Práctica:

Análisis VHDL.

1	

Determinar el comportamiento del siguiente circuito y simularlo. ¿ Por qué a la señal que se le asigna un rango entre 0 y 3 y no se la define directamente como un entero ? Determinar el retardo del circuito desde la matriz de retardos y desde el simulador gráfico.

```
entity proc is
port      ( d : in bit_vector( 2 downto 0 );
          q : out integer range 0 to 3
          );
end proc;

architecture maxpld of proc is
begin
process ( d )
variable num_bits : integer;
begin
          num_bits := 0;
          for i in d'range loop
                    if d( i ) = '1' then
                              num_bits := num_bits + 1;
                    end if;
          end loop;
          q <= num_bits;
end process;
end  maxpld;
```

2	

Determinar el comportamiento del siguiente circuito y simularlo. ¿Qué función realiza conv_integer() y en que biblioteca se encuentra? Determinar los retardos desde la matriz de reardos y desde la simulación, ¿Por qué difieren?

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

entity adder is
port      ( op1, op2 : in unsigned( 7  downto 0);
          result : out integer
          );
end adder;

architecture maxpld of adder is
begin
          result <= conv_integer( op1 + op2);
end  maxpld;
```

3	

Determinar el comportamiento del siguiente circuito y simularlo. Del sistema de ayuda del sistema determinar las características del componente dff y de un manual de la familia TTL las del integrado (macrofunción)74151.

```
library altera;
use altera.maxplus2.all;

library ieee;
use ieee.std_logic_1164.all;

entity compinst is
port    ( data, clock, clearn, presetn      : in std_logic;
          q_out  : out std_logic;
          a, b, c, gn : in std_logic;
          d : in std_logic_vector( 7 downto 0 );
          y, wn : out std_logic
          );
end compinst;

architecture a of  compinst is
begin
          dff1 : dff port map (d =>data, q => q_out, clk => clock, clrn => clearn, prn => presetn);
          mux      : a_74151b port map (c, b, a, d, gn, y, wn);
end a;
```

4	

Determinar el comportamiento del siguiente circuito y simularlo. Determinar las ecuaciones de síntesis. (Se trata de un multiplexor que en lugar de ser realizado como un proceso o mediante un componente se lo implementa mediante una asignación concurrente)

```
entity condsig is
port    ( input0, input1, sel : in bit;
          output   : out bit
          );
end condsig;

architecture maxpld of  condsig is
begin
          output <= input0 when sel = '0' else input1;
end maxpld;
```

5	

Determinar el comportamiento del siguiente circuito y simularlo. (Observe el empleo de la asignación concurrente condicional). ¿Qué ocurre cuando más de una entrada toma el valor 1? En la compilación emplear el Design Doctor y verificar en el simulador que los efectos predichos se cumplen. ¿Cuántos terminales se reservan para la señal q y por qué?

```
entity condsigm is
port    ( high, mid, low : in bit;
```

```
        q : out integer
        );
end condsigm;

architecture maxpld of condsigm  is
begin
q <=    3 when high = '1' else
        2 when mid  = '1' else
        1 when low  = '1' else
        0;
end maxpld;
```

6	

Determinar el comportamiento de cada uno de los siguientes contadores. ¿ Qué ocurre si en la vista en planta se pica dos veces con el mouse sobre una señal, o sobre una celda ? Emplear la familia FLEX.

```
entity counters is
port    (       d               : in integer range 0 to 255;
                clk             : in bit;
                clear           : in bit;
                ld              : in bit;
                enable          : in bit;
                up_down         : in bit;
                qa              : out integer range 0 to 255;
                qb              : out integer range 0 to 255;
                qc              : out integer range 0 to 255;
                qd              : out integer range 0 to 255;
                qe              : out integer range 0 to 255;
                qf              : out integer range 0 to 255;
                qg              : out integer range 0 to 255;
                qh              : out integer range 0 to 255;
                qi              : out integer range 0 to 255;
                qj              : out integer range 0 to 255;
                qk              : out integer range 0 to 255;
                ql              : out integer range 0 to 255;
                qm              : out integer range 0 to 255;
                qn              : out integer range 0 to 255
                );
end counters;

architecture a of counters is
begin
        -- Contador con habilitacion
        process ( clk )
        variable        cnt     : integer range 0 to 255;
        begin
                if ( clk'event and clk = '1' ) then
                        if enable = '1' then
                                cnt := cnt + 1;
                        end if;
                end if;
                qa      <= cnt;
```

```
end process;

-- Contador sincronico precargable
process ( clk )
variable        cnt     : integer range 0 to 255;
begin
        if ( clk'event and clk = '1' ) then
                if ld = '0' then
                        cnt := d;
                else
                        cnt := cnt + 1;
                end if;
        end if;
        qb      <=      cnt;
end process;

