Práctica

Martín Romera Sobrado
Ingeniería de Computadores 3
Centro Asociado de la UNED en Bizkaia
email: mromera95@alumno.uned.es

21 de septiembre de 2020

1. Ejercicio 1

En este ejercicio desarrollaremos un circuito que sigue la siguiente tabla para las entradas `x`, `y` y `z` y las salidas `F1` y `F2`:

x	y	z	F1	F2
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Cuadro 1: Tabla de verdad del circuito del Ejercicio 1

1.1. Apartado A

En este apartado se nos pide desarrollar la `entity` del circuito, definida por 3 entradas (`x`, `y` y `z`) y dos salidas (`F1` y `F2`). El código de este apartado es el siguiente:

```
----
-- ej1_a.vhd
-- Código relacionado al apartado a del ejercicio 1
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es

library IEEE;
use IEEE.std_logic_1164.all;

entity ej1 is
  port(
    x,y,z : in  std_logic;
    F1,F2 : out std_logic);
end entity ej1;
```

1.2. Apartado B

En este apartado se pide desarrollar la `architecture` para la `entity` del apartado anterior, definiendo el comportamiento del circuito. Para ello definimos las siguientes funciones booleanas:

$$F1 = yz + x(y + z)$$
$$F2 = \overline{x}(y \oplus z) + x\overline{(y \oplus z)}$$

En lo que se traduce en el siguiente código vhdl:

```
----
-- ej1_b.vhd
-- Código relacionado al apartado b del ejercicio 1
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es

library IEEE;
use IEEE.std_logic_1164.all;

architecture comportamiento of ej1 is
begin
  F1 <= ( ( y and z ) ) or ( x and ( y or z ) );
  F2 <= ( not x and ( y xor z ) ) or ( x and not ( y xor z ) );
end architecture comportamiento;
```

1.3. Apartado C

Para este apartado tenemos que diseñar un circuito con puertas lógicas que defina las funciones lógicas de la tabla 1. El circuito resultante es el siguiente:

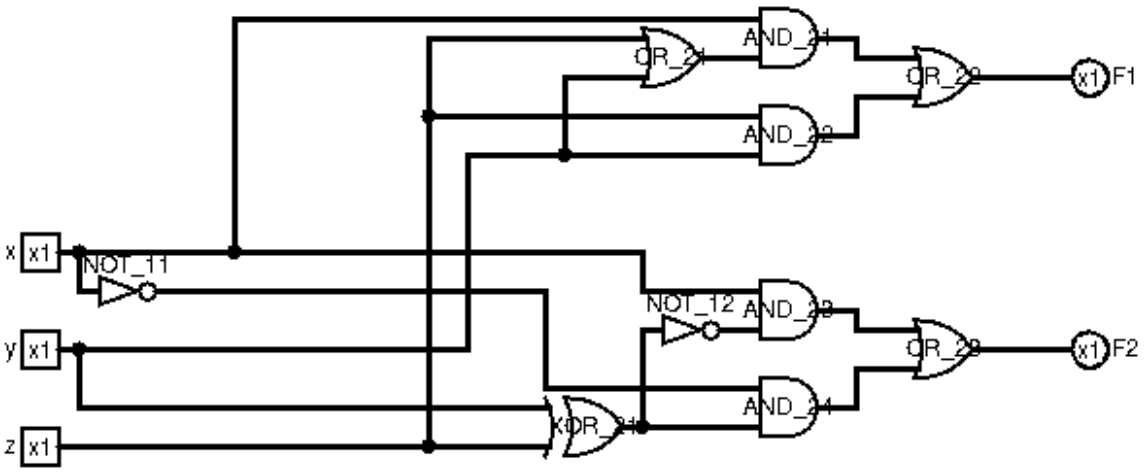


Figura 1: Circuito para el Apartado C del Ejercicio 1

A partir de este circuito tenemos que definir la `entity` y `architecture` de cada una de las puertas lógicas del circuito en vhdl. El código es el siguiente:

```
----
-- ej1_c.vhd
-- Código relacionado al apartado c del ejercicio 1
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es
```

```
library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica NOT de 1 entrada

entity not1 is
  port(
    a : in std_logic;
    o : out std_logic
    );
end entity not1;

architecture not1 of not1 is
begin
  o <= not a;
end architecture not1;

library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica AND de 2 entradas

entity and2 is
  port(
    a,b : in std_logic;
    o   : out std_logic
    );
end entity and2;

architecture and2 of and2 is
begin
  o <= a and b;
end architecture and2;

library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica OR de 2 entradas

entity or2 is
  port(
    a,b : in std_logic;
    o   : out std_logic
    );
end entity or2;

architecture or2 of or2 is
begin
```

```
  o <= a or b;
end architecture or2;

library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica XOR de 2 entradas

entity xor2 is
  port(
    a,b : in std_logic;
    o   : out std_logic
    );
end entity xor2;

architecture xor2 of xor2 is
begin
  o <= a xor b;
end architecture xor2;
```

1.4. Apartado D

Aprovechando las puertas lógicas del anterior apartado, en este las utilizaremos para definir `architecture` de la estructura de la `entity` del apartado 1. Siguiendo el circuito de la figura 1 obtenemos el siguiente código:

```
----
-- ej1_d.vhd
-- Código relacionado al apartado d del ejercicio 1
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es

library IEEE;
use IEEE.std_logic_1164.all;

architecture estructura of ej1 is

  -- Declaración de los componentes de la estructura

  component not1 is
    port(
      a : in std_logic;
      o : out std_logic
      );
  end component not1;

  component and2 is
    port(
```

```
      a,b : in std_logic;
      o   : out std_logic
      );
  end component and2;

  component or2 is
    port(
      a,b : in std_logic;
      o   : out std_logic
      );
  end component or2;

  component xor2 is
    port(
      a,b : in std_logic;
      o   : out std_logic
      );
  end component xor2;

  -- Declaración de señales auxiliares

  signal or_yz, and_yz, and_x_or_yz, not_x, xor_yz, not_xor_yz, and_not_x_xor_yz,
  ↪  and_x_not_xor_yz : std_logic;

begin
  -- Circuito F1
  Or_21  : or2  port map (y,z,or_yz);
  And_22 : and2 port map (y,z,and_yz);
  And_21 : and2 port map (x,or_yz,and_x_or_yz);
  Or_22  : or2  port map (and_yz,and_x_or_yz,F1);
  -- Circuito F2
  Not_11 : not1 port map (x, not_x);
  Xor_21 : xor2 port map (y,z,xor_yz);
  Not_12 : not1 port map (xor_yz,not_xor_yz);
  And_23 : and2 port map (x,not_xor_yz,and_x_not_xor_yz);
  And_24 : and2 port map (not_x,xor_yz,and_not_x_xor_yz);
  Or_23  : or2  port map (and_x_not_xor_yz,and_not_x_xor_yz,F2);
end architecture estructura;
```

