Linguaggi di descrizione dell'hardware

Progetti a.a. 2021/22

Lista provvisoria

I progetti vengono assegnati dal docente sulla base delle preferenze degli studenti. Si raccomanda di inserire come soggetto in qualsiasi mail riguardante i progetti la seguente stringa: `HDL - nome cognome`

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 3

Gestione del pessimismo nella simulazione di valori a X

In questo progetto si vuole realizzare un testbench parametrico in grado di verificare la funzionalitá di piccoli moduli combinatori (fino a 6-8 ingressi e un uscita) applicando tutti i possibili valori di `std_logic` o di un suo sottoinsieme quale `X01`. Il test bench deve anche riconoscere la possibile presenza di fenomeni di memoria.

In particolare, si vogliono considerare:

- `std_logic=(U,X,0,1,Z,W,L,H,-)`
- `X01` `std_logic=(X,0,1)`
- `X01Z` `std_logic=(X,0,1,Z)`

La generazione delle configurazioni in ingresso puó avvenire con un operazione di conteggio in modulo. L'attributo `'transaction` puó essere utilizzato per osservare i fenomeni di memoria. In caso il progetto venga selezionato da un gruppo da 2 si agginge la generazione di configurazioni pseudorandom.

Esempio nel caso di componente con 4 ingressi e segnali di tipo `X01` . Configurazioni generate (se interpretate X=2 diventa un conteggio modulo 3):

```
0000 01X1 101X 1X10 X101 XXXX
0001 01XX 10X0 1X11 X10X
000X 0X00 10X1 1X1X X110
0010 0X01 10XX 1XX0 X111
0011 0X0X 1100 1XX1 X11X
001X 0X10 1101 1XXX X1X0
00X0 0X11 110X X000 X1X1
00X1 0X1X 1110 X001 X1XX
00XX 0XX0 1111 X00X XX00
0100 0XX1 111X X010 XX01
0101 0XXX 11X0 X011 XX0X
010X 1000 11X1 X01X XX10
0110 1001 11XX X0X0 XX11
0111 100X 1X00 X0X1 XX1X
011X 1010 1X01 X0XX XXX0
01X0 1011 1X0X X100 XXX1
```

---

1. sviluppo del modulo parametrico per la generazione di sequenze

2. sviluppo del modulo che verifica le uscite
3. utilizzo di tali moduli per verificare gate combinatori forniti dal docente

Progetto 5

Detection di timing violation

Come é noto, nel caso di violazioni del timing, un errore puó essere prodotto e venire campionato dagli elementi di memoria. Per rivelare questi errori, occorre simulare il circuito e confrontare i valori campionati con quelli corretti. Si tratta quindi di costruire un miter per il circuito in esame. In questo progetto, si propone un alternativa pessimistica per tracciare questi errori. Si parte dall'ipotesi che il circuito sia sincrono, a ciclo singolo e che campioni agli istanti kT $k = 0....n$. In questo caso, si puó avere una timing violation se e solo se esiste almeno un gate nel circuito che viene valutato nell'intervallo $[(k-1)T, kT[$ e produce la sua risposta a un tempo $t > kT$.

Quindi se entro nel processo che calcola l'uscita del gate al tempo t (notate che si puó usare la funzione `now`) e l'uscita viene aggiornata a $t + del$, allora si ha un potenziale errore se $t \ mod \ T \neq (t + del) \ mod \ T$.

Si noti che in questo caso ci si accorge di errori senza bisogno di propagarne gli effetti alle uscite del circuito.

Nel progetto si richiede quindi di creare una libreria di gate logici standard con anche il periodo di clock passato come generic. Ogni gate deve essere in grado di valutare la condizione precedente producendo un indicazione di possibile erorre con una assert. Questa libreria va provata per diversi tipi di rete (ripple adder, CLA adder).

