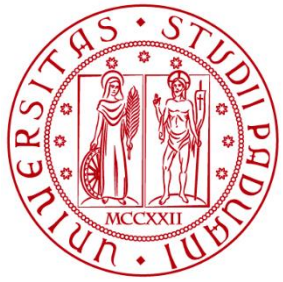
DEI
DIPARTIMENTO DI
INGEGNERIA DELL'INFORMAZIONE



UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Sistemi Digitali

Introduzione al VHDL

Marta Bagatin, marta.bagatin@unipd.it

Corso di Laurea in Ingegneria dell'Informazione

Anno accademico 2022-2023

Scopo della lezione

- Introdurre i concetti fondamentali alla base dei **linguaggi di descrizione dell'hardware**
- Introdurre le **parti principali di un codice VHDL**
  - Entity
  - Architecture
  - Librerie e package
  - Componenti
- Studiare alcuni **esempi di codici VHDL**
  - Descrizione con architettura Structural
  - Descrizione con architettura Dataflow
  - Descrizione con architettura Behavioral
  - Simulazione con Testbench

Cos’è il VHDL e come lo studieremo

HDL

- Gli strumenti **Computer-Aided Design (CAD)** sono essenziali per la progettazione dei sistemi digitali con miliardi di transistor
  - Progettazione/sintesi manuali sarebbero improponibili (tempo, probabilità di errore, costo, …)
- **HDL** (**H**ardware **D**escription **L**anguage): linguaggio di descrizione dell'hardware che consente descrizioni a diversi livelli di astrazione che possono essere simulate o sintetizzate (i.e. tradotte in hardware)
- **I più usati sono VHDL e Verilog**, entrambi supportati dall'IEEE (Institute of Electrical and Electronics Engineers)
  - Sono standardizzati, quindi possono essere scambiati tra diversi strumenti CAD

VHDL

- Nel linguaggi HDL le **dichiarazioni sono concorrenti, ovvero sono attive in parallelo, tutte allo stesso tempo** (le porte logiche in un circuito lavorano in parallelo)
- NON è un linguaggio di programmazione (es: Java, C++, Python)
  - Un linguaggio di programmazione viene tradotto in una sequenza di istruzioni, che viene eseguita in modo sequenziale (cioè una dopo l'altra) da un microprocessore
  - Un linguaggio HDL viene tradotto in un circuito digitale (sintesi logica)
- **VHDL: Very high speed integrated circuits Hardware Description Language**: nacque nel 1987 da un progetto del Dipartimento della difesa USA e venne riconosciuto come standard IEEE (Institute of Electrical and Electronics Engineers)
  - Sono state apportate diverse revisioni nel corso degli anni

VHDL: scopi di utilizzo

- **Documentazione** di un design
- **Simulazione** del comportamento di un design: studio dell'evoluzione dei segnali per valutare il corretto funzionamento del circuito
  - Applica una serie di segnali in ingresso al circuito e verifica le uscite
- **Sintesi automatica** di un design: passaggio da una descrizione comportamentale ad una descrizione a livello di porte logiche, i.e. traduzione del listato VHDL in una rete (netlist) di celle (logic cells) fisicamente realizzabili, che può essere a sua volta usata per produrre un circuito reale
  - Fornita da appositi programmi che si appoggiano a librerie dove sono descritte le porte logiche disponibili (fornite dal venditore)
- In questo corso NON ci occuperemo di sintesi di circuiti

VHDL sintetizzable e non sintetizzabile

- La sintesi è applicabile solo ad un sottoinsieme dei possibili comandi (VHDL sintetizzabile). **Non tutto ciò che è scritto in VHDL è sintetizzabile!**
- Al contrario, se la descrizione dell'hardware è pensata solo per essere simulata, può essere usato l'intero insieme dei comandi VHDL

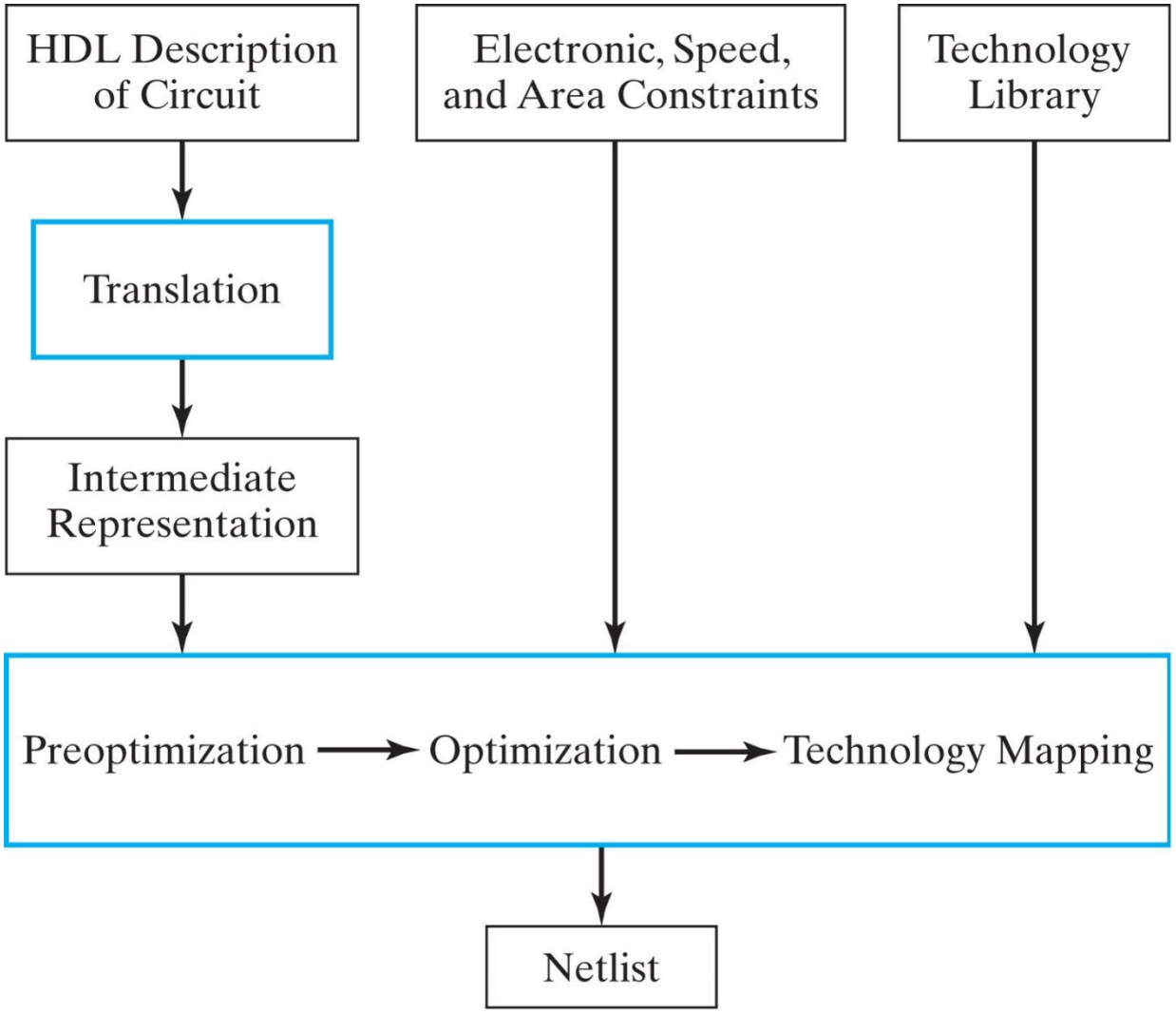
Livelli di astrazione

- Il VHDL permette la descrizione di un circuito a **diversi livelli di astrazione**
  - **Structural**: descrive il sistema in forma di componenti interconnessi, è equivalente ad uno schematico → basso livello di astrazione
  - **Behavioral**: descrive il comportamento e la funzionalità del sistema, a prescindere dall'effettiva implementazione a livello di componenti → alto livello di astrazione
  - **Dataflow**: livello di astrazione intermedio tra structural e behavioral, descrive il sistema in forma di flusso di dati

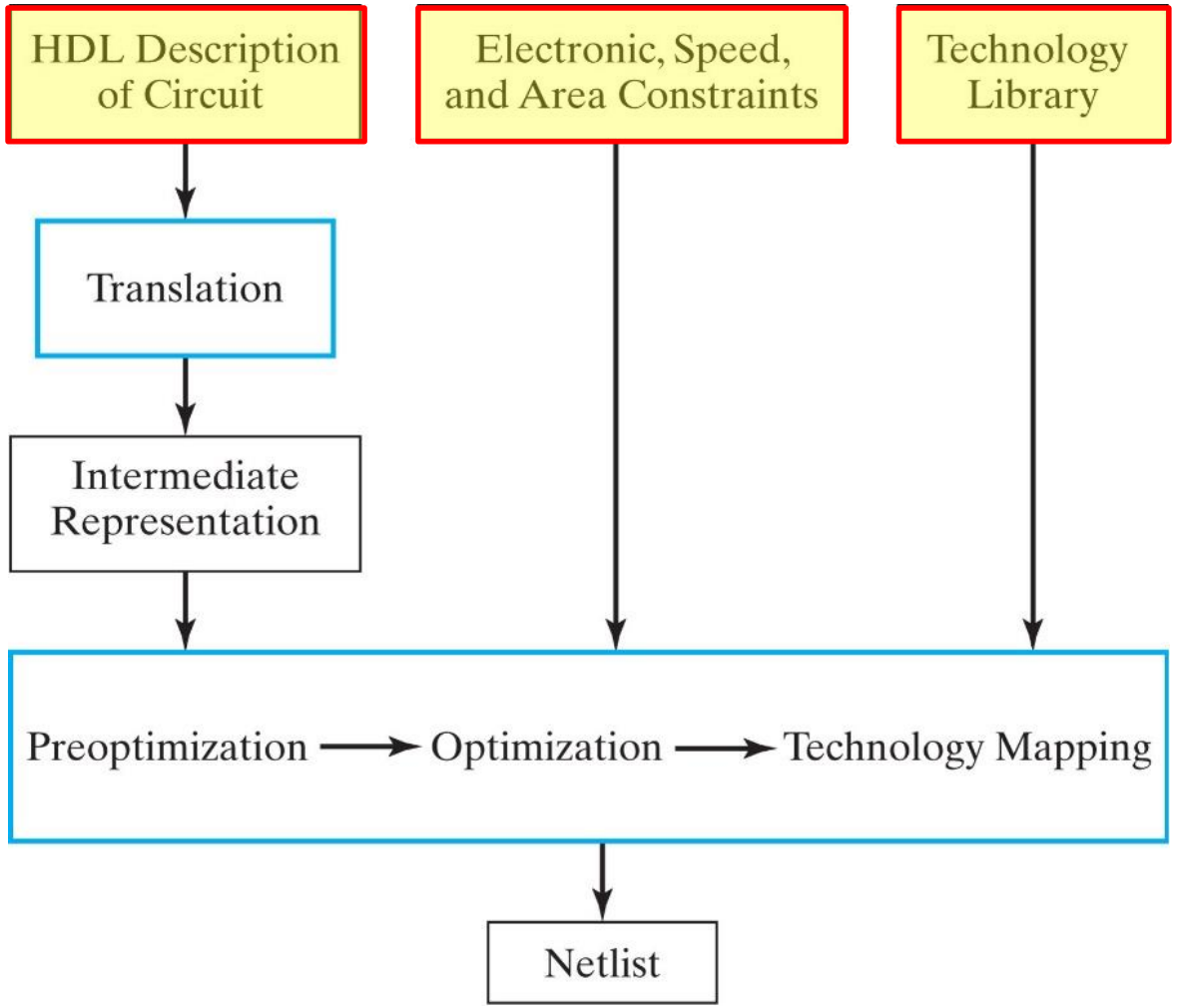
Esempio: Livelli di astrazione

- Codice VHDL di un circuito che somma due numeri A e B
- Descrizione **structural**: presenta al sintetizzatore una certa struttura, ovvero un insieme di blocchi interconnessi in cui viene suddivisa la funzionalità da realizzare
  - Il progettista ha il controllo sull'architettura del circuito e può suggerire al sintetizzatore gli schemi più performanti.
- Descrizione **behavioral**: dice al sintetizzatore di sommare A e B, senza specificare la struttura circuitale: `F <= A + B;`
  - Il progettista lascia al sintetizzatore la scelta di quale circuito usare

Flusso per la sintesi di un circuito

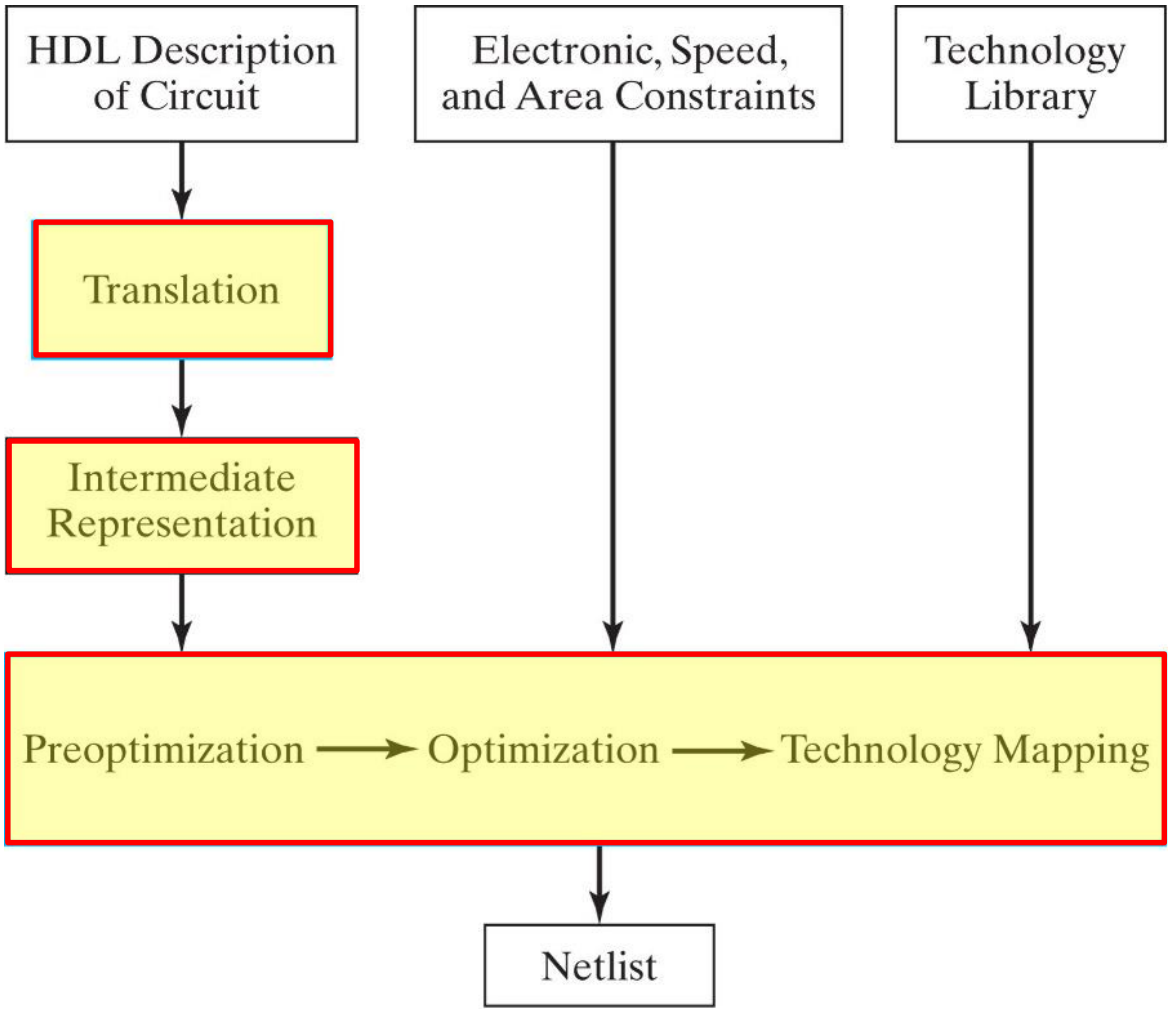


Flusso per la sintesi di un circuito



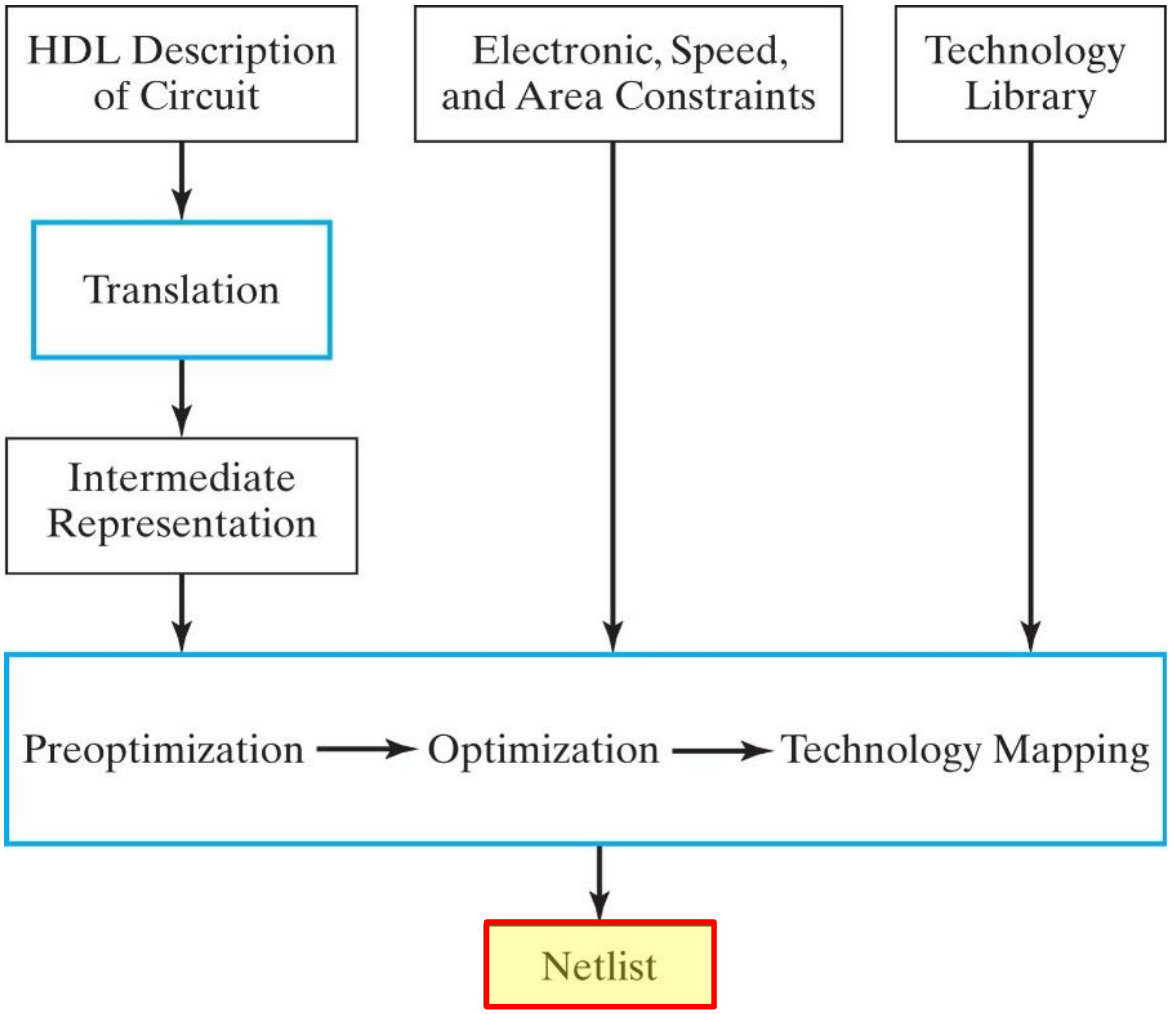
- **Descrizione del circuito** (codice HDL)
- **Vincoli/specifiche** di progetto (es: area, velocità, potenza, …)
- Eventuali informazioni addizionali contenute in **librerie tecnologiche** che descrivono i blocchi primitivi (es: ritardi, area, …)

Flusso per la sintesi di un circuito



- Traduzione in una **forma intermedia (RTL, Register Transfer Level)**: porte logiche ed elementi di memoria generici connessi tra loro, non legati ad una specifica tecnologia
- **Ottimizzazione** per soddisfare i vincoli e
- **Mappatura tecnologica** sostituisce i blocchi primitivi con le porte effettivamente disponibili nella libreria tecnologica
- Processo iterativo

Flusso per la sintesi di un circuito



- Il risultato è una **netlist** (**schema del circuito**): verrà usata dagli strumenti per la progettazione fisica circuitale per realizzare il layout finale per la fabbricazione del circuito integrato
- Nel caso di implementazione in circuiti programmabili (FPGA), viene usata per produrre il file che specifica come programmare la logica all'interno del dispositivo

Elementi di base del VHDL

Caratteristiche generali del VHDL

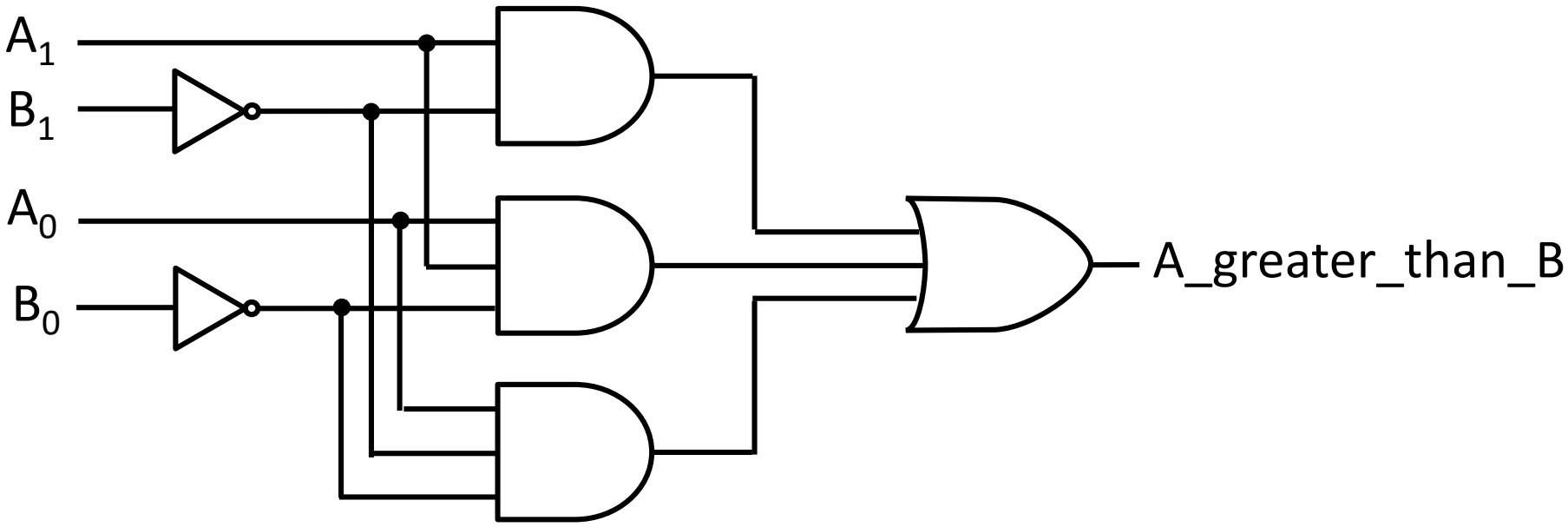
- E' un linguaggio **fortemente tipizzato** (**strongly-typed language**): dati di tipi diversi non possono essere assegnati gli uni agli altri senza usare funzioni di conversione
- E' **case insensitive** (non distingue le lettere minuscole dalle lettere maiuscole)
- Ogni riga può essere suddivisa su più linee: spazi, tabulazioni e ritorni a capo sono ignorati
- E' importante tenere sempre in mente l'**hardware** del circuito che si vuole descrivere!

Struttura del codice VHDL

- Primo esempio di codice VHDL: comparatore a due bit
- Illustrazione dei diversi tipi di descrizione di un circuito e studio degli elementi di base del VHDL
  - Entity
  - Architecture
  - Tipi di dati
  - Component
  - Signal …
- Partiremo dalla descrizione strutturale e poi vedremo livelli di astrazione via via più elevati, infine vedremo la simulazione tramite testbench

Comparatore a 2 bit

Schema del circuito



Due ingressi a 2 bit: A: A_1 A_0, B: B_1 B_0 (0 è il LSB, 1 è il MSB)

Una uscita a 1 bit: A_greater_than_B vale ‘1’ se A > B

Comparatore a 2 bit: VHDL

```
library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all;
```

Librerie e pacchetti

```
entity comparator_greater_than_structural is
  port (A: in std_logic_vector(1 downto 0);
    B: in std_logic_vector(1  downto 0);
    A_greater_than_B: out std_logic);
end comparator_greater_than_structural;
```

Entity

```
architecture structural of comparator_greater_than_structural is

 component NOT1
  port(in1: in std_logic;
    out1: out std_logic);
 end component;
 component AND2
  port(in1, in2: in std_logic;
   out1: out std_logic);
 end component;
 component AND3
  port(in1, in2, in3: in std_logic;
   out1: out std_logic);
 end component;
 component OR3
  port(in1, in2, in3 : in std_logic;
    out1: out std_logic);
 end component;
signal B1_n, B0_n, and0_out, and1_out, and2_out: std_logic;
begin
 inv_0: NOT1 port map (in1 => B(0), out1 => B0_n);
 inv_1: NOT1 port map (B(1), B1_n);
 and_0: AND2 port map (A(1), B1_n, and0_out);
 and_1: AND3 port map (A(1), A(0), B0_n, and1_out);
 and_2: AND3 port map (A(0), B1_n, B0_n, and2_out);
 or0: OR3 port map (and0_out,and1_out,and2_out, A_greater_than_B);
end structural;
```

Architecture

Entity

- Tutti i blocchi devono avere **una entity**
- E’ la **targa identificativa del blocco**, gli dà il nome e ne definisce l’interfaccia con l’esterno (quante, quali e di che tipo sono le **porte di ingresso e uscita**)
- Entity non dice cosa c’è dentro al blocco, è analoga ad un simbolo nello schema di un circuito

Entity

Sintassi per la dichiarazione di entity

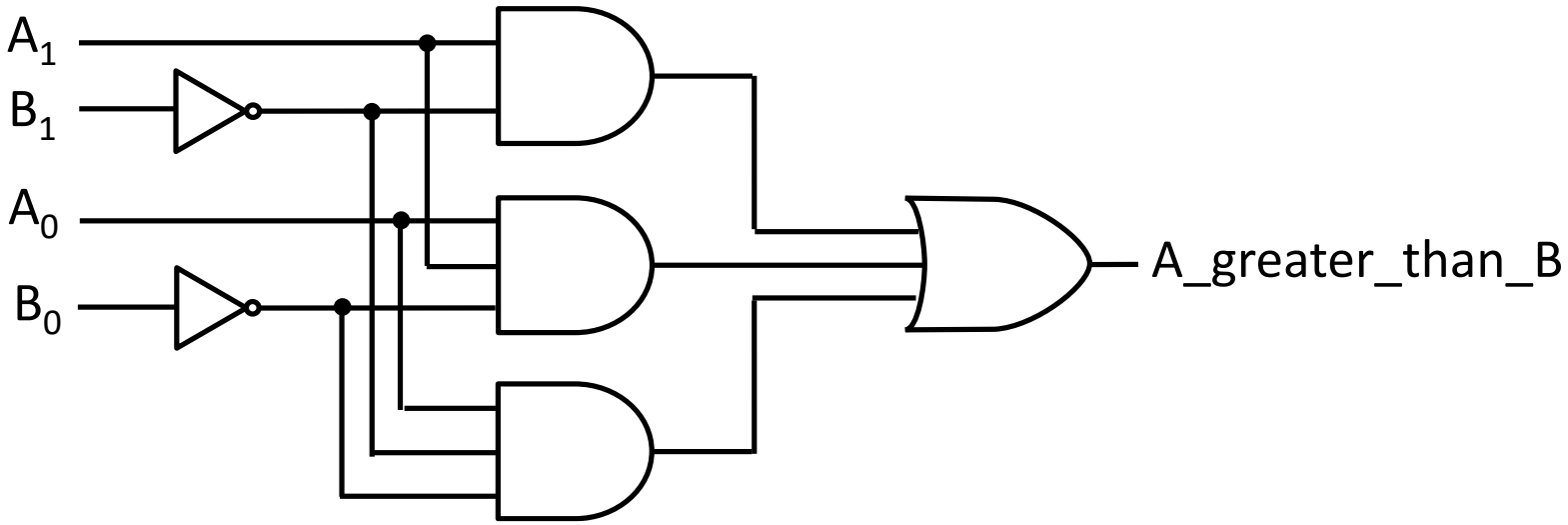
```
entity nameEntity is
--Dichiarazione ingressi (in) e uscite (out):
     port( namePort1, namePort2, …: in typeName;
           namePort11,namePort12,…: out typeName);
--“;” va dopo chiusura parentesi
end nameEntity;
```

Note

- Le **parole riservate** vanno obbligatoriamente nelle posizioni indicate
- Eventuali commenti possono essere scritti dopo --
- in/ out è la modalità (o direzione) delle porte
- typeName è il tipo di dato

Comparatore a 2 bit: Entity

```
entity comparator_greater_than_structural is
  port (A: in std_logic_vector(1 downto 0);
    B: in std_logic_vector(1  downto 0);
    A_greater_than_B: out std_logic);
end comparator_greater_than_structural;
```



Terminologia

- **Assegnazione: X <= Y**: assegno Y a X

  <= è da intendersi come una freccia che va da destra a sinistra
- Si dice che lo statement X <= Y rappresenta un driver per il segnale X
  - Ogni volta che Y cambia valore, lo cambia di conseguenza anche X
- **Driver**: è la sorgente di un segnale

Modalità

- La modalità indica **la direzione dei segnali nella entity** e indica anche se la porta può essere letta o pilotata dal blocco
  - **in** (modalità usata per gli **ingressi**): dato fluisce dentro alla entity. Il valore può essere letto ma non assegnato dalla entity, cioè possiamo solo leggere ma non pilotare quella porta. Il driver sta fuori dalla entity
  - **out** (modalità usata per le **uscite**): dato fluisce fuori dalla entity. Il valore può essere assegnato (pilotato) ma non letto dalla entity. Il driver sta dentro alla entity
  - **Inout**
  - **Buffer**

  } Non li vedremo!

Tipi di dati: std_logic

- Il tipo **std_logic** fa parte della libreria IEEE std_logic_1164
- E' usato per descrivere un **segnale logico da 1 bit** e può assumere i seguenti valori

Di interesse per i nostri scopi

  - '0': 0 logico
  - '1': 1 logico
  - 'X': valore non noto o non definito (può essere '0' logico o '1' logico)
  - '-': don't care (lascia libero il sintetizzatore di ottimizzare la funzione)
  - 'Z': alta impedenza (se si tratta di un output, significa che è sconnesso, flottante)
  - 'U': non inizializzato (non ha un valore iniziale), usato per simulazioni
  - …

- Viene indicato **tra apici singoli**, per es. '1'

Tipi di dati: std_logic_vector

- **Std_logic_vector** è usato per descrivere un segnale multi-bit, cioè una sequenza di **n bit** di tipo **std_logic**
- Gli indici sono indicati in ordine crescente o decrescente con le parole riservate **downto** / **to**, il cui uso determina l'indice del MSB

  Sintassi per una sequenza di n bit:

```
std_logic_vector (n-1 downto 0)   -- n-1 è il MSB
std_logic_vector (0 to n-1)       -- 0 è il MSB
```

- Per estrarre un bit da un vettore si usa la notazione nomeVettore(indiceBit): x(2) indica il bit con indice 2 nel vettore x
- Il valore viene indicato tra virgolette doppie, per es. "010 "

Attenzione a non mescolare nello stesso codice vettori definiti con downto e to!

Altri tipi di dati

- **Bit**: **1 bit**, può assumere solo **'0' e '1' logico**
- **Boolean**: **1 bit**, può assumere solo valore **'TRUE' e 'FALSE'**
  - VHDL-2008 converte automaticamente il tipo std_logic in boolean dove si aspetta un valore booleano (es: nella condizione di un if)
- **Integer**: rappresenta **numeri interi a 32 bit**, da $-(2^{31} - 1)$ a $2^{31} - 1$
  - Tipicamente è usato per specificare parametri come la lunghezza di un vettore o per l'indice in un ciclo for
  - Viene indicato in **base decimale** (cifre da 0 a 9) e **senza virgolette** (es: 100)
- Questi tipi di dati sono diversi da std_logic e std_logic_vector!

Non useremo questi tipi di dati per rappresentare segnali logici nei sistemi digitali, ma useremo sempre std_logic e std_logic_vector!

Architecture

- Dopo entity, **architecture** è la seconda fondamentale unità del codice VHDL
- E' una **particolare descrizione del blocco corrispondente ad una entity**: specifica cosa c'è dentro al circuito, come si comporta e come opera internamente
- Una **entity può essere mappata in diverse architecture** nello stesso design. Quindi un blocco VHDL deve avere un'unica entity, ma può avere architecture multiple

Architecture

- Tre principali tipi, in ordine di astrazione crescente
  - **Structural**: fa uso di **componenti**, è equivalente allo schematico di un circuito, è usata per strutture gerarchiche
  - **Dataflow**: descrive il **flusso di segnali** attraverso il circuito, è equivalente ad un insieme di equazioni booleane
  - **Behavioral**: si limita a **descrivere il comportamento** senza specificare la struttura, è usata per circuiti complessi quando non è necessario scendere nei dettagli dell'implentazione
- Combinazioni tra queste

Architecture

Sintassi per la dichiarazione di architecture

```
architecture nameArchitecture of nameEntity is
 -- eventuali dichiarazioni di componenti:
     component nameComponent
      port ( in1: in typeName;
             out1: out typeName);
     end component;
-- eventuali dichiarazioni di segnali e costanti:
     signal nameSignal1, nameSignal2, …: typeName;
     constant nameConstant1: typeName := constant1Value;
-- il corpo di architecture inizia dopo begin:
     begin
      ...
end nameArchitecture;
```

Component

- I componenti sono usati nelle **descrizioni strutturali**: possono essere connessi tra loro, anche in strutture annidate (gerarchie) e riusati più volte nel modello
- I componenti possono essere
  - **descritti dall'utente**, all'interno dello stesso file o in un altro file (ogni componente deve avere una propria entity e almeno una architecture)
  - oppure presi da **librerie predefinite** (es: fornite dai costruttori FPGA)
- Creare un'**istanza** di un componente significa inserire un blocco (che è già stato definito altrove) connettendone le porte di ingresso e uscita (definiti nella entity) ai segnali desiderati
- Si ottiene quindi una descrizione strutturale a livello di blocchi equivalente a quella che si avrebbe in uno **schematico**

Signal

- Mentre le porte di ingresso e uscita sono definite nella entity e comunicano con l'esterno, i **segnali interni sono definiti nella architecture**
- I segnali interni sono usati per le **connessioni interne al circuito,** per esempio per collegare nodi interni di un circuito o connettere tra loro blocchi diversi per formare un design più grande
- Devono essere definiti prima della parola chiave 'begin' dentro ad architecture. E' bene usare dei nomi descrittivi per i segnali
- Sintassi:

```
signal nameSignal: signalType [:= signalValue];
```

Constant

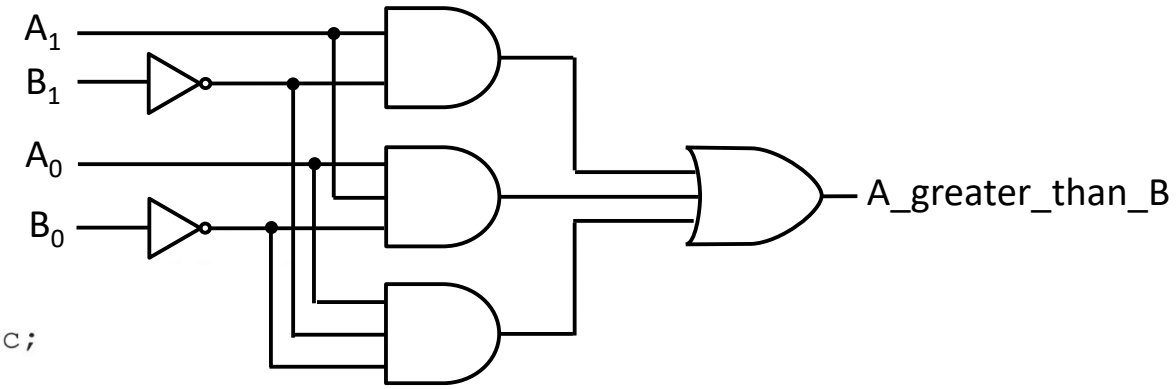
- Le costanti rappresentano dei **valori fissi** nel tempo a cui viene assegnato un nome e sono caratterizzate da un tipo di dato specifico
- Sono usate per **facilitare l'aggiornamento/modifiche del codice** e migliorarne la leggibilità
- Devono essere definite prima della parola chiave 'begin' dentro ad architecture. E' bene usare dei nomi simbolici per le costanti
- Sintassi:

```
constant nameConstant1: typeName := constant1Value;
```

Comparatore a 2 bit: Architecture Structural

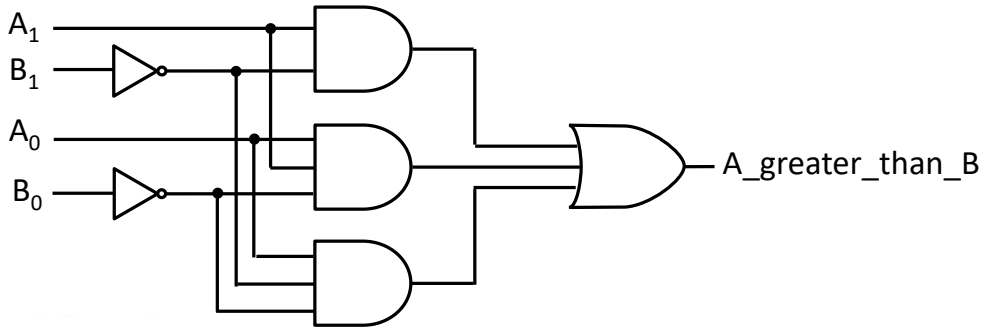
```
architecture structural of comparator_greater_than_structural is

 component NOT1
  port(in1: in std_logic;
    out1: out std_logic);
 end component;
 component AND2
  port(in1, in2: in std_logic;
   out1: out std_logic);
 end component;
 component AND3
  port(in1, in2, in3: in std_logic;
   out1: out std_logic);
 end component;
 component OR3
  port(in1, in2, in3 : in std_logic;
    out1: out std_logic);
 end component;
signal B1_n, B0_n, and0_out, and1_out, and2_out: std_logic;
begin
 inv_0: NOT1 port map (in1 => B(0), out1 => B0_n);
 inv_1: NOT1 port map (B(1), B1_n);
 and_0: AND2 port map (A(1), B1_n, and0_out);
 and_1: AND3 port map (A(1), A(0), B0_n, and1_out);
 and_2: AND3 port map (A(0), B1_n, B0_n, and2_out);
 or0: OR3 port map (and0_out,and1_out,and2_out, A_greater_than_B);
end structural;
```



Comparatore a 2 bit: Architecture Structural

```
architecture structural of comparator_greater_than_structural is
 component NOT1
  port(in1: in std_logic;
    out1: out std_logic);
 end component;
 component AND2
  port(in1, in2: in std_logic;
   out1: out std_logic);
 end component;
 component AND3
  port(in1, in2, in3: in std_logic;
   out1: out std_logic);
 end component;
 component OR3
  port(in1, in2, in3 : in std_logic;
    out1: out std_logic);
 end component;
signal B1_n, B0_n, and0_out, and1_out, and2_out: std_logic;
begin
 inv_0: NOT1 port map (in1 => B(0), out1 => B0_n);
 inv_1: NOT1 port map (B(1), B1_n);
 and_0: AND2 port map (A(1), B1_n, and0_out);
 and_1: AND3 port map (A(1), A(0), B0_n, and1_out);
 and_2: AND3 port map (A(0), B1_n, B0_n, and2_out);
 or0: OR3 port map (and0_out,and1_out,and2_out, A_greater_than_B);
end structural;
```

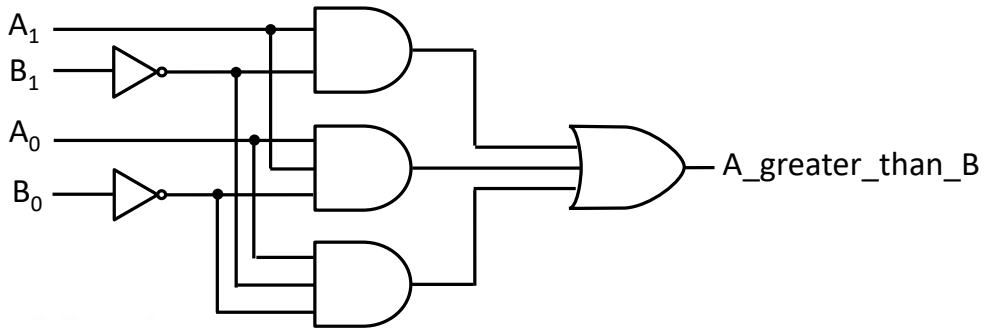


Dichiarazione dei componenti (porte logiche) che saranno poi istanziati nel design: inverter, AND a 2 ingressi, OR a 2-3 ingressi

Comparatore a 2 bit: Architecture Structural

```
architecture structural of comparator_greater_than_structural is

 component NOT1
  port(in1: in std_logic;
    out1: out std_logic);
 end component;
 component AND2
  port(in1, in2: in std_logic;
   out1: out std_logic);
 end component;
 component AND3
  port(in1, in2, in3: in std_logic;
   out1: out std_logic);
 end component;
 component OR3
  port(in1, in2, in3 : in std_logic;
    out1: out std_logic);
 end component;
 signal B1_n, B0_n, and0_out, and1_out, and2_out: std_logic;
begin
 inv_0: NOT1 port map (in1 => B(0), out1 => B0_n);
 inv_1: NOT1 port map (B(1), B1_n);
 and_0: AND2 port map (A(1), B1_n, and0_out);
 and_1: AND3 port map (A(1), A(0), B0_n, and1_out);
 and_2: AND3 port map (A(0), B1_n, B0_n, and2_out);
 or0: OR3 port map (and0_out,and1_out,and2_out, A_greater_than_B);
end structural;
```

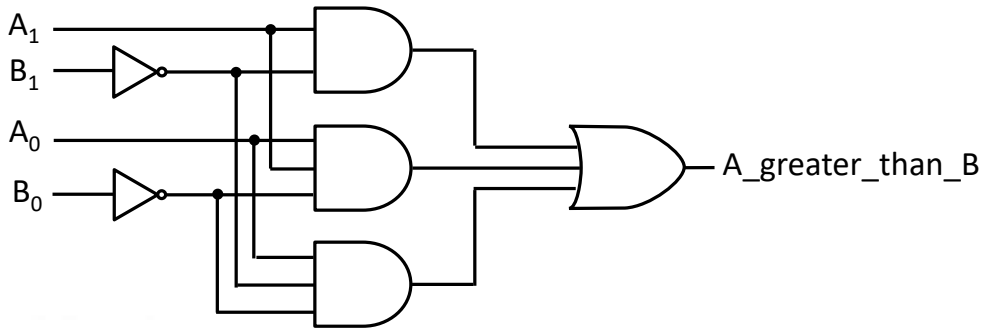


Dichiarazione dei segnali interni, cioè i nodi interni del circuito (uscite dei 2 inverter, uscite delle 3 porte AND)

Comparatore a 2 bit: Architecture Structural

```
architecture structural of comparator_greater_than_structural is

 component NOT1
  port(in1: in std_logic;
    out1: out std_logic);
 end component;
 component AND2
  port(in1, in2: in std_logic;
   out1: out std_logic);
 end component;
 component AND3
  port(in1, in2, in3: in std_logic;
   out1: out std_logic);
 end component;
 component OR3
  port(in1, in2, in3 : in std_logic;
    out1: out std_logic);
 end component;
signal B1_n, B0_n, and0_out, and1_out, and2_out: std_logic;
begin
 inv_0: NOT1 port map (in1 => B(0), out1 => B0_n);
 inv_1: NOT1 port map (B(1), B1_n);
 and_0: AND2 port map (A(1), B1_n, and0_out);
 and_1: AND3 port map (A(1), A(0), B0_n, and1_out);
 and_2: AND3 port map (A(0), B1_n, B0_n, and2_out);
 or0: OR3 port map (and0_out,and1_out,and2_out, A_greater_than_B);
end structural;
```



Concurrent statements: sono tutte valutate in contemporanea, l'ordine è irrilevante!

Istanziazione del componente e port map: i segnali sono assegnati alle porte di ingresso e uscita del componente

Librerie e pacchetti

- Le librerie sono delle collezioni di package, che a loro volta contengono codice VHDL (e.g. componenti, costanti) che si vuole riutilizzare in diversi progetti
- Ad esempio per le funzioni aritmetiche ci sono dei package standard IEEE, ma si usano spesso anche i package creati da Synopsys, prima della standardizzazione IEEE
  - A seconda del package usato, cambia il modo con cui i dati vengono trattati nelle operazioni aritmetiche. Per es: std_logic_vector può venire considerato come array di numeri con o senza segno

Package Synopsys	Package IEEE	Caratteristiche
`Std_logic_arith`	`Numeric_std`	Sono richieste dichiarazioni esplicite o specifiche conversioni
`Std_logic_signed`	`Numeric_std_signed`	Vettori sono considerati come numeri con segno (rappresentati con notazione complemento a 2)
`Std_logic_unsigned`	`Numeric_std_unsigned`	Vettori sono considerati come numeri senza segno

Librerie e pacchetti

Sintassi per utilizzare librerie e package all'interno di un design

```
library nameLibrary1, nameLibrary2, …;
use nameLibrary.namePackage.all;
```

Note

- `.all` permette di utilizzare tutto quello che è definito in quel package
- La dichiarazione di librerie e package da utilizzare va **ripetuta** prima di OGNI entity !

Comparatore a 2 bit: VHDL

```
library ieee, lcdf_vhdl;
use ieee.std_logic_1164.all, lcdf_vhdl.func_prims.all;
```

- Dichiarazione delle librerie e dei pacchetti usati nel design
  - Il package `std_logic_1164`, appartenente alla libreria `IEEE`, contiene i tipi di dati `std_logic` e `std_logic_vector`
  - Il package `func_prims`, appartenente alla libreria `lcdf_vhdl`, creata dagli autori del libro di testo, contiene l'implementazione VHDL delle porte logiche di base, registri e flip-flop (disponibile nel sito web del libro di testo)

Es: Creazione di un package

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity NOT1 is
        port (in1 : in std_logic; out1 : out std_logic);
end NOT1;

architecture NOT1_impl of NOT1 is
begin
        out1 <= not in1;
end NOT1_impl;

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
package MyPackage is
component NOT1 is
        port(in1 : in std_logic; out1 : out std_logic);
end component;
end;
```

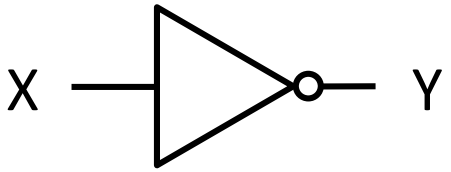
I nomi dei componenti e delle porte in, out devono essere gli stessi che compaiono nella entity!

Istanziazione dei componenti

- Un componente può essere istanziato in modi diversi all'interno di un design VHDL
  1) Istanziazione **per componente** (vista in precedenza nella realizzazione strutturale del comparatore, vedi slide 36)
  2) Istanziazione **diretta** (più compatta)
  3) Istanziazione **con package** (non la vedremo)

1) Istanziazione per componente

Descrizione di inverter:




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity inverter is
      port(X : in std_logic; Y : out std_logic);
end inverter;
architecture inverter_impl of inverter is
begin
      Y <= not X;
end inverter_impl;
```

1) Istanziazione per componente

Dichiarazione del componente in architecture (prima di `begin`), poi il componente viene istanziato con mappatura delle porte:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity TestCircuit is
    port ( input : in STD_LOGIC;
           output : out STD_LOGIC);
end TestCircuit;
architecture TestCircuit_impl of TestCircuit is
component inverter is
    port (X: in std_logic; Y: out std_logic);
end component;
begin
    INV1: inverter port map(input, output);
end TestCircuit_impl;
```

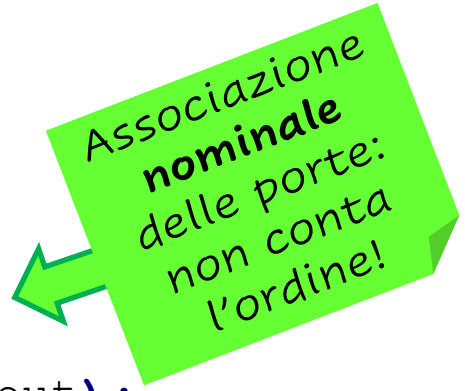
Stesso nome e porte di in/out che compaiono nella entity del componente

Associazione **posizionale** delle porte

1) Istanziazione per componente

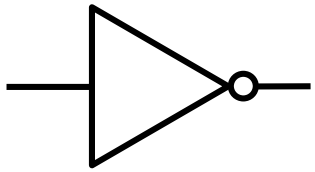
Dichiarazione del componente in architecture (prima di `begin`), poi il componente viene istanziato con mappatura delle porte:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity TestCircuit is
    port ( input : in STD_LOGIC;
           output : out STD_LOGIC);
end TestCircuit;
architecture TestCircuit_impl of TestCircuit is
component inverter is
    port (X: in std_logic; Y: out std_logic);
end component;
begin
    INV1: inverter port map(Y => output, X => input);
end TestCircuit_impl;
```



2) Istanziazione diretta

Descrizione di inverter:



```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity inverter is
      port(X : in std_logic; Y : out std_logic);
end inverter;
architecture inverter_impl of inverter is
begin
      Y <= not X;
end inverter_impl;
```

2) Istanziazione diretta

Istanziazione specificando entity e architecture, non serve la dichiarazione del componente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity TestCircuit is
    Port ( input : in STD_LOGIC;
           output : out STD_LOGIC);
end TestCircuit;

architecture TestCircuit_impl of TestCircuit is
begin
    INV1: entity work.inverter(inverter_impl) port map(input,output);
end TestCircuit_impl;
```

La parola riservata work indica che il componente è definito nella libreria corrente

3) Con package

Definizione del package che contiene il componente:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
package MyPackage is
component inverter is
      port(X : in std_logic; Y : out std_logic);
end component;
end;
```

3) Con package

Dichiarazione di uso del package e istanziazione con port map:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use work.MyPackage.all;
entity TestCircuit is
        port( input : in STD_LOGIC;
              output : out STD_LOGIC);
end TestCircuit;
architecture TestCircuit_impl of TestCircuit is
begin
        INV1: inverter port map(input, output);
end TestCircuit_impl;
```

Comparatore a 2 bit: VHDL dataflow

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

Librerie/ package

```
entity comparator_greater_than_dataflow is
  port (A: in std_logic_vector(1 downto 0);
    B: in std_logic_vector(1 downto 0);
    A_greater_than_B: out std_logic);
end comparator_greater_than_dataflow;
```

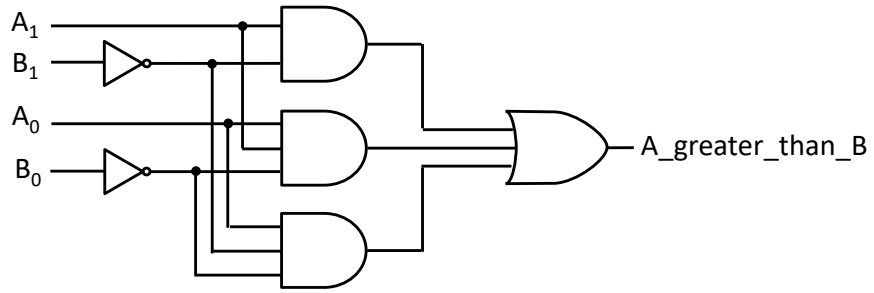
Entity

```
architecture dataflow of comparator_greater_than_dataflow is
signal B1_n, B0_n, and0_out, and1_out, and2_out: std_logic;
begin
 B1_n <= not B(1);
 B0_n <= not B(0);
 and0_out <= A(1) and B1_n;
 and1_out <= A(1) and A(0) and B0_n;
 and2_out <= A(0) and B1_n and B0_n;
 A_greater_than_B <= and0_out or and1_out or and2_out;
end dataflow;
```

Architecture

Architecture dataflow

```
architecture dataflow of comparator_greater_than_dataflow is
signal B1_n, B0_n, and0_out, and1_out, and2_out: std_logic;
begin
 B1_n <= not B(1);
 B0_n <= not B(0);
 and0_out <= A(1) and B1_n;
 and1_out <= A(1) and A(0) and B0_n;
 and2_out <= A(0) and B1_n and B0_n;
 A_greater_than_B <= and0_out or and1_out or and2_out;
end dataflow;
```



- Non specifica il dettaglio della struttura (non usa componenti), ma descrive la funzionalità del circuito attraverso i flussi di dati
- E' composta da equazioni booleane espresse con **assegnazioni concorrenti** (= eseguite tutte **in parallelo**): ogni volta che cambia il membro destro di un'assegnazione, al membro sinistro viene assegnato il nuovo valore
- Cambiando l'ordine delle assegnazioni, il circuito rimane lo stesso!

Comparatore a 2 bit: VHDL behavioral (1)

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

Librerie/ package

```
entity comparator_greater_than_behavioral is
  port (A: in std_logic_vector(1 downto 0);
   B: in std_logic_vector(1 downto 0);
   A_greater_than_B: out std_logic);
end comparator_greater_than_behavioral;
```

Entity

```
architecture when_else of comparator_greater_than_behavioral is
begin
 A_greater_than_B <= '1' when (A > B) else
                     '0';
end when_else;
```

Architecture con costrutto **when-else**

Comparatore a 2 bit: VHDL behavioral (2)

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

Librerie/ package

```
entity comparator_greater_than_behavioral2 is
  port (A: in std_logic_vector(1 downto 0);
   B: in std_logic_vector(1 downto 0);
   A_greater_than_B: out std_logic);
end comparator_greater_than_behavioral2;
```

Entity

