VHDL behavioral

M. Favalli

DE: Department of Engineering Ferrara

Obbiettivi

- Comprensione dei costrutti behavioral dal punto di vista della sintesi e della simulazione
- Espandere la conoscenza della sintassi del VHDL
  - funzioni e procedure
- Fornire alcuni esempi di VHDL comportamentale
  - bus resolution function, `std_logic`, file

Sommario

Modelli comportamentali

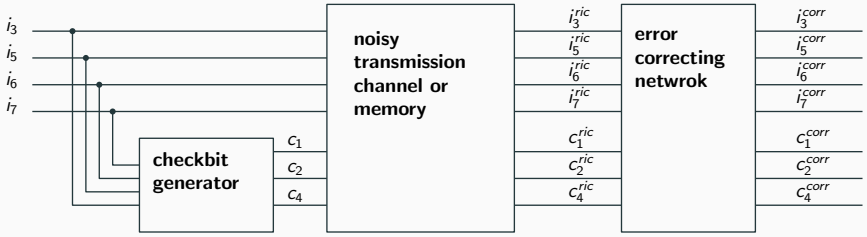
Behavioral modeling in VHDL

- Livelli di astrazione nei modelli VHDL
  - strutturale
  - dataflow
  - behavioral
- Nei modelli comportamentali interessano quasi esclusivamente gli aspetti funzionali (specifiche)
- Il timing é spesso opzionale, a meno che non si stia caratterizzando dal punto di vista dei ritardi una libreria di componenti
- Nello sviluppo di modelli comportamentali puó essere utile utilizzare tecniche di software engineering
  - progetto strutturato
  - rifinimento iterattivo
  - tipi di dato astratti

Esempio di VHDL comportamentale

Rete per la correzione di errori singoli tramite il **codice di Hamming**

1. Introduzione al codice di Hamming
2. Codificatore
3. Rete di correzione
4. Architetture comportamentali



Esempio di VHDL comportamentale: codice di Hamming

- Il codice di Hamming consente di correggere un errore in una parola mediante l'utilizzo di ridondanza di informazione
- Alla parola da trasmettere/memorizzare (I bit) viene aggiunto un numero opportuno di bit (C, $2^C \geq I + C + 1$)
- Tali bit sono calcolati come bit di paritá su sottoinsiemi indipendenti di I
- Nel caso in cui la parola di $I + C$ bit ricevuta/letta contenga un errore singolo, questo puó essere corretto
  - confrontando i checkbit ricevuti con quelli ricalcolati sui bit di informazione ricevuti e calcolando le sindromi di errore
  - tali sindromi di errore codificano in binario l'assenza di errori (0) o la posizione dell'errore
  - l'errore viene corretto complementando il bit errato

Esempio di VHDL comportamentale: Hamming checkbit generator

- Specifiche del codificatore (o checkbit generator)
  - $I = 4$, $C = 3$, organizzazione della parola $c_1 c_2 i_3 c_4 i_5 i_6 i_7$
  - equazioni dei checkbit:
    $c_1 = i_3 \oplus i_5 \oplus i_7$
    $c_2 = i_3 \oplus i_6 \oplus i_7$
    $c_4 = i_5 \oplus i_6 \oplus i_7$
- Questo componente rimane come esercizio, qui focalizzaimo l'attenzione sul correttore

Esempio di VHDL comportamentale: Hamming error corrector

- Specifiche del correttore
  - bit di informazione ricevuti $i_3^{ric}, i_5^{ric}, i_6^{ric}, i_7^{ric}$, bit di check ricevuti $c_1^{ric}, c_2^{ric}, c_4^{ric}$
  - sindromi di errore
    $s_1 = c_1^{ric} \oplus i_3^{ric} \oplus i_5^{ric} \oplus i_7^{ric}$
    $s_2 = c_2^{ric} \oplus i_3^{ric} \oplus i_6^{ric} \oplus i_7^{ric}$
    $s_4 = c_4^{ric} \oplus i_5^{ric} \oplus i_6^{ric} \oplus i_7^{ric}$
  - bit di errore: $o_i,\ i = 0..7$ dove $o_i = 1$ se $i = (s_4 s_2 s_1)_{2\to 10}$. altrimenti 0
  - uscite corrette
    $i_i^{corr} = i_i^{ric} \oplus o_i\ ,\ i = 3,5,6,7$
    $c_i^{corr} = c_i^{ric} \oplus o_i\ ,\ i = 1,2,4$
- Il calcolo di dei segnali o_i viene tipicamente fatto tramite un decoder

