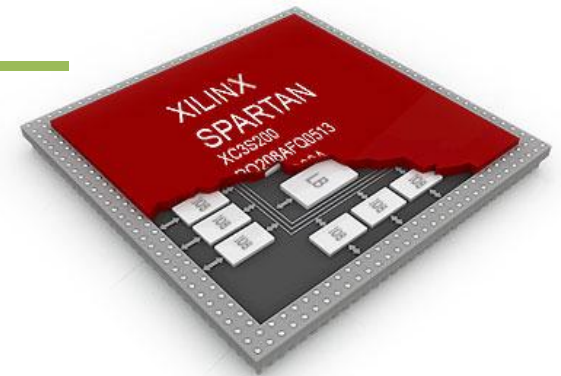


VHDL

Linguaggi di descrizione hardware, introduzione al VHDL e a Xilinx ISE Design Suite



31 ottobre 2012

Corso di Sistemi Embedded
Università degli Studi dell'Aquila

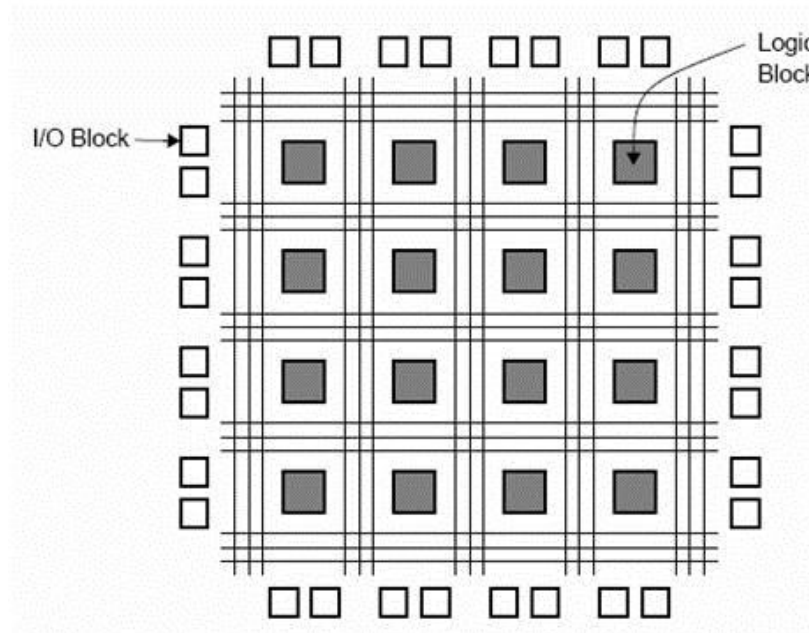


Panoramica

- Introduzione
- I linguaggi di descrizione hardware
- Introduzione al linguaggio VHDL
- Simulazione e sintesi
- Xilinx ISE Design Suite

Hardware Configurabile

- Un FPGA (Field Programmable Gate Array) è una piattaforma hardware configurabile.



- Sugli FPGA è possibile implementare microprocessori con le relative periferiche (system on chip)

Mapping degli algoritmi di elaborazione

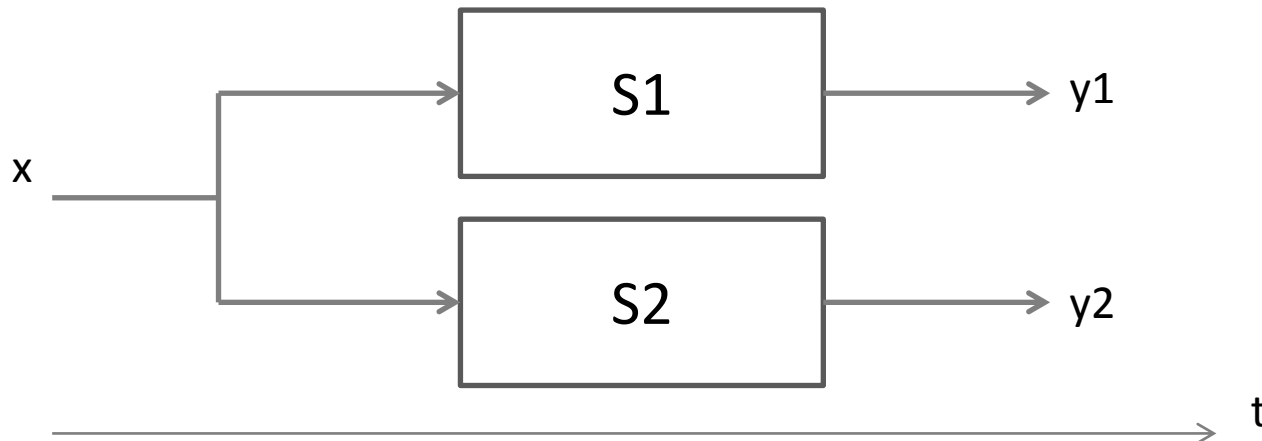
- Uno degli usi principali delle piattaforme configurabili resta in ogni caso il mapping diretto in hardware degli algoritmi di elaborazione del segnale.
- Gli FPGA consentono di sfruttare efficacemente i meccanismi di parallelismo e pipeline, anche simultaneamente (architetture ad array)
- Ovviamente anche un core ad array può vedere limitate le proprie prestazioni a seconda delle strutture dati in gioco

Linguaggi di descrizione dell'hardware

- Un HDL (Hardware Description Language) è un linguaggio per la descrizione formale di un circuito elettronico digitale
- La funzionalità dell'hardware viene descritta in modo indipendente dall'implementazione