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

Progetto 8

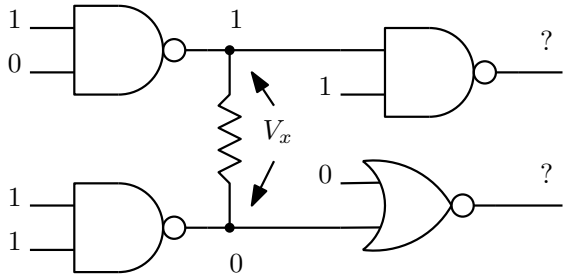
Simulazione di guasti di tipo bridging

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.

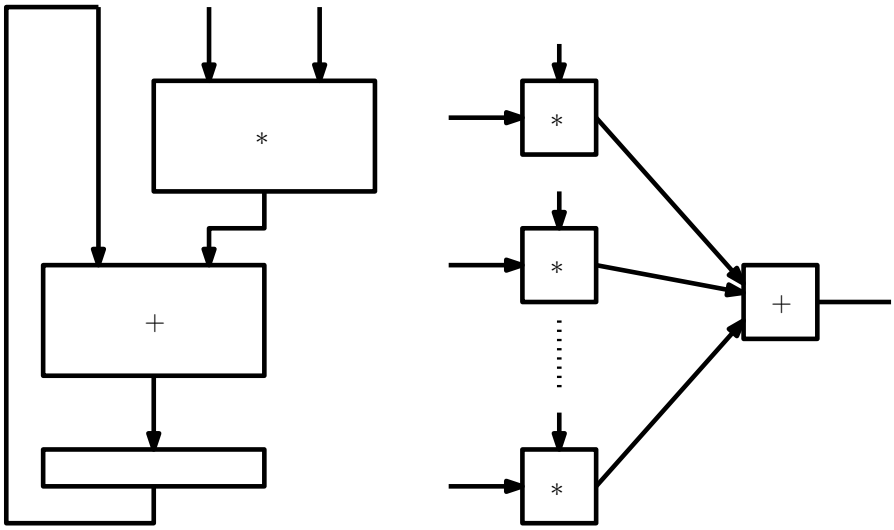
Si tratta di realizzare l'algebra prendendo esempio dal package `std_logic` o utilizzando i record, in questo caso si userebbero i valori delle colonne nella tabella. Poi si puó realizzare una libreria di gate e fare alcune prove su circuiti semplici come adder e ALU.

Esiste anche una versione per 2 persone che prevede di applicare tale algebra a semplici reti asincrone.

Progetto 10

Confronto fra MAC e architettura combinatoria

Una parte importante delle reti neurali esegue il calcolo di una somma pesata $\sum_i x_i w_i$, questa puó essere fatta tramite una rete Multiply and Accumulate (MAC, a sinistra) o tramite un implementazione combinatoria (a destra). Dal punto di vista funzionale le due architetture sono completamente equivalenti.



Non é cosí dal punto di vista di latenza, banda e consumo di potenza. Qui il consumo di potenza viene stimato banalmente come il numero complessivo di transizioni in salita delle porte logiche.

In questo progetto si tratta di valutare quantitativamente queste differenze fra i due errori sul risultato finale. I passi da realizzare sono i seguenti (per il caso di 4 termini da sommare):

- realizzazione strutturale delle due architetture (schematici dei componenti forniti dal docente)
- la realizzazione strutturale va fatta a partire da porte logiche che consentano di contare il numero di transizioni (ci sono diverse possibilitá da discutere con il docente)
- valutazione del ritardo massimo combinatorio tramite STA e dimensionamento del periodo di clock supponendo che gli ingressi siano forniti da dei registri e l'uscita sia campionata da un registro
- realizzazione di testbench per le due architetture e valutazione del consumo di potenza per una sequenza di vettori di ingresso.

Progetto 13

In questo progetto si chiede di modificare quanto visto sulla EFSM che calcola il massimo comune divisore fra due interi nel seguente modo:

- l'algoritmo, al segnale di `start`, inizia a calcolare i massimi comuni divisori per n coppie di valori di ingresso che si trovano memorizzate in una RAM
- ogni volta in cui l'algoritmo produce un risultato, questo viene scritto nella RAM (chiaramente in posizioni diverse dai dati di ingresso)
- dopo aver calcolato l'ultimo risultato ed averlo inserito nella RAM, viene fornito un segnale di `data_ready`

Il risultato é una EFSM (che chiaramente conterrá come sottografo quello della EFSM vista in laboratorio).

