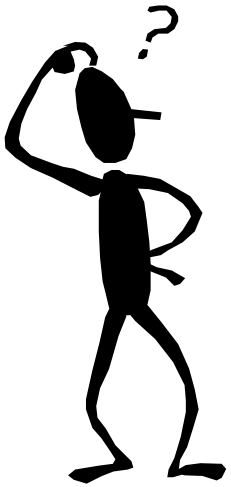


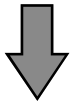
VHDL

Descrizione Introduttiva

VHDL

- Linguaggio utilizzato per descrivere circuiti digitali

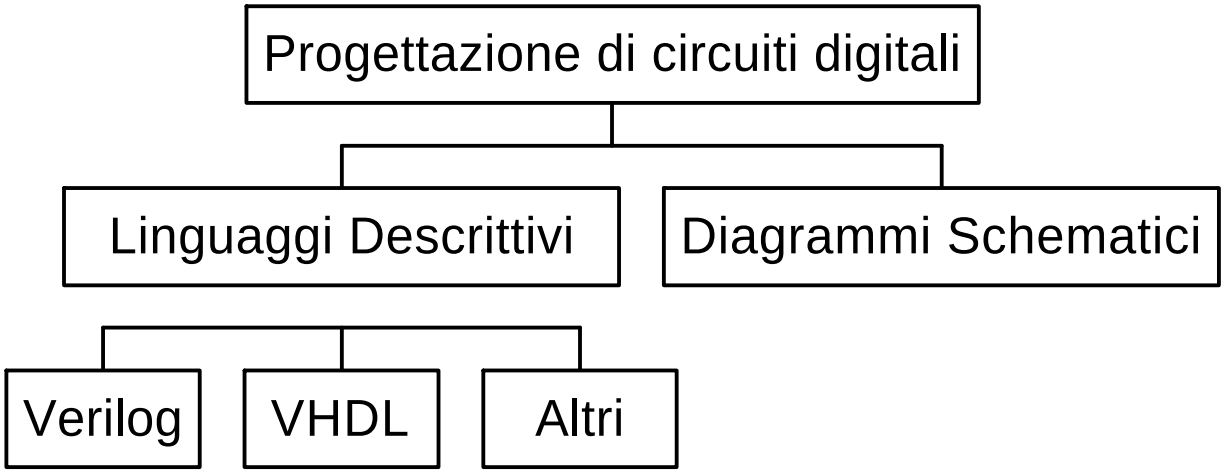
- **V**HSIC **H**ardware **D**escription **L**anguage