Elementi distintivi

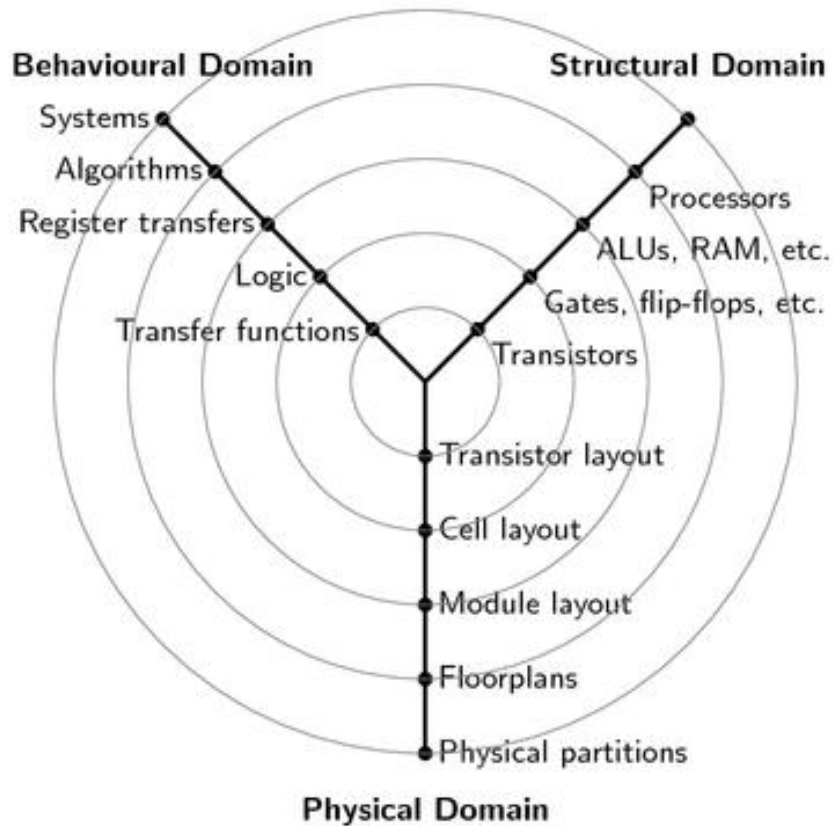
- Rispetto a un linguaggio di programmazione, un HDL permette di descrivere:
 - La temporizzazione : i sistemi reali sono caratterizzati da ritardi nella propagazione dei segnali
 - La concorrenza : i blocchi che compongono un certo sistema evolvono in contemporanea durante un certo intervallo di tempo



Domini e livelli di astrazione

- Di un circuito (digitale) si può fornire una:
 - Descrizione di livello fisico : si considerano i dispositivi fisici che compongono il sistema
 - Descrizione strutturale : si descrive il sistema in termini dei sotto-blocchi elementari che lo compongono
 - Descrizione comportamentale : si descrive solamente la funzionalità del sistema, ovvero il legame tra le uscite e gli ingressi

Y chart



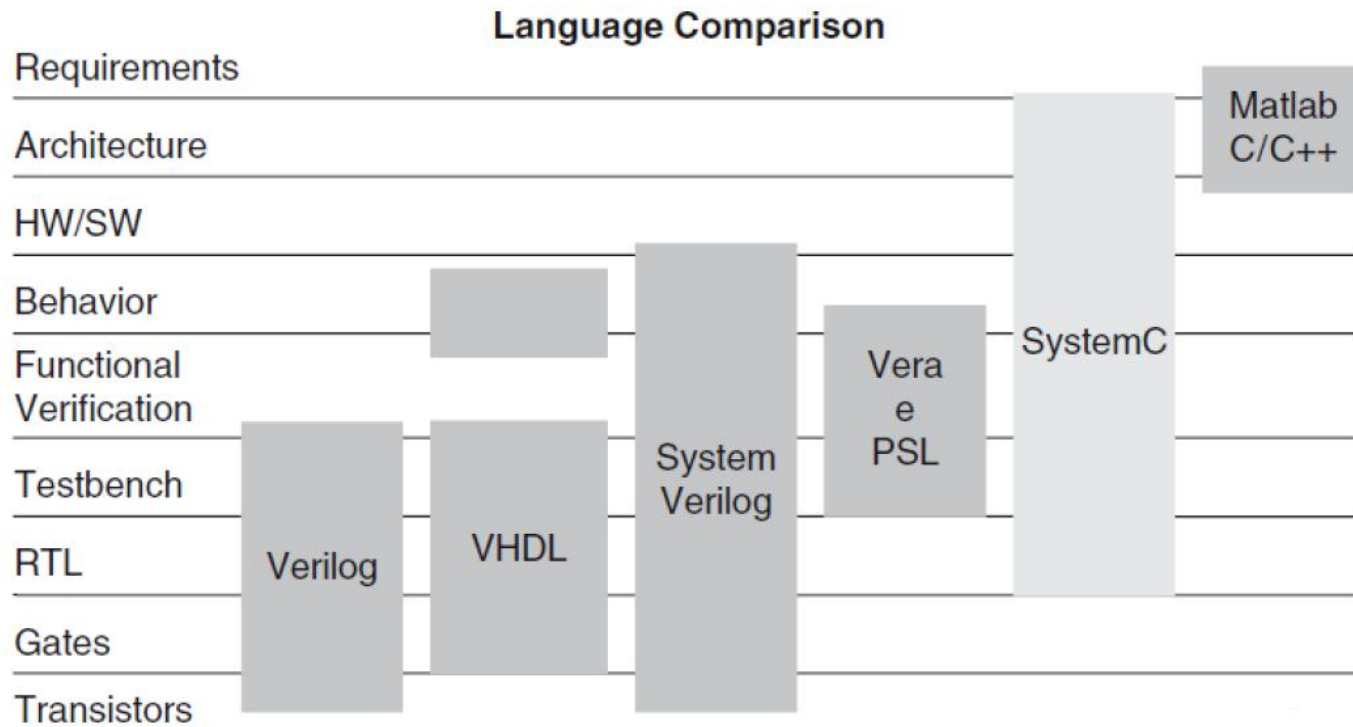
Non sempre il processo di progettazione rispetta il diagramma

Figure 1: Gajski-Kuhn Y-chart

Obiettivi della descrizione

- Un linguaggio di descrizione non è eseguibile
- La modellazione di un circuito digitale ha come scopo ultimo:
 - La simulazione: si verifica se il comportamento del circuito è quello atteso
 - La sintesi: si genera un file che permetta di configurare il dispositivo hardware target
- Non tutti i costrutti simulabili sono sintetizzabili