1.5. Apartado E

Finalmente en este apartado se nos pide comprobar el correcto funcionamiento de ambas `architecture` mediante un *testbench*, y comprobando visualmente los cronogramas resultantes. El código del testbench es el siguiente:

```
----
-- ej1_e.vhd
-- Código relacionado al apartado e del ejercicio 1
```

```
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es

library IEEE;
use IEEE.std_logic_1164.all;

entity bp_ej1 is
end entity bp_ej1;

architecture bp_ej1 of bp_ej1 is
  -- Señales para la depuración
  signal x,y,z : std_logic; -- Entradas
  signal F1,F2 : std_logic; -- Salida

  -- Componente del ejercicio 1
  component ej1 is
    port(
      x,y,z : in std_logic;
      F1,F2 : out std_logic
      );
  end component ej1;

begin
  -- Definir el componente del ejercicio 1
  uut : component ej1 port map (x,y,z,F1,F2);

  -- Proceso del banco de pruebas
  test : process
  begin
    x <= '0'; y <= '0'; z <= '0';
    wait for 10 ns;
    x <= '0'; y <= '0'; z <= '1';
    wait for 10 ns;
    x <= '0'; y <= '1'; z <= '0';
    wait for 10 ns;
    x <= '0'; y <= '1'; z <= '1';
    wait for 10 ns;
    x <= '1'; y <= '0'; z <= '0';
    wait for 10 ns;
    x <= '1'; y <= '0'; z <= '1';
    wait for 10 ns;
    x <= '1'; y <= '1'; z <= '0';
    wait for 10 ns;
    x <= '1'; y <= '1'; z <= '1';
    wait for 10 ns;
    report "Fin de la simulación";
    wait;
  end process test;
```

```
end architecture bp_ej1;
```

Para compilar y simularlo, en mi caso hago uso de *ghdl* junto al *Makefile* que he escrito para facilitarme la tarea. Para observar los cronogramas desde los archivos *vcd*, uso el programa *GTKWave*. Los resultados son los siguientes:

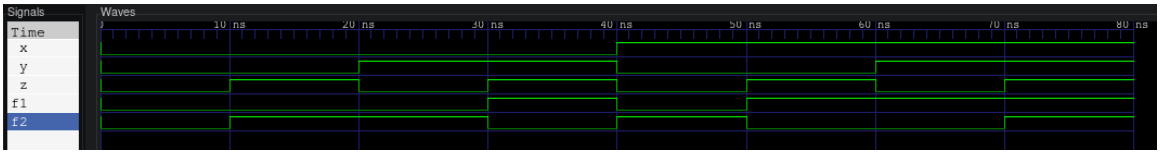


Figura 2: Cronograma del ejercicio 1 con la `architecture` apartado b

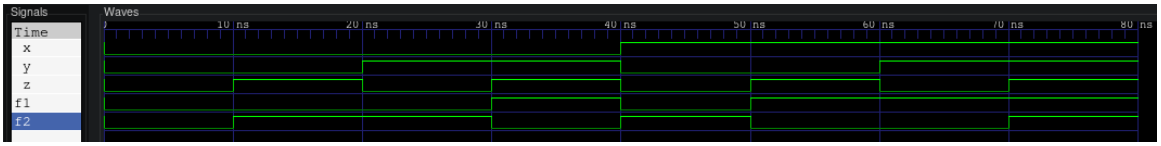


Figura 3: Cronograma del ejercicio 1 con la `architecture` apartado d

Si comprobamos ambos cronogramas con la tabla de verdad del cuadro 1 vemos que el circuito funciona correctamente.

2. Ejercicio 2

Para este ejercicio se nos propone un circuito como el siguiente:

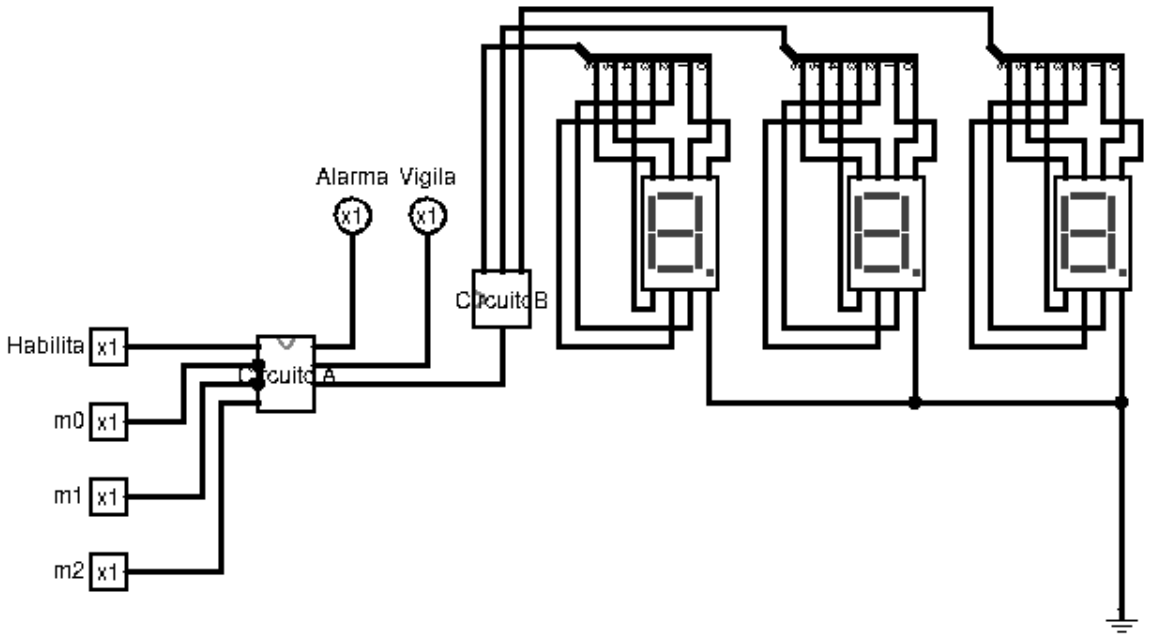
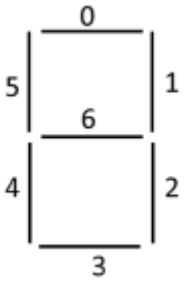


Figura 4: Circuito del ejercicio 2

Se trata de un circuito de la alarma de un museo que funciona de la siguiente manera:
Las señales `m0`, `m1` y `m2` corresponde a sensores de movimiento de las 3 habitaciones del museo. Si ninguno detecta movimiento (todos a '0'), entonces se activará la señal "vigila", que dirá al guarda que tiene que vigilar. La señal habilita indica si la alarma está encendida, la cual se activa si detecta movimiento en al menos dos habitaciones. Para indicar que la alarma está encendida, el `Circuito A` manda la señal "00" al `Circuito B` el cual es el responsable de de mostrar la palabra `On` en los 7 segmentos. En el caso de que la señal habilita esté a '0', la alarma estará apagada haciendo que el `Circuito A` mande la señal "00" al `Circuito B` mostrando este la palabra `Off` por los 7 segmentos. Los 7 segmentos se controlan mediante un vector de señales lógicas de 7 bits de la siguiente manera:



2.1. Apartado A

En este apartado tenemos que definir la estructura del `Circuito A` elabora las `entity` y `architecture` de cada una de las puertas lógicas, y usando estas últimas la `entity` y `architecture` del circuito.
El diagrama del circuito sería el siguiente:

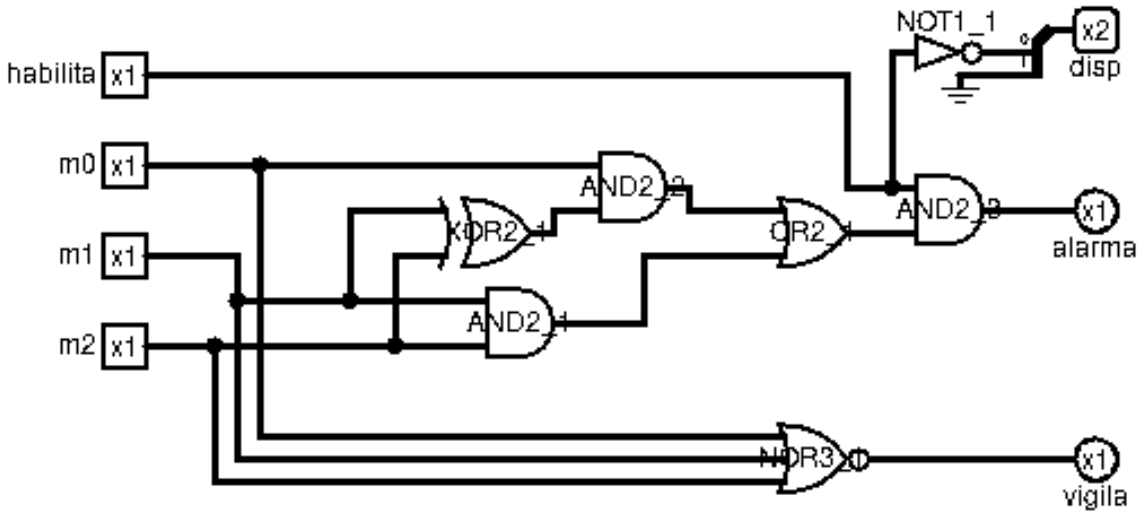


Figura 5: Circuito A

De lo que desarrollamos el siguiente código en vhdl:

```
----
-- ej2_a.vhd
-- Código relacionado al apartado a del ejercicio 2
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es

library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica NOT de una entrada

entity not1 is
  port(
    a : in std_logic;
    o : out std_logic
    );
end entity not1;

architecture not1 of not1 is
begin
  o <= not a;
```

```
end architecture not1;

library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica OR de dos entradas

entity or2 is
  port(
    a,b : in std_logic;
    o   : out std_logic
    );
end entity or2;

architecture or2 of or2 is
begin
  o <= a or b;
end architecture or2;

library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica AND de dos entradas

entity and2 is
  port(
    a,b : in std_logic;
    o   : out std_logic
    );
end entity and2;

architecture and2 of and2 is
begin
  o <= a and b;
end architecture and2;

library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica AND de tres entradas

entity xor2 is
  port(
    a,b : in std_logic;
    o : out std_logic
    );
end entity xor2;
```

```
architecture xor2 of xor2 is
begin
  o <= a xor b;
end architecture xor2;

library IEEE;
use IEEE.std_logic_1164.all;

-- Puerta lógica NOR de tres entradas

entity nor3 is
  port(
    a,b,c : in std_logic;
    o : out std_logic
    );
end entity nor3;

architecture nor3 of nor3 is
begin
  o <= not ( a or b or c );
end architecture nor3;

library IEEE;
use IEEE.std_logic_1164.all;

-- GND

entity gnd is
  port(o : out std_logic);
end entity gnd;

architecture gnd of gnd is
begin
  o <= '0';
end architecture gnd;

library IEEE;
use IEEE.std_logic_1164.all;

-- Circuito A

entity ej2_circuitoA is
  port(
    habilita, m0, m1, m2 : in std_logic;
    alarma, vigila : out std_logic;
    disp : out std_logic_vector(1 downto 0)
    );
end entity ej2_circuitoA;
```

```
architecture ej2_circuitoA of ej2_circuitoA is

  component not1 is
    port(
      a : in std_logic;
      o : out std_logic
      );
  end component not1;

  component or2 is
    port(
      a,b : in std_logic;
      o : out std_logic
      );
  end component or2;

  component and2 is
    port(
      a,b : in std_logic;
      o : out std_logic
      );
  end component and2;

  component xor2 is
    port(
      a,b : in std_logic;
      o : out std_logic
      );
  end component xor2;

  component nor3 is
    port(
      a,b,c : in std_logic;
      o : out std_logic
      );
  end component nor3;

  component gnd is
    port(o : out std_logic);
  end component gnd;

  signal and2_1s, xor2_1s, and2_2s, or2_1s : std_logic;

begin

  And2_1 : and2 port map(m1,m2,and2_1s);
  Xor2_1 : xor2 port map(m1,m2,xor2_1s);
```

```
  And2_2 : and2 port map(m0,xor2_1s,and2_2s);
  Or2_1  : or2  port map(and2_2s, and2_1s, or2_1s);
  And2_3 : and2 port map(habilita, or2_1s, alarma);
  Nor3_1 : nor3 port map(m0,m1,m2,vigila);
  Not1_1 : not1 port map(habilita,disp(0));
  GND_1  : gnd  port map(disp(1));


end architecture ej2_circuitoA;
```

