Rappresentazione dei segnali in VHDL

Approfondimento del corso di

Linguaggi di descrizione dell'hardware

Tipi di segnali (logici) predefiniti

- Tipo bit (definito nel package standard)

```
type Bit is ('0', '1');

type Bit_vector is array (Natural range <>) of Bit;
```

- Limitazioni nel descrivere numerosi comportamenti dei sistemi digitali

Estensioni nel package `ieee.std_logic`

- Significativa estensione delle condizioni modellabili
- Overloading degli operatori VHDL definiti per il tipo dato `bit`
- Funzioni di conversione e di utilità

Tipo di dato: `std_ulogic`

```
------------------------------------------------------------------
-- logic state system (unresolved)
------------------------------------------------------------------
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 for use with the resolution function
------------------------------------------------------------------
TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF std_ulogic;
```

Tipo di dato: `std_logic` (resolved)

Il tipo di segnale std_ulogic non consente di modellare segnali pilotati da più drivers quali ad esempio quelli dei bus. Per questo si usa una resolution function che modella un bus del tipo tristate (non wired-AND/OR).

```
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;
-------------------------------------------------------------------
-- *** industry standard logic type ***
-------------------------------------------------------------------
SUBTYPE std_logic IS resolved std_ulogic;
-------------------------------------------------------------------
-- unconstrained array of std_logic for use in declaring signal arrays
-------------------------------------------------------------------
TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;
```

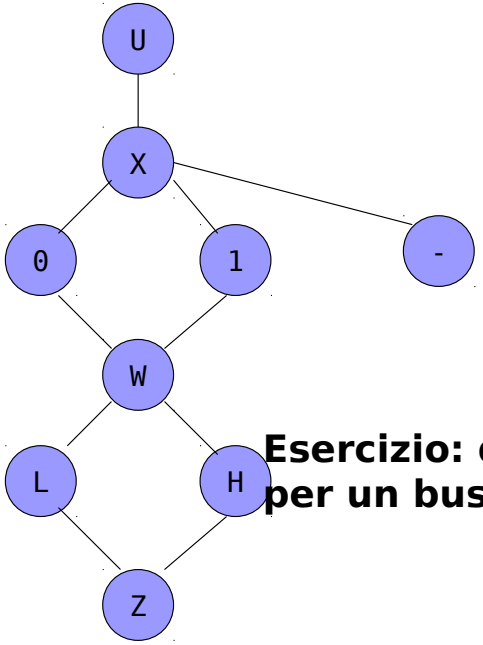
Tipo di dato: `std_logic` (resolved)

- Dato un insieme di driver che pilotano valori diversi su un unico segnale
- La resolution function determina il valore di tale segnale
- Questo viene fatto a partire da una tabella, che dati i valori di due driver determina il valore risolto
- Tale tabella determina un ordinamento parziale fra i segnali (multiple valued algebra)

Tipo di dato: `std_logic` (tabella di risoluzione)

```
--------------------------------------------------------------------
-- local types
--------------------------------------------------------------------
TYPE stdlogic_1d IS ARRAY (std_ulogic) OF std_ulogic;
TYPE stdlogic_table IS ARRAY(std_ulogic, std_ulogic) OF std_ulogic;
--------------------------------------------------------------------
-- resolution function
--------------------------------------------------------------------
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' )  -- | - |       );
```

Diagramma di Hasse di `std_logic`



Esercizio: descrivere un dri
per un bus wired-and

Resolution function di `std_logic`

```
FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic IS
       VARIABLE result : std_ulogic := 'Z';  -- weakest state default
BEGIN
       -- the test for a single driver is essential otherwise the
       -- loop would return 'X' for a single driver of '-' and that
       -- would conflict with the value of a single driver unresolved
       -- signal.
       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;
```

Operatori logici

- Overloading degli operatori logici AND, OR, NOT …. predefiniti sul tipo di dato `bit` e `bit_vector`
- Funzioni che utilizzano tabelle che definiscono l’operatore binario

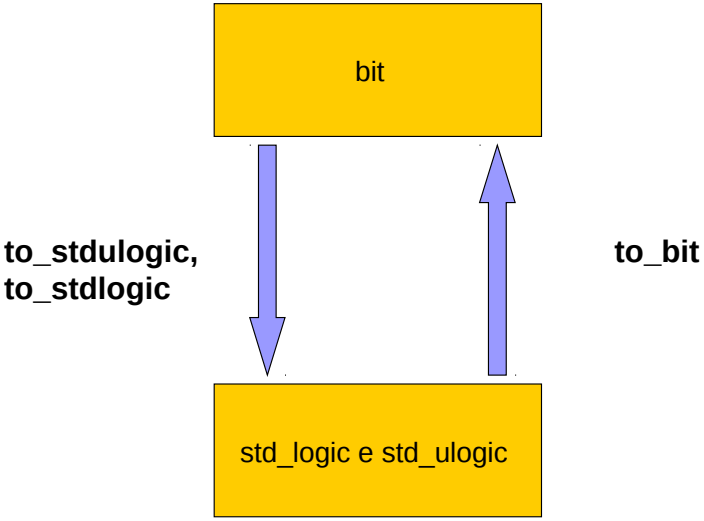
Operatori logici (OR table)

```
-- truth table for "or" function
CONSTANT or_table : stdlogic_table := (
-- ----------------------------------------------------
-- | U X 0 1 Z W L H - | |
-- ----------------------------------------------------
( 'U', 'U', 'U', '1', 'U', 'U', 'U', '1', 'U' ), -- | U |
( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ), -- | X |
( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | 0 |
( '1', '1', '1', '1', '1', '1', '1', '1', '1' ), -- | 1 |
( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ), -- | Z |
( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' ), -- | W |
( 'U', 'X', '0', '1', 'X', 'X', '0', '1', 'X' ), -- | L |
( '1', '1', '1', '1', '1', '1', '1', '1', '1' ), -- | H |
( 'U', 'X', 'X', '1', 'X', 'X', 'X', '1', 'X' )  -- | - |);
```

