www.delucagiovanni.com

Corsi, Forum, Docs – Robotics and A.I.

Presentazione del corso:

Progettazione elettronica avanzata con l'uso di Tools specializzati.

Giovanni De Luca

Sito personale e forum

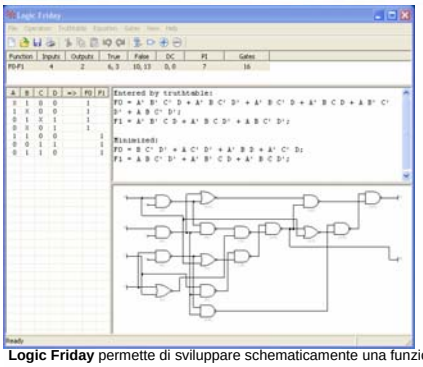
…….iniziamo

- Installazione dei seguenti software:

- **Logic Friday** (free)
- **Logic Works 5** (free)
- **Proteus ISIS** (no free - demo)
- **Quartus II v9** (student version)

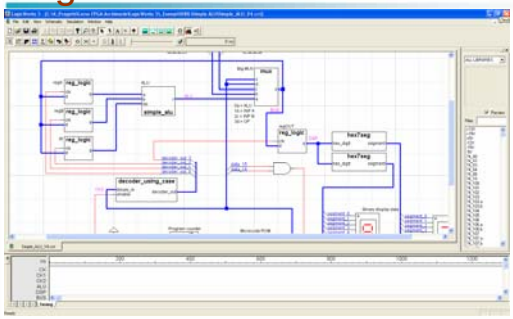
I software si trovano sulla cartella "SW_Corso_FPGA"

Logic Friday



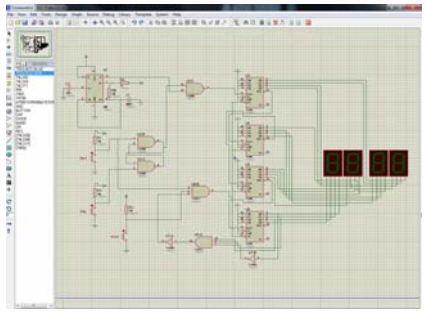
Logic Friday permette di sviluppare schematicamente una funzione logica combinatoria a partire dalla tabella della verità di n. variabili ingresso/uscita. La funzione può essere quindi minimizzata e convertita in schema Logico con la possibilità di scegliere il tipo di porte logiche da utilizzare nello schema finale.

LogicWorks 5



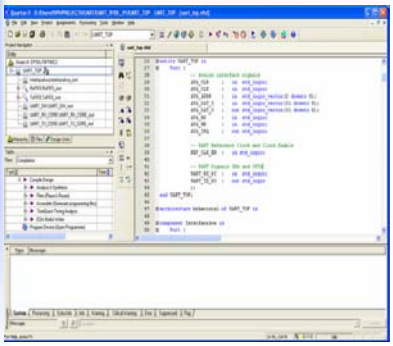
LogicWorks è un tool per lo sviluppo di circuiti elettronici interattivi a logica digitale.
Il pacchetto dà la potenza, la velocità e la flessibilità per creare e testare innumerevoli circuiti digitali direttamente sullo schermo del vostro computer.
Questo significa che si possono studiare concetti molto complessi in modo chiaro usando la simulazione senza spendere soldi, tempo e soprattutto senza danneggiare componenti.

Proteus ISIS



Proteus VSM è un software per la simulazione di circuiti elettronici analogici e digitali.
Esso si basa su modelli realizzati in PSpice. ISIS ha la possibilità di simulare microprocessori di quasi tutte le case produttrici. Inoltre può simulare circuiti digitali realizzati con integrati TTL e CMOS. Supporta il 95% di questi integrati con i quali realizzeremo gli esempi che man mano verranno presentati nel corso.

