PRÁCTICA 10: CONTROL DE APARCAMIENTO

Alumnos: Pablo Menéndez Ruiz de Azúa y Carles Olucha Royo
Fecha de realización: 03/12/2020
Fecha de entrega: 10/12/2020
Asignatura: Sistemas digitales I
Profesores: Julian Spahr y Álvaro Padierna

Índice

Objetivos

El propósito de esta práctica es:

- Comprender el funcionamiento de los contadores
- Diseñar y simular un circuito para controlar el aparcamiento e informar si hay sitios libres o no.
- Aprender a describir circuitos secuenciales con lenguajes de descripción de hardware VHDL.

Introducción

En esta práctica haremos un circuito digital secuencial para implementar un sencillo control de aparcamiento

El circuito contará ascendentemente cuando entre un coche y contará descendentemente cuando salga. Cuando haya sitios libres (contador sea menor al máximo de coches) habrá una luz verde encendida. Sin embargo, cuando no haya sitios libres (contador sea mayor igual al máximo de coches) habrá una luz roja y no dejará entrar coches, solamente dejará salir. Para arrancar el sistema hay que apretar el pulsador que hace de reset.

La entrada y la salida de los coches estará regulada mediante los pulsadores de la propia FPGA, y el número máximo de coches será determinado mediante los interruptores de esta (son 8 bits, por lo que se podrá poner un máximo de 255 coches)

Cálculos previos

Antes de empezar con la práctica, debemos programar todos los componentes que van a formar parte de nuestro circuito. En primer lugar, una máquina que actúe como un detector de flanco de bajada, en segundo lugar, un BinA7Seg y el control de aparcamiento propiamente dicho.

Detector de flanco de bajada

Antes de empezar a programar este bloque, debemos hacernos un diagrama de estados ya que nos va a ayudar a entender cómo funciona y además nos va a facilitar mucho el trabajo a la hora de realizar la programación en VHDL.

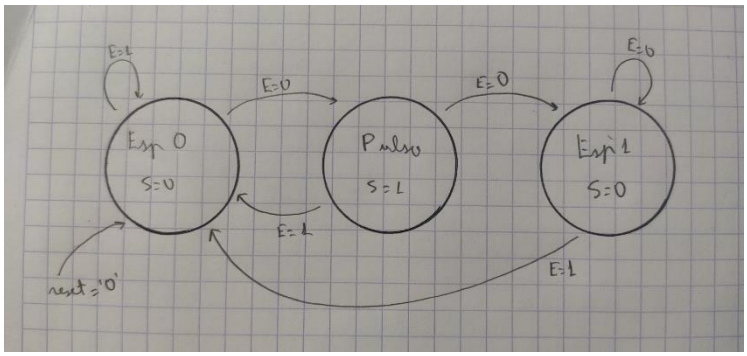


Figura 1 - Diagrama de estados del detector de flanco de bajada

Podemos ver que en este diagrama de estados tenemos tres estados: Esp0, Pulso y Esp1. Inicialmente, si reseteamos el circuito, nuestra entrada vale 1 y por tanto no pasa nada (Estado Esp0). En el momento en el que nuestra entrada vale 0 y el reloj se encuentra en un flanco, cambiamos al estado pulso y nuestra salida vale 1. Este pulso dura un ciclo y vuelve a 0. Si cuando vuelve a 0 la salida, E sigue valiendo 0, esperamos a que se ponga en 1 (Esp1). Por último, si estamos en Pulso o en Esp1 y la entrada vale 1, volvemos al estado inicial.

