VHDL

Very High Speed Integrated Circuit (VHSIC)

Hardware

Description

Language

Il VHDL è un linguaggio di tipo «Hardware Description Language» indipendente dalla tecnologia, usato per la modellizzazione, simulazione e sintesi di sistemi digitali.

VHDL

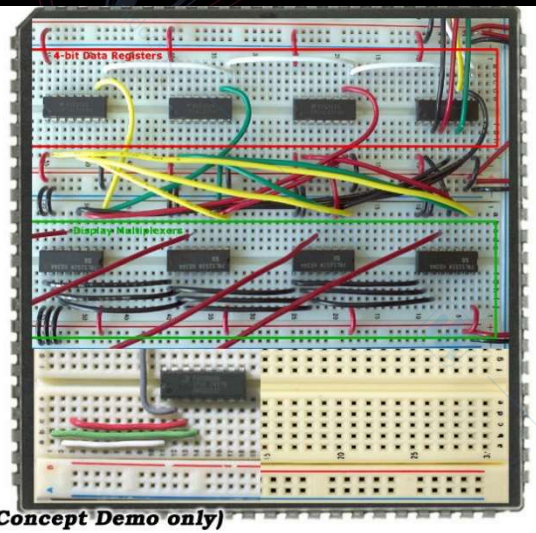
Quando è nato il VHDL?

Il VHDL nasce nel 1987 quando lo studio e la progettazione «a mano» di circuiti digitali non è più sostenibile e si deve cercare una strada alternativa.

Originariamente sviluppato dal Dipartimento della Difesa americano subisce modifiche e standardizzazioni nel corso degli anni.

Oggi è utilizzato in tutto il mondo, specialmente in Europa.

```
5   LIBRARY ieee;
6   USE ieee.std_logic_1164.all;
7
8   ENTITY LabExCG4 IS
9       PORT(   u, v, w, x, y : IN BIT;
10              s       :   IN STD_LOGIC_VECTOR(2 DOWNTO 0);
11              m : OUT BIT);
12  END LabExCG4;
13
14  ARCHITECTURE Behavior OF LabExCG4 IS
15  BEGIN
16  PROCESS(s)
17  BEGIN
18      CASE s is
19          WHEN "000" => m <= u;
20          WHEN "001" => m <= v;
21          WHEN "010" => m <= w;
22          WHEN "011" => m <= x;
23          WHEN "100" => m <= y;
24          WHEN OTHERS => m <= y;
25      END CASE;
26  END PROCESS;
27  END Behavior;
```



(NOT Actual equivalent Circuit - For Concept Demo only)

VHDL

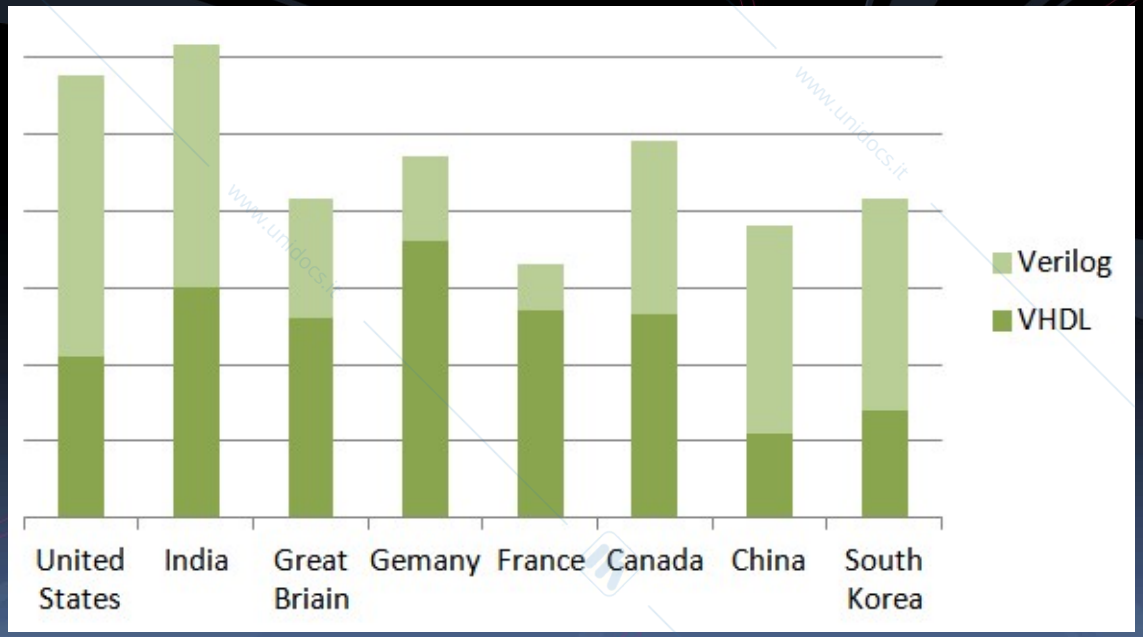
Evoluzione negli anni:

- 1981 Initiated by US DoD to address hardware life-cycle crisis
- 1983-85 Development of baseline language by Intermetrics, IBM and TI
- 1986 All rights transferred to IEEE
- 1987 Publication of IEEE Standard
- 1987 Mil Std 454 requires comprehensive VHDL descriptions to be delivered with ASICs
- 1994 Revised standard (named VHDL 1076-1993)
- 2000 Revised standard (named VHDL 1076 2000, Edition)
- 2002 Revised standard (named VHDL 1076-2002)
- 2007 VHDL Procedural Language Application Interface standard (VHDL 1076c-2007)
- 2009 Revised Standard (named VHDL 1076-2008)

HDLs

Ad oggi esistono due HDL principalmente utilizzati:

- VHDL (Europa)
- Verilog (USA)



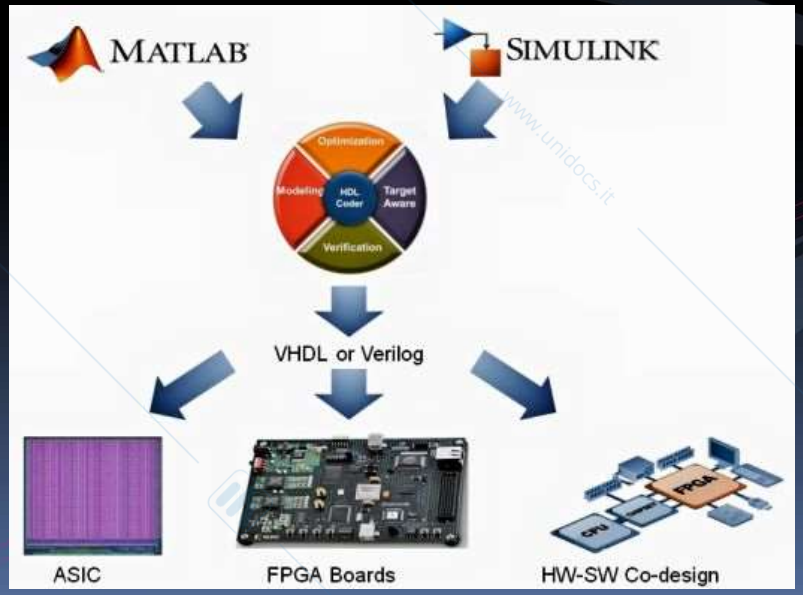
HDLs

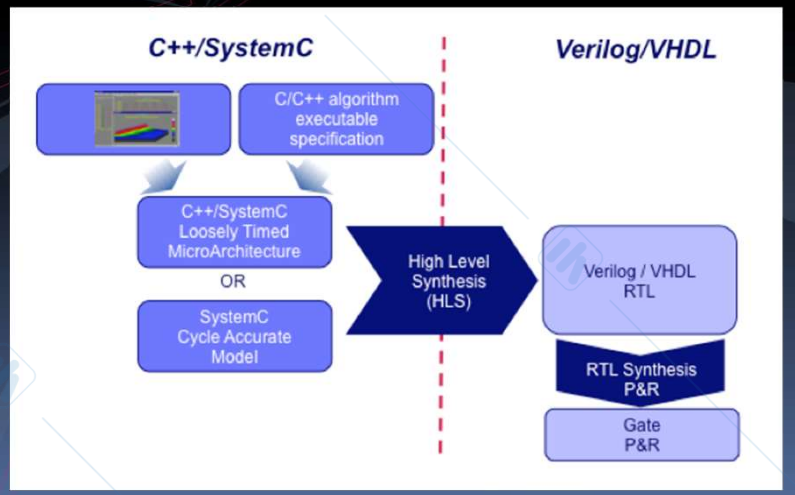
HDLs ARE NOT C, C++, MATLAB ecc

HLS

Negli ultimi anni le applicazioni che utilizzano FPGA stanno crescendo, il time-to-market si riduce progressivamente e l'accesso a programmatori non specializzati è molto difficile.

