Politecnico di Milano
V Facoltà di Ingegneria – DEI
Corso di Laurea in Ingegneria Informatica

Introduzione a VHDL e agli strumenti di sviluppo Xilinx

A cura di:
Giovanni Beltrame
Istituto CEFRIEL
Via Fucini 2, Milano

Sommario

[Introduzione]

Introduzione

Linguaggio e Sintassi

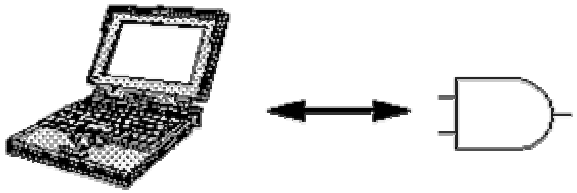
Sviluppo e Simulazione

Sintesi con Webpack

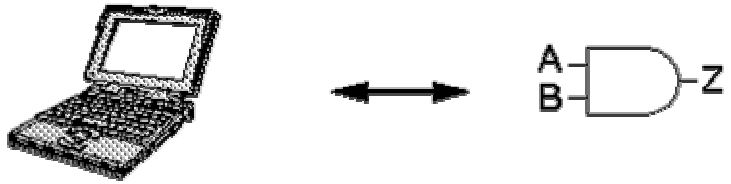
Esempi pratici

Definizione di HDL

- VHDL è un linguaggio di descrizione dell'hardware (Hardware Description Language)
- Per 'hardware' si intendono diverse cose a seconda del contesto



- L'hw si può descrivere a vari livelli di astrazione




```
if PRINTREQUEST then
   -- print protokoll
end if ;
```

```
Z <= A and B ;
```

- VHDL permette la specifica a tutti i livelli in modo generale

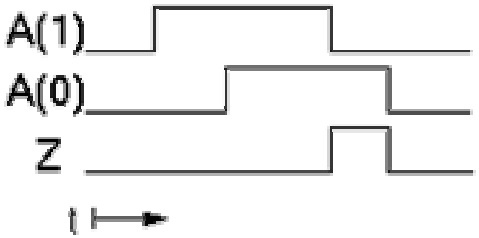
Utilizzo di un HDL

- Un HDL può essere utilizzato in tutte le fasi di sviluppo di un sistema hardware.

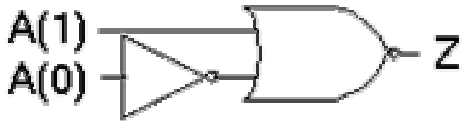
Modelling


```
Z <= '1' when A="01"
     else '0';
```

