Compito di linguaggi di descrizione dell'hardware

Esercizio 1

Si realizzi un modello comportamentale in VHDL di un full-adder. I ritardi di tale componente siano caratterizzati da: i) $t_r(sum)$ ritardo in salita per l'uscita sum; ii) $t_f(sum)$ ritardo in discesa per l'uscita sum; iii) $t_r(cout)$ritardo in salita per l'uscita $cout$; iv) $t_f(cout)$ ritardo in discesa per l'uscita $cout$. Tali parametri siano passati come generic. Le transizioni di un uscita da `0,1` a `X` devono utilizzare come ritardo il minimo valore dei ritardi che caratterizzano tale uscita.

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

entity fa is
  generic (trs,tfs,trcout,tfcout: time);
  port (a,b,cin: in std_logic;
        s,cout: out std_logic);
end entity fa;

architecture behav of fa is
begin
  process(a,b,cin)
    variable sum,co: std_logic;  -- temporary variables
    variable k:boolean:=TRUE;  -- flag
    variable tsmin,tcomin: time;
    begin
      -- compute the minimal delay at the first execution
      -- of the process
      if (k) then
        tsmin:=trs;
        tcomin:=trcout;
        if (tfs<tsmin) then
          tsmin:=tfs;
        end if;
        if (tfcout<tcomin) then
          tcomin:=tfcout;
        end if ;
        k:= FALSE;
      end if ;
      -- compute the outputs and select the delay
      sum:=a xor b xor cin;
      co:= (a and b) or (a and cin) or (b and cin);
      case sum is
      when '1' => s<='1' after trs;
      when '0' => s<='0' after tfs;
```

```
      when others => s<='X' after tsmin;
      end case;
      case co is
      when '1' => cout<='1' after trcout;
      when '0' => cout<='0' after tfcout;
      when others => cout<='X' after tcomin;
      end case;
    end process;
  end architecture behav;
```

Esercizio 2

Si descriva al livello comportamentale in VHDL un FF edge-triggered di tipo D con le seguenti caratteristiche: i) campionamento sul fronte di discesa del clock; ii) WE sincrono; iii) SET e RESET asincroni; iv) tempo di setup τ_{SU} e tempo di risposta τ_R passti come generic.

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

entity dff_cell is
   generic(tr,tsu: time);
   port(d,clk,we,set,reset: in std_logic;
        q: out std_logic);

end enitity dff;

architecture behav of dff_cell is
begin

  process(clk,set,reset)
  begin
    if (set='1') then
      q<='1' after tr;
    elsif (reset='1') then
         q<='0' after tr;
    elsif (set='0') and (reset='0') then
       if (clk='0') and (clk'event) and (clk'last_value='1')
          and (we='1') then
         if (d'stable(tsu)) then
           q<=d after tr;
         else
           q<='X';
         end if;
       end if;
    else
    -- set, reset have invalid values
      q<='X';
```

```
    end if;
    end process;
end architecture behav;
```

Esercizio 3

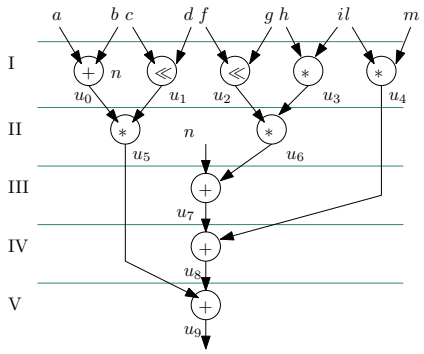
Si consideri il seguente algoritmo:

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

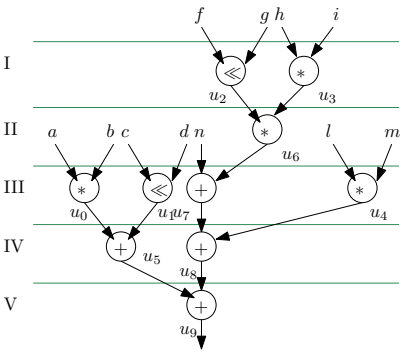
Si tracci il DFG e si determinino lo scheduling ASAP e quello ALAP. Si determini poi uno scheduling a latenza minima che ottimizzi il numero di risorse utilizzate (nell'ipotesi di ciclo singolo). Si determini poi il numero minimo di registri da utilizzare, il binding delle risorse e si descriva il componente al livello RTL. Si determini la differenza di costo fra l'allocazione ottenuta e quella che si avrebbe sostituendo gli shifter con moltiplicatori (si considerino i seguenti costi $add = 1$, $mult = 16$ $shifter = 2$).

Algoritmo

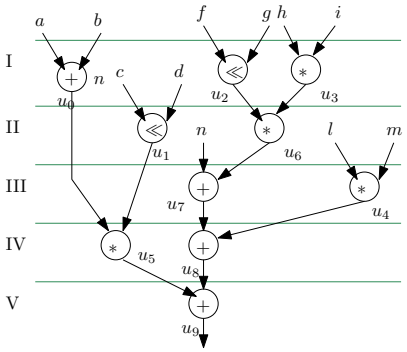
```
u0:=a+b;
u1:=c<<d;
u2:=f<<g;
u3:=h*i;
u4:=l*m;
u5:=u0+u1;
u6:=u2*u3;
u7:=u6+n;
u8:=u7+u4;
u9:=u8+u5;
```



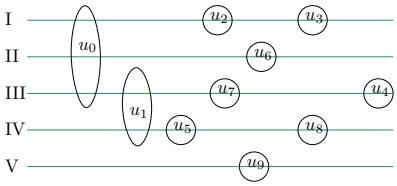
Scheduling ASAP: allocazione 2 shifter, 2 moltiplicatori, 1 sommatore



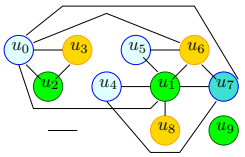
Scheduling ALAP: allocazione 2 moltiplicatori, 1 shifter, 2 sommatori



Scheduling a latenza minima con ottimizzazione del numero di risorse: allocazione 1 moltiplicatore, 1 shifter, 1 sommatore



Tempo di vita delle variabili



Ottimizzazione del numero di registri tramite graph coloring

Binding e descrizione RTL

```
    adder       shifter      mult
-------------------------------------------------------
I)  R1:=a+b;    R3:=f≪g:  R3:=h*i;
II)             R2:=c≪d   R1:=l*R1;
III) R4:=R2+n;            R1:=m*R1;
IV)  R2:=R4+R1;           R1:=R1*R3;
V)   R3:=R1+R2;
```