Universidad de Buenos Aires
Facultad De Ingeniería
Año 2022 – 2.do Cuatrimestre

(86.41) Sistemas digitales

Trabajo práctico N.º 1

Docentes a cargo:
Alvarez, Nicolás
Alpago, Octavio

Integrante	Padrón	Correo electrónico
Lützelschwab, Nahila	100686	nlützelschwab@fi.uba.ar

1. Introducción

El presente trabajo práctico tiene como objetivo realizar un circuito secuencial sincrónico aplicando el lenguaje de descripción de hardware VHDL.

2. Desarrollo

Para ello se implementó un circuito que controle dos semáforos en un cruce de calles, el cual presenta 6 salidas de acuerdo a las siguientes combinaciones posibles que llamaremos estados:

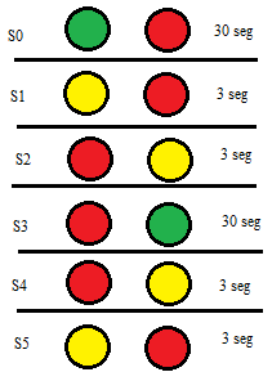

Figura 1

1. S0: Semáforo 1 en verde y semáforo 2 en rojo
2. S1: Semáforo 1 en amarillo y semáforo 2 en rojo
3. S2: Semáforo 1 en rojo y semáforo 2 en amarillo
4. S3: Semáforo 1 en rojo y semáforo 2 en verde
5. S4: Semáforo 1 en rojo y semáforo 2 en amarillo
6. S5: Semáforo 1 en amarillo y semáforo 2 en rojo

El tiempo en el que permanecerán en amarillo es de 3 segundos, mientras que permaneceran en verde y rojo durante 30 segundos. El reloj del sistema tendrá una frecuencia de operación de 50MHz.

Se puede representar el sistema mediante el siguiente diagrama de estados:

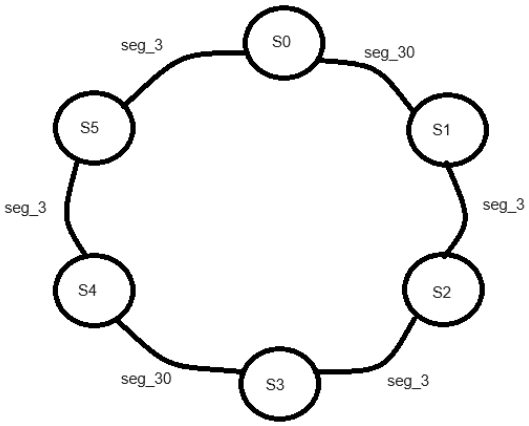

Figura 2

Teniendo en cuenta el circuito de la figura 4 se toman las siguientes consideraciones:

El clock tiene una frecuencia de 50MHz por lo tanto para que de una base de tiempo de 1 segundo debo contar 50 millones de pulsos, cada pulso dará $\frac{1}{50MHz} = 20ns$.El contador debe tener de cantidad de bits = $log_2(50{,}000{,}000) = 26$ bits y contará de 0 hasta 49.999.999.

En el pulso de $\frac{1}{50MHz} = 20ns$ hay un período completo del pulso de clock. Para la cuenta llegue a 30 se necesitan 5 bits ($2^5 = 32$).

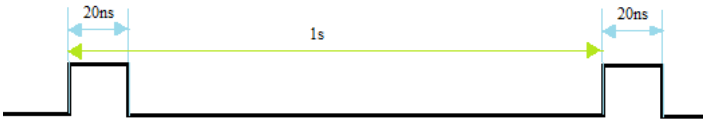

Figura 3: Counter

La comparación con cuenta = 29 o cuenta = 2 depende del estado en el que esté. Se requieren 30 segundos para los estados S0, S3 y para el resto se requieren 3 segundos. Por lo tanto se le agrega un multiplexor al contador que determine que tipo de cuenta se requiere para pasar de un estado al siguiente. Planteando un diagrama en bloques con dos contadores, uno que cuente por segundo y otro para los contadores de 3 y 30 segundos queda de la siguiente manera:

