Trabajo de VHDL

Cruce de Dos Semáforos

GRUPO 23

SISTEMAS ELECTRÓNICOS DIGITALES

CURSO 2022/2023

Alumnos:	**Nº de Matrícula:**
Aspiroz de la Calle, Miguel	54918
Bustillo Ergui, Jaime	54920
Pacheco Guiop, Jefferson Eduardo	55581

ÍNDICE

1. INTRODUCCIÓN

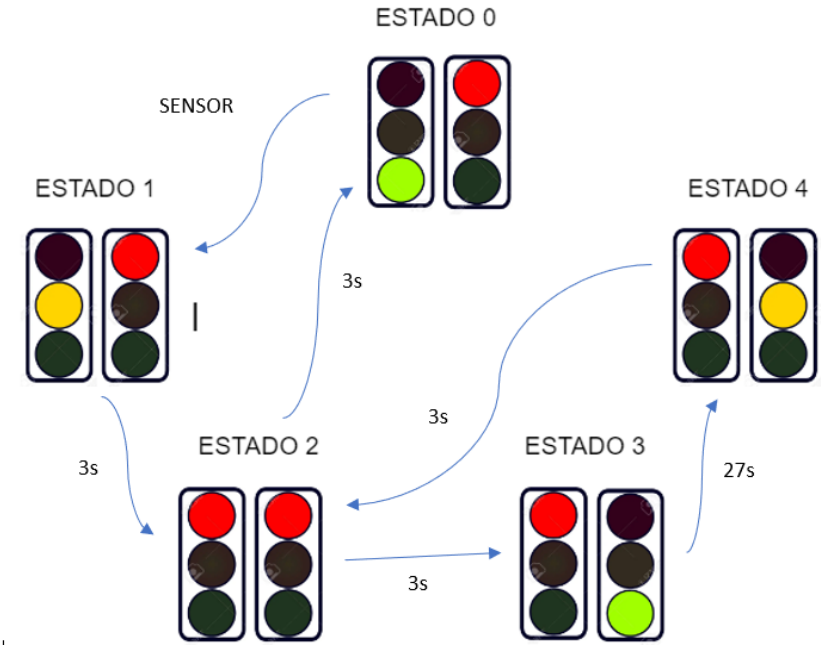
Para el trabajo de programación de la parte de VHDL de la asignatura de Sistemas Electrónicos Digitales se ha optado por llevar a cabo la tercera propuesta ofertada por los profesores.

Propuesta:
Cruce de dos semáforos: se trata de diseñar el circuito de control de dos semáforos que se encuentran en un cruce entre un camino rural y una carretera. Normalmente, el semáforo del camino estará siempre en rojo y el de la carretera en verde, pero en el camino hay instalado un sensor que permite detectar la presencia de un coche.

La finalidad del proyecto consistirá en que, cuando el sensor detecte un coche en el camino rural, se dé paso a una secuencia de estados con la cual el semáforo principal cambiará de verde a ámbar y posteriormente a rojo; instantes después, el semáforo secundario se pondrá en verde durante un tiempo determinado, tras el cual se volverá al estado inicial. Se ha decidido que este tiempo tenga un valor fijo para que el semáforo de la carretera principal no esté en rojo durante un período excesivo; lo que significa que, aunque el sensor detecte nuevos coches llegando por el camino rural mientras el semáforo del mismo está en verde, esto no incrementará el tiempo que este permanecerá abierto.

2. MÁQUINA DE ESTADOS

Como se ha descrito en el apartado anterior, este proyecto consta de una máquina de estados finitos que representan las diferentes combinaciones posibles de ambos semáforos:



De esta manera, los semáforos estarán en la configuración predeterminada (ESTADO 0) hasta que aparezca un coche en la carretera secundaria, el cual activará el sensor e iniciará toda la secuencia del semáforo. Inmediatamente tras la activación del sensor, el semáforo de la carretera principal se pone en ámbar (ESTADO 1) y, 3 segundos después, en rojo (ESTADO 2). Tras un margen de seguridad de otros 3 segundos, se

pone el semáforo de la carretera secundaria en verde, dejando pasar al coche y permaneciendo así (ESTADO 3) durante otros 27 segundos. Posteriormente, transcurrido este tiempo, el semáforo del camino rural se pone en ámbar (ESTADO 4) y luego vuelven a coincidir ambos semáforos en rojo (S2). Finalmente, se vuelve al estado predeterminado en el que está el semáforo de la carretera principal en verde y el de la secundaria en rojo.

