Subprograms

I subprograms sono strumenti del VHDL molto utili per eseguire operazioni frequenti o comuni a più parti del sistema e per aumentare la leggibilità del codice. Si dividono in:

procedures

- Concorrenti o sequenziali
- Possono restituire più di un argomento
- Possono avere parametri come input, output oppure inout
- Si presentano come istruzioni separate in un' architettura o in un processo

functions

- Concorrenti o sequenziali
- Possono restituire un solo argomento
- Possono avere parametri solo come input
- Generalmente sono utilizzate in istruzioni di assegnazione

Functions

```
FUNCTION add_bits (a, b : IN BIT) RETURN BIT IS
BEGIN
    RETURN (a XOR b);
END add_bits;
```

```
FUNCTION add_bits2 (a, b : IN BIT) RETURN BIT IS
 VARIABLE result : BIT;  -- variabile locale
    BEGIN
        result := (a XOR b);
        RETURN result;
END add_bits2;
```

Le 2 funzioni sono equivalenti

Le 2 funzioni non possono variare i parametri ad esse “passati”

E' necessario utilizzare la parola `RESULT`

Functions (cont.)

```
ARCHITECTURE behavior OF adder IS
BEGIN
    PROCESS (enable, x, y)
    BEGIN
    IF (enable = '1') THEN
        result <=   add_bits(x, y);
        carry  <= x AND y;
    ELSE
        carry, result <= '0';
    END PROCESS;
END behavior;
```

I parametri sono assegnati nell'ordine in cui compaiono nella dichiarazione

```
FUNCTION add_bits

(a, b : IN BIT)
```

Le funzioni possono essere chiamate in altre istruzioni

Functions (cont.)

```
USE LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
PACKAGE num_types IS
 TYPE log8 IS ARRAY(0 TO 7) OF std_logic;
END num_types;

USE LIBRARY IEEE; USE IEEE.std_logic_1164.ALL;
USE WORK.num_types.ALL;
ENTITY convert IS
 PORT(I1 : IN log8;
    O1 : OUT INTEGER);
END convert;

ARCHITECTURE behave OF convert IS
 FUNCTION vector_to_int(S : log8)
  RETURN INTEGER is
  VARIABLE result : INTEGER := 0;
 BEGIN
 FOR i IN 0 TO 7 LOOP
   result := result * 2;

   IF S(i) = '1' THEN
     result := result + 1;
   END IF;
 END LOOP;
 RETURN result;
 END vector_to_int;

BEGIN
 O1 <= vector_to_int(I1);
END behave;
```

Funzione dichiarata nell'`architecture`:disponibile in tutta l'architettura

Non è specificato cosa sia "S": è implicitamente assunta come una costante

Regione di dichiarazione simile a quella dei processi: si possono dichiarare variabili, costanti e tipi ma NON segnali

N.B.: nella descrizione della funzione, non è possibile effettuare alcun tipo di assegnazione ai parametri della funzione stessa che vengono considerati come INGRESSI

Functions (cont.)

```
 USE LIBRARY IEEE;
 USE IEEE.std_logic_1164.ALL;
 ENTITY dff IS
  PORT(d, clk : IN std_logic;
       q : OUT std_logic);

  FUNCTION rising_edge(SIGNAL S : std_logic)
   RETURN BOOLEAN IS
  BEGIN
  IF (S'EVENT) AND (S = '1') AND
       (S'LAST_VALUE = '0') THEN
    RETURN TRUE;
  ELSE
    RETURN FALSE;
  END IF;
 END rising_edge;
END dff;

ARCHITECTURE behave OF dff IS
BEGIN
 PROCESS( clk)
 BEGIN
  IF rising_edge(clk) THEN
   q <= d;
  END IF;
 END PROCESS;
END behave;
```

Funzione dichiarata nell' `entity`:disponibile in tutte le architectures associate all' `entity`

"S" è dichiarato come segnale: ad esso si possono applicare gli `attribute` (eccetto `'STABLE, 'QUIET, 'TRANSACTION,'DELAYED`)

Il compito più comune delle funzioni è quello di restituire un valore in una espressione. Vi sono altri 2 modi di impiegare una funzione: come *funzione di conversione* e come *funzione di risoluzione*

Conversion Functions

Sono utilizzate per convertire un oggetto di un tipo in un oggetto di un altro tipo.