-- Contador sincronico con clear
process ( clk )
variable        cnt     : integer range 0 to 255;
begin
        if ( clk'event and clk = '1' ) then
                if clear = '0' then
                        cnt := 0;
                else
                        cnt := cnt + 1;
                end if;
        end if;
        qc      <=      cnt;
end process;

-- Contador ascendente descendente
process ( clk )
variable        cnt     : integer range 0 to 255;
variable        direction       : integer;
begin
        if ( up_down = '1' ) then
                direction := 1;
        else
                direction := -1;
        end if;
        if ( clk'event and clk = '1' ) then
                cnt := cnt + direction;
        end if;
        qd      <=      cnt;
end process;

-- Contador sincronico con habilitacion precargable
process ( clk )
variable        cnt     : integer range 0 to 255;
begin
        if ( clk'event and clk = '1' ) then
                if ld = '0' then
                        cnt := d;
                else
```

```
                        if enable = '1' then
                                cnt := cnt + 1;
                        end if;
                end if;
        end if;
        qe      <=      cnt;
end process;

-- Contador con habilitacion ascendente descendente
process ( clk )
variable        cnt     : integer range 0 to 255;
variable        direction       : integer;
begin
        if ( up_down = '1' ) then
                direction := 1;
        else
                direction := -1;
        end if;

        if ( clk'event and clk = '1' ) then
                if enable = '1' then
                        cnt := cnt + direction;
                end if;
        end if;
        qf      <=      cnt;
end process;

-- Contador sincronico con habilitacion y clear
process ( clk )
variable        cnt     : integer range 0 to 255;
begin
        if ( clk'event and clk = '1' ) then
                if clear = '0' then
                        cnt := 0;
                else
                        if enable = '1' then
                                cnt := cnt + 1;
                        end if;
                end if;
        end if;
        qg      <=      cnt;
end process;

-- Contador con clear precargable
process ( clk )
variable        cnt     : integer range 0 to 255;
begin
        if ( clk'event and clk = '1' ) then
                if clear = '0' then
                        cnt := 0;
                else
                        if ld = '0' then
                                cnt := d;
                        else
```

```
                                cnt := cnt + 1;
                        end if;
                end if;
        end if;
        qh      <=      cnt;
end process;

-- Contador precargable ascendente descendente
process ( clk )
variable        cnt     : integer range 0 to 255;
variable        direction       : integer;
begin
        if (up_down = '1') then
                direction := 1;
        else
                direction := -1;
        end if;

        if ( clk'event and clk = '1' ) then
                if ld = '0' then
                        cnt := d;
                else
                        cnt := cnt + direction;
                end if;
        end if;
        qi      <=      cnt;
end process;

-- Contador sincronico precargable, con habilitacion ascendente descendente
process ( clk )
variable        cnt     : integer range 0 to 255;
variable        direction       : integer;
begin
        if ( up_down = '1' ) then
                direction := 1;
        else
                direction := -1;
        end if;
        if ( clk'event and clk = '1' ) then
                if ld = '0' then
                        cnt := d;
                else
                        if enable = '1' then
                                cnt := cnt + direction;
                        end if;
                end if;
        end if;
        qj      <=      cnt;
end process;

-- Contador sincronico precargable con clear y habilitacion
process ( clk )
variable        cnt     : integer range 0 to 255;
begin
```

```
        if ( clk'event and clk = '1' ) then
                if clear = '0' then
                        cnt := 0;
                else
                        if ld = '0' then
                                cnt := d;
                        else
                                if enable = '1' then
                                        cnt := cnt + 1;
                                end if;
                        end if;
                end if;
        end if;
        qk      <=      cnt;
end process;

-- Contador sincronico ascendente descendente con clear
process ( clk )
variable        cnt             : integer range 0 to 255;
variable        direction       : integer;
begin
        if ( up_down = '1' ) then
                direction := 1;
        else
                direction := -1;
        end if;
        if ( clk'event and clk = '1') then
                if clear = '0' then
                        cnt := 0;
                else
                        cnt := cnt + direction;
                end if;
        end if;
        ql      <=      cnt;
end process;

-- Contador sincronico ascendente descendente con habilitacion y clear
process ( clk )
variable        cnt     : integer range 0 to 255;
variable        direction       : integer;
begin
        if (up_down = '1') then
                direction := 1;
        else
                direction := -1;
        end if;
        if ( clk'event and clk = '1' ) then
                if clear = '0' then
                        cnt := 0;
                else
                        if enable = '1' then
                                cnt := cnt + direction;
                        end if;
                end if;
```

```
		end if;
		qm	<=	cnt;
	end process;