Esempio di VHDL comportamentale: esempio di correzione di errore

- Parola trasmessa $i_3 i_5 i_6 i_7 = 1001$, $s_4 s_2 s_1 = 100$
- Parola ricevuta $i_3^{ric} iric_5 i_6^{ric} i_7^{ric} = 1000$, $c_4^{ric} c_2^{ric} c_1^{ric} = 100$
  - si osserva che c'é un errore su i_7
- Sindromi di errore $s_4 s_2 s_1 = 111 \Rightarrow o_7 = 1$
- Uscite della rete di correzione $i_3^{corr} i_5^{corr} i_6^{corr} i_7^{corr} = 1001$, $c_1^{corr} c_2^{corr} c_4^{corr} = 100$

Esempio di VHDL comportamentale: entity del correttore

- per questa entity vedremo due architetture
  - la prima é piuttosto astratta, simulabile, ma potrebbe dare luogo a problemi di sintesi
  - la seconda é simulabile e aiuta il tool di sintesi a inferire la presenza di un decoder

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

entity corrector is
  -- mapping w1 w2 s3 w4 w5 w6 w7
  --         c1 c2 i3 c4 i5 i6 i7 (received)
  port(w: in std_logic_vector(1 to 7);
       wcorr: out std_logic_vector(1 to 7));
end entity corrector;
```


Esempio di VHDL comportamentale: architettura con calcolo dell'indice del bit errato

```
architecture behav_0 of corrector is
begin
  process(w)
  -- mapping s(2) s(1) s(0) = s4   s2   s1
  variable s: std_logic_vector(2 downto 0);
  variable tmp: std_logic_vector(1 to 7);
  variable index: integer; -- index of error
  begin
    tmp:=w;
    s(0):=w(1) xor w(3) xor w(5) xor w(7);
    s(1):=w(2) xor w(3) xor w(6) xor w(7);
    s(2):=w(4) xor w(5) xor w(6) xor w(7);
```

Esempio di VHDL comportamentale: architettura con calcolo dell'indice del bit errato

```
      index:=0;
      if (s(0)='1') then
        index:=index+1;
      end if;
      if (s(1)='1') then
        index:=index+2;
      end if;
      if (s(2)='1') then
        index:=index+4;
      end if;
      tmp(index):=not tmp(index);
      assert (index=0) report "corrected error" severity note;
      wcorr <= tmp;
    end process;
  end architecture behav_0;
```

conversione sostituibile con una funzione

Esempio di VHDL comportamentale: architettura con when-else

```
architecture behav_1 of corrector is
begin
  process(w)
  -- mapping s(2) s(1) s(0)
  --         s4   s2   s1
  variable s: std_logic_vector(2 downto 0);
  variable o: std_logic_vector(0 to 7);
  -- o(0)='1' denotes absence of errors
  begin
    s(0):=w(1) xor w(3) xor w(5) xor w(7);
    s(1):=w(2) xor w(3) xor w(6) xor w(7);
    s(2):=w(4) xor w(5) xor w(6) xor w(7);
```


Esempio di VHDL comportamentale: architettura con with-else

```
    o:="10000000" when s="000" else
       "01000000" when s="001" else
       "00100000" when s="010" else
       "00010000" when s="011" else
       "00001000" when s="100" else
       "00000100" when s="101" else
       "00000010" when s="110" else
       "00000001" when s="111" else
       "XXXXXXXX";
     assert (o(0)='1') report "corrected error" severity note;
     wcorr <= w xor o(1 to 7);
  end process;
end architecture behav_1;
```


Processi VHDL

- I processi VHDL visti fino ad ora sono eseguiti quando cambiano i segnali nella sensitivity list
- Un processo VHDL puó anche sospendere la sua esecuzione tramite l'istruzione **wait**