Simulation

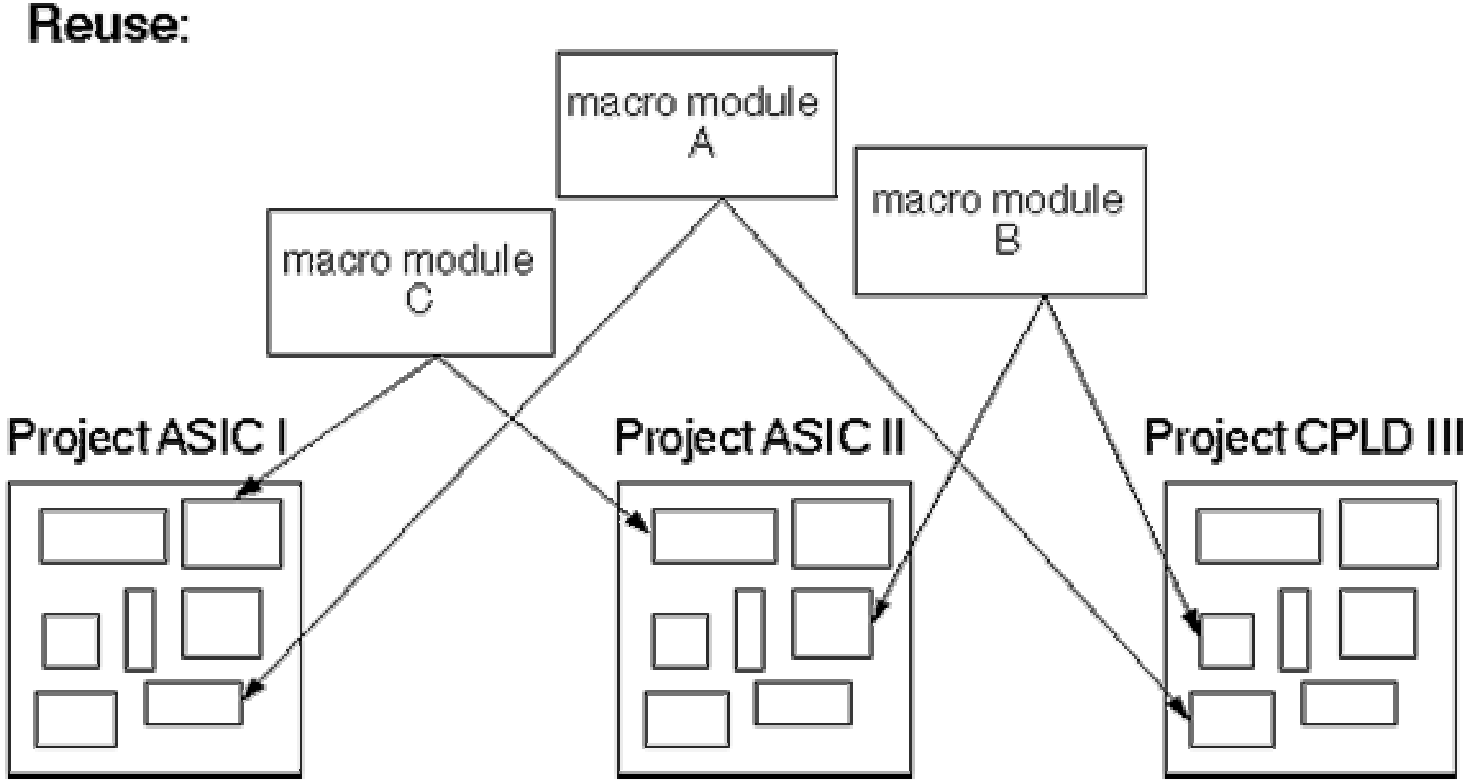


Synthesis



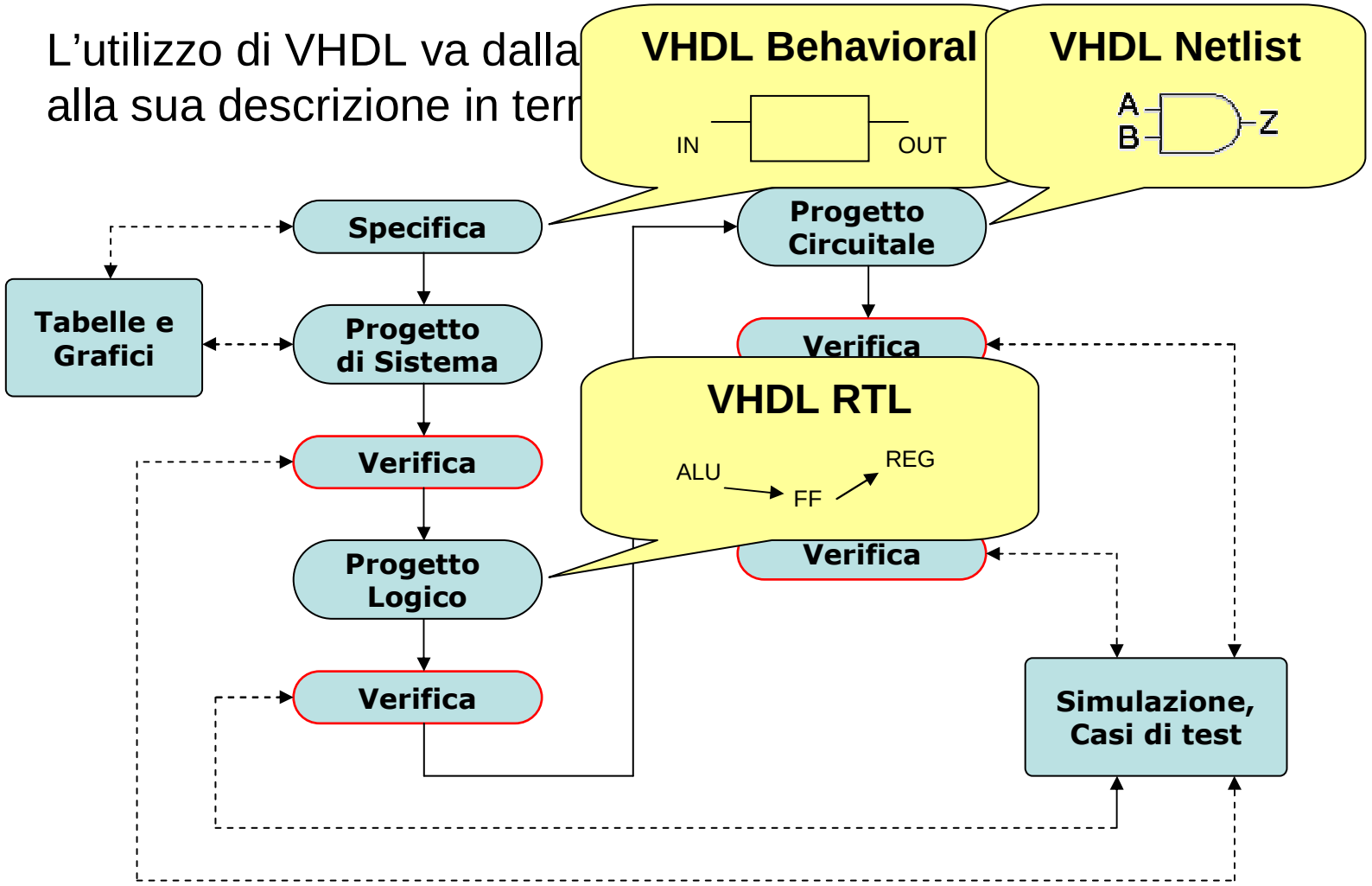
Utilizzo di un HDL

- Gli HDL permettono inoltre il riutilizzo del codice, e se il caso, lo sfruttamento della proprietà intellettuale (IP)



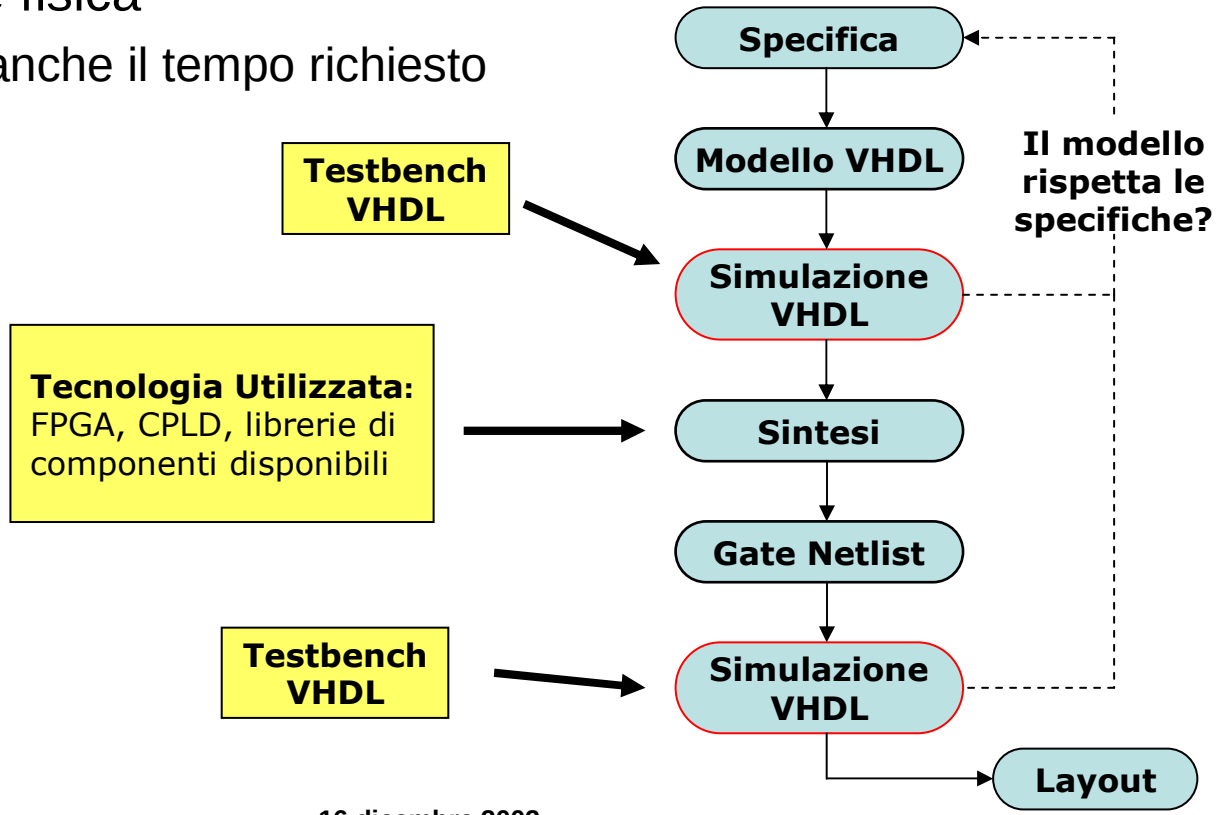
Spazio di utilizzo

- L'utilizzo di VHDL va dalla
  alla sua descrizione in terr



Sviluppo di ASIC

- L'utilizzo di VHDL per lo sviluppo ci circuiti integrati specifici segue un certo flusso
- La precisione della simulazione aumenta con l'approssimarsi alla realizzazione fisica
  - Aumenta anche il tempo richiesto



Sommario

[Linguaggio e Sintassi]

Introduzione

Linguaggio e Sintassi

Sviluppo e Simulazione

Sintesi con Webpack

Esempi pratici

Linguaggio

Concetti Fondamentali

- VHDL si differenzia da un normale linguaggio di programmazione su due punti:
- L’esecuzione delle istruzioni
  - Sequenziale
  - Parallela
- Le metodologie di sviluppo utilizzate
  - Astrazione
  - Modularità
  - Gerarchia

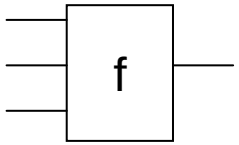
Linguaggio

Astrazione

- L’astrazione consiste nel nascondere dettagli
  - E’ la distinzione tra l’informazione essenziale e quella non essenziale
- Il concetto si applica creando diversi livelli di astrazione
  - Ad ogni livello viene considerata solo l’informazione essenziale
  - Tutto ciò che non è essenziale viene tralasciato
- Tutta l’informazione in un dato livello di astrazione, presenta lo stesso livello di essenzialità

