UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria
de Manresa

Pràctica 7:
Semàfor

Luca Di Iorio Casellas,
Victor Barbero Alcaide
2020 SisDig

Índex

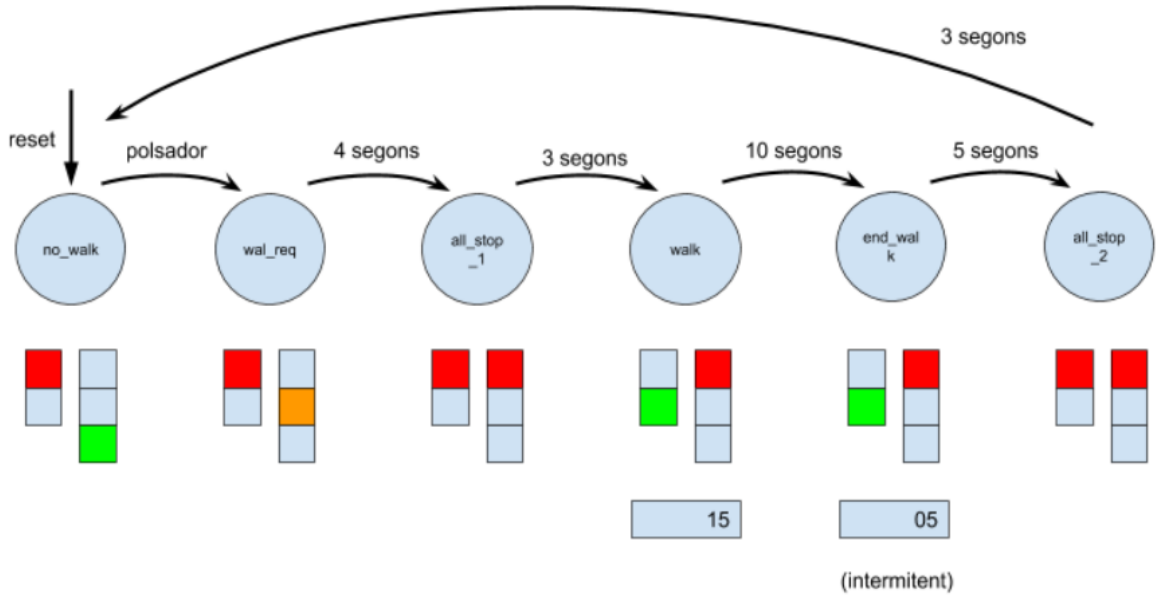
1 Introducció

L'objectiu d'aquesta pràctica és fer la descripció de hardware d'un dispositiu que controli un semàfor per a vianants.

El dispositiu controlarà els tres discs del semàfor de circulació de vehicles: verd (dg), ambre (da) i vermell (dr). Així com els indicadors per als vianants: verd (wg) i vermell (wr). A més, tindrà un indicador numèric que informarà del temps que els queda als vianants per a creuar el carrer. Disposarà d'un polsador de petició de pas per als vianants (p), que al ser activat s'iniciarà una seqüència temporitzada que es podrà implementar en la descripció mitjançant una màquina d'estats finits.

2 Desenvolupament

Un primer esquema en forma de diagrama de blocs del que pretenem assolir en l'elaboració de la màquina d'estats que representarà el fluxe del semàfor és el següent:



2.1 Top Level

Aquesta és la entitat que ens farà de mare del projecte. És a dir, és el projecte central del semàfor. En aquest, utilitzarem tant codi propi com altres components, per assolir les especificacions de la pràctica anteriorment esmentades.