Cuando vemos un detector de flanco, generalmente vemos que las entradas del diagrama de estados son al revés. Es decir, donde aparecen 1 deberían ser 0 y viceversa. Sin embargo, en esta práctica estamos trabajando con pulsadores que son activos a nivel bajo.
Una vez realizado nuestro diagrama de estados, nos disponemos a programar la máquina en VHDL.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity DetectorFlancobajada is
5     port(
6        e        : in std_logic;
7        reset_n  : in std_logic;
8        clk      : in std_logic;
9        s        : out std_logic
10    );
11 end DetectorFlancobajada;
12
13 architecture behavioral of DetectorFlancobajada is
14    type t_estado is (Esp0,Pulso,Esp1);
15
16    signal estado_act,estado_sig : t_estado;
17
18
19 TransicionEstados: process(estado_act,e)
20 begin
21    estado_sig <=estado_act;
22
23    case estado_act is
24       when Esp0 =>
25          if e = '0' then
26             estado_sig <= Pulso;
27          end if;
28       when Pulso =>
29          if e = '0' then
30             estado_sig <= Esp1;
31          end if;
32          if e = '1' then
33             estado_sig <= Esp0;
34          end if;
35       when Esp1 =>
36          if e = '1' then
37             estado_sig <= Esp0;
38          end if;
39       when others =>
40          estado_sig <=Esp0;
41    end case;
42 end process;
43
44 Salidas: process (estado_act)
45 begin
46
47    s<='0';
48    case estado_act is
49       when Esp0 =>
50          null;
51       when Pulso =>
52          s<='1';
53       when Esp1 =>
54          null;
55       when others =>
56          null;
57    end case;
58 end process;
59
60 VarEstado : process(clk,reset_n)
61 begin
62    if reset_n='0' then
63       estado_act <=Esp0;
64    else
65       if falling_edge(clk) then
66          estado_act <= estado_sig;
67       end if;
68    end if;
69 end process;
70 end behavioral;
```

Figura 2 – Código VHDL del detector de flanco de bajada

Una vez hecho el diagrama de estados, es muy fácil comprender el código. En primer lugar, tenemos que definir nuestros tres estados. Después vamos a programar 3 procesos: el primero para las transiciones entre estados dependiendo de la entrada, el segundo cuál es la salida dependiendo del estado actual y el último cómo se cambia de un estado a otro

BinA7Seg

Este bloque es un decodificador que tiene como función la representación en hexadecimal de un número de 4 bits en un display de 7 segmentos, tal como se muestra en la figura 1. El código VHDL de este bloque lo podemos ver en la figura 2.

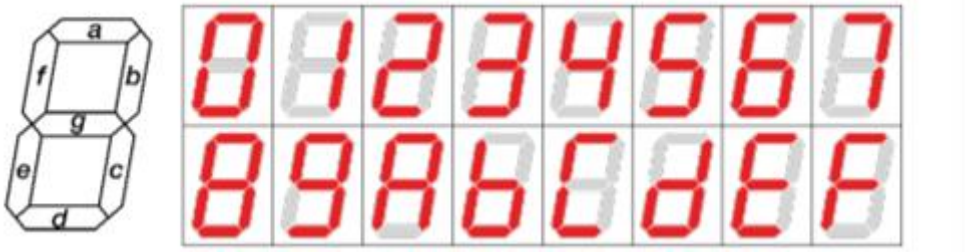


Figura 3 - Representación del 0 al 15 con el componente BinA7Seg

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

entity BinA7Seg is
    port(
        E       :in std_logic_vector(3 downto 0);
        salida  :out std_logic_vector(6 downto 0)
    );
end BinA7Seg;

architecture behavioral of BinA7Seg is

begin
with E select
    salida <=
    "0000001" when "0000",
    "1001111" when "0001",
    "0010010" when "0010",
    "0000110" when "0011",
    "1001100" when "0100",
    "0100100" when "0101",
    "0100000" when "0110",
    "0001111" when "0111",
    "0000000" when "1000",
    "0001100" when "1001",
    "0001000" when "1010",
    "1100000" when "1011",
    "0110001" when "1100",
    "1000010" when "1101",
    "0110000" when "1110",
    "0111000" when "1111",
    "1111111" when others;

end behavioral;
```

Figura 4 - Código VHDL del BinA7Seg

Control Aparcamiento

Este componente es un contador que lo que permite es si entra un coche (hemos pulsado uno de los pulsadores) aumentará el contador en 1 y si sale un coche (hemos pulsado el otro pulsador) disminuirá en uno el número.
Además, hemos añadido las salidas libre y ocupado que representarán las luces de colores (roja y verde) de la placa FPGA.
En este componente hemos tenido en cuenta que si el número de coches es mayor o igual que el máximo de coches permitido no puede permitir entrar más coches y habrá una luz roja.
Si el número de coches es menor al máximo, estará la luz en verde y podrán entrar coches.

