VHDL

Diego Trapero

Table of contents

1 VHDL syntax

1.1 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
```

1.2 Snippets

Proceso de reloj útil para un testbech

```
--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

```
--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;
```

2 VHDL component

2.1 Gates

NAND Gate

```
entity NANDGate is
   port (
      a, b : in bit;
      x : out bit
   );
end NANDGate;

architecture NANDArch of NANDGate is
begin
   x <= a NAND b;
end NANDArch;
```

2.2 Biestables

Latch Internal signal implementation

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

entity S_R_latch_top is
    Port ( S : in    STD_LOGIC;
           R : in    STD_LOGIC;
           Q : out   STD_LOGIC);
end S_R_latch_top;

architecture Behavioral of S_R_latch_top is
signal Q2   : STD_LOGIC;
signal notQ : STD_LOGIC;
begin

Q    <= Q2;
Q2   <= R nor notQ;
notQ <= S nor Q2;

end Behavioral;
```

Latch Inout ports implementation

```
entity SR_Latch is
  Port ( S,R : in STD_LOGIC;
         Q : inout STD_LOGIC;
         Q_n : inout STD_LOGIC);
 end SR_Latch;

 architecture SR_Latch_arch of SR_Latch is
 begin
 process (S,R,Q,Q_n)
       begin
             Q <= R NOR Q_n;
             Q_n <= S NOR Q;
       end process;

 end SR_Latch_arch;
```

```
-- http://vhdlbynaresh.blogspot.com.es/2013/07/design-of-sr-latch-using-behavior.html

library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity SR_Latch is
     port(
         enable : in STD_LOGIC;
         s : in STD_LOGIC;
         r : in STD_LOGIC;
         reset : in STD_LOGIC;
         q : out STD_LOGIC;
         qb : out STD_LOGIC
         );
end SR_Latch;

architecture SR_Latch_arc of SR_Latch is
begin

    latch : process (s,r,enable,reset) is
    begin
        if (reset='1') then
            q <= '0';
            qb <= '1';
        elsif (enable='1') then
            if (s/=r) then
                q <= s;
                qb <= r;
            elsif (s='1' and r='1') then
                q <= 'Z';
                qb <= 'Z';
            end if;
        end if;
    end process latch;

end SR_Latch_arc;
```

D Flip-FLop

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

entity DFF is
   port(
      D : in std_logic; --Data input
      C : in std_logic; --Clock signal
      Q : out std_logic --Data output
   );
end DFF;

architecture Behavioral of DFF is

begin
   process(C)
   begin
      if rising_edge(C) then
         Q <= D;
      end if;
   end process;
end Behavioral;
```