Principali linguaggi di descrizione



* Fonte: SystemC: from the ground up, Wiley, 2010

Il linguaggio VHDL

- VHDL sta per VHSIC (Very High Speed Integrated Circuits) Hardware Description Language
- È stato sviluppato all'inizio degli anni '80 dal Dipartimento per la Difesa degli Stati Uniti e rilasciato pubblicamente nel 1985.
- Standard IEEE 1076-1987, rivisto nel 1993 e nel 2008

ADA e VHDL

- Il VHDL fu sviluppato a partire dal linguaggio di programmazione ADA, e da esso ha ereditato molti elementi
 - A livello concettuale: linguaggio fortemente tipizzato, non case sensitive;
 - A livello sintattico: Ad esempio:

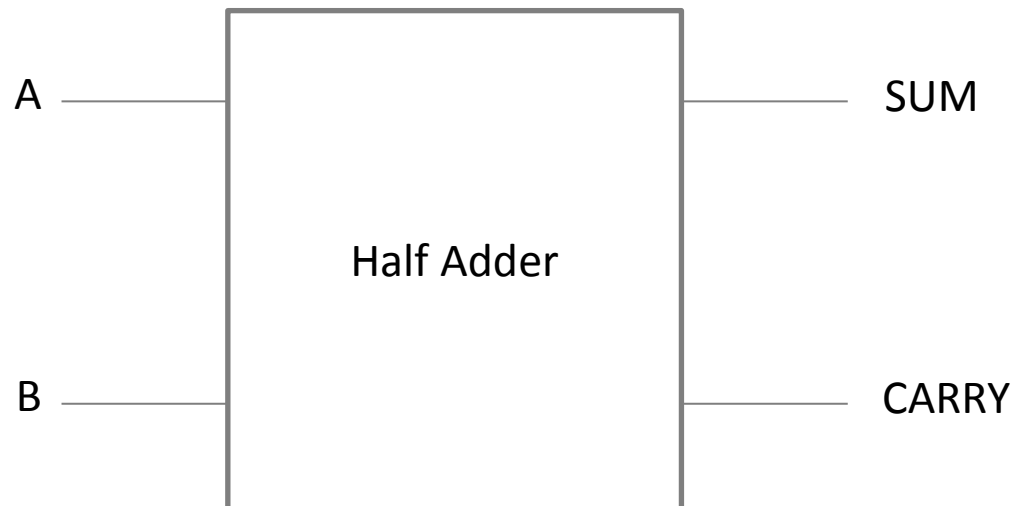
```
--Questo è un commento in VHDL
```

Design units

- Le Design Units sono segmenti di codice VHDL che possono essere compilati separatamente ed inclusi in una library.
- Una descrizione VHDL di un sistema digitale deve contenere almeno una entity e una architecture
 - Una entity specifica in modo univoco il nome identificativo di un sistema e le caratteristiche dei suoi ingressi e delle sue uscite
 - Nell'architecture viene invece descritta l'eettiva funzionalità del circuito

Un semplice esempio: half adder

- Un half adder riceve in ingresso due bit (A,B) e genera in uscita un bit corrispondente alla somma (SUM) e uno al riporto (CARRY)



Half adder: entity

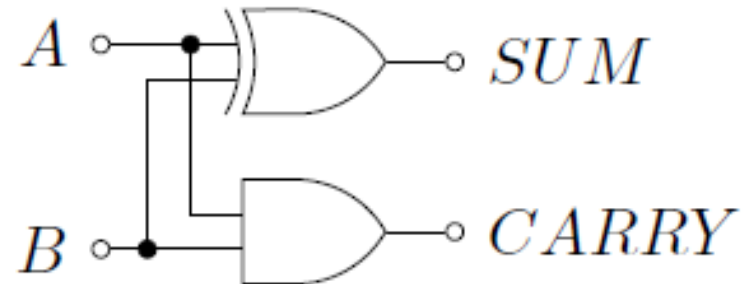
- Al top level del circuito corrisponde la entity VHDL:

```
-- Dichiarazione della entity half adder  
entity half_adder is  
port (A, B: in bit;  
        SUM , CARRY : out bit);  
end half_adder ;
```

Half adder: specifica funzionale

- Dalla descrizione del funzionamento e possibile compilare la tabella di verità e quindi arrivare alla struttura del circuito

A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1



Half adder: architecture

- La funzionalità viene descritta dall'architecture associata alla entity

```
--Dichiarazione dell'architecture ha_arch  
-- associata alla entity half adder
```

```
architecture ha_arch of half_adder is  
begin  
<area concorrente >  
end ha_arch ;
```

- Le istruzioni di area concorrente vengono eseguite in parallelo

Costrutti di area concorrente

- Assegnazione concorrente: si utilizza l'operatore `<=` per assegnare ad un segnale un valore

```
Z <= A;  
Z <= '1';  
Zvector <= "10001";
```

- Assegnazione condizionata: si assegna un valore ad un segnale quando si verifica una certa condizione

```
Z <= A when SEL = "00" else  
      B when SEL = "11" else  
      C;
```

Costrutti di area concorrente/2

- Assegnazione con selezione: simile all'assegnazione condizionata, permette di elencare rapidamente un numero elevato di casi (analogia: istruzioni if e switch-case)

```
with SEL select  
Z <= A when "000",  
      B when "001",  
      C when "010",  
      D when "011"  
      E when others ;
```

Half adder: architettura completa

- Si utilizzano gli operatori booleani e l'operatore <= (assegnazione) per descrivere la funzionalità:

```
--Dichiarazione dell'architecture ha_arch associata  
--alla entity half adder
```

```
architecture ha_arch of half_adder is  
begin  
    SUM <= A xor B;  
    CARRY <= A and B;  
end ha_arch ;
```