2.2. Apartado B

Para este apartado hay que definir la `entity` y `architecture` del `Circuito B`. Como este tiene una forma de "decodificador selector", haremos uso de un proceso con sentencias `case/when`:

```
----
-- ej2_b.vhd
-- Código relacionado al apartado B del ejercicio 2
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es


library IEEE;
use IEEE.std_logic_1164.all;

entity ej2_circuitoB is
  port(
    s_in : in std_logic_vector(1 downto 0); -- sin es una palabra reservada
    sout1, sout2, sout3 : out std_logic_vector(6 downto 0)
    );
end entity ej2_circuitoB;

architecture ej2_circuitoB of ej2_circuitoB is
begin

  circuito : process(s_in)
  begin
    case s_in is
      when "00" =>          -- ON
        sout1 <= "0000000";
        sout2 <= "0111111";
        sout3 <= "1010100";
      when "01" =>          -- OFF
        sout1 <= "0111111";
        sout2 <= "1110001";
        sout3 <= "1110001";
      when others =>        -- Caso imposible : Apagar display
        sout1 <= "0000000";
```

```
        sout2 <= "0000000";
        sout3 <= "0000000";
    end case;
  end process circuito;

end architecture ej2_circuitoB;
```

2.3. Apartado C

En este apartado pondremos en conjunto ambos circuitos para formar el circuito de la figura 4, con el siguiente código:

```
----
-- ej2_c.vhd
-- Código relacionado al apartado C del ejercicio 2
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es


library IEEE;
use IEEE.std_logic_1164.all;

entity ej2 is
  port(
    habilita, m0, m1, m2 : in std_logic;
    alarma, vigila : out std_logic;
    sout1, sout2, sout3 : out std_logic_vector(6 downto 0)
    );
end entity ej2;

architecture ej2 of ej2 is

  component ej2_circuitoA is
    port(
      habilita, m0, m1, m2 : in std_logic;
      alarma, vigila : out std_logic;
      disp : out std_logic_vector(1 downto 0)
      );
  end component ej2_circuitoA;

  component ej2_circuitoB is
    port(
      s_in : in std_logic_vector(1 downto 0); -- sin es una palabra reservada
      sout1, sout2, sout3 : out std_logic_vector(6 downto 0)
      );
  end component ej2_circuitoB;

  signal disp : std_logic_vector(1 downto 0);
```

```
begin

  CircuitoA : ej2_circuitoA port map(habilita, m0, m1, m2, alarma, vigila, disp);
  CircuitoB : ej2_circuitoB port map(disp, sout1, sout2, sout3);

end architecture ej2;
```