3. CÓDIGO

a. SEMAFORO.vhd

Este es el programa principal de la máquina de estados en el cual se controla la secuencia del semáforo basándose en el reloj y el sensor.
Para empezar, se declara la entidad *SEMAFORO*, en la cual se tienen cinco entradas y tres vectores de salida, que corresponden a las luces verde, ámbar y roja de sendos semáforos:

```
entity SEMAFORO is
    port (
        RESET : in std_logic;
        CLK : in std_logic;
        CLK2 : in std_logic;
        k : in natural;
        SENSOR : in std_logic;
        LUZ_R : out std_logic_vector(0 TO 1);
        LUZ_V : out std_logic_vector(0 TO 1);
        LUZ_A : out std_logic_vector(0 TO 1)
        );
end SEMAFORO ;
```

Por otro lado, la arquitectura comienza con la declaración de las diferentes señales auxiliares que van a ser necesarias durante el programa:

```
architecture behavioral of SEMAFORO  is
    type STATES is (S0, S1, S2, S3, S4);--los estados son posibles conbinaciones de los dos semaforos
    signal current_state: STATES := S0;--Indica el estado en el que se encuentra el programa
    signal next_state: STATES;--Indica el siguiente estado
    signal tiempo_inicio: NATURAL;
    signal a: NATURAL := 0;
```

A continuación, se inicia el proceso *state_register*, en el que se realiza un bucle *if* en el que con cada flanco de subida del reloj se introduce el siguiente estado en la variable del estado actual y, en caso de pulsarse el botón *RESET*, se vuelve al estado S0:

```
state_register: process (RESET, CLK)
begin

 if rising_edge(CLK ) then
    current_state<=next_state;--introduce el siguiente estado en el estado actual
    if RESET = '0' then--si pulsamos RESET volvemos al estado S0
        current_state <= S0;
    end if;
 end if;

end process;
```

A continuación se encuentra el decodificador, que se encarga de cambiar al siguiente estado. Para ello, se utiliza un *case-when* en el cual se establecen las condiciones para cambiar al siguiente estado dependiendo de en cuál se encuentre previamente. Si se parte del S0 y es la primera interacción (*a*=0), se introduce S1 en la variable del siguiente estado y se inicializa la variable *tiempo_inicio* en *k*; sin embargo, en caso de que ya no sea la primera interacción (*a*=1), se tendrá una nueva condición, que implica que el tiempo que ha pasado desde que se inició la última interacción tiene que ser mayor que 45 s, de manera que por muchos coches que vengan por la carretera rural el semáforo de la carretera principal no siempre va a estar en rojo, sino que permanecerá al menos 10 s en verde entre cada secuencia de cruce.

```
nextstate_decod: process (SENSOR, current_state, k)
 begin

    next_state <= current_state;
 case current_state is--cambio de un estado a otro
    when S0 =>
        if SENSOR = '1' and a=0 then --El programa debe esperar almenos 20s entre una secuencia y la siguiente.
            tiempo_inicio<=k;
            next_state <= S1;--se enciende la luz ambar del semaforo principal
            a<=1;
        elsif SENSOR = '1' and (k-tiempo_inicio)>=45 and a/=0 then
            tiempo_inicio<=k;
            next_state <= S1;--se enciende la luz ambar del semaforo principal
        end if;
```

