Compito di linguaggi di descrizione dell'hardware

Esercizio 1

Si realizzi il modello comportamentale in VHDL di un componente che riceve in ingresso due segnali x, y e un segnale di *reset*. Il componente verifica che il segnale y sia la copia di x ritardata di t. Se questo é verificato l'uscita *err* rimane a 0, altrimenti si porta a 1 e mantiene tale valore fino a quando $reset = 0$. Quando *reset* si porta a 1, l'uscita torna eventualmente a 0. **Soluzione**

Il parametro `tol` é una tolleranza (la forma d'onda ritardata di x e quella di y possono differire per `tol`), non era richiesto dall'esercizio, ma ha fisicamente senso.

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

entity check is
generic (t,tol: time);
port (x,y: in std_logic;
      reset: in std_logic;
      err: out std_logic);
end entity;

architecture behav of check is
signal xdel, cmp: std_logic;
begin
  xdel<=transport x after t;
  cmp<=xdel xor y after tol; -- tol << t

  process(cmp,reset)
  begin
    if (reset='0') and (rising_edge(cmp)) then
      err<='1';
    elsif (reset='1') then
         err<='0';
       elsif (is_x(reset)) then
            err<='X';
    end if;
  end process;
end architecture;
```

Esercizio 2

Si realizzi la descrizione comportamentale nel linguaggio VHDL di una rete combinatoria che riceve in ingresso due parole $a_{7..0}$ e $b_{1..0}$ che rappresentano due interi senza segno A e B. Compito della rete é produrre in uscita $A * B$. L'operazione deve rispettare i seguenti vincoli: 1) non deve essere usato il prodotto (evitando di

instanziare moltiplicatori nella fase di sintesi); 2) il dimensionamento dell'uscita y deve essere scelto in modo da rappresentare correttamente il risultato.

Soluzione

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

entity mult is
port (a: in std_logic_vector(7 downto 0);
      b: in std_logic_vector(1 downto 0);
      p: out std_logic_vector(9 downto 0));
end entity;

architecture behav of mult is
begin

  process(a,b)
  variable a1,a2:unsigned(9 downto 0);
  begin
    case b is
    when "00" => p<=(others=>'0');
    when "01" => p<="00" & a;
    when "10" => p<='0' & a & '0';
    when "11" => a2:=unsigned('0' & a & '0');
                 a1:=unsigned("00" & a);
                 p<=std_logic_vector(a2+a1);
    when others => p<=(others=>'X');
    end case;
  end process;
end architecture;
```

Esercizio 3

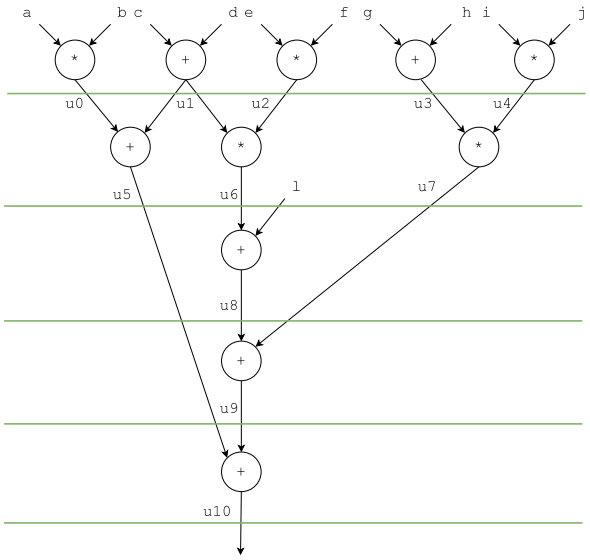
Si consideri il seguente algoritmo:

```
u0:=a*b;
u1:=c+d;
u2:=e*f;
u3:=g+h;
u4:=i*j;
u5:=u1+u0;
u6:=u2*u1;
u7:=u3*u4;
u8:=u6+l;
u9:=u8+u7;
u10:=u5+u9;
```