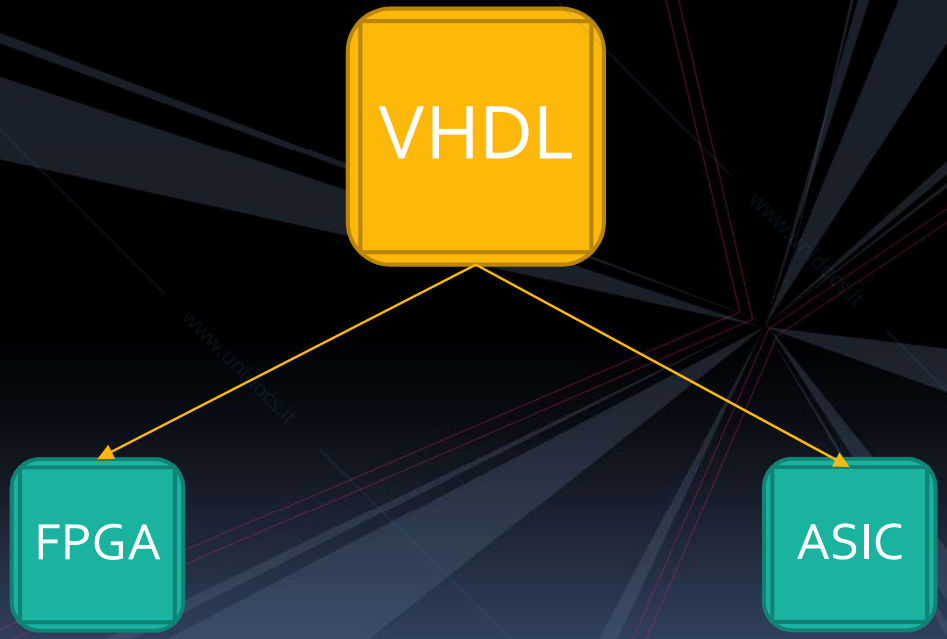
È nata l'esigenza di una metodologia di progettazione dei dispositivi diversa da quella tradizionale che si è concretizzata nei tool «High Level Synthesis».





Target applicativo

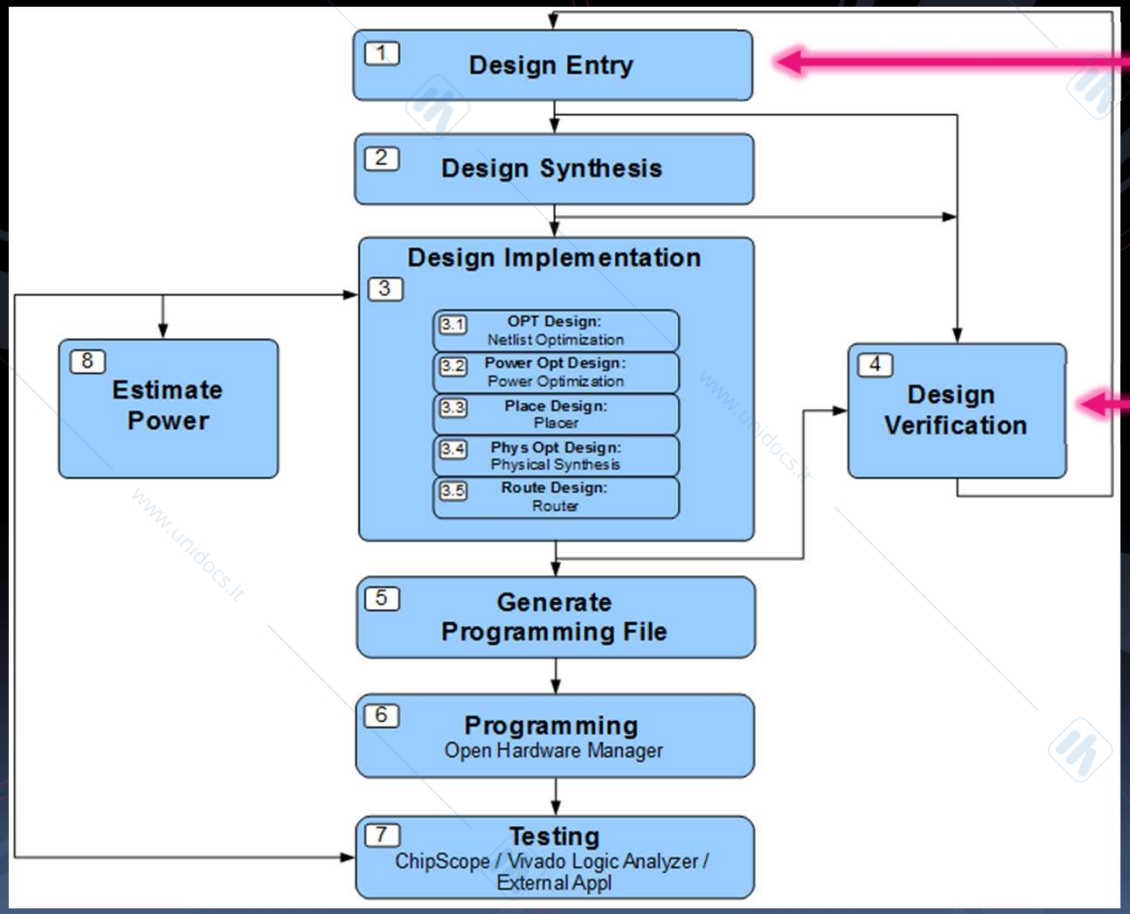
Il VHDL nasce come linguaggio di descrizione Hardware, non è legato a nessuna tipologia di dispositivo o produttore ma è uno standard IEEE.



