

Lucrarea de laborator 2

Sinteza și implementarea circuitelor logice combinate

Scopul lucrării: Proiectarea, testarea și simularea circuitelor logice combinate în mediul de dezvoltare software Altera Quartus II. Descrierea circuitului va fi efectuată în limbajul VHDL, folosind codificarea structurală, flux de date și comportamentală.

Proiectarea structurală a circuitului sumator/scăzător de 4 biți

Vom analiza etapele de proiectare a unui sumator/scăzător de 4 biți (Fig. 1). Circuitul constă din 4 sumatoare complete de un bit, unite consecutiv și 4 porți XOR.

Intrările circuitului:

- două numere A și B de patru biți;
- intrarea de selecție SEL pentru alegerea modului de lucru: adunare sau scădere.

Ieșirile circuitului:

- suma Sum de patru biți;
- transportul $Cout$.

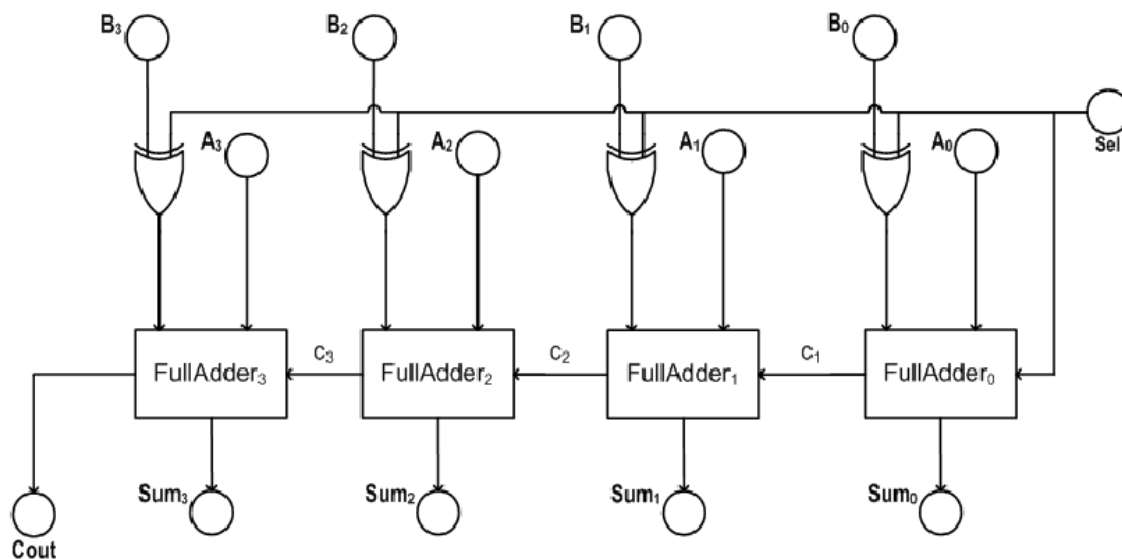


Figura 1. Sumator serial pe 4 biți.

Circuitul sumatorului complet este prezentat în Figura 2. Circuitul constă din două porți XOR cu 2 intrări, trei porți ȘI cu 2 intrări și o poartă SAU cu 3 intrări. Fiecare poartă logică va fi descrisă utilizând codificarea comportamentală, după care aceste coduri vor fi folosite în descrierea structurală a sumatorului complet.