Finalmente, para ser más precisos hemos tenido en cuenta que puede entrar un coche y salir otro a la vez (es un caso remoto pero posible). Por eso hemos realizado dos entradas entra_coche y sale_coche en lugar de hacer una ascendente_descendente como se suele hacer.

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

entity ControlAparcamiento is
    port(
    max_coches :    in std_logic_vector(7 downto 0);
    clk :           in std_logic;
    reset_n :       in std_logic;
    entra_coche :   in std_logic;
    sale_coche:     in std_logic;
    num_coches:     out std_logic_vector(7 downto 0);
    libre:          out std_logic;
    ocupado :       out std_logic);
end ControlAparcamiento;

architecture behavioral of ControlAparcamiento is
    signal contador : unsigned(7 downto 0);
    signal entra,sale: std_logic;

    component DetectorFlancoSubida
    port(
      e        : in std_logic;
      reset_n      : in std_logic;
      clk          : in std_logic;
      s        : out std_logic
    );
    end component;

begin
    i1_DetectorFlanco: DetectorFlancoSubida
    port map(
        e      => entra_coche,
      reset_n  => reset_n,
      clk      => clk,
      s     => entra
    );
    i2_DetectorFlanco: DetectorFlancoSubida
    port map(
        e      => sale_coche,
      reset_n  => reset_n,
      clk      => clk,
      s     => sale
    );
    process(clk,reset_n)
    begin
        if reset_n='0' then
            contador <= (others =>'0');
        else
            if rising_edge(clk) then
                if contador < unsigned(max_coches) and contador >"00000000"  then
                    if entra='1' then
                        contador <= contador+1;
                    end if;
                    if sale='1' then
                        contador <= contador-1;
                    end if;
                elsif contador = "00000000" and entra='1' then
                    contador <= contador+1;
                elsif contador = unsigned(max_coches) and sale='1' then
                    contador <= contador-1;
                end if;
            end if;
        end if;
    end process;
    num_coches <= std_logic_vector(contador);
    libre <= '0' when contador=unsigned(max_coches) else '1';
    ocupado <= '1' when contador=unsigned(max_coches) else '0';
end behavioral;
```

Figura 5 - Código VHDL del Contador de Aparcamiento

Diseño del circuito

Nuestro circuito va a estar formado por dos detectores de flanco de bajada para que solamente funcione el circuito cuando se pulse el botón, un control aparcamiento para que cuente los coches y dos componentes BinA7Seg que se encargarán de encender las luces de los displays.

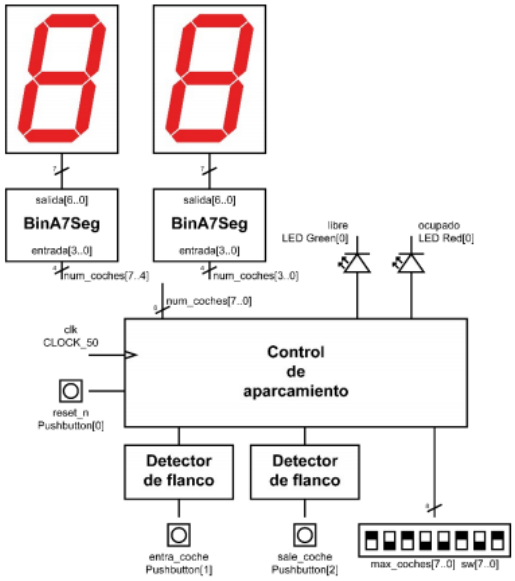


Figura 6 – Diagrama de bloques del circuito

Diseño del circuito en VHDL

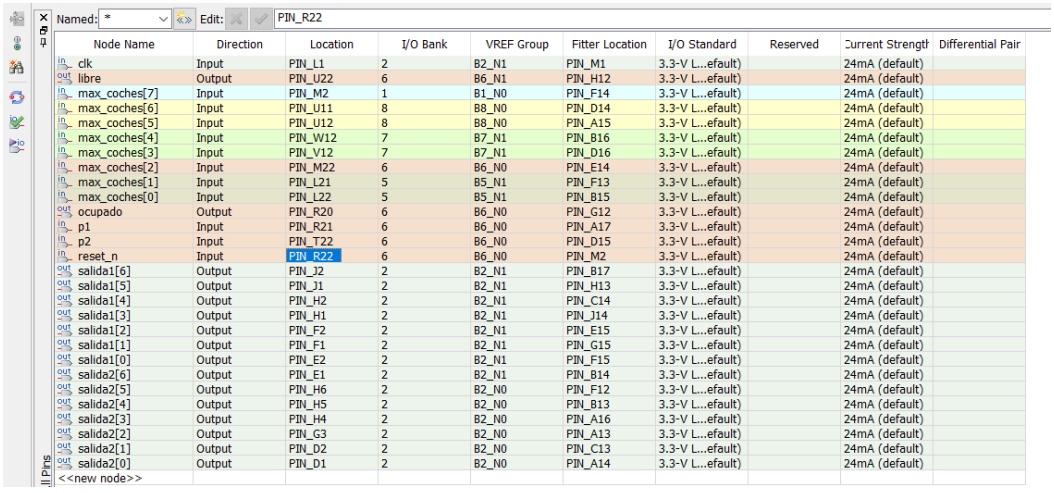