Design Flow

Device "Independent"

Device Dependent



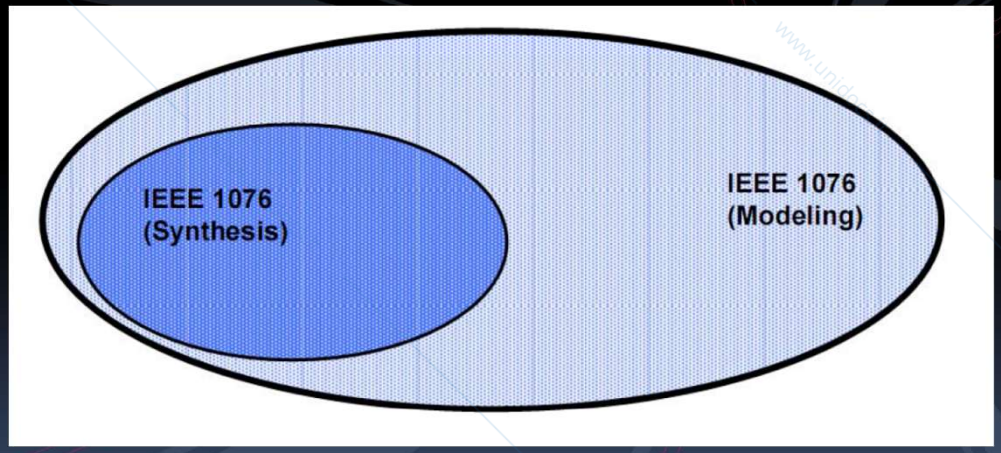
VHDL

VHDL

VHDL Synthesis - Modelization

Il VHDL è un linguaggio a 360 gradi per la progettazione hardware. Gli utilizzi si possono dividere in due gruppi:

- Descrizione Hardware: solitamente questa strada porta alla sintesi ed all'implementazione di un circuito => SINTESI.
- Modellizzazione Hardware: tramite il VHDL si descrive un comportamento di un certo dispositivo => SIMULAZIONE.



ATTENZIONE: non tutto lo standard VHDL può essere sintetizzato, molte parti e costrutti sono riservati alla modellizzazione.