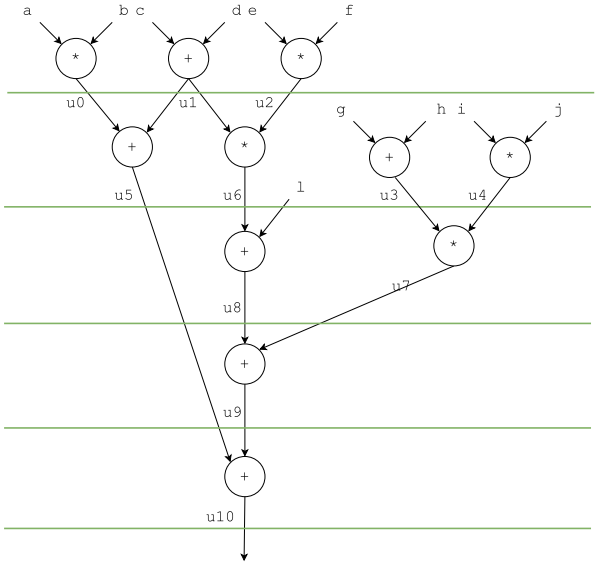
si tracci il DFG e si determinino lo scheduling ASAP e quello ALAP nell'ipotesi di ciclo singolo. Si determini poi uno scheduling che fissate le risorse (massime) a 2 adder e 1 moltiplicatore, minimizzi la latenza. Si mostrino poi i passi che svolgerebbe l'agoritmo di list based scheduling per risolvere lo stesso problema (indicando quale dei due metodi possibili é stato utilizzato per calcolare la prioritá).

Soluzione

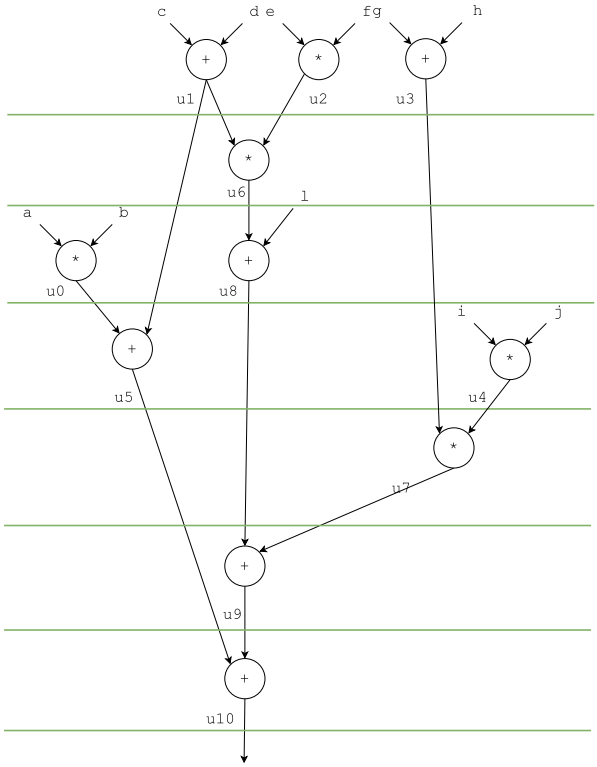
Scheduling ASAP: latenza relativa: 5, allocazione: 2 moltiplicatori e 3 adder.



Scheduling ALAP: latenza relativa 5, allocazione: 2 moltiplicatori e 2 adder.



Soluzione che ottimizza la latenza con risorse pari a 1 moltiplicatore e 2 adder.



List based scheduling con critical path come indice di prioritá. Nella ogni operazione é associata all'indice del risultato e la prioritá é data dal massimo numero di nodi nel DFG fra tale nodo e l'uscita:

nodo	prioritá
0 (*)	2
1 (+)	4
2 (*)	4
3 (+)	3
4 (*)	3
5 (+)	1
6 (*)	3
7 (*)	2
8 (+)	2
9 (+)	1
10 (+)	0

ciclo	nodi pronti	scheduling
1	0,1,2,3,4	1(+),2(*),3(+)
2	0,4,6	6(*)
3	0,4,8	4(*),8(+)
4	0,7	0(*)
5	5,7	5(+),7(*)
6	9	9(+)
7	10	10(+)

In questo caso non ci sono vantaggi rispetto alla soluzione illustrata in precedenza.