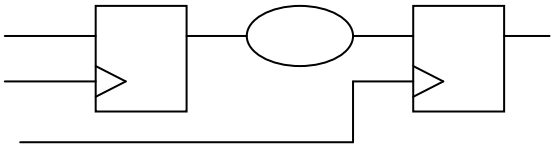
Linguaggio

Livelli di astrazione per IC



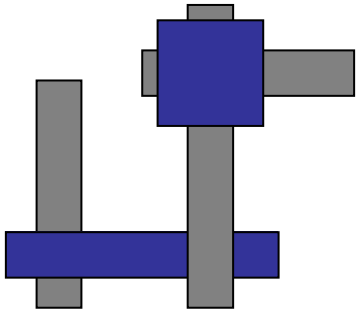
Behavior

RTL



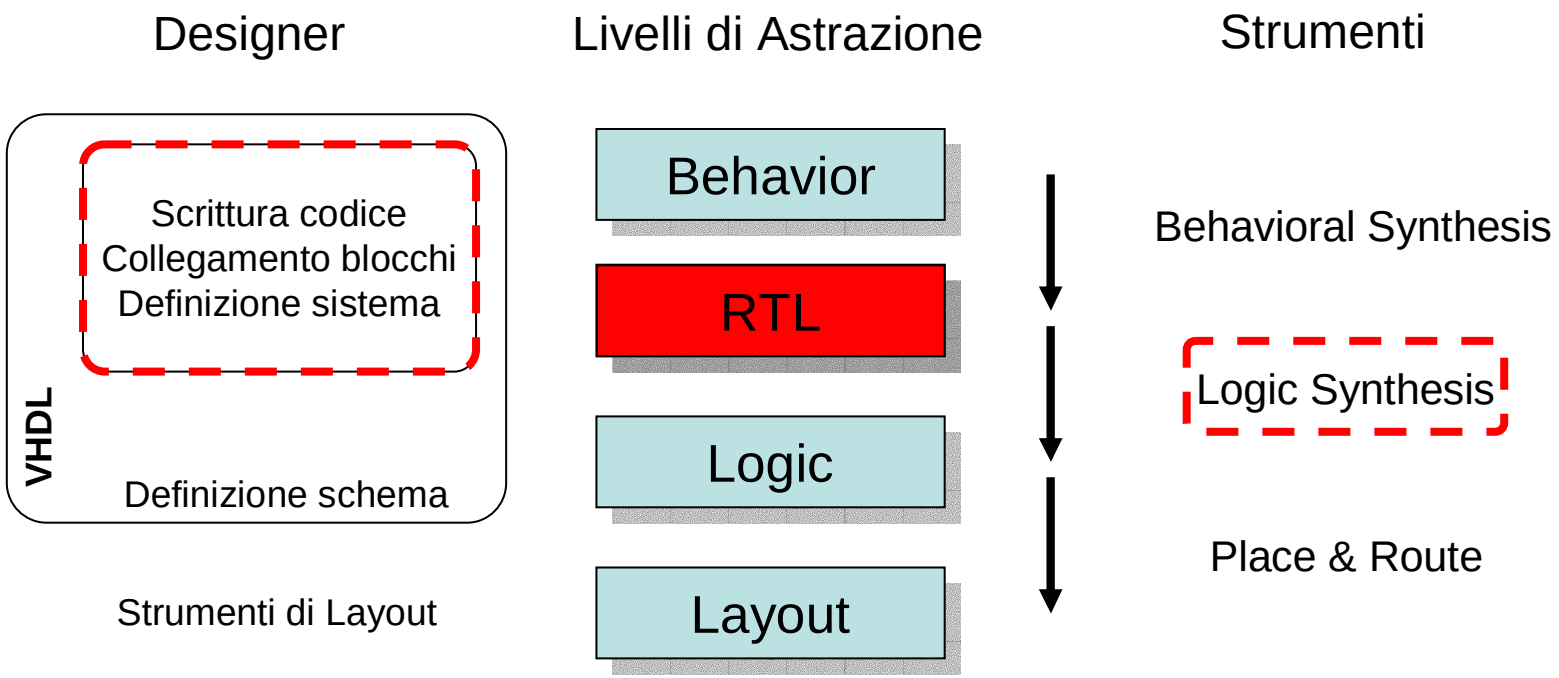
Logic

Layout



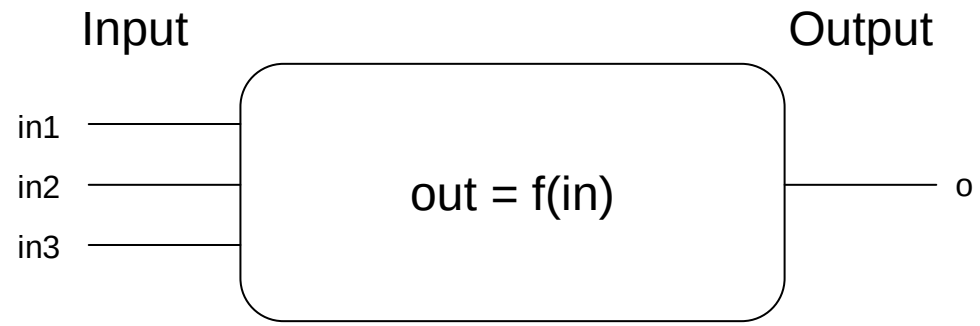
Linguaggio

Livelli di astrazione e VHDL

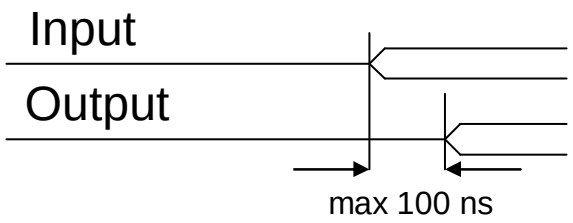


Linguaggio

Descrizione Behavioral in VHDL



Specification



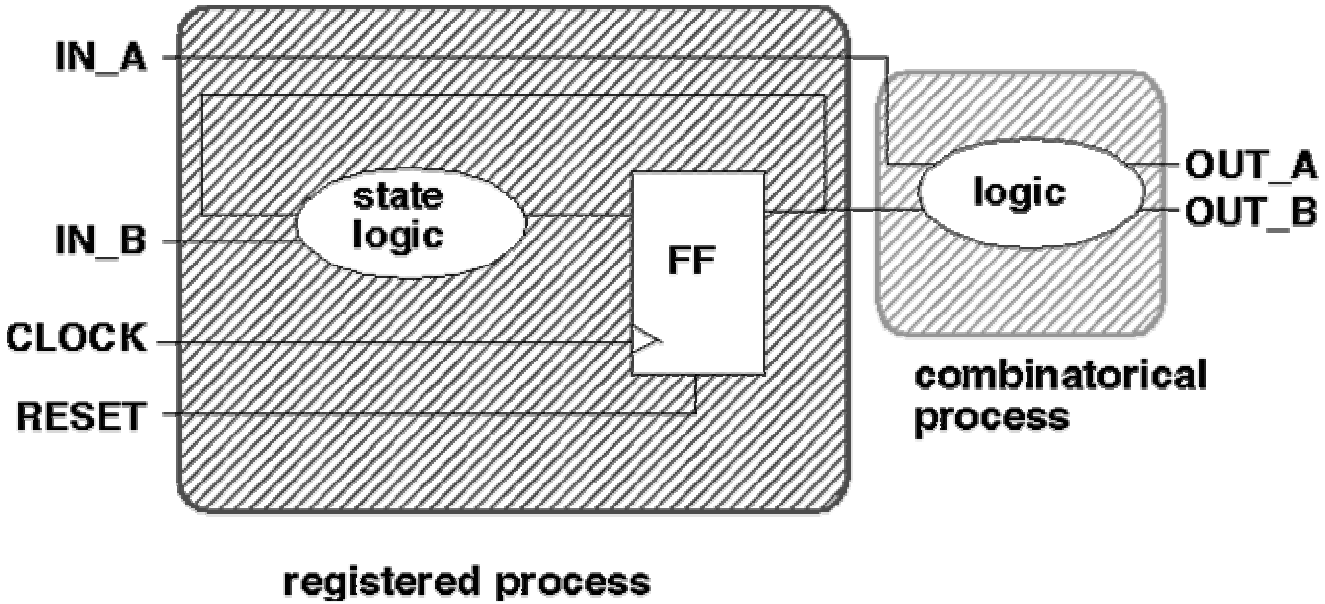
VHDL

```
O <= i1+i2*i3 after 100ns;
```

Linguaggio

Descrizione RT in VHDL

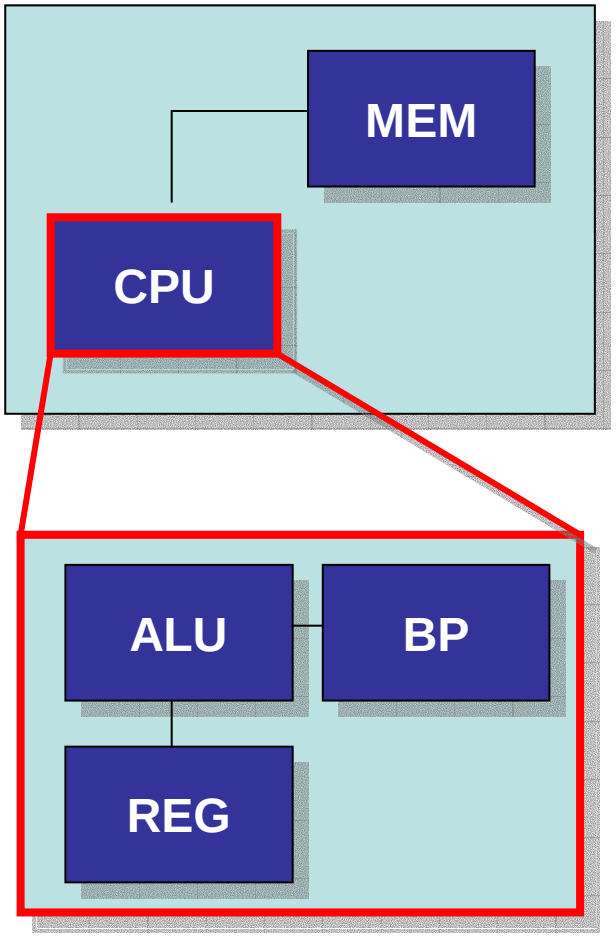
- E’ possibile separare il sistema in due processi
  - Una parte combinatoria
  - Una parte sequenziale
- La parte sequenziale avrà registri e logica di gestione della FSM