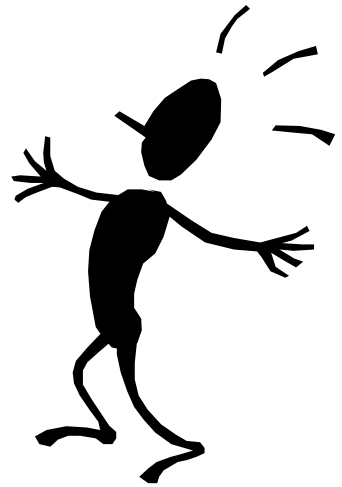


Very High Speed Integrated Circuits

- Standard IEEE 1987-1991

VHDL nella progettazione dei circuiti digitali

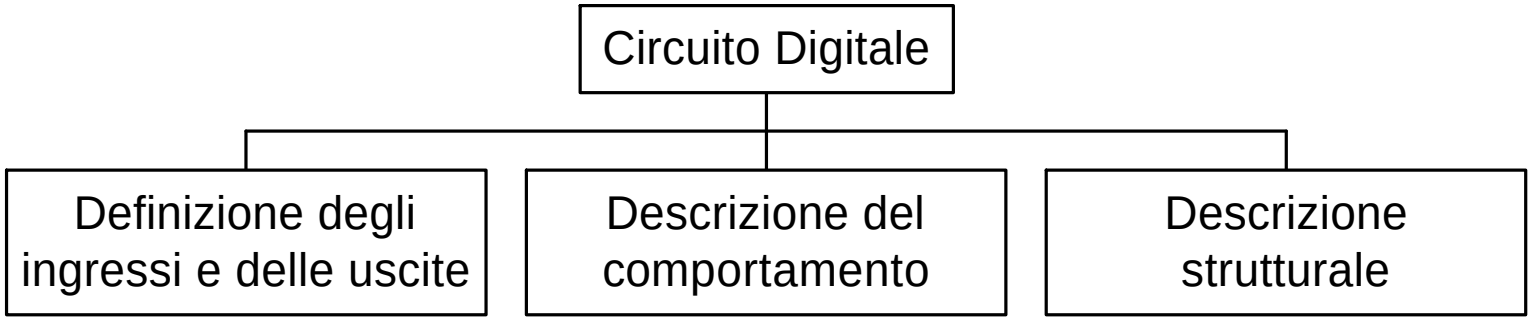




Caratteristiche principali

- È un linguaggio molto generale
- Permette di descrivere circuiti digitali in vari modi (comportamento, struttura interna, ecc.)
- È supportato dalla maggior parte dei tools di progettazione, simulazione e sintesi automatica
- Alcune istruzioni non sono sintetizzabili dai tools automatici
- È un linguaggio molto esteso

Tipi di descrizioni VHDL



ESEMPIO

In questo esempio si realizzerà una porta XOR utilizzando il linguaggio VHDL.

In particolare si mostreranno la descrizione comportamentale e quella strutturale della porta XOR.

La porta XOR

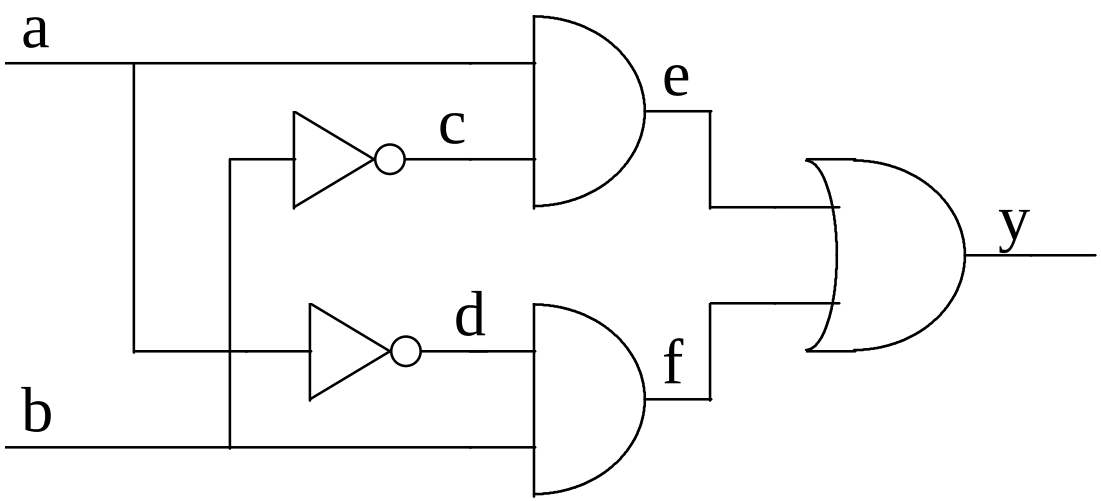
Tabella della verità

(Comportamento)

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

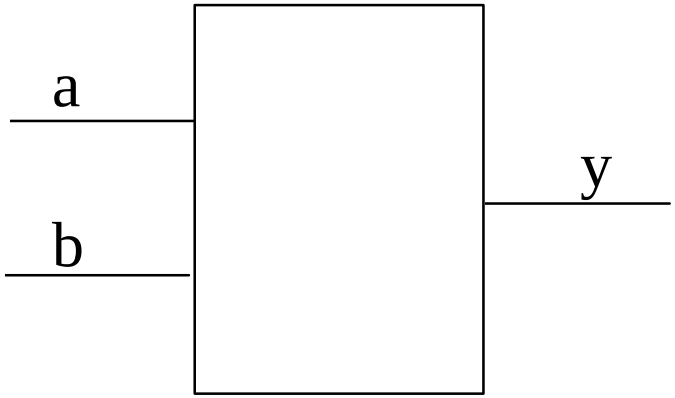
Circuito logico

(Struttura)



