PRACTICA LIBRE: VOLTIMETRO DIGITAL IMPLEMENTADO CON UNA FPGA

APARTADO 1

Centro:
Universidad de Alcalá
Grado:
Grado en Ingeniería de Computadores
Asignatura:
Modelado de Sistemas Computacionales
Curso:
Curso Académico 2020/2020
Curso 3º - Cuatrimestre 2º
Grupo:
A3
Alumnos:
Sergio Sanz Cacho – 03201575K
Carlos Tejeda Martínez – 03148129G

INDICE

DISEÑO DEL MÓDULO CNT_DISPLAY

PLANTEAMIENTO

De acuerdo con el desarrollo de la práctica, hemos tenido que desarrollar un controlador para el display de un voltímetro, que consiste en cuatro displays y un punto para la representación del valor medido que puede comprender desde 0 hasta 9999.

Este controlador se basa en la adquisición del valor a representar en código binario y mostrarlo en código decimal mediante los displays, tras el tratamiento de la señal adquirida. Para ello el módulo tendrá como entrada las señales de reloj (CLK), reset (RST), valor en binario (BCD), indicador de nuevo valor (BCD_OK); y como salidas el valor de la cifra a representar en el display (SEG_AG), el display al que va dirigido el valor (AND_30) y la señal de encendido del punto del display (DP).

DISEÑO

En primer lugar, a partir del diagrama expuesto en clase, estudiamos como ha de conformarse el interior del controlador a través de los componentes mostrados en el esquema, los cuales su función será implementada como procesos. Estos componentes son:

- Registro de 16bits:
  Este componente recibe por entrada el valor a representar en formato BCD y cuando la señal de validación del nuevo valor (BCD_OK) llega, se divide el valor de entrada de BCD en las cuatro cifras para cada display que se darán por las salidas BCD_U, BCD_D, BCD_C y BCD_M.
  En caso de que llegue la señal activa de reset (RST) se procede a poner las cuatro salidas de las cifras a cero.

```
process(CLK,RST,BCD_OK)
  begin
  --Si la señal de reset esta activa se pone todo a 0
  if RST = '1' then
    BCD_U <= (others => '0');
    BCD_D <= (others => '0');
    BCD_C <= (others => '0');
    BCD_M <= (others => '0');
  elsif CLK'event and CLK = '1' then
  --Comprobación del estado de la señal BCD_OK
    A <= BCD_OK;
    APTO <= (NOT A) and BCD_OK;
    --Si la señal BCD_OK esta activa se toma el valor por BCD
    if(APTO='1')then
      --Troceamos BCD
      BCD_U(3 downto 0) <= BCD(3 downto 0);
      BCD_D(3 downto 0) <= BCD(7 downto 4);
      BCD_C(3 downto 0) <= BCD(11 downto 8);
      BCD_M(3 downto 0) <= BCD(15 downto 12);
    end if;
  end if;
end process;
```

Ilustración 1-Implementación en VHDL del registro de 16bits

- Multiplexor:
  Este componente es regulado por la señal S proveniente del contador de 2bits la cual cada 1 milisegundo ira variando, dando por salida del multiplexor la diferente configuración a mostrar en cada uno de los displays dependiendo del tipo de cifra a

mostrar que es. Las salidas que tiene consisten en la señal para tener activo o desactivado el punto del display (DP), la señal que indica que display se está configurando (AND_30) y finalmente la señal en binario de la cifra a mostrar que pasará previamente a su visionado en el display por un conversor del número a binario a BCD para activar los segmentos apropiados del display (S_multiplexor).

```
process(S,BCD_U,BCD_D,BCD_C,BCD_M)
begin
    case S is
      --unidades
      when "00" =>
          S_multiplexor <= BCD_U;
          AND_30 <= "1110";
          DP <= '1';
      --decenas
      when "01" =>
          S_multiplexor <= BCD_D;
          AND_30 <= "1101";
          DP <= '1';
      --centenas
      when "10" =>
          S_multiplexor <= BCD_C;
          AND_30 <= "1011";
          DP <= '1';
      --millares
      when others =>
          S_multiplexor <= BCD_M;
          AND_30 <= "0111";
          DP <= '0';
     end case;
end process;
```

