VHDL strutturale

Obbiettivi del Modulo

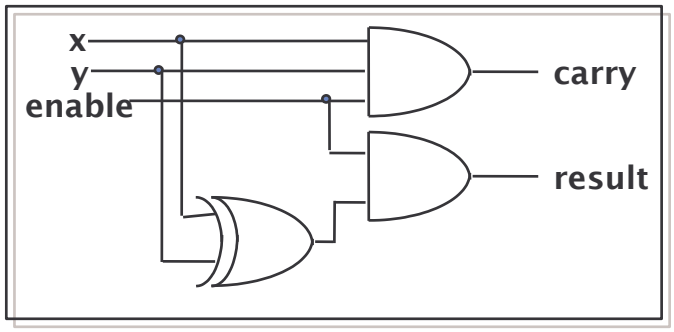
- **Introdurre i costrutti VHDL strutturali**
  - **Utilizzo dei *components***
    - ***Dichiarazione***
    - ***Instanza***
  - **Indicazioni sulla connessione dei componenti**
  - **Utilizzo delle dichiarazioni di configurazioni**
  - **Istruzione GENERATE**

Sommario

- Introduzione
- Includere oggetti VHDL
- Istruzione Generate
- Esempi
- Sommario

Introduzione

- I modelli possono essere costruiti interconnettendo dei sottocomponenti
  - Un modello strutturale elenca i componenti richiesti e le loro interconnessioni
  - Simile a un progetto schematico :



Meccanismi per Incorporare Oggetti VHDL

- **Per incorporare oggetti VHDL**
  - **Instanziazione *diretta (non disponibile. prima del* VHDL-93)**
  - **Utilizzo di dichiarazioni e istanze di *componenti***
    - **Creare c*omponenti* (i.e. dichiarazioni) e connetterli a segnali locali (i.e. istanze)**
    - ***Le istanze a componenti* si possono legare a oggetti VHDL sia :**
      - ➔ **Localmente – dentro l'architettura dove essi sono dichiarati**
      - ➔ **A livelli piu' alti nella gerarchia di progetto mediante *configurazioni***
- **Esempio di descrizione strutturale (registro):**

```
USE work.resources.all;

ENTITY reg4 IS -- 4-bit register with no
   enable
  GENERIC(tprop : delay := 8 ns;
          tsu   : delay := 2 ns);
  PORT(d0,d1,d2,d3 : IN level;
       clk : IN level;
       q0,q1,q2,q3 : OUT level);
END reg4;
```

5

Esempio di 4-Bit Register

- **Prima si devono trovare i blocchi costruttivi**
  - **Riutilizzo di un oggetto dagli esempi nel modulo precedente**

```
USE work.resources.all;

ENTITY dff IS

  GENERIC(tprop : delay := 8 ns;
          tsu   : delay := 2 ns);

  PORT(d      : IN level;
       clk    : IN level;
       enable : IN level;
       q      : OUT level;
       qn     : OUT level);

END dff;
```

```
ARCHITECTURE behav OF dff IS
  BEGIN
   one : PROCESS (clk)
     BEGIN
       -- first, check for rising clock edge
       -- and check that ff is enabled
       IF ((clk = '1' AND clk'LAST_VALUE = '0')
            AND enable = '1') THEN
         -- now check setup requirement is met
         IF (d'STABLE(tsu)) THEN
           -- now check for valid input data
           IF (d = '0') THEN
             q <= '0' AFTER tprop;
             qn <= '1' AFTER tprop;
           ELSIF (d = '1') THEN
             q <= '1' AFTER tprop;
             qn <= '0' AFTER tprop;
           ELSE -- else invalid data
             q <= 'X';
             qn <= 'X';
           END IF;
         ELSE -- else setup not met
           q <= 'X';
           qn <= 'X';
         END IF;
       END IF;
   END PROCESS one;
END behav;
```

6

Passi generali per incorporare oggetti VHDL

- **Per incorporare in architettura un oggetto VHDL, questo deve essere :**
  - **dichiarato – definendo un interfaccia locale**
  - **instanziato – connettendo i segnali locali all'interfaccia locale**
    - **Strutture regolari possono essere facilmente generate utilizzando l'istruzione *GENERATE* nelle istanze dei componenti**
  - **legato – selezionando una entity/architecture che lo realizza**