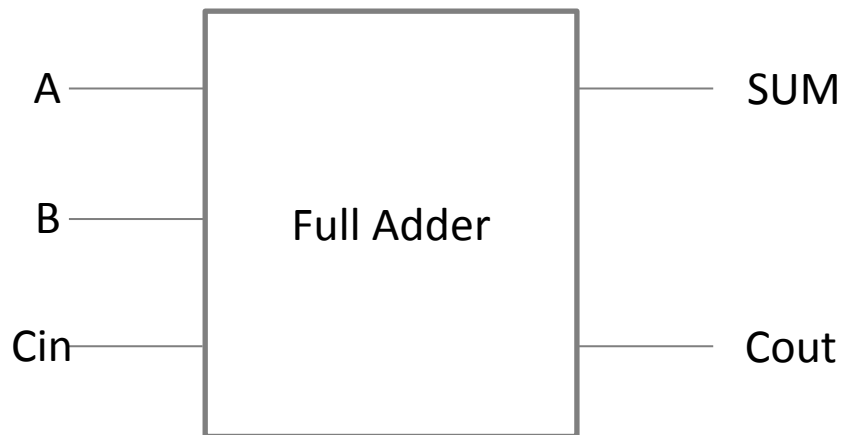
- Ogni architecture fa riferimento a una sola entity, ma ad una entity possono essere associate più architecture

I livelli di astrazione

- Il VHDL permette la descrizione di un circuito a due livelli di astrazione:
 - Behavioural level: si descrive la funzionalità del circuito in termini algoritmici oppure si specifica come i dati si muovono all'interno del sistema (dataflow)
 - Structural level: si descrive il sistema come interconnessione di componenti di base
- I vari tipi di descrizione possono essere combinati a seconda delle necessità

Esempio: full adder

Un full adder è un circuito digitale a tre ingressi e due uscite. Oltre a calcolare la somma dei due bit in ingresso esso è in grado di tenere conto di un riporto da una somma precedente



A	B	Cin	SUM	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Entity del full adder

Questa la entity del full adder:

```
entity FULL_ADDER is  
port (A, B, Cin : in bit ;  
        SUM , Cout : out bit );  
end FULL_ADDER ;
```

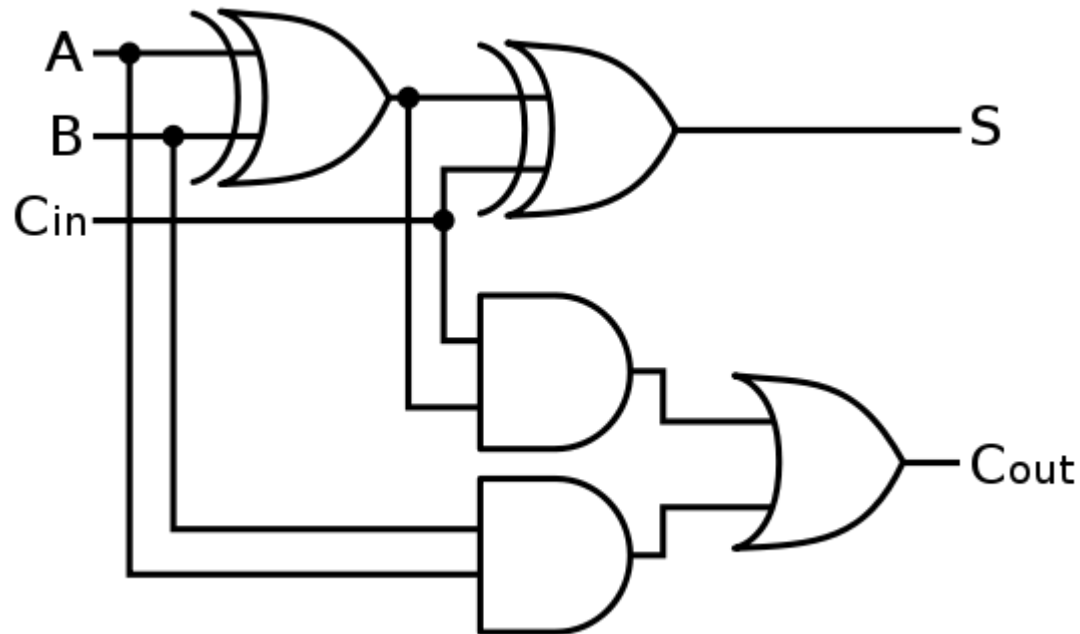
Algorithmic level

- Quella algoritmica e la descrizione di più alto livello ammessa dal VHDL
- Il costrutto di base è il processo: un processo permette di aprire un'area ad esecuzione sequenziale all'interno dell'area concorrente
- Questa la sintassi di base:

```
architecture alg of full_adder is  
begin  
  sum : process (A,B)  
  begin  
  
    <area sequenziale >  
  
    ...  
  
  end process
```

Descrizione del full adder

Possiamo sintetizzare il Full adder come fatto per gli Half adder e ottenere così una rete di porte logiche.



