Universidad Politécnica de Madrid

Cruce de semáforos

David Gay Alonso 52375

Raúl Güemes Sánchez 52397

A – 404

Sistemas Electrónicos Digitales

Índice

OBJETIVO

El objetivo de este trabajo es conseguir el correcto funcionamiento de dos semáforos con pulsador en el cruce de dos calles. Programaremos en Vivado los semáforos para tratar de alcanzar una perfecta sincronización entre ellos.

INTRODUCCIÓN

Diseñaremos el circuito de control de dos semáforos que se encuentran en un cruce entre un camino rural y una carretera. Estableceremos que el semáforo del camino estará siempre en rojo y el de la carretera en verde, pero en el del camino hay instalado un pulsador que al activarlo pone dicho semáforo en verde, y el otro en rojo.

MATERIAL DE TRABAJO

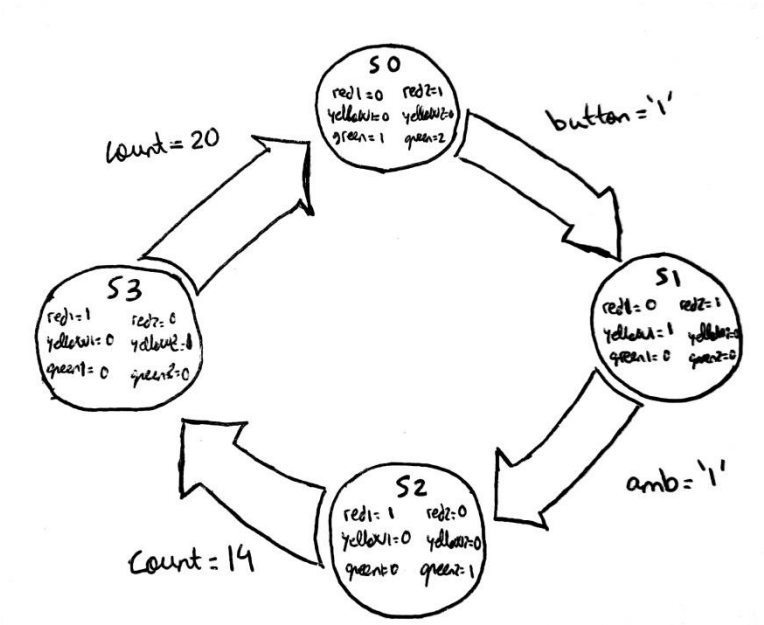
El material utilizado en el proyecto es:

Programa Vivado

Placa NEXYS 4 DDR

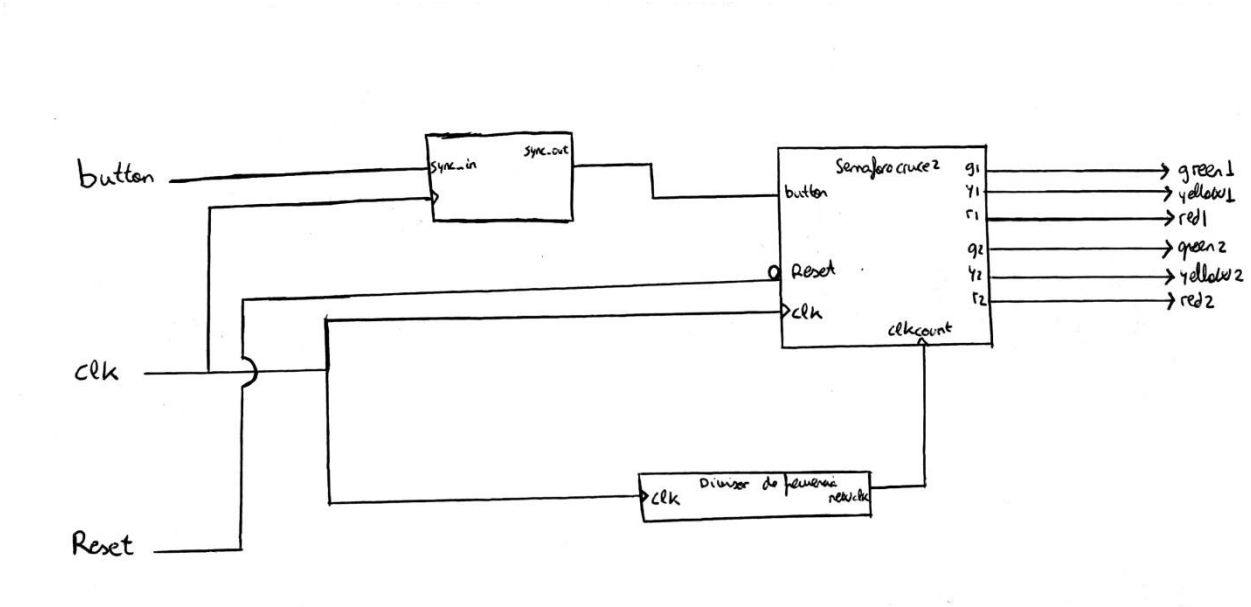
FUNCIONAMIENTO

Diseñaremos, implementaremos y programaremos la sincronización de dos semáforos en un cruce para evitar accidentes. Para la programación realizaremos una máquina de estados en la que las salidas dependerán del pulsador y el contador.



En el estado inicial, tenemos el semáforo 1 está en verde y el semáforo 2 está en rojo. Al darle al button el estado pasa al estado siguiente, donde el 1 se pone en ambar y el 2 en rojo. Cuando el contador está a 6, se activa la señal amb y pasa al siguiente estado, el cual se pone el 1 al rojo y el 2 en verde. Cuando el contador alcanza el valor 14, pasa al estado siguiente donde el primero se pone en rojo y el segundo en ambar. Al alcanzar el contador el valor 20 pasa al estado inicial.

A continuación, se mostrará nuestro diagrama de bloques, en función de nuestras entradas y salidas:



CÓDIGO

Código semaforocruce.vhd:

Estableceremos nuestras entradas (clk, clkcount, reset, button) y salidas de la máquina de estados (red1, red2, green1, green2, yelow1, yelow2). Definiremos los estados (s0, s1, s2, s3) y las señales (reloj, count, amb).

Al poner el reset a 0 va directamente al estado inicial, verde en 1 y rojo en dos, por tanto, los coches circularán por la vía principal. Al activar el pulsador cambia automáticamente al siguiente estado.

Definiremos un temporizador, de manera que al estar el reset en 0, la cuenta es nula, pero al presionar el pulsador comienza la cuenta. Al contar 6, activamos la señal “amb” a 1. En el segundo estado cuando la cuenta llegue a 14 pasará al tercer estado y al contar 20 regresa al estado inicial.

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;
USE IEEE.STD_LOGIC_ARITH.ALL;
library UNISIM;
use UNISIM.VComponents.all;

entity semaforocruce2 is
    Port (
        clk, clkcount, reset, button: in std_logic;
        red1, green1, yellow1, red2, green2, yellow2: out std_logic
     );
end semaforocruce2;

architecture Behavioral of semaforocruce2 is
type STATES is (S0, S1, S2, S3);
signal current_state, next_state: STATES;
signal reloj: std_logic;
signal count: integer range 0 to 20;
signal amb: std_logic;
begin

    mix_state: process (clk, reset, button)
    begin
        if reset = '0' then
            current_state <= S0;
        elsif rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;
```

Reloj para el temporizador:

```vhdl
    tempo: process(clkcount, reset)
    begin
        if reset ='0' then
            count <= 0;
         elsif rising_edge(clkcount) then
            count<=count+1;
            if count = 6 then
                amb <= '1';
            else
                amb <= '0';
            end if;
        end if;
    end process;
```

Defición de los estados:

```
45      output_decode: process (current_state)
46      begin
47          case (current_state) is
48              when S0 =>
49                  red1 <= '0';
50                  red2 <= '1';
51                  yellow1 <= '0';
52                  yellow2 <= '0';
53                  green1 <= '1';
54                  green2 <= '0';
55              when S1 =>
56                  red1 <= '0';
57                  red2 <= '1';
58                  yellow2 <= '0';
59                  green1 <= '0';
60                  green2 <= '0';
61                  yellow1 <= '1';
62              when S2 =>
63                  red1 <= '1';
64                  red2 <='0';
65                  yellow1 <= '0';
66                  yellow2 <= '0';
67                  green1 <= '0';
68                  green2 <= '1';
69              when S3 =>
70                  red1 <= '1';
71                  red2 <='0';
72                  yellow1 <= '0';
73                  green1 <= '0';
74                  green2 <= '0';
75                  yellow2 <= '1';
76          end case;
77      end process;
```

Cambio de estados:

```
78      mxstate_dec:process(current_state, button, count)
79      begin
80          next_state <= current_state;
81          case current_state is
82              when S0 =>
83                  if button = '1' then
84                      next_state <= S1;
85                  end if;
86              when S1 =>
87                  if amb = '1' then
88                      next_state <= S2;
89                  end if;
90              when S2 =>
91                  if count = 14 then
92                      next_state <= S3;
93                  end if;
94              when S3 =>
95                  if count = 20 then
96                      next_state <= S0;
97                  end if;
98              end case;
99      end process;
100 end Behavioral;
```

Código top:

Este código tiene como función instanciar todos los módulos creados con anterioridad.

Definición de la entidad:

```
22  library IEEE;
23  use IEEE.STD_LOGIC_1164.ALL;
24
25  -- Uncomment the following library declaration if using
26  -- arithmetic functions with Signed or Unsigned values
27  --use IEEE.NUMERIC_STD.ALL;
28
29  -- Uncomment the following library declaration if instantiating
30  -- any Xilinx leaf cells in this code.
31  --library UNISIM;
32  --use UNISIM.VComponents.all;
33
34  entity top is
35  Port (
36      clk, reset, button: in std_logic;
37      red1, green1, yellow1, red2, green2, yellow2: out std_logic
38  );
39  end top;
```

Componentes y señales:

```
41  architecture Behavioral of top is
42      component sincronizador
43          port(
44              sync_in: in std_logic;
45              clk: in std_logic;
46              sync_out: out std_logic
47          );
48      end component;
49      component divisordefrecuencia
50      port(
51          clk:    in STD_LOGIC;
52          new_clk:        out STD_LOGIC
53      );
54      end component;
55      component semaforocruce2
56      port(
57          clk, clkcount, reset, button: in std_logic;
58          red1, green1, yellow1, red2, green2, yellow2: out std_logic
59      );
60      end component;
61
62      signal new_clk: std_logic;
63      signal button_sync: std_logic;
```

Instanciación de los módulos:

```
begin
    inst_clk_divider: divisordefrecuencia port map(
        clk => clk,
        new_clk => new_clk
    );
    inst_semaforo: semaforocruce2 port map(
        clk => clk,
        clkcount => new_clk,
        reset => reset,
        button => button,
        red1 => red1,
        green1 => green1,
        yellow1 => yellow1,
        red2 => red2,
        yellow2 => yellow2,
        green2 => green2
    );
end Behavioral;
```

Código divisor de frecuencia:

El reloj de la palca es de 100 MHz por lo que daría 100 millones de flancos positivos del reloj en un segundo, por lo tanto, hago el divisor para que de un flanco positivo de reloj en cada segundo.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
entity divisordefrecuencia is
    generic (
        frec : integer := 100000000
    );
    Port (
        clk: in std_logic;
        new_clk: out std_logic
    );
end divisordefrecuencia;

architecture Behavioral of divisordefrecuencia is
signal n_clk: std_logic := '0';
begin
process (clk, n_clk)
    variable count: integer := 0;
    begin
        if rising_edge(clk) then
            count := count+1;
            if count = (frec/2) then
                n_clk <= not n_clk;
                count := 0;
            end if;
        end if;
        new_clk <= n_clk;
    end process;
end Behavioral;
```