```
architecture behavioral of clock_component is
begin
  process
  variable periodic: bit := '1';
  begin
    if en = '1' then
      periodic := not periodic;
    end if;
    ck <= periodic;
    wait for 1 ns;
  end process;
end architecture behavioral;
```

Sintassi

- I processi possono ricevere un etichetta utile quando nella stessa architettura sono dichiarati piú processi

```
[process_label :] process [(sensitivity_list)]
 [process_declarations]
begin
  process_statements;
end process [process_label];
```


Segmentazione dei processi

Un architettura comportamentale puó essere divisa in piú processi per motivi di leggibilitá o per fornire direttive alla sintesi per implementare processi diversi in moduli diversi

```
architecture two_segments of half_adder is
begin
  sum: process(a,b,cin)
       begin
         sum <= a xor b xor cin;
       end process sum;
  carry: process(a,b,cin)
         begiin
           cout <= (a and b) or (a and cin)
                   or (b and cin);
         end process carry;
end architecture two_segments;
```


Istruzioni VHDL sequenziali

- Assegnamenti a segnali o variabili
- Controllo di flusso
  - **`if <condition> then <statements> [elsif <condition>] [else <statements>] end if;`**
  - **`for <range> loop <statements> end loop;`**
  - **`while <condition> loop <statements> end loop;`**
  - **`case <condition> is when <value> => <statements> {when <value> => <statements>} [when others => <statements>] end case;`**
  - **`wait [on <signal>] [until <expression>] [for <time>];`**
  - **`assert <condition> [report <string>] [severity <level>];`**

Istruzione `case`

- Diversamente dalle istruzioni concorrenti **`when-else`** e **`with-select`**, l'istruzione **`case`** é sequenziale e va usata solo nei processi
  - ogni gruppo di istruzioni (**`<statements>`**) puó contenere una qualsiasi parte di codice sequenziale
- Sostituisce in maniera compatta un **`if-then-else`**
- Esempio
  - si vuole aggiungere al correttore del codice di hamming una statistica sugli errori

Esempio di VHDL comportamentale: architettura con case-when

```
architecture behav_stats of corrector is
begin
  process(w)
  -- mapping s(2) s(1) s(0)
  --         s4   s2   s1
  variable s: std_logic_vector(2 downto 0);
  variable o: std_logic_vector(0 to 7);
  -- o(0)='1' denotes absence of errors
  variable index: integer:=0;
  type int_array is array (integer range <>) of integer;
  variable errors: int_array(0 to 8):=(0,0,0,0,0,0,0,0,0);
  begin
    s(0):=w(1) xor w(3) xor w(5) xor w(7);
    s(1):=w(2) xor w(3) xor w(6) xor w(7);
    s(2):=w(4) xor w(5) xor w(6) xor w(7);
```


Esempio di VHDL comportamentale: architettura con case-when

```
    case s is
      when "000" => o:="10000000"; index:=0;
      when "001" => o:="01000000"; index:=1;
      when "010" => o:="00100000"; index:=2;
      when "011" => o:="00010000"; index:=3;
      when "100" => o:="00001000"; index:=4;
      when "101" => o:="00000100"; index:=5;
      when "110" => o:="00000010"; index:=6;
      when "111" => o:="00000001"; index:=7;
      when others => o:="XXXXXXXX"; index:=8;
    end case;
    errors(index):=errors(index)+1;
    if ((index/=0) and (index/=8)) then
      report "corrected␣error␣on␣bit␣"&integer'image(index)&
      "␣num␣=␣"&integer'image(errors(index));
    end if;
  .... -- same as behav_2
```


Esempio: contatore a 2 bit

