

**Ionel BOSTAN**

**Metode clasice și moderne  
în studiul circuitelor digitale**  
- lucrări practice de laborator -

**București - 2004**

## Lucrarea nr. 1: **Studiul parametrilor și caracteristicilor porților logice**

### 1. Scopul lucrării

În această lucrare sunt prezentate o serie de aspecte ce apar în funcționarea circuitelor logice realizate practic. Aceste aspecte nu pot fi puse în evidență prin analiza pur logică deoarece ele sunt induse de performanțele electrice limitate ce caracterizează circuite electronice aflate în spatele fiecărui simbol logic.

Cunoașterea caracteristicilor și a parametrilor electrici ai porților logice este strict necesară pentru implementarea cu succes în practică a circuitelor logice. Sunt prezentate: nivelurile de tensiune asociate stărilor logice; marginea de zgomot; modalitățile de realizare a etajelor de ieșire; timpul de propagare; factorul de încărcare; caracteristica de transfer în tensiune.

### 2. Considerente teoretice

Implementarea practică a schemelor logice se poate face apelând la circuite electronice, la circuite pneumatice, la circuite hidraulice etc.

În cazul implementărilor bazate pe circuite electronice, trebuie să avem în vedere că în spatele fiecărui simbol din schema logică se află un circuit electronic caracterizat de o serie de limitări (spre exemplu, viteza de propagare a semnalului electric prin circuit nu este infinită). Este evident că **aceste limitări vor influența performanțele circuitului logic**; mai puțin evident este faptul că **pot influența semnificativ chiar funcționarea schemei logice**, deci pot influența comportamentul în intrare-ieșire dorit. Din acest motiv, este necesar să cunoaștem foarte bine care sunt parametrii și limitările circuitelor electronice utilizate.

Din cele prezentate mai sus, rezultă că în proiectarea schemelor logice trebuie mers pe două planuri: unul în care se face analiza pur logică (în ipoteza că circuitele sunt ideale, fără limitări) iar altul în care trebuie să ținem cont de particularitățile și limitările introduse de tehnologia de realizare.

În mod uzual, circuitele electronice utilizate în electronica digitală lucrează în regim de comutație iar la intrare și la ieșire, vom regăsi doar două nivele de tensiune distincte.

**Parametrul unui circuit logic** = valoare de catalog pentru o mărime ce-i caracterizează funcționarea în condiții de test, sau la interconectarea cu alte circuite din aceeași familie.

Parametrii sunt aleși astfel încât să caracterizeze cât mai bine regimul de curent continuu, regimul tranzitoriu și comportamentul la zgomot al circuitului digital. Frecvent acești parametri sunt dați în cataloage ca valori tipice (normale) sau ca valori extreme (pentru cazul cel mai defavorabil).

**Familia de circuite logice** = grup de circuite logice cu caracteristici electrice similare, proiectate astfel încât să poată fi interconectate între ele în mod direct.

În proiectarea unor familii logice s-a pus accent fie pe creșterea vitezei de operare, fie pe reducerea consumului, sau s-a încercat obținerea unui compromis între viteza de operare și consum.

În prezent, cele mai utilizate familii logice sunt realizate pe suport de siliciu și folosesc tehnologii bipolare sau unipolare. O clasificare a acestora este prezentată în tabelul 1.

Tabelul 1

Tehnologie	Denumire familie	Seria
Bipolară - logică saturată	TTL (transistor transistor logic) standard	74***
	Schottky TTL	74S***
	Advanced Schottky TTL	74AS***
	Low-power Schottky TTL	74LS***
	Fast TTL	74F***
	Advanced Low-power Schottky TTL	74ALS***
Bipolară - logică nesaturată	ECL 10K (Emitter Coupled Logic)	
	ECL 100K (Emitter Coupled Logic)	
Unipolară de tip CMOS (Complementary MOS)	CMOS standard	CD4000
	High speed CMOS	74HC***
	Advanced CMOS	74AC***
	Advanced High speed CMOS	74AHC***
Unipolară de tip CMOS cu intrări compatibile TTL	High speed CMOS with TTL compatibility	74HCT***
	Advanced CMOS with TTL compatibility	74ACT***
	Advanced High speed CMOS with TTL compatibility	74AHCT***

## 2.1. Nivelurile de tensiune asociate stărilor logice

La prima vedere, modelarea celor două cifre binare în circuitele electronice s-ar putea face asociind prin convenție un nivel de tensiune pentru "unu logic" și un altul pentru "zero logic". Acest mod de lucru nu poate fi adoptat în practică deoarece nivelurile de tensiune sunt afectate de o serie de factori perturbatori precum: dispersia tehnologică, îmbătrânirea componentelor, variațiile tensiunii de alimentare etc. Din aceste motive, pentru fiecare stare logică se alocă câte o bandă de tensiuni permise. Pentru a putea face distincție între cele două stări logice, benzile de tensiune asociate sunt separate de o bandă interzisă (vezi fig. 1).

Semnificația mărimilor ce intervin în figura 1 este următoarea:

- $V_{OLmax}$  reprezintă valoarea maximă pentru tensiunea de ieșire corespunzătoare unei ieșiri logice aflată în starea "LOW";
- $V_{OHmin}$  reprezintă valoarea minimă pentru tensiunea de ieșire corespunzătoare unei ieșiri logice aflată în starea "HIGH";
- $V_{IHmin}$  reprezintă valoarea minimă necesară a tensiunii de intrare pentru a fi interpretată drept "unu logic" (stare "HIGH");
- $V_{OLmax}$  reprezintă valoarea maximă a tensiunii de intrare care este interpretată drept "zero logic" (starea "LOW").

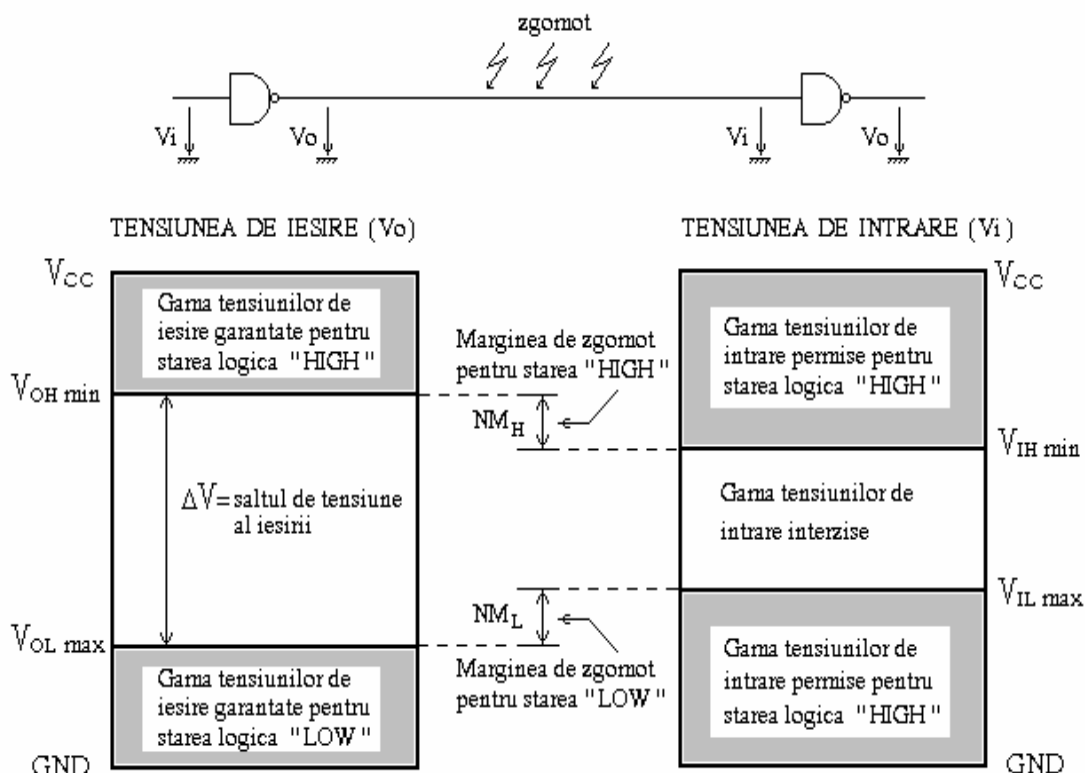


Fig.1. Nivelurile de tensiune asociate stărilor logice

Precizăm că valorile concrete ale tensiunilor  $V_{OLmax}$ ,  $V_{OHmin}$ ,  $V_{ILmax}$ ,  $V_{IHmin}$ , diferă de la o familie logică la alta, ele se regăsesc în foile de catalog ca parametri limită garantați de fabricant. În tabelul 2, sunt prezentate valorile de tensiune asociate stărilor logice pentru câteva familii logice.

Tabelul 2

Familie Parametru	TTL standard (74**)	74LS** 74AS** 74ALS**	CMOS (74HC**, 74AHC**)	CMOS compatibil TTL (74HCT**, 74AHCT**)	CMOS (CD***)
$V_{OLmax}$ [V]	0,4	0,4	0,1	0,4	99% din Vdd
$V_{OHmin}$ [V]	2,4	2,7	4,9	2,4	1% din Vdd
$V_{ILmax}$ [V]	0,8	0,8	1,5	2	30% din Vdd
$V_{IHmin}$ [V]	2	2	3,5	0,8	70% din Vdd

## 2.2. Marginea de zgomot

Pe traseul de legătură dintre ieșirea unui circuit și intrarea altuia se transmite un semnal util peste care se poate suprapune un semnal de zgomot. Se pune în mod firesc întrebarea: cât de mare poate fi acest zgomot pentru a nu perturba funcționarea sistemului?

Parametrul ce definește imunitatea la zgomot este denumit *margină de zgomot (noise margin)*, se notează cu **NM<sub>H</sub>** pentru starea logică "high", respectiv cu **NM<sub>L</sub>** pentru starea logică "low".

Marginea de zgomot reprezintă unul dintre cei mai importanți parametri ai circuitelor digitale deoarece oferă o măsură a imunității acestora la perturbații.

Marginea de zgomot statică este dată de amplitudinea maximă a semnalului de zgomot lent variabil care se poate suprapune peste semnalul util fără ca acesta să perturbe funcționarea normală a circuitului.

Analizând figura 1, se observă că între tensiunile garantate la ieșire și cele admisibile la intrare, apar diferențe. Rolul acestor diferențe este de a preîntâmpina efectul negativ pe care-l au zgomotele asupra semnalului util. Aceste diferențe nu sunt altceva decât valorile minime (garantate de fabricant) ale marginilor statice de zgomot. Ele se determină cu ajutorul relațiilor:

$$NM_H = V_{OH \min} - V_{IH \min}$$

$$NM_L = V_{OL \max} - V_{IL \max}$$

Facem precizarea că marginile de zgomot diferă de la o familie logică la alta. În plus, ele pot să nu fie egale pentru cele două stări logice.

## 2.3. Timpul de propagare

Acest parametru reprezintă întârzierea în timp dintre momentul aplicării unui semnal la intrarea unui circuit și momentul apariției răspunsului la ieșirea acestuia.

Timpul de propagare este un aspect nedorit în funcționarea circuitelor logice. Este necesar ca valoarea timpului de propagare să fie cât mai mică pentru a nu limita foarte mult viteza maximă de lucru a circuitelor. În funcție de tehnologia de realizare, întârzierea introdusă este de ordinul unităților sau chiar al zecilor de nanosecunde ( $1n = 10^{-9} s$ ).

Modul de definire a intervalelor de timp specifice semnalelor digitale se prezintă în figura 2, unde sunt prezentate semnalele de la intrarea și ieșirea unui inversor. Semnificația acestor mărimi temporale este următoarea:

- $t_r$  - (*rise time*), timpul de creștere al semnalului de intrare, se măsoară între 10% și 90% din amplitudinea tensiunii pentru nivelul de "unu logic";
- $t_f$  - (*fall time*), timpul de scădere al semnalului de intrare, se măsoară între 90% și 10% din amplitudinea tensiunii pentru nivelul de "unu logic";
- $t_{pHL}$ ,  $t_{pLH}$  - timpuri de propagare pentru tranziția ieșirii din "1" în "0", respectiv din "0" în "1";
- $t_{HL}$ ,  $t_{LH}$  - durată frontului căzător (respectiv crescător) al semnalului de ieșire;
- $t_p$  - timpul mediu de propagare definit prin relația:  $t_p = 0,5 \times (t_{pHL} + t_{pLH})$ .

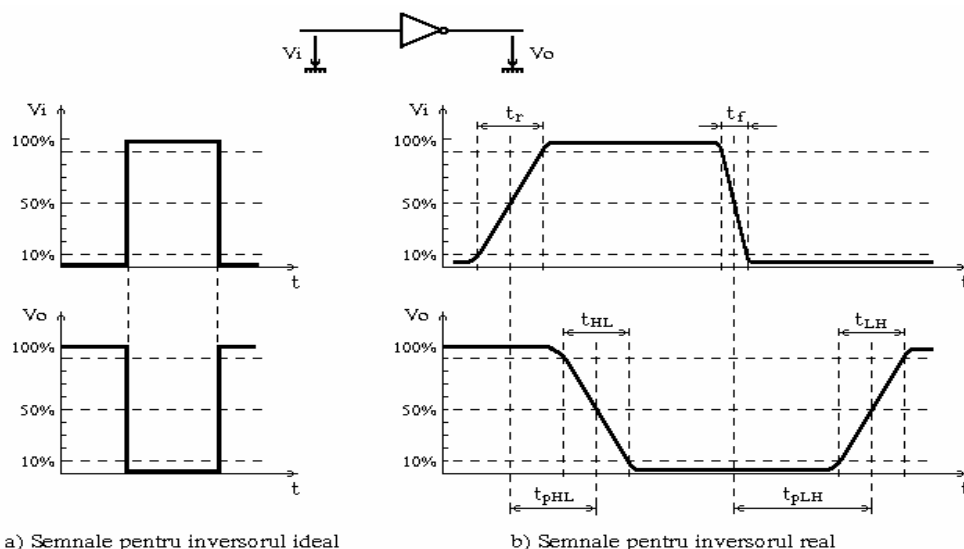


Fig. 2. Definirea timpilor de propagare

Facem precizarea că  $t_{pHL} \neq t_{pLH}$ . Pentru anumite familii logice, în foile de catalog vom găsi aceleași valori pentru  $t_{pHL}$  și  $t_{pLH}$ . Aceasta nu înseamnă că  $t_{pHL} = t_{pLH}$ , datele de catalog fac referire la valorile maxime pentru timpii de propagare. Valorile limită pot fi egale, dar nu și cele efective de la nivelul fiecărei porți.

Pentru a se asigura condiții optime de procesare a semnalelor digitale se recomandă ca perioada  $T$ , a semnalului de intrare, să satisfacă relația:

$$T \geq (20 \div 50) \times t_p$$

Dacă relația de mai sus nu este satisfăcută, există riscul ca semnalul de intrare să nu se mai regăsească la ieșirea circuitului.

## 2.4. Etaje de ieșire specifice familiei TTL standard

Schema bloc, de principiu, a unei porți realizate în tehnologie TTL este prezentată în figura 3. Dacă schema electrică a blocurilor componente diferă de la o subfamilie la alta, rolul lor funcțional se păstrează în totalitate.

Trebuie să precizăm faptul că familia TTL, conține circuite electronice ce lucrează în regim de comutație. În acest regim particular de lucru, fiecare tranzistor se comportă ca un comutator comandat electronic. În orice moment de timp el se poate afla în una din următoarele stări: **blocat** = **contact deschis**, sau **saturat** = **contact închis**. Din acest motiv, în figurile următoare, în schemele echivalente, tranzistoarele sunt înlocuite de comutatoare.

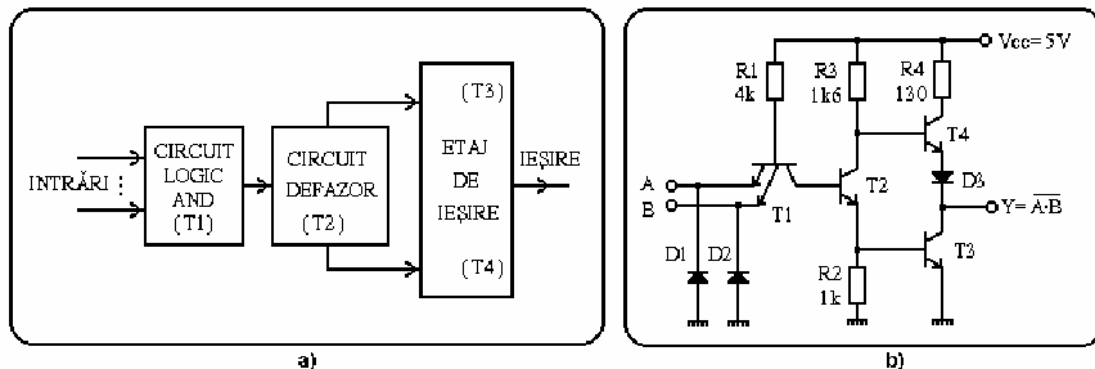


Fig.3. a) Schema bloc a unei porți realizată în tehnologie TTL; b) poartă NAND

**Etajul de intrare** este de regulă format dintr-un tranzistor multiemitor și are rolul de a realiza funcția logică. Acest etaj trebuie să fie proiectat astfel încât să nu necesite curenți mari de comandă și, în plus, să prezinte protecție la eventualele tensiuni negative ce pot fi aplicate intrărilor sale.

**Etajul defazor** are rolul de a genera două semnale în antifază ce sunt necesare pentru atacul etajului de ieșire.

**Etajul de ieșire** trebuie să asigure la ieșirea circuitului logic valori impuse de tensiune pentru fiecare stare logică.

Etajul final poate fi realizat în una din următoarele variante: **în contratimp** (etaj TOTEM POLE), **cu ieșire în gol** (OPEN COLLECTOR) sau **etaj THREE STATE** (TRISTATE). Așadar, aceeași poartă logică, poate fi realizată din punct de vedere tehnologic în trei variante distincte, funcție de etajul său final.

O serie de proprietăți ale porților logice sunt strâns legate de tipul etajului de ieșire. Din acest motiv, prezentăm pe scurt particularitățile fiecărui tip de etaj de ieșire.

### a) Etajul de ieșire în contratimp

Etajul de ieșire în contratimp, denumit și etaj TOTEM POLE, este etajul standard de ieșire al circuitelor logice realizate în tehnologie bipolară. Dacă în foile de catalog nu se fac referiri exprese la tipul etajului de ieșire, atunci, în mod implicit, acesta este de tip TOTEM POLE. În esență, un etaj în contratimp este format din două tranzistoare ce sunt conectate în serie între tensiunea de alimentare și masă (vezi fig. 4).

Schema simplificată a unui astfel de etaj, în care tranzistoarele au fost înlocuite prin comutatoare se prezintă în figura 4. a). Pe această figură se observă că ieșirea Y se află în unu logic, numai dacă avem simultan K4 închis și K3 deschis. Similar, ieșirea se află în starea zero logic dacă avem în același timp K4 deschis și K3 închis. Comanda de închidere/deschidere a comutatoarelor provine de la etajul defazor și este concepută astfel încât cele două comutatoare să fie acționate în contratimp.

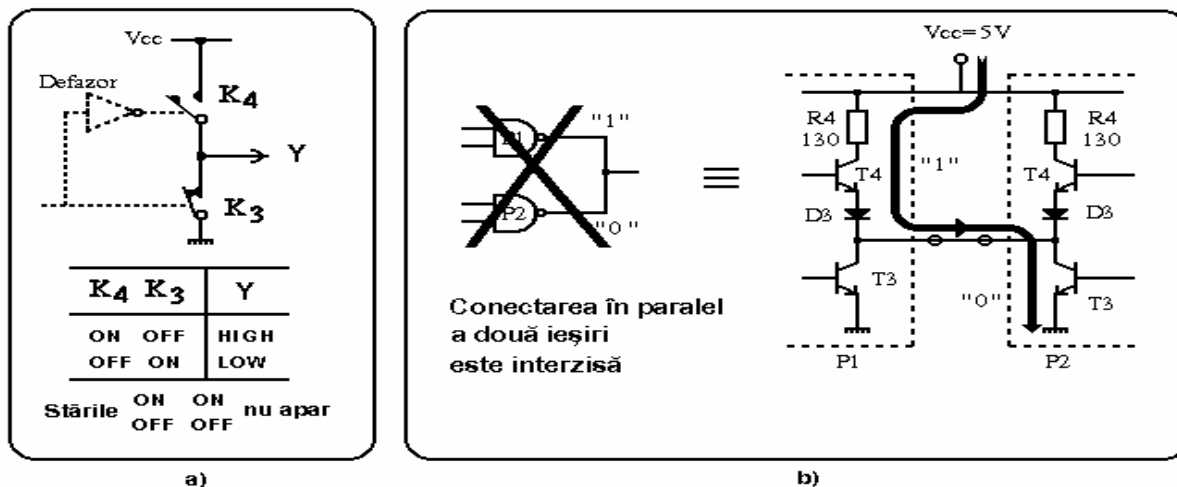


Fig. 4. Etajul de ieșire TOTEM POLE; a) schema electrică echivalentă; b) aspecte nedorite ce apar la conectarea în paralel a ieșirilor de tip TOTEM-POLE

*Proprietățile (particularitățile) etajelor de ieșire în contratimp:*

- Două sau mai multe ieșiri de acest tip nu pot fi conectate în paralel, deoarece apare o circulație de curent de valoare mare pentru cazul în care starea logică a ieșirilor este diferită. Această situație este prezentată în figura 4. b). Pentru aceste cazuri există riscul ca ambele circuitele să se distrugă.
- O ieșire de acest tip nu trebuie niciodată conectată la masă, la  $V_{cc}$ , sau la oricare altă sursă de semnal deoarece există riscul distrugerii circuitului.
- De regulă, o ieșire de acest tip este utilizată pentru comanda altor intrări digitale. În cazuri extreme poate fi utilizată și pentru comanda unor sarcini rezistive dacă sunt corect alese.
- Asigură cu resurse interne nivelele de tensiune necesare pentru ambele stări logice;
- În regim staționar acest etaj prezintă un tranzistor blocat iar celălalt saturat.
- Impedanța de ieșire este de același ordin de mărime atât pentru "zero logic" (T3 saturat și T4 blocat), cât și pentru starea de "unu logic" (T3 blocat și T4 saturat).
- Tranzistoarele T3 și T4 se află simultan în conducție pentru un interval scurt de timp ce corespunde tranziției din "1" în "0" a ieșirii. Pe acest interval, curentul absorbit de circuit este mare și conexiunile de alimentare ale circuitului răspund preponderent inductiv, provocând o scădere a tensiunii de alimentare. Din acest motiv, lângă capsula circuitului integrat, între  $V_{cc}$  și masă, trebuie conectat un condensator de decuplare de cca. 10 nF;

**b) Etajul final cu ieșire în gol (OPEN COLLECTOR)**

Etajul final de tip "colector în gol", se obține dintr-un etaj de ieșire în contratimp prin eliminarea repetorului pe emitor T4, rămâne tranzistorul T3 al cărui colector este conectat la ieșirea porții (vezi figura 5).

Acest etaj poate genera un bun "zero logic", prin saturarea tranzistorului T3, dar pentru "unu logic" va fi necesară, pe lângă blocarea lui T3, și utilizarea unei rezistențe externe conectate spre  $V_{cc}$ . Așadar, circuitul generează autoritar starea de "zero logic" și este doar permisiv pentru starea de "unu logic".

Valoarea rezistenței externe adăugată de către utilizator se calculează în funcție de condițiile concrete de lucru ale porții.

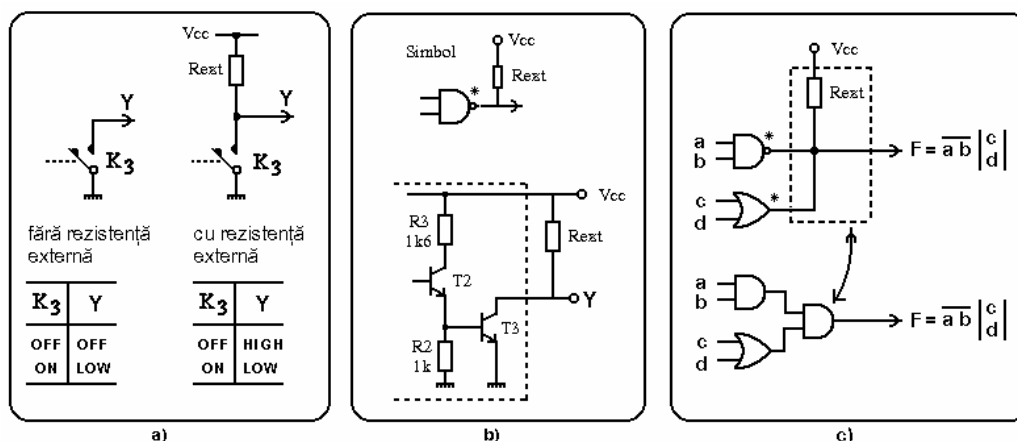


Fig. 5. Etajul de ieșire open collector; a) schema electrică echivalentă; b) simbolul și modul de conectare a rezistenței externe; c) efectul de AND cablat ce apare la conectarea în paralel a porților cu ieșire în gol pe o aceeași rezistență externă.

*Proprietățile (particularitățile) ieșirilor de tip open collector:*

- Funcționarea corectă a porții este posibilă doar în prezența rezistenței externe.
- Două sau mai multe ieșirile de acest tip pot fi conectate în paralel pe aceeași rezistență externă fără a exista riscul distrugerii circuitelor. Acest mod de lucru face ca nodul de conexiune să se comporte ca o poartă AND virtuală, denumită ȘI CABLAT, ale cărei intrări sunt chiar ieșirile porților concrete. Un exemplu de acest fel este prezentat în figura 5.c.
- O ieșire de acest tip poate fi utilizată pentru comanda unor sarcini ce operează la tensiuni de alimentare mai mari de 5V.
- Nivelul de "unu logic" este generat precar, prin intermediul rezistenței externe.
- Timpii de front pentru sarcini capacitive sunt inegali.
- Impedanțele de ieșire sunt net diferite pentru cele două stări logice.

**c) Etajul de ieșire tristate**

Ieșirea tristate prezintă, pe lângă cele două stări logice bine cunoscute LOW și HIGH, o stare suplimentară denumită stare de înaltă impedanță, notată HiZ. Circuitele logice care prezintă această facilități au o intrare suplimentară de comandă, denumită ENABLE, prin intermediul căreia se poate obține starea HiZ.

Starea HiZ înseamnă dezactivarea completă a ieșirii, lucru posibil prin blocarea simultană a celor două transistoare ale etajului final în contratimp.

În starea de înaltă impedanță tensiunea de ieșire are valoarea fixată de potențialul care există pe linia de magistrală la care este cuplată ieșirea porții (acest potențial este forțat pe magistrală de către o altă poartă). O structură de inversor tristate este prezentată în figura 6 b).

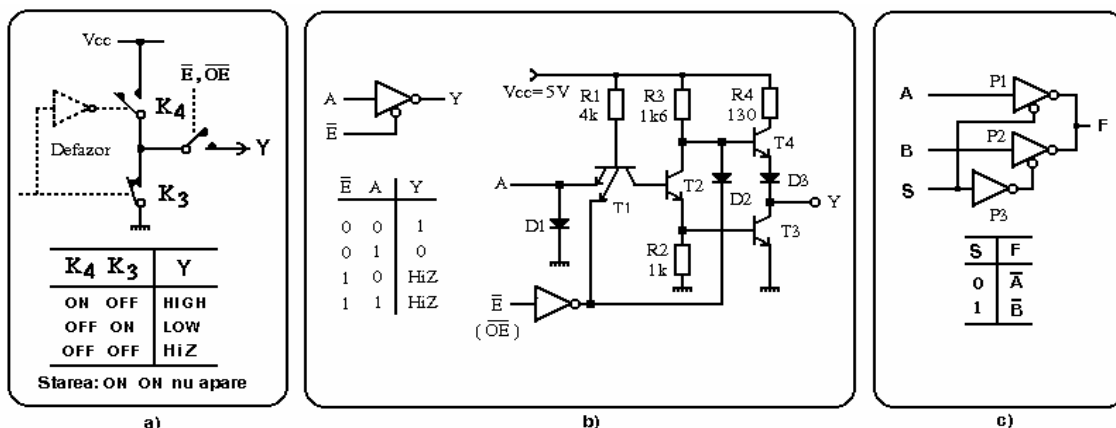


Fig. 6. Poarta cu ieșire tristate: a) schema electrică echivalentă; b) simbol, tabel de adevăr; c) exemplu de conectare în paralel a două inversoare cu ieșiri tristate

Proprietățile (particularitățile) etajelor de ieșire tristate:

- Permite conectarea în același punct a mai multor ieșiri, cu condiția ca numai una să fie validată (activată), la un moment dat, vezi figura 6.c.
- Asigură cu resurse proprii nivelele de tensiune pentru ambele stări logice.
- Circuitele prevăzute cu ieșiri tristate prezintă avantajul că se pot conecta ușor la magistralele de date sau adrese ale sistemelor cu microprocesoare.
- Oferă impedanțe mici la ieșire, și de același ordin de mărime, pentru ambele stări logice (ca la poarta TTL standard);
- Nu necesită rezistență externă ca în cazul etajelor open collector;
- În starea de înaltă impedanță, o ieșire tristate încarcă nesemnificativ circuitele cu care sunt cuplate la ieșire.

## 2.5. Factorul de încărcare

În foarte multe scheme apare nevoia ca ieșirea unui circuit logic să comande două sau mai multe intrări ale altor circuite logice. Analiza pur logică a circuitelor digitale nu impune nici o restricție în această privință. În practică trebuie impuse restricții deoarece ieșirea unui circuit are posibilități limitate de a genera sau prelua curenți.

Din figura 7 se observă că sensul de curgere al curenților depinde de starea logică transmisă pe linia de legătură. În plus, se remarcă faptul că, pe măsură ce crește numărul sarcinilor logice comandate de ieșirea unei porți, crește și valoarea curentului generat/prelucuat de către aceasta. Dacă numărul sarcinilor logice este prea mare, poarta logică nu va mai putea menține nivelele de tensiune acceptabile și informația logică se poate pierde.

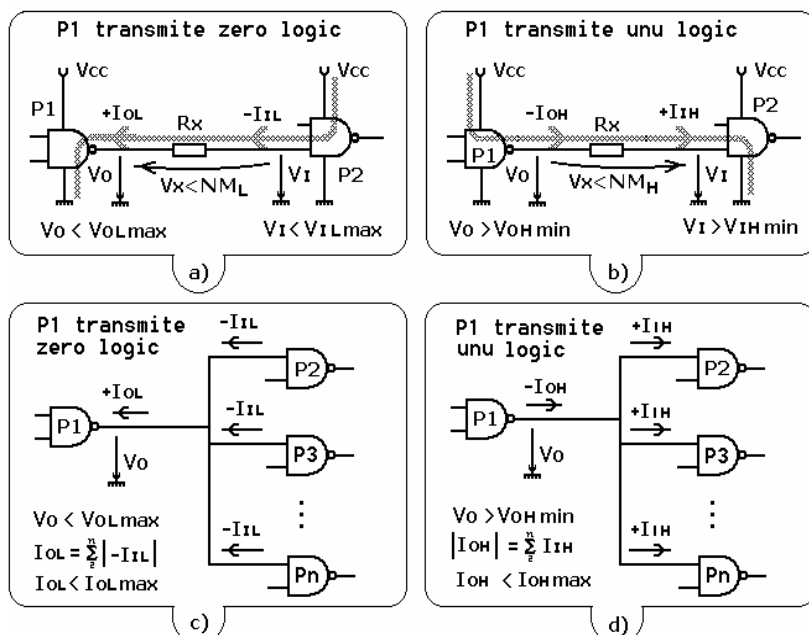


Fig. 7. Circulația curenților funcție de starea logică și factorul de încărcare

Așadar, un circuit logic trebuie să poată genera/prelua la ieșire un curent mai mare sau egal cu suma curenților preluați/generați de toate porțile care sunt conectate la aceea ieșire. În același timp el trebuie să asigure și nivelul garantat al tensiunii ce corespunde stării logice transmise.

De regulă, pentru stabilirea semnelor curenților se face apel la următoarea convenție: curentul care intră într-o bornă are semnul pozitiv, iar cel care iese dintr-o bornă are semnul negativ.

Semnificația curenților din figura 7 este următoarea:

- $I_{IL}$  - curentul de intrare în starea Low. Valoarea maximă a acestui curent este dependentă de familia logică din care provine circuitul (vezi tabelul 3).
- $I_{IH}$  - curentul de intrare în starea High. Valoarea maximă este dependentă de familia din care provine circuitul logic.
- $I_{OL}$  - curentul de ieșire în starea Low. Valoarea maximă este dependentă de familia din care provine circuitul logic.
- $I_{OH}$  - curentul de ieșire în starea High. Valoarea maximă este dependentă de familia din care provine circuitul logic.

Tabelul 3. Valorile curenților de intrare/ieșire pentru diferite familii logice

SERIA	INTRARE		IEȘIRE	
	nivel LOW $I_{IL}$ max [mA]	nivel HIGH $I_{IH}$ max [μA]	nivel LOW $I_{OL}$ max [mA]	nivel HIGH $I_{OH}$ max [mA]
74 **	- 1,6	40	16	- 0,4
74 S **	- 2,0	50	20	- 1,0
74 LS **	- 0,36	20	8	- 0,4
74 AS **	- 2,0	20	4 / 8	- 0,4
74 ALS **	- 0,1	20	8	- 0,4

Prin definiție **factorul de încărcare la ieșire**  $FO$ , (fan-out, output loading factor, sortance), este un număr ce indică capacitatea ieșirii de a comanda în siguranță, (cu asigurarea unor nivele corecte de tensiune), intrările altor circuite din aceeași familie. Fan-out este, în general, diferit pentru cele două stări logice, el se poate calcula cu relațiile:

$$FO_{LOW} = I_{OL\ min} / I_{IL\ max}$$

$$FO_{HIGH} = I_{OH\ min} / I_{IH\ max}$$

$$FO = \min \{ FO_{LOW}, FO_{HIGH} \}$$

Prin definiție **factorul de încărcare al intrării**,  $FI$ , (fan-in, input loading factor, facteur de charge) reprezintă numărul de unități de sarcină percepute la intrarea unui circuit digital.  $FI$  este dependent de complexitatea circuitului logic și poate avea valori mai mari decât 1. De exemplu, o ieșire TTL standard poate comanda 10 intrări cu  $FI=1$  sau 5 intrări cu  $FI=2$ .

