Linguaggi di descrizione dell'hardware

Progetti a.a. 2022/23

Lista provvisoria

I progetti vengono assegnati dal docente cercando di soddisare le preferenze degli studenti. Si raccomanda di inserire come soggetto in qualsiasi mail riguardante i progetti la seguente stringa: `HDL - nome cognome` . Si consiglia inoltre di fare riferimento ai progetti con il titolo e non il numero.

Si ricorda che:

- la consegna dei progetti é indipendente dal superamento della prova scritta
- il materiale da consegnare consiste di: 1) relazione sul lavoro svolto; 2) codice sorgente (in VHDL o altri linguaggi); 3) eventuali esempi per il codice.

La relazione consiste tipicamente di un inquadramento dell'argomento nell'ambito del corso. Una descrizione della metodologia seguita per realizzare il progetto mettendo in rilievo quelli che si pensano essere i contributi piú interessanti del lavoro svolto. Una descrizione ad alto livello del codice (non importa mettere tutti i listati) e di eventuali istruzioni per l'utilizzo e la compilazione. Una descrizione dei risultati ottenuti (simulazioni, statistiche, uscite di algoritmi). Quest'ultimo aspetto di solito viene trascurato.

Informazioni di carattere generale che possono essere utili in diversi progetti.
Ritardi realistici dei gate

In questo corso non si indirizzano aspetti specifici a una data tecnologia, ma comunque puó valere la pena mettere valori di ritardo compatibili con qualche tecnologia recente, per cui si consiglia di mettere questi valori (se servono).

Chiaramente, i ritardi accurati vengono calcolati dopo la sintesi fisica.

NOT	$(30 + \text{fan-out} \cdot 15)\ ps$
NAND/NOR	$(40 + \text{fan-out} \cdot 20)\ ps$
AND/OR	$(50 + \text{fan-out} \cdot 20)\ ps$
XOR	$(50 + \text{fan-out} \cdot 30)\ ps$

Modello VHDL di componente che genera sequenze pseudocasuali di vettori logici

```
entity lfsr is
  generic(n: natural);
  port(reset,clk: in std_logic;
       q: out std-logic_vector(0 o n-1));
end entity lfsr;

architecture behav of lfsr is
begin
  process (clk)
  constant initial:="01011101....";
  variable tmp: std_logic_vector(0 to n-1);
  variable bk: std_logic;
  -- any value different from all 0s
  begin
    if (rising_edge(clk)) then
      if (reset='0') then
        bk:=tmp(1) xor tmp(n-1);
        for i in 1 to n-1 loop
          tmp(i):=tmp(i-1);
        end loop;
        tmp(0):=bk;
      else
        tmp:=initial;
      end if;
    end if;
    q <= tmp;
  end process;
end architecture;
```

Progetto 2

Modello behavioral dei ritardi di un adder

Data una descrizione strutturale di un n-bit adder (il circuito verrá fornito dal docente), si vuole costruire un modello comportamentale dei ritardi fra ingresso e uscita.

Le operazioni da svolgere sono le seguenti:

1. costruire un modello strutturale del componente (schema logico)
2. eseguire la STA su tale modello determinando EAT e LST per ciascuna uscita
3. costruire una prima versione di modello VHDL in cui il cambiamento delle uscite avviene rispettando EAT e LST
   (a) se un uscita cambia in conseguenza di un cambiamento degli ingressi, l'uscita si porta prima a 'X' con ritardo EAT e poi al suo valore finale con ritardo LST
4. il modello ha la caratteristica rilevante di essere conservativo su ritardo minimo e massimo
5. il componente va simulato in parallelo alla sua realizzazione a livello gate valutando cosí l'errore medio (in pratica si tratta di determinare gli EAT e gli LST medi delle uscite nel modello a livello gate e di confrontarli con quelli a livello comportamentale);
6. é poi interessante provare a connettere in cascata un paio di questi adder e valutare di nuovo l'errore.

Progetto 4

Guasti nella rete di distribuzione del clock

Si vogliono valutare gli effetti di possibili guasti della rete di distribuzione del clock.