Dichiarazioni di Componenti e Legami Locali

- **Le dichiarazioni di componenti definiscono interfacce per oggetti locali**
- **Tali dichiarazioni si possono mettere sia nelle dichiarazioni di architetture che di package**
- **Le istanze a tali componenti connettono segnali locali ai segnali di interfaccia**

dichiarazione

legame (binding)

instanze

label

```
USE work.resources.all;

ARCHITECTURE struct_2  OF reg4 IS
  COMPONENT reg1 IS
     PORT (d, clk : IN level;
           q : OUT level);
  END COMPONENT reg1;
  CONSTANT enabled : level := '1';
  FOR ALL : reg1 USE work.dff(behav)
     PORT MAP(d=>d,clk=>clk,enable=>enabled,q=>q,qn=>OPEN);
  END FOR;
BEGIN
     r0 : reg1 PORT MAP (d=>d0,clk=>clk,q=>q0);
     r1 : reg1 PORT MAP (d=>d1,clk=>clk,q=>q1);
     r2 : reg1 PORT MAP (d=>d2,clk=>clk,q=>q2);
     r3 : reg1 PORT MAP (d=>d3,clk=>clk,q=>q3);
END struct_2;
```

Configurazione del port map la si usa quando il port map non corrisponde a quello di default specificato nell'interfaccia della entity instanziata

Configurazioni

- **Rappresentano il modo per legare un istanza di componente a una architettura di una entity**
- **In particolare per una data entity di un componenet instanziato, definiscono quale architettura utilizzare**
- **L'esempio di precedente è una configurazione di tio entity-architecture**
- **E' possibile anche utilizzare configurazioni al livello di architettura consentendo ad architetture a un livello superiore che istanziano quella considerata di determinare quale configurazione utilizzare**

Dichiarazioni di componenti e configurazioni

```
USE work.resources.all;

ARCHITECTURE struct_3  OF reg4 IS
  COMPONENT reg1 IS
    PORT (d, clk : IN level;
          q : OUT level);
  END COMPONENT reg1;
  CONSTANT enabled : level := '1';
 BEGIN
    r0 : reg1 PORT MAP (d<=d0,clk<=clk,q<=q0);
    r1 : reg1 PORT MAP (d<=d1,clk<=clk,q<=q1);
    r2 : reg1 PORT MAP (d<=d2,clk<=clk,q<=q2);
    r3 : reg1 PORT MAP (d<=d3,clk<=clk,q<=q3);
END struct_3;
```

```
USE work.resources.all;

CONFIGURATION reg4_conf_1 OF reg4 IS
  CONSTANT enabled : level := '1';
  FOR struct_3
    FOR all : reg1 USE work.dff(behav)
       PORT MAP(d=>d,clk=>clk,enable=>enabled,q=>q,qn=>OPEN);
    END FOR;
  END FOR;
END reg4_conf_1;
```

```
-- Architecture in which a COMPONENT for reg4 is declared
 ...
FOR ALL : reg4_comp USE CONFIGURATION work.reg4_conf_1;
 ...
```

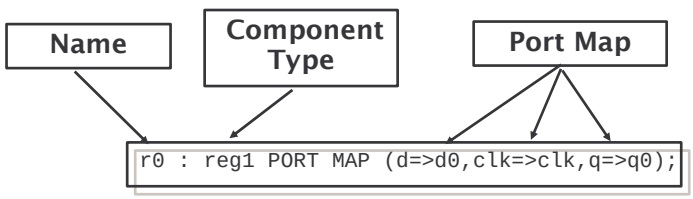
Dichiarazioni di configurazioni

- **Ragioni per utilizzarle:**
  - **grandi progetti spesso occupano livelli multipli di gerarchia**
  - **mentre si sviluppa l'architettura, puo' essere disponibile solo l'interfaccia del componente**
  - **meccanismo per mettere insieme diversi pezzi di progetto**
- **Le configurazioni si possono utilizzare per specializzare l'utilizzo di interfaccie secondo i bisogni :**
  - **il nome di entity puo' essere diverso da quello del componente**
  - **le entity possono avere piu' porte delle dichiarazioni dei componenti**
  - **le porte nella dichiarazione di entity possono avere nomi diversi da quelli nella dichiarazione del componente**

