Roadmap

- In questo capitolo vedremo gli elementi sintattici e semantici base del "linguaggio" VHDL.
- **Outline:**
  - Elementi lessicali
  - Identificatori
  - Oggetti (costanti, segnali, variabili)
  - Tipi (interi, physical, floating, enumerazione, …)
  - Array, Record
  - Aliasing
  - Subtype
  - Attributi
  - Operatori

Convenzioni Tipografiche

- In tutte le slide si useranno le seguenti convenzioni tipografiche:
  - Le keyword sono riportate in maiuscolo nei listati VHDL (ad es. **`ENTITY`**, **`ARCHITECTURE`**, **`FOR`**) e non possono essere usate come nomi assegnati dall'utente ad oggetti; comunque il VHDL non è case sensitive;
  - **`Nome_oggetto`** è un oggetto del VHDL a cui l'utente ha assegna l'identificativo nome_oggetto.;

Elementi Lessicali (1)

- Il VHDL non è **case sensitive** (a differenza del C) sia per gli identificatori che per le parole chiavi (*keyword*):

  ingresso = INGRESSO = INgrESso

  Architecture = aRCHITECTURE = aRcHiTeCtURe

- Ogni riga in VHDL può essere suddivisa su più linee e non c'è un particolare formato del testo in ingresso al simulatore: spazi, tabulazioni e ritorni a capo sono trattati allo stesso modo;

- **Commenti**: i commenti in VHDL incominciano in un qualunque punto di una riga, con due trattini consecutivi e terminano alla fine della linea;
  - I commenti sono del tutto ignorati dal simulatore;
  - Esempio:

```
-- io sono un commento e vengo ignorato!
```

Elementi Lessicali (2)

- **Numeri**: I numeri possono essere rappresentati come interi (integer) o come reali (real);

– Esempi:

```
0       1       987E6       -- numeri interi
0.0     0.5     2.71E-9     -- numeri reali
```

- **Caratteri**: sono composti da caratteri ASCII racchiusi fra apici;

– Esempi:

```
'A'     'a'     ''''        ' '     -- caratteri
```

- **Stringhe**: sono array di caratteri racchiusi fra doppi apici;

– Per includere in una stringa il carattere " si deve usare la sequenza " ";

– Esempi:

```
"Io sono una stringa"       -- stringa
"Io sono la stringa ""stringa"" e voi? "
-- è la sequenza di caratteri Io sono la stringa "stringa" e voi?
```

Elementi Lessicali (3)

- **Bit Strings**: per rappresentare le stringhe di bit si possono usare le notazioni binarie, ottali o esadecimali;

– Si possono usare gli underscore _ per separare le cifre in gruppi, senza che il simulatore VHDL li consideri;

– Esempio:

– il numero 86 in decimale può essere rappresentato alternativamente con:

```
B"1010110"  -- è la stringa di 7 bit 1010110
O"126" -- è la stringa di 3 cifre ottali (9 bit) equivalente a B"001_010_110"
X"56"  -- è la stringa di 2 cifre esadecimali (8 bit) equivalente a B"0101_0110“
```

anche se il numero di bit su cui il numero 86 è rappresentato è diverso nei vari casi;

Identificatori

- **Identificatori**: gli identificatori sono definiti dal programmatore per identificare degli oggetti VHDL;

– Esempi:

```
overflow        signal_1        -- esempi di identificatori
```

– Non si possono usare le parole riservate (BEGIN, COMPONENT, ARCHITECTURE, …) come identificatori. In tutte le slide, le parole chiave saranno riportate in maiuscolo, anche se il VHDL non è case sensitive;

– Gli identificatori leciti devono iniziare con una lettera e possono essere composti da una sequenza di underline (_), lettere o numeri

– Gli underline sono significativi (*questo_segnale* non è *questosegnale*)

Oggetti (1)

- Un **oggetto** è una entità VHDL che ha un suo *nome* (un identificatore alfanumerico unico), un suo *tipo* ed un suo *valore* (che deve appartenere all'insieme dei valori leciti all'interno del tipo);
- Esistono tre classi di oggetti in VHDL costanti (CONSTANT), variabili (VARIABLE) e segnali (SIGNAL);

1. Una **costante** è un oggetto che è inizializzato con un certo valore all'atto della sua definizione e che non può essere successivamente modificato:

```
CONSTANT delay : Time := 5 ns;
```

2. Una **variabile** è un oggetto il cui valore può essere modificato dopo che stato definito (può essere usato solo nei corpi sequenziali, *process*);

- Una variabile, appena viene definita, deve avere un suo valore. Questo valore iniziale può essere:
  - assegnato esplicitamente alla variabile (tramite un valore del tipo o attraverso una espressione),
  - assegnato implicitamente (per default) secondo delle regole fissate dallo standard.

Oggetti (2)

- Esempi di definizione di variabili sono (dove il tipo trace_array è definito precedentemente):

```
TYPE trace_array IS ARRAY (0 TO 99) OF BOOLEAN;
VARIABLE trace : trace_array;

VARIABLE count : NATURAL := 0;
```

3. I **segnali** sono usati per connettere dei moduli fra di loro (sono leciti solo nei corpi concorrenti, *architecture*);
- Anche un segnale deve avere un suo valore di inizializzazione, che può essere specificato esplicitamente o determinato implicitamente.