Codi:

Primer, definim les llibreries del nostre projecte.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

A continuació, definim la entitat, amb les seves corressponents entrades i sortides.

```
entity pr7 is
    Port (
        polsador : in std_logic;
        clk, reset : in STD_LOGIC;
        cat : out STD_LOGIC_VECTOR (7 downto 0);
        an : out STD_LOGIC_VECTOR (3 downto 0);
        LED : out STD_LOGIC_VECTOR (15 downto 0)
    );
end pr7;
```

Dins l'arquitectura, definim tant els senyals interns com els components que utilitzarem.

```
architecture Behavioral of pr7 is

    component clk_divider
        generic(eoc: integer := 1000000);
        Port ( clk : in STD_LOGIC;
               reset : in STD_LOGIC;
               clk_div : out STD_LOGIC);
    end component;
    component counter
        generic(num : std_logic_vector (4 downto 0));
        Port ( rst_en : in std_logic;
              enable : in STD_LOGIC;
               clk_div, reset : in STD_LOGIC;
               c : out STD_LOGIC_VECTOR (4 downto 0));
    end component;
    component seg7_coder
    Port ( char0 : in STD_LOGIC_VECTOR (3 downto 0);
           char1 : in STD_LOGIC_VECTOR (3 downto 0);
           char2 : in STD_LOGIC_VECTOR (3 downto 0);
           char3 : in STD_LOGIC_VECTOR (3 downto 0);
           clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           clk_div : in STD_LOGIC;
           cat : out STD_LOGIC_VECTOR (7 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
    end component;
    component bin2bcd
    generic(n: positive := 16);
    Port ( bin_in : in STD_LOGIC_VECTOR (n-1 downto 0);
           clk : in STD_LOGIC;
           reset : in STD_LOGIC;
           bcd0 : out STD_LOGIC_VECTOR (3 downto 0);
           bcd1 : out STD_LOGIC_VECTOR (3 downto 0);
           bcd2 : out STD_LOGIC_VECTOR (3 downto 0);
           bcd3 : out STD_LOGIC_VECTOR (3 downto 0);
           bcd4 : out STD_LOGIC_VECTOR (3 downto 0));
    end component;

    type state is (no_walk, wal_req, all_stop_1 , walk, end_walk, all_stop_2 );
    signal state_reg , state_next : state;
    signal c, c_b : std_logic_vector (4 downto 0);
    signal c0, c1 : std_logic_vector (3 downto 0);
    signal enable, enable_b : std_logic ;
    signal disp0, disp1, char0, char1 : std_logic_vector (3 downto 0);
    signal rst_en : std_logic ;
```

```
    signal clk_div_1s : std_logic;
    signal clk_div: std_logic;
    signal clk_div_blink : std_logic;
    signal blink : std_logic;
    alias L0 : STD_LOGIC is LED(15);
    alias L1 : STD_LOGIC is LED(14);
    alias L2 : STD_LOGIC is LED(13);
    alias L3 : STD_LOGIC is LED(12);
    alias L4 : STD_LOGIC is LED(11);

begin
```