Una vez que tenemos todos los componentes que necesitamos para programar nuestro circuito y entendiendo el diagrama de bloques de la figura 5, podemos escribir el código VHDL de nuestro circuito.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity Practica10 is
6       port(
7           p1          : in std_logic;
8           p2          : in std_logic;
9           reset_n     : in std_logic;
10          clk         : in std_logic;
11          max_coches  : in std_logic_vector(7 downto 0);
12          libre       : out std_logic;
13          ocupado     : out std_logic;
14          salida1     : out std_logic_vector(6 downto 0);
15          salida2     : out std_logic_vector(6 downto 0)
16      );
17  end Practica10;
18
19  architecture structural of Practica10 is
20      signal num_coches: std_logic_vector(7 downto 0);
21      signal entra_coche, sale_coche: std_logic;
22
23      component DetectorFlancoBajada
24      port(
25        e         : in std_logic;
26        reset_n   : in std_logic;
27        clk       : in std_logic;
28        s         : out std_logic
29      );
30      end component;
31
32      component ControlAparcamiento
33      port(
34          max_coches :    in std_logic_vector(7 downto 0);
35          clk :           in std_logic;
36          reset_n :       in std_logic;
37          entra_coche :   in std_logic;
38          sale_coche:     in std_logic;
39          num_coches:     out std_logic_vector(7 downto 0);
40          libre:          out std_logic;
41          ocupado :       out std_logic
42      );
43      end component;
```

```
45      component BinA7Seg
46      port(
47        E        :in std_logic_vector(3 downto 0);
48        salida   :out std_logic_vector(6 downto 0)
49      );
50      end component;
51
52  begin
53
54      i1_DetectorFlanco: DetectorFlancoBajada
55      port map(
56        e      => p1,
57        reset_n  => reset_n,
58        clk      => clk,
59        s      => entra_coche
60      );
61
62      i2_DetectorFlanco: DetectorFlancoBajada
63      port map(
64        e      => p2,
65        reset_n  => reset_n,
66        clk      => clk,
67        s      => sale_coche
68      );
69
70      i_ControlAparcamiento: ControlAparcamiento
71      port map(
72          max_coches  => max_coches,
73          clk         => clk,
74          reset_n     => reset_n,
75          entra_coche => entra_coche,
76          sale_coche  => sale_coche,
77          num_coches  => num_coches,
78          libre       => libre,
79          ocupado     => ocupado
80      );
81
82      i1_BinA7Seg: BinA7Seg
83      port map(
84        E      => num_coches(7 downto 4),
85        salida   => salidal
86      );
88      i2_BinA7Seg: BinA7Seg
89      port map(
90        E      => num_coches(3 downto 0),
91        salida   => salida2
92      );
93  end structural;
```

Figura 7 – Código VHDL del circuito

Finalmente, el último paso antes de realizar la simulación y la implantación física es la asignación de pines.



Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Differential Pair
clk	Input	PIN_L1	2	B2_N1	PIN_M1	3.3-V L...efault)		24mA (default)	
libre	Output	PIN_U22	6	B6_N1	PIN_H12	3.3-V L...efault)		24mA (default)	
max_coches[7]	Input	PIN_M2	1	B1_N0	PIN_F14	3.3-V L...efault)		24mA (default)	
max_coches[6]	Input	PIN_U11	8	B8_N0	PIN_D14	3.3-V L...efault)		24mA (default)	
max_coches[5]	Input	PIN_U12	8	B8_N0	PIN_A15	3.3-V L...efault)		24mA (default)	
max_coches[4]	Input	PIN_W12	7	B7_N1	PIN_B16	3.3-V L...efault)		24mA (default)	
max_coches[3]	Input	PIN_V12	7	B7_N1	PIN_D16	3.3-V L...efault)		24mA (default)	
max_coches[2]	Input	PIN_M22	6	B6_N0	PIN_E14	3.3-V L...efault)		24mA (default)	
max_coches[1]	Input	PIN_L21	5	B5_N1	PIN_F13	3.3-V L...efault)		24mA (default)	
max_coches[0]	Input	PIN_L22	5	B5_N1	PIN_B15	3.3-V L...efault)		24mA (default)	
ocupado	Output	PIN_R20	6	B6_N0	PIN_G12	3.3-V L...efault)		24mA (default)	
p1	Input	PIN_R21	6	B6_N0	PIN_A17	3.3-V L...efault)		24mA (default)	
p2	Input	PIN_T22	6	B6_N0	PIN_D15	3.3-V L...efault)		24mA (default)	
reset_n	Input	PIN_R22	6	B6_N0	PIN_M2	3.3-V L...efault)		24mA (default)	
salida1[6]	Output	PIN_J2	2	B2_N1	PIN_B17	3.3-V L...efault)		24mA (default)	
salida1[5]	Output	PIN_J1	2	B2_N1	PIN_H13	3.3-V L...efault)		24mA (default)	
salida1[4]	Output	PIN_H2	2	B2_N1	PIN_C14	3.3-V L...efault)		24mA (default)	
salida1[3]	Output	PIN_H1	2	B2_N1	PIN_J14	3.3-V L...efault)		24mA (default)	
salida1[2]	Output	PIN_F2	2	B2_N1	PIN_E15	3.3-V L...efault)		24mA (default)	
salida1[1]	Output	PIN_F1	2	B2_N1	PIN_G15	3.3-V L...efault)		24mA (default)	
salida1[0]	Output	PIN_E2	2	B2_N1	PIN_F15	3.3-V L...efault)		24mA (default)	
salida2[6]	Output	PIN_E1	2	B2_N1	PIN_B14	3.3-V L...efault)		24mA (default)	
salida2[5]	Output	PIN_H6	2	B2_N0	PIN_F12	3.3-V L...efault)		24mA (default)	
salida2[4]	Output	PIN_H5	2	B2_N0	PIN_B13	3.3-V L...efault)		24mA (default)	
salida2[3]	Output	PIN_H4	2	B2_N0	PIN_A16	3.3-V L...efault)		24mA (default)	
salida2[2]	Output	PIN_G3	2	B2_N0	PIN_A13	3.3-V L...efault)		24mA (default)	
salida2[1]	Output	PIN_D2	2	B2_N0	PIN_C13	3.3-V L...efault)		24mA (default)	
salida2[0]	Output	PIN_D1	2	B2_N0	PIN_A14	3.3-V L...efault)		24mA (default)	
<<new node>>									

