Progetto di sistemi elettronici LA - esercitazioni

- Corso di Laurea in Ing. elettronica
- Esercitazioni
  - copie dei lucidi presentati a lezione
  - breve guida all'utilizzo di QUARTUS
  - codici VHDL
    - sommatore a 4 bit
    - FF con ingresso di reset sincrono o asincrono e enable
    - contatore
  - Soluzione guidata di prove d'esame

Grazie a
Prof.ssa Eleonora Franchi, Fabio Campi e Antonello Deledda

Esercitazioni con QUARTUS

- Breve guida all'utilizzo di QUARTUS
- Software scaricabile gratuitamente dalla rete nella versione *web edition*

  www.altera.com -> downloads and licensing -> Quartus II Web Edition (Free)

- Registrazione obbligatoria
- Dal menù di installazione scegliere installazione personalizzata e deselezionare tutte le voci tranne le prime 3 (obbligatorie) e tutti i supporti di libreria Stratix.

Contatti

- Sito di riferimento:

  www-micro.deis.unibo.it -> Staff -> Tommaso DE MARCO

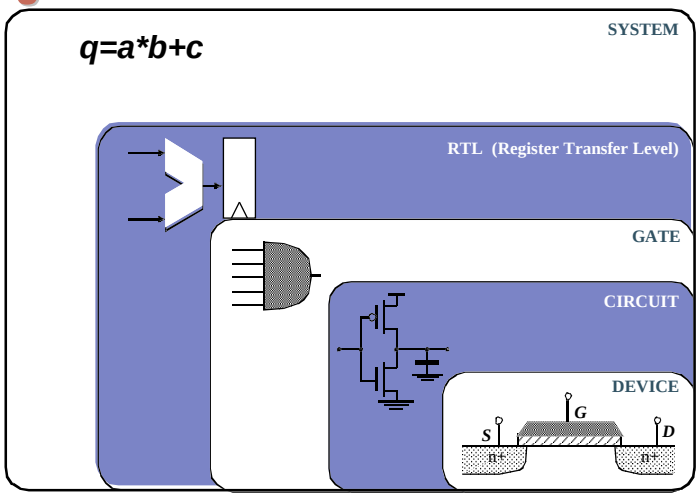
  - Materiale utilizzato durante le esercitazioni
  - Esercizi svolti e archivio temi d'esame
  - Informazioni utili
- Ricevimento: tdemarco@arces.unibo.it

Esame

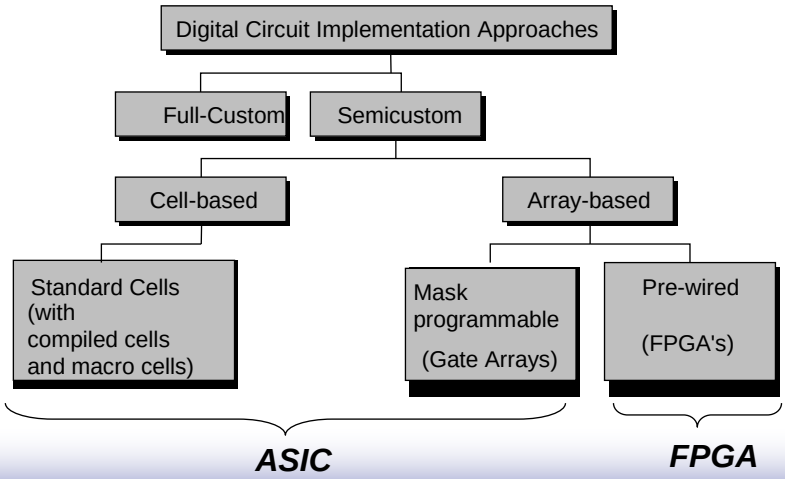
- si possono utilizzare supporti di memoria abilitati solo in lettura (CD ROM)
- non sarà visibile la propria area utente (il progetto sarà svolto nell'area tmp locale)
- si può consultare tutto il materiale cartaceo
- è accessibile la rete del DEIS