D Flip Flop with Enable and Reset

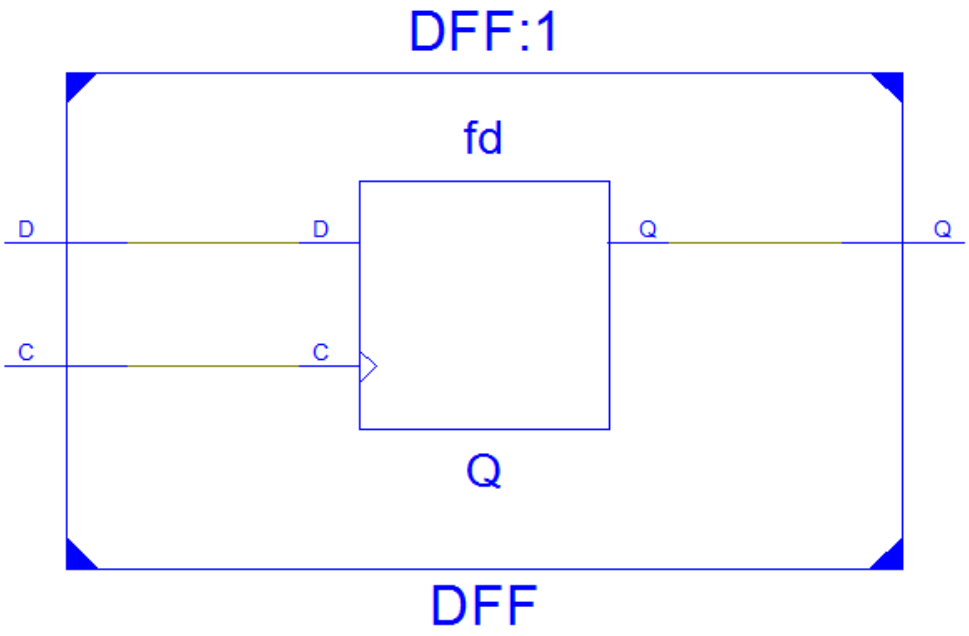

Figure 1: Synthesis Result

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

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity DFF is
   port(
      D  : in  std_logic; --Data input
      C  : in  std_logic; --Clock signal
      EN : in  std_logic; --Enable signal
      R  : in  std_logic; -- Asyncronous reset
      Q  : out std_logic  --Data output
   );
end DFF;

architecture Behavioral of DFF is
begin

   process(C, R)
   begin
      if (R = '1') then
         Q <= '0';
      else if rising_edge(C) AND EN='1' then
         Q <= D;
      end if;
      end if;
   end process;

end Behavioral;
```

D Flip Flop with Enable, Reset and Preset

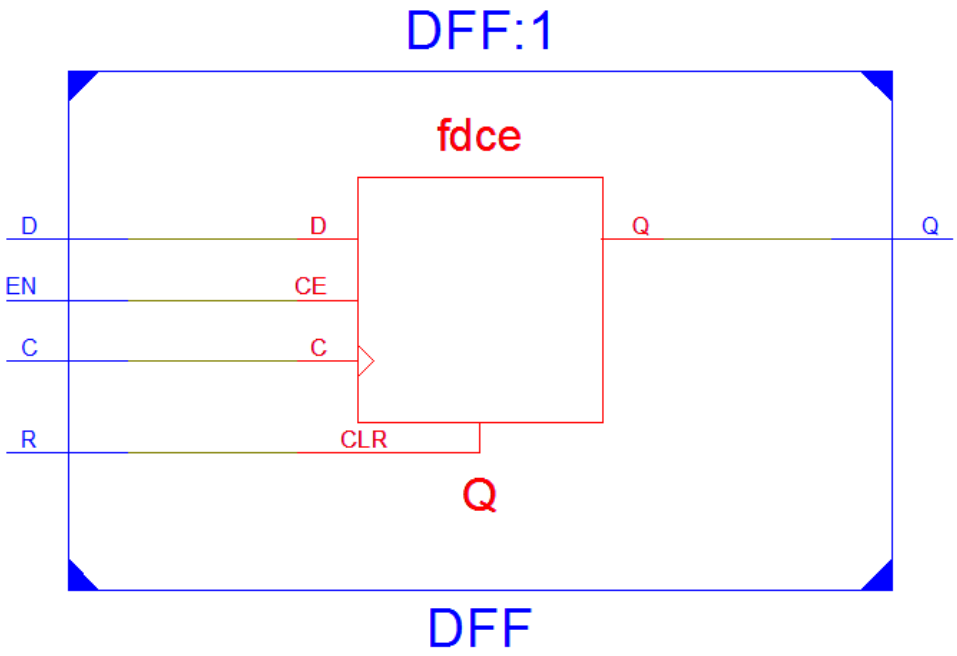

Figure 2: Synthesis Result

```
-- Code from Wikibooks:
-- https://en.wikibooks.org/wiki/VHDL_for_FPGA_Design/D_Flip_Flop

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity DFF is
   port
   (
      clk : in std_logic;

      rst : in std_logic;
      pre : in std_logic;
      ce  : in std_logic;

      d : in std_logic;

      q : out std_logic
   );
end entity DFF;

architecture Behavioral of DFF is
begin
   process (clk) is
   begin
      if rising_edge(clk) then
         if (rst='1') then
            q <= '0';
         elsif (pre='1') then
            q <= '1';
         elsif (ce='1') then
            q <= d;
         end if;
      end if;
   end process;
end architecture Behavioral;
```

With Guarded Block

```
-- D Flip Flop with Guarded Block, from Circuit Design with VHDL

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY dff IS
   PORT (
      d, clk, rst: IN STD_LOGIC;
      q : out std_logic
END dff;

ARCHITECTURE dff OF dff IS
BEGIN
    b1: BLOCK (clk'EVENT AND clk='1')
   BEGIN
      q <= GUARDED '0' WHEN rst='1' ELSE d;
    END BLOCK b1;
END dff;
```

2.3 FSM

FSM template

```
-- 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>;
```

3 Examples

4 More