11

Istruzione di Instanziazione

- **Connette un componente dichiarato a segnali nell' architettura**
- **L'instanziazione ha 3 parti**
  - **Name – identifica un unica *instanza* del componente**
  - **Component type – seleziona uno dei componenti dichiarati**
  - **Port map -- connette ai segnali dell'architettura**
    - **Eventualmente con Generic Map**



12

Generic Map

- **Permettono di specializzare il componente mentre questo viene instanziato**
  - **Le dichiarazioni di entity danno valori di default**
- **La GENERIC MAP e' simile alla PORT MAP perche' mappa specifici valori sui generics del componente**

```
USE Work.my_stuff.ALL
ARCHITECTURE test OF test_entity
               SIGNAL S1, S2, S3 : BIT;
BEGIN
       Gate1 : my_stuff.and_gate -- component found in package
               GENERIC MAP (tplh=>2 ns, tphl=>3 ns)
               PORT MAP (S1, S2, S3);
END test;
```


Specifiche per referenziare componenti

- **Mettono a disposizione le informazioni necessarie per fare riferimento a componenti**
  - **Singolo componente**

```
FOR A1 : and_gate USE binding_indication;
```

  - **Componenti multipli**

```
FOR A1, A2 : and_gate USE binding_indication;
```

  - **Tutti i componenti**

```
FOR ALL : and_gate USE binding_indication;
```

-- Condiziona tutti i componenti di tipo AND

  - **Altri componenti**

```
FOR OTHERS : and_gate USE binding_indication;
```

-- Si applica a tutti i componenti per cui non sono state date altre specifiche

Indicazioni per selezionare un componente da una libreria

- ***Binding indication*** **identifica il componente da instanziare**
- **Sono disponibili due meccanismi :**
  - **Riferimento a una entity/architecture del VHDL**

```
FOR ALL : reg1 USE work.dff(behav);
```

  - **Configurazione VHDL**

```
FOR reg4_inst : reg4_comp USE CONFIGURATION work.reg4_conf_1;
```

- **PORT MAP e/o GENERIC MAP possono essere poi utilizzate per specificare i singoli componenti**

Riferimenti diretti

- **Mettono a disposizione un semplice meccanismo per fare riferimento a componenti definiti precedentemente**
- **Sono disponibili solamente al livello gerarchico immediatamente superiore a quello che li utilizza**

```
USE work.resources.all;

ARCHITECTURE struct_1  OF reg4 IS
  CONSTANT enabled : level := '1';
 BEGIN
    r0 : ENTITY work.dff(behav)
         PORT MAP (d0,clk,enabled,q0,OPEN);
    r1 : ENTITY work.dff(behav)
         PORT MAP (d1,clk,enabled,q1,OPEN);
    r2 : ENTITY work.dff(behav)
         PORT MAP (d2,clk,enabled,q2,OPEN);
    r3 : ENTITY work.dff(behav)
         PORT MAP (d3,clk,enabled,q3,OPEN);
END struct_1;
```

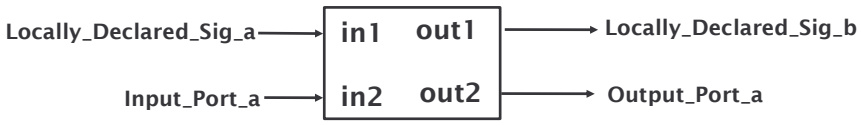
Riferimenti diretti

- **Mettono a disposizione un semplice meccanismo per fare riferimento a componenti definiti precedentemente**
- **Sono disponibili solamente al livello gerarchico immediatamente superiore a quello che li utilizza**

```
USE work.resources.all;

ARCHITECTURE struct_1  OF reg4 IS
  CONSTANT enabled : level := '1';
 BEGIN
    r0 : ENTITY work.dff(behav)
         PORT MAP (d0,clk,enabled,q0,OPEN);
    r1 : ENTITY work.dff(behav)
         PORT MAP (d1,clk,enabled,q1,OPEN);
    r2 : ENTITY work.dff(behav)
         PORT MAP (d2,clk,enabled,q2,OPEN);
    r3 : ENTITY work.dff(behav)
         PORT MAP (d3,clk,enabled,q3,OPEN);
END struct_1;
```


Actual e Local

- ***actual*** **puo' essere sia un segnale dichiarato dentro l'architettura, sia una porta nella dichiarazione di *entity***
  - **Una porta di un componente e' nota come *local* e deve corrispondere a un *actual compatibile***
- **Restrizioni**
  - **Local e actual devono essere dello stesso tipo**
  - **Local e actual devono avere modi compatibili**



Sommario per il VHDL strutturale

- **Il VHDL supporta vari livelli di astrazione**
  - **Instanze dirette richiedono una conoscenza dettagliata dei blocchi che vengono incorporati (bottom-up)**
  - **L'utilizzo dei *componenti* permette la definizione e l'utilizzo di interfaccie astratte per i componenti locali (top-down)**
    - **Si possono definire interfaccie locali per componenti connessi a segnali locali**
    - **I componenti possono essere collegati a oggetti VHDL eventualmente definiti in altri file (i.e. descrizioni di *entity/architecture*)**
      - **➔ Sia localmente, sia a livelli gerarchici piu' alti mediante l'utilizzo di configurazioni**
- ***Segnali actual* e *local* devono avere modi e tipi compatibili**

19

Istruzione Generate

- **Il VHDL mette a disposizione l'istruzione GENERATE per creare facilmente strutture ripetitive**
  - **Ad esempio, RAMs, adders**
- **Qualsiasi istruzione concorrente puo' essere inclusa in un istruzione GENERATE, includendo anche altre istruzioni GENERATE**
  - **In particolare, dentro a GENERATE, si possono instanziare componenti**

20

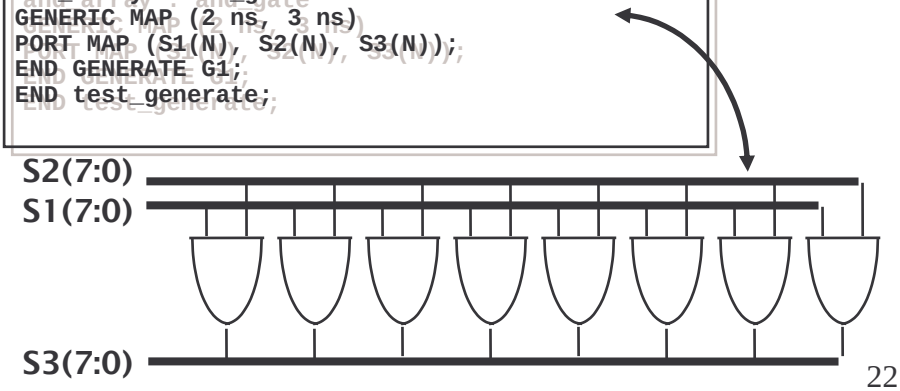
Istruzione Generate costrutto FOR

- **Serve per creare oggetti simili fra loro**
- **Il parametro di GENERATE deve essere discreto ed e' non definito al di fuori dell'istruzione GENERATE**
- **Il ciclo non puo' essere terminato anticipatamente**

```
name : FOR N IN 1 TO 8 GENERATE
       concurrent-statements
END GENERATE name;
```


FOR – Esempio

```
-- this uses the and_gate component from before
ARCHITECTURE test_generate OF test_entity IS
SIGNAL S1, S2, S3: BIT_VECTOR(7 DOWNTO 0);
BEGIN
G1 : FOR N IN 7 DOWNTO 0 GENERATE
and_array : and_gate
GENERIC MAP (2 ns, 3 ns)
PORT MAP (S1(N), S2(N), S3(N));
END GENERATE G1;
END test_generate;
```



Istruzione Generate
Costrutto IF

- **Permette di creare componenti in modo condizionale**
- **Non puo' usare condizioni ELSE o ELSIF dentro l'IF**

```
name : IF (boolean expression) GENERATE
       concurrent-statements
END GENERATE name;
```

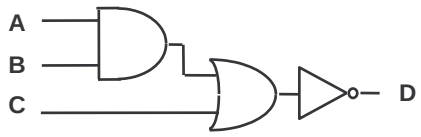

IF- Esempio


```
ARCHITECTURE test_generate OF test_entity
SIGNAL S1, S2, S3: BIT_VECTOR(7 DOWNTO 0);
BEGIN
G1 : FOR N IN 7 DOWNTO 0 GENERATE

G2 : IF (N = 7) GENERATE
or1 : or_gate
  GENERIC MAP (3 ns, 3 ns)
  PORT MAP (S1(N), S2(N), S3(N));
END GENERATE G2;

G3 : IF (N < 7) GENERATE
and_array : and_gate
  GENERIC MAP (2 ns, 3 ns)
  PORT MAP (S1(N), S2(N), S3(N));
END GENERATE G3;

END GENERATE G1;
END test_generate;
```

Esempi

- **Costruzione di moduli ad alto livello a partire da una libreria di gate CMOS**
  - **AND-OR-Invert**
  - **8 Bit Register utilizzando DFFs**
  - **8 Bit Shift Register utilizzando Multiplexers e DFFs**
- **Utilizzare questi moduli per costruire un datapath per la moltiplicazione di unsigned a 8 bit**

un gate And-Or-Invert Gate (Entity)




```
LIBRARY gate_lib;
USE gate_lib.resources.all;

ENTITY aoi2_str IS

  GENERIC(trise : delay := 12 ns;
          tfall : delay := 9 ns);

  PORT(a : IN level;
       b : IN level;
       c : IN level;
       d : OUT level);

END aoi2_str;
```

Descrizione strutturale di un gate And-Or-Invert (Architecture)

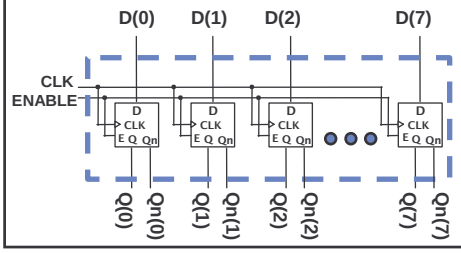
```
ARCHITECTURE structural OF aoi2_str IS

-- COMPONENT DECLARATIONS
  COMPONENT and2
    GENERIC(trise : delay;
            tfall : delay);
    PORT(a : IN level;
         b : IN level;
         c : OUT level);
  END COMPONENT;

  COMPONENT or2
    GENERIC(trise : delay;
            tfall : delay);
    PORT(a : IN level;
         b : IN level;
         c : OUT level);
  END COMPONENT;

  COMPONENT inv
    GENERIC(trise : delay;
            tfall : delay);
    PORT(a : IN level;
         b : OUT level);
  END COMPONENT;
```

```
  -- BINDING INDICATIONS
  FOR ALL : and2 USE ENTITY gate_lib.and2(behav);
  FOR ALL : or2  USE ENTITY gate_lib.or2(behav);
  FOR ALL : inv  USE ENTITY gate_lib.inv(behav);

  SIGNAL and_out : level;        -- signal for
   output
                                 -- of AND gate
  SIGNAL or_out  : level;        -- signal for
   output
                                 -- of OR gate
  BEGIN
    -- COMPONENT INSTANTIATIONS
    AND_1 : and2 GENERIC MAP(trise => trise,
                             tfall => tfall)
                 PORT MAP(a => a, b => b,
                          c => and_out);

    OR_1  : or2  GENERIC MAP(trise => trise,
                             tfall => tfall)
                 PORT MAP(a => and_out, b => c,
                          c => or_out);

    INV_1 : inv  GENERIC MAP(trise => trise,
                             tfall => tfall)
                 PORT MAP(a => or_out, b => d);
END structural;
```

27

Descrizione strutturale di un registro a 8 Bit Istruzione Generate

```
LIBRARY gate_lib;
USE gate_lib.resources.all;

ENTITY reg8_str IS

  GENERIC(tprop : delay := 8 ns;
          tsu   : delay := 2 ns);

  PORT(d      : IN level_vector(7 DOWNTO 0);
       clk    : IN level;
       enable : IN level;
       q      : OUT level_vector(7 DOWNTO 0);
       qn     : OUT level_vector(7 DOWNTO 0));

END reg8_str;
```



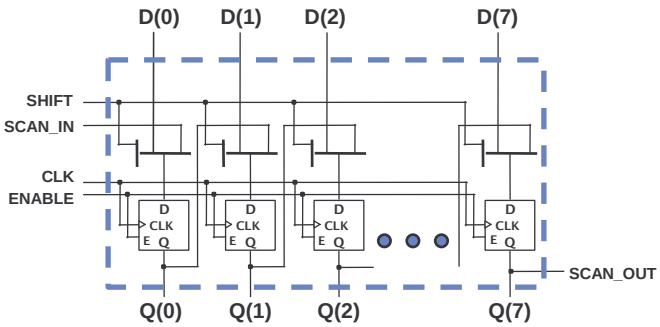

```
ARCHITECTURE structural OF reg8_str IS

  -- COMPONENT DECLARATION
  COMPONENT dff
    GENERIC(tprop : delay;
            tsu   : delay);
    PORT(d      : IN level;
         clk    : IN level;
         enable : IN level;
         q      : OUT level;
         qn     : OUT level);
  END COMPONENT;

  -- BINDING INDICATIONS
  FOR ALL : dff USE ENTITY gate_lib.dff(behav);

  BEGIN

  -- COMPONENT INSTANTIATION (GENERATE)
   R1:FOR i IN 1 TO 8 GENERATE
     I1:dff GENERIC MAP(tprop => tprop,
                        tsu => tsu)
          PORT MAP(d => d(i-1), clk => clk,
                   enable => enable,
                   q => q(i-1), qn => qn(i-1));
   END GENERATE R1;

END structural;
```

28

Descrizione strutturale di un 8 Bit Shift Register (Entity)


```
LIBRARY gate_lib;
USE gate_lib.resources.all;

ENTITY shift_reg8_str IS

  GENERIC(tprop : delay := 15 ns;
          tsu   : delay := 2 ns);
```

```
  PORT(d       : IN level_vector(7 DOWNTO 0);
       clk     : IN level;
       enable  : IN level;
       scan_in : IN level;
       shift   : IN level;
       scan_out : OUT level;
       q       : OUT level_vector(7 DOWNTO 0));

END shift_reg8_str;
```



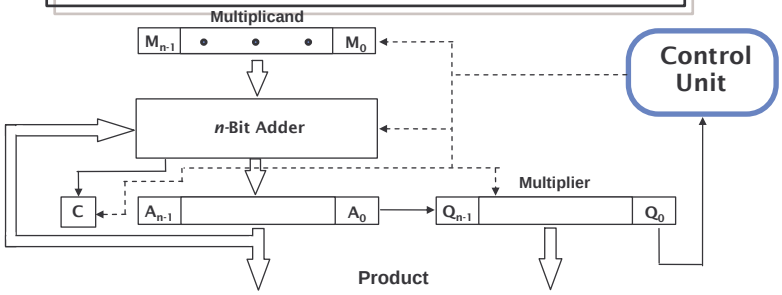
Descrizione strutturale di un 8 Bit Shift Register


```
ARCHITECTURE structural OF shift_reg8_str IS

  -- COMPONENT DECLARATION
  COMPONENT mux2
    GENERIC(tprop : delay);
    PORT(a   : IN level;
         b   : IN level;
         sel : IN level;
         c   : OUT level);
  END COMPONENT;


  COMPONENT dff
    GENERIC(tprop : delay;
            tsu   : delay);
    PORT(d      : IN level;
         clk    : IN level;
         enable : IN level;
         q      : OUT level;
         qn     : OUT level);
  END COMPONENT;

  -- BINDING INDICATIONS
  FOR ALL : mux2 USE ENTITY
            gate_lib.mux2(behav);
  FOR ALL : dff USE ENTITY
            gate_lib.dff(behav);

  SIGNAL mux_out : level_vector(7 DOWNTO 0);
  SIGNAL dff_out : level_vector(7 DOWNTO 0);

BEGIN
```

```
  -- COMPONENT INSTANTIATION (GENERATE W/ IF)
  G1:FOR i IN 0 TO 7 GENERATE
     G2 : IF (i = 0) GENERATE
       MUX1 : mux2 GENERIC MAP(tprop => tprop/2)
                   PORT MAP(a => scan_in,
                            b => d(i),
                            sel => shift,
                            c => mux_out(i));
       DFF1 : dff GENERIC MAP(tprop => tprop/2,
                              tsu => tsu)
                  PORT MAP(d => mux_out(i),
                           clk => clk,
                           enable => enable,
                           q => dff_out(i));
       q(i) <= dff_out(i);
     END GENERATE G2;
     G3 : IF (i > 0) GENERATE
       MUX1 : mux2 GENERIC MAP(tprop => tprop/2)
                   PORT MAP(a => dff_out(i-1),
                            b => d(i),
                            sel => shift,
                            c => mux_out(i));
       DFF1 : dff GENERIC MAP(tprop => tprop/2,
                              tsu => tsu)
                  PORT MAP(d => mux_out(i),
                           clk => clk,
                           enable => enable,
                           q => dff_out(i));
       q(i) <= dff_out(i);
     END GENERATE G3;
  END GENERATE G1;
  scan_out <= dff_out(7);
END structural;
```