http://www.deis.unibo.it/
http://www-micro.deis.unibo.it

Livelli di astrazione di un sistema digitale



Integrated Circuits Implementation Choices



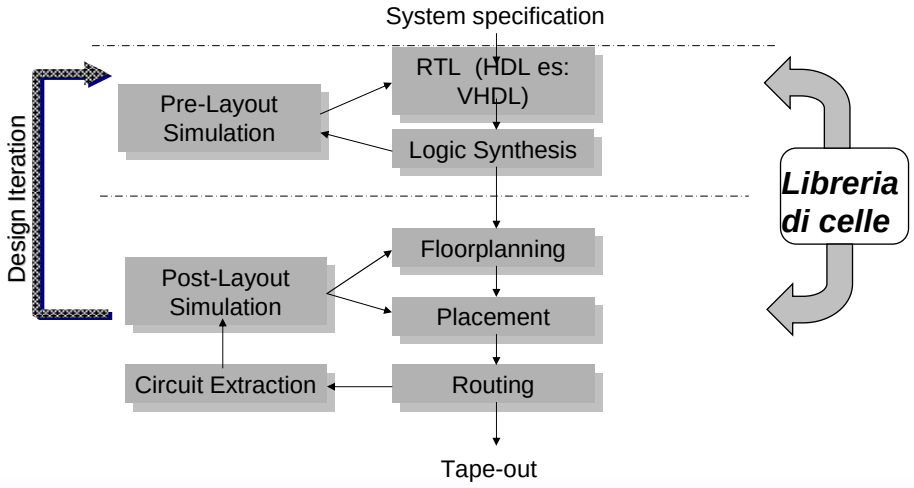
Semi-custom

- Il progettista può intervenire solo ai livelli più alti di astrazione (algoritmico e architetturale)
- Deve esistere una libreria di celle *progettate* e *caratterizzate* fornita dal costruttore
  - Caratterizzate: a ciascuna cella deve essere associato il valore dei parametri che permettono di calcolare Tp lungo il cammino critico, P e area di sistemi complessi
  - Le celle sono ovviamente state progettate a livello di transistor e il valore dei parametri per valutare le prestazioni è stato ricavato da simulazioni circuitali.

Full-custom

- Il progettista può ottimizzare a tutti i livelli di astrazione
- Mai attuabile per un intero sistema digitale, ma solo per moduli (macrocelle):
  - Riutilizzabili in differenti progetti (es: celle standard)
  - Con prestazioni particolarmente critiche per il sistema complessivo (es: ALU di un µP, MAC in DSP)
  - Regolari (es: memorie)
- In questo caso il progettista ha bisogno di modelli che permettano di scegliere fra differenti soluzioni. Deve poi disegnare lo schematico, simulare e caratterizzare la macrocella alle porte di IN/OUT affinché possa essere utilizzata in un flusso semi-custom.

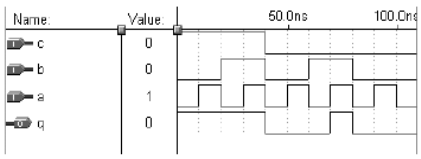
ASIC Design Flow



Flusso di Sviluppo di circuiti digitali : FRONT END

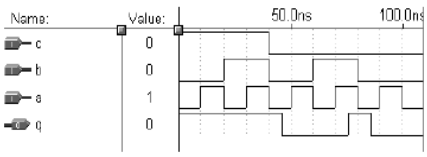
1) Definizione Algoritmica (linguaggio C): q=a * b + c;

2) Descrizione VHDL del circuito: q <= a and b or c;