- In generale lo scopo di ciascuna delle 3 classi degli oggetti è:
  - constanti → parametri fissi di un design;
  - variabili → per mantenere una informazione temporanea o lo stato nei process;
  - segnali → per rappresentare dei collegamenti hardware.

Tipi in VHDL (1)

- Il VHDL è un linguaggio fortemente tipizzato (*strongly typed*).
- Il VHDL mette a disposizione diversi tipi base, ma permette di definire i propri tipi (sia scalari che compositi).
- Esempi di tipi scalari sono *Boolenan*, *Character*, *Bit*, *Real*, etc.
- I tipi composti sono gli array e i record.

Tipi in VHDL (2)

- Il tipo *bit* ha due valori: '1' e '0';
- il tipo *bit_vector* rappresenta un array di *bit*.

- Normalmente nella descrizione di sistemi digitali è necessario rappresentare anche stati diversi dallo stato alto e basso, come lo stato di alta impedenza o quello di *unknown*.
- Sono perciò di uso comune i tipi *std_logic* e *std_logic_vector*, usati in sostituzione dei tipi *bit* e *bit_vector*, rispettivamente. I valori disponibili sono:
  - 'U' uninitialized
  - 'X' unknown
  - '0' low
  - '1' high
  - 'Z' high impedance
  - 'W' weak unknown
  - 'L' weak low
  - 'H' weak high
  - '-' don't care

Tipi interi

- Un tipo intero è un intervallo di numeri interi entro uno specifico range.
- Esempi:

  TYPE byte_int IS RANGE 0 to 255;

  TYPE signed_word_int IS RANGE –32768 to 32767;

  TYPE bit_index IS RANGE 31 downto 0;

- In VHDL è predefinito un tipo intero chiamato *integer*. L'intervallo di valori che può contenere dipende dall'implementazione, ma lo standard richiede che includa l'intervallo compreso fra –2147483647 e +2147483647 (codificabile con 32 bit).

Tipi physical (1)

- Il tipo *physical* è tipo numerico che permette di rappresentare alcune quantità fisiche (resistenza, tempo, tensione, corrente...).
- La dichiarazione di un tipo fisico include la specifica di una unità base e opzionalmente un insieme di unità secondarie (multipli o sottomultipli).
- Ad esempio il tipo lunghezza:

```
TYPE length IS RANGE 0 to 1E9
    UNITS       um;                 -- unità base
    mm = 1000 um; cm = 10 mm; m = 1000 mm; in = 25.4 mm; ft = 12 in;
      yd = 3 ft; rod = 198 in; chain = 22 yd; furlong = 10 chain;
    -- unità secondarie
END UNITS;
```

- Il tipo resistenza potrebbe essere definito come:

```
TYPE resistance IS RANGE 0 to 1E8
UNITS           ohms;                   -- unità base
kohms = 1000 ohms; Mohms = 1E6 ohms;    -- unità secondarie
end units;
```

Tipi physical (2)

- Un tipo physical predefinito in VHDL è il tipo *time*. Esso è usato per rappresentare i ritardi nelle simulazioni. La sua definizione è:

```
TYPE  time IS RANGE implementation_defined
        UNITS  fs;                      -- unità base
        ps = 1000 fs; ns = 1000 ps; us = 1000 ns; ms = 1000 us; sec = 1000 ms;
           min = 60 sec; hr = 60 min;
        -- unità secondarie
      END UNITS;
```

- Per far riferimento ad un valore appartenente ad un tipo fisico, è possibile usare le unità di misura. Ad esempio:

  10 mm

  1200 ohm

  23 ns

- I tipi physical (escluso il tipo time) servono a dare maggiore leggibilità al codice e sono usati raramente in VHDL.

Tipo floating point

- Il tipo *floating point* è una approssimazione discreta dell'insieme dei numeri reali in un determinato range.
- Esempi:

  TYPE signal_level IS RANGE –10.00 to +10.00;

  TYPE probability IS RANGE 0.0 to 1.0;
- In VHDL è predefinito un tipo floating point chiamato real. Il range di questo tipo dipende dall'implementazione, anche se include secondo lo standard l'intervallo compreso fra –1E38 e +1E38.

- I tipi floating point sono raramente usati in VHDL (difficilmente sono sintetizzabili).