Quartus II – Tool per FPGA



QUARTUS è un tool utilizzabile per effettuare, nell'ambito della progettazione di circuiti digitali:

- Sintesi logica
- Simulazione digitale
- Place and Route
- Analisi delle prestazioni

In questo corso verranno illustrati e commentati i vari passi che sarà necessario compiere nell'ambito del flusso di progetto di circuiti digitali tramite linguaggio VHDL.
QUARTUS è composto da diversi tools (Compiler, Simulator, Text Editor, etc.) ognuno dei quali serve per una fase specifica del flusso di progetto.

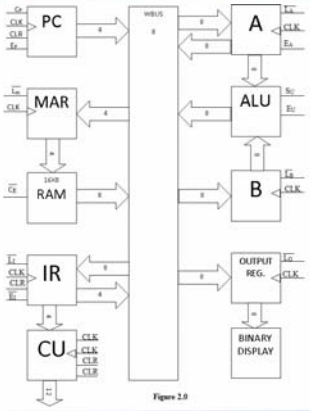
Finalità del corso

Apprendere le nozioni di base per la realizzazione di una CPU

SAP-1

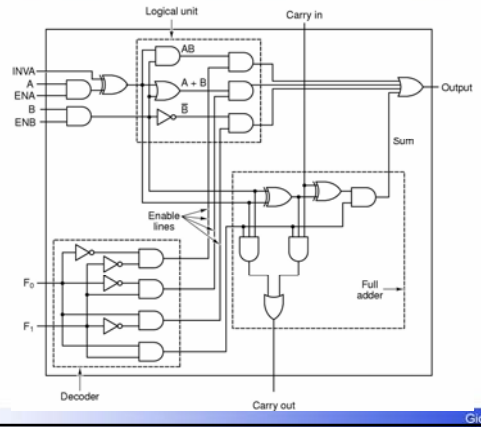
Simple as Computer 8bit

Schema a blocchi di una SAP

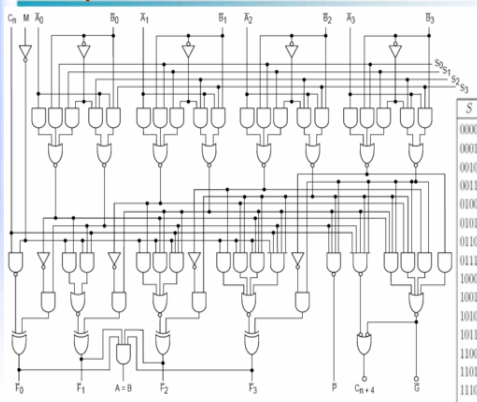


1. Program Counter (PC)
2. Memory Address Registers (MAR)
3. Random Access Memory (RAM)
4. Instruction Register (IR)
5. Controller Sequencer (CU)
6. Accumulator (A)
7. Adder-Subtracter (ALU)
8. B-Registers (B)
9. Output Register
10. Binary Display

Simple 2bit ALU



Simple 4bit ALU



S	Aritmético ($M=0$)	Lógico ($M=1$)
0000	$A+C_n$	$\overline{A}$
0001	$A\overline{B}+C_n$	$\overline{A\overline{B}}$
0010	$A\overline{B}+C_n$	$\overline{AB}$
0011	$-1+C_n$	0
0100	$A+A\overline{B}+C_n$	$\overline{AB}$
0101	$(A\overline{B})+A\overline{B}+C_n$	$\overline{B}$
0110	$A-B-1+C_n$	$A\oplus B$
0111	$A\overline{B}-1+C_n$	$A\overline{B}$
1000	$A+AB+C_n$	$\overline{A}B$
1001	$A+B+C_n$	$\overline{A\oplus B}$
1010	$(A\overline{B})+AB+C_n$	B
1011	$AB-1+C_n$	AB
1100	$A+A+C_n$	1
1101	$(A\overline{B})+A+C_n$	$A\overline{B}$
1110	$(A\overline{B})+A+C_n$	$A\overline{B}$
1111	$A-1+C_n$	A

