

# Trabajo Práctico 1 — Implementación de un Sistema Secuencial en VHDL

[6617] Sistemas Digitales  
Primer cuatrimestre de 2024

---

Padrón	Alumna	Dirección de correo
101456	Pérez Andrade, Violeta	<a href="mailto:viperez@fi.uba.ar">viperez@fi.uba.ar</a>

---

# Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Desarrollo</b>	<b>2</b>
2.1. Semáforos FSM . . . . .	3
2.2. Counter Dynamic . . . . .	3
2.3. Counter Time . . . . .	3
2.4. Semáforos - Top Level . . . . .	3
<b>3. Simulación</b>	<b>3</b>
<b>4. Síntesis</b>	<b>3</b>
<b>5. Anexo</b>	<b>5</b>
5.1. Código . . . . .	5

## 1. Introducción

El presente trabajo práctico tiene como objetivo fijar el concepto de circuito secuencial sincrónico aplicando el lenguaje de descripción de hardware VHDL. Para esto, se implementó un circuito para controlar dos semáforos en un cruce de calles.

## 2. Desarrollo

El circuito tiene seis posibles salidas, dependiendo del estado en el que se encuentre. En la siguiente figura en cada fila se puede ver para cada estado el color de cada semáforo.













S0		
S1		
S2		
S3		
S4		
S5		

Figura 1: Tabla de estados

El tiempo en amarillo será de 3 segundos mientras que en rojo y verde será de 30. A partir de esto, el sistema puede ser representado mediante el siguiente diagrama de estados

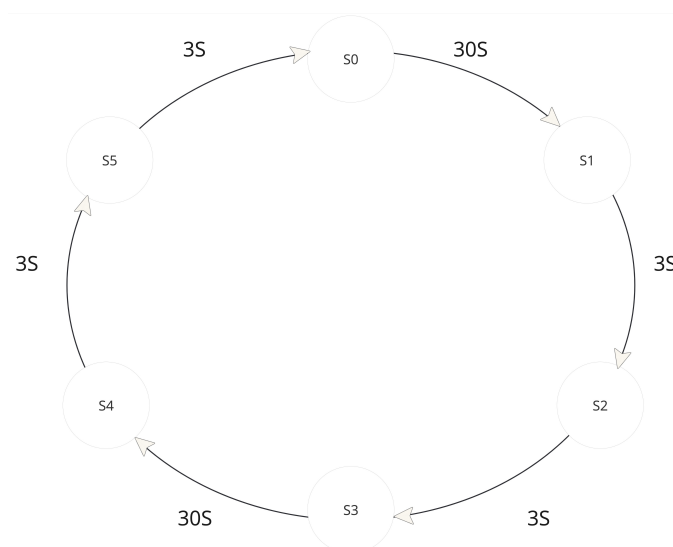


Figura 2: Diagrama de estados

Para esto fueron desarrollados tres módulos principales

## 2.1. Semáforos FSM

Este módulo es responsable de controlar los estados del sistema de semáforos. Implementa una máquina de estados finitos (FSM) con seis estados diferentes, como se muestra en la Figura 1. Cada estado corresponde a una combinación específica de luces en los dos semáforos.

## 2.2. Counter Dynamic

Este módulo es un contador binario asincrónico que cuenta hasta un valor máximo. Cuando el contador alcanza su valor máximo, envía una señal indicando que se completó un ciclo de conteo.

## 2.3. Counter Time

También es un contador pero cuenta eventos basados en el tiempo. Usa el período del reloj y el período de tiempo especificado para determinar cuándo incrementar el contador. Calcula cuantos ciclos de reloj se deben contar a partir del tiempo dado, y usa el modulo *counter dynamic* para contarlos.

## 2.4. Semaforos - Top Level

En este modulo se instancia al *semaforo\_fsm* además de dos contadores, uno para controlar el tiempo de duración del estado de 30 segundos y otro para controlar el tiempo de duración del estado de 3 segundos. El semáforo tiene dos señales de salida que son leídas por cada contador y les indica cuando debe empezar a contar. Además de dos señales de entrada enviadas por los contadores dando aviso de que el conteo finalizó y a partir de eso efectúa el cambio de estado.

## 3. Simulación

A continuación se puede ver una simulación del sistema, la misma se realizó usando la herramienta *GHDL*. Para poder apreciar varios estados, se utilizó un valor de frecuencia de clock menor (10 Hz).



Figura 3: Simulación

## 4. Síntesis

Se realizó la síntesis sobre el dispositivo *FPGA xc7a15tfg256-1* con el software Vivado, se puede ver el esquemático a continuación con los módulos, entradas y salidas descriptos en la sección 2.

