Práctica 1 - El lenguage de descripción de hardware VHDL

Sistemas Digitales - FIUBA

April 26, 2021

Herramientas
EDAplayground: `https://www.edaplayground.com/`
GTKWave, GHDL: `https://www.youtube.com/watch?v=H2GyAIYwZbw`
Modelsim: `https://www.mentor.com/company/higher_ed/modelsim-student-edition`

Nota: En todos los ejercicios se deberá implementar una entidad de simulación y simular la resolución del mismo.

Ejercicio 1 - Implementar en VHDL las compuertas: AND, OR, XOR, AOI, OAI.

Ejercicio 2 - Implementar en VHDL un multiplexor de dos líneas de selección y 4 entradas de datos.

Ejercicio 3 - Implementar en VHDL un multiplexor de 4 líneas de selección y 16 entradas de datos. Cada entrada de datos debe ser de un ancho de N bits.

Ejercicio 4 - Implementar en VHDL un sumador sin signo genérico de N bits.

Ejercicio 5 - Implementar en VHDL un restador con sign genérico de N bits.

Ejercicio 6 - Implementar en VHDL un multiplicador sin signo genérico de N bits.

Ejercicio 7 - Implementar en VHDL un multiplicador en complemento a dos genérico de N bits.

Ejercicio 8 - Implementar en VHDL cuatro diferentes flip flops D con las siguientes entradas en cada caso:

- Reset asincrónico
- Reset sincrónico
- Set asincrónico
- Set sincrónico

Ejercicio 9 - Implementar en VHDL cuatro differents registros de *N* bits con las siguientes entradas en cada caso:

- Reset asincrónico
- Reset sincrónico
- Set asincrónico
- Set sincrónico

Ejercicio 10 - Implementar en VHDL un contador genérico de *N* bits.

Ejercicio 11 - Implementar en VHDL un contador genérico de *N* bits y señal de carga. Cuando dicha señal de carga sea '1', el valor de la cuenta pasará a ser el valor de una señal de entrada de *N* bits.

Ejercicio 12 - Implementar en VHDL un contador genérico de *N* bits y señal de *enable*. Cuando dicha señal de *enable* sea '0', el valor de la cuenta no deberá ser modificado.

Ejercicio 13 - Implementar en VHDL un contador genérico de *N* bits con señal de *enable* y señal de carga. Cuando la señal de *enable* sea '0', el valor de la cuenta no deberá ser modificado. Cuando la señal de carga sea '1', el valor de la cuenta pasará a ser el valor de una señal de entrada de *N* bits.

Ejercicio 14 - Implementar en VHDL un registro de desplazamiento de *N* bits de largo.

Ejercicio 15 - Implementar en VHDL un registro de desplazamiento genérico de *N* bits y señal de carga. Cuando dicha señal de carga sea '1', el contenido del registro pasará a ser el valor de una señal de entrada de *N* bits.

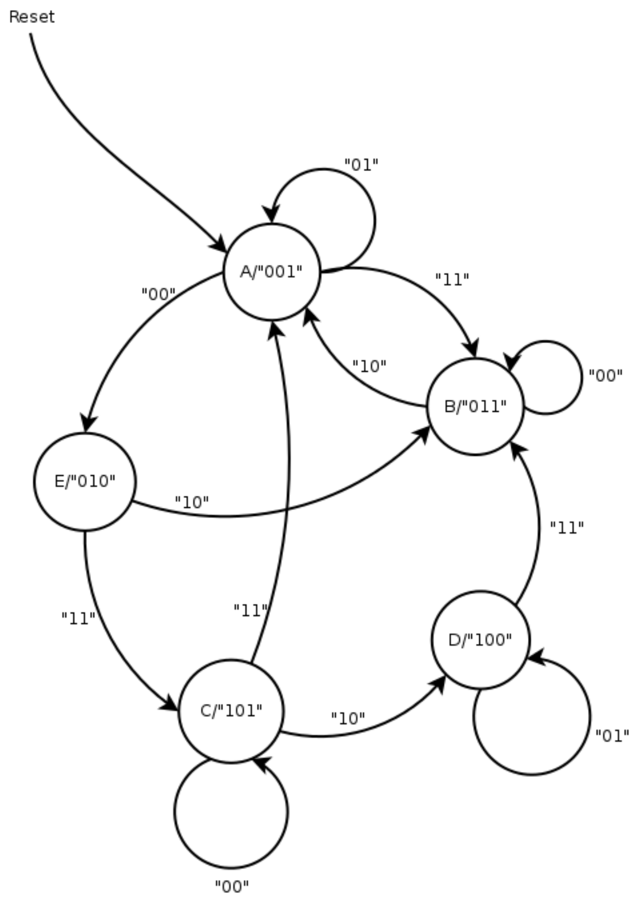
Ejercicio 16 - Implementar en VHDL un registro de desplazamiento genérico de *N* bits y señal de *enable*. Cuando dicha señal de *enable* sea

‘0’, el contenido del registro no deberá ser modificado. Implemetar también señal de carga.

Ejercicio 17 - Implementar en VHDL un registro de desplazamiento genérico de *N* etapas y de un ancho de palabra de *M* bits. El mismo deberá tener señal de carga y señal de *enable*.

Ejercicio 18 - Implementar en VHDL un circuito que detecte la secuencia “0010110” de una señal de entrada. Deberán relizarse tres implementaciones diferentes: mediante una FSM, mediante una FM on codificación one-hot y mediante un registro de desplazamiento y la lógica de detección adecuada.

Ejercicio 19 - Implementar en VHDL el circuito de Moore correspondiente al siguiente diagrama de estados:



Ejercicio 20 - Implementar en VHDL un detector de flancos. Este circuito tendrá dos salidas denominadas *rising* y *falling* las cuales serán '1' cuando en la señal de entrada se produzca un flanco de subida o de bajada respectivamente.

Ejercicio 1

Implementar en VHDL una compuerta AND.