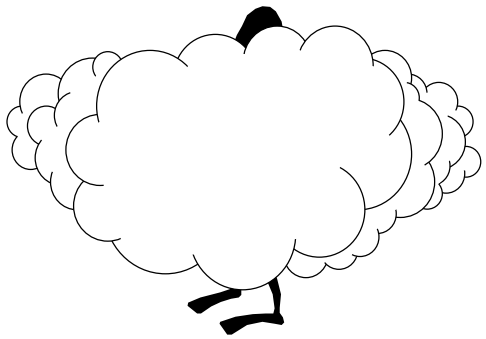
Definizione degli ingressi e delle uscite

```
ENTITY xor IS
   PORT
   (
         a : in Bit;
         b : in Bit;
         y : out Bit
   );
END xor;
```



Descrizione comportamentale (behavioural)

```
ARCHITECTURE beh_arch  OF xor IS
BEGIN
   PROCESS
   BEGIN
        y <= a  XOR  b;
   END PROCESS;
END beh_arch;
```



Descrizione comportamentale (behavioural)

```
ARCHITECTURE beh_arch  OF xor IS
    SIGNAL c : BIT;
    SIGNAL d : BIT;
    SIGNAL e : BIT;
    SIGNAL f : BIT;
BEGIN
    PROCESS
    BEGIN
        c <= NOT b;
        d <= NOT a;
        e <= a AND c;
        f <= b AND d;
        y <= e  OR  f;
    END PROCESS;
END beh_arch;
```

Descrizione Comportamentale (behavioural)

```
ARCHITECTURE beh_arch  OF xor IS
BEGIN
    PROCESS
        VARIABLE c : BIT;
        VARIABLE d : BIT;
        VARIABLE e : BIT;
        VARIABLE f : BIT;
    BEGIN
        c := NOT b;
        d := NOT a;
        e := a AND c;
        f := b AND d;
        y <= e  OR  f;
    END PROCESS;
END beh_arch;
```

Descrizione strutturale (structural)

```
ARCHITECTURE struc_arch  OF xor IS
    SIGNAL c : BIT;
    SIGNAL d : BIT;
    SIGNAL e : BIT;
    SIGNAL f : BIT;

COMPONENT        NOT
    PORT (I : IN BIT; O : OUT BIT);
END COMPONENT;

COMPONENT        AND
    PORT ( I1: IN BIT; I2: IN BIT; O: OUT BIT);
END COMPONENT;

COMPONENT        OR
    PORT ( I1: IN BIT; I2: IN BIT; O: OUT BIT);
END COMPONENT;
```

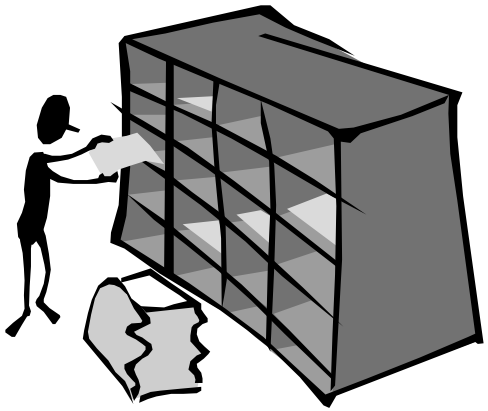
(continua)

Descrizione Strutturale (structural)

```
BEGIN
    PROCESS
    BEGIN
        u1: NOT( b => I , c => O );
        u2: NOT( a => I , d => O );
        u3: AND( a => I1 , c => I2 , e => O );
        u4: AND( b => I1 , d => I2 , f => O );
        u5: OR( e => I1 , f => I2 , y => O );
    END PROCESS;
END struc_arch;
```

Librerie di componenti

USE *ieee.std_logic_1164*.ALL;
USE *logic*.ALL



ENTITY *xor* IS
...
END *xor*;

ARCHITECTURE *arch* OF *xor* IS
BEGIN
...
END *arch*;

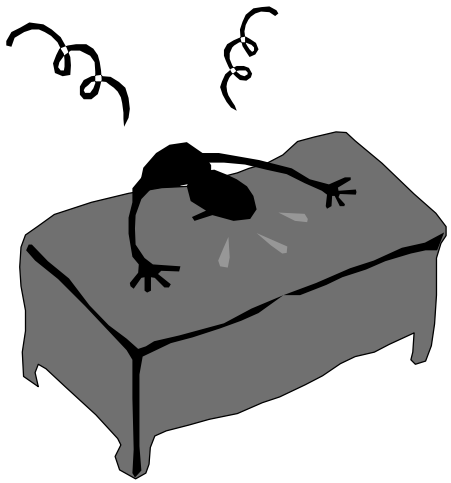
Tipi di Dati Standard

- BIT

  Dato il cui valore è ‘0’ o ‘1’

