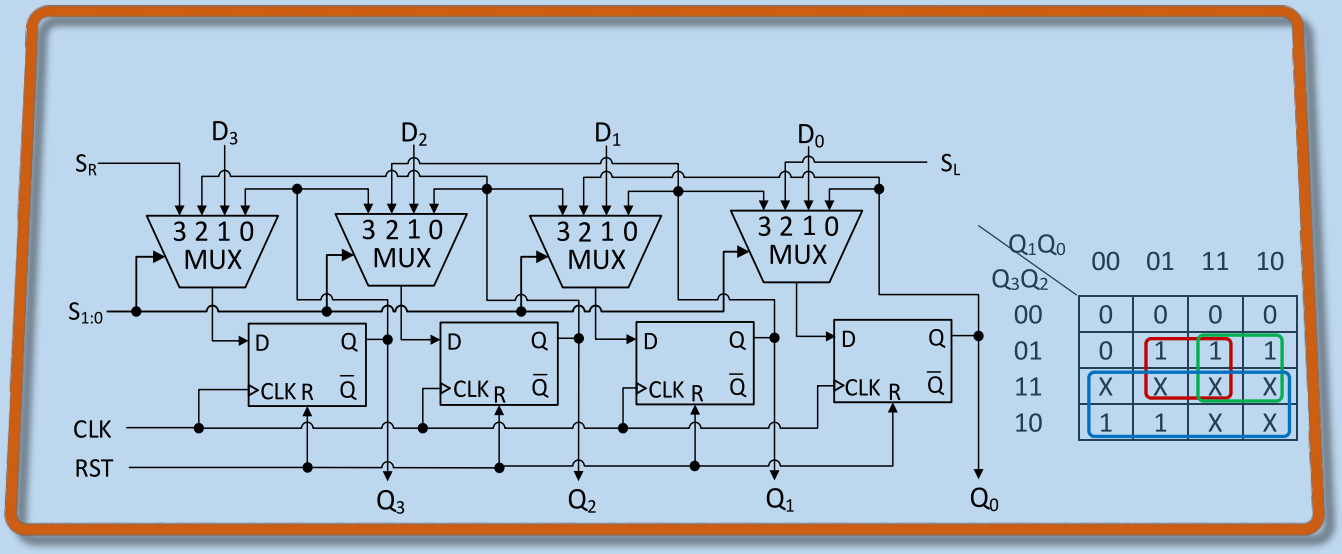
Cristian-Cosmin VANCEA

PROIECTARE LOGICĂ

Îndrumător de laborator



U.T.PRESS
Cluj-Napoca, 2026
ISBN 978-606-737-846-7

Cristian-Cosmin VANCEA

PROIECTARE LOGICĂ

Îndrumător de laborator

U.T.PRESS
Cluj-Napoca, 2026
ISBN 978-606-737-846-7

Editura U.T.PRESS
Str. Observatorului nr. 34
400775 Cluj-Napoca
Tel.:0264-401.999
e-mail: utpress@biblio.utcluj.ro
https://biblioteca.utcluj.ro/editura

Recenzia: Prof.dr.ing. Radu Gabriel Dănescu
Prof.dr.ing. Florin Ioan Oniga

Pregătire format electronic on-line: Gabriela Groza

ISBN 978-606-737-846-7

Prefață

Îndrumătorul de laborator se adreseaza studenților Facultății de Automatică și Calculatoare, disciplina fiind studiată în cadrul în ciclului de licență. De asemenea, îndrumătorul poate fi util oricui dorește să studieze tehnicile fundamentale de proiectare logică, care stau la baza implementării sistemelor de calcul digitale. Conținutul lucrării de față reprezintă materialul de bază pentru pregătirea lucrărilor practice din cadrul disciplinei și un important mijloc de verificare a cunoștințelor însușite, prin examinarea unei game variate de probleme specifice rezolvate.

Pe parcursul lucrării s-a urmărit o organizare didactică, axată pe capitole cu grad de dificultate progresiv. Se recomandă parcurgerea capitolelor în ordinea de apariție în cadrul îndrumătorului. Obiectivele urmărite sunt detaliate folosind un formalism teoretic, cu rol de fundamentare a cunoștințelor, dar avându-se în vedere o finalizare bazată pe implementare practică și testare a circuitelor în regim de funționare. Deoarece subiectul acoperă un domeniu vast s-a vizat în mod deosebit aprofundarea cunoștințelor legate de funcționarea componentelor esențiale din cadrul microarhitecturilor de calcul, în perspectiva abordării ulterioare a unor subiecte mai ample, precum implementarea automatelor și a microarhitecturilor de procesoare.

În ceea ce privește componentele studiate sunt prezentate atât detaliile de funcționare individuale, cât și modalități de integrare a acestora în proiectarea de circuite cu rol mai complex, redând astfel o perspectivă mai amplă, care combină întelegerea elementelor hardware cu formarea de abilități creative, în scopul obținerii unor capacități de calcul mai avansate. Pentru facilitarea părții practice, de implementare și testare, se utilizează un proiect cadru dezvoltat pentru aplicația ISE Project Navigator și o librărie de simulare dezvoltată pentru aplicația Logisim. Împreună, întregesc experiența cititorului atât la nivel practic, pe plăci FPGA, cât și la nivel de simulator.

În prezentarea lucrărilor s-a urmărit o succesiune constructivă a informațiilor prezentate. Fiecare capitol prezintă obiectivele, continuate de conceptele teoretice necesare aspectelor practice și implementării. Activitățile practice propuse la final urmăresc dezvoltarea abilităților de lucru la nivel hardware prin sesiuni concrete de implementare și testare. Atunci când sunt expuse mai multe variante posibile, se pot realiza analize comparative, similare cu activitățile de cercetare. Pentru a facilita însușirea noțiunilor prezentate, se recomandă parcurgerea fiecarui capitol înainte de activitatea de laborator. În cadrul anexelor sunt redate rezolvări ale problemelor de implementare hardware cu ajutorul circuitelor studiate pe parcursul lucrărilor.

Autorul vă urează lectură plăcută!

CUPRINS

1 Prezentarea metodologiei de lucru cu plăcile didactice

1.1 Obiective

Sunt enumerate resursele necesare pentru activitatea de laborator. Se prezintă noțiunile de bază în modelarea circuitelor numerice. Se studiază metodologia de lucru cu plăcile de dezvoltare precum și familiarizarea cu mediul de dezvoltare Xilinx ISE pentru programarea circuitelor logice programabile de tip FPGA (Field Programmable Gate Array). Se prezintă proiectul de lucru care conține biblioteca de componente TTL (Transistor-Transistor Logic) studiate și se implementează circuite simple folosind porți logice fundamentale. Se testează funcționarea acestora cu ajutorul tabelelor de adevăr corespunzătoare.

1.2 Resurse necesare

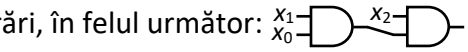
Pentru proiectarea și testarea circuitelor studiate sunt utilizate următoarele elemente:

- Placa de dezvoltare Nexys A7, Manualul de utilizare [1];
- Mediul de dezvoltare Xilinx ISE WebPACK 14.7 [2]; se va lucra în cadrul unui proiect existent, care pune la dispoziție o librărie cu circuitele TTL uzuale.

1.3 Elemente de bază în modelarea circuitelor numerice

Modelarea comportamentului circuitelor numerice se realizează folosind un formalism matematic care are la bază conceptele din *algebra booleană* (după numele matematicianului George Boole). Aceasta conține un set de operații logice și proprietăți ale acestora cu ajutorul cărora se poate descrie funcționalitatea circuitelor utilizate în proiectarea logică. Un astfel de circuit are unul sau mai multe intrări și ieșiri, denumite *terminale* (sau pini). Fiecare ieșire poate fi modelată ca o funcție de mai multe variabile. Variabilele sunt intrările circuitului. Deoarece în algebra booleană funcțiile și variabilele pot să aibă doar valorile 1 (adevărat) și 0 (fals) acestea se mai numesc și *funcții booleene*. Din punct de vedere electronic valorile sunt asociate cu proprietăți electrice precum tensiunea. De exemplu, pentru circuitele realizate cu tranzistoare în tehnologie Transistor-Transistor Logic (TTL) [3], valoarea 0 este asociată cu o tensiune mică și valoarea 1, cu o tensiune ridicată. Spunem că reprezentarea este în *logica pozitivă*. Atunci când se schimbă polaritatea (0 este asociat cu tensiuni ridicate și 1 este asociat cu tensiuni scăzute) reprezentarea este în *logica negativă*.

O funcție booleană se poate descrie folosind *tabelul de adevăr*. În partea stângă tabelul conține combinații de valori posibile pentru variabile, iar în dreapta sunt trecute valorile funcției pentru combinațiile respective. Un exemplu de tabel de adevăr pentru o funcție de 2 variabile x_1, x_0 este prezentat în Tabelul 1. 1, partea stângă. Conform acestuia, circuitul asociat va avea ieșirea 1, când intrarea x_1=1 și x_0=1, iar în rest 0. Din acest motiv mai poartă denumirea de funcția ȘI. Datorită simplității sale, un astfel de circuit se mai

numește *poartă logică fundamentală*. În algebra booleană operația ȘI se notează cu · (a nu se confunda cu operația de înmulțire), iar simbolul grafic, folosit în cadrul unui circuit, este  , cu intrările la stânga și ieșirea la dreapta. Există variante de porți ȘI cu număr extins de intrări, care realizează operația $x_{n-1}\cdot\ldots\cdot x_0$. În acest caz, rezultatul operației este 1, numai când toate intrările sunt 1. Fiind o operație asociativă, extinderea la mai multe intrări se poate realiza și conectând mai multe porți cu număr redus de intrări. De exemplu, o funcție ȘI cu 3 intrări se poate implementa conform relației $x_2\cdot x_1\cdot x_0 = x_2\cdot(x_1\cdot x_0)$, folosind porți ȘI cu 2 intrări, în felul următor: 

.

Tabelul 1. 1 Tabelele de adevăr ale porților ȘI (stânga), SAU (mijloc), NOT (dreapta)

x_1	x_0	ȘI
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_0	SAU
0	0	0
0	1	1
1	0	1
1	1	1

x	NOT
0	1
1	0

Există și alte porți fundamentale, precum poarta SAU și poarta NOT (sau INV). Operatorul pentru operația SAU este simbolul + (atenție!, nu reprezintă adunare). Expresia x_1+x_0 va avea rezultatul 1, dacă x_1=1 sau x_0=1 (vezi Tabelul 1. 1, mijloc). Simbolul porții SAU în circuite este  . O poartă logică NOT = $\bar{x}$ are o singură intrare și inversează valoarea acesteia la ieșire. Simbolul porții NOT este  . Cele 3 porți realizează operațiile de bază din algebra booleană. Prin interconectări ale acestora se pot realiza circuite corespunzătoare unui număr nelimitat de funcții booleene, cu oricâte variabile. De exemplu, expresia $x_2 + (\bar{x_1}\cdot x_0)$ se poate implementa cu circuitul: 

.

Într-o expresie ordinea priorităților este: 1) parantezele; 2) NOT; 3) ȘI; 4) SAU. Operațiile se realizează de la intrare spre ieșire, conform schemei circuitului, folosind tabelele de adevăr. Dacă x_2=0, x_1=0 și x_0=1, înlocuind în expresie, se obține $x_2 + (\bar{x_1}\cdot x_0) = 0 + (\bar{0}\cdot 1) = 0 + (1\cdot 1) = 0 + 1 = \mathbf{1}$. Similar, se poate calcula rezultatul expresiei pentru orice combinație de valori pe intrări și se poate determina tabelul de adevăr. În continuare, sunt prezentate alte combinații posibile și rezultatul obținut:

- x_2=1, x_1=0, x_0=0 → $x_2 + (\bar{x_1}\cdot x_0) = 1 + (\bar{0}\cdot 0) = 1 + (1\cdot 0) = 1 + 0 = \mathbf{1}$;
- x_2=0, x_1=1, x_0=1 → $x_2 + (\bar{x_1}\cdot x_0) = 0 + (\bar{1}\cdot 1) = 0 + (0\cdot 1) = 0 + 0 = \mathbf{0}$;
- x_2=0, x_1=0, x_0=0 → $x_2 + (\bar{x_1}\cdot x_0) = 0 + (\bar{0}\cdot 0) = 0 + (1\cdot 0) = 0 + 0 = \mathbf{0}$.

1.4 Ghid de lucru cu Xilinx ISE Project Navigator

Utilitarul Project Navigator face parte din pachetul Xilinx ISE și oferă suport pentru proiectarea circuitelor într-un editor schematic.

1.4.1 Mediul de lucru Project Navigator

După pornirea utilitarului Project Navigator se încarcă proiectul **ttl_env** [4] cu comanda File > Open Project… și se selectează fișierul ttl_env.xise. În cadrul utilitarului se pot distinge panourile de lucru (panels), ca în Figura 1. 1. O parte din acestea sunt folosite

pentru organizare (sunt amplasate în lateral), iar o parte afișează informații utile în procesul de prelucrare a unei scheme (sunt amplasate în partea inferioară) [**Notă**: Dacă interfața grafică este diferită se poate reseta cu comanda Layout > Load Default Layout.]:

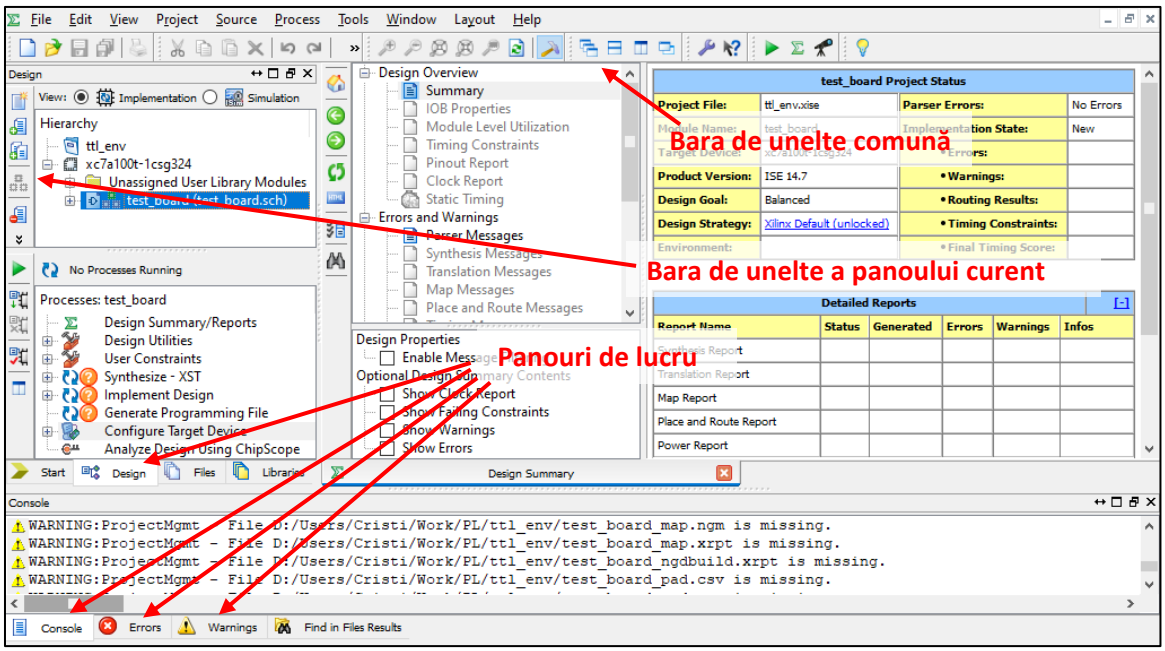


Figura 1. 1 Interfața utilitarului Project Navigator

- Panoul **Design** (Figura 1. 1) – conține structura proiectului organizată pe circuite; inițial, proiectul conține circuitul de test test_board. Dacă se apasă dublu-click pe numele circuitului se va afișa editorul schematic, care conține **schema** circuitului și se face automat saltul la panoul Symbols.
- Panoul **Symbols** (Figura 1. 2) – pune la dispoziție lista de componente, care se pot introduce în schema unui circuit, apăsând butonul Add Symbol în bara de unelte a editorului schematic. În cadrul panoului componentele sunt organizate pe categorii. Prin selectarea unei categorii în partea superioară se pot vizualiza dedesubt circuitele corespunzătoare. Dacă se selectează <--All Symbols--> va apărea lista tuturor componentelor, din toate categoriile. O componentă poate fi căutată, în cadrul categoriei curente, după numele său, prin introducerea acestuia în caseta Symbol Name Filter. Printre categoriile cele mai importante se pot enumera: *Arithmetic*, *Buffer*, *General*, *Logic*, *Mux*, *Decoder*. Întotdeauna, opțiunea <--All Symbols--> este secondată de librăria locală a proiectului, care conține toate circuitele TTL studiate. **Notă**: Circuitele TTL au numele simbolului format din prefixul **TTL_** urmat de codul circuitului.
- Panourile **Console**, **Errors**, **Warnings** (Figura 1. 1) – afișează informații de procesare a schemei curente. În **Console** apar toate informațiile, iar în **Errors** și **Warnings** doar cele legate de eventuale erori sau avertismente. De exemplu, pentru intrările neconectate, se emite avertisment de conectare implicită la 0 (masă sau Ground – GND). Ieșirile neconectate se vor elimina. În schimb, erorile detectate trebuie corectate.

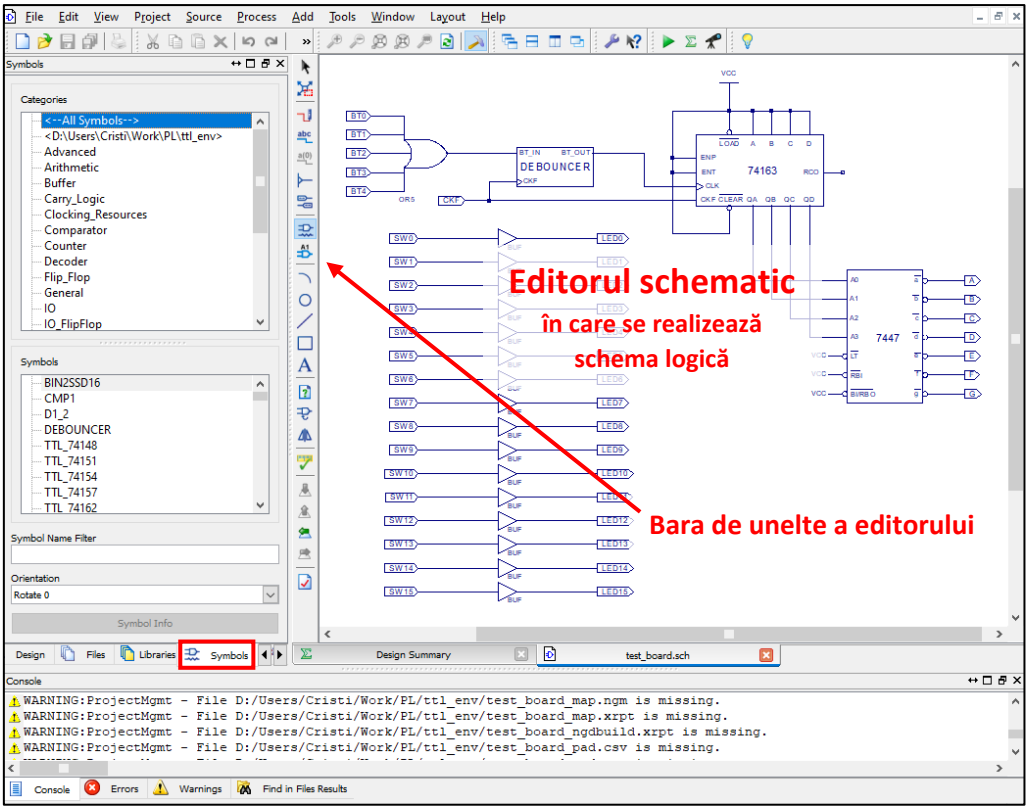


Figura 1. 2 Panoul Symbols și Editorul schematic

1.4.2 Crearea unei noi scheme

Pentru adăugarea unei noi scheme în cadrul proiectului, se revine în bara de unelte a panoului Design și se apasă butonul New Source . În fereastra următoare se selectează tipul Schematic și se introduce numele schemei (fără caractere speciale sau spații): de exemplu **lab01_01** (Figura 1. 3). **Notă**: verificați că opțiunea Add to project este bifată. Apoi se apasă Next și Finish. Pe ecran va apărea editorul schematic în care se poate defini schema circuitului prin adăugare de componente și interconectarea acestora. Aceasta se poate mări sau micșora apăsând pe Zoom In sau Zoom Out în bara de unelte comună amplasată în partea superioară.

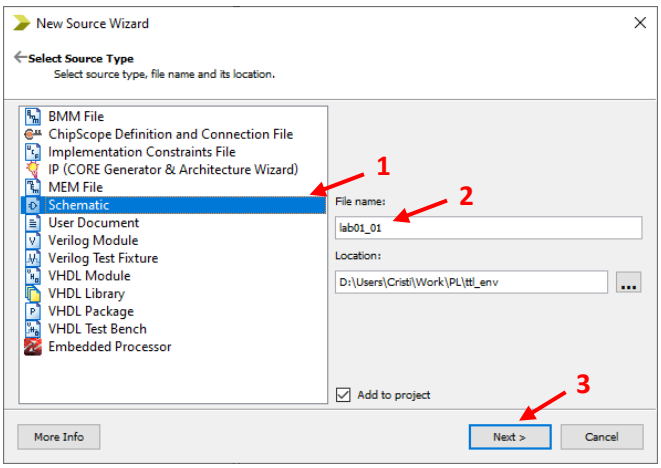


Figura 1. 3 Introducerea datelor necesare creării unui nou circuit

Cum proiectul poate conține mai multe scheme doar una poate să fie cea activă. Setarea schemei **lab01_01**, ca fiind cea activă, se realizează cu click-dreapta pe numele ei în panoul Design și alegerea opțiunii Set as Top Module. În dreptul numelui schemei active apare întotdeauna simbolul . Schema activă se poate schimba în orice moment. O schemă poate fi eliminată cu click-dreapta pe numele ei și opțiunea Remove.

În cadrul schemei se va introduce, pentru început, o poartă ȘI (engl. AND) cu 2 intrări. La panoul Symbols, în căsuța de căutare Symbols Name Filter, se introduce cuvântul *and2* (sufixul 2 reprezintă numărul de intrări), conform cu Figura 1. 4. Pentru a căuta în lista cu toate circuitele disponibile ne asigurăm că opțiunea <--All symbols--> este selectată în partea superioară. Se selectează simbolul **and2** din lista de simboluri afișate și se plasează pe schemă prin click. Prin apăsări repetate se poate introduce același simbol, în mod repetat. (**Notă**: Orientarea unui simbol poate fi modificată înainte de amplasare de la rubrica Orientation.)

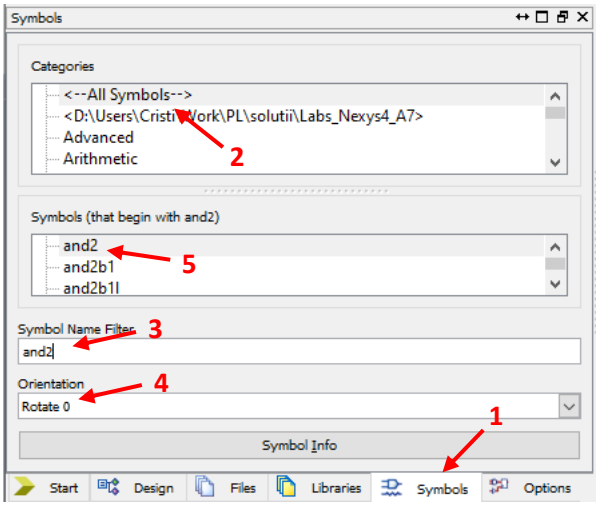


Figura 1. 4 Pașii pentru selectarea simbolului **and2** din lista de componente

Pentru selectarea oricărui element de pe schemă, trebuie să se intre în modul de selectare prin apăsarea butonului Select . Un element selectat poate fi deplasat cu mouse-ul sau poate fi șters, cu click-dreapta și Delete sau apăsând tasta *Del*.

În continuare, pe intrările și ieșirile componentei AND2 se vor trasa fire de conexiune. Prin apăsarea butonului Add Wire se trece în modul de inserare a firelor. Ulterior, firele se pot trasa apăsând mouse-ul între cele 2 puncte conectate. De remarcat faptul că apare un simbol grafic specific atunci când mouse-ul este deplasat deasupra punctelor de conexiune și prin simpla apăsare firul se va conecta automat la ele. Capetele rămase neconectate sunt marcate cu un dreptunghi roșu, ca în Figura 1. 5.

Figura 1. 5 Fire neconectate atașate la pinii componentei AND2

La capetele neconectate ale firelor se vor adăuga terminale de intrare și ieșire. Terminalele de intrare sunt utilizate pentru a introduce valori în circuit (0 sau 1), iar cele

de ieșire sunt necesare pentru vizualizarea rezultatelor în afara circuitului. Modul de inserare terminale se activează prin apăsarea butonului Add I/O Marker  . Ulterior, se pot introduce terminale apăsând pe capetele neconectate ale firelor. (**Notă**: Terminalele se pot adăuga și în mod direct la pinii simbolului, fără a mai trasa fire.) Tipul de terminal potrivit (intrare sau ieșire) este determinat automat. La inserare, terminalele primesc un nume generat automat. Acesta se poate schimba prin click-dreapta pe terminal și opțiunea Rename Port. Numele nu trebuie să conțină caractere speciale sau spațiu. Cele 2 terminale de intrare se vor redenumi la A, respectiv B, iar cel de ieșire la F1 (Figura 1. 6). **Notă**: Firele conectate la terminale primesc automat același nume. **Important**: Firele cu nume identic sunt considerate conectate (se numește *conexiune logică*).

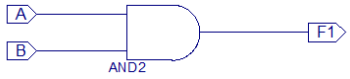


Figura 1. 6 Poarta AND2 conectată la terminale de intrare și ieșire

Se va adăuga în schema curentă o nouă funcție de ieșire F2, care realizează un ȘI extins la 3 intrări. Pentru aceasta se va introduce o nouă componentă AND care primește pe o intrare rezultatul porții AND existente și pe cealaltă un al 3-lea terminal de intrare C. Ieșirea acestei porți AND va fi legată la terminalul de ieșire F2. Pașii necesari sunt următorii:

1. Se introduce în schemă o nouă poartă AND2 din panoul Symbols.
2. Se trasează o conexiune dinspre ieșirea primului AND2 la una din intrările celui de-al doilea AND2. În consecință, va apărea o ramificație pe acest fir marcată printr-un dreptunghi. **Notă**: Toate ramificațiile sunt considerate unul și același fir.
3. Se trasează fire pe pinii neconectați ai noului AND2.
4. Se introduce un terminal C pe intrarea rămasă liberă a lui AND2 și un terminal F2 pe ieșire. Schema rezultată este prezentată în Figura 1. 7.
5. Se salvează schema în forma finală cu combinația de taste *Ctrl+S* sau cu comanda File > Save.

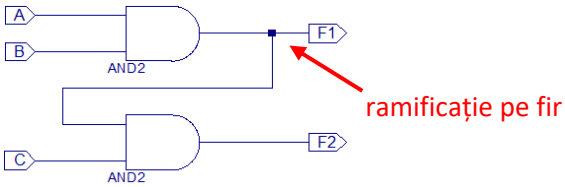


Figura 1. 7 Circuit care implementează funcțiile F1=A · B și F2=(A · B) · C

Ulterior, se va conecta terminalul de intrare C la o altă ieșire F3. **Notă**: Este permisă conectarea directă a unui terminal de intrare la un terminal de ieșire numai printr-un element buffer, denumit BUF, care se găsește în lista de simboluri la categoria *General*. După adăugarea acestei conexiuni prin intermediul unui BUF, circuitul devine cel din Figura 1. 8, în care se observă apariția unei noi ramificații de la intrarea C la buffer-ul BUF.

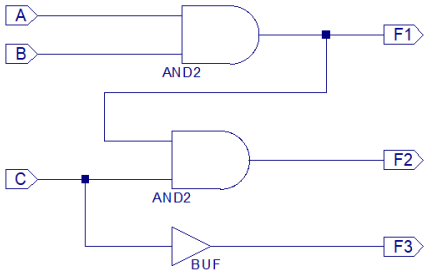


Figura 1. 8 Circuitul cu funcțiile F1=A · B, F2=(A · B) · C și F3=C

1.4.3 Definirea constrângerilor de implementare

Testarea circuitelor definite prin schema logică se realizează pe placa de dezvoltare din Figura 1. 9, care este dotată cu **16 comutatoare** și **5 butoane** pentru generarea valorilor pe terminalele de intrare. Vizualizarea rezultatelor înregistrate pe terminalele de ieșire se poate realiza pe cele **16 led-uri**.

La apăsarea unui **buton** acesta generează valoarea logică 1, altfel generează valoarea 0. Un **comutator** poziționat către led-uri generează valoarea 1 și valoarea 0 în poziția opusă. Un **led** care primește 0 este stins, iar dacă primește 1 luminează. Comutatoarele și led-urile sunt numerotate de la 0 la 15, de la dreapta spre stânga.

Asocierea terminalelor la butoane, comutatoare, și led-uri se realizează prin definirea unui fișier de constrângeri (nu este case sensitive) cu extensia .ucf (User Constraints File). Fiecare schemă trebuie să aibă asociat un astfel de fișier. Pentru simplificare, în directorul proiectului există un șablon predefinit de fișier de constrângeri denumit *_template.ucf*, care conține toate elementele plăcii, în comentarii (rânduri prefixate cu simbolul #). Se vor de-comenta doar rândurile asociate cu resursele necesare.

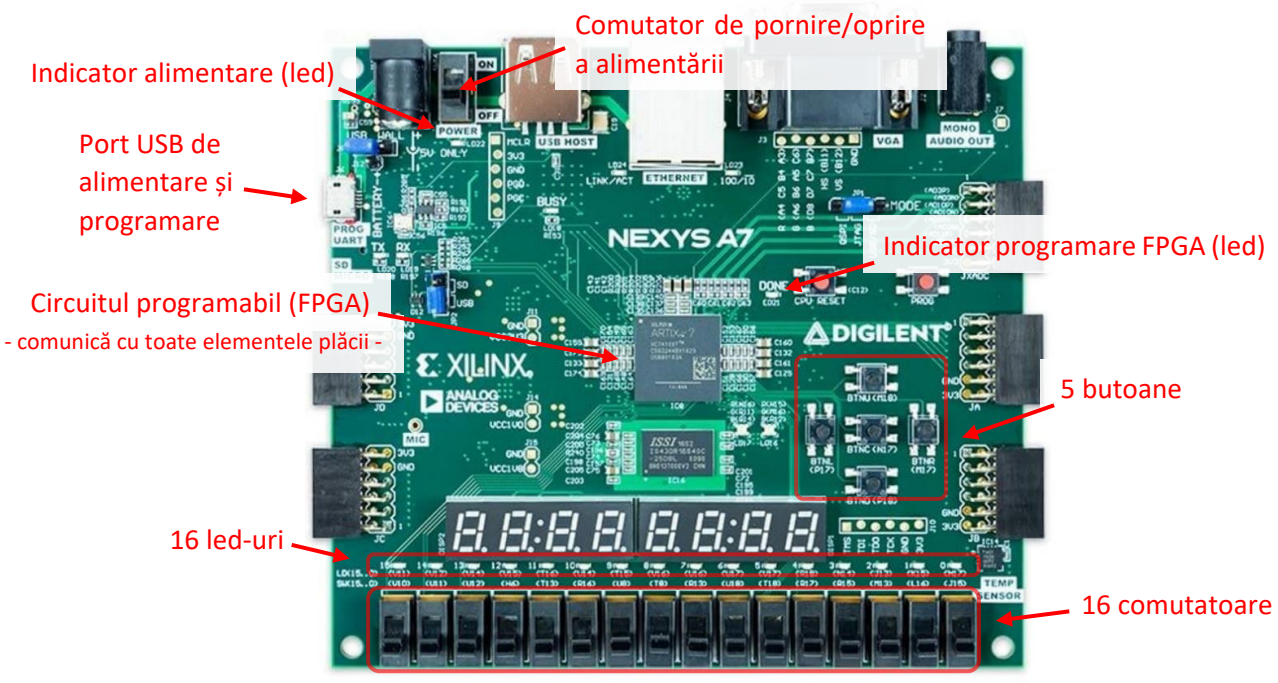
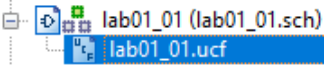


Figura 1. 9 Placa de dezvoltare

Înainte de asocierea unui fișier de constrângeri la o schemă este indicată crearea unei copii a fișierului *_template.ucf* cu numele schemei și extensia ucf. În cazul de față copia se va redenumi la *lab01_01.ucf*. Pentru ca asocierea să fie făcută la schema curentă aceasta trebuie sa fie desemnată top-module, lucru indicat de simbolul în dreptul ei, în panoul Design. În bara de unelte a panoului Design se va apăsa pe butonul Add Source și se va selecta fișierul *lab01_01.ucf* de pe disc. Ulterior încărcării acesta va apărea sub schema lab01_01 în structura proiectului 

și se va deschide pentru editare cu dublu-click pe numele său.

Se poate observa că în cadrul fișierului toate terminalele au nume predefinite între ghilimele, după cuvântul cheie NET. Se vor de-comenta (prin ștergerea simbolului #) numai rândurile asociate resurselor necesare de pe placă și numele terminalelor se va actualiza în conformitate cu cele de pe schemă. Presupunând că terminalele A, B, C vor fi asociate la comutatoarele 0, 1, respectiv 2, vor trebui de-comentate rândurile 13-15 și modificate astfel (vezi marcajul roșu):

```
## Switches
NET "A" LOC=J15 | IOSTANDARD=LVCMOS33 | CLOCK_DEDICATED_ROUTE=FALSE; # sw0
NET "B" LOC=L16 | IOSTANDARD=LVCMOS33 | CLOCK_DEDICATED_ROUTE=FALSE; # sw1
NET "C" LOC=M13 | IOSTANDARD=LVCMOS33 |CLOCK_DEDICATED_ROUTE=FALSE; # sw2
```

Notă: Valoarea asociată atributului LOC (de la locație) coincide cu codul înscris pe placă în dreptul comutatoarelor. Acest lucru este valabil și pentru butoane și led-uri.

Presupunând că ieșirile F1, F2 și F3 vor fi conectate la led-urile 0, 1 și 2 se vor de-comenta și ajusta rândurile 40-42 în felul următor:

```
## LEDs
NET "F1" LOC=H17 | IOSTANDARD=LVCMOS33; # led0
NET "F2" LOC=K15 | IOSTANDARD=LVCMOS33; # led1
NET "F3" LOC=J13 | IOSTANDARD=LVCMOS33; # led2
```

Se salvează fișierul de constrângeri cu *Ctrl+S* sau cu comanda File > Save.

1.4.4 Generarea fișierului de programare

Fișierul de programare se poate genera de la rubrica **Generate Programming File**, situată în partea inferioară a panoului Design (Figura 1. 10), cu click-dreapta pe ea și una din comenzile Run, ReRun sau Rerun All. Pe măsură ce procesarea avansează pot sa apară diverse erori sau avertismente. Dacă apar erori, în funcție de gravitatea acestora, întreg procesul se poate opri automat. Problemele sesizate trebuie eliminate și se va relua procesul de generare cu comanda Rerun All.

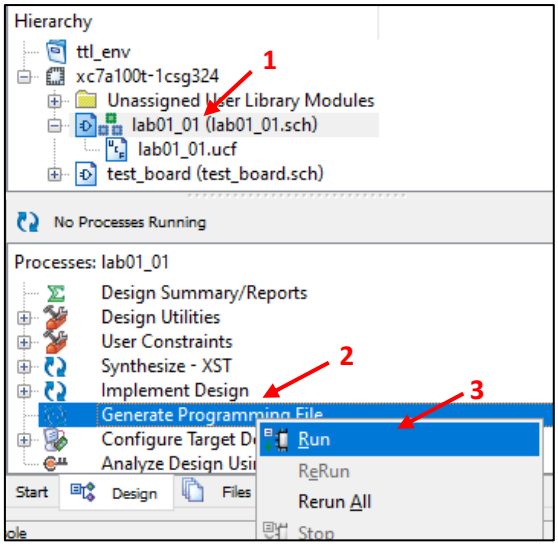


Figura 1. 10 Generarea fișierului de programare

Dacă generarea se încheie cu succes se va crea fișierul lab01_01.bit în directorul proiectului și va apărea iconița verde: Generate Programming File.

1.4.5 Programarea circuitului FPGA și testarea pe placa de dezvoltare

Important: Înainte de programarea circuitului, placa (Figura 1. 9) se conectează la stația de lucru folosind un cablu USB și se pornește alimentarea. Se va activa indicatorul led aflat sub comutatorul de alimentare.

Pentru programare, se lansează în execuție utilitarul ISE iMPACT cu click-dreapta pe pasul **Configure Target Device** și comanda Run (Figura 1. 11).

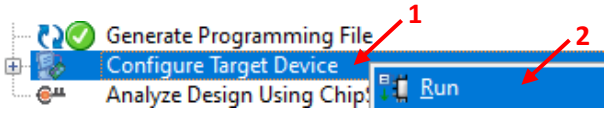


Figura 1. 11 Lansarea utilitarului ISE iMPACT

În cadrul utilitarului se deschide o sesiune de lucru cu dublu-click pe Boundary Scan (Figura 1. 12).

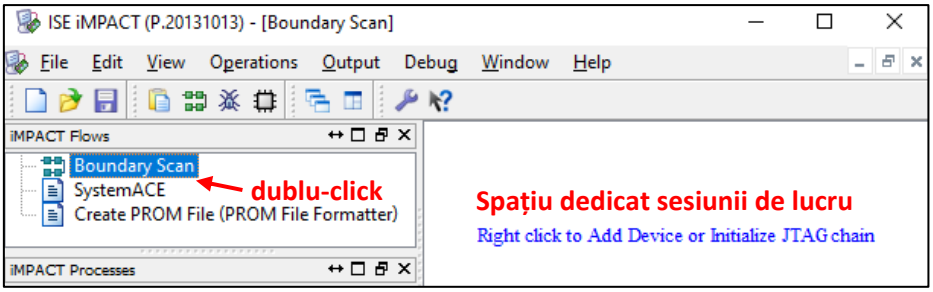
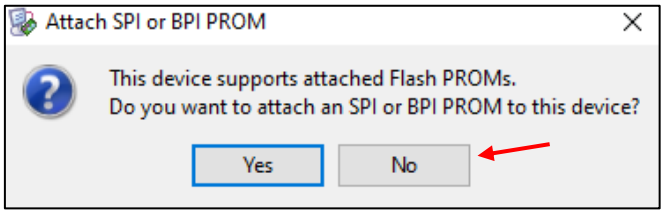


Figura 1. 12 Crearea unei sesiuni de lucru în ISE iMPACT

Pașii de realizare a unei conexiuni cu placa de dezvoltare:

1. Click-dreapta în spațiul dedicat sesiunii și se alege comanda Initialize Chain.

2. Se confirmă asocierea unui fișier de configurare și se selectează lab01_01.bit din directorul proiectului.
3. **Important**: Se va refuza atașarea unui modul SPI sau BPI PROM ori de câte ori apare această fereastră.



4. Se apasă OK în fereastra următoare de afișare a proprietăților de programare.

O conexiune funcțională va afișa în spațiul de lucru simbolul circuitului FPGA de pe placă, modelul acestuia și numele fișierului .bit de configurare curent, ca în Figura 1. 13.

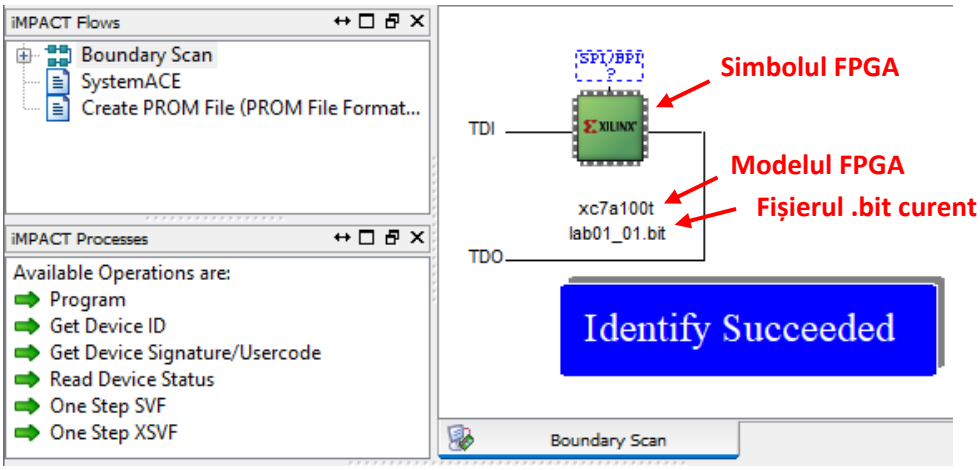


Figura 1. 13 ISE iMPACT recunoaște placa de dezvoltare

Urmează pasul final, de programare a circuitului FPGA, cu click-dreapta pe simbolul circuitului și se lansează comanda Program (Figura 1. 14). Încheierea acestei ultime etape este marcată de activarea pe placă a indicatorului led pentru programare. Din acest moment placa funcționează conform circuitului de pe schemă. Cu ajutorul comutatoarelor i se pot transmite valori de 0 sau 1 pe intrări, iar pe led-urile alocate se vor vizualiza rezultatele: F1=1, dacă A=B=1, F2=1, dacă A=B=C=1 și F3=C.

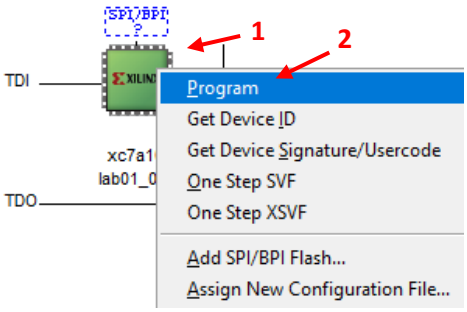


Figura 1. 14 Lansarea pasului de programare după atașarea unui fișier .bit

Configurarea cu circuitul curent se menține cât timp placa este alimentată. Sesiunea ISE iMPACT poate fi menținută pornită chiar dacă se aduc modificări la schemă.

Pentru a economisi timp, nu se recomandă închiderea ISE iMPACT după fiecare testare. În cazul unor modificări cu utilitarul Project Navigator, se va regenera fișierul de programare, se va reveni în ISE iMPACT și se va relua programarea cu pasul final, comanda Program, fără a parcurge pașii anteriori pentru realizarea conexiunii.

În cazul în care se dorește programarea cu un fișier .bit, având nume diferit de cel curent, se poate folosi comanda Assign New Configuration File… (Figura 1. 14), care va solicita alegerea de pe disc a noului fișier. Ulterior, se poate realiza programarea cu comanda Program. Nici în acest caz nu este necesară refacerea conexiunii.

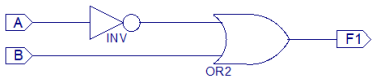
Notă: La închiderea utilitarului ISE iMPACT (la final de laborator) nu este recomandată salvarea proiectului din cadrul acestuia, dacă se solicită acest lucru.

Observație: Proiectul *ttl_env* poate fi curățat de fișierele generate pe parcurs, prin executarea scriptului clean_win.bat în Windows sau clean_lnx.sh în Linux, ambele aflate în folder-ul proiectului. Scripturile păstrează schemele .sch și fișierele de programare .ucf.

1.5 Activități practice

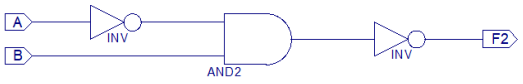
1. Adăugați în cadrul proiectului *ttl_env* următoarele circuite (în scheme diferite) și testați funcționalitatea acestora pe placă, folosind tabelele de adevăr alăturate. Denumirea schemelor și alegerea comutatoarelor și a led-urilor utilizate este la alegere.

a) $F1 = \overline{A} + B$



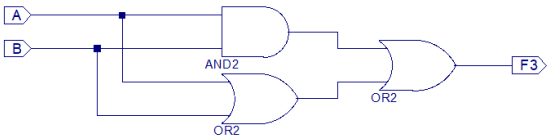
A	B	F1
0	0	1
0	1	1
1	0	0
1	1	1

b) $F2 = \overline{\overline{A} \cdot B}$



A	B	F2
0	0	1
0	1	0
1	0	1
1	1	1

c) $F3 = (A \cdot B) + (A + B)$



A	B	F3
0	0	0
0	1	1
1	0	1
1	1	1

1.6 Bibliografie

[1] Digilent, “Nexys A7 Reference Manual”, [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual

[2] AMD Xilinx, “ISE WebPACK Design Software”, [Online]. Available: https://wiki.archlinux.org/title/Xilinx_ISE_WebPACK

[3] Wikipedia, “Transistor–transistor logic”, [Online]. Available: https://en.wikipedia.org/wiki/Transistor-transistor_logic

[4] Cristian-Cosmin Vancea, “TTL_Env Project for Project Navigator”, [Online]. Available: https://drive.google.com/uc?export=download&id=1800AdR6Vdo8PDd1tB9n5LCQZNHAcgh9K

2 Porți logice fundamentale

2.1 Obiective

Sunt prezentate noțiuni de bază și proprietăți de implementare tehnologică a porților logice fundamentale. Sunt descrise porțile logice fundamentale și asocierea acestora cu operațiile de bază din algebra booleană. Sunt introduse prioritățile operatorilor și se studiază elaborarea tabelului de adevăr și a schemei logice pornind de la expresia booleană. Sunt analizate diverse transformări și proprietăți ale algebrei booleene, care permit utilizarea porților logice fundamentale pentru a genera noi funcționalități.

2.2 Considerații de implementare tehnologică

Porțile logice fundamentale sunt cele mai elementare circuite utilizate în implementarea dispozitivelor numerice. Ele sunt caracterizate de o funcționalitate simplă și au în componența lor un număr redus de tranzistoare sau semiconductori. Raportat la modelarea în algebra booleană acestea implementează operațiile de bază: Inversor (NOT sau INV), SAU (OR), ȘI (AND). Datorită complexității reduse, tot în această categorie se încadrează și circuite care s-ar putea implementa din operațiile de bază precum: SAU-NU (NOR), ȘI-NU (NAND), SAU-EXCLUSIV (XOR), COINCIDENȚĂ (XNOR).

Din punct de vedere al implementării la nivel electronic porțile logice fundamentale pot fi realizate în tehnologie TTL (Transistor-Transistor Logic) sau CMOS (Complementary Metal-Oxide Semiconductor). Există diferențe în ceea ce privește caracteristicile electrice ale celor două tehnologii, dintre care, printre cele mai cunoscute se pot enumera toleranța la variații de tensiune, viteza de comutare și puterea consumată.

- *toleranța la variații de tensiune* – conform valorilor din Figura 2. 1 se observă faptul că circuitele CMOS au un interval de tensiune mai extins asociat stării 0, ceea ce le conferă o toleranță mai bună la oscilații în jurul valorilor minime, probabilitatea ieșirii în mod necontrolat din această stare fiind mai scăzută, în comparație cu circuitele TTL.

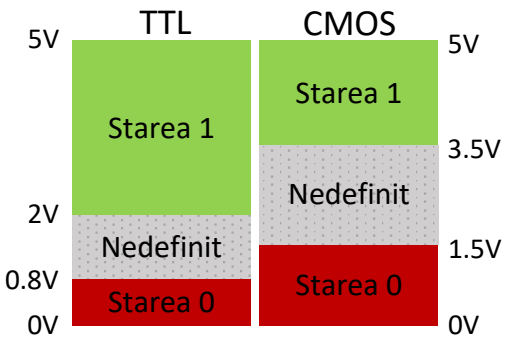


Figura 2. 1 Asocierea valorilor de tensiune pe intrări la nivelurile logice

- *viteza de comutare* – este strâns legată de timpul de reacție a circuitului, atunci când trebuie să-și schimbe starea pe ieșiri în urma unor modificări ale valorilor de intrare. Din acest punct de vedere circuitele CMOS sunt mai lente.
- *puterea consumată* – circuitele TTL consumă mai multă putere, motiv pentru care la echipamentele digitale mobile, cu alimentare pe bază de baterie sau acumulatori, este preferată implementarea cu circuite CMOS.

În continuare, vor fi studiate circuitele logice în tehnologie TTL. În catalogul de circuite ele sunt identificate prin coduri alcătuite din mai multe cifre, primele două fiind 74 sau 54. Conform proprietăților amintite anterior circuitele TTL se împart în mai multe subcategorii:

- standard;
- cu consum redus (low-power);
- rapide (high-speed);
- Schottky (cele mai rapide);
- combinații (ex. Low-Power Schottky).

2.3 Porțile logice fundamentale

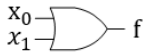
La nivelul algebrei booleene porțile logice fundamentale implementează operații elementare de una sau două variabile de intrare, dar pot fi extinse la mai multe intrări. Acestea sunt enumerate în Tabelul 2. 1 alături de expresiile matematice, simbolurile grafice folosite în schema logică și tabelele de adevăr asociate.

Tabelul 2. 1 Porțile logice fundamentale

Inversor (NOT) $f = \overline{x_0}$

x_0	f
0	1
1	0

SAU (OR) $f = x_1 + x_0$



x_1	x_0	f
0	0	0
0	1	1
1	0	1
1	1	1

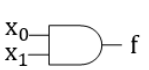
SAU-NU (NOR) $f = \overline{x_1 + x_0}$

x_1	x_0	f
0	0	1
0	1	0
1	0	0
1	1	0

SAU-EXCLUSIV (XOR) $f = x_1 \oplus x_0$

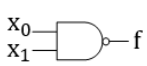
x_1	x_0	f
0	0	0
0	1	1
1	0	1
1	1	0

ȘI (AND) $f = x_1 \cdot x_0$



x_1	x_0	f
0	0	0
0	1	0
1	0	0
1	1	1

ȘI-NU (NAND) $f = \overline{x_1 \cdot x_0}$



x_1	x_0	f
0	0	1
0	1	1
1	0	1
1	1	0

COINCIDENȚĂ (XNOR) $f = \overline{x_1 \oplus x_0} = x_1 \odot x_0$

x_1	x_0	f
0	0	1
0	1	0
1	0	0
1	1	1

În ceea ce privește simbolurile folosite pentru reprezentarea grafică se observă faptul că operația de inversare este reprezentată de un cerculeț, iar în reprezentarea matematică, de o bară deasupra expresiei booleene. De asemenea, să nu se confunde operatorul + din algebra booleană, cu operația de adunare, respectiv operatorul ·, cu operația de înmulțire.

Conform tabelelor de adevăr asociate, efectul porților logice este următorul:

- operația NOT inversează valoarea de intrare;
- rezultatul unei operații SAU este 1 dacă cel puțin o intrare este 1 – echivalent cu valoarea maximă pe intrări;
- rezultatul unei operații ȘI este 1, dacă toate intrările sunt 1 – echivalent cu valoarea minimă pe intrări;
- operația SAU-NU este inversul operației SAU;
- operația ȘI-NU este inversul operației ȘI;
- rezultatul XOR este 1, dacă cele două intrări au valori diferite;
- rezultatul XNOR este 1, dacă cele două intrări au valori identice;
- operația XNOR este inversul operației XOR.

În funcție de operația implementată porțile logice pot fi identificate după codurile din tabelul următor:

Tabelul 2. 2 Codurile asociate porților logice fundamentale TTL

Poarta	Cod
NAND	7400
NOR	7402
NOT	7404
AND	7408
OR	7432
XOR	7486
XNOR	74266

Notă: În utilitarul Project Navigator porțile logice fundamentale sunt grupate în categoria *Logic* și se pot regăsi după denumirile lor (AND, OR, INV etc.). INV înlocuiește NOT.

Observație: Operația compusă SAU-NU se poate realiza și legând ieșirea porții SAU la intrarea porții NOT. O astfel de soluție presupune folosirea a două porți logice fundamentale, adică un număr dublu de resurse, o viteză de lucru mai scăzută din cauză că semnalele trebuie să se propage prin 2 porți și un consum de curent suplimentar. Din aceste motive se preferă implementarea operațiilor compuse de bază cu porți logice dedicate, care au comportamentul descris în tabelul de adevăr asociat. Același lucru este valabil și în cazul operațiilor ȘI-NU, XOR și XNOR.

2.4 Determinarea tabelului de adevăr al unei expresii booleene

La calculul rezultatului unei expresii se realizează operațiile, în ordinea priorităților (conform cu Tabelul 2. 3), pentru toate combinațiile de valori ale variabilelor. Dacă expresia are *n* variabile sunt 2^n combinații și tot atâtea valori de calculat pentru ieșire.

Considerând expresia $\overline{a \cdot b} + c$ se va realiza întâi operația ȘI-NU între variabilele a și b, urmată de operația SAU a rezultatului anterior cu variabila c. În calcule se vor utiliza tabelele de adevăr ale fiecărei operații în parte. Conform acestei metodologii valorile intermediare și finale sunt prezentate în Tabelul 2. 4.