Para el resto de los estados el cambio depende únicamente del tiempo transcurrido desde el inicio de la interacción (*k-tiempo_inicio*):

```
    when S1 =>
        if (k-tiempo_inicio)=2 then  --a los tres segundos el primer semaforo se pone en rojo
            next_state <= S2;
        end if;
    when S2 =>
         if (k-tiempo_inicio)=5 then   --tras otros 3s de seguridad el segundo semaforo se pone en verde
            next_state <= S3;
         elsif (k-tiempo_inicio)=32 then  --finalente el semaforo se pone en rojo y vuelve al primer estado en el que el primer semaforo
            next_state <= S0;
         end if;
    when S3 =>
         if (k-tiempo_inicio)=26 then  --el semaforo permanece en verde 20s , para que pasen los coches que esten parados pasen y luego
            next_state <= S4;
         end if;
    when S4 =>
         if (k-tiempo_inicio)=29 then  --El semaforo permanece en ambar 3s
            next_state <= S2;
         end if;
    when others =>--En caso de error se vuelve al primer estado
        next_state <= S0;
end case;
end process;
```

Finalmente, se declaran las acciones que suceden en cada estado. Para ello, se usa otro *case-when* en el proceso *output_decod*. Para cada estado se fijan qué luces tienen que estar encendidas y cuáles apagadas:

```
output_decod: process (current_state)
begin
case current_state is
    when S0 =>--   semaforo 1==>Verde    semaforo2==>rojo
      LUZ_R(1)<= '1';
       LUZ_R(0)<= '0';
       LUZ_A(1)<= '0';
       LUZ_A(0)<= '0';
       LUZ_V(1)<= '0';
       LUZ_V(0)<= '1';
    when S1 =>--   semaforo 1==>Ambar    semaforo2==>rojo
       LUZ_R(1)<= '1';
       LUZ_R(0)<= '0';
       LUZ_A(1)<= '0';
       LUZ_A(0)<= '1';
       LUZ_V(1)<= '0';
       LUZ_V(0)<= '0';
   when S2 =>--   semaforo 1==>Rojo    semaforo2==>rojo
       LUZ_R(1)<= '1';
       LUZ_R(0)<= '1';
       LUZ_A(1)<= '0';
       LUZ_A(0)<= '0';
       LUZ_V(1)<= '0';
       LUZ_V(0)<= '0';
   when S3 =>--   semaforo 1==>Rojo    semaforo2==>Verde
       LUZ_R(1)<= '0';
       LUZ_R(0)<= '1';
       LUZ_A(1)<= '0';
       LUZ_A(0)<= '0';
       LUZ_V(1)<= '1';
       LUZ_V(0)<= '0';
   when S4 =>--   semaforo 1==>Rojo    semaforo2==>Ambar
       LUZ_R(1)<= '0';
       LUZ_R(0)<= '1';
       LUZ_A(1)<= '1';
       LUZ_A(0)<= '0';
       LUZ_V(1)<= '0';
       LUZ_V(0)<= '0';
   when others =>-- todos los semaforos apagados en caso de error
       LUZ_R(1)<= '0';
       LUZ_R(0)<= '0';
       LUZ_A(1)<= '0';
       LUZ_A(0)<= '0';
       LUZ_V(1)<= '0';
       LUZ_V(0)<= '0';

 end case;
end process;
end behavioral;
```

b. RELOJ.vhd

En este segundo archivo del código se crea la entidad *Reloj*, la cual sirve para dividir el reloj de 100 MHz de la placa Nexys 4 y conseguir un reloj de 1 Hz a modo de cronómetro que pueda contar los segundos necesarios para los cambios de estado en el programa anterior.

Primero de todo, se crean dos entradas (*CLK* y *RESET*) y dos salidas (*CLK2* y *k*).

```
entity Reloj is
    Port (
        CLK2 : out STD_LOGIC;--reloj de 1Hz para contar los segundos
        CLK : in std_logic;--reloj basico de 100MHz
        RESET : in std_logic;
        k : out natural);--contador de segundos en el programa principal
end Reloj;
```

Posteriormente, se declaran en la arquitectura del componente las señales auxiliares que se usarán esta vez, de las cuales la más importante es *Cont1*, que se encarga de dividir el reloj.

```
architecture Behavioral of Reloj is


Signal Cont1:  integer range 0 to  100000000 := 0;--Primer divisor del reloj
signal Salida: std_logic;
signal segundo: natural;
```

Y finalmente, la parte funcional del programa se trata de un bucle *if*, el cual aumenta el contador una vez con cada flanco de subida del reloj principal, de manera que cuando llega a 100000000 se cambia el bit de la salida a su contrario, que posteriormente se mete en el reloj. Así, se multiplica el período de 10 ns por 100000000 y se obtiene como resultado 1 s.