  SIGNAL *dato* : BIT;

  dato <= ‘0’;



- INTEGER

  Numero compreso fra -2147483648 e 2147483647

  SIGNAL *dato* : INTEGER;

  dato <= 18;

Tipi di Dati Standard

- REAL

  Numero compreso fra -1xE38 e 1E38 con almeno 16 cifre significative

  SIGNAL *dato* : REAL;

  dato <= 1.24;

- BOOLEAN

  Dato il cui valore è FALSE o TRUE

  VARIABLE *dato* : BOOLEAN;

  dato := ‘0’;

Tipi di Dati Standard

- CHARACTER

  Un qualunque carattere ASCII

  VARIABLE *dato* : CHARACTER;

  dato := ‘A’;

Dati di Libreria

La libreria standard “ieee.std_logic_1164” aggiunge ai dati standard i seguenti tipi di dati elementari:

- STD_LOGIC

  Dato il cui valore è ‘0’, ‘1’, ‘Z’, ‘X’

  SIGNAL *dato* : STD_LOGIC;

  dato <= ‘1’;

Dati di Libreria

- STD_LOGIC_VECTOR

  Vettore di dati tipo STD_LOGIC

  SIGNAL *dato* : STD_LOGIC_VECTOR(7 downto 0);

  dato <= ‘110X11Z0’;

  dato(4 DOWNTO *2)* <= ‘X11’;

  dato(7 DOWNTO 6*)* <= *dato(1* DOWNTO *0)*;

- SIGNED, UNSIGNED

  Vettori di bit che rappresentano numeri

  SIGNAL *dato* : UNSIGNED;

  dato <= 18;

Struttura generale di un programma

```
USE libreria

ENTITY circuito IS
...
END circuito;

ARCHITECTURE arch OF circuito IS
    SIGNAL signal_globali
BEGIN
    PROCESS
        SIGNAL signal_locali
        VARIABLE variabili_locali
    BEGIN
    ...
    END PROCESS;

    PROCESS
    ...
    END PROCESS;
END arch;
```

Confronto fra Variable e Signal

SIGNAL

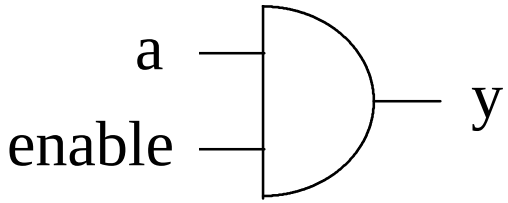
- Un nuovo valore viene aggiornato dopo un ritardo predefinito
- Ci sono SIGNAL locali ai PROCESS e globali alla ARCHITECT
- Rappresentano i fili di collegamento fra componenti

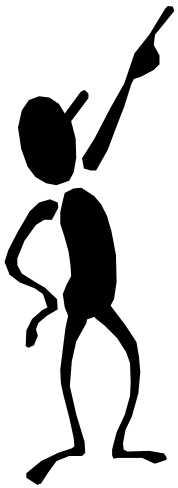
VARIABLE

- Un nuovo valore viene aggiornato immediatamente
- Ci sono VARIABLE soltanto locali ai PROCESS
- Non hanno nessuna rappresentazione

Controllo dei processi: IF-THEN-ELSE

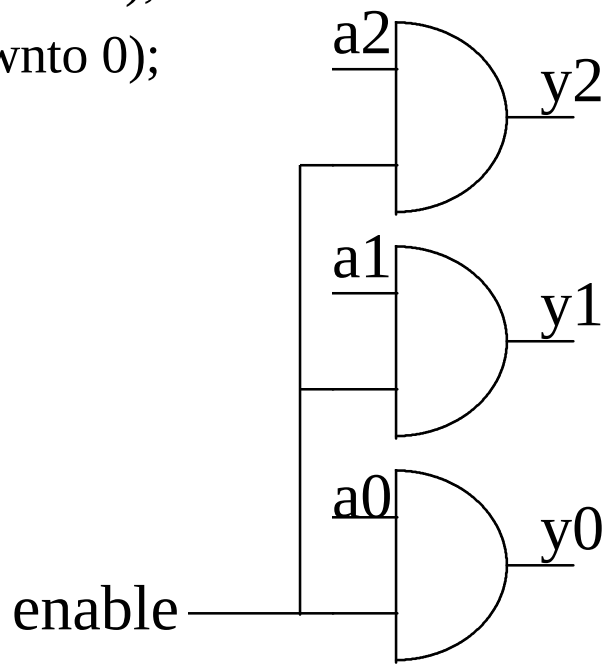
```
PROCESS
    SIGNAL a : STD_LOGIC;
    SIGNAL y : STD_LOGIC;
    SIGNAL enable : STD_LOGIC;
BEGIN
    IF (enable = ‘1’) THEN
        y <= a;
    ELSE
        y <= ‘0’;
    END IF;
END PROCESS;
```





