Universidad de Buenos Aires
Facultad de Ingeniería

Sistemas digitales (86.41)

Implementación de un sistema digital

en VHDL

Integrantes:

Puy Gonzalo 99784 gpuy@fi.uba.ar

1 er cuatrimestre 2022

1. Introducción

En el siguiente trabajo se implementará un circuito para controlar dos semáforos en un cruce de calles. Dicho circuito sera descripto en lenguaje VHDL, simulado y sintetizado sobre el dispositivo `xc7a15tftg256-1`.

2. Desarrollo

2.1. Especificación del diseño

El circuito a implementar debe controlar dos semáforos en un cruce de calles. Dispone de 6 salidas, las cuales indican el estado de las luces de los semáforos (Rojo, Amarillo y verde).

El tiempo en que los semáforos están en amarillo sera de 3 segundos, mientras que en rojo y verde sera de 30 segundos. Y por ultimo, el sistema cuenta con una frecuencia de operación de 50 MHz.

2.2. Estados del sistema

AL ser un cruce de calles y tener 2 semáforos, se planteó estados del sistema para que tenga coherencia con la implementación real. Dichos estados son

- E_0: `R1_V2` - Semáforo 1 en rojo, semáforo 2 en verde (estado inicial)
- E_1: `R1_A2` - Semáforo 1 en rojo, semáforo 2 en amarillo
- E_2: `A1_R2` - Semáforo 1 en amarillo, semáforo 2 en rojo
- E_3: `V1_R2` - Semáforo 1 en verde, semáforo 2 en rojo
- E_4: `A1_R2_p` - Semáforo 1 en amarillo, semáforo 2 en rojo
- E_5: `R1_A_p` - Semáforo 1 en rojo, semáforo 2 en amarillo

Notar que E_1 y E_2 son similares a E_5 y E_4 respectivamente, pero se los pone en estados diferentes porque provienen de estados diferentes. En E_4 el semáforo 1 viene de estar en verde mientras que en E_2 viene de estar en rojo.

Para mostrar los diferentes estados del sistema de una forma mas visual, se realizó el siguiente diagrama de estados

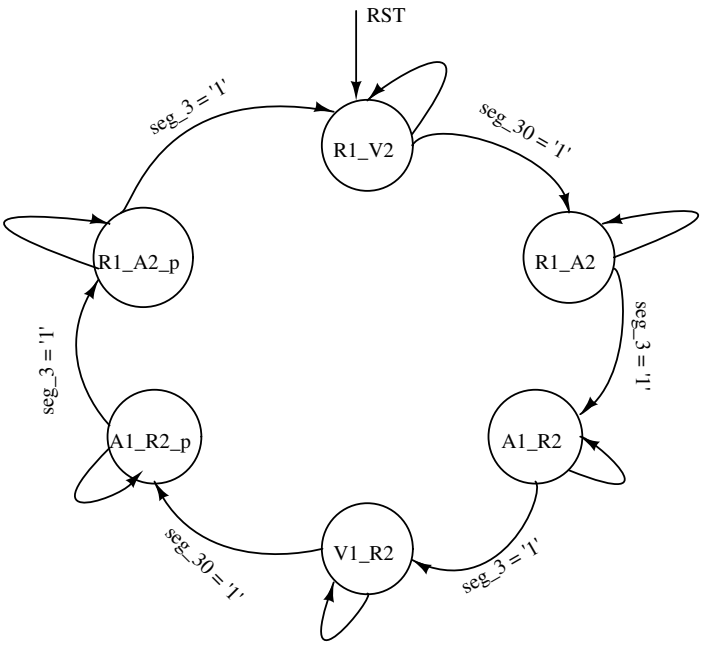

Figura 1: Diagrama de estados

2.3. Diseño propuesto

El siguiente esquema ilustra el circuito que será descripto en VHDL. Donde FSM es *Finite State Machine*.