Solución:

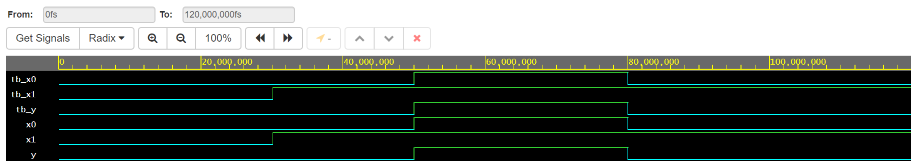
Diseño

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

entity gate_and is
    port ( x0  : in  std_logic;    -- AND gate input
           x1  : in  std_logic;    -- AND gate input
           y   : out std_logic     -- AND gate output
          );
end gate_and;

architecture behavioral of gate_and is
begin
    y <= x0 and x1;  -- 2 input AND gate
end behavioral;
```

Testbench

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

entity tb_gate_and is
end tb_gate_and;

architecture behavioral of tb_gate_and is

  component gate_and is
      port ( x0  : in  std_logic;    -- AND gate input
             x1  : in  std_logic;    -- AND gate input
             y   : out std_logic     -- AND gate output
      );
  end component;

    signal tb_x0   : std_logic;
    signal tb_x1   : std_logic;
    signal tb_y    : std_logic;

begin

  tb_x0  <= '0', '1' after 50 ns, '0' after 80 ns;
  tb_x1  <= '0', '1' after 30 ns, '0' after 120 ns;

  I1: gate_and port map (
    x0   => tb_x0,
    x1   => tb_x1,
    y    => tb_y
  );

end behavioral;
```

Simulación


From: 0fs
To: 120,000,000fs
Get Signals
Radix
100%
0
20,000,000
40,000,000
60,000,000
80,000,000
100,000,000
tb_x0
tb_x1
tb_y
x0
x1
y

Ejercicio 2

Implementar en VHDL un multiplexor de una línea de selección y dos entradas de datos.

Diseño

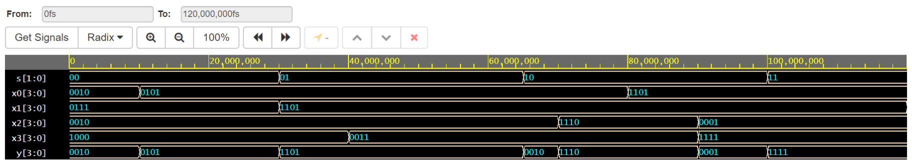
```
library ieee;
use ieee.std_logic_1164.all;

entity mux is
    port(
        x0 : in  std_logic_vector(3 downto 0);
        x1 : in  std_logic_vector(3 downto 0);
        x2 : in  std_logic_vector(3 downto 0);
        x3 : in  std_logic_vector(3 downto 0);
        s  : in  std_logic_vector(1 downto 0);
        y  : out std_logic_vector(3 downto 0)
    );
end mux;

architecture behavioral of mux is
begin
    process (x0, x1, x2, x3, s) is
    begin
        case s is
            when "00" =>
                y <= x0;
            when "01" =>
                y <= x1;
            when "10" =>
                y <= x2;
            when others =>
                y <= x3;
        end case;
    end process;
end behavioral;

architecture behavioral_2 of mux is
begin
    with s select y <=
        x0 when "00",
        x1 when "01",
```

```vhdl
        x2 when "10",
        x3 when others;
end behavioral_2;

architecture behavioral_3 of mux is
begin
    y <=
        x0 when s = "00" else
        x1 when s = "01" else
        x2 when s = "10" else
        x3;
end behavioral_3;
```

Testbench

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

entity tb_mux is
end tb_mux;

architecture behavioral of tb_mux is

    component mux is
        port(
            x0 : in  std_logic_vector(3 downto 0);
            x1 : in  std_logic_vector(3 downto 0);
            x2 : in  std_logic_vector(3 downto 0);
            x3 : in  std_logic_vector(3 downto 0);
            s  : in  std_logic_vector(1 downto 0);
            y  : out std_logic_vector(3 downto 0)
        );
    end component;

    signal tb_x0 : std_logic_vector(3 downto 0);
    signal tb_x1 : std_logic_vector(3 downto 0);
    signal tb_x2 : std_logic_vector(3 downto 0);
    signal tb_x3 : std_logic_vector(3 downto 0);
    signal tb_s  : std_logic_vector(1 downto 0);
    signal tb_y  : std_logic_vector(3 downto 0);

begin

  tb_x0 <= "0010", "0101" after 10 ns, "1101" after 80 ns;
  tb_x1 <= "0111", "1101" after 30 ns, "1100" after 120 ns;
  tb_x2 <= "0010", "1110" after 70 ns, "0001" after 80 ns;
  tb_x3 <= "1000", "0011" after 40 ns, "1111" after 90 ns;
  tb_s  <= "00"  , "01" after 30 ns, "10" after 65 ns, "11" after 100 ns;

  I1: mux
  port map (
    x0   => tb_x0,
    x1   => tb_x1,
    x2   => tb_x2,
    x3   => tb_x3,
    s    => tb_s,
    y    => tb_y
  );

end behavioral;
```

Simulación


From: 0fs
To: 120,000,000fs
Get Signals
Radix
100%
0
20,000,000
40,000,000
60,000,000
80,000,000
100,000,000
s[1:0]
x0[3:0]
x1[3:0]
x2[3:0]
x3[3:0]
y[3:0]
00
01
10
11
0010
0101
1101
0111
1101
0010
1110
0001
1000
0011
1111
0010
0101
1101
0010
1110
0001
1111

Ejercicio 3

Implementar en VHDL un multiplexor de 4 líneas de selección y 16 entradas de datos. Cada entrada de datos debe ser de un ancho de N bits.

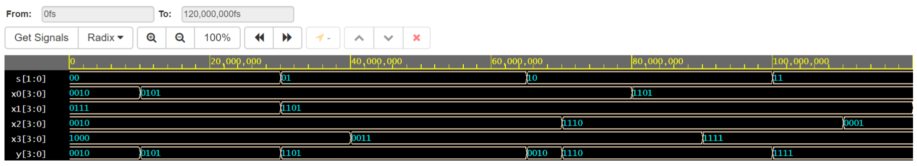
Diseño

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

entity mux is
    generic(N : natural := 8);
    port(
        x0 : in  std_logic_vector(N-1 downto 0);
        x1 : in  std_logic_vector(N-1 downto 0);
        x2 : in  std_logic_vector(N-1 downto 0);
        x3 : in  std_logic_vector(N-1 downto 0);
        s  : in  std_logic_vector(1   downto 0);
        y  : out std_logic_vector(N-1 downto 0)
    );
end mux;

architecture behavioral of mux is
begin
    process (x0, x1, x2, x3, s) is
    begin
        case s is
            when "00" =>
                y <= x0;
            when "01" =>
                y <= x1;
            when "10" =>
                y <= x2;
            when others =>
                y <= x3;
        end case;
    end process;
end behavioral;

architecture behavioral_2 of mux is
begin
    with s select y <=
        x0 when "00",
        x1 when "01",
        x2 when "10",
        x3 when others;
end behavioral_2;

architecture behavioral_3 of mux is
begin
    y <=
        x0 when s = "00" else
        x1 when s = "01" else
        x2 when s = "10" else
        x3;
end behavioral_3;
```