3) Simulazione Funzionale

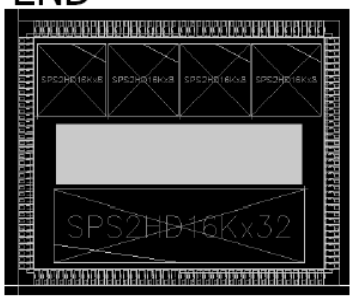
4) Sintesi Logica

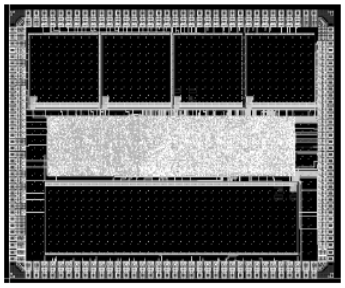


5) Simulazione Post-Sintesi

Flusso di Sviluppo di circuiti digitali: BACK END

6) Floorplanning



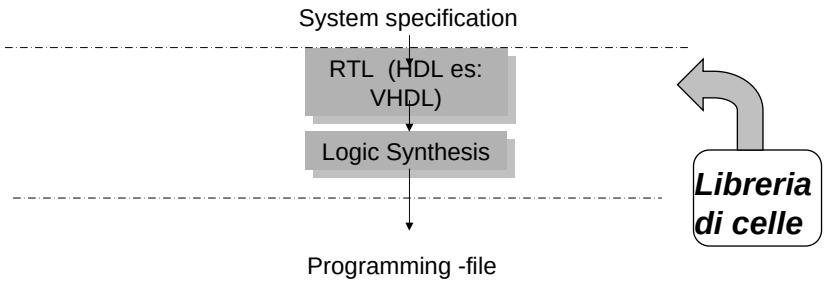


7) Place & Route

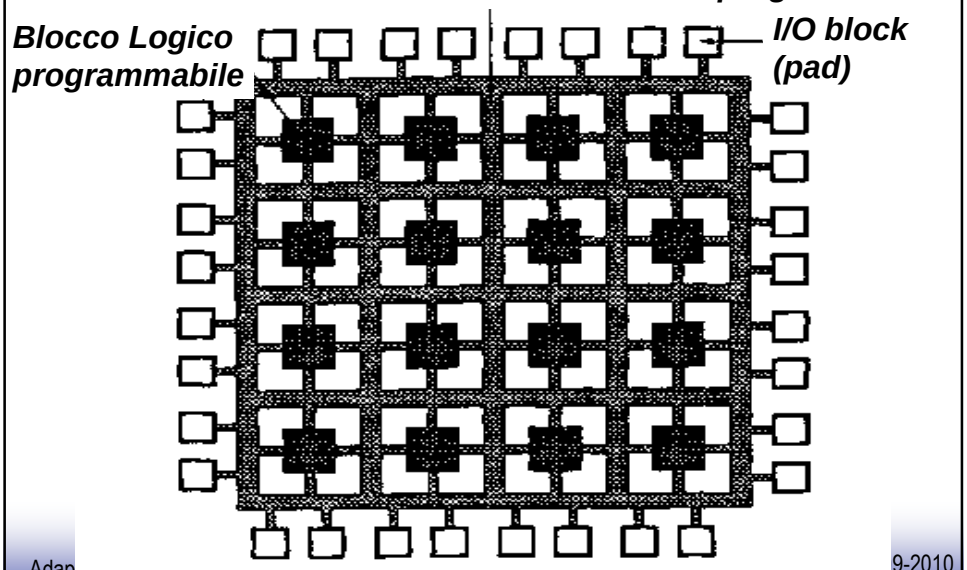
8) Parasitic extraction & backannotation

9) Simulazione post-layout

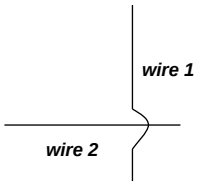
FPGA Design Flow



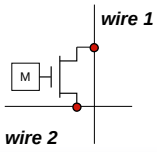
FPGA



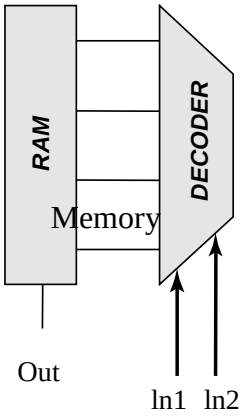
Programming Interconnect Technique based on SRAM or Flash