## 2.5. Caracteristica de transfer în tensiune a circuitelor logice / Tensiunea de prag

Caracteristica de transfer în tensiune reprezintă dependența statică între tensiunea de intrare în poartă și tensiunea de ieșire,  $V_o = f(V_i)$ . Această caracteristică prezintă o importanță deosebită deoarece oferă informații despre valorile efective ale unor mărimi ca: marginea de zgomot, nivelele limită ale tensiunii de intrare, lățimea benzii interzise, etc.

Pentru circuitele din aceeași familie logică, caracteristica de transfer în tensiune (CTT), este similară ca formă. Ea poate să difere puțin de la un circuit la altul numai prin valorile efective ale coordonatelor punctelor de frângere.

În mod curent, majoritatea circuitelor logice prezintă o caracteristică de transfer standard și numai o mică parte dintre ele prezintă o caracteristică specială de tip trigger Schmitt.

### a) CTT standard

Pentru exemplificare, în figura 8 se prezintă CTT tipică pentru inversorul SN7404 (familia TTL standard). Analizând caracteristica, se observă că segmentele AB și DE corespund benzilor permise ale tensiunilor de intrare pentru cele două stări logice, iar segmentele BC și CD corespund benzii interzise. Dintr-un alt punct de vedere, dacă ne raportăm la un semnal dreptunghiular aplicat la intrare, segmentele AB și DE corespund palielor, iar BC în prelungire cu CD fronturilor.

**Marginea de zgomot** efectivă se determină observând că semnalul sumă (semnal util + zgomot), nu trebuie să depășească abscisa punctului C pentru "unu logic", respectiv D pentru "zero logic".

**Nivelele de tensiune** garantate pentru intrare (0,8V și respectiv 2V), sunt, așa după cum se vede în figură, mult în afara zonei interzise efective, în scopul de a reduce efectul variațiilor de temperatură și dispersia tehnologică în buna funcționare a porții. În practică zona interzisă este considerată acoperitor în intervalul 0,8V÷2V.

**Tensiunea de prag** reprezintă acea valoare a tensiunii de intrare care, dacă este depășită, poate duce la schimbarea stării logice a ieșirii.

*Tensiunea de prag realizează separarea stărilor logice de la intrare pentru regimul dinamic.*

Pe caracteristica de transfer, tensiunea de prag este situată la mijlocul segmentului CD. Dacă tensiunea de intrare este menținută în regiunea CD, există riscul de apariție a unor oscilații de frecvență relativ mare la ieșirea circuitului. Pentru a evita amorsarea acestor oscilații, trebuie ca durata de traversare a zonei interzise (segmentul CD), de către semnalul de intrare, să nu depășească 40÷50 ns. În consecință, se recomandă ca durata fronturilor de atac ale semnalului de intrare să fie sub 50 ns. Cu cât caracteristica de transfer va fi mai verticală, se vor putea utiliza fronturi de atac mai lungi.

Din cele prezentate mai sus, se poate trage concluzia că CTT standard prezintă două particularități importante:

- are o singură tensiune de prag, indiferent dacă tensiunea de intrare evoluează în sens crescător sau în sens descrescător;



- în apropierea tensiunii de prag, panta CTT nu este perfect verticală, de aici necesitatea ca semnalele de atac ale acestor circuite să prezinte fronturi cu durata cât mai redusă.

### b) CTT de tip trigger Schmitt

Circuitele cu caracteristică de tip trigger Schmitt au în plus față de cele cu caracteristică standard, un etaj de amplificarea special plasat între circuitul de intrare și etajul defazor. Câștigul suplimentar datorat acestui etaj cu cuplaj în emitor, face ca zonele de tranziție să fie practic nule, caracteristica fiind verticală.

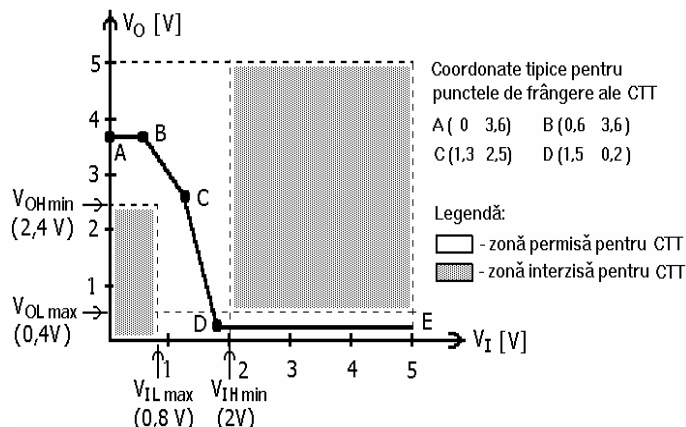


Fig. 8. Caracteristica de transfer în tensiune (CTT), pentru inversorului TTL - standard

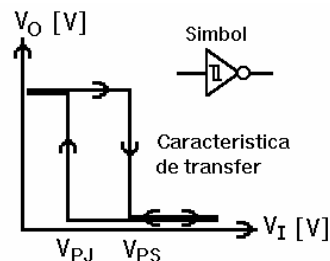


Fig. 9. Simbolul și caracteristica de transfer pentru inversorul trigger Schmitt SN7414

Pentru exemplificare, în figura 9 se prezintă o caracteristică de transfer de tip trigger Schmitt specifică unui inversor. Forma particulară a caracteristicii de tip trigger Schmitt face ca, la circuitele cu astfel de caracteristică, să apară următoarele proprietăți:

- semnalele de intrare **pot avea fronturi oricât de lente**, zona interzisă fiind foarte mică, traversare ei este posibilă fără amorsarea oscilațiilor;
- existența a două tensiuni de prag:  $V_{PS}$  - prag valabil pentru sensul crescător al tensiunii de intrare și  $V_{PJ}$  - prag valabil pentru sensul descrescător al tensiunii de intrare;
- apariția histerizisului (drumuri diferite de parcurgere a caracteristicii de transfer în funcție de sensul de evoluție al tensiunii de intrare) are ca efect creșterea marginii reale de zgomot. Spre exemplu, pentru inversorul SN7414, tensiunea de intrare admisă pentru zero logic poate urca până la cca. 1,6V iar pentru unu logic, tensiunea de intrare poate coborî până la cca. 0,8V. Creșterea marginii de zgomot a fost obținută prin suprapunere în zona centrală a benzilor de tensiune asociate stărilor logice de la intrarea circuitului.

Facem precizarea că singura deosebire dintre circuitele 7404 și 7414 este dată de caracteristica de transfer. Din punct de vedere logic ele realizează aceeași funcție – negarea valorii logice de la intrare.

Din punct de vedere electric, între răspunsul celor două circuite nu apar diferențe semnificative dacă semnalul de intrare are fronturi cu durată redusă și nu este afectat de zgomot (condiții favorabile de lucru). Diferențe semnificative apar atunci când semnalul de intrare are fronturi lent variabile și/sau este afectat de zgomot. În astfel de condiții, răspunsul circuitului cu caracteristică trigger Schmitt est net mai bun. În desfășurarea lucrării se va pune în evidență acest aspect.



## 3. Desfășurarea lucrării

### 3.1. Verificarea nivelelor de tensiune

Cu ajutorul montajului din figura 10 se determină modificarea nivelelor de tensiune asociate stărilor logice pentru diverse încărcări ale porții de test. Determinările se fac pentru următoarele tipuri de circuite: 7404; 74LS04; 74HCT04. Pentru fiecare circuit în parte, rezultatele măsurătorilor se trec în tabele similare tabelului 3.1.

Modul de lucru:

- se trec comutatoarele K1 și K2 pe poziția a;
- se introduce în soclu unul din circuitele specificate mai sus;
- se încarcă progresiv ieșirea porții testate prin realizarea de combinații diverse ON/OFF ale comutatoarelor din pachetul SW;
- pentru fiecare factor de încărcare și pentru fiecare stare logică în parte, se măsoară cu osciloscopul tensiunile din punctele de test C și D, și se completează tabelul 4;
- la rubrica de observații se specifică dacă în punctele X1, X2, X4, X8, nivelele de tensiune asociate stărilor logice mai au sau nu valori acceptabile;

**Întrebări:**

- Ce se întâmplă cu tensiunea de ieșire pentru fiecare stare logică pe măsură ce crește factorul de încărcare ?
- Există cazuri în care o ieșire logică trebuie să comande un număr mai mare de sarcini logice decât FAN OUT. Cum se poate rezolva o astfel de situație?

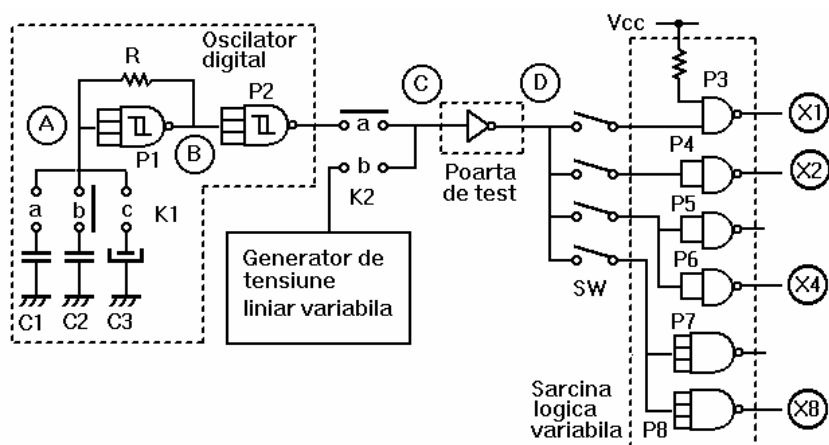


Fig. 10. Montaj experimental pentru determinarea parametrilor porților logice

Tabelul 4

Număr de sarcini logice	Tensiunea de intrare		Tensiunea de ieșire		Observații
	$V_{IL}$	$V_{IH}$	$V_{OL}$	$V_{OH}$	
0					
1					
.					
.					
15					

### 3.2. Determinarea tensiunilor de prag

Cu ajutorul montajului din figura 10 se determină tensiunile de prag atât pentru sensul crescător, cât și pentru cel descrescător al tensiunii de intrare. Pentru această determinare, la intrarea porții de test se aplică o tensiune în dinte de fierăstrău.

Determinările se fac pentru următoarele tipuri de circuite: 7404, 7414, 74HCT04, 74HCT14.

**Modul de lucru:**

- se trece comutatorul K2 pe poziția b;
- se încarcă ieșirea porții de test cu o singură sarcină logică;
- se introduce în soclu unul din circuitele specificate mai sus;
- se fac următoarele reglaje la osciloscop: se reglează atenuatorul pe poziția 1V/div pentru ambele canale; se reglează poziția de zero a ambelor trase astfel încât să se suprapună una peste alta (se are în vedere ca suprapunerea să se facă în dreptul unei gradații orizontale a ecranului gradat);
- pe canalul A al osciloscopului se aplică semnalul în dinte de fierăstrău (punctul C), iar pe canalul B semnalul de la ieșirea porții de test (punctul D);
- se vizualizează formele de undă și se determină punctele de intersecție ale celor două semnale - acestea sunt chiar valorile reale ale tensiunilor de prag;
- se completează tabelul 5 iar pentru circuitele 7404 și 7414 se desenează și formele de undă;

Tabelul 5

Tipul circuitului	$V_{OL}$	$V_{OH}$	$V_{TR+}$	$V_{TR-}$
7404				
7414				
74HCT04				
74HCT14				

**Întrebări:**

- Cum explicați faptul că, deși circuitele 7414 și 74HCT14 sunt inversoare trigger Schmitt, valorile tensiunilor de prag sunt diferite? Dar pentru circuitele 7404 și 74HCT04?

### 3.3. Vizualizarea caracteristici de transfer

Pentru vizualizarea acestei caracteristici se utilizează schema de la punctul anterior cu deosebirea că osciloscopul este configurat în modul de lucru XY.

Se desenează caracteristica de transfer pentru circuite specificate la punctul anterior al lucrării.

**Întrebări:**

- De ce nu apar pe osciloscop (sau sunt greu vizibile), ramurile verticale ale caracteristicilor de tip trigger Schmitt?

**3.4. Studiul funcționării oscilatorului de relaxare**

O aplicație foarte des întâlnită a porților cu caracteristică de tip trigger Schmitt o constituie oscilatorul de relaxare. Schema electrică a unui astfel de oscilator se poate identifica în figura 10. În componența oscilatorului intră poarta P1, rezistorul R și condensatorul selectat prin intermediul comutatorului K1.

**Modul de lucru:**

- pentru diverse valori ale condensatorului (selectabile prin intermediul comutatorului K1), se determină cu osciloscopul, frecvența semnalului generat;
- pentru o poziție convenabilă a comutatorului K1, se vizualizează și se desenează corelat în timp formele de undă din punctele A și B (vezi figura 10).

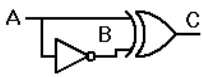
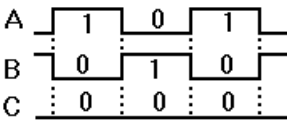
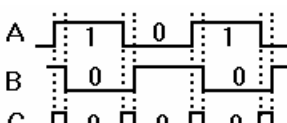
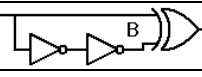
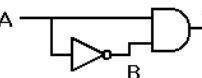
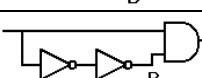
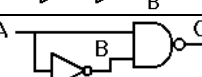
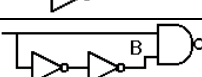
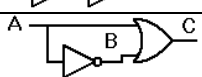
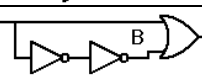
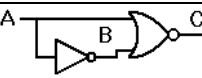
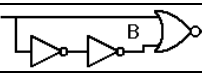
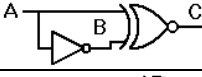
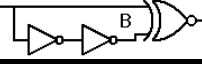
**Întrebări:**

- Analizând schema electrică din figura 10, se constată faptul că NAND-ul cu patru intrări lucrează în regim de inversor. Poate fi înlocuit acest NAND cu un inversor de tipul 7404? Dar cu unul de tip 7414? Motivați-vă răspunsul.
- Ce se întâmplă cu frecvența semnalului generat de oscilator dacă se mărește capacitatea condensatorului? Dar dacă valoarea condensatorului se mărește foarte mult?
- Poarta P2 (vezi fig. 10 ) este strict necesară pentru ca schema să oscileze?
- Poate fi utilizat acest oscilator pentru realizarea unui ceas electronic?
- Se modifică frecvența de oscilație dacă circuitul 7413 se înlocuiește cu altul care are tensiunile de prag mult diferite? Motivați-vă răspunsul.

**3.5. Efectul timpului de propagare - generarea de impulsuri din tranziția semnalului de intrare**

- a) Prin analiză logică, sau prin simulări pe calculator, se cere completarea tabelului de mai jos conform exemplului din linia 1.

Tabelul 6

Nr. crt.	Schema logică	Semnale de ieșire pentru cazul ideal ( $t_p=0$ )	Semnale de ieșire pentru cazul real ( $t_p \neq 0$ )
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			

**b)** Se realizează pe macheta de test schema din linia 1 a tabelului 6, după care se vizualizează cu ajutorul osciloscopului cu două canale semnalele din punctele A, B și C. Pentru a pune mai bine în evidență fenomenele, între A și B se vor lega în serie 3 inversoare. Se determină lățimea impulsurilor de ieșire ce sunt datorate timpului de propagare. Se desenează corelat în timp cele trei semnale și se compară cu cele determinate teoretic.

*Întrebări:*

- Precizați schemele din tabelul 6 care dau răspunsuri similare la ieșire.
- Cum se explică dependența lățimii impulsurilor de ieșire, în funcție de tranziția negativă sau pozitivă a semnalului de intrare (vezi experimentul de la subpunctul b)?

### 3.6. Efectul timpului de propagare - oscilatorului în inel

Se realizează pe macheta de test un oscilator în inel similar celui prezentat în figura 11.

- Pentru  $K2=a$  și pentru fiecare poziție a comutatorului  $K1$ , se vizualizează cu ajutorul osciloscopului semnalul din punctul Q. Pentru situațiile în care în punctul Q apar oscilații se va determina frecvența acestora.
- Se repetă subpunctul anterior pentru situația  $K2=b$ .

Determinările se vor face pentru următoarele tipuri de circuite: 7404, 7414, 74LS04, 74F04 și 74ALS04.

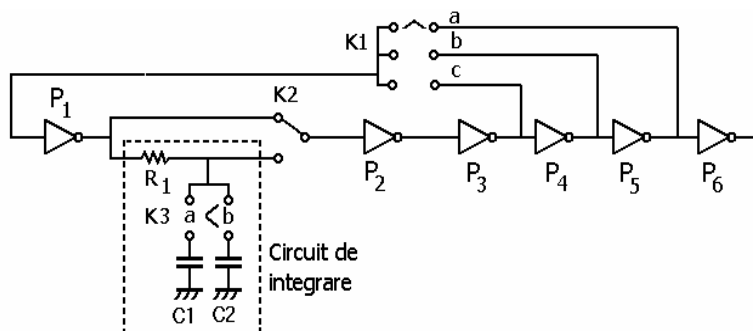


Fig. 11: Montaj experimental pentru studiul oscilatorului în inel

*Întrebări:*

- Care este condiția de oscilație pentru cazul în care  $K1$  este pe poziția  $a$ ?
- Cum explicați faptul că schema încetează să mai oscileze dacă în buclă se introduce un număr par de inversoare?
- Cum se modifică frecvența de oscilație prin creșterea numărului de inversoare în bucla de reacție?
- Cum explicați faptul că, pentru același număr de inversoare pe bucla de reacție, frecvența de oscilație diferă destul de mult pentru circuite analizate?
- Cum explicați modificarea frecvenței de oscilație atunci când se introduce circuitul de integrare?
- Cum explicați faptul că schema încetează să mai oscileze dacă capacitatea din circuitul de integrare este prea mare?
- În situația  $K2 = b$  și  $K1 = c$ , se mai poate introduce un integrator suplimentar corect calculat între  $P2$  și  $P3$ ? Ce se întâmplă în acest caz cu valoarea frecvenței semnalului generat?
- Pentru ca schema să oscileze este nevoie absolută de integrator?



## A T E N Ţ I E !

### Reguli de operare cu circuitele integrate digitale

#### Reguli privind alimentarea circuitelor integrate digitale

◆ Funcționarea circuitelor logice nu este posibilă fără conectarea acestora la o tensiune de alimentare corespunzătoare:

- $5V \pm 0,25V$  pentru toate seriile TTL;
- $5V \pm 0,5V$  pentru seria 74 HC \*\*, (CMOS);
- $3 \div 15V$  pentru seria CD 4000, (CMOS);

◆ Pentru menținerea constantă a tensiunii de alimentare, pe durata comutării ieșirii dintr-o stare în alta, este necesară utilizarea unui condensator de decuplare, plasat în imediata apropiere a circuitului integrat și conectat în paralel pe bornele de alimentare ale acestuia;

#### Reguli referitoare la conectarea intrărilor circuitelor integrate digitale

◆ Intrările neutilizate ale circuitelor digitale nu trebuie lăsate neconectate deoarece sunt sensibile la zgomot și pot altera funcționarea corectă a circuitului;

◆ Intrările neutilizate se vor conecta la stări logice alese astfel încât să nu intervină în funcționarea normală a circuitului. Spre exemplu, dacă se dorește starea de "unu logic" aceasta se poate obține în mai multe moduri:

- prin conectarea intrării la o sursă independentă de tensiune între  $2,4 \div 3,5V$ ;
- prin legare în paralel la intrări care îndeplinesc aceeași funcție logică - metoda prezintă dezavantajul că încarcă inutil poarta care comandă;

- prin conectarea la  $V_{cc}$  prin intermediul unei rezistențe de  $1k\Omega$ .

Starea de "zero logic" se obține prin conectarea directă la masă a intrării, pentru familia TTL, sau prin intermediul unei rezistențe pentru familia CMOS.

- ◆ Dacă totuși o intrare TTL este neconectată ("lăsată în aer"), aceasta va fi interpretată de circuit ca fiind în stare logică HIGH;

- ◆ Intrările porților pot fi conectate în paralel;

#### ***Reguli referitoare la conectarea ieșirilor circuitelor integrate digitale***

- ◆ De regulă, o ieșire digitală se conectează, după caz, la una sau mai multe intrări digitale;
- ◆ Este interzisă conectarea ieșirii unui circuit digital, chiar și pentru intervale scurte de timp, la masă, la tensiunea de alimentare, sau la oricare altă sursă de semnal;
- ◆ Ieșirile nu pot fi conectate în paralel decât în cazul utilizării etajelor tristate sau open collector;
- ◆ Dacă numărul de intrări este insuficient, mărirea acestuia se poate face prin:
  - cuplarea mai multor porți la intrarea alteia;
  - utilizarea unei porți expandoare;
  - utilizarea de funcții cablate utilizând porți open collector

Lucrarea nr. 2: **Implementarea funcțiilor binare cu rețele de porți logice****1. Scopul lucrării**

În această lucrare se face o prezentare sintetică a principalelor etape ce trebuie parcurse în procesul de implementare a funcțiilor binare cu ajutorul porților logice.

Prima parte a lucrării prezintă abordarea clasică (bazată pe utilizarea circuite integrate digitale de complexitate redusă) iar partea a doua prezintă abordarea modernă (bazată pe utilizarea de structuri logice reconfigurabile de tip CPLD sau FPGA). În strânsă legătură cu partea a doua a lucrării, se prezintă modul în care un circuit sau sistem digital poate fi implementat într-o structură reconfigurabilă cu ajutorul mediului de dezvoltare ISE-WebPack, produs de firma Xilinx. Totodată sunt prezentate primele noțiuni despre limbajul VHDL, urmând ca acestea să fie aprofundate în lucrările viitoare.

**2. Considerente teoretice****2. 1. Sinteza funcțiilor binare folosind rețele de porți logice – Metoda clasică**

Realizarea practică a funcțiilor binare se poate face în mai multe moduri: folosind porți logice, folosind circuite de complexitate medie (multiplexoare sau demultiplexoare), folosind memorii ROM sau EEPROM, folosind structuri logice programabile (PAL, GAL CPLD sau chiar FPGA), etc.

În această secțiune a lucrării se prezintă implementarea bazată pe porți logice. Succesiunea și denumirea etapelor necesare în procesul de sinteză sunt prezentate în tabelul 1.

La rândul său, implementarea cu porți logice, poate avea mai multe variante:

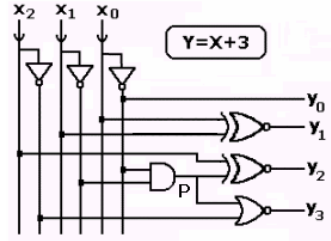
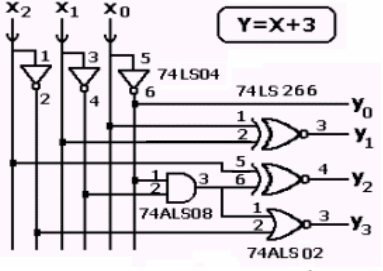
- folosind **logică combinată** – se permite utilizarea oricărui tip de poartă logică;
- folosind **logică de același tip** – se permite utilizarea unui singur tip de poartă logică, fie NAND, fie NOR;

**2.1.1. Logica combinată**

Tabelul 1

Tabelul

Nr. etapă	Denumire etapă	Exemplu																																																																																	
1	<b>Definirea foarte exactă a funcției logice ce trebuie realizate.</b>  Se pleacă de la descrierea în termeni naturali a funcționării circuitului.	<b>Ce trebuie să facă circuitul ?</b> Adună la numărul de intrare $X$ , constanta zecimală 3. Așadar, funcția de transfer a CLC- ului este: $Y = X + 3$ . Starea logică a intrărilor este interpretată ca fiind scrierea binară a numărului de intrare $X$ , iar starea logică a ieșirilor se consideră a fi scrierea binară a numărului $Y$ . <b>Câte intrări și câte ieșiri are circuitul ?</b> Din datele inițiale știm că are 3 intrări. Numărul de ieșiri trebuie deduse ținând seama de funcția realizată de circuit. Cel mai mare număr exprimat pe 3 biți este $X=7$ , caz în care $Y=7+3=10$ . Rezultă că circuitul trebuie să prezinte 4 ieșiri.																																																																																	
2	<b>Alegerea unei modalități de descriere a funcției logice.</b>  În principiu, se poate alege orice metodă de reprezentare cunoscută.	Pentru acest circuit, cea mai adecvată metodă de reprezentare este dată de tabelul de adevăr complet. <table><tr><th><math>X</math></th><th><math>x_2</math></th><th><math>x_1</math></th><th><math>x_0</math></th><th><math>y_3</math></th><th><math>y_2</math></th><th><math>y_1</math></th><th><math>y_0</math></th><th><math>Y</math></th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>3</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>4</td></tr><tr><td>2</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>5</td></tr><tr><td>3</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>6</td></tr><tr><td>4</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>7</td></tr><tr><td>5</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>8</td></tr><tr><td>6</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>9</td></tr><tr><td>7</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>10</td></tr></table>	$X$	$x_2$	$x_1$	$x_0$	$y_3$	$y_2$	$y_1$	$y_0$	$Y$	0	0	0	0	0	0	1	1	3	1	0	0	1	0	1	0	0	4	2	0	1	0	0	1	0	1	5	3	0	1	1	0	1	1	0	6	4	1	0	0	0	1	1	1	7	5	1	0	1	1	0	0	0	8	6	1	1	0	1	0	0	1	9	7	1	1	1	1	0	1	0	10
$X$	$x_2$	$x_1$	$x_0$	$y_3$	$y_2$	$y_1$	$y_0$	$Y$																																																																											
0	0	0	0	0	0	1	1	3																																																																											
1	0	0	1	0	1	0	0	4																																																																											
2	0	1	0	0	1	0	1	5																																																																											
3	0	1	1	0	1	1	0	6																																																																											
4	1	0	0	0	1	1	1	7																																																																											
5	1	0	1	1	0	0	0	8																																																																											
6	1	1	0	1	0	0	1	9																																																																											
7	1	1	1	1	0	1	0	10																																																																											
3	<b>Alegerea modalității (sau a tehnologiei) de implementare.</b>	Optăm pentru utilizarea porților logice deoarece scopul acestei lucrări constă în prezentarea acestei metode. În această etapă trebuie avute în vedere aspecte precum: <ul style="list-style-type: none"><li>- performanțele propuse (viteză de lucru, consum de energie, etc.);</li><li>- costul circuitelor utilizate;</li><li>- seria în care se va produce circuitul, etc.</li></ul>																																																																																	
4	<b>Simplificarea funcției logice.</b>  Această etapă nu este strict necesară pentru	Avem de implementat 4 funcții binare ce depind de aceleași variabile de intrare.  ♦ <b>Ieșirea <math>y_0</math>.</b> Din analiza tabelului de adevăr se observă că avem $y_0 = \overline{x_0}$ . Demonstratia acestei afirmații rămâne ca temă.																																																																																	

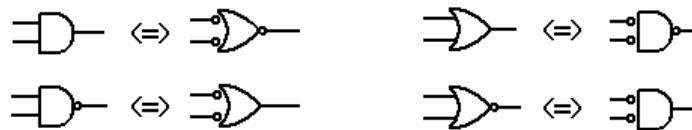
	<p>toate modalitățile de implementare a funcțiilor logice.</p> <p>Pentru implementarea cu porți, această etapă este necesară deoarece reduce numărul de circuite necesare.</p> <p>Proprietățile algebrei binare trebuie aplicate astfel încât să obține expresii finale cât mai simple.</p>	<p>♦ <b>Ieșirea <math>y_1</math>.</b> Folosind prima formă canonică obținem:</p> $y_1 = \begin{vmatrix} \bar{x}_2 \bar{x}_1 \bar{x}_0 \\ \bar{x}_2 x_1 x_0 \\ x_2 \bar{x}_1 \bar{x}_0 \\ x_2 x_1 x_0 \end{vmatrix} = \begin{vmatrix} \bar{x}_2 & \bar{x}_1 \bar{x}_0 \\ \bar{x}_2 & x_1 x_0 \\ x_2 & \bar{x}_1 \bar{x}_0 \\ x_2 & x_1 x_0 \end{vmatrix} = \begin{vmatrix} \bar{x}_2 & \bar{x}_1 \bar{x}_0 \\ x_2 & x_1 x_0 \end{vmatrix} = \begin{vmatrix} \bar{x}_1 \bar{x}_0 \\ x_1 x_0 \end{vmatrix}$ <p>♦ <b>Ieșirea <math>y_2</math>.</b> Folosind prima formă canonică obținem:</p> $y_2 = \begin{vmatrix} \bar{x}_2 \bar{x}_1 x_0 \\ \bar{x}_2 x_1 \bar{x}_0 \\ \bar{x}_2 x_1 x_0 \\ x_2 \bar{x}_1 \bar{x}_0 \end{vmatrix} = \begin{vmatrix} \bar{x}_2 & \bar{x}_1 x_0 \\ \bar{x}_2 & x_1 \bar{x}_0 \\ \bar{x}_2 & x_1 x_0 \\ x_2 & \bar{x}_1 \bar{x}_0 \end{vmatrix} = \begin{vmatrix} \bar{x}_2 & x_0 \\ \bar{x}_2 & x_1 \\ x_2 & \bar{x}_1 \bar{x}_0 \end{vmatrix} = \begin{vmatrix} \bar{x}_2 \bar{P} \\ x_2 P \end{vmatrix}; \text{ unde } P = \bar{x}_1 \bar{x}_0$ <p>♦ <b>Ieșirea <math>y_3</math>.</b> Folosind a prima formă canonică obținem:</p> $y_3 = \begin{vmatrix} x_2 \bar{x}_1 x_0 \\ x_2 x_1 \bar{x}_0 \\ x_2 x_1 x_0 \end{vmatrix} = x_2 \begin{vmatrix} \bar{x}_1 x_0 \\ x_1 \bar{x}_0 \\ x_1 x_0 \end{vmatrix} = x_2 \begin{vmatrix} x_0 \\ x_1 \\ x_0 \end{vmatrix}$ <p>Aplicând teoremele lui DeMorgan obținem:</p> $y_3 = x_2 \begin{vmatrix} x_1 \\ x_0 \end{vmatrix} = x_2 \begin{vmatrix} \bar{x}_1 \\ \bar{x}_0 \end{vmatrix} = \begin{vmatrix} x_2 \\ \bar{x}_1 \bar{x}_0 \end{vmatrix} = \begin{vmatrix} x_2 \\ P \end{vmatrix}$ <p><b>Observații:</b></p> <ul style="list-style-type: none"> <li>- Funcția <math>y_1</math>, se implementează cu o poartă sau exclusiv negat;</li> <li>- Produsul <math>P</math>, apare în expresia a două funcții binare, el va fi calculat o singură dată și apoi folosit în sinteza ambelor funcții;</li> </ul>
5	<p><b>Deducerea schemei logice.</b></p> <p><b>Atenție:</b> Există o mulțime de scheme logice pentru același tabel de adevăr! Schema logică depinde foarte mult de modul în care s-au aplicat proprietățile algebrei binare în procesul de simplificarea expresiei algebrice.</p>	<p>Schema logică rezultă din relațiile obținute la punctul anterior.</p>  <p>Se observă că această implementare necesită 7 porți logice dar acestea se găsesc în 4 circuite integrate diferite.</p>
6	<p><b>Implementarea hardware.</b></p> <p>În această etapă, după ce s-a stabilit familia de circuite logice, se identifică codurile circuitelor ce conțin porțile ce ne interesează, apoi se face alocarea pinilor.</p> <p>După terminarea acestei etape avem toate datele necesare pentru a trece la realizarea sa practică.</p>	<p>Alegem familia 74LS***.</p> <p>Consultând un catalog de firmă găsim următoarele coduri:</p> <ul style="list-style-type: none"> <li>- pentru inversoare: 74LS04;</li> <li>- pentru poarta AND cu două intrări: 74LS08;</li> <li>- pentru poarta SAU exclusiv negat: 74LS266;</li> <li>- pentru poarta NOR cu două intrări: 74LS02;</li> </ul>  <p>Se observă că am avut nevoie de 4 circuite integrate însă gradul lor de utilizare este următorul: 1/2 din 74LS04, 1/2 din 74LS266, 1/4 din 74LS08 și 1/4 din 74LS02.</p>

### 2.1.2. Logica de același tip

Așa după cum se știe, sistemele de operatori: (AND, NOT), (OR, NOT), (NOR), (NAND) sunt sisteme complete de operatori - ceea ce înseamnă că pot fi utilizate în descrierea oricărei funcții binare. Interes practic mai mare prezintă ultimele două deoarece ne indică posibilitatea de a implementa orice funcție logică cu ajutorul unui singur tip de poartă logică, fie NAND fie NOR.