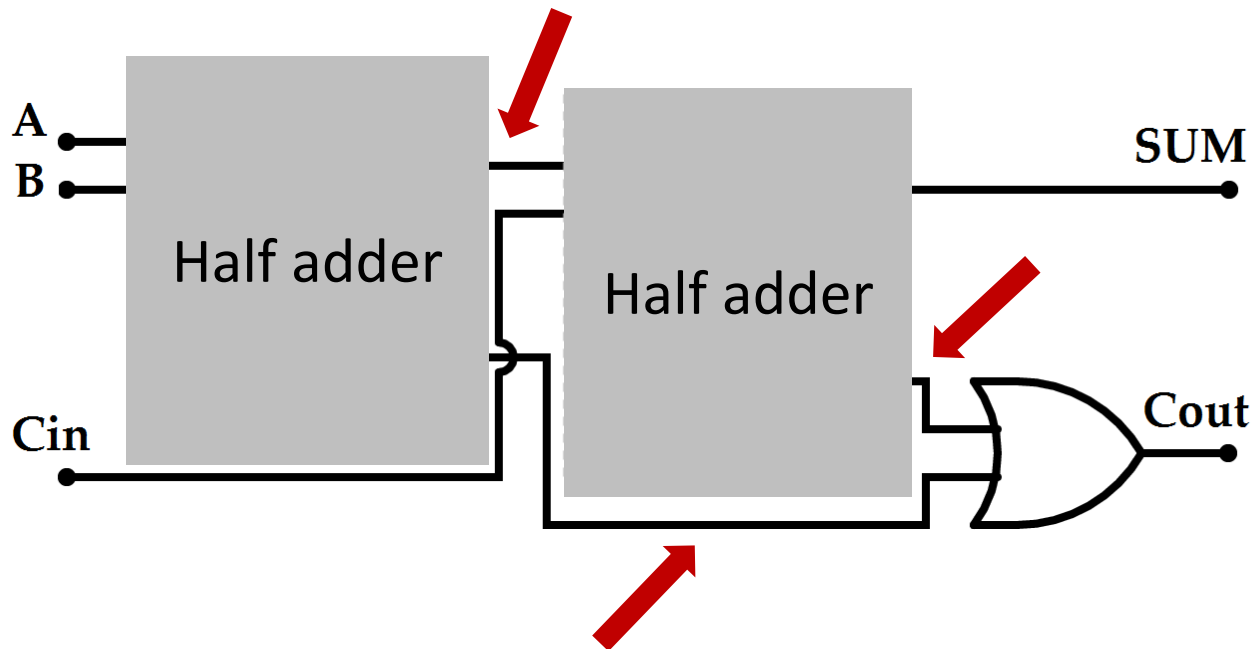
Dataflow level

- Si descrive la propagazione del flusso di dati all'interno del circuito
- Ad esempio per il full-adder:

```
architecture dataflow of full_adder is  
  
SUM <= A xor B xor C_in ;  
CARRY <= (A and B) or ( C_in and (A or B) ) ;  
  
end dataflow ;
```

Descrizione del full adder

Si nota comunque che la funzionalità del full adder può essere ottenuta combinando due half adder in maniera opportuna



Segnali e componenti

- L'architecture descrive la funzionalita vera e propria di un circuito istanziato tramite una entity.
- Oltre al corpo vero e proprio, nell'architecture e presente anche un'area dichiarativa in cui è possibile dichiarare segnali e componenti:

```
architecture name of entity_name is  
component INV  
port ( ... );  
end component ;  
signal x,y,z : bit ;  
begin
```

- I segnali non sono visibili all'infuori del modulo

Structural level

- Utilizzando i component possiamo modellare un sistema come interconnessione di sottosistemi (precedentemente definiti)
- In VHDL si utilizza il costrutto portmap per istanziare i componenti e definirne le interconnessioni. I segnali servono a rappresentare i collegamenti tra componenti.

```
architecture struct of full_adder is  
  signal s1 ,c1 ,c2: bit ;  
  component half_adder  
  port (A,B:in bit ;  
    sum , Carry :out bit );  
  end component ;  
  
  begin  
    HA1 : half_adder port map(x,y, s1 ,c1 );  
    HA2 : half_adder port map(s1 ,cin , s,c2 );  
    cout <= c1 or c2;  
  end struct ;
```

Configuration

- A una entity possono corrispondere diverse architecture, magari differenti per il livello di astrazione della descrizione;
- L'associazione tra entity e architecture che si vuole effettivamente usare si fa attraverso un'ulteriore design unit, la configuration

```
configuration default_conf of full_adder is  
<corpo : associa alla entity un'architecture >  
end default_conf ;
```

Tipi di dato in VHDL

- Il VHDL dispone dei tipi bit e bitvector per la rappresentazione di dati digitali
- Solitamente comunque si preferisce usare i tipi std logic e std logic vector definiti nella library std logic 1164
- La library va sempre dichiarata nella maniera seguente:

```
library IEEE ;  
use IEEE.std_logic_1164.ALL;
```

Tipi IEEE std logic 1164

- I tipi definiti nella libreria `std_logic_1164` utilizzano un sistema logico a
- 9 valori:
 - '0': valore logico zero;
 - '1': valore logico uno;
 - 'Z': alta impedenza;
 - 'U': non inizializzato;
 - 'X': non noto;
 - '-': don't care;
- Dichiareremo allora i seguenti tipi IEEE std logic1164

```
library IEEE ;  
use IEEE.STD_LOGIC_1164.ALL;  
...  
signal x: std_logic;  
signal y: std_logic_vector (7 downto 0);
```

Circuiti combinatori

- Un circuito combinatorio può essere descritto da:
 - Un'espressione logica combinazione degli operatori booleani and, or, not. Il VHDL mette a disposizione tali operatori. Ad esempio:

```
y <= not(a) and b;
```

- Una tabella di verità anche non completamente specificata. In VHDL si utilizza un'assegnazione condizionale concorrente:

```
with x select  
y <= "01" when "00",  
  "10" when "11",  
  "  --" when others ;
```

Circuiti sequenziali

- Possono essere di tipo sincrono o asincrono. Nella progettazione digitale, a parte casi particolari, si preferisce usare sistemi sincroni.
- In VHDL la descrizione dei circuiti sequenziali è realizzata tramite process

```
count : process (x)
variable cnt: integer := -1;
begin
cnt := cnt + 1;
end process ;
```

Processi

- Un processo può essere dichiarato ovunque nel corpo dell'architettura di un modulo e viene pertanto eseguito parallelamente ad ogni altra istruzione concorrente.
- Un processo apre un'area sequenziale all'interno di un'area concorrente: le istruzioni contenute in un processo vengono infatti eseguite una dopo l'altra.
- L'esecuzione avviene comunque solo in corrispondenza di un evento specificato nella sensitivity list

Sensitivity list

- La Sensitivity List contiene tutti i segnali ai quali il processo deve essere sensibile
- Ogni volta che un segnale della sensitivity list cambia valore, il processo viene attivato
- Nel corpo del processo si utilizzano gli attributi dei segnali per distinguere i vari tipi di eventi.