2.1.1 Maquina d'estats:

El codi corresponent a la màquina d'estats consta de tres processos: un descodificador d'estats, un descodificador de sortides i un registre de sortides.

```
status_decoder: process(clk, reset) begin
if rising_edge(clk) then
    if reset='1' then state_reg <= no_walk;
    else state_reg <= state_next;
    end if;
end if;
end process;

output_decoder: process(state_reg) begin
case state_reg is
    when no_walk =>
        L0 <= '1';
        L1 <= '0';
        L2 <= '0';
        L3 <= '0';
        L4 <= '1';
        enable <= '0';
        enable_b <= '0';
        rst_en <= '1';
    when wal_req =>
        L0 <= '1';
        L1 <= '0';
        L2 <= '0';
        L3 <= '1';
        L4 <= '0';
        enable <= '1';
        enable_b <= '0';
        rst_en <= '0';
    when all_stop_1 =>
```

```
            L0 <= '1';
            L1 <= '0';
            L2 <= '1';
            L3 <= '0';
            L4 <= '0';
            enable <= '1';
            enable_b <= '0';
            rst_en  <= '0';
        when walk =>
            L0 <= '0';
            L1 <= '1';
            L2 <= '1';
            L3 <= '0';
            L4 <= '0';
            enable <= '1';
            enable_b <= '1';
            rst_en  <= '0';
        when end_walk =>
            L0 <= '0';
            L1 <= blink;
            L2 <= '1';
            L3 <= '0';
            L4 <= '0';
            enable <= '1';
            enable_b <= '1';
            rst_en  <= '0';
        when all_stop_2 =>
            L0 <= '1';
            L1 <= '0';
            L2 <= '1';
            L3 <= '0';
            L4 <= '0';
            enable <= '1';
            enable_b <= '0';
            rst_en  <= '0';
    end case;
    end process;

     status_register : process(state_reg, polsador, c0, c1) begin
    state_next <= state_reg;
    case state_reg is
        when no_walk =>
            if polsador='1' then state_next <= wal_req;
            else state_next <= no_walk;
            end if;
```

```
    when wal_req =>
        if c0="0001" and c1="0010" then state_next <= all_stop_1;
        else state_next <= wal_req;
        end if;
    when all_stop_1 =>
        if c0="1000" and c1="0001" then state_next <= walk;
        else state_next <= all_stop_1;
        end if;
    when walk =>
        if c0="1000" and c1="0000" then state_next <= end_walk;
        else state_next <= walk;
        end if;
    when end_walk =>
        if c0="0011" and c1="0000" then state_next <= all_stop_2;
        else state_next <= end_walk;
        end if;
    when all_stop_2 =>
        if c0="0000" and c1="0000" then state_next <= no_walk;
        else state_next <= all_stop_2;
        end if;
end case;
end process;
```

2.1.2 Comptadors:

En aquesta pràctica, utilitzarem dos processos comptadors, que es basen en un únic component que descriurem més endavant. El primer comptador, serveix per mantenir el flux de la màquina d'estats. El segon, per visualitzar per pantalla el temps en que el semàfor de vianants està en verd, permeten el pas.

El codi corresponent la declaració dels components és el següent:

```
u2: counter
    generic map(num => "11001")
    port map (rst_en => rst_en,
              enable => enable,
              clk_div => clk_div_1s,
              reset => reset,
              c => c);
u7: counter
    generic map(num => "01111")
    port map (rst_en => rst_en,
              enable => enable_b,
              clk_div => clk_div_1s,
              reset => reset,
              c => c_b);
```

Veiem com els components utilitzen una senyal de rellotge dividida per generar un pols cada segon. El procés divisor de rellotge, com ja hem vist en pràctiques anteriors i no aprofundirem més en aquesta pràctica, es declara de la següent forma:

```
u1: clk_divider
    generic map (eoc => 100000000)
    port map (clk => clk,
              reset => reset,
              clk_div => clk_div_1s);
```

2.1.3 Conversió de binari a BCD:

Els processos comptadors anteriors generen com a resultat un nombre binari. Per a poder-lo visualitzar pels displays, necessitem que aquest sigui del format BCd. Per tant, utilitzarem convertidors de binari a BCD, que ja hem descrit en profunditat en pràctiques anteriors. A més a més, per a simplificar la comprensió del codi, hem decidit que el procés comptador que utilitzem per mantenir el flux d'execució de la màquina d'estats també l'utilitzarem amb BCD en lloc de binari i, per tant, necessitem un compnent conversor.

Aleshores, el codi de la crida dels components és el següent:

```
u5: bin2bcd
    generic map (n => 5)
    port map (bin_in => c,
              clk => clk,
              reset => reset,
              bcd0 => c0,
              bcd1 => c1,
              bcd2 => open,
              bcd3 => open,
              bcd4 => open);

u8: bin2bcd
    generic map (n => 5)
    port map (bin_in => c_b,
              clk => clk,
              reset => reset,
              bcd0 => disp0,
              bcd1 => disp1,
              bcd2 => open,
              bcd3 => open,
              bcd4 => open);
```

2.1.4 Intermitencia:

Com diu a l'enunciat de la practica, hi ha un estat de la màquina d'estats en que, tant el LED del semàfor de vianants com el temps als displays, es mostra de forma intermitent

per avisar que s'aproxima el final d'estat. Per aconseguir-ho, necessitem primer de tot crear un tren d'ones a la freqüència desitjada.