Data path di un moltiplicatore a 8 Bit (Entity)

```
LIBRARY ieee;
LIBRARY gate_lib;
USE gate_lib.resources.all;
ENTITY mult_datapath IS
  PORT(multiplicand : IN level_vector(7 DOWNTO 0);
       multiplier   : IN level_vector(7 DOWNTO 0);
       a_enable     : IN level;  -- clock enable for A register
       a_reset      : IN level;  -- Reset control for A register
       a_mode       : IN level;  -- Shift or load mode for A
       c_enable     : IN level;  -- clock enable for c register
       m_enable     : IN level;  -- clock enable for M register
       q_enable     : IN level;  -- clock enable for Q register
       q_mode       : IN level;  -- Shift or load mode for Q
       clk          : IN level;
       product      : OUT level_vector(15 DOWNTO 0));
END mult_datapath;
```



31

Data path di un moltiplicatore a 8 Bit Multiplier (Architecture)

```
ARCHITECTURE structural OF mult_datapath IS

  COMPONENT dff
    GENERIC(tprop : delay;
            tsu   : delay);
    PORT(d      : IN level;
         clk    : IN level;
         enable : IN level;
         q      : OUT level;
         qn     : OUT level);
  END COMPONENT;

  COMPONENT reg8_str
    GENERIC(tprop : delay;
            tsu   : delay);
    PORT(d      : IN level_vector(0 TO 7);
         clk    : IN level;
         enable : IN level;
         q      : OUT level_vector(0 TO 7);
         qn     : OUT level_vector(0 TO 7));
  END COMPONENT;
```

```
COMPONENT shift_reg8_str
  GENERIC(tprop : delay;
          tsu   : delay);
  PORT(d        : IN level_vector(0 TO 7);
       clk      : IN level;
       enable   : IN level;
       scan_in  : IN level;
       shift    : IN level;
       scan_out : OUT level;
       q        : OUT level_vector(0 TO 7));
END COMPONENT;

COMPONENT alu_str
  GENERIC(tprop : delay);
  PORT(a    : IN level_vector(7 DOWNTO 0);
       b    : IN level_vector(7 DOWNTO 0);
       mode : IN level;
       cin  : IN level;
       sum  : OUT level_vector(7 DOWNTO 0);
       cout : OUT level);
END COMPONENT;

FOR ALL : and2  USE ENTITY gate_lib.and2(behav);
FOR ALL : dff   USE ENTITY gate_lib.dff(behav);
FOR ALL : reg8_str  USE ENTITY
                      work.reg8_str(structural);
FOR ALL : shift_reg8_str  USE ENTITY
                  work.shift_reg8_str(structural);
FOR ALL : alu_str  USE ENTITY
                     work.alu_str(structural);
```