Memory M stores the gate voltage of the MOS transistor

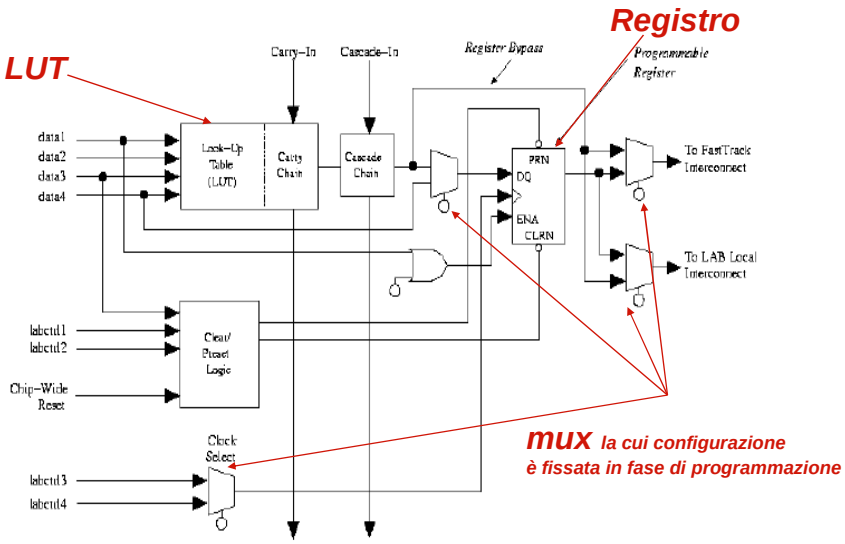


Programmable Logic Block based on Look-up Table (LUT)



In	Out
00	0
01	1
10	1
11	0

Flex10KE: Logic Element (LE) *(from Altera)*



Flusso di progetto in laboratorio

- boot Windows
- Creare un direttorio di lavoro
- QuartusII9.sp2
- Descrizione vhdl del sistema
- Verifica funzionale
- Sintesi del sistema su fpga
- Verifica timing

Flusso di progetto in laboratorio

- Descrizione RTL del circuito utilizzando il linguaggio VHDL (.vhd)
  - Verifica codice: *start analysis and elaboration*
  - Analisi dello schematico associato al file: *RTL viewer*
- Simulazione funzionale
  - Creazione netlist: *generate functional simulation netlist*
  - Creazione file con forme d'onda (.vwf)
  - Specificare: *simulation mode = functional*
  - Simulazione: *start simulation*

- Sintesi logica su FPGA
  - Sintesi: *start compilation*
    - *Nel summary compaiono i dati relativi all'utilizzo delle risorse*
    - *Nel project navigator la lista degli elementi utilizzati (Logic Cell, Logic Cell Register,Pin,..)*
    - *Nella finestra di log compare già l'informazione della massima frequenza a cui il circuito può funzionare*
      - *Info: Clock "CLOCK" has Internal fmax of 174.86 MHz between source register "cicli[7]" and destination register "cicli[13]" (period= 5.719 ns)*
- Simulazione post-sintesi (timing)
  - Specificare: *simulation mode = timing*
  - Simulazione: *start simulation*
  - Analisi delle prestazioni*: start timing analyzer*

Menu Assignments ->Settings:

- Files: permette di selezionare fra differenti file .vhd all'interno dello stesso progetto l'unico attivo (eseguire *remove* e lasciare un solo file .vhd)
- General: permette di definire la *top-level entity* (solo dopo che sia stato eseguito il comando *start analysis and elaboration* il sistema riconosce automaticamente tutte le entità). Di default il nome della top-level entity che il sistema si aspetta coincide con il nome del *project*
- Simulator: permette di definire il "tipo" di simulazione (*functional o timing*) e di specificare il file con gli stimoli (.vwf)

Suggerimenti per la simulazione

- Mantenere stabili le configurazioni degli ingressi per un tempo sufficientemente lungo da assicurarsi che nelle simulazioni *timing* i transitori siano esauriti prima della nuova commutazione degli ingressi
  - Il periodo di un segnale di tipo clock ha di default assegnato un periodo di 10 ns (per simulazioni timing NON è in generale sufficiente)
- Controllare la durata della simulazione assegnando un valore ragionevole a *end time*
  - *Attenzione: di default è 1 us*
  - *Il valore ragionevole è definito dal numero di configurazioni distinte in ingresso che fornite*
- Controllare *grid time (a metà del Tck)*