Por otro lado, cada vez que el reloj generado tiene un flanco de bajada, a la variable k se le suma 1.

```
 Signal Cont1:  integer range 0 to 100000000 := 0;
 signal Salida: std_logic;


 begin

process(CLK, RESET)
 begin
if RESET = '0' then
     Cont1 <= 0;
     Salida<='0';
 elsif rising_edge (CLK) then
     if (cont1 = 100000000) then
             salida<=NOT(salida);
             cont1<=0;
     else
         cont1<=cont1+1;
     end if;
end if;
end process;

 CLK2<=salida;
end Behavioral;
```

c. Top.vhd

Por último, está la entidad top, la cual se encarga de englobar todas las otras entidades y de ejecutarlas correctamente. Para ello, en su interfaz se declaran las entradas y salidas reales del sistema, además de los otros dos componentes (*SEMAFORO* y *RELOJ*), con las entradas y salidas necesarias.

```
entity top is
    Port ( LUZ_R : out std_logic_vector(0 TO 1);--Luces rojas de los semaforos
           LUZ_V : out std_logic_vector(0 TO 1);--luces verdes de los semaforos
           LUZ_A : out std_logic_vector(0 TO 1);--Luces ambar de los semaforos
           SENSOR : in std_logic;--Sensor que detecta el coche que viene
           CLK : in std_logic; --Reloj Basico
           RESET : in std_logic);--Volver al primer estado
end top;

architecture Behavioral of top is

signal CLK2 :std_logic ;

component Reloj is
    port(
        CLK : in std_logic;
        RESET : in std_logic;
        CLK2 : out std_logic
        );
end component;
component SEMAFORO is--Programa que gestiona los estados de los semaforos
    port (
        RESET : in std_logic;
        CLK : in std_logic;
        CLK2 : in std_logic;
        SENSOR : in std_logic;
        LUZ_R : out std_logic_vector(0 TO 1);
        LUZ_V : out std_logic_vector(0 TO 1);
        LUZ_A : out std_logic_vector(0 TO 1)
        );
end component ;
```

A cada una de las entidades anteriores se le pasan las entradas correspondientes y, posteriormente, se recuperan las salidas.

```
signal synk_out: std_logic;
signal edge: std_logic;

begin
Inst_REL: Reloj PORT MAP
(
    CLK2 => CLK2,
    CLK=>CLK,
    RESET=>RESET
 );
Inst_SEM: SEMAFORO PORT MAP
 (
    CLK2 => CLK2,
    CLK => CLK,
    SENSOR => SENSOR,
    RESET => RESET,
    LUZ_R =>LUZ_R,
    LUZ_A =>LUZ_A,
    LUZ_V =>LUZ_V
);
end Behavioral;
```

4. RESTRICCIONES

En el archivo de restricciones de la Nexys 4 DDR se han descomentado las líneas de código necesarias para las necesidades específicas de este programa, quedando de la manera que se muestra abajo.

Entradas:

```
## Clock signal
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK }]; #IO_L12P_T1_MRCC_35 Sch=clk100mhz
##create_clock -add -name CLK2 -period 1 -waveform {0 0.5} [get_ports { CLK2 }];


##Switches

set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { SENSOR }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { RESET }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn
```