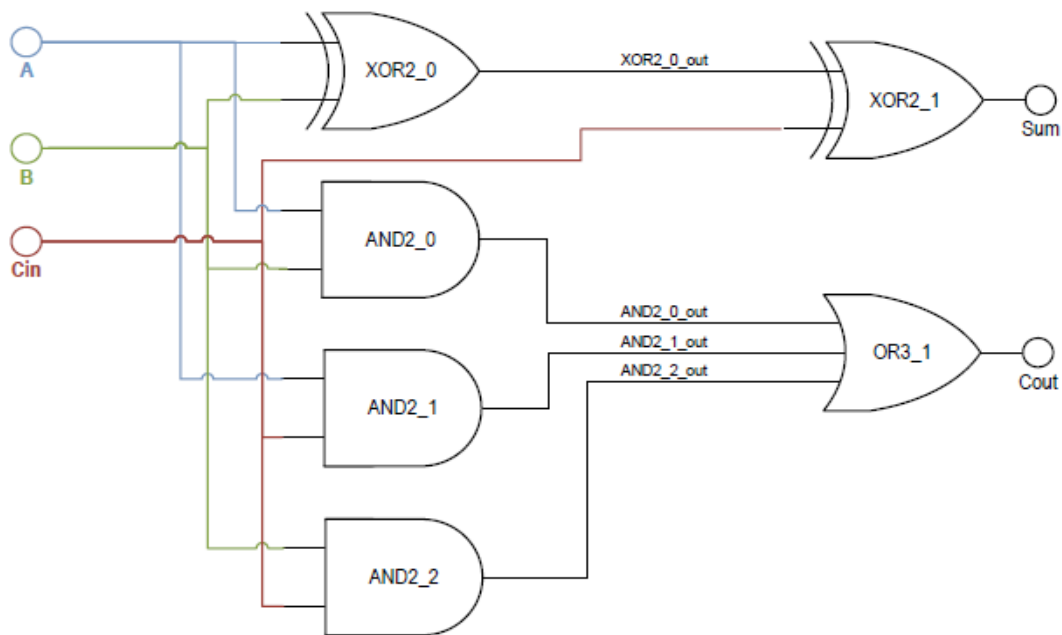


Figura 2. Circuitul logic al sumatorului complet

Proiectarea sumatorului

1. Crearea unui proiect nou

2. Crearea, adăugarea și compilarea fișierelor de proiect

a) Se selectează *File* → *New File* din bara de meniu. Se alege *VHDL File* din lista fișierelor de proiect.

Se introduce codul pentru descrierea comportamentală a porții logice *AND* cu două intrări și se salvează cu numele *AND2*.

Program comportamental pentru poarta logică AND cu două intrări

```

library ieee;
use ieee.std_logic_1164.all;
entity Gate_And2 is
  port (x, y: in std_logic;
        F: out std_logic);
end Gate_And2;
architecture Gate_And2_beh of Gate_And2 is
begin
  process (x, y)
  begin
    F <= x and y;
  end process;
end Gate_And2_beh;

```

Codul VHDL reprezintă un program comportamental pentru poarta logică AND cu două intrări. În lista de sensibilitate a procesului sunt incluse semnalele *x*

și y . Deci, de fiecare dată, când unul dintre aceste semnale își va schimba valoarea, procesul se va executa și va genera un nou rezultat.

b) Se creează un fișier VHDL nou și se introduce codul comportamental pentru poarta logică **XOR**. Fișierul se salvează cu numele **XOR2**.

Program comportamental pentru poarta logică **XOR** cu două intrări

```
library ieee;
use ieee.std_logic_1164.all;
entity Gate_XOR2 is
  port (x, y: in std_logic;
        F: out std_logic);
end Gate_XOR2;
architecture Gate_XOR2_beh of Gate_XOR2 is
begin
  process (x, y)
  begin
    F <= x XOR y;
  end process;
end Gate_XOR2_beh;
```

c) Se creează un fișier VHDL nou și se introduce codul comportamental pentru poarta logică **OR** cu trei intrări. Fișierul se salvează cu numele **OR3**.

Program comportamental pentru poarta logică **OR** cu trei intrări

```
library ieee;
use ieee.std_logic_1164.all;
entity Gate_OR3 is
  port (x, y, z: in std_logic;
        F: out std_logic);
end Gate_OR3;
architecture Gate_OR3_beh of Gate_OR3 is
begin
  process (x, y, z)
  begin
    F <= x OR y OR z;
  end process;
end Gate_OR3_beh;
```

d) Se creează un fișier VHDL nou și se introduce codul structural pentru sumatorul complet pe un bit (Tabelul 4). Fișierul se salvează cu numele FullAdder.