Operatori logici (OR table)

```
---------------------------------------------------------------------
-- or
---------------------------------------------------------------------
FUNCTION "or" ( l,r : std_logic_vector ) RETURN std_logic_vector IS
  ALIAS lv : std_logic_vector ( 1 TO l'LENGTH ) IS l;
  ALIAS rv : std_logic_vector ( 1 TO r'LENGTH ) IS r;
  VARIABLE result : std_logic_vector ( 1 TO l'LENGTH );
BEGIN
  IF ( l'LENGTH /= r'LENGTH ) THEN
    ASSERT FALSE
    REPORT "arguments of overloaded 'or' operator are not of the same length"
    SEVERITY FAILURE;
  ELSE
    FOR i IN result'RANGE LOOP
      result(i) := or_table (lv(i), rv(i));
    END LOOP;
  END IF;
RETURN result;
END “or”;
```

Funzioni di conversione



Funzioni di utilità

- Analisi dei fronti:
  - `rising_edge`
  - `falling_edge`
- Rivelazione di valori logici non definiti (‘U’,’X’,’W’,’Z’,’-’)
  - `is_x`

Funzioni di utilità

```
-------------------------------------------------------------------------
-- edge detection
 -------------------------------------------------------------------------
FUNCTION rising_edge (SIGNAL s : std_ulogic) RETURN BOOLEAN IS
BEGIN
  RETURN (s'EVENT AND (To_X01(s) = '1') AND (To_X01(s'LAST_VALUE) = '0'));
END;
-------------------------------------------------------------------------
-- object contains an unknown
-------------------------------------------------------------------------
FUNCTION Is_X ( s : std_logic_vector ) RETURN BOOLEAN IS
BEGIN
  FOR i IN s'RANGE LOOP
    CASE s(i) IS
      WHEN 'U' | 'X' | 'Z' | 'W' | '-' => RETURN TRUE;
      WHEN OTHERS => NULL;
    END CASE;
  END LOOP;
RETURN FALSE;
END;
```

Rappresentazione di segnali aritmetici

- Il tipo di dato `std_logic` da luogo a problemi nel caso di circuiti aritmetici
- Ad esempio due `std_logic_vector` non sono sommabili
- Scrivere una funzione opportuna per la somma?
- Utilizzo di interi?
- Il VHDL preferisce invece mettere a disposizione tipi di dato specifici per le operazioni aritmetiche

Libreria: `ieee.numeric_std;`

- Una volta che si sia attribuita una semantica di tipo numerico a un vettore logico bisogna individuare il tipo di rappresentazione dei dati numerici
- cosa rappresenta “1010”?
- tipi di dato unsigned
  - `signal a,b:unsigned(n-1 downto 0);`
- tipi di dato signed
  - `signal a,b:signed(n-1 downto 0);`
- Questi dati mantengono una struttura di tipo array
- Vantaggi dal punto di vista della sintesi

Esempio di somma di due unsigned

- con perdita del bit piu’ significativo del risultato

```
signal a,b,c: unsigned(7 donwto 0);
.......
a<=“01101110”; -- 110
b<=“11011010”; -- 218
c<=a+b; -- “01001000” 72
```

- senza perdita del bit piu’ significativo del risultato

```
signal a,b: unsigned(7 donwto 0);
signal d: unsigned(8 downto 0);
.......
a<=“01101110”; -- 110
b<=“11011010”; -- 218
d<=(“0”&a)+(“0”&b); -- “101001000”
328
```

Esempio di somma di due signed

- senza overflow

```
signal a,b,c: signed(7 donwto 0);
.......
a<=“01101110”; -- 110
b<=“11011010”; -- -38
c<=a+b; -- “01001000” 72
```

- con overflow (e soluzione del problema)

```
signal a,b,c: signed(7 donwto 0);
.......
a<=“10001110”; -- -114
b<=“11011010”; -- -38
c<=“01101000”; -- 104
d<=(“1”&a)+(“1”&b); -- “101101000”
-152
```

Funzioni di conversione

- std_logic_vector => unsigned : `unsigned()`
- std_logic_vector => signed : `signed()`
- unsigned/signed => std_logic_vector: `std_logic_vector()`