Tipo enumerazione (1)

- Un tipo enumerazione è un insieme ordinato di identificatori di caratteri, tutti distinti fra loro, anche se in tipi di enumerazione distinti lo stesso identificativo può essere usato più volte.
- Ad esempio, la definizione TYPE qit IS ('0', '1', 'Z', 'X') definisce un nuovo tipo di logica multivalore, in cui:
  - *qit* è l'identificatore di tipo
  - *'0', '1', 'Z', 'X'* sono gli elementi del tipo (definiti per enumerazione)
  - *'0'* è il valore assegnato per default
- Altri esempi possono essere:

  TYPE logic_level IS (unknown, low, undriven, high);

  TYPE alu_function IS (disable, pass, add, subtract, multiply, divide);

  TYPE octal_digit IS ('0', '1', '2', '3', '4', '5', '6', '7');

Tipo enumerazione (2)

- Alcuni tipi enumerazione sono predefiniti in VHDL, fra cui:

```
TYPE boolean IS (false, true);
TYPE bit IS ('0', '1');
TYPE severity_level IS (note, warning, error, failure);
TYPE character IS (
    NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL,
    BS, HT, LF, VT, FF, CR, SO, SI,
    DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB,
    CAN, EM, SUB, ESC, FSP, GSP, RSP, USP,
    ' ', '!', '"', …
```

Array (1)

- Gli array sono tipi di dati fondamentali nel VHDL.
- Un array in VHDL è una collezione di elementi indicizzati tutti dello stesso tipo.

  TYPE word IS ARRAY (31 DOWNTO 0) of bit;

  TYPE memory IS ARRAY (address) of word;
- Gli array possono essere monodimensionali (vettori) o multidimensionali.

  TYPE transform IS ARRAY (1 to 4, 1 to 4) of real;
- Si noti che le seguenti definizioni di tipo definiscono tipi diversi

  TYPE word_0 IS ARRAY (31 DOWNTO 0) of bit;

  TYPE word_1 IS ARRAY (0 TO 31) of bit;

  Perché il primo elemento word_0 è != dal primo elemento word_1
- Un array può essere constrained, ovvero i limiti degli indici sono stabiliti, oppure unconstrained se i limiti sono stabiliti all'atto della dichiarazione di una variabile di quel tipo.

  TYPE vector IS ARRAY (integer range <>) of real;

Array (2)

- Un elemento di un array è riferito usando gli indici ed il nome dell'oggetto: a(1), b(1, 1)
- Una slice di elementi contigui di un array monodimensionale può essere riferita usando un *range* degli indici: a(8 TO 15) è un vettore di 8 elementi parte dell'array a.
- Si supponga di aver definito un array di caratteri in questo modo:

  TYPE c IS ARRAY (1 TO 4) of bit;

  è possibile scrivere un valore in ogni posizione di un oggetto (ad es. segnale) di tipo c o usando una notazione posizionale:

  segnale_c <= ('1','0','1','1');

  oppure usando una notazione con associazione per nome (in cui l'ordine non è importante):

  segnale_c <= (1 => '1', 3 => '1', 4 => '1', 2 => '0');

  segnale_c <= (1 => '1', 2 => '0', 3 => '1', 4 => '1');
- Le notazioni posizionale e per associazione nominale possono essere mischiate, usando la keyword *others* per assegnare gli elementi non specificati con un unico valore:

  segnale_c <= ('1', 2 => '0', OTHERS => '1');

Array (3)

- Qualche esempio interessante di array (Navabi) basati sul tipo:

  TYPE qit IS ('0', '1', 'Z', 'X');

```
TYPE qit_nibble IS ARRAY ( 3 DOWNTO 0 ) OF qit;
TYPE qit_byte IS ARRAY ( 7 DOWNTO 0 ) OF qit;
-- due vettori (un nibble e un byte di qit)

TYPE qit_4by8 IS ARRAY ( 3 DOWNTO 0, 0 TO 7 ) OF qit;
-- il tipo qit_4by8 è una matrice bidimensionale

TYPE qit_nibble_by_8 IS ARRAY ( 0 TO 7 ) OF qit_nibble;
-- il tipo qit_nibble_by_8 è un vettore di qit_nibble
```

- In generale qit_4by8 e qit_nibble_by_8 specificano differenti tipi e sono accessibili attraverso un diverso indirizzamento degli indici.

```
SIGNAL sq8 : qit_byte := "ZZZZZZZZ";
-- inizializzazione con costante
sq8 <= "Z101000Z";
-- assegnazione con costante
```

Array (4)

```
TYPE qit IS ('0', '1', 'Z', 'X');
TYPE qit_nibble IS ARRAY (3 DOWNTO 0) OF qit;
TYPE qit_byte IS ARRAY (7 DOWNTO 0) OF qit;
TYPE qit_word IS ARRAY (15 DOWNTO 0) OF qit;
TYPE qit_4by8 IS ARRAY (3 DOWNTO 0, 0 TO 7) OF qit;
TYPE qit_nibble_by_8 IS ARRAY (0 TO 7) OF qit_nibble;

SIGNAL sq1 : qit;
SIGNAL sq4 : qit_nibble;
SIGNAL sq8 : qit_byte;
SIGNAL sq16 : qit_word;
SIGNAL sq_4_8 : qit_4by8;
SIGNAL sq_nibble_8 : qit_nibble_by_8;

sq8 <= sq16(11 DOWNTO 4);          -- estrae una slice di 8-bit da sq16
sq16(15 DOWNTO 12) <= sq4;         -- assegna una slice di 4 bit di sq16
sq1 <= sq_4_8(0, 7);               -- estrae un bit da sq_4_8
sq4 <= sq_nibble_8(2);             -- assegna il 3-o nibble di sq_nibble_8 a sq4
sq8 <= sq8(0) & sq8(7 DOWNTO 1);   -- shift a destra di sq8
sq4 <= sq8(2) & sq8(3) & sq8(4) & sq8(5);
-- concatenazione di alcuni bit di sq8 in un nibble
```