Serve introdurre un po di terminologia, si chiama clock source la sorgente di tale segnale supposto ideale e clock sinks gli ingressi di tipo clock dei FF. Per diversi motivi (difetti e disturbi) il segnale di clock puó non arrivare in maniera corretta a uno piú FF.

Si noti che la rete di distribuzione di clock presenta inevitabilmente degli skew, che peró qui non verranno tenuti in conto perché facciamo l'ipotesi che siano giá stati tenuti in conto durante il progetto. Quindi supporremo di avere una rete che in assenza di disturbi ha un funzionamento ideale (il clock arriva in maniera sincrona a tutti i sink).

In questo progetto si vuole modellare in VHDL questo tipo di problemi nel caso in cui siano dovuti a disturbi (ad esempio crosstalk). In particolare, si fa l'ipotesi che il problema riguardi un singolo FF e che al suo sink si presenti uno dei seguenti problemi:

- il clock ritarda rispetto a quello ideale
- il clock anticipa rispetto a quello ideale
- il clock presenta un fronte di salita indesiderato oltre a quello normale (jitter)

tali difetti sono parametrici, ovvero il loro effetto dipende dal valore di ritardo/anticipo che presentano. L'effetto di questi guasti va modellato in VHDL e analizzato tramite la simulazione di descrizioni strutturali di FSM. I guasti possono essere modellati sia inserendo opportuni buffer nella rete di distribuzione del clock o modificando il modello VHDL dei FF.

In particolare, si dovrá prima costruire il modello di rete sincrona in assenza di guasti e poi quello in cui selettivamente si possono iniettare i diffetti considerati. I due modelli vanno simulati insieme confrontando le uscite per determinare quando i guasti producono un errore. I guasti stessi vanno simulati per diversi valori di ritardo/anticipo.

Progetto 6

Static timing analysis in VHDL

In questo progetto si vuole implementare l'algoritmo di static timing analysis sfruttando il meccanismo di simulazione event-driven del VHDL. L'idea quella di avere, per ciascun modello di gate, un architettura che, diversamente da quella funzionale, si limita a cambiare il valore dell'uscita in risposta a un cambiamento del valore di uno qualsiasi dei suoi ingressi. Cambiando contemporaneamente tutti gli ingressi, si propagano degli eventi attraverso tutto il circuito e la differenza fra l'istante di tempo in cui si ha l'ultima transizione delle uscite e quello in cui si sono cambiati agli ingressi é pari al ritardo massimo della rete cosí come viene calcolato da static timing analysis.

Si fornisce di seguito un esempio del codice di tale modello nel caso di un gate a due ingressi. Si richiede di realizzare anche un modello per i flip-flop in modo da poter considerare anche reti sincrone calcolando i ritardi massimi delle diverse parti combinatorie. Il modello del flip-flop deve commutare l'uscita solo in presenza del fronte di salita (discesa) del segnale di clock.

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

entity plb is
generic (delay: time);
port (a,b: in std_logic;
      output: out std_logic);
end entity plb;