	-- Contador de modulo 200
	process ( clk )
	variable	cnt		: integer range 0 to 255;
	constant	modulus		: integer := 200;
	begin
		if ( clk'event and clk = '1' ) then
			if cnt = modulus then
				cnt := 0;
			else
				cnt := cnt + 1;
			end if;
		end if;
		qn	<=	cnt;
	end process;
end a;
```

7	

Determinar el diagrama de estados del circuito y simularlo.

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

entity enumsmch is
port	(	updown : in std_logic;
		clock	: in std_logic;
		lsb	: out std_logic;
		msb	: out std_logic
	);
end enumsmch;

architecture firstenumsmch of enumsmch is
type count_state is (zero, one, two, three);
signal present_state, next_state		: count_state;
begin
process  ( present_state, updown )
	begin
		case present_state is
			when zero =>
				if (updown = '0') then
					next_state <= one;
					lsb <= '0';
					msb <= '0';
				else
					next_state <= three;
					lsb <= '1';
					msb <= '1';
				end if;
			when one =>
```

```
                        if (updown = '0') then
                                next_state <= two;
                                lsb <= '1';
                                msb <= '0';
                        else
                                next_state <= zero;
                                lsb <= '0';
                                msb <= '0';
                        end if;
                when two =>
                        if (updown = '0') then
                                next_state <= three;
                                lsb <= '0';
                                msb <= '1';
                        else
                                next_state <= one;
                                lsb <= '1';
                                msb <= '0';
                        end if;
                when three =>
                        if (updown = '0') then
                                next_state <= zero;
                                lsb <= '1';
                                msb <= '1';
                        else
                                next_state <= two;
                                lsb <= '0';
                                msb <= '1';
                        end if;
        end case;
    end process;

    process
    begin
        wait until clock'event and clock = '1';
        present_state <= next_state;
    end process;
end firstenumsmch;
```

8	

Determinar el comportamiento del siguiente circuito y simularlo. Interprete cada uno de los terminales de la RAM que se desea implementar. (Consultar la ayuda del sistema). a) Desde el simulador (Initialize menu) inicialice algunos lugares de memoria y léalos mediante una simulación. b) Mediante el editor de ondas y el simulador escriba un lugar de memoria y luego léalo.

```
--Si se colocan las constantes en el lugar deseado esta parte es opcional.

package ram_constants is
    constant addr_width : integer := 8;
    constant data_width : integer := 8;
end ram_constants;
```

```
library ieee;
use ieee.std_logic_1164.all;
library lpm;
use lpm.lpm_components.all;
library work;
use work.ram_constants.all;

entity ram256x8 is
port    ( data    :       in std_logic_vector ( data_width - 1 downto 0 );
          address :       in std_logic_vector ( addr_width - 1 downto 0 );
          we              :       in std_logic;
          inclock :       in std_logic;
          outclock:       in std_logic;
          q               :       out std_logic_vector ( data_width - 1 downto 0 )
          );
end ram256x8;

architecture example of ram256x8 is
begin
inst_1  : lpm_ram_dq
                        generic map (lpm_widthad => addr_width, lpm_width => data_width)
                        port map (data => data, address => address, we => we,  inclock => inclock,
                        outclock => outclock,  q => q);
end example;
```

(El archivo de inicialización se puede crear con un editor de texto ASCII o desde la ventana de inicialización, también se lo puede llamar desde aquí o especificar con el parámetro lpm_file).

Interprete el contenido de los archivos *.hex y *.mif.

9	

Determinar el comportamiento del siguiente circuito y simularlo. Este circuito será invocado por el circuito del ejecicio 10.

```
entity reg12 is
port    (       d               : in   bit_vector( 11 downto 0 );
                clk             : in   bit;
                q               : out  bit_vector( 11 downto 0 )
        );
end reg12;

architecture a of reg12 is
begin
        process
        begin
                wait until clk = '1' and clk'event;
                q <= d;
        end process;
end a;
```

10	

Determinar el comportamiento del siguiente programa y simularlo. Para llamar un componente ya definido se crea primero un paquete dentro del archivo de menor jerarquía o en un archivo aparte y luego se lo invoca como un componente.

```
package reg24_package is
        component reg12
                port(
                        d       : in    bit_vector(11 downto 0);
                        clk     : in    bit;
                        q       : out   bit_vector(11 downto 0));
        end component;
end reg24_package;

library work;
use work.reg24_package.all;

entity reg24 is
port    (       d               : in   bit_vector( 23 downto 0 );
                clk             : in   bit;
                q               : out  bit_vector( 23 downto 0 )
        );
end reg24;

architecture a of reg24 is
begin
        reg12a  : reg12 port map (d => d(11 downto 0), clk => clk,

                                q => q(11 downto  0));
        reg12b  : reg12 port map (d => d(23 downto 12), clk => clk,
                                q => q(23 downto 12));
end a;
```

11	

Determinar el comportamiento del siguiente programa y simularlo. Determinar las caractrísticas del componente lpm_ff.