```
entity count2 is
  generic(prop_delay time:=10 ns);
  port(clock: in bit;
       q1, q0: out bit);
end entity count2;

architecture behav of count2 is
begin
  count: process(clock)
    variable count_value: natural:=0;
    begin
      if (clock='1') then
        count_value:=(count_value+1) mod 4;
        q0 <= bit'val(count_value mod 2) after prop_delay;
        q1 <= bit'val(count_value/2) after prop_delay;
      end if;
    end process count;
end architecture behav;
```


Processi sincronizzati con wait

Istruzione `wait`

- L'istruzione wait provoca la sospensione di un processo o di una procedura
- Sintassi: **`wait [sensitivity_clause] [condition_clause] [time_out_clause];`**
  - **`sensitivity_clause::= on signal_name , signal_name`**

```
wait on clock;
```

  - **`condition_clause::= boolean_expression`**

```
wait until clock='1';
```

  - **`timeout_clause::= for time_expression`**

```
wait for 150 ns;
```


processi equivalenti: sensitivity list vs wait on

```
sum: process(a,b,cin)
  begin
    sum <= a xor b xor cin;
end process sum;
```

=

```
sum: process
  begin
    sum <= a xor b xor cin;
    wait on a, b, cin;
end process sum;
```

Se si mette una sensitity list in un processo non si puó usare l'istruzione wait e viceversa

`wait on` vs `wait until`

Genera un impulso di 1 ns dopo un evento su q

```
pulse: process
  begin
    p <= '0';
    wait on q;
    p <= '1';
    wait for 1 ns;
  end process pulse;
```

Genera un impulso di 1 ns dopo che q si é portato a 1

```
pulse: process
  begin
    p <= '0';
    wait until q='1';
    p <= '1';
    wait for 1 ns;
  end process pulse;
```


Testbench

- Un testbench é il componente top-level nella simulazione
  - la sua dichiarazione non contiene alcuna **`port`**
  - instanzia tutti i componenti del sistema
- I testbench servono anche per:
  - generare gli stimoli per la simulazione tramite descrizioni behavioral
  - confrontare risposte di uscita con valori attesi o verificare alcune proprietá del modello simulato

Schema di testbench

```
entity testbench is
end entity testbench;

architecture example of testbench is
component entity_under_test
      port(....);
end component;
for all: .... -- binding
begin
  generate_waveforms;
  instantiate_component;
  monitoring_statements;
end architecture example;
```


Funzioni e procedure

Sottoprogrammi

- Simili a quelli tipici dei linguaggi software
- Premettono di utilizzare ripetutamente del codice senza riscriverlo
- Consentono di dividere grossi blocchi di software in parti piú piccole e facilemente maneggiabili
- Il VHDL mette a disposizione sia funzioni che procedure

Funzioni

- Producono un singolo valore di ritorno
- Chiamate all'interno di espressioni con parametri passati per posizione
- Non possono modificare i parametri che le sono passati (nessun side-effect)
- Devono contenere un istruzione di **`return`** e devono tornare un oggetto VHDL (nessun void come in C)
- Possono essere dichiarate nei package, nella parte dichiarativa delle architetture o nei processi
- Spazio di nomi locale (fra **`is`** e **`begin`**)

Funzioni: esempi

```
function add_bits(a, b: in bit) return bit is
begin
  return (a xor b);
end function add_bits;
```

```
function add_bits2(a, b: in bit) return bit is
variable result: bit;
begin
  result:=a xor b;
  return result;
end function add_bits2;
```


Funzioni: esempio di chiamata

```
library IEEE;
use IEEE.std_logic_1164.all;

entity aoi_cmos is
  port(a,b,c: in std_logic;
       x: out std_logic);
end entity aoi_cmos;

architecture behav of aoi_cmos is
function aoi (a,b,c: in std_logic) return std_logic is
variable y: std_logic;
begin
  y:=not (a and (b or c));
  return y;
end function aoi;
begin
  x <= aoi(a,b,c);
end architecture behav;
```


Procedure

- Valori di uscita multipli
- Chiamabili autonomamente
- Possono modificare i parametri
- Non richiedono l'istruzione di **`return`**