architecture delay of plb is
begin

  process(a,b)
  variable x: std_logic:='0';
    begin

      if (a'event or b'event) then
        x:=not x;
        output <= transport x after delay;
      end if;
    end process;
end architecture;
```

Progetto 7

Simulazione di guasto

In questo progetto si deve realizzare un programma in un qualsiasi linguaggio (C, Java etc. etc.) che legge una descrizione dataflow di una una qualsiasi rete combinatoria, determina la lista dei segnali di tale rete (uscite e segnali interni) e realizza una versione modificata del componente e un testbench che siano in grado di eseguire una simulazione di guasti di tipo stuck-at sotto l'ipotesi di guasto singolo. Questa implementazione deve utilizzare le istruzioni messe a disposizione dallo standard VHDL 2008 (`force` , `release`).

In particolare, si tratta di:

- costruire un modello VHDL del circuito in cui si aggiunge un segnale intero di ingresso x e un segnale di tipo `std_logic` t. Il segnale x corrisponde all'indice del segnale che deve essere bloccato al valore specificato da$t$ (se $x = 0$ nessun segnale é guasto)

- il nuovo modello aggiunge alle istruzioni dataflow precedenti un processo con x e t nella sensitivity list che sulla base del valore di x utilizzando la `force` blocca a t un segnale fino a quando non cambiano tali parametri e allora con la `release` ne ripristina il funzionamento corretto per poi passare al guasto successivo

- il testbench istanzia la rete corretta e quella col guasto poi ha un processo che cambia il valore x (inizialmente a 0), si sospende attivando un ulteriore processo che genera un insieme di stimoli applicati alle due copie del circuito e si sospende fino a quando tale processo non termina

- serve anche una rete che confronti le uscite dei due circuiti determinando se il guasto viene rivelato o meno come un errore

- alla fine si dovrebbe avere la percentuale (*fault coverage*) di guasti rivelati dalla sequenza in esame

Il progetto é supportato questo materiale.

Progetto 8

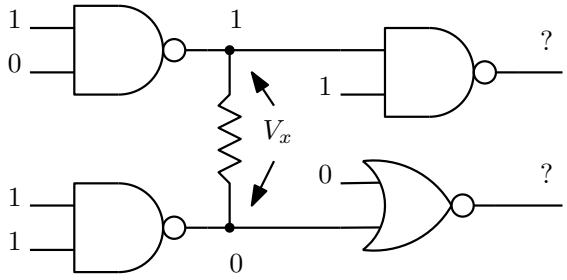
Simulazione di guasti di tipo bridging

In questo progetto si vuole simulare il comportamento di un circuito in presenza di guasti di tipo bridging che danno luogo a cortocircuiti non previsti fra uscgate. In presenza di questo tipo di guasti, se nel circuito privo di guasti le uscite dei gate cortocircuitate pilotano valori diversi, in quello guasto si ha un conflitto il cui esito dipende dalle conduttanze di uscita dei gate. Tipicamente, il segnale cortocircuitato si porta a un valore di tensione intermedio fra alimentazione e massa. I gate nel fan-out andranno a risolvere tale valore sulla base della loro soglia logica.

Per poter simulare questo tipo di comportamenti, bisogna per prima cosa caratterizzare i segnali di uscita dei gate aggiungendo informazioni sulla loro conduttanza e aggiungere una opportuna funzione di risoluzione del bus. Il VHDL in questo caso consente di utilizzare segnali di tipo record in cui alcuni campi possono portare informazioni sulla conduttanza di uscita. Una opportuna funzione di risoluzione del bus puó poi ottenere per il nodo cortocircuitato la conduttanza equivalente verso l'alimentazione e massa.

```
package bridgings is

type signal_u is record
  value: std_logic;
  p_up_g: real;
  p_down_g: real;
end record;

type signal_u_array is array (natural range <>) of signal_u;

function resolved (s: signal_u_array) return signal_u;

subtype signal_r is resolved signal_u;

type signal_r_array is array (natural range <>) of signal_r;

end package;

package body bridgings is

function resolved (s: signal_u_array) return signal_u is
variable x: signal_u;
```