Ilustración 2-Implementación en VHDL del multiplexor

Tabla de combinaciones del multiplexor:

S	S_multiplexor	AND_30	DP
00	BCD_U	1110	1
01	BCD_D	1101	1
10	BCD_C	1011	1
11	BCD_M	0111	0

- Preescaler:

Este componente produce la señal CE_preescaler a partir de la señal de reloj (CLK), pasando de recibir pulsos cada 10 nanosegundos a producir una señal de salida con pulsos cada 1 milisegundo para activar el funcionamiento del contador de 2bits.
En caso de que llegue la señal activa de reset (RST) se procede a reiniciar la emisión de pulsos de 1ms.

```
process (CLK, RST)
begin
 --Si la señal de reset esta activa se reinicia la creación de pulsos de 1ms
  if RST = '1' then
    counter_reg   <= 0;
  elsif CLK'event and CLK = '1' then
    --Si se supera el tiempo para el pulso se reinicia para uno nuevo
    if counter_reg = CLKDIV-1 then
      counter_reg <= 0;
    --Si aun no se ha llegado al tiempo para el pulso, se incrementa el contador
    else
      counter_reg <= counter_reg+1;
    end if;
  end if;
end process;
--Se emite el pulso cuando se ha contado el paso de 1 milisegundo
CE_preescaler <= '1' when counter_reg = CLKDIV-1 else '0';
```

Ilustración 3-Implementación en VHDL del preescaler

- Contador de 2bits:

  Este componente es regulado por la señal CE_preescaler la cuál porta un pulso cada 1 milisegundo que activará al contador para incrementarse en uno su valor de salida S. Cuando llega a 3 (“11”) vuelve a 0 (“00”) por tanto podemos indicar las salida de las 4 cifras en el multiplexor en bucle, refrescando los datos mostrados cada 4 milisegundos.

  En caso de que llegue la señal activa de reset (RST) se procede a iniciar desde 0 el contador.

```
process (CLK, RST, CE_preescaler)
begin
    --Si la señal de reset esta activa se reinicia el contador a 0
    if (RST = '1') then
      S <= (others => '0');
    elsif (CLK'event and CLK = '1')then
      --Si hemos recibido un nuevo pulso se incrementa el contador
      if (CE_preescaler = '1') then
          S <= S+1;
      end if;
    end if;
end process;
```

Ilustración 4-Implementación en VHDL del contador de 2bits

- Conversor BCD-7segmentos:

  Este componente recibe la cifra que se ha de mostrar en formato BCD (S_multiplexor) y en función del número le asigna su combinación para la visualización en 7 segmentos, de acuerdo con los segmentos que han de encenderse para la representación de la cifra, que se trasmitirá a través de la salida SEG_AG al display correspondiente.

```
process(S_multiplexor)
begin
  case S_multiplexor is
      when "0000" =>
          SEG_AG <= "1000000";--0
       when "0001" =>
          SEG_AG <= "1111001";--1
       when "0010" =>
          SEG_AG <= "0100100";--2
       when "0011" =>
          SEG_AG <= "0110000";--3
       when "0100" =>
          SEG_AG <= "0011001";--4
       when "0101" =>
          SEG_AG <= "0010010";--5
       when "0110" =>
          SEG_AG <= "0000010";--6
       when "0111" =>
          SEG_AG <= "1111000";--7
       when "1000" =>
          SEG_AG <= "0000000";--8
       when "1001" =>
          SEG_AG <= "0011000";--9
       when others =>
          SEG_AG <= "0111111";--central
       end case;
end process;
```

Ilustración 5-Implementación en VHDL del conversor de BCD a 7 segmentos

Tabla de combinaciones del conversor:

S_multiplexor	SEG_AG
0000	1000000
0001	1111001
0010	0100100
0011	0110000
0100	0110001
0101	0010010
0110	0000010
0111	1111000
1000	0000000
1001	0011000
XXXX	0111111

Combinando estos componentes el diagrama completo de la estructura del cnt_display queda asi:

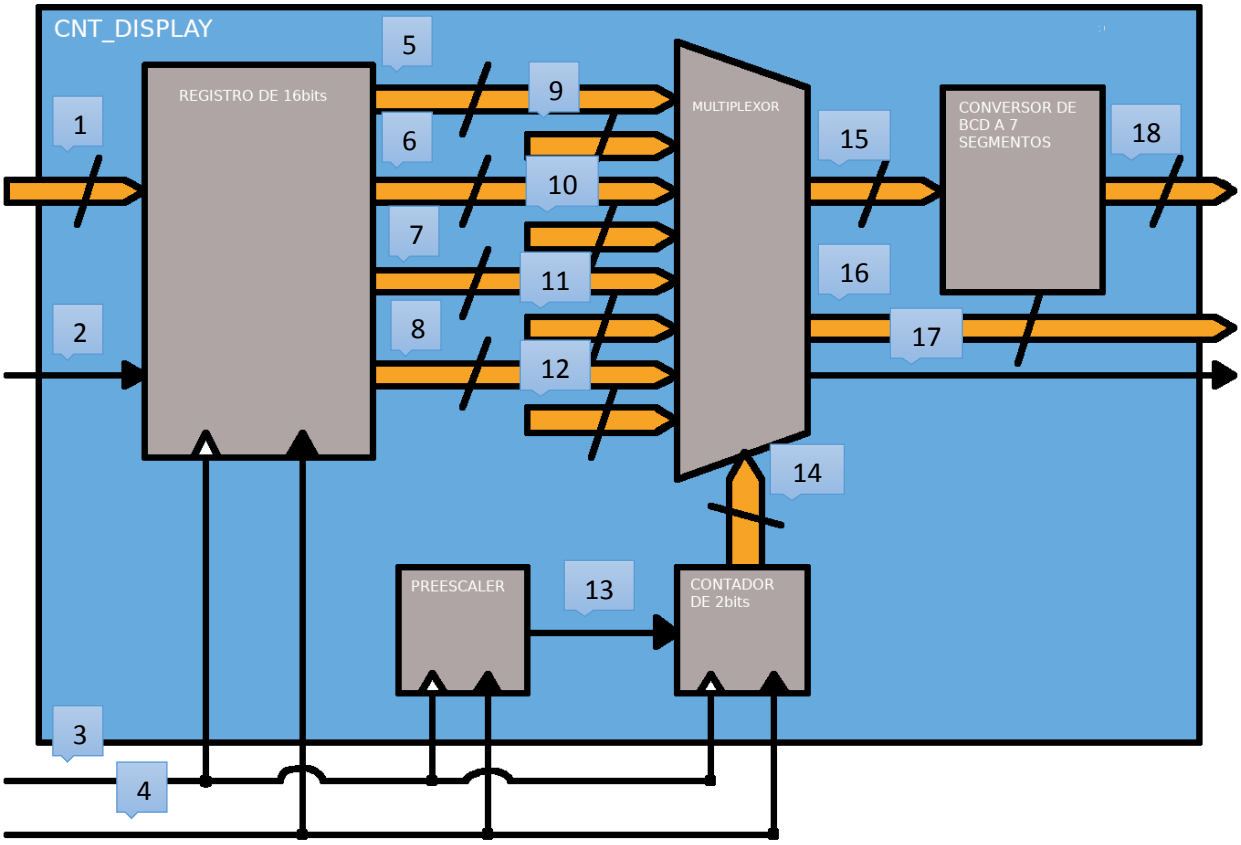


Ilustración 6-Esquema completo de la estructura de la entidad cnt_display con las señales utilizadas