**Metoda grafică.** Transpunerea unei scheme logice oarecare într-o logică de același tip se face, utilizând în mod convenabil, următoarele reguli:

- **R1:** La ieșirea unui etaj logic se poate introduce un cerculeț de negare dacă, corespunzător, la intrarea nivelului logic următor se face același lucru ("dubla negare este adevăr").
- **R2:** Introducerea la o ieșire a unui cerculeț de negare trebuie urmată de adăugarea unui inversor. Introducerea la o intrare a unui cerculeț de negație trebuie precedată de negarea respectivei variabile de intrare.
- **R3:** Trecerea de la o poartă la alta se face conform echivalențelor de mai jos (deduse din teoremele lui DeMorgan):



În funcție de schema logică inițială rezultatul transpunerii în logică de același tip ne poate conduce spre scheme mai simple sau, dimpotrivă, mai complicate decât schema inițială. Câteva transpuneri în logică de același tip sunt exemplificate în figura 1.

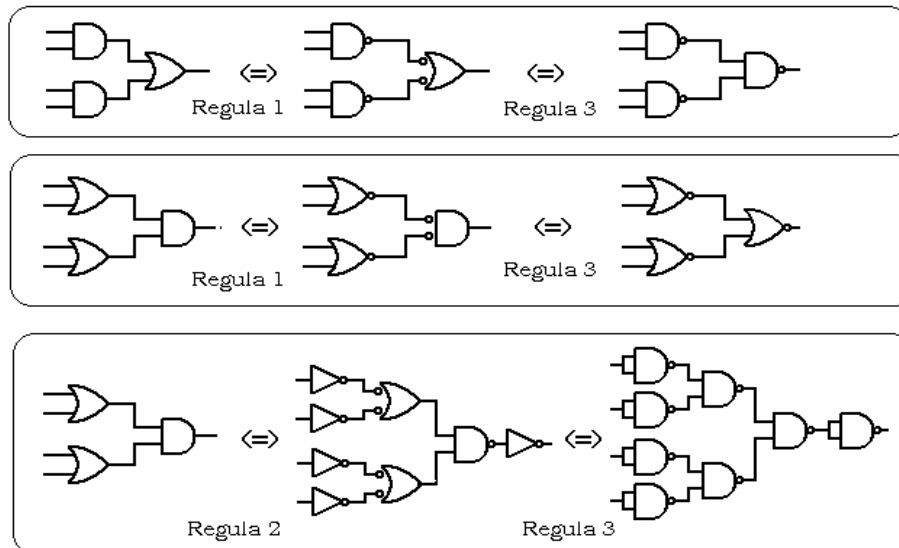


Fig. 1. Transpunerea unor scheme logice în logică de același tip

Pentru situațiile în care logica de același tip nu se impune în mod strict, aplicarea regulilor R1, R2, R3, pe porțiuni convenabile ale schemei, poate să determine reducerea necesarului de porți al schemei. De exemplu, aplicarea regulii R3 pentru funcția  $y_2$ , are ca efect reducerea necesarului de porți de la 4 la 2 și, în plus, implementarea se face folosind două tipuri de porți față de trei cât era inițial. Această situație este prezentată în figura 2.

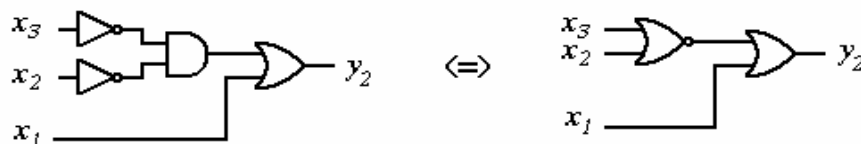
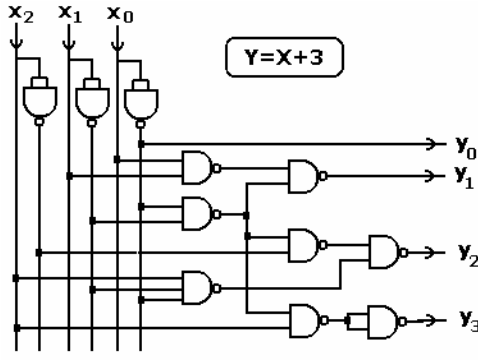


Fig. 2. Reducerea numărului de porți folosind echivalența dintre acestea

**Metoda analitică.** Această metodă presupune aplicarea repetată și în mod convenabil a teoremelor lui DeMorgan pentru o expresie inițială a funcției logice. Dacă se dorește implementarea în logică de tip NAND-NAND, se urmărește transformarea operațiilor de adunare în operații de înmulțire, iar dacă se dorește implementarea în logică de tip NOR-NOR, se urmărește transformarea operațiilor de înmulțire în operații de adunare.

În tabelul 2, se prezintă modul de trecere la logica de tip NAND-NAND a circuitului logic prezentat în tabelul 1.



Trecerea la logica de tip NAND-NAND	Schema logică/Observații
<p>Se pornește de la expresiile reduse obținute în tabelul 1.</p> $y_0 = \bar{x}_0$ $y_1 = \frac{x_1 \bar{x}_0}{x_1 x_0} = \frac{\bar{x}_1 \bar{x}_0}{x_1 x_0}$ $y_2 = \frac{\bar{x}_2 x_0}{x_2 \bar{x}_1 \bar{x}_0} = \frac{\bar{x}_2 \bar{x}_1 \bar{x}_0}{x_2 \bar{x}_1 \bar{x}_0} = \frac{\bar{x}_2 \bar{x}_1 \bar{x}_0}{x_2 \bar{x}_1 \bar{x}_0}$ $y_3 = x_2 \frac{x_1}{x_0} = x_2 \frac{x_1}{x_0} = x_2 \frac{\bar{x}_1 \bar{x}_0}{x_0} = x_2 \bar{x}_1 \bar{x}_0$	 <p>• Această realizare necesită 10 porți NAND cu 2 intrări și o poartă NAND cu 3 intrări. Așadar, avem nevoie de 4 circuite: 3 circuite 74LS00 și un circuit de tip 74LS10 .</p>

## 2. 2. Sinteza funcțiilor binare folosind structuri logice programabile de tip CPLD

CPLD-urile sunt structuri logice programabile, mai corect spus reconfigurabile, care într-o singură capsulă înglobează o mulțime de resurse hardware digitale. Spre exemplu, circuitul din machetele de laborator, XC95108, conține echivalentul a cca. 2000 de porți convenționale. Există însă structuri programabile mult mai complexe.

Realizarea de aplicații folosind astfel de circuite nu se mai face manual, fiecare firmă producătoare de circuite reconfigurabile asigură și un mediu software integrat care ușurează foarte mult dezvoltarea aplicațiilor. De regulă, aceste medii software acceptă mai multe modalități de descriere a funcționării circuitului (sistemului) logic. Cele mai întâlnite modalități de descriere sunt: folosirea de scheme logice; folosirea de fișiere VHDL; folosirea de diagrame de tranziție a stărilor; etc.

În desfășurarea acestei lucrări se folosește mediul de dezvoltare ISE - WebPack, produs de firma Xilinx, iar ca metode de descriere a funcționării circuitului folosim fie schemele logice, fie limbajul de descriere hardware VHDL.

### 2.2.1. Descrierea circuitelor combinaționale cu ajutorul editorului de scheme

Pentru a realiza o aplicație în mediul **ISE-WebPack** pe baza editorului de scheme, este necesar să parcurgem următoarele etape:

- **Deschiderea unui nou proiect.** Pentru aceasta este necesară parcurgerea următoarei secvențe de acțiuni: se alege **File** → **New Project** → în fereastra apărută se specifică **numele** proiectului (spre exemplu **circuit\_sumare**), **locul** unde acesta va fi salvat și proprietățile proiectului → **OK**;
- **Adăugarea unui fișier sursă conținând schema logică** (fișier de tip **.sch**). Din fereastra **Project Navigator** se alege **Project** → **New Source** → în fereastra apărută se alege **Schematic**, se specifică denumirea noului fișier (spre exemplu **abc**) → **Next** → **Finish**. După aceste comenzi, în fereastra **Source in Project** se va adăuga fișierul **abc.sch**, apoi, în mod automat se lansează în execuție editorul de scheme **Xilinx ECS**.
- **Desenarea schemei logice.** Deoarece modul de editare a schemei logice este similar oricărui editor de scheme electrice, nu vom insista asupra acestui aspect. În final trebuie avut grijă ca schema logică să fie salvată.
- **Adăugarea fișierului de constrângeri** (fișier de tip **.ucf**). Acest fișier este foarte important deoarece prin intermediul sau sunt precizați pinii circuitului CPLD unde vor fi conectate intrările și ieșirile proiectat de noi. Pentru adăugarea acestui fișier se procedează astfel: din fereastra **Project Navigator** se alege **Project** → **New Source** → în fereastra apărută se alege **Implementation Constrain File**, se specifică denumirea noului fișier (spre exemplu **cons\_sumator**) → **Next** → în fereastra apărută se alege fișierul sursă căruia i se asociază acest fișier de constrângeri (în cazul de față avem o singură sursă **abc.sch**) → **Next** → **Finish**. În urma acestor operații în fereastra surselor implicate în proiect (**Source in Project**) apare o nouă componentă **cons\_sumator.ucf**.

Dacă se face dublu clic pe **abc.ucf** se lansează un utilitar care ne ajută să edităm fișierul de constrângeri. Din fereastra apărută se selectează tabul **Ports**. După această alegere, pe ecran apare lista porturilor de intrare și de ieșire ale sistemului digital proiectat de noi. Pentru fiecare port trebuie să specificăm pinul CPLD-ului unde dorim conectarea respectivului port, aceasta presupune completarea coloanei denumită **Location**. După completare se face o salvare a fișierului de constrângeri.

Deoarece fișierul de constrângeri este de tip text, există și posibilitatea editării manuale a acestui fișier prin următoarea secvență de comenzi: din fereastra **Source in Project** se alege fișierul ce descrie funcționarea circuitului proiectat de noi (în cazul de față **abc.sch**) → din fereastra **Processes for Current Source** se face dublu clic pe opțiunea **Edit Constrains**. După aceste comenzi apare o fereastră în care se introduc constrângerile, cu respectarea sintaxei.

Trebuie avut grijă ca numărul pinului completat în coloana **Location** să corespundă cu schema hardware a machetei de laborator. Pentru acesta este absolut obligatoriu să consultați tabelul de conexiuni prezentat în anexe.

- **Implementarea circuitului în CPLD.** După introducerea surselor în proiect (etapă denumită *Design Entry*), urmează câteva etape ce se desfășoară în mod automat. Ultima etapă constă în generarea fișierelor de configurare a circuitului CPLD, etapă denumită incorect de generare a fișierelor de programare (*Generate Programming Files*). Pentru generarea fișierului de configurare, în fereastra resurselor se alege fișierul principal (**abc.sch**) iar în fereastra proceselor disponibile pentru acest fișier se face dublu clic pe opțiunea **Configure Device (iMPACT)**. În ferestrele următoare se aleg următoarele comenzi: **Configure Devices → Next → Boundary Scan Mode → Next → Automatically connect to cable. → Finish → OK** → se alege fișierul **abc.jed** → **Open** → clic dreapta pe icoana circuitului XC95108 → **Program...** → se alege opțiunea **Erase Before Programming** → **OK**. Dacă totul este în regulă, după câteva secunde va apare mesajul **Programming Succeeded** și se poate trece la verificarea funcționării circuitului.

## 2.2.2. Descrierea circuitelor combinaționale cu ajutorul limbajului VHDL

Limbajul VHDL a fost conceput în vederea simulării și sintezei cât mai ușoare a unui circuit/sistem logic. La ora actuală, limbajul VHDL este considerat un limbaj suficient de puternic pentru descrierea unor sisteme digitale complexe dar și suficient de ușor de asimilat pentru cei care sunt familiarizați cu un program de nivel înalt precum Pascal sau C.

Proiectarea și sintetizarea unui sistem digital în limbajul VHDL presupune deschiderea unui proiect în care sunt implicate mai multe categorii de fișiere sursă. Mediul ISE-WebPack are posibilitatea de a sintetiza scheme logice pornind de la fișiere sursă de tip VHDL. Modul de lucru este similar cu cel prezentat în secțiunea anterioară, cu singura deosebire că atunci când se introduce o nouă sursă în proiect, în loc de opțiunea **Schematic** se alege **VHDL Module**.

Descrierea unui circuit logic cu ajutorul VHDL presupune utilizarea unei entități (**entity**) la care trebuie asociată în mod obligatoriu cel puțin o arhitectură (**architecture**).

**Entitatea** este folosită în principal pentru a specifica conexiunile exterioare, denumite porturi, prin specificarea numelui, a direcției datelor, a tipului de date vehiculate, etc.

**Arhitectura** este folosită pentru descrierea comportamentului intern al circuitului ce urmează a fi proiectat, cu alte cuvinte conține o descriere a funcționării circuitului sau schema logică a acestuia.

Înainte de a trece la prezentarea exemplelor facem precizarea că limbajul VHDL permite descrierea circuitelor logice în mai multe moduri:

- **descriere structurală** - proiectantul impune schema logică iar sarcina mediului software este de a "amplasa" corect această schemă în resursele interne ale circuitului în care se dezvoltă aplicația;
- **descriere comportamentală** - proiectantul face o descriere de nivel înalt a circuitului/sistemului urmând ca sinteza propriu-zisă a schemei logice să rămână în sarcina mediului software.
- **metode combinate** - proiectantul poate să opteze pentru descriere structurală pentru anumite blocuri funcționale și descriere comportamentală pentru altele.

O sursă VHDL este un fișier text în care, în secțiunea de descriere a funcționării circuitului (în arhitectură), găsim o listă formată din **declarații concurente** și **declarații secvențiale**. Ajunși aici trebuie să precizăm că ceea ce în programare se cheamă *instrucțiune*, în descrierea hardware a circuitelor poate denumi de *declarație* (statement). Aceasta nu este singura diferență între programare și descrierea hardware, deosebirile de substanță, după cum vom vedea imediat, sunt date de modul în care sunt executate cele două tipuri de declarații.

Toate declarațiile concurente sunt executate simultan. Acest lucru se explică prin faptul că, în urma sintezei, fiecare declarație concurentă este implementată de un circuit hardware separat (implementare paralelă). Din acest motiv ordinea în care aceste declarații apar în listingul arhitecturi nu are nici o importanță.

Declarațiile secvențiale trebuie executate una după alta, într-o ordine bine stabilită, și numai după ce anumite condiții de declanșare au fost îndeplinite. În limbajul VHDL, pentru a separa cele două tipuri de declarații se folosește noțiunea de proces (**process**). În interiorul unui proces sunt incluse numai operațiile secvențiale ce trebuiesc executate atunci când anumite condiții de declanșare sunt îndeplinite. Condițiile de declanșare a procesului, altfel spus de declanșare a unei serii de acțiuni cu succesiune precisă, sunt specificate într-o așa zisă listă de sensivitate a procesului. Așadar, în interiorul procesului, ordinea declarațiilor este foarte importantă, ea indică succesiunea în care operațiile se execută.

Din punct de vedere al execuției, fiecare proces are același statut cu o declarație concurentă, cu alte cuvinte, toate procesele și toate declarațiile concurente se execută în același timp, deci sunt implementate în hardware prin scheme distincte.

În final precizăm că, în mod curent, procesele sunt folosite pentru implementarea circuitelor secvențiale și uneori pentru circuite combinaționale iar declarațiile concurente doar pentru circuite combinaționale.

Modul de utilizare al celor două tipuri de declarații precum și multitudinea de posibilități puse la dispoziție de limbajul VHDL se poate urmări în exemplele ce urmează, exemple ce au ca scop implementarea unei funcții binare de trei variabile având tabelul de adevăr prezentat în tabelul 3.

Tabelul 3

c	b	a	yout
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

◆ **Exemplul 1:** Descrierea funcției logice **yout** pornind de la tabelul de adevăr

Observații	Codul VHDL
<b>Secțiune dedicată includerii de biblioteci</b>	<pre>library IEEE; use IEEE.std_logic_1164.all;</pre>
<b>Secțiune dedicată descrierii entității.</b> Aici se declară intrările și ieșirile circuitului precum și tipul de date vehiculate prin acestea. În cazul de față: - numele entității este: <b>clc1</b> - avem 3 intrări notate <b>a, b, c</b> și o ieșire notată <b>yout</b> ; - toate intrările și ieșirile sunt de tip <i>std_logic</i> , deci pot avea următoarele stări: H, L, X, HiZ	<pre>entity clc1 is port (a, b, c : in std_logic;       you : out std_logic); end clc1;</pre>
<b>Secțiune dedicată descrierii arhitecturii.</b> Aici se precizează efectiv schema sau funcționarea entității declarate anterior. În cazul de față: - numele arhitecturii este: <b>arh_1</b> ; - asocierea arhitecturii se face cu entitatea <b>clc1</b> ; - expresia algebrică a funcției logice este: $yout = \bar{c}\bar{b}a + c\bar{b}\bar{a} + cb\bar{a}$	<pre>architecture arh_1 of clc1 is begin -- o singura declaratie concurenta you &lt;= ((not c) and (not b) and a) or       (c and (not b) and (not a)) or       (c and b and (not a)); end arh_1;</pre>

**Exemplul 2:** O altă modalitate de descriere a funcției logice **yout**

Observații	Codul VHDL
<b>Secțiune dedicată includerii de librării</b>	<pre>library IEEE; use IEEE.std_logic_1164.all;</pre>
<b>Secțiune dedicată descrierii entității.</b> (Secțiune identică cu cea din exemplul anterior).	<pre>entity clc1 is port (a, b, c : in std_logic;       you : out std_logic); end clc1;</pre>
<b>Secțiune dedicată descrierii arhitecturii.</b> În această secțiune apar câteva modificări: - numele arhitecturii este: <b>arh_2</b> ; - sunt declarate 3 semnale: <b>s1, s2, s3</b> - sunt folosite 4 declarații concurente; Semnalul din limbajul VHDL are ca echivalent fizic un fir de legătură. Semnalele trebuie declarate între <b>architecture</b> și <b>begin</b> . Starea logică a ieșirii <b>yout</b> este reactualizată de fiecare dată când apare o modificare la oricare din semnalele <b>s1, s2, s3</b> . Semnalele declarate într-o arhitectură sunt recunoscute numai în interiorul acesteia.	<pre>architecture arh_2 of clc1 is signal s1, s2, s3 : std_logic; begin -- mai multe declaratii concurente s1 &lt;= (not c) and (not b) and a; s2 &lt;= c and (not b) and (not a); s3 &lt;= c and b and (not a); you &lt;= s1 or s2 or s3; end arh_2;</pre>

**Exemplul 3:** Utilizarea unui proces pentru descrierea funcției **yout**

Observații	Codul VHDL
<b>Secțiune dedicată includerii de librării</b>	<pre>library IEEE; use IEEE.std_logic_1164.all;</pre>
<b>Secțiune dedicată descrierii entității.</b> (Secțiune identică cu cea din exemplul anterior).	<pre>entity clc1 is port (a, b, c : in std_logic;       you : out std_logic); end clc1;</pre>
<b>Secțiune dedicată descrierii arhitecturii.</b> - În descriere se folosește un singur proces denumit <b>p1</b> . Atribuirea unui nume pentru proces este opțională. - Activarea procesului (altfel spus, lansarea în execuție) se face pentru orice eveniment apărut pe intrările <b>a, b, c</b> . - Se recomandă ca în proces să existe declarații pentru ambele valori logice ale ieșirii. - Declarațiile de tip <b>if – then</b> nu pot fi folosite decât în interiorul unui proces. - În lista de sensibilități a procesului trebuie incluse toate semnalele ce apar în partea dreaptă a declarațiilor din interiorul procesului. - Este necesar ca rezultatul evaluării condiției de test din paranteza lui <b>if</b> să fie TRUE sau FALSE, de aceea se folosește <b>c='0' and b='1'</b> și nu <b>not c and b</b> .	<pre>architecture arh_3 of clc1 is begin p1 : process (a, b, c) begin you &lt;= '0'; if ((c='0') and (b='0') and (a='1')) then you &lt;= '1'; end if; if ((c='1') and (b='0') and (a='0')) then you &lt;= '1'; end if; if ((c='1') and (b='1') and (a='0')) then you &lt;= '1'; end if; end process p1; end arh_3;</pre>
<b>Descrierea arhitecturii - variantă alternativă.</b> O altă posibilitate de descriere a arhitecturii funcției binare <b>yout</b> se	<pre>architecture arh_3_bis of clc1 is begin</pre>

<p>bazează pe utilizarea unei declarații de tip <b><i>if – then – else</i></b>.</p>	<pre> p2 : process (a, b, c) begin   if (     (( c='0') and ( b='0') and (a='1')) or     (( c='1') and ( b='0') and (a='0')) or     (( c='1') and ( b='1') and (a='0'))   ) then     yout &lt;= '1' ;   else     yout &lt;= '0' ;   end if; end process p2; end arh 3 bis; </pre>
---	---

**Exemplul 4:** Utilizarea declaratiei de tip **when – else** pentru descrierea functiei logice **yout**

<b>Observații</b>	<b>Codul VHDL</b>
<b>Secțiune dedicată includerii de librării</b>	<b>library IEEE;</b> <b>use IEEE.std_logic_1164.all;</b>
<b>Secțiune dedicată descrierii entității.</b>  (Secțiune identică cu cea din exemplul anterior).	<b>entity clc1 is</b> <b>port</b> (a, b, c : <b>in</b> std_logic; yout : <b>out</b> std_logic); <b>end clc1;</b>
<b>Secțiune dedicată descrierii arhitecturii.</b> - numele arhitecturii este: <b>my_clc4</b> ; - utilizarea declarației <b>when-else</b> trebuie făcută astfel încât să apară ambele valori logice ale ieșirii; - Este necesar ca rezultatul evaluării condiției de test din paranteza lui <b>when</b> să fie TRUE sau FALSE, de aceea se folosește <b>c='0' and b='1'</b> și nu <b>not c and b</b> .	<b>architecture my_clc4 of clc1 is</b> <b>begin</b> yout <= '1' <b>when</b> ( (c='0' and b='0' and c='1') or (c='1' and b='0' and c='0') or (c='1' and b='1' and c='0') ) <b>else</b> '0' ; <b>end my_clc4;</b>

### Exemplul 5: Utilizarea vectorilor în descrierea funcției logice **yout**

În limbajul VHDL, prin declarația **vector** se introduc magistralele, fie de intrare fie de ieșire. Spre exemplu, prin declarația `d_out : out std_logic_vector (3 downto 0);` se descrie o magistrală de ieșire de 4 biți în care `d_out(3)` este cel mai semnificativ bit. După cum se va vedea în acest exemplu, utilizarea vectorilor simplifică destul de mult descrierea circuitelor digitale.

Există mai multe modalități de atribuire a stării logice a unei magistrale de date:

```
d_out <= "1100 ";           -- atribuirea tuturor biților din magistrală
d_out(3) <= `1 `;          -- atribuirea unui singur bit din magistrală (bitul D3)
d_out(2 downto 1) <= "1100 "; -- atribuirea unei porțiuni din magistrală (biții D2 și D1)
```

Pentru utilizarea vectorilor în implementarea funcției **youst** este necesar să redenumim intrările circuitului combinational astfel încât acestea să aparțină unei magistrale de intrare de 3 biti.

<b>Observații</b>	<b>Codul VHDL</b>
<b>Secțiune dedicată includerii de librării</b>	<pre>library IEEE; use IEEE.std_logic_1164.all;</pre>
<b>Secțiune dedicată descrierii entității.</b> În acest exemplu se folosește un vector de intrare format din 3 biți, <b><i>d_in</i></b> , prin care sunt introduse cele 3 intrări ale circuitului logic. Am folosit următoarea asociere : - <b><i>d_in(2)</i></b> pentru intrarea <b><i>a</i></b> ; <b><i>d_in(1)</i></b> pentru intrarea <b><i>b</i></b> ; - <b><i>d_in(0)</i></b> pentru intrarea <b><i>c</i></b> ;	<pre>entity clc1 is port (     d_in:in std_logic_vector(2 downto 0);     yout : out std_logic); end clc1;</pre>
<b>Secțiune dedicată descrierii arhitecturii.</b>  (varianta 1)	<pre>architecture my_clc5 of clc1 is begin yout &lt;= '1' when (     (d_in = "001") or (d_in = "100") or     (d_in = "100" ) )     else '0' ; end my_clc5;</pre>
<b>Secțiune dedicată descrierii arhitecturii.</b>  (varianta 2)	<pre>architecture my_clc5_bis of clc1 is begin p3 : process (d_in ) begin yout &lt;= '0' ; if ((d_in = "001" ) or(d_in = "100")or(d_in = "100" ) ) then     yout &lt;= '1' ; else     yout &lt;= '0' ; end if;</pre>

	<pre>end process p3; end my_clc5_bis;</pre>
--	---

♦ **Exemplul 6:** Descrierea comportamentală a unui circuit logic.

Una dintre cele mai puternice facilități oferite de VHDL o constituie descrierea comportamentală a unui circuit logic. Această facilitate permite proiectantului să dezvolte sisteme logice foarte complexe fără a fi preocupat de schema logică detaliată, ci doar de descrierea comportamentală de nivel înalt a respectivului circuit.

Spre exemplu, un circuit de adunare a două numere reprezentate pe 3 biți fiecare, se poate descrie printr-o singură declarație de forma  $z \leq x + y$ . Sarcina sintetizării scheme logice a respectivului sumator cade în sarcina mediului software în care se dezvoltă aplicația.

Observații	Codul VHDL
<b>Secțiune dedicată includerii de librării</b>	<pre>-- Exemplu de CLC descris comportamental library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_unsigned.all; use IEEE.std_logic_arith.all;</pre>
<b>Secțiune dedicată descrierii entităților.</b> În cazul de față: - nume entitate este: <b>sum</b> - avem 2 magistrale de intrare de câte 3 biți; - o ieșire <b>z</b> pe 5 biți; - toate intrările și ieșirile sunt de tip <i>std_logic</i> .	<pre>entity sum is port(     x,y:in std_logic_vector(2 downto 0);     z:out std_logic_vector(4 downto 0)); end sum;</pre>
<b>Secțiune dedicată descrierii arhitecturii.</b>	<pre>architecture arh_sum of sum is begin     z &lt;= x + y; end arh_sum;</pre>



### 3. Desfășurarea lucrării

#### 3.1. Implementarea funcțiilor logice folosind rețele de porți (Metoda clasică)

- A. Pentru circuitul logic cu o intrare și o ieșire având tabelul de adevăr prezentat în tabelul 3 se cere:
  - implementarea funcției **yout** folosind forma canonică disjunctivă și un număr cât mai redus de porți;
  - implementarea funcției **yout** folosind forma canonică conjunctivă și un număr cât mai redus de porți;
  - implementarea funcției **yout** folosind doar porți NAND;
  - implementarea funcției **yout** folosind doar porți NOR;
- B. Pentru circuitul logic cu 3 intrări și 4 ieșiri ce realizează funcția  $Y=X+3$  (vezi tabelul 1 din această lucrare) se cere:
  - implementarea funcțiilor logice folosind cealaltă formă canonică;
  - implementarea funcțiilor logice folosind doar porți NAND cu două intrări (vezi tabelul 2);
  - implementarea funcțiilor logice folosind doar porți NOR;

#### 3.2. Utilizarea ISE WebPack pentru descrierea aplicațiilor sub formă de scheme logice

- A. Implementarea și verificarea pe macheta de laborator cu CPLD a schemei logice din tabelul 1.
- B. Implementarea și verificarea pe macheta de laborator cu CPLD a schemei logice din tabelul 2.
- C. Implementarea și verificarea pe macheta de laborator cu CPLD a schemei logice din figura 3.

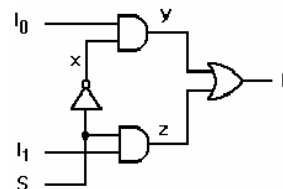


Fig.3

**Mod de lucru:** -vezi secțiunea 4 (Indicații privind modul de lucru).

#### 3.3. Implementarea CLC-urilor folosind limbajul VHDL

##### Partea I: Descrierea structurală (pornind de la o schemă logică impusă)

- A. Pentru schema logică din tabelul 1 se cere:
  - O descriere în limbaj VHDL folosind modul de lucru din exemplu 1.
  - O descriere în limbaj VHDL folosind modul de lucru din exemplu 2.
 Pentru fiecare caz în parte realizați o implementare pe macheta de laborator cu CPLD și verificați dacă circuitul implementat funcționează corect.

**B.** Pentru schema logică din tabelul 2 se cere:

- O descriere în limbaj VHDL folosind modul de lucru din exemplu 1.
- O descriere în limbaj VHDL folosind modul de lucru din exemplu 2.

Pentru fiecare caz în parte realizați o implementare pe macheta de laborator cu CPLD și verificați dacă circuitul implementat funcționează corect.

**Mod de lucru:** -vezi secțiunea 4 (Indicații privind modul de lucru).

#### **Partea a II-a: Descrierea structurală (pornind de la tabelul de adevăr)**

**C.** Pornind de la tabelul de adevăr al clc-ului din tabelul 1, se cere:

- O descriere în limbaj VHDL folosind modul de lucru din exemplu 3.
- O descriere în limbaj VHDL folosind modul de lucru din exemplu 4.
- O descriere în limbaj VHDL folosind modul de lucru din exemplu 5.

Pentru fiecare caz în parte realizați o implementare pe macheta de laborator cu CPLD și verificați dacă circuitul implementat funcționează corect.

**D.** Pentru un circuit logic combinațional cu 5 intrări și o ieșire, ce realizează funcția de vot majoritar (ieșirea este în "1", atunci când la intrări predomină valoarea logică "1") se cere:

- tabelul de adevăr redus;
- programul VHDL folosind modul de lucru din exemplul 1;
- verificarea programului pe macheta de laborator;

**Mod de lucru:** -vezi secțiunea 4 (Indicații privind modul de lucru).

#### **Partea a III-a: Descrierea comportamentală**

**E.** Folosind descrierea comportamentală (similară exemplului 6), se cere descrierea și verificarea pe macheta de laborator a unui circuit de sumare a două numere exprimate pe 4 biți.

**F.** Folosind descrierea comportamentală (similară exemplului 6), se cere descrierea și verificarea pe macheta de laborator a unui circuit de comparare a două numere exprimate pe 4 biți.

**Mod de lucru:** -vezi secțiunea 4 (Indicații privind modul de lucru).

### **4. Indicații privind modul de lucru**

Pentru fiecare aplicație este necesară deschiderea unui nou proiect după metodologia prezentată în secțiunea 2.2.2. a prezentei lucrări de laborator.

Toate aplicațiile din această lucrare necesită doar un singur fișier sursă (fie schemă logică, fie sursă VHDL) și un singur fișier de constrângeri.

Referitor la conectarea intrărilor și a ieșirilor din circuit facem următoarele precizări:

- Variabilele de intrare se vor conecta la switch-urile (comutatoare cu două poziții) de pe macheta de laborator. În acest mod, trecerea comutatorului de pe o poziție pe alalta echivalează cu schimbarea stării logice a variabilei de intrare. Seschiderea.
- Variabilele de ieșire se vor conecta la LED-urile de pe macheta de laborator. În acest mod, în momentul în care o variabilă de ieșire este în unu logic, LED-ul asociat luminează.
- În cazul aplicațiilor de sumatoare sau comparatoare binare, fiecare număr de intrare respectiv ieșire va fi reprezentat printr-un vector. Un vector de intrare trebuie conectat la un pachet de switch-uri (un număr corespunzător de switch-uri ce sunt amplasate unul lângă altul) iar vectorul de ieșire va fi reprezentat pe un pachet de LED-uri.

NET "x<0>" LOC = "P37";  
NET "x<1>" LOC = "P40";  
NET "x<2>" LOC = "P43";

NET "y<0>" LOC = "P54";  
NET "y<1>" LOC = "P52";  
NET "y<2>" LOC = "P50";

NET "z<0>" LOC = "P75";  
NET "z<1>" LOC = "P71";  
NET "z<2>" LOC = "P67";  
NET "z<3>" LOC = "P65";  
NET "z<4>" LOC = "P62";

Fișierul de constrângeri pentru exemplul 6 este prezentat alăturat. Pentru introducerea numărului x s-au folosit primele trei comutatoare (SW1, SW2, SW3), iar pentru y s-au folosit ultimele trei comutatoare (SW6, SW7, SW8). Afișarea rezultatului se face pe primele 5 LED-uri (LD1÷LD5).



## Lucrarea nr. 3: **Implementarea funcțiilor binare folosind DCD/DMUX**

### 1. Scopul lucrării

În prima parte a lucrării, după o scurtă prezentare teoretică referitoare la structura internă și la funcționarea circuitelor decodificatoare (DCD) respectiv demultiplexoare (DMUX), se prezintă o metodă de utilizare a acestor circuite în implementarea funcțiilor binare.

În partea a doua se arată modul în care aceste circuite pot fi descrise în limbaj VHDL în vederea implementării lor în structuri de tip CPLD sau FPGA.

### 2. Considerente teoretice

O posibilă clasificare a circuitelor integrate digitale, din punctul de vedere al gradului de integrare, este prezentată în tabelul 1.

Tabelul 1

Gradul de integrare	Numărul de porți	Circuite combinaționale (CLC)	Circuite secvențiale
MIC <b>SSI (Small Scale Integration)</b>	$\leq 12$	- porți logice elementare;	- bistabili; - latch-uri;
MEDIU <b>MSI (Medium Scale Integration)</b>	$12 \div 100$	- codificatoare, decodificatoare; - multiplexoare, demultiplexoare; - sumatoare; - comparatoare;	- registre; - numărătoare;
MARE <b>LSI (Large Scale Integration)</b>	$> 100$	- memorii fixe (ROM); - matrici logice programabile (PLA, PLD);	- registre mari; - memorii RAM;
FOARTE MARE <b>VLSI (Very Large Scale Integration)</b>	$> 1000$	- structuri programabile de tip CPLD; - structuri programabile de tip FPGA;	- memorii RAM;

În proiectarea sistemelor digitale moderne, datorită apariției circuitelor realizate în tehnologie VLSI, tot mai des se pune problema realizării de sisteme pe un singur chip (System on Chip - **SoC**). În acest context, studiarea circuitelor combinaționale de complexitate medie pare nejustificată. Această părere este greșită deoarece toate circuitele de complexitate medie se regăsesc ca părți integrante, sau ca blocuri funcționale, în structura circuitelor VLSI. Așadar, cunoașterea foarte exactă a tuturor facilităților oferite de aceste circuite, ne permite fie înțelegerea funcționării unui sistem complex realizat în tehnologie VLSI, fie proiectarea eficientă a unui astfel de sistem.