```
begin
  x.p_up_g:=0.0;
  x.p_down_g:=0.0;
  for i in s'range loop
         x.p_up_g:=x.p_up_g+s(i).p_up_g;
         x.p_down_g:=x.p_down_g+s(i).p_down_g;
  end loop;
  return x;
end function;
-- inizializzare a 0 le conduttanze del segnale
```



---

1. descrivere il tipo di segnale e la funzione di risoluzione del bus in un package
2. descrivere delle porte logiche adatte per questo tipo di segnali, con particolare riferimento alla descrizione della soglia che deve risolvere i valori in ingresso ($[0, V_L] =' 0'$, $]V_L, V_H[=' X'$ e $[V_H, V_{DD}] =' 1'$).

Progetto 9

Algebra per hazard detection

In questo progetto si vuole costruire un algebra che sia in grado di rappresentare i possibili hazard presenti in una rete combinatoria quando viene applicata una sequenza di due configurazioni in ingresso. Il valore di un segnale é quindi denotato da un valore iniziale, uno finale e dalla possibile presenza di hazard.

value	initial	hazard	final
`s0`	0	no	0
`s1`	1	no	1
`s0h`	0	?	0
`s1h`	1	?	1
`r`	0	no	1
`f`	1	no	0
`rh`	0	?	1
`fh`	1	?	0
`x`	X		X
`x0`	X		0
`x1`	X		1
`0x`	0		X
`1x`	1		X

Ad esempio, se un AND ha `r` su un ingresso e `f` sull'altro, esiste la possibilitá che ci sia un hazard in uscita che assume quindi il valore `s0h` . Il fatto che poi ci sia nel circuito dipende dal timing attuale.

Si tratta di realizzare l'algebra prendendo esempio dal package `std_logic` o utilizzando i record, in questo caso si userebbero i valori delle colonne nella tabella. Poi si puó realizzare una libreria di gate e fare alcune prove su circuiti semplici come adder e ALU.

In alternativa, c'é una versione di questo progetto che utilizza i record ed é parzialmente supportata da questo materiale

Progetto 11

Interfaccia per l'inserimento di descrizioni strutturali in VHDL

In questo progetto (per due persone) si vuole costruire un sistema in grado di facilitare la costruzione di modelli VHDL strutturali. Si tratta di provare alcuni concetti senza sviluppare un sistema completo. Il progetto richiede l'utilizzo di finestre di testo, forms, bottoni .... ma non grafica vera e propria. Gli strumenti di programmazione possono essere i piú diversi, da Java a C con librerie tipo Qt, python, tcl/tk ....

In pratica, il sistema deve consentire l'inserimento di componenti e la possibilitá di richiamarli in fase di instanziazione, il mantenimento di una lista di segnali disponibili associabili alle porte dei componenti e la possibilitá di salvare i risultati in un formato intermedio. Devono essere messe poi a disposizione utilitá che svolgano funzioni piú o meno utili che non sono messe a disposizione dal VHDL.

Ad esempio, dato un segnale il sistema deve fornire la lista dei driver e il suo fan-out, oppure dato un ingresso il sistema deve fornire i cammini da quell'ingresso alle uscite. Ed eventuali altre funzioni ritenute utili dagli autori del progetto.

Esempio di possibile layout

Componenti	Segnali	Istanze
and2	x	a0: or2 port map(x,y,w);
or2	y	a1: and2 port map(x,y,w);
....	w	a2: and2 port map(x,w,z);
....	z	....
working area with buttons and windows		

In presenza di specifiche competenze, il progetto puó essere svolto anche utilizzando un interfaccia web (js, php ....).

Progetto 16
Simulazione Montecarlo di guasti transitori
Simulazione di guasti transitori

Guasti dovuti a radiazioni che generano impulsi di corrente nei dispositivi colpiti Un guasto transitorio puó alterare temporaneamente il valore di un uscita di gate o pu dare luogo a un cambiamento di stato di un flip-flop. Il loro effetto dipende da diversi fattori: 1) dimensioni dellimpulso, 2) istante in cui si ha levento, 3) cammino lungo cui si propagano gli effetti (elettrico e logico). Si tratta di applicare un numero sufficiente di vettori casuali in ingresso e di iniettare gli errori confrontando le uscite con quelle del circuito corretto (dopo il campionamento da parte di FF).
Gestione del non-determinismo utilizzabile in questo progetto e nel successivo

- Supporto per alcuni dei progetti precedenti
- Libreria per la generazione di numeri casuali
- Realizzazione di un gate con possibilitá di:
  - Ritardo variabile
  - Iniezione di errori transitori
- La stessa libreria si puó utilizzare per realizzare test bench con generazione pseudocasuale di vettori di collaudo
- Il package di base é tratto da un libro di C. Myers sulle reti asincrone

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

package nondeterminism is

  shared variable s1:integer:=844396720;
  shared variable s2:integer:=821616997;

  -- Returns a number between 1 and num.
  impure function selection(constant num:in integer) return integer;
  -- Returns a std_logic_vector of size bits between 1 and num.
  impure function selection(constant num:in integer;
                            constant size:in integer) return std_logic_vector;
```

```
  -- Returns random delay between lower and upper. (m.u. ps)
  impure function delay(constant l:in integer;
                        constant u:in integer) return time;

end nondeterminism;

package body nondeterminism is ..

library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;

entity inverter is
generic (index: integer; -- index of the element
              low_del,high_del: integer); -- low and high bounds on propagation delay
port(a: in std_logic;
      b: out std_logic;
      set: in integer:=0); -- signal provoking a pulse at gate output
end entity inverter;

architecture behav of inverter is
signal indx: integer:=index;
begin
  process(a,set)
  variable del: time:=0 ps;
  variable k: boolean := TRUE;
  variable set_time: time;

begin
  -- at the beginning of the process set a
  -- random quantity for propagation delay
  if (k) then
    del:=delay(low_del,high_del);
    k:= not k;
 end if;
  if (set/=index) then -- if the element is not seleted
  if (not(set'event)) then -- normal inverter behavior
  b <= not a after del;
  else
  b <= not a; -- ripristinate the correct output value
  end if;
  elsif (set=index) and (set'event) then -- single error transient
  b<=a;
```

```
  end if;

 end process;
end architecture behav;


library ieee;
use ieee.std_logic_1164.all;
use work.nondeterminism.all;

entity testbench is
end entity testbench;

architecture prg of testbench is

-- to be stored in a package
constant low_sens: time:= 1000 ps;
constant up_sens: time:= 1600 ps;
constant gates_no: integer:=2;
constant set_duration: time := 55 ps; -- duration of the SET

signal a,b,c: std_logic:='1';
signal set: integer:=0;

Begin

--stimuli generation
a<=not a after delay(380,400)

-netlist

not0: entity work.inverter(behav) generic map (1,50,60) port map (a,b,set);
not1: entity work.inverter(behav) generic map (2,50,60) port map (a,c,set);

process -- set a random time for the start of the SET
variable set_time: time;
begin
  set_time:=delay(low_sens,up_sens); -- set start
  wait for set_time;
  set <= selection(gates_no); -- select a random component
  wait for set_duration;    -- select pulse duration
```

```
  set <= 0;
  wait;
end process;

end architecture prg;
```