Si fornisce una possibile descrizione behavioral della RAM.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.Numeric_Std.all;
entity sync_ram is
  port (
    clock   : in  std_logic;
    we      : in  std_logic;
    address : in  std_logic_vector;
    datain  : in  std_logic_vector;
    dataout : out std_logic_vector
  );
end entity sync_ram;
architecture RTL of sync_ram is
   type ram_type is array (0 to (2**address'length)-1) of
   std_logic_vector(datain'range);
   signal ram : ram_type;
   signal read_address : std_logic_vector(address'range);
begin
  RamProc: process(clock) is
  begin
    if rising_edge(clock) then
      if we = '1' then
        ram(to_integer(unsigned(address))) <= datain;
      end if;
      read_address <= address;
```

```
    end if;
  end process RamProc;
  dataout <= ram(to_integer(unsigned(read_address)));
end architecture RTL;
```

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 possibilit 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 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.

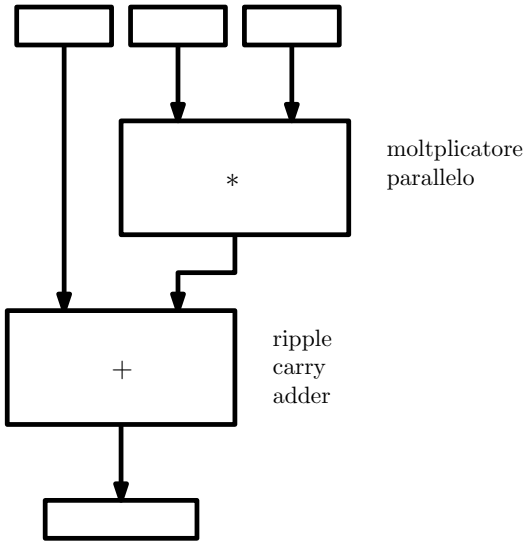
Poi si cambiano i ritardi e si ripete l'esperimento.

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 18

Realizzazione di un simulatore logico event-driven

In questo progetto, si vuole realizzare un simulatore logico di tipo event-driven utilizzando un qualsiasi linguaggio ad alto livello (C, Java). Il simulatore legge una descrizione di una rete al livello gate con ritardi e un insieme di configurazioni di ingresso per tale rete e produce una previsione sul comportamento dei segnali della rete in risposta a tali stimoli.

Il formato della descrizione potrebbe essere una restrizione del dataflow del VHDL:

```
p <= a nand b after 2 ns;
q <= c and p after 1 ns;
....
```

Il programma legge tale descrizione e memorizza le informazioni in una tabella che ha un record per ogni nodo contenente le informazioni sul tipo di funzione, gli operandi e il ritardo. C'é poi una struttura dati che si utilizza per la simulazione che é essenzialmente una lista gerarchica con un elemento per ogni istante in cui la rete é attiva e tale elemento punta a una lista contenente i segnali da aggiornare in tale istante. L'algoritmo di simulazione inizializza tale struttura dati con gli eventi sugli ingressi e poi itera cambiando di valore ai segnali nella lista corrente e poi passa a valutare i gate nel fan-out di tali segnali programmando in istanti futuri gli eventi in uscita a tali gate. Questa metodologia é descritta anche sul testo.

Progetto 19

STA su descrizioni di reti in VHDL

In questo progetto, si vuole realizzare un algoritmo che esegua la static timing analysis utilizzando un qualsiasi linguaggio ad alto livello (C, Java). Il simulatore legge una descrizione di una rete combinatoria al livello gate con ritardi.

Il formato della descrizione potrebbe essere una restrizione del dataflow del VHDL:

```
library ieee; -- ignored
use ieee.std_logic_1164.all; -- ignored

p <= a nand b after 2 ns;

q <= c and p after 1 ns;