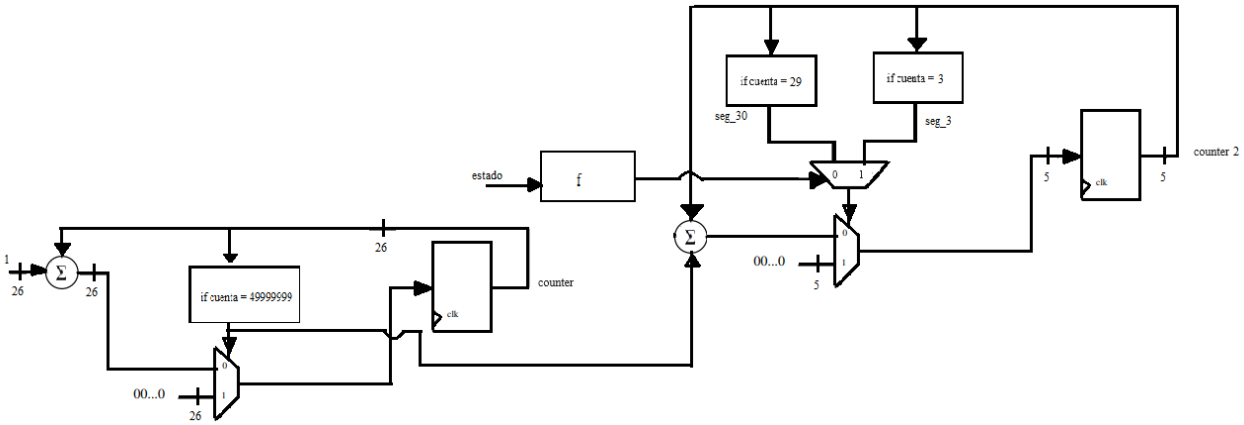

Figura 4

Para minimizarlo se puede utilizar un único contador como se puede observar en la figura 5. Haciendo uso de un contador con carga que permita elegir el modo de contar dependiendo del estado actual y siguiente. En caso que la cuenta llegue a 3 o 30, se setea $seg\_30 = 1$ o $seg\_3 = 1$, respectivamente. Dicha señal será útil para la máquina de estados finita, la cual describe la lógica del diagrama de estados presentado en la figura 2 y cuyas salidas son los estados de las luces (rojo_1, amarillo_1, verde_1, rojo_2, amarillo_2, verde_2). Para el caso de 30 segundos se debe contar hasta $30s \cdot 50MHz = 1{,}500{,}000{,}000$ y para 3 segundos $3s \cdot 50MHz = 150{,}000{,}000$, siendo entonces los rangos de [0; 1.499.999.999] y [0; 149.999.999] respectivamente. Por lo tanto, el contador debe poder contar de 0 de 1.499.999.999, siendo enoncesla cantidad de bits del contador de $log_2(1500000000) = 30,48 \approx 31$ bits.

El contador y la máquina de estados son circuitos sincróncos, se utiliza rising edge para el clock y nivel alto para el reset.