#### 2. 1. Decodificatorul (DCD)

Decodificatorul este un circuit combinațional prevăzut cu  $n$  intrări (denumite cel mai adesea **intrări de selecție**) și  $2^n$  ieșiri. Circuitul are proprietatea de a **recunoaște o combinație binară** aplicată pe intrările de selecție **prin activarea unei singure ieșiri**.

Numărul combinațiilor binare distincte (altfel spus, numărul de coduri) ce pot fi recunoscute de către un DCD este dependent de numărul intrărilor de selecție, iar numărul combinațiilor ce sunt semnalizate depinde de numărul de ieșiri disponibile.

În figura 1 se prezintă simbolul, schema logică de principiu și tabelul de adevăr pentru un decodor cu 8 ieșiri active pe zero logic. Așadar, recunoașterea unui cod se face prin trecerea în zero logic a unei singure ieșiri (cea asociată codului respectiv) iar restul ieșirilor se mențin în starea lor inactivă (în cazul de față unu logic). *Facem precizarea că există și circuite DCD care au ieșirile active pe unu logic, schema lor internă fiind similară celei din figura 1.*

Pentru a face distincție între intrările/ieșirile active pe unu și cele active pe zero, prin convenție internațională, intrările/ieșirile active pe zero sunt însoțite de un cerculeț în schema logică iar denumirea intrărilor/ieșirilor este însoțită de o bară similară celei de negație ( $\bar{Y}_{out}$ ).

Printre circuitele uzuale disponibile pe piață putem enumera:

- decodoare BCD/zecimal: 7442, 7445, 74141, 74145, aceste circuite prezintă patru intrări de selecție și numai 10 ieșiri (active în "zero logic") din cele 16 posibile;
- decodoare BCD/7segmente: 7446, 7447 circuite utilizate pentru comanda afișajelor numerice;
- 

#### 2. 2. Demultiplexorul (DMUX)

Acest circuit poate fi privit ca un DCD prevăzut cu o intrare suplimentară de validare a funcționării circuitului. În consecință un DMUX prezintă:  $n$  intrări de selecție, o intrare de validare și  $2^n$  ieșiri.

Intrarea de validare, denumită de regulă **Enable**, controlează funcționarea DMUX. Pentru cazul unui DMUX cu intrarea de validare activă pe zero logic, așa cum este cazul celui prezentat în figura 2, putem avea următoarele situații:

- dacă intrarea de validare este activată,  $\bar{E} = 0$ , funcționarea demultiplexorului este identică cu a unui DCD (în funcție de codul aplicat pe intrările de selecție se activează doar o singură ieșire);

- dacă intrarea de validare este neactivată,  $\bar{E} = 1$ , toate ieșirile sunt forțate în starea lor inactivă, indiferent de codul binar aplicat pe intrările de selecție.

La o primă vedere, se poate spune că DMUX este un DCD mai flexibil deoarece prin intermediul intrării de validare se poate controla momentele de timp în care DMUX are voie să funcționeze.

Principala funcție a unui DMUX este aceea de a distribui informația digitală prezentă pe intrarea de validare  $\bar{E}$ , spre una din liniile de ieșire. Este evident că selectarea liniei de ieșire se face prin intermediul intrărilor de selecție. **Deci, din punct de vedere funcțional, un DMUX poate fi asemănat cu un comutator rotativ.**

Pentru a demonstra această ultimă proprietate a unui DMUX, ce nu este evidentă la prima vedere, să considerăm că pe intrările de selecție ale circuitului din figura 2, aplicăm combinația binară  $BA=01$ , ceea ce înseamnă că ieșirea aleasă este  $\bar{1}$ . În funcție de stare logică a intrării  $\bar{E}$  pot exista două situații:

- dacă  $\bar{E} = 0$ , circuitul DMUX lucrează ca un DCD și, datorită codului aplicat intrărilor de selecție, se activează ieșirea  $\bar{1}$  (trece în starea "0") în timp ce restul ieșirilor rămân inactive (starea logică "1");
- dacă  $\bar{E} = 1$ , DMUX nu este validat, deci toate ieșirile sunt forțate în starea inactivă (starea logică "1").

Din analiza celor două situații se observă că starea logică a ieșirii selectate, în cazul de față  $\bar{1}$ , este identică cu starea logică a intrării de validare  $\bar{E}$ . Cu alte cuvinte putem afirma că **informația digitală prezentă pe intrarea  $\bar{E}$  este transmisă (directionată) spre o ieșire indicată de codul aplicat intrărilor de selecție.**

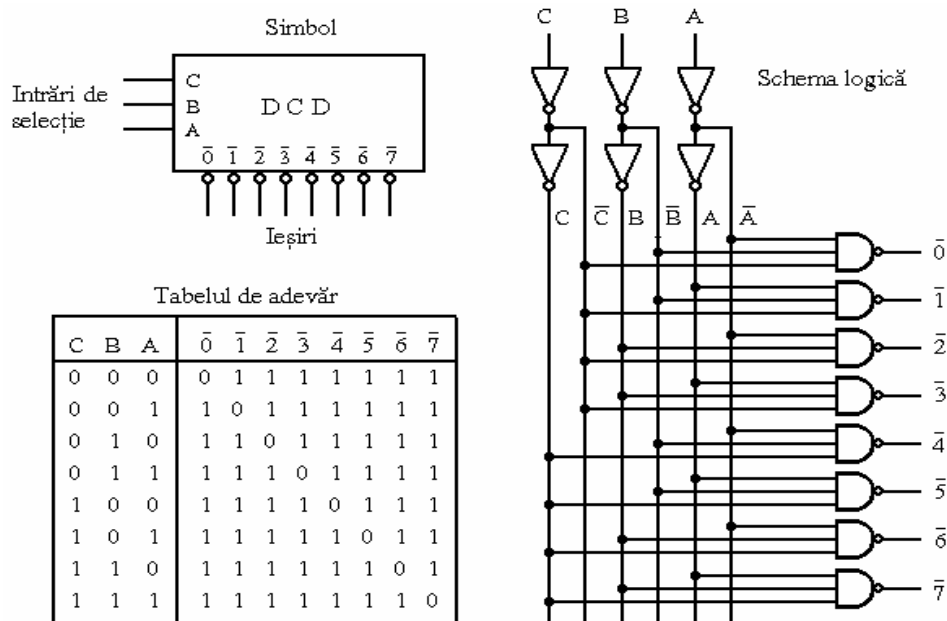


Fig.1. Schema logică, tabelul de adevăr și simbolul unui decodificator cu 8 ieșiri active pe zero logic

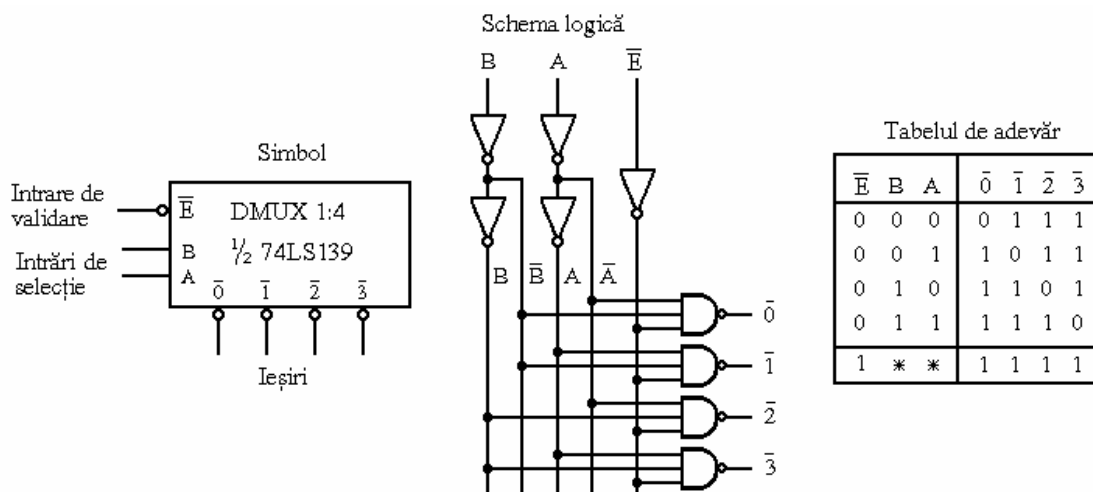


Fig.2. Schema logică, simbolul și tabelul de adevăr pentru DMUX 1:4 de tip 74LS139



### 2.3. Exemple de DCD/DMUX realizate în structuri integrate

În această secțiune sunt prezentate câteva circuite uzuale realizate în structuri integrate.

Observații	Simbol
<b>Circuitul 74LS138</b> <ul style="list-style-type: none"> <li>- circuitul prezintă: <ul style="list-style-type: none"> <li>- 8 ieșiri de date active pe zero logic;</li> <li>- 3 intrări de validare (două active pe zero logic iar a treia pe unu logic);</li> </ul> </li> <li>- pentru acest circuit activarea unei ieșiri se poate face numai dacă simultan sunt îndeplinite condițiile: <math>G1A=1</math>, <math>\overline{G2A}=0</math> și <math>\overline{G2B}=0</math>;</li> <li>- la intrarea A trebuie aplicat cel mai puțin semnificativ bit din codul de selecție;</li> <li>- o altă posibilitate de notare a intrărilor de selecție este următoarea: <math>A0=A</math>, <math>A1=B</math>, <math>A2=C</math>;</li> </ul>	
<b>Circuitul 74LS139 (2 x DMUX 1:4)</b> <ul style="list-style-type: none"> <li>- circuitul integrat conține 2 circuite DMUX 1:4 cu ieșiri active pe zero logic;</li> <li>- cele 2 circuite DMUX sunt complet separate;</li> <li>- pentru primul circuit DMUX avem: <ul style="list-style-type: none"> <li>- <math>\overline{1G}</math> intrarea de validare/intrarea de date, activă pe zero logic;</li> <li>- <math>\overline{1B}</math>, <math>\overline{1A}</math> intrările de selecție;</li> <li>- <math>\overline{1Y0}</math>, <math>\overline{1Y1}</math>, <math>\overline{1Y2}</math>, <math>\overline{1Y3}</math> ieșirile de date ale demultiplexorului, active pe zero logic;</li> </ul> </li> <li>- pentru al doilea circuit DMUX avem: <ul style="list-style-type: none"> <li>- <math>\overline{2G}</math> intrarea de validare/intrarea de date, activă pe zero logic;</li> <li>- <math>\overline{2B}</math>, <math>\overline{2A}</math> intrările de selecție;</li> <li>- <math>\overline{2Y0}</math>, <math>\overline{2Y1}</math>, <math>\overline{2Y2}</math>, <math>\overline{2Y3}</math> ieșirile de date ale demultiplexorului, active pe zero logic;</li> </ul> </li> </ul>	
<b>Circuitul 74LS442 (decodificator zecimal)</b> <ul style="list-style-type: none"> <li>- circuitul prezintă 4 intrări de selecție (A pentru cel mai puțin semnificativ bit iar D pentru cel mai semnificativ bit al codului de selecție);</li> <li>- din punct de vedere teoretic, cu 4 intrări de selecție avem posibilitatea de a comanda 16 ieșiri;</li> <li>- din punct de vedere practic, în circuitul 7442, sunt implementate doar primele 10 ieșiri;</li> <li>- pentru coduri de selecție mai mari de 1001, toate ieșirile sunt în starea lor inactivă (zero logic);</li> </ul>	

### 2.4. Utilizarea DCD în implementarea funcțiilor binare

Analizând schema logică a unui decodificator se observă că aceasta este organizată pe două nivele:

- un nivel de inversoare pentru calculul complementelor variabilelor de intrare;
- un nivel de NAND-uri (dacă ieșirile decodificatorului sunt active pe zero logic) sau un nivel de AND-uri (dacă ieșirile decodificatorului sunt active pe unu logic).

În ipoteza că pe intrările de selecție ale unui decodificator binar se aplică variabilele unei funcții binare, la ieșirile acestuia se vor regăsi fie mintermenii respectivei funcții (dacă DCD are ieșiri active pe unu logic) fie complementul mintermenilor (dacă DCD are ieșiri active pe zero logic).

Dacă variabilele funcției sunt conectate la intrările de selecție cu respectarea ponderilor, atunci numerotarea ieșirilor DCD-ului va fi identică cu numerotarea mintermenilor ( $m_0$  este disponibil la ieșirea 0,  $m_1$  este disponibil la ieșirea 1, ...).

Așadar, utilizarea DCD-ului este avantajoasă acolo unde este nevoie de implementarea mai multor funcții binare ce depind de aceleași variabile de intrare deoarece, mintermenii se calculează o singură dată și pot fi utilizați pentru fiecare funcție.

#### Regulile de conectare sunt următoarele:

- dacă în tabelul de adevăr funcția este definită cu mai puține stări de "1" decât stări de "0", atunci vom utiliza un NAND conectând la intrările lui mintermenii marcați cu "1";
- dacă în tabelul de adevăr funcția este definită cu mai puține stări de "0" decât stări de "1", atunci vom utiliza un AND conectând la intrările lui mintermenii marcați cu "0".

Pentru o înțelegere mai bună a modului de utilizare a unui DCD în implementarea funcțiilor binare, în cele ce urmează se exemplifică modul de conectare a ieșirilor DCD în funcție de tabelul de adevăr al funcției ce trebuie implementate. Ca exemplu se consideră funcția **F** dată prin tabelul de adevăr alăturat.

**Exemplu 1:** Se cere implementarea funcției folosind un DCD cu ieșiri active pe unu logic;

Deoarece DCD-ul are ieșirile active pe unu logic înseamnă că la ieșirile sale vom obține mintermenii funcției. Pentru rezolvarea acestei probleme vom scrie funcția **F** folosind formele canonice, după care, acestea vor fi prelucrate până la punerea în evidență a mintermenilor.

Prima formă canonică a funcției **F** are expresia:

$$F = \left[ \begin{array}{c} \bar{x}_2 x_1 \bar{x}_0 \\ x_2 \bar{x}_1 x_0 \end{array} \right] = \left[ \begin{array}{c} m_2 \\ m_5 \end{array} \right] \quad (1)$$

Relația (1) conduce la o schemă de implementare a funcției **F** ca cea din figura 3.a)

$x_2$	$x_1$	$x_0$	$F$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

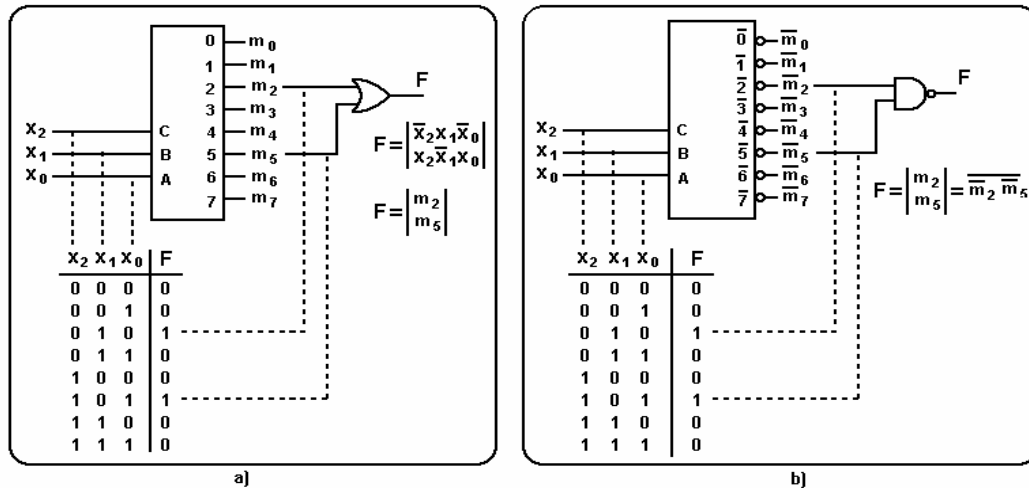


Fig. 3. Implementarea funcției **F** folosind circuite DCD

**Exemplu 2:** Se cere implementarea funcției folosind un DCD cu ieșiri active pe zero logic;

Deoarece DCD-ul are ieșirile active pe zero logic înseamnă că la ieșirile sale vom obține complementul mintermenii funcției. Pentru rezolvarea acestei probleme vom scrie funcția **F** folosind formele canonice, după care, acestea vor fi prelucrate până la punerea în evidență a complementelor mintermenilor.

Aplicând teoremele lui DeMorgan relația (1) se obține:

$$F = \left[ \begin{array}{c} \bar{x}_2 x_1 \bar{x}_0 \\ x_2 \bar{x}_1 x_0 \end{array} \right] = \left[ \begin{array}{c} m_2 \\ m_5 \end{array} \right] = \overline{\overline{m_2} \overline{m_5}} \quad (2)$$

Implementarea funcției după relația (2) este prezentată în figura 3.b).

## 2. 5. Utilizarea DMUX în implementarea funcțiilor binare

Este cunoscut faptul că un DMUX poate fi transformat ușor în DCD prin menținerea permanentă a intrării de validare în starea sa activă. Toate precizările făcute la utilizarea DCD în implementarea funcțiilor binare rămân valabile.

## 3. Utilizarea limbajului VHDL pentru descrierea DCD/DMUX

♦ **Exemplul 3:** Descrierea în limbaj VHDL a unui decodificator zecimal cu ieșiri active pe unu logic

Observații	Codul VHDL
Secțiune dedicată includerii de librării	<code>library IEEE; use IEEE.std_logic_1164.all;</code>
Secțiune dedicată descrierii entității.  <b>Obs.:</b> elementele unui vector pot fi declarate în ordine descrescătoare (cazul intrării <b>bcd</b> ) sau în ordine crescătoare (cazul ieșirii <b>zec</b> ).	<code>entity bcd_zec is port (   bcd : in std_logic_vector(3 downto 0);   zec : out std_logic_vector(0 to 9)); end bcd_zec;</code>
Secțiune dedicată descrierii arhitecturii.	<code>architecture dec_arh of bcd_zec is</code>

<p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- numele arhitecturii este: <b>dec_arh</b>;</li> <li>- arhitectura se asociază cu entitatea <b>bcd_zec</b>;</li> <li>- procesul <b>P1</b>, este sensibil la codul de intrare <b>bcd</b>;</li> <li>- descrierea este corectă deoarece se face o singură atribuire pentru ieșirea <b>zec</b>;</li> <li>- ieșirile sunt active pe unu logic;</li> </ul>	<pre> begin P1: process (bcd) begin     case bcd is         when "0000" =&gt; zec &lt;= "1000000000";         when "0001" =&gt; zec &lt;= "0100000000";         when "0010" =&gt; zec &lt;= "0010000000";         when "0011" =&gt; zec &lt;= "0001000000";         when "0100" =&gt; zec &lt;= "0000100000";         when "0101" =&gt; zec &lt;= "0000010000";         when "0110" =&gt; zec &lt;= "0000001000";         when "0111" =&gt; zec &lt;= "0000000100";         when "1000" =&gt; zec &lt;= "0000000010";         when "1001" =&gt; zec &lt;= "0000000001";         when others =&gt; zec &lt;= "0000000000";     end case; end process P1; end dec_arh;         </pre>
--	---

◆ **Exemplul 4:** Descrierea în limbaj VHDL a unui DMUX 1:8 (Implementare secvențială)

Observații	Codul VHDL
Secțiune dedicată includerii de librării	<pre> library IEEE; use IEEE.std_logic_1164.all;         </pre>
<p>Secțiune dedicată descrierii entității.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- nume entitate este: <b>dmux_A</b></li> <li>- intrările de selecție sunt introduse prin: <b>code_in</b>;</li> <li>- ieșirea de date este notată <b>Yout</b>;</li> <li>- intrarea de validare este declarată: <b>En</b>.</li> </ul>	<pre> entity dmux_A is port (     code_in :in std_logic_vector(2 downto 0);     En:in std_logic;     Yout:out std_logic_vector(7 downto 0)); end dmux_A;         </pre>
<p>Secțiune dedicată descrierii arhitecturii.</p> <p><b>Observații referitoare la procese:</b></p> <ul style="list-style-type: none"> <li>- Un proces este parcurs pentru orice schimbare de stare logică apărută la oricare variabilă din lista de senzitivități;</li> <li>- Declarațiile din corpul procesului sunt parcurse una după alta (execuție secvențială) și nu în paralel;</li> </ul> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- descrierea funcționării se face cu procesul P1, ce are în lista de senzitivități toate intrările DMUX;</li> <li>- procesul este parcurs la fiecare modificare apărută pe oricare intrare a DMUX;</li> <li>- intrarea de validare și ieșirile sunt active pe unu logic;</li> <li>- declarația <b>case</b> este parcursă doar dacă E=1;</li> </ul>	<pre> architecture dmux_arh of dmux_A is begin P1: process (code_in, En) begin     if (En='0') then         Yout &lt;= (others =&gt; '0');     else         case code_in is             when "000" =&gt; Yout &lt;= "00000001";             when "001" =&gt; Yout &lt;= "00000010";             when "010" =&gt; Yout &lt;= "00000100";             when "011" =&gt; Yout &lt;= "00001000";             when "100" =&gt; Yout &lt;= "00010000";             when "101" =&gt; Yout &lt;= "00100000";             when "110" =&gt; Yout &lt;= "01000000";             when "111" =&gt; Yout &lt;= "10000000";             when others =&gt; Yout &lt;= "00000000";         end case;     end if; end process; end dmux_arh;         </pre>

◆ **Exemplul 5:** Sumator pe 2 biți cu afișarea rezultatului pe un digit cu 7 segmente

În acest exemplu se descrie un CLC care realizează sumarea a doi operanzi, fiecare fiind exprimat pe 2 biți. Rezultatul se afișează pe un digit cu 7 segmente. Deoarece cel mai mare număr pe 2 biți este 3, rezultatul maxim al operației de adunare este 6, deci poate fi afișat pe un singur digit.

Observații	Codul VHDL
Secțiune dedicată includerii de librării	<pre> library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_unsigned.all; use IEEE.std_logic_arith.all;         </pre>
<p>Secțiune dedicată descrierii entității.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- avem două intrări de 2 biți: <b>x, y</b>;</li> <li>- o ieșire pe 7 biți pentru comanda segmentelor afișajului <b>afis</b>;</li> <li>- o ieșire pentru comanda catodului comun al afișajului <b>d_afis</b>;</li> </ul>	<pre> entity sum2 is port( d_activare: out std_logic;     x,y:in std_logic_vector(1 downto 0);     afis:out std_logic_vector(6 downto 0)); end sum2;         </pre>

<p>Secțiune dedicată descrierii arhitecturii. În cazul de față:</p> <ul style="list-style-type: none"> <li>- numele arhitecturii este: <b>arh_sum2</b>;</li> <li>- asocierea arhitecturii se face cu entitatea <b>sum2</b>;</li> <li>- descrierea folosește numai declarații concurente (pentru operația de sumare, pentru activare afișaj, pentru decodificare BCD-7seg.);</li> </ul> <p><b>Observații:</b></p> <ul style="list-style-type: none"> <li>- o declarație concurentă este executată pentru orice modificare de stare logică a semnalelor implicate în ea, deci <b>sum</b> este reactualizat pentru orice modificare apărută pe <b>x</b> sau pe <b>y</b>;</li> </ul>	<pre> <b>architecture</b> arh_sum2 <b>of</b> sum2 <b>is</b> <b>signal</b> suma : <b>std_logic_vector</b>(3 <b>downto</b> 0); <b>begin</b> suma &lt;= x + y; d_activare &lt;= '1'; <b>with</b> suma <b>select</b> afis&lt;=  "1000000" <b>when</b> "0000", --0         "1111001" <b>when</b> "0001", --1         "0100100" <b>when</b> "0010", --2         "0110000" <b>when</b> "0011", --3         "0011001" <b>when</b> "0100", --4         "0010010" <b>when</b> "0101", --5         "0000010" <b>when</b> "0110", --6         "1111000" <b>when</b> "0111", --7         "0000000" <b>when</b> "1000", --8         "0010000" <b>when</b> "1001", --9         "1111111" <b>when</b> <b>others</b>; --alte situatii <b>end</b> arh_sum2; </pre>
<p><b>Fisierul de constrângeri</b></p> <ul style="list-style-type: none"> <li>- rezultatul se afișează pe primul digit al machetei (prima linie din fișierul de constrângeri);</li> <li>- următoarele 7 linii sunt pentru comanda segmentelor digitului;</li> <li>- introducerea operandului <b>x</b> se face prin SW1 și SW2;</li> <li>- introducerea operandului <b>y</b> se face prin SW7 și SW8;</li> </ul>	<pre> NET "d_activare" LOC = "P70"; NET "afis&lt;0&gt;" LOC = "P39"; NET "afis&lt;1&gt;" LOC = "P41"; NET "afis&lt;2&gt;" LOC = "P44"; NET "afis&lt;3&gt;" LOC = "P46"; NET "afis&lt;4&gt;" LOC = "P48"; NET "afis&lt;5&gt;" LOC = "P51"; NET "afis&lt;6&gt;" LOC = "P53"; NET "x&lt;1&gt;" LOC = "P37"; NET "x&lt;0&gt;" LOC = "P40"; NET "y&lt;1&gt;" LOC = "P52"; NET "y&lt;0&gt;" LOC = "P54"; </pre>



## 4. Desfășurarea lucrării

### 4.1. Studiul circuitelor decodificatoare (DCD).

**A.** Referitor la circuitul decodificator din figura 1, răspundeți la următoarele întrebări:

- Ce formă canonică a fost folosită în proiectarea sa ?
- Se poate modifica schema logică pentru a obține un circuit cu ieșirile active pe 1 logic ? Cum ?
- Cât timp se menține activă o ieșire ?
- Dacă C=0, B=0 și A=1, codul perceput de circuit este 001 sau 100? Ce ieșire se activează în acest caz?
- Ce succesiune de coduri trebuie aplicate pe intrările de selecție dacă dorim activarea ieșirilor în ordinea:  $\bar{3}, \bar{5}, \bar{7}, \bar{9}$  ?

**B.** Pentru un circuit decodificator BCD-zecimal, spre exemplu circuitul 74LS442, se cere desenarea corelată în timp a semnalelor de ieșire dacă pe intrările de selecție sunt aplicate semnalele din figura alăturată în ordinea:

- a) Q1 → A, Q2 → B, Q3 → C, Q4 → D ;  
b) Q1 → D, Q2 → C, Q3 → B, Q4 → A ;

Cum explicați diferențele apărute ?

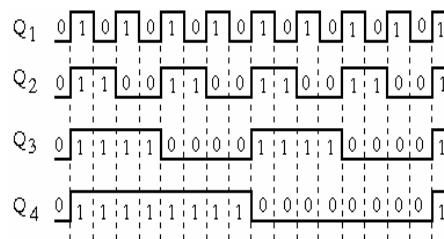


Fig. 4

**C.** Scrieți tabelul de adevăr pentru un decodificator cu 4 intrări și 7 ieșiri destinat comenzii unui afișaj numeric. Intrările sunt notate prin DCBA, iar ieșirile a b c d e f g. Ieșirile sunt active pe unu logic (unu logic = segment aprins).  
Tabelul trebuie realizat astfel încât cifrele să arate ca în figura alăturată.



Fig. 5

### 4.2. Studiul circuitelor demultiplexoare (DMUX).

**A.** Referitor la circuitul decodificator din figura 2, răspundeți la următoarele întrebări:

- Se poate modifica schema logică pentru a obține un circuit cu ieșirile active pe 1 logic? Cum ?
- Dacă  $\bar{E} = 0$ , B=0 și A=1, codul perceput de circuit este 01 sau 10? Ce ieșire se activează în acest caz?

- Care sunt cele două moduri în care se poate face dezactivarea unei ieșiri ?
- Ce succesiune de coduri trebuie aplicate pe intrările de selecție dacă dorim activarea ieșirilor în ordinea: 1, 3, 2, 0 ?

**B.** Care este modul de conectare al unui decodificator zecimal, spre exemplu circuitul 74LS442, pentru a obține o funcționare similară unui DUX 1:8 ?

#### 4.3. Implementarea funcțiilor binare cu DCD.

**A.** Folosind un DCD cu ieșiri active pe zero logic și porți logice cu număr cât mai mic de intrări (vezi exemplul 2), se cere implementarea unui circuit logic cu 3 intrări și 4 ieșiri având tabelul de adevăr din fig. 6;

X	x <sub>2</sub>	x <sub>1</sub>	x <sub>0</sub>	y <sub>3</sub>	y <sub>2</sub>	y <sub>1</sub>	y <sub>0</sub>	Y
0	0	0	0	0	0	1	1	3
1	0	0	1	0	0	0	0	0
2	0	1	0	0	1	0	1	5
3	0	1	1	0	1	1	0	6
4	1	0	0	0	0	0	1	1
5	1	0	1	1	0	0	0	8
6	1	1	0	1	0	0	1	9
7	1	1	1	0	0	1	0	2

Fig. 6

#### 4.4. Utilizarea ISE WebPack pentru descrierea aplicațiilor sub formă de scheme logice

- A.** Folosind facilitatea mediului de dezvoltare ISE WebPack, prin care se permite descrierea sistemelor digitale prin intermediul schemelor logice, se cere:
- Implementarea și verificarea pe macheta de laborator cu CPLD a schemei de DCD din figura 1.
  - Implementarea și verificarea pe macheta de laborator cu CPLD a schemei de DMUX din figura 2.
  - Implementarea și verificarea pe macheta de laborator cu CPLD a schemelor logice din exemplul 2.

**Mod de lucru:** Se folosesc indicațiile de la sfârșitul lucrării.

#### 4.5. Implementarea circuitelor logice cu ajutorul limbajului VHDL

**A.** Ținând cont de descrierile VHDL prezentate în exemplele anterioare, se cere codul VHDL și implementarea pe macheta de laborator pentru:

- Un decodificator zecimal cu ieșiri active pe zero logic;
- Un DMUX 1:8 ;

După implementare, verificați că starea logică a intrării de date selectate este transmisă și la ieșirea circuitului.

**B.** Implementați pe macheta de laborator cu CPLD, sumatorul pe 2 biți prezentat în exemplul 5.

**C.** Adaptați metoda de descriere a unui decodificator (exemplul 3), pentru implementarea decodificatorului BCD-7 segmente de la punctul 4.1.C. din desfășurarea lucrării.

**D.** Realizați o descriere în limbaj VHDL și implementați pe macheta cu CPLD, un circuit de transcodare pe 4 biți care să facă trecerea de la coul binar natural la codul Gray.

**Mod de lucru:** Se folosesc indicațiile de la sfârșitul lucrării.

### 5. Indicații privind modul de lucru

Pentru fiecare aplicație este necesară deschiderea unui nou proiect după metodologia prezentată într-o lucrare de laborator anterioară. Toate aplicațiile din această lucrare necesită doar un singur fișier sursă (fie schemă logică, fie sursă VHDL) și un singur fișier de constrângeri .

Referitor la conectarea intrărilor și a ieșirilor din circuit facem următoarele precizări:

- Intrările de selecție și cele de date se vor conecta la switch-urile (comutatoare cu două poziții) de pe macheta de laborator. În acest mod, trecerea comutatorului de pe o poziție pe alata echivalează cu schimbarea stării logice a variabilei de intrare.
- Variabilele de ieșire se vor conecta la LED-urile de pe macheta de laborator. În acest mod, în momentul în care o variabilă de ieșire este în unu logic, LED-ul asociat luminează.
- În cazul aplicației de sumare fiecare număr de intrare respectiv ieșire va fi reprezentat printr-un vector. Un vector de intrare trebuie conectat la un pachet de switch-uri (un număr corespunzător de switch-uri ce sunt amplasate unul lângă altul) iar vectorul de ieșire va fi reprezentat pe un pachet de LED-uri.
- Fișierul de constrângeri pentru exemplul 3 este prezentat alăturat. Pentru introducerea numărului  $x$  s-au folosit primele trei comutatoare (SW1, SW2, SW3), iar pentru  $y$  s-au folosit ultimele trei comutatoare (SW6, SW7, SW8). Afișarea rezultatului se face pe primele 5 LED-uri (LD1÷LD5).

NET "x<0>" LOC = "P37";  
NET "x<1>" LOC = "P40";  
NET "x<2>" LOC = "P43";

NET "y<0>" LOC = "P54";  
NET "y<1>" LOC = "P52";  
NET "y<2>" LOC = "P50";

NET "z<0>" LOC = "P75";  
NET "z<1>" LOC = "P71";  
NET "z<2>" LOC = "P67";  
NET "z<3>" LOC = "P65";  
NET "z<4>" LOC = "P62";



## Lucrarea nr. 4: **Implementarea funcțiilor binare folosind MUX**

### 1. Scopul lucrării

În partea teoretică se pune accent pe cunoașterea multiplexorului (simbol, structură internă, tabel de adevăr) și înțelegerea modului de utilizare a multiplexorului în implementarea funcțiilor binare.

În partea aplicativă se urmărește fixarea deprinderilor acumulate în lucrările anterioare, privind dezvoltarea aplicațiilor cu structuri logice programabile de tip CPLD, fie pe baza schemelor logice, fie folosind limbajul VHDL.

### 2. Considerente teoretice

Realizarea funcțiilor binare numai cu ajutorul porților logice este, de cele mai multe ori, greoaie și implică un efort mare de calcul în vederea simplificării - mai ales când numărul variabilelor de intrare este mare. O soluție mai bună constă în utilizarea circuitelor de complexitate medie, în special a multiplexoarelor (MUX), a decodificatoarelor (DCD) sau a demultiplexoarelor DMUX.