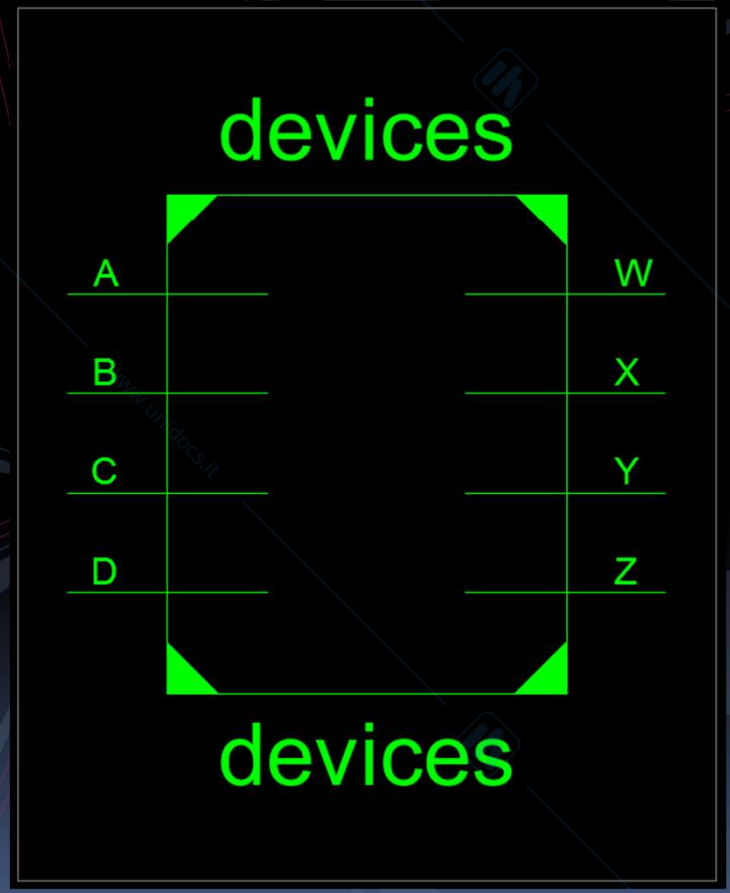
Entity

Ogni design VHDL è composto da «mattoni» fondamentali chiamati ENTITY. Ogni Entity è una black-box con ingressi ed uscite

```
entity my_entity is
  port(
    in_port1 : in std_logic;
    in_port2 : in std_logic;
    out_port1 : out std_logic;
    out_port2 : out std_logic
  );
end my_entity;
```

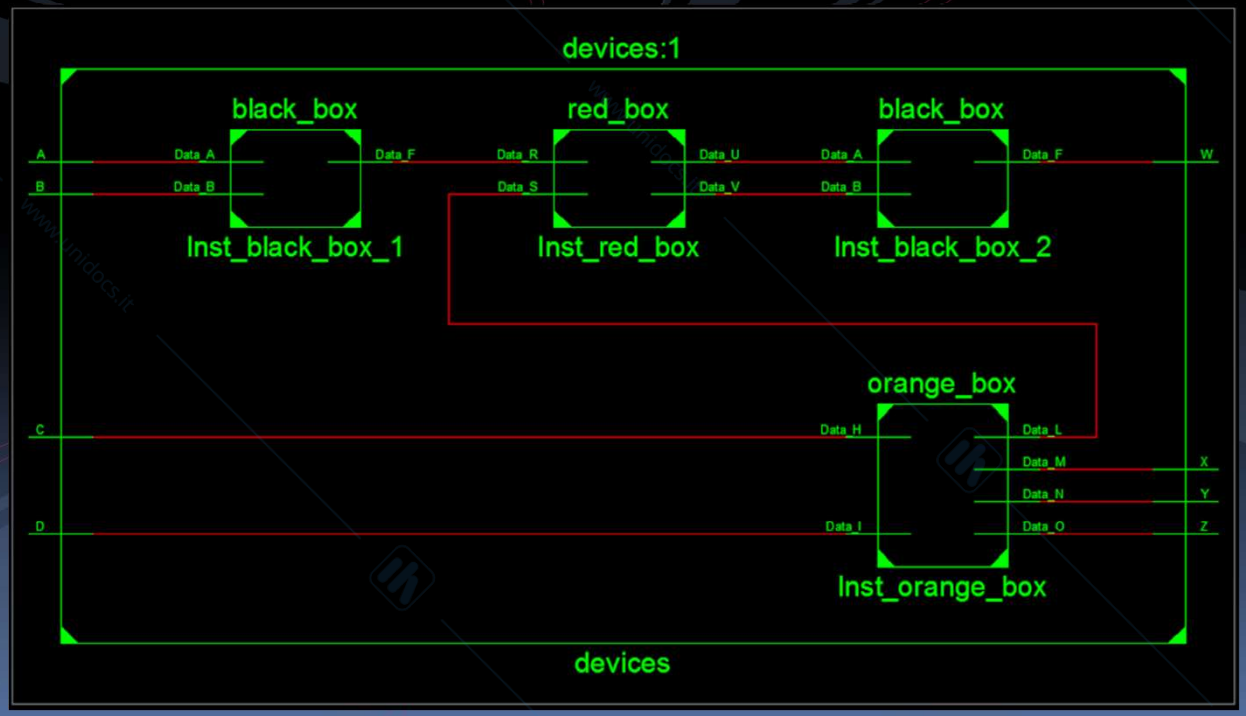
Top

Un design VHDL completo nasce con una Entity principale chiamata «top», la quale comunica direttamente con il mondo esterno tramite ingressi ed uscite specificate nel relativo «port».