```
architecture with_select of comparator_greater_than_behavioral2 is
begin
 with A select
 A_greater_than_B <= '0' when "00",
           B(0) nor B(1) when "01",
           not B(1) when "10",
           B(0) nand B(1) when "11",
           'X' when others;
end with_select;
```

Architecture con costrutto **with-select**

Comparatore a due bit: tabella di verità

A_1	A_0	B_1	B_0	A_greater_than_B
0	0	0	0	**0**
0	0	0	1	**0**
0	0	1	0	**0**
0	0	1	1	**0**
0	1	0	0	**1**
0	1	0	1	**0**
0	1	1	0	**0**
0	1	1	1	**0**
1	0	0	0	**1**
1	0	0	1	**1**
1	0	1	0	**0**
1	0	1	1	**0**
1	1	0	0	**1**
1	1	0	1	**1**
1	1	1	0	**1**
1	1	1	1	**0**

Costrutti ‘When-else’ e ‘With-select’

- Sono costrutti per istruzioni concorrenti (eseguite in parallelo)
  - **When-else**: si usa per assegnazioni condizionali (logica prioritaria), per valutare **priorità in cascata**
  - **With-select**: si usa per assegnazioni di segnali selezionati con logica parallela, per valutare diverse **scelte con la stessa priorità**
- Es: il bus di dati del PC riceve dati da diversi dispositivi (dal processore, dalla memoria, dal disco, dagli I/O) - ciascuno di questi dispositivi comanda lo stesso bus, siamo noi a decidere quale viene selezionato

When-else

Sintassi per il costrutto when-else

```
name <= espressione1 when condizione1 else
      { espressione2 when condizione2 else }
        espressione3;
```

Logica **prioritaria**:

- Condizione1 ha la priorità, indipendentemente dal fatto che sia o meno soddisfatta la condizione2
- Se condizione1 è soddisfatta, non vengono valutate le successive e viene selezionata la prima scelta
- Permette decisioni su segnali multipli (annidando when multipli, ognuno condizionato su un segnale diverso)

With-select

Sintassi per il costrutto with-select

```
with espressioneDiScelta select
      Z <= espressione1 when scelta1,
         { espressione2 when scelta2,
           espressioneN when sceltaN, }
           espressioneX when others;
```

Logica **parallela**:

- Le condizioni hanno tutte la stessa priorità
- Tutti i casi devono essere coperti da una (e una sola) scelta
- Tutte le scelte devono essere contemplate (è obbligatorio inserire la condizione di default ‘when others’)
- Permette decisioni in base ad una singola condizione booleana (tipicamente corrisponde a strutture meno complesse rispetto al when-else)

Testbench

- Il testbench (= banco di prova) è un codice VHDL per **simulare il comportamento di un circuito**
- Simula l'applicazione di segnali in ingresso al circuito da simulare (DUT, device under test) e ne rileva le uscite
  - Può essere pensato come un banco di prova che applica stimoli al circuito progettato e opzionalmente verifica se il circuito simulato (DUT) si comporta nel modo atteso
- Si tratta di codice che **non verrà sintetizzato in hardware**, ma che serve solo per la simulazione/ validazione di un design

Comparatore a 2 bit: Testbench

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

Librerie/package

```
entity greater_testbench is
end greater_testbench;
```

Entity (vuota: non ci sono porte di in e out)

```
architecture testbench of greater_testbench is
signal A, B: std_logic_vector (1 downto 0);
signal struct_out: std_logic;
component comparator_greater_than_structural is
   port (A: in std_logic_vector(1 downto 0);
   B: in std_logic_vector(1 downto 0);
   A_greater_than_B: out std_logic);
end component;
begin
u1: comparator_greater_than_structural port map(A,B, struct_out);
tb: process
begin
 A <= "10";
 B <= "00";
 wait for 10 ns;
 B <= "01";
 wait for 10 ns;
 B <= "10";
 wait for 10 ns;
 B <= "11";
 wait; -- halt the process
end process;
end testbench;
```

Architecture

Comparatore a 2 bit: Testbench

```
architecture testbench of greater_testbench is
signal A, B: std_logic_vector (1 downto 0);
signal struct_out: std_logic;
component comparator_greater_than_structural is
   port (A: in std_logic_vector(1 downto 0);
   B: in std_logic_vector(1 downto 0);
   A_greater_than_B: out std_logic);
end component;
begin
u1: comparator_greater_than_structural port map(A,B, struct_out);
tb: process
begin
 A <= "10";
 B <= "00";
 wait for 10 ns;
 B <= "01";
 wait for 10 ns;
 B <= "10";
 wait for 10 ns;
 B <= "11";
 wait; -- halt the process
end process;
end testbench;
```

Dichiarazione dei segnali per stimolare ingressi del DUT e del segnale che sarà connesso all'uscita del DUT

Comparatore a 2 bit: Testbench

```
architecture testbench of greater_testbench is
signal A, B: std_logic_vector (1 downto 0);
signal struct_out: std_logic;
component comparator_greater_than_structural is
   port (A: in std_logic_vector(1 downto 0);
   B: in std_logic_vector(1 downto 0);
   A_greater_than_B: out std_logic);
end component;
begin
u1: comparator_greater_than_structural port map(A,B, struct_out);
tb: process
begin
 A <= "10";
 B <= "00";
 wait for 10 ns;
 B <= "01";
 wait for 10 ns;
 B <= "10";
 wait for 10 ns;
 B <= "11";
 wait; -- halt the process
end process;
end testbench;
```

Dichiarazione del componente che servirà da DUT

Istanziazione del componente (con associazione posizionale): A,B vengono connessi agli ingressi, struct_out all'uscita

Comparatore a 2 bit: Testbench

```
architecture testbench of greater_testbench is
signal A, B: std_logic_vector (1 downto 0);
signal struct_out: std_logic;
component comparator_greater_than_structural is
   port (A: in std_logic_vector(1 downto 0);
   B: in std_logic_vector(1 downto 0);
   A_greater_than_B: out std_logic);
end component;
begin
u1: comparator_greater_than_structural port map(A,B, struct_out);
tb: process
begin
 A <= "10";
 B <= "00";
 wait for 10 ns;
 B <= "01";
 wait for 10 ns;
 B <= "10";
 wait for 10 ns;
 B <= "11";
 wait; -- halt the process
end process;
end testbench;
```

Process: contiene gli **stimoli al DUT**, cioè combinazioni di valori assegnate agli ingressi una dopo l'altra, con attesa di un tempo di 10 ns, cioè 10^{-9} s (usati per la simulazione).
Nel process gli statement vengono eseguiti in modo sequenziale e non concorrente!
La parola riservata wait sospende il processo (per un certo tempo o indefinitamente)

Process

- Un process è una dichiarazione concorrente (in parallelo alle altre presenti nell'architecture), all'interno del quale gli **statement vengono eseguiti in modo sequenziale** (uno dopo l'altro) nell'ordine in cui sono scritti
- Un process viene avviato immediatamente se non è specificata una lista di sensibilità (esempio precedente) o quando varia uno dei segnali nella lista di sensibilità (prossima slide)
- Il valore dei segnali assegnato all'interno di un processo viene **aggiornato soltanto quando il processo viene sospeso** (o perché sono state eseguiti tutti gli statement all'interno del processo o perché c'è un'istruzione wait)
  - Se ci sono assegnazioni multiple allo stesso segnale, il segnale prende l'ultimo valore che gli viene assegnato all'interno del processo

Process

Sintassi per il costrutto process

```
process [ ( <lista_di_sensibilità> ) ]
        signal …
        component …
begin
        <istruzioni_sequenziali>
end process;
```

Process: Lista di sensibilità

- Nel testbench abbiamo visto l'uso del costrutto process senza una lista di sensibilità (sensitivity list)
- La lista di sensibilità è opzionale e contiene i segnali al cui cambiamento il processo è sensibile
  - Se è vuota, il processo viene eseguito di continuo, in assenza di comandi espliciti per fermare il processo (per es. wait)
- Se presente, la lista di sensibilità può contenere solo segnali che possono essere letti (input o signal, non output!)
- Vedremo meglio l'uso del costrutto process in futuro

Disclaimer

Figures from *Logic and Computer Design Fundamentals*, Fifth Edition, GE Mano | Kime | Martin