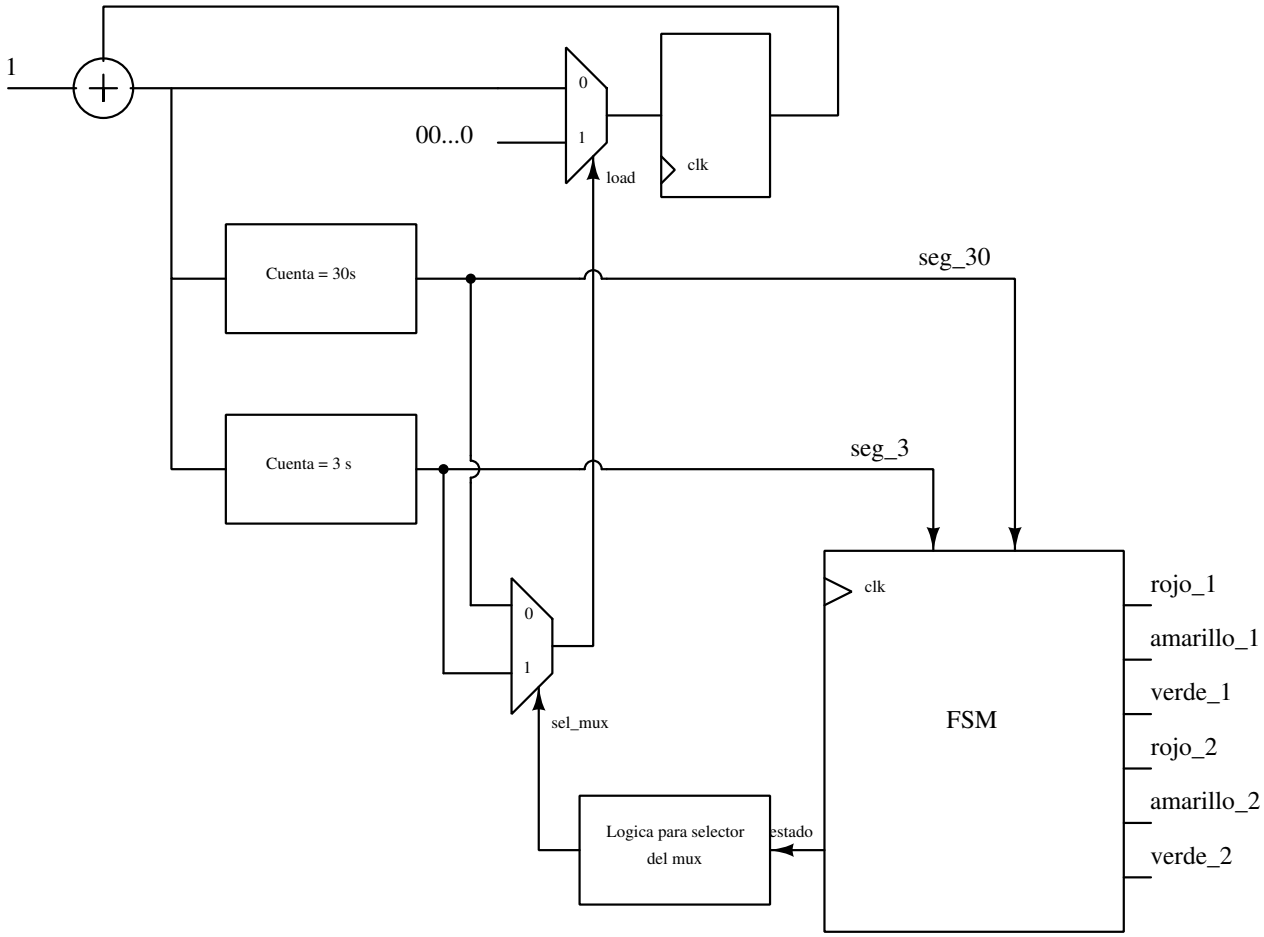

Figura 2

Esta implementación esta compuesta por 1 contador de 31 bits con carga, que se selecciono en base a la frecuencia de trabajo y la cantidad de tiempo en la que deben permanecer los diferentes estados. Para el caso de nuestro trabajo, se tiene una frecuencia de 50 MHz, por lo que sabemos que el contador tendrá que contar hasta

$$30\,\mathrm{s} \cdot 50\,\mathrm{MHz} = 1500000000$$

para 30 segundos. Esto es, contar desde 0 hasta 1499999999. Para 3 segundos:

$$3\,\mathrm{s} \cdot 50\,\mathrm{MHz} = 150000000$$

Es decir, contar desde 0 hasta 149999999. Por lo tanto, el contador debe ser de $\log_2(1500000000) = 30{,}48 \approx 31$ bits.