32

moltiplicatore a 8 Bit Multiplier (Architecture)


```
SIGNAL gnd                    : level := '0';
SIGNAL c_out, a_scan_out, carry_out : level;
SIGNAL a_out, alu_out, m_out
                    : level_vector(7 DOWNTO 0);

BEGIN

 -- A, C, M, and Q registers
 A1 : shift_reg8_str GENERIC MAP(6 ns, 1 ns)
      PORT MAP(d(0)=>alu_out(0),d(1)=>alu_out(1),
               d(2)=>alu_out(2),d(3)=>alu_out(3),
               d(4)
   =>alu_out(4),d(5)=>alu_out(5),
               d(6)=>alu_out(6),d(7)=>alu_out(7),
               clk => clk, enable => a_enable,
               scan_in => c_out, shift => a_mode,
               scan_out => a_scan_out,
               q(0)=>a_out(0),q(1)=>a_out(1),
               q(2)=>a_out(2),q(3)=>a_out(3),
               q(4)=>a_out(4),q(5)=>a_out(5),
               q(6)=>a_out(6),q(7)=>a_out(7));

 C1 : dff GENERIC MAP(5 ns, 1 ns)
      PORT MAP( d => carry_out, clk => clk,
                enable => c_enable, q => c_out);

 M1 : reg8_str GENERIC MAP(4 ns, 1 ns)
      PORT MAP(d => multiplicand, clk => clk,
               enable => m_enable, q => m_out);
```

```
Q1 : shift_reg8_str GENERIC MAP(6 ns,1 ns)
     PORT MAP(d(0) => multiplier(0),
              d(1) => multiplier(1),
              d(2) => multiplier(2),
              d(3) => multiplier(3),
              d(4) => multiplier(4),
              d(5) => multiplier(5),
              d(6) => multiplier(6),
              d(7) => multiplier(7),
              clk => clk,
              enable => q_enable,
              scan_in => a_scan_out,
              shift => q_mode,
              q(0) => product(0),
              q(1) => product(1),
              q(2) => product(2),
              q(3) => product(3),
              q(4) => product(4),
              q(5) => product(5),
              q(6) => product(6),
              q(7) => product(7));
 -- ALU
 ALU1 : alu_str GENERIC MAP(8 ns)
        PORT MAP(a => m_out, b => a_out,
                 mode => a_reset,
                 cin => gnd,
                 sum => alu_out,
                 cout => carry_out);

 -- connect A register output to product
 product(15 DOWNTO 8)<=a_out(7 DOWNTO 0);
END structural;
```


Sommario

- **Il VHDL strutturale descrive la topologia dei componenti**
- **Questi possono essere dati a qualsiasi livello di astrazione – da gate a basso livello a blocchi complessi**
- **I parametri di tipo generic sono ereditati dall'architettura di qualsiasi componente di quel tipo**
- **L'istruzione GENERATE puo' creare facilmente grandi blocchi di logica regolare**
- **Le configurationi danno al progettista il controllo sulle entity e architecture utilizzate per il componente**

References

[Bhasker95] Bhasker, J. A VHDL Primer, Prentice Hall, 1995.

[Coelho89] Coelho, D. R., The VHDL Handbook, Kluwer Academic Publishers, 1989.

[Lipsett89] Lipsett, R., C. Schaefer, C. Ussery, VHDL: Hardware Description and Design, Kluwer Academic Publishers, , 1989.

[LRM94] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-1993, 1994.

[Navabi93] Navabi, Z., VHDL: Analysis and Modeling of Digital Systems, McGraw-Hill, 1993.

[Menchini94] Menchini, P., "*Class Notes for Top Down Design with VHDL*", 1994.

[MG90] An Introduction to Modeling in VHDL, Mentor Graphics Corporation, 1990.

[MG93] Introduction to VHDL, Mentor Graphics Corporation, 1993.

[Perry94] Perry, D. L., VHDL, McGraw-Hill, 1994.

[Calhoun95] Calhoun, J.S., Reese, B., *"Class Notes for EE-4993/6993: Special Topics in Electrical Engineering (VHDL)"*, Mississippi State University, http://www.erc.msstate.edu/mpl/vhdl-class/html, 1995.

[Williams94] Williams, R. D., *"Class Notes for EE 435: Computer Organization and Design"*, University of Virginia, 1994.