Testbench

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

entity tb_mux is
end tb_mux;

architecture behavioral of tb_mux is

    constant TB_N : natural := 4;

    signal tb_x0 : std_logic_vector(TB_N-1 downto 0);
    signal tb_x1 : std_logic_vector(TB_N-1 downto 0);
    signal tb_x2 : std_logic_vector(TB_N-1 downto 0);
    signal tb_x3 : std_logic_vector(TB_N-1 downto 0);
    signal tb_s  : std_logic_vector(1      downto 0);
    signal tb_y  : std_logic_vector(TB_N-1 downto 0);

begin

  tb_x0 <= "0010", "0101" after 10 ns, "1101" after 80 ns;
  tb_x1 <= "0111", "1101" after 30 ns, "1100" after 120 ns;
  tb_x2 <= "0010", "1110" after 70 ns, "0001" after 110 ns;
  tb_x3 <= "1000", "0011" after 40 ns, "1111" after 90 ns;
  tb_s  <= "00"  , "01" after 30 ns, "10" after 65 ns, "11" after 100 ns;

  I1: entity work.mux(behavioral)
  generic map (N => TB_N)
  port map (
    x0   => tb_x0,
    x1   => tb_x1,
    x2   => tb_x2,
    x3   => tb_x3,
    s    => tb_s,
    y    => tb_y
  );

end behavioral;
```

Simulación



From: 0fs To: 120,000,000fs

Ejercicio 4

Implementar en VHDL un sumador sin signo genérico de N bits.

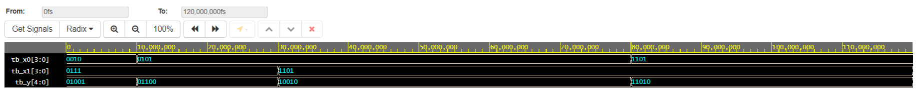
Diseño

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

entity adder is
    generic(N : natural := 8);
    port(
        x0 : in  std_logic_vector(N-1 downto 0);
        x1 : in  std_logic_vector(N-1 downto 0);
        y  : out std_logic_vector(N   downto 0)
    );
end adder;

architecture behavioral of adder is
    signal extended_x0 : unsigned(N downto 0);
    signal extended_x1 : unsigned(N downto 0);
begin
    extended_x0 <= '0' & unsigned(x0);
    extended_x1 <= '0' & unsigned(x1);
    y <= std_logic_vector(extended_x0 + extended_x1);
end behavioral;
```

Testbench

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

entity tb_adder is
end tb_adder;

architecture behavioral of tb_adder is

    constant TB_N : natural := 4;

    signal tb_x0 : std_logic_vector(TB_N-1 downto 0);
    signal tb_x1 : std_logic_vector(TB_N-1 downto 0);
    signal tb_y  : std_logic_vector(TB_N   downto 0);

begin

  tb_x0 <= "0010", "0101" after 10 ns, "1101" after 80 ns;
  tb_x1 <= "0111", "1101" after 30 ns, "1100" after 120 ns;

  I1: entity work.adder(behavioral)
  generic map (N => TB_N)
  port map (
    x0   => tb_x0,
    x1   => tb_x1,
    y    => tb_y
  );

end behavioral;
```

Simulación


From: 0fs
To: 120,000,000fs
Get Signals
Radix
100%
0
10,000,000
20,000,000
30,000,000
40,000,000
50,000,000
60,000,000
70,000,000
80,000,000
90,000,000
100,000,000
110,000,000
tb_x0[3:0] 0010 0101 1101
tb_x1[3:0] 0111 1101
tb_y[4:0] 01001 01100 10010 11010

Ejercicio 5

Implementar en VHDL un sumador con signo genérico de N bits.

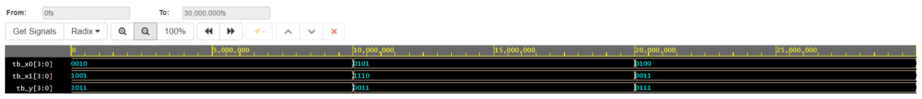
Diseño

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

entity adder is
    generic(N : natural := 8);
    port(
        x0 : in  std_logic_vector(N-1 downto 0);
        x1 : in  std_logic_vector(N-1 downto 0);
        y  : out std_logic_vector(N-1 downto 0)
    );
end adder;

architecture behavioral of adder is
begin
    y <= std_logic_vector(unsigned(x0) + unsigned(x1));
    -- Podría ser también con el tipo signed en lugar de
    -- unsigned. Notar que lo que cambia entre un sumador
    -- con signo y sin signo es el tamaño de la salida y.
    -- En el caso de signado, para que la representación
    -- complemento a 2 funcione, las dos entradas deben ser
    -- del mismo tamaño que la salida, mientras que en el
    -- caso de sin signo, la salida debe tener un bit más
    -- las entradas.
end behavioral;
```

Testbench

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

entity tb_adder is
end tb_adder;

architecture behavioral of tb_adder is

    constant TB_N : natural := 4;

    signal tb_x0 : std_logic_vector(TB_N-1 downto 0);
    signal tb_x1 : std_logic_vector(TB_N-1 downto 0);
    signal tb_y  : std_logic_vector(TB_N-1 downto 0);

begin

  tb_x0 <= "0010", "0101" after 10 ns, "0100" after 20 ns, "0000" after 30 ns;
  tb_x1 <= "1001", "1110" after 10 ns, "0011" after 20 ns, "0000" after 30 ns;