2.4. Apartado D

Finalmente, de manera similar al ejercicio anterior, desarrolaremos un *testbench* para comprobar el correcto funcionamiento del circuito entero, solo que este deberá de comprobar los casos mediante `asserts`:

```
----
-- ej2_d.vhd
-- Código relacionado al apartado D del ejercicio 2
-- Autor: Martín Romera Sobrado
-- Contacto: mromera95@alumno.uned.es


library IEEE;
use IEEE.std_logic_1164.all;

entity bp_ej2 is
end entity bp_ej2;

architecture bp_ej2 of bp_ej2 is

  -- Señales para depuración del utt
  signal habilita, m0, m1, m2 : std_logic;
  signal alarma, vigila : std_logic;
  signal sout1, sout2, sout3 :std_logic_vector(6 downto 0);

  -- Componente que vamos a testear
  component ej2 is
    port(
      habilita, m0, m1, m2 : in std_logic;
      alarma, vigila : out std_logic;
      sout1, sout2, sout3 : out std_logic_vector(6 downto 0)
      );
  end component ej2;

begin

  uut : ej2 port map(habilita, m0, m1, m2, alarma, vigila, sout1, sout2, sout3);

  test : process
```

```
begin

  habilita <= '0'; m0 <= '0'; m1 <= '0'; m2 <= '0';
  wait for 10 ns;
  assert (alarma = '0' and vigila = '1' and sout1 = "0111111" and sout2 =
  ↪  "1110001" and sout3 = "1110001")
    report "Falla para habilita <= 0; m0 <= 0; m1 <= 0; m2 <= 0" severity
    ↪  failure;


  habilita <= '0'; m0 <= '0'; m1 <= '0'; m2 <= '1';
  wait for 10 ns;
  assert (alarma = '0' and vigila = '0' and sout1 = "0111111" and sout2 =
  ↪  "1110001" and sout3 = "1110001")
    report "Falla para habilita <= 0; m0 <= 0; m1 <= 0; m2 <= 1" severity
    ↪  failure;


  habilita <= '0'; m0 <= '0'; m1 <= '1'; m2 <= '0';
  wait for 10 ns;
  assert (alarma = '0' and vigila = '0' and sout1 = "0111111" and sout2 =
  ↪  "1110001" and sout3 = "1110001")
    report "Falla para habilita <= 0; m0 <= 0; m1 <= 1; m2 <= 0" severity
    ↪  failure;


  habilita <= '0'; m0 <= '0'; m1 <= '1'; m2 <= '1';
  wait for 10 ns;
  assert (alarma = '0' and vigila = '0' and sout1 = "0111111" and sout2 =
  ↪  "1110001" and sout3 = "1110001")
    report "Falla para habilita <= 0; m0 <= 0; m1 <= 1; m2 <= 1" severity
    ↪  failure;


  habilita <= '0'; m0 <= '1'; m1 <= '0'; m2 <= '0';
  wait for 10 ns;
  assert (alarma = '0' and vigila = '0' and sout1 = "0111111" and sout2 =
  ↪  "1110001" and sout3 = "1110001")
    report "Falla para habilita <= 0; m0 <= 1; m1 <= 0; m2 <= 0" severity
    ↪  failure;


  habilita <= '0'; m0 <= '1'; m1 <= '0'; m2 <= '1';
  wait for 10 ns;
  assert (alarma = '0' and vigila = '0' and sout1 = "0111111" and sout2 =
  ↪  "1110001" and sout3 = "1110001")
    report "Falla para habilita <= 0; m0 <= 1; m1 <= 0; m2 <= 1" severity
    ↪  failure;
```

```
habilita <= '0'; m0 <= '1'; m1 <= '1'; m2 <= '0';
wait for 10 ns;
assert (alarma = '0' and vigila = '0' and sout1 = "0111111" and sout2 =
↪  "1110001" and sout3 = "1110001")
