Universidad Nacional Autonoma de Mexico Facultad de ingeniería

Diseño Digital VLSI

Grupo : 4

Práctica: 1 Diseño de un reloj digital

Edgar Daniel Barcenas Martinez

Objetivo:

Demostrar mediante el diseño de un reloj digital, que las declaraciones concurrentes se efectúan al mismo tiempo (en paralelo), el orden de escritura en las instrucciones concurrentes no afecta el resultado de síntesis o de simulación.

Especificaciones:
Utilizando un FPGA de 7 Segmentos, diseñar un reloj digital, en el cual se visualice en los 2 primeros displays las horas y en los siguientes los minutos. Cada vez que se llegue a 23 horas se reinicie el conteo de horas y minutos.

Diagrama de Bloques:

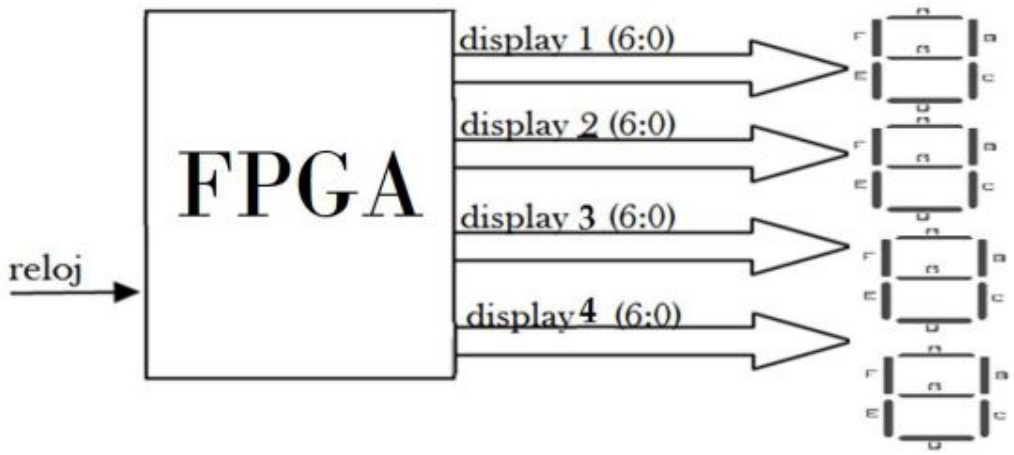


Figura 1.1. Diagrama de bloques del sistema Reloj Digital

Bloques funcionales:

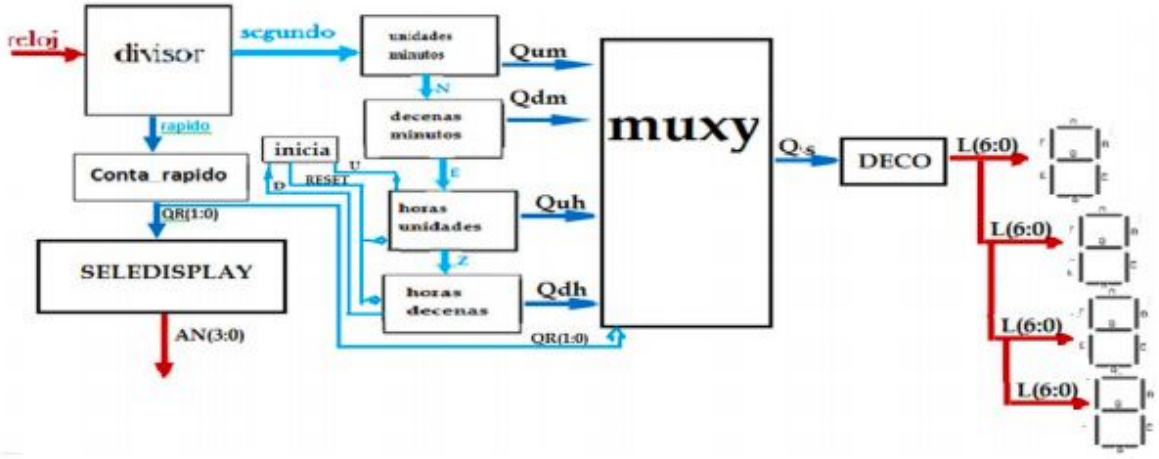


Figura 1.2. Bloques funcionales del sistema Reloj Digital

Codigo:
Entidad del sistema Reloj Digital

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity reldig is
Port ( reloj : in  std_logic;
       AN : out  std_logic_vector (3 downto 0);
       L : out  std_logic_vector (6 downto 0));
end reldig;
```

Arquitectura del sistema Reloj Digital