Per tant, utilitzem un divisor de rellotge i un procés que generi el tren de polsos esmentat, el senyal del qual hem anomenat 'blink'. El codi és el següent:

```
u6: clk_divider
    generic map (eoc => 10000000)
    port map (clk => clk,
              reset => reset,
              clk_div => clk_div_blink);

process( clk_div_blink ) begin
if rising_edge ( clk_div_blink ) then blink <= NOT blink;
end if ;
end process;
```

2.1.5 Visualitació als displays:

Per visualitzar el temps al display, necessitem el component que ja hem vist en pràctiques anteriors. A més a més, utilitzem un altre divisor de rellotge per aconseguir la freqüència adequada.

El codi és el següent:

```
u3: clk_divider
    generic map (eoc => 100000)
    port map (clk => clk,
              reset => reset,
              clk_div => clk_div);

u4: seg7_coder
    port map (char0 => char0,
              char1 => char1,
              char2 => "1111",
              char3 => "1111",
              clk => clk,
              reset => reset,
              clk_div => clk_div,
              cat => cat,
              an => an);
```

Com hem vist al subapartat anterior, hi ha un estat que requereix la intermitència del valor visualitzat pels displays. Per aconseguir-ho, utiltzem a següent assignació:

```
char0 <= disp0 when state_reg=walk else
         disp0 when state_reg=end_walk and blink='0' else
         "1111";
```

```
char1 <= disp1 when state_reg=walk else
         disp1 when state_reg=end_walk and blink='0' else
         "1111";

end Behavioral;
```

3 Comptador

Aquest component és l'encarregat de realitzar un compte enrere.

Com hem esmentat anteriorment, l'utilitzarem dues vegades: la primera, per mantenir el flux d'execució de la màquina d'estats. Quan premem el polsador, aleshores anem canviant d'estat segons un determinat temps que ha de transcòrrer. Una forma de mantenir el temps transcorregut sense utilitzar un comptador per cada estat és fer un compte enrrere del total de temps.L'altre ús que li donarem serà pel compte enrrere del temps de pas dels vianants.

Codi:

Primer, definim les llibreries del nostre projecte.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
```

A continuació, definim la entitat, amb les seves corressponents entrades i sortides.

```
entity counter is
    generic(num : std_logic_vector (4 downto 0));
    Port (
        rst_en : in std_logic;
        enable : in std_logic;
        clk_div, reset : in STD_LOGIC;
        c: out std_logic_vector (4 downto 0));
end counter;
```

I finalment elaborem el cos del component.

```
architecture Behavioral of counter is
begin
```

El cos del component consta d'un únic procés de compte enrere des d'un nombre genèric en binari fins a 0. Com a toc d'atenció, cal esmentar el senyal 'rst-en' que s'utilitza en aquest cas per mantenir el sincronisme davant les les repeticions de la màquina d'estats. Sense aquest senyal que actua com un habilitador de lectura de dades, es produeix un retard d'un segon, cosa que és necessària ja que els procéssos necessiten un sincronisme.

```
down_counter: process(clk_div, reset)
variable cnt : std_logic_vector (4 downto 0) := num;
begin
if reset='1' or rst_en='1' then cnt := num;
elsif rising_edge(clk_div) then
    if enable='1' then
        if cnt="00000" then cnt := num;
        else cnt := std_logic_vector (to_unsigned(to_integer(unsigned(cnt)) -1, 5));
        end if;
    end if;
end if;
c <= cnt;
end process;

end Behavioral;
```

4 Funcionament:

Per exemplificar el seu funcionament, adjuntem una imatge d'un dels estats: