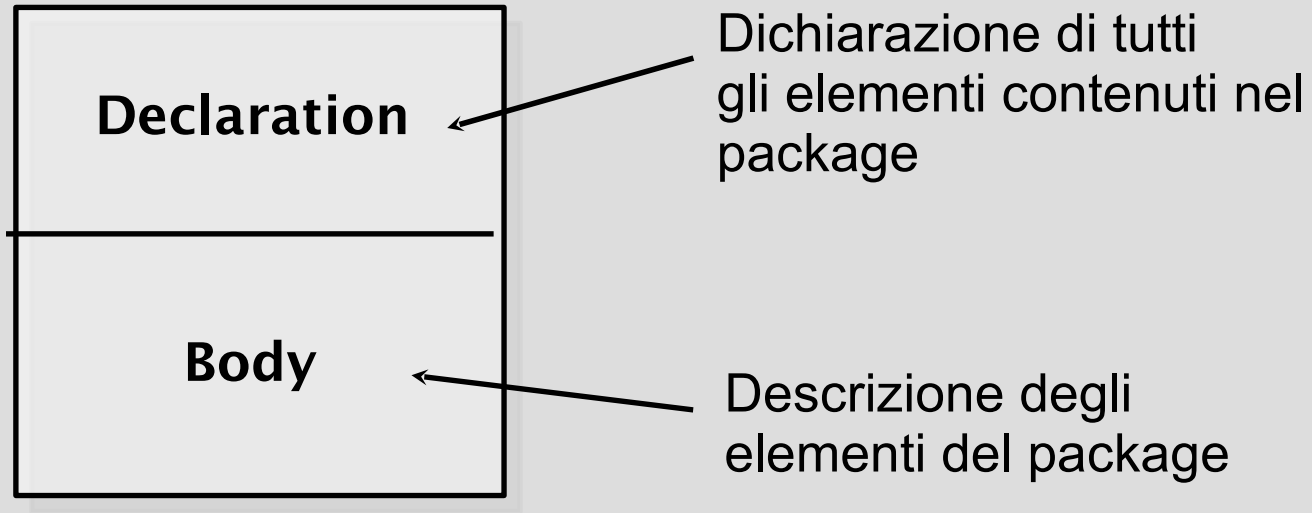
VHDL Packages

- I **Packages** incapsulano elementi che possono essere condivisi tra più entity
- Un **package** consiste di due parti:



Packages

- Nei `package` possiamo inserire:
  - Subprograms (functions and procedures)
  - Dati e dichiarazione di tipo:
    - User record definitions
    - User types and enumerated types
    - Constants
    - Files
    - Aliases
    - Attributes
  - Dichiarazione di `component`
- Entities ed Architectures non possono essere dichiarate
- Un package viene richiamato con la keyword ***use***

Package Declaration

```
PACKAGE resources IS

 -- user defined enumerated type
   TYPE level IS ('X', '0', '1', 'Z');

 -- type for vectors (buses)
   TYPE level_vector IS ARRAY (NATURAL RANGE <>) OF level;

 -- subtype used for delays
   SUBTYPE delay IS time;

 -- resolution function for level
   FUNCTION wired_x (input : level_vector) RETURN level;

 -- subtype of resolved values
   SUBTYPE level_resolved_x IS wired_x level;

 -- type for vectors of resolved values
   TYPE level_resolved_x_vector IS
      ARRAY (NATURAL RANGE <>) OF level_resolved_x;

 END resources;
```

Package Body

```
PACKAGE BODY resources IS
  -- resolution function
  FUNCTION wired_x (input : level_vector) RETURN level IS

    VARIABLE has_driver : BOOLEAN := FALSE;
    VARIABLE result     : level   := 'Z';
    BEGIN
      L1 : FOR i IN input'RANGE LOOP

        IF(input(i) /= 'Z') THEN
          IF(NOT has_driver) THEN
            has_driver := TRUE;
            result := input(i);
          ELSE                    -- has more than one driver
            result := 'X';
            EXIT L1;
          END IF;
        END IF;

      END LOOP L1;

      RETURN result;
  END wired_x;
END resources;
```

Type std_logic

```
PACKAGE std_logic_1164 IS

TYPE std_ulogic IS ( 'U', -- Uninitialized
                     'X', -- Forcing Unknown
                     '0', -- Forcing 0
                     '1', -- Forcing 1
                     'Z', -- High Impedance
                     'W', -- Weak Unknown
                     'L', -- Weak 0
                     'H', -- Weak 1
                     '-' -- Don't care
                   );
-------------------------------------------------------
-- unconstrained array of std_ulogic
-------------------------------------------------------
TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;
-------------------------------------------------------
-- resolution function
-------------------------------------------------------
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;
-------------------------------------------------------
-- *** industry standard logic type ***
-------------------------------------------------------
SUBTYPE std_logic IS resolved std_ulogic;
```

Type std_logic

```
PACKAGE BODY std_logic_1164 IS
CONSTANT resolution_table : stdlogic_table := (
-- ---------------------------------------------------
--| U   X    0    1    Z    W    L    H    -        |   |
-- ---------------------------------------------------
( 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' ), -- | U |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | X |
( 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' ), -- | 0 |
( 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' ), -- | 1 |
( 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' ), -- | Z |
( 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' ), -- | W |
( 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' ), -- | L |
( 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' ), -- | H |
( 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' ), -- | - |
);
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
VARIABLE result : std_ulogic := 'Z'; -- weakest state
BEGIN
-- the test for a single driver is essential
      IF (s'LENGTH = 1) THEN RETURN s(s'LOW);
      ELSE
          FOR i IN s'RANGE LOOP
          result := resolution_table(result, s(i));
          END LOOP;
      END IF;
      RETURN result;
END resolved;
```

Type std_logic

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY test IS
PORT ( i0, i1, i2, i3: IN std_logic;
       o0,01,02,o3,o4      : OUT std_logic);
END test;

ARCHITECTURE a OF test IS
BEGIN
o0 <= '0' WHEN i0 = ‘0’ AND i1 = ‘0’ ELSE
      '1' ;
o1 <= '-' WHEN i0 = ‘0’ AND i1 = ‘0’ ELSE
      '1' WHEN i0 = ‘1’ AND i1 = ‘1’ ELSE
      '0';

o2 <= '0' WHEN i0 = ‘0’ AND i1 = ‘0’ ELSE
      '1';

o2 <=  '0' WHEN i0 = ‘0’ AND i1 = ‘0’ ELSE
      '1' WHEN i0 = ‘1’ AND i1 = ‘1’ ELSE
      '0';

o3 <= '0' WHEN i0 = ‘0’ AND i1 = ‘0’;

o4 <= 'Z' WHEN i0 = ‘0’ AND i1 = ‘0’ ELSE   '1';

o4 <= '0' WHEN i0 = ‘0’ AND i1 = ‘0’ ELSE   'Z';
END a;
```

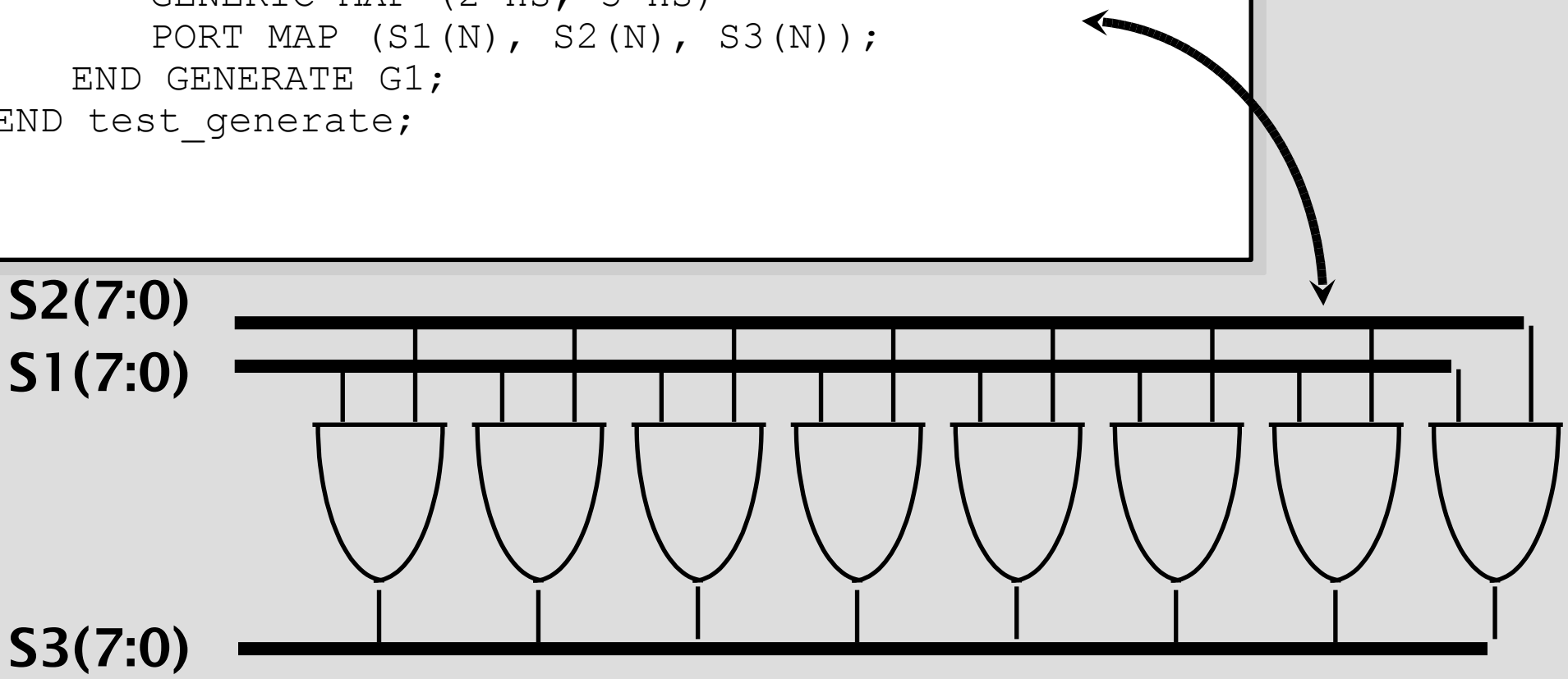
Generate Statement

- Il VHDL permette di creare strutture ripetitive con la keyword GENERATE
  - (e.g. RAMs, adders)
- Uno statement GENERATE statement può essere usato in ogni assegnazione concorrente

```
name : FOR parameter_specification GENERATE
       [Declaration_statements
BEGIN]
       {concurrent_statements}
END GENERATE [name];
```

Esempio FOR

```
-- 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;
```



Esempio IF

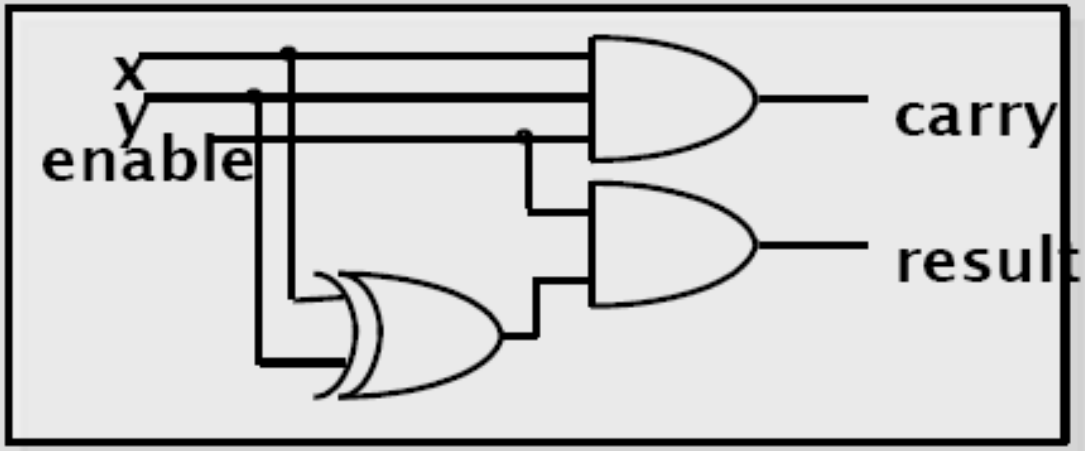
```
name : IF boolean_expression GENERATE
       [Declaration_statements
BEGIN]
       {concurrent_statements}
END GENERATE [name];
```

```
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;
```

Configurations

Un entity può avere più descrizioni (architecture).

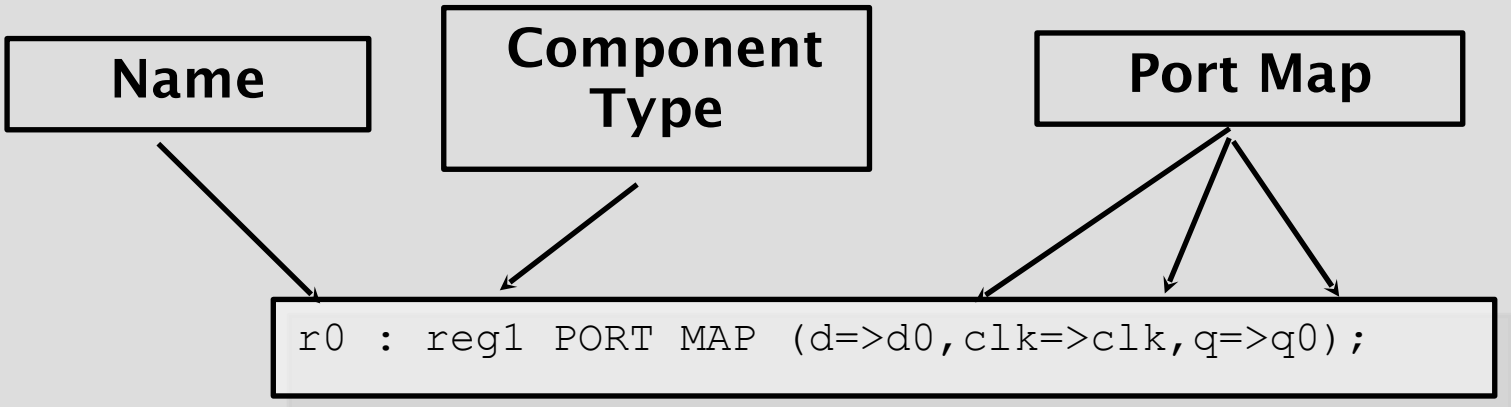
Come faccio a scegliere quale architecture simulare? La **configuration** permette in fase di instanziazione di un componente di specificare quale architecture usare




```
FOR ALL : and2 USE ENTITY gate_lib.and2_Nty(and2_a);
    FOR ALL : and3 USE ENTITY gate_lib.and3_Nty(and3_a);
    FOR ALL : xor2 USE ENTITY gate_lib.xor2_Nty(xor2_a);
```

Instanziazione

- Per istasnzaiare un componente servono:
  - Name: identifica un istanza *unica* del componente
  - Component type: seleziona uno dei componenti dichiarati
  - Port map: per collegarlo ai segnali dell'architettura



Generic Map

- Permette di "*personalizzare*" l' instanziazione del componente
  - Nell'Entity sono previsti i valori di default
- E' simile al PORT MAP

```
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;
```

Binding

- Associa un componente istanziato ad unentity/architecture
  - Single component

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

  - Multiple components

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

  - All components

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

  - Other components

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

Binding

- Si possono utilizzare due meccanismi:
  - VHDL entity/architecture design object

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

  - VHDL configuration

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

- Puo includere il PORT MAP e/o il GENERIC MAP

Esempio

```
USE work.resources.all;

ARCHITECTURE struct_3 OF reg4 IS
  COMPONENT reg1 IS
    PORT (d, clk : IN level; q : OUT level);
  END COMPONENT reg1;
   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;
```