#### 2.1. Circuitul multiplexor (MUX)

Multiplexorul este un circuit logic combinațional prevăzut cu:  $n$  intrări de selecție,  $2^n$  intrări de date și o singură ieșire de date. Prin intermediul unui cuvânt de cod de  $n$  biți (adresa de selecție), multiplexorul conectează la ieșirea de date una din intrările sale de date. Funcționarea acestui circuit poate fi asemănată cu cea a unui comutator rotativ cu mai multe poziții de intrare, așa cum se prezintă în figura 1.b. Intrarea de validare  $\bar{E}$ , permite validarea funcționării ( $\bar{E}=0$ ) sau blocarea funcționării ( $\bar{E}=1$ ) circuitului.

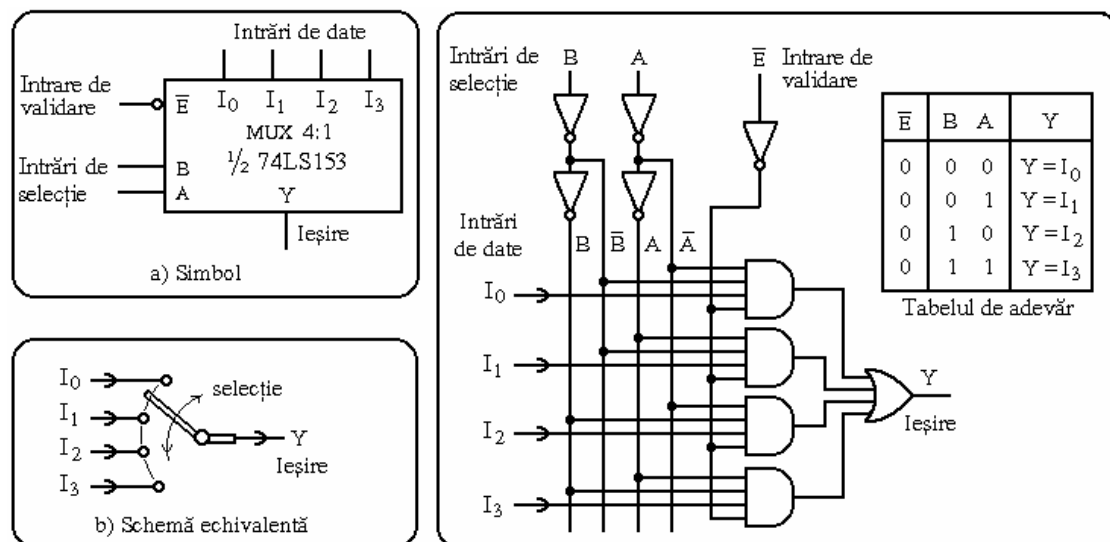
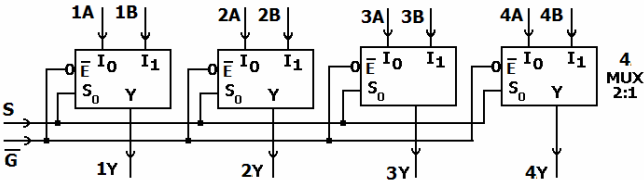
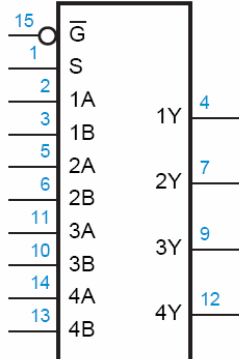
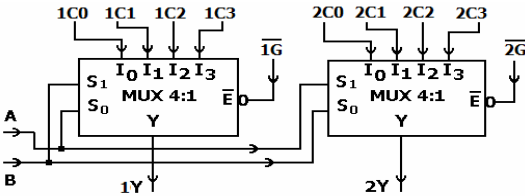
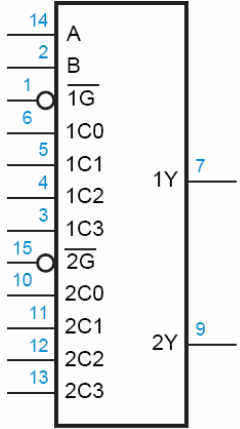
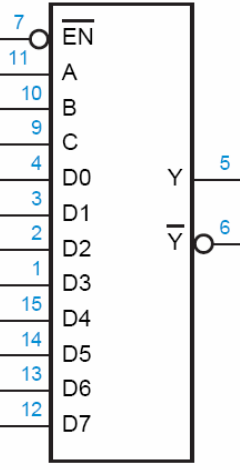


Fig. 1. Schema logică, tabelul de adevăr și simbolul pentru un MUX 4:1 de tip 74LS153

#### 2.2. Exemple de multiplexoare realizate în structuri integrate

Observații	Simbol
<b>Circuitul 74LS157 (4xMUX2:1)</b> <ul style="list-style-type: none"> <li>- circuitul conține 4 multiplexoare de tip MUX 2:1;</li> <li>- toate multiplexoarele folosesc aceeași intrare de selecție (S), respectiv aceeași intrare de validare <math>\bar{G}</math>;</li> <li>- datorită modului de conectare, multiplexoarele nu pot fi comandate separat;</li> <li>- modul de conectare a celor 4 multiplexoare din capsula 74LS157 este prezentat în figura de mai jos:</li> </ul>  <ul style="list-style-type: none"> <li>- Intrarea <math>\bar{G}</math> este folosită pentru validarea funcționării multiplexoarelor:</li> <li>- dacă <math>\bar{G}=0</math>, funcționarea multiplexoarelor este validată (permisă);</li> </ul>	<b>74LS157</b> 

<p>- dacă <math>\overline{G} = 1</math>, ieșirile multiplexoarelor nu se modifică indiferent de starea logică a celorlalte intrări;</p>	
<p><b>Circuitul 74LS153 (2xMUX4:1)</b></p> <ul style="list-style-type: none"> <li>- circuitul conține 2 multiplexoare de tip MUX 4:1;</li> <li>- cele două multiplexoare folosesc în comun intrările de selecție;</li> <li>- datorită modului de conectare, multiplexoarele nu pot fi comandate separat;</li> <li>- intrările de validare sunt separate;</li> <li>- modul de conectare a multiplexoarelor din capsula 74LS153 este prezentat în figura de mai jos:</li> </ul>  <p>- intrarea de selecție cu ponderea cea mai mică este A, iar intrarea de selecție cu ponderea cea mai mare este B;</p>	<p><b>74LS153</b></p> 
<p><b>Circuitul 74LS151 (MUX8:1)</b></p> <ul style="list-style-type: none"> <li>- circuitul conține un multiplexor de tip MUX 4:1;</li> <li>- pe lângă ieșirea Y specifică fiecărui MUX, capsula 74LS151 ne oferă și complementul acesteia</li> <li>- intrarea de selecție cu ponderea cea mai mică este A, iar intrarea de selecție cu ponderea cea mai mare este C;</li> </ul>	<p><b>74 LS 151</b></p> 

### 2. 3. Utilizarea MUX în implementarea funcțiilor binare

Dacă implementăm o funcție logică direct după prima formă canonică (fără a face nici o simplificare), obținem o schemă logică structurată pe trei nivele:

- primul nivel este format din inversoare și este folosit pentru calculul complementelor variabilelor de intrare;
- nivelul doi este format din porți AND și este folosit pentru calculul mintermenilor funcției (reamintim că nu trebuie calculați toți mintermenii, este suficient să calculăm doar pe aceia pentru care funcția are valoarea "1");
- nivelul trei este format dintr-o singură poartă OR pentru a realiza sumarea mintermenilor.

Dacă analizăm structura internă a unui multiplexor, (vezi lucrarea anterioară), găsim o arhitectură similară cu cea rezultată în urma implementării după prima formă canonică. Diferențe apar pe nivelul doi: există câte o poartă AND pentru fiecare mintermen, în plus, fiecare poartă de pe acest nivel are o intrare suplimentară ce se constituie în intrarea de date a MUX.

Datorită acestor asemănări structurale a apărut destul de repede ideea că multiplexorul se poate folosi în implementarea de funcții binare. Intuitiv ne dăm seama că pe intrările de selecție trebuie conectate variabilele funcției ce trebuie implementate iar pe intrările de date valorile funcției. În funcție de raportul dintre numărul de variabile ale funcției binare și numărul de intrări de selecție ale MUX pot exista următoarele situații:

- Numărul intrărilor de selecție este egal cu numărul variabilelor funcției.** În acest caz, implementare funcției binare se face fără efort de minimizare și nu mai necesită nici un alt circuit suplimentar. Dacă conectarea variabilelor funcției la intrările de selecție se face cu respectarea ponderilor, ( $x_0 \rightarrow A$ ,  $x_1 \rightarrow B$ , ...), nu trebuie să facem altceva decât să copiem valorile funcției din tabelul de adevăr pe intrările de date ale MUX.
- Numărul intrărilor de selecție este mai mic decât numărul variabilelor funcției.** În acest caz, o parte din variabilele funcției vor fi folosite pentru comanda intrărilor de selecție iar restul vor intra în calculul unor subfuncții binare ce se vor aplica la intrările de date ale MUX. Dacă numărul variabilelor funcției,  $x$ , este mai mare cu  $p$  față de numărul  $n$  al intrărilor de selecție ale MUX, atunci trebuiesc calculate  $2^n$  funcții logice de  $p = x - n$  variabile. Calculul celor  $n$  subfuncții binare necesită circuite suplimentare pe lângă MUX și necesită și un oarecare efort de simplificare. Oricum, cele  $n$  subfuncții sunt mult mai ușor de implementat decât funcția inițială deoarece au un număr mult mai mic de variabile de intrare.

**C. Numărul intrărilor de selecție este mai mare decât numărul variabilelor funcției.** Această situație corespunde unei utilizări neeficiente a MUX și nu prezintă un interes prea mare din punct de vedere practic. Totuși, dacă într-un anumit context suntem forțați să folosim așa ceva, se procedează astfel: variabilele funcției se conectează la intrările de selecție cu respectarea ponderilor; intrările de selecție nefolosite se conectează la masă; pe intrările de date se copiază valorile funcției; intrările de date nefolosite se conectează la masă.

Pentru o înțelegere mai bună a modului de utilizare a MUX-ului în implementarea funcțiilor binare sunt prezentate două exemple considerând că avem de implementat o funcție binară de trei variabile ce este dată prin tabelul de adevăr alăturat.

$x_2$	$x_1$	$x_0$	$F$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

**Exemplul 1: Se cere implementarea funcției F folosind un circuit MUX 8:1**

Deoarece un MUX8:1 are trei intrări de selecție se constată că suntem într-o situație fericită în care numărul intrărilor de selecție este egal cu cel al variabilelor funcției.

Conectarea variabilelor funcției la intrările de selecție cu respectare ponderilor presupune următoarele conexiuni:  $x_0 = A$ ,  $x_1 = B$ ,  $x_2 = C$ .

Din prelucrarea primei forme canonice (forma disjunctivă) a funcției **F**, se obține relația (1). Relația (2) nu este altceva decât funcția de transfer a multiplexorului 8:1.

$$F = \begin{vmatrix} \bar{x}_2 \bar{x}_1 \bar{x}_0 \\ \bar{x}_2 \bar{x}_1 x_0 \\ \bar{x}_2 x_1 \bar{x}_0 \\ \bar{x}_2 x_1 x_0 \\ x_2 \bar{x}_1 \bar{x}_0 \\ x_2 \bar{x}_1 x_0 \\ x_2 x_1 \bar{x}_0 \\ x_2 x_1 x_0 \end{vmatrix} = \begin{vmatrix} 1 \bar{x}_2 \bar{x}_1 \bar{x}_0 \\ 1 \bar{x}_2 \bar{x}_1 x_0 \\ 1 \bar{x}_2 x_1 \bar{x}_0 \\ 1 \bar{x}_2 x_1 x_0 \\ 1 x_2 \bar{x}_1 \bar{x}_0 \\ 1 x_2 \bar{x}_1 x_0 \\ 1 x_2 x_1 \bar{x}_0 \\ 1 x_2 x_1 x_0 \end{vmatrix} = \begin{vmatrix} 0 \bar{x}_2 \bar{x}_1 \bar{x}_0 \\ 1 \bar{x}_2 \bar{x}_1 \bar{x}_0 \\ 1 \bar{x}_2 \bar{x}_1 x_0 \\ 1 \bar{x}_2 x_1 \bar{x}_0 \\ 0 \bar{x}_2 x_1 x_0 \\ 1 x_2 \bar{x}_1 \bar{x}_0 \\ 1 x_2 \bar{x}_1 x_0 \\ 1 x_2 x_1 \bar{x}_0 \\ 0 x_2 x_1 x_0 \\ 1 x_2 x_1 x_0 \end{vmatrix} = \begin{vmatrix} 0 S_0 \\ 1 S_1 \\ 1 S_2 \\ 1 S_3 \\ 0 S_4 \\ 1 S_5 \\ 1 S_6 \\ 1 S_7 \\ 0 S_8 \\ 1 S_9 \end{vmatrix} \quad (1)$$

$$Y = \begin{vmatrix} I_0 \bar{C} \bar{B} \bar{A} \\ I_1 \bar{C} \bar{B} A \\ I_2 \bar{C} B \bar{A} \\ I_3 \bar{C} B A \\ I_4 C \bar{B} \bar{A} \\ I_5 C \bar{B} A \\ I_6 C B \bar{A} \\ I_7 C B A \end{vmatrix} = \begin{vmatrix} I_0 S_0 \\ I_1 S_1 \\ I_2 S_2 \\ I_3 S_3 \\ I_4 S_4 \\ I_5 S_5 \\ I_6 S_6 \\ I_7 S_7 \end{vmatrix} \quad (2)$$

Din compararea relațiilor (1) și (2), prin identificare, rezultă:  $I_0=0$ ,  $I_1=1$ ,  $I_2=1$ ,  $I_3=0$ ,  $I_4=1$ ,  $I_5=1$ ,  $I_6=0$ ,  $I_7=1$ . Schema de conectare a MUX 8:1 pentru a realiza funcției cerute se prezintă în figura 2.a.

Conectarea variabilelor funcției la intrările de selecție în ordinea ponderilor ușurează modul de amplasare a valorilor funcției la intrările de date: valoarea funcției pentru combinația 0 se conectează la  $I_0$ , valoarea funcției pentru combinația 1 se conectează la  $I_1$ , etc.

**Atenție:** Conectarea fără respectarea ponderilor nu este interzisă însă, în astfel de situații, modul de conectare al valorilor funcției la intrările de date este cu totul altul.

**Exemplul 2: Se cere implementarea funcției F folosind un circuit MUX 4:1**

Deoarece un MUX 4:1 are două intrări de selecție se constată că numărul intrărilor de selecție este mai mic cu o unitate decât numărul de variabile ale funcției F.

Pentru astfel de cazuri, în mod arbitrar se alege două variabile ale funcției pentru comanda intrărilor de selecție ale multiplexorului. **În cazul de față alege următoarea variantă:  $x_2 = A$ ,  $x_0 = B$ .**

Prelucrarea primei forme canonice (forma disjunctivă) a funcției **F**, se face în concordanță cu alegerea deja făcută și necesită o succesiune de câteva etape:

- scrierea primei forme canonice după tipicul deja cunoscut;
- rearanjarea variabilelor funcției în produse după ponderea intrărilor de selecție pe care le comandă (în cazul nostru,  $x_0$  trebuie să apară pe prima poziție,  $x_2$  pe poziția a doua după care vin restul de variabile);
- se dau factori comuni toate combinațiile posibile ale variabilelor ce comandă intrările de selecție (în cazul nostru, toate combinațiile posibile ale variabilelor  $x_0 x_2$ );
- rezultatul acestor prelucrări efectuate asupra formei canonice se compară cu funcția de transfer a MUX-ului folosit și de deduc expresiile logice ale subfuncțiilor ce trebuie conectate la intrările de date ale MUX (în cazul de față se compară relațiile 3 și 4).

$$F = \begin{vmatrix} \bar{x}_2 \bar{x}_1 \bar{x}_0 \\ \bar{x}_2 \bar{x}_1 x_0 \\ \bar{x}_2 x_1 \bar{x}_0 \\ \bar{x}_2 x_1 x_0 \\ x_2 \bar{x}_1 \bar{x}_0 \\ x_2 \bar{x}_1 x_0 \\ x_2 x_1 \bar{x}_0 \\ x_2 x_1 x_0 \end{vmatrix} = \begin{vmatrix} x_0 \bar{x}_2 \bar{x}_1 \\ x_0 \bar{x}_2 x_1 \\ \bar{x}_0 \bar{x}_2 \bar{x}_1 \\ \bar{x}_0 \bar{x}_2 x_1 \\ x_0 \bar{x}_2 \bar{x}_1 \\ x_0 \bar{x}_2 x_1 \\ \bar{x}_0 x_2 \bar{x}_1 \\ \bar{x}_0 x_2 x_1 \end{vmatrix} = \begin{vmatrix} \bar{x}_0 \bar{x}_2 \bar{x}_1 \\ \bar{x}_0 \bar{x}_2 x_1 \\ \bar{x}_0 x_2 \bar{x}_1 \\ \bar{x}_0 x_2 x_1 \\ x_0 \bar{x}_2 \bar{x}_1 \\ x_0 \bar{x}_2 x_1 \\ x_0 x_2 \bar{x}_1 \\ x_0 x_2 x_1 \end{vmatrix} = \begin{vmatrix} \bar{x}_0 \bar{x}_2 \bar{x}_1 \\ \bar{x}_0 \bar{x}_2 x_1 \\ \bar{x}_0 x_2 \bar{x}_1 \\ \bar{x}_0 x_2 x_1 \\ x_0 \bar{x}_2 \bar{x}_1 \\ x_0 \bar{x}_2 x_1 \\ x_0 x_2 \bar{x}_1 \\ x_0 x_2 x_1 \end{vmatrix} = \begin{vmatrix} S_0 x_1 \\ S_1 \bar{x}_1 \\ S_2 \bar{x}_1 \\ S_3 1 \end{vmatrix} \quad (3)$$

$$Y = \begin{vmatrix} I_0 \bar{B} \bar{A} \\ I_1 \bar{B} A \\ I_2 B \bar{A} \\ I_3 B A \end{vmatrix} = \begin{vmatrix} I_0 S_0 \\ I_1 S_1 \\ I_2 S_2 \\ I_3 S_3 \end{vmatrix} \quad (4)$$

Comparând relațiile (3) și (4) se constată că intrările multiplexorului trebuie conectate astfel:  $I_0 = x_1$ ,  $I_1 = \bar{x}_1$ ,  $I_2 = \bar{x}_1$ ,  $I_3 = 1$ . Schema de conectare a MUX pentru realizarea funcției cerute se prezintă în figura 2. c).

O altă variantă de realizare a funcției binare F cu ajutorul unui MUX4:1, este prezentată în figura 2.b.



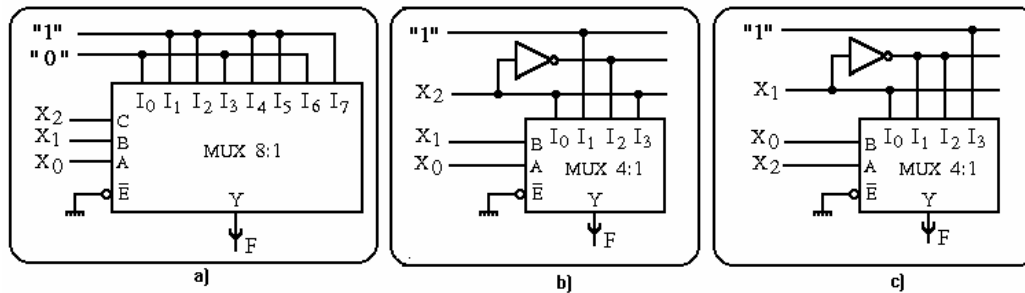


Fig. 2. Variante de implementare ale funcției F folosind multiplexoare de tip MUX8:1 respectiv MUX4:1.

### • Concluzie

Multiplexorul poate fi privit ca un circuit universal pentru implementare funcțiilor logice dar prezintă inconvenientul că mintermenii (calculați la nivelul porților AND) nu pot fi utilizați decât o singură dată. În consecință, MUX-ul nu se pretează pentru implementarea mai multor funcții binare de aceleași variabile deoarece nu există posibilitatea ca mintermenii calculați pentru o funcție să fie utilizați și de celelalte funcții.

## 3. Utilizarea limbajului VHDL pentru descrierea MUX

După cum este deja cunoscut, limbajul VHDL permite o descriere foarte ușoară a funcționării unui circuit/sistem logic. Reamintim că limbajul VHDL, permite descrierea circuitelor/sistemelor logice în câteva moduri:

- **descriere structurală** - proiectantul impune schema logică iar sarcina mediului software este de a "amplasa" corect această schemă în resursele interne ale circuitului în care se dezvoltă aplicația;
- **descriere comportamentală** - proiectantul face o descriere de nivel înalt a circuitului/sistemului urmând ca sinteza propriu-zisă a schemei logice să rămână în sarcina mediului software de dezvoltare.
- **Metode combinate** - proiectantul poate să opteze pentru descriere structurală pentru anumite blocuri funcționale și descriere comportamentală pentru altele.

De regulă, limbajul VHDL este conceput pentru implementarea în paralel a circuitelor logice, de aceea, de cele mai multe ori, ordinea introducerii declarațiilor nu este importantă. Singura modalitate permisă de VHDL de a introduce operații cu execuție secvențială (una după alta, în ordinea scrierii lor în program) constă în folosirea declarației de tip **process**. *Atenție, de această dată, ordinea declarațiilor din corpul unui proces are importanță !*

### ◆ Exemplul 3: Descrierea în limbaj VHDL a unui MUX 4:1 ( Implementare concurentă )

Observații	Codul VHDL
Secțiune dedicată includerii de librării	<pre>library IEEE; use IEEE.std_logic_1164.all;</pre>
Secțiune dedicată descrierii entității. În cazul de față: - nume entitate este: <b>mux4</b> - intrările de selecție sunt: <b>s1 și s0</b> ; - intrările de date sunt: <b>d0, d1, d2, d3</b> ; - ieșirea de date este notată <b>Yout</b> ;	<pre>entity mux4 is port (d0, d1, d2, d3 : in std_logic;       s1, s0 : in std_logic;       yout : out std_logic); end mux4;</pre>
Secțiune dedicată descrierii arhitecturii. În cazul de față: - numele arhitecturii este: <b>abc_arh</b> ; - arhitectura este asociată cu entitatea: <b>mux4</b> ; - descrierea funcționării se face cu două declarații concurente: una calculează semnalul intern <b>sel</b> , iar cealaltă calculează ieșirea <b>yout</b> ; - la prima vedere descrierea pare greșită deoarece semnalul intern <b>sel</b> , este folosit înainte de a fi calculat; - descriere este corectă deoarece succesiunea declarațiilor concurente nu contează, ele sunt parcurse în paralel; - semnalul <b>sel</b> , este recunoscut numai în interiorul arhitecturii <b>abc_arh</b> ;	<pre>architecture abc_arh of mux4 is signal sel: integer begin with sel select yout &lt;= d0 when 0,         d1 when 1,         d2 when 2,         d3 when 3,         'X' when others; sel &lt;= 0 when s0='0' and s1='0' else       1 when s0='1' and s1='0' else       2 when s0='0' and s1='1' else       3 when s0='1' and s1='1' else       4; end abc_arh;</pre>

### ◆ Exemplul 4: Descrierea în limbaj VHDL a unui MUX 4:1 (Implementare concurentă – variantă greșită)

Observații	Codul VHDL
Secțiune dedicată includerii de librării	<pre>library IEEE; use IEEE.std_logic_1164.all;</pre>

<p>Secțiune dedicată descrierii entității. În cazul de față:</p> <ul style="list-style-type: none"> <li>- nume entitate este: <b><i>mux4</i></b></li> <li>- intrările de selecție sunt: <b><i>s1 și s0</i></b>;</li> <li>- intrările de date sunt: <b><i>d0, d1, d2, d3</i></b>;</li> <li>- ieșirea de date este notată <b><i>yout</i></b>;</li> </ul>	<pre>entity mux4 is port (d0, d1, d2, d3 : in std_logic;       s1, s0 : in std_logic;       yout : out std_logic); end mux4;</pre>
<p>Secțiune dedicată descrierii arhitecturii. În cazul de față:</p> <ul style="list-style-type: none"> <li>- descrierea se face cu 4 declarații concurente, la prima vedere pare corectă dar în realitate este <b>greșită</b>;</li> </ul> <p><b>Unde este greșeala?</b></p> <ul style="list-style-type: none"> <li>- fiecare atribuire pentru <b><i>yout</i></b>, generează un nou driver pentru comanda semnalului de ieșire <b><i>yout</i></b>;</li> <li>- așadar avem 4 drivere, fiecare încearcă să impună propria sa valoare logică asupra firului de ieșire <b><i>yout</i></b>, lucru ce conduce la conflict;</li> </ul>	<pre>architecture gresit of mux4 is begin yout &lt;= d0 when s0='0' and s1='0' else '0'; yout &lt;= d1 when s0='1' and s1='0' else '0'; yout &lt;= d2 when s0='0' and s1='1' else '0'; yout &lt;= d3 when s0='1' and s1='1' else '0'; end gresit;</pre>
<p><b>Cum se poate corecta greșeala?</b></p> <ul style="list-style-type: none"> <li>- varianta corectă trebuie să folosească o singură atribuire pentru <b><i>yout</i></b>;</li> </ul>	<pre>architecture corect of mux4 is begin yout &lt;= d0 when s0='0' and s1='0' else,         d1 when s0='1' and s1='0' else,         d2 when s0='0' and s1='1' else,         d3 when s0='1' and s1='1' else,         'X'; -- pentru necunoscut end corect;</pre>

♦ **Exemplul 5:** Descrierea în limbaj VHDL a unui MUX 4:1 (Implementare secvențială - varianta 1)

Observații	Codul VHDL
<p>Secțiune dedicată includerii de librării</p>	<pre>library IEEE; use IEEE.std_logic_1164.all;</pre>
<p>Secțiune dedicată descrierii entității. În cazul de față:</p> <ul style="list-style-type: none"> <li>- nume entitate este: <b><i>mux4</i></b></li> <li>- vectorul de selecție cu două componente: <b><i>sel</i></b>;</li> <li>- intrările de date sunt: <b><i>d0, d1, d2, d3</i></b>;</li> <li>- ieșirea de date este notată <b><i>Yout</i></b>;</li> </ul>	<pre>entity mux4 is port ( d0, d1, d2, d3 : in std_logic; sel: in std_logic_vector(1 downto 0); Yout : out std_logic ); end mux4;</pre>
<p>Secțiune dedicată descrierii arhitecturii. În cazul de față:</p> <ul style="list-style-type: none"> <li>- descrierea funcționării se face cu un proces ce are ca listă de sensibilități toate intrările MUX;</li> <li>- procesul este parcurs pentru orice modificare de stare logică apărută pe oricare intrare;</li> <li>- declarația <b><i>case</i></b> poate fi folosită numai în interiorul unui proces;</li> </ul> <p><b>Observații referitoare la procese:</b></p> <ul style="list-style-type: none"> <li>- Un proces este parcurs pentru orice schimbare de stare logică apărută la oricare variabilă din lista de sensibilități;</li> <li>- Declarațiile din corpul procesului sunt parcurse una după alta (execuție secvențială) și nu în paralel;</li> </ul>	<pre>architecture arh_3 of mux4 is begin P1: process (d0, d1, d2, d3, sel) begin case sel is when "00" =&gt; Yout &lt;= d0; when "01" =&gt; Yout &lt;= d1; when "10" =&gt; Yout &lt;= d2; when others =&gt; Yout &lt;= d3; end case; end process P1; end arh_3;</pre>

♦ **Exemplul 6:** Descrierea în limbaj VHDL a unui MUX4:1 (Implementare secvențială - varianta 2)

Observații	Codul VHDL
<p>Secțiune dedicată includerii de librării</p>	<pre>library IEEE; use IEEE.std_logic_1164.all;</pre>
<p>Secțiune dedicată descrierii entității. În cazul de față:</p> <ul style="list-style-type: none"> <li>- nume entitate este: <b><i>mux4_1</i></b>;</li> <li>- selecția se face cu un vector pe 2 biți: <b><i>adr</i></b>;</li> <li>- datele de intrare: <b><i>d0, d1, d2, d3</i></b>;</li> <li>- ieșirea de date: <b><i>data_out</i></b>;</li> </ul>	<pre>entity mux4_1 is port ( adr : in std_logic_vector(1 downto 0); d0,d1,d2,d3 : in std_logic; data_out : out std_logic); end mux4_1;</pre>
<p>Secțiune dedicată descrierii arhitecturii.  În cazul de față:</p> <ul style="list-style-type: none"> <li>- numele arhitecturii este: <b><i>arch4</i></b>;</li> <li>- arhitectura se asociază cu entitatea <b><i>mux4_1</i></b>;</li> <li>- procesul <b><i>P1</i></b>, este sensibil la intrarea de selecție și la intrările de date;</li> <li>- în funcție de combinația de pe intrările de selecție, ieșirea <b><i>data_out</i></b> are o singură atribuire;</li> </ul>	<pre>architecture arch4 of mux4_1 is begin P1: process (adr, data_in) begin case adr is when "00" =&gt; data_out &lt;= d0; when "01" =&gt; data_out &lt;= d1; when "10" =&gt; data_out &lt;= d2; when "11" =&gt; data_out &lt;= d3;</pre>

```

when others => data_out <= "---";
end case;
end process P1;
end arch4;

```



## 4. Desfășurarea lucrării

### 4.1. Studiul circuitelor multiplexoare (MUX).

A. Referitor la circuitul MUX 4:1 din figura 1, răspundeți la următoarele întrebări:

- În ce stare logică se află ieșirea de date dacă  $\bar{E} = 1$  ?
- Dacă  $\bar{E} = 0$ ,  $B=0$  și  $A=1$ , intrarea selectată este I0 sau I2? Ce intrare este conectată la ieșirea de date ?
- Care este intrarea de selecție cu ponderea cea mai mare ?
- Acest circuit poate fi implementat doar cu porți NAND? Dacă da, care este schema logică?

B. În ce ordine trebuie conectate semnalele  $x$ ,  $y$ ,  $z$  la intrările unui MUX8:1 astfel încât la ieșire să obținem semnalul  $Sout$  ?

Modul de comandă a selecțiilor și forma semnalelor  $x$ ,  $y$ ,  $z$  se prezintă în figura 3.

Semnalul  $Sout$  poate fi obținut prin utilizarea unui MUX 4:1? Dacă da, care este schema de conectare?

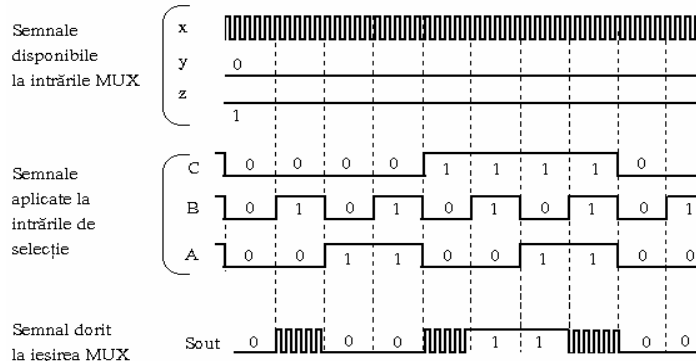


Fig. 3

### 4.1. Implementarea funcțiilor binare cu MUX.

A. Referitor la circuitele logice din fig. 2, răspundeți la următoarele întrebări:

- Cum trebuie conectate intrările de date ale MUX8:1, pentru a realiza aceeași funcție logică, în situația în care comanda intrărilor de selecție se face astfel:  $x_0 \rightarrow C$ ,  $x_1 \rightarrow B$ ,  $x_2 \rightarrow A$ ;
- Cum trebuie conectate intrările de date ale MUX4:1, pentru a realiza aceeași funcție logică, în situația în care comanda intrărilor de selecție se face astfel:  $x_1 \rightarrow B$ ,  $x_2 \rightarrow A$ ;

B. Folosind modul de lucru prezentat în exemplul 2, se cere implementarea unei funcții logice cu tabelul de adevăr din figura 4, folosind un MUX 4:1.

c	b	a	yout
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Fig. 4

### 4.4. Utilizarea ISE WebPack pentru descrierea aplicațiilor sub formă de scheme logice

Folosind facilitatea mediului de dezvoltare ISE WebPack, prin care se permite descrierea sistemelor digitale prin intermediul schemelor logice, se cere:

- Implementarea și verificarea pe macheta de laborator cu CPLD a schemei din figura 1.
- Implementarea și verificarea pe macheta de laborator cu CPLD a schemelor logice din figura 2.

Mod de lucru: vezi indicațiile de la "Implementarea funcțiilor binare cu rețele de porți logice"

### 3.4. Implementarea circuitelor logice cu ajutorul limbajului VHDL

- Realizați o descriere comportamentală, în limbaj VHDL, pentru circuitele logice din figura 2, după care verificați funcționarea acestora pe macheta de laborator cu CPLD.
- Tinând cont de descrierile VHDL prezentate în exemplele 3÷6, se cere codul VHDL și implementarea pe macheta de laborator pentru un circuit de MUX8:1 cu intrare de validare activă pe zero logic; După implementare, verificați că starea logică a intrării de date selectate este transmisă și la ieșirea circuitului.



## Lucrarea nr. 5: **Aplicații cu circuite basculante monostabile**

### 1. Scopul lucrării

În această lucrare se prezintă câteva modalități de realizarea a circuitelor de generare a impulsurilor cu durată controlată din tranzițiile unui semnal de intrare precum și aplicațiile tipice ale acestor circuite. Se pune accent pe înțelegerea: modalităților de obținere a impulsurilor din tranziții; a diferențelor dintre monostabilul retriggerabil și cel nertriggerabil; funcționării și utilizării principalelor monostabile realizate în structuri integrate; aplicațiilor tipice; structurii și funcționării unui monostabil numeric; implemetarea CBM numerice în CPLD.