Ad esempio:

```
-- Commutazione sul fronte di clock:  
(clock ='1' and clock'event)
```

Costrutti di area sequenziale

- All'interno di un processo le istruzioni VHDL vengono eseguite in sequenza, come in un normale linguaggio di programmazione.
- Si dispone ad esempio della scelta condizionata:

```
if condition then  
    sequential statements  
[ elsif condition then  
    sequential statements ]  
[ else  
    sequential statements ]  
end if;
```

Costrutti di area sequenziale

- Scelta multipla: e l'analogo del costrutto switch-case:

```
case expression is  
  when choice =>  
    sequential statements  
  when choice =>  
    sequential statements  
  [when others => seq. statements]  
end case ;
```

Costrutti di area sequenziale

- Cicli: individuati dalla keyword `loop`, hanno la seguente sintassi di base:

```
[loop_label:] loop  
sequential statements  
[next [label] [when condition];  
[exit [label] [when condition];  
end loop [loop_label];
```

- Esistono anche versioni più raffinate di questo costrutto (`while – loop`, `for – loop`)

Esempio: flip flop di tipo D

```
-- D type Flip Flop
entity dff is
port (data_in : in std_logic;
       clock   : in std_logic;
       data_out : out std_logic);
end dff;

architecture arch1 of dff is
begin

process (clock)
begin
    if (clock = '1' and clock'event) then
        data_out <= data_in;
    end if;
end process;
end arch1;
```

Strucutural modelling

- Solitamente si utilizzano le descrizioni structural level per definire sistemi complessi composti da sotto-sistemi precedentemente specificati.
- Progetti complessi fanno solitamente uso anche delle design unity finora non trattate: package e package bodies
- Si utilizza di solito un package per raggruppare tutti i componenti di base, da riutilizzare come sotto-sistemi per la definizione di sistemi più complessi

Istanziamento di componenti

- Una volta definito un certo sistema (e magari dopo averlo incluso in un package) è possibile riutilizzarlo in un sistema più complesso di livello superiore
- Come visto, un componente va dichiarato tramite la keyword `component` e istanziato tramite port map. Dello stesso componente possono ovviamente coesistere più istanze
- Esiste in VHDL'93 una sintassi alternativa che riunisce la dichiarazione e l'istanziamento di un componente:

```
U0: entity dut_ent (dut_arch) port map(a, b);
```

Componenti generici

- VHDL supporta la costruzione di componenti parametrici tramite il costrutto generic

```
entity half_adder is  
generic (N : integer );  
port (A, B: in std_logic_vector (0 to N-1);
```

- Nell'istanziamento specificheremo, oltre alla port map, anche una generic map

```
generic map (10 => N)
```

Simulazione

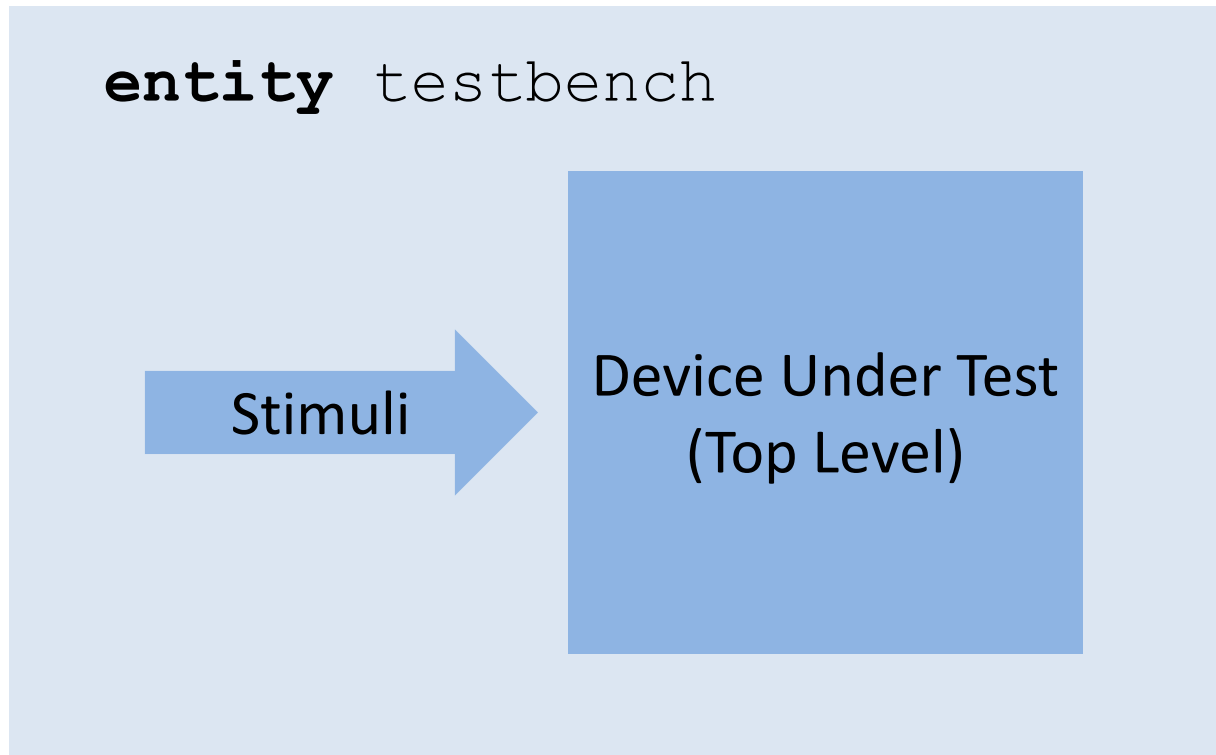
- La simulazione è il processo mediante il quale si verifica il corretto funzionamento del circuito modellato
- Essa può essere:
 - Funzionale: si utilizza il modello HDL, opportunamente sollecitato, per verificare l'effettivo funzionamento del circuito.
 - Temporale: si verificano le prestazioni vere e proprie del circuito.
- Questa simulazione è di solito possibile solo una volta definito il dispositivo target.

Testbenches

- La simulazione di un circuito richiede, oltre al modello dello stesso, anche la descrizione della sollecitazione da applicare.
- In VHDL si specifica la sollecitazione da applicare scrivendo un testbench, ovvero un modulo privo di ingressi ed uscite, che includa il sistema da simulare e la cui unica funzionalità è quella di applicare sugli ingressi di tale sistema i segnali voluti con la temporizzazione desiderata
- I testbench sono di solito l'unica parte del codice VHDL in cui si fa uso delle specifiche temporali (non sintetizzabili) e dell'istruzione `wait` all'interno dei `process`

```
1 y <= "00" after 5s;
```

Schema concettuale



Realizzazione di un testbench

- Un testbench è di solito l'entità di livello più alto: per tale motivo la sua entity non ha alcuna porta

```
entity testbench1 is  
end testbench1;
```

- Esso includerà come componente il dispositivo da testare e tanti segnali quanti gli ingressi da sollecitare:

```
component device_under_test is  
port (...);  
end device_under_test ;
```