Tabelul 2. 3 Prioritatea operațiilor în algebra booleană

Operator	Prioritate
()	mare
¯	
•	
⊕, ⊙	
+	mică

Tabelul 2. 4 Calculul tabelului de adevăr pentru expresia $\overline{a \cdot b} + c$

a b c	$\overline{a \cdot b} + c$
0 0 0 :	$\overline{0 \cdot 0} + 0 = 1 + 0 = 1$
0 0 1 :	$\overline{0 \cdot 0} + 1 = 1 + 1 = 1$
0 1 0 :	$\overline{0 \cdot 1} + 0 = 1 + 0 = 1$
0 1 1 :	$\overline{0 \cdot 1} + 1 = 1 + 1 = 1$
1 0 0 :	$\overline{1 \cdot 0} + 0 = 1 + 0 = 1$
1 0 1 :	$\overline{1 \cdot 0} + 1 = 1 + 1 = 1$
1 1 0 :	$\overline{1 \cdot 1} + 0 = 0 + 0 = 0$
1 1 1 :	$\overline{1 \cdot 1} + 1 = 0 + 1 = 1$

a b c	$\overline{a \cdot b} + c$
0 0 0	1
0 0 1	1
0 1 0	1
0 1 1	1
1 0 0	1
1 0 1	1
1 1 0	0
1 1 1	1

2.5 Realizarea schemei logice pentru o expresie

Conectarea porților în cadrul schemei logice este strict legată de ordinea operațiilor din expresia care trebuie implementată.

De exemplu, considerând funcția $f = b + a \cdot \overline{a + (b + a)}$ și ținând cont de prioritatea operațiilor (din Tabelul 2. 3) rezultă următorii pași de efectuare a operațiilor:

1. P1 = $b + a$ (operația SAU din paranteză)
2. P2 = $\overline{a + P1}$ (operația SAU-NU)
3. P3 = $a \cdot P2$ (operația ȘI)
4. f = $b + P3$ (operația SAU)

Dacă amplasarea porților se face de la stânga la dreapta, dinspre intrări spre ieșiri, atunci schema logică rezultată este prezentată în Figura 2. 2.

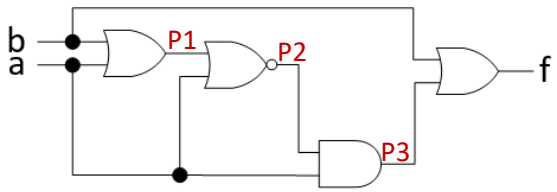


Figura 2. 2 Schema logică pentru funcția $f = b + a \cdot \overline{a + (b + a)}$

2.6 Proprietăți ale algebrei booleene

Două funcții booleene sunt considerate echivalente, dacă pentru toate combinațiile de intrare au rezultate identice.

Folosind tabelele de adevăr ale operațiilor de bază se pot demonstra următoarele proprietăți (egalități):

- Comutativitate: $a + b = b + a$; $a \cdot b = b \cdot a$
- Asociativitate: $(a + b) + c = a + (b + c)$; $(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- Distributivitate: $a + (b \cdot c) = (a + b) \cdot (a + c)$; $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
- Element neutru: $a \cdot 1 = a$; $a + 0 = a$
  - Consecințe ale elementului neutru: $a + 1 = 1$; $a \cdot 0 = 0$
- Idempotența: $a + a = a$; $a \cdot a = a$
- Dubla negație: $\bar{\bar{a}} = a$
- Operația XOR: $a \oplus b = \bar{a} \cdot b + a \cdot \bar{b}$
- Operația XNOR: $a \odot b = a \cdot b + \bar{a} \cdot \bar{b}$
- De Morgan: $\overline{a + b} = \bar{a} \cdot \bar{b}$; $\overline{a \cdot b} = \bar{a} + \bar{b}$

2.7 Transformări ale porților logice și generarea de noi funcționalități

Se pot realiza diverse tipuri de porți din alte tipuri folosind proprietățile algebrei booleene. Un exemplu elocvent îl reprezintă reducerea numărului de intrări la porțile ȘI, ȘI-NU, SAU și SAU-NU. O poartă ȘI cu 4 intrări care realizează expresia $a \cdot b \cdot c \cdot d$, poate fi redusă la 2 intrări $a \cdot b$ în mai multe feluri, conform echivalențelor: $a \cdot b = a \cdot 1 \cdot b \cdot 1 = a \cdot b \cdot a \cdot b = a \cdot b \cdot b \cdot b = a \cdot a \cdot a \cdot b = \cdots$, etc. Valoarea constantă 1 reprezintă sursa de alimentare a circuitului (5V) și poartă denumirea de VCC (Voltage Common Collector). Ea este disponibilă în lista de simboluri a utilitarului Project Navigator, în categoria *General*. Câteva soluții pentru implementarea unei porți ȘI cu 2 intrări, dintr-o poartă ȘI cu 4 intrări sunt prezentate în Figura 2. 3. Soluțiile sunt similare în cazul porții ȘI-NU.

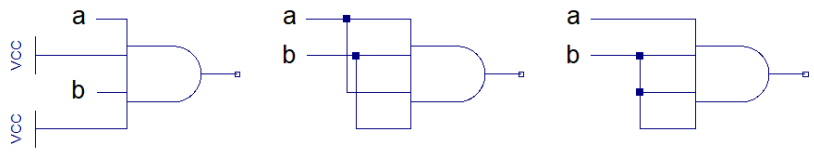


Figura 2. 3 Variante de transformare a unei porți ȘI cu 4 intrări, în poartă ȘI cu 2 intrări

Expresiile echivalente în cazul porții SAU implică folosirea constantei 0, denumită GND (Ground), adică masa circuitului (0V): $a + b = a + 0 + b + 0 = a + b + a + b = a + b + b + b = a + a + a + b = \cdots$, etc. O parte din soluții sunt prezentate în Figura 2. 4 și sunt identice și pentru poarta SAU-NU. Simbolul GND se află în categoria *General*.

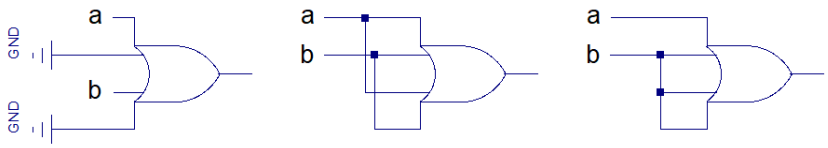


Figura 2. 4 Variante de transformare a unei porți SAU cu 4 intrări, în poartă SAU cu 2 intrări

O poartă inversor se poate realiza dintr-o poartă ȘI-NU sau o poartă SAU-NU cu 2 intrări conform expresiilor: $\bar{a} = \overline{a \cdot 1} = \overline{a \cdot a} = \overline{a + a} = \overline{a + 0}$ (Figura 2. 5).

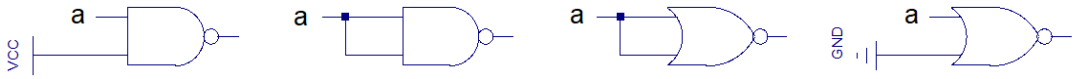


Figura 2. 5 Variante de realizare a unei porți NOT din alte porți

Folosind relațiile De Morgan, o poartă SAU se poate realiza cu ȘI-NU și inversoare, iar o poartă ȘI se poate realiza cu SAU-NU și inversoare: $a \cdot b = \overline{\bar{a} + \bar{b}}$, $a + b = \overline{\bar{a} \cdot \bar{b}}$ (Figura 2. 6).

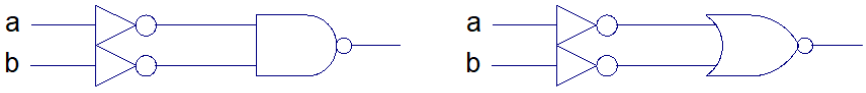
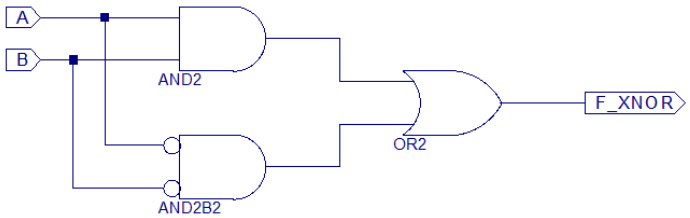


Figura 2. 6 Alternative de implementare a porților SAU (stânga) și ȘI (dreapta)

2.8 Activități practice

1. Implementați în cadrul proiectului *ttl_env*, într-o singură schemă, porțile fundamentale NOT, NOR și XOR cu 2 intrări și testați funcționalitatea acestora pe placă folosind tabelele de adevăr din Tabelul 2. 1.
2. Adăugați la proiect schema din figura următoare, care implementează expresia $a \cdot b + \bar{a} \cdot \bar{b}$ și testați pe placă faptul că această expresie are funcționalitatea porții XNOR folosind tabelul de adevăr al acesteia (din Tabelul 2. 1).



3. Pentru fiecare din expresiile următoare determinați tabelul de adevăr, adăugați câte o schemă corespunzătoare în cadrul proiectului și testați funcționalitatea pe placă cu tabelul de adevăr. Expresiile nu vor fi modificate folosind proprietățile algebrei booleene. Completați cerințele pentru fiecare punct, înainte de a trece la următorul.
   a. $\bar{a} \cdot b + \bar{c}$
   b. $a \cdot c + \overline{b \cdot c}$
   c. $a + \overline{b \cdot c}$
   d. $\overline{a + \bar{b}} \cdot c$
   e. $\overline{(a + b) \cdot \overline{a + c}}$
   f. $a + \overline{b \cdot \overline{a + c}}$

3 Editarea schematică și simularea funcționării circuitelor cu software specializat (I)

3.1 Obiective

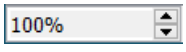
Se prezintă utilitarul Logisim care pune la dispoziție facilități de proiectare și simulare a circuitelor numerice digitale. Sunt prezentate elementele componente ale utilitarului și modul de lucru cu controalele active din componența simulatorului. Sunt redate câteva exemple de circuite, atât din punct de vedere al modului de implementare cât și în ceea ce privește procesul de simulare a funcționalității acestora. Se prezintă editorul de simboluri și posibilități de organizare a circuitelor în cadrul librăriei proiectului.

3.2 Utilitarul Logisim

Utilitarul Logisim (versiunea 2.7.1) oferă suport pentru descrierea circuitelor numerice digitale și simularea funcționalității acestora, prin aplicarea de stimuli pe terminalele de intrare și observarea rezultatelor care apar pe terminalele de ieșire. Descrierea unui circuit se face sub forma unei scheme de simboluri grafice interconectate, care poartă denumirea de *schemă logică*. Logisim pune la dispoziție un set de circuite logice de bază și oferă posibilitatea extinderii acestora cu alte circuite, printr-o implementare în stil ierarhic și grupare în librării de circuite reutilizabile. Implementarea unui circuit presupune două faze de dezvoltare:

1. Descrierea schemei logice în editorul schematic, prin care se definește structura circuitului. În modul de descriere a schemei logice se poate activa sau dezactiva modulul de simulare a funcționalității circuitului.
2. Definirea simbolului reprezentativ circuitului în vederea reutilizării acestuia în componența altor circuite. În acest fel se facilitează dezvoltarea ierarhică la niveluri de abstractizare, de complexitate progresivă.

3.2.1 Editorul schematic

Editorul schematic este disponibil imediat după lansarea în execuție a utilitarului Logisim. Acesta conține o zonă de lucru în care se poate realiza schema logică a circuitului (Figura 3. 1). Dimensiunea elementelor de pe schemă se poate schimba (zoom in/zoom out) cu ajutorul casetei 

din colțul stânga-jos.

3.2.1.1 Uneltele de lucru și organizarea pe librării (Toolbox)

În partea laterală a zonei de lucru se află uneltele de lucru pentru realizarea unei scheme logice (zona Toolbox). Acesta conține lista librăriilor disponibile, începând din partea superioară cu librăria proiectului denumită implicit *Untitled* (acesta este dealtfel și numele proiectului curent). Inițial, în cadrul acesteia se află doar circuitul curent – cu denumirea implicită *main* – a cărui schemă logică este în dezvoltare. Apăsând ⊞ în dreptul unei librării se pot vizualiza uneltele și circuitele din componența sa, sub formă de simboluri grafice asociate la numele lor. Astfel, în cadrul librăriei *Gates* se pot accesa

porțile logice fundamentale, iar în celelalte librării se pot descoperi circuite logice de complexitate mai ridicată sau diverse unelte necesare în realizarea schemei logice.

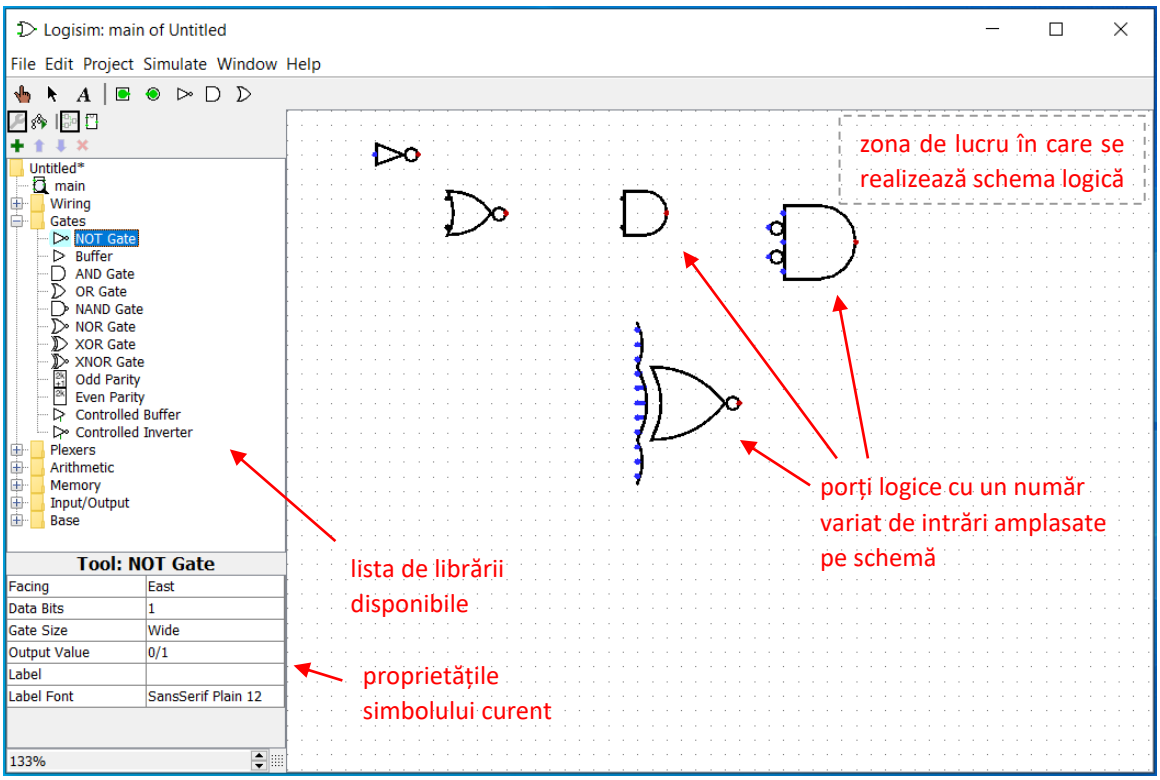


Figura 3. 1 Editorul schematic al utilitarului Logisim

3.2.1.2 Simbolurile, amplasarea pe schemă și proprietățile acestora

Simbolurile circuitelor puse la dispoziție de librăriile existente se pot introduce pe schemă prin selectare și apăsarea mouse-ului în poziția dorită a zonei de lucru. Ulterior, pot fi reamplasate în altă poziție prin selectare și deplasare (drag-and-drop). Un simbol de pe schemă se poate multiplica cu opțiunile de copiere (Copy) și lipire (Paste) disponibile în secțiunea Edit a meniului din partea superioară a ferestrei principale sau cu combinația de taste *Ctrl+C*, urmată de *Ctrl+V* sau simplu *Ctrl+Insert*. Ori de câte ori un simbol este selectat, va apărea în partea inferioară a listei de librării numele simbolului și o listă de proprietăți specifice acestuia. De exemplu, prin selectarea inversorului NOT (Figura 3. 1) setul de proprietăți cuprinde:

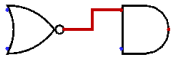
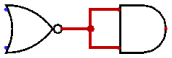
- `Facing` – direcția de amplasare pe schemă (rotirea simbolului);
- `Data Bits` – numărul de biți pentru care se aplică funcția logică NOT;
- `Gate Size` – dimensiunea simbolului pe schemă;
- `Output Value` – comportamentul la ieșire: poate avea “0 logic” sau “1 logic”;
- `Label` – eticheta de identificare pe schemă, care este opțională;
- `Label Font` – dimensiunea textului etichetei.

Proprietățile diferă de la simbol la simbol, dar o parte dintre acestea sunt comune la majoritatea. Valorile proprietăților se pot modifica, ceea ce poate avea efect asupra funcționalității circuitului asociat simbolului și/sau asupra aspectului acestuia în zona de

lucru, în funcție de caz. De exemplu: o poartă ȘI-NU (NAND) cu 2 intrări (`Number Of Inputs = 2`) are simbolul  ; dacă are 5 intrări simbolul devine  ; dacă intrările 2 și 4 sunt inversate (`Negate 2 = Yes; Negate 4 = Yes`) atunci apare un cerculeț în dreptul acestora:  .

3.2.1.3 Realizarea conexiunilor pe schema logică

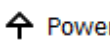
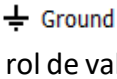
Se observă faptul că intrările și ieșirile apar, de regulă, sub forma unor puncte mai proeminente amplasate la extremitățile simbolului, având culori diferite. Pentru o localizare fiabilă a acestora, amplasarea mouse-ului în dreptul lor duce la apariția unui cerculeț verde în jur:  .

Conectarea intrărilor și ieșirilor se realizează prin apăsarea butonului de mouse și deplasarea acestuia pe schemă de la un pin la celălalt:  . Crearea unei ramificații într-un punct al unui fir se poate realiza prin trasarea unui nou fir, începând cu acel punct până la o intrare/ieșire neconectată:  . Ramificația se distinge pe fir printr-o bulină de culoarea acestuia. Astfel, tipurile de conexiuni posibile pot fi:

- conexiune directă de la o ieșire la o intrare;
- conexiune de la o ieșire la un fir existent sau de la un fir existent la o intrare; de menționat faptul că două fire separate devin unul singur prin conectare.

Un cablu cu mai multe fire se poate crea cu elementul  din librăria *Wiring*, pentru care atributele `Fan Out` și `Bit Width In` se setează la numărul de fire. **Notă**: Orice element de pe schemă (simbol sau segment de fir) se poate șterge prin click-dreapta pe acesta și alegerea opțiunii de ștergere (Delete) sau apăsând tasta *Del*, după selectare.

3.2.1.4 Elemente de circuit cu valori constante

Printre alte elemente de circuit puse la dispoziție de librăria *Wiring* se pot remarca  și  . reprezintă sursa de curent, care are valoarea "1 logic" (VCC), iar  reprezintă masa, care are valoarea "0 logic" (GND). Ambele pot fi folosite cu rol de valori constante în circuit. Utilizarea acestor elemente este exemplificată în Figura 3. 2.

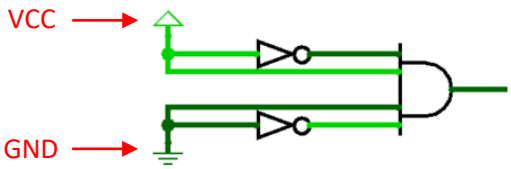


Figura 3. 2 Circuit cu intrările conectate la VCC și GND

Datorită faptului că simularea este activată implicit în Logisim, valorile constante se propagă producând efecte vizibile la ieșirile circuitelor și pe fire. Firele care transportă

valoarea “1 logic” au culoarea verde-deschis, iar cele care transportă valoarea “0 logic” sunt colorate cu verde-închis.

3.2.1.5 Unealta de text

Unealta Text Tool este disponibilă în cadrul librăriei *Base* sau în bara de unelte (Toolbar) situată sub meniu . Cu ajutorul ei se pot insera fragmente de text, fără o funcționalitate propriu-zisă. Textul poate conține explicații sau etichete relevante pentru analizarea schemei.

3.2.1.6 Terminale de intrare și ieșire

Orice schemă logică a unui circuit necesită, pe lângă valori constante, atât terminale de intrare prin care se transmit valori către circuit cât și terminale de ieșire prin care se pot prelua rezultatele produse de acesta. Numărul de terminale de intrare și ieșire depinde de structura circuitului. În timpul simulării, terminalele evidențiază valorile logice. În Logisim, terminalul de intrare are simbolul Pin și este accesibil în librăria *Wiring*. Atributul său numit `Output?` definește tipul terminalului: de intrare sau de ieșire. În cazul în care terminalul este de ieșire (`Output? = True`) simbolul adoptă o formă circulară , ca în Figura 3. 3. Un terminal poate avea un număr variabil de biți. Pentru terminale de 1 bit este necesar ca proprietatea `Data Bits = 1`. Terminalul se poate denumi dând valori proprietății `Label`.

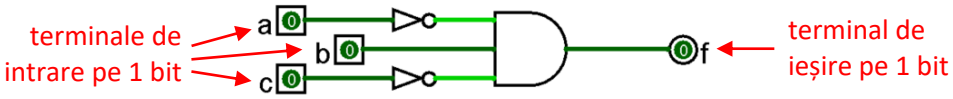


Figura 3. 3 Circuit cu terminale de intrare și ieșire pe care se pot identifica valorile logice în timpul simulării

3.2.1.7 Simularea în Logisim

În momentul plasării terminalelor de intrare, valorile implicite ale acestora se propagă prin circuit producând efecte corespunzătoare. În modul de editare a schemei nu se pot aduce modificări valorilor asociate terminalelor de intrare. Pentru a activa modificarea terminalelor de intrare trebuie să se acceseze unealta Poke Tool, situată atât în cadrul librăriei *Base*, cât și în bara de unelte . Ulterior, valorile terminalelor se pot schimba apăsând pe simbolul acestora în schemă. Revenirea la modul de editare a schemei logice se face accesând una din uneltele Select Tool sau Edit Tool situate tot în librăria *Base* sau în bara de unelte .

Dacă simulatorul detectează erori sau conexiuni incomplete în circuit, atunci terminalele de ieșire pot indica valori necunoscute sau eronate . Toate comenzile legate de sesiunea de simulare se află la secțiunea Simulate a meniului. De exemplu, oprirea simulării se poate realiza cu comanda Simulation Enabled sau cu combinația de taste *Ctrl+E*. Tot cu această comandă se realizează reactivarea simulării. Resetarea simulării se realizează cu comanda Reset Simulation sau cu combinația de taste *Ctrl+R*.

Aceasta este utilă după oprirea simulării, dacă se dorește resetarea ultimelor semnale valorice înregistrate în circuit și forțarea tuturor terminalelor la “0 logic”.

3.2.1.8 Bara de unelte a editorului schematic (Toolbar)

Bara de unelte  este situată dedesubtul meniului și conține un subset din elementele de circuit, dar și unelte care pot fi accesate cu ușurință de utilizator. Este posibilă modificarea componenței acesteia accesând opțiunea Options din secțiunea Project a meniului. În fereastra de opțiuni care apare pe ecran se va accesa tab-ul Toolbar care permite adăugarea și ștergerea de elemente.

3.2.2 Editorul de simboluri

Circuitul descris de schema logică se poate încapsula într-un simbol reprezentativ, de forma unui dreptunghi cu terminalele de intrare și ieșire (pini). Simbolul se salvează în librăria proiectului și poate fi utilizat în cadrul altor scheme logice, având funcționalitatea circuitului pe care îl încapsulează. Interacțiunea cu circuitul se va face prin terminale. Comutarea de la modul de editare a schemei logice la modul de editare a simbolului asociat, și invers, se realizează cu comenzile Edit Circuit Appearance, respectiv Edit Circuit Layout din secțiunea Project a meniului. Pentru o comutare rapidă, cele două comenzi apar și dedesubtul barei de unelte având simbolurile:  . Pentru circuitul din Figura 3. 3 se generează automat simbolul prezentat în Figura 3. 4.

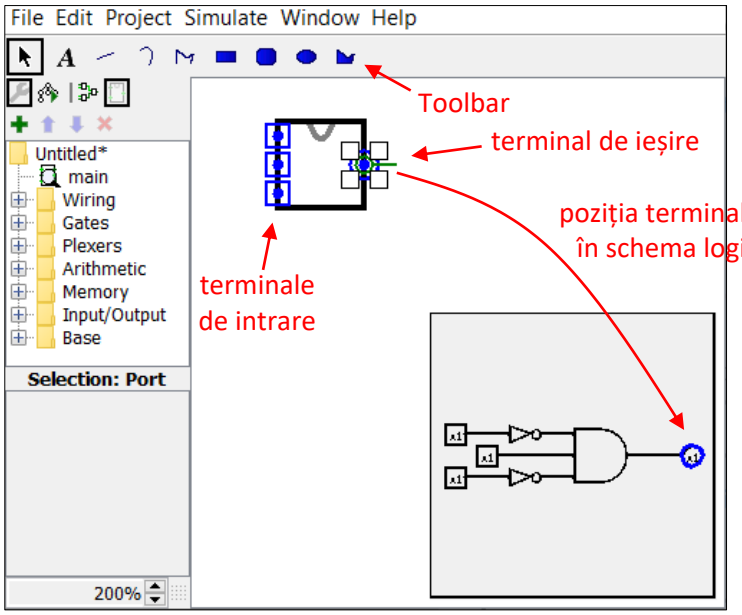


Figura 3. 4 Editarea simbolului asociat schemei logice curente

Simbolul poate suferi modificări ulterioare. Terminalele de intrare sunt marcate cu un dreptunghi albastru, iar cele de ieșire cu un cerculeț. Pentru identificarea unui terminal, selectarea acestuia pe simbol va evidenția poziția sa în schema logică. Opțional, denumirile terminalelor sau chiar a circuitului se pot amplasa pe simbol folosind unealta de text din Toolbar-ul editorului de simboluri:  .

3.2.3 Organizarea circuitelor în cadrul librăriei proiectului

În cadrul librăriei proiectului se pot crea unul sau mai multe circuite. Adăugarea unui circuit se realizează cu comanda Add circuit situată deasupra listei de librării. Programul va solicita numele noului circuit și se va crea o schemă logică fără elemente. Alături de această comandă se află comenzile de deplasare a circuitului curent într-o altă poziție, în cadrul librăriei , și comanda de ștergere a circuitului .

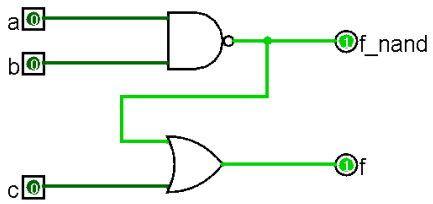
Se poate comuta între circuitele librăriei proiectului apăsând dublu-click pe numele circuitului, în Toolbox. Va apărea schema logică a circuitului în zona de lucru și se vor putea aduce modificări. În meniu, la secțiunea File, sunt comenzi de salvare a proiectului – Save, de încărcare a unui alt proiect – Open sau de creare a unuia nou – New. Un proiect se va salva cu tot cu librăria de circuite a acestuia.

Notă: Salvați proiectul din când în când și la final pentru a nu pierde circuitele realizate.

Prin selectarea unei scheme din librăria curentă se vor putea vizualiza și modifica proprietățile acesteia, printre care și numele indicat de atributul `Circuit Name`.

3.3 Activități practice

1. Creați un proiect nou în Logisim și realizați un circuit care să conțină porțile logice fundamentale NOT, AND, XOR, OR din librăria *Gates*. Configurați porțile să fie pe 1-bit, cu 2 intrări (cu excepția NOT). Introduceți 2 terminale de intrare, folosite în comun de către toate porțile și 4 terminale de ieșire, câte unul pentru fiecare poartă. Testați funcționarea porților logice cu ajutorul simulatorului și a tabelelor de adevăr corespunzătoare.
2. Adăugați un nou circuit în cadrul proiectului, în care să realizați schema logică din figura următoare. Calculați tabelul de adevăr al circuitului. Testați circuitul aplicând toate combinațiile posibile de valori pe intrările a, b, c și confruntați rezultatele cu valorile din tabel. Vizualizați simbolul generat pentru circuit și adăugați nume corespunzătoare în dreptul terminalelor de intrare și ieșire folosind unealta de text din Toolbar.



3. Adăugați câte un circuit separat în cadrul proiectului pentru următoarele funcții logice și testați-le în simulator confruntând rezultatele cu cele din tabelul de adevăr pentru toate combinațiile valorilor de intrare:

a) $f_1 = a + \bar{b} + c$
b) $f_2 = (a + b) \cdot (a + c)$
c) $f_3 = a + \overline{b \cdot \bar{c}}$
d) $f_4 = b \cdot (\overline{a + c} + b \cdot \bar{c})$
e) $f_5 = \overline{a \cdot \overline{b \cdot c}}$
f) $f_6 = \bar{a} + \bar{b} + \overline{a + c}$
g) $f_7 = \overline{a \cdot b} + b \cdot \bar{c}$

4 Editarea schematică și simularea funcționării circuitelor cu software specializat (II)

4.1 Obiective

Se prezintă modalități de gestionare a librăriilor în utilitarul Logisim și posibilitatea de dezvoltare ierarhică a circuitelor. Se exemplifică un circuit construit în stil ierarhic. Sunt redate modalități de analiză a circuitelor folosind utilitarul de analiză combinațională.

4.2 Gestionarea librăriilor

Un proiect în Logisim are atașată în mod implicit librăria proiectului, din care vor face parte circuitele dezvoltate sub formă de scheme logice. Fișierul în care se salvează proiectul conține librăria acestuia împreună cu toate schemele logice și dependențele sale. Acest lucru dă posibilitatea încărcării librăriei din fișierul respectiv într-un alt proiect, astfel că se pot include circuitele sale ca și componente ale altor scheme logice. Utilizarea într-o schemă logică a unor circuite din librăria curentă sau din alte librării poartă denumirea de *dezvoltare ierarhică*. În Logisim, încărcarea unei librării se realizează cu comanda Load Library > Logisim-library… din secțiunea Project a meniului. În pasul următor, se va selecta un fișier de proiect de pe disc (Figura 4. 1) din care Logisim va încărca librăria acestuia.

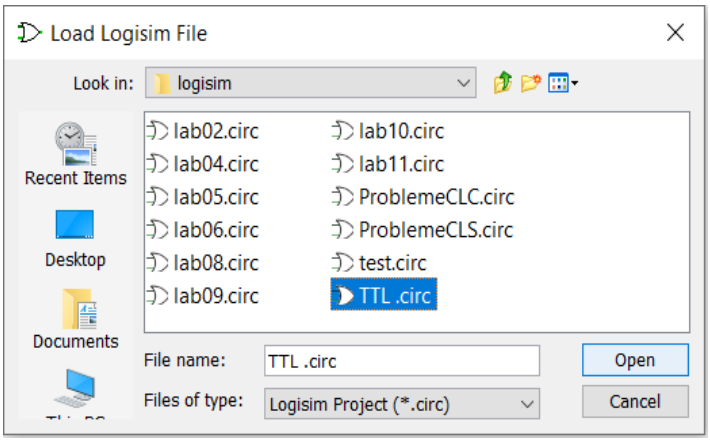


Figura 4. 1 La încărcarea unei librării se selectează fișierul proiectului care implementează librăria respectivă

Ulterior, librăria încărcată va apărea în Toolbox, între celelalte librării disponibile, iar circuitele sale vor fi disponibile sub forma simbolurilor asociate. În Figura 4. 2 este exemplificat accesul la circuitele din librăria TTL [1]. În cadrul unui proiect se pot încărca oricâte librării sunt necesare, ceea ce face posibilă utilizarea componentelor din librării multiple.

Atunci când se dorește eliminarea unei librării din proiect se poate apăsa butonul din dreapta al mouse-ului pe numele său și se alege opțiunea Unload Library. Alegând Reload Library se poate actualiza o librărie care este deja în Toolbox.

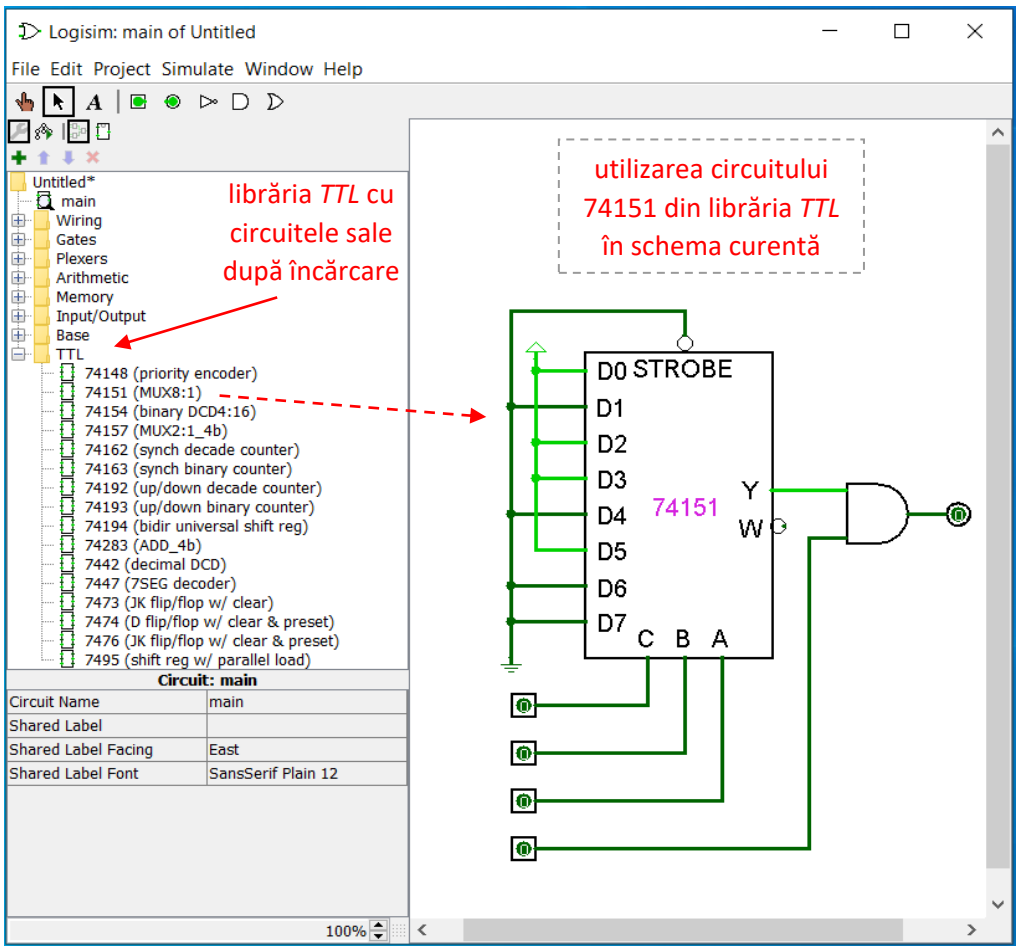


Figura 4. 2 După încărcare, librăria *TTL* va apărea în Toolbox și circuitele sale se pot accesa cu ajutorul simbolurilor asociate

4.3 Analiza unui circuit

Odată schema logică finalizată, se pot realiza următoarele tipuri de analiză:

- Analiza circuitului;
- Crearea de statistici.

Analiza circuitului se realizează cu comanda Analyze Circuit din secțiunea Project a meniului. Ulterior, va apărea utilitarul de analiză combinațională în fereastra Combinational Analysis, care are mai multe tab-uri (Figura 4. 3):

- Inputs – conține lista de terminale de intrare detectate. Ele sunt identificate după eticheta lor sau primesc automat un nume în lipsa etichetei.
- Outputs – conține lista de terminale de ieșire detectate. Similar cu intrările, ieșirile sunt identificate după etichetă sau primesc automat un nume.
- Table – evidențiază tabelul de adevăr al ieșirilor detectate în cadrul circuitului. Apăsând în tabel pe valorile de ieșire acestea își schimbă valoarea, oscilând între 0, 1 și x.

- Expression – evidențiază expresiile funcțiilor logice aferente fiecărei ieșiri, în Forma Disjunctivă Minimă sau în Forma Conjunctivă Minimă, în funcție de configurările din tab-ul Minimized.
- Minimized – evidențiază expresiile minimizate ale funcțiilor logice aferente fiecărei ieșiri, ca sumă de produse (Forma Disjunctivă Minimă) sau ca produs de sume (Forma Conjunctivă Minimă). Dacă sunt maxim 4 intrări, atunci se va evidenția și Diagrama Karnaugh a funcțiilor, împreună cu grupările de minimizare. Apăsând în celulele diagramei, valorile se modifică între 0, 1 și x, iar grupările și minimizarea se actualizează automat. Dacă butonul Set As Expression este activ, atunci trebuie apăsat pentru ca expresia minimizată să fie luată în considerare.

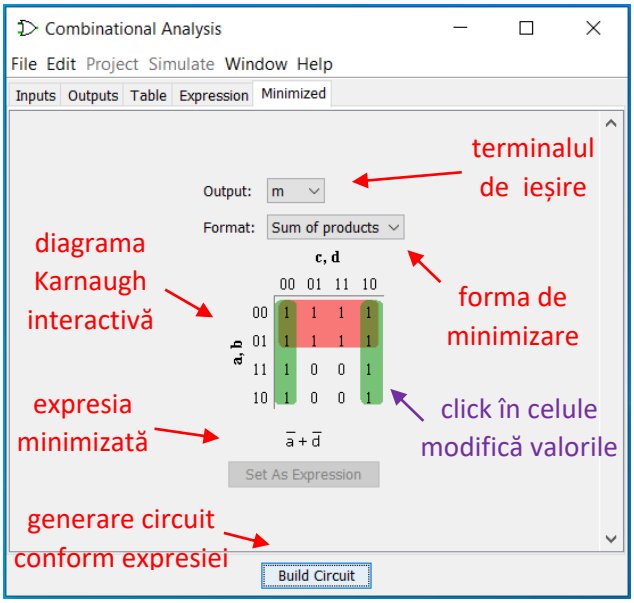


Figura 4. 3 Analiza combinațională a circuitului din Figura 4. 4

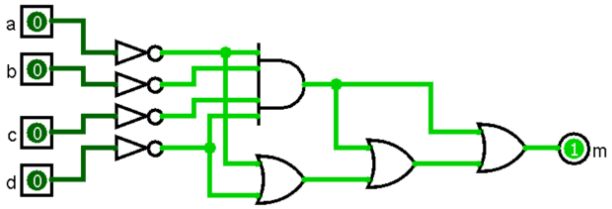


Figura 4. 4 Schemă logică complexă a cărei expresie se poate simplifica cu utilitarul de analiză combinațională

Notă: Analiza circuitului funcționează corect numai pentru circuite combinaționale: ale căror ieșiri depind strict de valorile de pe intrări.

Oricare dintre tab-uri conține butonul Build Circuit cu ajutorul căruia se va genera circuitul corespunzător funcțiilor logice curente din utilitarul Combinational Analysis. Se va solicita numele noului circuit și apoi va fi inclus în librăria proiectului. De exemplu, circuitul rezultat după minimizarea schemei din Figura 4. 4, cu utilitarul de analiză combinațională, este redat în Figura 4. 5.

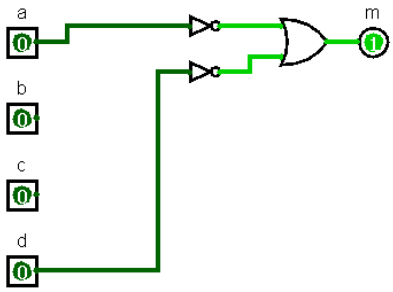


Figura 4. 5 Schema logică rezultată prin minimizarea circuitului din Figura 4. 4

Utilitarul de analiză se poate folosi și pentru generarea schemei unui circuit definit în formă analitică sau grafică. Chiar dacă nu există niciun element pe schemă, utilitarul se poate lansa și cu comanda Combinational Analysis de la secțiunea Window a meniului. În cadrul acestei ferestre, la tab-ul Inputs se vor introduce etichetele terminalelor de intrare, iar la tab-ul Outputs, pe cele ale terminalelor de ieșire. Funcțiile logice corespunzătoare terminalelor de ieșire se pot defini analitic la tab-ul Expression, prin tabel de adevăr la tab-ul Table sau prin diagramă Karnaugh, dacă au cel mult 4 intrări, la tab-ul Minimized. După definirea funcțiilor într-una din forme se poate crea circuitul corespunzător cu comanda Build Circuit.

Notă: La introducerea prin expresie analitică se acceptă operatorii NOT, AND, OR, XOR. Inversarea NOT se reprezintă cu caracterul ~, operația AND se reprezintă cu un spațiu, OR cu caracterul +, iar XOR cu caracterul ^. Se pot folosi paranteze dacă este necesar. După introducerea unei expresii se apasă butonul Enter (Figura 4. 6).

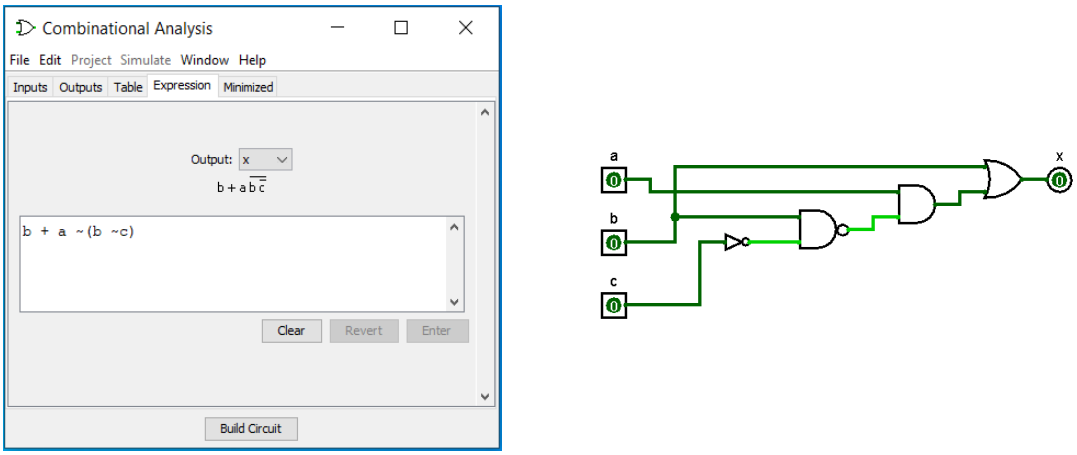
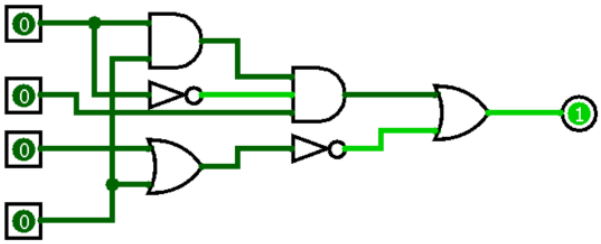


Figura 4. 6 Introducerea unei expresii în utilitarul de analiză combinațională (stânga). Circuitul generat corespunzător expresiei (dreapta)

Generarea de statistici se realizează cu comanda Get Circuit Statistics din secțiunea Project a meniului. Va apărea o fereastră cu diverse statistici legate de circuitele din componența schemei logice curente.

4.4 Activități practice

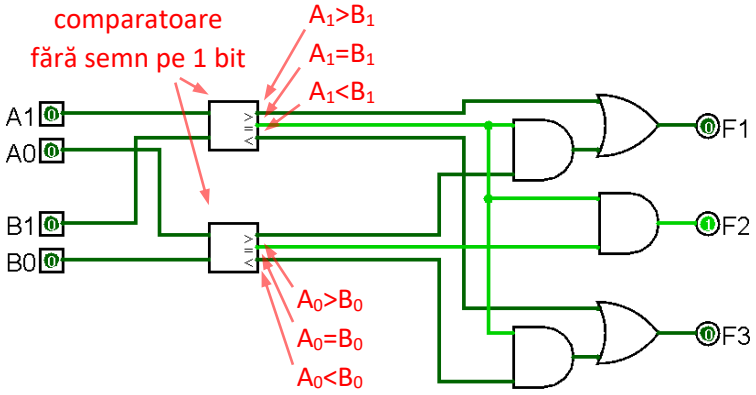
1. Creați un proiect nou în Logisim și încărcați librăria *TTL*. Vizualizați lista circuitelor din componența sa.
2. Creați o nouă schemă pentru circuitul din figura următoare și folosiți utilitarul de analiză combinațională pentru a genera circuitul echivalent obținut prin minimizare cu tab-ul Minimize.



3. Folosiți tab-urile Inputs, Outputs și Expression ale utilitarului de analiză combinațională pentru a implementa următoarele funcții logice într-o singură schemă logică:
f1 = ~a + b + ~c f2 = ~(a + b) (a + c) f3 = ~a + ~(~b c)
4. Realizați schema din figura următoare, care reprezintă un comparator fără semn pe 2 biți folosind comparatoare fără semn pe 1 bit (cu atributul `Numeric Type=Unsigned`) din librăria *Arithmetic*, și alte porți logice fundamentale necesare. Testați funcționarea comparatorului în simulator, pentru toate combinațiile posibile pe intrări.

Explicații la schemă: Cele 2 numere pe 2 biți care se compară sunt A_1A_0 și B_1B_0. Se iau în considerare următoarele reguli pentru cele 3 situații posibile:

- $F_1 = A_1A_0 > B_1B_0$: F_1=1 dacă $A_1 > B_1$ **SAU** ($A_1 = B_1$ **ȘI** $A_0 > B_0$);
- $F_2 = A_1A_0 = B_1B_0$: F_2=1 dacă $A_1 = B_1$ **ȘI** $A_0 = B_0$;
- $F_3 = A_1A_0 < B_1B_0$: F_3=1 dacă $A_1 < B_1$ **SAU** ($A_1 = B_1$ **ȘI** $A_0 < B_0$).



4.5 Bibliografie

[1] Cristian-Cosmin Vancea, "Librăria TTL pentru Logisim", [Online]. Available: https://drive.google.com/uc?export=download&id=1j4kRe9JXdQi6MqnqsB5nrXfX1uwoVc43

5 Circuite logice combinaționale – optimizare și sinteză

5.1 Obiective

Se prezintă funcțiile incomplet definite și modul de reprezentare a acestora. Se studiază tehnici de simplificare a funcțiilor booleene. Se realizează circuite combinaționale pentru implementarea unor funcții booleene în formă simplificată. Se studiază și se verifică funcționarea unor circuite combinaționale: generator de cod Excess-3, comparator de numere pe 2 biți, sumatoare pe 1 sau mai mulți biți.

5.2 Considerații teoretice

Circuitele logice se pot clasifica în *combinaționale* și *secvențiale*. Circuitele *combinaționale* sunt caracterizate de faptul că ieșirile implementează funcții a căror variabile sunt determinate numai de intrările circuitului. Circuitele combinaționale compuse din porți logice fundamentale sunt ușor de identificat, prin faptul că nu prezintă conexiuni cu reacție, adică de la pini de ieșire înapoi la pini de intrare. Funcțiile implementate pot fi de două feluri: *incomplet definite* și *complet definite*.

O funcție este *incomplet definită* când nu se cunoaște sau nu reprezintă interes rezultatul pentru un subset din combinațiile de intrare. Pentru aceste combinații rezultatul se notează cu X sau Ø. De exemplu, un circuit care primește cifre zecimale codificate binar pe 4 biți de intrare, va avea un comportament nedefinit pentru valori binare în intervalul 1010-1111, deoarece nu au echivalent cifre zecimale. Un astfel de caz este circuitul care implementează codul Excess-3 (BCD incrementat cu 3) din Tabelul 5. 1. Circuitul va prezenta 4 biți pentru codul binar de intrare și 4 biți pentru codul Excess-3 de ieșire. Ieșirile vor implementa funcțiile W, X, Y, Z cu variabilele comune A, B, C, D.

Tabelul 5. 1 Tabelul de adevăr pentru codul Excess-3 (BCD incrementat cu 3)

A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

5.2.1 Tehnici de optimizare a circuitului

Prin procesul de optimizare se determină un circuit cu aceeași funcționalitate, dar cu o structură mai simplă. Se urmărește reducerea numărului de operații și a numărului de variabile implicate, ceea ce va determina reducerea numărului de porți, a conexiunilor necesare și a lungimii acestora. O parte din tehnicile folosite sunt descrise în continuare:

a) **Folosirea variabilelor auxiliare**: De exemplu, expresiile $\begin{cases} a = (c \odot d) \cdot (e \oplus f) \\ b = \overline{c \odot d} + (e \oplus f) \end{cases}$ se pot rescrie cu variabilele auxiliare $m = c \odot d$, $n = e \oplus f$, astfel: $\begin{cases} a = m \cdot n \\ b = \overline{m} + n \end{cases}$.

b) **Aplicarea proprietăților algebrei booleene**: Implică aplicarea de relații din algebra booleană, precum: $\overline{\overline{x}} = x$, $x + 1 = 1$, $x \cdot 1 = x$, $x + \overline{x} = 1$ sau $x \cdot \overline{x} = 0$, etc. Astfel, în expresia $x \cdot y \cdot z + x \cdot y \cdot \overline{z}$ se poate da factor comun $x \cdot y$ și se obține echivalența: $x \cdot y \cdot z + x \cdot y \cdot \overline{z} = x \cdot y \cdot (z + \overline{z}) = x \cdot y \cdot 1 = x \cdot y$.

c) **Utilizarea de tehnici cu caracter general**, precum minimizarea cu diagrame Karnaugh sau metoda Quine-McCluskey.

5.2.2 Sinteza funcțiilor booleene

Procesul de sinteză a funcțiilor booleene presupune definirea funcțiilor sub o formă grafică (tabel de adevăr, diagrama Karnaugh), aducerea la o reprezentare analitică optimizată și implementarea circuitului corespunzător. În continuare, se vor studia câteva exemple de sinteză a unor circuite uzuale.

5.2.2.1 Generator de cod Excess-3

Folosind reprezentarea funcțiilor W, X, Y, Z din Tabelul 5. 1, care realizează conversia de la codul binar la codul Excess-3, se realizează diagramele Karnaugh pentru fiecare dintre acestea, ca în Figura 5. 1.

Minimizarea funcțiilor booleene la Forma Disjunctivă Minimă (FDM): În cadrul diagramelor se urmărește realizarea de grupuri dreptunghiulare cu celule vecine de 1 și X, care se încercuiesc (Figura 5. 1). Prima și ultima coloană, precum și primul și ultimul rând, sunt considerate vecine. Grupurile trebuie să conțină un număr de celule putere a lui 2 cât mai mare, iar numărul grupurilor să fie cât mai mic, încât toate celulele de 1 din diagramă să fie incluse în cel puțin un grup. Nu se vor realiza grupuri care să conțină numai celule de X. De asemenea, este posibil ca celule de X să nu fie incluse în niciun grup. O celulă de 1 sau X poate să facă parte din mai multe grupuri, dacă acest lucru ajută la creșterea dimensiunii acestora.

Pentru fiecare grup identificat se va obține un termen, prin aplicarea operației de conjuncție (ȘI) peste variabilele cu valori constante în dreptul celulelor grupului. În cadrul termenului variabilele apar negate, dacă în dreptul celulelor din grup au valoarea constantă 0 sau nenegate, dacă au valoarea 1. Forma Disjunctivă Minimă se obține ca disjuncție peste termenii rezultați asociați grupurilor.

Minimizarea funcțiilor booleene la Forma Conjunctivă Minimă (FCM): Tehnica este simetrică celei de minimizare la forma disjunctivă, deoarece se urmărește, după

aceleași criterii, formarea de grupuri cu celule de 0. La nevoie, grupurile pot include și celule de X. Termenii rezultați din grupuri aplică o disjuncție (SAU) numai peste variabilele constante în dreptul grupului: variabilele apar negate, dacă sunt constante la 1 și nenegate, altfel. Forma Conjunctivă Minimă se obține prin conjuncția termenilor rezultați din grupuri.

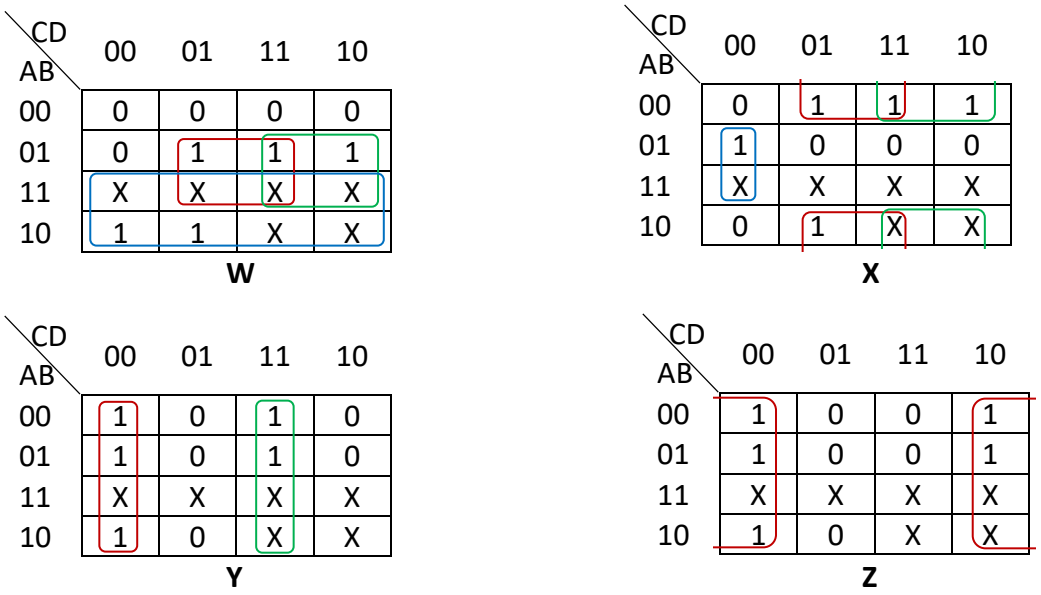


Figura 5. 1 Diagramele Karnaugh pentru ieșirile generatorului de cod Excess-3

Din Figura 5. 1 rezultă următoarele expresii la Forma Disjunctivă Minimă, pentru cele 4 funcții de ieșire ale circuitului:

$$\begin{aligned} W &= B \cdot D + B \cdot C + A \\ X &= \overline{B} \cdot D + \overline{B} \cdot C + B \cdot \overline{C} \cdot \overline{D} \\ Y &= \overline{C} \cdot \overline{D} + C \cdot D \\ Z &= \overline{D} \end{aligned} \qquad (5.\ 1)$$

5.2.2.2 Comparator de numere fără semn, pe 2 biți

Circuitul de comparare a 2 numere fără semn, pe 2 biți, $N_1=A_1A_0$, respectiv $N_2=B_1B_0$, generează 3 ieșiri F_1, F_2, F_3, care indică relația de mărime dintre acestea, după următoarele reguli:

- F_1=1, dacă $N_1>N_2$, altfel 0;
- F_2=1, dacă $N_1=N_2$, altfel 0;
- F_3=1, dacă $N_1<N_2$, altfel 0.