Linguaggio

Modularità e Gerarchia

- Permettono di scomporre il progetto in sottosezioni
  - Si riduce la complessità
- Permettono l'esplorazione dello spazio di progetto e la simulazione dei singoli componenti



Sintassi Generale VHDL

```
--------------------------------
-- Example VHDL Code --
--------------------------------
-- an example signal
signal mySignal: bit;

MYsignal <= '0',
-- start with '0‘
            ’1' AFTER 10 ns,
-- and toggle after
            '0' after 10 ns,
-- every 10 ns
            '1' afTer 10 ns;
```

- VHDL è case-insensitive
- I commenti sono indicati da ‘--’ a fine riga
- Tutte le istruzioni sono terminate da ‘;’ e possono occupare più linee
- Gli elementi di una lista sono delimitati da ‘,’
- I segnali sono assegnati da ‘<=‘
- Gli identificativi definiti dall’utente sono composti da lettere, numeri o ‘_’, ma devono iniziare con una lettera

Linguaggio

Costrutti di VHDL

- In VHDL sono presenti diversi costrutti
- **Entity**: descrive l'interfaccia di un componente
- **Architecture**: descrive l'implementazione o il comportamento di un componente
- **Configuration**: permette di strutturare la gerarchia e il collegamento tra moduli
- **Process**: gestione del parallelismo e degli eventi
- **Package**: fornisce moduli progettuali, tipi, ecc. Punta al riuso del codice
- **Library**: fornisce codice compilato

Costrutti di VHDL

- In VHDL esistono anche oggetti, dichiarabili all'interno dei costrutti visti secondo la tabella

	Entity	Architecture	Process	Package
Subprogram	X	X	X	X
Component		X		X
Configuration		X		
Constant	X	X	X	X
Datatype	X	X	X	X
Port	X			
Signal		X	X	X
Variable			X	

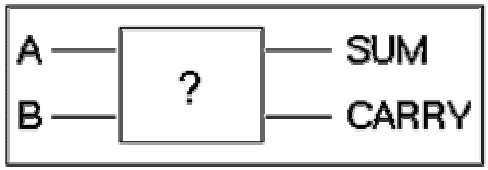
Linguaggio

Il costrutto Entity

- Permette di dichiarare l'interfaccia di un modulo o componente senza descriverne il comportamento
- Contiene il costrutto **`port`**, con cui vengono rappresen gresso e uscita

Nome entità

```
entity HALFADDER is
  port(
    A, B:        in  bit;
    SUM, CARRY: out bit);
end HALFADDER;
```



Direzione porta

```
entity ADDER is
  port(A,B:   in  integer range 0 to 3;
       SUM:   out integer range 0 to 3;
       CARRY: out bit );
end ADDER;
```

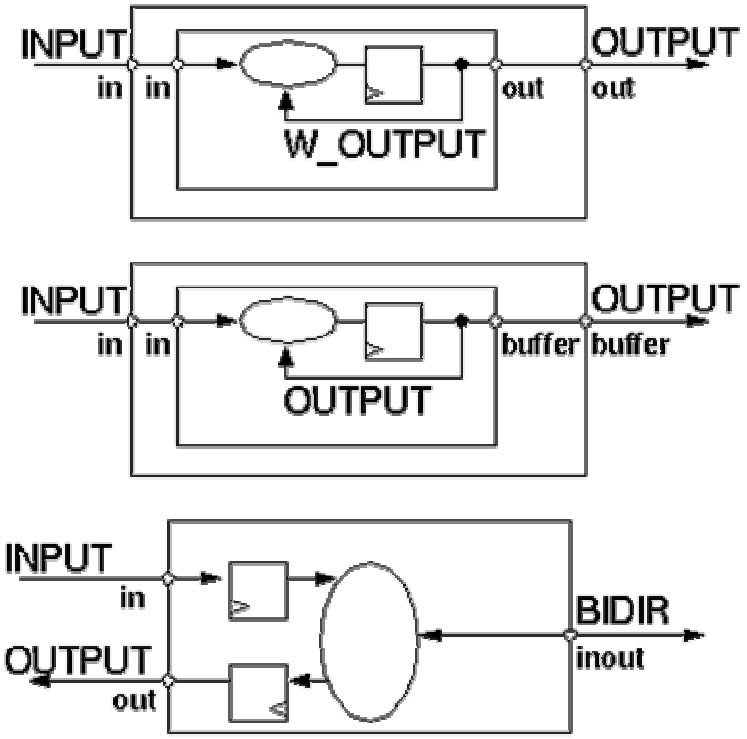
Tipo di segnale

Linguaggio

Linguaggio

Modalità di utilizzo delle porte

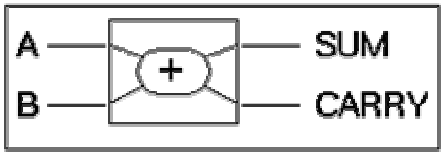
- **in**:
  - Segnali in sola lettura
- **out**:
  - Segnali in sola scrittura
- **buffer**:
  - Simile ad out
  - I segnali possono essere anche letti
- **inout**:
  - Porta bidirezionale
- NB: I tipi delle porte devono coincidere!



Linguaggio

Il costrutto Architecture

- Costituisce l'implementazione di un modulo
- E' sempre collegata ad una specifica entity
  - Una entity può avere diverse architecture
- Le porte dell'entità sono disponibili come segnali all'interno dell'architettura
- Tutte le istruzioni presenti sono eseguite in parallelo




```
entity HALFADDER is
  port(A,B: in   bit;
       SUM, CARRY: out bit);
end HALFADDER;

architecture RTL of
HALFADDER is
begin
  SUM   <= A xor B;
  CARRY <= A and B;
end RTL;
```

Identificatore Architettura

Istruzioni (implementazione)

Linguaggio

Il costrutto Architecture: struttura

- Ha due parti, una dichiarativa:
  - Tipi d
  - Costa
  - Segnali addizionali (di collegamento)
  - Componenti
- Una parte di definizione (dopo il begin):
  - Assegnamento di segnali
  - Processi
  - Creazione di componenti
- Tutte le istruzioni sono svolte in parallelo: l'ordine non conta

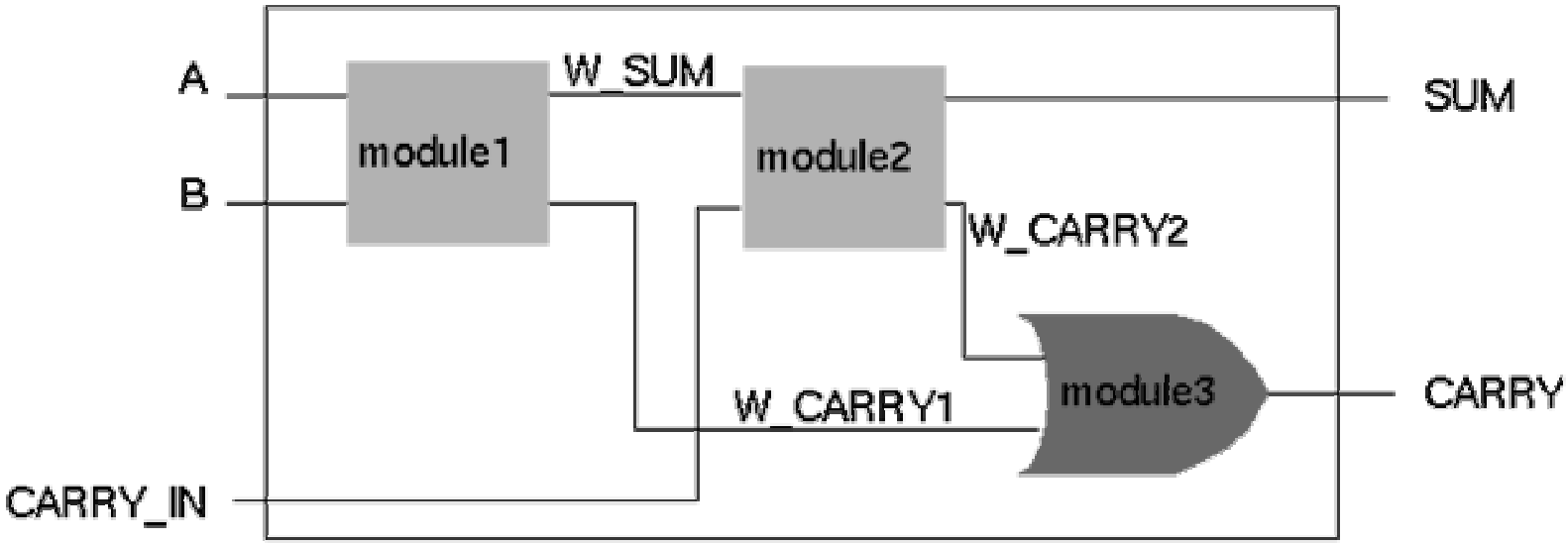
Parte dichiarativa

```
architecture EXAMPLE of STRUCTURE is
  subtype DIGIT is integer
                   range 0 to 9;
  constant BASE: integer := 10;
  signal DIGIT_A, DIGIT_B: DIGIT;
  signal CARRY: DIGIT;
begin
  DIGIT_A <= 3;
  SUM <= DIGIT_A + DIGIT_B;
  DIGIT_B <= 7;
  CARRY <= 0 when SUM < BASE
           else 1;
end EXAMPLE;
```

