Compito di linguaggi di descrizione dell'hardware

Esercizio 1

Si realizzi un modello comportamentale in VHDL di un latch D trasparente che é nella fase di sample con il segnale di clock alto. Il modello del latch riceve come parametri generic t_1 e t_{cq} (tempo di risposta). Il parametro t_1 é di tipo time e rappresenta il tempo minimo per cui il clock deve stare a 1. Se tale condizione viene riconosciuta come non valida, allora q si porta a X. Si trascuri il problema del tempo di setup (pt. 5.0).

Si noti che, dato un segnale s, `s'event` e `s'stable{t}` (con `t` > 0) non possono mai essere vere contemporaneamente.

Soluzione

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

entity latch is
 generic(tcq,t1: time);
 port(d,clk: in std_logic;
      q: out std_logic);
end entity latch;

architecture behav of latch is
signal tmp_clk: std_logic;
begin
 tmp_clk <= clk after t1; -- exploit inertial delay
 process(d,clk)
 begin
  if (clk='1') then
    q <= d after tcq; -- sample
  elsif ((clk='0') and (clk'event)
         and (tmp_clk/='1')) then
    q <= 'X' after tcq;
  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 di 8 bit ciascuna. Tali parole rappresentano numeri naturali (A, B e C). La rete deve determinare il massimo e il minimo fra

questi valori e produrre in uscita (sempre a 8 bit) la differenza fra tali due valori (pt. 5.0).

Si raccomanda di descrivere tutte le possibili condizioni (ordinamenti) in ingresso, altrimenti la sintesi vi mette dei latch.

Soluzione

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

entity max_min is
 port(a,b,c: in std_logic_vector(7 downto 0);
      y: out std_logic_vector(7 downto 0));
end entity rete;

architecture behav of max_min is
begin
 process(a,b,c)
 variable max,min: unsigned(7 downto 0);
 begin
  max:=unsigned(a);
  min:=unsigned(a);
  if (b>max) then
    max:=unsigned(b);
  end if;
  if (b<min) then
    min:=unsigned(b);
  end if;
  if (c>max) then
    max:=unsigned(c);
  end if;
  if (c<min) then
    min:=unsigned(c);
  end if;

  y<=std_logic_vector(max-min);
 end process;
end architecture;
```

Esercizio 3

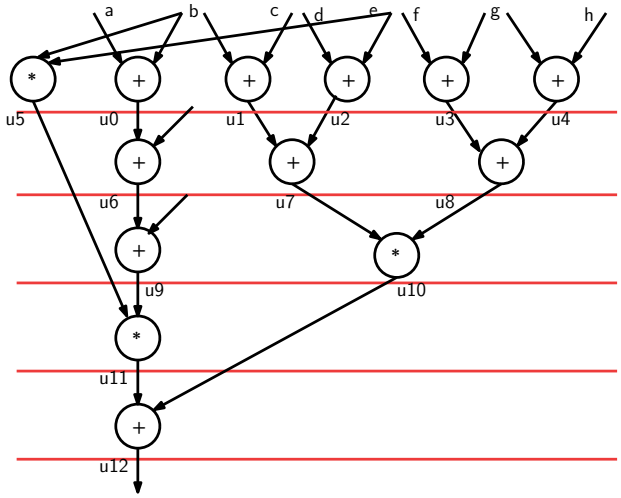
Si consideri questo algoritmo e se ne tracci il DFG. Si traccino poi i DFG per lo scheduling ASAP e quello ALAP nell'ipotesi di ciclo singolo indicando il numero di risorse da allocare.

```
0. u0:=a+b;     6. u6:=j+u0;
1. u1:=b+c;     7. u7:=u1+u2;
2. u2:=d+e;     8. u8:=u3+u4;
3. u3:=f+g;     9. u9:=u6+k;
4. u4:=g+h;     10. u10:=u7*u8;
5. u5:=b*e;     11. u11:=u5*u9;
                12. u12:=u10+u11;
```

Poi si determini uno scheduling che, per una latenza pari a quella minima, minimizzi il costo della rete nell'ipotesi che il costo dei moltiplicatori sia molto piú grande di quello dei sommatori (pt. 4.0). Si discuta il motivo per cui in questo caso é stato necessario aggiungere questa ipotesi sul costo dei componenti, mostrando eventualmente uno scheduling alternativo utilizzabile se i due costi dovessero essere confrontabili (pt. 1.0).

Soluzione

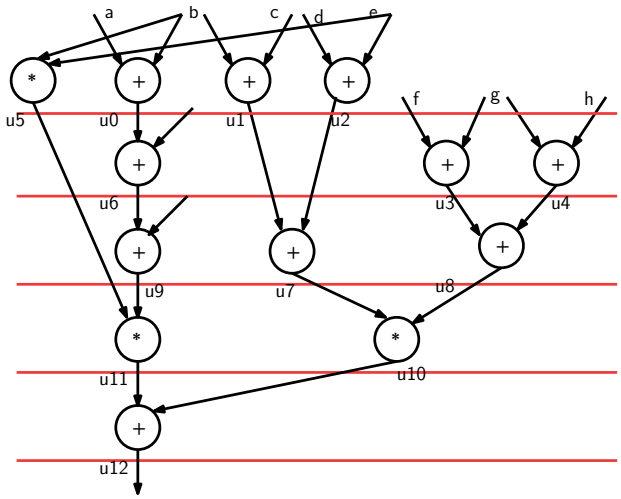
Scheduling ASAP 1 moltiplicatore e 5 adder



Lo scheduling ALAP non ḿostrato e ha 2 moltiplicatori e 5 adder.

Lo scheduling ASAP nell'ipotesi che il moltiplicatore sia molto piú costoso dell'adder ha il costo minimo

Se invece, i due costi sono confrontabili, questo scheduling ha il costo minore in quanto l'allocazione richiede 2 moltiplicatori e 3 adder. Questo ci consente di dire che questa soluzione conviene se il moltiplicatore costa 2 volte un adder (cosa difficilmente verificata).


a
b
c
d
e
f
g
h
*
+
+
+
+
+
+
+
+
*
*
u5
u0
u1
u2
u6
u3
u4
u9
u7
u8
u11
u10
u12