Schema bloc a circuitului este redată în figura următoare.

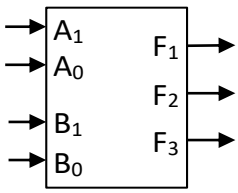


Figura 5. 2 Schema bloc a comparatorului fără semn, pe 2 biți

Așa cum s-a văzut în lucrarea anterioară, comparatorul fără semn, pe 2 biți se poate realiza cu comparatoare fără semn, pe 1 bit și porți logice. O soluție alternativă, bazată integral pe porți logice fundamentale, presupune generarea diagramelor Karnaugh ale ieșirilor, ca funcții de cele 4 intrări: A_1, A_0, respectiv B_1, B_0. De exemplu, în diagrama pentru F_1 se completează cu 1 în celulele pentru care $A_1A_0>B_1B_0$ și cu 0 în rest. Completarea pentru F_3 folosește o abordare simetrică, iar în cazul F_2 se pune 1, dacă $A_1A_0=B_1B_0$. Diagramele rezultate sunt redate în figura următoare:

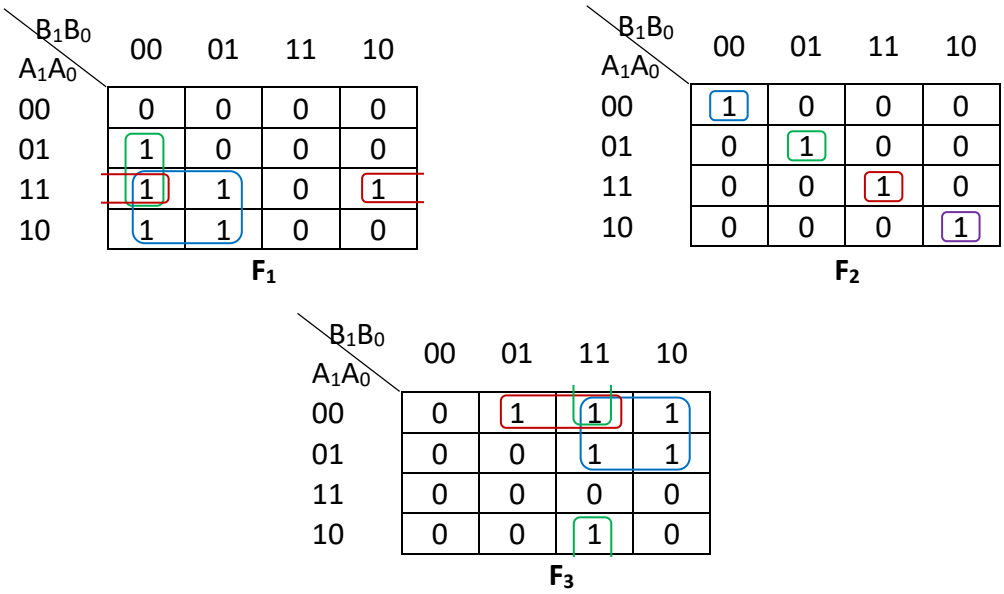


Figura 5. 3 Diagramele Karnaugh pentru ieșirile comparatorului fără semn, pe 2 biți

Expresiile la Forma Disjunctivă Minimă rezultate pe baza grupărilor efectuate în diagramele Karnaugh sunt următoarele:

$$\begin{aligned} F_1 &= A_1 \cdot \overline{B_1} + A_0 \cdot \overline{B_1} \cdot \overline{B_0} + A_1 \cdot A_0 \cdot \overline{B_0} \\ F_2 &= \overline{A_1} \cdot \overline{A_0} \cdot \overline{B_1} \cdot \overline{B_0} + \overline{A_1} \cdot A_0 \cdot \overline{B_1} \cdot B_0 + A_1 \cdot A_0 \cdot B_1 \cdot B_0 + A_1 \cdot \overline{A_0} \cdot B_1 \cdot \overline{B_0} \\ F_3 &= \overline{A_1} \cdot B_1 + \overline{A_0} \cdot B_1 \cdot B_0 + \overline{A_1} \cdot \overline{A_0} \cdot B_0 \end{aligned} \quad (5.\ 2)$$

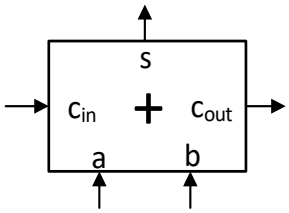
Aplicând axiomele și teoremele algebrei booleene funcția F_2 se poate rescrie într-o formă simplificată:

$$\begin{aligned} F_2 &= \overline{A_1} \cdot \overline{B_1} \cdot (\overline{A_0} \cdot \overline{B_0} + A_0 \cdot B_0) + A_1 \cdot B_1 \cdot (\overline{A_0} \cdot \overline{B_0} + A_0 \cdot B_0) = \\ &= (\overline{A_1} \cdot \overline{B_1} + A_1 \cdot B_1) \cdot (A_0 \odot B_0) = (\mathbf{A_1} \odot \mathbf{B_1}) \cdot (\mathbf{A_0} \odot \mathbf{B_0}) \end{aligned} \quad (5.\ 3)$$

5.2.2.3 Sumatorul complet pe 1 bit

Sumatorul complet pe 1 bit este unitatea de bază utilizată la implementarea sumatoarelor pe mai mulți biți, prin cascadarea acestuia de *n* ori, unde *n* este numărul de biți pe care se dorește să se realizeze operația de adunare. Concret, această unitate realizează suma a 3 intrări pe 1 bit și returnează rezultatul pe 2 biți. Cele 3 intrări sunt valorile a, b și transportul c_{in} (Carry In) rezultat din suma biților de rang inferior. Cei 2 biți de ieșire sunt transportul rezultat c_{out} (Carry Out) către rangul superior, respectiv

rezultatul s (Sum), de la rangul curent. Schema bloc a circuitului și tabelul de adevăr sunt prezentate în Figura 5. 4.



a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Figura 5. 4 Schema bloc și tabelul de adevăr pentru sumatorul complet pe 1 bit

Pe baza tabelului de adevăr se pot realiza diagramele Karnaugh pentru cele 2 ieșiri:

Figura 5. 5 Diagramele Karnaugh pentru ieșirile sumatorului complet pe 1 bit

Expresiile rezultate prin minimizare la Forma Disjunctivă Minimă sunt următoarele:

$$c_{out} = a \cdot c_{in} + b \cdot c_{in} + a \cdot b$$
$$s = a \cdot \bar{b} \cdot \overline{c_{in}} + \bar{a} \cdot \bar{b} \cdot c_{in} + a \cdot b \cdot c_{in} + \bar{a} \cdot b \cdot \overline{c_{in}} = \ldots = a \oplus b \oplus c_{in} \quad (5.\ 4)$$

5.2.2.4 Sumatorul pe 4 biți (semioctet) obținut prin cascadare

Implementarea unui sumator pe 4 biți (semioctet) între 2 numere $A_{3:0}$ și $B_{3:0}$ se poate realiza cu 4 sumatoare pe 1 bit, conectate încât transportul să se propage de la unitatea de rang inferior spre cele superioare (cascadare). Intrarea de transport la rangul cel mai mic va fi conectată la 0 (GND) – se obține un semi-sumator pe 4 biți. Transportul pe ieșire la rangul cel mai mare va indica depășirea numărului de 4 biți rezervat rezultatului $S_{3:0}$. Schema logică este prezentată în figura următoare:

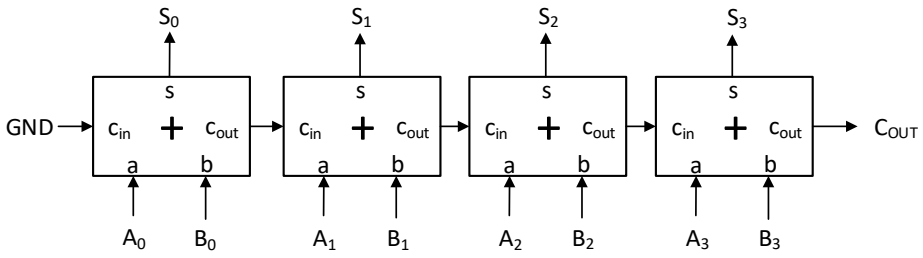


Figura 5. 6 Sinteza unui semi-sumator pe 4 biți prin cascadarea sumatoarelor pe 1 bit

Notă: Într-o manieră asemănătoare, sumatoarele pe 4 biți pot fi conectate pentru a implementa sumatoare pe orice număr de biți, multiplu de 4: intrarea de transport la rangul cel mai mic va fi conectată la 0, iar celelalte linii de transport vor fi interconectate pentru a realiza propagarea acestuia la sumatoarele de rang superior.

5.2.2.5 Implementarea sumatoarelor pe mai mulți biți folosind tehnici de minimizare

Alternativ, se pot implementa sumatoare pe mai mulți biți exprimând cifrele rezultatului ca funcții de biții de intrare. Astfel, se generează tabelul de adevăr al ieșirilor, se aplică tehnici de minimizare și se implementează expresiile obținute cu porți logice fundamentale. De exemplu, un semi-sumator pe 2 biți, care realizează suma între numerele A_1A_0 și B_1B_0, va prezenta la ieșire suma S_1S_0 și transportul C_{OUT}, cu funcționalitatea descrisă în Tabelul 5. 2. Cele 3 funcții, S_1, S_0, respectiv C_{OUT}, se pot minimiza folosind diagrame Karnaugh și proprietățile algebrei booleene, expresiile rezultate fiind următoarele:

Tabelul 5. 2 Semi-sumator pe 2 biți

A_1	A_0	B_1	B_0	C_{OUT}	S_1	S_0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

$$\begin{aligned} C_{OUT} &= A_1 \cdot B_1 + A_0 \cdot B_1 \cdot B_0 + A_1 \cdot A_0 \cdot B_0 \\ S_1 &= A_1 \oplus B_1 \oplus (A_0 \cdot B_0) \\ S_0 &= A_0 \oplus B_0 \end{aligned} \quad (5.5)$$

Observație: Avantajul acestei metode constă în faptul că numărul de porți logice cumulate este mai redus față de metoda implementării cu cascadare de sumatoare pe număr redus de biți. Dezavantajul îl reprezintă procesul de minimizare costisitor, odată cu creșterea exponențială a numărului de combinații posibile pe intrări. Diagramele Karnaugh pot fi folosite pentru minimizări de funcții cu maxim 4 variabile. Așadar, pentru sumatoare cu intrări pe mai mult de 2 biți sunt necesare tehnici de minimizare alternative, precum metoda Quine-McCluskey.

5.3 Activități practice

1. Aduceți la Forma Disjunctivă Minimă funcția: $f = \sum(0,4,7,10,12,13) + \sum_{\Phi}(1,8,15)$.
2. Implementați și testați pe placă ieșirea W a generatorului de cod Excess-3.
3. Implementați și testați pe placă funcția F_2 a comparatorului fără semn, pe 2 biți, cu porți XNOR.
4. Implementați și testați pe placă sumatorul complet pe 1 bit. Pentru implementarea ieșirii s (Sum) se va folosi varianta cu porți XOR.
5. Implementați și testați în Logisim un semi-sumator pe 4 biți folosind sumatoare pe 1 bit (cu atributul `Data Bits = 1`) din librăria *Arithmetic* .
6. Implementați și testați în Logisim un convertor de cod din 8421 (cod BCD) în 2421 (cod Aiken, https://en.wikipedia.org/wiki/Aiken_code). Pentru implementare, se va realiza tabelul de adevăr și se vor determina Formele Disjunctive Minime ale ieșirilor.

6 Circuite logice combinaționale elementare din categoria MSI

6.1 Obiective

Se analizează și se verifică funcționarea celor mai utilizate componente integrate din clasa MSI (Medium Scale Integration) precum demultiplexoarele, multiplexoarele și decodificatoarele. Se studiază codificatorul prioritar și convertorul din cod binar în cod Gray.

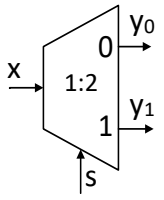
6.2 Considerații teoretice

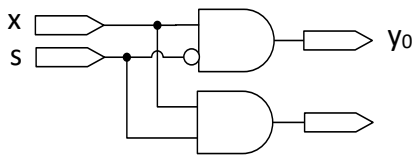
Circuitele TTL din categoria MSI au în componența lor un număr de 50-500 tranzistoare. Datorită funcționalității mai complexe față de porțile logice fundamentale, acestea sunt preferate în procesul de implementare a circuitelor, fiindcă pot reduce dimensiunea acestora, înlocuind un număr semnificativ de porți logice și conexiunile dintre ele. În general, se folosește o abordare mixtă, care combină circuitele MSI cu porțile logice fundamentale. Se vor studia în continuare câteva circuite MSI cu structură simplă, dar având o largă răspândire în implementarea hardware.

6.2.1 Circuite de demultiplexare

Demultiplexorul 1:*n* (DMUX 1:*n*) are un semnal de intrare de date **x**, *n* semnale de ieșire $\mathbf{y}_i$ și *m* semnale de selecție $\mathbf{s}_k$, unde $n=2^m$. Semnalul de ieșire cu indicele indicat de valoarea zecimală înscrisă pe semnalele de selecție va prelua valoarea semnalului de intrare, iar celelalte semnale de ieșire vor avea valoarea 0. Efectul demultiplexorului este descris de următoarea relație: $y_i = x$, dacă $i = \sum_{k=0}^{m-1} s_k \times 2^k$, altfel $y_i = 0$.

Circuitul care implementează demultiplexorul 1:2 este prezentat în Figura 6. 1, împreună cu simbolul asociat și tabelul de adevăr. Expresiile $y_0 = x \cdot \bar{s}$, $y_1 = x \cdot s$ se pot determina din tabelul de adevăr, prin minimizare la Forma Disjunctivă Minimă.





s	x	y_0	y_1
0	0	0	0
0	1	1	0
1	0	0	0
1	1	0	1

Figura 6. 1 Simbolul demultiplexorului DMUX 1:2 (stânga), schema logică (mijloc) și tabelul de adevăr (dreapta)

Demultiplexoarele cu număr mai mare de ieșiri se pot implementa folosind atât porți logice fundamentale, cât și tehnica de cascadare a unor demultiplexoare mai mici. În ceea ce privește tehnica de cascadare, pentru a implementa un demultiplexor DMUX 1:4 se pot folosi 3 demultiplexoare DMUX 1:2, cascadate pe două niveluri logice, ca în figura următoare:

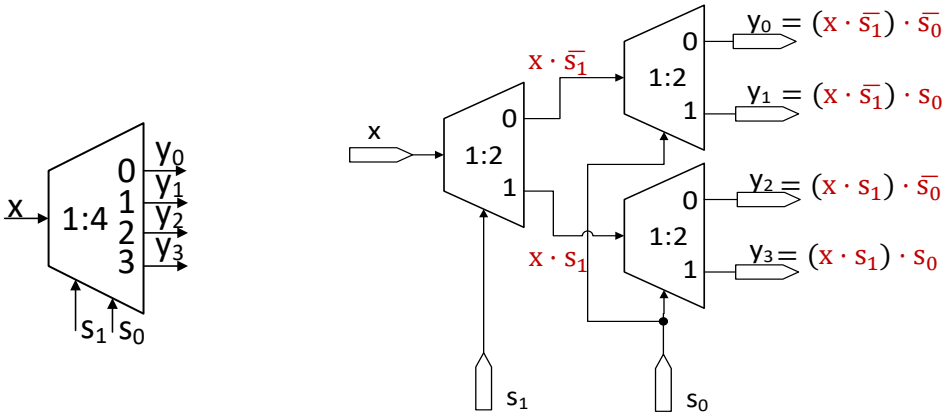


Figura 6. 2 Simbolul DMUX 1:4 (stânga) și implementarea prin cascadare (dreapta)

Notă: Dacă x = 1, atunci demultiplexorul activează numai ieșirea indicată de semnalele de selecție, având astfel funcționalitatea unui decodificator.

Există demultiplexoare cu un număr mai mare de ieșiri precum DMUX 1:8, 1:16, 1:32, etc.

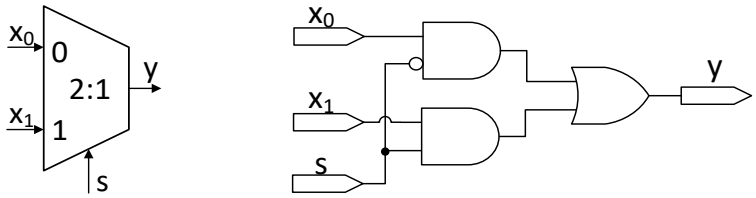
În Logisim, demultiplexoarele se regăsesc în librăria *Plexers*. Dimensiunea lor se poate stabili cu atributul `Select Bits`, care definește numărul de biți de selecție. Pentru ca intrarea și ieșirile de date să fie pe 1 bit se va seta `Data Bits = 1`. Opțional, demultiplexoarele pot prezenta o intrare de activare, care se poate elimina cu opțiunea `Include Enable = No`.

În Project Navigator, demultiplexoarele 1:4, 1:8, 1:16 se pot realiza cu ajutorul decodificatoarelor din categoria *Decoder*, denumite D2_4E, D3_8E, respectiv D4_16E. Intrarea de activare E (Enable) a decodificatoarelor poate fi folosită cu rol de intrare de date a demultiplexorului, iar intrările de date A_i, cu rol de selecție. Demultiplexorul 1:2 se poate regăsi în librăria locală a proiectului *ttl_env*, cu denumirea D1_2.

6.2.2 Circuite de multiplexare

Multiplexorul *n*:1 (MUX *n*:1) are *n* semnale de intrare de date $\mathbf{x}_i$, un semnal de ieșire **y** și *m* semnale de selecție $\mathbf{s}_k$, unde $n=2^m$. Semnalul de ieșire va prelua valoarea semnalului de intrare cu indicele indicat de valoarea zecimală înscrisă pe semnalele de selecție. În gereral, multiplexoarele sunt utilizate atunci când se dorește selectarea pe ieșire a unui singur semnal, din mai multe semnale de intrare. Efectul multiplexorului poate fi descris astfel: $y = x_i$, unde $i = \sum_{k=0}^{m-1} s_k \times 2^k$.

Multiplexorul 2:1 prezentat în Figura 6. 3 selectează o singură intrare, pe baza selecției s. Pornind de la tabelul de adevăr și realizând minimizarea la Forma Disjunctivă Minimă, se poate deduce următoarea expresie a ieșirii: $y = x_0 \cdot \bar{s} + x_1 \cdot s$.



s	x_0	x_1	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Figura 6. 3 Simbolul multiplexorului MUX 2:1 (stânga), schema logică (mijloc) și tabelul de adevăr (dreapta)

Similar cu tehnica de la demultiplexoare, un multiplexor cu mai multe intrări se poate implementa prin cascadare de multiplexoare de dimensiuni reduse sau cu porți logice fundamentale. De exemplu, un multiplexor MUX 4:1 se poate realiza prin cascadarea a 3 multiplexoare MUX 2:1 amplasate pe 2 niveluri logice:

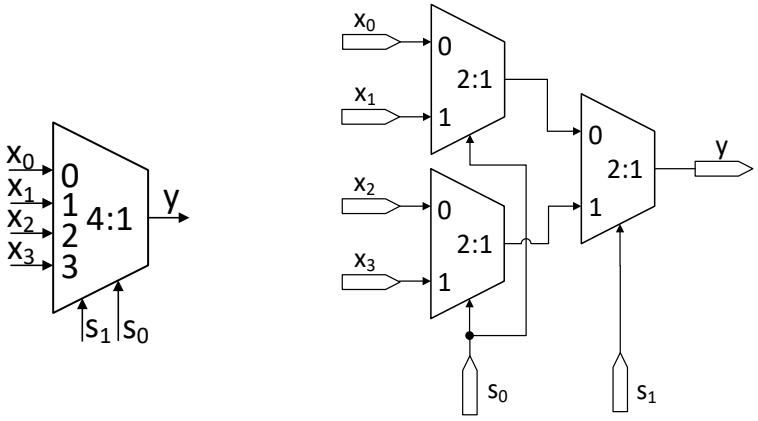


Figura 6. 4 Simbolul MUX 4:1 (stânga) și implementarea prin cascadare (dreapta)

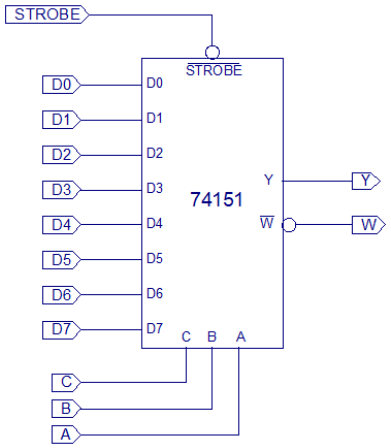
Se pot implementa astfel multiplexoare extinse precum: MUX 8:1, 16:1, 32:1, etc.

În Logisim, multiplexoarele se află în librăria *Plexers*. Atributul `Select Bits` definește numărul de biți de selecție; `Data Bits` setează numărul de biți pe intrările și ieșirea de date. Intrarea de activare se poate elimina setând `Include Enable = No`.

În Project Navigator, multiplexoarele sunt amplasate în categoria *Mux*, cu denumirile: M2_1, M2_1E, M4_1E, M8_1E. Sufixul E indică prezența intrării de activare E (Enable), care trebuie conectată la VCC, altfel ieșirea va fi 0.

Circuitul integrat 74151 (Figura 6. 5) implementează un MUX 8:1 cu intrările de date D_0, …, D_7, intrările de selecție A, B, C și 2 ieșiri complementare: Y, respectiv W. Ieșirea Y funcționează în logica pozitivă (activă pe 1), iar ieșirea W în logica negativă (activă pe 0): $W = \overline{Y}$. Selecțiile codifică numărul binar CBA_2. Suplimentar, există și o intrare de activare STROBE (S), în logica negativă. Dacă STROBE = 0 circuitul prezintă un regim de funcționare normal, altfel Y = 0 și $W = \overline{Y} = 1$.

În Logisim, toate circuitele integrate se regăsesc în librăria TTL. În proiectul *ttl_env*, integratele se găsesc în librăria proiectului, cu prefixul **TTL_** urmat de codul circuitului.



STROBE	C	B	A	Y	W
1	X	X	X	0	1
0	0	0	0	D_0	$\overline{D_0}$
0	0	0	1	D_1	$\overline{D_1}$
0	0	1	0	D_2	$\overline{D_2}$
0	0	1	1	D_3	$\overline{D_3}$
0	1	0	0	D_4	$\overline{D_4}$
0	1	0	1	D_5	$\overline{D_5}$
0	1	1	0	D_6	$\overline{D_6}$
0	1	1	1	D_7	$\overline{D_7}$

Figura 6. 5 Circuitul 74151 implementează un MUX 8:1

6.2.3 Circuite de decodificare

Decodificatorul are *n* semnale de intrare și maxim 2^n ieșiri. Decodificatorul activează ieșirea cu indicele indicat de codul aplicat pe intrări. Toate celelalte ieșiri sunt menținute inactive. Decodificatoarele pot avea ieșirile în logica pozitivă sau negativă.

Integratul 7442 (Figura 6. 6 – stânga) este un decodificator BCD-zecimal cu 4 intrări, care alcătuiesc codul binar $DCBA_2$, și 10 ieșiri în logica negativă, notate de la 0 la 9. Funcționarea acestuia este în conformitate cu Tabelul 6. 1. Pentru codurile binare în intervalul 1010_2-1111_2 toate ieșirile sunt inactive (au valoarea 1).

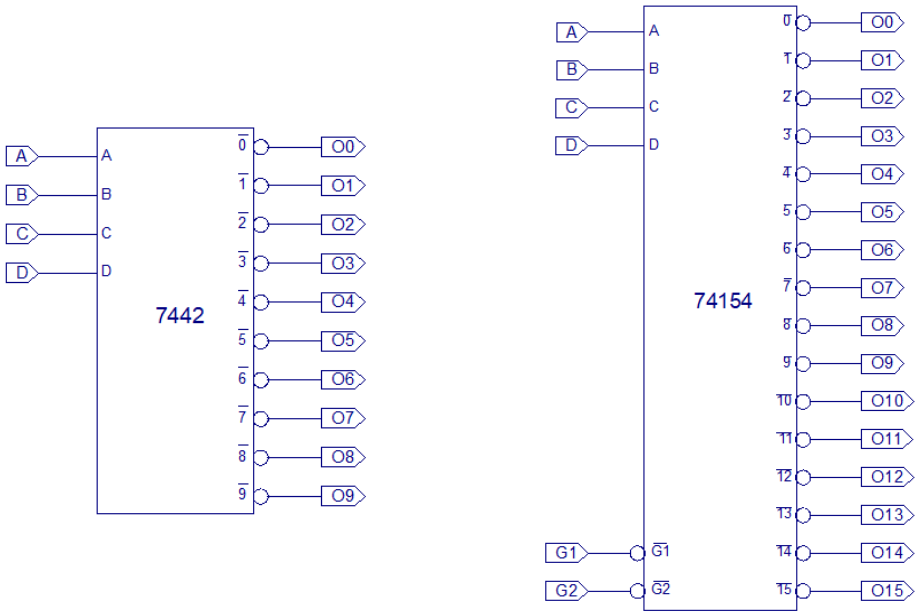


Figura 6. 6 Decodificatorul zecimal 7442 și decodificatorul hexazecimal 74154

Există și varianta cu 16 ieșiri, sub forma integratului 74154, denumit decodificator hexazecimal 4:16 (DCD 4:16). Orice combinație pe intrări, cu valori între 0 și 15, va activa ieșirea corespunzătoare (valoarea 0). Acest circuit are 2 intrări suplimentare de activare numite G_1 și G_2, care funcționează în logica negativă. Pentru activarea circuitului este necesar ca $G_1 = G_2 = 0$. Orice altă combinație inactivează circuitul plasând ieșirile pe 1.

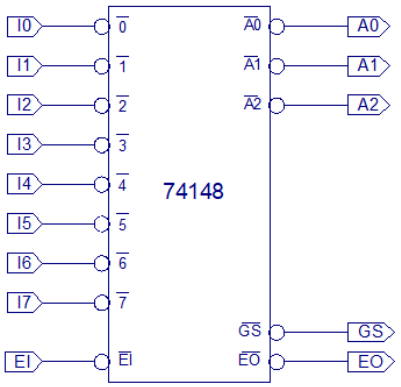
Tabelul 6. 1 Tabelul de adevăr al decodificatorului 7442

BCD				Zecimal									
D	C	B	A	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	1	1	1	1	1	1	1	1
0	0	0	1	1	0	1	1	1	1	1	1	1	1
0	0	1	0	1	1	0	1	1	1	1	1	1	1
0	0	1	1	1	1	1	0	1	1	1	1	1	1
0	1	0	0	1	1	1	1	0	1	1	1	1	1
0	1	0	1	1	1	1	1	1	0	1	1	1	1
0	1	1	0	1	1	1	1	1	1	0	1	1	1
0	1	1	1	1	1	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0
1	0	1	0	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1

6.2.4 Circuite de codificare – codificatorul prioritar

Codificatorul prioritar are 2^n intrări și n ieșiri. Ieșirile codifică indicele intrării active, cu prioritatea cea mai mare. Prioritatea crește progresiv cu indicele.

Circuitul 74178 este un codificator prioritar cu 8 intrări și 3 ieșiri, în logica negativă (Figura 6. 7). Suplimentar, intrarea EI are rol de activare a circuitului (0 – activ, 1 – inactiv). Ieșirea GS se activează (GS = 0), când codul binar $A_{2:0}$ expus la ieșire este valid. Codul poate fi invalid când circuitul este dezactivat (EI = 1) sau dacă nicio intrare nu este activă. La rândul ei, ieșirea EO se activează (EO = 0), dacă circuitul este activ și toate intrările sunt inactive. Comportamentul circuitului este prezentat în tabelul din figura următoare:



	Intrări								Ieșiri				
EI	0	1	2	3	4	5	6	7	A_2	A_1	A_0	GS	EO
1	X	X	X	X	X	X	X	X	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	X	X	X	X	X	X	X	0	0	0	0	0	1
0	X	X	X	X	X	X	0	1	0	0	1	0	1
0	X	X	X	X	X	0	1	1	0	1	0	0	1
0	X	X	X	X	0	1	1	1	0	1	1	0	1
0	X	X	X	0	1	1	1	1	1	0	0	0	1
0	X	X	0	1	1	1	1	1	1	0	1	0	1
0	X	0	1	1	1	1	1	1	1	1	0	0	1
0	0	1	1	1	1	1	1	1	1	1	1	0	1

Figura 6. 7 Codificatorul prioritar 74148 (stânga) și tabelul de adevăr (dreapta)

6.2.5 Circuite de conversie – convertorul binar-Gray pe 4 biți

Convertorul realizează conversia între coduri de reprezentare. În general, sunt folosite pentru comunicația între sisteme care prelucrează informația în codificări diferite.

Convertorul binar-Gray pe 4 biți prezintă 4 intrări și 4 ieșiri, pe care se realizează codificările respective. La intrare se aplică un cuvânt în cod binar, iar la ieșire se obține codul Gray corespunzător (similar, se poate implementa și conversia inversă Gray-binar). Funcționalitatea circuitului este prezentată în tabelul următor:

B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

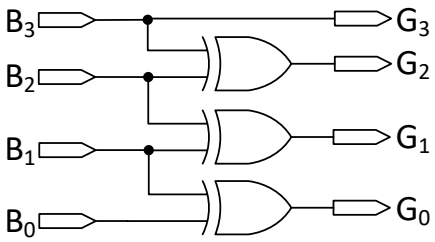


Figura 6. 8 Conversia binar-Gray (stânga) și implementarea convertorului (dreapta)

Fiecare ieșire G_i poate fi tratată ca o funcție logică aparte. În urma simplificării cu tehnici de minimizare și folosind proprietățile algebrei booleene, se obțin următoarele expresii pentru ieșiri: $G_3 = B_3$, $G_2 = B_2 \oplus B_3$, $G_1 = B_1 \oplus B_2$, $G_0 = B_0 \oplus B_1$.

6.3 Activități practice

1. Implementați și testați pe placă multiplexorul 74151.
2. Implementați și testați pe placă decodificatorul zecimal 7442.
3. Implementați și testați pe placă codificatorul prioritar 74148.
4. Implementați cu porți logice și testați în Logisim un demultiplexor DMUX 1:2.
5. Implementați cu demultiplexoare DMUX 1:2 și testați în Logisim un demultiplexor DMUX 1:4.
6. Implementați cu multiplexoare MUX 2:1 și testați în Logisim un multiplexor MUX 4:1.
7. Implementați și testați în Logisim decodificatorul hexazecimal 74154.
8. Implementați cu porți XOR și testați în Logisim convertorul binar-Gray pe 4 biți.
9. Implementați funcția logică $f(A, B, C, D, E) = \overline{B} \cdot \overline{C} \cdot D + C \cdot \overline{D} \cdot E + B \cdot \overline{C} \cdot \overline{E} + A \cdot \overline{B} \cdot \overline{D} + A \cdot B \cdot C \cdot D$, folosind numai un multiplexor, semnalele 0 și 1 și variabilele conectate direct, fără a fi inversate.

7 Circuite logice combinaționale complexe din categoria MSI

7.1 Obiective

Se analizează și se verifică funcționarea unor componente integrate MSI care implementează funcții mai complexe precum: multiplexorul cu calea de date pe mai mulți biți, sumatorul, unitatea aritmetică-logică și decodificatorul BCD-7 segmente. Se proiectează un sumator-scăzător cu ajutorul unui sumator și se testează funcționalitatea acestuia. Se extinde numărul de biți prin cascadare.

7.2 Considerații teoretice

Circuitele MSI complexe au o largă utilizare în aplicațiile bazate pe sisteme numerice. Operațiile întâlnite cel mai frecvent sunt cele aritmetice și logice, în consecință, o unitate aritmetică-logică este practic nelipsită. Creșterea numărului de biți de calcul este o abordare naturală în contextul în care aceste sisteme prelucrează date reprezentate pe multiplu de 8 biți (1 octet / byte). În general, afișarea rezultatelor se realizează în format zecimal. Decodificatoarele BCD-7 segmente facilitează afișarea valorilor binare în baza 10, cu ajutorul afișoarelor cu 7 segmente.

7.2.1 Multiplexoare și demultiplexoare cu mai multi biți pe calea de date

Multiplexoarele realizează selecția datelor de la un set de intrări către o singură ieșire, în funcție de valorile semnalelor de selecție. Demultiplexoarele realizează plasarea datelor de la o intrare unică la una din ieșirile indicate de semnalele de selecție. Multiplexoarele și demultiplexoarele se pot proiecta astfel încât intrările și ieșirile de date să fie reprezentate pe mai mulți biți. De exemplu, un MUX 4:1 cu calea de date pe 3 biți, are intrările și ieșirile de date pe 3 biți. El poate fi interpretat ca 3 unități MUX 4:1 cu calea de date pe 1 bit, ca în Figura 7. 1 – stânga, care funcționează concomitent și partajează semnalele de selecție.

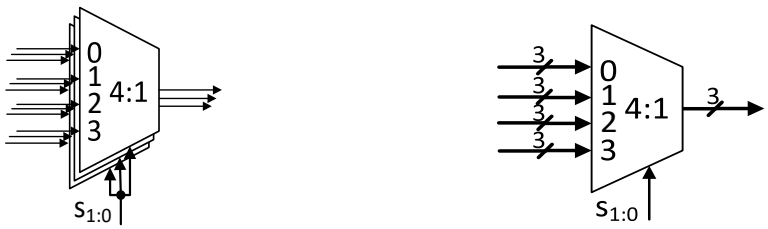


Figura 7. 1 MUX 4:1 cu calea de date pe 3 biți – implementarea cu unități MUX 4:1 având calea de date pe 1 bit (stânga) și simbolul asociat (dreapta)

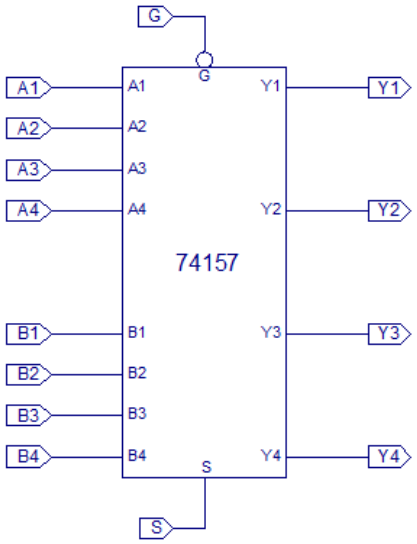
Similar, un DMUX 1:4 cu calea de date pe 3 biți poate fi implementat cu un set de 3 unități DMUX 1:4 cu calea de date pe 1 bit având selecțiile partajate (Figura 7. 2).

Figura 7. 2 DMUX 1:4 cu calea de date pe 3 biți – implementarea cu unități DMUX 1:4 având calea de date pe 1 bit (stânga) și simbolul asociat (dreapta)

Circuitul 74157 implementează un MUX 2:1 cu calea de date pe 4 biți. Simbolul circuitului este prezentat în Figura 7. 3. Intrările de date $A_{4:1}$, $B_{4:1}$ și ieșirea $Y_{4:1}$ sunt reprezentate pe 4 biți. Selecția S pe 1 bit stabilește intrarea care va fi înaintată pe ieșire, după regula:

$$Y_{4:1} = \begin{cases} A_{4:1}, \text{dacă } S = 0 \\ B_{4:1}, \text{dacă } S = 1 \end{cases} \quad (7.\ 1)$$

Intrarea G (Strobe) activează circuitul dacă G=0, altfel ieșirile sunt menținute la 0.



Intrări				Ieșire
G (Strobe)	S (Select)	A_i	B_i	Y_i
1	X	X	X	0
0	0	0	X	0
0	0	1	X	1
0	1	X	0	0
0	1	X	1	1

Figura 7. 3 Multiplexorul 74157: simbolul (stânga) și tabelul de adevăr (dreapta)

7.2.2 Circuite care realizează operații aritmetice și logice

Operațiile aritmetice de bază sunt implementate cu sumatoare și scăzătoare. Circuitul 74283, prezentat în Figura 7. 4, realizează adunarea pe 4 biți a două numere binare $A_{4:1}$ și $B_{4:1}$. Rezultatul este calculat pe ieșirile $S_{4:1}$. Circuitul prezintă o intrare de transport (Carry In), la nivelul biților de rang 0, denumită C_0, și o ieșire de transport (Carry Out), de rang 4, denumită C_4, cu rol de semnalizare a depășirii. Liniile de transport se pot folosi la implementarea de sumatoare pe un număr extins de biți, pentru cascadarea mai multor circuite 74283. **Notă**: Activarea intrării C_0 are ca efect incrementarea rezultatului cu valoarea 1. Operația aritmetică efectuată de sumatorul 74283 este:

$$(C_4S_4S_3S_2S_1)_2 = (A_4A_3A_2A_1)_2 + (B_4B_3B_2B_1)_2 + (000C_0)_2 \quad (7.\ 2)$$

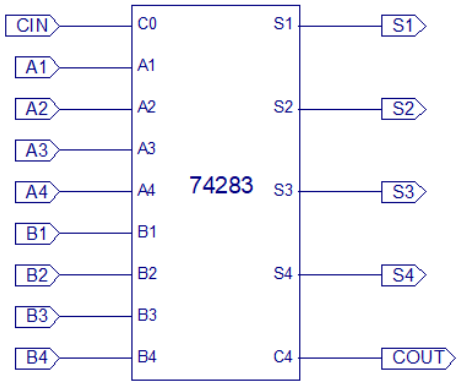


Figura 7. 4 Sumatorul 74283 pe 4 biți

7.2.2.1 Implementarea unui sumator-scăzător pe 4 biți

La reprezentarea în Complement față de 2, operația de scădere se poate realiza prin adunarea primului termen cu complementul față de 2 al celui de-al doilea termen, care se poate exprima la rândul său prin complementul față de 1: $A - B = A + \overline{\overline{B}} = A + \overline{B} + 1$. La numere pe 4 biți expresia devine:

$$(A_4A_3A_2A_1)_2 - (B_4B_3B_2B_1)_2 = (A_4A_3A_2A_1)_2 + (\overline{B}_4\overline{B}_3\overline{B}_2\overline{B}_1)_2 + (0001)_2 \quad (7.\ 3)$$

În algebra booleană, pentru o expresie ȩ avem relațiile: ȩ ⊕ 0 = ȩ și ȩ ⊕ 1 = $\overline{ȩ}$. În consecință, operațiile de adunare și scădere pot fi exprimate unitar, cu ajutorul XOR:

$$\begin{cases}(A_4A_3A_2A_1)_2 + (B_4B_3B_2B_1)_2 = (A_4A_3A_2A_1)_2 + (B_4B_3B_2B_1 \oplus 0000)_2 + (0000)_2\\ (A_4A_3A_2A_1)_2 - (B_4B_3B_2B_1)_2 = (A_4A_3A_2A_1)_2 + (B_4B_3B_2B_1 \oplus 1111)_2 + (0001)_2\end{cases}$$

Cele 2 operații se pot contopi în expresia $(C_4S_4S_3S_2S_1)_2 = (A_4A_3A_2A_1)_2 + (B_4B_3B_2B_1 \oplus \text{Sel Sel Sel Sel})_2 + (0\ 0\ 0\ \text{Sel})_2$, unde **Sel=0 pentru adunare** și **Sel=1 pentru scădere**. Expresia poate fi implementată cu sumatorul 74283 și cu 4 porți XOR, în felul următor:

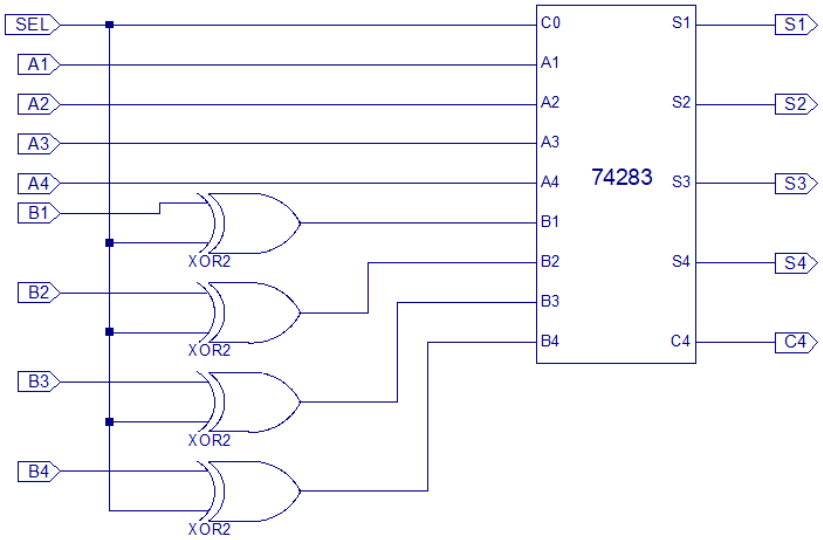


Figura 7. 5 Implementarea unui sumator-scăzător pe 4 biți, folosind circuitul 74283

7.2.2.2 Implementarea unui sumator-scăzător pe 8 biți

Pentru proiectarea pe 8 biți, expresia se poate extinde în felul următor:

$$(C_8S_8S_7S_6S_5S_4S_3S_2S_1)_2 = (A_8A_7A_6A_5A_4A_3A_2A_1)_2 + \\ + (B_8B_7B_6B_5B_4B_3B_2B_1 \oplus \text{Sel Sel Sel Sel Sel Sel Sel Sel})_2 + (0\ 0\ 0\ 0\ 0\ 0\ 0\ \text{Sel})_2 \quad (7.4)$$

Se vor utiliza 2 sumatoare 74283 cascadate și 8 porți XOR. Unul dintre sumatoare va realiza calculul pe biții de rang 1:4, cu semnalul Sel conectat la intrarea C_0. Cel de-al doilea sumator va realiza calculul pe biții de rang 5:8, cu transportul C_4 de la sumatorul anterior conectat la C_0:

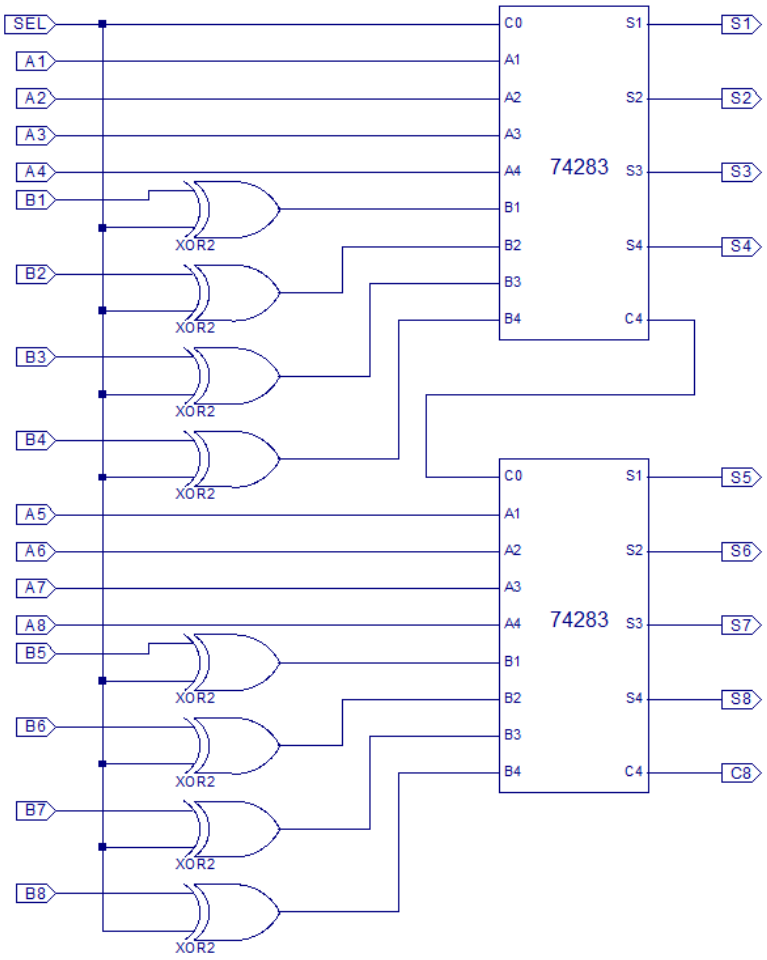
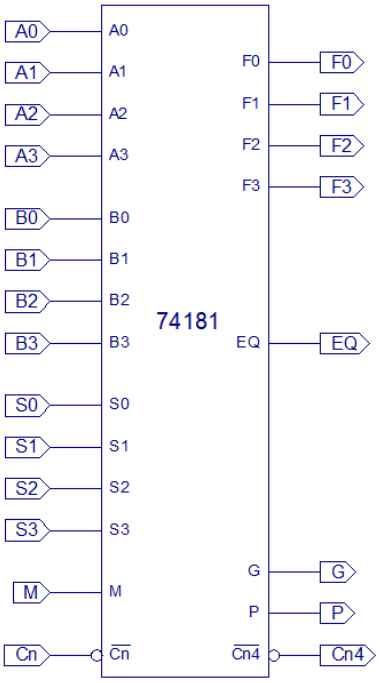


Figura 7. 6 Implementarea unui sumator-scăzător pe 8 biți prin cascadare

7.2.2.3 Unități aritmetice-logice

Circuitul 74181 (Figura 7. 7) implementează operații aritmetice-logice pe 4 biți între operanzii $A_{3:0}$ și $B_{3:0}$, reprezentați în complement față de 2. Operația efectuată se stabilește cu ajutorul codului aplicat pe intrările $S_{3:0}$. Pentru fiecare cod se pot realiza două categorii de operații: o operație logică bit-cu-bit, dacă intrarea M = 1 sau una aritmetică, dacă M = 0. Rezultatul este calculat pe ieșirile $F_{3:0}$. Semnalele de transport C_n (C_{in}) și C_{n4} (C_{out}) sunt active pe 0. Acestea ajută la extinderea numărului de biți prin cascadare. Ieșirea EQ (sau A=B) testează egalitatea între $A_{3:0}$ și $B_{3:0}$ la efectuarea operației A minus B minus 1 ($S_{3:0}$=0110). Setul complet de operații poate fi urmărit în tabelul următor:



Intrări de selecție				Operații efectuate	
S_3	S_2	S_1	S_0	Logice M = 1	Aritmetice M = 0, C_n = 1
0	0	0	0	$\overline{A}$	A
0	0	0	1	$\overline{A+B}$	$A+B$
0	0	1	0	$\overline{A}\cdot B$	$A+\overline{B}$
0	0	1	1	0	minus 1
0	1	0	0	$\overline{A\cdot B}$	A plus $(A\cdot\overline{B})$
0	1	0	1	$\overline{B}$	$(A+B)$ plus $(A\cdot\overline{B})$
0	1	1	0	$A\oplus B$	A minus B minus 1
0	1	1	1	$A\cdot\overline{B}$	$(A\cdot B)$ minus 1
1	0	0	0	$\overline{A}+B$	A plus $(A\cdot B)$
1	0	0	1	$\overline{A\oplus B}$	A plus B
1	0	1	0	B	$(A+\overline{B})$ plus $(A\cdot B)$
1	0	1	1	$A\cdot B$	$(A\cdot B)$ minus 1
1	1	0	0	1	A plus A
1	1	0	1	$A+\overline{B}$	$(A+B)$ plus A
1	1	1	0	$A+B$	$(A+\overline{B})$ plus A
1	1	1	1	A	A minus 1

Figura 7. 7 Circuitul 74181: simbolul (stânga) și operațiile pe care le efectuează (dreapta)
(+ = SAU, · = ȘI, ⊕ = XOR, plus = adunare, minus = scădere)

7.2.3 Afișarea cifrelor zecimale

Placa de dezvoltare prezintă 8 afișoare cu 7 segmente pentru afișarea în baza 10 (Figura 7. 8) [1]. Fiecare afișor poate afișa o cifră zecimală, construită din cele 7 segmente, prin activarea sau inactivarea lor (Figura 7. 9). Segmentele sunt controlate de 7 semnale, denumite *catozi*, indexate de la A la G. Catozii sunt partajați de cele 8 afișoare. Fiecare afișor are un semnal de activare, denumit *anod*. Anozii sunt numerotați de la 0 la 7, fiind asociați afișoarelor de la dreapta la stânga. **Notă**: Datorită catozilor partajați, pe afișoarele active va apărea aceeași cifră, dar există tehnici prin care se pot afișa cifre diferite pe fiecare afișor, astfel încât să se poată vizualiza valori cu mai multe cifre zecimale. Catozii și anozii sunt activi pe 0. Implicit, catozii au valoarea 1 și anozii au valoarea 0, ceea ce înseamnă că afișoarele sunt active, dar nu afișează nimic pe segmente, fiind inactive.

Figura 7. 8 Afișoarele cu 7 segmente de pe placa de dezvoltare

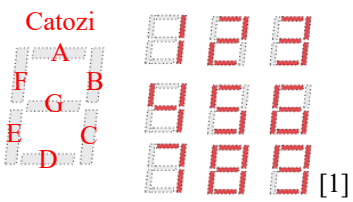


Figura 7. 9 Poziția segmentelor pe afișor și configurarea cifrelor zecimale

Circuitul 7447 este un decodificator BCD-7 segmente, care primește valori binare pe 4 biți și activează catozii care alcătuiesc cifra zecimală corespunzătoare, în vederea afișării acesteia pe afișorul cu 7 segmente. Asocierea este prezentată în Figura 7. 10.

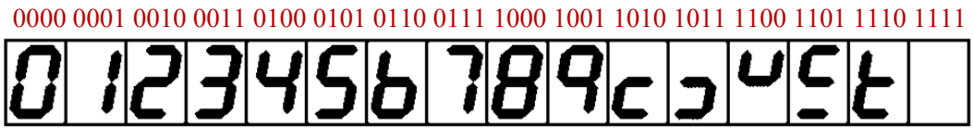


Figura 7. 10 Asocierea codurilor binare la simbolurile afișate

Ieșirile decodificatorului asociate catozilor A-G funcționează în logica negativă (Figura 7. 11). Intrările $\overline{LT}$, $\overline{RBI}$, $\overline{BI}/\overline{RBO}$ sunt rezervate pentru configurații de test, motiv pentru care, la o funcționare normală, acestea se dezactivează prin conectare la VCC.

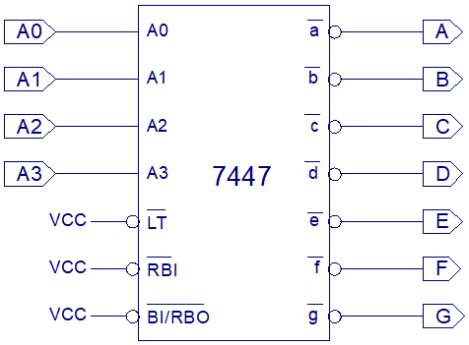


Figura 7. 11 Decodificatorul BCD-7 segmente 7447

În Logisim, afișorul pe 7 segmente se află în librăria *Input*/*Output* și funcționează în logica pozitivă, deci ieșirile decodificatorului 7447 trebuie inversate.

În Project Navigator, în fișierul .ucf, catozii și anozii se află într-o secțiune separată:

```
## 7 segment display
NET "A" LOC=T10 | IOSTANDARD=LVCMOS33; # cat a
NET "B" LOC=R10 | IOSTANDARD=LVCMOS33; # cat b
. . .
```

7.3 Activități practice

1. Implementați și testați pe placă multiplexorul 74157 cu calea de date pe 4 biți.
2. Implementați și testați pe placă sumatorul 74283 pe 4 biți.
3. Implementați și testați pe placă un sumator-scăzător implementat cu circuitul 74283.
4. Implementați și testați pe placă decodificatorul BCD-7 segmente 7447.
5. Implementați și testați în Logisim un sumator pe 8 biți cu sumatoare 74283.
6. Implementați și testați în Logisim un sumator-scăzător pe 8 biți cu sumatoare 74283.
7. Implementați și testați în Logisim unitatea aritmetică-logică 74181 pe 4 biți.