Top - Modularity

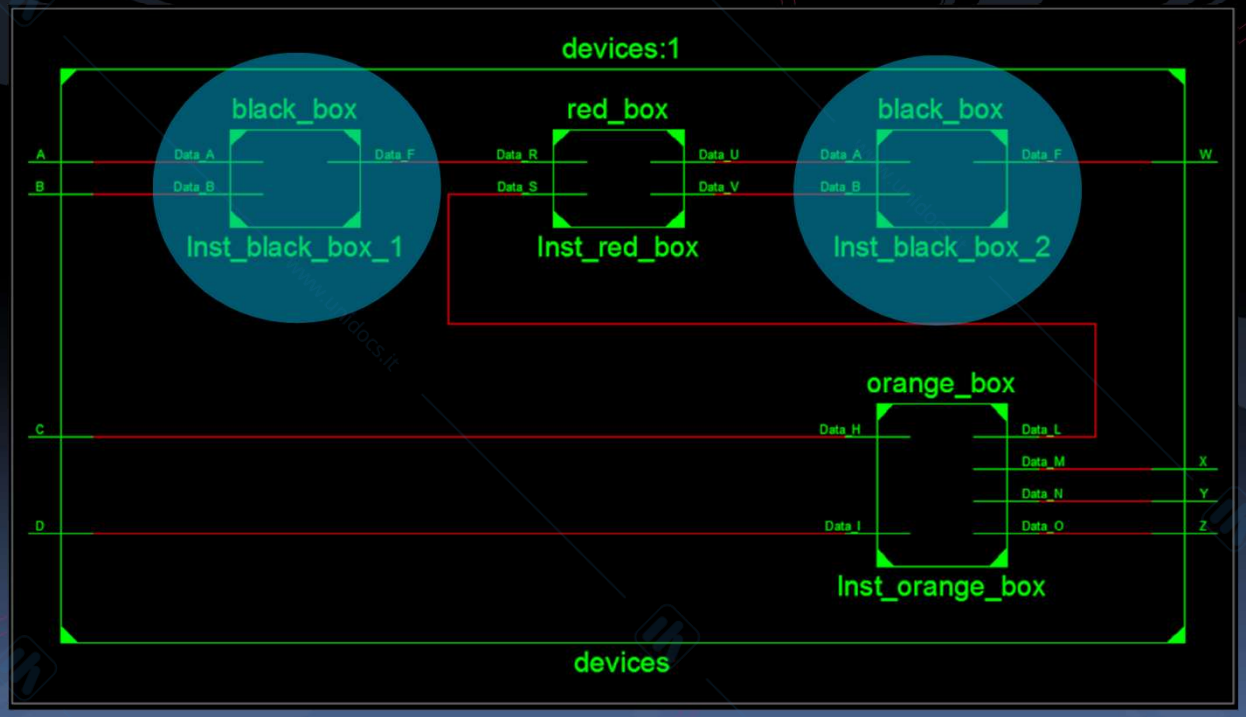
Il VHDL prevede una gerarchia modulare, varie Entity possono essere «instanziate» e collegate tramite ingressi ed uscite, andando a formare il sistema finale. Questo tipo di approccio è molto pratico, velocizza sia la scrittura che il debug.



NOTA: per convenzione le porte d'uscita sono a destra mentre gli ingressi a sinistra

Top - Modularity

Un ulteriore vantaggio della modularità è il riutilizzo delle Entity nel design. Ad esempio in un addizionatore si può riutilizzare il blocco fondamentale Full-Adder fino ad ottenere la larghezza della somma desiderata.



Architecture

Cosa succede all'interno delle «Black-Box»?

La descrizione del comportamento o dei sub-componenti istanziati all'interno delle Entity viene descritto in un'apposita sezione chiamata **«Architecture»**.

```
entity and2 is
  port(
      a : in std_logic;
      b : in std_logic;
      c : out std_logic
  );
end my_entity;

architecture and2_a of and2 is
Begin
  c <= a and b;
end and2_a;
```

Architecture Name

Architecture

Lo standard permette l'utilizzo di più Architecture per Entity.

Si può descrivere in forme diverse lo stesso componente e poi scegliere l'implementazione che si desidera.

Good design rule:

- **Utilizzare solo una architecture per entity**
- **Utilizzare lo stesso file .vhd per architecture e entity**
- **Dare lo stesso nome della entity al file**

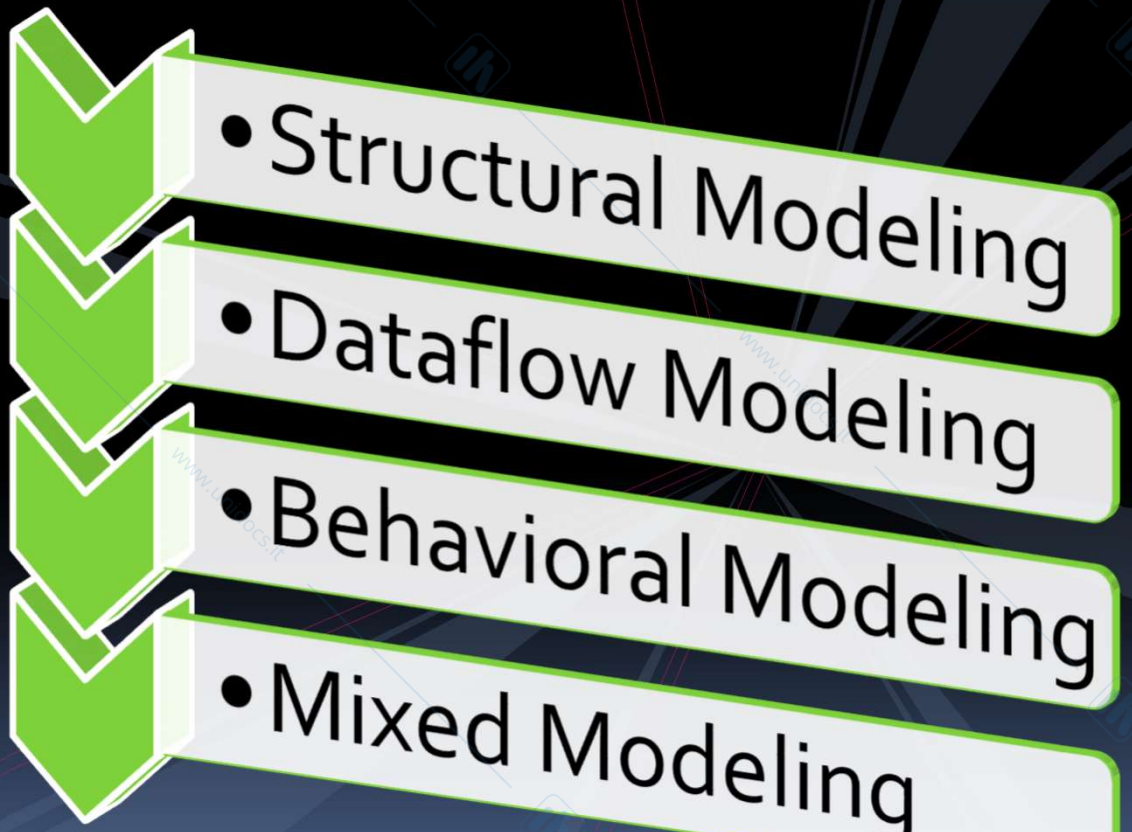
  (e.g. entity = and2 => file = and2.vhd)

Modeling style

- **Structural Modeling:** Implementazione come unione di porte o strutture, questo tipo di approccio è strettamente legato alla tecnologia utilizzata.
- **Dataflow Modeling:** Implementazione come descrizione combinatoria tramite porte logiche di base, "teoricamente" indipendente dalla tecnologia.
- **Behavioral Modeling:** Implementazione come descrizione del comportamento che avrà l'entity. Non riflette direttamente ciò che verrà implementato.
- **Mixed Modeling:** La combinazione di una qualsiasi delle tre precedent.

Levels of Abstraction

Low Abstraction



High Abstraction

Library

- Esistono delle librerie che definiscono i tipi base o il comportamento delle funzioni elementari. Alcune sono praticamente un obbligo in ogni design.

- Tutte le librerie devono essere dichiarate manualmente ad eccezione per una speciale libreria chiamata «standard», questa include delle informazioni base e direttive che vengono lette direttamente dall'analizzatore. Ad esempio il tipo «boolean» è incluso in questa libreria.

Integer

Type Integer: Definito nello «standard» package, da cui derivano tutti gli altri tipi numerici di base. Consente le classiche operazioni con i numeri. Lunghezza massima predefinita nella dichiarazione nel package.

```
type INTEGER is range -2147483647 to 2147483647;
```

$2^{31} - 1$

Se non diversamente esplicitato utilizza **32 bit** di larghezza

```
my_number INTEGER range 0 to 64;
```

Attenzione: non si può fare affidamento sul roll-up di un integer, anche se si utilizza un constraint di lunghezza, il comportamento dalla simulazione all'implementazione può essere diverso.

Natural e Positive

Entrambi sono delle derivazioni **«SUBTYPE»** del tipo integer.

```
subtype NATURAL is INTEGER range 0 to INTEGER'HIGH;
```

```
subtype POSITIVE is INTEGER range 1 to INTEGER'HIGH;
```

Attenzione: sono derivati dal tipo integer, questo non vuol dire che utilizzano 32 bit.