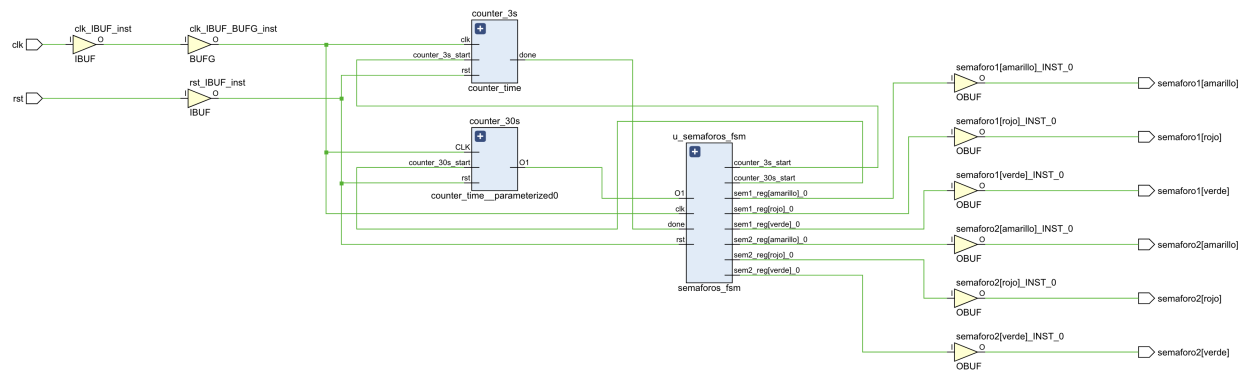


Figura 4: Síntesis

## 5. Anexo

Se muestra a continuación el código desarrollado

### 5.1. Código

Top level:

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use work.tp1_pkg.all;
4
5 entity semaforos is
6     generic (
7         constant CLK_PERIOD : time := 20 ns
8     );
9     port(
10         rst           : in std_logic;
11         clk           : in std_logic;
12         semaforo1     : out semaforo;
13         semaforo2     : out semaforo
14     );
15 end semaforos;
16
17 architecture behavioral of semaforos is
18     signal counter_30s_end, counter_3s_end : std_logic;
19     signal counter_30s_start, counter_3s_start : std_logic;
20
21     signal estado : t_semaforo_state;
22 begin
23
24     counter_3s : entity work.counter_time
25         generic map (
26             CLK_PERIOD => CLK_PERIOD,
27             COUNTER_PERIOD => 3 sec
28         )
29         port map (
30             rst => rst,
31             start => counter_3s_start,
32             done => counter_3s_end,
33             clk => clk
34         );
35
36     counter_30s : entity work.counter_time
37         generic map (
38             CLK_PERIOD => CLK_PERIOD,
39             COUNTER_PERIOD => 30 sec
40         )
41         port map (
42             rst => rst,
43             start => counter_30s_start,
44             done => counter_30s_end,
45             clk => clk
46         );
47
48     u_semaforos_fsm : entity work.semaforos_fsm
49         port map(
50             rst => rst,
51             clk => clk,
```

```

52         state => estado,
53         semaforo1 => semaforo1,
54         counter_30s_start => counter_30s_start,
55         counter_3s_start => counter_3s_start,
56         counter_30s_end => counter_30s_end,
57         counter_3s_end => counter_3s_end,
58         semaforo2 => semaforo2
59     );
60
61 end behavioral;

```

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  package tp1_pkg is
5      type t_semaforo_state is (S0, S1, S2, S3, S4, S5);
6
7      type semaforo is record
8          verde      : std_logic;
9          rojo       : std_logic;
10         amarillo: std_logic;
11     end record;
12
13 end tp1_pkg;

```

Semaforos\_FSM:

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use work.tp1_pkg.all;
4
5  entity semaforos_fsm is
6      port(
7          rst           : in std_logic;
8          clk           : in std_logic;
9          counter_3s_start: out std_logic;
10         counter_30s_start: out std_logic;
11         counter_3s_end: in std_logic;
12         counter_30s_end: in std_logic;
13         state          : out t_semaforo_state;
14         semaforo1      : out semaforo;
15         semaforo2      : out semaforo
16     );
17 end semaforos_fsm;
18
19 architecture behavioral of semaforos_fsm is
20     signal sem1, sem2 : semaforo;
21 begin
22     process(clk, rst)
23     begin
24         if rst='1' then
25             state <= S0;
26             counter_30s_start <= '1';
27         elsif clk = '1' and clk'event then
28             counter_30s_start <= '0';
29             counter_3s_start <= '0';
30             state <= state;
31             case state is
32                 when S0 =>

```

```
33         if (counter_30s_end) then
34             state <= S1;
35             counter_3s_start <= '1';
36         end if;
37     when S1 =>
38         if (counter_3s_end) then
39             state <= S2;
40             counter_3s_start <= '1';
41         end if;
42     when S2 =>
43         if (counter_3s_end) then
44             state <= S3;
45             counter_30s_start <= '1';
46         end if;
47     when S3 =>
48         if (counter_30s_end) then
49             state <= S4;
50             counter_3s_start <= '1';
51         end if;
52     when S4 =>
53         if (counter_3s_end) then
54             state <= S5;
55             counter_3s_start <= '1';
56         end if;
57     when S5 =>
58         if (counter_3s_end) then
59             state <= S0;
60             counter_30s_start <= '1';
61         end if;
62     end case;
63
64     sem1 <= (verde => '0', rojo => '0', amarillo => '0');
65     sem2 <= (verde => '0', rojo => '0', amarillo => '0');
66
67     case state is
68     when S0 =>
69         sem1.verde <= '1';
70         sem1.rojo <= '0';
71         sem1.amarillo <= '0';
72         sem2.verde <= '0';
73         sem2.rojo <= '1';
74         sem2.amarillo <= '0';
75     when S1 =>
76         sem1.verde <= '0';
77         sem1.rojo <= '0';
78         sem1.amarillo <= '1';
79         sem2.verde <= '0';
80         sem2.rojo <= '1';
81         sem2.amarillo <= '0';
82     when S2 =>
83         sem1.verde <= '0';
84         sem1.rojo <= '1';
85         sem1.amarillo <= '0';
86         sem2.verde <= '0';
87         sem2.rojo <= '0';
88         sem2.amarillo <= '1';
89     when S3 =>
90         sem1.verde <= '0';
```