Definizione del modulo

Linguaggio

Modellazione Gerarchica: Esempio



Full adder: 2 halfadders + 1 OR-gate

Linguaggio

Modellazione gerarchica: componenti

- I componenti di un modulo sono specificati nella parte dichiarativa di architecture
- In pratica definiscono una interfaccia

```
entity FULLADDER is
 port (A,B,CARRY_IN: in   bit;
       SUM,CARRY: out bit);
end FULLADDER;

architecture STRUCT of FULLADDER is
 signal W_SUM,W_CARRY1, W_CARRY2 : bit;

 component  HALFADDER
  port (A,B: in   bit;
        SUM,CARRY: out bit);
  end component;
 component  ORGATE
  port (A, B: in bit;
        RES: out bit);
 end component;
begin
. . .
```

Linguaggio

Modellazione gerarchica: componenti

```
architecture STRUCT of FULLADDER is
 component HALFADDER
   port (A,B: in   bit;
         SUM,CARRY: out bit);
 end component;
 component ORGATE
                    bit;
                  t bit);
 end component;
 signal W_SUM,W_CARRY1,W_CARRY2: bit;
begin
  MODULE1: HALFADDER
      port map(A,B,W_SUM,W_CARRY1);
  MODULE2: HALFADDER
      port map(W_SUM,CARRY_IN,SUM,
               W_CARRY2);
  MODULE3: ORGATE
      port map(W_CARRY2,W_CARRY1,
               CARRY);
end STRUCT;
```

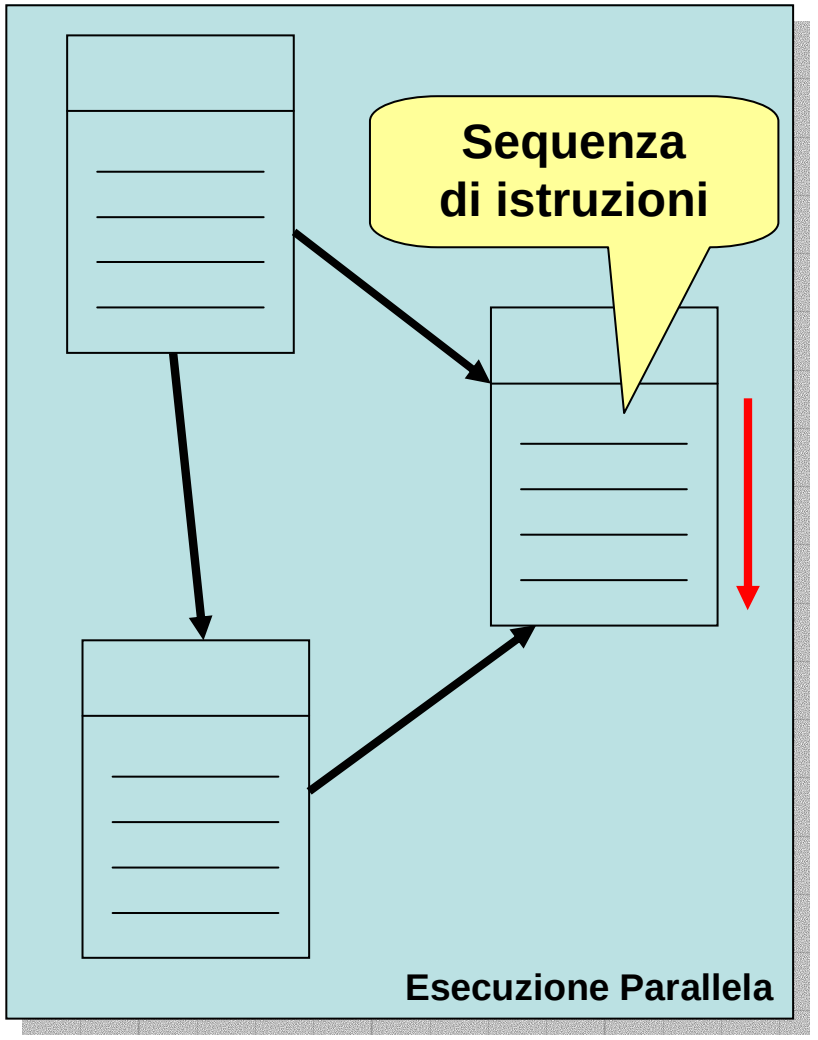
Creazione di una istanza

- Le istanze dei componenti sono create nella parte di definizione
- I segnali di questi vengono assegnati ai segnali dell'architettura tramite **`port map`**
- L'associazione dei segnali è posizionale

Linguaggio

Il costrutto Process

- Il costrutto process permette l'esecuzione sequenziale di operazioni
- I processi esistono solo all'interno di una architettura
- All'interno di una architettura possono essere presenti più processi concorrenti
- L'esecuzione di questi è controllata da *sensitivity list* o dall'istruzione *wait*
- I processi sono dotati di un nome opzionale



Linguaggio

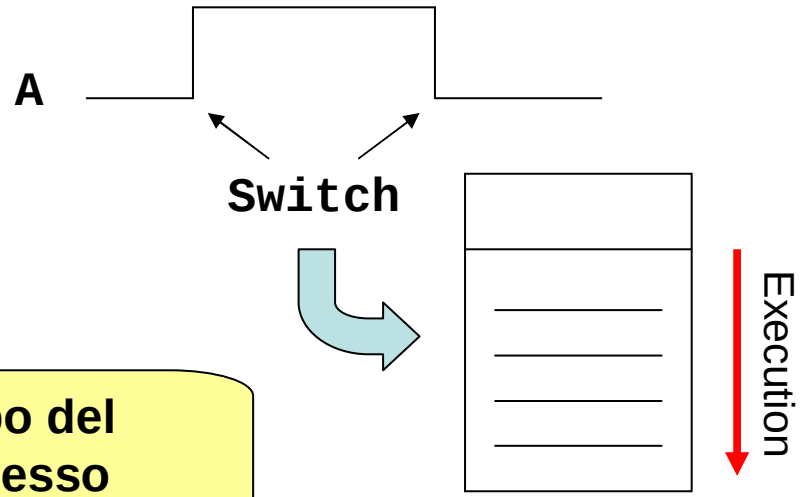
Il costrutto Process: sintassi

```
entity AND_OR_XOR is
 port(A,B : in bit;
      Z_OR,Z_AND,Z_XOR:out
end AND
architect      RTL OF AND
begin
 A_O_X: process(A, B)
  begin
   Z_OR  <= A or  B;
   Z_AND <= A and B;
   Z_XOR <= A xor B;
  end process A_O_X ;
end RTL;
```

Etichetta del processo

Sensitivity List: il processo viene attivato da ogni variazione dei segnali della lista