El contador además tiene una lógica que compara la cuenta actual con los números calculados anteriormente, si estas comparaciones son verdaderas pondrá en 1 la salida `seg_30` o `seg_3` según corresponda.

Con el multiplexor lo que se hace es aprovechar al contador con carga para reiniciar la cuenta, con esto nos ahorramos el hecho de tener que hacer 2 contadores (uno para 30 segundos y otro para 3 segundos).

Por otro lado, la maquina de estados recibe estas señales `30_seg` y `3_seg`, con las cuales realizará la lógica para el cambio de estados. En base a los estados, se colocaran las respectivas salidas y el valor correspondiente al selector del multiplexor que hará que se reinicie o no la cuenta.

Por ultimo, se aclara que tanto el contador como la maquina de estados son circuitos sincrónicos, con el mismo dominio de reloj. El camino del reloj no dispone de ninguna lógica y solo se utiliza el flanco ascendente.

Además cuentan con un solo *reset* asincrónico, que tampoco dispone de ninguna lógica en su camino y se utiliza solo el nivel alto (`'1'`) del mismo.

3. Implementación en VHDL

El código del circuito descripto en la sección anterior, se implementó en VHDL y se presenta a continuación

```vhdl
-- TP 1
-- Materia: Sistemas digitales
-- Alumno: Gonzalo Puy
-- Padron: 99784

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;


entity semaforos is
    port(
        rst         : in std_logic;
        clk         : in std_logic;
        rojo_1      : out std_logic;
        amarillo_1  : out std_logic;
        verde_1     : out std_logic;
        rojo_2      : out std_logic;
        amarillo_2  : out std_logic;
        verde_2     : out std_logic
    );
end semaforos;

architecture behavioral of semaforos is

    -- Definicion de constantes
    constant N_counter : natural := 31;

    -- Definicion del tipo de dato "t_state"
    type t_state is (R1_V2, R1_A2, A1_R2, V1_R2, A1_R2_p, R1_A2_p);

    -- Definicion de señales a usar

```

```
34      signal state     : t_state;
35      signal value     : std_logic_vector(N_counter-1 downto 0) := (others => '0');
36      signal mux_out   : std_logic;
37      signal sel_mux   : std_logic;
38      signal seg_30    : std_logic;
39      signal seg_3     : std_logic;
40
41  begin
42
43      -- Componentes
44      mux: entity work.mux
45      port map(
46          x0 => seg_30,
47          x1 => seg_3,
48          s  => sel_mux,
49          y  => mux_out
50      );
51
52      contador: entity work.counterN
53      generic map(N => N_counter)
54      port map(
55          rst => rst,
56          clk => clk,
57          load => mux_out,
58          value => value,
59          count => open,
60          seg_30_count => seg_30,
61          seg_3_count => seg_3
62      );
63
64      fsm: process(clk, rst) -- Maquina de estados
65      begin
66          if rst = '1' then
67              state <= R1_V2;
68
69          elsif clk = '1' and clk'event then
70              case state is
71
72                  when R1_V2 =>
73                      if seg_30 = '1' then
74                          state <= R1_A2;
75                      end if;
76
77                  when R1_A2 =>
78                      if seg_3 = '1' then
79                          state <= A1_R2;
80                      end if;
81
82                  when A1_R2 =>
```

```
83                      if seg_3 = '1' then
84                          state <= V1_R2;
85                      end if;
86
87                  when V1_R2 =>
88                      if seg_30 = '1' then
89                          state <= A1_R2_p;
90                      end if;
91
92                  when A1_R2_p =>
93                      if seg_3 = '1' then
94                          state <= R1_A2_p;
95                      end if;
96
97                  when R1_A2_p =>
98                      if seg_3 = '1' then
99                          state <= R1_V2;
100                     end if;
101             end case;
102         end if;
103     end process;
104
105     -- Selector del mux:
106     sel_mux <=
107                 '1' when ((state = R1_A2) or (state = R1_A2_p) or (state = A1_R2) or (state = A1_R2_p))
108                 '0';
109
110     -- Salidas:
111     rojo_1 <=
112             '1' when ((state = R1_V2) or (state = R1_A2) or (state = R1_A2_p)) else
113             '0';
114
115     amarillo_1 <=
116                 '1' when ((state = A1_R2) or (state = A1_R2_p)) else
117                 '0';
118
119     verde_1 <=
120              '1' when state = V1_R2 else
121              '0';
122
123     rojo_2 <=
124             '1' when ((state = A1_R2) or (state = V1_R2) or (state = A1_R2_p)) else
125             '0';
126
127     amarillo_2 <=
128                 '1' when ((state = R1_A2) or (state = R1_A2_p)) else
129                 '0';
130
131     verde_2 <=
```

```
132             '1' when state = R1_V2 else
133             '0';
134 
135 end behavioral;
```

