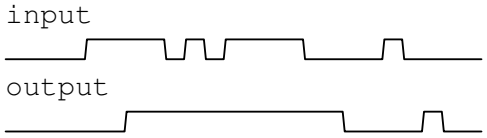
Compito di linguaggi di descrizione dell'hardware

Esercizio 1

Si realizzi un modello comportamentale in VHDL di un buffer che riporta in uscita il valore dell'ingresso ritardato di t (da fornire come generic). Il componente deve filtrare gli impulsi a 0 piú piccoli di t, ma non quelli a 1.




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

entity buff is
   generic(t: time);
   port(a: in std_logic;
        b: out std_logic);
end entity buff;

architecture behav of buff is
signal x,y: std_logic;
begin
    x <= a after t;
    y <= transport a after t;
    b <= x or y;
end architecture;
```

Esercizio 2

Si descriva in maniera comportamentale un componente combinatorio che ha in ingresso due parole di 8 bit ciascuna ($a_{7..0}$ e $b_{7..0}$) che rappresentano interi senza segno A e B e 2 bit (p_a e p_b) che rappresentano rispettivamente i bit di paritá di a e b. La rete produce in uscita ($x_{7..0}$) la somma $X = A + B$, il bit di paritá p_x di x e un bit di errore che assume il valore 1 in caso di errore su a o b.

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

entity add is
   port(a: in std_logic_vector(7 downto 0);
        b: in std_logic_vector(7 downto 0);
        p_a: in std_logic;
        p_b: in std_logic;
```

```
        x: out std_logic_vector(7 downto 0);
        p_x: out std_logic;
        e: out std_logic);
end entity add;

architecture behav of add is
signal s_a,s_b: std_logic;

signal sum: std_logic_vector(7 downto 0);
begin
    process(a,b)
    begin
      sum<=std_logic_vector(unsigned(a)+unsigned(b));
    end process;

    s_a <= a(0) xor a(1) xor a(2) xor a(3) xor a(4) xor a(5)
           xor a(6) xor a(7);
    s_b <= b(0) xor b(1) xor b(2) xor b(3) xor b(4) xor b(5) xor
           b(6) xor b(7);
    e <= (p_a xor s_a) or (p_b xor s_b);
    x <= sum;
    p_x <= sum(0) xor sum(1) xor sum(2) xor sum(3) xor sum(4) xor sum(5)
           xor sum(6) xor sum(7);

end architecture;
```

Esercizio 3

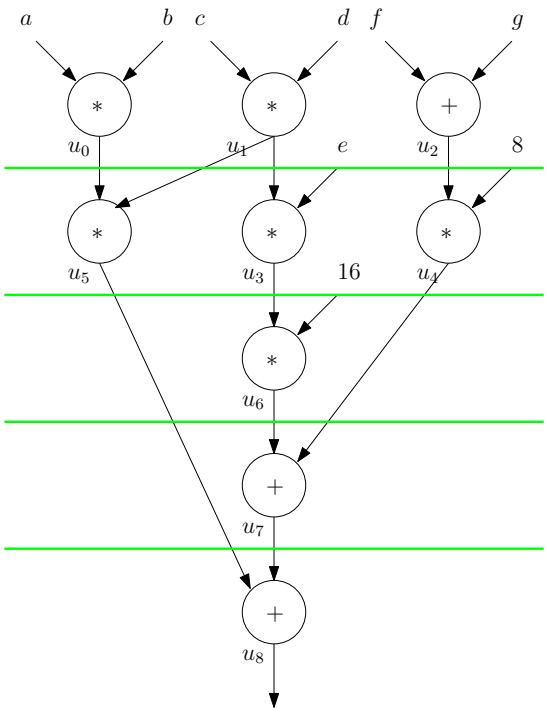
Si consideri il seguente algoritmo:

```
0. u0:=a*b;
1. u1:=c*d;
2. u2:=f+g;
3. u3:=u1*e;
4. u4:=u2*8;
5. u5:=u0*u1;
6. u6:=u3*16;
7. u7:=u6+u4;
8. u8:=u5+u7;
```

Si tracci il DFG e si determinino lo scheduling ASAP e quello ALAP. Si determini poi uno scheduling che utilizza un moltiplicatore e un sommatore minimizzando la latenza (nell'ipotesi di ciclo singolo). Si consideri poi la possibilitá di aggiungere uno shifter $\ll$ per realizzare le operazioni di prodotto con le costanti. Si tracci prima il DFG con tale nuovo operatore e si determini se, nelle condizioni specificate dalla prima domanda, si ottiene qualche vantaggio nella latenza.