### 2. Considerente teoretice

#### 2.1. Caracteristica de transfer în tensiune a unei porți logice

Înainte de a trece la prezentarea propriu zisă a circuitelor ce fac tema acestei lucrări, vom face câteva precizări privind caracteristica de transfer în tensiune a unei porți logice. Într-o lucrare anterioară, am arătat că o poartă logică poate avea o caracteristică de transfer normală (standard) sau una de tip trigger Schmitt.

- În cazul porților logice cu o caracteristică de transfer normală (standard), există o singură valoare a tensiunii de prag,  $V_T$ , indiferent dacă tensiunea de intrare este crescătoare sau descrescătoare. Spre exemplu, pentru inversorul 7404 aparținând familiei TTL standard, valoarea tensiunii de prag este de cca. 1,4V.

- În cazul porților logice cu o caracteristică de transfer de tip trigger Schmitt, există două tensiuni de prag: o tensiune  $V_P$ , valabilă pentru sensul crescător al tensiunii de intrare și respectiv o tensiune  $V_N$ , valabilă pentru sensul descrescător al tensiunii de intrare. Spre exemplu, pentru inversorul trigger Schmitt 7414 din familia TTL standard, pragul pozitiv  $V_P \approx 1,7V$  iar cel negativ  $V_N \approx 0,7V$ .

Printre avantajele utilizării circuitelor logice cu caracteristică trigger Schmitt, cele mai importante sunt:

- schimbarea stării logice de la ieșirea circuitului se face foarte rapid, din acest motiv semnalul de la ieșirea unui trigger Schmitt se apropie foarte mult de semnalul digital ideal, adică durata fronturilor este foarte redusă;
- marginea de zgomot este mult mai mare decât în cazul circuitelor cu caracteristică normală;
- acceptă semnale de intrare cu fronturi oricât de lente.

Datorită acestor proprietăți, circuitele logice cu caracteristică de transfer de tip trigger Schmitt sunt deosebit de utile pentru situații în care:

- semnalul de intrare aplicat unui sistem logic este deformat (spre exemplu are fronturi foarte lente);
- peste semnalul util se suprapun zgomote;
- transmisia semnalelor între două componente ale aceleiași sistem logic este însoțită de reflexii datorate neadaptărilor de impedanță.

Pe lângă aceste aplicații, circuitele logice cu caracteristică de tip trigger Schmitt mai pot fi utilizate și pentru realizarea de oscilatoare digitale, de astabile sau de monostabile. Aceste aplicații vor fi descrise succint în partea teoretică a acestei lucrări.

#### 2.2. Obținerea unei caracteristici trigger Schmitt folosind circuite normale

Dacă dintr-un motiv sau altul, nu dispunem de circuite cu caracteristică trigger Schmitt, există posibilitatea de a obține o astfel de caracteristică folosind circuite cu caracteristică normală de transfer. Un exemplu de acest fel este prezentat în figura 1.

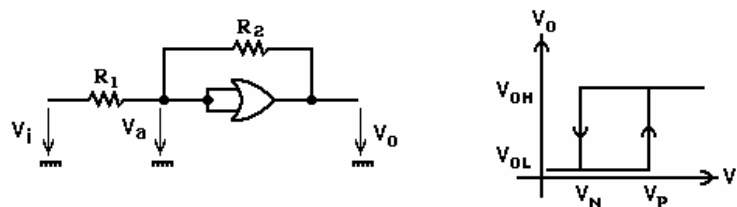


Fig. 1. Realizarea unei caracteristici de transfer de tip trigger Schmitt folosind porți cu caracteristică normală de transfer

Pentru a determina tensiunile de prag  $V_N$ , respectiv  $V_P$ , vom neglija curentul de intrare prin poarta logică. Această presupunere este foarte apropiată de adevăr pentru cazul circuitelor logice realizate în tehnologie CMOS și va genera mici erori în cazul circuitelor realizate în tehnologie TTL.

**A). Determinarea pragului superior,  $V_P$ .** Vom considera ieșirea porții în starea logică zero, (în acest caz  $V_o = V_{OL}$ ), și începem să creștem tensiunea de intrare  $V_i$ . Pentru o valoare oarecare a tensiunii de intrare  $V_i$ , expresia tensiunii  $V_a$  de la intrarea porții este dată de relația:

$$V_a = \frac{R_2}{R_1 + R_2} V_i + \frac{R_1}{R_1 + R_2} V_{OL} \quad (1)$$

În momentul în care tensiunea de intrare a crescut suficient de mult astfel încât la intrarea porții să se atingă tensiunea de tranziție, ( $V_a = V_T$ ), starea logică a ieșirii se va schimba, va trece în unu logic. Pentru întreg circuitul, poartă plus rezistențe, acest eveniment corespunde atingerii pragului superior. Pentru acest moment, ecuația (1) devine:

$$V_T = \frac{R_2}{R_1 + R_2} V_P + \frac{R_1}{R_1 + R_2} V_{OL} \quad (2)$$

din care deducem expresia tensiunii de prag superior:

$$V_P = \left(1 + \frac{R_1}{R_2}\right) V_T - \frac{R_1}{R_2} V_{OL} \quad (3)$$

**B). Determinarea pragului inferior,  $V_N$ .** Vom considera ieșirea porții în starea logică unu, (în acest caz  $V_O = V_{OH}$ ), și începem să scădem tensiunea de intrare  $V_i$ . Pentru o valoare oarecare a tensiunii de intrare  $V_i$ , expresia tensiunii  $V_a$  de la intrarea porții este dată de relația:

$$V_a = \frac{R_2}{R_1 + R_2} V_i + \frac{R_1}{R_1 + R_2} V_{OH} \quad (4)$$

În momentul în care tensiunea de intrare a scăzut suficient de mult astfel încât la intrarea porții să se atingă tensiunea de tranziție, ( $V_a = V_T$ ), starea logică a ieșirii se va schimba, va trece în zero logic. Pentru întreg circuitul, poartă plus rezistențe, acest eveniment corespunde atingerii pragului inferior. Pentru acest moment, ecuația (4) devine:

$$V_T = \frac{R_2}{R_1 + R_2} V_N + \frac{R_1}{R_1 + R_2} V_{OH} \quad (5)$$

din care deducem expresia tensiunii de prag superior:

$$V_N = \left(1 + \frac{R_1}{R_2}\right) V_T - \frac{R_1}{R_2} V_{OH} \quad (6)$$

Trebuie remarcat faptul că schema din figura 1 nu permite alegerea independentă a pragurilor, prin fixarea raportului  $R_1/R_2$  se acționează în același timp asupra ambelor praguri.

### 2.3. Generarea de impulsuri din tranziția semnalului de intrare

Există două metode de generare a impulsurilor din tranzițiile unui semnal de intrare: o metodă se bazează pe efectul timpului de propagare prin porțile logice reale, iar cealaltă metodă se bazează pe încărcarea-descărcarea unui condensator dintr-un circuit RC.

Prima metodă, bazată pe efectul timpilor de propagare, este mai puțin utilizată deoarece generează impulsuri scurte și lățimea lor nu poate fi modificată. Cealaltă metodă, bazată pe încărcarea-descărcarea unui condensator, este foarte des folosită deoarece prin modificarea valorilor circuitului RC se poate modifica și lățimea impulsului generat.

Înainte de a prezenta efectiv câteva scheme de generare a impulsurilor din tranzițiile semnalului de intrare, facem precizarea că descărcarea unui condensator într-un circuit RC, se face după relația:

$$u(t) = u(\infty) - [u(\infty) - u(0)] \exp\left(\frac{-t}{RC}\right) \quad (7)$$

Dacă ne interesează timpul scurs de la începutul descărcării până la atingerea tensiunii  $U_x$ , acesta este dat de relația:

$$\tau = -RC \ln \frac{U_x - u(\infty)}{u(0) - u(\infty)} \quad (8)$$

• **O schemă electrică sensibilă la tranziția pozitivă** a semnalului de intrare se prezintă în figura 2. Din formele de undă se observă că impulsul de ieșire este activ pe zero și corespunde descărcării condensatorului între cele două praguri ale caracteristicii trigger Schmitt. Durata impulsului generat ( $\tau$ ), depinde de circuitul RC dar și de caracteristicile electrice ale porții utilizate.

Pentru circuitul 74LS132 avem următoarele date de catalog: - tensiunea maximă de intrare pentru starea zero logic,  $V_{OLmax} = 0,8V$ ; - tensiunea tipică de ieșire în starea unu logic,  $V_{OH} = 3,6V$ ; - curentul maxim la intrare pentru starea zero logic  $I_{ILmax} = -0,4mA$ ; - tensiunea pentru pragul negativ,  $V_N = 1,1V$ ;

Pentru determinarea valorii maxime a rezistenței  $R$ , trebuie respectată relația:

$$I_{ILmax} R \leq V_{ILmax}$$

cea ce înseamnă că  $R \leq 2k\Omega$ .

După alegerea rezistenței  $R$ , valoarea condensatorului  $C$ , se determină în funcție de durata dorită a impulsului de ieșire folosind în mod adecvat relația (8). Ținem cont că descărcarea condensatorului începe de la  $u(0)=V_{OH}$  și se termină la  $u(\infty)=V_{OL}=I_{ILmax} R$ . Impulsul de ieșire începe odată cu descărcarea condensatorului și ține până când tensiunea pe rezistență atinge valoarea de prag negativ  $V_N$ .

În aceste condiții se obține: 
$$\tau = -RC \ln \frac{V_N - I_{ILmax} R}{V_{OH} - I_{ILmax} R}$$

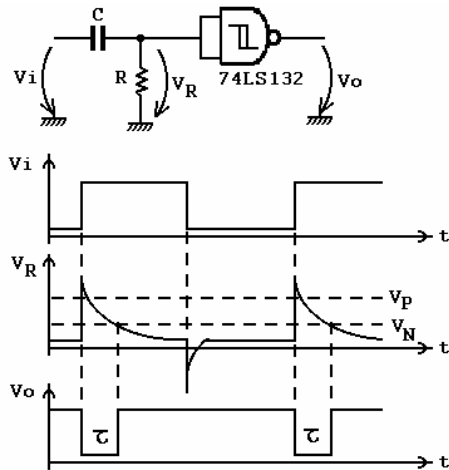


Fig. 2. Schema electrică a unui monostabil activ pe tranziția pozitivă a semnalului de intrare

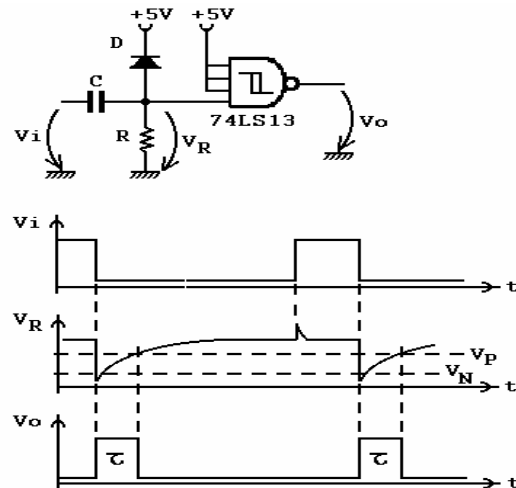


Fig. 3. Schema electrică a unui monostabil activ pe tranziția negativă a semnalului de intrare

• O schemă electrică sensibilă la tranziția negativă a semnalului de intrare se prezintă în figura 3. De această dată, impulsul de ieșire este activ pe unu logic.

Din analiza formelor de undă din figura 3, se vede că încărcarea condensatorului se face între  $V_{OL}$  și  $V_{OH}$  însă impulsul de ieșire este menținut activ doar până ce tensiunea de pe condensator atinge pragul pozitiv.

Valoarea rezistenței  $R$  se alege de cca. 20 kΩ pentru ca intrarea porții logice să se comporte ca și cum ar fi în unu logic (vezi structura internă a porții TTL). Valoarea condensatorului se calculează din considerente legate de lățimea dorită a impulsului de ieșire.

În condițiile particulare ale acestei scheme, relația (8) se scrie: 
$$\tau = -RC \ln \frac{V_{OH} - V_P}{V_{OH} - V_{OL}}$$

## 2.4. CBM realizate în structuri integrate

Circuitele logice special destinate generării de impulsuri din tranzițiile unui semnal de intrare poartă denumirea de circuite basculante monostabile (CBM). Lățimea impulsurilor generate este controlată prin intermediul unui circuit RC extern. Cel mai adesea, CBM-urile realizate integrat sunt dotate cu două intrări de comandă: una sensibilă la tranziția pozitivă (TR+) și alta la tranziția negativă (TR-), precum și cu două ieșiri complementare: Q respectiv  $\bar{Q}$ .

CBM-urile prezintă o stare stabilă în care ieșirea este în zero logic ( $Q=0$ ) și o stare metastabilă în care ieșirea se află în starea de unu logic ( $Q=1$ ).

Referitor la starea metastabilă putem face următoarele observații:

- intrarea în această stare este o consecință a unei tranziții active aplicată pe intrarea de comandă (denumită uneori și intrare de declanșare) atunci când CBM-ul se află în starea stabilă;
- durata stării metastabile ( $\tau$ ), este dependentă de valorile concrete ale circuitului RC extern;
- după expirarea intervalului de timp  $\tau$ , monostabilul revine singur (fără nici o comandă externă) în starea stabilă;
- în funcție de modul de tratare a unei comenzi de declanșare apărută pe durata unei stări metastabile, CBM-urile se împart în două categorii:

- **CBM neretrigerabile** – nu se acceptă nici o comandă de declanșare pe durata unei stări metastabile, este ca și cum intrarea de declanșare nu este analizată pe starea  $Q=1$ ;
- **CBM retrigerabile** – se acceptă comenzi de declanșare indiferent de starea monostabilului, circuitul va rămâne în starea metastabilă un interval de timp  $\tau$  măsurat de la ultima comandă de declanșare.

Diferențele care apar în funcționare celor două tipuri de monostabili se pot vedea în figura 4, unde, pentru același semnal de comandă aplicat pe intrarea de declanșare TR+ și aceeași lățime  $\tau$  a impulsului, se prezintă în mod comparativ răspunsul unui CBM neretrigerabil și al unui CBM retrigerabil. Analizând aceste forme de undă se pot face următoarele observații:

- dacă intervalul de timp dintre două comenzi de declanșare este mai mare decât  $\tau$ , cele două monostabile funcționează identic (cazul intervalelor de timp dintre tranzițiile 1,2,3);
- diferențe semnificative în funcționare apar atunci când intervalul de timp dintre comenzile de declanșare este mai mic decât  $\tau$  (cazul intervalelor de timp dintre tranzițiile 3,4,5,6,7);

- pentru CBM-ul neretrigerabil, tranzițiile 4 și 6 nu au nici un efect deoarece, la apariția lor, monostabilul se află în starea metastabilă provocată de tranzițiile 3, respectiv 5;
- pentru CBM-ul retrigerabil, toate tranzițiile au efect, fiecare tranziție inițiază un nou interval de timp  $\tau$ , indiferent dacă intervalul anterior s-a terminat sau nu.
- ieșirea CBM-ului retrigerabil rămâne în starea  $Q=1$  în zona în care semnalul de intrare are tranziții dese și prezintă căderi în zero în zona în care semnalul de intrare are tranziții mai rare - din acest motiv, o aplicație tipică a CBM retrigerabil o constituie detecția lipsei de impuls.

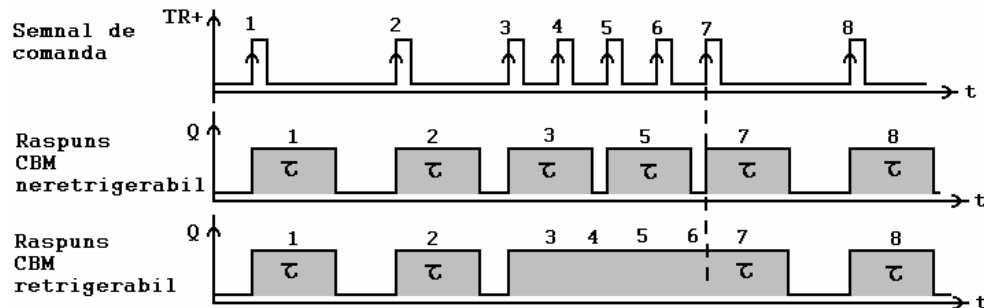


Fig. 4. Răspunsul celor două tipuri de CBM în condiții identice de comandă (același semnal de comandă pe intrarea TR+, același interval de timp  $\tau$ )

Tabelul 1: Exemple reprezentative de CBM

Observații	Simbol
<b>74LS121</b> <ul style="list-style-type: none"> <li>- circuitul conține 1 CBM neretrigerabil;</li> <li>- intrările A1 și A2 sunt sensibile pe tranziția negativă, iar intrarea B la tranziția pozitivă și poate fi comandată și de fronturi lente;</li> <li>- dacă se folosește o intrare de tip A, celelalte intrări de declanșare trebuie legate la +5V;</li> <li>- dacă se folosește intrarea B, cel puțin o intrare A trebuie conectată la masă;</li> <li>- datorită unei rezistențe interne, sunt posibile următoarele moduri de conectare a circuitului RC;</li> </ul> <p><math>\tau = C(R + R_i) \ln 2</math>      <math>\tau = CR_i \ln 2</math>      <math>\tau = CR \ln 2</math></p>	<p>74LS121</p>
<b>74LS123</b> <ul style="list-style-type: none"> <li>- circuitul conține 2 CBM retrigerabile;</li> <li>- B, intrare de declanșare sensibilă pe tranziția pozitivă, dacă nu este folosită intrarea se leagă la +5V;</li> <li>- A, intrare de declanșare sensibilă pe tranziția negativă, dacă nu este folosită intrarea se leagă la masă;</li> <li>- intrarea de ștergere <math>\bar{R}</math>, este activă pe zero logic;</li> <li>- dacă A=0 și B=1, o tranziție pozitivă pe <math>\bar{R}</math> declanșează generarea unui impuls;</li> </ul>	<p>1/2 74LS123</p>
<b>MMC4098</b> <ul style="list-style-type: none"> <li>- circuitul conține 2 CBM ce operează în mod normal în regim retrigerabil, dar poate opera și în modul neretrigerabil dacă se face o legătură externă între <math>\bar{Q}</math> și TR-;</li> <li>- TR+, intrare de declanșare sensibilă pe tranziția pozitivă, dacă nu este folosită intrarea se leagă la +Vss;</li> <li>- TR-, intrare de declanșare sensibilă pe tranziția negativă, dacă nu este folosită intrarea se leagă la VDD;</li> <li>- intrarea de ștergere <math>\bar{R}</math>, este activă pe zero logic;</li> </ul>	<p>1/2 4098</p>

## 2.5. Aplicații cu CBM

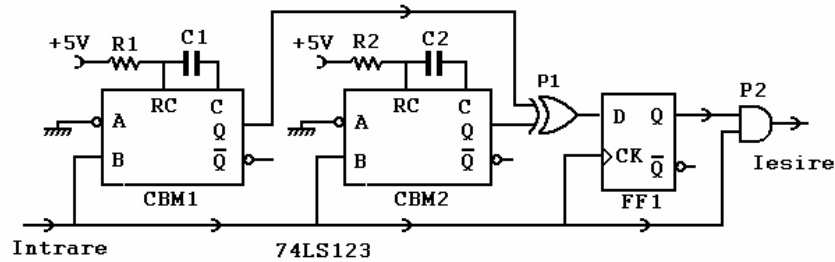
**A. Filtru trece bandă.** Un comportament de filtru trece bandă pentru semnale digitale se obține cu ajutorul schemei din figura 5. Dacă frecvența semnalului de intrare variază, această schemă este concepută astfel încât spre ieșire sunt lăsate să treacă doar impulsurile a căror frecvență este între o valoare minimă  $f_{min}$  și o valoare maximă  $f_{max}$ .

Referitor la funcționarea schemei di figura 5, se pot face următoarele observații:

- ambele monostabile sunt retrigerabile;

- un monostabil este folosit pentru fixarea frecvenței minime și celălalt pentru fixarea frecvenței maxime ce are permisiunea să treacă spre ieșire;
- poarta P1 și bistabilul D au ca rol deschiderea corectă a porții P2 astfel încât să se realizeze funcția dorită;

Fig. 5. Schema electrică a unui filtru trece bandă realizat cu două CBM retrigerabile.



### B. Generator de semnal digital.

O posibilitate de a obține un semnal digital folosind 2 monostabile este prezentată în figura 6.

Se observă că la terminarea impulsului generat de CBM1 se constituie în comandă de declanșare pentru CBM2 iar terminarea impulsului generat de CBM2 devine comandă de declanșare pentru CBM1. Această reacție negativă (conexiunea de la ieșirea CBM2 la intrarea CBM1) menține funcționarea generatorului de semnal.

Un CBM fixează durata de unu iar celălalt durata de zero a semnalului digital de la ieșire. Dacă rezistențele sunt înlocuite cu potențiometre, durata ambelor stări logice poate fi controlată.

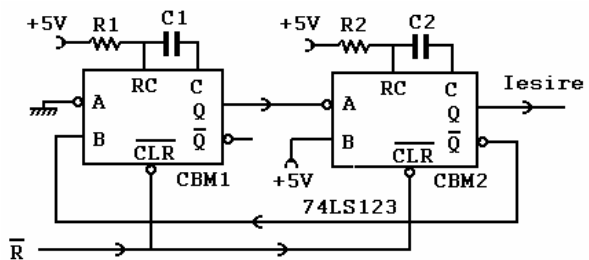


Fig. 6. Generator de semnale digitale realizat cu 2 CBM retrigerabile

### C) Detector pentru impulsuri lipsă

O aplicație tipică a circuitelor monostabile retrigerabile o constituie detecția impulsurilor lipsă dintr-un tren de impulsuri de frecvență cunoscută. Acest gen de aplicație este deosebit de utilă în sistemele de alarmare bazate pe bariere de infraroșu. O barieră în infraroșu se realizează relativ simplu, pe o parte a zonei protejate (spre exemplu o ușă sau o fereastră) se montează o sursă pulsatorie de infraroșu iar pe cealaltă parte un receptor de infraroșu. Amplasarea ansamblului emițător-receptor trebuie făcută astfel încât să existe vizibilitate directă între ele. În momentul în care un corp opac trece prin zona protejată bariera este întreruptă și receptorul nu mai vede unul sau mai multe impulsuri ce au fost transmise de către emițător. Lipsa impulsului/impulsurilor, trebuie să fie detectată de circuitul logic ce prelucrează semnalele provenite de la receptor pentru a declanșa alarma (cineva a pătruns neautorizat în zona protejată). Este evident că, pentru o protecție eficientă, zona protejată trebuie să fie acoperită cu mai multe bariere infraroșii astfel încât să formeze o veritabilă "plasă de păianjen".

O schemă bloc de principiu pentru un sistem de alarmară cu o singură barieră de infraroșu este prezentată în figura 7. Dacă bariera nu este obturată (întreruptă), impulsurile primite de receptor au o cadență suficient de mare pentru a menține ieșirea CBM retrigerabil în starea  $Q=1$ ,  $Q'=0$ .

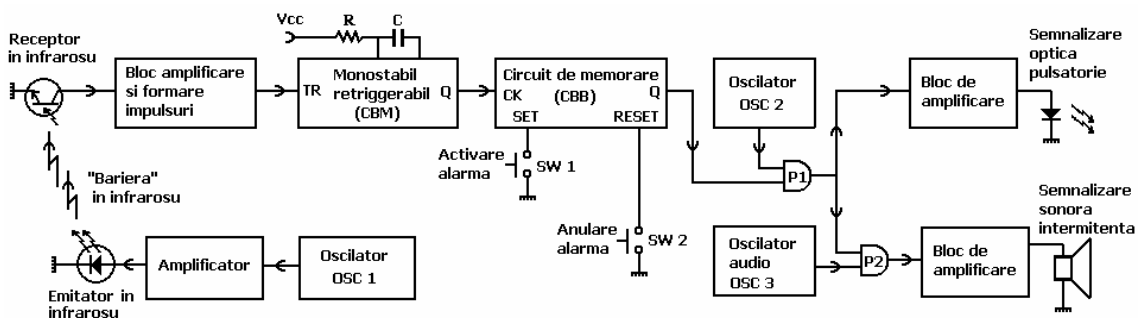


Fig. 7. Schema bloc a unui bariere în infraroșu bazată pe detecția lipsei de impuls

În momentul în care un obiect întrerupe bariera de infraroșu, receptorul nu mai primește impulsuri, în consecință intrarea de declanșare a CBM nu mai este stimulată și, la scurt timp după primul impuls lipsă, va trece în starea  $Q=0$ ,  $Q'=1$ . Noua stare a CBM va fi sesizată și memorată de circuitul bistabil (CBB) în vederea declanșării și menținerii stării de alarmă.

După ce obiectul a trecut prin barieră, la intrarea CBM reapar impulsurile transmise de emițător și ieșirea acestuia va reveni în starea  $Q=1$ ,  $Q'=0$ . Se remarcă astfel că starea CMB este alterată doar atâta timp cât bariera a fost întreruptă, adică atâta timp cât a trecut obiectul prin zona protejată. Din acest motiv, circuitul de avertizare nu poate fi comandat direct de către CBM, pentru că durata de alarmare ar fi foarte scurtă și poate trece neobservată. De aici necesitatea folosirii unui bistabil (CBB), el are rolul de a memora starea de alarmă până când aceasta este anulată prin intermediul acționării SW2; evident SW2 nu este amplasat în zona protejată.



Odată apărută o stare de alarmă, circuitul CBB menține deschisă poarta P1 pentru ca semnalul generat de OSC2 să acționeze circuitul de semnalizare optică intermitentă și, în combinație cu OSC3 și P2, să acționeze semnalizarea acustică intermitentă.

Menționăm că OSC2 și P1 nu sunt strict necesare, semnalizarea acustică și sonoră se face și fără acestea dar, ambele semnalizări nu mai sunt intermitente.

## 2.6. CBM numerice

Un dezavantaj al structurilor logice programabile este acela că nu permit realizarea monostabilelor din cauza faptului că în interiorul acestor structuri nu dispunem de rezistențe și nici de condensatoare.

Dacă într-o anumită aplicație utilizarea CBM este strict necesară, există două soluții de rezolvare a problemei: conectarea circuitului RC în exteriorul structurii logice programabile; respectiv utilizarea de monostabile numerice.

CBM-urile numerice realizează o funcționare similară unui CBM clasic numai că lățimea impulsului generat este egală cu un număr întreg de perioade provenite de la un semnal de ceas a cărui frecvență este cunoscută.

O schemă de principiu pentru un monostabil numeric este prezentată în figura 8. În această figură se observă prezența următoarelor componente:

- un bistabil D activ pe tranziția pozitivă a semnalului de ceas;
- un numărător presetabil configurat pentru numărarea înapoi;
- o poartă AND.

Modul de funcționare al schemei din figura 8 este următorul:

- la apariția unei tranziții pozitive pe intrarea TR+, bistabilul D trece în starea  $Q=1$ , deoarece intrarea de date este menținută în permanență în unu logic;
- starea  $Q=1$ , determină trecerea în unu logic a semnalului  $Q\_CBM$  și deschiderea porții AND pentru impulsurile de pe intrarea de ceas  $ck\_in$ ;
- fiecare impuls ce trece prin poarta AND decrementează cu o unitate starea numărătorului presetabil  $N1$ ;
- în momentul în care numărătorul s-a golit (trece prin starea 0) se activează ieșirea de semnalizare a golirii ( $Bw=0$ );
- activarea ieșirii de terminare numărare, are ca efect resetarea bistabilului (forțare în starea  $Q=0$ ) și încărcarea paralelă a numărătorului cu constanta binară  $M$ ;
- resetarea bistabilului are ca efect trecerea în zero a ieșirii  $Q\_CBM$  și totodată blochează trecerea impulsurilor prin poarta AND, în consecință starea numărătorului nu se mai poate modifica;
- schema rămâne în această stare (numărător încărcat cu constanta  $M$ , bistabil în starea  $Q=0$  și poartă AND blocată) până la o nouă tranziție pozitivă pe intrarea de declanșare  $TR+$ , după apariția acesteia funcționarea se repetă;

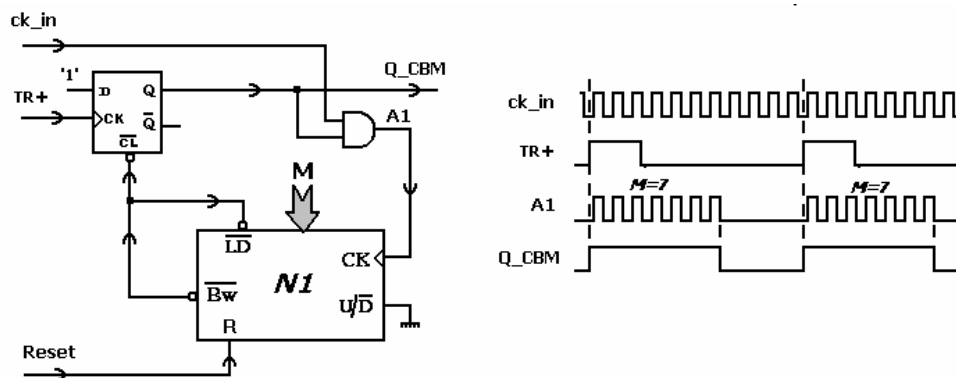


Fig. 8. Schema bloc de principiu și formele de undă pentru un monostabil numeric

### Observații:

- mărirea sau micșorarea duratei impulsului generat de schema din figura 8 se face prin modificarea constantei  $M$ ;
- pentru un control fin al duratei impulsului generat este nevoie de un semnal de ceas cu o perioadă cât mai mică;
- capacitatea numărătorului presetabil trebuie aleasă în funcție de frecvența semnalului de ceas și de lățimea dorită a impulsurilor ce trebuie generate;
- avantajul schemei este dat de faptul că permite programarea numerică a duratei impulsului generat;
- schema este utilă și acolo unde este nevoie să dec număr de impulsuri din semnalul  $ck\_in$  în ritmul tranzițiilor semnalului  $TR+$ , în această situație semnalul de ieșire se culege de la  $A1$ ;
- resetarea (terminarea) unui impuls în curs de derulare se face prin simpla resetare a numărătorului;
- dezavantajul schemei, așa cum este ea prezentată în figura 8, este acela că nu lucrează corect la prima comandă de declanșare;



### 3. Desfășurarea lucrării

#### 3.1. Studiul unor aplicații tipice cu CBM

- A.** Referitor la schema din figura 5, răspundeți la următoarele întrebări:
- Exemplificați cu forme de undă funcționarea schemei;
  - Este strict necesar ca CBM-urile să fie retrigerabile sau pot fi înlocuite și cu CBM neretigerabile ?
  - Are importanță care CBM fixează frecvența minimă de trecere și care pe cea maximă ?
  - Care este legătura dintre perioada semnalului cu frecvență minimă, respectiv maximă, și durata impulsurilor generate de CBM ?
  - Care sunt valorile componentelor R1, C1, R2, C2 astfel încât filtrul să aibă frecvența minimă de 50kHz iar frecvența maximă 100kHz ?
- B.** Referitor la schema din figura 6, răspundeți la următoarele întrebări:
- Exemplificați cu forme de undă funcționarea schemei;
  - Care CBM fixează durata de zero logic și care pe cea de unu logic a semnalului generat?
  - Cum se amorsează funcționarea schemei ?
  - Este strict necesar ca CBM-urile să fie retrigerabile, sau nu are importanță pentru această aplicație ?
  - Semnalul de ieșire poate fi preluat și de la ieșirea primului CBM ? Dacă da, ce diferențe apar față de situația prezentată pe schemă ?
- C.** Referitor la schema din figura 7, răspundeți la următoarele întrebări:
- Dacă frecvența semnalului generat de OSC1 este de 40kHz, în ce gamă poate varia constanta de timp a CBM astfel încât să declanșeze alarma pentru un singur impuls lipsă ? Dar pentru a se declanșa alarma când lipsesc mai mult de 10 impulsuri ?
  - Ce modificări trebuie făcute în schema bloc pentru a putea accepta mai multe bariere de infraroșu ?
  - Care ar fi frecvența acceptabilă pentru OSC2 ? Dar pentru OSC3 ?
  - Poate fi "păcălită" o astfel de schemă (chiar dacă are mai multe bariere de infraroșu) ? Ce măsuri de precauție se pot lua ?

#### 3.1. Studiul CBM numerice cu implementare în structuri CPLD

##### 3.1.1. Descrierea aplicațiilor sub formă de scheme logice în mediul ISE-WebPack

- A.** Folosind simbolurile existente în editorul de scheme al mediului ISE-WebPack, categoria TTL, realizați o implementare a schemei din figura 8 și verificați funcționarea acesteia pe macheta de laborator cu CPLD. Urmăriți pe bareta de LED-uri succesiunea stărilor prin care trece numărătorul și determinați lățime impulsului generat.

##### 3.1.2. Utilizarea limbajului VHDL

- A.** Folosind modul de lucru prezentat la studiul automatelor FSM, concepeți în limbaj VHDL și apoi verificați pe macheta de laborator cu CPLD, un automat sincron care să comande în mod convenabil elementele schemei din figura 8 astfel încât aceasta să funcționeze corect de la prima comandă de declanșare.

Pentru rezolvarea acestei aplicații se recomandă:

- intrările automatului vor fi: intrarea de declanșare TR+, ieșirea de semnalizare a terminării numărării  $\overline{B_w}$  și eventual intrarea de Reset a CBM numeric;
- ieșirile automatului vor fi folosite pentru: comanda încărcării paralele a numărătorului, pentru comanda bistabilului D (setarea și resetarea sa);
- stările automatului: o stare de WAIT în care se așteaptă comanda de declanșare pe intrarea TR+; o stare intermediară INIT în care se face inițializarea schemei (încărcarea numărătorului și setarea bistabilului); o stare PULS în care se generează impulsul de ieșire, în această stare se rămâne până când numărătorul se golește sau până când avem comandă de reset;
- codificarea stărilor se face la alegere;

- B.** Rezolvați problema de la punctul anterior folosind utilitarul StateCAD ce vă permite descrierea funcționării automatului direct sub formă de diagramă de tranziție a stărilor.