Figura 8 – Asignación de pines

Resultados experimentales

Simulación

Antes de realizar la simulación es necesario crear un archivo con extensión .vht en el que se indique los intervalos de tiempos y cómo cambia cada variable de entrada. En este caso, no vamos a realizar la simulación de todo nuestro circuito, sino que únicamente lo vamos a hacer de nuestro componente principal que es el control de aparcamiento. Además, deberemos tener esto en cuenta a la hora de realizar la simulación ya que no haremos una gate level simulation sino una rtl simulation.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
ENTITY ControlAparcamiento_vhd_tst IS
END ControlAparcamiento_vhd_tst;
ARCHITECTURE ControlAparcamiento_arch OF ControlAparcamiento_vhd_tst
IS
-- constants
-- signals
SIGNAL clk : STD_LOGIC:='0';
SIGNAL entra_coche : STD_LOGIC;
SIGNAL libre : STD_LOGIC;
SIGNAL max_coches : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL num_coches : STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL ocupado : STD_LOGIC;
SIGNAL reset_n : STD_LOGIC;
SIGNAL sale_coche : STD_LOGIC;
COMPONENT ControlAparcamiento
PORT (
 clk : IN STD_LOGIC;
 entra_coche : IN STD_LOGIC;
 libre : OUT STD_LOGIC;
 max_coches : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
 num_coches : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
 ocupado : OUT STD_LOGIC;
 reset_n : IN STD_LOGIC;
 sale_coche : IN STD_LOGIC);
END COMPONENT;
BEGIN
i1 : ControlAparcamiento
PORT MAP (
 -- list connections between master ports and signals
 clk => clk,
 entra_coche => entra_coche,
 libre => libre,
 max_coches => max_coches,
 num_coches => num_coches,
 ocupado => ocupado,
 reset_n => reset_n,
 sale_coche => sale_coche);
init : PROCESS
-- variable declarations
BEGIN
-- code that executes only once
WAIT;
END PROCESS init;
clk <= not clk after 50 ns;
always : PROCESS
BEGIN
 reset_n <= '0';
 entra_coche <= '0';
 sale_coche <= '0';
 max_coches <= X"07";
 wait for 160 ns;
 reset_n <= '1';
 wait for 100 ns;
 for n in 0 to 8 loop
 entra_coche <= '1';
 wait for 500 ns;
 entra_coche <= '0';
 wait for 600 ns;
 end loop;
 assert num_coches = max_coches
 report "Error el contador no satura"
 severity failure;
 assert ocupado <= '1'
 report "Error, la salida ocupado no funciona"
```