Program structural pentru sumatorul complet pe un bit

```
library ieee;
use ieee.std_logic_1164.all;
entity FullAdder is
  port (A, B, Cin: in std_logic; SUM, Cout: out std_logic);
end FullAdder;
architecture FullAdder_struct of FullAdder is
  component Gate_And2 is
    port (x, y: in std_logic; f: out std_logic); end component;
  component Gate_XOR2 is
    port (x, y: in std_logic; f: out std_logic); end component;
  component Gate_OR3 is
    port (x, y, z: in std_logic; f: out std_logic);
  end component;
  signal XOR2_0_out, AND2_0_out, AND2_1_out, AND2_2_out: std_logic;
  begin
    XOR2_0: Gate_XOR2 port map (A, B, XOR2_0_out);
    XOR2_1: Gate_XOR2 port map (XOR2_0_out, Cin, Sum);
    AND2_0: Gate_AND2 port map (A, B, AND2_0_out);
    AND2_1: Gate_AND2 port map (A, Cin, AND2_1_out);
    AND2_2: Gate_AND2 port map (B, Cin, AND2_2_out);
    OR3_1: Gate_OR3 port map (AND2_0_out, AND2_1_out, AND2_2_out, Cout);
  end FullAdder_struct;
```

Acest exemplu reprezintă un program VHDL structural care utilizează mai multe componente, create anterior. În program sunt folosite două instanțieri pentru poarta *XOR*, trei – pentru poarta *AND* și una – pentru poarta *OR*. Fiecare componentă utilizează modul pozițional de atribuire a semnalelor. Semnalele *XOR2_0_out*, *AND2_0_out*, *AND2_1_out* și *AND2_2_out* sunt semnale interne ale circuitului.

e) În final se creează un fișier VHDL nou și se introduce codul structural pentru sumatorul serial de patru biți (Tabelul 5). *Fișierul se salvează cu același nume ca și numele entităților.*


Program structural pentru sumatorul serial de patru biți

```
library ieee;
use ieee.std_logic_1164.all;
entity Lab2 is
  port (A, B : in std_logic_vector (3 downto 0);
        Sel: in std_logic;
        Sum: out std_logic_vector (3 downto 0);
        Cout: out std_logic);
end Lab2;
```

```

architecture AddSub_struct of Lab2 is
  component Gate_XOR2 is
    port (x, y: in std_logic;
          f: out std_logic);
  end component;
  component FullAdder is
    port (A, B, Cin: in std_logic;
          Sum, Cout: out std_logic);
  end component;
  signal XOR2_0_out, XOR2_1_out, XOR2_2_out, XOR2_3_out: std_logic;
  signal c1, c2, c3: std_logic;
  begin
  XOR2_0: Gate_XOR2 port map (B(0), Sel, XOR2_0_out);
  XOR2_1: Gate_XOR2 port map (B(1), Sel, XOR2_1_out);
  XOR2_2: Gate_XOR2 port map (B(2), Sel, XOR2_2_out);
  XOR2_3: Gate_XOR2 port map (B(3), Sel, XOR2_3_out);
  FullAdder_0: FullAdder port map (A(0), XOR2_0_out, Sel, Sum(0), c1);
  FullAdder_1: FullAdder port map (A(1), XOR2_1_out, c1, Sum(1), c2);
  FullAdder_2: FullAdder port map (A(2), XOR2_2_out, c2, Sum(2), c3);
  FullAdder_3: FullAdder port map (A(3), XOR2_3_out, c3, Sum(3), Cout);
  end AddSub_struct;

```

f) Proiectul se compilează tastând butonul **Start Compilation**  de pe Bara de instrumente.