  I1: entity work.adder(behavioral)
  generic map (N => TB_N)
  port map (
    x0   => tb_x0,
    x1   => tb_x1,
    y    => tb_y
  );

end behavioral;
```

Simulación



```
tb_x0[3:0]  0010  0101  0100
tb_x1[3:0]  1001  1110  0011
tb_y[3:0]   1011  0011  0111
```

Ejercicio 6

Implementar en VHDL un multiplicador sin signo genérico de N bits.

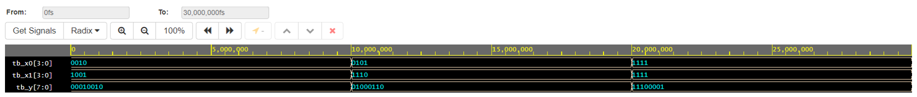
Diseño

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

entity mult is
    generic(N : natural := 8);
    port(
        x0 : in  std_logic_vector(N-1 downto 0);
        x1 : in  std_logic_vector(N-1 downto 0);
        y  : out std_logic_vector(2*N-1 downto 0)
    );
end mult;

architecture behavioral of mult is
begin
    y <= std_logic_vector(unsigned(x0) * unsigned(x1));
end behavioral;
```

Testbench

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

entity tb_mult is
end tb_mult;

architecture behavioral of tb_mult is

    constant TB_N : natural := 4;

    signal tb_x0 : std_logic_vector(TB_N-1 downto 0);
    signal tb_x1 : std_logic_vector(TB_N-1 downto 0);
    signal tb_y  : std_logic_vector(2*TB_N-1 downto 0);

begin

  tb_x0 <= "0010", "0101" after 10 ns, "1111" after 20 ns, "0000" after 30 ns;
  tb_x1 <= "1001", "1110" after 10 ns, "1111" after 20 ns, "0000" after 30 ns;

  I1: entity work.mult(behavioral)
  generic map (N => TB_N)
  port map (
    x0   => tb_x0,
    x1   => tb_x1,
    y    => tb_y
  );

end behavioral;
```

Simulación


From: 0fs To: 30,000,000fs

Ejercicio 7

Implementar en VHDL un multiplicador con signo genérico de N bits.

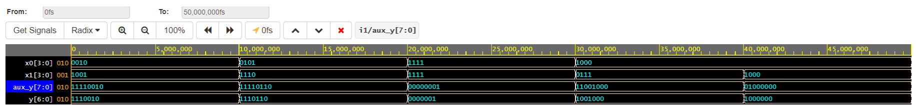
Diseño

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

entity mult is
    generic(N : natural := 8);
    port(
        x0 : in  std_logic_vector(N-1 downto 0);
        x1 : in  std_logic_vector(N-1 downto 0);
        y  : out std_logic_vector(2*N-1 downto 0)
    );
end mult;

architecture behavioral of mult is
begin
    y <= std_logic_vector(signed(x0) * signed(x1));
end behavioral;
```

Testbench

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

entity tb_mult is
end tb_mult;

architecture behavioral of tb_mult is

    constant TB_N : natural := 4;

    signal tb_x0 : std_logic_vector(TB_N-1 downto 0);
    signal tb_x1 : std_logic_vector(TB_N-1 downto 0);
    signal tb_y  : std_logic_vector(2*TB_N-2 downto 0);

begin

  --          +2      +5                  -1                  -8
  tb_x0 <= "0010", "0101" after 10 ns, "1111" after 20 ns, "1000" after 30 ns,
  -- -8
  "1000" after 40 ns, "0000" after 50 ns;

  --          -7      -6                  -1                  +7
  tb_x1 <= "1001", "1110" after 10 ns, "1111" after 20 ns, "0111" after 30 ns,
  -- -8
  "1000" after 40 ns, "0000" after 50 ns;

  -- tb_y:     -14     -30      +1     -56     -64
  --       1110010 1100010 0000001 1001000 1000000

  I1: entity work.mult(behavioral)
  generic map (N => TB_N)
  port map (
    x0   => tb_x0,
    x1   => tb_x1,
    y    => tb_y
  );

end behavioral;
```

Simulación



```
x0[3:0]   010 0010      0101      1111      1000
x1[3:0]   001 1001      1110      1111      0111      1000
aux_y[7:0] 010 11110010 11110110  00000001  11001000  01000000
y[6:0]    010 1110010   1110110   0000001   1001000   1000000
```

Ejercicio 8

Implementar en VHDL cuatro diferentes flip flops D con las siguientes entradas en cada caso:

- Reset asincrónico
- Reset sincrónico
- Set asincrónico
- Set sincrónico

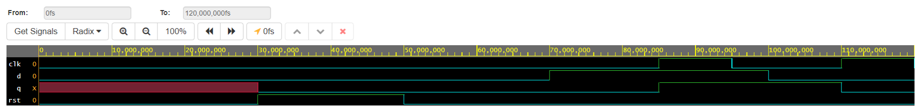
Diseño

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

entity dff is
    port(
        d   : in  std_logic;
        rst : in  std_logic;
        clk : in  std_logic;
        q   : out std_logic
    );
end dff;

architecture behavioral of dff is
begin
    process(clk,rst)
    begin
        if rst='1' then
            q <= '0';
        elsif clk = '1' and clk'event then
            q <= d;
        end if;
    end process;
end behavioral;
```

Testbench

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

entity tb_dff is
end tb_dff;

architecture behavioral of tb_dff is

    signal tb_d   : std_logic;
    signal tb_clk : std_logic;
    signal tb_rst : std_logic;
    signal tb_q   : std_logic;

begin

  tb_rst <= '0', '1' after 30 ns, '0' after 50 ns;
  tb_d   <= '0', '1' after 70 ns, '0' after 100 ns;
  tb_clk <= '0', '1' after 85 ns, '0' after 95 ns, '1' after 110 ns, '0' after 120 ns;

  I1: entity work.dff(behavioral)
  port map(
        rst =>  tb_rst,
        clk =>  tb_clk,
        d   =>  tb_d,
        q   =>  tb_q
   );

end behavioral;
```

Simulación



Notar que el tipo de dato *bit* no representa correctamente el comportamiento de un flip flop, ya que al no poseer el valor 'X' o 'U' como una posibilidad, una señal que no es inicializada es entonces inicializada por el primer valor que se declara en la definición del tipo (en un tipo enumerado, el primer elemento de la enumeración es el valor por defecto de una señal no inicializada):

```
type bit is ('0', '1');
```

Mientras que

```
package std_logic_1164 is
  type std_ulogic is ('u',  -- uninitialized
                      'x',  -- forcing  unknown
                      '0',  -- forcing  0
                      '1',  -- forcing  1
                      'z',  -- high impedance
                      'w',  -- weak unknown
                      'l',  -- weak 0
                      'h',  -- weak 1
                      '-'   -- don't care
  );
  type std_ulogic_vector is array ( natural range <> ) of std_ulogic;
  function resolved ( s : std_ulogic_vector ) return std_ulogic;
  subtype std_logic is resolved std_ulogic;
  type std_logic_vector is array ( natural range <> ) of std_logic;