7.4 Bibliografie

[1] Digilent, “Nexys A7 Reference Manual”, [Online]. Available: https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual

8 Circuite logice secvențiale – bistabile

8.1 Obiective

Se studiază circuitele basculante bistabile uzuale, implementarea lor cu porți logice fundamentale și funcționalitatea acestora sub acțiunea comenzilor primite pe semnalele de intrare. Se analizează efectul comenzilor asincrone și al celor sincrone pe nivel sau pe frontul semnalului de tact, precum și avantajele și dezavantajele care decurg din modul în care se realizează sincronizarea. Se studiază metoda de realizare a unui tip de bistabil cu un alt tip de bistabil și porți logice adiționale.

8.2 Considerații teoretice

Circuitele logice secvențiale sunt automate de ordinul 1, care prezintă o stare internă și ieșiri ce depind de această stare și de comenzile primite pe intrări. Din categoria circuitelor logice secvențiale fac parte circuitele basculante bistabile, numite pe scurt *bistabile*. Acestea sunt cele mai elementare structuri secvențiale, care au două stări distincte, reprezentate prin valorile 0 sau 1, așadar sunt capabile să memoreze 1 bit. Starea circuitului se poate menține oricât de mult, dar poate să se schimbe sub acțiunea comenzilor pe care circuitul le primește pe intrările sale. Bistabilul prezintă 2 ieșiri complementare, reprezentând starea internă și inversul ei.

În funcție de reacția la comenzile primite pe semnalele de intrare, bistabilele pot fi a*sincrone* sau *sincrone*. Cele asincrone reacționează imediat la comenzile primite pe intrări, iar cele sincrone reacționează la momente bine determinate de un semnal de sincronizare numit *tact* sau *ceas*. Tactul este un semnal oscilatoriu între 0 și 1, care se repetă cu o perioadă T și are frecvența *f*=1/T. Bistabilele sincrone pot să prezinte și intrări de comandă asincrone, care au efect imediat și prioritar față de intrările sincrone și sunt folosite, în general, la inițializarea stării bistabilului cu valoarea 0 sau 1.

8.2.1 Bistabilul RS asincron

Bistabilul RS asincron are 2 intrări de comandă asincrone: R (reset) și S (set). Funcționarea circuitului este prezentată în Tabelul 8. 1.

Tabelul 8. 1 Funcționarea RS asincron (Q^t = starea curentă, Q^{t+1} = starea viitoare)

S	R	Q^{t+1}
0	0	Q^t
0	1	0
1	0	1
1	1	*

Implementarea cu porți ȘI-NU se poate obține din expresiile în forma disjunctivă minimă (FDM) pentru ieșiri, aplicând dubla negație și relațiile lui De Morgan (Figura 8. 1).
Notă: Funcționalitatea comenzilor Sn, Rn pentru circuitul obținut este în logica negativă.

Q^t S R	Q^{t+1}	$\overline{Q^{t+1}}$
0 0 0	0	1
0 0 1	0	1
0 1 0	1	0
0 1 1	X	X
1 0 0	1	0
1 0 1	0	1
1 1 0	1	0
1 1 1	X	X

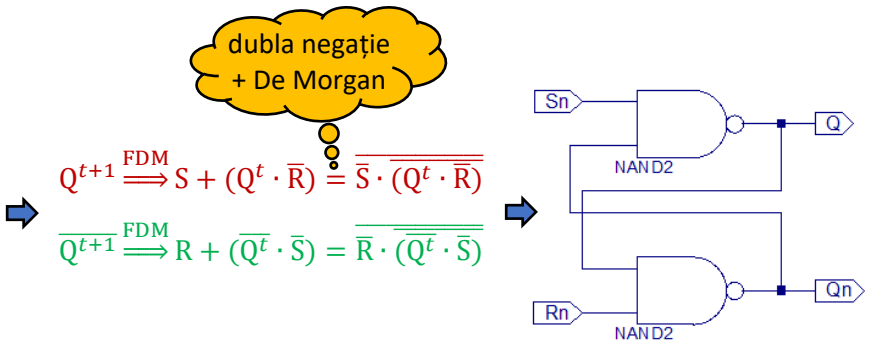
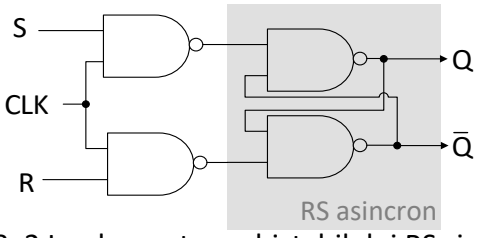


Figura 8. 1 Implementarea bistabilului RS asincron cu porți ȘI-NU (Sn=$\overline{S}$, Rn=$\overline{R}$, Qn=$\overline{Q}$)

8.2.2 Bistabilul RS sincron

Bistabilul RS sincron prezintă semnalul de tact CLK și 2 intrări de comandă S, R, sincrone cu semnalul de tact, în logica pozitivă. Efectul intrărilor este identic cu cel din Tabelul 8. 1, doar că se manifestă atunci când CLK=1. Implementarea cu porți ȘI-NU se obține prin extinderea variantei asincrone, conform schemei din Figura 8. 2.



Q^t	Q^{t+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Figura 8. 2 Implementarea bistabilului RS sincron cu porți ȘI-NU (stânga) și tabelul de excitație al acestuia (dreapta)

Ținând cont de efectul comenzilor sincrone se poate deduce *tabelul de excitație*, redat în Figura 8. 2 – dreapta. Acesta precizează valorile care trebuie să fie pe intrările de comandă sincrone astfel încât să se obțină o anumită evoluție a stării bistabilului. Aceste valori se menționează pentru toate combinațiile posibile de stare curentă și stare viitoare. De exemplu, primul rând din tabel precizează faptul că dacă se dorește păstrarea stării curente Q^t=0, astfel încât Q^{t+1}=0, atunci trebuie ca S=0 și R=X (R poate fi 0 sau 1). Conform rândului al 2-lea, pentru tranziția din Q^t=0 în Q^{t+1}=1, trebuie ca S=0 și R=1.

8.2.3 Bistabilul D (data/delay)

Bistabilul D are o intrare sincronă de comandă, denumită D. În momentul sincronizării cu semnalul de tact CLK starea bistabilului preia valoarea prezentă pe D. Simbolul bistabilului și implementarea cu porți ȘI-NU sunt prezentate în Figura 8. 3.

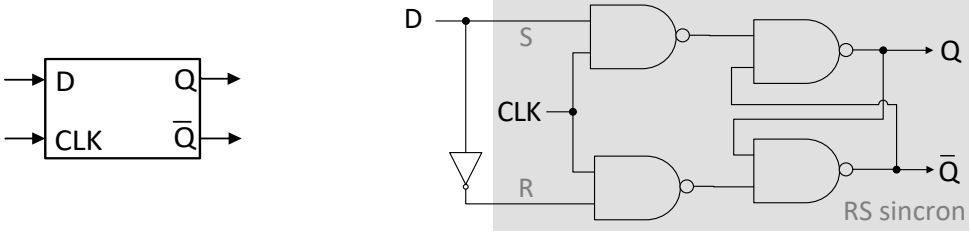


Figura 8. 3 Simbolul bistabilului D (stânga) și implementarea cu ȘI-NU (dreapta)

Notă: Un bistabil D se obține dintr-un bistabil RS sincron cu intrările S=D și R=$\overline{D}$.

Tabelele de adevăr și de excitație (Tabelul 8. 2) evidențiază relația $Q^{t+1} = D$ dintre starea viitoare și intrarea D. Expresia poartă denumirea de *ecuația caracteristică*.

Tabelul 8. 2 Tabelele de adevăr (stânga) și de excitație (dreapta) ale bistabilului D

CLK	D	Q^{t+1}
0	X	Q^t
1	0	0
1	1	1

Q^t	Q^{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Comportamentul bistabilului este evidențiat în diagrama de timp următoare. Intrarea D are efect cât timp CLK=1 – sincronizare pe *nivelul pozitiv* – și este blocată când CLK=0. Cele 2 intervale sunt marcate cu verde, respectiv roșu, în diagramă.

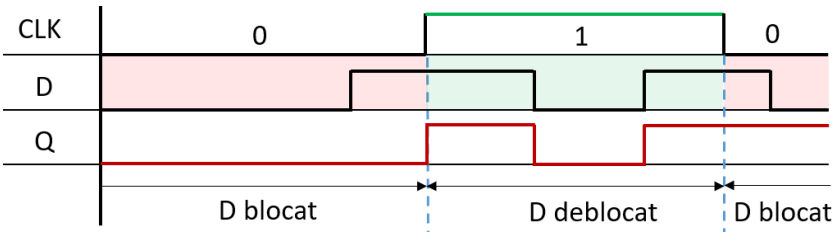
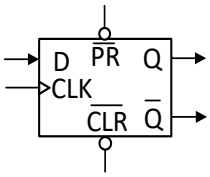


Figura 8. 4 Funcționarea bistabilului D cu sincronizare pe nivelul pozitiv

Notă: Există și varianta de bistabil D cu sincronizare pe *nivelul negativ*, efectul intrării D fiind valabil când CLK=0.

Circuitul TTL 7474 implementează bistabilul D cu 2 intrări asincrone suplimentare, de set ($\overline{PR}$) și reset ($\overline{CLR}$), funcționale în logica negativă. Comenzile $\overline{PR}$ (preset) și $\overline{CLR}$ (clear) au efect imediat și sunt prioritare intrării D. Când $\overline{PR} = \overline{CLR} = 1$ (sunt inactive) intrarea D are efect pe *frontul ascendent* (rising edge). Frontul ascendent îl reprezintă comutarea semnalului CLK din 0 în 1. Bistabilele care sunt sincrone pe nivel mai poartă denumirea de *latch*, iar cele care comută pe front se mai numesc *flip-flop*. Funcționarea este prezentată în diagrama de timp din Figura 8. 5, care evidențiază efectul intrării D pe frontul ascendent, cu comutarea stării bistabilului, dacă este cazul, imediat după front.



$\overline{PR}$	$\overline{CLR}$	CLK	D	Q^{t+1}
0	0	X	X	*
0	1	X	X	1
1	0	X	X	0
1	1	0	X	Q^t
1	1	↑	D	D

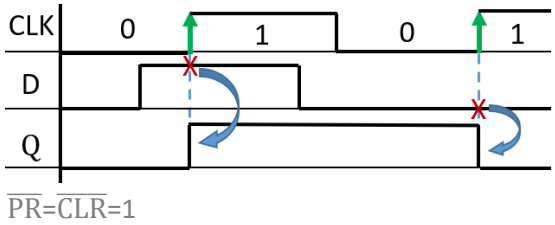


Figura 8. 5 Bistabilul D 7474 (stânga), tabelul de adevăr (mijloc) și sincronizarea pe frontul ascendent (dreapta)

Observație: În Project Navigator, în cadrul proiectului *ttl_env*, toate circuitele TTL secvențiale prezintă o intrare suplimentară denumită CKF (ClocK-Fpga), care trebuie

conectată la semnalul de ceas al plăcii cu același nume. Pentru aceasta se va introduce (decomenta) în fișierul .ucf secțiunea legată de semnalul de ceas:

```
## Clock signal
NET "CKF" LOC = "E3" | IOSTANDARD = "LVCMOS33";
NET "CKF" TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100 MHz HIGH 50%;
```

Important: Pe plăcile de dezvoltare semnalul de ceas utilizator va fi generat cu ajutorul butoanelor. La fiecare apăsare și relaxare a acestora se pot genera mai multe impulsuri nedorite, în loc de unul singur, datorate uzării mecanice. Pentru eliminarea impulsurilor false este necesară utilizarea circuitului DEBOUNCER (Figura 8. 6), disponibil în librăria proiectului *ttl_env*. Pe intrarea BT_IN se aplică semnalul de la buton, iar pe ieșirea BT_OUT se obține semnalul filtrat. Semnalul CKF se conectează la ceasul plăcii. Circuitul de test pentru bistabilul 7474 este prezentat în figura următoare:

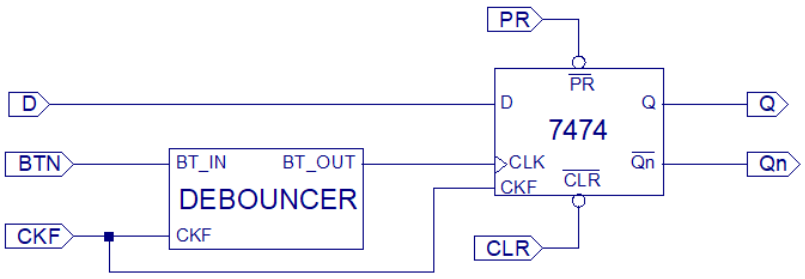


Figura 8. 6 Circuitul de test în Project Navigator pentru bistabilul 7474

În fișierul .ucf există o secțiune dedicată declarării butoanelor:

```
## Buttons
NET "BTN" LOC=N17 | IOSTANDARD=LVCMOS33; # center
```

În Logisim, semnalul CKF nu apare, fiind specific plăcilor FPGA, și nici folosirea unui DEBOUNCER nu este necesară, fiindcă simulatorul nu reproduce zgomotele mecanice.

Notă: Există și bistabile D cu sincronizare pe *frontul descendent* (falling edge), la comutarea CLK din 1 în 0. Soluțiile de implementare a bistabilelor cu comutare pe front variază. O posibilitate de implementare a bistabilului D flip-flop cu sincronizare pe frontul descendent este prin conectarea, în configurație *master-slave*, a două bistabile RS cu sincronizare pe nivel. Intrarea D este conectată la bistabilul master, iar ieșirea Q provine de la bistabilul slave. Ieșirile bistabilului master sunt conectate la intrările bistabilului slave. Figura 8. 7 prezintă implementarea cu porți ȘI-NU a bistabilului D master-slave. Semnalul CLK este conectat direct la master și inversat la slave, ceea ce provoacă o comutare alternativă, în timp, a bistabilelor. **Funcționare**: O comandă aplicată pe intrarea D are efect asupra bistabilului master dacă CLK=1, dar nu se va propaga pe bistabilul slave, decât după ce CLK=0. Efectul apare imediat după frontul descendent. Diagrama de timp din Figura 8. 8 prezintă propagarea semnalelor în funcție de valoarea semnalului CLK și scoate în evidență alternanța perioadelor de blocare și deblocare. Se observă că intrarea D activată înainte de frontul descendent are efect pe ieșirea Q imediat după front.

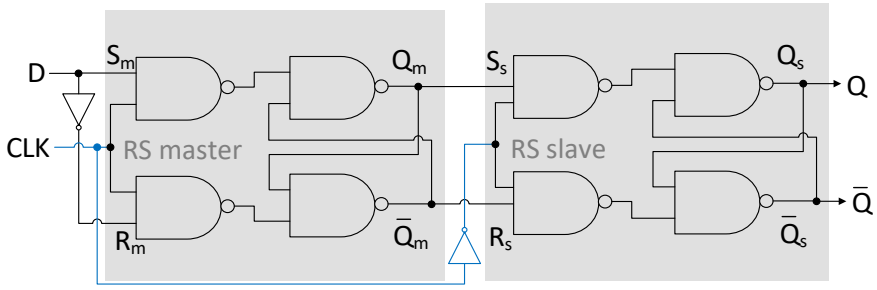


Figura 8. 7 Bistabilul D master-slave implementat cu porți ȘI-NU

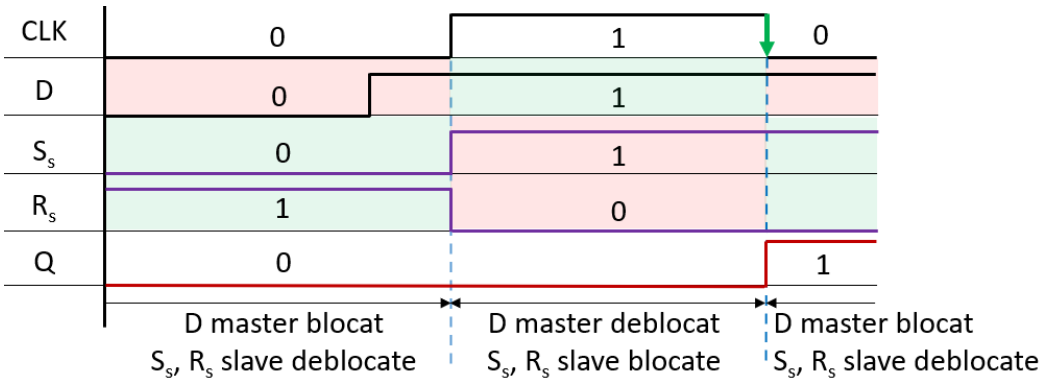


Figura 8. 8 Funcționarea în timp a bistabilului D master-slave

8.2.4 Bistabilul JK

Bistabilul JK prezintă intrările sincrone J și K, și are funcționalitatea bistabilul RS sincron (J≈S, K≈R), cu deosebirea că dacă J=K=1, starea bistabilului se inversează. Tabelele de adevăr și de excitație sunt următoarele:

Tabelul 8. 3 Tabelele de adevăr (stânga) și de excitație (dreapta) ale bistabilului JK

CLK	J	K	Q^{t+1}
0	X	X	Q^t
1	0	0	Q^t
1	0	1	0
1	1	0	1
1	1	1	$\overline{Q^t}$

Q^t	Q^{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Implementarea variantei latch cu porți ȘI-NU presupune extinderea bistabilului RS sincron cu expresiile: $S = \overline{Q^t} \cdot J$ și $R = Q^t \cdot K$ (Figura 8. 9 – stânga). Porțile ȘI se pot contopi cu porțile ȘI-NU, și se obține un circuit cu 2 niveluri logice (Figura 8. 9 – dreapta).

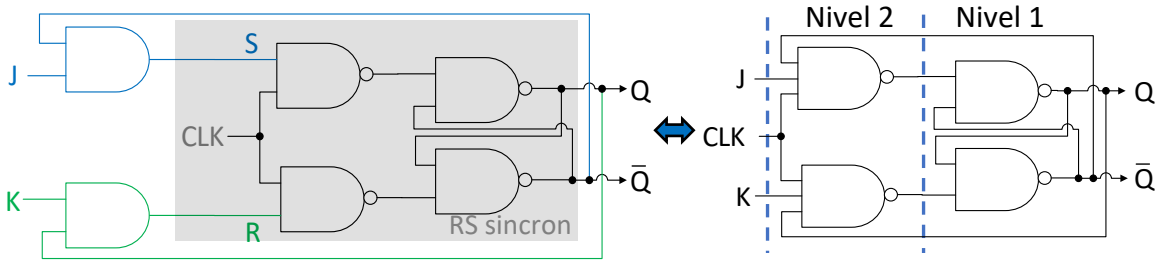


Figura 8. 9 Bistabilul JK latch implementat cu porți logice

În varianta master-slave bistabilul JK este sincron pe frontul descendent și presupune conectarea reacțiilor de la ieșirile bistabilului slave pe intrările bistabilului master, ca în Figura 8. 10.

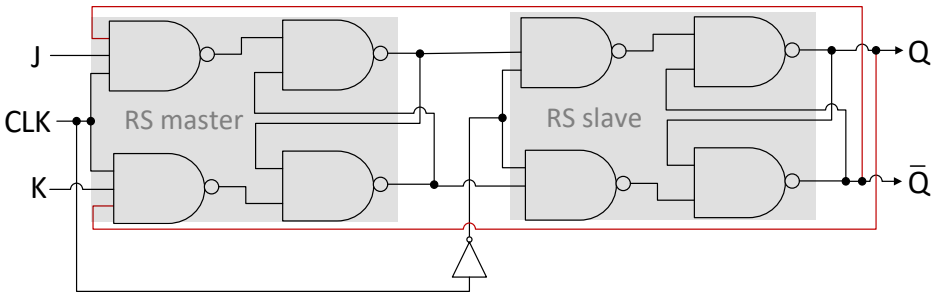


Figura 8. 10 Bistabilului JK master-slave implementat cu porți ȘI-NU

Avantajul sincronizării pe front îl reprezintă faptul că, atunci când J=K=1, are loc o singură comutare în decursul unei perioade de tact, la momentul frontului (Figura 8. 11). La sincronizarea pe nivel pot avea loc mai multe comutări, în funcție de durata perioadei de ceas, ceea ce introduce o situație de incertitudine și limitări de utilizare.

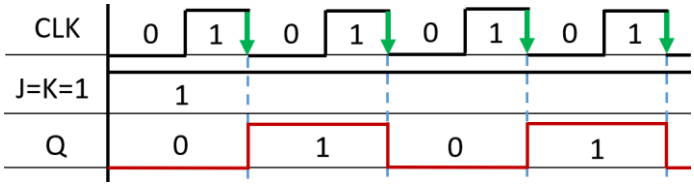


Figura 8. 11 Bistabilul JK master-slave comută o singură dată în decursul unei perioade

Circuitele 7473 și 7476 implementează bistabile JK master-slave. Bistabilul JK 7473 are o intrare asincronă de reset $\overline{\text{CLR}}$ (clear), iar JK 7476 are o intrare asincronă de reset $\overline{\text{CLR}}$ (clear) și una de set $\overline{\text{PR}}$ (preset). Intrările asincrone sunt funcționale în logica negativă și prioritare față de cele sincrone. Circuitele sunt prezentate în Figura 8. 12.

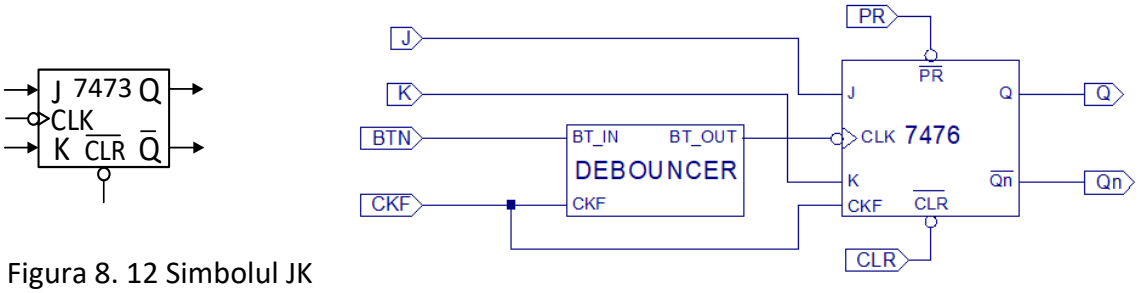
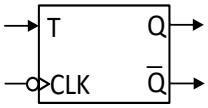


Figura 8. 12 Simbolul JK 7473 (stânga) și circuitul de test pentru JK 7476 (dreapta)

8.2.5 Bistabilul T (toggle)

Bistabilul T are o singură intrare sincronă, denumită T. Dacă T=0 bistabilul își menține starea curentă, iar dacă T=1 bistabilul își inversează starea (toggling). Bistabilul T există atât în varianta latch cât și flip-flop. În Figura 8. 13 este prezentat bistabilul în varianta flip-flop cu sincronizare pe frontul descendent. Din tabelul de adevăr reiese *ecuația caracteristică*: $\text{Q}^{t+1} = \text{T} \oplus \text{Q}^{t}$. Comportamentul său este identic cu al unui bistabil JK, ale cărui intrări de comandă sunt conectate la semnalul T. În Figura 8. 13 – dreapta este prezentată implementarea cu un bistabil JK 7476.



CLK	T	Q^{t+1}
0	X	Q^t
1	X	Q^t
↓	0	Q^t
↓	1	$\overline{Q^t}$

Q^t	Q^{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

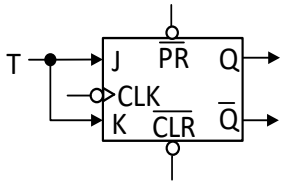


Figura 8. 13 Bistabilul T flip-flop sincron pe frontul descendent (stânga), tabelele de adevăr și de excitație (mijloc) și implementarea cu un bistabil JK 7476 (dreapta)

8.2.6 Implementarea unui bistabil folosind un alt bistabil

Atunci când se dorește implementarea unui bistabil de tip A cu un bistabil de tip B se utilizează tabelele de excitație ale acestora, astfel:

1. Se asociază perechile pentru care stările curentă și următoare (Q^t, Q^{t+1}) sunt identice și se creează un tabel de adevăr în care starea curentă Q^t și intrările bistabilului de tip A apar în partea stângă. În partea dreaptă apar doar intrările bistabilului de tip B.
2. Din tabelul de adevăr se determină expresiile intrărilor bistabilului B.

Exemple de implementare a unui bistabil cu un alt tip de bistabil:

a) Implementarea D cu JK

Implementarea unui bistabil D cu JK folosește tabelele de excitație ale lui D și JK:

Q^t	Q^{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Q^t	Q^{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Prin împerecherea perechilor (Q^t, Q^{t+1}) comune în cele 2 tabele de excitație se obține tabelul de adevăr de mai jos, care exprimă intrările J, K în funcție de starea Q^t și intrarea D. Prin minimizare la Forma Disjunctivă Minimă cu diagrame Karnaugh se obțin expresiile intrărilor bistabilului JK: $J = D$ și $K = \overline{D}$. Implementarea corespunzătoare acestor expresii este prezentată în Figura 8. 14.

Q^t	D	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

$J = D$
$K = \overline{D}$

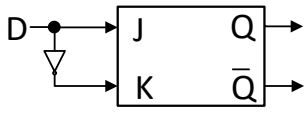


Figura 8. 14 Tabelul de adevăr pentru intrările J, K și implementarea bistabilului D cu JK

b) Implementarea D cu T

Se folosesc tabelele de excitație ale lui D și T din care reiese tabelul de adevăr pentru intrarea T și expresia acestuia pe baza căreia se realizează schema logică:

Q^t	Q^{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Q^t	Q^{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Q^t	D	T
0	0	0
0	1	1
1	0	1
1	1	0

$$T = D \oplus Q^t$$

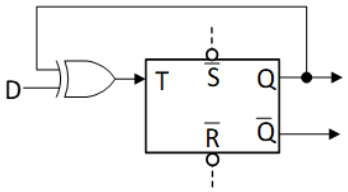


Figura 8. 15 Implementarea bistabilului D cu bistabilul T – pași de rezolvare

c) Implementarea T cu JK

Pe baza tabelelor de excitație ale lui T și JK se generează tabelul de adevăr pentru intrările J și K din care se deduc expresiile logice ale acestora și circuitul rezultat:

Q^t	Q^{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Q^t	Q^{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Q^t	T	J	K
0	0	0	X
0	1	1	X
1	1	X	1
1	0	X	0

$$J = T$$
$$K = T$$

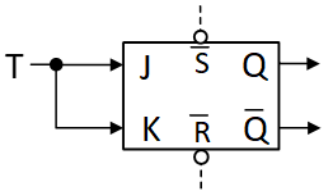


Figura 8. 16 Implementarea bistabilului T cu bistabilul JK – pași de rezolvare

8.3 Activități practice

1. Implementați cu porți ȘI-NU și testați pe placă bistabilul RS asincron.
2. Implementați pe placă bistabilul D 7474. Testați comenzile sincrone și asincrone.
3. Implementați pe placă bistabilul JK 7476. Testați comenzile sincrone și asincrone.
4. Implementați cu porți ȘI-NU și testați în Logisim bistabilul RS sincron.
5. Implementați în Logisim un bistabil T folosind bistabilul JK 7476. Testați comenzile sincrone și asincrone.
6. Implementați în Logisim bistabilul JK 7473. Testați comenzile sincrone și asincrone.
7. Implementați în Logisim un bistabil JK folosind bistabilul D 7474. Inactivați comenzile asincrone prin conectare la VCC și testați comenzile sincrone.

9 Circuite logice secvențiale – numărătoare

9.1 Obiective

Se definesc numărătoarele și se studiază proprietățile lor generale. Se analizează avantajele și dezavantajele implementării numărătoarelor asincrone și sincrone cu bistabile JK în configurație *toggle*. Se studiază numărătoare directe și reversibile pe 4 biți, disponibile sub formă de circuite integrate, comportamentul și particularitățile de funcționare ale acestora.

9.2 Considerații teoretice

Numărătoarele sunt circuite logice secvențiale care contorizează numărul de impulsuri aplicate pe intrarea de ceas. Numărătoarele sunt implementate cu bistabile. Numărul de bistabile determină numărul de biți pe care se face numărarea. Numărul de valori din secvența de numărare definește *capacitatea numărătorului*. Un numărător este *asincron*, dacă bistabilele din componența sa comută la momente diferite, și este *sincron*, dacă bistabilele comută simultan cu impulsul de ceas. În funcție de direcția de numărare, numărătoarele pot fi *directe*, *inverse* sau *reversibile*. În general, numărarea directă este crescătoare, cea inversă este descrescătoare, iar numărătoarele reversibile pot număra în ambele direcții. Numărarea poate fi *binară*, *binar-zecimală* sau *modulo p*. La numărarea *modulo p* secvența de numărare conține p valori ($p<2^n$) pe n biți.

9.2.1 Numărătorul binar asincron direct

Numărătorul binar asincron direct are structura cea mai simplă, bazată pe bistabile JK în configurație *toggle* (J=K=1), fără alte circuite adiționale. Comutarea este asincronă deoarece, exceptând bistabilul de rang 0, care este conectat la CLK, celelalte bistabile au ca semnal de ceas ieșirea bistabilului de rang inferior. Schema logică de implementare a unui numărător pe 3 biți, folosind 3 bistabile JK, cu sincronizare pe frontul descendent, este prezentată în Figura 9. 1. Extinderea la un număr mai mare de biți se poate realiza prin adăugarea de bistabile adiționale. **Notă**: Comanda $\overline{RST}$ activă pe 0, realizează resetarea asincronă a numărătorului prin aducerea imediată la valoarea 0.

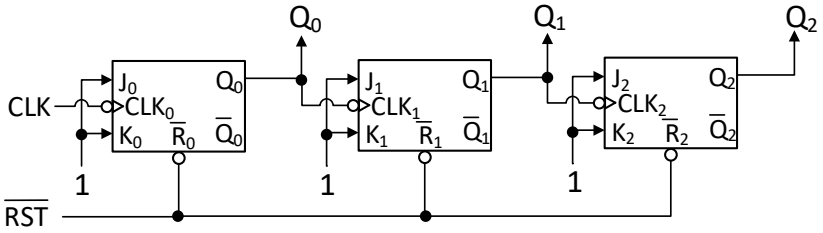


Figura 9. 1 Implementarea numărătorului binar asincron direct, pe 3 biți, cu bistabile JK

Diagrama de timp a funcționării numărătorului asincron, prezentată în Figura 9. 2, evidențiază valoarea pe ieșirile numărătorului la fiecare impuls de ceas. Numărarea se realizează în bucla 0-7. Numărarea avansează la fiecare front descendent al CLK. Se

observă faptul că un front descendent generat pe ieșirea unui bistabil determină comutarea bistabilului de rang superior. Tranziția de la 111_2 la 000_2 este cea mai lentă deoarece necesită comutarea consecutivă a tuturor bistabilelor. Perioada de ceas trebuie să depășească durata pentru tranziția cea mai lentă. Creșterea numărului de bistabile reprezintă un dezavantaj, deoarece limitează frecvența de lucru a numărătorului.

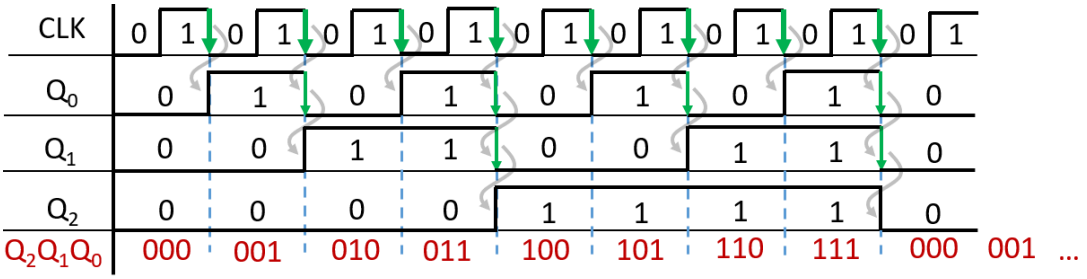


Figura 9. 2 Diagrama de funcționare a numărătorului binar asincron direct, pe 3 biți

Analizând formele de undă generate pe ieșiri se observă faptul că fiecare bistabil generează pe ieșirea sa un semnal de ceas cu perioada dublă față de cel de rang inferior. Bistabilele JK în configurație *toggle* funcționează ca *divizoare de frecvență* a semnalului primit pe intrarea de ceas. Raportat la semnalul CLK, cu cât crește rangul unui bistabil, cu atât crește factorul de divizare: Q_0, Q_1, Q_2 au factorul de divizare 2, 4, respectiv 8.

9.2.2 Numărătorul binar sincron serie direct

Problema frecvenței de lucru reduse la numărătorul asincron este parțial rezolvată de numărătorul sincron serie, la care comutarea bistabilelor JK este sincronă, prin conectarea acestora la semnalul CLK, ca în Figura 9. 3. Intrările sincrone J și K sunt conectate, dar pentru identificarea expresiilor aplicate pe acestea se analizează comportamentul fiecărui bit în parte, în cadrul buclei de numărare. Tabelul 9. 1 prezintă valorile înregistrate pe ieșirile bistabilelor în bucla 0-15.

Tabelul 9. 1 Ieșirile numărătorului pe 4 biți

Nr.	Q_3	Q_2	Q_1	Q_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
0	0	0	0	0

Analizând fiecare coloană Q_i în parte, se constată următoarele asocieri: bistabilul Q_0 comută la fiecare perioadă de ceas; bistabilul Q_1 comută când Q_0=1; bistabilul Q_2 comută când Q_1=1 și Q_0=1; bistabilul Q_3 comută când Q_2=1 și Q_1=1 și Q_0=1. Prin analogie, un bistabil comută întotdeauna atunci când toate cele de rang inferior au valoarea 1. Pe baza acestor observații se deduc următoarele expresii pentru intrările de comandă J_i, K_i:

$$\begin{aligned} J_0 = K_0 = 1 \\ J_1 = K_1 = Q_0 \\ J_2 = K_2 = Q_1 \cdot Q_0 \\ J_3 = K_3 = Q_2 \cdot Q_1 \cdot Q_0 \end{aligned} \qquad (9.\ 1)$$

Pentru implementarea operațiilor ȘI se utilizează numai porți cu 2 intrări, conectate serial, deoarece au structură mai simplă. Dezavantajul îl constituie creșterea timpului de propagare, odată cu creșterea numărului de porți conectate

serial, dacă se dorește extinderea numărului de biți. Timpul de propagare determină perioada de ceas, limitând astfel frecvența de lucru.

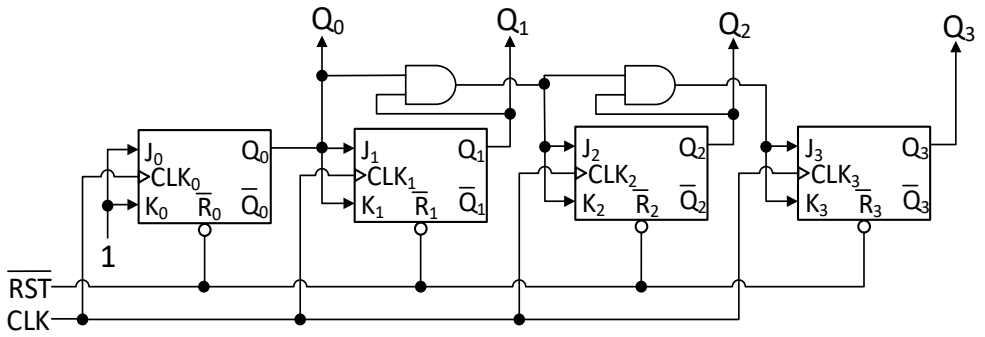


Figura 9. 3 Implementarea unui numărător sincron serie direct, pe 4 biți, cu bistabile JK

9.2.3 Numărătorul binar sincron paralel direct

Limitarea timpului de propagare pe porțile conectate serial are ca soluție utilizarea, în paralel, de porți ȘI cu mai mult de 2 intrări, ca în Figura 9. 4. Astfel, timpul de propagare se rezumă la întârzierea indusă de poarta ȘI cea mai complexă (cu cel mai mare număr de intrări), care este semnificativ mai mică decât cea corespunzătoare mai multor porți ȘI conectate în serie. În consecință, frecvența de lucru este mai puțin afectată, odată cu creșterea numărului de bistabile.

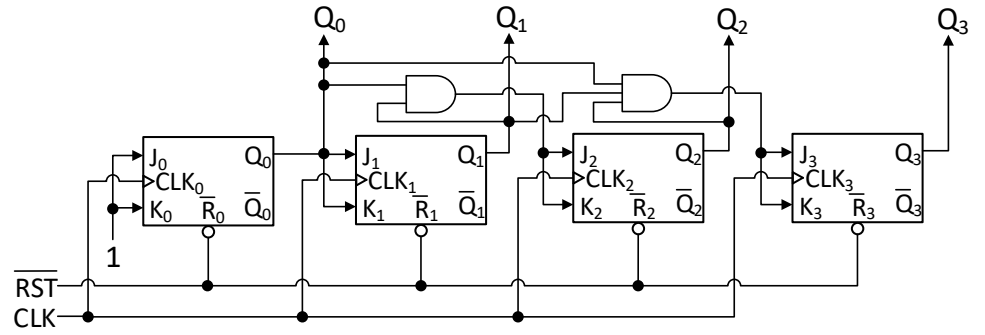
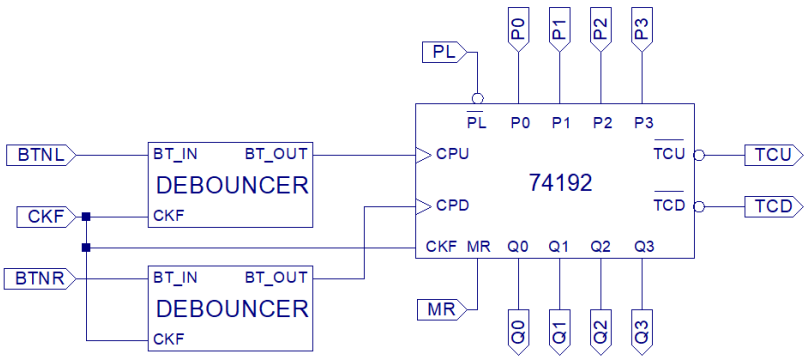


Figura 9. 4 Implementarea unui numărător sincron paralel direct, pe 4 biți

9.2.4 Numărătorul zecimal sincron reversibil 74192

Circuitul TTL 74192, din Figura 9. 5, este un numărător reversibil pe 4 biți, care are bucla de numărare 0-9, asociată cifrelor zecimale. Acesta prezintă comanda asincronă MR (Master Reset), de reset la 0, și intrarea asincronă $\overline{PL}$ (Parallel Load), de încărcare paralelă a valorii pe 4 biți de pe intrările $P_{3:0}$. Comanda de reset are prioritate față de comanda de încărcare paralelă, care funcționează în logica negativă. Valoarea curentă a numărătorului este disponibilă pe ieșirile $Q_{3:0}$. Rolul semnalelor este prezentat în tabelul din Figura 9. 5.

Pentru numărare directă, se aplică semnalul de ceas pe intrarea CP_U (Count Up) și valoarea 1 intrarea CP_D (Count Down). Pentru numărare inversă, se aplică semnalul de ceas pe CP_D și valoarea 1 pe CP_U.



MR	$\overline{PL}$	CP_U	CP_D	$Q_{3:0}^{t+1}$
1	X	X	X	$(0000)_2$
0	0	X	X	$P_{3:0}$
0	1	↑	1	$Q_{3:0}^{t}+1$
0	1	1	↑	$Q_{3:0}^{t}-1$
0	1	1	1	$Q_{3:0}^{t}$

Figura 9. 5 Testarea numărătorului 74192 (stânga) și tabelul de adevăr (dreapta)

La numărare directă, ieșirea $\overline{TC}_U$ (Terminal Count Up) semnalizează prin $\overline{TC}_U = 0$ (în logica negativă), atingerea valorii zecimale maxime $Q_{3:0}$=1001. Similar, la numărare inversă, ieșirea $\overline{TC}_D$ (Terminal Count Down) semnalizează prin $\overline{TC}_D = 0$, atingerea valorii minime $Q_{3:0}$=0000. **Notă**: Semnalizarea are loc în a doua parte a perioadei de ceas, precum în Figura 9. 6. Expresiile ieșirilor $\overline{TC}_U$ și $\overline{TC}_D$ sunt:

$$\overline{TC}_U = \overline{Q_3 \cdot Q_0 \cdot \overline{CP}_U}$$
$$\overline{TC}_D = \overline{\overline{Q_3} \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot \overline{Q_0} \cdot \overline{CP}_D} \quad (9.\ 2)$$

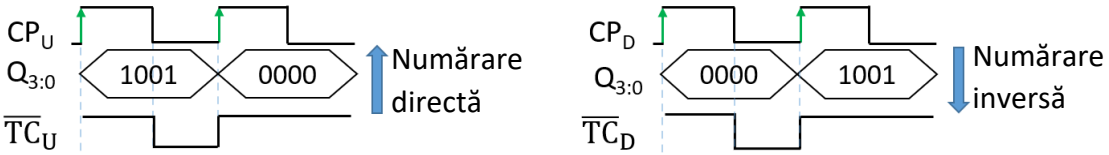
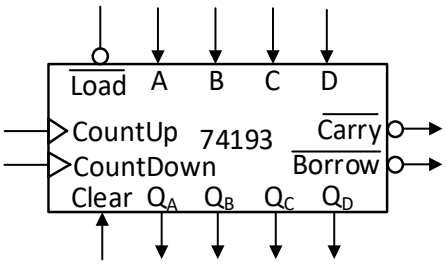


Figura 9. 6 Activarea indicatorilor la atingerea limitelor de numărare ale 74192

Numărătorul 74192 este implementat cu un mecanism de *autocorecție sincronă*: dacă valoarea curentă este în afara buclei 0-9, atunci va reveni la o valoare din cadrul acesteia, după câteva perioade de ceas, reluându-și funcționarea normală.

9.2.5 Numărătorul binar sincron reversibil 74193

Circuitul TTL 74193, din Figura 9. 7, reprezintă varianta binară a numărătorului 74192, având bucla de numărare 0-15. Intrările și ieșirile au denumire diferită, dar funcționalitate similară cu cele ale numărătorului 74192.



Clear	$\overline{Load}$	CountUp	CountDown	$Q_{D:A}^{t+1}$
1	X	X	X	$(0000)_2$
0	0	X	X	$(DCBA)_2$
0	1	↑	1	$Q_{D:A}^{t}+1$
0	1	1	↑	$Q_{D:A}^{t}-1$
0	1	1	1	$Q_{D:A}^{t}$

Figura 9. 7 Simbolul numărătorului 74193 (stânga) și tabelul de adevăr (dreapta)

Semnalele Clear și $\overline{Load}$ reprezintă comenzile asincrone de reset, respectiv de încărcare cu valoarea binară $DCBA_2$. Intrările CountUp și CountDown sunt dedicate pentru

semnalul de ceas folosit la numărare directă, respectiv inversă. Intrarea de ceas secundară trebuie conectată la 1. Valoarea numărătorului este disponibilă pe ieșirile $Q_{D:A}$. Conform diagramelor din Figura 9. 8, ieșirea $\overline{\text{Carry}}$ semnalizează valoarea maximă 15 la numărare directă și ieșirea $\overline{\text{Borrow}}$ semnalizează valoarea minimă 0 la numărare inversă. Semnalizarea are loc numai când semnalul de ceas este 0. Expresiile celor două ieșiri sunt:

$$\begin{aligned}\overline{\text{Carry}} &= \overline{Q_3 \cdot Q_2 \cdot Q_1 \cdot Q_0 \cdot \overline{\text{CountUp}}} \\ \overline{\text{Borrow}} &= \overline{\overline{Q_3} \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot \overline{Q_0} \cdot \overline{\text{CountDown}}}\end{aligned} \tag{9. 3}$$

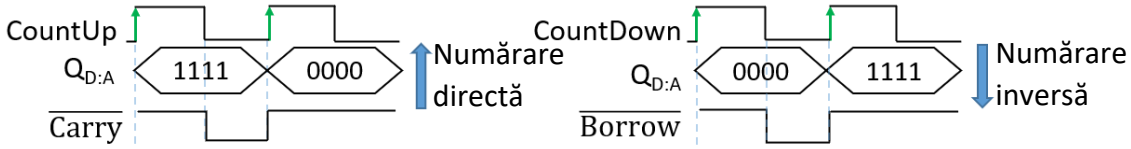
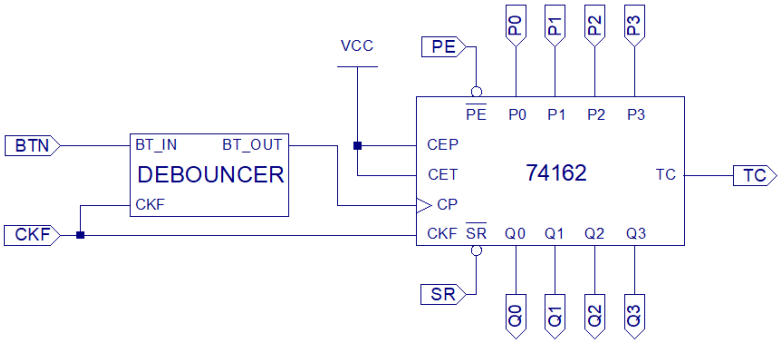


Figura 9. 8 Activarea indicatorilor la atingerea limitelor de numărare ale 74193

9.2.6 Numărătorul zecimal sincron direct 74162

Numărătorul 74162 (Figura 9. 9) numără cifrele zecimale pe 4 biți, în bucla 0-9. Starea acestuia se poate citi pe ieșirile $Q_{3:0}$. Semnalul de ceas se aplică pe CP (Clock Pulse). Comanda $\overline{\text{SR}}$ (Synchronous Reset), în logica negativă, realizează resetarea sincronă (pe frontul de ceas), punând starea pe 0. Comanda $\overline{\text{PE}}$ (Parallel Enable), funcțională tot în logica negativă, realizează încărcarea sincronă a numărătorului cu valoarea de pe intrările $P_{3:0}$. Conform tabelului de funcționare din Figura 9. 9, comanda $\overline{\text{SR}}$ este prioritară față de $\overline{\text{PE}}$. Ambele au prioritate față de operația de numărare.



$\overline{\text{SR}}$	$\overline{\text{PE}}$	CET	CEP	CP	$Q_{3:0}^{t+1}$
0	X	X	X	↑	$(0000)_2$
1	0	X	X	↑	$P_{3:0}$
1	1	1	1	↑	$Q_{3:0}^{t}+1$
1	1	0	X	X	$Q_{3:0}^{t}$
1	1	X	0	X	$Q_{3:0}^{t}$

Figura 9. 9 Testarea numărătorului 74162 (stânga) și tabelul de adevăr (dreapta)

Pentru a activa modul de numărare directă este necesar ca semnalele CEP și CET să fie active (CEP=CET=1) și comenzile $\overline{\text{SR}}$ și $\overline{\text{PE}}$ să fie inactive ($\overline{\text{SR}} = \overline{\text{PE}} = 1$). Ieșirea TC (Terminal Count) semnalizează (prin TC=1) atingerea maximului, când $Q_{3:0}$=1001 (Figura 9. 10). Funcționarea ieșirii TC este condiționată de activarea intrării CET, conform expresiei:

$$\text{TC} = Q_3 \cdot \overline{Q_2} \cdot \overline{Q_1} \cdot Q_0 \cdot \text{CET} \tag{9. 4}$$

Având în vedere faptul că se pot încărca valori din afara buclei de numărare, circuitul este dotat cu un mecanism de *autocorecție sincronă*, astfel încât se revine în buclă, într-un număr de maxim 2 perioade de ceas.

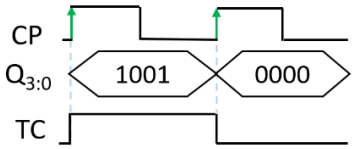
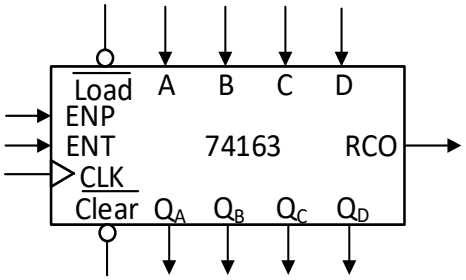


Figura 9. 10 Ieșirea TC indică atingerea valorii maxime la numărătorul 74162

9.2.7 Numărătorul binar sincron direct 74163

Numărătorul 74163 este asemănător cu 74162 din secțiunea anterioară, dar are bucla de numărare 0-15. Conform simbolului din Figura 9. 11, denumirea pinilor diferă, însă păstrează funcționalitatea de la varianta zecimală, așa cum este descrisă în tabel.



$\overline{\text{Clear}}$	$\overline{\text{Load}}$	ENP	ENT	CLK	$Q_{D:A}^{t+1}$
0	X	X	X	↑	$(0000)_2$
1	0	X	X	↑	$(DCBA)_2$
1	1	1	1	↑	$Q_{D:A}^{t}+1$
1	1	0	X	X	$Q_{D:A}^{t}$
1	1	X	0	X	$Q_{D:A}^{t}$

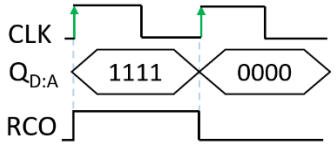


Figura 9. 11 Numărătorul 74163, tabelul de adevăr și semnalizarea limitei superioare

Comenzile $\overline{\text{Clear}}$ și $\overline{\text{Load}}$, funcționale în logica negativă, realizează reset sincron, respectiv încărcare sincronă cu valoarea binară $DCBA_2$. Ambele au prioritate față de modul de numărare directă, care este activ numai dacă ENP=ENT=1. Valoarea numărătorului este disponibilă pe ieșirile $Q_{D:A}$. Ieșirea RCO (Ripple Carry Output) semnalizează atingerea valorii maxime 15, conform diagramei de timp din Figura 9. 11, după formula:

$$\text{RCO} = \text{Q}_\text{D} \cdot \text{Q}_\text{C} \cdot \text{Q}_\text{B} \cdot \text{Q}_\text{A} \cdot \text{ENT} \tag{9. 5}$$

9.3 Activități practice

1. Implementați pe placă numărătorul 74192. Testați comenzile de reset și încărcare paralelă, numărarea în ambele sensuri, autocorecția și semnalizarea limitelor buclei.
2. Implementați pe placă numărătorul 74162. Testați comenzile de reset și încărcare paralelă, numărarea, autocorecția și semnalizarea limitei buclei.
3. Implementați în Logisim numărătorul 74193. Testați comenzile de reset și încărcare paralelă, numărarea în ambele sensuri și semnalizarea limitelor buclei.
4. Implementați în Logisim numărătorul 74163. Testați comenzile de reset și încărcare paralelă, numărarea și semnalizarea limitei buclei.
5. Implementați în Logisim un numărător binar asincron direct, pe 3 biți, cu bistabile JK 7473. Testați bucla de numărare.
6. Implementați în Logisim un numărător binar sincron serie direct, pe 4 biți, cu bistabile JK 7473. Testați bucla de numărare.

10 Circuite logice secvențiale – aplicații ale numărătoarelor

10.1 Obiective

Se studiază diverse modalități de extindere a domeniului de numărare folosind cascadarea numărătoarelor cu număr redus de biți. Se analizează diferențele de interpretare a informației la numărarea binară și binar-zecimală cu numărătoare multiple. Se studiază implementarea numărătoarelor *modulo p* prin diverse strategii adaptate la particularitățile de funcționalitate ale circuitelor folosite.

10.2 Considerații teoretice

În sinteza sistemelor numerice cu reprezentarea informației pe un număr mare de biți se folosesc numărătoare cu capacitate extinsă. Acestea pot fi implementate prin cascadarea mai multor numărătoare de capacitate redusă. De exemplu, având la dispoziție numărătoare pe 4 biți, precum cele studiate în lucrarea anterioară, se poate extinde numărul de biți la un multiplu de 4, prin conectarea acestora în mod repetat. În acest fel se poate extinde domeniul de numărare de la 2^4 valori la $2^{n\times4}$, unde n este numărul de numărătoare cascadate.

O altă aplicație frecvent utilizată în sinteza sistemelor numerice o reprezintă reducerea domeniului de numărare la un subset de valori din cele care se pot reprezenta pe n biți. Un numărător care numără p valori din cele 2^n, unde $p<2^n$ poartă denumirea de numărător *modulo p*. Alegerea celor p valori poate fi arbitrară, motiv pentru care se pot genera mai multe numărătoare *modulo p*, pentru același p, în funcție de secvența de valori numărată. De exemplu, în categoria numărătoarelor *modulo 10* se încadrează și numărătorul *zecimal* sau *decadic*, care numără consecutiv în bucla 0-9, reprezentând cifrele zecimale.

10.2.1 Extinderea domeniului de numărare prin cascadare

Extinderea domeniului de numărare prin cascadarea numărătoarelor are la bază următorul principiu: numărătorul cu rangul cel mai inferior numără la fiecare impuls de ceas; orice alt numărător din șirul de numărătoare cascadate își activează numărarea o singură dată, pentru un impuls de ceas, la fiecare finalizare a buclei de numărare pentru numărătorul de rang inferior.

