VHDL strutturale

M. Favalli

DE: Department of Engineering Ferrara

VHDL strutturale

- Introduzione
- Includere oggetti VHDL
  - dichiarazione
  - binding
  - instanza
  - connessione
- Istruzione **`generate`**
- Esempi
- Sommario

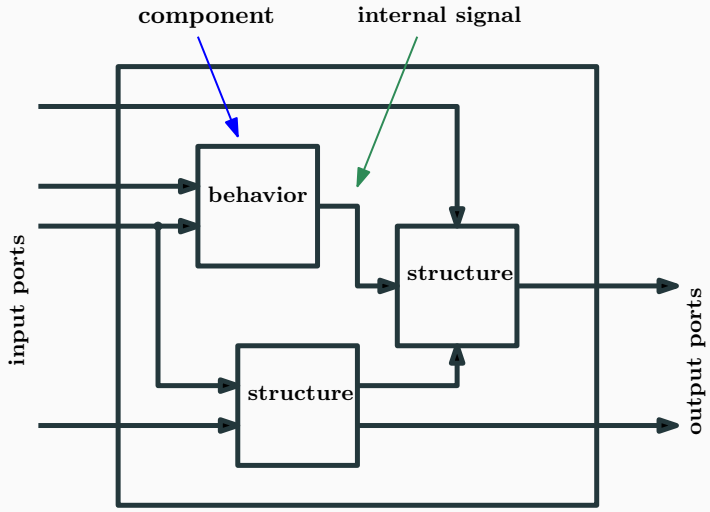
Modello strutturale in HDL

- Rete di componenti interconnessi fra loro in maniera simile a un progetto schematico
- Il modello VHDL strutturale deve consentire di:
  - definire l'interfaccia verso l'esterno (porte)
  - elencare i componenti e i segnali interni utilizzati
  - descrivere le interconnessioni fra i componenti e le porte
  - fornire indicazioni su dove reperire i modelli VHDL di tali componenti

Nota

Il VHDL non ha componenti predefiniti, quindi una descrizione strutturale é comunque di tipo gerarchico e al livello piú basso ci saranno descrizioni comportamentali dei componenti utilizzati

Esempio di struttura gerarchica



VHDL strutturale: obbiettivi

- Capacitá di descrivere semplici reti al livello strutturale
- Supporto a metodologie di progetto basate su strutture gerarchiche
- Supporto alla descrizione di strutture ripetitive
- Capacitá di interpretare modelli VHDL strutturali

Ruolo del VHDL strutturale nella sintesi di sistemi digitali

- Metodologie di progetto di tipo top-down supportate da algoritmi e tool di EDA
  - **behavioral** $\Rightarrow$ **RTL** $\Rightarrow$ **gate**
- Metodologie di progetto di tipo bottom-up supportate da descrizioni strutturali di tipo gerarchico
  - **componenti elementari** $\Rightarrow$ **blocchi funzionali** $\Rightarrow$ **micro-architettura** $\Rightarrow$ **....**
- In entrambi i casi si arriva a una descrizione strutturale in termini di componenti elementari interconnessi
  - ingresso ai tool di sintesi fisica

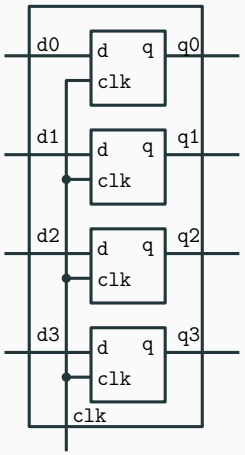
VHDL strutturale: introduzione

Tecniche per incorpare oggetti VHDL

- utilizzo di dichiarazioni e instanze di componenti
  - creare componenti (i.e. dichiarazioni) e connetterli a segnali locali (i.e. instanze)
  - le instanze a componenti si possono legare a oggetti VHDL sia:
    - localmente: dentro l'architettura dove essi sono dichiarati
    - a livelli piú alti nella gerarchia di progetto: mediante configurazioni
- **instanziazione diretta** (non disponibile prima del VHDL-93)

Esempio di VHDL strutturale - specifiche

Descrizione strutturale di un registro a 4 bit realizzato con FF D



- approccio bottom-up, si suppone che siano disponibili FF D
- problema: i FF D disponibili (indicati nel lucido seguente) hanno funzionalitá non richieste
- la descrizione é molto semplice
- i modelli VHDL possono essere nello stesso file o in file separati

Esempio di VHDL strutturale - basic building block

```
library ieee;
use ieee.std_logic_1164.all

entity dff is
  generic(tcq : time := 100 ps;
          tdc : time := 50 ps);
  port(d,clk,enable : in std_logic;
       q, qn : out std_logic);
end entity dff;
```

- rising edge triggered D flip-flop
- `tcq`: tempo di risposta
- `tdc`: tempo di setup

Esempio di VHDL strutturale - basic building block

```
architecture behav of dff is
begin
  one : process (clk)
  variable state: std_logic;
  begin
    -- check for rising clock edge and enable
    if ((clk = '1' and clk'last_value = '0')
        and enable = '1') then
       if (d'stable(tdc)) then -- setup requirement is met
         state := d;
       else -- setup not met
         state := 'X';
       end if;
    end if;
    q <= state after tcq;
    qn <= not state after tcq;
  end process one;
end architecture behav;
```

`rising_edge(clk)`

Esempio di VHDL strutturale - entity

Descrizione strutturale di un registro a 4 bit realizzato con FF D:

```
library ieee;
use ieee.std_logic_1164.all;
use work.all; -- optional
-- 4-bit register with no enable
entity reg4 is
  generic(tcq : time := 140 ps;
          tdc : time := 60 ps);
  port(d0,d1,d2,d3: in std_logic;
       clk: in std_logic;
       q0,q1,q2,q3: out std_logic);
end entity reg4;
```


Passi per incorporare un oggetto in un modello VHDL

- Si fa riferimento al modello entity/architecture
- Per incorporare un oggetto VHDL in un architettura di tipo strutturale, questo deve essere:
  - **dichiarato**: definendo un interfaccia locale
  - **instanziato**: connettendo i segnali locali (all'architettura) all'interfaccia locale
  - **legato**: selezionando una entity/architecture che lo realizza

Esempio di VHDL strutturale - architettura

```
architecture struct  of reg4 is
  component reg1 is
    port (d, clk : in std_logic;
          q : out std_logic);
  end component reg1;
  constant enabled : std_logic := '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;
```

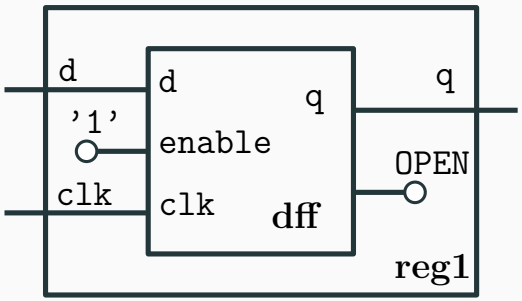
dichiarazione

legame (binding) e configurazione

instanza di componente

Nota

La configurazione inserita nel binding costruisce un wrapper intorno al modello VHDL **`dff`** che mette a disposizione l'interfaccia del componente **`reg1`**



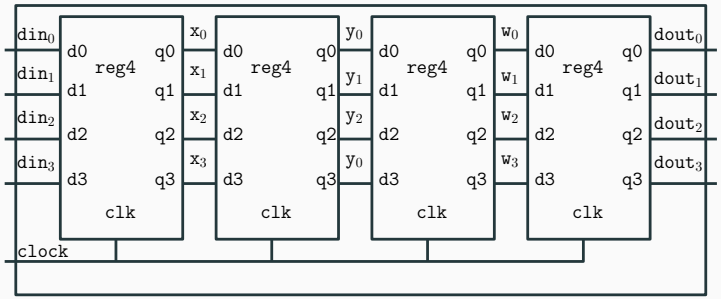
Configurazioni

- Le configurazioni sono utilizzate per legare un instanza di componente a un architettura e una entity
- Nell'esempio precedente si utilizza una configurazione di tipo **`entity-architecture`**
- Nel caso di descrizioni strutturali con piú di due livelli gerarchici é possibile utilizzare configurazioni al livello di architettura
- In pratica, si consente ad architetture a un livello piú alto che instanziano quella considerata di determinare quale configurazione questa debba utilizzare per i suoi componenti
- In questo modo, se si vogliono ad esempio cambiare i gate e i flip-flop di una libreria tecnologica, non bisogna iterare attraverso tutti i livelli del progetto

Esempio di configurazioni

- Si supponga che il componente **`reg4`** sia utilizzato in uno shift-register (parallelo - FIFO) a 4 stadi
- La rete ha 4 ingressi piú il clock e 4 uscite
- Il progetto consiste dei seguenti oggetti VHDL
  - decrizione comportamentale del flip-flop (slides precedenti)
  - descrizione strutturale del registro a 4 bit (senza configurazione)
  - descrizione strutturale dello shift-register
  - configurazione per il registro a 4 bit

Schema logico



VHDL - register with no binding

```
-- library and entity omitted

architecture struct_2  of reg4 is
component reg1 is
  port (d, clk : in std_logic;
        q : out std_logic);
end component reg1;
-- no binding
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 architecture struct_2;
```


VHDL - shift register entity I

```
library ieee;
use ieee.std_logic_1164.all;

entity sr_4x4 is
  port(din: in std_logic_vector(0 to 3);
       clock: in std_logic;
       dout: out std_logic_vector(0 to 3));
end entity sr_4x4;

architecture struct of sr_4x4 is
component register4 is
  port(d0,d1,d2,d3: in std_logic;
       clk: in std_logic;
       q0,q1,q2,q3: out std_logic);
end component register4;
for all : register4 use configuration work.reg4_conf;
signal x,y,w: std_logic_vector(0 to 3);
```

external configuration

VHDL - shift register architecture II

```
begin
 r0: register4
     port map(d0=>din(0),d1=>din(1),d2=>din(2),d3=>din(3),
              clk=>clock,
              q0=>x(0),q1=>x(1),q2=>x(2),q3=>x(3));
 r1: register4
     port map(d0=>x(0),d1=>x(1),d2=>x(2),d3=>x(3),
              clk=>clock,
              q0=>y(0),q1=>y(1),q2=>y(2),q3=>y(3));
 r2: register4
     port map(d0=>y(0),d1=>y(1),d2=>y(2),d3=>y(3),
              clk=>clock,
              q0=>w(0),q1=>w(1),q2=>w(2),q3=>w(3));
 r3: register4
     port map(d0=>w(0),d1=>w(1),d2=>w(2),d3=>w(3),
              clk=>clock,
              q0=>dout(0),q1=>dout(1),q2=>dout(2),q3=>dout(3))
end architecture sr_4x4;
```

VHDL - configuration

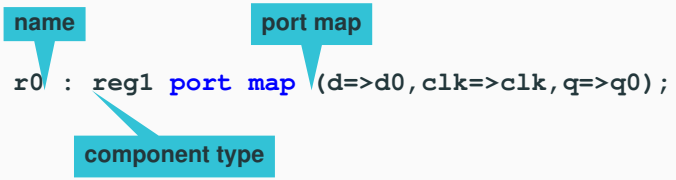
Se si vuole cambiare il componente al livello piú basso (flip-flop D) basta cambiare la configurazione

```
library ieee;
use ieee.std_logic_1164.all;
use work.all;

configuration reg4_conf of reg4 is
  constant enabled : std_logic := '1';
  for struct_2
    for all : reg1 use entity work.dff(behav)
       port map(d=>d,clk=>clk,enable=>enabled,q=>q,qn=>OPEN);
    end for;
  end for;
end reg4_conf;
```


Istruzione di instanziazione

- Genera una copia di un componente dichiarato e lo connette ai segnali dell'architettura
- Nel caso piú semplice, l'instanziazione ha 3 parti
  - **name** - identifica una specifica instanza del componente
  - **component type** - seleziona uno dei componenti dichiarati
  - **port map** - connette le porte del componente ai segnali dell'architettura



Generic map

- Permettono di specializzare il componente quando viene instanziato
- Esempio
  - nel modello di una porta logica i parametri **generic** possono essere utilizzati per definire dei valori di default di ritardo
  - in un circuito il ritardo varia da instanza a instanza dipendetemente dal fan-out di tali gate
- Tali ritardi possono essere mappati sulle varie instanze del gate tramite **generic map** che é simile alla **port map**

Esempio

```
....
entity nor2 is
  generic(trise: time:=100 ps;
          tfall: time:=80 ps);
  port(a,b: in std_logic;
       y: out std_logic);
end entity nor2;
          ....
          architecture test of test_entity is
          signal a,b,....,x,y,w: std_logic;
          ....
          begin
            gate0: nor_gate
                   generic map(trise=>110 ps,tfall=>90 ps)
                   port map(a=>x,b=>y,y=>w);
            gate1: nor_gate
                   generic map(trise=>200 ps, tfall=>160 ps)
                   port map(a=>w,b=>a,y=>b);
```

modello del gate

architettura che usa tale componente

parametri di ritardo specifici dell'istanza

Specifiche per referenziare componenti

- Si utilizzano nella parte dichiarativa dell'architettura
- Mettono a disposizione le informazioni che che servono per fare riferimento ai 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;
    ```


Indicazioni di binding

- Consentono di identificare il modello di un componente che viene instanziato
- Sono possibili diverse alternative
  - riferimento a entity/architecture

    ```
    for all: reg1 use work.dff(behav);
    ```

  - riferimento a una configurazione VHDL

    ```
    for all : register4 use configuration work.reg4_conf;
    ```

  - riferimenti diretti (VHDL93)

Riferimenti diretti

- Mettono a disposizione un semplice meccanismo per fare riferimento a oggetti definiti precedentemente
- Si perde la flessibilitá delle configurazioni e vanno rispettate alcune regole sull'ordine di analisi degli oggetti VHDL

```
-- entity in other slides
architecture struct_1  of reg4 is
  constant enabled : std_logic := '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 architecture struct_1;
```

port map per posizione

Actual e local

- Sono i termini utilizzati per indicare
  - le porte della entity o i segnali dichiarati nella architecture di una descrizione strutturale (**actual**)
  - le porte dei componenti (**local**)
- Regole di compatibilitá (statiche)
  - local e actual devono essere dello stesso tipo
  - local e actual devono avere modi compatibili

Sommario

- I diversi meccanismi di instanziazione possono luogo a un po di confusione
- L'instanziazione diretta é conveniente in piccoli progetti, quando si vuole realizzare un testbench o quando si usa un approccio bottom-up
- Il meccanismo entity-architecture conviene quando si realizzano progetti cooperativi (i componenti sono instanziabili anche senza un binding) o quando si vogliono riutilizzare oggetti esistenti con qualche modifica
- Le configurazioni sono da utilizzare in progetti complessi distribuiti su diversi livelli gerarchici

Istruzione `generate`

- Il VHDL mette a disposizione l'istruzione **`generate`** per creare strutture ripetitive quali RAM, adders
- Qualsiasi istruzione concorrente puó essere inclusa in un istruzione **`generate`**, comprese anche altre istruzioni dello stesso tipo
- Un utilizzo tipico é nelle descrizioni strutturali, dove dentro a un istruzione **`generate`** si possono instanziare componenti

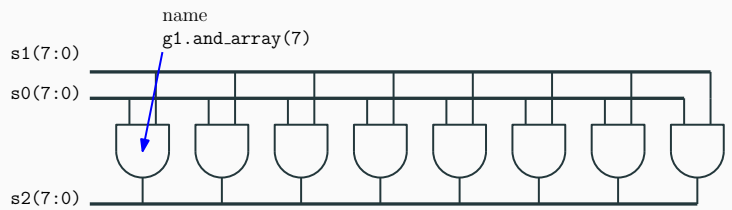
Sintassi

- Il costrutto **`for/generate`** serve per creare oggetti simili fra loro
- Il parametro di tale istruzione deve essere discreto e non é definito al di fuori dell'istruzione **`generate`**
  - facendo riferimento all'hardware, non é pensabile che il numero di gate in un integrato varii dinamicamente (almeno con le attuali tecnologie)
- Il ciclo non puó essere terminato anticipatamente

```
name : for n in 1 to 8 generate
       -- concurrent-statements
end generate name;
```


Esempio

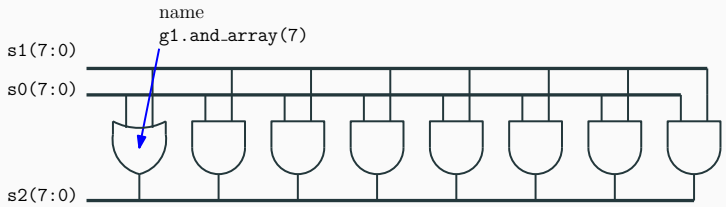
```
architecture test_generate of test_entity is
-- and_gate component declaration and binding
signal s0, s1, s2: std_logic_vector(7 downto 0);
begin
g1 : for n in 7 downto 0 generate
and_array : and_gate
generic map (100 ps, 150 ps)
port map (s0(n), s1(n), s2(n));
end generate g1;
end test_generate;
```



Utilizzo di `if` nell'istruzione `generate`

- L'istruzione **`generate`** non consente di gestire eccezioni
- Ad esempio un sommatore ripple-carry adder composto da n full-adder in cascata non sarebbe descrivibile perché il primo e l'ultimo full-adder sono diversi da quelli interni
- Il costrutto **`if`** consente di risolvere questo problema
  - si noti che non puó essere seguito da **`else - elsif`** (fino al VHDL-2008)

```
name : if (boolean expression) generate
       -- concurrent-statements
end generate name;
```



Esempio: array di gate con un or seguito da 7 and

```
architecture test_generate of test_entity
signal s0, s1, s2: std_logic_vector(7 downto 0);
begin
g1 : for n in 7 downto 0 generate
g2 : if (n = 7) generate
or1 : or_gate
  generic map (110 ps, 180 ps)
  port map (s0(n), s1(n), s2(n));
end generate g2;
g3 : if (n < 7) generate
and_array : and_gate
  generic map (100 ps, 160 ps)
  port map (s0(n), s1(n), s2(n));
end generate g3;
end generate g1;
end architecture test_generate;
```


Esempio precedente con `else/elsif` (VHDL-2008)

```
architecture test_generate of test_entity
signal s0, s1, s2: std_logic_vector(7 downto 0);
begin
g1 : for n in 7 downto 0 generate
g2 : if (n = 7) generate
or1 : or_gate
  generic map (110 ps, 180 ps)
  port map (s0(n), s1(n), s2(n));
elsif (n < 7) generate
and_array : and_gate
  generic map (100 ps, 160 ps)
  port map (s0(n), s1(n), s2(n));
else generate
end generate g2;
end generate g1;
end architecture test_generate;
```


Conclusioni

- Per quanto sia in teoria possibile avere un flusso automatico di progetto interamente top-down, l'utilizzo del VHDL strutturale é necessario per
  - comprendere i risultati della sintesi
  - scrivere i testbench durante la verifica di progetto
- Lo stile di progetto bottom-up é comunque molto seguito e reso necessario in applicazioni in cui
  - si vogliano riutilizzare moduli esistenti
  - gli strumenti di sintesi automatica si dimostrino inferiori alle conoscenze del progettista

Libreria di gate VHDL