  ...

end std_logic_1164;
```

(Ya veremos más adelante funciones de resolución, no preocuparse por ahora por tipos resueltos o no resueltos).

Ejercicio 9

Implementar en VHDL cuatro differents registros de *N* bits con las siguientes entradas en cada caso:

- Reset asincrónico
- Reset sincrónico
- Set asincrónico
- Set sincrónico

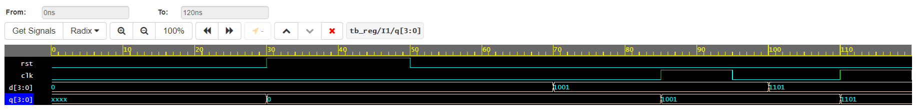
Diseño

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

entity reg is
    generic (
        N : natural := 8
    );
    port(
        d   : in  std_logic_vector(N-1 downto 0);
        rst : in  std_logic;
        clk : in  std_logic;
        q   : out std_logic_vector(N-1 downto 0)
    );
end reg;

architecture behavioral of reg is
begin
    process(clk,rst)
    begin
        if rst='1' then
             q <= (others => '0');
            -- q <= (others => '1');
        elsif clk = '1' and clk'event then
            q <= d;
        end if;
    end process;
end behavioral;
```

Testbench

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

entity tb_reg is
end tb_reg;

architecture behavioral of tb_reg is

    constant TB_N         : natural := 4;

    signal tb_d   : std_logic_vector(TB_N-1 downto 0);
    signal tb_clk : std_logic;
    signal tb_rst : std_logic;
    signal tb_q   : std_logic_vector(TB_N-1 downto 0);

begin

  tb_rst <= '0', '1' after 30 ns, '0' after 50 ns;
  tb_d   <= "0000", "1001" after 70 ns, "1101" after 100 ns;
  tb_clk <= '0', '1' after 85 ns, '0' after 95 ns, '1' after 110 ns, '0' after 120 ns;

  I1: entity work.reg(behavioral)
  generic map(
        N      => TB_N
  )
  port map(
        rst =>  tb_rst,
        clk =>  tb_clk,
        d   =>  tb_d,
        q   =>  tb_q
   );

end behavioral;
```

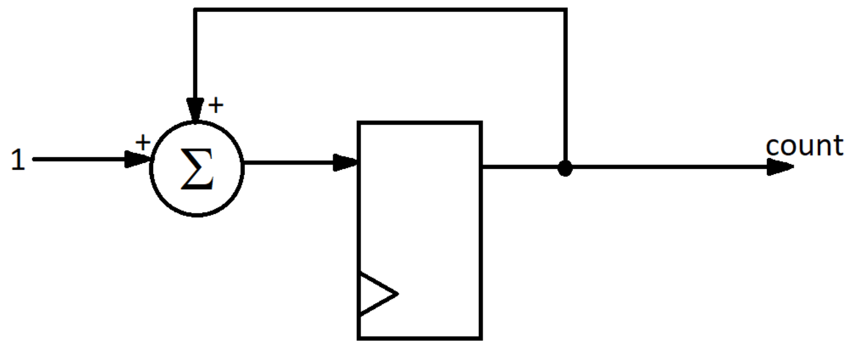
Simulación


From: 0ns
To: 120ns
Get Signals
Radix
100%
tb_reg/I1/q[3:0]
0
10
20
30
40
50
60
70
80
90
100
110
rst
clk
d[3:0]
q[3:0]
0
1001
1101
xxxx
0
1001
1101

Ejercicio 10

Implementar en VHDL un contador genérico de N bits.

Diseño



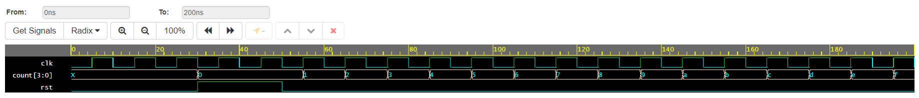
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
    generic (
        N : natural := 8
    );
    port(
        rst   : in  std_logic;
        clk   : in  std_logic;
        count : out std_logic_vector(N-1 downto 0)
    );
end counter;

architecture behavioral of counter is
    signal aux_count : unsigned(N-1 downto 0);
begin
    process(clk,rst)
    begin
        if rst='1' then
```

```
                aux_count <= (others => '0');
            elsif clk = '1' and clk'event then
                aux_count <= aux_count + 1;
            end if;
        end process;

        count <= std_logic_vector(aux_count);

    end behavioral;
```

Testbench

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

entity tb_counter is
end tb_counter;

architecture behavioral of tb_counter is

    constant SIM_TIME_NS : time := 200 ns;
    constant TB_N        : natural := 4;

    signal tb_clk   : std_logic := '0';
    signal tb_rst   : std_logic;
    signal tb_count : std_logic_vector(TB_N-1 downto 0);

begin

  tb_rst <= '0', '1' after 30 ns, '0' after 50 ns;
  tb_clk <= not tb_clk after 5 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.counter(behavioral)
  generic map(
        N     =>  TB_N
  )
  port map(
        rst   =>  tb_rst,
        clk   =>  tb_clk,
        count =>  tb_count
   );

end behavioral;
```

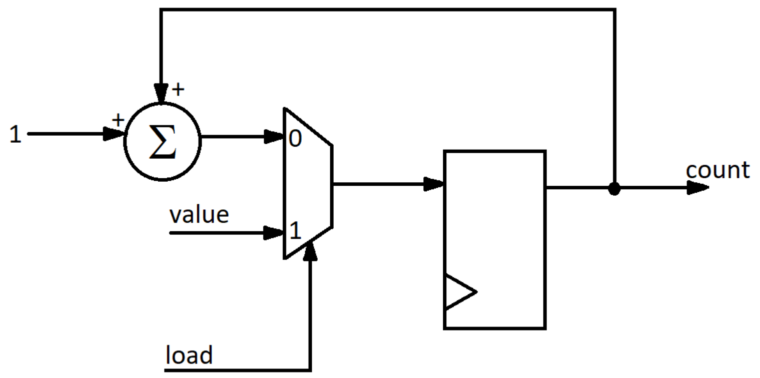
Simulación


From: 0ns
To: 200ns
Get Signals
Radix
100%
0
20
40
60
80
100
120
140
160
180
clk
count[3:0]
x
0
1
2
3
4
5
6
7
8
9
a
b
c
d
e
f
rst

Ejercicio 11

Implementar en VHDL un contador genérico de N bits y señal de carga. Cuando dicha señal de carga sea '1', el valor de la cuenta pasará a ser el valor de una señal de entrada de N bits.

Diseño



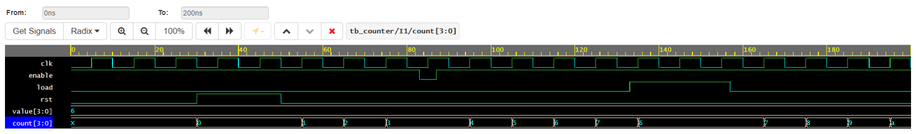
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter 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)
    );
end counter;

architecture behavioral of counter is
    signal aux_count : unsigned(N-1 downto 0);
```

```
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);

end behavioral;
```

