Compito di linguaggi di descrizione dell'hardware

Esercizio 1 Si descriva al livello comportamentale del VHDL un flip-flop di tipo D positive edge-triggered con comandi di write enable WE e reset RES sincroni. Il flip-flop é caratterizzato anche da un tempo di risposta tr e da un tempo di setup tsu (ogni violazione del tempo di setup risulta nell'uscita Q al valore `'X'`). Oltre a rilevare le violazioni di tempo di setup dovute a D, il modello deve fare la stessa operazione anche per WE e RES (pt. 5.0).

Soluzione

Questo esercizio puó essere affrontato in vari modi che dipendono essenzialmente dalla prioritá che hanno i diversi comandi fra loro (ad esempio quando RES va a 1, non é rilevante conoscere il comporatmento di D e WE). Una soluzione conservativa che produce forse piú valori a `'X'` rispetto a molte possibili implementazioni é quella indicata di seguito.

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

entity dff is
  generic(tsu,tr: time);
  port(d,we,res: in std_logic;
       clk: in std_logic;
       q: out std_logic);
end entity dff;

architecture behav of dff is
begin

  process(clk)
  begin
    if (clk='1') and (clk'last_value='0') then
      if (res'stable(tsu) and we'stable(tsu) and d'stable(tsu)) then
        if (res='1') then
          q <= '0' after tr;
        elsif (res='0') and (we='1') then
           q <= d after tr;
        end if;
      else
        q <= 'X' after tr;
      end if;
    end if;
  end process;
```

```
end architecture;
```

Nota: gli esercizi 2.1 e 2.2 sono in alternativa per chi ha frequentato nell'a.a. 2018/19. L'esercizio 2.1 é sconsigliato a chi abbia frequentato negli anni precedenti.

Esercizio 2.1 Si descriva in VHDL comportamentale un componente combinatorio che riceve in ingresso due parole x e y di 8 bit (`std_logic_vector(7 downto 0)`). Tali vettori contengono due numeri reali rappresentati in virgola fissa in maniera corrispondente ai seguenti formati: `ufixed (4 downto -3)` per x e `ufixed (3 downto -4)` per y. Il componente deve produrre sull'uscita *sum* (sempre di tipo `std_logic_vector`) il risultato della somma senza che ci siano problemi di overflow o perdite di precisione (pt. 5.0).

Soluzione

```
library ieee;
use ieee.std_logic_1164.all, ieee.fixed.all;

entity adder is
  port(x,y: in std_logic_vector(7 downto 0);
       sum: out std_logic _vector(9 downto 0));
end entity adder;

architecture behav of adder is
begin

  process(a,b)
  variable tmpx,tmpy: std_logic_vector(9 downto 0);
  variable ux,uy: ufixed(5 downto -4);
  begin
    tmpx:='0' & x & '0';
    tmpy:="00" & y;       -- now they are aligned
    ux:=to_ufixed(x,5,-4);
    uy:=to_ufixed(y,5,-4);
    sum <= to_slv(ux+uy);
  end process;
end architecture;
```

Esercizio 2.2 Si descriva in VHDL comportamentale un componente combinatorio che riceve in ingresso una parola a da 16 bit e una b da 3 bit. Entrambe rappresentano interi senza segno. La rete deve produrre in uscita il prodotto dei due numeri rappresentati da a e b. L'uscita deve essere di tipo `std_logic_vector` come i due ingressi e va dimensionata in maniera tale da evitare perdite di informazioni. L'esercizio deve

essere risolto senza l'utilizzo esplicito di operazioni di tipo prodotto nel codice VHDL (pt. 5.0).

Soluzione

```
library ieee;
use ieee.std_logic_1164.all, numeric_std.all;

entity multiplier is
  port(a: in std_logic_vector(15 downto 0);
       b: in std_logic_vector(2 downto 0);
       y: out std_logic _vector(18 downto 0));
end entity multipleir;