Struttura file .vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
```

link a librerie e package

```
entity nome_del_modulo is
port ( term1,term2 : in std_logic;
       term3,term4 : out std_logic) ;
end nome_del_modulo;
```

entity
interfaccia I/O del modulo

```
architecture tipo_architettura of nome_del_modulo is
begin
  .......
end tipo_archiettura;
```

architecture
descrizione del circuito (comportamentale o strutturale)

Esempio

```
library IEEE;
use IEEE.std_logic_1164.all;


entity HA is
port ( I1,I2 : in std_logic;
       SUM, CO : out std_logic);
end HA;

architecture BEHAVIOR of HA is
begin
  SUM <= (I1 xor I2);
  CO <= (I1 and I2);
end BEHAVIOR;
```

assegnamenti eseguiti simultaneamente

Tipi di dato

- std_logic e std_logic_vector
  richiedono il link al package
  use IEEE.std_logic_1164.all;

- unsigned/signed
  richiedono il link ai package

  use IEEE.std_logic_unsigned.all;
  use IEEE.std_logic_signed.all;
  use IEEE .std_logic_arith.all;

Operazioni logiche

- not, and, or, nand, nor, xor

operandi std_logic

Operatori relazionali

- = uguale
- /= diverso
- > maggiore
- >= maggiore o uguale
- < minore
- <= minore o uguale

operandi *signed, unsigned*

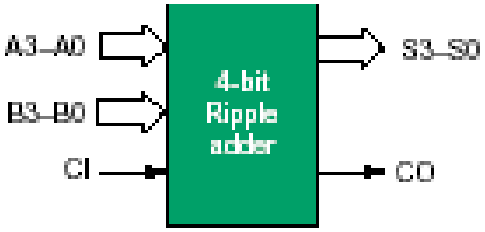
OPERAZIONI ARITMETICHE

- + - *

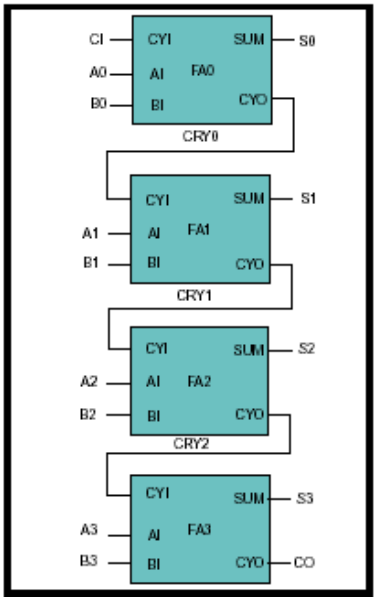
(e .. pagina 39 manuale)

- somme: Quartus richiede che gli operandi e il risultato siano espressi dallo stesso numero n di bit
- moltiplicazioni: operandi a n bit, risultato a 2*n bit
- su operandi di tipo *signed* o *unsigned*

Example: 4 bit adder



The Ripple-Carry Adder



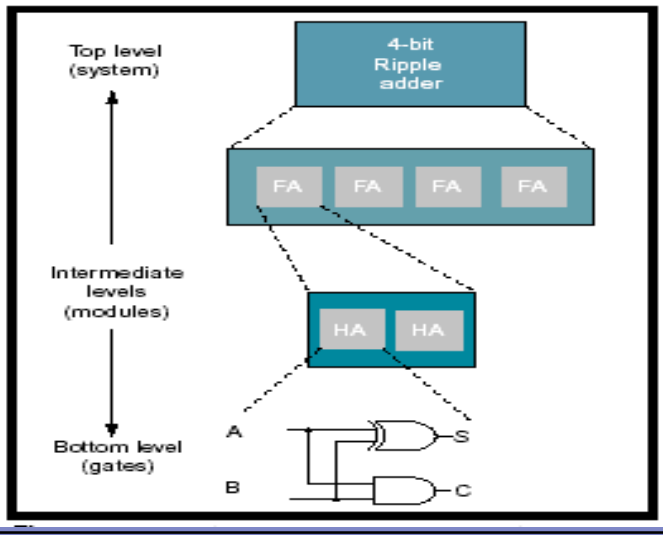
Full-Adder

$Sum_i = A_i \oplus B_i \oplus Cin_i = P_i \oplus Cin_i$

$Cout_i = A_iB_i + Cin_i(A_i \oplus B_i) = G_i + Cin_i P_i$

A	B	C_i	S	C_o	*Carry status*
0	0	0	0	0	delete
0	0	1	1	0	delete
0	1	0	1	0	propagate
0	1	1	0	1	propagate
1	0	0	1	0	propagate
1	0	1	0	1	propagate
1	1	0	0	1	generate
1	1	1	1	1	generate

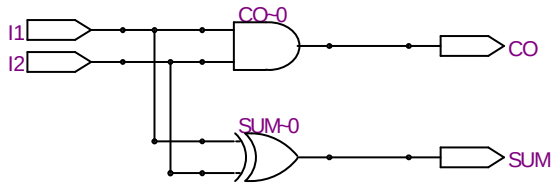
Progetto gerarchico



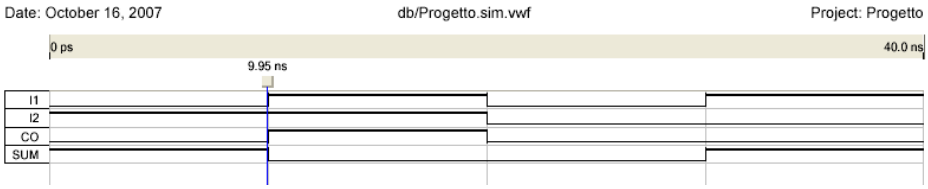
VHDL description of an Half - Adder

