DIPARTIMENTO DI ELETTRONICA E INFORMAZIONE

POLITECNICO
DI MILANO

VHDL - Esempi





Martedì 13 Gennaio 2009

Processi

- Un **process** è un'istruzione concorrente che contiene un'area sequenziale.
- Un processo viene eseguito parallelamente alle altre istruzioni concorrenti.
  - L'esecuzione del suo body può essere condizionata da una **sensitivity list**, una lista di segnali.
  - La sensitivity list non si usa quando il body contiene un'istruzione wait.
    - In questo caso l'esecuzione viene avviata e si sospende nel modo indicato da *wait*.
- Note:
  - I segnali assegnati all'interno di un processo ricevono il valore solo dopo la sospensione del processo.
  - In un processo le variabili locali conservano in proprio valore nel tempo tra un'esecuzione e l'altra.

Processi - Esempio


```
p1: process (B, C)
begin
    A <= B and C;
    C <= '0';
end;
```

```
p1_2: process
begin
    A <= B and C;
    C <= '0';
    wait on B, C;
end;
```

Struttura generale di un programma


```
use libreria

entity circuito is
...
end circutio;

architecture archi of circuito is
-- istruzione concorrente 1
-- istruzione concorrente 2
begin
        pa: process
        begin
                -- istruzione sequenziale pa_1
                -- istruzione sequenziale pa_2
                -- ...
        end;

        pb: process
        begin
                -- istruzione sequenziale pa_2
                -- ...
        end;
end;
```

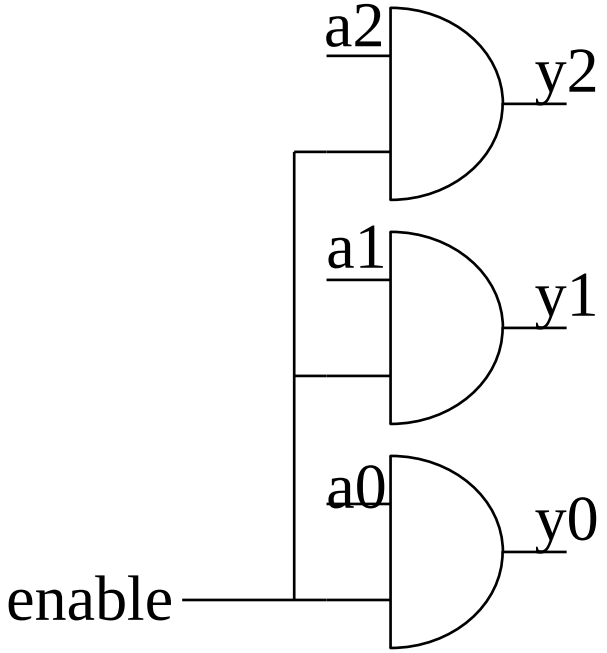
Area concorrente

Area sequenziale

Area sequenziale

Controllo dei processi: IF-THEN-ELSE


```
SIGNAL a : STD_LOGIC_VECTOR(2 downto 0);
SIGNAL y : STD_LOGIC_VECTOR(2 downto 0);
SIGNAL enable : STD_LOGIC;

PROCESS( enable)
BEGIN
    IF (enable = ‘1’) THEN
        y <= a;
    ELSE
        y <= ”000”;
    END IF;
END PROCESS;
```

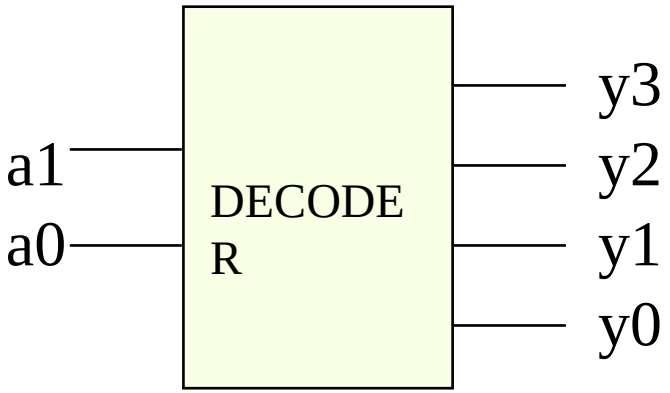


Controllo dei processi: CASE


```
SIGNAL a : STD_LOGIC;
SIGNAL y : STD_LOGIC;
SIGNAL enable : STD_LOGIC_VECTOR(1 downto 0);

PROCESS ( enable )
BEGIN
    CASE enable IS
        WHEN “00” =>
            y <= a;
        WHEN “01” =>
            y <= ‘0’;
        WHEN “10” =>
            y <= ‘0’;
        WHEN OTHERS =>
            y <= ‘1’;
    END CASE;
END PROCESS;
```