3. Simularea funcțională și temporală a proiectului


a) Pentru a putea realiza simularea funcțională trebuie să creăm un fișier cu stimulii pentru intrări. Pentru a realiza acest lucru, din meniul *File* se selectează opțiunea *New*. În fereastra care apare selectăm opțiunea *Vector Waveform File* din submeniul *Other Files*.

b) Adăugarea semnalelor pentru simulare.

Se selectează **Edit** → **Insert** → **Insert Node or Bus**.

Se tastează butonul **Node Finder** în fereastra **Insert Node or Bus**

În meniul **Filter** se selectează **Pins: all** și se tastează butonul **List** (Figura 3).

Se selectează pe rând pinii **A, B, Sel și Sum** și se transferă în fereastra **Selected Nodes**, tastând butonul . Apoi se tastează **OK**.

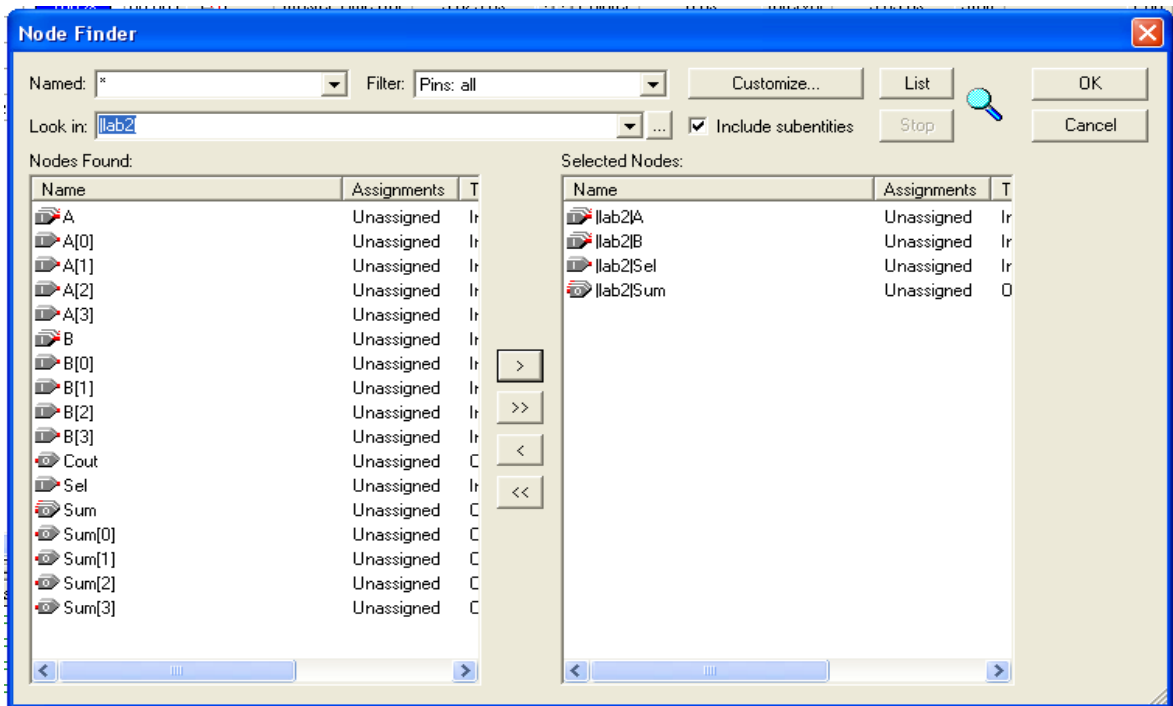


Figura 3. Selectarea nodurilor pentru simulare

În fereastra **Insert Node or Bus** se selectează **Unsigned Decimal** în opțiunea **Radix** pentru a ușura modul de introducere a valorilor de intrare și de vizualizare a rezultatelor obținute la ieșire.