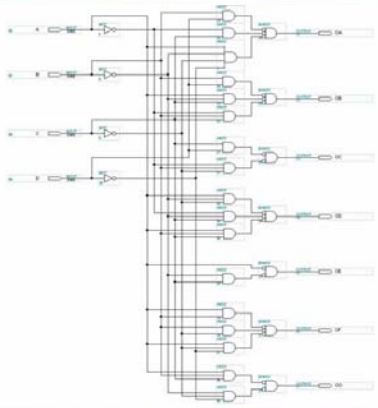
BCD to 7 segment

Truth table for Hex to 7 Segment decoder

ABCD	Segments						
Bcd(3:0)	a	b	c	d	e	f	g
0000	0	0	0	0	0	0	1
0001	1	0	0	1	1	1	1
0010	0	0	1	0	0	1	0
0011	0	0	0	0	1	1	0
0100	1	0	0	1	1	0	0
0101	0	1	0	0	1	0	0
0110	0	1	0	0	0	0	0
0111	0	0	0	1	1	1	1
1000	0	0	0	0	0	0	0
1001	0	0	0	0	1	0	0
1010	0	0	0	1	0	0	0
1011	1	1	0	0	0	0	0
1100	0	1	1	0	0	0	1
1101	1	0	0	0	0	1	0
1110	0	1	1	0	0	0	0
1111	0	1	1	1	0	0	0

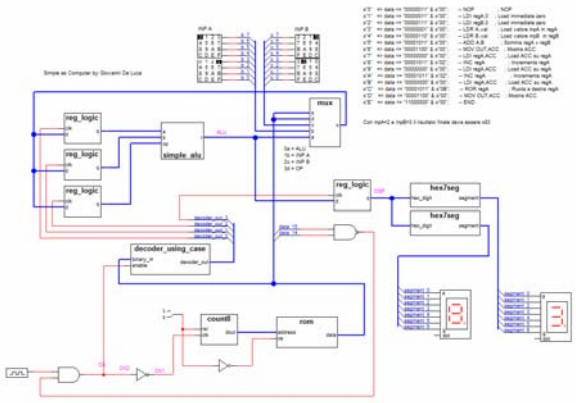
s	BCD	G	F	E	D	C	B	A
0	0000	0	1	1	1	1	1	1
1	0001	0	0	0	0	1	1	0
2	0010	1	0	1	1	0	1	1
3	0011	1	0	0	1	1	1	1
4	0100	1	1	0	0	1	1	0
5	0101	1	1	0	1	1	0	1
6	0110	1	1	1	1	1	0	1
7	0111	0	0	0	0	1	1	1
8	1000	1	1	1	1	1	1	1
9	1001	1	1	0	1	1	1	1

BCD to 7 segment decoder



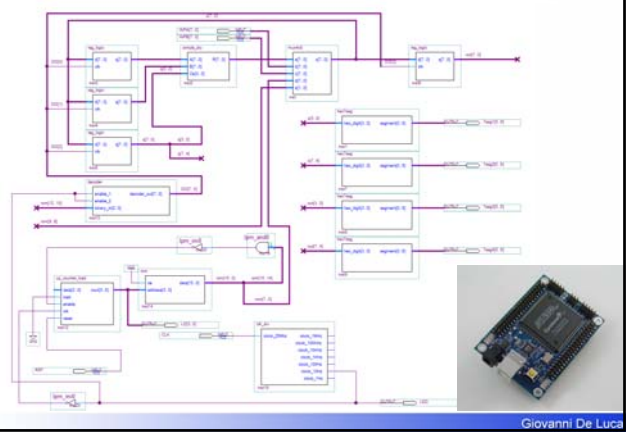
Giovanni De Luca

My Simple as Computer (LW5)



Giovanni De Luca

My Simple as Computer (Quartus)



Giovanni De Luca

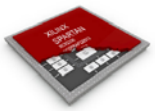
Di cosa parliamo ?