Salidas:

```
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LUZ_R[0] }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LUZ_A[0] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LUZ_V[0] }]; #IO_L17N_T2_A25_15 Sch=led[2]
#set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED[3] }]; #IO_L8P_T1_D11_14 Sch=led[3]
#set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED[4] }]; #IO_L7P_T1_D09_14 Sch=led[4]
#set_property -dict { PACKAGE_PIN V17   IOSTANDARD LVCMOS33 } [get_ports { LED[5] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
#set_property -dict { PACKAGE_PIN U17   IOSTANDARD LVCMOS33 } [get_ports { LED[6] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
#set_property -dict { PACKAGE_PIN U16   IOSTANDARD LVCMOS33 } [get_ports { LED[7] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
#set_property -dict { PACKAGE_PIN V16   IOSTANDARD LVCMOS33 } [get_ports { LED[8] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
#set_property -dict { PACKAGE_PIN T15   IOSTANDARD LVCMOS33 } [get_ports { LED[9] }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
#set_property -dict { PACKAGE_PIN U14   IOSTANDARD LVCMOS33 } [get_ports { LED[10] }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
#set_property -dict { PACKAGE_PIN T16   IOSTANDARD LVCMOS33 } [get_ports { LED[11] }]; #IO_L15N_T2_DQS_DOUT_CSO_B_14 Sch=le
#set_property -dict { PACKAGE_PIN V15   IOSTANDARD LVCMOS33 } [get_ports { LED[12] }]; #IO_L16P_T2_CSI_B_14 Sch=led[12]
set_property -dict { PACKAGE_PIN V14   IOSTANDARD LVCMOS33 } [get_ports { LUZ_R[1] }]; #IO_L22N_T3_A04_D20_14 Sch=led[13]
set_property -dict { PACKAGE_PIN V12   IOSTANDARD LVCMOS33 } [get_ports { LUZ_A[1] }]; #IO_L20N_T3_A07_D23_14 Sch=led[14]
set_property -dict { PACKAGE_PIN V11   IOSTANDARD LVCMOS33 } [get_ports { LUZ_V[1] }]; #IO_L21N_T3_DQS_A06_D22_14 Sch=led[1
```

5. TESTBENCH

a. tb_SEMAFORO

En este primer *testbench* se pretende simular la llegada de un coche por el camino rural a los 5 segundos, el cual se va después de que el semáforo se ponga en verde. Esta es la simulación más simple para plasmar el funcionamiento básico del programa.

El código del *testbench* es el siguiente:

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

entity tb_SEMAFORO is
end tb_SEMAFORO;

architecture tb of tb_SEMAFORO is

    component TOP
        port (RESET  : in std_logic;
              CLK    : in std_logic;
              SENSOR : in std_logic;
              LUZ_R  : out std_logic_vector (0 to 1);
              LUZ_V  : out std_logic_vector (0 to 1);
              LUZ_A  : out std_logic_vector (0 to 1));
    end component;

    signal RESET  : std_logic;
    signal CLK    : std_logic;
    signal SENSOR : std_logic;
    signal LUZ_R  : std_logic_vector (0 to 1);
    signal LUZ_V  : std_logic_vector (0 to 1);
    signal LUZ_A  : std_logic_vector (0 to 1);
    constant TbPeriod : time := 10 ns;
    constant TbPeriod2 : time := 1000000000 ns;
    signal TbClock : std_logic := '0';
    signal TbSimEnded : std_logic := '0';

begin

    dut : TOP
    port map (RESET  => RESET,
              CLK    => CLK,
              SENSOR => SENSOR,
              LUZ_R  => LUZ_R,
              LUZ_V  => LUZ_V,
              LUZ_A  => LUZ_A);

    -- Clock generation
    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';
    CLK <= TbClock;


    stimuli : process
    begin

        SENSOR <= '0';
        -- Reset generation
        -- EDIT: Check that RESET is really your reset signal
        RESET <= '0';
        wait for 100 ns;
        RESET <= '1';
        wait for 100 ns;

        wait for 5*TbPeriod2;
        SENSOR<='1';
        wait for TbPeriod2*10;
        SENSOR<='0';

        wait for 50 * TbPeriod2;

        -- Stop the clock and hence terminate the simulation
        TbSimEnded <= '1';
        wait;
    end process;

end tb;
```

b. tb_SEMAFORO2

En el segundo *testbench* se hace una simulación en la que aparece un coche por el camino rural a los 5 segundos, poniéndose en verde el semáforo. 15 segundos después del primer coche, mientras sigue en verde, llega un segundo coche que atraviesa el cruce sin afectar al programa.

El código es el siguiente:

```
entity tb_SEMAFORO2 is
end tb_SEMAFORO2;

architecture tb of tb_SEMAFORO2 is

    component TOP
        port (RESET  : in std_logic;
              CLK    : in std_logic;
              SENSOR : in std_logic;
              LUZ_R  : out std_logic_vector (0 to 1);
              LUZ_V  : out std_logic_vector (0 to 1);
              LUZ_A  : out std_logic_vector (0 to 1));
    end component;

    signal RESET  : std_logic;
    signal CLK    : std_logic;
    signal SENSOR : std_logic;
    signal LUZ_R  : std_logic_vector (0 to 1);
    signal LUZ_V  : std_logic_vector (0 to 1);
    signal LUZ_A  : std_logic_vector (0 to 1);

    constant TbPeriod : time := 10 ns;
    constant TbPeriod2 : time := 1000000000 ns;
    signal TbClock : std_logic := '0';
    signal TbSimEnded : std_logic := '0';

begin

    dut : TOP
    port map (RESET  => RESET,
              CLK    => CLK,
              SENSOR => SENSOR,
              LUZ_R  => LUZ_R,
              LUZ_V  => LUZ_V,
              LUZ_A  => LUZ_A);

    -- Clock generation
    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';
    CLK <= TbClock;


    stimuli : process

begin

    SENSOR <= '0';
    -- Reset generation
    -- EDIT: Check that RESET is really your reset signal
    RESET <= '0';
    wait for 100 ns;
    RESET <= '1';
    wait for 100 ns;

    wait for 5*TbPeriod2;--primer coche que llega y se queda parado 10s hasta que pasa (T=5s)
    SENSOR<='1';
    wait for TbPeriod2*10;
    SENSOR<='0';


     wait for 5*TbPeriod2;--Segundo coche que llega que pasa de largo porque el semaforo ya esta en verde (t=20s)
    SENSOR<='1';
    wait for TbPeriod2;
    SENSOR<='0';


    wait for 50 * TbPeriod2;
```

```
        -- Stop the clock and hence terminate the simulation
        TbSimEnded <= '1';
        wait;
    end process;

end tb;

-- Configuration block below is required by some simulators. Usually no need to edit.

configuration cfg_tb_SEMAFORO of tb_SEMAFORO is
    for tb
    end for;
end cfg_tb_SEMAFORO;
```

c. tb_SEMAFORO3

Por último, en el tercer *testbench* se simula, igual que en los anteriores casos, la llegada de un coche por el camino rural a los 5 segundos. Posteriormente, pasa otro coche mientras el semáforo está en verde y, finalmente, cuando el semáforo vuelve a estar en rojo, llega un último coche que tiene que esperar el intervalo mínimo establecido para que se vuelva a poner en verde.

El código de este tercer *testbench* es el siguiente:

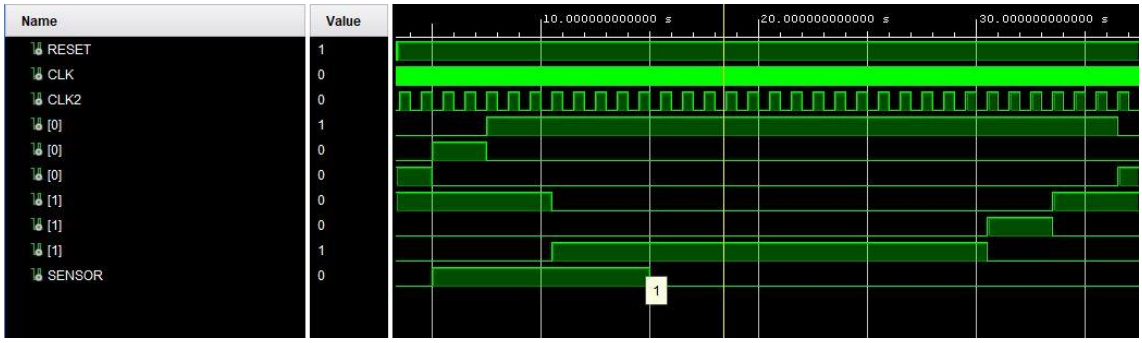
```
library ieee;
use ieee.std_logic_1164.all;

entity tb_SEMAFORO3 is
end tb_SE)3;