INTEGER'HIGH = 2^31-1 = 2147483647

Library IEEE

Le più famose librerie sono standardizzate direttamente dalla IEEE, chi fornisce gli strumenti di sviluppo «deve» garantire portabilità e insensibilità al dispositivo utilizzato.

```
library IEEE;
  use IEEE.std_logic_1164.all;
  use IEEE.std_logic_textio.all;
  use IEEE.std_logic_arith.all;
  use IEEE.numeric_bit.all;
  use IEEE.numeric_std.all;
  use IEEE.std_logic_signed.all;
  use IEEE.std_logic_unsigned.all;
  use IEEE.math_real.all;
  use IEEE.math_complex.all;
```

Packages

https://www.csee.umbc.edu/portal/help/VHDL/stdpkg.html

Std_ulogic

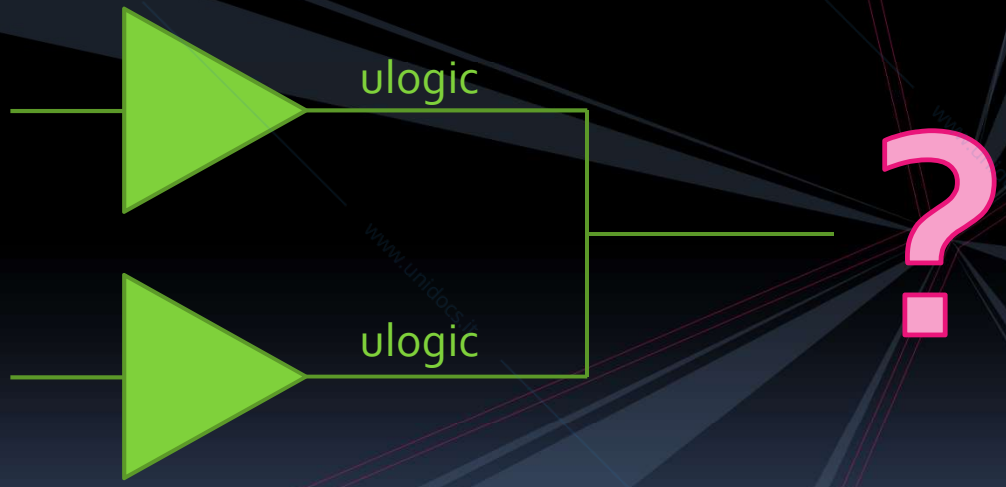
Std_ulogic definisce i possibili stati di un buffer in uscita. È definito nel package std_logic_1164.

```
------------------------------------------------------
-- 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
                    );
```

Std_ulogic

Cosa succede se due buffer hanno le uscite collegate insieme?

Il tipo std_ulogic non sa cosa fare, per questo motivo si dice «unresolved»



Std_logic

Per ovviare al problema si è creato il tipo std_logic che è uguale al precedente solo che è «risolto».

Risolvere vuol dire definire per ogni combinazione di stati che valore assumerà l'uscita

```
SUBTYPE std_logic IS resolved std_ulogic;
```

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

Std_logic_vector

Std_logic_vector è la naturale estensione di std_logic sotto forma di vettore

```
mio_byte_a STD_LOGIC_VECTOR(7 DOWNTO 0);
mio_byte_a STD_LOGIC_VECTOR(14 DOWNTO 8);
mio_byte_a STD_LOGIC_VECTOR(9 DOWNTO 2);
mio_byte_a STD_LOGIC_VECTOR(0 DOWNTO 7);
mio_byte_a STD_LOGIC_VECTOR(4 DOWNTO -3);

mio_byte_b STD_LOGIC_VECTOR(0 TO 7);
mio_byte_b STD_LOGIC_VECTOR(8 TO 14);
mio_byte_b STD_LOGIC_VECTOR(2 TO 9);
mio_byte_b STD_LOGIC_VECTOR(7 TO 0);
mio_byte_b STD_LOGIC_VECTOR(-3 TO 4);
```

Unsigned - Signed

Quindi il VHDL è limitato a rappresentare un massimo numero di $2^{31} - 1$?

NO!

```
--============================================================
-- Numeric array type definitions
--============================================================

type UNSIGNED is array ( NATURAL range <> ) of STD_LOGIC;
type SIGNED is array ( NATURAL range <> ) of STD_LOGIC;
```

Si può utilizzare la libreria numeric_std dove vengono definiti i tipi SIGNED e UNSIGNED con tutte le loro operazioni.
Questi due tipi sono visti come array di std_logic, praticamente non c'è limite alla dimensione che possono assumere.

Attenzione: Al contrario degli integer e derivati, con signed ed unsigned possiamo fare affidamento sul Roll-Up

Conversion

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
```