```
architecture behavioral of reldig is
   signal segundo : std_logic;
   signal rapido: std_logic;
   signal n : std_logic;
   signal Qs: std_logic_vector(3 downto 0);
   signal Qum: std_logic_vector(3 downto 0);
   signal Qdm:   std_logic_vector(3 downto 0);
   signal e : std_logic;
   signal Qr: std_logic_vector(1 downto 0);
   signal Quh: std_logic_vector(3 downto 0);
   signal Qdh: std_logic_vector(3 downto 0);
   signal z : std_logic;
   signal u : std_logic;
   signal d : std_logic;
   signal reset : std_logic;
```

Parte Operatoria de la arquitectura del sistema Reloj Digital

```
begin
   divisor : process (reloj)
      variable CUENTA: STD_LOGIC_VECTOR(27 downto 0) :=  X"0000000";
   begin
      if rising_edge (reloj) then
         if CUENTA =X"48009E0" then
            CUENTA :=  X"0000000";
         else
            CUENTA := CUENTA +1;
         end if;
      end if;
      segundo <= CUENTA(22);
      rapido  <= CUENTA(10);
   end process;


   UNIDADES: process (segundo)
      variable CUENTA: STD_LOGIC_VECTOR(3 downto 0) := "0000";
   begin
      if rising_edge (segundo) then
         if  CUENTA ="1001" then
            CUENTA :="0000";
            N  <= '1';
         else
            CUENTA := CUENTA +1;
            N  <= '0';
```

```
        end if;
    end if;
    Qum <= CUENTA;
end process;
```

```
DECENAS: process (N)
   variable CUENTA: STD_LOGIC_VECTOR(3 downto 0) := "0000";
begin
   if rising_edge (N) then
      if  CUENTA  ="0101" then
         CUENTA :="0000";
         E <= '1';
      else
         CUENTA := CUENTA +1;
         E <= '0';
      end if;
   end if;
   Qdm <= CUENTA;
end process;

HoraU: Process(E,reset)
   variable cuenta: std_logic_vector(3 downto 0):="0000";
begin
   if rising_edge(E) then
      if cuenta="1001" then
         cuenta:= "0000";
         Z<='1';
      else
         cuenta:=cuenta+1;
         Z<='0';
      end if;
   end if;
   if reset='1' then
      cuenta:="0000";
   end if;
   Quh<=cuenta;
   U<=cuenta(2);
end Process;

HORAD: Process(Z, reset)
   variable cuenta: std_logic_vector(3 downto 0):="0000";
begin
   if rising_edge(Z) then
      if cuenta="0010" then
         cuenta:= "0000";
      else
         cuenta:=cuenta+1;
      end if;
   end if;
   if reset='1' then
      cuenta:="0000";
   end if;
   Qdh<=cuenta;
   D <=cuenta(1);
end Process;
```

```
   inicia: process (U,D)
   begin
      reset <= (U and D);
   end process;

   CONTRAPID: process (rapido)
      variable CUENTA: STD_LOGIC_VECTOR(1 downto 0) := "00";
   begin
      if rising_edge (rapido) then
         CUENTA := CUENTA +1;
      end if;
      Qr <= CUENTA;
   end process;

   MUXY: PROCESS (Qr)
   BEGIN
      if Qr = "00"  then
         Qs<= Qum;
      elsif Qr = "01" then
         Qs<= Qdm;
      elsif Qr = "10" then
         Qs<= Quh;
      elsif Qr = "11" then
         Qs<= Qdh;
      end if;
   end process;

   seledisplay:  process (Qr)
   BEGIN
      case Qr is
         when "00" =>
            AN<= "1110";
         when "01" =>
            AN<= "1101";
         when "10" =>
            AN<= "1011";
         when others =>
            AN<= "0111";
      end case;
   end process;