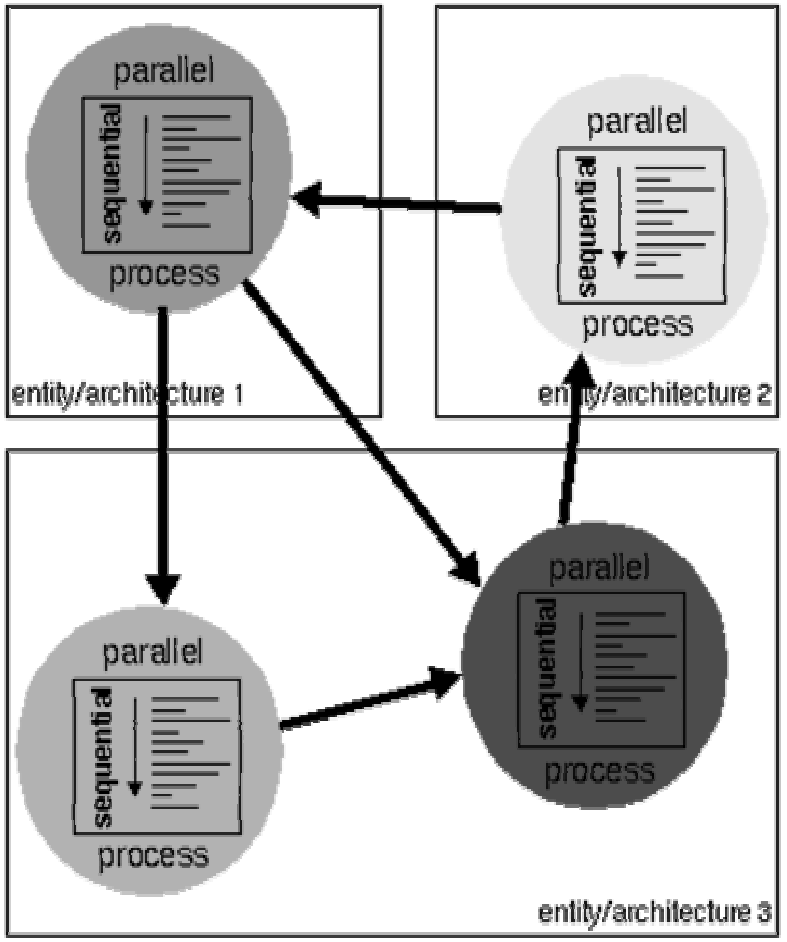
Corpo del processo



Linguaggio

Modello di comunicazione di VHDL

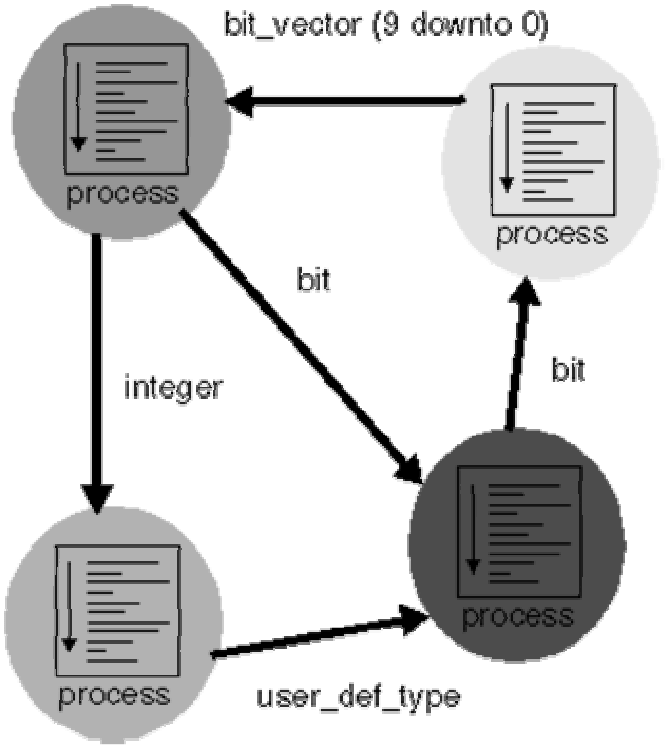
- I processi comunicano tra di loro attraverso segnali e sensitivity list
- Processi su architetture differenti possono comunicare tramite le interfacce fornite dalle entity
- Tutti i processi vengono eseguiti in parallelo e sono costituiti da una sequenza di operazioni



Linguaggio

I Segnali

- Ogni segnale di è caratterizzato da un tipo che ne specifica i possibili valori
- Esistono diversi tipi predefiniti
  - Esempio: bit, bit_vector, integer, real, ecc.
- L'utente può definire a sua volta dei tipi
  - Migliorano il modello dell'hardware, la leggibilità, la possibilità di trovare errori, ecc.
  - Esempio: segnale ad alta impedenza



Le Librerie

- Una libreria è una collezione di elementi VHDL precompilati
- E’ un nome logico con cui vengono raggruppate una serie di strutture
  - Contiene la locazione fisica di ogni file
- La libreria “corrente”, ossia top-level è la libreria WORK
- Dentro la libreria WORK possono essere presenti altre librerie definite dall’utente

Linguaggio

Linguaggio

Le Librerie

- Una libreria fondamentale è la libreria IEEE
  - Contiene i tipi STD_LOGIC, utilissimi per i segnali binari
- I tipi standard IEEE introducono diversi valori aggiuntivi:
  - U: undefined
  - X: bus contention (2 segnali sullo stesso canale)
  - Z: alta impedenza
  - W: weak unknown (U debole elettricamente)
  - L: 0 debole elettricamente
  - H: 1 debole elettricamente

Esempio di utilizzo:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Linguaggio

Definizione di tipi

- L’utente può definire nuovi tipi con il costrutto type
- Un tipo viene definito come un elenco di valori

```
type MY_STATE is (RESET, ST1, ST2);
type std_logic is (‘U’, -- undefined
                   ‘X’, -- bus contention
                   ...
```

- Segnali di un dato tipo sono assegnabili solamente con i valori specificati

```
signal STATE : MY_STATE;
...
STATE <= RESET;
```

Linguaggio

Strutture di controllo

- IF-THEN-ELSIF

```
if condition then
        -- statements
elsif condition then
        -- statements
end if;
```

- FOR

```
for i in 0 to 30 loop
        -- statements
end loop;
```

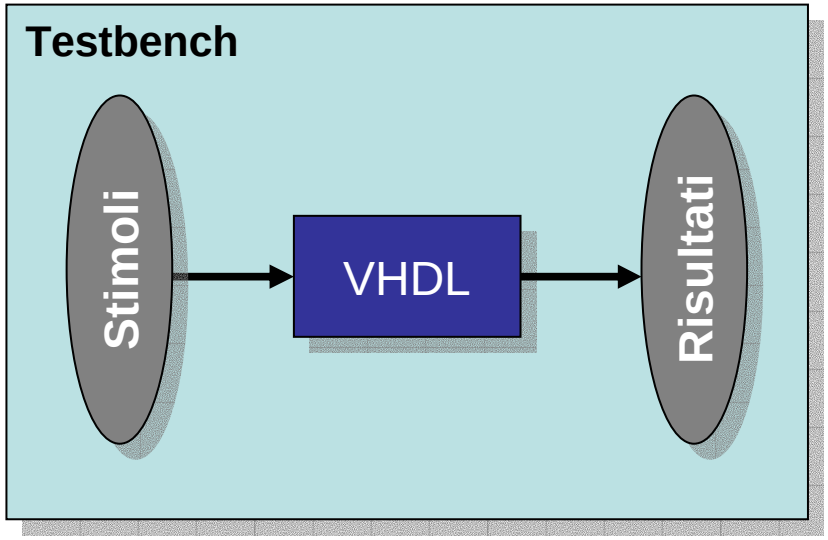
- CASE

```
case expr is
        when val_1 => statements;
        when val_2 => statements;
end case;
```

Linguaggio

I Testbench VHDL

- Per poter testare i propri design, il progettista può crearsi dei testbench
- Sono entità top-level che forniscono i segnali ai moduli da provare e ne valutano le risposte
- Solitamente non hanno segnali di ingresso e uscita



Linguaggio

Esempio di Testbench

Componente sotto test

```
entity ADDER IS
 port (A,B: in   bit;
       CARRY,SUM : out bit);
end ADDER;

architecture RTL of ADDER is
begin
  ADD: process (A,B)
  begin
   SUM <= A xor B;
   CARRY <= A and B;
  end process ADD;
end RTL;
```

```
architecture TEST of TB_ADDER is
 component ADDER
  port (A, B:  in   bit;
        CARRY, SUM: out bit);
 end component;
 signal A_I,B_I,
        CARRY_I,SUM_I: bit;
 begin
  DUT: ADDER
  port map(A_I,B_I,
           CARRY_I, SUM_I);
  STIMULUS: process
   begin
    A_I <= '1'; B_I <= '0';
    wait for 10 ns;
    A_I <= '1'; B_I <= '1';
    wait for 10 ns;
      -- and so on ...
   end process STIMULUS;
end TEST;
```