Progetto 17

Simulazione Montecarlo di errori timing

Simulazione di un circuito sincrono con diversi valori di ritardo dei gate secondo una distribuzione uniforme di probabilitá dei ritardi dei gate.

Analisi della dipendenza dell'errore dal periodo di clock e dalla configurazione di ingresso applicata. Bisogna applicare un insieme sufficentemente grande di vettori di ingresso e simulare con tali stimoli lo stesso circuito per diverse configurazioni dei ritardi. Problema: come determinare se l'uscita campionata é corretta?

Risultati per una singola configurazione dei ritardi: se si applicano N vettori con un periodo di clock T, si conta il numero di vettori E campionati in uscita al componente che risultano errati e quindi si ha una probabilitá di errore stimata E/N. Poi si puó rendere l'analisi piú accurata considerando il numero di errori per ciascuna uscita E_i e stimare la probabilitá di errore per tali uscite E_i/N.

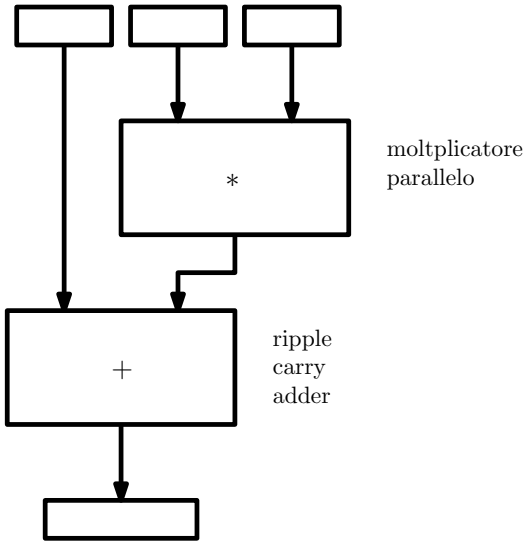
Poi si cambiano i ritardi e si ripete l'esperimento.