```
procedure full_adder(signal a, b, cin: in bit;
                     signal s, cout: out bit) is
begin
  s <= (a xor b) xor cin;
  cout <= (a and b) or (a and cin) or (b and cin);
end procedure full_adder;
```


Esempio

```
....
entity 2add is
  port(a,b: in std_logic_vector(1 downto 0);
       cin: in std_logic;
       out: out std_logic_vector(1 downto 0);
       cout: out std_logic);
end entity 2add;
architecture dataflow of 2add is
procedure full_adder(signal a, b, cin: in bit;
                     signal s, cout: out bit) is
begin
  s <= (a xor b) xor cin;
  cout <= (a and b) or (a and cin) or (b and cin);
end procedure full_adder;
signal carry: std_logic;
begin
  full_adder(a(0),b(0),cin,sum(0),carry);
  full_adder(a(1),b(1),carry,sum(1),cout);
end architecture dataflow;
```

Note

Una procedura puó avere sia segnali che variabili come argomenti, ma se ha variabili puó essere chiamata solo dall'interno di un processo

```
procedure full_adder(signal a, b, cin: in bit;
procedure <procedure_name>
  (signal|variable|constant <name1> : in|out|inout <type>;
   signal|variable|constant <name2> : in|out|inout <type>;
                            ... ) is
    <signal constant or variable declarations
    for use within the procedure>
begin
    <code performed by the procedure here>
end procedure;
```


Bus resolution

Risoluzione dei segnali

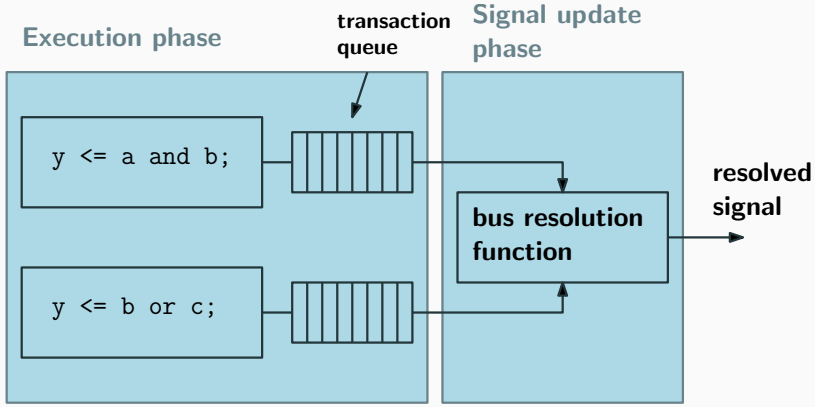
- Il problema é quello di segnali caratterizzati da piú driver
  - possibilitá di conflitti fra i valori pilotati

```
j <= a and b;
j <= a or b;
```

- Da un punto di vista circuitale l'esito di tali conflitti dipende dalla tecnologia e dalle caratteristiche del circuito
  - porte logiche in tecnologia CMOS
  - buffer tristate in tecnologia CMOS
- I tipi di dato base associati ai segnali VHDL non consentono di avere piú driver (es. assegnamenti multipli concorrenti allo stesso segnale)
- **Il VHDL mette a disposizione un meccanismo per descrivere al livello logico il comportamento di segnali pilotati da piú driver**

Bus resolution function

I segnali con piú driver devono avere associata una funzione di risoluzione del bus



Bus resolution function: esempio

- Si vuole costruire una funzione di risoluzione che modelli un segnale con un comportamento di tipo wired-and
- In tale comportamento un segnale alto prevale su quelli bassi
- Il tipo di dato di partenza sia il **bit**
- Il tipo di dato risolto viene poi definito in un package tramite una funzione di risoluzione del bus

Bus resolution function: esempio

```
package bus_resolution is
function wired_and (drivers: in bit_vector) return bit;
subtype wand_bit is wired_and bit;
end package bus_resolution;

package body bus_resolution is
function wired_and (drivers: in bit_vector) return bit is
variable accumulate: bit:='1';
begin
  for i in drivers'range loop
    accumulate:=accumulate and drivers(i);
  end loop;
  return accumulate;
end function wired_and;
end package body bus_resolution;
```


Bus resolution function: esempio

- qui si ha un esempio di utilizzo in un modello con piú driver