Esempio

```
TYPE fourval IS (X,L,H,Z)
```

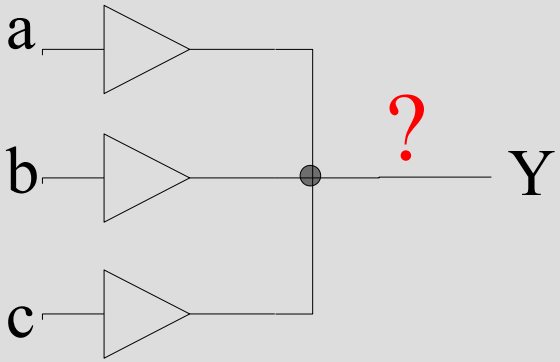


```
TYPE fourval IS ('X','0','1','Z')
```

```
FUNCTION convert4val(S : fourval) RETURN fourvalue IS
BEGIN
 CASE S IS
  WHEN X =>
   RETURN 'X';
  WHEN L =>
   RETURN '0';
  WHEN H =>
   RETURN '1';
  WHEN Z =>
   RETURN 'Z';
 END CASE;
END convert4val;
```

Resolution Functions

Vengono utilizzate per decidere il valore del segnale quando questo sia pilotato da driver multipli (il VHDL non ammette driver multipli senza resolution function)



Esempio: definito il tipo

```
TYPE fourval IS (X,L,H,Z)
```

ove

Z → alta impedenza (debole)
H → '1' logico (“sovrascrivibile” da 'X')
L → '0' logico (“sovrascrivibile” da 'X')
X → condizione non nota (non sovrascrivibile)

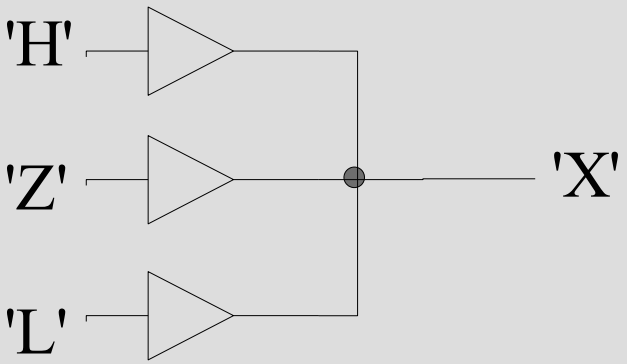
è possibile scrivere una *resolution function* per il tipo `fourval`

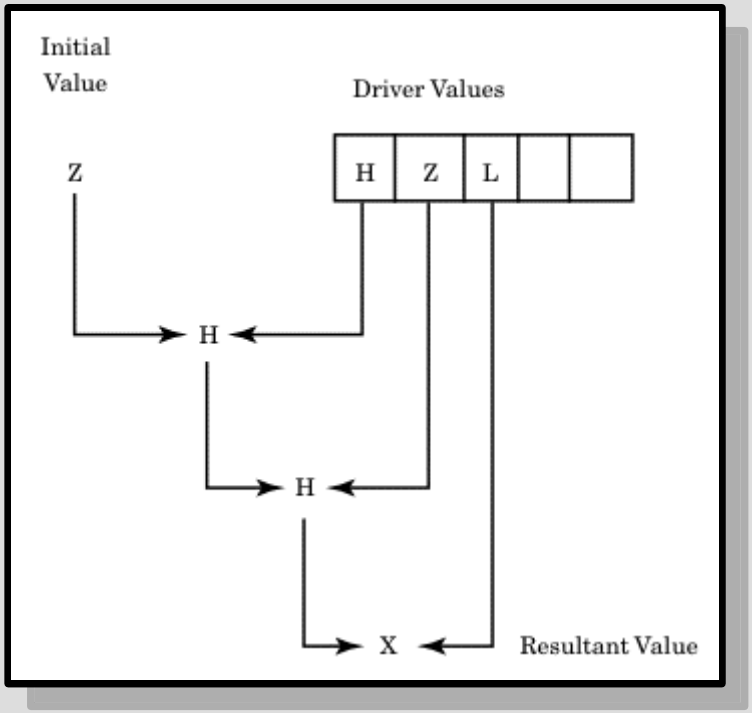
Resolution Functions (cont)

Una volta stabiliti i livelli di “forza” di tutti i possibili valori che il segnale può assumere (es.: X, L, H, Z), è possibile costruire una tabella della verità per 2 valori di ingresso:

	Z	L	H	X
Z	Z	L	H	X
L	L	L	X	X
H	H	X	H	X
X	X	X	X	X

La tabella è utilizzabile anche nel caso di N>2 ingressi in quanto si considerano valide le proprietà associativa e commutativa