Las señales indicadas en la figura superior son detalladas a continuación:

Número	Señal	Número	Señal
1	BCD (16bits)	10	Constante para AND_30 relativa a BCD_C (4bits)
2	BCD_OK	11	Constante para AND_30 relativa a BCD_D (4bits)
3	CLK	12	Constante para AND_30 relativa a BCD_U (4bits)
4	RST	13	CE_preescaler
5	BCD_M (4bits)	14	S (2bits)
6	BCD_C (4bits)	15	S_multiplexor (4bits)
7	BCD_D (4bits)	16	AND_30 (4bits)
8	BCD_U (4bits)	17	DP
9	Constante para AND_30 relativa a BCD_M (4bits)	18	SEG_AG (7bits)

Resumiendo, el desarrollo completo se basa en la llegada valores en codificación BCD por la señal BCD, con los cuales se empieza a trabajar en el registro de 16 bits una vez que llega un pulso a nivel alto de la señal BCD_OK.

Cuando se valida el valor obtenido por BCD, se procede a dividir la señal recogida en los 4 segmentos relativos a cada cifra del valor en las señales de menor a mayor peso de BCD_U, BCD_D, BCD_C y BCD_M, llegando esta señal al multiplexor.

Estos valores serán pasados a los displays indicados de manera secuencial cada 1ms, refrescando cada display cada 4ms mediante el uso de la señal de control del multiplexor S, generada por el contador binario de 2bits alimentando su funcionamiento mediante una señal generada en el preescaler que ocasiona las variaciones cada 1ms.

Finalmente, como último paso al visionado del valor por el display se traduce el valor BCD a 7 segmentos para su representación en el display.

SIMULACIÓN FUNCIONAL

En la creación del testbench para ambas simulaciones, declaramos los componentes y señales apropiadas del cnt_display y asignamos una señal inicial de RESET activa durante 123ns, desactivándose después; además de declarar la señal de RELOJ de 10ns de período

```
    --COMPONENTES
    component test_ap1
        port (
            CLK    : in  std_logic;
            RST    : in  std_logic;
            SW_OK  : in  std_logic;
            SW     : in  std_logic_vector(15 downto 0);
            AND_30 : out std_logic_vector(3 downto 0);
            DP     : out std_logic;
            SEG_AG : out std_logic_vector(6 downto 0));
    end component;
    --SEÑALES
    signal RELOJ : std_logic := '0';
    signal RESET : std_logic := '1';
    signal SW_NUMERO : std_logic_vector(15 downto 0);
    signal SW_OK_NUMERO :  std_logic := '0';
    signal AND_30_SALIDA_DISPLAY :  std_logic_vector(3 downto 0);
    signal DP_PUNTO     :  std_logic;
    signal SEG_AG_DISPLAY_SELEC :  std_logic_vector(6 downto 0);

    signal D_Display_Aux : std_logic_vector(3 downto 0);

    signal BCD_U : std_logic_vector(3 downto 0);
    signal BCD_D : std_logic_vector(3 downto 0);
    signal BCD_C : std_logic_vector(3 downto 0);
    signal BCD_M : std_logic_vector(3 downto 0);

    signal D_Display : std_logic_vector(15 downto 0);

begin

    DUT: test_ap1
        port map(
            CLK => RELOJ,
            RST => RESET,
            SW => SW_NUMERO,
            SW_OK => SW_OK_NUMERO,
            AND_30 => AND_30_SALIDA_DISPLAY,
            DP => DP_PUNTO,
            SEG_AG => SEG_AG_DISPLAY_SELEC);
    --Se pone la señal de reset (rst) a bajo nivel a los 123ns
    RESET <= '0' after 123 ns;
    --Se aplica una señal de reloj de 10ns de período
    RELOJ <= NOT RELOJ after 5 ns;
```

Ilustración 6-Asignación de señales en el testbench

En primer lugar, tenemos un proceso en el cual se atribuye una secuencia de estimulaciones que consisten en la introducción de un valor al cnt_display para posteriormente tras 6 milisegundos se pasa un nuevo dato. En esta secuencia se ha simulado el pulso del SW_OK entre dos flancos de bajada para tener el pulso de 10 nanosegundos.