În cazul numărătoarelor binare reversibile 74193, cascadarea presupune conectarea indicatorilor de finalizare a buclei fiecărui numărător la semnalele de ceas ale numărătorului de rang superior (dacă există), ca în Figura 10. 1:

- La numărarea directă, numărătorul care numără pe biții $Q_{3:0}$ primește semnalul de ceas pe intrarea CountUp, și intrarea CountDown se conectează la 1. Ieșirea $\overline{\text{Borrow}}$ va fi menținută inactivă la 1, ceea ce determină activarea numărării (incrementarea) pe biții $Q_{7:4}$, la fiecare front ascendent generat de $\overline{\text{Carry}}$. Acest

front are loc la inactivarea semnalului $\overline{Carry}$, când numărătorul de rang inferior trece din 15 în 0 (Figura 10. 2 – stânga).

- La numărarea inversă, numărătorul cu ieșirile $Q_{3:0}$ primește semnalul de ceas pe CountDown, și CountUp se conectează la 1. În consecință, $\overline{Carry}$ va fi menținut la 1 și frontul ascendent generat pe $\overline{Borrow}$, la fiecare trecere din 0 în 15 (Figura 10. 2 – dreapta), va determina o decrementare a numărătorului pe $Q_{7:4}$.

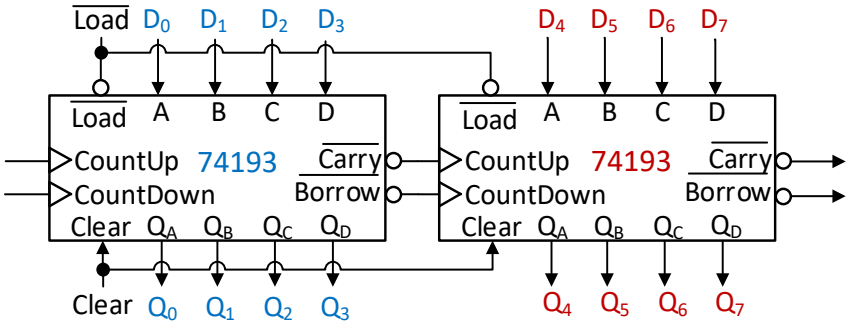


Figura 10. 1 Cascadarea a două numărătoare 74193

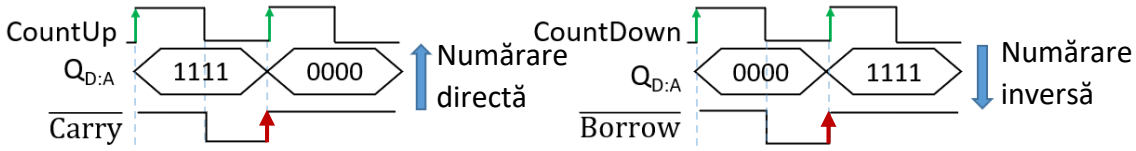


Figura 10. 2 Generarea frontului ascendent pe $\overline{Carry}$ sau $\overline{Borrow}$ la numărătorul 74193, pe final de buclă

Notă: Bucla de numărare pe 8 biți, înregistrată pe ieșiri, conține valori în intervalul 0-255 (Figura 10. 4 – stânga).

La cascadarea numărătoarelor zecimale reversibile 74192 conexiunile se realizează în mod similar: indicatorii $\overline{TC_U}$ și $\overline{TC_D}$ se conectează la CP_U, respectiv CP_D, ca în Figura 10. 3. Valorile înregistrate trebuie interpretate în baza 10, convertind perechi de câte 4 ieșiri, la cifrele zecimale corespunzătoare. La fiecare 10 valori consecutive ale $Q_{3:0}$ are loc o modificare a valorii pe $Q_{7:4}$. Bucla pe ieșirile $Q_{7:0}$ conține următoarele valori zecimale: 00, 01, 02, … 09, 10, 11, 12, … 19, 20, 21, 22, … 99, 00, 01, 02, … (Figura 10. 4 – dreapta).

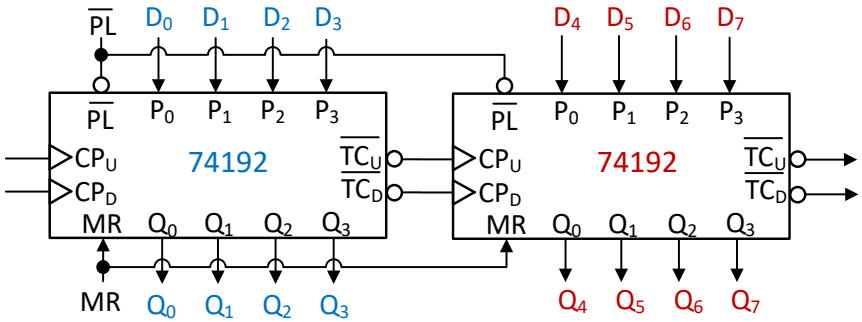


Figura 10. 3 Cascadarea a două numărătoare 74192

Observație: Comenzile asincrone de reset și încărcare cu valoarea extinsă $D_{7:0}$ sunt conectate în comun la toate numărătoarele, astfel că efectul lor este simultan.

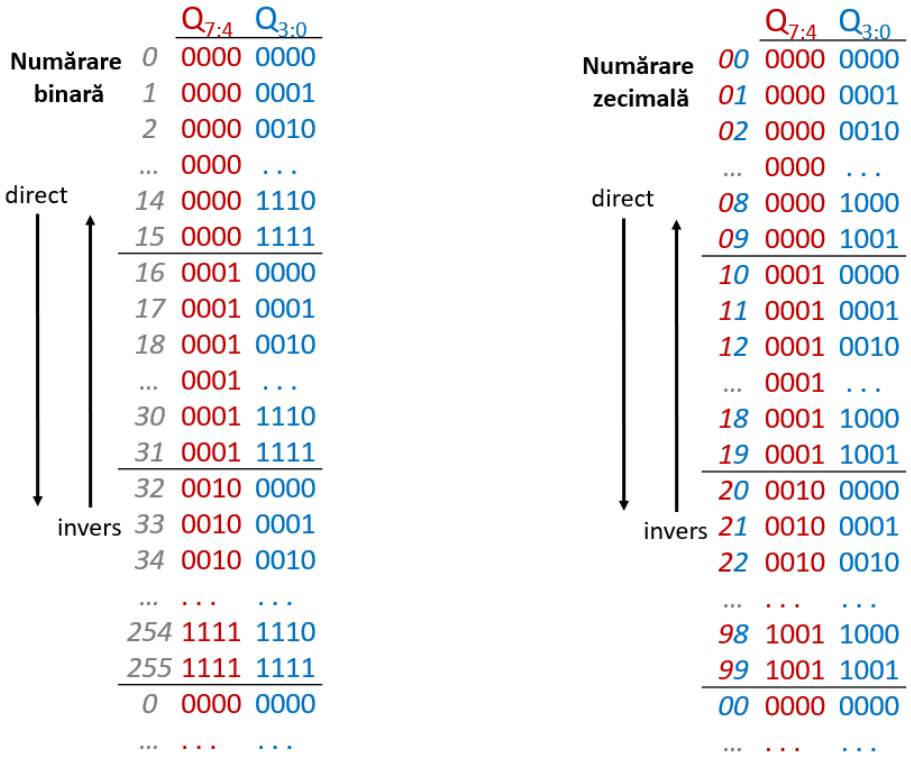


Figura 10. 4 Buclele de numărare pe 8 biți: binară (stânga) și zecimală (dreapta)

Există și posibilitatea cascadării mixte între numărătorul binar 74193 cu numărătorul zecimal 74192. Dacă se dorește obținerea unei numărări binare, atunci pentru biții mai puțin semnificativi $Q_{3:0}$ trebuie folosit un numărător binar 74193 (Figura 10. 5). Se va obține astfel o numărare binară în bucla 0-159 (159_{10}=$1001\,1111_2$). În caz contrar, folosind un numărător zecimal 74192 pe $Q_{3:0}$, bucla va fi discontinuă, fiindcă după fiecare 10 valori consecutive se face un salt cu 7 valori: 0, 1, … 9, 16, 17, 18, … 25, 32 … .

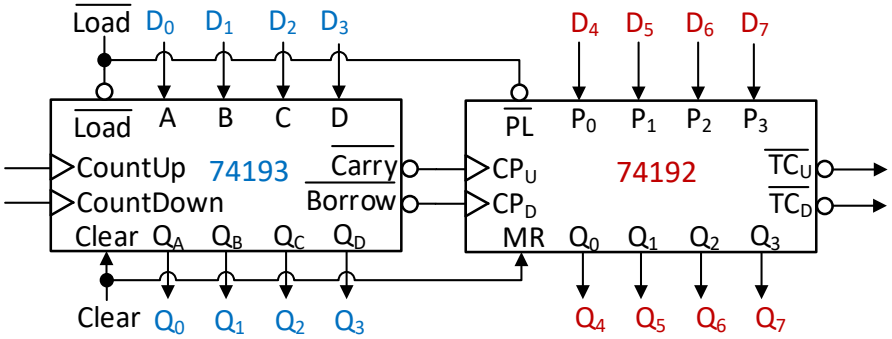


Figura 10. 5 Cascadarea numărătorului 74192 cu 74193, la numărarea binară

În cazul cascadării numărătoarelor binare directe 74163 (Figura 10. 6 – stânga), acestea vor avea intrarea de ceas conectată, în comun, la semnalul de tact al circuitului. Numărătorul pentru biții $Q_{3:0}$ va număra la fiecare impuls prin activarea intrărilor ENP=ENT=1. Restul numărătoarelor vor avea ENP=1, și intrarea ENT conectată la indicatorul RCO, de finalizare a buclei numărătorului de rang inferior. RCO se activează pe

durata impulsului în care $Q_{3:0}$=1111, iar la trecerea $Q_{3:0}$ în 0000, provoacă incrementarea numărătorului de rang superior, după care se va dezactiva, conform diagramei de timp din Figura 10. 6. Bucla de numărare binară rezultată pe 8 biți este 0-255.

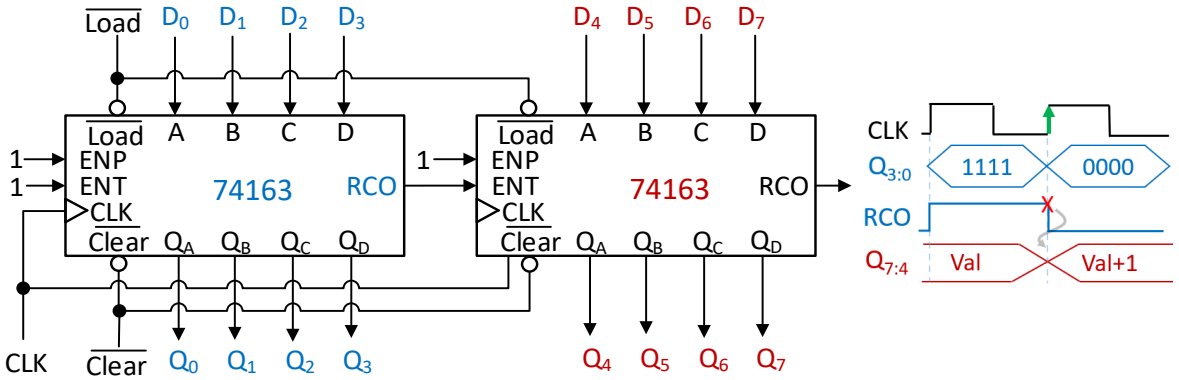


Figura 10. 6 Cascadarea a două numărătoare 74163 (stânga) și activarea numărătorului de rang superior la finalul de buclei numărătorului de rang inferior (dreapta)

Cascadarea numărătoarelor zecimale directe 74162 se realizează în mod similar (Figura 10. 7) și funcționarea urmează aceleași principii. Bucla de numărare zecimală rezultată pe 8 biți (2 cifre zecimale) este 00-99. Numărarea este directă.

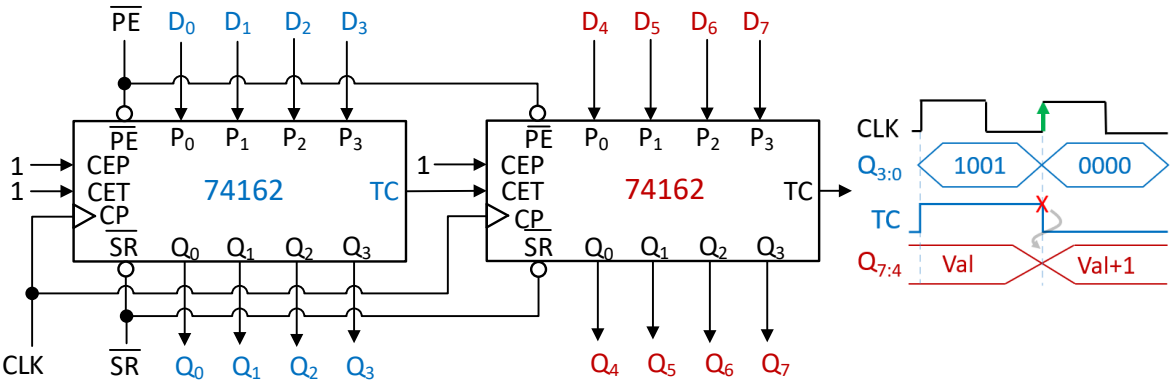


Figura 10. 7 Cascadarea a două numărătoare 74162 (stânga) și activarea numărătorului de rang superior la finalul de buclei numărătorului de rang inferior (dreapta)

Figura 10. 8 prezintă cascadarea mixtă între 74162 și 74163. Pentru a elimina discontinuitățile, este necesară amplasarea unui numărător binar pentru ieșirile $Q_{3:0}$.

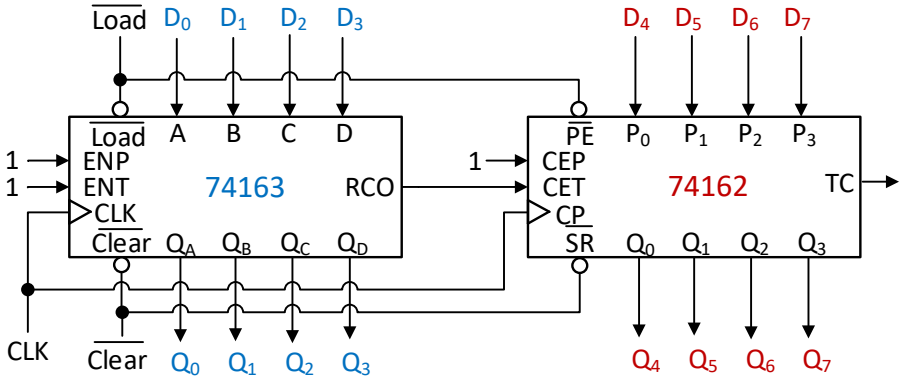


Figura 10. 8 Cascadarea numărătorului 74162 cu 74163, la numărarea binară

Bucla de numărare binară pentru numărătorul binar pe 8 biți, din Figura 10. 8, este 0-159 (159_{10}=10011111_2).

10.2.2 Numărătoare *modulo p* cu secvența de numărare continuă

Numărătoarele *modulo p*, numără în bucle cu *p* valori de *n* biți, din cele 2^n posibile. La numărare continuă valorile buclei sunt consecutive, dar începutul, finalul și direcția de numărare pot să varieze. Având în vedere faptul că ieșirile numărătoarelor sunt citite de alte circuite sincrone pe același front de ceas, în timpul funcționării contează doar acele valori care sunt vizibile (stabile) în proximitatea și în timpul frontului de ceas. Alte valori înregistrate între fronturi sunt ignorate, nefiind încadrate bucla de numărare.

Implementarea numărătoarelor *modulo p* cu secvență continuă, folosind circuitele 74192, 74193, 74162, respectiv 74163 se realizează în felul următor: se numără până la finalul buclei, apoi se realizează încărcarea (load) cu valoarea de început a buclei. Dacă valoarea de început este 0, atunci se poate înlocui cu comanda de reset. Finalul de buclă se poate detecta cu logică suplimentară sau, în unele cazuri particulare, cu indicatorii de final de buclă de tip *carry* și *borrow*. **Observație**: Se vor folosi numai numărătoare care au în bucla lor originală toate valorile prevăzute în bucla *modulo p*. De exemplu, cu un numărător zecimal se pot implementa numai bucle *modulo p* cu valori în intervalul 0-9.

Notă: Inițial, numărătoarele pot avea valoarea 0, iar dacă 0 nu face parte din bucla de numărare, vor fi necesare un număr de impulsuri de ceas până la intrarea în buclă.

La implementarea cu circuitele 74192 sau 74193, având comenzile de reset și load asincrone, considerând că valoarea de final a buclei este *Val*, atunci activarea uneia din cele două comenzi la atingerea *Val*, va duce la o resetare sau o încărcare imediată cu valoarea de început a buclei, ceea ce înseamnă că valoarea *Val* va fi suprascrisă înaintea frontului de ceas următor, deci nu va face parte din bucla de numărare. Soluția constă în întârzierea revenirii la începutul buclei, cu un impuls de ceas, adică atunci când se detectează *Val*+1 la numărare directă, sau *Val*-1 la numărare inversă. Considerând valoarea $D_{3:0}$ de început a buclei, implementarea cu numărătorul 74193 este redată în Figura 10. 9. În practică, intrările A, B, C, D vor fi conectate la 0 (GND) sau 1 (VCC) în funcție de valoarea de start a buclei. Dacă $D_{3:0}$=7 (7_{10}=0111_2), atunci A=1, B=1, C=1, respectiv D=0. **Notă**: Dacă nu se folosește comanda Clear aceasta se inactivează. Valorile semnalelor la finalul și reluarea buclei sunt prezentate în diagramele de timp din Figura 10. 10.

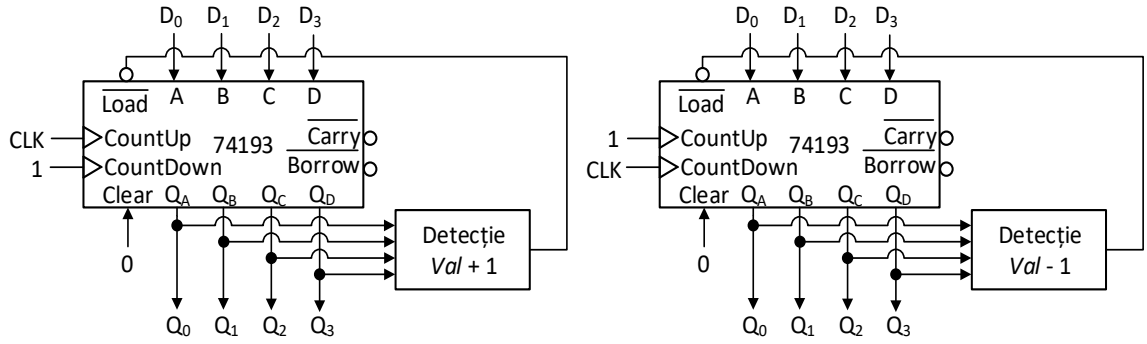


Figura 10. 9 Implementarea buclei $D_{3:0}$-*Val* cu numărătorul 74193, la numărarea directă (stânga) și inversă (dreapta)

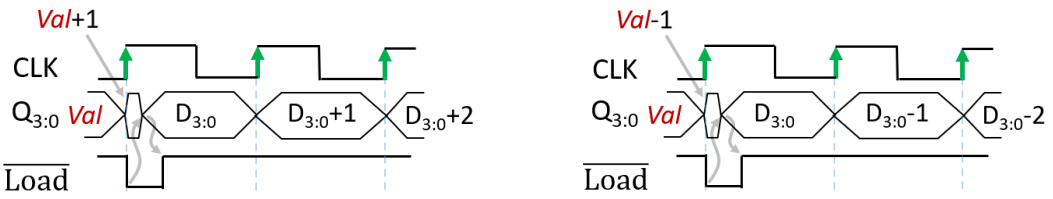


Figura 10. 10 Valorile semnalelor la finalul și la reluarea buclei $D_{3:0}$-*Val*, implementată cu numărătorul 74193, în cazul numărării directe (stânga) și inverse (dreapta)

Detecția unei valori pe ieșirile numărătorului se poate realiza folosind poarta ȘI, prin implementarea mintermului corespunzător. Fiindcă $\overline{\text{Load}}$ este activ pe 0, este mai oportun să se folosească poarta ȘI-NU. Dacă valoarea de detectat este 15 sau 0 se pot folosi indicatorii $\overline{\text{Carry}}$, respectiv $\overline{\text{Borrow}}$. De exemplu, pentru a implementa un numărător cu bucla 0-14, se va detecta valoarea 15 cu ajutorul $\overline{\text{Carry}}$ și pentru revenirea la 0 se poate folosi comanda Clear, iar $\overline{\text{Load}}$ se inactivează (Figura 10. 11 – stânga). Dacă se dorește bucla 7-15, se detectează 0 (0=15+1) și se încarcă 7 (Figura 10. 11 – dreapta).

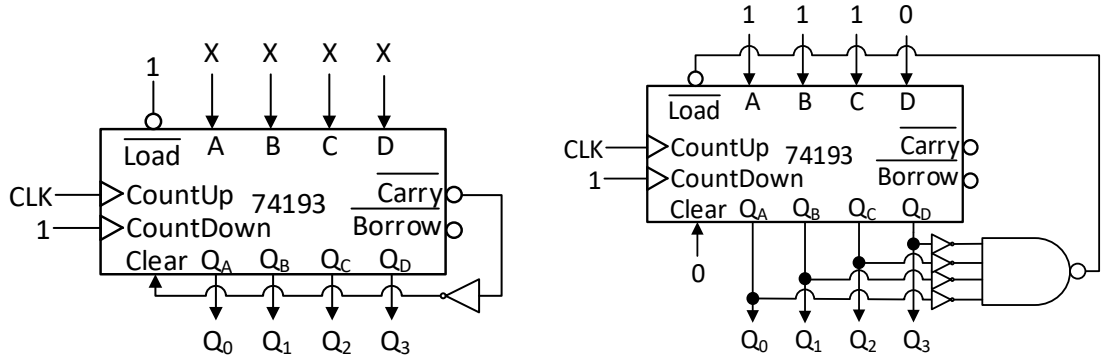


Figura 10. 11 Numărătoare cu buclele 0-14 (stânga) și 7-15 (dreapta), folosind 74193

Utilizarea numărătorului zecimal 74192 urmează principii similare, doar că se ține cont de bucla 0-9 a acestuia. Figura 10. 12 prezintă numărătoare inverse cu buclele 6-0 (se detectează 9 și se încarcă 6), respectiv 8-1 (se detectează 0 și se încarcă 8).

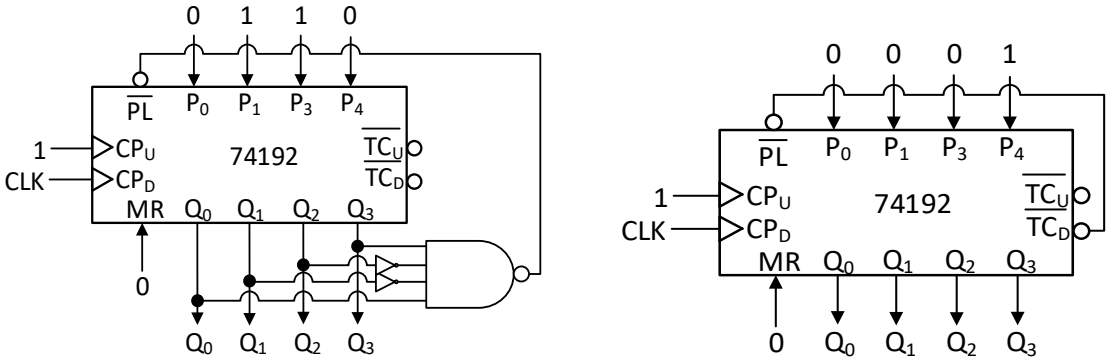


Figura 10. 12 Numărător cu bucla 6-0 (stânga) și 8-1 (dreapta), folosind 74192

În cazul numărătoarelor 74162 și 74163 se pot implementa numai bucle directe în intervalul 0-9, respectiv 0-15. Se detectează valoarea de final a buclei și se lansează o comandă de reset sau încărcare cu prima valoare din buclă, în funcție de caz. Deoarece aceste comenzi sunt sincrone, efectul lor nu va avea loc decât după următorul front de

ceas, în consecință valoarea curentă nu se pierde prin suprascriere. Implementarea buclei $D_{3:0}$-*Val* cu numărătorul 74163, și valorile semnalelor la finalul, respectiv reluarea buclei, sunt prezentate în Figura 10. 13.

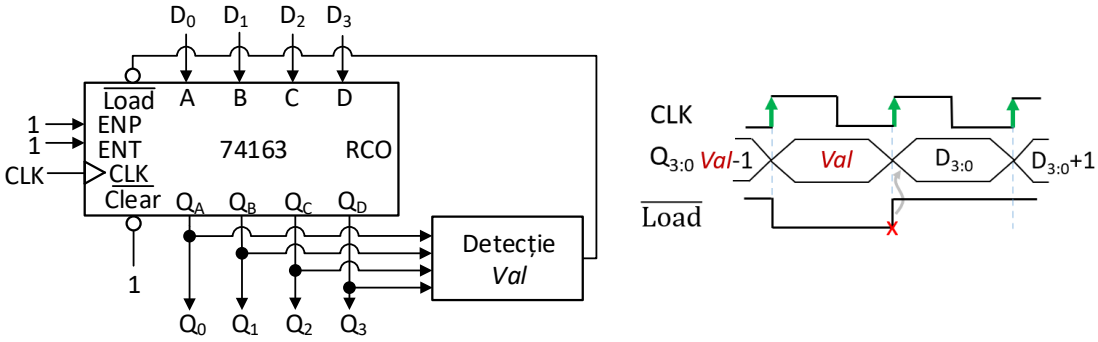


Figura 10. 13 Implementarea buclei $D_{3:0}$-*Val* cu numărătorul 74163 (stânga), și valorile semnalelor la finalul și la reluarea buclei

La un numărător cu bucla 10-12 folosind numărătorul 74163, se detectează valoarea 12 pe ieșiri și se încarcă 10. Circuitul de test este prezentat în Figura 10. 14.

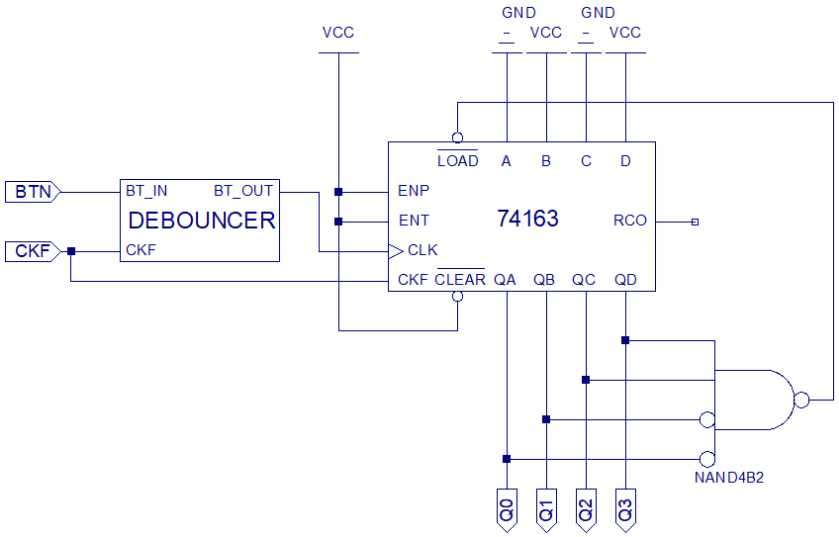


Figura 10. 14 Circuitul de test al numărătorului cu bucla 10-12, folosind 74163

Pentru a implementa buclele directe 4-9 și 0-8, cu numărătorul zecimal 74162, se folosesc configurațiile următoare:

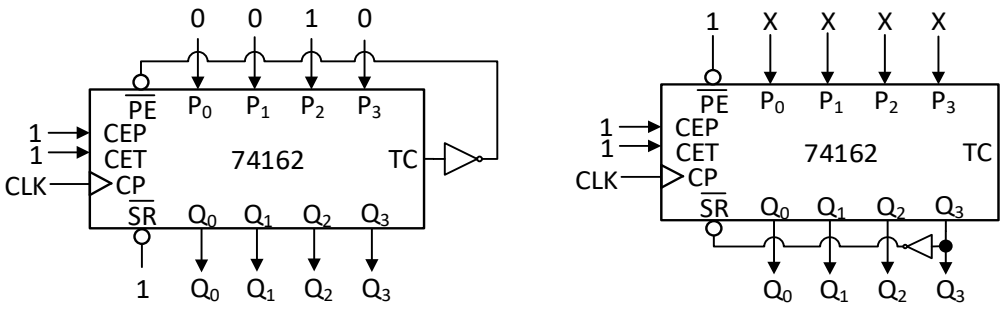


Figura 10. 15 Numărătoare cu buclele 4-9 (stânga) și 0-8 (dreapta), folosind 74162

Notă: În primul caz, detecția valorii 9 se realizează cu indicatorul TC. În al doilea caz, detecția valorii 8 se poate reduce la bitul Q_3, deoarece se activează numai pentru valoarea 8 din cadrul buclei, în rest fiind 0.

Dacă se dorește implementarea unei bucle cu valori mai mari se pot folosi numărătoare cascadate. De exemplu, bucla 37-74 se poate implementa cu numărătoare 74193 cascadate, încărcând valoarea 37 (00100101_2) la detectarea valorii 75 (01001011_2):

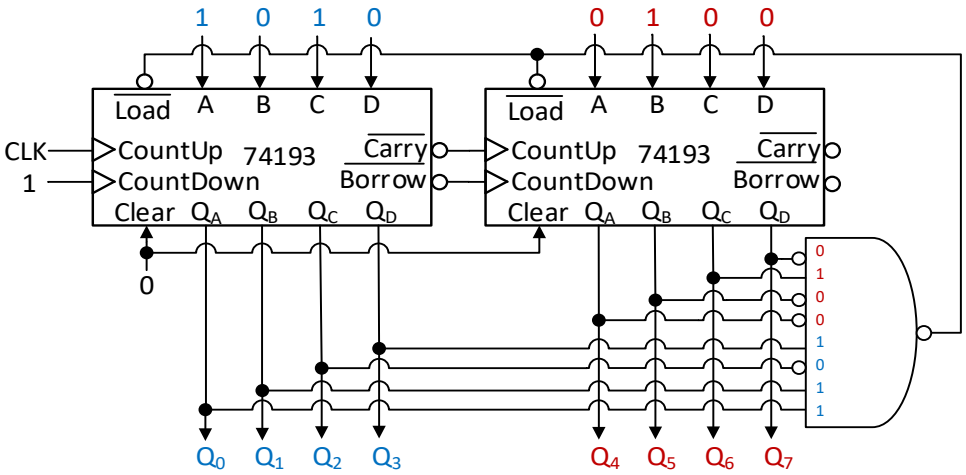
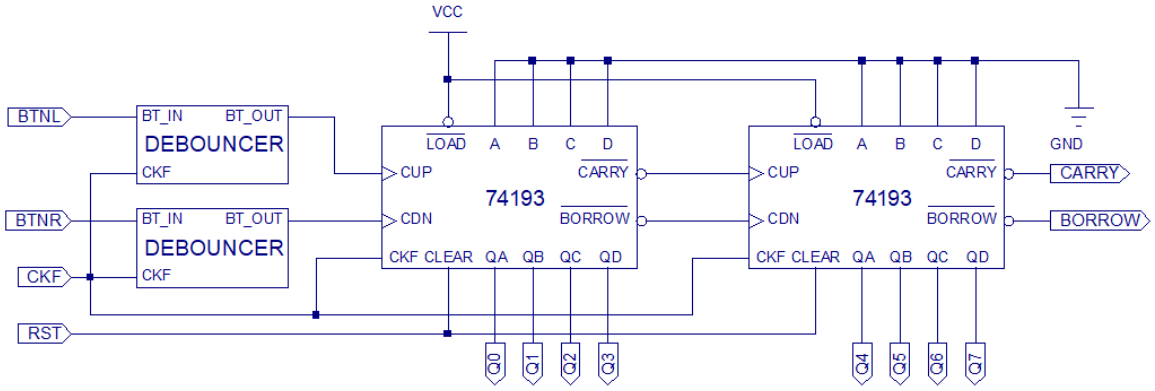


Figura 10. 16 Numărător cu bucla 37-74, folosind circuite 74193 cascadate

10.3 Activități practice

1. Implementați pe placă numărătorul de mai jos, cu numărătoare 74193 cascadate. Testați comanda asincronă de reset și numărarea în bucla 0-255, în ambele sensuri.



2. Implementați pe placă numărătorul *modulo 3* direct, pe 4 biți, cu bucla 10-12, implementat cu circuitul 74163. Testați numărarea în buclă.
3. Implementați în Logisim un numărător binar pe 8 biți implementat prin cascadarea unui 74162 cu 74163. Testați resetarea, încărcarea sincronă și numărarea în bucla 0-159.
4. Implementați în Logisim numărătorul *modulo 7* invers, pe 4 biți, cu bucla 6-0, implementat cu circuitul 74192. Testați numărarea în buclă.
5. Implementați în Logisim numărătorul *modulo 6* direct, pe 4 biți, cu bucla 4-9, implementat cu circuitul 74162. Testați numărarea în buclă.
6. Implementați în Logisim numărătorul *modulo 38* direct, pe 8 biți, cu bucla 37-74, implementat cu circuite 74193 cascadate. Testați numărarea în buclă.

11 Circuite logice secvențiale – registre

11.1 Obiective

Sunt prezentate categoriile de registre existente, clasificate după funcționalitatea acestora. Este studiată implementarea registrului universal cu bistabile și multiplexoare. Se analizează funcționalitatea registrelor sub formă de circuite integrate. Sunt prezentate modalități de implementare a generatoarelor de secvențe numerice cu ajutorul registrelor.

11.2 Considerații teoretice

Registrele sunt circuite logice secvențiale folosite pentru stocarea și deplasarea datelor. Implementarea acestora se realizează cu bistabile, câte un bistabil pentru fiecare bit de date stocat. Datorită simplității circuitului rezultat, tipurile de bistabile cel mai des utilizate sunt bistabilele D. Numărul de biți reprezintă *capacitatea registrului*. După funcționalitatea lor, registrele se pot clasifica astfel:

- *de memorie* – pot să încarce și să memoreze date la fiecare impuls de ceas;
- *de deplasare* – pot să deplaseze datele, pe biți, câte o poziție, la fiecare impuls de ceas, în paralel cu încărcarea serială a unui bit: BitSerial→Q_A→Q_B→Q_C→ …;
- *combinate* – combină funcționalitatea registrelor de memorie și deplasare;
- *universale* – sunt registre combinate, cu deplasare bidirecțională.

11.2.1 Registrul universal

În Figura 11. 1 este prezentată implementarea unui registru universal pe 4 biți, folosind bistabile D flip-flop și multiplexoare. Fiecare bit are asociat un bistabil și un multiplexor care decide funcționalitatea. Bistabilele se pot reseta asincron.

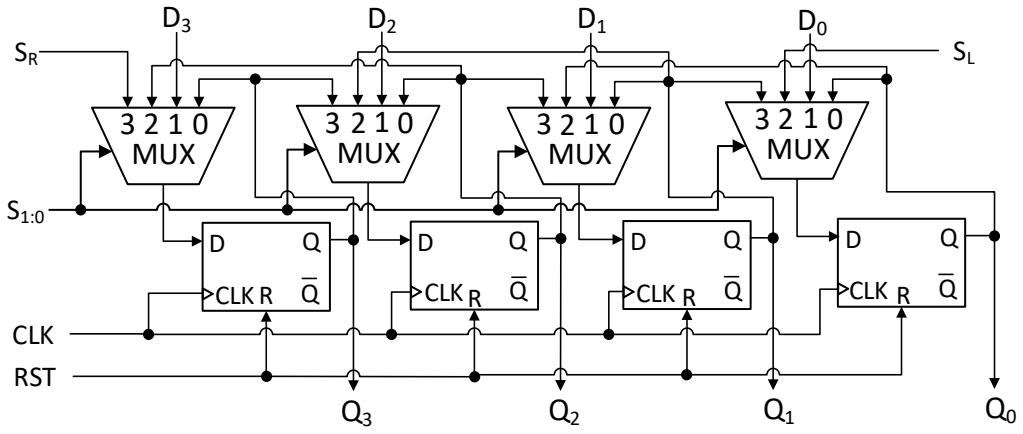


Figura 11. 1 Implementarea unui registru universal pe 4 biți cu bistabile D flip-flop

Selecția pe 2 biți $S_{1:0}$ aplicată multiplexoarelor definește funcționalitatea registrului în felul următor:

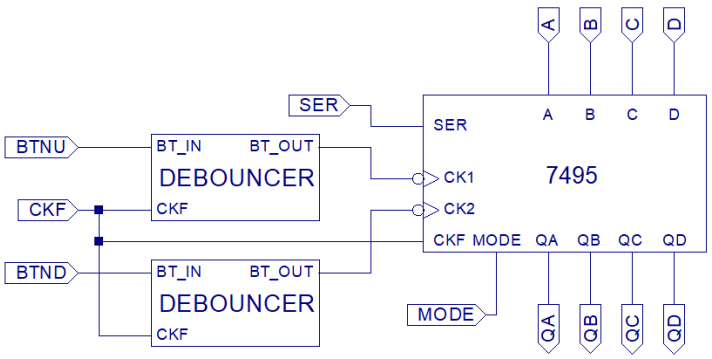
- $S_{1:0}$=00 – memorare: $Q_3Q_2Q_1Q_0{}^{t+1} = Q_3Q_2Q_1Q_0{}^{t}$;

- $S_{1:0}$=01 – încărcare paralelă: $Q_3Q_2Q_1Q_0{}^{t+1} = D_3D_2D_1D_0$;
- $S_{1:0}$=10 – încărcare serială a intrării S_L (Serial Left) cu deplasare la stânga: $Q_3Q_2Q_1Q_0{}^{t+1} = Q_2Q_1Q_0S_L{}^{t}$;
- $S_{1:0}$=11 – încărcare serială a intrării S_R (Serial Right) cu deplasare la dreapta: $Q_3Q_2Q_1Q_0{}^{t+1} = S_RQ_3Q_2Q_1{}^{t}$.

Notă: Încărcarea serială presupune încărcarea unui bit de date la fiecare impuls de ceas. La registrul pe 4 biți sunt nevoie de 4 impulsuri de ceas pentru a încărca în întregime registrul, spre deosebire de încărcarea paralelă, care încarcă registrul într-un singur impuls.

11.2.2 Registrul combinat 7495

Circuitul 7595 este un registru combinat pe 4 biți, care implementează operațiile de încărcare, memorare și deplasare serială. Circuitul de test și tabelul de adevăr sunt prezentate în Figura 11. 2. Registrul are două moduri de funționare: dacă MODE=0, se face deplasare cu încărcare serială pe intrarea SER (în direcția SER→Q_A→Q_B→Q_C→Q_D), sincron cu frontul descentent al semnalului de ceas CK_1; dacă MODE=1, se încarcă valoarea $(ABCD)_2$ în registru, sincron cu frontul descentent al semnalului de ceas CK_2. **Notă**: Modificarea modului de funcționare se face numai când CK_1=CK_2=0.



MODE	CK_1	CK_2	SER	$Q_{A:D}^{t+1}$
0	↓	X	0	0 Q_A^t Q_B^t Q_C^t
0	↓	X	1	1 Q_A^t Q_B^t Q_C^t
1	X	↓	X	A B C D

Figura 11. 2 Circuitul de test pentru registrul 7495 (stânga) și tabelul de adevăr (dreapta)

Prin cascadarea mai multor registre 7495 se obține aceeași funcționalitate, dar pe un număr multiplu de 4 biți. Astfel, un registru combinat pe 8 biți se obține prin cascadarea a două registre pe 4 biți, conform cu Figura 11. 3.

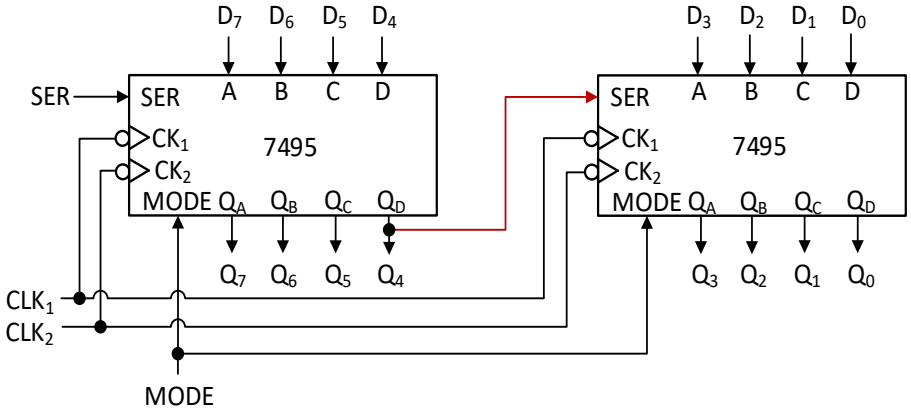


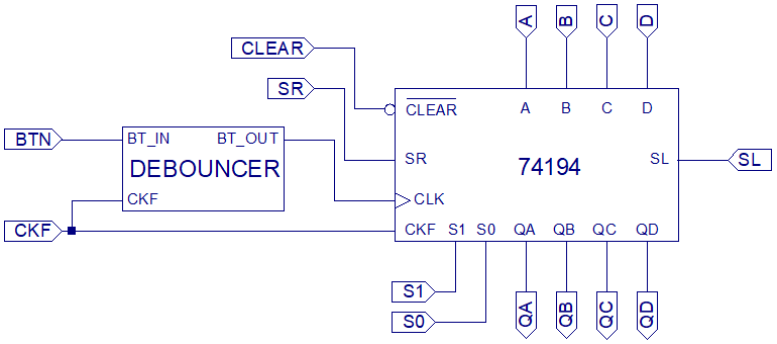
Figura 11. 3 Cascadarea a două registre 7495

Se observă partajarea semnalelor de ceas și a comenzii MODE de către toate circuitele implicate. Pentru a realiza deplasarea biților de la un registru, la vecinul său din dreapta, se conectează ieșirea Q_D la intrarea SER a vecinului, realizând astfel deplasarea bitului Q_4 către bitul Q_3.

11.2.3 Registrul universal 74194

Registrul universal pe 4 biți 74194 (Figura 11. 4) este sincron pe frontul ascendent al semnalului de ceas. Intrarea $\overline{CLEAR}$, activă pe 0, este prioritară și realizează reset asincron. Cu ajutorul semnalelor de selecție S_1 și S_0 se determină cele 4 operații sincrone pe frontul ascendent, conform tabelului din Figura 11. 4:

- $S_{1:0}$=00 – pentru memorare;
- $S_{1:0}$=01 – pentru deplasare la dreapta cu încărcare serială pe S_R;
- $S_{1:0}$=10 – pentru deplasare la stânga cu încărcare serială pe S_L;
- $S_{1:0}$=11 – pentru încărcare cu valoarea $(ABCD)_2$.



$\overline{CLR}$	S_1	S_0	CLK	$Q^{t+1}_{A:D}$			
0	X	X	X	0	0	0	0
1	0	0	↑	Q^t_A	Q^t_B	Q^t_C	Q^t_D
1	0	1	↑	S_R	Q^t_A	Q^t_B	Q^t_C
1	1	0	↑	Q^t_B	Q^t_C	Q^t_D	S_L
1	1	1	↑	A	B	C	D

Figura 11. 4 Testarea registrului 74194 (stânga) și tabelul de funcționare (dreapta)

Pentru extinderea numărului de biți se pot cascada mai multe registre 74194. Figura 11. 5 prezintă cascadarea a două registre pentru realizarea unui registru universal pe 8 biți. Circuitele 74194 parteajează semnalul de ceas, comanda $\overline{CLEAR}$ și semnalele de selecție S_1, S_0, asfel că ambele registre vor avea același regim de lucru. Pentru deplasarea biților între registre se conectează Q_D la S_R ($Q_4 \rightarrow Q_3$) și Q_A la S_L ($Q_3 \rightarrow Q_4$).

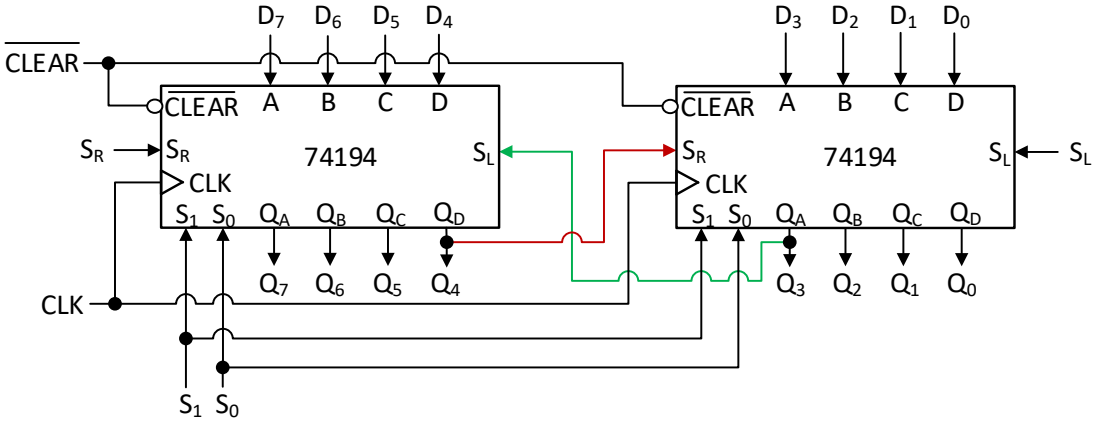


Figura 11. 5 Cascadarea a două registre 74194

11.2.4 Generatoare de secvențe numerice

Secvențele numerice sunt șiruri de numere care se repetă în aceeași ordine. Astfel de secvențe se pot genera în mod facil folosind registre de deplasare și logică suplimentară. O secvență care conține 2^n-1 valori nenule pe *n* biți, într-o succesiune aparent aleatoare, se numește *secvență pseudoaleatoare*. Astfel de secvențe se pot genera cu registre, încărcate inițial cu o valoare nenulă, care efectuează operația de deplasare, concomitent cu încărcarea serială a rezultatului operației XOR între bitul cel mai semnificativ și bitul cel mai putin semnificativ de pe ieșirile sale. Figura 11. 6 ilustrează implementarea unui generator de numere pseudoaleatoare pe 4 biți, cu registrul 74194, în două variante. Inițial, se încarcă valoarea 0001_2, setând $S_{1:0}$=11 pentru un impuls.

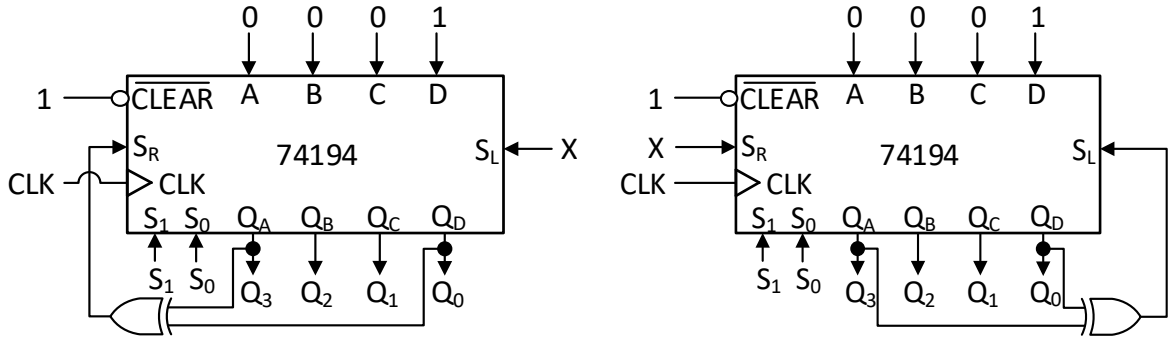


Figura 11. 6 Generator de numere pseudoaleatoare pe 4 biți, folosind registrul 74194, cu încărcare serială pe S_R (stânga) și cu încărcare serială pe S_L (dreapta)

În prima variantă (Figura 11. 6 – stânga), pentru următoarele impulsuri de ceas după inițializare, se setează $S_{1:0}$=01 și se trece în modul de deplasare la dreapta cu încărcare serială pe S_R. Secvența repetitivă generată pe ieșirea $Q_{3:0}$ conține 15 valori: 0001 → 1000 → 1100 → 1110 → 1111 → 0111 → 1011 → 0101 → 1010 → 1101 → 0110 → 0011 → 1001 → 0100 → 0010. După ultima valoare secvența se reia în aceeași ordine.

În varianta a doua (Figura 11. 6 – dreapta), după inițializare, se setează $S_{1:0}$=10 și se trece în modul de deplasare la stânga cu încărcare serială pe S_L. Secvența repetitivă generată este: 0001 → 0011 → 0111 → 1111 → 1110 → 1101 → 1010 → 0101 → 1011 → 0110 → 1100 → 1001 → 0010 → 0100 → 1000.

Implementarea primei secvențe cu circuitul 7495 este ilustrată în Figura 11. 7.

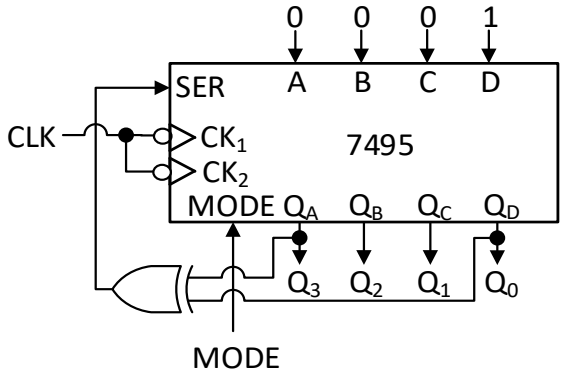


Figura 11. 7 Generator de numere pseudoaleatoare pe 4 biți, folosind registrul 7495

Inițial, se setează MODE=1 pentru cel puțin un impuls de ceas și se încarcă valoarea $(0001)_2$. Ulterior, se setează MODE=0 și se generează restul secvenței la următoarele impulsuri de ceas. Tranziția între valori are loc pe frontul descendent.

Notă: Indiferent de tipul de registru utilizat, trebuie evitată încărcarea acestuia cu valoarea 0000_2, deoarece ieșirea se blochează în această valoare.

Pentru generarea unor secvențe pseudoaleatoare pe mai mult de 4 biți se pot folosi registre cascadate, ca în Figura 11. 8. Inițial, se încarcă o valoare nenulă, la alegere, iar apoi se trece în modul de deplasare. Pentru o secvență pe 6 biți se pot folosi 4 biți de la un registru și 2 biți de la al doilea registru.

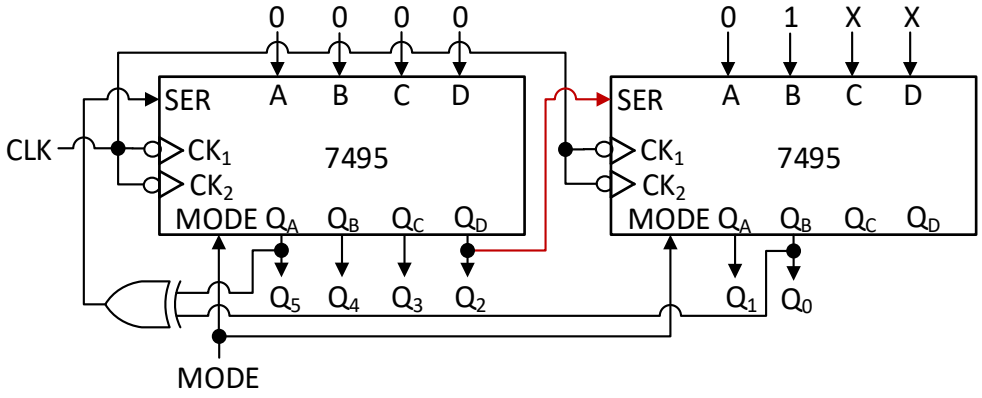


Figura 11. 8 Generator de numere pseudoaleatoare pe 6 biți, folosind registre 7495 cascadate

11.3 Activități practice

1. Implementați pe placă registrul 7495. Testați regimurile de funcționare pe frontul descendent al semnalelor de ceas. Considerând ieșirea Q_A=bitul cel mai semnificativ, încărcați paralel, și serial, valorile: 13, 15, 7.
2. Implementați pe placă registrul 74194. Testați resetul asincron și regimurile de funcționare pe frontul ascendent. Considerând ieșirea Q_A=bitul cel mai semnificativ, după câte un reset asincron, încărcați paralel, și serial (în ambele direcții), valorile: 10, 12, 6.
3. Implementați și testați în Logisim un generator de secvență pseudoaleatoare pe 4 biți, care începe cu valoarea 0001_2, încărcată inițial. Folosiți registrul 7495.
4. Implementați și testați în Logisim un generator de secvență pseudoaleatoare pe 6 biți, care începe cu valoarea 000001_2, încărcată inițial. Folosiți registre 7495.
5. Implementați în Logisim un registru universal pe 8 biți prin cascadarea a două registre 74194. Testați resetul asincron și regimurile de funcționare pe frontul ascendent. După câte un reset asincron, încărcați paralel, și serial (în ambele direcții), valorile: 17, 31, 67.