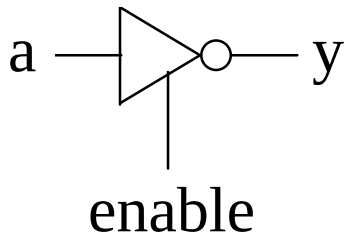
Controllo dei processi: IF-THEN-ELSE

```
PROCESS
    SIGNAL a : STD_LOGIC_VECTOR(2 downto 0);
    SIGNAL y : STD_LOGIC_VECTOR(2 downto 0);
    SIGNAL enable : STD_LOGIC;
BEGIN
    IF (enable = ‘1’) THEN
        y <= a;
    ELSE
        y <= ‘0’;
    END IF;
END PROCESS;
```



Controllo dei processi: IF-THEN-ELSE

```
PROCESS
    SIGNAL a : STD_LOGIC;
    SIGNAL y : STD_LOGIC;
    SIGNAL enable : STD_LOGIC;
BEGIN
    IF (enable = ‘1’) THEN
        y <= NOT a;
    ELSE
        y <= ‘Z’;
    END IF;
END PROCESS;
```



Controllo dei processi: IF-THEN-ELSIF-ELSE

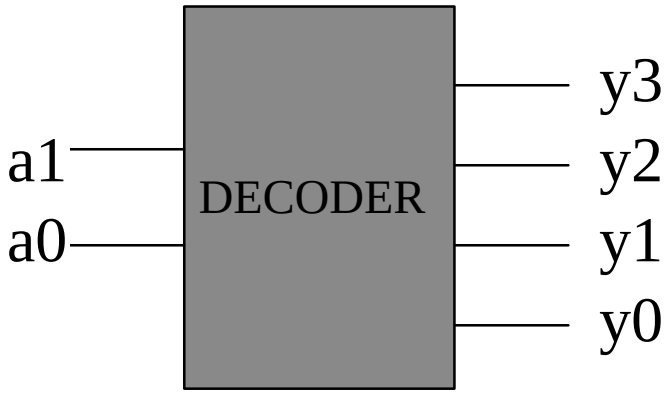
```
PROCESS
    SIGNAL a : STD_LOGIC;
    SIGNAL y : STD_LOGIC;
    SIGNAL enable : STD_LOGIC_VECTOR(1 downto 0);
BEGIN
    IF (enable = ‘00’) THEN
        y <= a;
    ELSIF (enable = ‘01’ OR enable = ‘10’) THEN
        y <= ‘0’;
    ELSE
        y <= ‘1’;
    END IF;
END PROCESS;
```

Controllo dei processi: CASE

```
PROCESS
    SIGNAL a : STD_LOGIC;
    SIGNAL y : STD_LOGIC;
    SIGNAL enable : STD_LOGIC_VECTOR(1 downto 0);
BEGIN
    CASE enable IS
        WHEN ‘00’ =>
            y <= a;
        WHEN ‘01’ =>
            y <= ‘0’;
        WHEN ‘10’ =>
            y <= ‘0’;
        WHEN OTHERS =>
            y <= ‘1’;
    END CASE;
END PROCESS;
```