Se selectează **Edit** → **End Time** pentru a deschide fereastra **End Time**. Se alege timpul de simulare de 500 ns și se tastează **OK**.

c) Atribuirea valorilor de intrare.

Se selectează intrarea **A** și se tastează **Ctrl + Alt + B** pentru a deschide fereastra **Arbitrary Value** (Figura 4). În această fereastră se aleg următoarele valori:

Start time: 40 ns
 End time: 160 ns
 Numeric or named value: 10

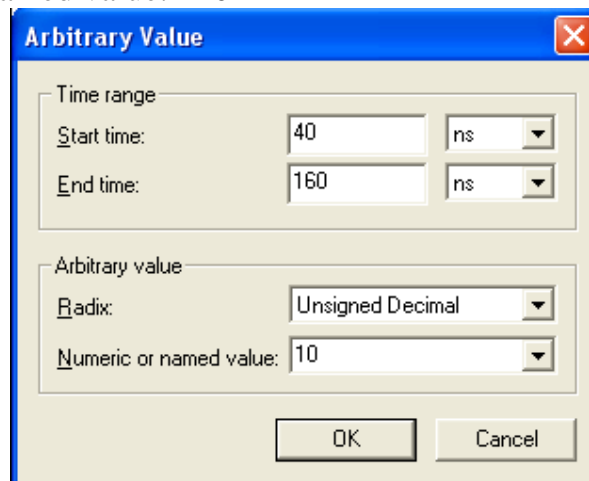


Figura 4 Atribuirea valorilor de intrare

Procedura se repetă pentru a atribui intrării **A** valoarea 7 pe intervalul 160 - 310 ns. Intrarea **B** va primi valoarea 5 pe intervalul 100 – 230 ns și valoarea 2 pe intervalul 230 – 390 ns.

Semnalul **Sel** va primi valoarea 1 logic pe intervalele 130 - 200 ns și 270 - 350 ns. Pentru aceasta se selectează pe rând intervalele menționate în fișierul de stimuli și se tastează **Ctrl+Alt+I**.

Fișierul de stimuli este prezentat în Figura 5. Acest fișier se salvează cu numele **Lab2.vwf**.

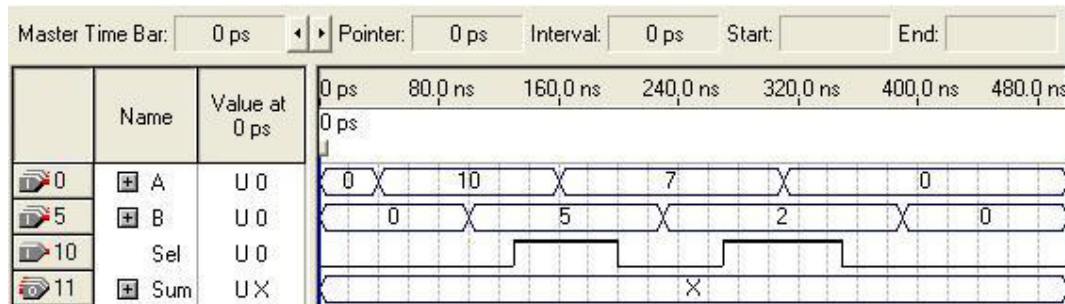



Figura 5. Fișierul de stimuli

d) Din bara de meniu se selectează **Assignments** → **Settings**. Se alege **Simulator Settings** și se setează **Simulation Mode** la **Functional**. Fișierul de intrare trebuie să fie **Lab2.vwf**.

Proiectul se compilează tastând butonul **Start Compilation**  de pe Bara de instrumente.

Se selectează **Processing** → **Generate Functional Simulation Netlist** și se tastează **OK** când procesul este complet.

Proiectul se simulează, tastând butonul **Start Simulation** de pe bara de instrumente. Formele de undă pentru **Sum** și **Cout** după simulare sunt prezentate în Figura 6.