In questa lezione si descrivono i dispositivi logici programmabili

- Dispositivi logici programmabili
- Struttura degli FPGA
- VHDL

Elementi di calcolo

- Vari tipi di elementi di calcolo per sistemi embedded:
  - ASIC
  - Microcontrollori
  - Microprocessori
  - Dispositivi logici programmabili
- Le logiche programmabili sono spesso convenienti
  - economiche
  - efficienti
  - riconfigurabili / aggiornabili



Dispositivi logici programmabili

Le *logiche programmabili* (o *PLD*, Programmable Logic Device) integrano risorse logiche e linee di interconnessione

Sono chip *programmabili* nel senso che:

- ciascuna risorsa logica può essere configurata per svolgere una specifica funzione
- ciascuna linea di interconnessione può essere collegata o meno a varie risorse logiche

Esistono differenti gradi di *programmabilità*:

- one-time programmable (OTP): la configurazione del chip è irreversibile ed è ottenuta applicando tensioni elettriche più alte di quelle della normale alimentazione
- riprogrammabili: la configurazione può essere effettuata più volte; le interconnessioni sono transistor pilotati dai bit di un circuito di memoria volatile (RAM statica) oppure persistente (EEPROM, Flash)
- riconfigurabili: la configurazione può essere effettuata più volte mentre il circuito è in funzione ed in modo selettivo

Complessità e organizzazione dei PLD

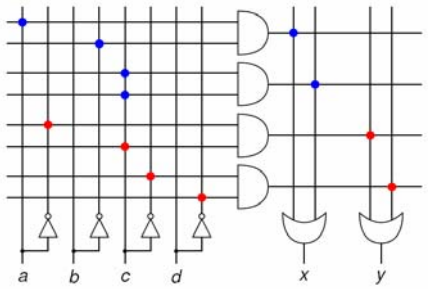
I PLD si distinguono anche in base alla complessità delle risorse logiche elementari (*celle*)

- Ad un estremo, una cella è costituita da una o due porte logiche, oppure da un multiplexer a 2 o 4 ingressi, oppure da un latch o flip-flop
  - Vantaggio: Il silicio è maggiormente sfruttato, perché il rischio di sotto-utilizzare una cella è ridotto
- All'altro estremo, una cella contiene una o due circuiti in grado di realizzare qualunque funzione booleana a 4 ingressi, qualche porta logica, uno o due multiplexer e qualche flip-flop
  - Vantaggio: per realizzare la logica dell'applicazione occorrono meno celle complesse, perciò le interconnessioni sono più corte e facili da realizzare

Tipologia di PLD

- PLA (Programmable Logic Array): costituito da zone di porte AND e OR e da aree di interconnessione configurabili (OTP)
- PAL (Programmable Array Logic): come i PLA, ma più semplici: la configurazione delle porte OR è prefissata
- GAL (Generic Array Logic): include diversi PAL, con multiplexer per retro-azionare le uscite, e con flip-flop; inoltre può essere riprogrammato
- CPLD (Complex PLD): basato su celle complesse (GAL) ed un bus comune configurabile per mezzo di una memoria EEPROM
- FPGA (Field Programmable Gate Array): celle complesse e interconnessioni sono distribuite regolarmente nel chip; la configurazione è in memoria generalmente volatile

Funzionamento di un PLA



- $x = (a \wedge \overline{b}) \vee c$
- $y = (\overline{a} \wedge c) \vee (\overline{c} \wedge \overline{d})$

Tecnologia di interconnessione

I PLD sono programmati configurando opportunamente le connessioni elettriche del chip

Diverse tecnologie per realizzare le interconnessioni:

- OTP: One Time Programmable
  - Fuse technology: connessioni normalmente chiuse, una corrente elevata fonde il collegamento ed apre il circuito
  - Antifuse technology: connessioni normalmente aperte, l'applicazione di una tensione elevata allinea gli atomi di silicio amorfo e li trasforma in silicio policristallino conduttore
- Riprogrammabili, Riconfigurabili
  - Ciascuna connessione è controllata da un transistor pilotato dalla linea di uscita di un bit di memoria statica, EEPROM o flash

Tipicamente i PLD più semplici sono OTP, quelli più complessi sono riprogrammabili e riconfigurabili

FPGA

- FPGA: Field Programmable Gate Array
- Il primo FPGA è stato costruito dalla Xilinx nel 1985
- Tradizionalmente utilizzati in
  - elaborazione di segnali digitali
  - sistemi aereospaziali
  - sistemi di difesa
  - prototipizzazione di ASIC
  - crittoanalisi
  - bio-informatica
  - calcolo scientifico
  - ...
- Maggiori venditori:
  - Xilinx (quota di mercato: > 50%)
  - Altera (quota di mercato: ~ 30%)
  - Lattice Semiconductor, Actel, SiliconBlue Technologies, Achronix, QuickLogic, ...
- Diffusione sempre crescente!