```
use work.bus_resolution.all;
entity bus_wand is
  port(a,b,c: in bit; z: out bit);
end entity bus_wand;
architecture wired_and_behav of bus_wand is
signal resolved_node: wand_bit;
begin
  resolved_node <= a;
  resolved_node <= b;
  resolved_node <= c;
  z <= resolved_node;
end architecture wired_and_behav;
```


Tipo di dato `std_logic`

- Il package IEEE std_logic_1164 definisce prima il tipo di dato non risolto **`std_ulogic`**
- A partire da questo viene definito il tipo di dato **`std_logic`** che é di tipo **resolved**

```
-- 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`

Rappresenta lo standard nei progetti industriali ed é ottenuto tramite una funzione di risoluzione

```
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
-------------------------------------------------------
type std_logic_vector is array (natural range <>) of std_logic;
```


Funzione di risoluzione per `std_logic`

- Determina il valore di un segnale pilotato da piú driver
- Il modello per questa funzione é dato da un ordinamento parziale fra gli elementi della multiple valued algebra (**`std_ulogic`**)
- La funzione realizza tale ordinamento tramite una tabella indirizzata dai valori di due driver

Diagramma di Hasse

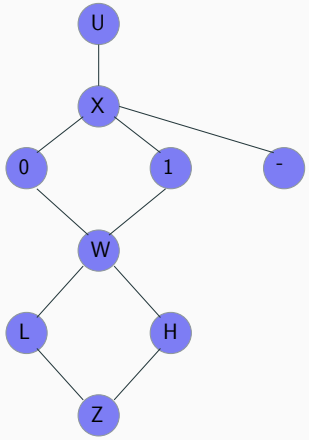


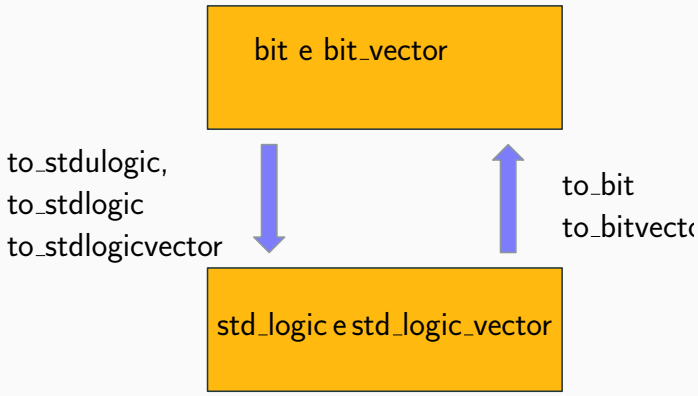
Tabella di risoluzione per `std_logic`

```
-- 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' )  -- | - | );
```


Funzione di risoluzione per `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;
```

Funzioni di conversione fra `std_logic` e `bit`



Utilitá per `std_logic`

- Il VHDL mette a disposizione un insieme di valori che in alcune applicazioni puó risultare ridondante
- Questo problema viene affrontato mettendo a disposizione
  - Sottotipi semplificati **`x01`** e **`x01z`**
  - Funzioni di utilitá generale per l'analisi dei fronti (**`rising_edge`** e **`falling_edge`**) e la rivelazione di valori logici non definiti (**`'U'`**,**`'X'`**,**`'W'`**,**`'Z'`**,**`'-'`**) tramite **`is_x`**

Utilitá per `std_logic`

- Sottotipi semplificati **`x01`** e **`x01z`**

```
subtype x01  is resolved std_ulogic range 'X' to '1';
-- ('X','0','1')
subtype x01z is resolved std_ulogic range 'X' to 'Z';
-- ('X','0','1','Z')
```

e le relative funzioni di conversione da **`std_logic`**