El contador y el multiplexor se implementaron en archivos aparte,

```
1  -- TP 1
2  -- Materia: Sistemas digitales
3  -- Alumno: Gonzalo Puy
4  -- Padron: 99784
5  
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8  use IEEE.numeric_std.all;
9  
10 -- Contador generico de N bits con señal del carga
11 entity counterN is
12     generic(
13         N : natural := 8
14     );
15     port(
16         rst             : in std_logic;
17         clk             : in std_logic;
18         load            : in std_logic;
19         value           : in std_logic_vector(N-1 downto 0);
20         count           : out std_logic_vector(N-1 downto 0);
21         seg_30_count    : out std_logic;
22         seg_3_count     : out std_logic
23     );
24 end counterN;
25 
26 architecture behavioral of counterN is
27 
28     constant N30_SEG : natural := 1499999999;
29     constant N3_SEG : natural := 149999999;
30 
31     signal  aux_count : unsigned(N-1 downto 0);
32 
33 begin
34 
35     process(clk,rst)
36     begin
37         if rst = '1' then
38             aux_count <= (others => '0');
39         elsif clk = '1' and clk'event then
40             if load = '1' then
41                 aux_count <= unsigned(value);
```

```
42              else
43                  aux_count <= aux_count + 1;
44              end if;
45          end if;
46      end process;
47
48      count <= std_logic_vector(aux_count);
49
50      seg_30_count <= '1' when (aux_count = N30_SEG) else '0';
51
52      seg_3_count <= '1' when (aux_count = N3_SEG) else '0';
53
54
55  end behavioral;
```

```
1   -- TP 1
2   -- Materia: Sistemas digitales
3   -- Alumno: Gonzalo Puy
4   -- Padron: 99784
5
6   library IEEE;
7   use IEEE.std_logic_1164.all;
8   use IEEE.numeric_std.all;
9
10  entity mux is
11      port(
12          x0  : in std_logic; -- Entrada 0 mux
13          x1  : in std_logic; -- Entrada 1 mux
14          s   : in std_logic; -- Selectro mux
15          y   : out std_logic -- Salida
16      );
17  end mux;
18
19  architecture behavioral of mux is
20  begin
21      process(x0,x1,s)
22      begin
23          case s is
24              when '0' =>
25                  y <= x0;
26
27              when others =>
28                  y <= x1;
29          end case;
30      end process;
31  end behavioral;
```

4. Simulación

Simular el circuito descripto es problemático por la cantidad de tiempo que hay que simular para ver el correcto funcionamiento y todos los estados. Por lo tanto se hizo un escalaje en los números que se comparan con el contador para que se puedan apreciar todos los estados en una simulación de menor duración. Por lo tanto se reemplazo $N30\_SEG = 1499999999$ con $N30\_SEG = 149$ y $N3\_SEG = 149999999$ con $N3\_SEG = 14$. En base a esto se realizó el siguiente banco de pruebas o *test bench*

```
1  -- TP 1
2  -- Materia: Sistemas digitales
3  -- Alumno: Gonzalo Puy
4  -- Padron: 99784
5
6  library IEEE;
7  use IEEE.std_logic_1164.all;
8  use IEEE.numeric_std.all;
9
10 entity tb_semaforos is
11 end tb_semaforos;
12
13 architecture behavioral of tb_semaforos is
14
15     constant SIM_TIME : time := 10000 ns;
16     constant N_TB : natural := 31;
17
18     signal tb_rst           : std_logic;
19     signal tb_clk           : std_logic := '0';
20     signal tb_rojo_1        : std_logic;
21     signal tb_amarillo_1    : std_logic;
22     signal tb_verde_1       : std_logic;
23     signal tb_rojo_2        : std_logic;
24     signal tb_amarillo_2    : std_logic;
25     signal tb_verde_2       : std_logic;
26
27 begin
28
29     tb_rst <= '0', '1' after 1 ns, '0' after 20 ns;
30     tb_clk <= not tb_clk after 10 ns; -- Clock con freq : 50 MHz
31
32     stop_simulation : process
33     begin
34         wait for SIM_TIME;
35         assert false
36             report "Simulacion terminada."
37             severity failure;
38     end process;
39
40     I1: entity work.semaforos(behavioral)
```

```
41      port map(
42          rst         => tb_rst,
43          clk         => tb_clk,
44          rojo_1      => tb_rojo_1,
45          amarillo_1  => tb_amarillo_1,
46          verde_1     => tb_verde_1,
47          rojo_2      => tb_rojo_2,
48          amarillo_2  => tb_amarillo_2,
49          verde_2     => tb_verde_2
50      );
51
52  end behavioral;
```