Lucrarea nr. 6 : **Studiul registrelor de deplasare****1. Scopul lucrării**

În partea teoretică se face o prezentare succintă a structurii interne a principalelor tipuri de registre, după care sunt prezentate câteva registre realizate în structuri integrate (se pune accent pe semnificația pinilor și pe unele particularități ce apar în funcționarea acestor circuite).

Ca aplicații ale registrelor de deplasare se studiază: realizarea divizoarelor de frecvență; realizarea luminilor dinamice și a generatoarelor de numere pseudo-aleatoare. O parte din aceste aplicații sunt prezentate ca implementări cu structuri integrate distincte sau ca descrieri în limbaj VHDL.

**2. Considerente teoretice****2.1. Introducere**

La nivel de bit, memorarea informației se face cu ajutorul latch-urilor sau a bistabililor. La nivel de cuvânt, memorarea și procesarea informației se face cu ajutorul registrelor. Registrul este un circuit logic secvențial format dintr-o succesiune de  $n$  latch-uri sau bistabili.

Încărcarea unui registru (operația de introducere a cuvântului de  $n$  biți în registru) se poate face în două moduri:

- *serial* – încărcarea se face bit cu bit în ritmul unui semnal de ceas, motiv pentru care timpul de încărcare este egal cu  $n$  perioade de ceas;
- *paralel* – toți biții sunt introduși în același timp, pe tranziția activă a unui semnal de ceas.

Extragerea informației din registru (operație denumită citire) se poate face tot în două moduri:

- *serial* – citirea se face bit cu bit pe  $n$  tranziții active ale unui semnal de ceas;
- *paralel* – toți biții sunt citiți în același timp.

În general, un registru este definit ca o structură liniară de celule de memorare ce prezintă una sau mai multe din următoarele funcții: - acces serial; - acces paralel; - ieșire serială; - ieșire paralelă.

**Clasificarea registrelor:**

Clasificarea registrelor se poate face după mai multe criterii însă cel mai important este cel care indică modul de intrare și de ieșire a informației.

- După tipul celulelor de memorie utilizate:
  - cu latch-uri;
  - cu bistabili;
- După tipul celulelor de memorie utilizate:
  - cu intrare paralelă și ieșire paralelă (parallel in/parallel out);
  - cu intrare serială și ieșire paralelă (serial in/parallel out);
  - cu intrare paralelă și ieșire serială (parallel in/serial out);
  - cu intrare serială și ieșire serială (serial in/serial out);

Registrele cu încărcare serială mai sunt denumite și *registre de deplasare* și, la rândul lor, pot fi împărțite în două categorii: a) registre cu deplasare la dreapta și b) registre bidirecționale.

Registrele prevăzute cu ambele tipuri de intrări (serie și paralel) și ieșire paralelă sunt denumite registre universale.

**2.2. Structura internă a registrelor**

• **Registru cu intrare paralelă și ieșire paralelă (registru paralel/paralel).** O structură de registru cu încărcare paralelă și ieșire paralelă se obține relativ ușor prin utilizarea unui număr convenabil de bistabili de tip D. Un exemplu de registru paralel/paralel pe 4 biți se prezintă în figura 1.

**Funcționare:**

- toate intrările de ceas ale bistabililor se conectează împreună și formează intrarea de comandă a încărcării paralele **CKP**;
- la apariția unei tranziții active pe intrarea **CKP**, starea logică a intrărilor **PI** este copiată în celulele de memorie și va fi păstrată nealterată până la următoarea tranziție activă;
- încărcarea paralelă se poate face numai când pe intrarea **CKP** se aplică tranziție activă, spunem că încărcarea paralelă se face sincron cu semnalul aplicat pe **CKP**;
- intrările de tip D joacă rol de intrări paralele de date;
- ieșirile bistabililor joacă rol de ieșiri paralele de date;
- citirea informației din registru se poate face oricând;
- intrarea de ștergere **MR** (*Master Reset*), este activă pe zero logic și este folosită pentru ștergerea registrului (aducerea în starea  $Q=0$  a tuturor bistabililor din structură);
- ștergerea registrului se poate face oricând prin aducerea intrării **MR** în starea 0, spunem că ștergerea se face asincron față de semnalul aplicat pe intrarea **CKP**;

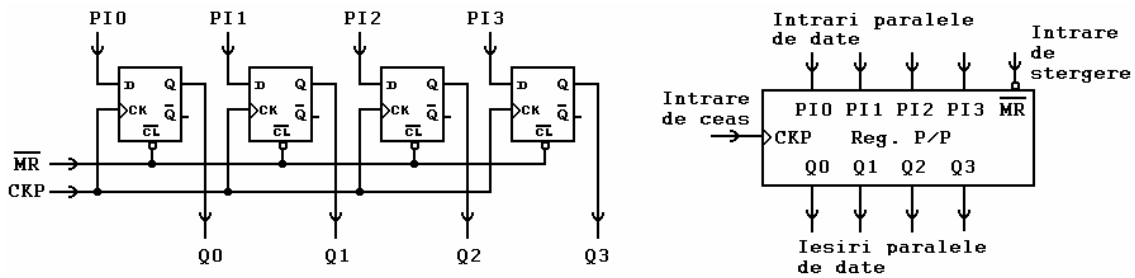


Fig. 1. Structura internă și simbolul pentru un registru paralel/paralel pe 4 biți

• **Registru cu intrare serie și ieșire paralelă (registru serie/paralel).**

O structură posibilă de registru cu încărcare serială se prezintă în figura 2. Analizând schema de conectare se constată că acest tip de registru prezintă ambele tipuri de ieșiri, atât paralelă cât și serială.

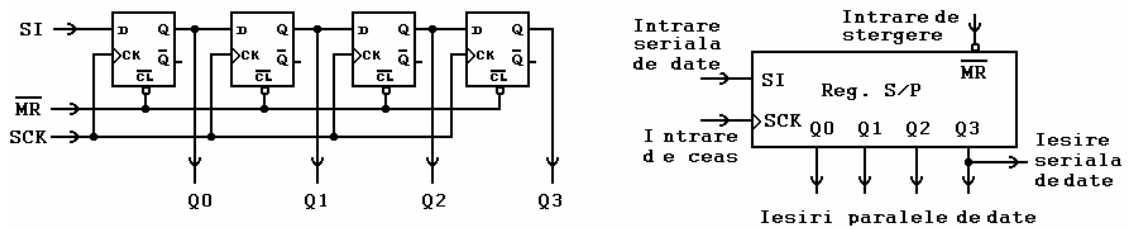


Fig. 2. Structura internă și simbolul pentru un registru serie/paralel pe 4 biți

**Funcționare:**

- intrarea D a primului bistabil joacă rol de intrare serială de date, **SI** (Serial Input);
- toate intrările de ceas ale bistabililor se conectează împreună și formează intrarea de comandă a încărcării seriale **SCK** (Serial Clock);
- la apariția unei tranziții active pe intrarea **SCK**, starea logică a intrării **SI** este copiată în primul bistabil iar conținutul celulelor de memorie este deplasat spre dreapta cu o poziție (de aici și denumirea de registru de deplasare);
- încărcarea acestui registru nu se poate face decât în mod serial și necesită 4 perioade de ceas (în fig. 3 se prezintă modul de încărcare serială a informației 1111 într-un registru ce prezenta inițial informația 0000, pentru a putea urmări deplasarea informației în registru au fost utilizate fonturi diferite pentru fiecare bit de intrare);
- ieșirile bistabililor joacă rol de ieșiri paralele de date;
- ieșirea ultimului bistabil joacă și rol de ieșire serială;
- citirea informației din registru se poate face oricând, independent de **SCK**;
- intrarea de ștergere **MR** (Master Reset), este activă pe zero logic și are o execuție asincronă;

Registreele de deplasare sunt foarte utile în realizarea rapidă a operațiilor de înmulțire/împărțire cu puteri ale lui 2. Se poate arăta că înmulțirea unui număr binar (stocat în registru) cu numărul  $2^p$  înseamnă deplasarea spre stânga cu  $p$  poziții iar împărțirea cu  $2^p$  necesită deplasarea spre dreapta cu  $p$  poziții.

**Exemplu:**

- scriere zecimală:  $5 \times 4 = 20 \Leftrightarrow 5 \times 2^2 = 20$ , deci trebuie efectuată o deplasare spre stânga cu două poziții;
- scriere binară:  $101 \times 100 = 10100$
- conținutul inițial al registrului: 101
- după două deplasări la stânga, cu introducerea de zerouri pe pozițiile rămase libere se obține: **10100**;

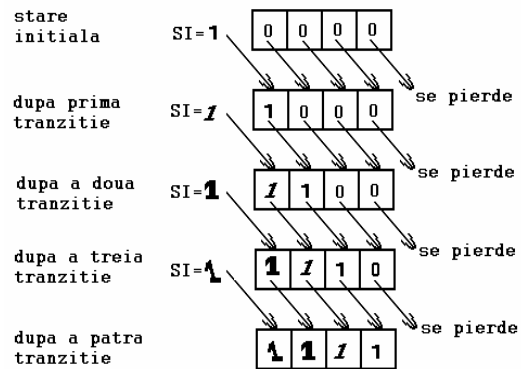


Fig. 3. Exemplu de încărcare serială a unui registru de 4 biți

• **Registru universal.**

Dacă în fața fiecărui bistabil din schema prezentată în figura 2 se introduce un multiplexor cu 2 intrări, se permite conectarea intrării D fie la intrarea de încărcare paralelă, fie la ieșirea bistabilului de pe poziția anterioară. Se obține astfel un registru universal, un registru ce se poate încărcă paralel sau serie și poate fi citit serie sau paralel.

Un exemplu de registru universal pe 4 biți se prezintă în figura 4.

**Funcționare:**

- modul de încărcare a registrului depinde de starea logică a intrării  $S/\bar{P}$ , dacă  $S/\bar{P} = 0$  încărcarea se face paralel iar dacă  $S/\bar{P} = 1$  încărcarea se face serie;
- multiplexoarele 2:1 se comportă ca niște comutatoare ce conectează intrările D fie la PI, fie la ieșirile bistabililor de pe poziția anterioară;
- ambele moduri de încărcare se fac sincron cu semnalul aplicat pe intrarea **CKU**;
- citirea informației din registru se poate face paralel sau serie în mod independent de **CKU**;

Facem precizarea că unele variante de registre universale au intrări separate de ceas pentru încărcarea serie respectiv paralel.

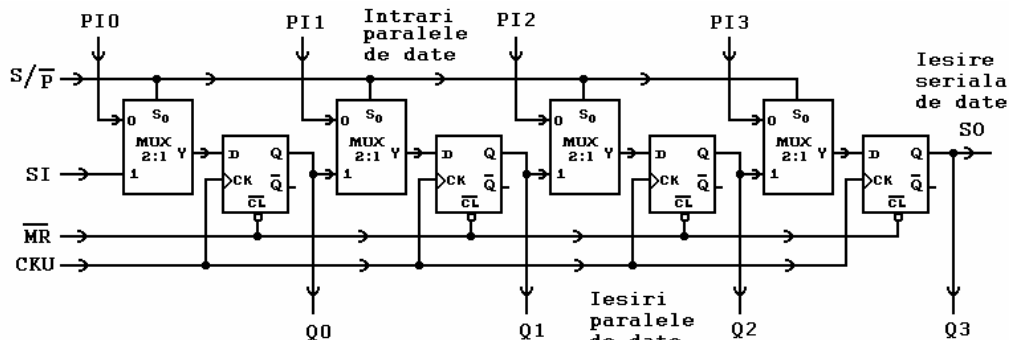


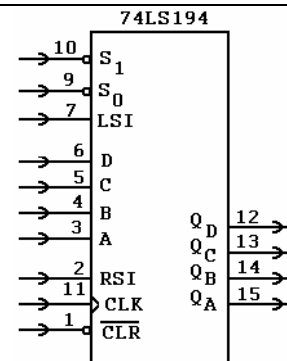
Fig. 4. Structura internă pentru un registru universal pe 4 biți

### 2.3. Exemple de registre realizate în structuri integrate

Observații	Simbol
<b>Registru 74LS171</b> <ul style="list-style-type: none"> <li>- circuitul conține 4 bistabili de tip D, conectați într-o manieră similară celei din figura 1;</li> <li>- intrarea de ștergere (<math>\overline{CLR}</math>), este activă pe zero logic;</li> <li>- ieșirile de date sunt disponibile și sub formă negată;</li> <li>- încărcarea paralelă se face pe tranziția pozitivă a semnalului de ceas;</li> </ul>	
<b>Registru 74LS377</b> <ul style="list-style-type: none"> <li>- circuitul conține 8 bistabili de tip D;</li> <li>- intrarea de validare a accesului informației la celulele de memorie (<math>\overline{EN}</math>), este activă pe zero logic;</li> <li>- încărcarea paralelă se face pe frontul pozitiv al semnalului de ceas numai atunci când <math>\overline{EN} = 0</math>;</li> <li>- dacă <math>\overline{EN} = 1</math>, informația de intrare nu are acces spre celulele de memorare, indiferent de starea semnalului de ceas;</li> </ul>	
<b>Registru 74LS670</b> <ul style="list-style-type: none"> <li>- circuitul dispune de 4 registre de 4 biți fiecare;</li> <li>- ieșirile sunt de tip <i>tristate</i>;</li> <li>- intrarea de validare a scrierii (<math>\overline{WE}</math>) și cea de validare a citirii (<math>\overline{RE}</math>) sunt active pe zero logic;</li> <li>- datorită faptului că avem semnale diferite de validare pentru citire și pentru scriere, este posibilă scrierea unei locații simultan cu citirea alteia;</li> <li>- adresarea registrului se face prin aplicarea unui cod binar pe liniile <math>\overline{RB}</math>, <math>\overline{RA}</math> (dacă este operație de citire) respectiv <math>\overline{WB}</math>, <math>\overline{WA}</math> (dacă este operație de scriere);</li> <li>- circuitul se comportă ca o memorie RAM cu acces dual.</li> </ul>	
<b>Registru 74LS95</b> <ul style="list-style-type: none"> <li>- modul de lucru se alege prin intermediul intrării <math>\overline{CM}</math>:                         <ul style="list-style-type: none"> <li>- <math>\overline{CM} = 0</math>, deplasare spre dreapta în ritmul tranzițiilor negative ale semnalului aplicat pe intrarea <math>\overline{CKS}</math>;</li> <li>- <math>\overline{CM} = 1</math>, încărcare paralelă în ritmul tranzițiilor negative ale semnalului aplicat pe intrarea <math>\overline{CKP}</math>;</li> </ul> </li> <li>- intrarea serială de date este <math>\overline{SI}</math>;</li> <li>- poate fi folosit ca :                         <ul style="list-style-type: none"> <li>- registru cu intrare serie și ieșire paralel;</li> <li>- registru cu intrare serie și ieșire serie;</li> <li>- registru cu intrare paralelă și ieșire paralelă;</li> <li>- registru bidirecțional (necesită conexiuni externe);</li> </ul> </li> </ul>	

### Registru universal 74LS194

- registru universal pe 4 biți;
- modul de lucru se alege prin intermediul intrărilor de selecție  $S_1 S_0$ , după cum urmează:
  - $S_1 S_0 = 00$ , memorare
  - $S_1 S_0 = 01$ , deplasare informație de la  $Q_A \rightarrow Q_D$
  - $S_1 S_0 = 10$ , deplasare informație de la  $Q_D \rightarrow Q_A$
  - $S_1 S_0 = 11$ , încărcare paralelă ( $Q_D Q_C Q_B Q_A = DCBA$ )
- intrarea de ștergere asincronă ( $CLR$ ) este activă pe zero logic;
- intrarea serială de date pentru deplasarea spre stânga este  $LSI$ ;
- intrarea serială de date pentru deplasarea spre dreapta este  $RSI$ ;
- operațiile de deplasare la stânga, deplasare la dreapta și încărcare paralelă sunt efectuate sincron cu semnalul de ceas  $CLK$ , pe tranziția pozitivă;
- pentru comanda de memorare, semnalul de ceas nu are efect asupra funcționării circuitului;



## 2.4. Aplicații ale registrelor

• **Registru în inel.** Am arătat în fig. 3 că încărcarea serie a unui registru de deplasare se face cu pierderea informației inițiale din registru. Dacă se conectează ieșirea serială la intrarea serie se obține un registru în inel care recirculează la nesfârșit o secvență binară. Secvența dorită poate fi încărcată paralel sau serie, înainte de închiderea legăturii dintre ieșirea serială și intrarea serială.

Ca aplicații ale registrelor în inel se pot enumera: realizarea de numărătoare Johnson; realizarea unor divizoare digitale de frecvență; realizarea unor lumini dinamice.

Un exemplu de lumină dinamică este prezentată în figura 5. Funcționarea acestei scheme este relativ simplă:

- Imediat după conectarea sursei de alimentare, circuitul format din  $C2$ ,  $R1$  și  $P2$ , generează o comandă de încărcare paralelă a stărilor logice fixate de către  $K1 \div K4$ .
- Apariția comenzii de încărcare paralelă se explică astfel: inițial  $C2$  este descărcat și intrarea porții  $P2$  se află în zero logic, iar ieșirea în unu logic, deci se obține  $CM=1$ .
- La scurt timp după conectarea sursei de alimentare,  $C2$  se încarcă, la intrarea porții  $P2$  ajunge o tensiune suficient de mare să fie interpretată ca unu logic - fapt ce conduce la apariția unui zero logic la ieșire. Deci,  $CM=0$ , ceea ce înseamnă că registrul primește o comandă de încărcare serie cu deplasare la dreapta.
- Datorită conexiunii dintre  $Q_D$  (ieșire serială) și  $SI$  (intrare serială), informația încărcată paralel va fi recirculată în inel;
- Comutatoarele  $K1 \div K4$  sunt folosite pentru stabilirea secvenței binare, starea lor este citită doar în primele momente de după conectarea sursei de alimentare.
- Dacă secvența binară încărcată paralel a fost 1000 pe bareta de LED-uri se va deplasa un LED aprins, iar dacă secvența a fost 0111, se va deplasa un LED stins;
- semnalul de ceas este asigurat prin intermediul unui oscilator realizat cu un inversor trigger Schmitt (poarta  $P1$ ).

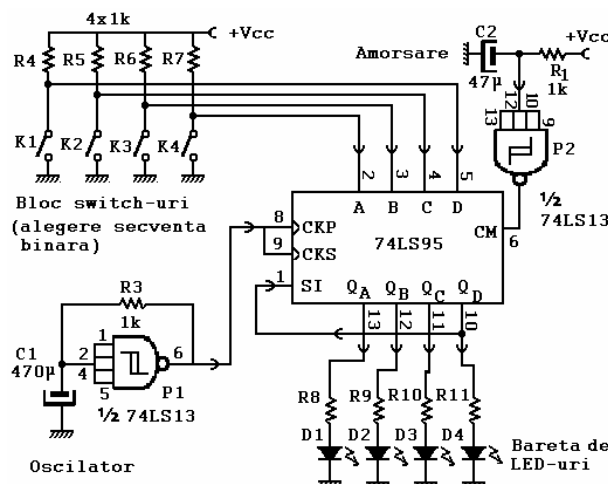


Fig. 5. Schema logică a unei lumini dinamice

• **Generarea de numere pseudoaleatoare.** Acolo unde este nevoie de generarea unor numere aleatoare, o soluție de compromis constă în generarea unor numere pseudoaleatoare, adică în generarea unei secvențe foarte lungi de numere. Deși succesiunea numerelor este mereu aceeași, datorită lungimii mari a secvenței, utilizatorul are senzația că numerele sunt generate aleator.

O posibilitate de generare a unei secvențe de numere pseudoaleatoare constă în utilizarea unui registru de deplasare la care se aplică o reacție liniară (Linear Feedback Shift Register). Spre deosebire de registrul în inel – unde reacția este realizată printr-o legătură între ieșirea serială de date și intrarea serială de date -, în cazul generatoarelor de numere pseudoaleatoare, reacția este asigurată printr-o poartă XOR conectată ca în figura 6.

Pentru fiecare tranziție activă a semnalului de ceas, starea registrului se modifică pe de o parte datorită faptului că informația este deplasată la dreapta iar pe de altă parte datorită faptului că intrarea serială de date primește informația  $Q_n \oplus Q_m$ .

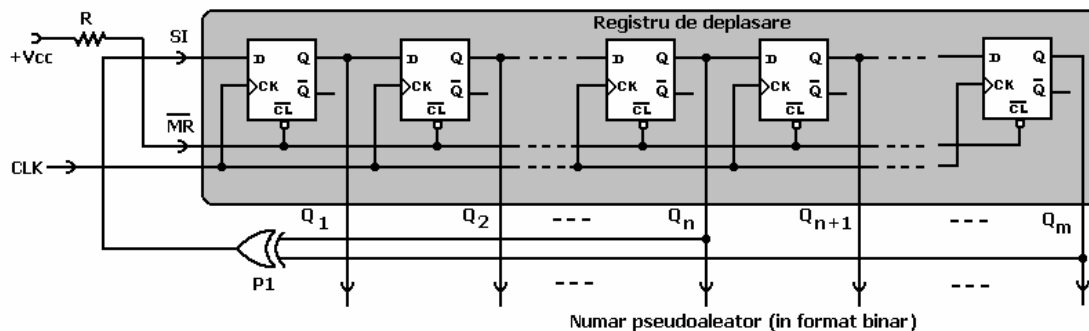


Fig. 6. Schema de principiu a unui generator de numere pseudoaleatoare

Din analiza tabelului 1 (unde se prezintă legătura dintre poziția reacției ( $n$ ), lungimea registrului ( $m$ ) și numărul de stări distincte), se observă că pentru dimensiuni mari ale registrului se obțin secvențe lungi și astfel se poate crea aparența de generare aleatoare a numerelor.

Tabelul 1

Lungime registru (m)	Poziție reacție (n)	Numărul de stări distincte	Lungime registru (m)	Poziție reacție (n)	Numărul de stări distincte
4	3	16	20	17	1.048.475
5	3	31	21	19	2.097.151
6	5	63	22	21	4.194.303
7	6	127	23	18	8.388.607
9	5	511	25	22	33.554.431
10	7	1.023	28	25	268.435.455
11	9	2.047	29	27	536.870.911
15	14	32.767	31	28	2.147.483.647

• **Conversia între formatele de reprezentare a informației digitale.** Informația vehiculată în sistemele digitale se face fie în format serial fie în format paralel. Trecerea de la un format la altul se face cu ajutorul registrelor de deplasare. Modul concret de utilizare se va studia într-o altă lucrare de laborator.

## 2.5. Descrierea registrelor cu ajutorul limbajului VHDL

♦ **Exemplul 1:** Descrierea în limbaj VHDL a unui registru pe 8 biți cu încărcare paralelă pe frontul pozitiv al semnalului de ceas. Intrarea de ștergere a registrului este activă pe nivelul de unu logic.

Observații	Codul VHDL
Secțiune dedicată includerii de librării.	-- Registru pe 8 biți cu încărcare paralelă <b>library</b> IEEE; <b>use</b> IEEE.std_logic_1164.all;
Secțiune dedicată descrierii entității. În cazul de față: - nume entitate este: <b>reg_8</b> ; - intrarea de ceas: <b>clk</b> ; - comandă încărcare paralelă: <b>ld</b> ; - intrări paralele de date: <b>din</b> ; - ieșiri paralele de date: <b>dout</b> ;	<b>entity</b> reg8 <b>is</b> <b>port</b> ( clk: <b>in</b> std_logic; reset: <b>in</b> std_logic; ld: <b>in</b> std_logic; din: <b>in</b> std_logic_vector(7 downto 0); dout: <b>out</b> std_logic_vector(7 downto 0) ); <b>end</b> reg8;
Secțiune dedicată descrierii arhitecturii. În cazul de față: - numele arhitecturii este: <b>arh8</b> ; - arhitectura este asociată cu entitatea: <b>reg8</b> ;  <b>Observații:</b> - procesul <b>p1</b> este sensibil la intrarea de ceas <b>clk</b> ; - resetul este activ pe unu logic; - declarația <b>clk'event and clk='1'</b> este folosită pentru a specifica tranziția pozitivă a semnalului de ceas;	<b>architecture</b> arh8 <b>of</b> reg8 <b>is</b> <b>begin</b> <b>process</b> (clk) <b>begin</b> <b>if</b> (reset = '1') <b>then</b> dout <= "00000000"; <b>elsif</b> (clk'event and clk='1') <b>then</b> <b>if</b> (ld = '1') <b>then</b> dout <= din; <b>end if</b> ; <b>end if</b> ; <b>end process</b> p1; <b>end</b> arh8;

♦ **Exemplul 2:** Descrierea în limbaj VHDL a unui registru cu intrare serie și ieșire paralelă pe 8 biți. Intrarea de ceas este activă pe tranziția pozitivă. Circuitul nu are intrare de ștergere.

Observații	Codul VHDL
Secțiune dedicată includerii de librării	-- Exemplu de registru de deplasare <b>library</b> IEEE; <b>use</b> IEEE.std_logic_1164.all;
Secțiune dedicată descrierii entității. În cazul de față: - nume entitate este: <b>reg_8</b> ; - intrarea de ceas: <b>clk</b> ; - intrarea seriala de date: <b>si</b> ; - ieșirile de date: <b>dq</b> ;	<b>entity</b> reg_8 <b>is</b> <b>port</b> ( si, clk : <b>in</b> std_logic; dq : <b>out</b> std_logic_vector(7 downto 0)); <b>end</b> reg_8;
Secțiune dedicată descrierii arhitecturii. În cazul de față: - numele arhitecturii este: <b>arh_reg</b> ; - arhitectura este asociată cu entitatea: <b>reg_8</b> ; - descrierea funcționării se face cu un proces ce este sensibil la semnalul de ceas <b>clk</b> ; <b>Observații:</b> Deplasarea se face pe frontul crescător al semnalului de ceas; Data de pe intrarea serie este copiată în prima celulă a registrului de memorie.	<b>architecture</b> arh_reg <b>of</b> reg_8 <b>is</b> <b>begin</b> <b>process</b> ( clk ) <b>begin</b> <b>if</b> (( clk 'event ) <b>and</b> ( clk='1' )) <b>then</b> dq( 7 downto 1 ) <= dq( 6 downto 0 ); dq(0) <= si ; -- descriere alternativa -- dq <= dq( 6 downto 0 ) & si; <b>end if</b> ; <b>end process</b> ; <b>end</b> arh_reg ;

◆ **Exemplul 3:** Descrierea în limbaj VHDL a unui registru pe 4 biți cu încărcare paralelă și posibilitatea de deplasare a informației spre stânga sau spre dreapta. Metoda folosită în acest exemplu este preferabilă celei anterioare.

Observații	Codul VHDL
<b>Package</b> -ul conține elemente ce trebuie recunoscute în mai multe proiecte. Elementele dintr-un pachet sunt recunoscute într-un proiect dacă se face apel la instrucțiunea <b>use</b> . În cazul de față se definește un pachet pentru a specifica formatul <b>bit4</b> .	<b>library</b> IEEE; <b>use</b> IEEE.std_logic_1164.all; <b>package</b> reg_types <b>is</b> <b>subtype</b> bit4 <b>is</b> std_logic_vector(3 downto 0); <b>end</b> reg_types;
Secțiune dedicată includerii de librării. Se remarcă faptul că se include și pachetul descris mai sus.	-- Exemplu de registru pe 4 biți <b>library</b> IEEE; <b>use</b> IEEE.std_logic_1164.all; <b>use</b> WORK.reg_types.all;
Secțiune dedicată descrierii entității. În cazul de față: - nume entitate este: <b>reg_A</b> ; - intrarea de ceas: <b>clk</b> ; - comandă încărcare paralelă: <b>load</b> ; - intrare pentru specificarea sensului de deplasare a informației din registru: <b>shift_sens</b> ; - intrări paralele de date: <b>din</b> ; - ieșiri paralele de date: <b>dout</b> ;	<b>entity</b> reg_A <b>is</b> <b>port</b> ( clk, shift_sens, load : <b>in</b> std_logic; din : <b>in</b> bit4; dout : <b>inout</b> bit4); <b>end</b> nr_A;
Secțiune dedicată descrierii arhitecturii. În cazul de față: - numele arhitecturii este: <b>arh_A</b> ; - arhitectura este asociată cu entitatea: <b>reg_A</b> ; <b>Observații:</b> - procesul <b>pr_1</b> este responsabil de menținerea stării registrului; - procesul: <b>pr_2</b> este folosit pentru transferul stării registrului spre ieșirile circuitului; - pentru descrierea deplasării spre dreapta, în locul declarațiilor: temp_val (2 downto 0) <= temp_val (3 downto 1); temp_val (3) = '0' ; se poate folosi temp_val <= '0' & temp_val (3 downto 1); - pentru descrierea deplasării spre stânga, în locul declarațiilor: temp_val (3 downto 1) <= temp_val (2 downto 0); temp_val (0) = '0' ; se poate folosi temp_val <= temp_val (2 downto 0) & '0' ;	<b>architecture</b> arh_A <b>of</b> reg_A <b>is</b> <b>signal</b> temp_val: bit4; <b>begin</b> <b>pr_1: process</b> (shift_sens, load, din, dout) <b>begin</b> <b>if</b> load = '1' <b>then</b> temp_val <= din ; <b>elsif</b> shift_sens = '1' <b>then</b> -- deplasare spre dreapta temp_val (2 downto 0) <= temp_val (3 downto 1); temp_val (3) = '0' ; <b>else</b> -- deplasare spre stanga temp_val (3 downto 1) <= temp_val (2 downto 0); temp_val (0) <= '0' ; <b>end if</b> ; <b>end process</b> pr_1 ; <b>pr_2: process</b> <b>begin</b> <b>wait until</b> clk 'event <b>and</b> clk ='1' dout <= temp_val ; <b>end process</b> pr_2 ; <b>end</b> arh_A ;





### 3. Desfășurarea lucrării

#### 3.1. Studiul registrelor realizate în structuri integrate.

- A.** Cum trebuie conectat un circuit 74LS95 pentru a obține un registru de deplasare bidirecțional, cu intrare serie și ieșire paralelă ?
- B.** Referitor la schema din figura 5, răspundeți la următoarele întrebări:
- Pentru a obține efectul de "lumină curgătoare" este nevoie de mai multe LED-uri așezate în linie (bareta luminoasă). Dacă menținem capacitatea registrului la 4 biți cum trebuie conectate din punct de vedere electric LED-urile pentru a obține o bareta luminoasă de 32 poziții ?
  - Care este informația ce trebuie încărcată paralel dacă dorim deplasarea unui singur LED stins ? Dar pentru un singur LED aprins ?
  - Asupra cărei componente trebuie acționat pentru a reduce viteza de curgere ?
  - Dacă schema este pornită și deplasează un LED aprins, ce trebuie făcut ca ea să deplaseze un LED stins ?
  - Care va fi efectul vizibil pe bareta de LED-uri dacă legătura dintre Qd și SI se face printr-un inversor logic ?
- C.** Folosind registre de tip 74LS95 și alte circuite auxiliare necesare, concepeți o schemă de lumină dinamică cu 8 LED-uri știind că LED-ul aprins trebuie să parcurgă următorul ciclu: LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED6, LED5, LED4, LED3, LED2, LED1, LED0, LED1 ..., cu alte cuvinte sensul de deplasare se schimbă după ce LED-ul aprins a ajuns la una din extremitățile baretei.

#### 3.2. Utilizarea ISE WebPack pentru descrierea aplicațiilor sub formă de scheme logice

- A.** Folosind facilitatea ISE-WebPack prin care se permite descrierea proiectelor la nivel de schemă logică se cere implementarea și testarea schemelor din figurile 1, 2 și 4 pe macheta de laborator cu CPLD;
- B.** Folosind facilitatea ISE-WebPack prin care se permite descrierea proiectelor la nivel de schemă logică se cere implementarea și testarea unui generator de numere pseudoaleatoare (vezi figura 6) cu  $m=4$  și  $n=2$ .

#### 3.3. Utilizarea limbajul VHDL

- A.** Implementați și verificați pe macheta de laborator cu CPLD, descrierile în limbaj VHDL prezentate ca exemple. Cum se execută funcția de reset la registrul din exemplul 1?
- B.** Modificați descrierea VHDL de la exemplul 2 pentru adăugarea unei intrări de ștergere cu execuție asincronă față de semnalul de ceas.
- C.** În exemplul 3, deplasarea informației ce a fost încărcată paralel se face cu pierderea treptată (câte un bit la fiecare perioadă a semnalului de ceas) a datelor. Modificați descrierea VHDL de la exemplul 3 astfel încât informația încărcată paralel să fie rotită (recirculată) la nesfârșit.
- D.** Concepeți o descriere comportamentală pentru circuitul 74LS194 după care verificați corectitudinea ei pe macheta de laborator cu CPLD.
- E.** Concepeți o descriere în limbaj VHDL și implementați pe macheta de laborator cu CPLD, o lumină dinamică pe 8 biți care să funcționeze similar cu cea din figura 5.
- F.** Concepeți o descriere în limbaj VHDL și implementați pe macheta de laborator cu CPLD, o lumină dinamică cu 8 LED-uri știind că LED-ul aprins trebuie să parcurgă următorul ciclu: ..., LED0, LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED6, LED5, LED4, LED3, LED2, LED1, LED0, LED1 ..., cu alte cuvinte LED-ul aprins se deplasează de la stânga la dreapta și când atinge capătul baretei se deplasează în sens invers.
- G.** Concepeți o descriere în limbaj VHDL și implementați pe macheta de laborator cu CPLD, o lumină dinamică cu 8 LED-uri pentru a obține următorul efect: bareta se aprinde de la stânga la dreapta - dând efectul de umplere, iar după ce întreaga bareta este aprinsă se stinge de la dreapta la stânga - dând efectul de golire.
- H.** Concepeți o descriere în limbaj VHDL și implementați pe macheta de laborator cu CPLD, un generator de numere pseudoaleatoare cu o secvență de cel puțin 63 numere. Afișarea numerelor se va face în cod binar pe bareta de LED-uri sau în zecimal pe două afișaje cu 7 segmente.