Array (5)

```
TYPE qit_4by8 IS ARRAY (3 DOWNTO 0, 0 TO 7) OF qit;
SIGNAL sq_4_8 : qit_4by8 :=
                  (
        ('0', '0', '1', '1', 'Z', 'Z', 'X', 'X'),  -- sq_4_8 (3, 0 TO 7)
        ('X', 'X', '0', '0', '1', '1', 'Z', 'Z'),  -- sq_4_8 (2, 0 TO 7)
        ('Z', 'Z', 'X', 'X', '0', '0', '1', '1'),  -- sq_4_8 (1, 0 TO 7)
        ('1', '1', 'Z', 'Z', 'X', 'X', '0', '0')   -- sq_4_8 (0, 0 TO 7)
                  );
```

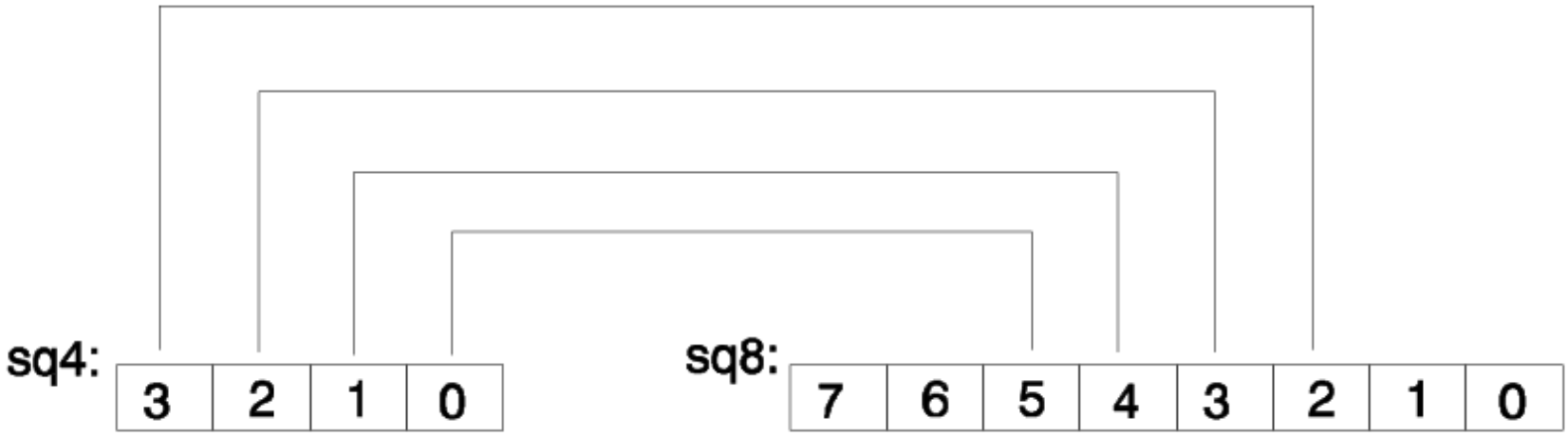
- L'uso di parentesi innestate permette di accedere ad array multidimensionali;
- Il set di parentesi più interno corrisponde agli indici più a destra;
- In figura c'è l'inizializzazione di un segnale con una costante di tipo qit_4by8, ed è anche riportata nei commenti come accedere a delle slice "orizzontali" di un oggetto di tipo qit_4by8.

Array (6)

```
TYPE qit_nibble IS ARRAY(3 DOWNTO 0) OF qit;
TYPE qit_byte IS ARRAY (7 DOWNTO 0) OF qit;
SIGNAL sq4 : qit_nibble;
SIGNAL sq8 : qit_byte;

sq4 <= sq8(2) & sq8(3) & sq8(4) & sq8(5);
-- concatenazione di alcuni bit di sq8 in un nibble
```

- La corrispondenza degli elementi fra il nibble costruito a destra dell'assegnazione e la definizione del tipo del segnale a sinistra è riportata in figura:



Array (7)

- In VHDL è possibile usare un tipo discreto (ad es. qit) come indice per un array;
- Una costante è usata per implementare una tabella di verità;

```
TYPE qit IS ('0', '1', 'Z', 'X');
TYPE qit_2d IS ARRAY (qit, qit) OF qit;

ENTITY nand2_q IS
    PORT (i1, i2 : IN qit; o1 : OUT qit);
END nand2_q;
--
ARCHITECTURE average_delay OF nand2_q IS
CONSTANT qit_nand2_table : qit_2d := (
        -- '0' '1' 'Z' 'X'
        -- ---------------------------------------
        ('1', '1', '1', '1'), -- '0'
        ('1', '0', '0', 'X'), -- '1'
        ('1', '0', '0', 'X'), -- 'Z'
        ('1', 'X', 'X', 'X')); --'X'
BEGIN
        o1 <= qit_nand2_table (i1, i2) AFTER 5ns;
END average_delay;
```

Record (1)

- I record sono una collezione di elementi anche di diverso tipo, ad es.

```
TYPE a_new_record_type IS RECORD
    id1, id2 : type_1;
    id3 : array_type_2;
    id4, id5, id6 : record_type_3;
END RECORD;
```

- Per far riferimento ad un elemento in un oggetto record, si può usare il suo identificativo.
- Ad esempio:

```
VARIABLE example_1 : a_new_record_type ;
...
instr_1.id1 := elem_type_1;
instr_1.id2 := elem_type_1;
instr_1.id3 := elem_array_type_2;
```

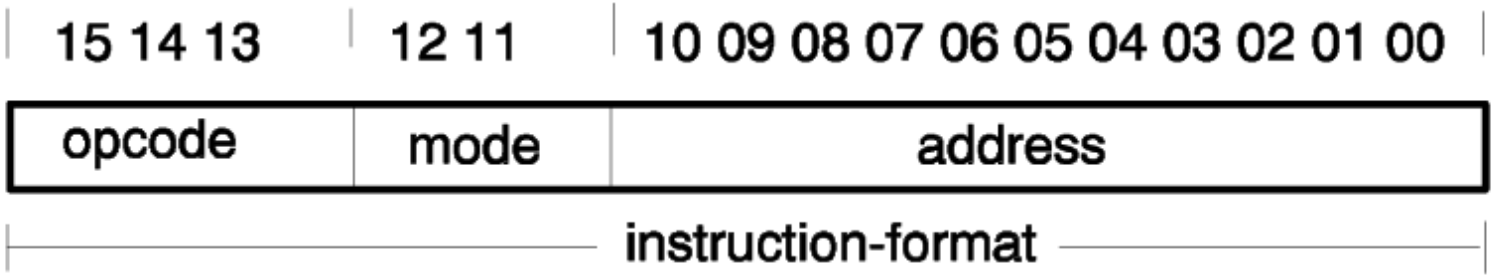
- Così come per gli array, anche per i record è possibile usare sia la notazione posizionale che una corrispondenza per nome.

Record (2)

```
TYPE opcode IS (sta, lda, add, sub, and, nop, jmp, jsr);
TYPE mode IS RANGE 0 TO 3
TYPE address IS BIT_VECTOR (10 DOWNTO 0);

TYPE instruction_format IS RECORD
     opc : opcode;
     mde : mode;
     adr : address;
END RECORD;
```

- Una definizione di record è utile ad es. per rappresentare il formato e la codifica di una istruzione di un processore;



Record (3)

```
TYPE opcode IS (sta, lda, add, sub, and, nop, jmp, jsr);
TYPE mode IS RANGE 0 TO 3
TYPE address IS BIT_VECTOR (10 DOWNTO 0);

TYPE instruction_format IS RECORD
     opc : opcode;
     mde : mode;
     adr : address;
END RECORD;

SIGNAL instr : instruction_format := (nop, 0, "00000000000");

instr.opc <= lda;
instr.mde <= 2;
instr.adr <= "00011110000";

instr <= (lda, 2, "00011110000");
```

- Una costante è definita usando gli elementi del record (ordinatamente secondo la definizione del record) tra parentesi;
- L'assegnazione di un segnale di tipo record può essere fatta per gli elementi del record individualmente oppure per tutti gli elementi assieme;

Aliasing (1)

- Una volta definito un tipo, è possibile assegnare un nome alternativo al tipo o ad una sua parte, usando la dichiarazione *alias*, in modo da potervi fare riferimento più comodamente.
- L'aliasing chiaramente è semplicemente un renaming e non definisce nessun oggetto.
- Ad esempio:

```
VARIABLE instr : BIT_VECTOR(31 DOWNTO 0);
ALIAS op_code : BIT_VECTOR(7 DOWNTO 0) IS instr(31 DOWNTO 24);
```

dichiara op_code come alias per gli 8 bit più significativi di instr;

- Un altro esempio dell'utilizzo degli alias, può essere nell'assegnazione di nomi simbolici ai bit di un registro di stato:

```
ALIAS c_flag : BIT IS flag_register(3);
ALIAS v_flag : BIT IS flag_register(2);
ALIAS n_flag : BIT IS flag_register(1);
ALIAS z_flag : BIT IS flag_register(0);
```

Aliasing (2)

```
TYPE opcode IS (sta, lda, add, sub, and, nop, jmp, jsr);
TYPE mode IS RANGE 0 TO 3
TYPE address IS BIT_VECTOR (10 DOWNTO 0);

TYPE instruction_format IS RECORD
    opc : opcode;
    mde : mode;
    adr : address;
END RECORD;

SIGNAL instr : instruction_format := (nop, 0, "00000000000");

ALIAS page : BIT_VECTOR (2 DOWNTO 0) IS instr.adr (10 DOWNTO 8);
ALIAS offset : BIT_VECTOR (7 DOWNTO 0) IS instr.adr (7 DOWNTO 0);
page <= "001"; -- 3 bits
offset <= X"F1"; -- 8 bits
offset <= B"1111_0001"; -- 8 bits
```

- Usando l'aliasing è possibile separare nell'indirizzo address, una parte che fa riferimento alla pagina ed una che fa riferimento all'offset nella pagina;

Subtype (1)

1. L'uso di un *subtype* permette di usare un sottoinsieme dei valori assunti da un tipo:

   SUBTYPE digits IS character RANGE '0' to '9';

2. Un subtype può essere anche usato per rendere constrained una dichiarazione di array unconstrained, specificando i limiti per gli indici.

- Ad esempio:

  SUBTYPE id IS string(1 TO 20);

  SUBTYPE word IS bit_vector(31 DOWNTO 0);

- Ci sono alcuni subtype di tipo numerico predefiniti in VHDL:

  SUBTYPE natural IS integer RANGE 0 TO highest_integer;

  SUBTYPE positive IS integer RANGE 1 TO highest_integer;

Subtype (2)

- Un subtype è completamente compatibile con il suo tipo base, ad es. il subtype:

  SUBTYPE compatible_nibble_bits IS BIT_VECTOR (3 DOWNTO 0);

  è completamente compatibile con il tipo BIT_VECTOR.

- Mentre il tipo:

  TYPE nibble_bits IS ARRAY (3 DOWNTO 0) OF BIT;

  non è compatibile con il tipo BIT_VECTOR.

Attributi (1)

- Un largo insieme di oggetti VHDL (es. tipi enumerativi e discreti, tipi array, segnali, etc.) dichiarati in una descrizione VHDL possono avere delle informazioni addizionali, associando loro degli attributi.
- Ci sono un gran numero di attributi predefiniti per i tipi, gli array ed i segnali, ed altri attributi possono essere definiti dall'utente (raramente usati).
- In generale gli attributi sono poco usati in VHDL per descrivere dei sistemi. Alcuni attributi utili sono ('event, 'stable).
- Una applicazione molto utile invece è per dare delle direttive ad un sintetizzatore (ad es. rimuovi la gerarchia, tratta come black box, …).
- Su Navabi c'è una trattazione in dettaglio degli attributi per i tipi scalari, per i discreti, per gli array e per i segnali.

- Per riferirsi ad un attributo si usa l'apice ' (tick),

```
obj'attr    -- fa riferimento all'attributo attr dell'oggetto obj
```

Attributi (2)

- Esempi di attributi definiti sui range (tipi discreti) sono:

```
T'left          -- limite sinistro di T
                -- (l'elemento definito più "a destra")
T'right         -- analogamente a sinistra
T'low           -- limite inferiore di T (l'elemento "minore")
T'high          -- analogamente per il limite superiore
```

- Esempi dell'effetto degli attributi sui range:

```
TYPE T1 IS RANGE(0 to 10);              -- ascending range

T'left = T'low = 0;
T'right = T'high = 10;
```

```
TYPE T2 IS RANGE(10 downto 0);          -- descending range

T'left = T'high = = 10;
T'right =  T'low = 0;
```

Attributi (3)

- Un esempio degli attributi predefiniti per un tipo enumerativo:

```
TYPE qit IS ('0', '1', 'Z', 'X');
SUBTYPE tit IS qit RANGE '0' TO 'Z';
```

Attribute	Description	Example	Result
'BASE	Base of type	tit'BASE	qit
'LEFT	Leftbound of type or subtype.	tit'LEFT qit'LEFT	'0' '0'
'RIGHT	Right bound of type or subtype.	tit'RIGHT qit'RIGHT	'Z' 'X'
'HIGH	Upper bound of type or subtype.	INTEGER'HIGH tit'HIGH	Large 'Z'
'LOW	Lower bound of type or subtype.	POSITIVE'LOW qit'LOW	1 '0'
'POS(V)	Position of value V in base of type.	qit'POS('Z') tit'POS('X')	2 3
'VA1(P)	Value at Position P in base of type	qit'VAL(3) tit'VAL(3)	'X' 'X'
'SUCC(V)	Value, after value V in base of type.	tit'SUCC('Z')	'X'
'PRED(V)	Value, before value V in base of type.	tit'PRED('1')	'0'
'LEFTOF(V)	Value, left of value V in base of type.	tit'LEFTOF('1') tit'LEFTOF('0')	'0' Error
'RIGHTOF(V)	Value,right of value V in base of type.	tit'RIGHTOF('1') tit'RIGHTOF('Z')	'Z' 'X'

Attributi (4)

- Un esempio degli attributi predefiniti per gli array:

```
TYPE qit_4by8 IS ARRAY (3 DOWNTO 0, 0 TO 7) OF qit;
SIGNAL sq_4_8 : qit_4by8;
```

Attribute	Description	Example	Result
'LEFT	Left bound	sq_4_8'LEFT(1)	3
'RIGHT	Right bound	sq_4_8'RIGHT sq_4_8'RIGHT(2)	0 7
'HIGH	Upper bound	sq_4_8'HIGH(2)	7
'LOW	Lower bound	sq_4_8'LOWS(2)	0
'RANGE	Range	sq_4_8'RANGE(2) sq_4_8'RANGE(1)	0 TO 7 3 DOWNTO 0
'REVERSE_RANGE	Reverse Range	sq_4_8'REVERSE_RANGE(2) sq_4_8'REVERSE_RANGE(1)	7 DOWNTO 0 0 TO 3
'LENGTH	Length	sq_4_8'LENGTH	4