- Si realizza un testbench collegando (tramite port map) i segnali dichiarati agli ingressi del circuito da provare e quindi facendo variare i segnali come necessario

Tool di simulazione

- Gli ambienti di sviluppo includono di solito un tool di simulazione o permettono di usarne uno esterno
- Uno degli strumenti di simulazione più noti è ModelSim di Mentor Graphics. Esistono versioni di questo tool ottimizzate per i dispositivi delle varie case produttrici ben integrate all'interno delle Design Suite.
- Un tool open source da prendere in considerazione: GHDL (Gnu HDL)

Funzionalità dei tool di simulazione

- Un tool di simulazione permette la verifica (functional o timing) di un sistema dato l'insieme dei segnali di ingresso.
- Oltre alla possibilità di utilizzare i testbench, molti simulatori permettono di specificare manualmente l'insieme delle sollecitazioni.
- I Waveform Generator sono di solito tool ad interfaccia grafica che consentono all'utente di costruire con relativa facilità i diagrammi temporali dei segnali in ingresso.

Tool di sintesi

- La sintesi è il processo di generazione di un file di configurazione dell'hardware (bitstream) a partire da una sua descrizione HDL.
- Le case produttrici di FPGA mettono di solito a disposizione opportuni tool di sintesi ottimizzati per i dispositivi target.
- Non necessariamente un modello HDL simulabile e anche sintetizzabile

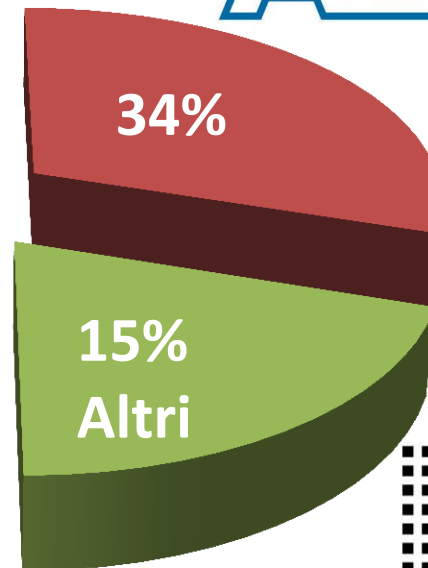
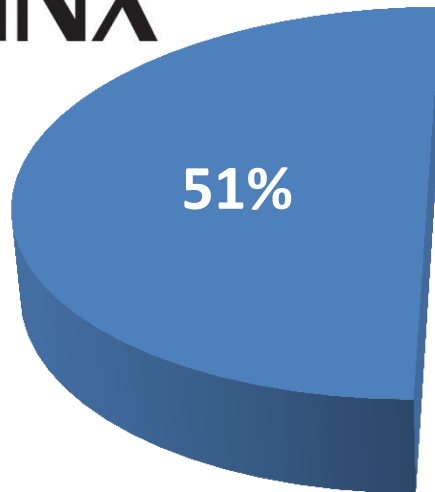
Tool di sintesi

- I tool di sintesi VHDL devono interpretare i costrutti del linguaggio per tradurli nella corrispondente implementazione circuitale
- Il problema fondamentale è appunto quello di fare in modo che il processo di traduzione sia corretto e il risultato ottenuto quello desiderato
- L'errore di sintesi più frequente è quello di ottenere nel circuito dei latch indesiderati

Vincoli

- Il processo di sintesi si basa, oltre che sul modello, anche su una serie di vincoli (constraints) fissati dall'utente
- Tramite i vincoli si può cercare di imporre al tool di sintesi un determinato comportamento, in modo da ottimizzare il risultato finale
- La maggior parte degli ambienti di sviluppo supporta la definizione dei vincoli in opportuni file (es. User Constraints Files, .ucf)

Manufacturers e quote di mercato



* dati aggiornati al 2010,
fonte: Xilinx Fact Sheet

Xilinx ISE Design Suite

- ISE Design Suite e l'ambiente di progettazione di Xilinx per lo sviluppo di sistemi su CPLD e FPGA.
- ISE è un ambiente integrato per la simulazione e la sintesi:
 - Simulazione: tramite ISim (ISE Simulator), con supporto a tool esterni (ModelSim, NC-Sim, VCS);
 - Sintesi: tramite XST (Xilinx Synthesis Technology), con supporto a tool esterni (Precision RTL, Synplify)
- Il concorrente Altera mette invece a disposizione l'ambiente di sviluppo Quartus II