Scelta del perido di clock: se il periodo di clock soddisfa i vincoli sui ritardi non si hanno errori, quindi si potrebbe determinare tale valore e poi fare qualche simulazione per $0.95T$ e $0.9T$.

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

entity inverter is
generic (index: integer; -- index of the element
         low_del,high_del: integer); -- low and high bounds on propagation delay
port(a: in std_logic;
     b: out std_logic;
     set: in boolean); -- signal provoking a new evaluation of delays
end entity inverter;

architecture behav of inverter is
signal indx: integer:=index;
begin
  process(a,set)
  variable del: time:=0 ps;

begin
  -- at the beginning of a new simulation sets a
  -- random quantity for propagation delay
  if (set'event) then
    del:=delay(low_del,high_del);
  end if;
```

```
  b <= not a after del;

 end process;
end architecture behav;
```

Circuito da considerare (descritto al livello strutturale)



Progetto 27
Utilizzo dei tipi di dato fixed e floating point in VHDL

Le applicazioni di machine learning basate su reti neurali richiedono spesso l'uso di valori reali. Se realizzate direttamente in hardware tali tecniche richiedono di coniugare esigenze di accuratezza con obbiettivi di costo, prestazioni e consumo di potenza. Questi obbiettivi possono essere raggiunti utilizzando rappresentazioni dei numeri reali che non corrispondono a quelli standard utilizzati nelle CPU: é possibile utilizzare dati rappresentati in virgola fissa of in virgola mobile con un numero diverso di bit rispetto alle notazioni standard (IEEE-754).

Lo standard VHDL 2008 mette a disposizione tipi dato customizzabili sia in virgola fissa che in virgola mobile. Questi sono supportati sia da tool di simulazione che di sintesi.

Progetto 28
Utilizzo del PSL nella verifica di macchine a stati finiti

Il linguaggio Property Specification Language é stato inizialmente sviluppato per il System Verilog, a partire dallo standard 2008 é stato esteso al VHDL. Questo linguaggio serve per semplificare il compito di scrivere testbench.

L'analisi del funzionamento di un sistema basandosi solo sulle forme d'onda é un operazione quasi impossibile e non sempre é possibile confrontare la rete corrente con un modello corretto.

Il PSL basandosi anche sulla logica temporale mette a disposizione meccanismi potenti per verificare le proprietá di un sistema digitale consentendo di specificare proprietá tipo "dopo che un segnale si é portato a 1, un altro segnale si deve portare a 1 entro 4 peridi di clock".