Attributi (5)

- Un esempio degli attributi predefiniti per i segnali:

```
SIGNAL s1 : BIT;
```

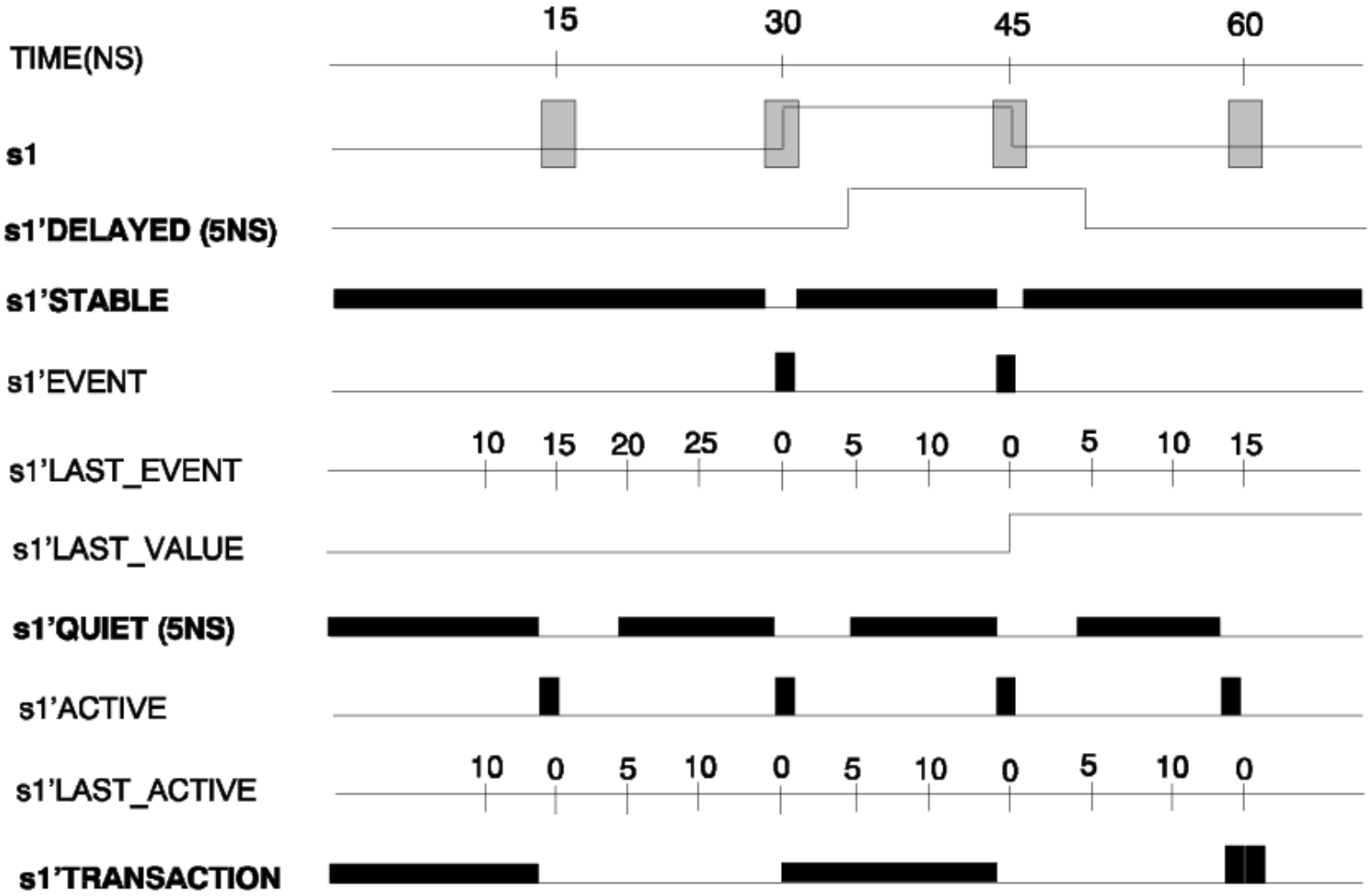
Attribute	T/E	Example Description	Kind	Type
'DELAYED	-	s1'DELAYED(5 NS)	SIGNAL	As s1
A copy of s1, but delayed by 5 NS. If no parameter or 0, delayed by delta. Equivalent to TRANSPORT delay of s1.				
'STABLE	EV	s1'STABLE(5 NS)	SIGNAL	BOOLEAN
A signal that is TRUE if s1 has not changed in the last 5 NS. If used with no parameter or 0, the resulying signal is TRUE if s1 has not changed in the current simulation time.				
'EVENTS	EV	s1'EVENT	Value	BOOLEAN
If s1 changes in the current simulation cycle, s1'EVENT will be TRUE for this cycle (delta time).				
'LAST_EVENT	EV	s1'LAST_EVENT	Value	Time
The amount of time since the last value change on s1. If s1'EVENT is TRUE, the value of s1'LAST_EVENT is 0.				
'LAST_VALUE	EV	s1'LAST_VALUE	Value	As s1
The value of s1 before the most recent event occurs on it.				