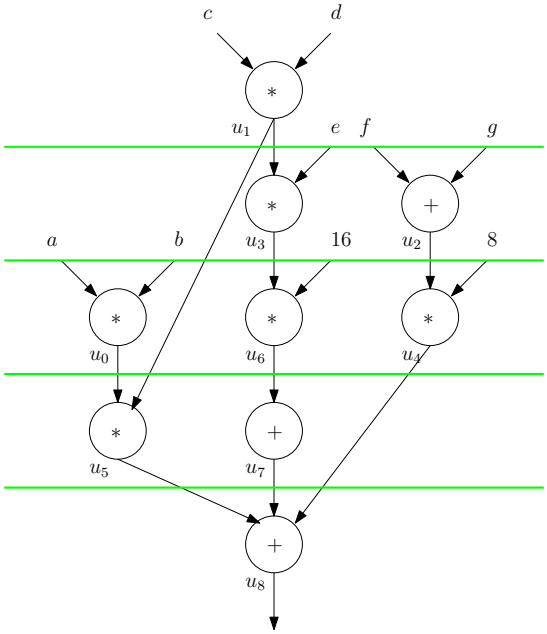
Soluzione

Scheduling ASAP



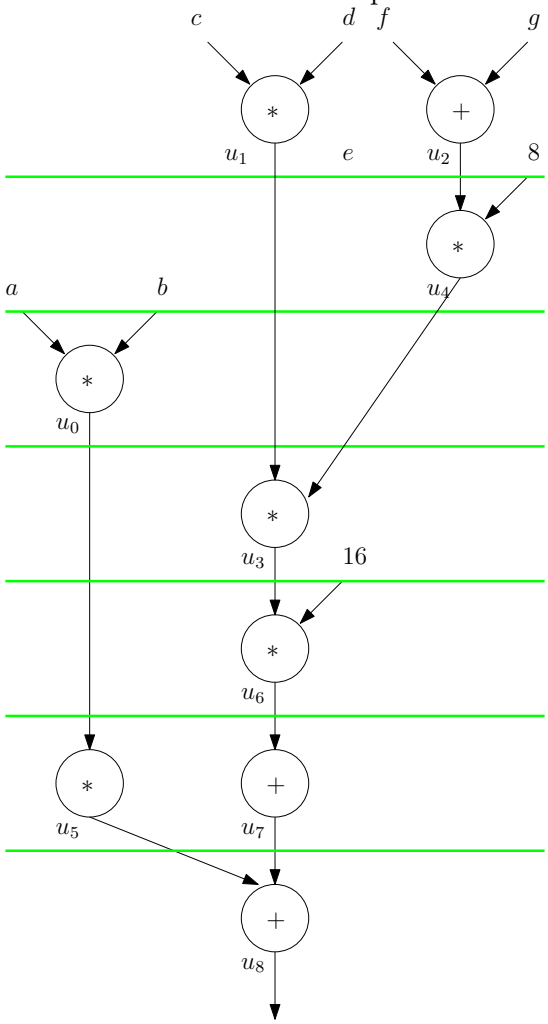
latenza = 5, risorse = 2 multipier, 1 adder

Scheduling ALAP



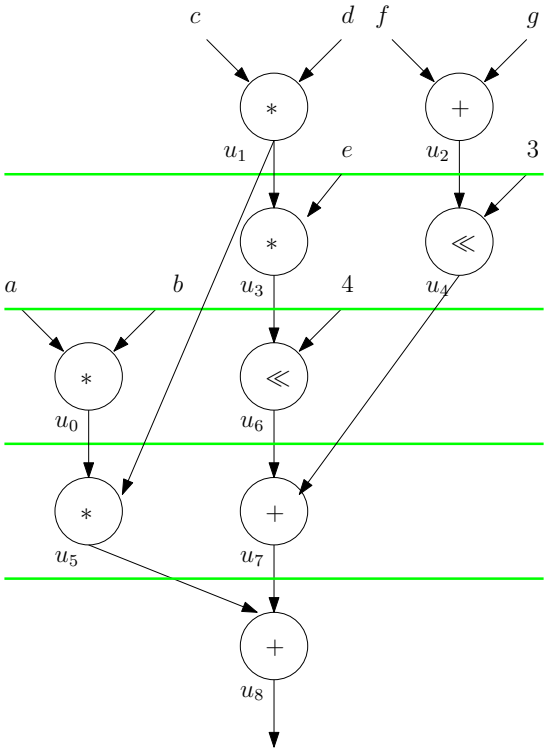
latenza = 5, risorse = 3 multiplier, 1 adder

Scheduling con risorse limitate a 1 adder e 1 multiplier e latenza ottimizzata



latenza = 7

Scheduling con risorse limitate a 1 adder, 1 multiplier e 1 shifter. Con latenza ottimizzata



latenza = 5