La metodología seguida es en cargar un nuevo valor en SW_NUMERO, ocasionar el pulso SW_OK como hemos detallado anteriormente y esperar un tiempo para repetir esta secuencia de instrucciones con otro valor.

```
process
begin
    --Esperamos 200ns
    wait for 200ns;
    --Cargamos un nuevo valor "1999"
    SW_NUMERO <= "0001100110011001";
    --Esperamos al flanco de bajada
    wait until RELOJ = '0';
    --Cargamos que hay nuevo valor (se pulsa boton)
    SW_OK_NUMERO <= '1';
    --Esperamos al flanco de bajada
    wait until RELOJ = '0';
    --Desactivamos que hay nuevo valor (no se pulsa boton)
    SW_OK_NUMERO <= '0';
    --Esperamos 6 ms
    wait for 6 ms;
    --Cargamos un nuevo valor "0000"
    SW_NUMERO <= "0000000000000000";
    --Esperamos al flanco de bajada
    wait until RELOJ = '0';
    --Cargamos que hay nuevo valor (se pulsa boton)
    SW_OK_NUMERO <= '1';
    --Esperamos al flanco de bajada
    wait until RELOJ = '0';
    --Desactivamos que hay nuevo valor (no se pulsa boton)
    SW_OK_NUMERO <= '0';
    --Esperamos 6 ms
    wait for 6 ms;
    report "fin controlado d ela simulación" severity failure;
end process;
```

Ilustración 7-Implementación en testbench en VHDL de una secuencia de actuación del cnt_display

El siguiente conjunto de procesos su función comprende en convertir el valor mostrado por los displays al valor decimal para su comprobación del resultado en la señal D_DISPLAY.

El siguiente proceso examina el dígito mostrado por el display para traducirlo a BCD

```
process(SEG_AG_DISPLAY_SELEC,AND_30_SALIDA_DISPLAY)
    Variable Display_Bin : std_logic_vector(3 downto 0);
begin
    case SEG_AG_DISPLAY_SELEC is
        when  "1000000" =>
            Display_Bin := "0000";--0
        when "1111001" =>
            Display_Bin := "0001";--1
        when "0100100" =>
            Display_Bin := "0010";--2
        when "0110000" =>
            Display_Bin := "0011";--3
        when "0011001" =>
            Display_Bin := "0100";--4
        when "0010010" =>
            Display_Bin := "0101";--5
        when "0000010" =>
            Display_Bin := "0110";--6
        when "1111000" =>
            Display_Bin := "0111";--7
        when "0000000" =>
            Display_Bin := "1000";--8
        when "0011000" =>
            Display_Bin := "1001";--9
        when others =>
            Display_Bin := "1111";--central
    end case;
    --Escribimos el valor a mostrar del display
    D_Display_Aux<=Display_bin;
end process;
```

Ilustración 8 Conversor de 7 segmentos a BCD

Este proceso a partir de la obtención del valor en BCD y dependiendo del display del que se ha obtenido la información, recabamos los valores de cada posición.

```
process (D_Display_Aux,AND_30_SALIDA_DISPLAY)
begin
    case AND_30_SALIDA_DISPLAY is
        --unidades
        when "1110" =>
            BCD_U <= D_Display_Aux;
        --decenas
        when "1101" =>
            BCD_D <= D_Display_Aux;
        --centenas
        when "1011" =>
            BCD_C <= D_Display_Aux;
        --millares
        when others =>
            BCD_M <= D_Display_Aux;
    end case;
end process;
```