```

91         sem1.rojo      <= '1';
92         sem1.amarillo <= '0';
93         sem2.verde     <= '1';
94         sem2.rojo      <= '0';
95         sem2.amarillo <= '0';
96         when S4 =>
97             sem1.verde  <= '0';
98             sem1.rojo   <= '1';
99             sem1.amarillo <= '0';
100            sem2.verde  <= '0';
101            sem2.rojo   <= '0';
102            sem2.amarillo <= '1';
103            when S5 =>
104                sem1.verde  <= '0';
105                sem1.rojo   <= '0';
106                sem1.amarillo <= '1';
107                sem2.verde  <= '0';
108                sem2.rojo   <= '1';
109                sem2.amarillo <= '0';
110            end case;
111        end if;
112    end process;
113
114    semaforo1 <= sem1;
115    semaforo2 <= sem2;
116
117 end behavioral;

```

Contadores:

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.math_real.all;
5
6  entity counter_dynamic is
7      generic (
8          constant COUNTS : natural
9      );
10     port(
11         rst : in std_logic;
12         clk : in std_logic;
13         enable : in std_logic;
14         done : out std_logic
15     );
16 end counter_dynamic;
17
18 architecture behavioral of counter_dynamic is
19     constant N : integer := integer(ceil(log2(real(COUNTS))));
20     signal counter : unsigned(N-1 downto 0);
21 begin
22     process(clk,rst)
23     begin
24         if rst='1' then
25             counter <= (others => '0');
26             done <= '0';
27         elsif clk = '1' and clk'event then
28             if (enable) then
29                 if counter = COUNTS-1 then

```

```

30         counter <= (others => '0');
31         done <= '1';
32     else
33         counter <= counter + 1;
34         done <= '0';
35     end if;
36 end if;
37 end if;
38 end process;
39 end behavioral;

```

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use ieee.math_real.all;
5
6  entity counter_time is
7      generic (
8          constant CLK_PERIOD : time;
9          constant COUNTER_PERIOD : time
10     );
11     port(
12         rst : in std_logic;
13         clk : in std_logic;
14         start : in std_logic;
15         done : out std_logic
16     );
17 end counter_time;
18
19 architecture behavioral of counter_time is
20     constant COUNTS : natural := COUNTER_PERIOD/CLK_PERIOD;
21     signal counting : std_logic;
22 begin
23     process (clk,rst) begin
24         if rst = '1' then
25             counting <= '0';
26         elsif clk = '1' and clk'event then
27             if start then
28                 counting <= '1';
29             elsif done then
30                 counting <= '0';
31             else
32                 counting <= counting;
33             end if;
34         end if;
35     end process;
36
37     counter: entity work.counter_dynamic
38         generic map (
39             COUNTS => COUNTS
40         )
41         port map (
42             rst => rst,
43             clk => clk,
44             enable => counting,
45             done => done
46         );
47

```

```
48 end behavioral;
```

Test bench:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use work.tp1_pkg.all;
5
6  entity tb is
7  end tb;
8
9  architecture sim of tb is
10
11     constant CLK_PERIOD : time := 100 ms;
12     constant TEST_TIME : time := 100 sec;
13
14     signal clk, rst : std_logic := '0';
15
16     signal semaforo1 : semaforo;
17     signal semaforo2 : semaforo;
18 begin
19
20     u_semaforos: entity work.semaforos
21         generic map (
22             CLK_PERIOD => CLK_PERIOD
23         )
24         port map (
25             rst => rst,
26             clk => clk,
27             semaforo1 => semaforo1,
28             semaforo2 => semaforo2
29         );
30
31     -- Clock process
32     clk_process : process
33     begin
34         while now < TEST_TIME loop
35             clk <= '0';
36             wait for CLK_PERIOD / 2;
37             clk <= '1';
38             wait for CLK_PERIOD / 2;
39         end loop;
40         wait;
41     end process clk_process;
42
43     -- Reset process
44     reset_process : process
45     begin
46         rst <= '1'; -- Assert reset
47         wait for CLK_PERIOD * 2; -- Hold reset for 2 clock cycles
48         rst <= '0'; -- De-assert reset
49         wait;
50     end process reset_process;
51
52 end sim;
```