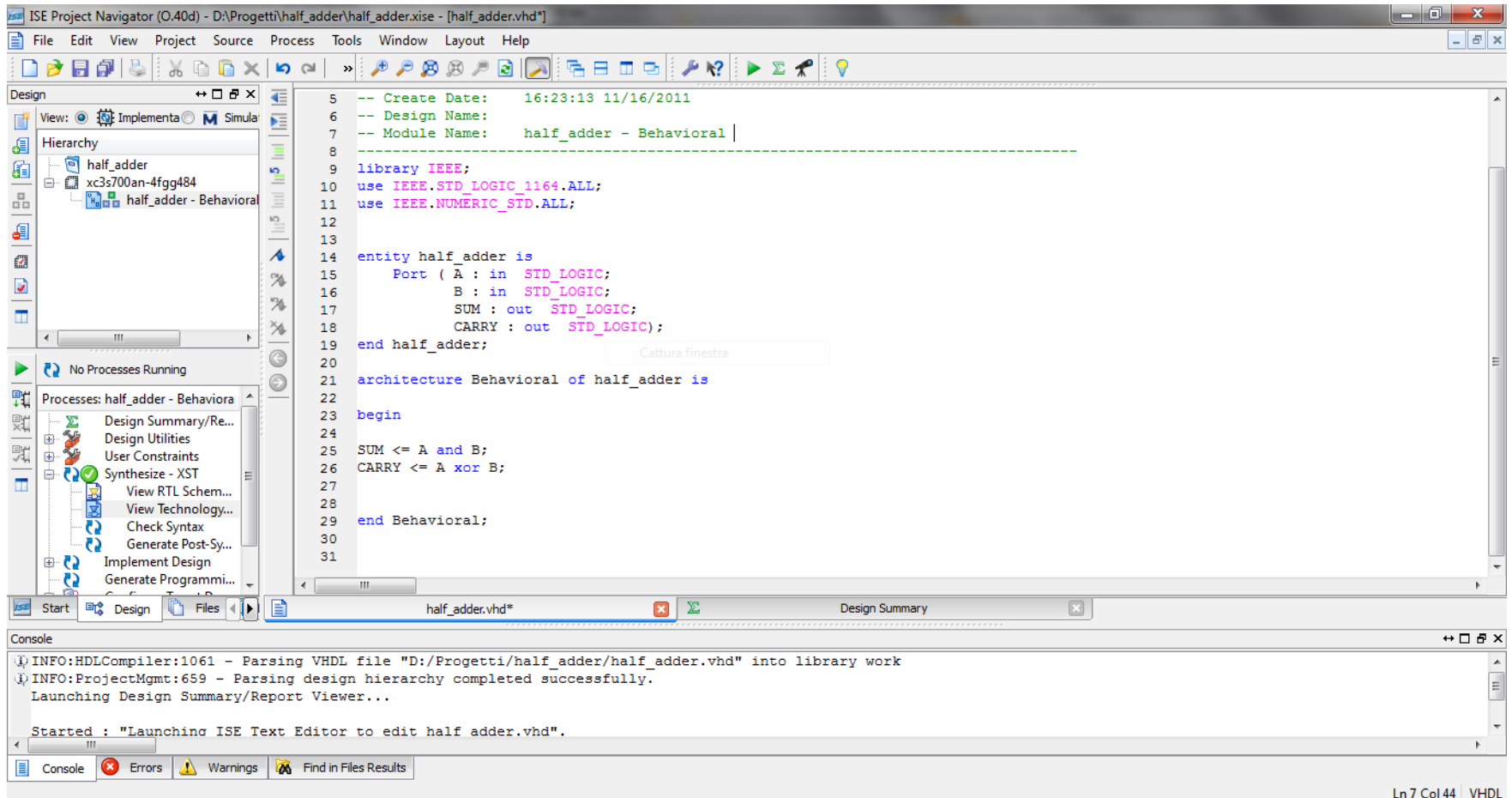
I tool di ISE Design Suite

- Strumenti di base: editor (con ambiente grafico in alternativa), tool di simulazione, tool di sintesi
- Sviluppo di sistemi embedded: implementazione rapida di system on chip tramite l'Embedded Design Kit
- Strumenti di gestione: gestione efficace del flusso di progetto attraverso PlanAhead
- Altre utility e strumenti: progettazione DSP tramite il System Generator, strumenti di analisi delle prestazioni o dell'area occupata, ecc.

Project Navigator

- Il Project Navigator è l'ambiente centrale della ISE Design Suite
- Costituisce l'entry point per scrittura del codice, la simulazione e la sintesi
- Permette inoltre una gestione coerente ed integrata degli altri strumenti software

Project Navigator



EDK, Embedded Development Kit

- EDK e l'ambiente di progettazione di system on chip della ISE Design Suite. Esso si compone di:
 - XPS: ambiente per la progettazione dell'hardware (sistemi a microprocessore, anche multicore)
 - SDK: ambiente di sviluppo software
- Alternativa Altera: Nios II Embedded Design Suite (EDS);

System Generator

- System Generator è un ambiente di sviluppo di applicazioni Digital Signal Processing su FPGA
- Il tool è sviluppato all'interno di Simulink e permette di sfruttare tutte le funzionalità dello stesso per lo sviluppo di hardware dedicato
- Oltre alla presenza di un gran numero di blocchi DSP immediatamente disponibili, una caratteristica interessante del system generator è la possibilità di sfruttare Matlab per generare vettori di sollecitazione
- System Generator supporta la co-simulazione hardware
- Anche Altera mette a disposizione un tool analogo (DSP Builder)

Vivado Design Suite

- La novità introdotta nell'ultima edizione della suite di sviluppo è il supporto alla high level synthesis
- ISE Design Suite è attualmente affiancato da un ulteriore pacchetto software, Vivado Design Suite, comprendente strumenti di sintesi da descrizioni C, C++ e SystemC

IP Cores

- Un vantaggio dell'uso degli ambienti di progettazione integrati sviluppati dalle case produttrici e la disponibilit  di IP Cores gi  pronti ed ottimizzati per i dispositivi disponibili
- Xilinx mette a disposizione vari IP cores all'interno dell'Embedded Development Kit, e un tool dedicato per i core dedicati al signal processing (Core Generator)
- Gli IP Cores sono ovviamente messi a disposizione, gratuitamente o a pagamento, anche da entit  diverse rispetto ai produttori di FPGA
- OpenCores.org rende disponibili gratuitamente (previa registrazione) un gran numero di IP Cores per molte aree applicative

Processori Embedded

- Xilinx mette a disposizione due tipologie di processori per lo sviluppo di sistemi embedded:
 - Hard-processor: PowerPC, disponibile in vari modelli di FPGA Xilinx e ARM, disponibile sulla piattaforma Zynq-7000
 - Soft-processor: MicroBlaze, PicoBlaze
- PowerPC e MicroBlaze sono direttamente supportati all'interno dell'Embedded Development Kit

PicoBlaze

- PicoBlaze è in realtà più una macchina a stati microprogrammata che un processore vero e proprio (KCPSM, Constant(K) Coded Programmable State Machine)
- Viene spesso usato all'interno del System Generator come elemento di controllo dei datapath sviluppati all'interno del tool
- Architettura RISC a 8 bit
- Programmabile in assembly (programmi di max 1024 istruzioni)

Riferimenti

- ANSI/IEEE Std 1076
- Peter J. Ashenden, The Designer's Guide to VHDL, Morgan Kaufmann Publishers, 2008
- Volnei A. Pedroni, Circuit Design and Simulation with VHDL - 2nd Edition, MIT Press, 2010
- Pong P. Chu, FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Version, Wiley, 2008

Risorse in rete

- vhdl.org
- OpenCores, sviluppo hardware open-source
- Vendors: Xilinx, Altera
- GHDL, simulatore con licenza GPL
- Simulatore ActiveHDL e Tutorial vari

Grazie per l'attenzione