Ilustración 9-Proceso en VHDL para obtener valores BCD de los displays

El último proceso coloca el conjunto de los valores de los displays, tratado por los anteriores procesos, acorde a sus pesos por el display que ocupan y obtenemos el valor completo mostrado en los displays

```
process (BCD_U,BCD_D,BCD_C,BCD_M)
begin
    D_Display(3 downto 0) <= BCD_U(3 downto 0);
    D_Display(7 downto 4) <= BCD_D(3 downto 0);
    D_Display(11 downto 8) <= BCD_C(3 downto 0);
    D_Display(15 downto 12) <= BCD_M(3 downto 0);
end process;
```

Ilustración 10-Proceso en VHDL para obtener el valor en BCD

Con el proyecto sintetizado lanzamos la simulación funcional de duración total de 12 milisegundos, resumiéndose la ejecución completa en la siguiente imagen:

Ilustración 11-Diagrama de la ejecución completa del testbench del ap1

Las señales del sistema, las cuales están divididas en diferentes grupos, son:

- Las dos primeras señales son:
  - RELOJ : Señal de reloj que sincroniza todos los eventos del sistema.
  - RESET : Señal asíncrona encargada de reiniciar el sistema.
- Las señales de entrada
  - SW_NUMERO : Valor de entrada en este caso (1999 y 0000), que será mostrado en los displays
  - SW_OK_NUMERO : Entrada por la que se comunica mediante un pulso de 10ns que tenemos un valor nuevo en la entrada
- Señal interna:
  - BCD_OK: Entrada del CNT_DISPLAY, que es excitada por la señal generada por SW_OK
- Señales de salida:
  - AND_30_SALIDA_DISPLAY : Señal encargada de indicar en cuál de los 4 displays nos encontramos
  - DP_PUNTO : Señal que indica si se debe o no encender el punto decimal
  - SEG_AG_DISPLAY_SELEC : Valor en 7 segmentos a escribir en el display indicado por AND_30
- Señales auxiliares del testBench

- D_Display_Aux : Señal que guarda el valor en BCD del display actual.
- BCD_U : Señal que guarda el valor en Bin del display correspondiente a las unidades.
- BCD_D : Señal que guarda el valor en Bin del display correspondiente a las decenas.
- BCD_C : Señal que guarda el valor en Bin del display correspondiente a las centenas.
- BCD_M : Señal que guarda el valor en Bin del display correspondiente a los millares.

- Señal que muestra el estado de los displays
  - D_DISPLAY : Esta señal recoge el valor completo escrito por los 4 displays, para poder comprobar su correcto funcionamiento.

En la imagen anterior observamos la estimulación de la SW_NUMERO con dos valores distintos, “1999” primero y después “0000”. Además, se estimula la entrada BCD_OK mediante un pulso de 10ns, para indicar la validación de un nuevo dato de entrada.

La recepción de los nuevos valores, acorde a la llegada de señal activa de BCD_OK, y su posterior muestreo en los displays, se comprueba en las siguientes dos imágenes.

Ilustración 12 -Estimulo inicial “1999”

Lo primero que observamos en estas dos imágenes, es que la señal SW_OK no es la que desencadena la aceptación de un nuevo valor, sino que esta señal genera el pulso de 10ns(BCD_OK) encargado de validar el nuevo valor de entrada.

Ilustración 13-Segundo estimulo "0000"

También cabe destacar que, debido al funcionamiento de los procesos en el cuál las señales se actualizan al final de este, podemos observar que la actualización del valor BCD se produce en el flanco de subida del reloj posterior al que detecta el BCD_OK.

SIMULACIÓN TEMPORAL

Para esta simulación se emplea el mismo banco de pruebas que para la simulación funcional lo que implica que el código es el mismo que el explicado en el apartado anterior.

La simulación temporal, que es lanzada tras la implementación, posee una duración total de 12 ns.