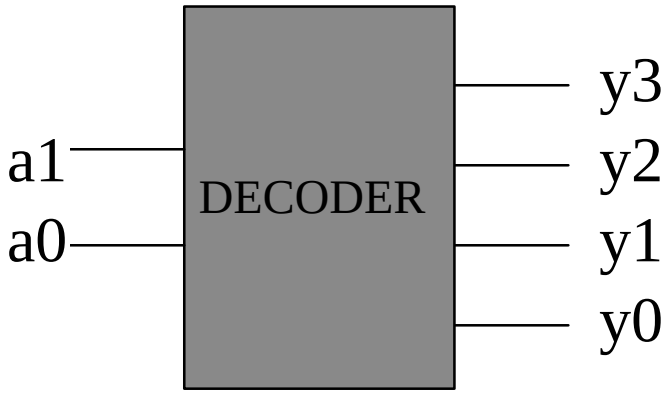
Controllo dei processi: CASE

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



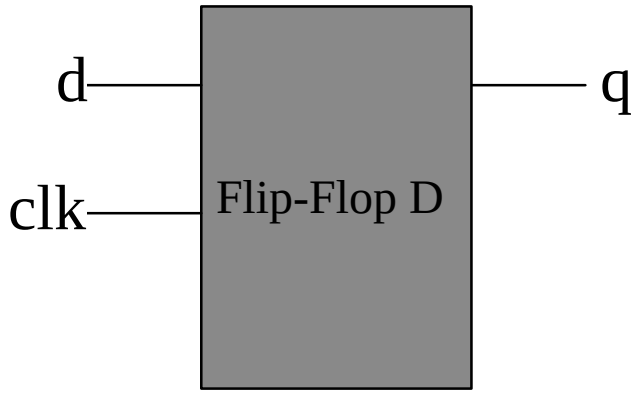
Controllo dei processi: SELECT

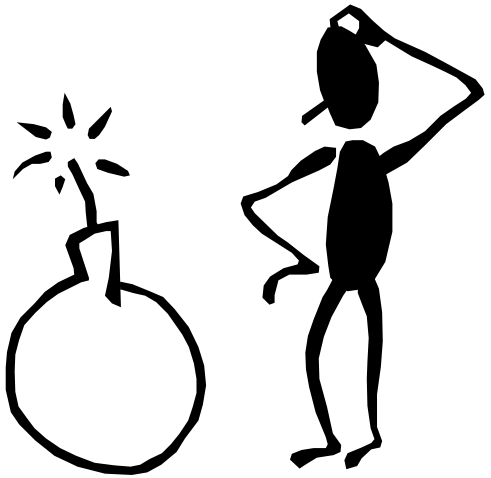
```
PROCESS
    SIGNAL y : STD_LOGIC_VECTOR(3 downto 0);
    SIGNAL a : STD_LOGIC_VECTOR(1 downto 0);
BEGIN
    WITH a SELECT
        y   <=   ‘0001’ WHEN ‘00’,
                 ‘0010’ WHEN ‘01’,
                 ‘0100’ WHEN ‘10’,
                 ‘1000’ WHEN ‘11’;
END PROCESS;
```



Circuiti Sequenziali: FF-D

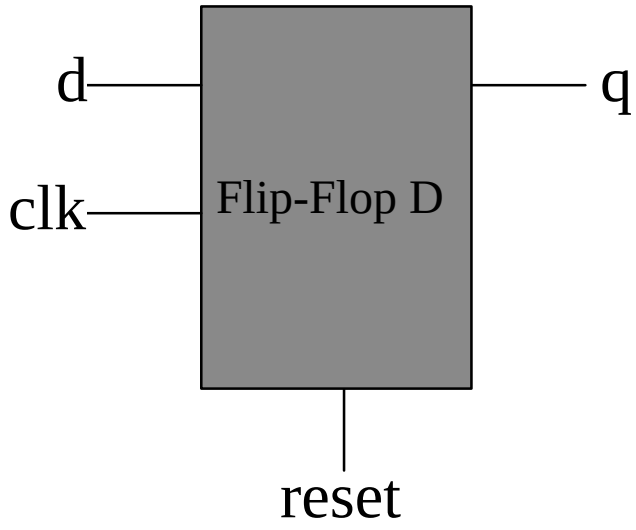
```
ARCHITECTURE arch OF ffd IS
    SIGNAL d : STD_LOGIC;
    SIGNAL q : STD_LOGIC;
    SIGNAL clk : STD_LOGIC;
BEGIN
    PROCESS(clk)
    BEGIN
        IF (clk'EVENT AND clk = '1') THEN
            q <= d;
        END IF;
    END PROCESS;
END arch;
```