```
subtype x01  is resolved std_ulogic range 'X' to '1';
-- ('X','0','1')
subtype x01z is resolved std_ulogic range 'X' to 'Z';
-- ('X','0','1','Z')
```


Funzioni di utilitá per `std_logic`

```
-------------------------------------------------------------
-- 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;
```

Input/Output

File

- Il VHDL mette a disposizione il package **`textio`**
- Esempio di dichiarazione

```
use std.textio.all;
file data:text open read_mode is "data.txt";
```

- Funzioni associate ai file:
  `file_open();`
  `file_close();`
  `read(); write();`
  `readline(); writeline();`
  `endfile();`
- Le funzioni di **`read`** e **`write`** sono applicabili anche a stringhe

Esempio di utilizzo di file

- Una lista di vettori logici si trova in un file nel seguente formato
  0100 10 ns
  1010 40 ns
  ....
  1111 200 ns
- Si vuole realizzare un architettura che legga tali vettori di test e li applichi in maniera ordinata ai suoi segnali

Esempio di operazioni su file

```
library ieee;
use ieee.std_logic_1164.all, std.textio.all;

entity prova is
end entity prova;

architecture leggi_file of prova is
file test_vectors: text;
constant n: natural := 4;
signal x: std_logic_vector(n-1 downto 0);

begin
p: process
variable inline : line; -- stringa
variable ch : character;
variable del : time;
variable y: bit_vector(n-1 downto 0);
```

read non legge std_logic_vector

Esempio di operazioni su file

```
begin
  file_open(test_vectors,"test.vec",read_mode);
  while not endfile(test_vectors) loop
    readline(test_vectors,inline);
  for i in n-1 downto 0 loop
    read(inline,y(i));
  end loop;
  read(inline,ch);
  read(inline,del);
  x <= to_stdlogicvector(y);
  wait for del;
end loop;
file_close(test_vectors);
wait;
end process p;
end architecture leggi_file;
```


Dati di itipo numerico

Null transactions

Rappresentazione dei segnali aritmetici

- Se vogliamo salire con il livello di astrazione nella rappresentazione dei sistemi digitali bisogna introdurre la possibilitá di calcolare espressioni numeriche come ad esempio nei DSP
- I possibili candidati per rappresentare questi dati sono **`std_logic_vector`** e **`integer`**, peró
  - gli operatori aritmetici non sono applicabili a **`std_logic_vector`**
  - gli interi sono molto lontani da rappresentazioni al livello logico
- Il VHDL mette a disposizione tipi d dato specifici per le operazioni aritmetiche con accessibilitá al livello di singolo bit

Package `numeric_std`

- Appartiene alla libreria **`ieee`**
- Supporta sia rappresentazioni con segno che enza segno:
  - tipo di dato **unsigned**:
    `signal a,b:unsigned(n-1 downto 0);`
  - tipo di dato **signed**:
    `signal a,b:signed(n-1 downto 0);`
- Questi dati mantengono una struttura di tipo array con vantaggi dal punto di vista della sintesi
- L'assegnamento avviene in maniera simile a **`std_logic_vector`**
  - esempio: $n = 4$, **`a<="1010";`**, e **`a`** é di tipo unsigned il valore in base 10 é 10, se **`a`** é di tipo signed il valore in base 10 é -6

Somma di unsigned

- con perdita del bit piú significativo del risultato (somma in modulo 2^n)

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

- senza perdita del bit piú 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

- risultato senza overflow

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

- risultato con overflow e con soluzione del problema

```
signal a,b,c: signed(7 donwto 0);
signal d: signed(8 downto 0);
.......
a<="10001110"; -- -114
b<="11011010"; -- -38
c<=a+b; -- "01101000" 104 overflow error
d<=('1' & a)+('1' & b); -- "101101000" -152
```


Funzioni di conversione

- **`unsigned()`** converte da **`std_logic_vector`** a **`unsigned`**
- **`signed()`** converte da **`td_logic_vector`** a **`signed`**
- **`st_logic_vector()`** coverte da **`signed/unsigned`** a **`std_logic_vector()`**
- La dimensione dei dati deve essere la stessa