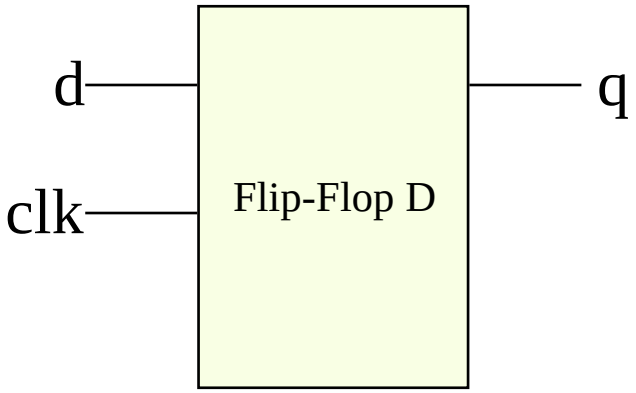
Controllo dei processi: CASE - DECODER


```
SIGNAL y : STD_LOGIC_VECTOR(3 downto 0);
SIGNAL a : STD_LOGIC_VECTOR(1 downto 0);

PROCESS (a)
BEGIN
    CASE a IS
        WHEN “00” =>
            y <= “0001”;
        WHEN “01” =>
            y <= “0010”;
        WHEN “10” =>
            y <= “0100”;
        WHEN “11” =>
            y <= “1000”;
        WHEN OTHERS => ;
    END CASE;
END PROCESS;
```



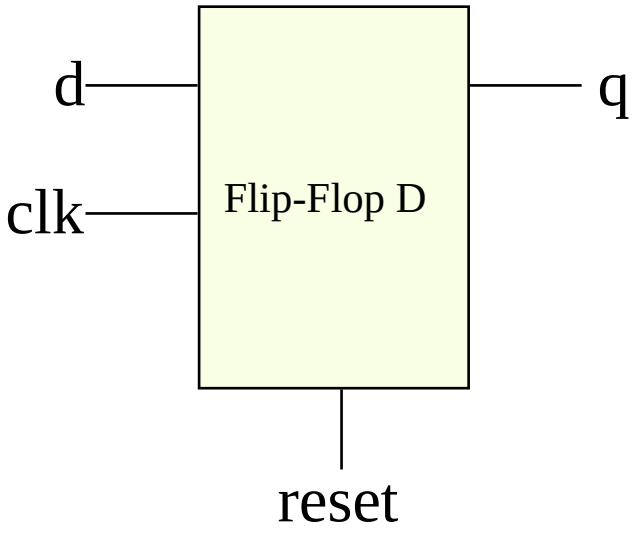
Circuiti Sequenziali: FF-D


```
entity ffd is
port(
    d:          in std_logic;
    clk:        in std_logic;
    q:          out std_logic
);
end ffd;

ARCHITECTURE arch OF ffd IS
BEGIN
    PROCESS(clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            q <= d;
        END IF;
    END PROCESS;
END arch;
```



Circuiti Sequenziali: FF-D con RESET


```
entity ffd is
port(
    d:          in std_logic;
    clk:        in std_logic;
    reset:      in std_logic;
    q:          out std_logic
);
end ffd;

ARCHITECTURE arch OF ffd IS
BEGIN
    PROCESS(clk, reset)
    BEGIN
        IF (reset = '1') THEN
            q <= '0';
        ELSIF (clk'EVENT AND clk = '1') THEN
            q <= d;
        END IF;
    END PROCESS;
END arch;
```