```
67      severity failure;
68      for n in 0 to 3 loop
69      sale_coche <= '1';
70      wait for 500 ns;
71      sale_coche <= '0';
72      wait for 600 ns;
73      end loop;
74      wait for 100 ns;
75      assert num_coches = X"03"
76      report "Error: el contador cuenta mal hacia abajo"
77      severity failure;
78      assert ocupado <= '0'
79      report "Error, la salida libre no funciona"
80      severity failure;
81      wait for 100 ns;
82      assert false
83      report "Fin de la simulacion"
84      severity failure;
85      WAIT;
86  END PROCESS always;
87  END ControlAparcamiento_arch;
```

Figura 9 - Código de la simulación

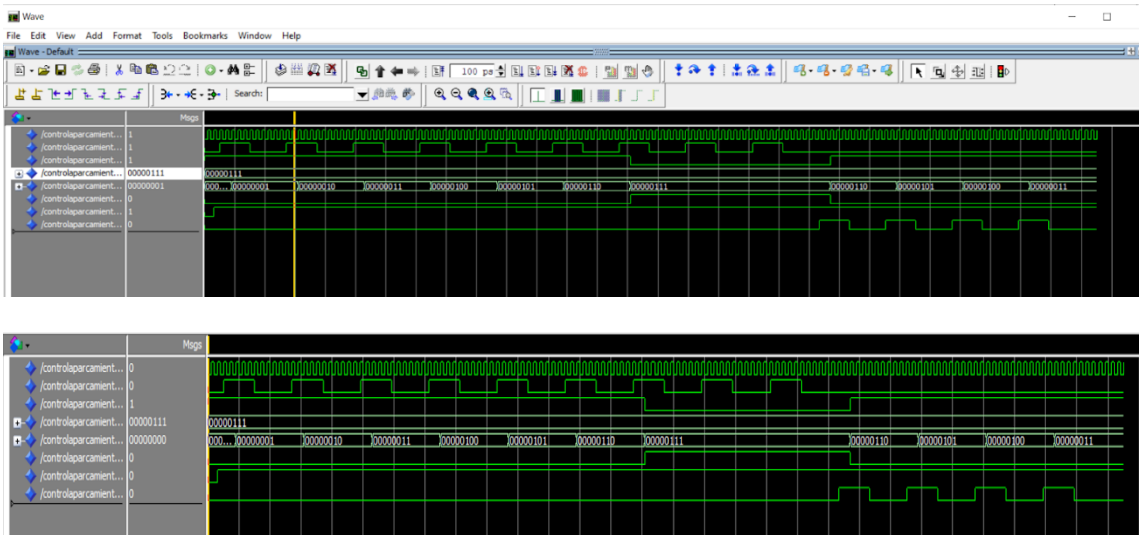


Figura 10 - Simulación

Implantación física

Tras haber realizado la simulación y haber comprobado que todo funcionaba correctamente, ya podemos implementar nuestro esquema en la placa y comprobar si nuestro esquema realiza la acción para la que lo hemos diseñado. En la placa hemos asignado el número máximo de coches mediante los 8 interruptores. Las entradas y salidas del circuito serán asignadas mediante dos pulsadores y la luz verde y roja determinará si hay sitio para aparcar o no. Además, el número de coches que hay en el aparcamiento se verá siempre en el display de números. En nuestro caso, cometimos un pequeño error a la hora de asignar los displays, ya que sería más visible que los bits menos significativos estuvieran en el display de la derecha y los menos a la izquierda pero nos equivocamos a la hora de asignar los pines.
Los resultados obtenidos han sido los siguientes:

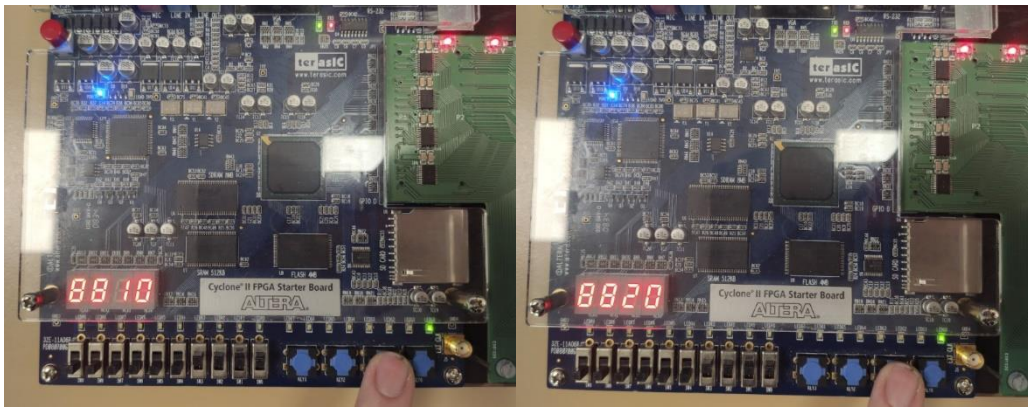