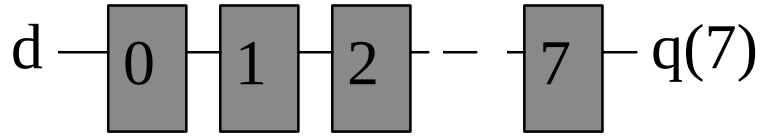
Circuiti Sequenziali: FF-D con RESET

```
ARCHITECTURE arch OF ffd IS
    SIGNAL d : STD_LOGIC;
    SIGNAL q : STD_LOGIC;
    SIGNAL clk : STD_LOGIC;
    SIGNAL reset : STD_LOGIC;
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;
```



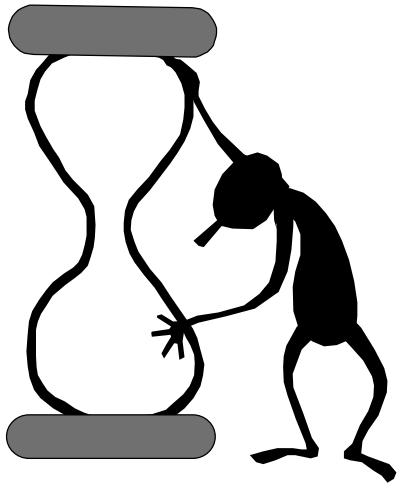
Registro di scorrimento

```
ARCHITECTURE arch OF shift IS
    SIGNAL d : STD_LOGIC;
    SIGNAL q : STD_LOGIC_VECTOR(7 downto 0);
    SIGNAL clk : STD_LOGIC;
    SIGNAL reset : STD_LOGIC;
BEGIN
    PROCESS(clk, reset)
    BEGIN
        IF (reset = '1') THEN
            q <= '00000000';
        ELSIF (clk'EVENT AND clk = '1') THEN
            q(0) <= d;
            q(7 downto 1) <= q(6 downto 0);
        END IF;
    END PROCESS;
END arch;
```



Contatore

```
ARCHITECTURE arch OF counter IS
    SIGNAL q : UNSIGNED(7 downto 0);
    SIGNAL clk : STD_LOGIC;
    SIGNAL reset : STD_LOGIC;
BEGIN
    PROCESS(clk, reset)
    BEGIN
        IF (reset = ‘1’) THEN
            q <= ‘00000000’;
        ELSIF (clk’EVENT AND clk = ‘1’) THEN
            q <= q + 1;
        END IF;
    END PROCESS;
END arch;
```



Sommatore

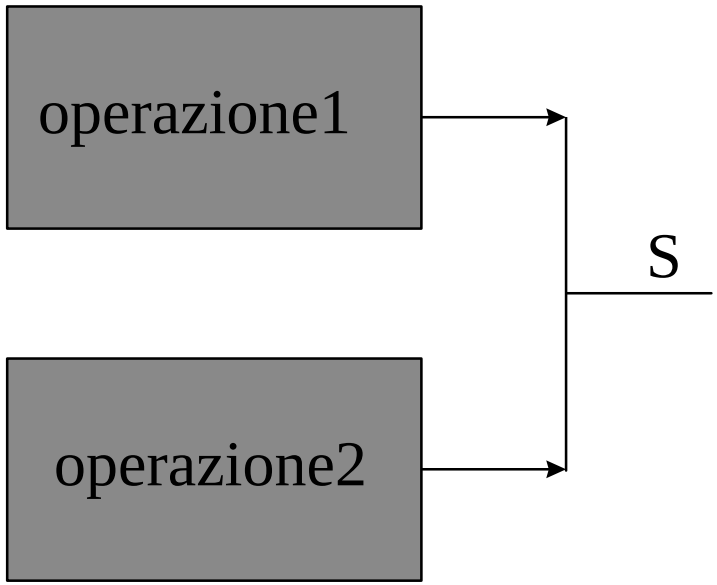
```
ARCHITECTURE arch OF adder IS
    SIGNAL d1 : SIGNED(7 downto 0);
    SIGNAL d2 : SIGNED(7 downto 0);
    SIGNAL y : SIGNED(8 downto 0);
    SIGNAL clk : STD_LOGIC;
    SIGNAL reset : STD_LOGIC;
BEGIN
    PROCESS(clk, reset)
    BEGIN
        IF (reset = ‘1’) THEN
            y <= ‘000000000’;
        ELSIF (clk’EVENT AND clk = ‘1’) THEN
            y <= d1 + d2;
        END IF;
    END PROCESS;
END arch;
```