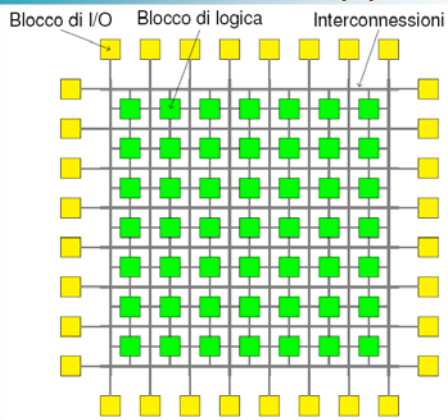
Struttura di un PFGA

Un FPGA è costituito da:

- Un insieme di interfacce di I/O digitali
- Un insieme di blocchi di logica programmabile distribuiti lungo la superfice del chip
  - Xilinx li chiama CLB (Configurable Logic Block)
  - Altera li chiama LAB (Logic Array Block)
- Unità di calcolo specializzate (ad es, moltiplicatori)
- Blocchi di RAM statica
- Collegamenti di interconnessione tra i vari elementi dell'FPGA

I dettagli variano con il venditore ed il modello

Struttura di un FPGA (2)



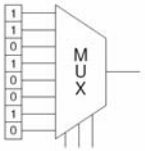
Gerarchie di interconnessione

- I blocchi di logica programmabili sono distribuiti sulla superficie di silicio del chip
- Un dato può essere trasferito da un blocco ad un altro utilizzando interconnessioni programmabili
- Esiste una gerarchia di inteconnessione:
  - Blocchi logici vicini sono collegati tramite interconnessioni dotate di bassa latenza di trasmissione
  - Gruppi locali di blocchi logici sono collegati tramite interconnessioni di capacità maggiore chiamate linee di routing

Celle logiche

Ogni blocco di logica programmabile (CLB o LAB) è costituito da tipicamente poche celle logiche

- Sono i circuiti più piccoli individualmente programmabili
- Xilinx: LC (Logic Cell), Altera: LE (Logic Element)
- Ciascuna cella è tipicamente costituita da
  - uno o due LUT (Lookup Table)
  - un circuito addizionatore con riporto
  - alcuni elementi di memoria (flip-flop)
  - alcuni multiplexer
- Una LUT con n ingressi realizza una funzione combinatoria di arità n
  - È costituita da 2^n celle SRAM (flip-flop) ed un multiplexer
  - Generalmente $n \in \{2, \ldots, 6\}$
  - I flip-flop possono essere usati anche come registro



Caratteristiche costruttive di un FPGA

Principali caratteristiche per la scelta di un FPGA:

- Supporto fisico, ad esempio:
  - numero di pin del chip
  - integrazione su scheda con bus ad alta velocità per l'interfacciamento ad un calcolatore
- Consumo di energia e potenza
- Densità e quantità dei blocchi logici
- Quantità di RAM utilizzabile
  - contenuta nelle LUT
  - integrata nel chip come blocchi specializzati
- Funzionalità implementate in hardware
- Disponibilità di IP già pronte
- Costi:
  - del chip
  - degli strumenti di sviluppo
  - delle IP

Programmazione di un FPGA

- Gli FPGA sono programmati, nel senso che lo sviluppatore definisce completamente il funzionamento del chip
- Quando si programma un microprocessore od un microcontrollore lo sviluppatore definisce le istruzioni macchina eseguite dall'unità di calcolo
  - "Programmare" significa costruire il software
- Quando si programma una dispositivo di logica programmabile (FPGA) lo sviluppatore definisce il funzionamento dei circuiti interni del dispositivo
  - "Programmare" significa definire l'hardware
- Per programmare l'hardware si utilizzano linguaggi chiamati generalmente HDL (Hardware Description Language)
- I due linguaggi HDL più diffusi:
  - VHDL (Europa, Asia)
  - Verilog (USA)