Testbench

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

entity tb_counter is
end tb_counter;

architecture behavioral of tb_counter is

        constant SIM_TIME_NS : time := 200 ns;

    constant TB_N        : natural := 4;

    signal tb_clk   : std_logic := '0';
    signal tb_rst   : std_logic;
    signal tb_load  : std_logic;
    signal tb_value : std_logic_vector(TB_N-1 downto 0);
    signal tb_count : std_logic_vector(TB_N-1 downto 0);

begin

  tb_rst   <= '0', '1' after 30 ns, '0' after 50 ns;
  tb_clk   <= not tb_clk after 5 ns;
  tb_value <= "0110";
  tb_load  <= '0', '1' after 133 ns, '0' after 157 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.counter(behavioral)
  generic map(
        N     =>  TB_N
  )
  port map(
        rst   =>  tb_rst,
        clk   =>  tb_clk,
        load  =>  tb_load,
        value =>  tb_value,
        count =>  tb_count
    );

end behavioral;
```

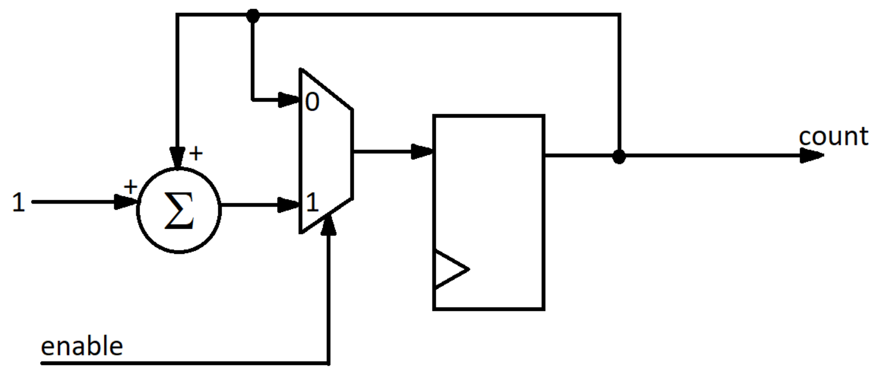
Simulación


From: 0ns
To: 200ns
Get Signals
Radix
100%
tb_counter/I1/count[3:0]
0
20
40
60
80
100
120
140
160
180
clk
enable
load
rst
value[3:0]
count[3:0]
6
x
0
1
2
3
4
5
6
7
6
7
8
9
a

Ejercicio 12

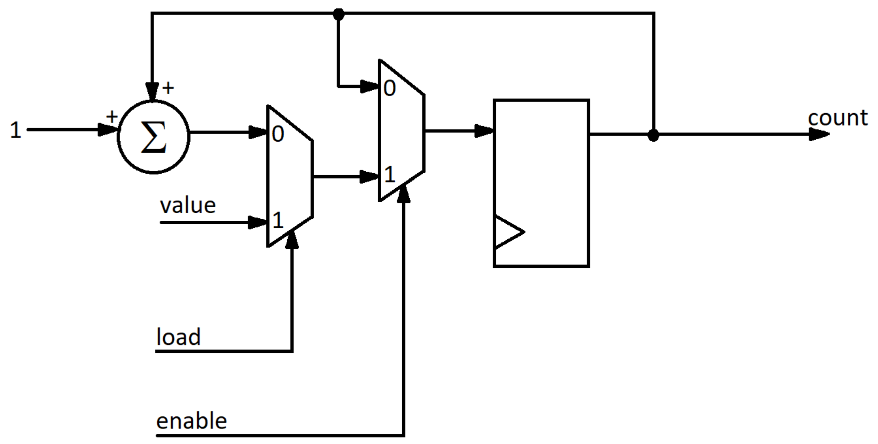
Implementar en VHDL un contador genérico de *N* bits y señal de *enable*. Cuando dicha señal de *enable* sea '0', el valor de la cuenta no deberá ser modificado.

Diseño



Ejercicio 13

Implementar en VHDL un contador genérico de N bits con señal de *enable* y señal de carga. Cuando la señal de *enable* sea '0', el valor de la cuenta no deberá ser modificado. Cuando la señal de carga sea '1', el valor de la cuenta pasará a ser el valor de una señal de entrada de N bits.



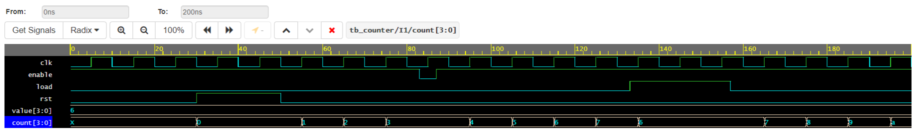
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
    generic (
        N : natural := 8
    );
    port(
        rst   : in  std_logic;
        clk   : in  std_logic;
        enable: 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)
    );
end counter;
```

```
architecture behavioral of counter is
    signal aux_count : unsigned(N-1 downto 0);
begin
    process(clk,rst)
    begin
        if rst='1' then
             aux_count <= (others => '0');
        elsif clk = '1' and clk'event then
            if enable = '1' then
                if load = '1' then
                    aux_count <= unsigned(value);
                else
                    aux_count <= aux_count + 1;
                end if;
            end if;
        end if;
    end process;

    count <= std_logic_vector(aux_count);

end behavioral;
```

