VHDL

VHDL

when / else : concurrent signal assignment statement

A sequential signal assignment statement is also a concurrent signal assignment statement. Additional control is provided by the use of postponed and guarded.

```
[ label : ] sequential signal assignment statement

[ label : ] [ postponed ] conditional_signal_assignment_statement ;

[ label : ] [ postponed ] selected_signal_assignment_statement ;
```

The optional guarded causes the statement to be executed when the guarded signal changes from False to True. conditional signal assignment statement

A conditional assignment statement is also a concurrent signal assignment statement.

```
target <= waveform when choice; -- choice is a boolean expression
target <= waveform when choice else waveform;

sig <= a_sig when count>7;
sig2 <= not a_sig after 1 ns when ctl='1' else b_sig;

-- "waveform" for this statement seems to include [ delay_mechanism ]
-- See sequential signal assignment statement
```

FSM

```
-- FSM template: from Circuit Design with VHDL

LIBRARY ieee;
USE ieee.std_logic_1164.all;
```

```
-------------------------------------------------
ENTITY <entity_name> IS
   PORT ( input: IN <data_type>;
          reset, clock: IN STD_LOGIC;
          output: OUT <data_type>);
END <entity_name>;
-------------------------------------------------
ARCHITECTURE <arch_name> OF <entity_name> IS
TYPE state IS (state0, state1, state2, state3, ...);
   SIGNAL pr_state, nx_state: state;
BEGIN
   ---------- Lower section: -----------------------
   PROCESS (reset, clock)
   BEGIN
      IF (reset='1') THEN
         pr_state <= state0;
      ELSIF (clock'EVENT AND clock='1') THEN
         pr_state <= nx_state;
      END IF;
   END PROCESS;
   ---------- Upper section: -----------------------
   PROCESS (input, pr_state)
   BEGIN
      CASE pr_state IS
         WHEN state0 =>
            IF (input = ...) THEN
               output <= <value>;
               nx_state <= state1;
ELSE ...
            END IF;
         WHEN state1 =>
            IF (input = ...) THEN
               output <= <value>;
               nx_state <= state2;
ELSE ...
            END IF;
         WHEN state2 =>
            IF (input = ...) THEN
               output <= <value>;
               nx_state <= state3;
ELSE ...
END IF; ...
      END CASE;
   END PROCESS;
END <arch_name>;
```

```
--proceso de reloj, útil para un testbech

-- inicializar la variable en la zona de señales
SIGNAL CLK  :  STD_LOGIC := '0';


clock : process(clk)
begin
   clk <= not clk after 20ns;
end process;


--todos los procesos sincronos con resets:

process (clk, reset)
   if reset = '1' then
      -- do RESET, other asyncronous taska ;
   elseif rising_edge(clk) then
      -- SINCRONOUS TASKS
      if enable = '1' then
         -- ENABLED TASKS

   endif;
end process;
```