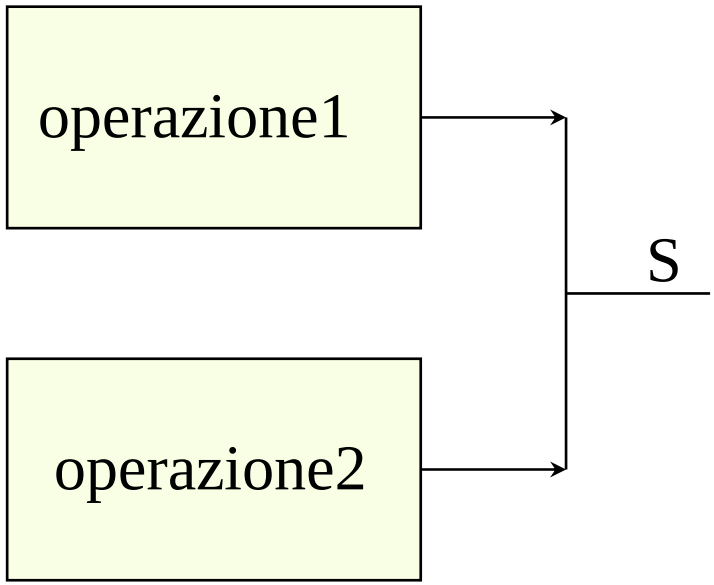


Contatore

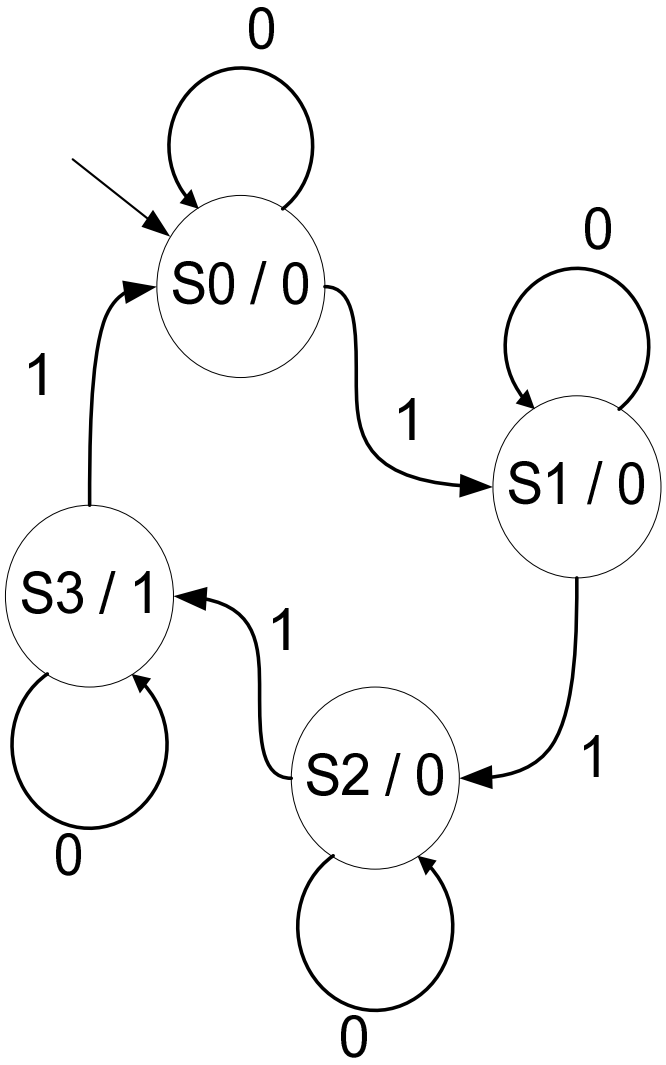

```
ARCHITECTURE arch OF counter IS
BEGIN
    PROCESS(clk, reset)
    BEGIN
        IF (reset = '1') THEN
            q <= '000';
        ELSIF (clk'EVENT AND clk = '1') THEN
            IF q >= 9 THEN
                q <= '000';
            ELSE
                q <= q + 1;
            END IF;
        END IF;
    END PROCESS;
END arch;
```

Double DRIVEN


```
ARCHITECTURE arch  OF circ IS
SIGNAL s : BIT;
BEGIN
    PROCESS
    BEGIN
        s <= operazione1;
    END PROCESS;

    PROCESS
    BEGIN
        s <= operazione2;
    END PROCESS;
END arch;
```



Macchina a stati finiti: FSM



FSM 1/4 - Entity

```
-------------------------------------------------

library ieee ;
use ieee.std_logic_1164.all;

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

entity seq_design is
port(
     a:                 in std_logic;
     clock:                        in std_logic;
     reset:                        in std_logic;
     x:                 out std_logic
);
end seq_design;

-------------------------------------------------
```

FSM 2/4 – Architecture: process#1


```
architecture FSM of seq_design is

    -- define the states of FSM model

    type state_type is (S0, S1, S2, S3);
    signal next_state, current_state: state_type;

begin

    -- cocurrent process#1: state registers
    state_reg: process(clock, reset)
    begin

      if (reset='1') then
            current_state <= S0;
      elsif (clock'event and clock='1') then
          current_state <= next_state;
      end if;

    end process;
```

FSM 3/4 – Architecture: process#2


```
-- cocurrent process#2: combinational logic
   comb_logic: process(current_state, a)
   begin

      -- use case statement to show the
      -- state transistion

      case current_state is

         when S0 =>      x <= '0';
                         if a='0' then
                            next_state <= S0;
                         elsif a ='1' then
                            next_state <= S1;
                         end if;

         when S1 =>      x <= '0';
                         if a='0' then
                            next_state <= S1;
                         elsif a='1' then
                            next_state <= S2;
                         end if;
```

FSM 4/4 – Architecture: process#2

```
        when S2 =>          x <= '0';
                            if a='0' then
                               next_state <= S2;
                            elsif a='1' then
                               next_state <= S3;
                            end if;

        when S3 =>          x <= '1';
                            if a='0' then
                               next_state <= S3;
                            elsif a='1' then
                               next_state <= S0;
                            end if;

        when others =>
                            x <= '0';
                            next_state <= S0;

    end case;

  end process;

end FSM;
```

WebPACK

- ISE *WebPACK* è un tool gratuito e scaricabile dal web che supporta la scrittura, la sintesi e la verifica di specifiche in HDL sia per CPLD che FPGA.
- E' possibile scaricarlo gratuitamente da:
  - www.xilinx.com