1+1=3

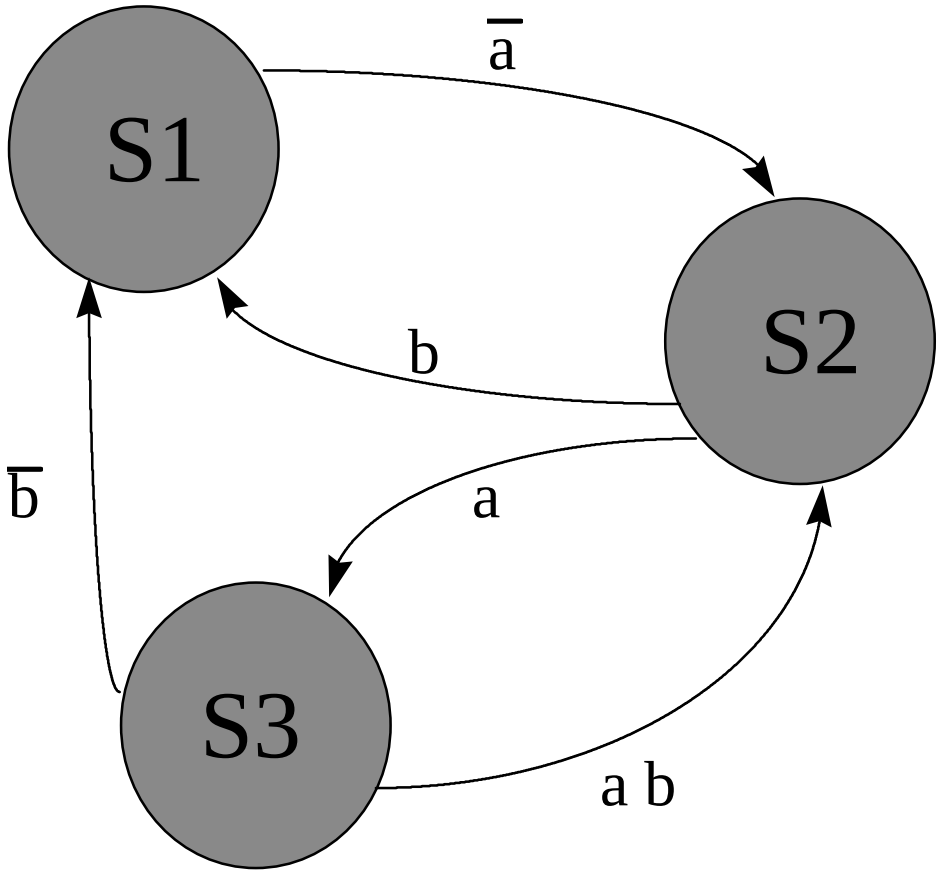
Attenzione agli errori!!
(o orrori?)

```
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 (FSA)



Macchina a stati finiti (FSA)

```
ENTITY fsm IS
  PORT
  (
    clk : IN STD_LOGIC;
    reset : IN STD_LOGIC;
    a, b : IN STD_LOGIC;
    q : OUT STD_LOGIC
  );
END fsm;

ARCHITECTURE arch OF fsm IS
  TYPE STATE_TYPE IS (s1, s2, s3);
  SIGNAL stato: STATE_TYPE;
BEGIN
  PROCESS (clk)
BEGIN
  IF reset = '1' THEN
  stato <= s1;
  ELSIF clk'EVENT AND clk = '1' THEN
    CASE stato IS
      WHEN s1 =>
        IF (NOT a) THEN
          stato <= s2;
        END IF;
      WHEN s2 =>
        IF (b) THEN
          stato <= s1;
        ELSIF (a) THEN
          stato <= s3;
        END IF;
```

Macchina a stati finiti (FSA)

```
    WHEN s3 =>
      IF (NOT b) THEN
        stato <= s1;
      ELSIF (a AND b) THEN
        stato <= s2;
      END IF;
    END CASE;
  END IF;
END PROCESS;

WITH state SELECT
  q <= '0' WHEN s1,
       '0' WHEN s2,
       '1' WHEN s3;
END arch;
```

Bibliografia

VHDL: Hardware Description and Design

Roger Lipsett
Carl Schaefer
Cary Ussery

Kluwer Academic Publisher

Bib. Elettronica L4 - 58