12 Dezvoltarea cu circuite logice programabile de tip FPGA

12.1 Obiective

Se prezintă arhitectura unui circuit FPGA (Field Programmable Gate Array). Se detaliază componența celulei logice de bază și se studiază principiul de implementare a circuitelor cu astfel de celule. Este descrisă modalitatea de generare a simbolului unui circuit. Sunt enumerați pașii necesari realizării unei simulări cu utilitarul ISim, configurarea valorilor de intrare și se exemplifică utilizarea acestuia.

12.2 Considerații teoretice

Placa de dezvoltare utilizată pe parcursul lucrărilor prezintă un circuit programabil de tip FPGA, alcătuit din celule logice de bază (Configurable Logic Blocks – CLB) interconectate, care sunt organizate într-o rețea de tip matrice (Figura 12. 1 – stânga). O celulă conține 2 sau mai multe *slice*-uri. Un *slice* conține memorii RAM *Look-Up Table* (LUT), logică de *carry* (transport), necesară la operații aritmetice, și bistabile. Conținutul memoriilor LUT poate fi configurat încât să implementeze funcții booleene cu număr de variabile identic cu numărul de biți de adresă. Logica de *carry* facilitează propagarea rapidă, între biți, a transportului generat la operații aritmetice. Bistabilele reprezintă suportul pentru generarea diverselor circuite logice secvențiale necesare în cadrul unei arhitecturi. De exemplu, la modelul Artix-7 prezent pe placă, un *slice* conține 4 memorii LUT și 8 bistabile (Figura 12. 1 – dreapta). Ieșirea unei memorii LUT poate fi înaintată direct la exterior sau prin intermediul unui bistabil. Prin rețeaua de interconectare, ieșirile se pot distribui la intrările aceluiași *slice* și la *slice*-uri din celule identice sau diferite.

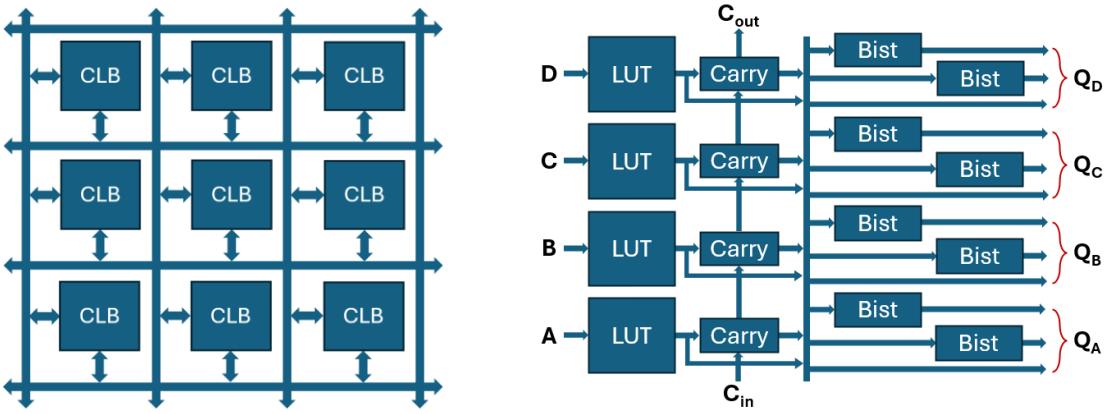


Figura 12. 1 Organizarea celulelor într-un FPGA (stânga) și structura unui *slice* (dreapta)

Pentru interconectare, circuitul FPGA conține un grid de linii intersectate în jurul celulelor (Figura 12. 2). Fiecare intersecție reprezintă o conexiune programabilă (Programmable Interconnect Point – PIP) și este controlată de câte un bit dintr-o memorie RAM a circuitului FPGA. Această memorie este separată de rețeaua de celule de bază și este responsabilă pentru toate conexiunile active din FPGA. În momentul configurării unui FPGA, memoria RAM de conexiuni este scrisă corespunzător, definind astfel rețeaua de linii interconectate, în concordanță cu arhitectura de implementat.

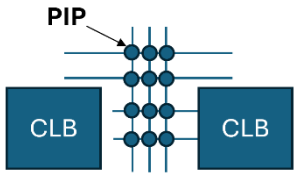
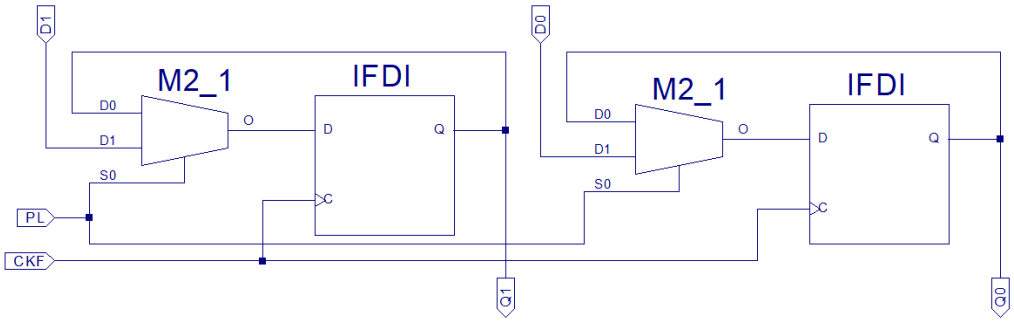


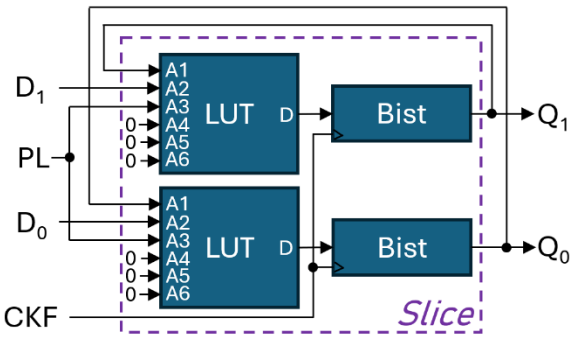
Figura 12. 2 Secvență din gridul de conexiuni programabile

12.2.1 Implementarea unui circuit în FPGA

În cazul implementării cu circuite FPGA, o arhitectură descrisă prin schema logică (sau limbaj de descriere hardware) este redefinită integral cu resursele hardware existente în celulele de bază, fără ca funcționalitatea rezultată să sufere modificări. Uneltele de sinteză și implementare analizează arhitectura propusă și realizează conversia, stabilind celulele CLB utilizate, configurarea lor internă la nivel de *sclice*-uri și conexiunile dintre ele. De exemplu, registrul pe 2 biți din Figura 12. 3 (a), poate fi implementat în cadrul unui *slice*, folosind 2 memorii LUT și 2 bistabile D, conectate ca în Figura 12. 3 (b). Memoriile LUT sunt configurate să funcționeze ca multiplexoare MUX 2:1, pe baza tabelului din Figura 12. 3 (c), reprezentând harta de biți corepunzătoare acestora. Astfel, un LUT va expune pe ieșirea D valoarea bitului de adresă A_1 sau A_2, în funcție de valoarea selecției conectate pe A_3. Biții de adresă A_4, A_5, A_6 vor fi conectați la 0, în interiorul *slice*-ului și nu vor folosi fire de conexiune din rețea.



(a)



(b)

A_6	A_5	A_4	A_3	A_2	A_1	D
0	0	0	0	0	0	0
0	0	0	0	0	1	1
0	0	0	0	1	0	0
0	0	0	0	1	1	1
0	0	0	1	0	0	0
0	0	0	1	0	1	0
0	0	0	1	1	0	1
0	0	0	1	1	1	1
0	0	1	X	X	X	X
0	1	X	X	X	X	X
1	X	X	X	X	X	X

(c)

Figura 12. 3 Implementarea unui registru pe 2 biți (a) folosind un *slice* în FPGA (b) și harta de biți a unei memorii LUT 64x1 (c) configurată ca MUX 2:1

Etapele de sinteză și implementare premerg pasul de generare a fișierului de configurare .bit. În Project Navigator, acest lucru este evidențiat în Figura 12. 4. Pocesul de programare cu fișierul .bit, folosind utilitarul ISE iMPACT, constă în scrierea memoriei RAM pentru conexiunile programabile, și scrierea memoriilor LUT din *slice*-urile alocate, cu hărțile de biți corespunzătoare funcțiilor booleene de implementat.

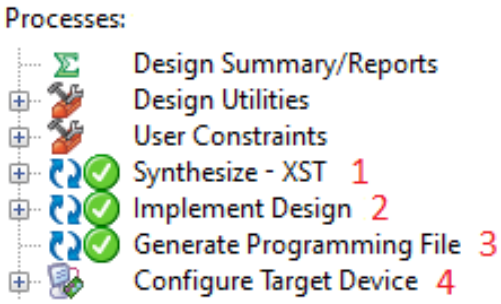


Figura 12. 4 Pașii de implementare a unui circuit în Project Navigator

12.2.2 Generarea simbolului unui circuit

Un circuit se poate încapsula într-un simbol reprezentativ, pentru utilizare ulterioară în cadrul altor circuite. Generarea simbolului în Project Navigator se realizează selectând schema în panoul Design. În partea inferioară, se extinde rubrica Design Utilities se apasă click dreapta pe opțiunea Create Schematic Symbol și se alege opțiunea Run. Pașii sunt prezentați în Figura 12. 5 – stânga. Simbolul generat va fi disponibil în librăria proiectului și va purta numele schemei. Pentru registrul pe 2 biți din Figura 12. 3 (a), aspectul simbolului va fi cel din Figura 12. 5 – dreapta. În cadrul simbolului de formă dreptunghiulară sunt evidențiați pinii reprezentând intrările și ieșirile registrului.

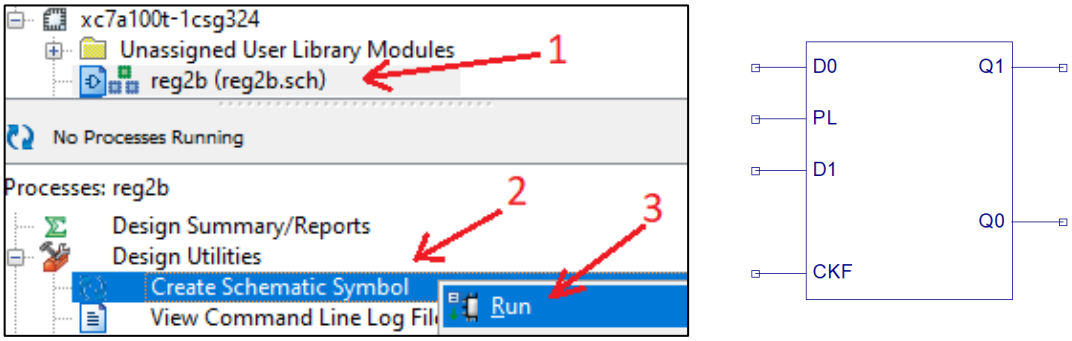


Figura 12. 5 Pașii de generare a simbolului unui circuit în Project Navigator (stânga) și simbolul generat pentru registrul pe 2 biți de la secțiunea anterioară (dreapta)

12.2.3 Simularea funcționării unui circuit

Modulul de simulare poartă denumirea ISim și se poate lansa din Project Navigator pentru oricare din circuitele dezvoltate. Pașii de lasare a simulatorului necesită selectarea opțiunii Simulation în partea superioară a panoului Design, urmată de selectarea schemei a cărei funcționalitate se va simula; se va extinde rubrica ISim Simulator în partea inferioară și aplicând click dreapta pe opțiunea Simulate Behavioral Model, se

alege opțiunea Run, ca în Figura 12. 6. După lansare, utilitarul ISim va conține o fereastră cu forme de undă pentru semnalele circuitului.

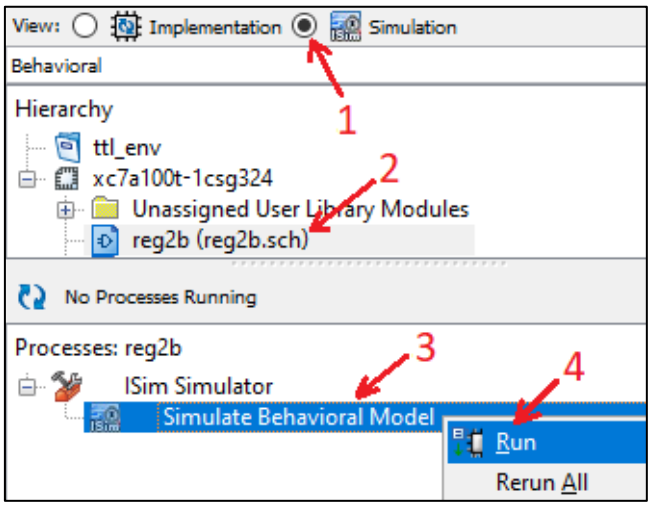
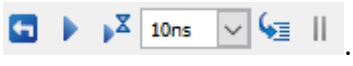


Figura 12. 6 Lansarea simulării în Project Navigator pentru un circuit

Pentru registrul din Figura 12. 3 (a) lista de semnale va avea în componența sa intrările și ieșirile circuitului, ca în Figura 12. 7, eventual alte semnale interne. Inițial, valorille intrărilor sunt necunoscute, motiv pentru care în dreptul lor apare valoarea U (Unknown).

Name	Value
ckf	U
d0	U
d1	U
pl	U
q0	U
q1	U

Figura 12. 7 Lista semnalelor unui circuit în fereastra de simulare și valorile lor

Pentru aducerea simulării la momentul inițial 0 se va apăsa butonul Restart  în bara de control din partea superioară: 

. Fiecărui semnal i se pot atribui valori, cu click dreapta pe numele lor și opțiunea Force Constant…. În fereastra ulterioară, se stabilește valoarea dorită la rubrica Force to Value (Figura 12. 8).

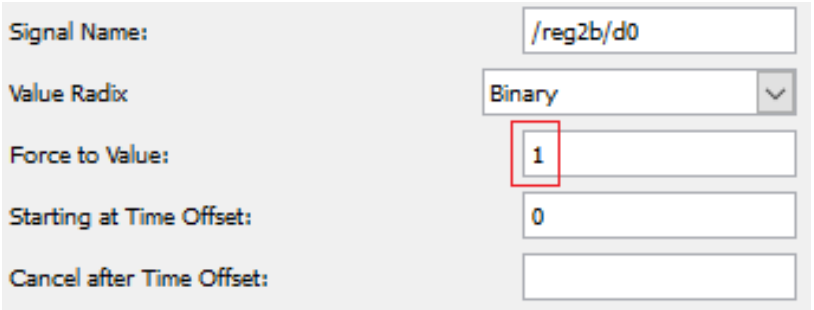


Figura 12. 8 Setarea unui semnal din simulator la valoarea 1

Semnalul de ceas se poate inițializa alegând Force Clock…, în loc de Force Constant…. În fereastra asociată (Figura 12. 9), se setează atributele Leading Edge Value = 0, Trailing Edge Value = 1 și Period = 10ns (echivalentul unei frecvențe de 100MHz). **Notă**: Perioada ceasului Period poate primi orice valoare.

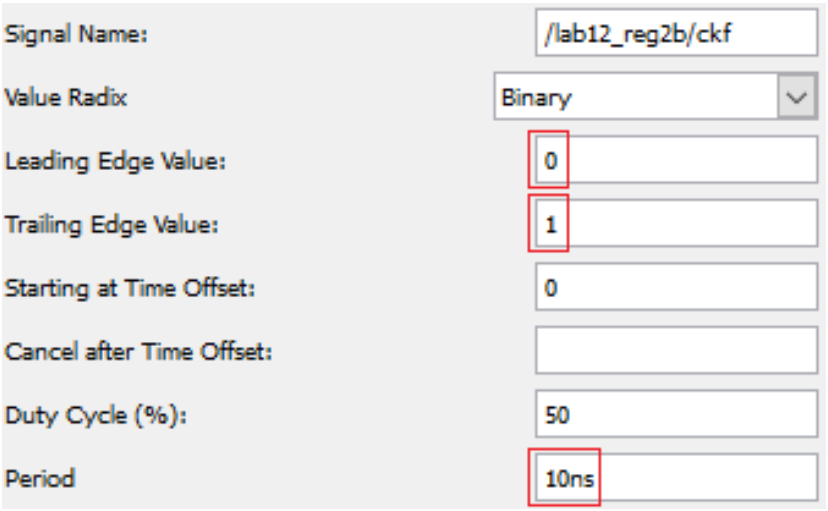


Figura 12. 9 Configurarea în simulator a unui semnal de ceas cu perioada de 10ns

Simularea se realizează pe pași, cu o durată configurabilă. Pentru simplificare, un pas se poate configura cât intervalul unui impuls de ceas, adică 10ns. Valoarea se introduce în bara de control menționată anterior: 

. Execuția unui pas de simulare are loc la apăsarea butonului  . La fiecare apăsare se vor afișa formele de undă corespunzătoare semnalelor, pe durata a 10ns. Valorile intrărilor se pot modifica înainte de fiecare pas, cu opțiunea Force Constant…. Figura 12. 10 prezintă un exemplu de simulare a comportamentului registrului pe 2 biți, în 4 pași, pe durata a 40ns. În primul ciclu, se scrie valoarea 2 pe frontul ascendent al CKF, prin setarea PL=1 și D_1D_0=10. În ciclurile următoare, prin dezactivarea PL=0 se observă că starea registrului Q_1Q_0 rămâne nealterată, indiferent de valorile pregătite pe D_1D_0. **Notă**: Pentru îmbunătățirea aspectului, dimensiunea formelor de undă se poate modifica folosind comenzile Zoom In  , Zoom Out  , respectiv Zoom to Full View  din bara de control.

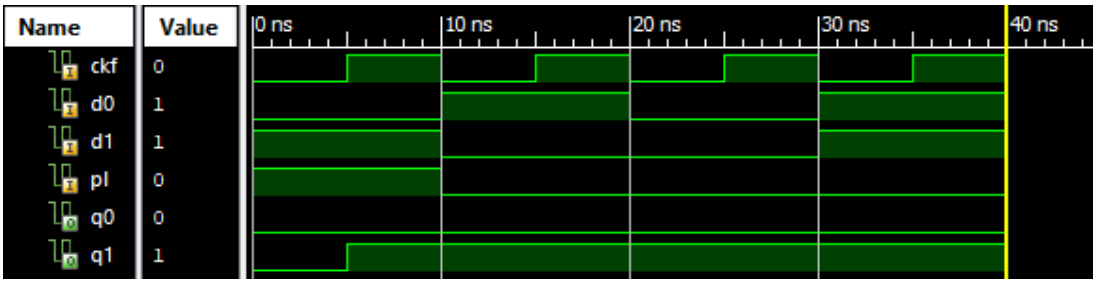


Figura 12. 10 Testarea unui registru pe 2 biți folosind simulatorul ISim

În panoul de simulare se pot adăuga semnale prin *drag&drop* din fereastra alăturată Objects (Figura 12. 11 – stânga). Semnalele se pot elimina cu click dreapta pe acestea și opțiunea Delete (Figura 12. 11 – dreapta).

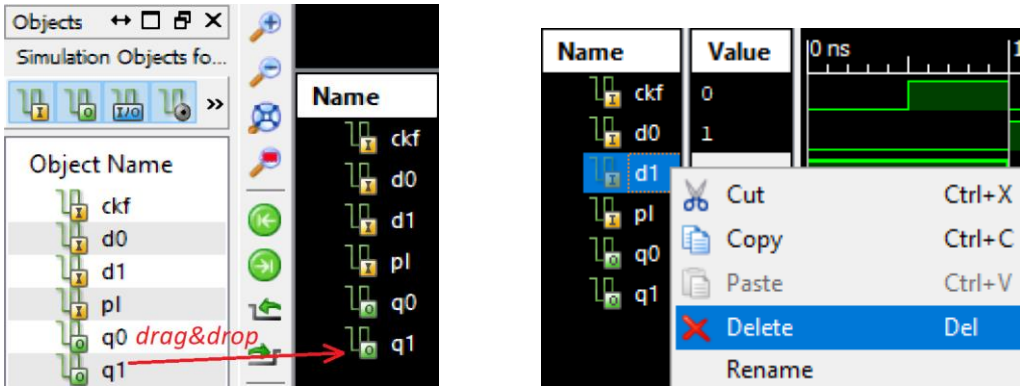
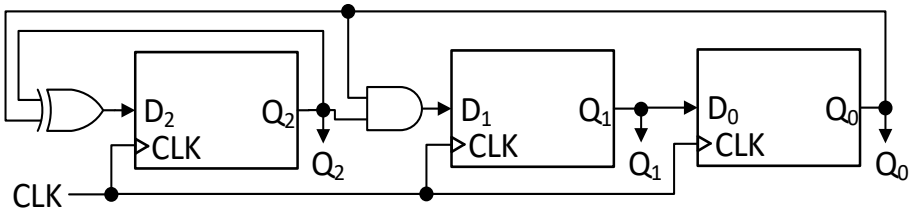


Figura 12. 11 Adăugarea semnalelor pentru simulare în panoul cu formele de undă (stânga) și eliminarea semnalelor de pe panou (dreapta)

12.3 Activități practice

1. Implementați cu *slice*-uri circuitul din schema următoare și definiți hărțile de biți ale memoriilor LUT implicate:

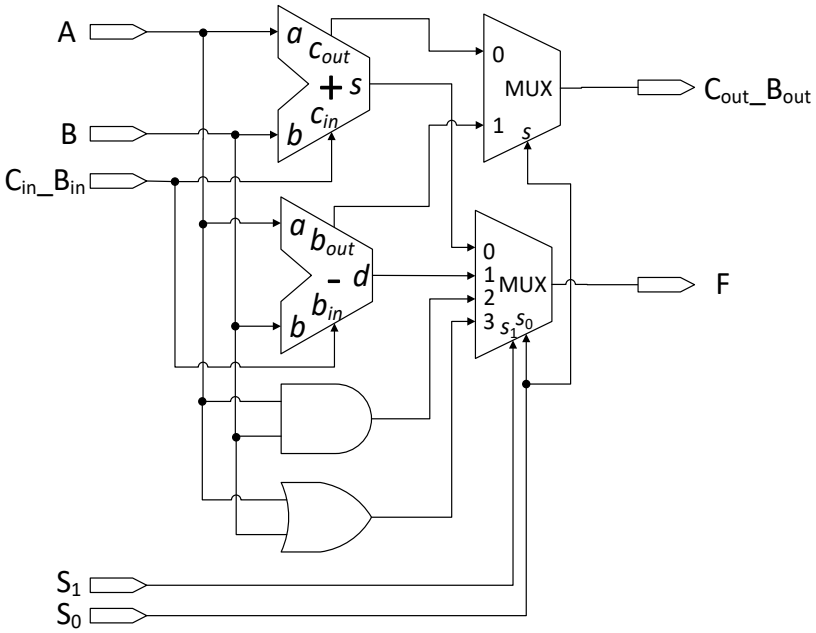


2. Implementați în Project Navigator registrul pe 2 biți din Figura 12. 3 (a), creați simbolul circuitului și simulați funcționarea sa cu ISim.
3. Implementați în Project Navigator un sumator complet pe 1 bit, folosind porți logice fundamentale. Generați simbolul circuitului și realizați, într-o nouă diagramă, un sumator complet pe 3 biți, prin cascadarea celui pe 1 bit. Testați funcționarea sumatorului pe 3 biți folosind simulatorul ISim.

A. Anexa 1 – Probleme cu circuite logice combinaționale

1. Să se proiecteze o UAL (Unitate Aritmetică-Logică) cu 2 operanzi A și B, a câte 1 bit, care execută următoarele operații în funcție de valoarea de selecție S_1S_0: 00 -> adunare fără semn; 01 -> scădere (A-B) fără semn; 10 -> ȘI; 11 -> SAU. Unitatea va avea un semnal comun Carry/Borrow la intrare și ieșire.

Rezolvare: UAL (ALU – Arithmetic Logic Unit) se întâlnește în majoritatea microprocesoarelor și este capabilă să implementeze un număr ridicat de operații aritmetice și logice. Funcțional, operațiile se realizează concomitent, dar pe ieșire se va pune la dispoziție doar rezultatul uneia din operații, în funcție de valorile de selecție. Deoarece se implementează 4 operații, selecția rezultatului pe ieșirea F se va realiza cu un multiplexor MUX 4:1. Pentru Carry și Borrow se folosesc semnale comune, atât la intrare cât și la ieșire, sub forma C_{in}_B_{in}, respectiv C_{out}_B_{out}. La ieșire este suficient un MUX2:1, care selectează între Carry sau Borrow, în funcție de S_0. Se observă faptul că pentru S_1 = 1 ieșirea C_{out}_B_{out} nu are sens, fiindcă se realizează una din operațiile AND sau OR. În timpul testării circuitului, ieșirea C_{out}_B_{out} are sens dacă S_1 = 0, altfel se va ignora.



Sumatorul pe 1 bit se realizează cu porți logice fundamentale conform expresiilor:

$$c_{out} = (b \cdot c_{in}) + (a \cdot c_{in}) + (a \cdot b)$$

$$s = a \oplus b \oplus c_{in}$$

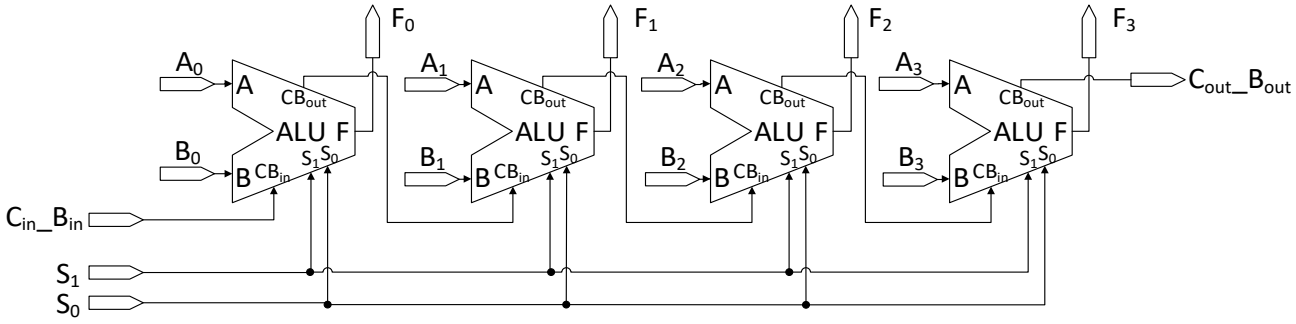
Scăzătorul pe 1 bit se realizează cu porți logice fundamentale conform expresiilor:

$$b_{out} = (\bar{a} \cdot b_{in}) + (\bar{a} \cdot b) + (b \cdot b_{in})$$

$$d = a \oplus b \oplus b_{in}$$

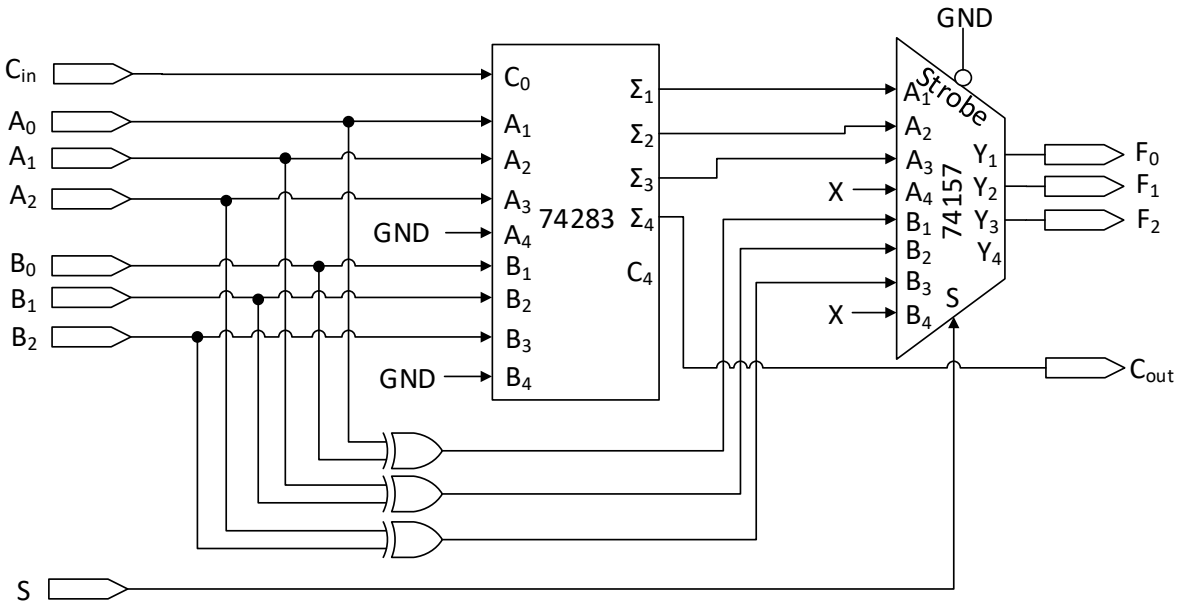
2. Să se proiecteze o UAL (Unitate Aritmetică-Logică) cu 2 operanzi A și B, a câte 4 biți, folosind mai multe UAL pe 1 bit (vezi Problema 1), care execută următoarele operații în funcție de valoarea de selecție S_1S_0: 00 -> adunare fără semn; 01 -> scădere (A-B) fără semn; 10 -> ȘI pe biți; 11 -> SAU pe biți. Unitatea va avea un semnal comun Carry/Borrow la intrare și ieșire.

Rezolvare: UAL pe mai mulți biți se poate poate realiza prin cascadarea în serie a mai multor UAL pe mai puțini biți. Semnalele de selecție sunt folosite în comun de toate unitățile UAL cascadate. Semnalul comun de intrare Carry/Borrow se aplică pe UAL de rang cel mai puțin semnificativ și se preia de la UAL cel mai semnificativ. Semnalele Carry/Borrow intermediare sunt conectate de la ieșirea unui UAL de rang inferior către intrarea UAL de rang imediat superior. Operațiile se realizează în paralel pe toți biții, însă propagarea semnalului Carry/Borrow are loc serial, ceea ce crește timpul după care rezultatul devine stabil pentru citire.



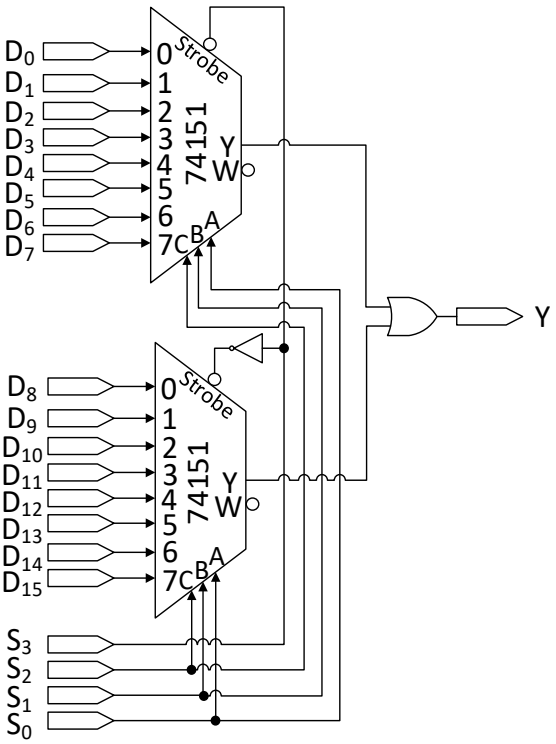
3. Să se proiecteze o UAL (Unitate Aritmetică-Logică) cu 2 operanzi A și B, a câte 3 biți, care execută următoarele operații în funcție de valoarea de selecție S: 0 -> adunare fără semn; 1 -> A⊕B pe biți. Unitatea va avea un semnal de Carry la intrare și la ieșire. Se va folosi sumatorul 74283 și multiplexorul 74157.

Rezolvare: Fiind vorba de o UAL cu două operații, selecția rezultatului expus pe ieșirea F se face cu multiplexorul MUX 2:1 (74157). Pe baza selecției S, se alternează între rezultatul adunării și rezultatul operației XOR. Adunarea se va implementa cu integratul 74283. Se vor utiliza cei 3 biți mai puțin semnificativi ai circuitelor, iar ceilalți vor fi conectați la 0 (GND), pentru a nu afecta rezultatul, în cazul adunării. În cazul multiplexorului 74157, intrările A_4 și B_4 nu contează (X = se conectează la GND sau VCC).



4. Să se proiecteze un MUX 16:1 folosind 2 multiplexoare 74151 și cel mult încă 2 porți logice fundamentale.

Rezolvare: Multiplexoarele 74151 au intrarea *Strobe*, care poate fi folosită pentru a activa selectiv una din cele 2 unități, cealaltă rămânând inactivă. Intrarea de selecție cea mai semnificativă S_3 va determina unitatea activă, conectând-o direct la *Strobe* pentru un multiplexor și inversată pentru al 2-lea. Cele 2 multiplexoare vor utiliza în comun intrările de selecție S_2, S_1, S_0. Peste ieșirile celor 2 unități 74151 se aplică operația SAU pentru a genera valoarea finală. Astfel, pentru numerele de combinație 0-7 reprezentate pe $S_{3:0}$, se activează multiplexorul superior, iar pentru numerele de combinație 8-15 se activează multiplexorul inferior. Multiplexorul rămas inactiv va avea pe ieșire valoarea 0 și nu va afecta valoarea finală datorită operației SAU.



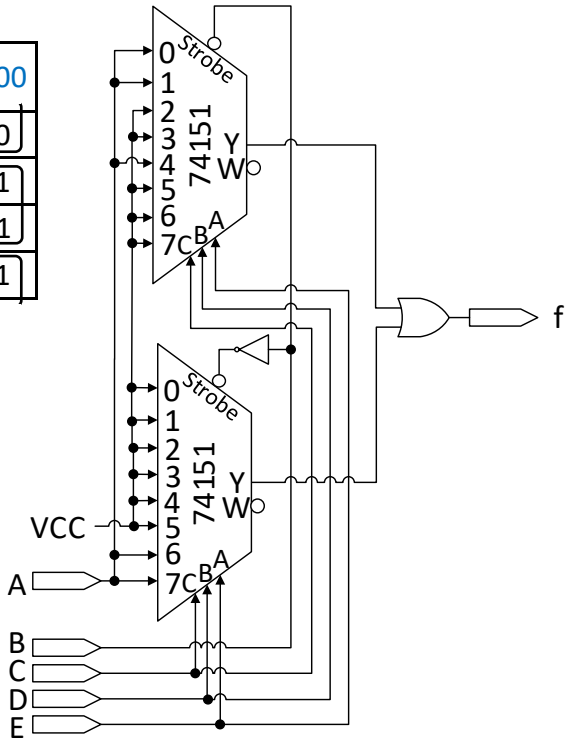
5. Să se implementeze cu multiplexoare 74151 funcția: $f(A, B, C, D, E) = A + \overline{C} \cdot D + B \cdot \overline{D} + \overline{B} \cdot D + \overline{B} \cdot C \cdot E$. Sunt disponibile semnalele 0 (GND), 1 (VCC) şi variabilele numai în forma directă, nu şi negate.

Rezolvare: Deoarece variabila A nu apare negată în expresie, se aleg variabilele B, C, D, E ca intrări de selecție într-un MUX 16:1. În consecință, celulele din diagrama Karnaugh se grupează după perechi de valori comune pentru B, C, D, E în dreptul lor. Dacă o grupare conține aceeași valoare, aceasta se aplică pe intrarea de date a multiplexorului asociată selecției B, C, D, E. Dacă o grupare conține valori distincte atunci rezultatul pe intrarea de date asociată selecției B, C, D, E se calculează în funcție de celula care conține valoarea 1, în felul următor:

- Dacă celula corespunde unei valori de 0 pentru A atunci pe intrare se aplică $\overline{A}$.
- Dacă celula corespunde unei valori de 1 pentru A atunci pe intrare se aplică A.

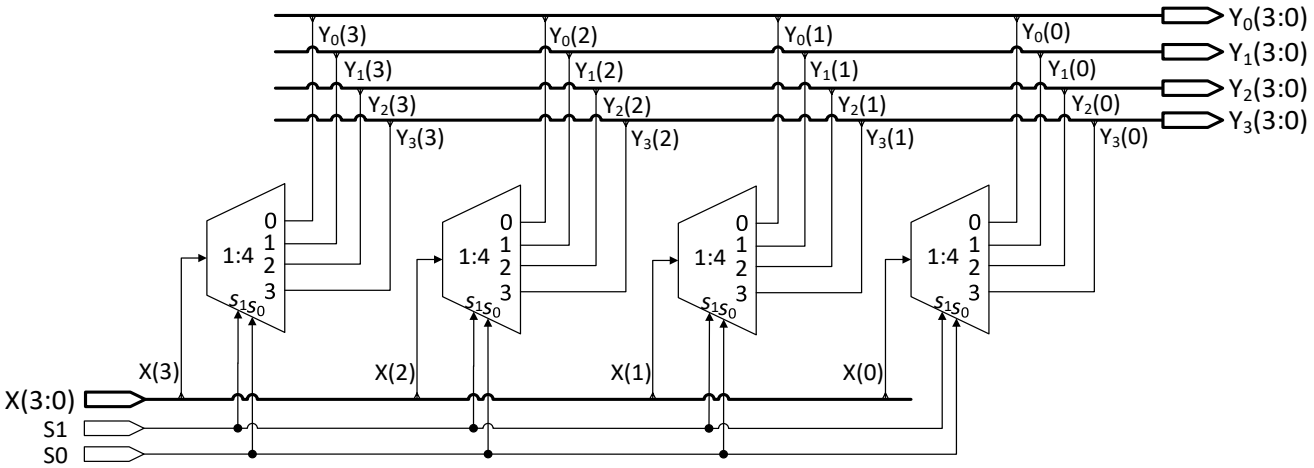
Notă: Implementarea unui MUX 16:1 cu 2 unități 74151 a fost prezentată în problema anterioară.

AB \ CDE	000	001	011	010	110	111	101	100
00	0	0	1	1	1	1	1	0
01	1	1	1	1	0	0	1	1
11	1	1	1	1	1	1	1	1
10	1	1	1	1	1	1	1	1



6. Să se proiecteze un DMUX 1:4 cu lățimea căii de date pe 4 biți folosind demultiplexoare cu lățimea căii de date pe 1 bit.

Rezolvare: Implementarea unui DMUX 1:4 cu calea de date pe 4 biți necesită folosirea a 4 unități DMUX 1:4 cu calea de date pe 1 bit. Unitățile DMUX pe 1 bit vor folosi în comun semnalele de selecție S_1, S_0 și vor realiza demultiplexarea biților cu același rang de pe cele 4 ieșiri. Primul DMUX va realiza demultiplexarea pentru biții de rang 0, următorul DMUX va realiza demultiplexarea pentru biții de rang 1, etc. Intrările celor 4 unități DMUX vor fi conectate la biții de intrare ai căii de date.



7. Să se proiecteze cu porți logice fundamentale un codificator prioritar 4:2 la care intrarea cea mai prioritară este cea cu indexul 3. Toate intrările și ieșirile lucrează în logica pozitivă. Dacă nicio intrare nu este activă se returnează 00.

Rezolvare: Se scrie tabelul de adevăr și se minimizează ieșirile cu Diagrame Karnaugh. Valorile X în tabelul de adevăr se înlocuiesc cu toate combinațiile de 0 și 1 atunci când se face completarea diagramelor Karnaugh, generând astfel mai multe combinații, care vor corespunde la mai multe celule din diagramă.

Tabelul de adevăr

X_3	X_2	X_1	X_0	Y_1	Y_0
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

Y_1:

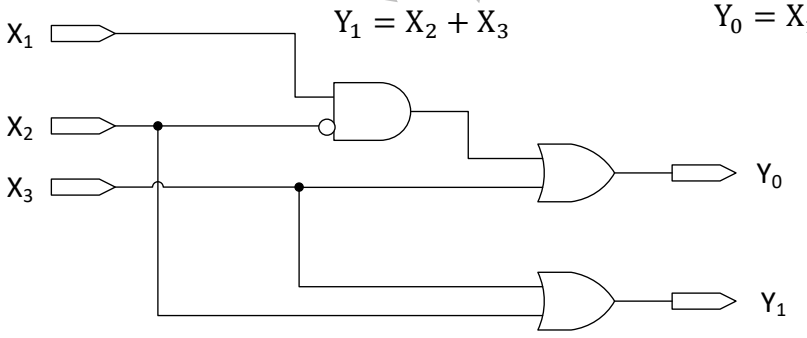
X_3X_2 \ X_1X_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

Y_0:

X_3X_2 \ X_1X_0	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$Y_1 = X_2 + X_3$

$Y_0 = X_1 \cdot \overline{X_2} + X_3$



8. Să se implementeze funcția: $f = \sum(0,1,2,4,6)$ în 3 variante posibile.

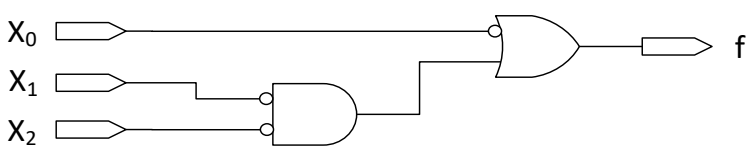
Rezolvare: Fiindcă mintermul corespunzător celui mai mare număr de combinație este 6, înseamnă că este o funcție de 3 variabile.

Varianta 1

Se implementează cu porți logice fundamentale, ceea ce presupune o minimizare a funcției la forma disjunctivă minimă (FDM).

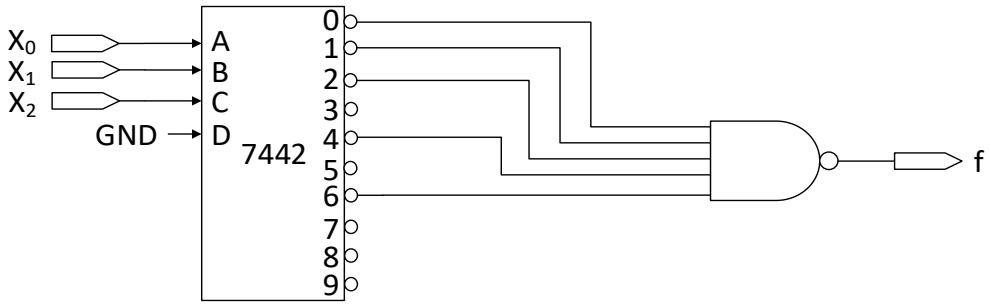
X_2 \ X_1X_0	00	01	11	10
0	1	1	0	1
1	1	0	0	1

$f = \overline{X_0} + \overline{X_2} \cdot \overline{X_1}$



Varianta 2

Se implementează cu decodificator și se folosește forma canonică disjunctivă (FCD). Avem la dispoziție decodificatorul BCD-zecimal 7442, care implementează mintermii inversați la ieșire. Aplicând ȘI-NU peste ieșirile corespunzătoare mintermilor inversați ai funcției, se obține disjuncția acestora (cf. De Morgan), deci funcția în forma canonică disjunctivă. Se vor folosi cele 3 intrări mai puțin semnificative ale decodificatorului pentru variabile, iar a 4-a va fi conectată la 0 (GND).

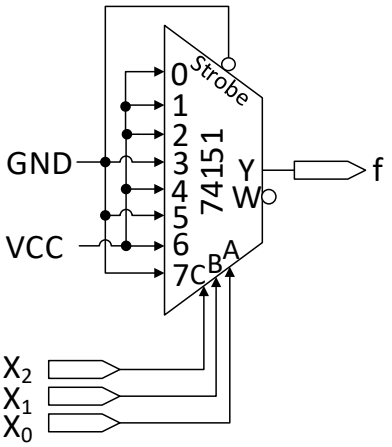


Varianta 3

Se implementează cu integratul 74151 (un multiplexor MUX 8:1), care va avea conectate pe selecțiile C, B, A cele 3 variabile de intrare, cu același rang. Se folosește forma canonică disjunctivă (FCD) formată din disjuncția mintermilor vizibili în tabelul de adevăr. Pe intrările de date ale multiplexorului, corespunzătoare mintermilor, se conectează 1 (VCC), iar pe celelalte, 0 (GND). Se poate copia coloana valorilor de ieșire, din tabelul de adevăr, la intrările de date ale multiplexorului.

Tabelul de adevăr

	x_2	x_1	x_0	f_2
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	0



9. Să se implementeze funcția: $f = (\overline{A + B} \oplus B) \cdot C$ numai cu porți ȘI-NU.

Rezolvare: Pentru a implementa numai cu porți ȘI-NU orice funcție se aduce la forma disjunctivă minimă (FDM) folosind minimizare cu Karnaugh. Diagrama Karnaugh se stabilește din tabelul de adevăr, care se construiește dând valori variabilelor pentru toate combinațiile de intrare posibile. Pe FDM se aplică dubla negație și apoi relația De Morgan, rezultatul fiind o funcție care conține numai porți ȘI-NU. Dacă în expresie apare o variabilă

de intrare în forma negată aceasta se poate obține, folosind poarta ȘI-NU, cu una din relațiile: $\overline{X} = \overline{X \cdot X} = \overline{X \cdot 1}$.

Tabelul de adevăr

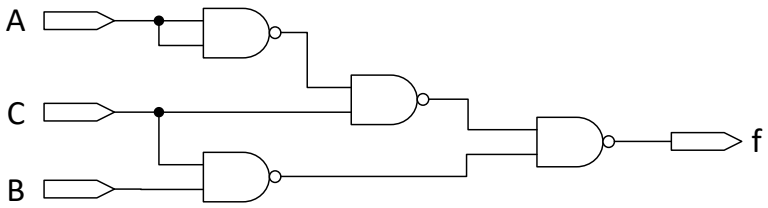
A	B	C	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

A \ BC	00	01	11	10
0	0	1	1	1
1	0	0	1	1

$$f \overset{FDM}{\Longrightarrow} \overline{A} \cdot C + B \cdot C = \overline{\overline{\overline{A} \cdot C + B \cdot C}} = \overline{\overline{\overline{A} \cdot C} \cdot \overline{B \cdot C}} = \overline{\overline{\overline{A \cdot A} \cdot C} \cdot \overline{B \cdot C}}$$

Dubla negație

De Morgan

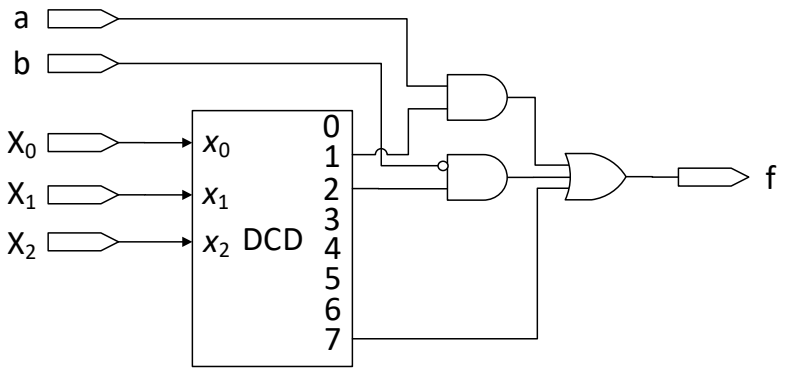


10. Să se implementeze cu decodificator, având semnalele în logica pozitivă, funcția f cu expresii înglobate definită de următoarea diagramă Karnaugh.

X_2 \ X_1X_0	00	01	11	10
0	0	a	0	$\overline{b}$
1	X	X	1	0

Rezolvare: Implementarea cu decodificator se realizează exprimând funcția în forma canonică disjunctivă (FCD). În acest sens, diagrama Karnaugh reprezintă o disjuncție OR a mintermilor corespunzători celulelor care conțin valori de 1. Atunci când o celulă conține o expresie înglobată înseamnă că mintermul corespunzător celulei realizează o conjuncție ȘI cu expresia din celulă. Celulele din diagramă, care conțin valoarea X, se vor înlocui cu valoarea 0, pentru a reduce numărul de mintermi implicați, ceea ce simplifică expresia în FCD a funcției. Așadar, funcția se poate scrie: $f = a \cdot P_1 + \overline{b} \cdot P_2 + 1 \cdot P_7$, unde $P_1 = \overline{X_2} \cdot \overline{X_1} \cdot X_0$, $P_2 = \overline{X_2} \cdot X_1 \cdot \overline{X_0}$, $P_7 = X_2 \cdot X_1 \cdot X_0$ sunt mintermi de variabilele X_2, X_1, X_0. Variabilele X_2, X_1, X_0 vor fi intrările decodificatorului, în consecință ieșirile vor reprezenta mintermii asociați acestora. Se utilizează doar ieșirile corespunzătoare mintermilor implicați în disjuncția OR (mintermii pentru care celula din diagramă conține 1 sau o expresie înglobată). La nivelul acestor ieșiri se realizează conjuncția ȘI cu expresia înglobată, unde este cazul, iar apoi se realizează o disjuncție OR peste toate ramurile

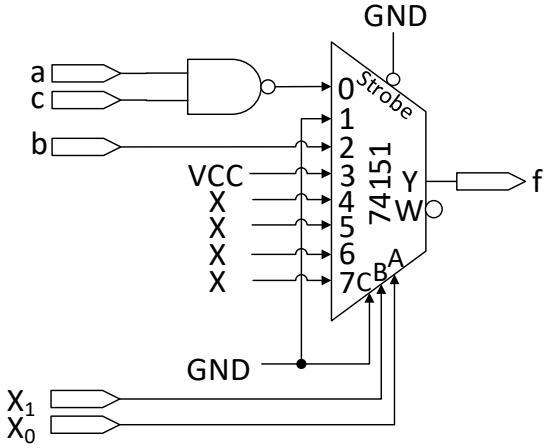
implicate. Rezultatul disjuncției oferă valoarea funcției. **Notă**: Implementarea se poate realiza direct pe baza conținutului diagramei Karnaugh, fără a mai rescrie funcția în FCD.



11. Să se implementeze cu multiplexorul 74151 funcția f cu expresii înglobate definită de următoarea diagramă Karnaugh.

$X_1 X_0$	0	1
0	$\overline{a \cdot c}$	X
1	b	1

Rezolvare: Implementarea cu multiplexor se realizează exprimând funcția în forma canonică disjunctivă (FCD). Diagrama Karnaugh reprezintă o disjuncție OR a mintermilor corespunzători celulelor care conțin valori de 1. Atunci când o celulă conține o expresie înglobată înseamnă că mintermul corespunzător celulei realizează o conjuncție ȘI cu expresia din celulă. Celulele din diagramă, care conțin valoarea X, se înlocuiesc cu valoarea 0, pentru a reduce numărul de mintermi implicați, ceea ce simplifică expresia în FCD a funcției. Așadar, funcția se poate scrie: $f = \overline{a \cdot c} \cdot P_0 + b \cdot P_2 + 1 \cdot P_3$, unde $P_0 = \overline{X_1} \cdot \overline{X_0}$, $P_2 = X_1 \cdot \overline{X_0}$, $P_3 = X_1 \cdot X_0$ sunt mintermi de variabilele X_1, X_0. Deoarece multiplexorul 74151 are 3 semnale de selecție, ambele variabile X_1, X_0 vor fi conectate la selecțiile mai puțin semnificative ale acestuia. Selecția mai semnificativă se conectează la 0 (GND). Conținutul celulelor din diagrama Karnaugh va indica ce se conectează pe intrările 0-3 corespunzătoare ale multiplexorului. Fiindcă C = 0, intrările 4-7 nu contează (X = se pot conecta la GND sau VCC). **Obs**: Implementarea se poate realiza direct pe baza conținutului diagramei Karnaugh, fără a mai rescrie funcția în FCD.



12. Să se implementeze cu porți logice fundamentale funcția cu expresii înglobate $f(A,B,C,D) = \bar{b} \cdot P_2 + P_7 + (a \oplus c) \cdot P_{13} + P_{14} + P_{15} + \sum_{\Phi}(1,3,4,5,6,8,9,10,11)$, unde P_i reprezintă mintermii de variabilele A, B, C, D (în ordine, A este cea mai semnificativă și D este cea mai puțin semnificativă).

Rezolvare: Implementarea cu porți logice se realizează aducând funcția la forma disjunctivă minimă (FDM) cu ajutorul diagramei Karnaugh. Se compune diagrama Karnaugh punând expresiile înglobate în celulele asociate mintermilor cu care acestea apar în conjuncție.

AB \ CD	00	01	11	10
00	0	X	X	$\bar{b}$
01	X	X	1	X
11	0	$a \oplus c$	1	1
10	X	X	X	X

Minimizare

Pas₁: Se înlocuiesc expresiile înglobate cu 0 și se face un număr minim de grupări dreptunghiulare maximale (de dimensiune putere a lui 2) cu valori de 1 și X, care acoperă toate celulele cu 1.

Pas₂: Se înlocuiesc valorile 1 cu X și se face un număr minim de grupări dreptunghiulare maximale (de dimensiune putere a lui 2) cu expresiile înglobate și X-uri, care acoperă toate celulele cu expresii. O grupare nu are voie să conțină două expresii înglobate. Se realizează o conjuncție ȘI între expresia înglobată și termenul rezultat din grupul care conține expresia.

Pas₃: Se realizează disjuncția SAU între termenii obținuți la pașii anteriori.