Casi di test

Sommario

[Sintesi con WebPack]

Introduzione
Linguaggio e Sintassi
Sviluppo e Simulazione
Sintesi con Webpack
Esempi pratici

Flusso FPGA

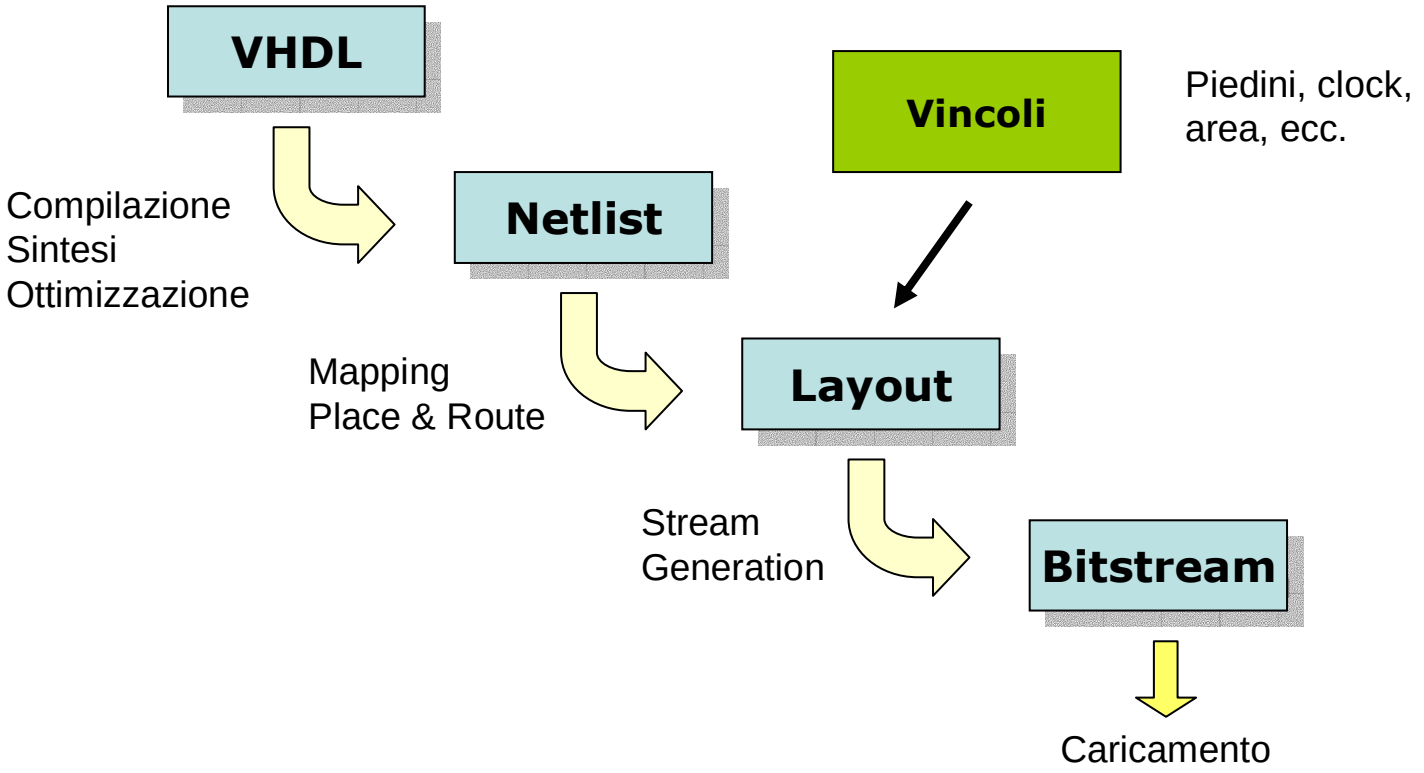
- Un FPGA (*Field Programmable Gate Array)* è un dispositivo hardware riprogrammabile
- E' composto da una serie di blocchi (CLBs, *Complex Logic Blocks*) la cui funzionalità e i collegamenti sono definiti dall'utente
- Esistono FPGA con migliaia di blocchi
- Ogni blocco può fornire funzionalità complesse, equivalenti a centinaia di gate
  - In pratica esistono FPGA da centinaia di migliaia di gate equivalenti
- WebPack fornisce strumenti per la sintesi hardware su FPGA

Flusso FPGA

- La configurazione (i.e., la definizione del circuito) di un FPGA viene caricata nel silicio tramite un opportuno canale
- Il file di configurazione viene chiamato *bitstream*
  - E' una sequenza di bit che fornita ai piedini dell'FPGA la configura
- Il bitstream può essere presente in una ROM sulla stessa scheda dell'FPGA, oppure può essere caricato esternamente

Flusso FPGA

- I tool per la sintesi FPGA (come WebPack) seguono il flusso seguente per realizzare il bitstream:



La nostra board

- Gli esempi di oggi sono stati realizzati su una FPGA Xilinx XC4010XL, da 20K gate, con un clock massimo di 100MHz
- WebPack non supporta dispositivi della classe XC4000, per via della licenza free
  - Il bitstream che vedrete è stato realizzato con ISE Foundation, la versione a pagamento dello stesso tool
- Gli esempi verranno mostrati per l'FPGA Spartan, del tutto analoga (ma per cui avete la licenza!)

Installazione e setup di WebPack

- L'applicazione si può scaricare dal sito www.xilinx.com
  - Sezione prodotti -> Free ISE WebPack
  - E' necessario registrarsi sul sito di Xilinx
- Verrà comunque fornito al supporto tecnico qui a Cremona – Sono 175Mb di archivio!
  - Vi daranno dei CD?
- Xilinx fornisce anche un simulatore, sviluppato da Mentor Graphics (ModelSim)
- ModelSim può essere scaricato assieme a WebPack, ma necessita una licenza separata
  - La licenza va richiesta sul sito di Mentor e vi verrà spedita via mail nel giro di 2 gg

Sommario

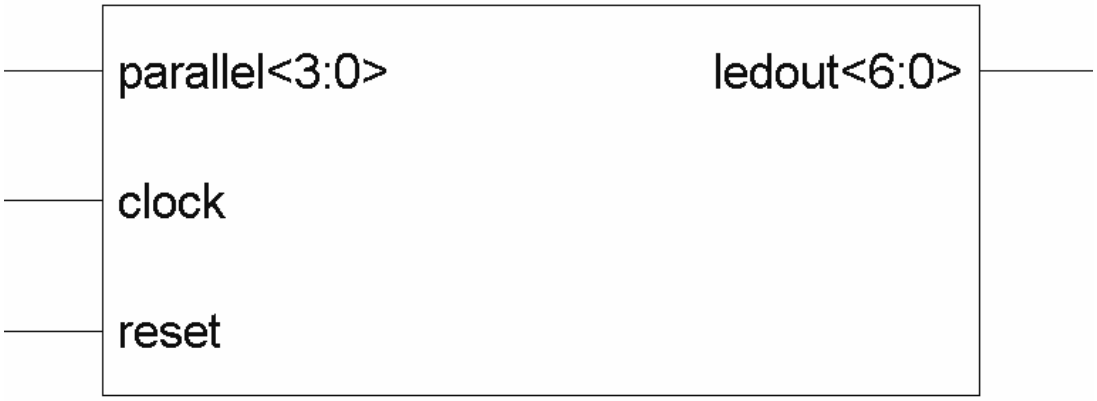
[Esempi Pratici]

Introduzione
Linguaggio e Sintassi
Sviluppo e Simulazione
Sintesi con Webpack
Esempi pratici