Resolution Functions (cont)

```
PACKAGE fourpack IS
  TYPE fourval IS (X, L, H, Z);
  TYPE fourval_vector IS ARRAY (natural RANGE <> ) OF
      fourval;

  FUNCTION resolve( s: fourval_vector) RETURN fourval;
END fourpack;

PACKAGE BODY fourpack IS
  FUNCTION resolve( s: fourval_vector) RETURN fourval IS
   VARIABLE result : fourval := Z;
  BEGIN
   FOR i IN s'RANGE LOOP
    CASE result IS
     WHEN Z =>
       CASE s(i) IS
         WHEN H =>
           result := H;
         WHEN L =>
           result := L;
         WHEN X =>
           result := X;
         WHEN OTHERS =>
           NULL;
       END CASE;
```

```
     WHEN L =>
       CASE s(i) IS
         WHEN H =>
           result := X;
         WHEN X =>
           result := X;
         WHEN OTHERS =>
           NULL;
       END CASE;

     WHEN H =>
       CASE s(i) IS
         WHEN L =>
           result := X;
         WHEN X =>
           result := X;
         WHEN OTHERS =>

           NULL;
       END CASE;

       WHEN X =>
         result := X;

    END CASE;
   END LOOP;
   RETURN result;
 END resolve;
END fourpack;
```

Procedure

- Le procedure possono avere parametri di `IN, OUT e INOUT`
- La chiamata della procedura è essa stessa una istruzione
- Possono modificare i parametri “passati”
- Non richiedono il `RETURN`
- La principale caratteristica per la quale vengono utilizzate è la capacità di restituire più valori

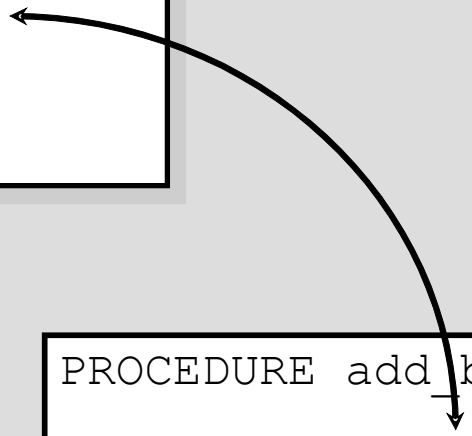
```
PROCEDURE add_bits3 (SIGNAL a, b, en : IN BIT;
            SIGNAL temp_result, temp_carry : OUT BIT) IS

BEGIN
       temp_result <= (a XOR b) AND en;
       temp_carry <= a AND b AND en;

END add_bits3;
```

Procedure (cont.)

```
ARCHITECTURE behavior OF adder IS
BEGIN
  PROCESS (enable, x, y)
  BEGIN
  add_bits3(x, y, enable,
            result, carry);
  END PROCESS;
END behavior;
```



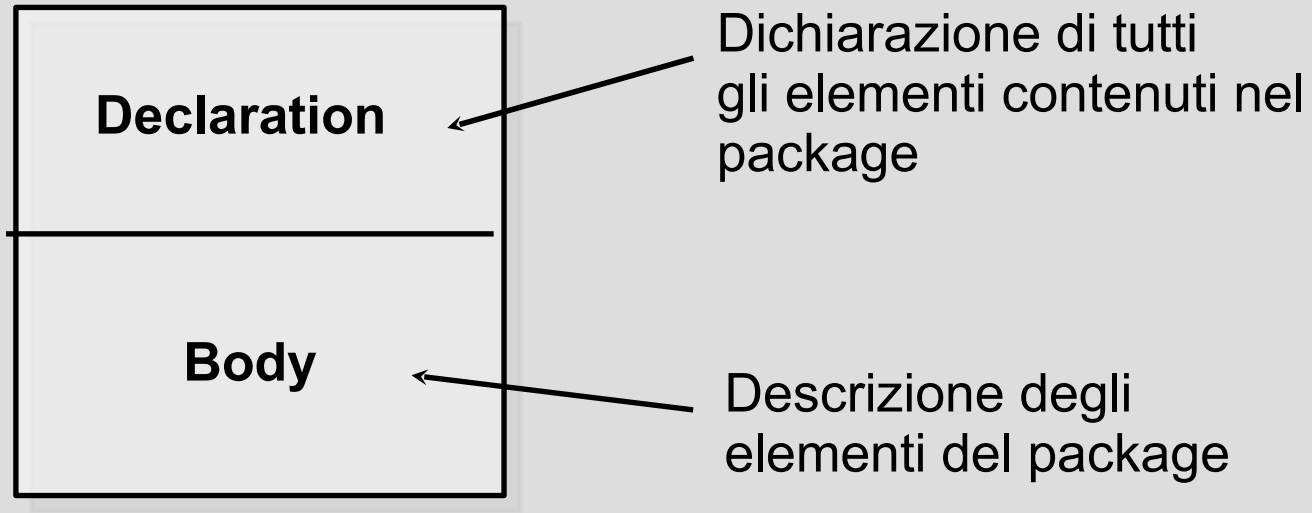
```
PROCEDURE add_bits3

(SIGNAL a, b, en : IN BIT;
 SIGNAL temp_result,
    temp_carry : OUT BIT)
```

I parametri devono essere compatibili in termini di data-flow (direzione) e tipo

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;
```