architecture tb of tb_SEMAFORO3 is

    component TOP
        port (RESET  : in std_logic;
              CLK    : in std_logic;
              SENSOR : in std_logic;
              LUZ_R  : out std_logic_vector (0 to 1);
              LUZ_V  : out std_logic_vector (0 to 1);
              LUZ_A  : out std_logic_vector (0 to 1));
    end component;

    signal RESET  : std_logic;
    signal CLK    : std_logic;
    signal SENSOR : std_logic;
    signal LUZ_R  : std_logic_vector (0 to 1);
    signal LUZ_V  : std_logic_vector (0 to 1);
    signal LUZ_A  : std_logic_vector (0 to 1);

    constant TbPeriod : time := 10 ns;
    constant TbPeriod2 : time := 1000000000 ns;
    signal TbClock : std_logic := '0';
    signal TbSimEnded : std_logic := '0';

begin

    dut : TOP
    port map (RESET  => RESET,
              CLK    => CLK,
              SENSOR => SENSOR,
              LUZ_R  => LUZ_R,
              L      UZ_V,
              LUZ_A  => LUZ_A);

    -- Clock generation
    TbClock <= not TbClock after TbPeriod/2 when TbSimEnded /= '1' else '0';
    CLK <= TbClock;


    stimuli : process
    begin
```

```
    -- Reset generation
    -- EDIT: Check that RESET is really your reset signal
    RESET <= '0';
    wait for 100 ns;
    RESET <= '1';
    wait for 100 ns;

    wait for 5*TbPeriod2;--primer coche que llega y se queda parado 10s hasta que pasa (T=5s)
    SENSOR<='1';
    wait for TbPeriod2*10;
    SENSOR<='0';

    wait for 25*TbPeriod2;   --Segundo coche que llega, llega cuando el semaforo vuelve a estar en rojo. (t=41s)
    SENSOR<='1';             --tiene que esperar a que pasen los 20s minimos para que se vuelva a poner en verde
    wait for TbPeriod2*20;
    SENSOR<='0';


    wait for 50 * TbPeriod2;

    -- Stop the clock and hence terminate the simulation
    TbSimEnded <= '1';
    wait;
end process;
```

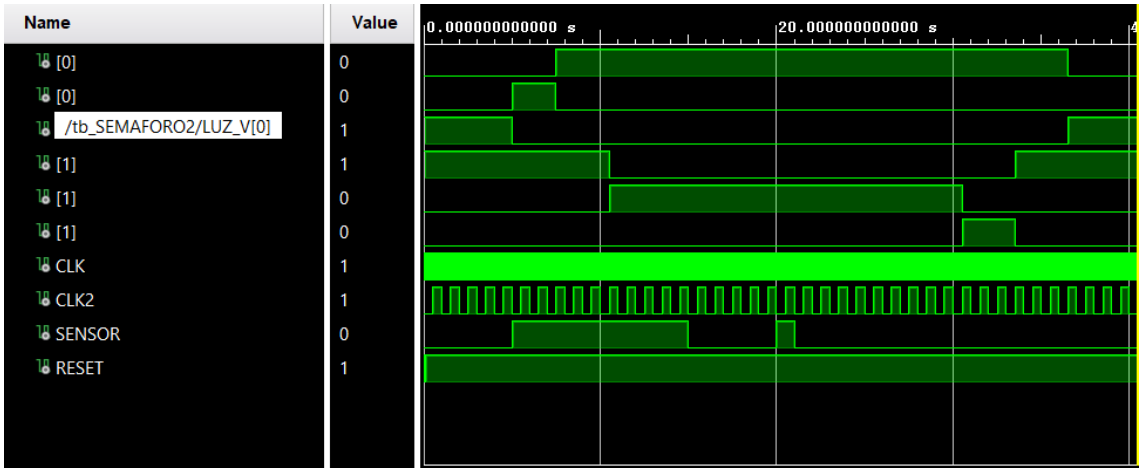
6. RESULTADOS

A continuación se muestran las simulaciones de cada uno de los tres *testbench* anteriormente expuestos.

tb_SEMAFORO.vhd



tb_SEMAFORO2.vhd



tb_SEMAFORO3.vhd