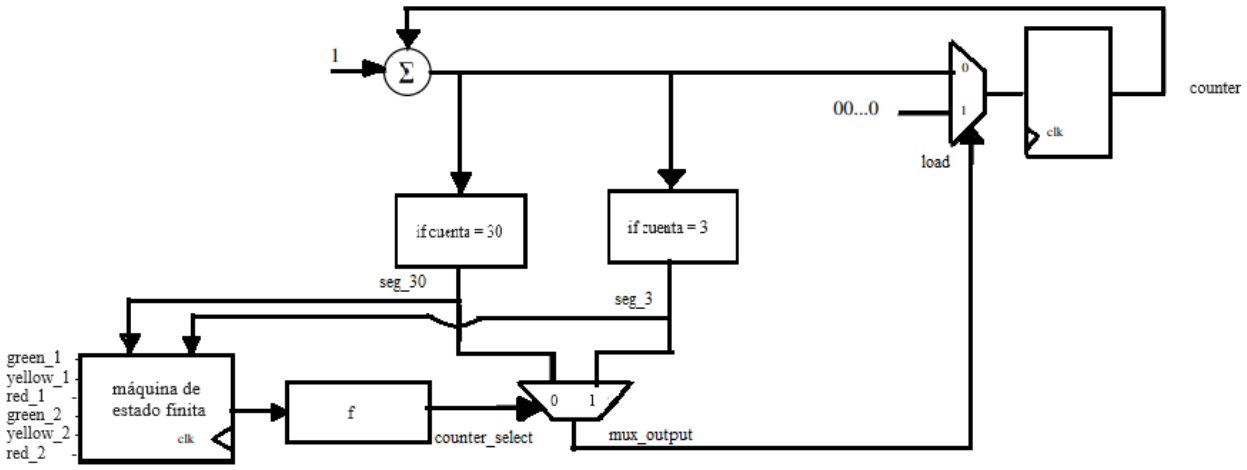

Figura 5

La implementación en VHDL es la siguiente:

```vhdl
-- Nahila Lützelschwab - TP1 - padron: 100686 - packages

-- Declaration of common library and use
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

-- Declaration data type for the states of traffic lights
package traffic_state_pkg is
  type traffic_state is (
            S0, -- green_1 & red_2
        S1, -- yellow_1 & red_2
        S2, -- red_1 & yellow_2
        S3, -- red_1 & green_2
        S4, -- red_1 & yellow_2
        S5  -- yellow_1 & red_2
    );
end package traffic_state_pkg;

```

```vhdl
-- Nahila Lützelschwab - TP1 - padron: 100686 - traffic_lights design

-- Declaration of common library and use
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.traffic_state_pkg.all;

entity traffic_lights is
        port(
        rst: in std_logic;
        clk: in std_logic;

        green_1: out std_logic;
        yellow_1: out std_logic;
        red_1: out std_logic;

        green_2: out std_logic;
        yellow_2: out std_logic;
        red_2: out std_logic
     );
end traffic_lights;

architecture behavioral of traffic_lights is
        -- Declaration of constants and signals to use
        constant counter_number : natural := 31;
```

```
    signal value : std_logic_vector(counter_number-1 downto 0) := (others => '0');
    signal seg_3 : std_logic;
    signal seg_30 : std_logic;
    signal counter_select : std_logic;
    signal mux_output : std_logic;
    signal state : traffic_state;
    --signal aux_state : traffic_state;

begin
        mux: entity work.mux
    port map(
      x0 => seg_3,
      x1 => seg_30,
      s => counter_select,
      y => mux_output
    );

    counter: entity work.mycounter
    generic map(
            N => counter_number
    )
        port map(
      rst => rst,
      clk => clk,
      load => mux_output,
      value => value,
      count => open,
      seg_3_reached => seg_3,
      seg_30_reached => seg_30
    );

    fsm : entity work.fsm
          port map(
      rst => rst,
      clk => clk,
      seg_3 => seg_3,
      seg_30 => seg_30,
      state => state
    );

    process(state)
    begin
      if state = S0 or state = S3 then
        counter_select <= '1';
      else
        counter_select <= '0';
      end if;
    end process;

    -- Outputs
        green_1 <= '1' when state = S0 else '0';
    yellow_1 <= '1' when ((state = S1) or (state = S5)) else '0';
    red_1 <= '1' when ((state = S2) or (state = S3) or (state = S4)) else '0';

    green_2 <= '1' when state = S3 else '0';
    yellow_2 <= '1' when ((state = S2) or (state = S4)) else '0';
    red_2 <= '1' when ((state = S0) or (state = S1) or (state = S5)) else '0';

end behavioral;