Il linguaggio VHDL

- Nasce nel 1980 per aiutare il ministero della difesa USA a documentare il funzionamento degli ASIC
- Successivamente sono stati creati dei simulatori logici che consentono di simulare l'hardware descritto da VHDL
- Ancora più tardi sono stati creati dei sintetizzatori logici in grado di produrre una descrizione fisica del circuito hardware descritto dal codice VHDL
- Concetti e sintassi del VHDL sono molto influenzati dal linguaggio di programmazione del software Ada
- Oggi il VHDL è definito dallo standard IEEE 1164 (ultima versione: VHDL-4.0 del gennaio 2009)
- Esistono inoltre numerosi sotto-standard IEEE che definiscono aspetti di interesse per applicazioni specifiche

Struttura di un file VHDL

```
-- Questo e' un commento

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity UNCOMPONENTE is
    port ( ... );
end UNCOMPONENTE;

architecture NOMEARCH of UNCOMPONENTE is
...
begin
...
end NOMEARCH;
```


Struttura di un file VHDL (2)

- Il linguaggio VHDL è case-insensitive
  - Nessuna differenza tra MAIUSCOLE e minuscole
- Le librerie consentono di utilizzare
  - Tipi di dati predefiniti
  - Componenti hardware standard
  - Funzioni
- La dichiarazione `entity` descrive l'interfaccia di un componente hardware: quali segnali di input e di output lo collegano agli altri componenti
- La dichiarazione `architecture` descrive come è fatto il componente hardware
- Due modalità diverse per definire l'architettura:
  - Descrizione strutturale: si descrive il circuito definendo i collegamenti tra porte logiche ed i vari componenti
  - Descrizione comportamentale: si descrive il circuito definendo il comportamento che dovrà avere

Definizione dell'interfaccia

```
entity Prova is
    port(a, b, c:  IN std_logic;
         x:  OUT std_logic;
         y:  OUT std_logic;
    );
end Prova;    -- oppure end entity;
```

- Il nome del componente è `Prova` (o `PROVA`, `prova`, ...)
- Riceve tre ingressi chiamati `a`, `b` e `c` di tipo `std_logic` (segnale logico standard)
- Produce due uscite chiamate `x` e `y` di tipo `std_logic`

Tipi di dati predefiniti

- `integer`: numero intero con segno di (almeno) 32 bit
- `natural`: numero intero senza segno ≥ 0
- `positive`: numero intero senza segno > 0
- `real`: numero in virgola mobile con mantissa e esponente
- `time`: intervallo temporale
  - valore qualificato da: `fs ps ns us ms sec min hr`
- `character`: carattere codificato con lo standard ISO 8859 Latin 1 (8 bit)
- `boolean`: valori logici `false` e `true`

Tipi di dati predefiniti (2)

- `bit`: i valori `'0'` (valore logico basso) e `'1'` (alto)
- `std_logic`: un segnale logico standard:
  - `'U'`: non inizializzato
  - `'Z'`: stato alta impedenza
  - `'0'`: forzato logico basso
  - `'1'`: forzato logico alto
  - `'X'`: valore forzato sconosciuto
  - `'W'`: debole sconosciuto
  - `'L'`: debole logico basso
  - `'H'`: debole logico alto
  - `'-'`: non importa

Un valore "debole" è usato ad esempio per lo stato di una linea derivato da resistenze di "pull-up" o "pull-down"

Vettori

- È possibile definire vettori (`array`)
  - Ad esempio: `array (1 to 4) of boolean`
- Alcuni tipi di vettori sono predefiniti:
  - `string`: sequenza di `character`
  - `bit_vector`: sequenza di `bit`
  - `std_logic_vector`: vettore di valori logici

La direzione in cui si indicano gli indici dei vettori è importante:

`array (1 to 4) of integer` ⇒ big endian

`array (4 downto 1) of integer` ⇒ little endian

Descrizione strutturale

```
architecture Strut of Prova is
    signal tmp:  std_logic;
begin
    x <= a and b;
    y <= tmp xor b;
    tmp <= c or a;
end Strut;  -- oppure end architecture;
```