architecture behav of multiplier is
begin

  process(a,b)
  variable p: unsigned(18 dwonto 0);
  begin
    p:=(others => '0');
    if (b(0)='1') then
      p:=p+unsigned("000" & a);
    end if;
    if (b(1)='1') then
      p:=p+unsigned("00" & a & "0");
    end if;
    if (b(2)='1') then
      p:=p+unsigned("0" & a & "00");
    end if;
    y <= std_logic_vector(p);
  end process;
end architecture;
```

Esercizio 3

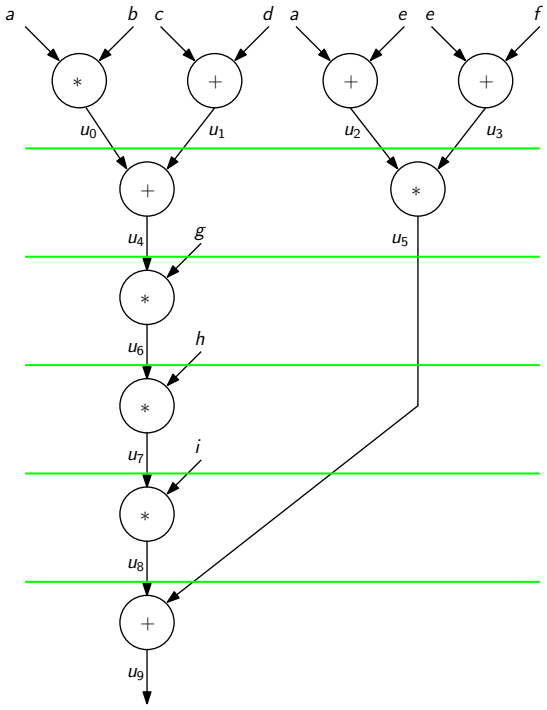
Si consideri il seguente algoritmo:

```
u0:=a*b;      u5:=u2*u3;
u1:=c+d;      u6:=u4*g;
u2:=a+e;      u7:=u6*h;
u3:=e+f;      u8:=u7*i;
u4:=u0+u1;    u9:=u8+u5;
```

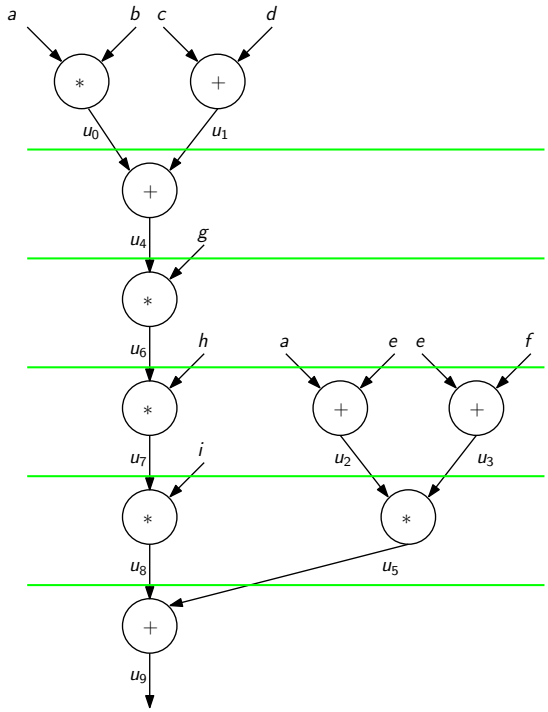
si tracci il DFG. Si tracci poi il DFG con scheduling ASAP e ALAP con l'ipotesi di ciclo singolo. Si determini, utilizzando tale ipotesi, uno scheduling a latenza minima che ottimizzi il numero complessivo di risorse utilizzate. Si determini poi uno scheduling, sempre a latenza minima, che ottimizzi il costo della rete nell'ipotesi che l'area dei moltiplicatori sia 10 volte quella dei sommatori. Si commentino sinteticamente i due risultati ottenuti (pt. 5.0).

Soluzione

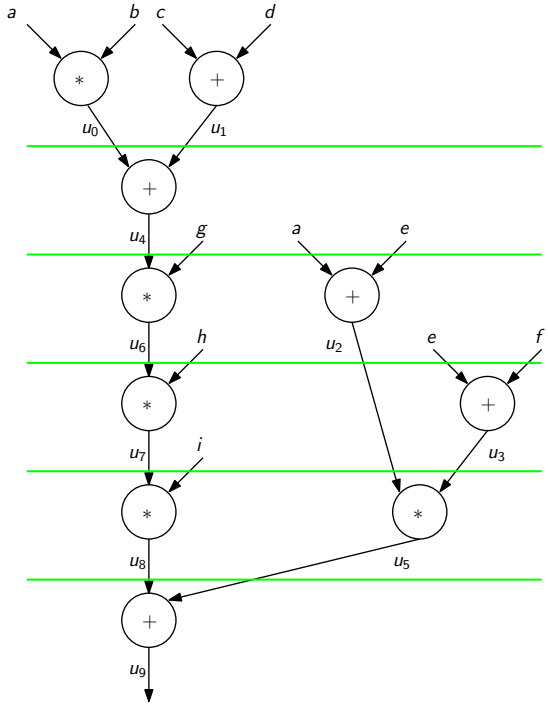
Scheduling ASAP, allocazione 3 adder e 1 moltiplicatore.



Scheduling ALAP, 2 adder e 2 moltiplicatori



Scheduling che minimizza il numero di risorse (che vengono tutte considerate con peso 1)



Si puó verificare che invece lo scheduling ASAP da luogo al minor costo ($1 \times 10 + 3 \times 1$) quando si tenga conto dell'area. Questo é un esempio in cui il problema non puó essere semplificato limitandosi a minimizzare il numero di risorse