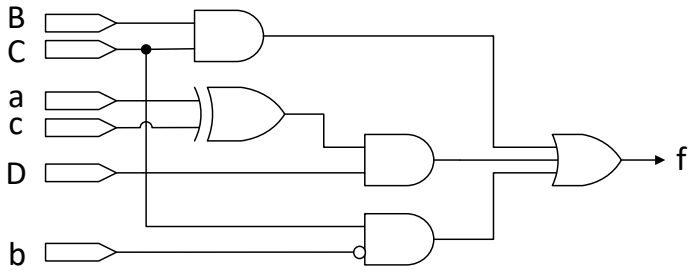
Pas₁

AB \ CD	00	01	11	10
00	0	X	X	0
01	X	X	1	X
11	0	0	1	1
10	X	X	X	X

Pas₂

AB \ CD	00	01	11	10
00	0	X	X	$\bar{b}$
01	X	X	X	X
11	0	$a \oplus c$	X	X
10	X	X	X	X

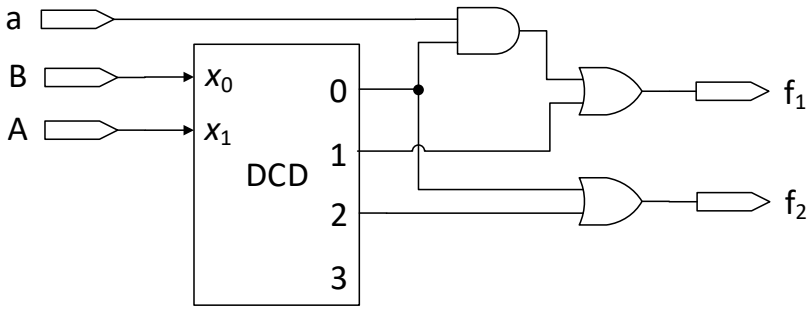
Pas₃ $f = B \cdot C + (a \oplus c) \cdot D + \bar{b} \cdot C$



13. Să se implementeze cu decodificator având semnalele în logica pozitivă, funcțiile cu expresii înglobate $f_1(A,B) = a \cdot P_0 + P_1$, $f_2(A,B) = \sum(0,2)$, unde P_i reprezintă mintermii

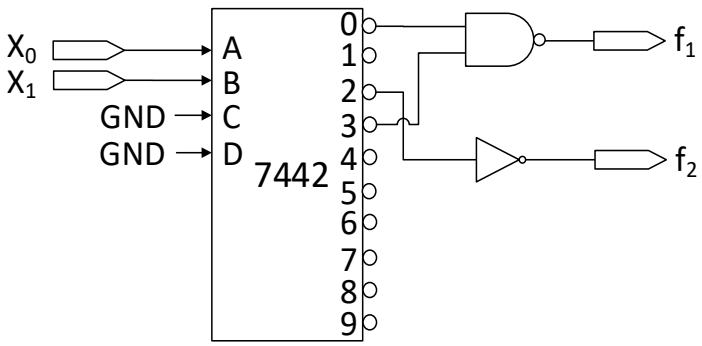
de variabilele A, B (în ordine, A este cea mai semnificativă și B este cea mai puțin semnificativă).

Rezolvare: Implementarea cu decodificator se realizează exprimând funcția în forma canonică disjunctivă (FCD). Fiindcă ambele funcții sunt definite prin mintermii variabilelor A și B, ele sunt în forma canonică disjunctivă. Deoarece au variabile comune se poate folosi același decodificator. Variabilele A, B vor fi intrările decodificatorului și ieșirile vor reprezenta mintermii asociați acestora. Pentru fiecare funcție se utilizează doar ieșirile corespunzătoare mintermilor necesari. La nivelul acestor ieșiri se realizează conjuncția ȘI cu expresia înglobată, unde este cazul, iar apoi se realizează o disjuncție OR peste toate ramurile implicate.



14. Să se implementeze cu decodificatorul 7442 funcțiile $f_1 = \sum(0,3)$ și $f_2 = \sum(2)$.

Rezolvare: Implementarea cu decodificator se realizează exprimând funcția în forma canonică disjunctivă (FCD). Fiindcă ambele funcții sunt definite prin disjuncție de mintermi, ele sunt în forma canonică disjunctivă. Se poate folosi același decodificator pentru implementarea ambelor funcții. Avem la dispoziție integratul 7442, un decodificator BCD-zecimal, care implementează mintermii inversați la ieșire. Aplicând ȘI-NU peste ieșirile corespunzătoare mintermilor din f_1 se obține disjuncția lor (cf. De Morgan), în consecință funcția f_1. Peste ieșirea corespunzătoare mintermului care implementează funcția f_2 se aplică o inversare. Deoarece integratul 7442 este un decodificator BCD-zecimal cu 4 intrări, iar funcțiile au doar 2 variabile, se vor folosi cele 2 intrări mai puțin semnificative, iar celelalte vor fi conectate la 0 (GND). Ca urmare, mintermii relevanți se vor afla pe ieșirile 0-3.



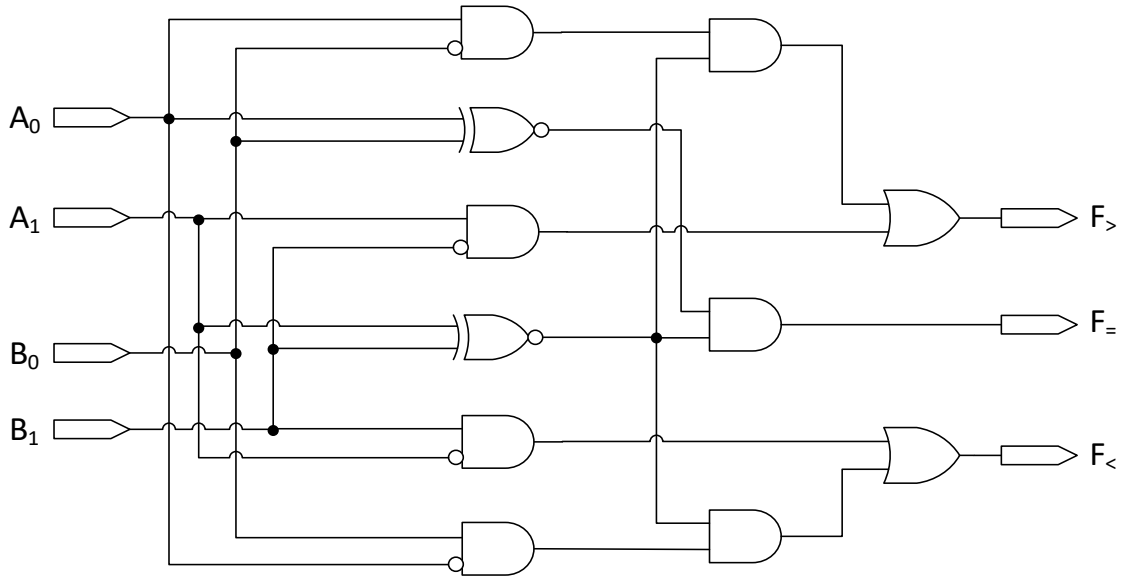
15. Să se proiecteze cu porți logice fundamentale un comparator care implementează relațiile >, <, = între 2 operanzi fără semn, pe 2 biți.

Rezolvare: O variantă de implementare ar fi să se construiască tabelul de adevăr care să conțină pe intrări cei 4 biți ai celor 2 operanzi și cele 16 combinații posibile, iar pe ieșiri cele 3 funcții. Se minimizează cu diagrama Karnaugh funcțiile $F_>$, $F_<$, $F_=$.

O altă variantă ar fi să se utilizeze proprietățile aritmetice ale numerelor fără semn și anume:

- $F_>$ = 1, dacă A > B, adică $A_1>B_1$ sau ($A_1=B_1$ și $A_0>B_0$) $\Leftrightarrow F_> = A_1 \cdot \overline{B_1} + (A_1 \odot B_1) \cdot (A_0 \cdot \overline{B_0})$
- $F_<$ = 1, dacă A < B, adică $A_1<B_1$ sau ($A_1=B_1$ și $A_0<B_0$) $\Leftrightarrow F_< = \overline{A_1} \cdot B_1 + (A_1 \odot B_1) \cdot (\overline{A_0} \cdot B_0)$
- $F_=$ = 1, dacă A = B, adică $A_1=B_1$ și $A_0=B_0$ $\Leftrightarrow F_= = (A_1 \odot B_1) \cdot (A_0 \odot B_0)$

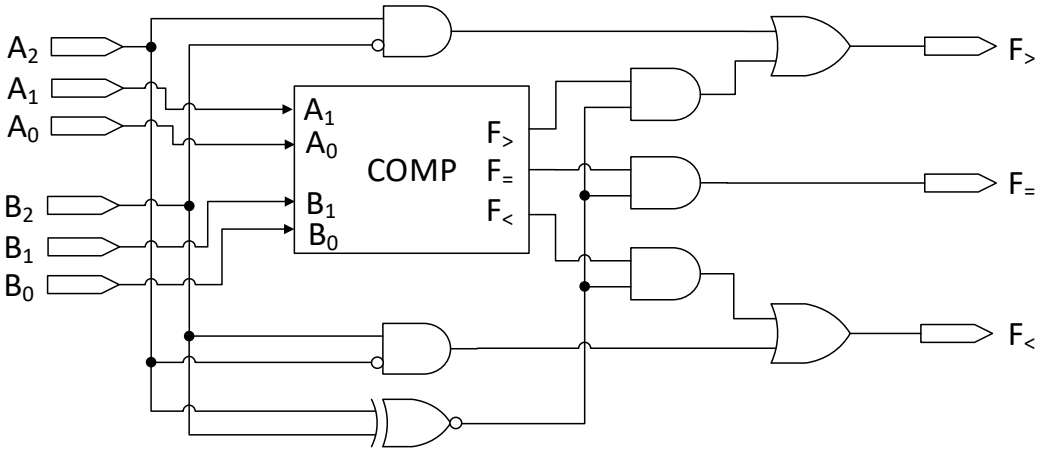
Obs: Chiar dacă expresiile obținute după minimizare ar fi mai simple decât cele de mai sus, trebuie ținut cont de faptul că minimizarea devine costisitoare dacă numărul de biți ar depăși 2. Ca urmare este de preferat varianta bazată pe proprietățile aritmetice.



16. Să se proiecteze un comparator care implementează relațiile >, <, = între 2 operanzi fără semn, pe 3 biți. Se vor utiliza porți logice fundamentale și comparatoare, care implementează relațiile pe operanzi fără semn, pe 2 biți.

Rezolvare: Se utilizează următoarele proprietăți ale numerelor fără semn:

- $F_>$ = 1, dacă A > B, adică $A_2>B_2$ sau ($A_2=B_2$ și $A_{1:0}>B_{1:0}$) $\Leftrightarrow F_> = A_2 \cdot \overline{B_2} + (A_2 \odot B_2) \cdot F_{>_{1:0}}$
- $F_<$ = 1, dacă A < B, adică $A_2<B_2$ sau ($A_2=B_2$ și $A_{1:0}<B_{1:0}$) $\Leftrightarrow F_< = \overline{A_2} \cdot B_2 + (A_2 \odot B_2) \cdot F_{<_{1:0}}$
- $F_=$ = 1, dacă A = B, adică $A_2=B_2$ și $A_{1:0}=B_{1:0}$ $\Leftrightarrow F_= = (A_2 \odot B_2) \cdot F_{=_{1:0}}$



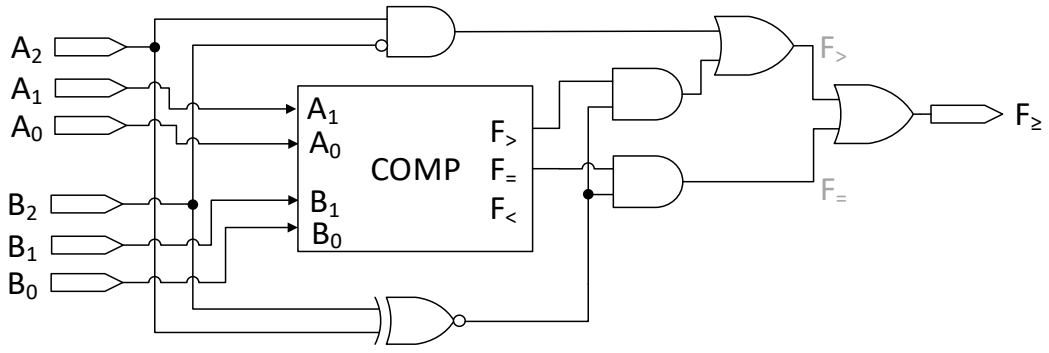
17. Să se proiecteze un comparator care implementează relația ≥ între 2 operanzi fără semn, pe 3 biți. Se vor utiliza porți logice fundamentale și comparatoare, care implementează relațiile pe operanzi fără semn, pe 2 biți.

Rezolvare:

Varianta 1

$F_{\geq}$ = 1, dacă A ≥ B, adică A=B sau A>B ⇔ $F_{\geq} = F_{>} + F_{=}$. Pentru a implementa $F_{>}$ și $F_{=}$ se utilizează următoarele proprietăți ale numerelor fără semn:

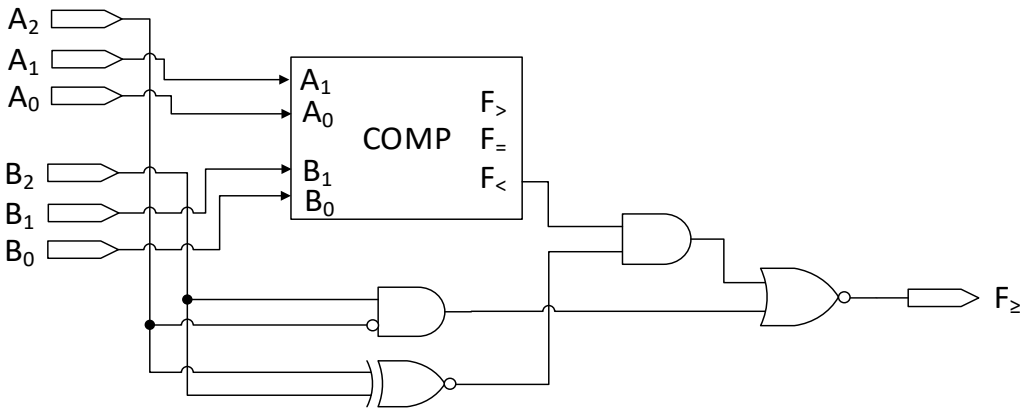
- $F_{>}$ = 1, dacă A > B, adică $A_2>B_2$ sau ($A_2=B_2$ și $A_{1:0}>B_{1:0}$) ⇔ $F_{>} = A_2 \cdot \overline{B_2} + (A_2 \odot B_2) \cdot F_{>_{1:0}}$
- $F_{=}$ = 1, dacă A = B, adică $A_2=B_2$ și $A_{1:0}=B_{1:0}$ ⇔ $F_{=} = (A_2 \odot B_2) \cdot F_{=_{1:0}}$



Varianta 2

$F_{\geq}$ = 1, dacă A ≥ B, adică not(A<B) ⇔ $F_{\geq} = \overline{F_{<}}$. Pentu a implementa $F_{<}$ se utilizează următoarea proprietate a numerelor fără semn:

- $F_{<}$ = 1, dacă A < B, adică $A_2<B_2$ sau ($A_2=B_2$ și $A_{1:0}<B_{1:0}$) ⇔ $F_{<} = \overline{A_2} \cdot B_2 + (A_2 \odot B_2) \cdot F_{<_{1:0}}$

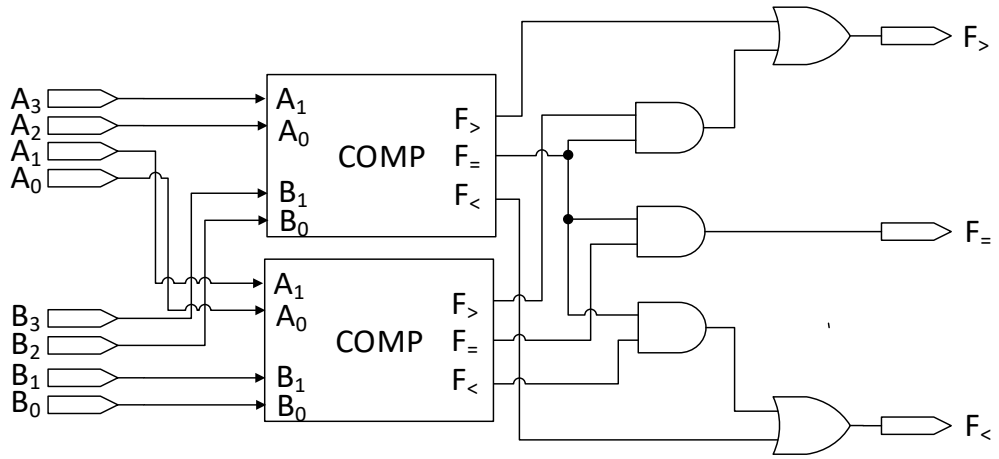


18. Să se proiecteze un comparator care implementează relațiile >, <, = între 2 operanzi fără semn, pe 4 biți. Se vor utiliza porți logice fundamentale și comparatoare care implementează relațiile pe operanzi fără semn, pe 2 biți.

Rezolvare: Se utilizează următoarele proprietăți ale numerelor fără semn:

- $F_{>}$ = 1, dacă A > B, adică $A_{3:2}>B_{3:2}$ sau ($A_{3:2}=B_{3:2}$ și $A_{1:0}>B_{1:0}$) $\Leftrightarrow F_{>} = F_{>_{3:2}} + F_{=_{3:2}} \cdot F_{>_{1:0}}$
- $F_{<}$ = 1, dacă A < B, adică $A_{3:2}<B_{3:2}$ sau ($A_{3:2}=B_{3:2}$ și $A_{1:0}<B_{1:0}$) $\Leftrightarrow F_{<} = F_{<_{3:2}} + F_{=_{3:2}} \cdot F_{<_{1:0}}$
- $F_{=}$ = 1, dacă A = B, adică $A_{3:2}=B_{3:2}$ și $A_{1:0}=B_{1:0}$ $\Leftrightarrow F_{=} = F_{=_{3:2}} \cdot F_{=_{1:0}}$

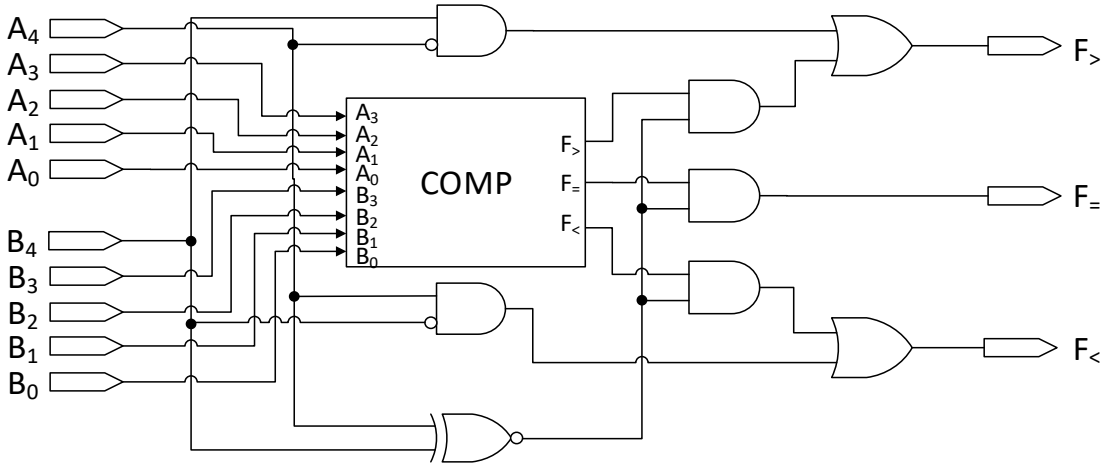
Se folosește un comparator pe 2 biți pentru biții 3:2 și un alt comparator pe 2 biți pentru biții 1:0.



19. Să se proiecteze un comparator care implementează relațiile >, <, = între 2 operanzi, numere cu semn, în complement față de 2, pe 5 biți. Se vor utiliza porți logice fundamentale și comparatoare, care implementează relațiile pe operanzi fără semn, pe 4 biți.

Rezolvare: Se utilizează următoarele proprietăți ale numerelor cu semn, în complement față de 2:

- $F_> = 1$, dacă A > B, adică (A_4=0 și B_4=1) sau (A_4=B_4 și $A_{3:0}$>$B_{3:0}$) $\Leftrightarrow F_> = \overline{A_4} \cdot B_4 + (A_4 \odot B_4) \cdot F_{>_{3:0}}$
- $F_< = 1$, dacă A < B, adică (A_4=1 și B_4=0) sau (A_4=B_4 și $A_{3:0}$<$B_{3:0}$) $\Leftrightarrow F_< = A_4 \cdot \overline{B_4} + (A_4 \odot B_4) \cdot F_{<_{3:0}}$
- $F_= = 1$, dacă A = B, adică A_4=B_4 și $A_{3:0}$=$B_{3:0}$ $\Leftrightarrow F_= = (A_4 \odot B_4) \cdot F_{=_{3:0}}$

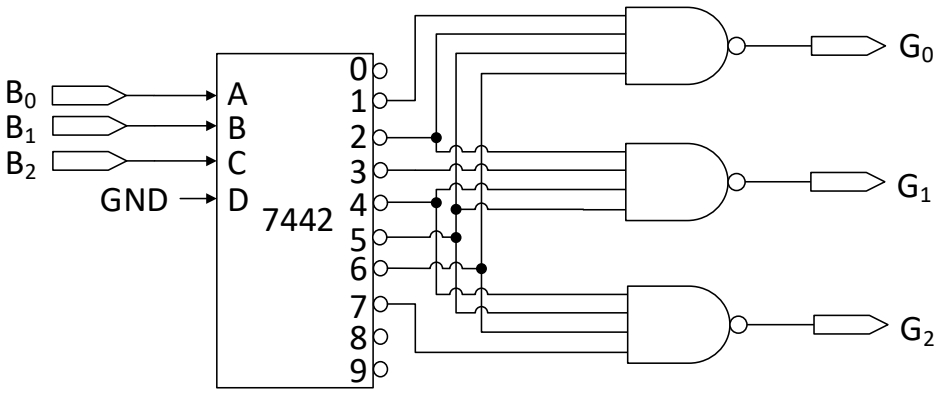


20. Să se proiecteze un convertor BCD-Gray pe 3 biți cu decodificatorul 7442 și alte porți logice fundamentale necesare.

Rezolvare: Pentru implementarea cu decodificator se folosește forma canonică disjunctivă (FCD) sau tabelul de adevăr. Avem la dispoziție integratul 7442, un decodificator BCD-zecimal, care implementează mintermii inversați la ieșire. Aplicând ȘI-NU peste ieșirile corespunzătoare mintermilor se obține disjuncția lor (cf. De Morgan), în consecință FCD a funcției. Fiindcă integratul 7442 este un decodificator BCD-zecimal cu 4 intrări, iar funcțiile au doar 3 variabile, se vor folosi cele 3 intrări mai puțin semnificative, iar cea de-a 4-a va fi conectată la 0 (GND). Deoarece funcțiile sunt compuse din disjuncția a câte 4 mintermi, se vor utiliza porți ȘI-NU cu 4 intrări.

Tabelul de adevăr

	B_2	B_1	B_0	G_2	G_1	G_0
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

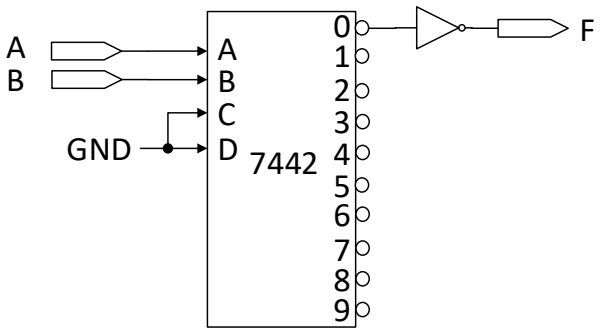


21. Să se proiecteze poarta SAU-NU cu 2 intrări folosind decodificatorul 7442 și alte porți logice fundamentale necesare.

Rezolvare: Pentru implementarea cu decodificator se folosește forma canonică disjunctivă (FCD) sau tabelul de adevăr. Avem la dispoziție integratul 7442, un decodificator BCD-zecimal, care implementează mintermii inversați la ieșire. Aplicând NOT peste ieșirea corespunzătoare se obține mintermul funcției. Fiindcă integratul 7442 este un decodificator BCD-zecimal cu 4 intrări, iar funcția are 2 variabile, se vor folosi cele 2 intrări mai puțin semnificative, iar celelalte vor fi conectate la 0 (GND).

Tabelul de adevăr

	B	A	F
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	0

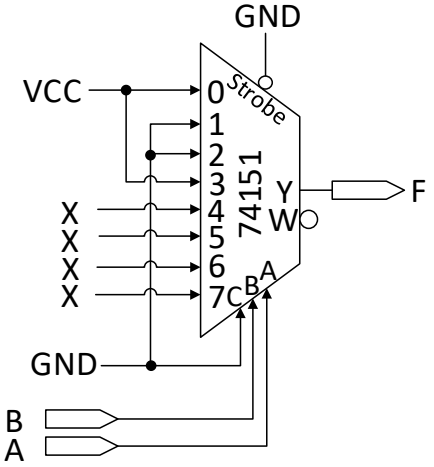


22. Să se proiecteze poarta XNOR cu 2 intrări folosind un multiplexor 74151.

Rezolvare: Implementarea cu multiplexor se realizează exprimând funcția în forma canonică disjunctivă (FCD) sau cu tabelul de adevăr care evidențiază mintermii componenți. Ambele intrări A, B vor fi conectate la selecțiile mai puțin semnificative ale multiplexorului 74151, iar celelalte se conectează la 0 (GND). Se poate copia coloana valorilor de ieșire, din tabelul de adevăr, la intrările de date ale multiplexorului.

Tabelul de adevăr

	B	A	F
0	0	0	1
1	0	1	0
2	1	0	0
3	1	1	1



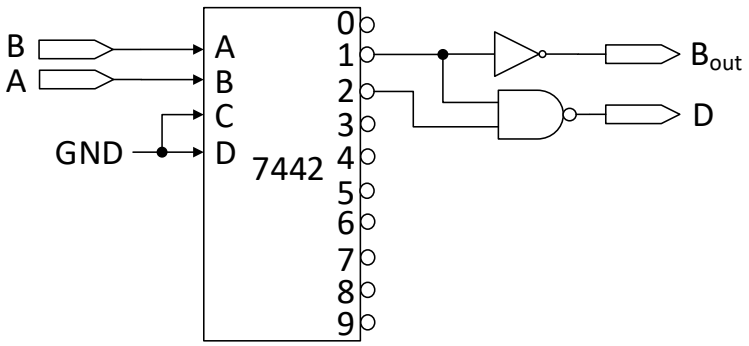
23. Să se proiecteze un semi-scăzător pe 1 bit folosind decodificatorul 7442 și alte porți logice fundamentale necesare.

Rezolvare: Pentru implementarea cu decodificator se folosește forma canonică disjunctivă (FCD) sau tabelul de adevăr. Avem la dispoziție integratul 7442, un decodificator BCD-zecimal, care implementează mintermii inversați la ieșire. Aplicând ȘI-NU peste ieșirile corespunzătoare mintermilor funcției D se obține disjuncția acestora (cf.

De Morgan), în consecință funcția D. Peste ieșirea corespunzătoare mintermului care implementează funcția B_{out} se aplică o inversare. Deoarece integratul 7442 este un decodificator BCD-zecimal cu 4 intrări, iar funcțiile au doar 2 variabile, se vor folosi cele 2 intrări mai puțin semnificative, iar celelalte vor fi conectate la 0 (GND).

Tabelul de adevăr

	A	B	B_{out}	D
0	0	0	0	0
1	0	1	1	1
2	1	0	0	1
3	1	1	0	0

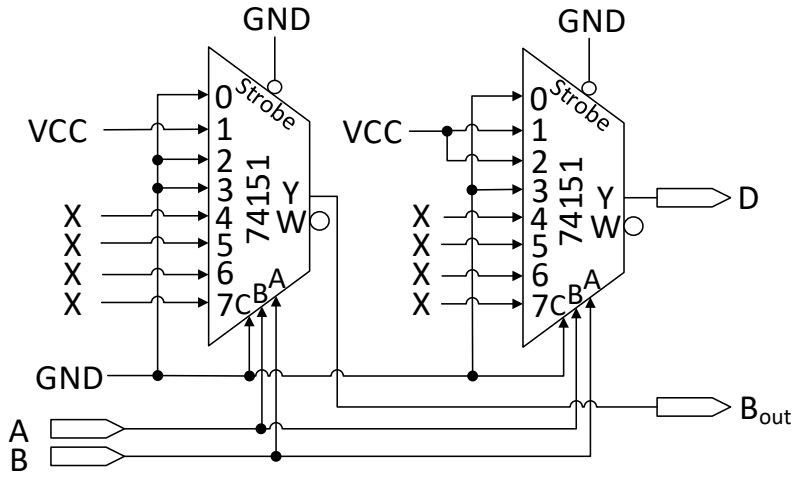


24. Să se proiecteze un semi-scăzător pe 1 bit folosind 2 multiplexoare 74151.

Rezolvare: Implementarea cu multiplexor se realizează exprimând funcțiile în forma canonică disjunctivă (FCD) sau cu tabelul de adevăr, care evidențiază mintermii componenți. Ambele intrări A, B vor fi conectate la selecțiile mai puțin semnificative ale multiplexorului 74151. Selecția mai semnificativă se conectează la 0 (GND). Se poate copia coloana valorilor de ieșire, din tabelul de adevăr, la intrările de date ale multiplexorului. Pentru cele 2 funcții B_{out} și D implementarea se va realiza individual, cu câte un multiplexor 74151.

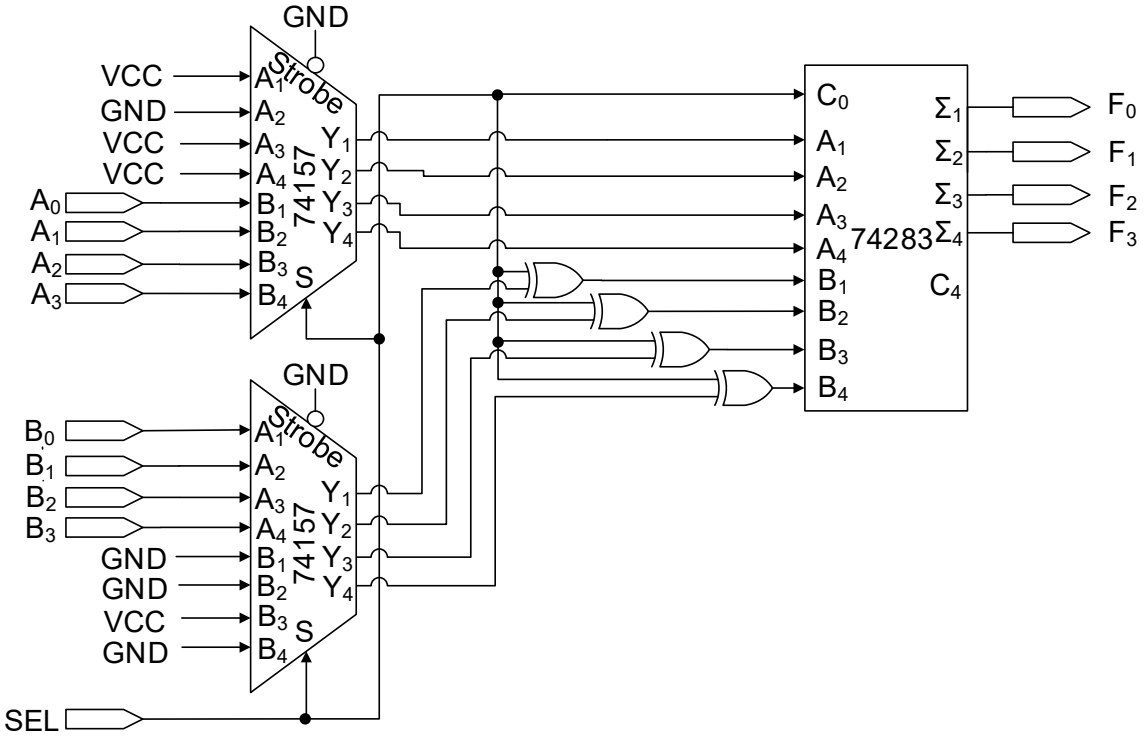
Tabelul de adevăr

	A	B	B_{out}	D
0	0	0	0	0
1	0	1	1	1
2	1	0	0	1
3	1	1	0	0



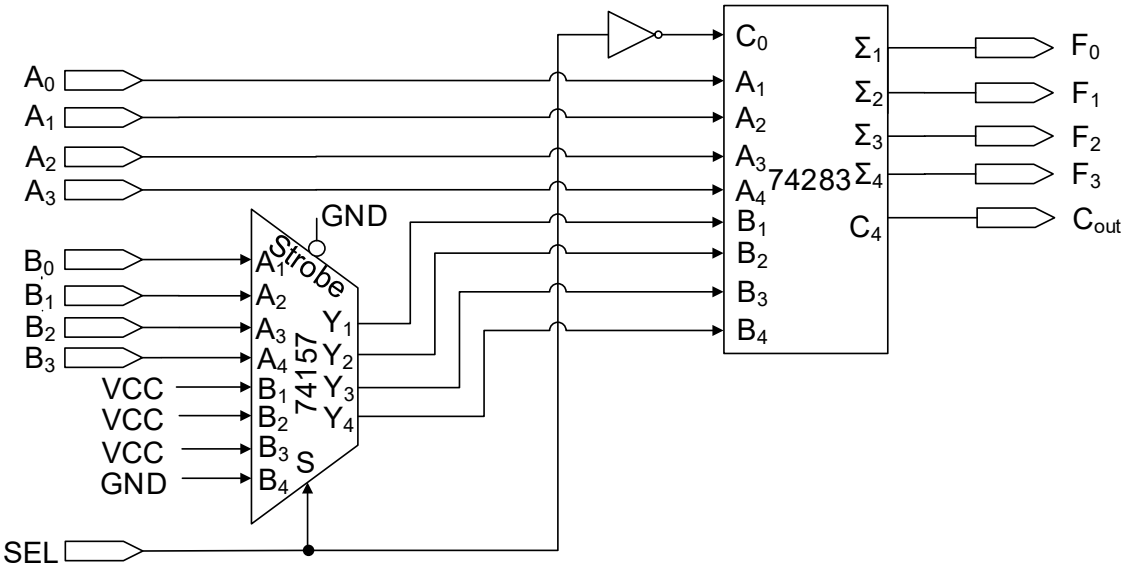
25. Să se proiecteze o UAL (Unitate Aritmetică-Logică) pe 2 operanzi A, B de câte 4 biți, care execută următoarele operații în funcție de valoarea de selecție SEL: 0 -> 13 + B; 1 -> A - 4 în complement față de 2. Se va folosi sumatorul 74283 și multiplexoare 74157.

Rezolvare: Se observă faptul că cele 2 operații se pot efectua cu un sumator-scăzător (realizat cu circuitul 74283 și porți XOR) ai cărui termeni diferă în funcție de semnalul SEL. Se pot folosi 2 unități 74157 (multiplexoare MUX 2:1 cu calea de date pe 4 biți) cu ajutorul cărora termenii implicați în operații se pot inițializa corespunzător, în funcție de SEL.



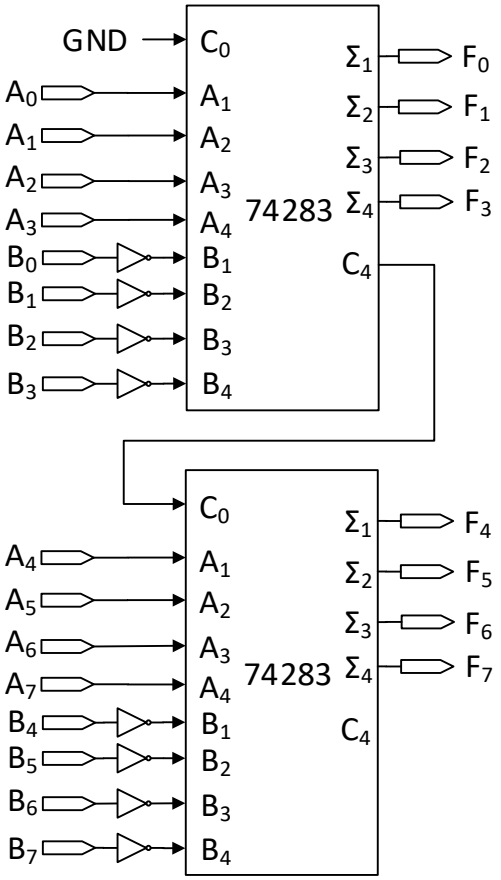
26. Să se proiecteze o UAL (Unitate Aritmetică-Logică) pe 2 operanzi A, B de câte 4 biți, care execută următoarele operații în funcție de valoarea de selecție SEL: 0 -> A + B + 1; 1 -> A + 7. Se va folosi sumatorul 74283 și multiplexorul 74157.

Rezolvare: Se observă faptul că cele 2 operații se pot efectua cu un sumator 74283 pentru care termenul al 2-lea diferă în funcție de semnalul SEL: poate să fie B, dacă SEL=0 sau 0111_2, dacă SEL=1. Pentru selecție se poate utiliza multiplexorul 74157 (un MUX 2:1 cu calea de date pe 4 biți). Adunarea cu +1 (de la A + B + 1) se poate realiza conectând semnalul $\overline{SEL}$ la intrarea C_0. Astfel, dacă SEL=0 atunci C_0=1 și se adună +1, iar dacă SEL=1 atunci C_0=0, deci nu va afecta rezultatul.



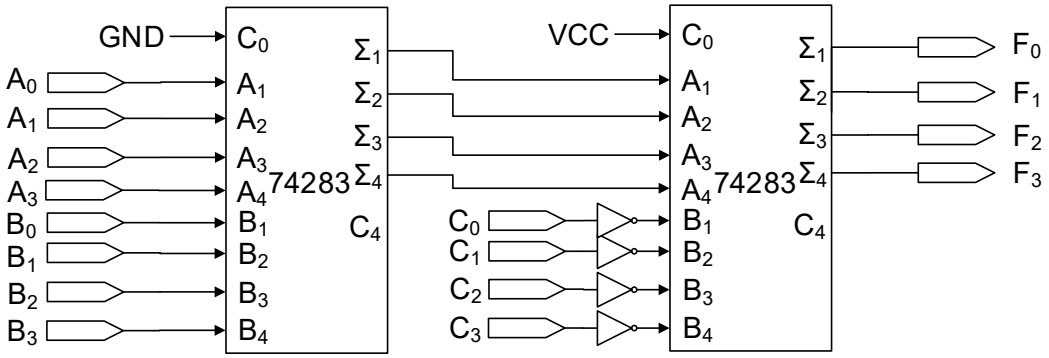
27. Să se proiecteze o unitate aritmetică pe 8 biți, care calculează A - B - 1 în Complement față de 2, folosind două sumatoare 74283 și porți logice.

Rezolvare: În complement față de 2, întreaga expresie se poate rescrie astfel: A - B - 1 = A + $\overline{B}$ + 1 - 1 = A + $\overline{B}$. Aceasta se poate implementa cu două sumatoare 74283 cascadate și inversoare. C_0 se va conecta la 0 (GND).



28. Să se proiecteze o unitate aritmetică pe 4 biți care calculează A + B - C în Complement față de 2, folosind două sumatoare 74283 și porți logice.

Rezolvare: În complement față de 2, întreaga expresie se poate rescrie astfel: A + B - C = A + B + $\overline{C}$ + 1. Aceasta se poate implementa cu două sumatoare 74283 și inversoare. Primul C_0 se va conecta la 0 (GND), iar al 2-lea C_0 se va conecta la 1 (VCC) pentru a realiza incrementarea din finalul expresiei.

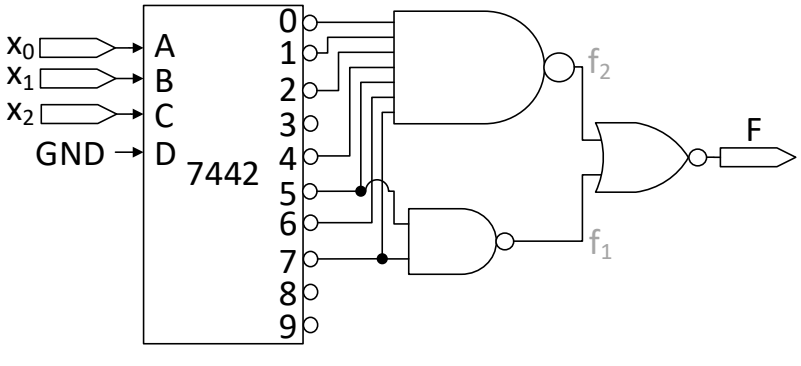


29. Să se realizeze un circuit care implementează funcția logică $F(x_2, x_1, x_0) = \overline{f_1(x_2, x_1, x_0) + f_2(x_2, x_1, x_0)}$, unde $f_1(x_2, x_1, x_0)=\sum(5, 7)$ și $f_2(x_2, x_1, x_0) = \overline{\overline{x_1} \cdot \overline{x_2}} + \overline{x_0 \cdot x_1}$. Se va folosi un singur decodificator 7442 și porți logice fundamentale pentru a implementa f_1 și f_2.

Rezolvare: Implementarea cu decodificator se realizează exprimând funcțiile f_1 și f_2 în forma canonică disjunctivă (FCD). Fiindcă f_1 este definită prin mintermi, aceasta este în forma canonică disjunctivă. În cazul funcției f_2 se va construi tabelul de adevăr pentru a identifica mintermii. Variabilele x_2, x_1, x_0 vor fi intrările decodificatorului. În consecință, ieșirile decodificatorului 7442 vor reprezenta mintermii negați asociați. Deoarece integratul 7442 este un decodificator BCD-zecimal cu 4 intrări, iar funcțiile au doar 3 variabile, se vor folosi cele 3 intrări mai puțin semnificative, iar a 4-a va fi conectată la 0 (GND). Pentru fiecare funcție se utilizează doar ieșirile corespunzătoare mintermilor implicați, ca intrări ale unei porți ȘI-NU. Rezultatele porților ȘI-NU generează cele 2 funcții f_1 și f_2 peste care se aplică operația SAU-NU a cărei ieșire va genera rezultatul final F.

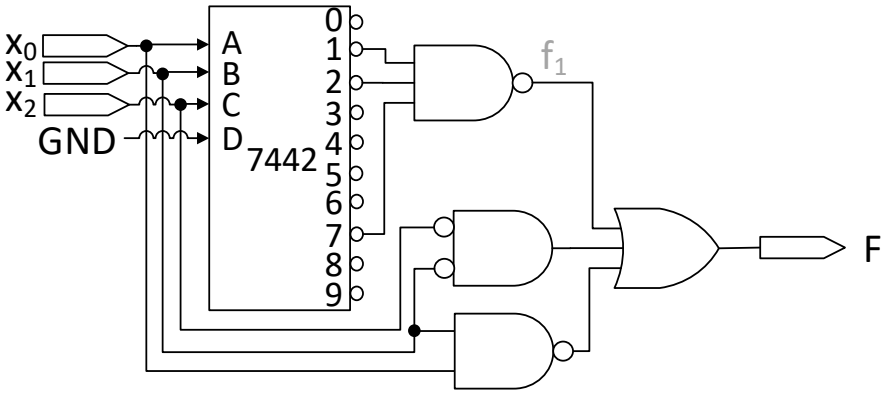
Tabelul de adevăr pentru f_2

	x_2	x_1	x_0	f_2
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	0	1	1
7	1	1	1	1



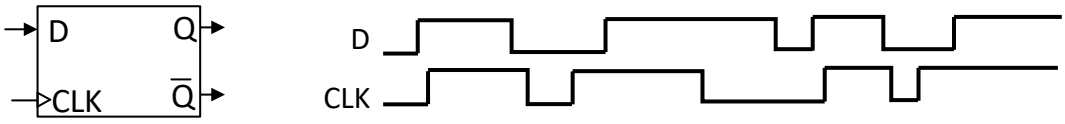
30. Să se realizeze un circuit care implementează funcția logică $F(x_2, x_1, x_0) = f_1(x_2, x_1, x_0) + \overline{x_1} \cdot \overline{x_2} + \overline{x_0 \cdot x_1}$, unde $f_1(x_2, x_1, x_0)=\sum(1, 2, 7)$. Se va folosi un singur decodificator 7442 și porți logice fundamentale pentru a implementa f_1.

Rezolvare: Implementarea cu decodificator se realizează exprimând funcția f_1 în forma canonică disjunctivă (FCD). Fiindcă f_1 este definită prin mintermi, aceasta este în forma canonică disjunctivă. Variabilele x_2, x_1, x_0 vor fi intrările decodificatorului. În consecință, ieșirile decodificatorului 7442 vor reprezenta mintermii negați asociați. Deoarece integratul 7442 este un decodificator BCD-zecimal cu 4 intrări, iar funcțiile au doar 3 variabile, se vor folosi cele 3 intrări mai puțin semnificative, iar a 4-a va fi conectată la 0 (GND). Pentru f_1 se utilizează doar ieșirile corespunzătoare mintermilor implicați, ca intrări ale unei porți ȘI-NU. Ulterior implementării lui f_1, restul expresiei se poate implementa cu porți logice.


x0
x1
x2
GND
A
B
C
D
7442
0
1
2
3
4
5
6
7
8
9
f1
F

B. Anexa 2 – Probleme cu circuite logice secvențiale

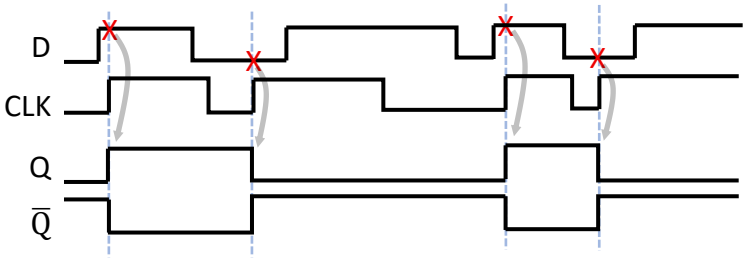
1. Determinați ieșirile bistabilului D flip-flop, care primește pe intrări impulsurile descrise în diagrama următoare.



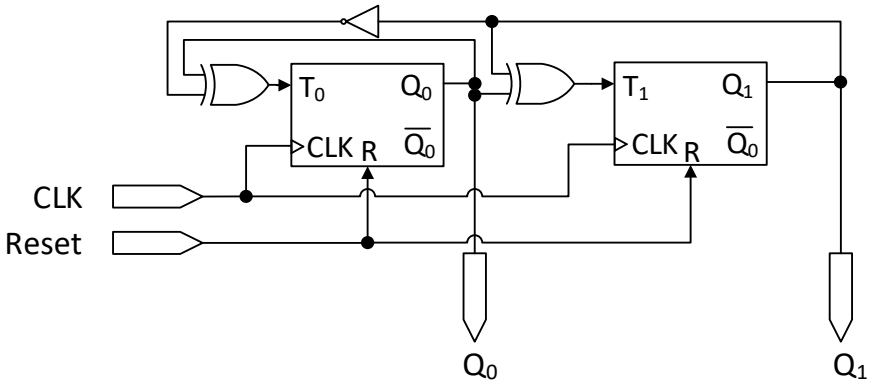
Rezolvare: Bistabilul D cu comutare pe frontul ascendent copiază la ieșirea Q valoarea semnalului de comandă D existentă pe frontul crescător al semnalului. Ieșirea $\overline{Q}$ reprezintă valoarea complementară a lui Q.

Tabelul de adevăr

D	Q^{t+1}
0	0
1	1



2. Determinați graful de tranziții al circuitului de mai jos.



Rezolvare:

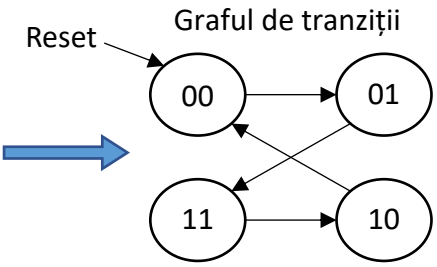
Se generează expresiile pentru intrările bistabilelor T_1, T_0. Se generează un tabel de adevăr care conține pe linii toate stările curente $Q_1^t Q_0^t$ și efectul lor asupra intrărilor de comandă T_1, T_0. Pe baza valorilor (Q_1^t, T_1) și (Q_0^t, T_0) se determină care va fi starea următoare $Q_1^{t+1} Q_0^{t+1}$ a circuitului și se introduce în același tabel adăugând coloane suplimentare (la dreapta). Se generează un graf care are în noduri toate stările curente. Perechile de forma ($Q_1^t Q_0^t$, $Q_1^{t+1} Q_0^{t+1}$) din tabel reprezintă arce orientate de la starea curentă la cea următoare în cadrul grafului de tranziții.

$T_0=\overline{Q_1^t}\oplus Q_0^t$ $T_1=Q_1^t\oplus Q_0^t$

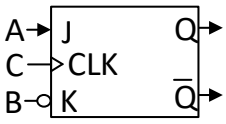
Tabelul de adevăr

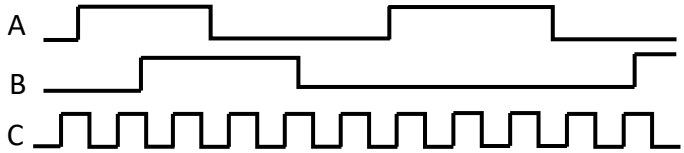
T	Q^{t+1}
0	Q^t
1	$\overline{Q^t}$

Q_1^t	Q_0^t	T_1	T_0	Q_1^{t+1}	Q_0^{t+1}
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	0	0	0
1	1	0	1	1	0



3. Determinați ieșirile bistabilului JK flip-flop, care primește pe intrări impulsurile descrise mai jos.

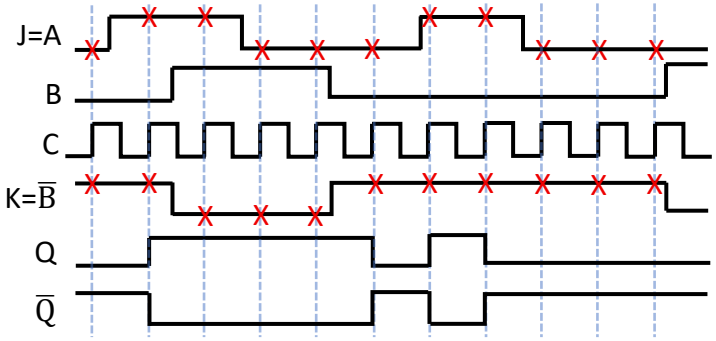




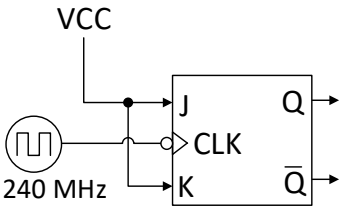
Rezolvare: Bistabilul JK cu comutare pe frontul ascendent funcționează după următorul tabel de adevăr. Ieșirea $\overline{Q}$ reprezintă valoarea inversată a lui Q. Se citesc intrările la fiecare front ascendent.

Tabelul de adevăr

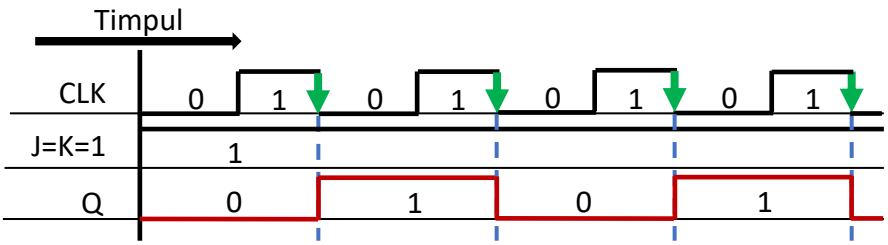
J K	Q^{t+1}
0 0	Q^t
0 1	0
1 0	1
1 1	$\overline{Q^t}$



4. Dacă frecvența semnalului de tact pe intrarea bistabilului de mai jos este de 240Hz, care este frecvența pe ieșirile bistabilului?



Rezolvare: Bistabilul JK în configurație Toggle (cu intrările conectate la 1 logic) își schimbă starea la fiecare front descrescător, conform diagramei de timp de mai jos. Se observă faptul că la ieșire se obține un semnal de tact cu perioadă dublă față de perioada semnalului CLK. Așadar, frecvența este de 2 ori mai mică (funcționează ca un divizor de frecvență cu factorul 2). Ca urmare, frecvența pe ieșiri este 240/2 = 120Hz.



5. Implementați un bistabil JK folosind un bistabil D, un MUX 2:1 și un inversor.

Rezolvare: Se utilizează tabelele de excitație ale bistabilelor de tip JK și D. Se asociază combinațiile pentru care stările curentă și următoare (Q^t, Q^{t+1}) sunt identice și se creează un tabel de adevăr în care starea curentă Q^t și intrările bistabilului JK apar în partea stângă. În partea dreaptă apar doar intrările bistabilului D. Din tabel se determină expresiile intrărilor bistabilului D folosind o metodă de minimizare.