- Le "istruzioni" sono "eseguite" in parallelo, non in sequenza!
- L'ordine delle "istruzioni" non è importante
- `tmp` è un segnale interno dell'architettura (una linea)

Se `a`, `b`, `c` sono eguali a `'1'`, quanto vale `y`? `'0'`

Simulazione e sintesi

Il codice VHDL può essere utilizzato per

1. Simulare il funzionamento dell'hardware
2. Sintetizzare una descrizione fisica dell'hardware

Le fasi della "compilazione":

- Controllo della sintassi
- Sintesi della descrizione RTL del circuito
  - ⇒ lista di componenti e collegamenti chiamata *netlist*
- Simulazione del circuito (facoltativa)
- Implementazione del progetto
  - Traduzione della netlist in un design adatto al tipo di chip
  - Mappatura del design sulle risorse a disposizione nel chip
  - Piazzamento dei componenti e instradamento dei segnali
- Generazione del flusso di bit (*bitstream*) con cui programmare il chip

II parte - VHDL

In questa lezione si continua a descrivere il linguaggio di programmazione dell'hardware VHDL

1. La descrizione comportamentale
2. Il costrutto process
3. Sincronizzazione con segnale di clock
4. Variabili
5. Segnali
6. Collegamento di componenti

La descrizione comportamentale

- La descrizione strutturale si basa sulla semplice descrizione dei collegamenti tra porte logiche e componenti
  - I "puristi" distinguono anche tra descrizione strutturale e descrizione del flusso di dati
- La descrizione comportamentale consente di specificare l'architettura (implementazione) di un componente descrivendo come deve comportarsi
- La descrizione dell'interfaccia del componente è identica
  - basata sul costrutto `entity`
- È possibile utilizzare contemporaneamente diversi tipi di descrizione
  - una parte dell'architettura ha una descrizione comportamentale
  - un'altra parte ha una descrizione strutturale

La descrizione comportamentale (2)

- La descrizione comportamentale è particolarmente adatta quando il circuito è facilmente esprimibile come un algoritmo
- In particolare questa descrizione è conveniente implementando automi a stati finiti
- In generale è necessario preservare le informazioni interne del componente (ad esempio, lo stato dell'automa)
  - uso di flip-flop e registri
  - la logica prodotta è sequenziale, non combinatoria
- Per ottenere buone prestazioni con elevate frequenze di funzionamento si implementano circuiti sincroni
  - le macchine sincrone sono pilotate dal fronte attivo di un segnale di clock

Il costrutto process

La descrizione comportamentale è basata sul costrutto `process`

```
NOMEPROC : process( <lista segnali> ) is
    <elenco segnali e variabili>
begin
    <istruzioni in sequenza>
end process;    -- oppure end NOMEPROC;
```

- Le istruzioni all'interno del costrutto `process` sono eseguite in sequenza
  - analogo ad un linguaggio di programmazione del software
  - diversi costrutti `process` sono "eseguiti" in parallelo tra loro
- I circuiti corrispondenti ad un `process` sono attivati solo nelle circostanze definite dalla "`lista segnali`" dopo la parola chiave "`process`"
- La memoria "locale" del `process` è definita prima del "`begin`"

Sensitivity List

- È l'elenco dei segnali che segue il costrutto "process"
- Indica i segnali che attivano il circuito descritto
- In effetti il circuito è attivato per ogni cambiamento di stato di uno qualsiasi dei segnali nella lista

```
entity Prova is
   port (a :  IN std_logic;
         z :  OUT std_logic);
end entity;
architecture Behav of Prova is
begin
   process (a) is
   begin
      z <= not a;
   end process;
end architecture;
```


Sincronizzazione con il segnale di Clock

Per realizzare un componente attivato da un fronte di un segnale di clock:

- Elencare il segnale di clock come ingresso logico nell'interfaccia del componente
- Elencare il segnale di clock nella sensitivity list
- All'interno del costrutto "process", utilizzare il costrutto `if...then` per condizionare l'esecuzione
- Le condizioni possono essere meglio specificate con gli *attributi* dei segnali , ad esempio:
  - `clk'event`: vale `True` se il segnale `clk` cambia stato
  - `clk'last_value`: valore prima del cambiamento di stato
  - Esistono vari altri attributi, meno utilizzati