Esempi Pratici

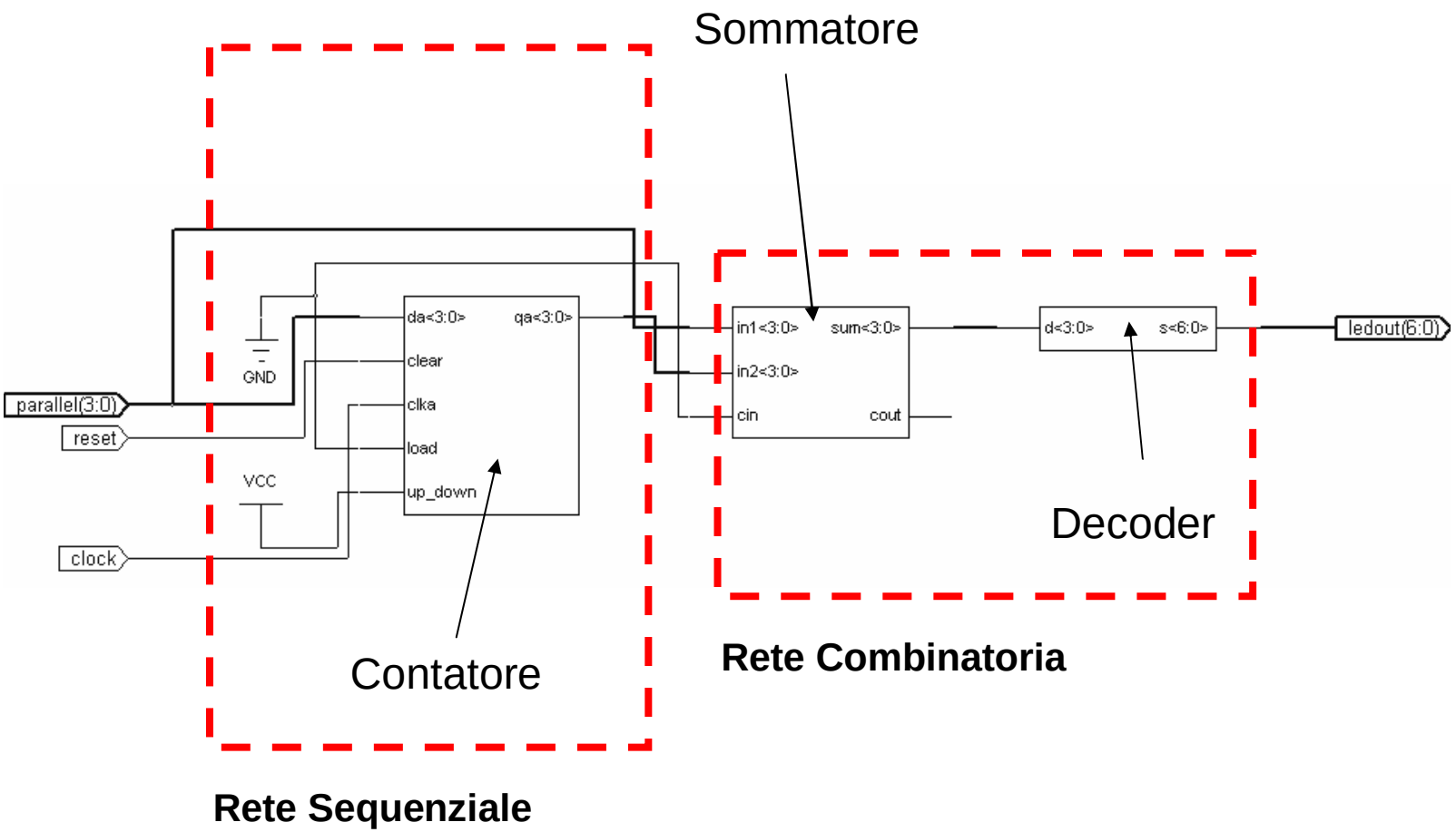
Esempio

- Cosa andremo a realizzare?
- Un circuito con in ingresso un segnale parallelo a 4 bit, un clock e un reset
- In uscita avremo un numero codificato per la visualizzazione su un display a 7 segmenti
- Il sistema somma il segnale parallelo ad un suo contatore progressivo interno e fornisce questa somma codificata



Esempi Pratici

Esempio



Esempi Pratici

Esempio

- Per realizzare la parte sequenziale è necessario creare un registro
  - Deve contenere il valore attuale del contatore
- Tutti i segnali utilizzati da un processo sincrono saranno sintetizzati in registri

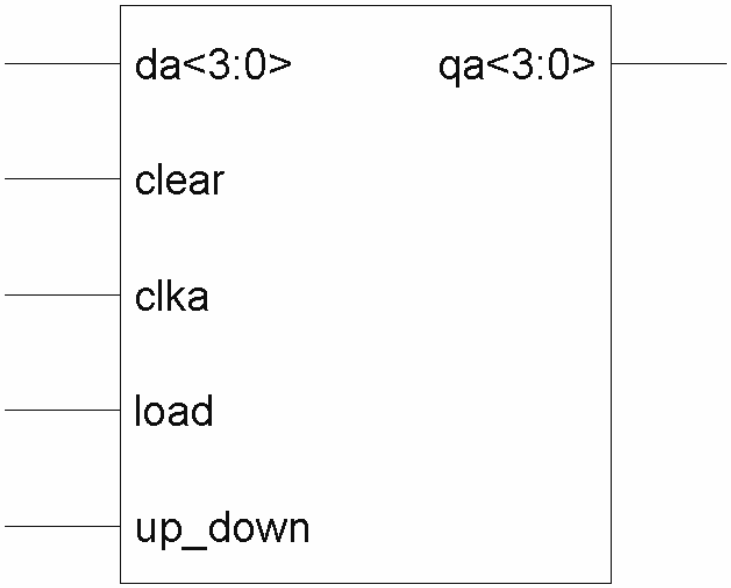
```
architecture Behavioral of counter is
signal count : std_logic_vector( 3 downto 0 );
begin
 process(clka)
 begin
  if (clka'event and clka='1') then
   if( load = '1' ) then
    count <= da;
   ...
```

Registro a 4 bit

Esempi Pratici

Esempio

- Realizziamo un contatore un po' più complesso di quello che ci serve realmente
- In ingresso abbiamo:
  - Un vettore a 4 bit
  - Un segnale di reset (clear) che azzera il contatore
  - Un clock
  - Un segnale che permette il caricamento di un valore nel registro (load)
  - Un indicatore della direzione del contatore (up_down)



Esempi Pratici

Esempio

- Il contatore conterrà un processo che crea un registro e lo modifica al variare del clock, in base ai segnali di ingresso

```
architecture Behavioral of counter is
signal count : std_logic_vector( 3 downto 0 );
begin
  process(clka)
  begin
    if (clka'event and clka='1') then
      if( load = '1' ) then
        count <= da;
      elsif( clear = '1' ) then
          count <= "0000";
      elsif( up_down = '1' ) then
          count <= count+1;
      else
          count <= count-1;
      end if;
    end if;
  end process;
  qa <= count;
end Behavioral;
```

Sensibile al clock

Reagisce solo al fronte di salita del clock

L'uscita è sempre uguale al registro del contatore

Esempi Pratici

Esempio

- Sommatore e decodificatore sono parti di logica combinatoria pura
- Il sommatore a 4 bit sarà costituito da 4 full-adder a 1 bit
  - Si utilizza la gerarchia per includerli in un bit4Adder
- Il decoder avrà semplicemente una descrizione behavioral, indicata come ingresso -> uscita

```
architecture Structural of bit4adder is
SIGNAL carry  :   STD_LOGIC_VECTOR(2 DOWNTO 0);
component full_add is
    Port ( a : in std_logic;
           b : in std_logic;
           c_in : in std_logic;
           sum : out std_logic;
           c_out : out std_logic);
end component;
begin
          c0   : full_add
            PORT MAP (in1(0),in2(0),cin,sum(0),carry(0));
          c1   : full_add
            PORT MAP (in1(1),in2(1),carry(0),sum(1),carry(1));
          c2   : full_add
            PORT MAP (in1(2),in2(2),carry(1),sum(2),carry(2));
          c3   : full_add
            PORT MAP (in1(3),in2(3),carry(2),sum(3),cout);
end Structural;
```

Bus a 3 bit

4 full-adder

Esempi Pratici

Esempio: simulazione

- Simuliamo il sistema sintetizzato tramite ModelSim
- I comandi fondamentali di ModelSim sono:
  - **`force <signal> <value> [time], <value> [time], ...`**
    forza un segnale ad un dato valore in un certo istante di tempo
  - **`run [time]`**
    esegue la simulazione fino all'istante specificato
  - **`restart`**
    ritorna all'istante 0, annullando quanto fatto
- Per impostare un clock per la simulazione occorre usare un comando speciale:
  - **`force clock 0 50, 1 100 -repeat 100`**
    l'opzione repeat 100 ripete per sempre i primi 100 istanti
- Nella finestra wave è possibile osservare il comportamento di tutti i segnali della nostra architettura

Sommario

[Fine]