Testbench

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

entity tb_counter is
end tb_counter;

architecture behavioral of tb_counter is

    constant SIM_TIME_NS : time := 200 ns;
    constant TB_N        : natural := 4;

    signal tb_clk    : std_logic := '0';
    signal tb_rst    : std_logic;
    signal tb_enable: std_logic;
    signal tb_load   : std_logic;
    signal tb_value  : std_logic_vector(TB_N-1 downto 0);
    signal tb_count  : std_logic_vector(TB_N-1 downto 0);

begin

  tb_rst     <= '0', '1' after 30 ns, '0' after 50 ns;
  tb_clk     <= not tb_clk after 5 ns;
  tb_value   <= "0110";
  tb_enable  <= '1', '0' after 83 ns, '1' after 87 ns;
  tb_load    <= '0', '1' after 133 ns, '0' after 157 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.counter(behavioral)
  generic map(
        N     =>  TB_N
  )
  port map(
        rst   =>  tb_rst,
        clk   =>  tb_clk,
        enable=>  tb_enable,
        load  =>  tb_load,
        value =>  tb_value,
        count =>  tb_count
   );

end behavioral;
```

Simulación


From:
0ns
To:
200ns
Get Signals
Radix
100%
tb_counter/I1/count[3:0]
0
20
40
60
80
100
120
140
160
180
clk
enable
load
rst
value[3:0]
count[3:0]
6
x
0
1
2
3
4
5
6
7
6
7
8
9
a

Ejercicio 14

Implementar en VHDL un registro de desplazamiento de N bits de largo.

Diseño

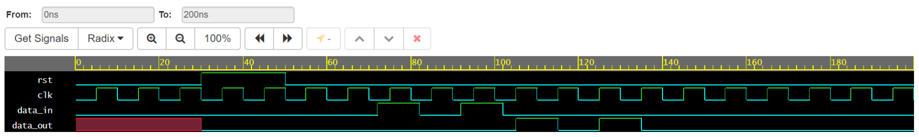
```
library ieee;
use ieee.std_logic_1164.all;

entity shift_reg is
    generic (
        N        : natural := 8
    );
    port(
        rst      : in  std_logic;
        clk      : in  std_logic;
        data_in  : in  std_logic;
        data_out : out std_logic
    );
end shift_reg;

architecture behavioral of shift_reg is
    signal sr : std_logic_vector(N-1 downto 0);
begin
    process(clk,rst)
    begin
        if rst='1' then
            sr <= (others => '0');
        elsif clk = '1' and clk'event then
            sr(N-1 downto 1) <= sr(N-2 downto 0);
            sr(0) <= data_in;
        end if;
    end process;

    data_out <= sr(N-1);

end behavioral;
```

Testbench

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

entity tb_shift_reg is
end tb_shift_reg;

architecture behavioral of tb_shift_reg is

    constant SIM_TIME_NS : time := 200 ns;
    constant TB_N        : natural := 4;

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

begin

  tb_rst      <= '0', '1' after 30 ns, '0' after 50 ns;
  tb_clk      <= not tb_clk after 5 ns;
  tb_data_in  <= '0', '1' after 72 ns, '0' after 82 ns, '1' after 92 ns,
                 '0' after 102 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.shift_reg(behavioral)
  generic map(
        N =>  TB_N
  )
  port map(
        rst      =>  tb_rst,
        clk      =>  tb_clk,
        data_in  =>  tb_data_in,
        data_out =>  tb_data_out
   );
```

```
end behavioral;
```

Simulación


From: 0ns
To: 200ns
Get Signals
Radix
100%
0
20
40
60
80
100
120
140
160
180
rst
clk
data_in
data_out

Ejercicio 17

Implementar en VHDL un registro de desplazamiento genérico de N etapas y de un ancho de palabra de M bits. El mismo deberá tener señal de carga y señal de *enable*.

Diseño

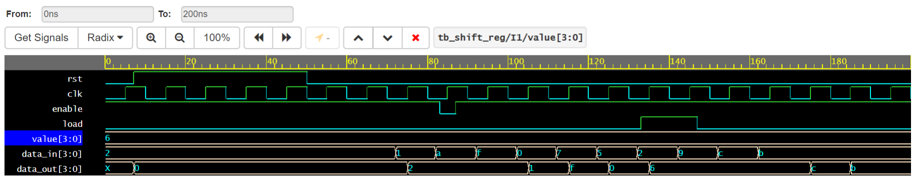
```
use ieee.std_logic_1164.all;

package my_pkg is

        type std_logic_matrix is array(natural range <>) of std_logic_vector;

end package my_pkg;

--------------------------------------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use work.my_pkg.all;

entity shift_reg is
    generic (
        N_BITS   : natural := 8;
        M_STAGES : natural := 10
    );
    port(
        rst      : in  std_logic;
        clk      : in  std_logic;
        enable   : in  std_logic;
        load     : in  std_logic;
        value    : in  std_logic_vector(N_BITS-1 downto 0);
        data_in  : in  std_logic_vector(N_BITS-1 downto 0);
        data_out : out std_logic_vector(N_BITS-1 downto 0)
    );
end shift_reg;

architecture behavioral of shift_reg is
    signal sr :  std_logic_matrix(M_STAGES-1 downto 0)(N_BITS-1 downto 0);
begin
    process(clk,rst)
    begin
```

```
        if rst='1' then
             sr <= (others=>(others=>'0'));
        elsif clk = '1' and clk'event then
                if enable = '1' then
                    if load = '1' then
                        sr <= (others => value);
                else
                    sr(M_STAGES-1 downto 1) <= sr(M_STAGES-2 downto 0);
                    sr(0) <= data_in;
                end if;
            end if;
        end if;
    end process;

    data_out <= sr(M_STAGES-1);

end behavioral;
```

Testbench

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

entity tb_shift_reg is
end tb_shift_reg;
use work.my_pkg.all;

architecture behavioral of tb_shift_reg is

        constant SIM_TIME_NS : time := 200 ns;

    constant TB_N_BITS   : natural := 4;
    constant TB_M_STAGES : natural := 3;

    signal tb_clk      : std_logic := '0';
    signal tb_rst      : std_logic;
    signal tb_value    : std_logic_vector(TB_N_BITS-1 downto 0);
    signal tb_load     : std_logic;
    signal tb_enable   : std_logic;
    signal tb_data_in  : std_logic_vector(TB_N_BITS-1 downto 0);
    signal tb_data_out : std_logic_vector(TB_N_BITS-1 downto 0);

begin

  tb_rst      <= '0', '1' after 7 ns, '0' after 50 ns;
  tb_clk      <= not tb_clk after 5 ns;
  tb_value    <= "0110";
  tb_enable   <= '1', '0' after 83 ns, '1' after 87 ns;
  tb_load     <= '0', '1' after 133 ns, '0' after 147 ns;
  tb_data_in  <= "0010", "0001" after 72 ns, "1010" after 82 ns, "1111" after 92 ns, "000