....
```

Il programma legge tale descrizione e memorizza le informazioni in una tabella che ha un record per ogni nodo contenente le informazioni sul tipo di funzione, gli operandi e il ritardo. Utilizzando questa tabella si puó poi realizzare l'algoritmo di STA e produrre anche la lista di tutti i possibili cammini ingresso-uscita con i relativi ritardi.

Progetto 23

Approximate computing

L'approximate computing consiste nel consentire all'hardware di effettuare calcoli di tipo aritmetico con una precisione variabile. Dipendentemente dall'applicazione e dalle condizioni al contorno (energia disponibile), la rete puó decidere di approssimare in vari modi i bit meno significativi del risultato, consentendo di mantenere non attive le porte logiche che in condizioni normali dovrebbero calcolarle. Questo chiaramente consente di risparmiare potenza dinamica.

In questo progetto c' da implementare un adder in grado di calcolare una somma in maniera approssimata sulla base di un segnale di controllo. L'idea é quella di utilizzare solo un numero limitato di bit degli operandi in ingresso. L'adder é descritto in un articolo piuttosto semplice.

Progetto 24

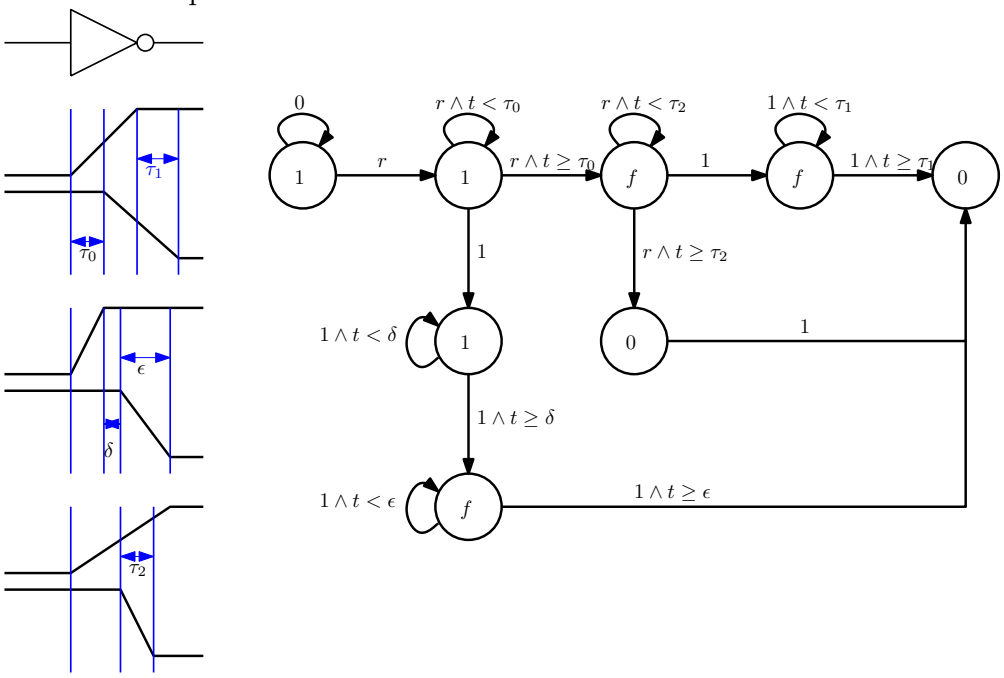
Caratterizzazione di un invertitore come timed automata

Il modello di ritardo di propagazione non tiene conto della pendenza dei segnali di ingresso e uscita. Questo puó essere verificato con simulazioni al livello elettrico.

Questi aspetti possono essere modellati in VHDL utilizzando un estensione delle FSM detta timed automata che nel caso piú semplice rappresenta un sistema in cui le transizioni avvengono a causa di cambiamenti degli ingressi o del passare del tempo.

In pratica, quando si entra in uno stato viene messa a 0 una variabile t che poi avanza col tempo di simulazione fino a quando una condizione su una transizione non diventa vera.

Un esempio di parte di un timed automata in grado di gestire tre diverse possibili condizioni sulle pendenze di ingressi e uscite di un invertitore é illustrato nella figura seguente. Tale automa va completato utilizzando simulazioni con SPICE per determinare i parametri.



Progetto 25

Euristico di tipo force-directed scheduling

Si chiede di implementare il force directed scheduling in una versione piú accurata di quella vista a lezione (tenendo conto delle predecessor-successor forces).

In particolare, il programma riceve in ingresso un file di testo con la descrizione dell'algoritmo che deve includere somme, sottrazioni, moltiplicazioni e divisioni. I costi dei componenti possono essere dati come parametri da linea di comando o essere inseriti in un file di configurazione. Il programma deve poi calcolare scheduling ASAP e ALAP e da questi applicare l'euristico fornendo una traccia delle operazionis volte e il risultato finale, ovvero uno scheduling che minimizzi l'area per una latenza uguale a quella minima.

Progetto 26

Euristico di tipo list-based scheduling

Si chiede di implementare il list based scheduling per il problema della minimizzazione della latenza per un costo prefissato.

In particolare, il programma riceve in ingresso un file di testo con la descrizione dell'algoritmo che deve includere somme, sottrazioni, moltiplicazioni e divisioni. I costi dei componenti devono essere inseriti in un file di configurazione contenente anche l'allocazione. Il programma deve poi applicare l'euristico fornendo una traccia delle operazioni volte e il risultato finale. Come misura di prioritá si consiglia di utilizzare il critical-path relativo.

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 30
Algoritmo approssimato per il calcolo della radice quadrata (interi)

Si fornisce la descrizione comportamentale di un algoritmo che calcolo in maniera approssimata la radice quadrata (intera) di un numero intero positivo.

Si tratta di verificare se questa descrizione comportamentale funziona e di ottenere da questa una descrizione del tipo EFSM in maniera simile a quanto visto per il massimo comune divisore (quindi con segnali di start e data ready). Dopo aver realizzato questa si deve realizzare una descrizione strutturale supponendo di disporre di una ALU (semplicissima), uno shifter programmabile e un comparatore oltre a registri ed eventuali multiplexer. Tali componenti possono essere descritti al livello behavioral. Tale data-path deve poi essere controllato da una FSM estratta dalla EFSM.