Tabel excitație JK

Q^t	Q^{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Tabel excitație D

Q^t	Q^{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

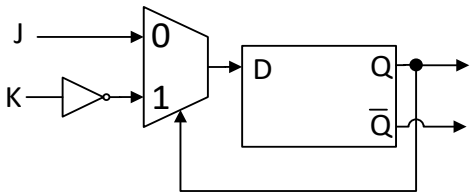
Q^t	J	K	D
0	0	X	0
0	1	X	1
1	X	1	0
1	X	0	1

D:

Q^t \ J K	00	01	11	10
0	0	0	1	1
1	1	0	0	1

$$D = (\overline{Q^t} \cdot J) + (Q^t \cdot \overline{K})$$

Se observă faptul că expresia lui D se poate implementa cu un MUX 2:1 cu selecția legată la Q^t. Rezultă următorul circuit:



6. Un bistabil PN are 4 operații în funcție de intrările sale de comandă P și N: 00 → reset, 01 → memorare, 10 → complementare; 11 → set. Se cer următoarele: a) Tabelul de adevăr al bistabilului; b) Ecuația caracteristică (ecuația lui Q^{t+1}); c) Tabelul de excitație; d) Arătați modul în care un bistabil D poate fi convertit într-un bistabil PN.

Rezolvare:

a) Tabelul de adevăr

Q^t	P	N	Q^{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

b) Ecuația caracteristică

Q^{t+1}:

Q^t \ P N	00	01	11	10
0	0	0	1	1
1	0	1	1	0

$$D = (\overline{Q^t} \cdot P) + (Q^t \cdot N)$$

c) Tabelul de excitație

Q^t	Q^{t+1}	P N
0	0	0 X
0	1	1 X
1	0	X 0
1	1	X 1

d) Se utilizează tabelele de excitație ale bistabilelor de tip PN și D. Se asociază combinațiile pentru care stările curentă și următoare (Q^t, Q^{t+1}) sunt identice și se creează un tabel de adevăr în care starea curentă Q^t și intrările bistabilului PN apar în partea stângă. În partea dreaptă apar doar intrările bistabilului D. Din tabel se determină expresiile intrărilor bistabilului D folosind o metodă de minimizare.

Tabel excitație JK

Q^t	Q^{t+1}	P N
0	0	0 X
0	1	1 X
1	0	X 0
1	1	X 1

Tabel excitație D

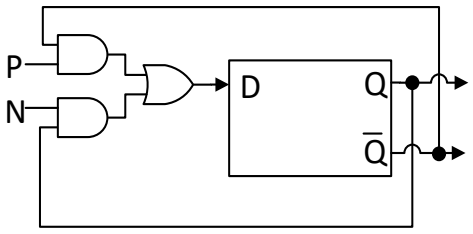
Q^t	Q^{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Q^t	J	K	D
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

D:

Q^t \ P N	00	01	11	10
0	0	0	1	1
1	0	1	1	

$$D = (\overline{Q^t} \cdot P) + (Q^t \cdot N)$$

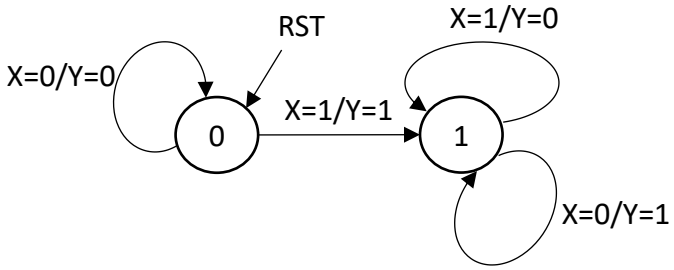


7. Proiectați un circuit serial de complementare în complement față de 2 cu o intrare X, o intrare de reset, una de tact și o ieșire Y. Circuitul acceptă un șir de biți la intrarea X și generează la ieșirea Y complementul față de 2 al valorii primite. Biții se primesc începând

cu cel mai puțin semnificativ avansând către cel mai semnificativ. Circuitul poate fi resetat asincron pentru a porni și opri operația.

Rezolvare:

Se implementează algoritmul de complementare față de 2. Se începe cu bitul cel mai puțin semnificativ și se avansează către bitul cel mai semnificativ. Până la primul bit de 1 întâlnit (inclusiv bitul de 1) se lasă neschimbați. Toți biții care urmează ulterior se inversează. Circuitul are astfel 2 stări, una în care nu inversează biții de pe intrare (Y = X) și una în care inversează biții de pe intrare (Y = $\overline{X}$). Se folosește un bistabil care memorează starea curentă. Starea de neinversare se codifică cu 0 și starea altenativă se codifică cu 1. După *reset* se pornește în starea 0. Dacă se primește un bit X=1 starea se modifică în 1. Pe baza acestor reguli se poate scrie următorul tabel de funcționare a bistabilului și tabelul de adevăr al ieșirii Y în conformitate cu *graful de tranziții*:



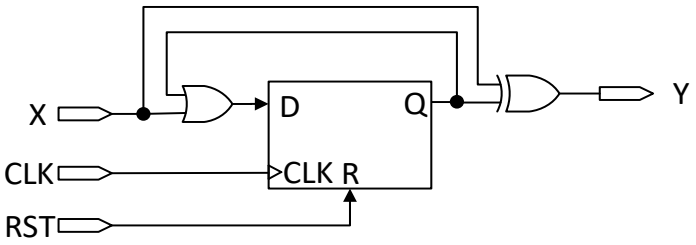
Q^t	X	Q^{t+1}
0	0	0
0	1	1
1	0	1
1	1	1

$$Q^{t+1} = Q^t + X$$

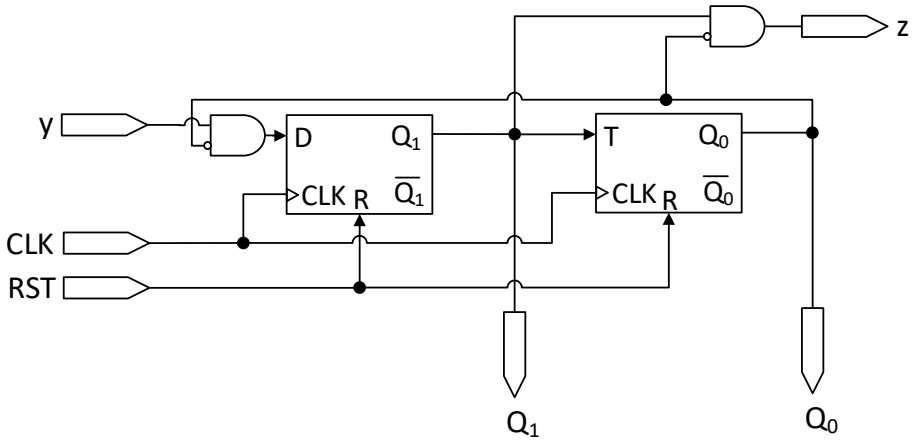
Q^t	X	Y
0	0	0
0	1	1
1	0	1
1	1	0

$$Y = Q^t \oplus X$$

Alegem implementarea cu bistabil D flip-flop pentru care știm relația $Q^{t+1} = D$ => $D = Q^t + X$. Circuitul rezultat este următorul:



8. Determinați graful de tranziții al circuitului de mai jos cu intrarea y și ieșirea z.



Rezolvare: Se generează expresiile pentru intrările bistabilelor D și T. Se generează un tabel de adevăr care conține pe linii toate variantele de intrare cu starea curentă (y Q_1^t Q_0^t) și efectul asupra intrărilor de comandă D și T. Pe baza valorilor (Q_1^t, D) și (Q_0^t, T) se determină care va fi starea următoare ($Q_1^{t+1}Q_0^{t+1}$) și ieșirea z în funcție de ($Q_1^tQ_0^t$) și se introduc în același tabel adăugând coloane la dreapta. Se generează un graf care are în noduri toate combinațiile posibile de 2 biți. Perechile obținute de forma ($Q_1^tQ_0^t$, $Q_1^{t+1}Q_0^{t+1}$) reprezintă arce orientate de la starea curentă la cea următoare, în cadrul grafului de tranziții. Condiția de tranziție este dată de y. În graf, ieșirea z poate apărea pe arce sau atașată la noduri (stări), în funcție de caz.

$D=\overline{Q_0^t}\cdot y$ $\quad T=Q_1^t$

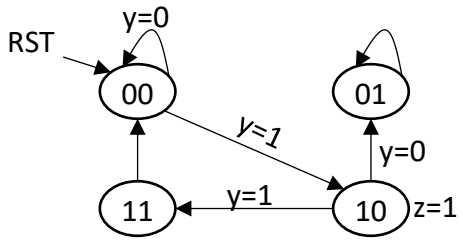
Tabele de adevăr

D	Q^{t+1}
0	0
1	1

T	Q^{t+1}
0	Q^t
1	$\overline{Q^t}$

y	Q_1^t	Q_0^t	D	T	Q_1^{t+1}	Q_0^t	z
0	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0
0	0	1	0	0	0	1	0
1	0	1	0	0	0	1	0
0	1	0	0	1	0	1	1
1	1	0	1	1	1	1	1
0	1	1	0	1	0	0	0
1	1	1	0	1	0	0	0

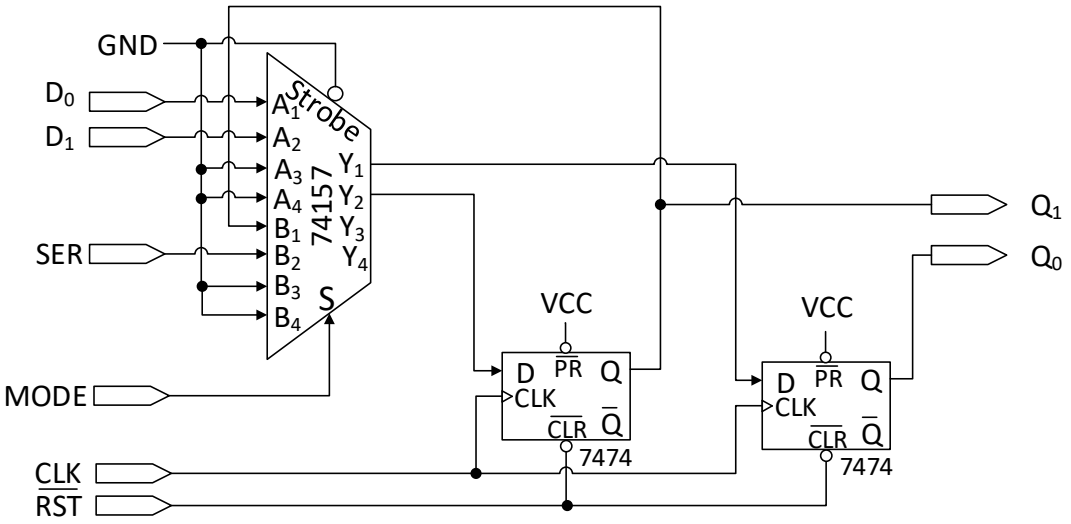
Graful de tranziții



9. Să se proiecteze cu bistabile D 7474 și multiplexorul 74157 un registru combinat pe 2 biți cu următoarele operații. Dacă intrarea MODE = 0 se realizează încărcare paralelă de pe intrările D_1D_0. Dacă MODE = 1 se realizează încărcare serială de pe intrarea SER cu deplasare la dreapta. Registrul operează pe frontul ascendent.

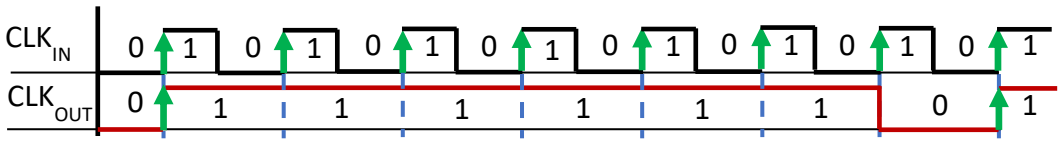
Rezolvare:

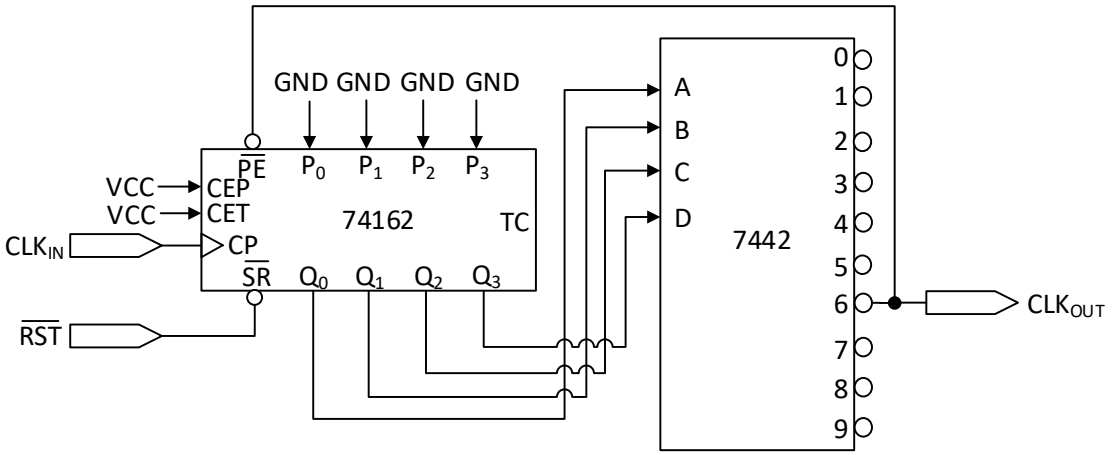
Integratul 7474 conține 2 bistabile D. Operațiile se vor implementa cu MUX 2:1 cu calea de date pe 2 biți. Pentru selecția modului de lucru se va folosi integratul 74157 care are calea de date pe 4 biți, din care se vor utiliza doar 2 biți.



10. Să se proiecteze un circuit divizor de frecvență cu factorul de divizare 7 folosind numărătorul 74162 și decodificatorul 7442. Circuitul va avea posibilitatea de reset.

Rezolvare: Circuitul trebuie să producă pe ieșire un semnal de tact CLK_{OUT} cu frecvența de 7 ori mai mică decât frecvența semnalului de tact de intrare CLK_{IN} ceea ce înseamnă o perioadă de 7 ori mai mare pe CLK_{OUT} decât pe CLK_{IN}. Deoarece factorul de umplere nu contează, CLK_{OUT} poate avea o formă de undă care are valoarea 0 o dată la 7 perioade CLK_{IN}, iar în rest 1. Acest lucru se poate genera folosind un numărător care numără direct în bucla 0-6. Semnalul CLK_{OUT} va fi 0 când se detectează 6, altfel 1. Decodificarea valorii 6 pe ieșirile numărătorului se poate face cu decodificatorul zecimal 7442. Pentru numărare, se va folosi numărătorul zecimal direct 74162.





11. Să se proiecteze un registru de deplasare la dreapta pe 3 biți cu multiplexoare 4:1 și bistabile de tip T. Registrul operează pe frontul ascendent și are o intrare serială SER.

Rezolvare:

Pentru a implementa un registru de deplasare se folosesc bistabile D. Atunci când se dorește implementarea cu bistabile de tip T, se poate face o implementare a bistabilului D cu bistabile T și logică suplimentară. Pentru aceasta se utilizează tabelele de excitație ale bistabilelor de tip D și T. Se asociază combinațiile pentru care stările curentă și următoare (Q^t, Q^{t+1}) sunt identice și se creează un tabel de adevăr în care starea curentă Q^t și intrările bistabilului D apar în partea stângă. În partea dreaptă apar doar intrările bistabilului T. Tabelul obținut pentru T se poate folosi la implementarea cu un MUX 4:1 având intrările de selecție conectate la Q^t și D și cele de date la valorile din coloana T.

Tabel excitație D

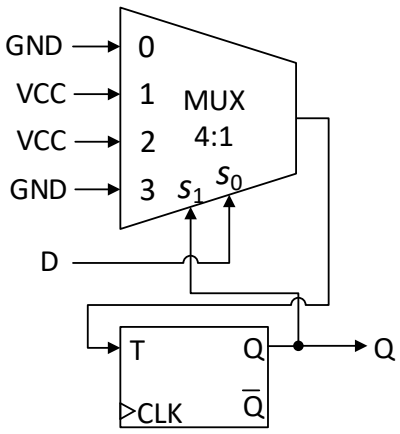
Q^t	Q^{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Tabel excitație T

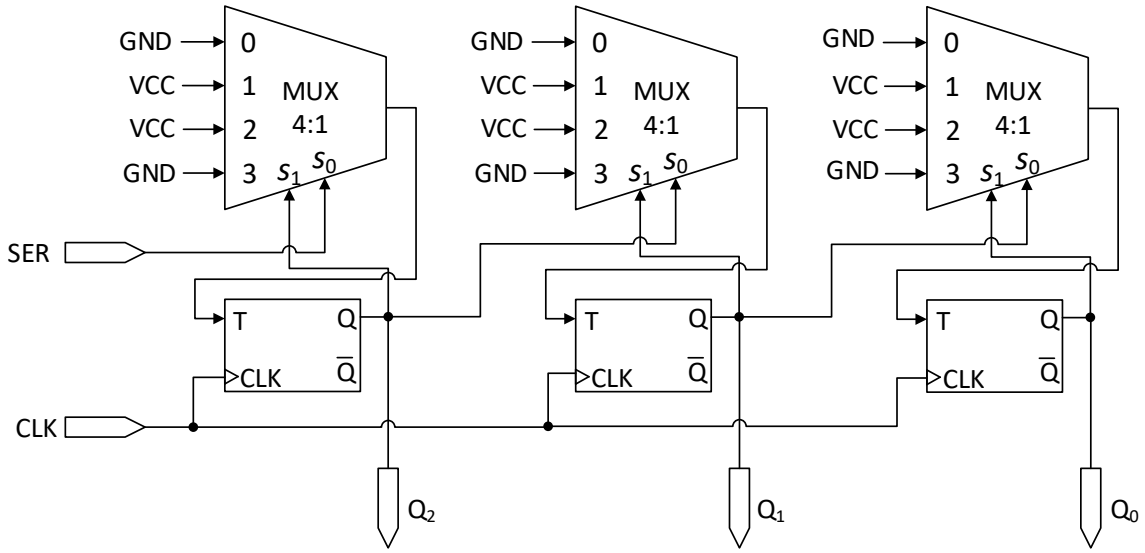
Q^t	Q^{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

Q^t	D	T
0	0	0
0	1	1
1	0	1
1	1	0

Bistabil D implementat cu bistabil T și MUX 4:1



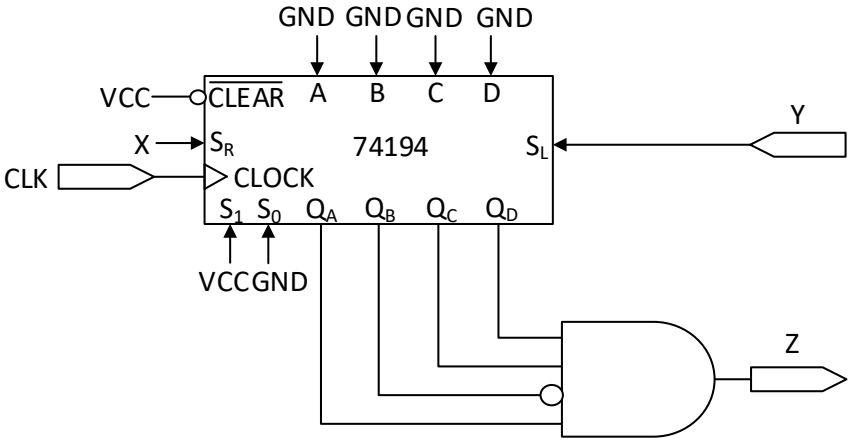
Rezultă următorul circuit pentru registrul de deplasare la dreapta pe 3 biți:



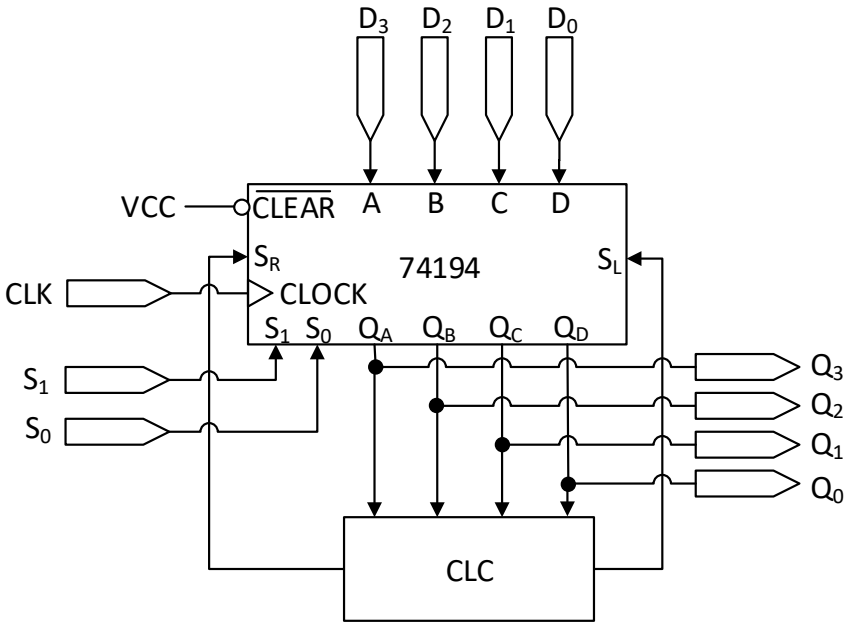
12. Să se proiecteze cu registrul universal 74194 circuitul care detectează secvența “1011”. Secvența se inserează serial (bit cu bit) pe intrarea Y începând cu bitul cel mai semnificativ (din stânga) al acesteia. Circuitul are o ieșire Z, care indică 1 numai când ultimii 4 biți inserați formează secvența de mai sus, iar în rest indică 0.

Rezolvare:

Se poate utiliza registrul universal 74194 în modul de lucru cu încărcare serială și deplasare la stânga. Se detectează cu o poartă ȘI cu 4 intrări dacă s-a înscris secvența “1011” în registru și se semnalează pe ieșirea Z.



13. Să se detecteze secvența de pe ieșirile $Q_{3:0}$ ale generatorului de secvențe de mai jos având în vedere că la primul impuls de tact se încarcă paralel valoarea binară $D_{3:0}$ = "1011", iar apoi se face numai deplasare la stânga. $S_L=\overline{Q_A} \cdot Q_D$ și S_R=0.



(CLC – Circuit Logic Combinațional)

Rezolvare:

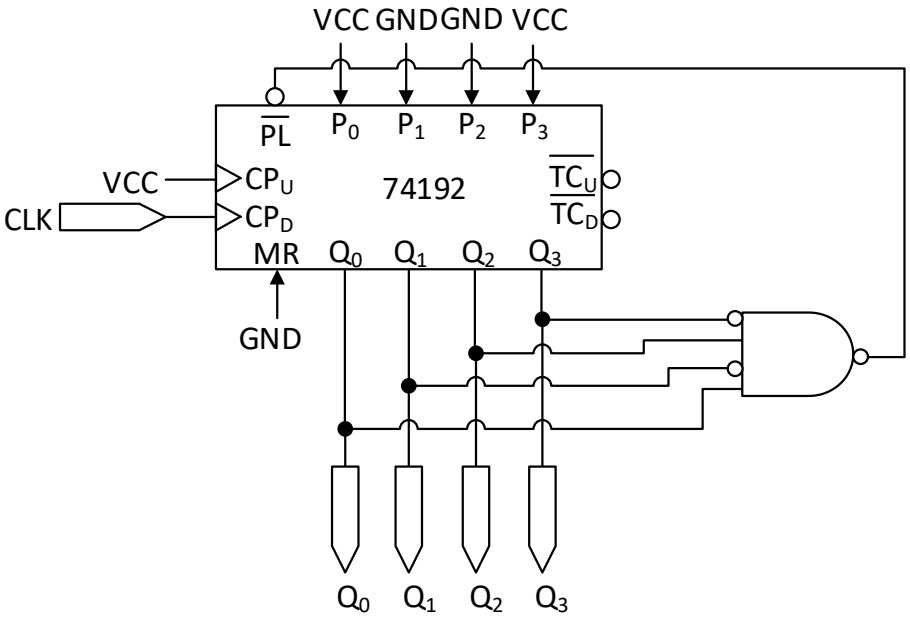
La fiecare impuls de tact, cei 3 biți mai puțin semnificativi (din dreapta) se deplasează la stânga cu o poziție. Noua valoare a bitului cel mai puțin semnificativ va fi dată de funcția pentru intrarea S_L calculată pe valorile biților de stare curentă.

Secvența generată va fi: 1011 (inițializare), 0110, 1100, 1000, 0000, 0000, … (se repetă valoarea 0000).

14. Să se proiecteze un numărător zecimal sincron cu integratul 74192, care să numere invers în bucla 9 – 6.

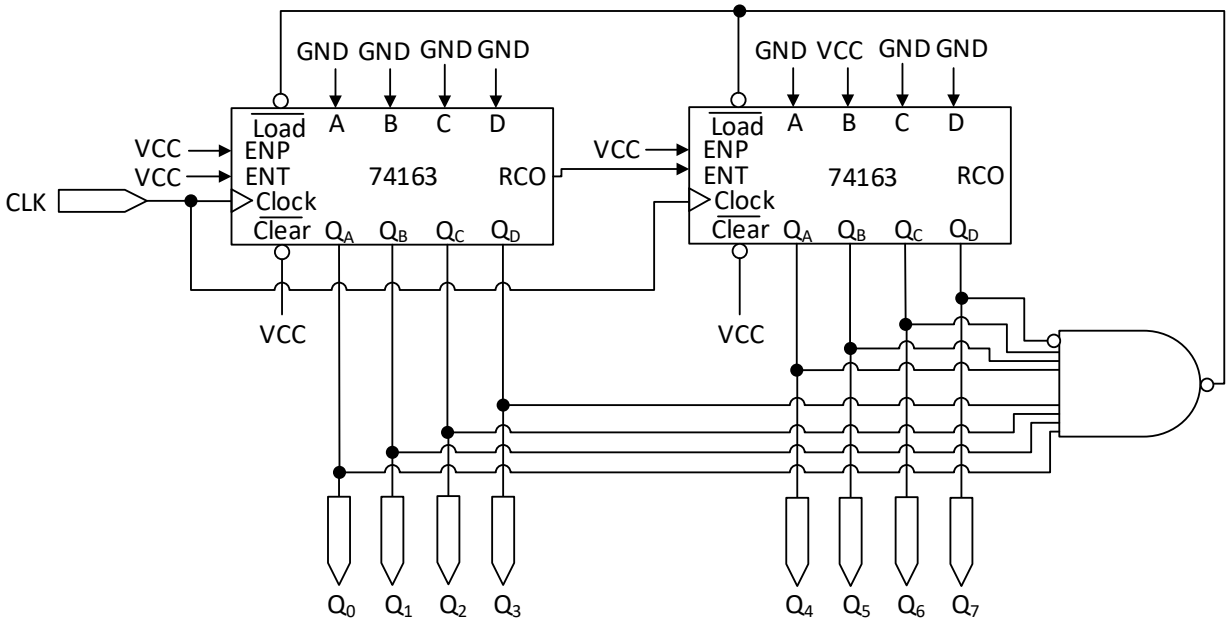
Rezolvare:

Integratul 74192 este un numărător sincron zecimal reversibil pe 4 biți care are comandă de încărcare paralelă asincronă. Pe liniile de date paralele se va încărca valoarea 9 (exprimată în binar), deoarece este prima valoare din bucla de numărare inversă. Fiindcă încărcarea paralelă este asincronă, este nevoie ca această comandă să fie întârziată cu 1 impuls după încheierea buclei de numărare, pentru a nu pierde valoarea 6 care încheie bucla. Din acest motiv comanda de încărcare paralelă se va iniția atunci când se detectează valoarea 5. Semnalul de tact pentru numărare directă se menține la valoarea 1 (VCC), iar cel de reset se dezactivează cu 0 (GND).



15. Să se proiecteze un numărător binar sincron prin cascadarea a 2 integrate 74163 care să numere direct în bucla 32 – 127.

Rezolvare: Integratul 74163 este un numărător sincron binar direct pe 4 biți, care are comandă de încărcare paralelă sincronă. Pentru a obține un numărător pe 8 biți se cascadează 2 integrate 74163. Pe liniile de date paralele se va încărca valoarea 32 (exprimată în binar pe 8 biți = 0010 0000), deoarece este prima valoare din bucla de numărare directă. Comanda de încărcare paralelă se va iniția atunci când se detectează valoarea 127 (exprimată în binar pe 8 biți = 0111 1111). Circuitul de detecție pentru valoarea 127 se implementează cu o poartă ȘI-NU, deoarece comanda de încărcare paralelă se inițiază cu 0 logic, deci se inversează rezultatul operației ȘI pe biți. Semnalul de reset se dezactivează cu valoarea 1 (VCC).



16. Să se proiecteze un numărător binar sincron direct pe 2 biți cu bistabile 7474 de tip D și posibilitatea de resetare asincronă. Numărătorul operează pe frontul ascendent.

Rezolvare:

Pentru a implementa un numărător binar sincron direct se folosesc bistabile T. Atunci când se dorește implementarea cu bistabile de tip D se poate face o implementare a bistabilului T cu bistabile D și logică suplimentară. Pentru aceasta, se utilizează tabelele de excitație ale bistabilelor de tip T și D. Se asociază combinațiile pentru care stările curentă și următoare (Q^t, Q^{t+1}) sunt identice și se creează un tabel de adevăr în care starea curentă Q^t și intrările bistabilului T apar în partea stângă. În partea dreaptă apar doar intrările bistabilului D. Folosind tabelul, se determină expresia comenzii bistabilului D, care corespunde unei porți XOR.

Tabel excitație T

Q^t	Q^{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

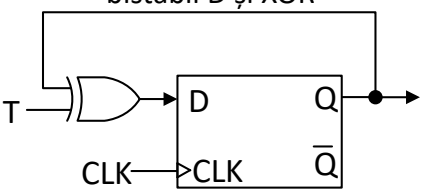
Tabel excitație D

Q^t	Q^{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

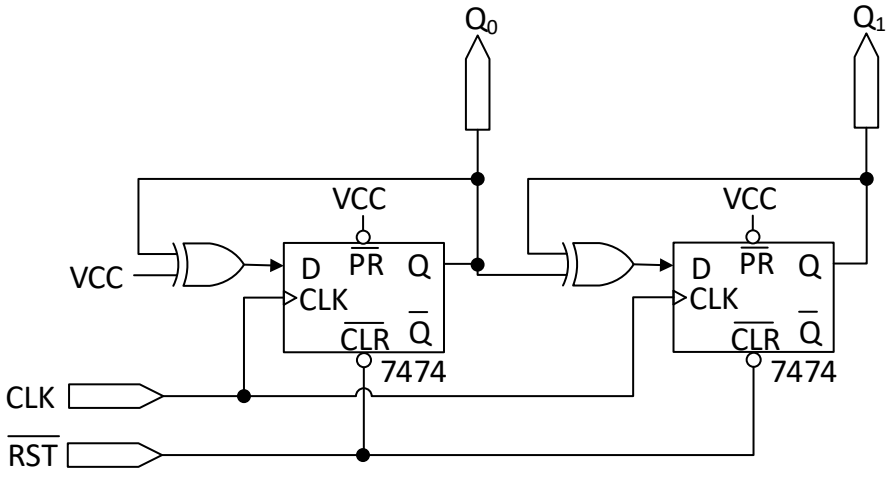
Q^t	T	D
0	0	0
0	1	1
1	1	0
1	0	1

$D = T \oplus Q^t$

Bistabil T implementat cu bistabil D și XOR



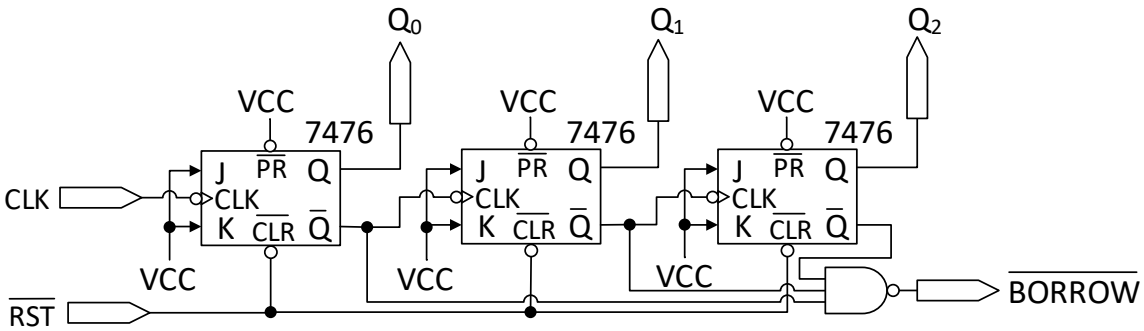
Rezultă următorul circuit pentru numărătorul binar sincron direct, pe 2 biți, cu integratul 7474:



17. Să se proiecteze un numărător binar asincron invers pe 3 biți cu bistabile 7476 de tip JK și posibilitatea de resetare asincronă. Numărătorul operează pe frontul descendent și trebuie să aibă un semnal $\overline{\mathrm{BORROW}}$ activ pe 0.

Rezolvare:

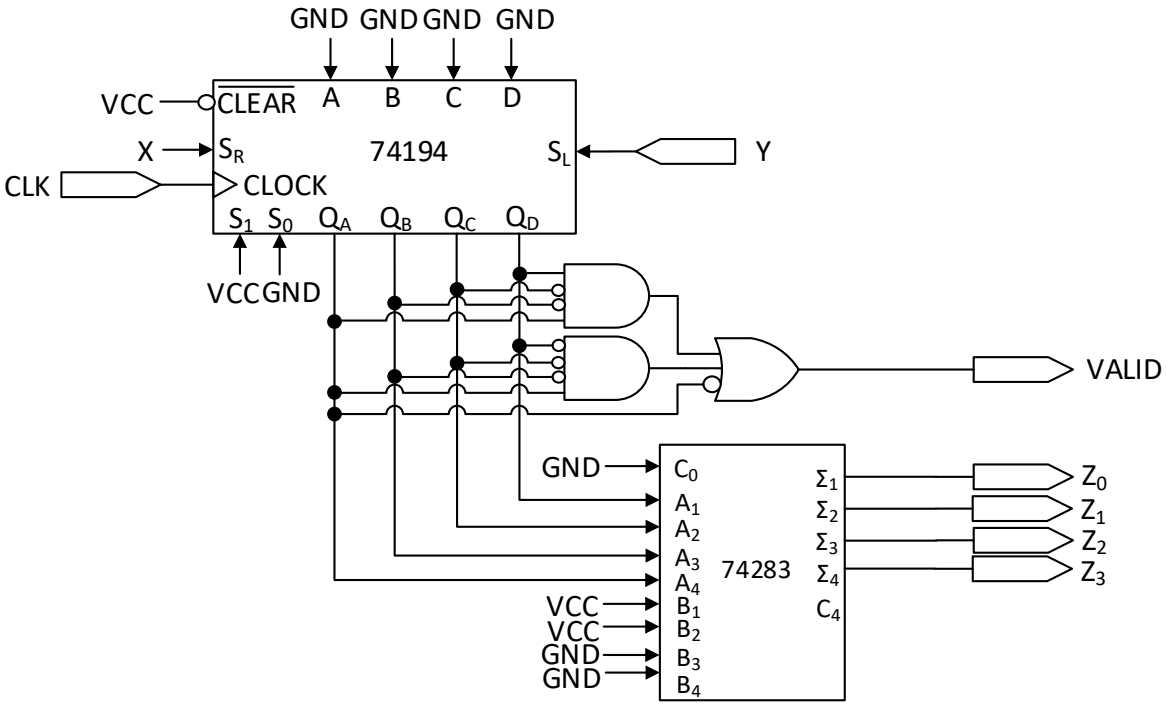
Pentru implementarea unui numărător binar asincron invers se folosesc bistabile JK în configurație oscilatorie (cu 1 logic pe intrări). Semnalul de tact este conectat la bistabilul cel mai puțin semnificativ, iar intrările de tact pentru bistabilele biților mai semnificativi sunt conectate la ieșirile negate ale bistabilelor învecinate mai puțin semnificative. Astfel, bistabilul cel mai puțin semnificativ va oscila la fiecare impuls de tact, iar celelalte vor oscila atunci când cel anterior comută din 0 în 1. Pentru implementarea semnalului $\overline{\mathrm{BORROW}}$, activ pe 0, este suficientă o poartă ȘI-NU peste ieșirile bistabilelor, pentru a se activa în valoarea 000. $\overline{\mathrm{BORROW}} = \overline{\overline{\mathrm{Q_2}} \cdot \overline{\mathrm{Q_1}} \cdot \overline{\mathrm{Q_0}}}$.



18. Să se proiecteze cu registrul universal 74194, sumatorul 74283 și porți logice fundamentale un circuit care primește date serial (bit cu bit) pe intrarea Y. Circuitul va indica pe ieșirea $Z_{3:0}$ codul Excess 3 al valorii binare formate din cei mai recenți 4 biți încărcați. Bitul cel mai semnificativ dintre cei 4 este primul încărcat. Pe ieșirea VALID, circuitul va indica 1 dacă valoarea din registru pentru care se calculează codul Excess 3 este mai mică decât 10_{10}, altfel indică 0, deoarece codul Excess 3 se calculează pentru valori de la 0 la 9.

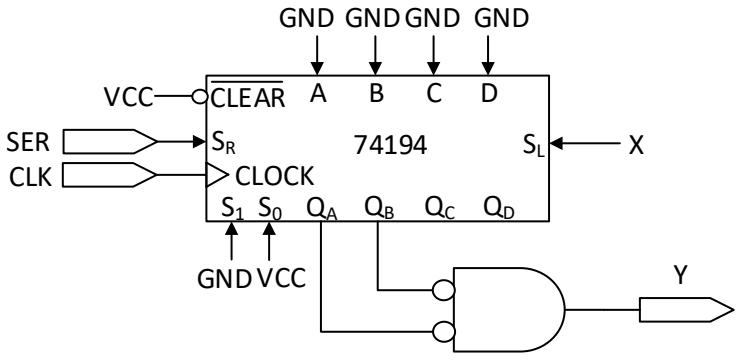
Rezolvare:

Se poate utiliza registrul universal 74194 în modul de lucru cu încărcare serială și deplasare la stânga. Codul Excess 3 se poate calcula folosind sumatorul 74283, care va însuma valoarea din registru cu 0011_2 (valoarea 3). O posibilitate pentru a detecta dacă valoarea din registru este mai mică decât 10_{10} este să se testeze dacă valoarea este 9 sau 8 sau dacă bitul cel mai semnificativ este 0, deoarece la valorile mai mici decât 8 bitul, cel mai semnificativ este 0. VALID = $\mathrm{Q_A} \cdot \overline{\mathrm{Q_B}} \cdot \overline{\mathrm{Q_C}} \cdot \mathrm{Q_D} + \mathrm{Q_A} \cdot \overline{\mathrm{Q_B}} \cdot \overline{\mathrm{Q_C}} \cdot \overline{\mathrm{Q_D}} + \overline{\mathrm{Q_A}}$.



19. Să se realizeze cu registrul universal 74194 și porți logice fundamentale un circuit care setează o ieșire Y=1 dacă cei mai recenți 2 biți primiți serial (bit cu bit) sunt 0, altfel Y=0. Registrul trebuie configurat pentru încărcare serială de pe intrarea SER, cu deplasare la dreapta.

Rezolvare: Se va utiliza registrul universal 74194 în modul de lucru cu încărcare serială și deplasare la dreapta. În acest caz, cei mai recenți 2 biți primiți se află pe ieșirile Q_A și Q_B. Se detectează cu o poartă ȘI cu 2 intrări, dacă aceștia sunt 00_2 și se semnalează pe ieșirea Y. $Y = \overline{Q_A} \cdot \overline{Q_B}$ – mintermul P_0.



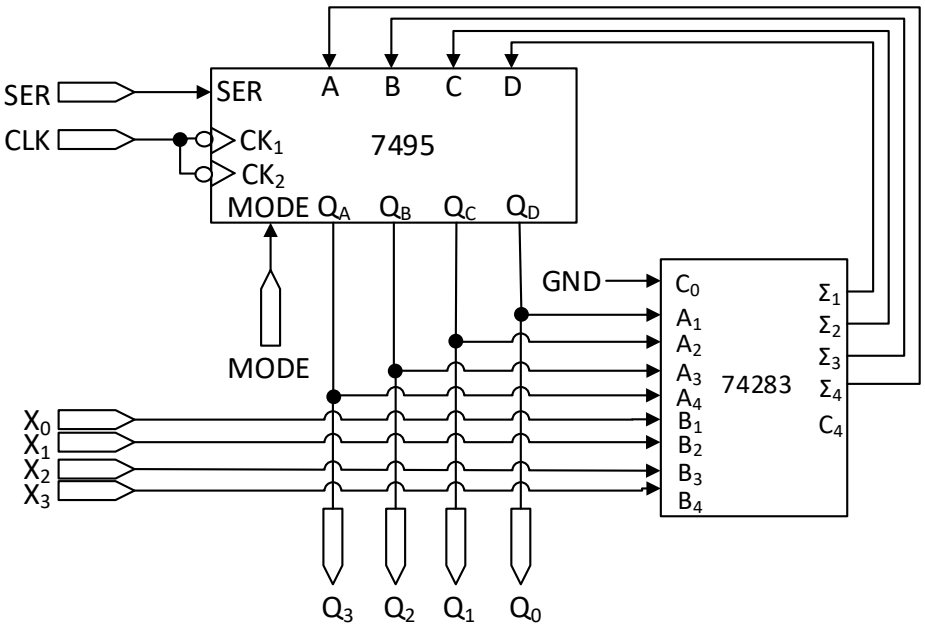
20. Să se realizeze cu registrul combinat 7495 un circuit care efectuează următoarele operații în funcție de intrarea de comandă MODE:

- MODE = 0 – încărcare serială de pe intrarea SER cu deplasare la dreapta;
- MODE = 1 – încărcare paralelă cu suma valorii din registru și valoarea $X_{3:0}$ calculată folosind sumatorul 74283 pe 4 biți (la care ieșirea de transport se ignoră).

Ieșirile registrului vor fi legate la terminalele de ieșire $Q_{3:0}$.

Rezolvare:

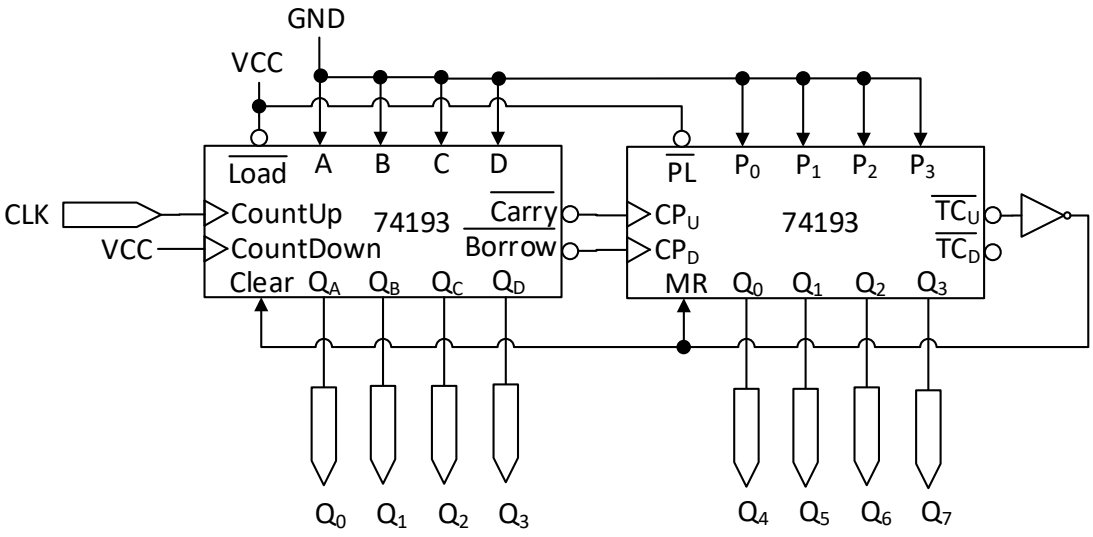
Se va utiliza registrul 7495 pentru care intrările de ceas vor fi legate la un semnal de tact comun. Sumatorul 74283 va primi pe cele 2 intrări atât ieșirea registrului cât și pe $X_{3:0}$. C_0 va fi legat la 0 (GND). Rezultatul va fi înaintat către intrările de date folosite la încărcarea paralelă.



21. Să se proiecteze un numărător binar sincron prin cascadarea unui numărător 74193 cu un numărător 74192, care să numere direct în bucla 0 – 158.

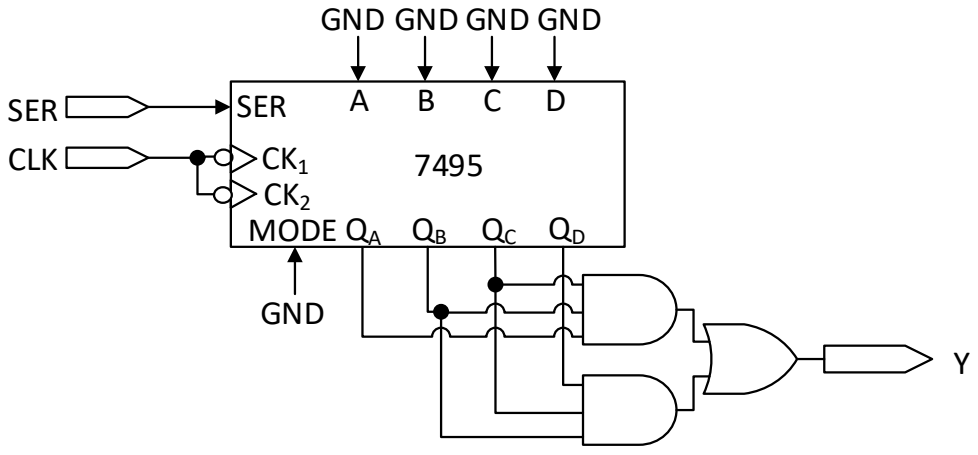
Rezolvare:

Integratele 74193 și 74192 sunt numărătoare sincrone bidirecționale pe 4 biți. Pentru a obține un numărător binar pe 8 biți prin cascadare, numărătorul binar 74193 va număra pe biții mai puțin semnificativi $Q_{3:0}$, iar cel zecimal, 74192 va număra pe biții mai semnificativi $Q_{7:4}$. Deoarece prima valoare din buclă este 0 aceasta se poate încărca prin resetarea numărătoarelor. Comanda de resetare fiind asincronă ea se va iniția atunci când se detectează valoarea 159 pe ieșiri, pentru a nu pierde valoarea 158. Fiind numărul maxim posibil, valoarea 159 $(1001\ 1111)_2$ se poate detecta folosind semnalul TC_U (Terminal Count Up) al numărătorului mai semnificativ, adică numărătorul 74192. Semnalul TC_U trebuie inversat deoarece este activ pe 0 și comanda de reset este activă pe 1. Încărcarea paralelă este dezactivată cu 1 (VCC).



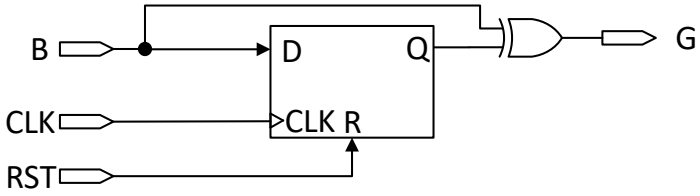
22. Să se proiecteze cu registrul combinat 7495 și porți logice fundamentale un circuit în care se inserează serial (bit cu bit) pe intrarea SER o secvență binară. Circuitul are o ieșire Y, care indică 1 numai când ultimii 4 biți inserați conțin 3 valori de 1 consecutive, iar în rest indică 0.

Rezolvare: Se va utiliza registrul 7495 pentru care intrările de ceas vor fi legate la un semnal de tact comun. Cele 3 valori consecutive de 1 pot să fie înscrise pe Q_A, Q_B, Q_C sau pe Q_B, Q_C, Q_D. În consecință, detecția se poate realiza cu expresia booleană $Y = Q_A \cdot Q_B \cdot Q_C + Q_B \cdot Q_C \cdot Q_D$. Modul de lucru va fi conectat la 0 (GND) pentru a activa încărcarea serială.



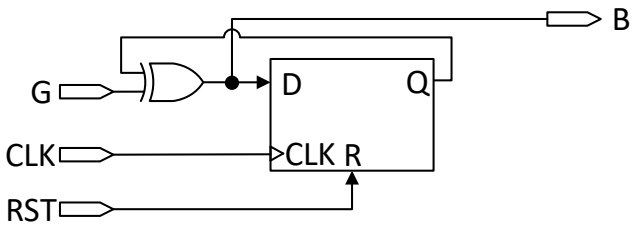
23. Proiectați un circuit serial de conversie BCD->Gray cu o intrare B, o intrare de reset, una de tact și o ieșire G. Circuitul primește serial un șir de biți în cod BCD la intrarea B și generează la ieșirea G codul Gray al valorii primite. Biții se primesc începând cu cel mai semnificativ, avansând către cel mai puțin semnificativ. Circuitul poate fi resetat asincron pentru a porni și opri operația.

Rezolvare: Se consideră relația de conversie BCD->Gray la bitul de rang *i*: $G_i = B_i \oplus B_{i+1}$. Bitul curent B_i se salvează într-un bistabil D și va fi disponibil pe ieșirea Q la impulsul următor, când apare B_{i-1}. Operația $B_{i-1} \oplus Q$ va genera bitul de rang inferior din codul Gray, după relația: $G_{i-1} = B_{i-1} \oplus Q = B_{i-1} \oplus B_i$. Inițial, bistabilul are valoarea 0, ceea ce înseamnă că bitul rezultat G_n, de la rangul maxim, este $G_n = B_n \oplus 0 = B_n$ și se salvează B_n. La ciclul următor, $G_{n-1} = B_{n-1} \oplus Q = B_{n-1} \oplus B_n$ și se salvează B_{n-1}. Urmează $G_{n-2} = B_{n-2} \oplus Q = B_{n-2} \oplus B_{n-1}$ și se salvează B_{n-2}, etc.



24. Proiectați un circuit serial de conversie Gray->BCD cu o intrare G, o intrare de reset, una de tact și o ieșire B. Circuitul primește serial un șir de biți în cod Gray la intrarea G și generează la ieșirea B codul BCD al valorii primite. Biții se primesc începând cu cel mai semnificativ, avansând către cel mai puțin semnificativ. Circuitul poate fi resetat asincron pentru a porni și opri operația.

Rezolvare: Se consideră relația de conversie Gray->BCD la bitul de rang *i*: $B_i = G_n \oplus G_{n-1} \oplus \ldots \oplus G_i$. Bitul calculat B_i se salvează într-un bistabil D și va fi disponibil pe ieșirea Q la impulsul următor, când apare G_{i-1}. Operația $Q \oplus G_{i-1}$ va genera bitul de rang inferior din codul BCD, după relația: $B_{i-1} = Q \oplus G_{i-1} = G_n \oplus G_{n-1} \oplus \ldots \oplus G_i \oplus G_{i-1}$. Inițial, bistabilul are valoarea 0, ceea ce înseamnă că bitul rezultat B_n, de la rangul maxim, este $B_n = 0 \oplus G_n = G_n$ și se salvează G_n. La ciclul următor, $B_{n-1} = Q \oplus G_{n-1} = G_n \oplus G_{n-1}$ și se salvează valoarea $G_n \oplus G_{n-1}$. Urmează $B_{n-2} = Q \oplus G_{n-2} = G_n \oplus G_{n-1} \oplus G_{n-2}$ și se salvează $G_n \oplus G_{n-1} \oplus G_{n-2}$, etc.



Bibliografie

1. Lucia Văcariu, Octavian Creț, “Analiza și Sinteza Dispozitivelor Numerice – Îndrumător de laborator”, Ediția a 3-a, U.T. Press, 2009.

2. Lucia Văcariu, Octavian Creț, “Probleme de Proiectare Logică a Sistemelor Numerice / Logic Design Problems for Digital Systems”, Ediția a 2-a, U.T. Press, 2013.

3. Randy H. Katz, Gaetano Borriello, “Contemporary Logic Design”, 2nd Edition, Pearson, 2004.

4. John Francis Wakerly, “Digital Design Principles and Practices”, 5th Edition, Pearson, 2018.

5. Charles H. Roth, Larry L. Kinney, “Fundamentals of Logic Design”, Enhanced 7th Edition, Cengage Learning, 2020.

6. Wayne Wolf, “FPGA-based System Design”, Prentice Hall, 2004.

7. Sarah L. Harris, David Harris, “Digital Design and Computer Architecture”, RISC-V Edition, Morgan–Kaufmann, 2021.

8. M. Morris Mano, Michael D. Ciletti, “Digital Design with an introduction to the Verilog, VHDL, HDL, and SystemVerilog”, 6th Edition, global edition, Pearson, 2018.

9. R. P. Jain, Kishor Sarawadekar, “Modern Digital Electronics”, 5th Edition, Mc Graw Hill India, 2022.

10. Thomas Lawrence Floyd, “Digital Fundamentals”, 11th Edition, Global Edition, Pearson, 2015.

11. Cristian-Cosmin Vancea, “TTL_Env Project for Project Navigator”, [Online]. Available: https://drive.google.com/uc?export=download&id=1800AdR6Vdo8PDd1tB9n5LCQZNHAcgh9K

12. Cristian-Cosmin Vancea, “Librăria TTL pentru Logisim”, [Online]. Available: https://drive.google.com/uc?export=download&id=1j4kRe9JXdQi6MqnqsB5nrXfX1uwoVc43