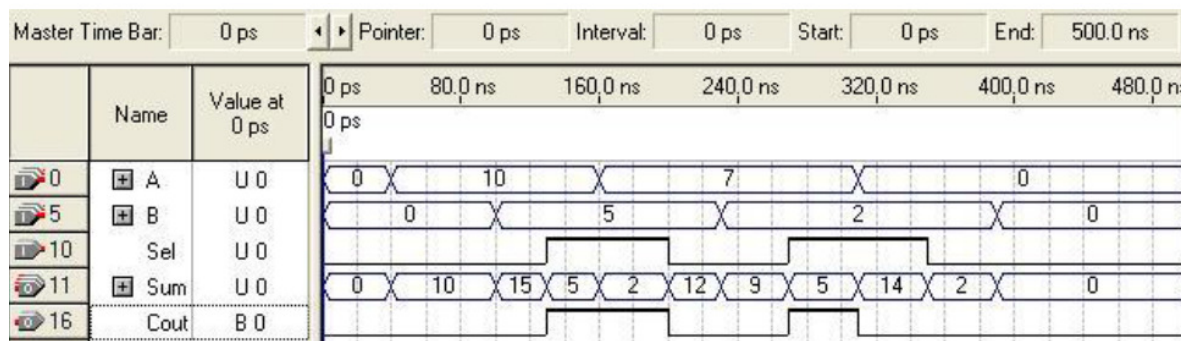


Figura 6. Formele de undă după simularea funcțională

e) Se alege modul de simulare temporal și se repetă simularea. De această dată, diagrama de timp va conține anumite hazarduri de semnal (Figura 7).

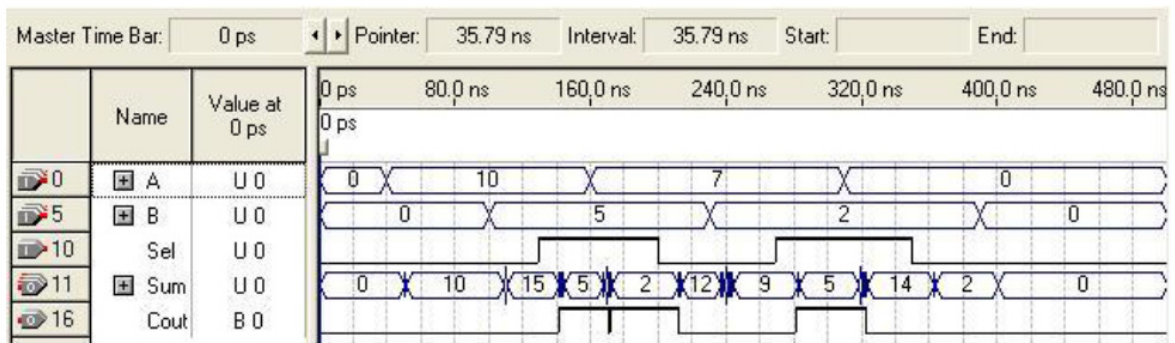


Figura 7. Formele de undă după simularea temporală

4. Asignarea pinilor

Pentru a efectua implementarea practică a schemei în circuitul FPGA de pe placa de dezvoltare DE0 este necesar ca pinii din schemă să fie atribuiți unor pini fizici ai circuitului FPGA Cyclone III EP3C16F484C6. Acest lucru se realizează prin selectarea opțiunii **Pins** din meniul **Assignments**. În fereastra care se deschide atribuim pinii.