Figura 11 – Entrando coches

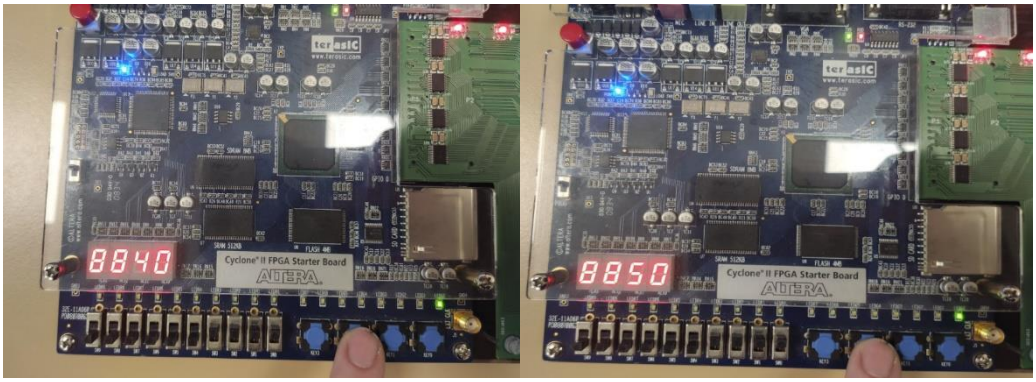


Figura 12- Saliendo coches

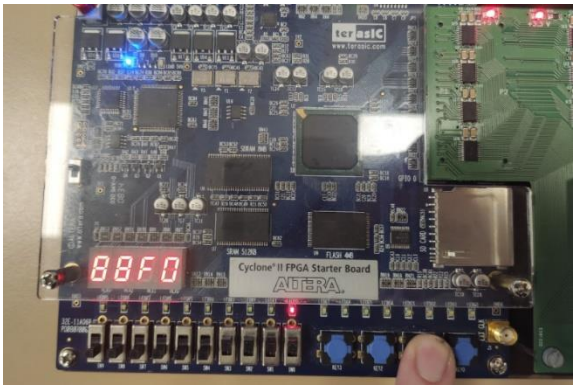


Figura 13 – Máximo de coches alcanzado (no suma más)

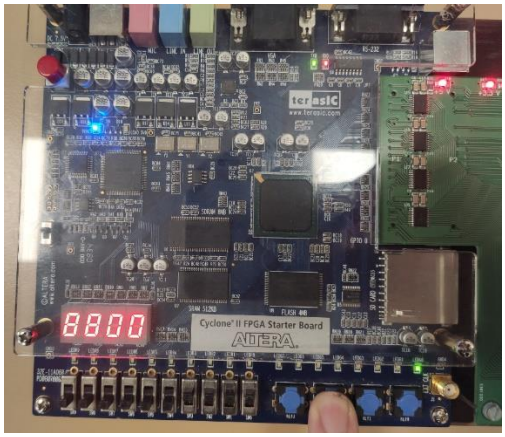


Figura 14 – Mínimo de coches alcanzado (no resta más)

Conclusiones y análisis de resultados

Antes de empezar esta práctica nos fijamos unos objetivos y ahora que hemos llegado al final de esta, debemos analizar si hemos cumplido esos objetivos.

El primero de estos objetivos era el de comprender el funcionamiento de los contadores. Es cierto, que el contador que hemos programado era parecido a alguno que ya habíamos estudiado. Sin embargo, había una serie de detalles que diferenciaban a este contador de otros como el detector de flanco o que te permitía tanto restar como sumar dependiendo de la entrada. Por lo tanto, programar este contador nos ha servido para profundizar más en los contadores y comprender mejor su funcionamiento.

Otro objetivo que nos hemos fijado al principio de esta práctica era el de diseñar y simular un control de aparcamiento. Este objetivo lo hemos cumplido porque hemos terminado la práctica y nuestro código ha funcionado correctamente tanto en la simulación como en la placa.

El último de los objetivos que nos habíamos fijado era el de escribir circuitos secuenciales utilizando VHDL. Como se puede observar, hemos utilizado procesos que dependían de un reloj en todos nuestros diseños, tanto en el detector de flanco como en el contador. Por lo tanto, hemos aprendido y dominado como describir circuitos secuenciales con VHDL.