   with   Qs SELECT
      L <= "1000000" when "0000",    --0
           "1111001" when "0001",    --1
           "0100100" when "0010",    --2
           "0110000" when "0011",    --3
           "0011001" when "0100",    --4
           "0010010" when "0101",    --5
           "0000010" when "0110",    --6
           "1111000" when "0111",    --7
           "0000000" when "1000",    --8
           "0010000" when "1001",    --9
           "1000000" when others;    --F
end Behavioral;
```

Actividades complementarias:

La actividad complementaria consiste en determinar que se encendiera un LED en una determinada hora, también consistió en apagar ese led con un push button.

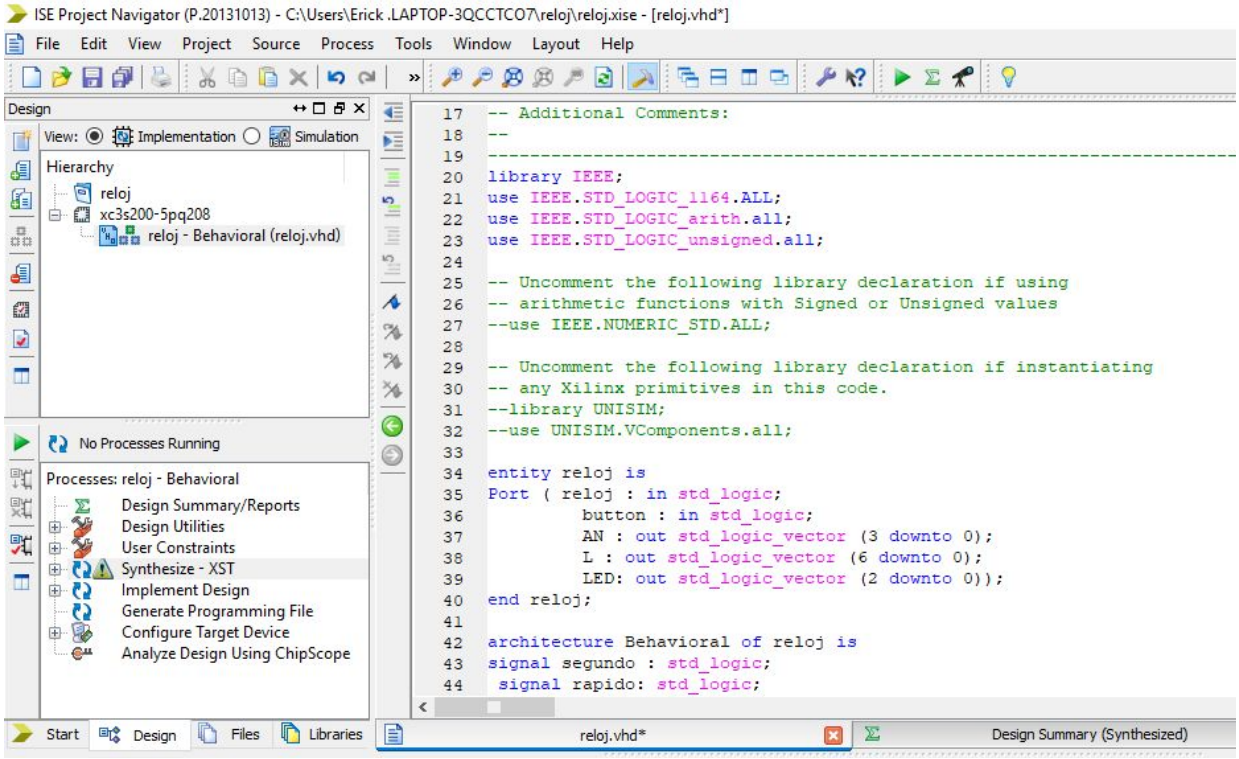
Primero tenemos que modificar la entidad. En esta agregamos un nuevo puerto al que llamaremos LED y lo declararemos como std_logic_vector (2 downto 0)

```
34  entity reloj is
35  Port ( reloj : in std_logic;
36           button : in std_logic;
37           AN : out std_logic_vector (3 downto 0);
38           L : out std_logic_vector (6 downto 0);
39           LED: out std_logic_vector (2 downto 0));
40  end reloj;
41
```

En la parte de la arquitectura debemos de mantenerlo igual. Debemos agregar en la parte operatoria de la arquitectura un nuevo proceso para que se ejecute en paralelo.

```
60  Alarma: process(Qum,Qdm,Quh,Qdh)
61     variable boton: STD_LOGIC;
62     BEGIN
63     ---Encender LED
64        if Qum = "0001" and
65           Qdm = "0001" and
66           Quh = "0000" and
67           Qdh = "0000" then
68           boton := '1';
69        end if;
70        if boton = '1' then
71           LED <= "111";
72        end if;
73        ---Apagar LED
74        if button = '1' then
75           LED <= "000";
76        end if;
77     end process;
78
```

Compilacion:



Conclusiones:

En esta práctica aprendí que los FPGA funcionan en paralelo, es decir los procesos dentro de nuestro código. Tambien aprendi que el orden de escritura en las instrucciones concurrentes no afecta el resultado de síntesis o de simulación. Entendí que en VHDL tenemos una estructura del código. En la Entidad tenemos una interfaz en la que indicamos la entrada y las salidas, en la arquitectura describimos el funcionamiento de la entidad y tenemos los procesos que se ejecutan de forma concurrente, tambien aprendi a como implementar variables, constantes, señales y sentencias de control.

Respecto al programa Xilinx aprendí la configuración del FPGA, su carga y la asignación de pines.