Cómo depurar una máquina de estados en VHDL

Creador: David Rubio G.

Entrada: https://soceame.wordpress.com/2024/11/18/como-depurar-una-maquina-de-estados-en-vhdl/

Blog: https://soceame.wordpress.com/

GitHub: https://github.com/DRubioG

Fecha última modificación: 23/02/2025

Uno de los problemas más típicos en VHDL es intentar depurar una máquina de estados, y esto es debido a la forma en la que se enseñan a crear máquinas de estados.

El formato normal en el que se enseña a crear una máquina de estados es utilizando un *type*.

```
type fsm is (S0, S1, S2);

signal state, next_state : fsm;

begin


    process(clk, rst_n, state)
    begin
        if rst_n = '0' then
            next_state <= S0;
        elsif rising_edge(clk) then
            case state is
                when S0 =>
                    next_state <= S1;
                when S1 =>
                    next_state <= S2;
                when S2 =>
                    next_state <= S0;
                when others=> null;
            end case;
        end if;
    end process;
```

El problema que tiene este formato está en que en caso de querer depurar o querer sacar del estado(*state*) al exterior, no es fácil, porque se desconoce el tamaño que tiene la señal de cambio de estados.

Solución

Este problema viene resuelto por las Safe Finite State Machines (o máquinas de estados seguras).

Lo primero es entender que una máquina de estado se puede realizar utilizando un *case*, pero también se puede utilizar un *if*.

```
constant S0 : std_logic_vector(2 downto 0):="001";
constant S1 : std_logic_vector(2 downto 0):="010";
constant S2 : std_logic_vector(2 downto 0):="100";

signal state, next_state : std_logic_vector(2 downto 0);

begin


    process(clk, rst_n, state)
    begin
        if rst_n = '0' then
            next_state <= S0;
        elsif rising_edge(clk) then
            if state = S0 then
                next_state <= S1;
            elsif state = S1 then
                next_state <= S2;
            elsif state = S2 then
                next_state <= S0;
            end if;
        end if;
    end process;
```

Entonces, los estados en un *if* se pueden llegar a considerar constantes(*constant*).

Pues este es el truco para generar una máquina de estados que se puede depurar. Aquí un ejemplo de cómo quedaría.

```
constant S0 : std_logic_vector(2 downto 0):="001";
constant S1 : std_logic_vector(2 downto 0):="010";
constant S2 : std_logic_vector(2 downto 0):="100";

signal state, next_state : std_logic_vector(2 downto 0);

begin

    process(clk, rst_n, state)
    begin
        if rst_n = '0' then
            next_state <= S0;
        elsif rising_edge(clk) then
            case state is
                when S0 =>
                    next_state <= S1;
                when S1 =>
                    next_state <= S2;
                when S2 =>
                    next_state <= S0;
                when others=> null;
            end case;
        end if;
    end process;
```

Perfeccionamiento de las máquinas de estados

Si lo que quieres es que la máquina de estados también se vea como las máquinas de estados que te han enseñado, solo necesitas añadir un *subtype* que se llame como el *type* que utilizabas en las máquinas de estados y queda todo arreglado.

```
subtype fsm is std_logic_vector(2 downto 0);

constant S0 : fsm:="001";
constant S1 : fsm:="010";
constant S2 : fsm:="100";

signal state, next_state : fsm;

begin


    process(clk, rst_n, state)
    begin
        if rst_n = '0' then
            next_state <= S0;
        elsif rising_edge(clk) then
            case state is
                when S0 =>
                    next_state <= S1;
                when S1 =>
                    next_state <= S2;
                when S2 =>
                    next_state <= S0;
                when others=> null;
            end case;
        end if;
    end process;
```