Compito di linguaggi di descrizione dell'hardware

Esercizio 1

Si realizzi un modello comportamentale in VHDL di un registro a n bit che campiona sul fronte positivo del segnale di clock. Tale registro riceve una parola $d_{0..n-1}$ in ingresso e produce $q_{0..n-1}$ in uscita. Il componente riceve anche il valore del tempo di risposta t_{cq} e di quello di setup t_{dc} passati come generic insieme alla sua dimensione n. In caso di violazioni del tempo di setup, si devono portare a X solamente i segnali (o il segnale) di uscita che corrispondono agli ingressi (all'ingresso) che hanno (ha) effettivamente violato tale condizione (pt. 5.0).

Soluzione

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

entity reg is
 generic(n: integer;
         tcq, tdc: time);
 port(d: in std_logic_vector(0 to n-1);
      clk: in std_logic;
      q: out std_logic_vector(0 to n-1));
end entity;

-- soluzione molto semplificata
architecture behav of reg is
begin
 p: process(clk)
    begin
     if (clk='1' and clk'last_value='0') then
       for i in 0 to n-1 loop
         if (d(i)'stable(tdc)) then
           q(i) <= d(i) after tcq;
         else
           q(i) <= 'X' after tcq;
         end if;
       end loop;
     end if;
    end process;
end architecture;
```

Esercizio 2

Si descriva al livello comportamentale in VHDL un componente combinatorio che riceve in ingresso tre parole a, b e c. Tali parole rappresentano interi con segno (A, B e C) in complemento a 2. Tali parole hanno dimensioni diverse: a ha 16 bit, b e c ne hanno 8. L'uscita s deve contenere (sempre in complemento a 2) il valore di $A + B + C$ senza che ci siano overflow o underflow (pt. 5.0).

Soluzione

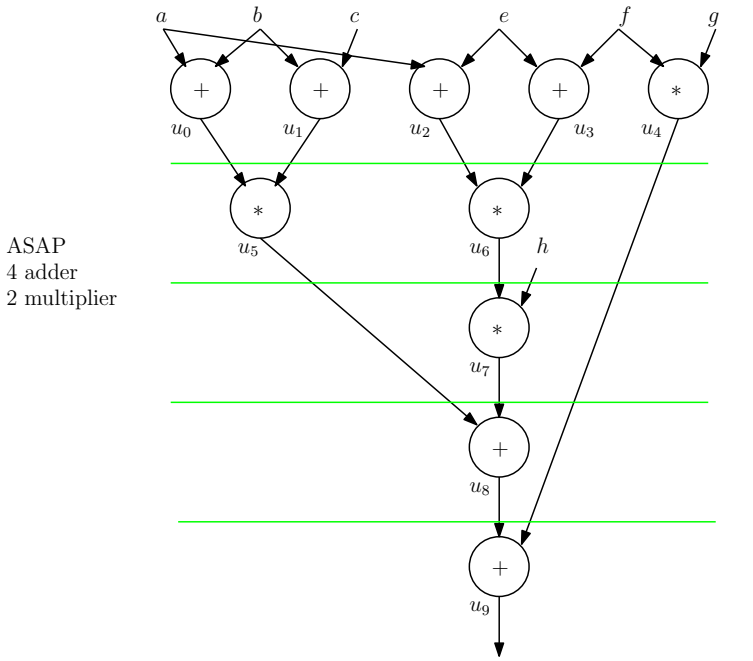
```
library ieee;
use ieee.std_logic_1164.all, ieee.numeric_std.all;
-- -2^14 <= a <= 2^14-1, -2^7 <= b,c, <= 2^7-1
-- -2^14-2^8 <= a+b+c <= 2^14+2^8-2
-- quindi per l'uscita vanno bene 17 bit
entity adder is
 port(a: in std_logic_vector(15 downto 0);
      b,c: in std_logic_vector(7 downto 0);
      s: out std_logic_vector(16 downto 0));
end entity adder;

architecture behav of adder is
begin

 process(a,b,c)
 variable tmpa,tmpb,tmpc: signed(16 downto 0);
 begin
  tmpa:=signed(a(15) & a);
  tmpb:=signed((others => b(7)) & b);
  tmpc:=signed((others => c(7)) & c);
  s <= std_logic_vector(a+b+c);
 end process;

end architecture;
```

Esercizio 3

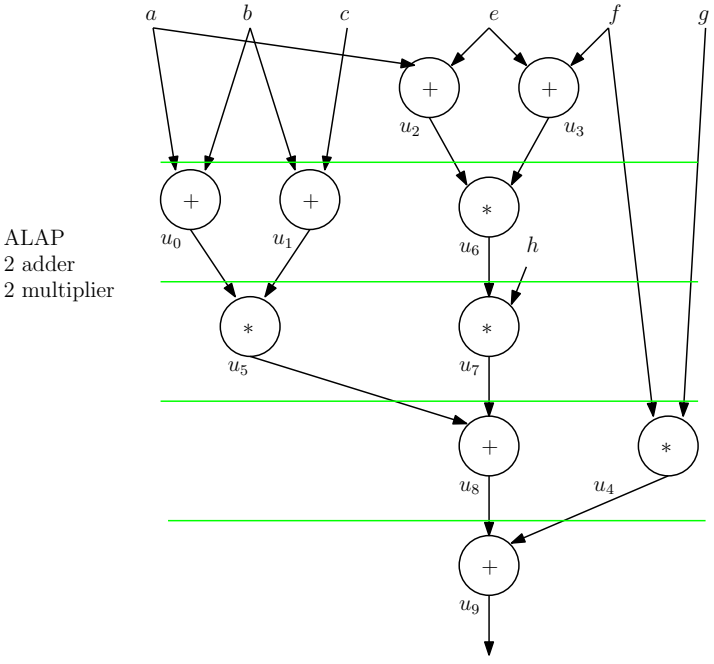
Si consideri questo algoritmo e se ne tracci il DFG. Si traccino poi i DFG per lo scheduling ASAP e quello ALAP indicando il numero di risorse da allocare.

```
0. u0:=a+b;
1. u1:=b+c;
2. u2:=a+e;
3. u3:=e+f;
4. u4:=f*g;
5. u5:=u0*u1;
6. u6:=u2*u3;
7. u7:=u6*h;
8. u8:=u7+u5;
9. u9:=u8+u4;
```

Nell'ipotesi di ciclo singolo, si determini poi uno scheduling che, per una latenza pari a quella minima, minimizzi il numero di risorse utilizzate. Si supponga che il ritardo di un adder sia 2.0ns e quello di un moltiplicatore sia pari a 4.8ns. Supponendo poi di utilizzare 0.5ns per multiplexer e tempi di risposta e setup dei FF, si determini il periodo minimo di clock utilizzabile e la latenza assoluta associata alla rete (in ns, non in cicli di clock) (pt. 5.0).
Soluzione



ASAP
4 adder
2 multiplier



ALAP
2 adder
2 multiplier

Si noti che lo scheduling ALAP é anche uno dei possibili scheduling a risorse minime.

Nell'ipotesi di ciclo singolo, il ritardo di cui tenere conto é quello del moltiplicatore cui vanno sommati i $0.5ns$ di mpx e flip-flop. Quindi deve essere $T > 5.3ns$, la latenza minima é quindi pari a $5 \times 5.3ns = 26.5ns$.