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

entity reg24lpm is
port    (       d               : in  std_logic_vector( 23 downto 0 );
                clk             : in  std_logic;
                q               : out std_logic_vector( 23 downto 0 )
        );
end reg24lpm;

architecture a of reg24lpm is
begin
        reg12a  : lpm_ff
```

```
            generic map (lpm_width => 12)
            port map (data => d(11 downto 0), clock => clk,
                        q => q(11 downto 0));
    reg12b  : lpm_ff
            generic map (lpm_width => 12)
            port map (data => d(23 downto 12), clock => clk,
                        q => q(23 downto 12));
end a;
```

12	

Determinar el comportamiento del siguiente programa y simularlo.

```
entity reginf is
port    (       d, clk, clr, pre, load, data        : in bit;
                q1, q2, q3, q4, q5, q6, q7          : out bit
        );
end reginf;

architecture maxpld of reginf is
begin

        -- register with active-high clock
        process
        begin
           wait until clk = '1';
           q1 <= d;
        end process;

        -- register with active-low clock
        process
        begin
           wait until clk = '0';
           q2 <= d;
        end process;

        -- register with active-high clock & asynchronous clear
        process (clk, clr)
        begin
           if clr = '1' then
              q3 <= '0';
           elsif clk'event and clk = '1' then
              q3 <= d;
           end if;
        end process;


        -- register with active-low clock & asynchronous clear
        process (clk, clr)
        begin
           if clr = '0' then
              q4 <= '0';
           elsif clk'event and clk = '0' then
```

```
            q4 <= d;
        end if;
    end process;

    -- register with active-high clock & asynchronous preset
    process (clk, pre)
    begin
        if pre = '1' then
            q5 <= '1';
        elsif clk'event and clk = '1' then
            q5 <= d;
        end if;
    end process;

    -- register with active-high clock & asynchronous load
    process (clk, load, data)
    begin
        if load = '1' then
            q6 <= data;
        elsif clk'event and clk = '1' then
            q6 <= d;
        end if;
    end process;

    -- register with active-high clock & asynchronous clear & preset
    process (clk, clr, pre)
    begin
        if clr = '1' then
            q7 <= '0';
                elsif pre = '1' then
                        q7 <= '1';
        elsif clk'event and clk = '1' then
            q7 <= d;
        end if;
    end process;
end maxpld;
```

13	

Determinar el comportamiento del siguiente programa y simularlo. (Asignación concurrente con selección).

```
entity selsig is
port    (       d0, d1, d2, d3    : in bit;
                s                                 : in integer range 0 to 3;
                output                    : out bit
        );
end selsig;

architecture maxpld of selsig is
begin

with s select               -- creates a 4-to-1 multiplexer
```

```
        output <=       d0 when 0,
                        d1 when 1,
                        d2 when 2,
                        d3 when 3;

end maxpld;
```

14	

Determinar el comportamiento del siguiente programa y simularlo.

```
package meals_pkg is
        type meal is (breakfast, lunch, dinner, midnight_snack);
end meals_pkg;

use work.meals_pkg.all;

entity selsigen is
port    (       previous_meal   : in meal;
                next_meal               : out meal
        );
end selsigen;

architecture maxpld of selsigen is
begin

with previous_meal      select
        next_meal <=    breakfast       when dinner | midnight_snack,
                                        lunch           when breakfast,
                                        dinner          when lunch;

end maxpld;
```

15	

Determinar el comportamiento del siguiente programa y simularlo. (Asignaciones concurrentes)

```
entity simpsig is
port    (       a, b, e   : in  bit;
                c, d      : out bit
        );
end simpsig;

architecture maxpld of simpsig is
begin

        c <= a and b;
        d <= e;

end maxpld;
```

16	

Determinar el comportamiento del siguiente programa y simularlo. Realizar la tabla y el diagrama de estados. El reset es sincrónico o asincrónico. Observe en el editor de ondas como se identifican los estados enumerados.

```
entity statmach is
port    (       clk             : in    bit;
                input   : in    bit;
                reset   : in    bit;
                output  : out   bit);
end statmach;

architecture a of statmach is
        type state_type is (s0, s1);
        signal state    : state_type;
begin
        process (clk)
        begin
                if reset = '1' then
                        state <= s0;
                elsif (clk'event and clk = '1') then
                        case state is
                                when s0=>
                                        state <= s1;
                                when s1=>
                                        if input = '1' then
                                                state <= s0;
                                        else
                                                state <= s1;
                                        end if;
                        end case;
                end if;
        end process;

        output <= '1' when state = s1 else '0';
end a;
```