```
library IEEE;
use IEEE.std_logic_1164.all;


entity HA is
port ( I1,I2 : in std_logic;
       SUM, CO : out std_logic);
end HA;

architecture BEHAVIOR of HA is
begin
   SUM <= (I1 xor I2);
   CO <= (I1 and I2);
end BEHAVIOR;
```

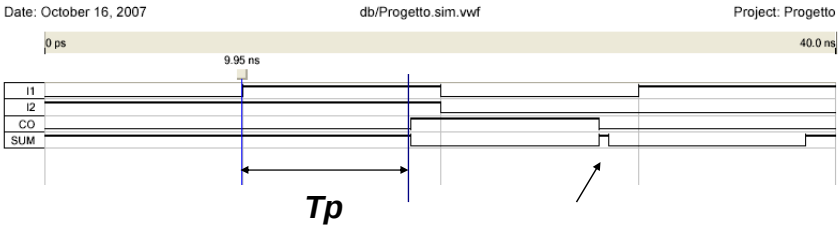

RTL viewer



Simulazione funzionale



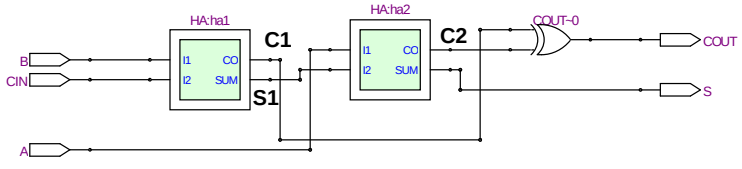
Simulazione dopo la sintesi



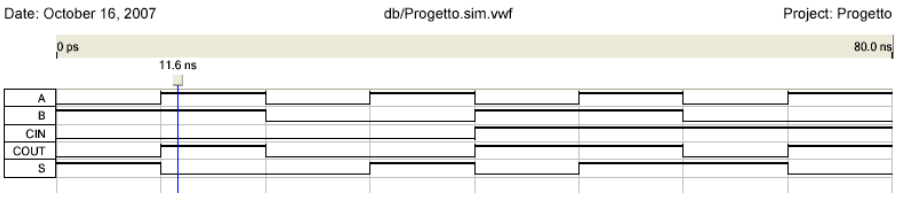
Structural description of a Full-Adder