Attributi (6)

Attribute	T/E	Example Description	Kind	Type
'QUIET	TR	s1'QUIET(5 NS)	SIGNAL	BOOLEAN
A signal that is TRUE if no transaction has been placed on s1 in the last 5 NS. If no parameter of 0, for current simulation cycle is assumed.				
'ACTIVE	TR	s1'ACTIVE	Value	BOOLEAN
If s1 has had a transaction in the current simulation cycle, s1'ACTIVE will be TRUE for this simulation cycle, for * time.				
'LAST_ACTIVE	TR	s1'LAST_ACTIVE	Value	Time
Tthe amount of time since the last transaction occurred on s1. If s1'ACTIVE is TRUE, s1'LAST_ACTIVE is 0.				
'TRANSACTION	TR	s1'TRANSACTION	SIGNAL	BIT
A signal that toggles each time a transaction occurs on s1.				

Attributi (7)

- Un esempio con un segnale:



Attributi (8)

- Un tipico esempio di utilizzo degli attributi sui segnali è nella individuazione di un fronte attivo di un clock;
- Ad es. un flip-flop D attivo sul fronte di discesa:

```
ENTITY brief_d_flip_flop IS
        PORT (d, c : IN BIT; q : OUT BIT);
END brief_d_flip_flop;
--
ARCHITECTURE falling_edge OF brief_d_flip_flop IS
        SIGNAL tmp : BIT;
BEGIN
        tmp <= d WHEN (c = '0' AND NOT c'STABLE) ELSE tmp;
        q <= tmp AFTER 8 NS;
END falling_edge;
```

Operatori (1)

- Il VDHL include un vasto insieme di operatori:
  - operatori vari : ** abs not
  - operatori moltiplicativi : * / mod rem
  - operatori di segno : + -
  - operatori di addizione : + - &
  - operatori di shift : sll srl sla sra rol ror
  - operatori relazionali : = /= < <= > >=
  - operatori logici : and or nand nor xor xnor

- Le diverse classi di operatori sono riportate secondo il loro ordine di priorità.
- Naturalmente, possono essere usate le parentesi per cambiare tale ordine quando si scrivono espressioni.

Operatori vari

- ** esponenziazione
- ABS valore assoluto

Esempi:

```
VARIABLE A,B,C :Integer;

A:= 2; B:= 8;
C:= A ** B;     -- C = 2 ** 8 = 256

A:= 4; B:= -2;
C:= A ** B;     -- C = 4 ** (-2) = 1 / (4**2) = 0.0625

A:= -14;
C:= ABS (A);    -- C = 14
```

Operatori moltiplicativi

- * moltiplicazione
- / divisione
- mod modulo
- rem resto

Esempi:

```
VARIABLE A,B,C :Integer;
VARIABLE X,Y,Z :Real;

A:= 2; B:= 8;
C:= A * B;    -- C = 2 * 8 = 16

X:= 4; Y:= -8;
Z:= X / Y;    -- Z = 4 / (-8) = -0.5

A:= 9; B:= 7;
C:= A MOD B;   -- C = 9 mod 7 = 2
```

Operatori di addizione

- + addizione
- - sottrazione
- & concatenamento

Esempi:

```
VARIABLE A,B,C :Integer;

A:= 2; B:= 8;
C:= A + B;    -- C = 2 + 8 = 10


SIGNAL sq4 : std_logic_vector(3 DOWNTO 0);
SIGNAL sq8 : std_logic_vector(7 DOWNTO 0);

sq4 <= sq8(2) & sq8(3) & sq8(4) & sq8(5);
```

Operatori di shift

- sll Shift left logical
- srl Shift right logical
- sla Shift left arithmetic
- sra Shift right arithmetic
- rol Rotate left logical
- ror Rotate right logical

```
variable A, C : BIT_VECTOR(3 downto 0);

A := ('1','0','1','1');
C := A sll 1 ;    -- ('0', '1', '1', '0')
C := A sll 3 ;    -- ('1', '0', '0', '0')

A := ('1','0','1','1');
C := A sla 1 ;    -- ('0', '1', '1', '1')
C := A sla 3 ;    -- ('1', '1', '1', '1')

A := ('1','0','1','1');
C := A rol 1 ;    -- ('0', '1', '1', '1')
C := A rol 2 ;    -- ('1', '1', '0', '1')
```

Operatori relazionali

- = uguaglianza
- /= diseguaglianza
- < minore di
- <= minore o uguale
- > maggiore di
- >= maggiore o uguale

```
variable A: REAL := 100.0;
variable B : BIT_VECTOR(7 downto 0) :=
('0','0','0','0','0','0','0','0');
variable C, D : BIT_VECTOR(1 to 0);

A /= 342.54      -- True
A = 100.0        -- True
B /= ('1', '0', '0', '0', '0', '0', '0', '0') -- True
C = D            -- True
A > 42.54        -- True
A >= 100.0       -- True
B < ('1', '0', '0', '0', '0', '0', '0', '0') -- True
C <= B           -- True
```

Operatori logici

- and
- or
- nand
- nor
- xor
- xnor
- not

```
variable A, B, C, D, Y : BIT := '1';

Y := (A AND (B XNOR C)) NOR D;
```