Los resultados de la simulación se muestran a continuación

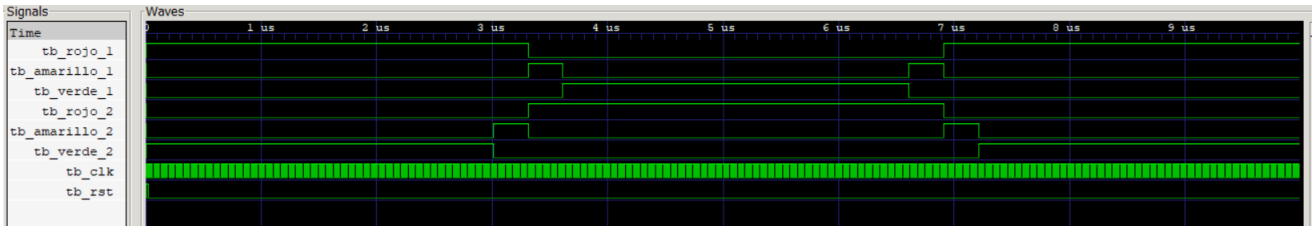

Figura 3

La figura 3 muestra la señal de clock, reset y las 6 salidas del sistema. Con estas ultimas se puede apreciar todos los estados de la *fsm*.

Para mostrar el correcto funcionamiento también se decidió mostrar las señales de cuenta, `seg_30`, `seg_3` y `sel_mux` como se puede apreciar en la figura 4.

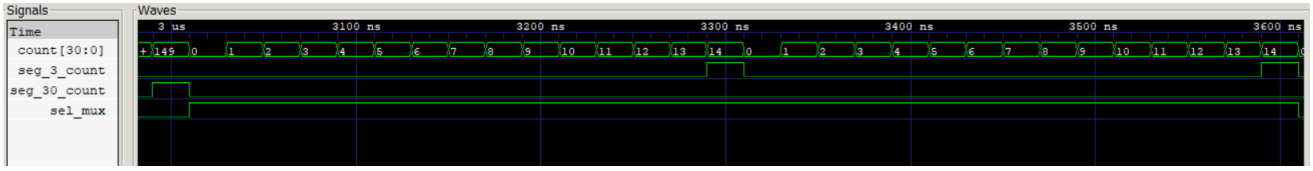

Figura 4

La captura de la figura 4 se realizo en el cambio del primer al segundo y del segundo al tercer estado. Se puede apreciar como se reinicia el clock al terminar de contar los '30 segundos' y luego de los '3 segundos' necesarios para cambiar del segundo al tercer estado.

5. Síntesis

Para esta sección, se realizo una síntesis sobre el dispositivo FPGA `xc7a15tftg256-1` mediante el *software* Vivado. En esta herramienta, se muestra como resulta el circuito descripto mediante VHDL implementado en el dispositivo.

En especial, es interesante observar el esquemático RTL (es decir, la abstracción del circuito descripto). Dicho esquemático se puede ver en la figura 5.