  report "Falla para habilita <= 0; m0 <= 1; m1 <= 1; m2 <= 0" severity
  ↪  failure;


habilita <= '0'; m0 <= '1'; m1 <= '1'; m2 <= '1';
wait for 10 ns;
assert (alarma = '0' and vigila = '0' and sout1 = "0111111" and sout2 =
↪  "1110001" and sout3 = "1110001")
  report "Falla para habilita <= 0; m0 <= 1; m1 <= 1; m2 <= 1" severity
  ↪  failure;


habilita <= '1'; m0 <= '0'; m1 <= '0'; m2 <= '0';
wait for 10 ns;
assert (alarma = '0' and vigila = '1' and sout1 = "0000000" and sout2 =
↪  "0111111" and sout3 = "1010100")
  report "Falla para habilita <= 1; m0 <= 0; m1 <= 0; m2 <= 0" severity
  ↪  failure;


habilita <= '1'; m0 <= '0'; m1 <= '0'; m2 <= '1';
wait for 10 ns;
assert (alarma = '0' and vigila = '0' and sout1 = "0000000" and sout2 =
↪  "0111111" and sout3 = "1010100")
  report "Falla para habilita <= 1; m0 <= 0; m1 <= 0; m2 <= 1" severity
  ↪  failure;


habilita <= '1'; m0 <= '0'; m1 <= '1'; m2 <= '0';
wait for 10 ns;
assert (alarma = '0' and vigila = '0' and sout1 = "0000000" and sout2 =
↪  "0111111" and sout3 = "1010100")
  report "Falla para habilita <= 1; m0 <= 0; m1 <= 1; m2 <= 0" severity
  ↪  failure;


habilita <= '1'; m0 <= '0'; m1 <= '1'; m2 <= '1';
wait for 10 ns;
assert (alarma = '1' and vigila = '0' and sout1 = "0000000" and sout2 =
↪  "0111111" and sout3 = "1010100")
  report "Falla para habilita <= 1; m0 <= 0; m1 <= 1; m2 <= 1" severity
  ↪  failure;
```

```
    habilita <= '1'; m0 <= '1'; m1 <= '0'; m2 <= '0';
    wait for 10 ns;
    assert (alarma = '0' and vigila = '0' and sout1 = "0000000" and sout2 =
    ↪  "0111111" and sout3 = "1010100")
      report "Falla para habilita <= 1; m0 <= 1; m1 <= 0; m2 <= 0" severity
      ↪  failure;


    habilita <= '1'; m0 <= '1'; m1 <= '0'; m2 <= '1';
    wait for 10 ns;
    assert (alarma = '1' and vigila = '0' and sout1 = "0000000" and sout2 =
    ↪  "0111111" and sout3 = "1010100")
      report "Falla para habilita <= 1; m0 <= 1; m1 <= 0; m2 <= 1" severity
      ↪  failure;


    habilita <= '1'; m0 <= '1'; m1 <= '1'; m2 <= '0';
    wait for 10 ns;
    assert (alarma = '1' and vigila = '0' and sout1 = "0000000" and sout2 =
    ↪  "0111111" and sout3 = "1010100")
      report "Falla para habilita <= 1; m0 <= 1; m1 <= 1; m2 <= 0" severity
      ↪  failure;


    habilita <= '1'; m0 <= '1'; m1 <= '1'; m2 <= '1';
    wait for 10 ns;
    assert (alarma = '1' and vigila = '0' and sout1 = "0000000" and sout2 =
    ↪  "0111111" and sout3 = "1010100")
      report "Falla para habilita <= 1; m0 <= 1; m1 <= 1; m2 <= 1" severity
      ↪  failure;


    report "Simulación del banco de pruebas finalizado";
    wait;
  end process test;

end architecture bp_ej2;
```

El cual al ejecutar, no hace que salte ningun error, logrando además el siguiente cronograma:

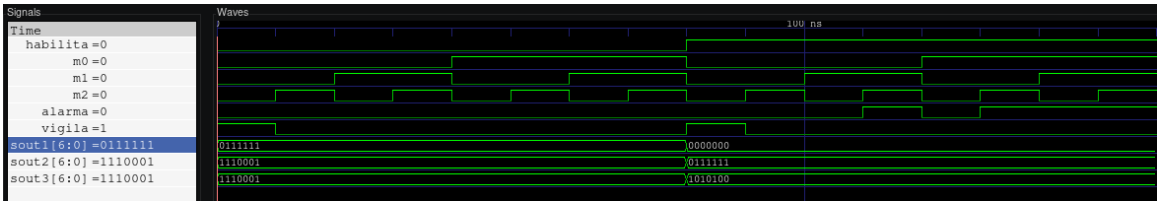


Figura 6: Cronograma del circuito completo del Ejercicio 2