```
library IEEE;
use IEEE.std_logic_1164.all;

entity FA is
port ( A,B,CIN : in std_logic;
       S, COUT : out std_logic);
end FA;

architecture STRUCTURAL of FA is
  component HA                          dichiarazione del componente HA
  port ( I1,I2   : in std_logic;
         SUM, CO : out std_logic);
  end component;
signal S1, C1, C2 : std_logic;          definzione di segnali interni

begin
  ha1 : HA port map( I1 => B, I2 => CIN, SUM => S1, CO =>C1); istanzia i moduli
  ha2 : HA port map( I1 => A, I2 => S1, SUM => S, CO => C2);

   COUT <= C2 xor C1;      Att: nel file .vhd deve esserci la descrizione di
                           tutti I moduli della gerarchia
   end STRUCTURAL;
```

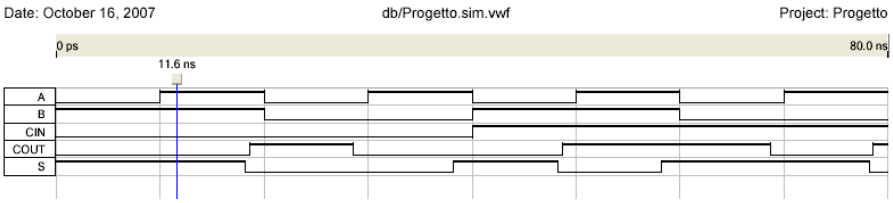

RTL viewer



Simulazione funzionale



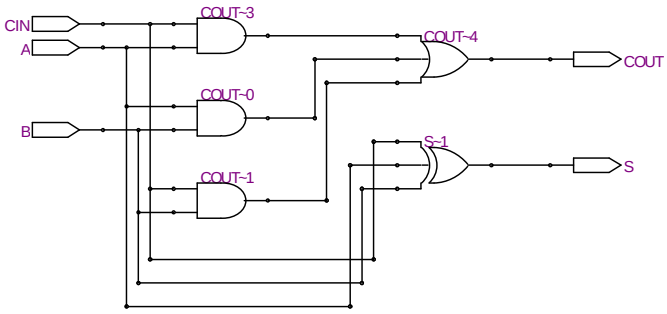
Simulazione dopo la sintesi



Behavioral description of a Full-Adder


```
library IEEE;
use IEEE.std_logic_1164.all;


entity FA is
port ( A,B,CIN : in std_logic;
       S, COUT : out std_logic);
end FA;

architecture BEHAVIOR of FA is
begin
   S <= (A xor B) xor CIN;
   COUT <= (A and B) or (B and CIN) or (A and CIN);
end BEHAVIOR;
```


RTL Viewer

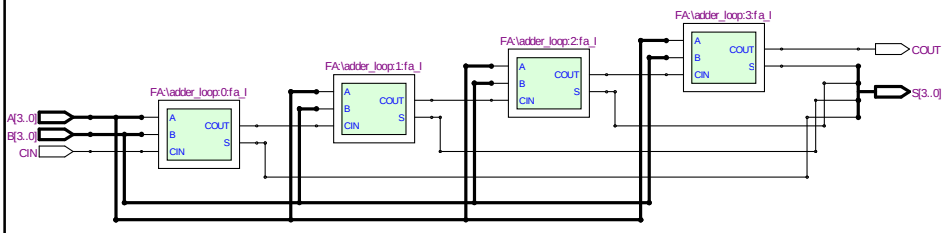


Structural description of the Adder

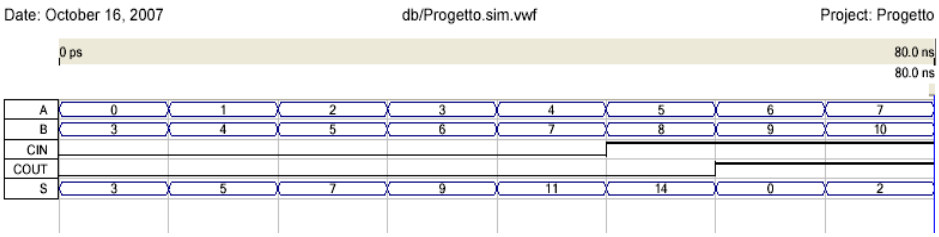
```
library IEEE;
use IEEE.std_logic_1164.all;