```
   n: in std_logic_vector(7 downto 0);
   sqrt: out std_logic_vector(7 downto 0);

....
 process(n)
 variable x,c,d,u: unsigned(7 downto 0);

 constant zero: unsigned(7 downto 0):=(others=>'0');

 begin

  x:=unsigned(n);
  c:=zero;

  d:='1' & (others=>'0');
  while (d > n) loop
   d:="00" & d(n-1 downto 2);
  end loop;

  -- now d contains the higher power of 4 smaller than n

  while (d /= zero) loop
      u:=c+d;
      c:='0' & c(n-1 downto 1);
      if (x>=u) then
        x:=x-u;
```

```
        c:=c+d;
      end if;
      d:="00" & d(n-1 downto 2);
    end loop;
  sqrt<=std_logic_vector(c);

end process;
```

Progetto 31

Ottimizzazione dei registri

Dato un data-flow graph con scheduling, si chiede di implementare l'ottimizzazione dei registri tramite graph-coloring.

In particolare, il programma riceve in ingresso un file di testo con la descrizione dell'algoritmo in cui ciascuna operazione é annotata con il ciclo di clock in cui viene eseguita e la risorsa funzionale che la esegue (binding). A questo punto per ogni variabile utilizzata (a sinistra degli assegnamenti), si calcola il tempo di vita e da questo grafo si ottiene il numero di colori (registri) da utilizzare per il graph coloring e il grafo di incompatibilitá delle variabili.

Per colorare quest'ultimo esistono delle tecniche esatte, ma qui si suggerisce di utilizzare un euristico di tipo greedy. Si tratta semplicemente di definire un ordinamento per i vertici e uno per i colori. Si parte dal primo vertice e si procede assegnando a ciascun vertice un colore fra quelli diversi da quelli dei vertici adiacenti a quello considerato.

Dopo aver fatto questa operazione, si chiede di scrivere la tabella con la descrizione RTL delle operazioni.