Tabelul 1. Asignarea pinilor pentru întrerupătoare

Numele semnalului	Nr. Pinului de pe FPGA	Descriere
SW[0]	PIN_J6	Slide Switch[0]
SW[1]	PIN_H5	Slide Switch[1]
SW[2]	PIN_H6	Slide Switch[2]
SW[3]	PIN_G4	Slide Switch[3]
SW[4]	PIN_G5	Slide Switch[4]
SW[5]	PIN_J7	Slide Switch[5]
SW[6]	PIN_H7	Slide Switch[6]
SW[7]	PIN_E3	Slide Switch[7]
SW[8]	PIN_E4	Slide Switch[8]
SW[9]	PIN_D2	Slide Switch[9]

Tabelul 2. Asignarea pinilor pentru butoane

Numele semnalului	Nr. Pinului de pe FPGA	Descriere
BUTTON [0]	PIN_H2	Pushbutton[0]
BUTTON [1]	PIN_G3	Pushbutton[1]
BUTTON [2]	PIN_F1	Pushbutton[2]

Tabelul 3. Asignarea pinilor pentru led-uri

Numele semnalului	Nr. Pinului de pe FPGA	Descriere
LEDG[0]	PIN_J1	LED Green[0]
LEDG[1]	PIN_J2	LED Green[1]
LEDG[2]	PIN_J3	LED Green[2]
LEDG[3]	PIN_H1	LED Green[3]
LEDG[4]	PIN_F2	LED Green[4]
LEDG[5]	PIN_E1	LED Green[5]

LEDG[6]	PIN_C1	LED Green[6]
LEDG[7]	PIN_C2	LED Green[7]
LEDG[8]	PIN_B2	LED Green[8]
LEDG[9]	IN_B1	LED Green[9]

Se salvează această atribuire și se compilează încă o dată proiectul.

După aceasta se poate trece la implementarea proiectului pe placa de dezvoltare.

Desfășurarea lucrării

1. Se va studia exemplul de proiectare structurală a circuitului sumator/scăzător de 4 biți și, în baza lui, se va proiecta circuitul logic combinațional din lucrarea de laborator Nr. 1. Pentru descrierea componentelor (porților logice) din circuit se va utiliza codificarea comportamentală.
2. Se va proiecta circuitul logic combinațional din lucrarea de laborator Nr. 1 folosind codificarea flux de date (trei proiecte separate pentru atribuirea directă, condițională și selectivă concurrentă de semnal).
3. Se va proiecta un sumator binar-zecimal pe o tetradă pentru codul 8421 (figura 8). Se va utiliza proiectarea ierarhică. Descrierea circuitului va fi efectuată în limbajul VHDL, folosind codificarea structurală și comportamentală.

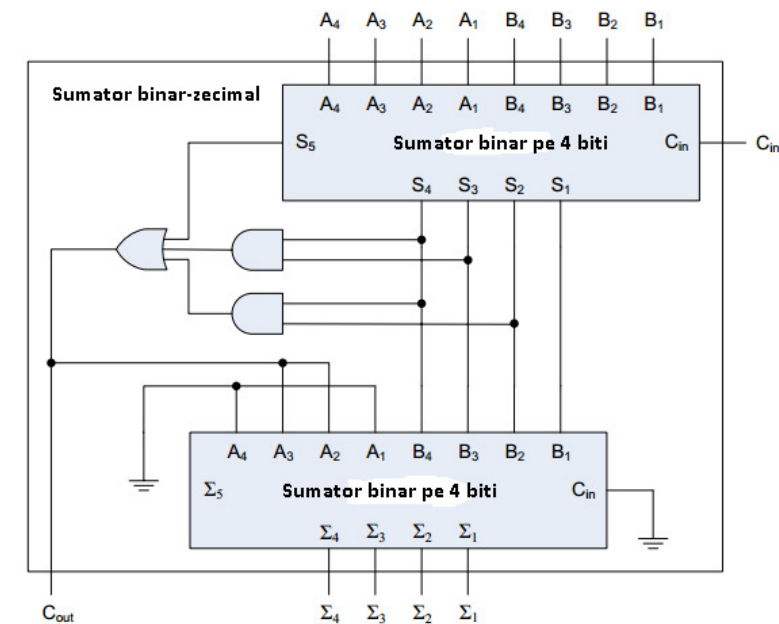


Figura 8. Sumator binar-zecimal