entity ADDER4 is
port ( A, B : in  std_logic_vector(3 downto 0);
       CIN  : in  std_logic;
       COUT : out std_logic;
       S    : out std_logic_vector(3 downto 0) );
end ADDER4;

architecture STRUCTURAL of ADDER4 is
  component FA
  port ( A,B,CIN : in std_logic;
         S, COUT : out std_logic);
  end component;
signal K : std_logic_vector(4 downto 0);
begin

   adder_loop : for I in 0 to 3 generate
     fa_I : FA port map ( A => A(I), B => B(I), CIN => K(I), COUT => K(I+1), S => S(I) );
   end generate;

  K(0) <= CIN;
   COUT <= K(4);
end STRUCTURAL;
```

Att: nel file .vhd deve esserci la descrizione di tutti I moduli della gerarchia

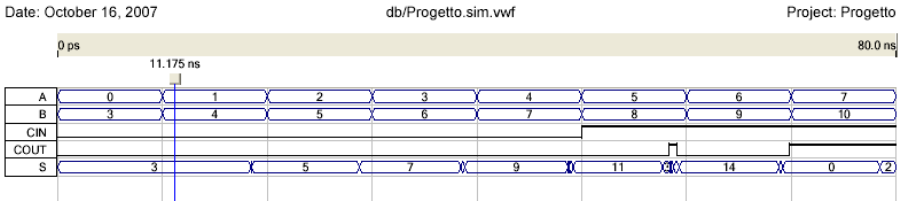
RTL Viewer



Simulazione funzionale



Simulazione dopo la sintesi

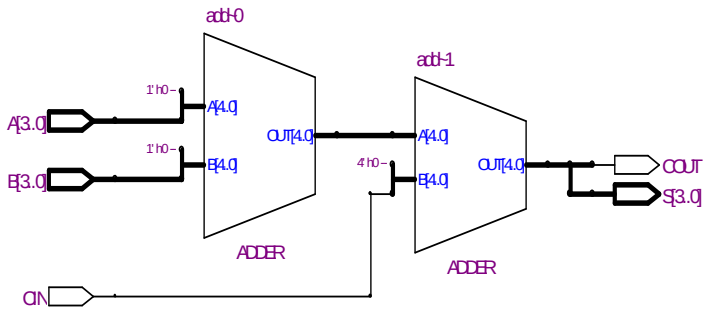


Behavioral description of the Adder: 1

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity ADDER4 is
port ( A, B : in  unsigned (3 downto 0);
       CIN  : in  std_logic;
            COUT : out std_logic;
       S    : out unsigned(3 downto 0) );
end ADDER4;

architecture BEHAVIORAL of ADDER4 is

signal K : unsigned (4 downto 0);
signal Aint, Bint, Cint : unsigned(4 downto 0);
begin
   Aint <=conv_unsigned(A,5);
   Bint <=conv_unsigned(B,5);
   Cint <= conv_unsigned(CIN,5);
   K <= Aint+Bint+Cint;

   S <= K(3 downto 0);
   COUT <= K(4);
end BEHAVIORAL;
```

somme: Quartus richiede che gli operandi e il risultato siano rappresentati con lo stesso numero di bit

RTL Viewer



Behavioral description of the Adder: 2

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;

entity ADDER4bis is
port ( A, B : in  unsigned (4 downto 0);
       CIN  : in  std_logic;
              COUT : out std_logic;
       S    : out unsigned(3 downto 0) );
end ADDER4bis;

architecture BEHAVIORAL of ADDER4bis is

signal K : unsigned (4 downto 0);
signal Cint : unsigned(4 downto 0);

begin

   Cint <= conv_unsigned(CIN,5);
   K <= A+B+Cint;

   S <= K(3 downto 0);
   COUT <= K(4);

end BEHAVIORAL;
```


RTL Viewer