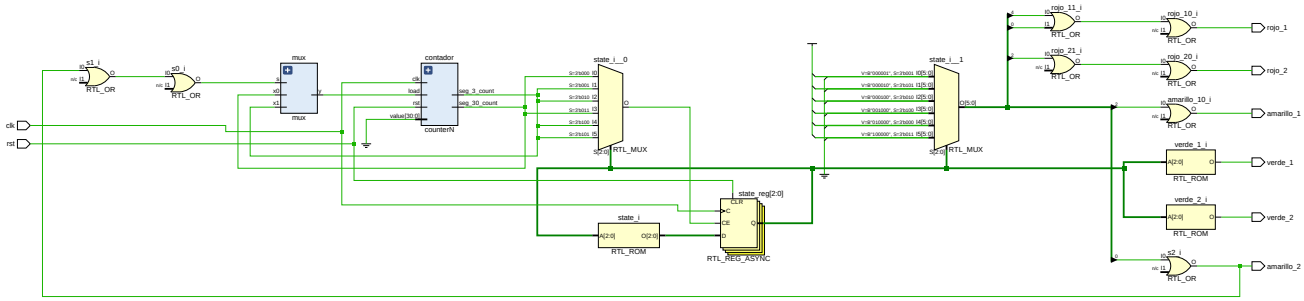

Figura 5: Esquemático RTL

Otro de los resultados de la síntesis, es lo que se llama esquemático de implementación. Este muestra el conexionado interno que se realiza en la FPGA. Dicho esquemático se muestra a continuación

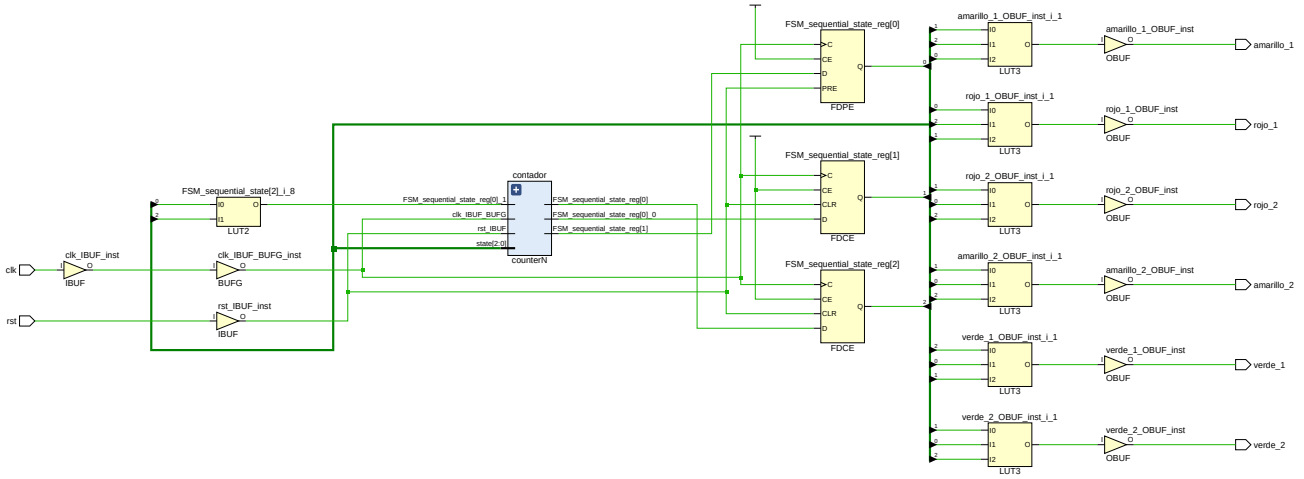

Figura 6: Esquemático de Implementación

Por ultimo resulta de interés, ver el resumen de recursos utilizados en el diseño. El cual muestra la cantidad utilizada de Flip-Flops, LUT y puertos IO que se serian necesarios en la implementación en el dispositivo. Este resumen y el resumen de tiempos se muestra en las figuras 7 y 8 respectivamente.

Resource	Utilization	Available	Utilization %
LUT	49	10400	0.47
FF	34	20800	0.16
IO	8	170	4.71

Figura 7: Resumen de los recursos utilizados en el dispositivo `xc7a15tftg256-1`

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	74	Total Number of Endpoints:	74	Total Number of Endpoints:	NA

There are no user specified timing constraints.

Figura 8: Resumen de tiempos

6. Conclusiones

Como análisis final, se puede acotar que el trabajo permitió no solo dar una buena introducción al lenguaje de descripción de hardware VHDL, sino también poder llevar un diseñó digital y sincrónico totalmente abstracto, a algo real.

Otro aspecto importante tratado en este trabajo fue la síntesis realizada mediante el *software* Vivado, que permite acercarnos al dispositivo FPGA y poder observar como quedaría nuestra implementación en este.