Durante los cuales podemos observar el efecto de los estímulos aplicados(Señales de entrada "1999" y "0000", y el indicador de nuevo valor SW_OK) a las distintas entradas, en las diferentes salidas del sistema, como muestra la siguiente imagen.

En la siguiente ilustración podemos observar todas las señales del sistema las cuales se encuentran explicadas en el apartado de la simulación funcional.

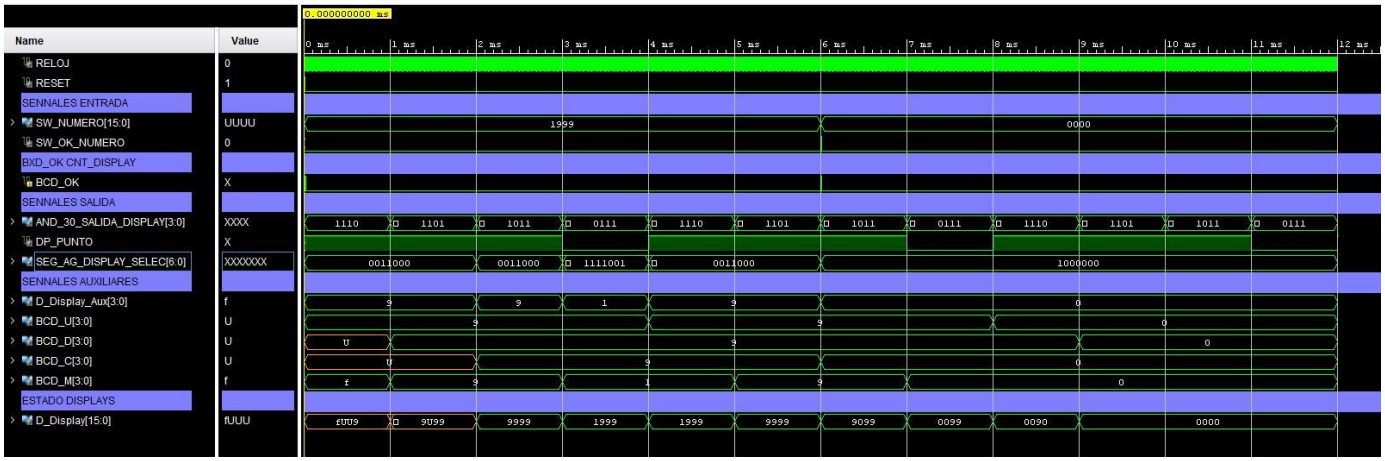


Ilustración 14-Simulación temporal completa

En las 2 imágenes siguientes se observa el efecto de la activación del SW_OK que a su vez desencadena la activación del BCD_OK, cuya activación carga el nuevo valor de entrada en los displays.

Ilustración 15 -Primer estimulo "1999" y tiempo de retardo

En este primer estimulo podemos observar el retardo 10.422 ns entre el flanco de subida en el cuál debería haberse producido el cambio del valor BCD a escribir en los displays y el momento en el cual se produce.

En la siguiente imagen observamos el segundo estimulo el cambio entre "1999" y "0000" y al igual que con el primer estimulo observamos un retardo entre el flanco de subida en el cual se debería haber producido el cambio en dicho valor BCD y el momento en el que ocurre, aunque esta vez dicho retardo es de 9.246ns.

Ilustración 16-Segundo estimulo y tiempo de retardo

Tras esta simulación temporal para la cual hemos tenido que realizar la implantación del diseño creado, podemos obtener datos sobre la utilización de los recursos de la FPGA por nuestro diseño.