```

```
-- Nahila Lützelschwab - TP1 - padron: 100686 - multiplexor

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

entity mux is
```

```
  port(
    x0 : in std_logic;
    x1 : in std_logic;
    s : in std_logic;
    y : out std_logic
  );
end mux;

architecture behavioral of mux is
begin
  process (x0, x1, s) is
    begin
    case s is
      when '0' =>
              y <= x0;
      when others =>
              y <= x1;
    end case;
  end process;
end behavioral;
```

```
-- Nahila Lützelschwab - TP1 - padron: 100686 - counter

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

-- N-bit generic counter with load signal

entity mycounter is
  generic (
          N : natural := 8
  );
  port(
    rst : in std_logic;
    clk : in std_logic;
    load : in std_logic;
    value : in std_logic_vector(N-1 downto 0);
    count : out std_logic_vector(N-1 downto 0);
    -- indicators when counter reaches 3 or 30 seconds
    seg_3_reached : out std_logic;        -- seg_3_reached = 1 if reached otherwise 0
    seg_30_reached : out std_logic        -- seg_30_reached = 1 if reached otherwise 0
  );
end mycounter;

architecture behavioral of mycounter is
        signal aux_count : unsigned(N-1 downto 0);
    constant N_SEG3 : natural := 149999999 ;        -- 3s * 50MHz = 150.000.000 -> 0 - 149999999
    constant N_SEG30 : natural := 1499999999;        -- 30s * 50MHz = 1.500.000.000 -> 0 - 1499999999

begin
  process(clk,rst)
  begin
    if rst='1' then
            aux_count <= (others => '0');
    elsif clk = '1' and clk'event then
      if load = '1' then
              aux_count <= unsigned(value);
      else
              aux_count <= aux_count + 1;
      end if;
    end if;
  end process;

  count <= std_logic_vector(aux_count);
  seg_3_reached <= '1' when (aux_count = N_SEG3) else '0';
```

```
  seg_30_reached <= '1' when (aux_count = N_SEG30) else '0';

end behavioral;

```

```
-- Nahila Lützelschwab - TP1 - padron: 100686 - fsm

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

entity fsm is
  port(
    rst : in std_logic;
    clk : in std_logic;
    seg_3 : in std_logic;
    seg_30 : in std_logic;
    state : out traffic_state
  );
end fsm;

architecture behavioral of fsm is

  -- Declaration of signals to use
    signal aux_state : traffic_state;
begin
  process(clk,rst)
  begin
    if rst='1' then
            aux_state <= S0;
    elsif clk = '1' and clk'event then
            case aux_state is
        -- green_1 & red_2 -> yellow_1 & red_2
                when S0 =>
            if seg_30 = '1' then
                    aux_state <= S1;
            end if;
        -- yellow_1 & red_2 -> red_1 & yellow_2
          when S1 =>
            if seg_3 = '1' then
                    aux_state <= S2;
            end if;
        -- red_1 & yellow_2 -> red_1 & green_2
          when S2 =>
            if seg_3 = '1' then
                    aux_state <= S3;
            end if;
        -- red_1 & green_2 -> red_1 & yellow_2
          when S3 =>
            if seg_30 = '1' then
                    aux_state <= S4;
            end if;
        -- red_1 & yellow_2 -> yellow_1 & red_2
          when S4 =>
            if seg_3 = '1' then
                    aux_state <= S5;
            end if;
        -- yellow_1 & red_2 -> green_1 & red_2
          when S5 =>
            if seg_3 = '1' then
                    aux_state <= S0;
            end if;
        end case;
          end if;
  end process;
  state <= traffic_state(aux_state);
end behavioral;
```

2.1. Simulación - Test bench

Para simular el circuito se reemplazaron los valores por otros más chicos para que los resultados sean apreciables y se realizó mediante el siguiente test bench:

```
-- Nahila Lützelschwab - TP1 - padron: 100686 - traffic lights testbench

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

entity tb_traffic_lights is
end tb_traffic_lights;

architecture behavioral of tb_traffic_lights is

  constant SIM_TIME_NS : time := 10000 ns;
  constant TB_N : natural := 31;

  signal tb_rst : std_logic;
  signal tb_clk : std_logic := '0';

  signal tb_green_1 : std_logic;
  signal tb_yellow_1 : std_logic;
  signal tb_red_1 : std_logic;

  signal tb_green_2 : std_logic;
  signal tb_yellow_2 : std_logic;
  signal tb_red_2 : std_logic;

begin

  tb_rst <= '0', '1' after 1 ns, '0' after 20 ns;
  -- 50 MHz clock frequency
  tb_clk <= not tb_clk after 10 ns;

  stop_simulation : process
  begin
    wait for SIM_TIME_NS; --run the simulation for this duration
    assert false
            report "Simulation finished."
            severity failure;
  end process;

  I1: entity work.traffic_lights(behavioral)
  port map(
    rst => tb_rst,
    clk => tb_clk,

    green_1 => tb_green_1,
    yellow_1 => tb_yellow_1,
    red_1 => tb_red_1,

    green_2 => tb_green_2,
    yellow_2 => tb_yellow_2,
    red_2 => tb_red_2
  );

end behavioral;
```

El resultado de la simulación se puede observar en la figura 6 las 6 salidas y las entradas de clock y de reset, donde se pueden apreciar los cambios de estado tal como lo describe el diagrama de estados.