In questo progetto verrá applicato alle macchine a stati finiti per determinare la presenza di possibili errori.

Progetto 29
Applicazione di un generatore di parser (ANTLR) al linguaggio VHDL.

Il sistema ANTLR data una grammatica consente di ottenere un parser per il relativo linguaggio in maniera molto semplice. Tale parser puó poi essere utilizzato per riempire strutture dati opportune, che nel caso del VHDL potrebbero descrivere la connettivitá o espressioni logiche ad esempio.

Il sistema puó generare parser per VHDL in java o python ed é giá stato usato con successo in progetti precedenti.

Il progetto consiste in una semplice applicazione in cui si utilizza il parser per analizzare una descrizione strutturale e produrre informazioni su tale struttura: fan-out, path. La grammatica del VHDL é giá disponibile.

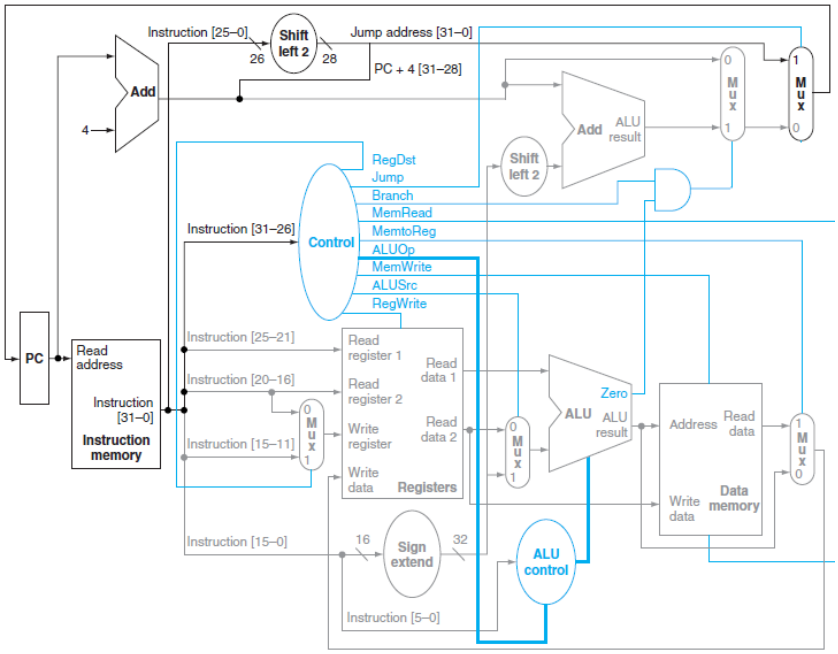
Progetto 32

Realizzazione di una CPU a ciclo singolo

In questo progetto di deve realizzare la versione semplificata della CPU di tipo MIPS a ciclo singolo indicata in figura.

In particolare, i componenti combinatori possono essere realizzati al livello comportamentale e lo stesso vale per le due memorie e il register file (notate che la memoria istruzioni puó essere semplificata come una ROM perché non viene mai scritta). Il controllo di tale CPU é combinatorio e quindi non ci sono da realizzare FSM. Il test bench deve consistere nell'esecuzione di un piccolo frammento di codice assembly (da fornire in binario, ovvero non é necessario fare un interfaccia fra assembly e VHDL).

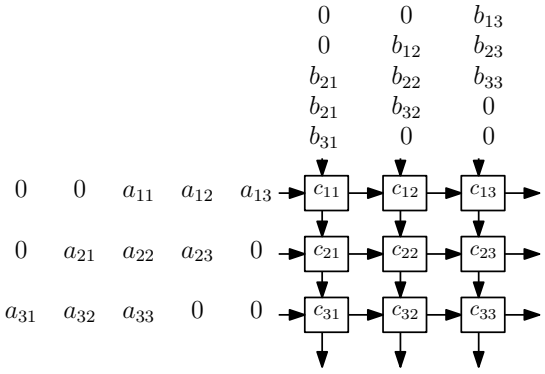
Verrá fornita ulteriore documentazione.



Progetto 33

Array sistolici

Gli array sistolici sono architetture regolari utilizzate per realizzare algoritmi specifici come, ad esempio, la moltiplicazione fra matrici. Nell'esempio, si ha una moltiplicazione fra due matrici A e B di dimensione $3x3$. Il cui risultato é una matrice $3x3$ i cui elementi sono $c_{ij} = \sum_{k=1}^{3} a_{ik} b_{kj}$.



I dati in ingresso vengono letti da una RAM e inseriti in FIFO (shift-register) prima dell'elaborazione. I processing element (i quadrati) moltiplicano i valori che hanno in input e li sommano a un valore accumulato in precedenza. Poi passano i valori di B ai processing element di sotto e quelli di A a quelli a destra (a parte gli ultimi). Alla fine, gli elementi contengono la matrice prodotto che comunque deve poi essere scritta in RAM.

Nel progetto, i processing element (che hanno un moltiplicatore, un adder, piú registri di uscita e multiplexer) possono essere descritti al livello strutturale utilizzando blocchi descritti al livello comportamentale.