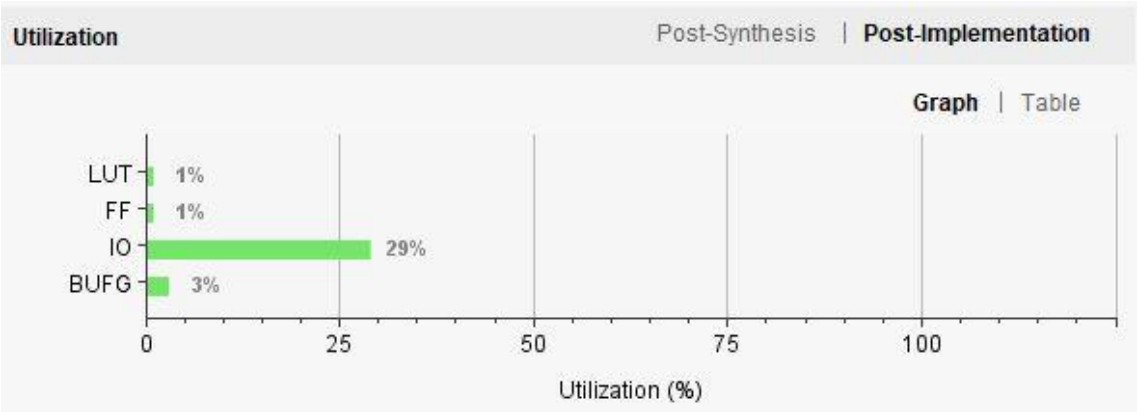
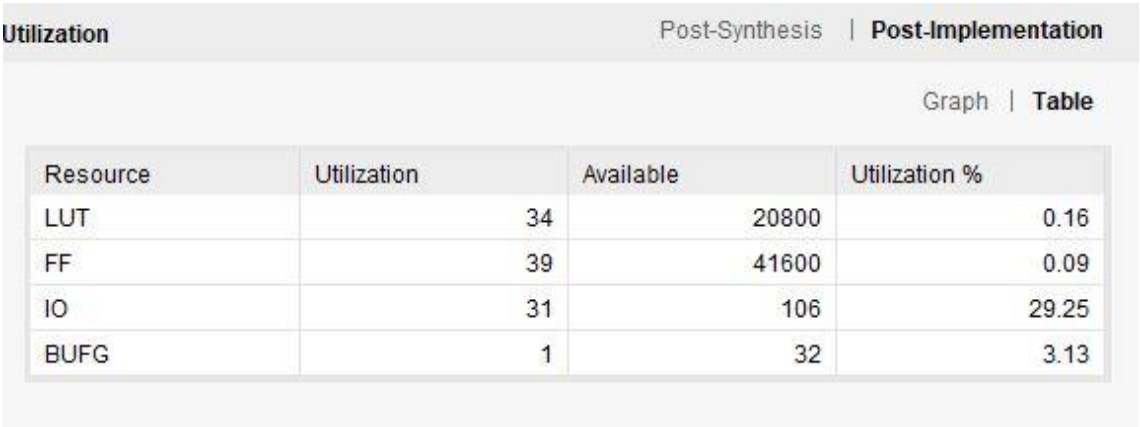


Ilustración 17- Grafica de utilización

En la gráfica de la ilustración anterior, podemos observar que, de la cantidad de recursos disponibles de cada tipo, son los recursos de tipo IO(Entrada y Salida) de los que más se usan, un 29% de los mismos.

En la siguiente tabla observamos, que el recurso más empleado por nuestro proyecto es el de FF(Registros).



Resource	Utilization	Available	Utilization %
LUT	34	20800	0.16
FF	39	41600	0.09
IO	31	106	29.25
BUFG	1	32	3.13

Ilustración 18-Tabla de utilización

CONCLUSIÓN

Tras la realización de este apartado, hemos llegado a una serie de conclusiones principales.

La primera de ellas seria la de no pensar que estamos programando código y tener siempre presente que se trata de modelar un hardware que tiene que poder ser sintetizable e implementable.

La segunda conclusión es la de lo importante que es tener cuidado con el lugar y la forma de emplear las señales para así evitar la inferencia de latches.

Y como última conclusión, pero no menos importante por ello, tendríamos la importancia de realizar un testbench robusto para así poder detectar cualquier pequeño error en nuestro diseño.