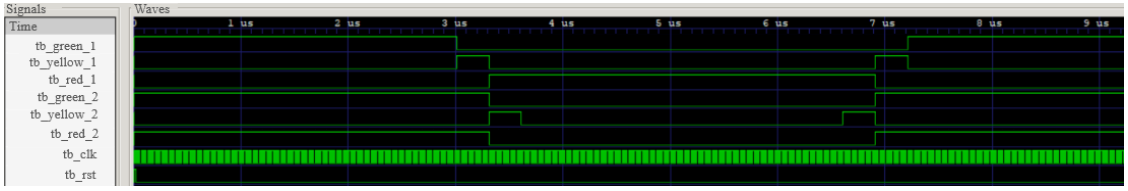

Figura 6

2.2. Síntesis

Se realizó la síntesis sobre el dispositivo FPGA xc7a15tftg256-1 con el software Vivado, donde se demuestra la correcta implementación del diseño en el mismo. Se puede observar en la figura **??** el register transfer level, el cual es un nivel de abstracción intermedio que se encuentra entre la descripción del hardware a nivel de comportamiento y la descripción a nivel de puertas lógicas.

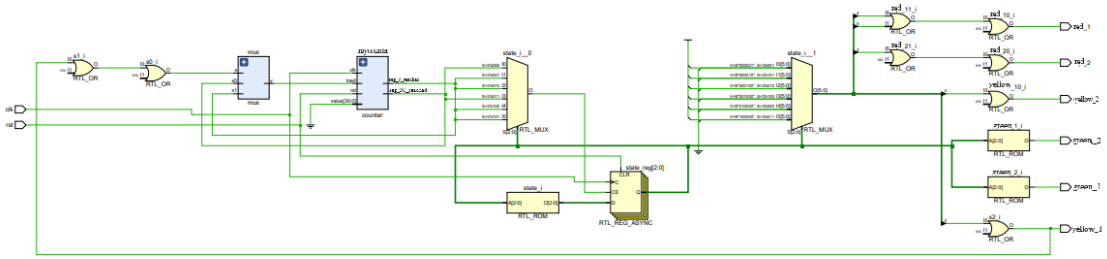

Figura 7

En la figura 8 se puede observar el esquemático de implementación que muestra el conexionado interno de la FPGA.

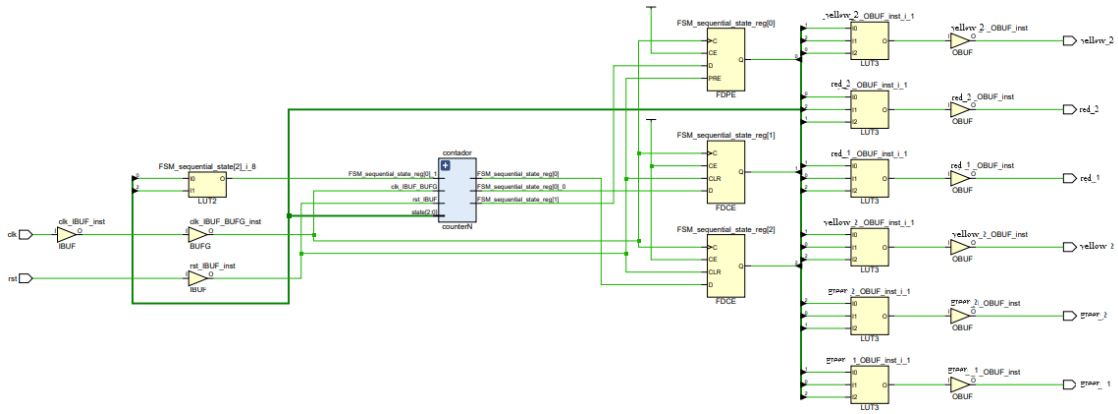

Figura 8

3. Conclusión

Se puede concluir que el presente trabajo práctico permitió fijar el concepto de circuito secuencial sincrónico y aprender el manejo tanto del lenguaje de descripción de hardware VHDL como del software Vivado que permitió realizar la síntesis sobre un dispositivo FPGA específico.