Sincronizzazione con il segnale di Clock (2)

```
entity Prova is
   port (a :  IN std_logic;
        clk :  IN std_logic;
          z :  OUT std_logic);
end entity;
architecture Behav of Prova is
begin
   process (clk) is
   begin
      -- fronte attivo: salita
      if clk'event and clk = '1' then
         z <= not a;
      end if;
   end process;
end architecture;
```

Variabili

- Le *variabili* consentono di memorizzare dati all'interno di un costrutto `process`
- Esempio (tra `process` e `begin`):

```
variable STATO : integer := -1;
```

- Se non indicato, il valore iniziale dipende dal tipo di dati
  - Per gli scalari, è il valore "all'estrema sinistra" dell'intervallo
- Generalmente le variabili sono "private"
  - Una variabile può essere visibile da un solo costrutto `process`
  - Tipicamente è dichiarata all'interno di un `process`
- Per assegnare un nuovo valore: `STATO := 42;`
- Le modifiche al valore di una variabile sono "immediate"
  - I flip-flop sono modificati mentre si "esegue" l'assegnazione

Variabili (2)

```
process (clk) is
   variable state :  integer := 0;
begin
   if clk'event and clk = '1' then
      case state is
         when 0 =>
           z <= (not a); state := 1;
         when 1 =>
           z <= a;       state := 2;
         when 2 =>
           z <= '0';     state := 0;
         when others =>
           z <= '0';     state := 0;
      end case;
   end if;
end process;
```


Segnali

- I *segnali* consentono di memorizzare dati all'interno di un costrutto `process`
- Esempio (tra `architecture` e `begin`):

```
signal STATO : integer := -1;
```

- Se il segnale appare soltanto in descrizioni strutturali, è considerato un semplice collegamento senza memoria
- Se invece il segnale appare entro un costrutto `process`, è un elemento di memoria simile ad una variabile, ma
  - I segnali possono essere condivisi tra più processi
  - La modifica del valore di un segnale all'interno di un costrutto `process` avviene solo al termine del costrutto
- Per assegnare un nuovo valore: `STATO <= 42;`

Segnali (2)

```
architecture Behav of Prova is
   signal state :  integer := 0;
begin
   process (clk) is
   begin
      if clk'event and clk = '1' then
         case state is
           when 0 =>
             z <= (not a); state <= 1;
           when 1 =>
             z <= a;       state <= 2;
           when 2 =>
             z <= '0';     state <= 0;
           when others =>
             z <= '0';     state <= 0;
      ⋮
```


Differenza tra variabili e segnali

```
architecture Behav of Quiz is
   signal S : integer := 0;
begin
   process is
      variable V : integer := 0;
   begin
      S <= 1;
      V := 1;
      x <= S;
      y <= V;
   end process;
end architecture;
```

Se all'inizio S e V sono 0, che valore hanno x e y al termine del processo? ***x = 0 y = 1***

Collegamento di componenti

- Nella descrizione strutturale si possono collegare tra loro componenti complessi
  - definiti dall'utente
  - contenuti in librerie
- Si deve ripetere la definizione dell'interfaccia del componente utilizzando il costrutto `component`
- Si deve poi istanziare ciascun componente indicando
  - Il nome dell'istanza
  - Il tipo di componente
  - Come instradare i segnali di ingresso e uscita

```
nome_comp :  tipo_comp port map (sig1 => sig2, ...);
```

Collegamento di componenti (2)

```
entity Esempio is
   port(a :  in STD_LOGIC;
        z :  out STD_LOGIC));
end entity;

architecture Struct of Esempio is
   component BlackBox is
      port(ing :  in STD_LOGIC;
           usc :  out STD_LOGIC));
   end component;
   signal tmp :  STD_LOGIC;
begin
   bb1 :  BlackBox port map (a, tmp);
   bb2 :  BlackBox port map (
             ing => tmp, usc => z);
end architecture;
```