Compito di linguaggi di descrizione dell'hardware

Esercizio 1

Si realizzi un modello comportamentale in VHDL di un flip-flop di tipo D. Il flip-flop ha in ingresso il dato D e il clock. Sono poi dati due parametri δ e t_{cq} come generic. Il segnale D viene campionato tre volte sul fronte di salita del clock, dopo δ e dopo 2δ ottenendo cosí 3 valori utilizzati per calcolare Q che assume un valore pari alla maggioranza dei valori campionati (il nuovo valore di Q viene prodotto con un ritardo t_{cq} rispetto all'ultimo campionamento).

Soluzione

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

entity ff is
  generic(d,tcq: time);
  port(d,clk: in std_logic;
       q: out std_logic);
end entity ff;

architecture behav of ff is
signal clk1, clk2: std_logic;
begin
  clk1<=clk after d;
  clk2<=clk after 2*d;

  process(clk,clk1,clk2)
  variable tmp,tmp1,tmp2: std_logic;
  begin
    if (rising_edge(clk)) then
      tmp:=d;
    elsif (rising_edge(clk1)) then
         tmp1:=d;
    elsif (rising_edge(clk2)) then
         tmp2:=d;
    end if;
    -- we use the equation of a majority voter
    -- if-then-else description works as well
    q<=(tmp and tmp1) or (tmp and tmp2) or (tmp1 and tmp2) after tcq;
  end process;
end architecture;
```

Esercizio 2 Si realizzi la descrizione comportamentale di una rete sincrona che riceve in ingresso una parola $a_{7..0}$, un segnale di *reset* e il segnale di *clock*. Compito della rete é sommare (modulo 2^8) i valori (di tipo unsigned) che arrivano su a e produrre continuamente il risultato in uscita $o_{7..0}$. Quando il segnale di *reset* si porta 1 la somma viene posta a 0. Il segnale di reset agisce in maniera sincrona e la rete si sincronizza sui fronti di salita del segnale di clock.

Soluzione

Per far funzionare la rete descritta, c'é bisogno di un segnale di reset che non era richiesto dal testo

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

entity add_seq is
  port(a: in std_logic_vector(7 downto 0);
       clk: in std_logic;
       o: out std_logic_vector(7 downto 0));
end entity add_seq;

architecture behav of add_seq is
begin
  process(clk)
  variable sum: unsigned(7 downto 0);
  begin
    if (rising_edge(clk)) then
      sum:=sum+unsigned(a);
      o<=std_logic_vector(sum);
    end if;
  end process;
end architecture;
```

Esercizio 3

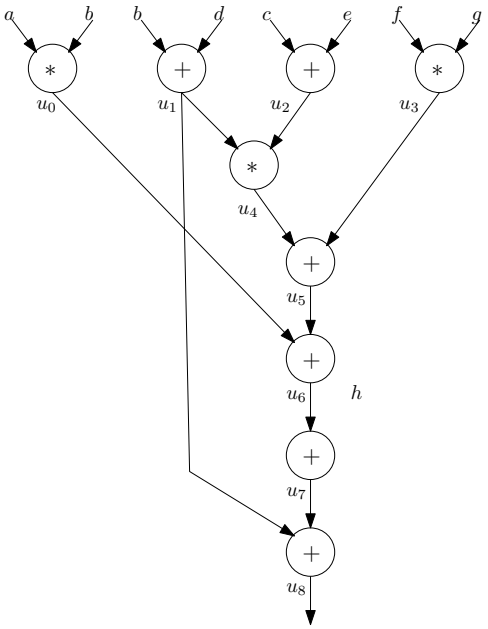
Si consideri il seguente algoritmo:

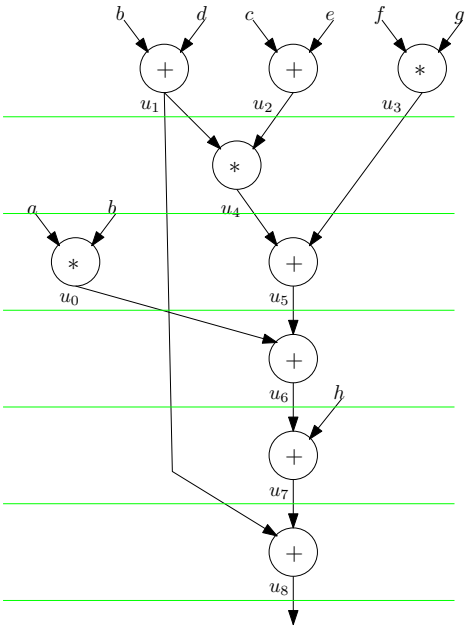
```
u0:=a*b;
u1:=b+d;
u2:=c+e;
u3:=f*g;
u4:=u1*u2;
u5:=u4+u3;
u6:=u0+u5;
u7:=u6+h;
u8:=u7+u1;
```

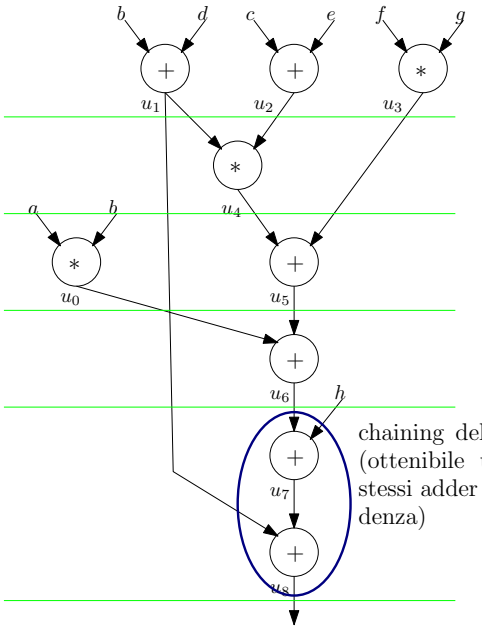
si tracci il DFG e, nell'ipotesi di ciclo singolo, si determini poi uno scheduling a latenza minima minimizzando le risorse utilizzate. Supponendo che il ritardo massimo di un sommatore sia pari a 1 ns e quello di un moltiplicatore a 8 ns, si calcoli il periodo di clock con cui tale rete puó operare supponendo che i ritardi associati a multiplexer e flip-flop non superino 0.5 ns. Si calcoli la latenza in ns. Si abbandoni l'ipotesi di ciclo singolo e si individui una tecnica che consenta di ridurre la latenza, si descriva brevemente tale soluzione mostrando il DFG con il nuovo scheduling. Si spieghi il motivo per cui tale operazione puó essere fatta senza il bisogno di aggiungere nuove unitá funzionali.

Soluzione

Lo scheduling a latenza minima richiede un moltiplicatore e due adder, la sua latenza relativa é 6. La rete combinatoria con il ritardo massimo é il moltiplicatore che ha 8 ns di ritardo massimo cui si possono sommare gli 0.5 ns relativi ai multiplexer e ai flip-flop per cui si puó usare un periodo $T = 8.5ns$ cui corrisponde una latenza assoluta pari a $6T = 51ns$. Se si osserva che nel DFG con tale scheduling le due ultime operazione sono delle somme e che possono essere concatenate in maniera combinatoria con la tecnica detta operation chaining, che é possibile perché la somma dei loro ritardi rimane inferiore al periodo di clock. Questa operazione ha un costo trascurabile perché si possono utilizzare i due sommatori utilizzati per le operazioni precedenti. La latenza relativa diventa pari a 5 e quella assoluta $5T = 42.5ns$.







chaining delle due somme (ottenibile utilizzando gli stessi adder usati in precedenza)