  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.shift_reg(behavioral)
  generic map(
        N_BITS   =>  TB_N_BITS,
        M_STAGES =>  TB_M_STAGES
  )
  port map(
        rst      =>  tb_rst,
        clk      =>  tb_clk,
        enable   =>  tb_enable,
        load     =>  tb_load,
        value    =>  tb_value,
        data_in  =>  tb_data_in,
        data_out =>  tb_data_out
   );

end behavioral;
```

Simulación


From: 0ns
To: 200ns
Get Signals
Radix
100%
tb_shift_reg/I1/value[3:0]
0
20
40
60
80
100
120
140
160
180
rst
clk
enable
load
value[3:0]
data_in[3:0]
data_out[3:0]
6
2
1
a
f
0
7
5
2
9
c
b
x
0
2
1
f
0
6
c
b

Ejercicio 18

Implementar en VHDL un circuito que detecte la secuencia “0010110” de una señal de entrada. Deberán relizarse dos implementaciones diferentes: mediante una FSM y mediante un registro de desplazamiento y la lógica de detección adecuada.

Diseño

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

entity fsm is
    port(
        rst      : in  std_logic;
        clk      : in  std_logic;
        data_in  : in  std_logic;
        detected : out std_logic
    );
end fsm;

architecture behavioral of fsm is

    type t_state is (S0, S1, S2, S3, S4, S5, S6);
    signal state : t_state;

begin
    process(clk,rst)
    begin
        if rst='1' then
            state <= S0;
        elsif clk = '1' and clk'event then
            case state is
                when S0 =>
                    if data_in = '0' then
                        state <= S1;
                    end if;
                 when S1 =>
                     if data_in = '1' then
                        state <= S2;
                     end if;
                 when S2 =>
                     if data_in = '0' then
```

```
                        state <= S3;
                    else
                        state <= S0;
                    end if;
                when S3 =>
                    if data_in = '1' then
                        state <= S4;
                    else
                        state <= S0;
                    end if;
                when S4 =>
                    if data_in = '1' then
                        state <= S5;
                    else
                        state <= S0;
                    end if;
                when S5 =>
                    if data_in = '0' then
                        state <= S6;
                    else
                        state <= S0;
                    end if;
                when S6 =>
                    if data_in = '0' then
                        state <= S1;
                    else
                        state <= S0;
                    end if;
            end case;
        end if;
    end process;

    detected <= '1' when state = S6 else '0';

end behavioral;

------------------------------------------------------------

architecture one_hot of fsm is

    constant S0: std_logic_vector(6 downto 0) := "0000001";
    constant S1: std_logic_vector(6 downto 0) := "0000010";
    constant S2: std_logic_vector(6 downto 0) := "0000100";
    constant S3: std_logic_vector(6 downto 0) := "0001000";
    constant S4: std_logic_vector(6 downto 0) := "0010000";
    constant S5: std_logic_vector(6 downto 0) := "0100000";
    constant S6: std_logic_vector(6 downto 0) := "1000000";

        signal state : std_logic_vector(6 downto 0);

begin
    process(clk,rst)
    begin
        if rst='1' then
             state <= S0;
        elsif clk = '1' and clk'event then
            case state is
                when S0 =>
                    if data_in = '0' then
                        state <= S1;
                    end if;
                when S1 =>
                    if data_in = '1' then
                        state <= S2;
                    end if;
                when S2 =>
                    if data_in = '0' then
                        state <= S3;
                    else
                        state <= S0;
                    end if;
                when S3 =>
                    if data_in = '1' then
                        state <= S4;
                    else
                        state <= S0;
                    end if;
                when S4 =>
                    if data_in = '1' then
                        state <= S5;
                    else
                        state <= S0;
                    end if;
                when S5 =>
                    if data_in = '0' then
                        state <= S6;
```

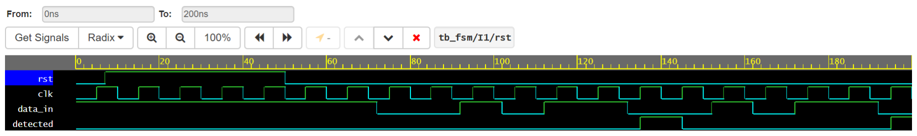
```vhdl
                     else
                        state <= S0;
                     end if;
                when S6 =>
                    if data_in = '0' then
                        state <= S1;
                    else
                        state <= S0;
                    end if;
                when others =>
                    state <= S0;
             end case;
        end if;
    end process;

    detected <= '1' when state = S6 else '0';

end one_hot;

-------------------------------------------------------------

architecture shiftreg of fsm is

    signal sr: std_logic_vector(6 downto 0);

begin
    process(clk,rst)
    begin
        if rst='1' then
            sr <= (others => '0');
        elsif clk = '1' and clk'event then
            sr(6 downto 1) <= sr(5 downto 0);
            sr(0) <= data_in;
        end if;
    end process;

    detected <= '1' when sr = "0010110" else '0';

end shiftreg;
```

Testbench

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

entity tb_fsm is
end tb_fsm;

architecture behavioral of tb_fsm is

    constant SIM_TIME_NS : time := 200 ns;

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

begin

  tb_rst      <= '0', '1' after 7 ns, '0' after 50 ns;
  tb_clk      <= not tb_clk after 5 ns;
  tb_data_in  <= '1', '0' after 72 ns, '0' after 82 ns, '1' after 92 ns,
                 '0' after 102 ns, '1' after 112 ns,  '1' after 122 ns,
                 '0' after 132 ns, '0' after 142 ns, '1' after 152 ns,
                 '0' after 162 ns, '1' after 172 ns, '1' after 182 ns,
                 '0' after 192 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.fsm(behavioral)
  --I1: entity work.fsm(one_hot)
  --I1: entity work.fsm(shiftreg)
  port map(
        rst      =>  tb_rst,
        clk      =>  tb_clk,
        data_in  =>  tb_data_in,
        detected =>  tb_detected
```

```
   );

end behavioral;
```

Simulación


From: 0ns
To: 200ns
Get Signals
Radix
100%
tb_fsm/I1/rst
0
20
40
60
80
100
120
140
160
180
rst
clk
data_in
detected