### 4. Indicații privind modul de lucru

Pentru fiecare aplicație este necesară deschiderea unui nou proiect după metodologia prezentată într-o lucrare de început.

Toate aplicațiile din această lucrare necesită doar un singur fișier sursă (fie schemă logică, fie sursă VHDL) și un singur fișier de constrângeri .

Referitor la conectarea intrărilor și a ieșirilor din circuit facem următoarele precizări:

- pentru comanda intrărilor de date se vor folosi switch-urile SW1, SW2, SW3, SW4;
  - pentru afișarea informației din registru se vor folosi LED-urile LED1, LED2, LED3 și LED4;
  - pentru intrarea de ceas se folosește BTN1, pentru intrarea de reset BTN2 iar pentru intrarea de încărcare paralelă BTN3;
  - pentru intrarea de comandă a sensului de deplasare se va utiliza SW8;
  - pentru intrarea serială de date se va utiliza BTN8;
- Toate aceste precizări trebuie să se regăsească în conținutul fișierului de constrângeri, alcătuirea sa, sperăm, nu mai ridică nici o problemă.*

Referitor la obținerea semnalului de comandă a intrărilor de ceas, facem următoarele precizări:

- Pentru a crea un efect perceptibil de lumină dinamică este nevoie ca frecvența semnalului ce comandă deplasarea informației din registru să fie sub 2Hz. Dacă frecvența este prea mare, vom percepe toată bareta aprinsă.
- Pentru comanda intrării de ceas se folosește oscilatorul de pe macheta de laborator cu CPLD. Deoarece frecvența acestuia este de 25,1758MHz, el trebuie mai întâi divizat în frecvență și abia apoi folosit pentru comanda intrării de ceas a registrului de deplasare. Dacă ne propunem un semnal cu frecvența de 1Hz, constanta de divizare este egală cu 25.175.000.
- Din punct de vedere al sintezei este mai bine să folosim o constantă de divizare ce reprezintă o putere a lui 2, spre exemplu  $2^{23} = 8.388.608$ , ceea ce înseamnă o frecvență de ieșire de cca. 3 Hz, valoare acceptabilă pentru aplicațiile propuse.
- În fișierul VHDL, pe lângă descrierea propriu-zisă a funcționării, se va introduce un proces pentru divizarea semnalului preluat de la oscilatorul machetei.
- Modul de intercalare al acestui process cu restul aplicației se prezintă pe coloana alăturată. Cea mai mare divizare în frecvență se obține dacă semnalul de atac al registrului **clk\_reg** este preluat de la cel mai semnificativ bit al semnalului **cnt**. Pentru aceasta este necesar să facem o declarație de forma:
 

```
clk_reg <= cnt(22);
```
- Starea registrului de deplasare se afișează pe bareta de LED-uri de pe macheta de laborator cu CPLD.

```
-- Exemplu de structura
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity lumina_dinamica is
port (
    osc_25M : in std_logic;
    -- se declara restul de intrari si
    -- iesiri necesare
);
end lumina_dinamica;

architecture arh_2 of lumina_dinamica is
    signal cnt: std_logic_vector(22 downto 0);
    signal clk_reg: std_logic;
    -- alte declaratii,daca este cazul
begin
    -- proces pentru divizare ceas
    divizare: process (osc_25M)
    begin
        if (osc_25M 'event and osc_25M='1') then
            cnt <= cnt + 1;
        end if;
    end process divizare;
    clk_reg <= cnt(22);

    -- alte procese sau declaratii concurente
    -- pentru descrierea aplicatiilor cerute
    -- în lucrare

end arh_2;
```



## Lucrarea nr. 7: **Studiul numărătoarelor asincrone**

### 1. Scopul lucrării

Lucrarea are ca scop familiarizarea studentului cu numărătoarele asincrone. Pentru aceasta, se pune accent pe: înțelegerea structurii interne și a modului de funcționare; prezentarea avantajelor și dezavantajelor ce decurg din structura internă; prezentarea numărătoarelor asincrone realizate în structură integrată ce sunt disponibile pe piață (semnificația pinilor, particularitățile în funcționare, modul de cascaderare și modul de operare cu acestea); prezentarea modului de implementare în structuri programabile de tip CPLD folosind editorul de scheme din mediul ISE-WebPack; utilizarea numărătoarelor în realizarea divizoarelor digitale de frecvență.

### 2. Considerente teoretice

#### 2.1. Clasificarea numărătoarelor

Numărătoarele sunt structuri secvențiale care pot parcurge un număr de stări distincte ca urmare a aplicării unor impulsuri la intrarea de numărare, intrare ce este de regulă notată prin CK, CP sau  $\Phi$ . Numărul stărilor distincte este dependent de numărul bistabililor din structură și de modalitatea de codificare a stărilor.

Cele mai importante criterii de clasificare a numărătoarelor sunt:

- După sensul de numărare:
  - înainte (sens direct);
  - înapoi (sens invers);
  - bidirecționale sau reversibile (pot număra fie înainte fie înapoi)
    - cu două intrări de numărare;
    - cu o intrare de numărare și o intrare de sens.
- După codul utilizat:
  - numărătoare în cod binar natural;
  - numărătoare în cod BCD;
  - numărătoare Johnson (numărătoare cu ieșirile decodate).
- După modul de comutare a bistabililor:
  - asincrone, semnalul de ceas comandă în mod direct numai primul bistabil, după care ieșirea unui bistabil comandă pe următorul;
  - sincrone, fiecare bistabil este comandat direct de către semnalul de ceas.

#### 2.2. Numărătoare binare asincrone

##### Caracteristici generale:

- Prezintă cele mai simple scheme de realizare, motiv pentru care prețul lor este redus.
- Sunt realizate prin cascaderare unor celule divizoare cu 2 a frecvenței. Aceste celule provin din configurarea corespunzătoare a unor anumite tipuri de bistabili (vezi fig. 1).
- Numărul stărilor este:  $N \leq 2^{NUMAR\_BISTABILI}$ .
- Semnalul de numărare se aplică numai primului bistabil al structurii. Comanda celorlalți bistabili se face dinspre bistabilul  $p$  spre bistabilul  $p + 1$ .
- Ca o consecință a timpului de propagare prin bistabili, cât și a modului de conectare a acestora în structura numărătorului asincron, trecerea de la numărul  $i$  la  $i + 1$  nu se face direct, așa cum ar fi de dorit, ci se face printr-o serie de stări intermediare de scurtă durată. Spre exemplu, trecerea de la numărul binar **1111** la **0000** se face prin lanțul de stări intermediare **1111** → **1110** → **1100** → **1000** → **0000**.
- Numărul stărilor intermediare este cu atât mai mare cu cât numărătorul bistabililor din structură este mai mare.
- Stările intermediare sunt fenomene nedorite de care trebuie să se țină seama la proiectarea circuitelor comandate de către ieșirile numărătorului. Dacă ieșirile numărătorului sunt conectate, spre exemplu, la un circuit decodor acesta generează "ciocuri" la anumite ieșiri pe durata stărilor intermediare ale numărătorului.
- Aceste structuri sunt utilizate îndeosebi la realizarea integrată a numărătoarelor de capacități mari: 10, 12, 14 sau chiar 20 de biți.

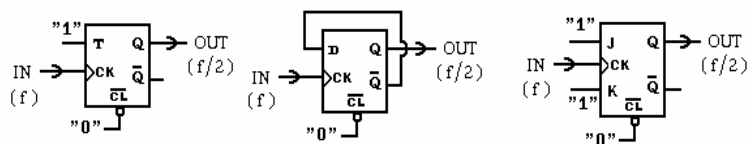


Fig. 1. Celule de divizarea cu doi a frecvenței unui semnal digital

**Exemplu:** În fig. 2 se prezintă schema de realizare și simbolul unui numărător binar asincron pe 4 biți constituit din bistabili JK activi pe tranziția negativă a semnalului de ceas. Structuri asemănătoare se pot realiza și cu bistabili de alt tip configurați ca în fig.1.

Pentru acest caz, avem următoarea semnificație a semnalelor:

- $\Phi$  intrare de numărare, sensibilă (activă) la tranziția negativă a semnalului aplicat;
- $\overline{R}$  intrare de RESET, de aducere la zero a numărătorului, activă pe nivelul LOW al semnalului aplicat acestei intrări;

- Qd, Qb, Qc, Qa - ieșirile de numărare a căror stare logică, citite în ordinea indicată, indică în cod binar starea numărătorului (numărul impulsurilor acumulate). Ieșirea Qd, indică cel mai semnificativ bit al stării numărătorului iar Qa pe cel mai puțin semnificativ.

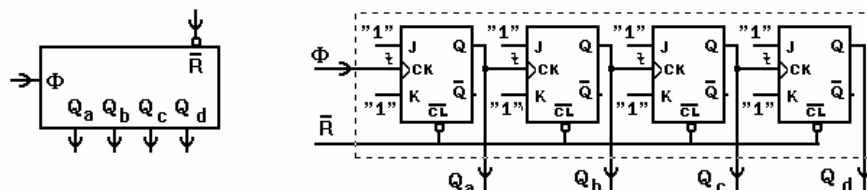


Fig. 2. Exemplu de numărător binar asincron pe 4 biți realizat cu bistabili JK

**Observații:**

- După starea maximă, în cazul de față 1111, la următorul impuls numărătorul trece de la sine în starea zero și continuă numărarea.
- Frecvența semnalelor de ieșire scade la jumătate după fiecare bistabil.
- Numărătorul poate fi utilizat și ca divizor de frecvență a semnalelor digitale.
- În mod natural (fără conexiuni suplimentare, altele decât cele necesare cascaderii), factori de divizare în frecvență ai semnalului de intrare sunt puteri ale lui doi.
- În fig. 4 se arată, pentru situația cea mai defavorabilă, modul de apariție a stărilor intermediare datorită comenzii succesive a bistabililor și timpului de întârziere  $\tau$  al acestora.
- Numărătoarele asincrone lucrează corect dacă intervalul de timp dintre două tranziții active ale semnalului de ceas este mai mare decât suma întârzierilor introduse de lanțul bistabililor.

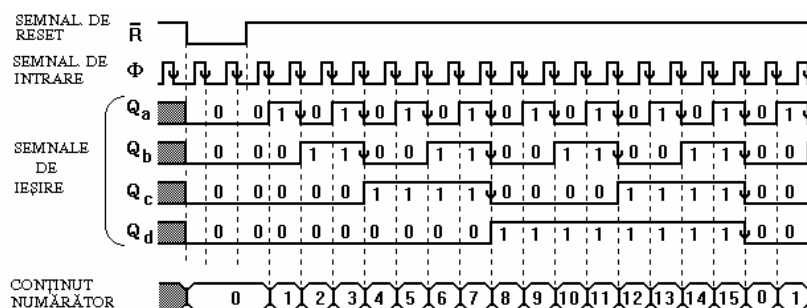


Fig. 3. Forma ideală a semnalelor pentru un numărător binar pe 4 biți

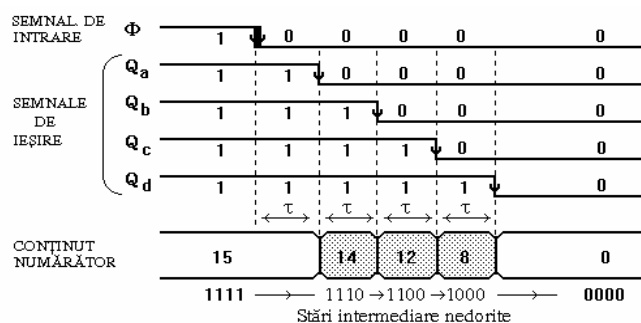


Fig. 4. Exemplificarea modului de apariție a stărilor intermediare nedorite – cazul unui numărător asincron pe 4 biți care trece din starea 15 → 0. Stările nedorite (1110, 1100 și 1000) sunt de scurtă durată însă pot fi sesizate de alte circuite dintr-un sistem digital mai amplu.

### 2.3. Prezentarea structurilor asincrone de numărare realizate în circuite integrate MSI

În această secțiune se prezintă câteva tipuri de numărătoare asincrone, mai des utilizate, ce sunt disponibile sub formă de circuite integrate. Pentru fiecare circuit se prezintă semnificația pinilor și particularitățile funcționale ale acestuia.

Din analiza foilor de catalog, în dorința de a fi cât mai versatile, se constată că o serie de numărătoare sunt prevăzute cu facilități suplimentare. O parte a acestor funcții se prezintă pe scurt în continuare pentru a facilita înțelegerea mai ușoară a circuitelor în momentul prezentării lor.

• **Funcția de reversibilitate**

Numărătoarele reversibile pot realiza numărarea impulsurilor de intrare fie în sens crescător, fie în sens descrescător. Aceste circuite se împart în două categorii: a) circuite cu două intrări de ceas; b) circuite cu o intrare de ceas și o intrare de comandă a sensului de numărare. Pentru primul caz, cele două intrări de ceas sunt denumite COUNT UP respectiv COUNT DOWN. În cazul doi, intrarea de ceas este denumită CLOCK iar cea de comandă UP / DOWN.

### • Funcția de încărcare paralelă (PRESET)

Numărătoarele ce dispun de această facilități, prezintă avantajul că pot starta numărarea dintr-o stare particulară, ce se încarcă în prealabil în mod paralel. Pentru a permite acest lucru, circuitul este prevăzut cu:

- intrări de date, notate  $P_0, P_1, \dots$ , prin intermediul cărora se specifică constanta binară ce trebuie încărcată paralel;
- o intrare de control prin intermediul căreia se comandă introducerea în bistabili a datelor de pe intrările paralele PI.

Intrarea poate fi întâlnită sub diverse denumiri: LOAD, LOAD ENABLE, PARALLEL LOAD etc.

Facem precizarea că funcția de încărcare paralelă poate fi executată sincron sau asincron față de semnalul de ceas aplicat numărătorului. La numărătoarele cu presetare asincronă, încărcarea se execută odată cu activarea intrării LD și nu ține cont de starea logică a semnalului de ceas. Pentru numărătoarele cu presetare sincronă, încărcarea efectivă se execută la prima tranziție activă a semnalului de ceas care apare după activarea intrării LD.

Trebuie reținut că, pe durata activării funcției de încărcare paralelă, procesul de numărare este inhibat.

### • Funcția de ștergere a numărătorului (RESET)

Pentru aducerea în starea zero a unui numărător, marea majoritate a numărătoarelor sunt prevăzute cu o intrare denumită RESET. Funcție de circuitul utilizat, funcția de ștergere poate fi executată sincron, sau asincron, față de semnalul de ceas aplicat numărătorului. Pentru numărătoarele cu resetare asincronă, ștergerea se face la momentul activării intrării RESET, deci nesincronizat cu semnalul de ceas. În celălalt caz, numărătoare cu resetare sincronă, ștergerea se face efectiv la prima tranziție activă a semnalului de ceas ce apare după activarea intrării de RESET.

Unele firme, pentru a face distincție între cele două modalități de ștergere, notează prin: MASTER RESET resetul asincron, și prin SYNCHRONOUS RESET pe cel sincron.

### • Funcția de semnalizarea a terminării numărării (TERMINAL COUNT)

În aplicații precum: extinderea capacității de numărare, realizarea divizoarelor programabile de frecvență etc, este foarte utilă semnalizarea în exterior a momentelor de umplere, sau după caz, de golire a numărătoarelor. Prin umplere se înțelege trecerea numărătorului prin starea sa maximă, iar prin golire trecerea sa prin zero.

De regulă, numărătoarele reversibile au două ieșiri destinate acestui scop: a) TERMINAL COUNT UP sau CARRY, pentru semnalizarea umplerii; b) TERMINAL COUNT DOWN sau BORROW, pentru semnalizarea golirii.

Numărătoarele unidirecționale prezintă o ieșire TERMINAL COUNT prin care semnalizează trecerea prin starea maximă a acestora.

## ◆ Exemple de numărătoare asincrone realizate în structuri integrate

### • Circuitul 74 LS 93 - numărător binar asincron pe 4 biți

Circuitul este activ pe tranziția negativă a semnalului de ceas.

Este organizat în două secțiuni: prima realizează o divizare cu 2 a frecvenței de intrare, iar a doua o divizare cu 8.

Un numărător binar pe 4 biți, cu intrarea pe CP0, se obține prin realizarea unei conexiuni externe între Q0 și CP1.

Ștergerea numărătorului se execută asincron prin  $MR1 = MR2 = 1$ .

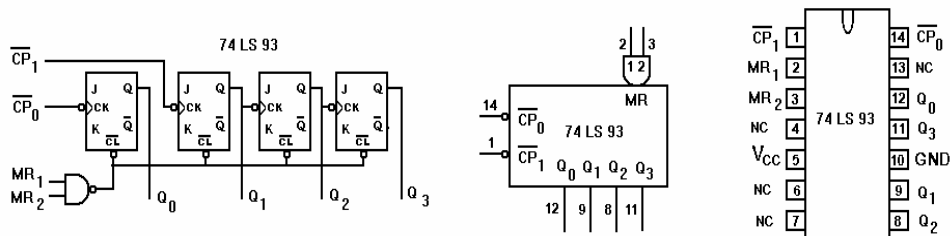


Fig. 5. Circuitul 74LS93: schemă logică, simbol, pinout

### • Circuitul 74 LS 92 - numărător modulo 12 (divizor cu 12)

Circuitul este activ pe tranziția negativă a semnalului de ceas.

Este organizat în două secțiuni: prima realizează o divizare cu 2 a frecvenței de intrare, iar a doua o divizare cu 6.

Un divizor cu 12, cu intrarea pe CP0, se obține prin realizarea unei conexiuni externe între Q0 și CP1.

Ștergerea numărătorului se execută în mod asincron prin  $MR1 = MR2 = 1$ .

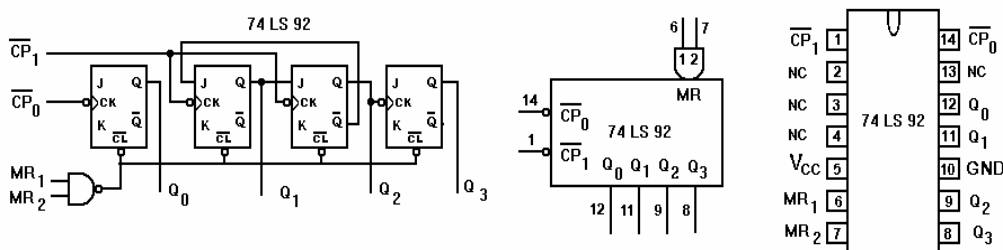


Fig. 6. Circuitul 74LS 92: schemă logică, simbol, pinout

### • Circuitul 74 LS 90 - numărător BCD asincron

Circuitul este activ pe tranziția negativă a semnalului de ceas.

Este organizat în două secțiuni: prima realizează o divizare cu 2 a frecvenței de intrare, iar a doua o divizare cu 5.

Un numărător BCD, cu intrarea pe CP0, se obține prin realizarea unei conexiuni externe între Q0 și CP1.

Ștergerea numărătorului se execută în mod asincron prin  $MR1 = MR2 = 1$ .

Prin  $MS1 = MS2 = 1$ , în numărător se încarcă în mod asincron constanta 9.

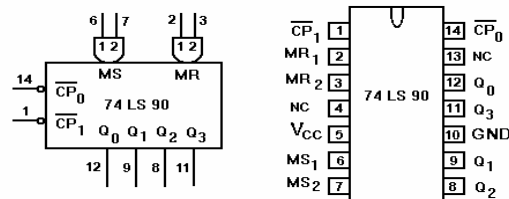


Fig. 7. Circuitul 74LS90: simbol, pinout

- Circuitele: **74 LS 196** - numărător BCD asincron presetabil

**74 LS 197** - numărător binar presetabil pe 4 biți

Circuitele sunt active pe tranziția negativă a semnalului de ceas.

Circuitele sunt organizate în două secțiuni: prima realizează o divizare cu 2 a frecvenței de intrare, iar a doua o divizare cu 5 respectiv cu 8.

Pentru LS196 un numărător BCD, cu intrarea pe CP0, se obține prin realizarea unei conexiuni externe între Q0 și CP1.

Pentru LS197 un numărător binar pe 4 biți, cu intrarea pe CP0, se obține prin realizarea unei conexiuni externe între Q0 și CP1.

Ștergerea numărătoarelor se execută în mod asincron pentru  $\overline{MR}=0$ .

Încărcarea paralelă se face în mod asincron prin  $\overline{PL}=0$ .

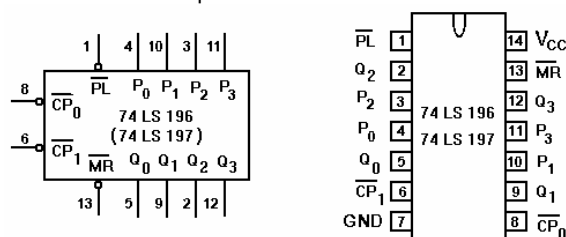


Fig. 8. Circuitul 74LS196 / (74LS197): simbol logic, pinout

## 2.4. Cascadarea numărătoarelor

Extinderea capacității de numărare se face prin conectarea convenabilă a mai multor circuite de capacitate mai mică. Modul de conectare este dependent de circuitele utilizate și de performanțele impuse numărătorului mare ce trebuie realizat.

*Cascadarea numărătoarelor asincrone* se face simplu, prin interconectarea bitului MSB al numărătorului  $n-1$ , la intrarea de ceas a numărătorului  $n$ . Această metodă este cunoscută sub denumirea de *Ripple Count*. Noua structură, de capacitate mai mare, are tot comportament de numărător asincron.

## 2.5. Realizarea divizoarelor de frecvență cu ajutorul numărătoarelor asincrone

Divizorul digital de frecvență este un circuit ce realizează o reducere, cu un anumit coeficient, a frecvenței semnalului de intrare. De regulă, coeficientul de divizare este un număr întreg. Atragem atenția asupra faptului că semnalele de intrare și de ieșire sunt digitale, deci trebuie să respecte nivelele de tensiune asociate prin standard pentru ambele stări logice.

Din analiza semnalelor prezentate în figura 3 se observă destul de clar că un numărător binar poate fi utilizat pentru obținerea unor factori de divizare ce reprezintă puteri ale cifrei 2 ( spre exemplu divizare cu 2, cu 4, cu 8, cu 16, etc.).

În diverse aplicații practice apare necesitatea utilizării unor factori de divizare ce nu reprezintă puteri ale cifrei 2, (spre exemplu divizări cu 10). Pentru astfel de cazuri este necesar să utilizăm în mod convenabil facilitatea de ștergere (*Reset*).

În principiu, realizarea unui divizor de frecvență cu factorul de divizare  $k$ , este echivalent cu sinteza unui numărător cu  $k$  stări distincte. Pentru aceasta este necesar să alegem un numărător binar cu  $2^n$  stări distincte, însă trebuie avut grijă să fie îndeplinită condiția  $k < 2^n$ , unde  $n$  reprezintă numărul de bistabili ai numărătorului binar.

Dacă nu intervenim în nici un fel asupra numărătorului binar acesta va avea  $2^n$  stări distincte și nu  $k$  stări distincte cum dorim noi. Pentru a rezolva această problemă este necesar să împărțim stările numărătorului în două categorii:

- stări permise (stările de la 0 la  $k-1$ ) – pe durata acestor stări funcția de ștergere nu trebuie activată;
- stări nepermise (stările mai mari decât  $k$ ) – pe durata acestor stări funcția de ștergere trebuie activată.

Așadar, trebuie proiectat un CLC care primește la intrare starea numărătorului și generează la ieșire un semnal de comandă a intrării de reset a numărătorului binar.

**Exemplu:** Realizarea unui divizor de frecvență cu 12 folosind structura de numărător binar din figura 2.

Tabelul 1.

Nr. etapă	Denumire etapă	Observații
1	Alegerea numărătorului binar și alcătuirea unui tabel cu succesiunea stărilor prin care trece acesta	Avem nevoie de un numărător binar de cel puțin 4 biți
2	Separarea stărilor permise de cele nepermise	Numărătoarea stărilor permise se începe în mod obligatoriu cu starea 0;

	$Q_D$	$Q_C$	$Q_B$	$Q_A$	$\bar{R}$	Observatii
	0	0	0	0	1	
	0	0	0	1	1	
	0	0	1	0	1	
	0	0	1	1	1	Stari permise
	0	1	0	0	1	(Funcția de ștergere trebuie să fie inactivă $\bar{R} = 1$ )
	0	1	0	1	1	
	0	1	1	0	1	
	0	1	1	1	1	
	1	0	0	0	1	
	1	0	0	1	1	
	1	0	1	0	1	
	1	0	1	1	1	
	1	1	0	0	0	Stari nepermise
	1	1	0	1	*	(Funcția de ștergere trebuie să fie activă $\bar{R} = 0$ )
	1	1	1	0	*	
	1	1	1	1	*	

În momentul în care circuitul ajunge în starea 1100, intrarea de ștergere se activează și numărătorul este forțat să treacă în starea 0000;

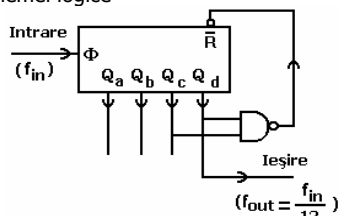
Prima stare nedorită este "atinsă tangențial", se trece prin ea doar câteva nanosecunde. Această stare nu este lăsată să existe.

Funcția de *Reset* este necesar să fie activă doar în prima stare nedorită. În restul stărilor nedorite funcția de ștergere poate avea valoarea *\*=don't care*, deoarece numărătorul nu mai are cum să treacă prin aceste stări.

3 Sinteza CLC-ului pentru comanda intrării de ștergere.  
 $\bar{R} = Qd + Qc = Qd \cdot Qc$

Se aplică a doua formă canonică și se ține cont de faptul că \*, poate fi considerată '0' sau '1'.

4 Realizarea schemei logice



Durata de '1' a semnalului de ieșire este egală cu 4 perioade ale semnalului de intrare;  
 Durata de '0' a semnalului de ieșire este egală cu 8 perioade ale semnalului de intrare;

## 2.6. Utilizarea editorului de scheme din mediul WebPack

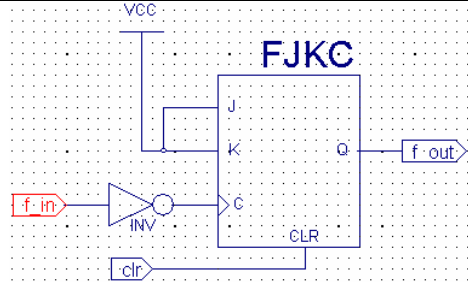
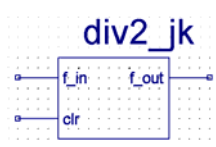
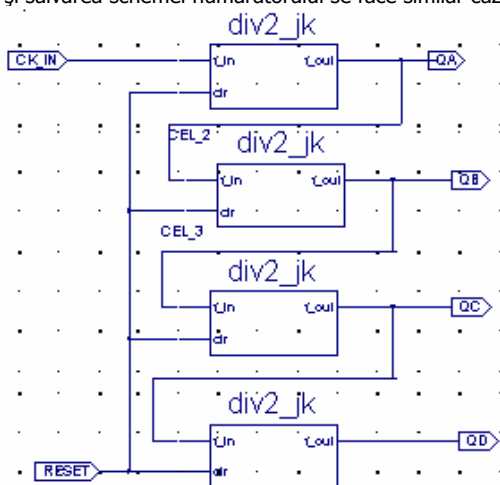
Foarte multe scheme digitale sunt formate dintr-o celulă de bază care este repetată de un anumit număr de ori (în această categorie se încadrează și numărătoare binare asincrone). Pentru astfel de scheme este utilă definirea unui simbol propriu (conceput de către noi) prin care să reprezentăm celula de bază a schemei.

Modul în care se definește un nou simbol precum și asocierea acestui simbol cu o anumită schemă internă se prezintă în exemplul de mai jos.

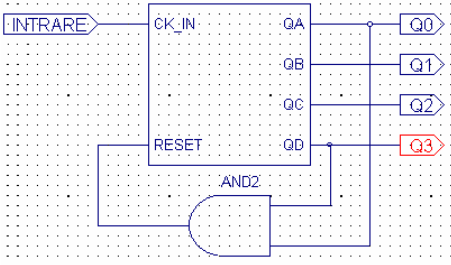
**Exemplu:** Realizarea unui divizor de frecvență folosind un simbol atașat structurii de numărător din figura 2.

Tabelul 2.

Nr.	Denumire etapă	Mod de lucru
1	Deschiderea unui nou proiect	Din fereastra <b>Project Navigator</b> se execută următoarea secvență de comenzi: <b>File</b> → <b>New Project</b> → se atribuie un nume (spre exemplu <b>num_4bit</b> ) → <b>OK</b> → ...
2	Adăugăm la proiect un fișier nou de tip schemă logică în care descriem schema internă a celulei de bază.  În exemplul de față, celula de bază este un bistabil JK conectat astfel încât să realizeze o divizare cu 2 a frecvenței semnalului de intrare.  <b>Atenție:</b> Bistabilii din interiorul CPLD au toate intrările active pe '1' iar intrarea de ceas este activă pe tranziția pozitivă.  După generarea simbolului grafic, avem la dispoziție un simbol (definit de noi, cu numele <b>div_jk</b> ), simbol ce poate fi folosit drept componentă în alte scheme.	1. Secvența de comenzi este: <b>Project</b> → <b>New Sources</b> → <b>Schematic</b> și se atribuie un nume (spre exemplu <b>div_jk</b> ) → <b>Next</b> → <b>Finish</b> . În urma acestor comenzi se lansează editorul de scheme logice.  2. Se desenează schema unui divizor de frecvență cu 2 realizată cu bistabil JK: <ul style="list-style-type: none"> <li>se aduce un bistabil JK în fereastra de lucru prin secvența de comenzi: <b>Symbols</b> → <b>Flip-flops</b> → <b>fjkc</b> → deplasarea mouse în câmpul de desen → clic stânga pentru plasare componentă;</li> <li>trasare conexini prin secvența de comenzi: <b>Add</b> → <b>Wire</b> → deplasare mouse până la un capăt al firului → clic dreapta cu menținere și deplasare până la celălalt capăt al firului → eliberare buton stânga;</li> <li>adăugarea de pini de intrare/ieșire prin secvența de comenzi: <b>Add</b> → <b>I/O Marker</b> → din fereastra <b>Options</b> se alege după caz <b>Add an input marker</b> sau <b>Add an output marker</b> → se deplasează mouseul în zona de lucru și se face clic dreapta pentru amplasare → se apasă butonul săgeată orientată spre stânga (de pe toolbarul superior) → se face dublu clic pe marker → în fereastra apărută se modifică proprietatea <b>Name</b> cu denumirea dorită a intrării/ieșirii → <b>OK</b>.</li> <li>După terminarea desenului se salvează prin <b>File</b> → <b>Save</b>.</li> </ul>

	<p>Denumirea asociată simbolului este aceeași cu denumirea fișierului ce descrie conținutul.</p>	 <p>3. Se asociază schemei logice un simbol grafic</p> <ul style="list-style-type: none"> <li>Se revine în <b>Project Navigator</b> și se execută următoarea secvență de comenzi: în fereastra <b>Source in Project</b> se alege <b>div_jk</b> → în fereastra <b>Processes for current sources</b> se face dublu clic pe <b>Create Schematic Symbol</b>.</li> </ul> 
3	<p>Adăugăm la proiect un nou fișier de tip schemă în care descriem structura internă a numărătorului.</p> <p><i>În exemplul de față, vom lega în cascadă 4 simboluri de <b>div_jk</b> pentru a realiza un numărător binar pe 4 biți.</i></p> <p><i>Din schemă se observă că avem 4 blocuri <b>div2_jk</b>, conectate similar schemei din figura 2.</i></p>	<p>1. Din fereastra <b>Project Navigator</b>, printr-o secvență de comenzi ce a fost deja descrisă, se introduce un nou fișier de tip schemă logică, denumit <b>n_asincron4</b>.</p> <p>2. În editorul de scheme se verifică lista simbolurilor disponibile pentru a constata prezența simbolului generat de noi, <b>div_jk</b>. Pentru aceasta, în textboxul <b>Categories</b> trebuie să existe o linie de forma <b>&lt;c:/cid/proiecte_vhdl/num_4bit&gt;</b>, linie ce indică locul unde se află amplasat proiectul nostru. Dacă se selectează această linie, în textboxul <b>Symbols</b> trebuie să existe și <b>div_jk</b>.</p> <p>3. Desenarea și salvarea schemei numărătorului se face similar cazului anterior.</p>  <p>4. Generarea unui simbol grafic pentru numărător (similar cazului anterior)</p>
4	<p>Adăugarea fișierului de constrângeri.</p> <p><b>Atenție:</b> Această etapă este necesară numai dacă dorim testarea structurii numărătorului.</p>	<p>1. Din fereastra <b>Project Navigator</b>, printr-o secvență de comenzi deja cunoscută se adaugă un nou fișier de tip <b>Implementation Constrains File</b>. După declararea numelui, acest fișier trebuie asociat cu schema <b>n_asincron4</b>.</p> <ul style="list-style-type: none"> <li>Pentru comanda intrării de ceas se folosește contactul cu revenire BTN1; NET "RESET" LOC = "P83"; NET "CK_IN" LOC = "P56";</li> <li>Pentru comanda intrării de ștergere se folosește BTN5; NET "QD" LOC = "P62"; NET "QC" LOC = "P65";</li> <li>Afișarea stării numărătorului se face pe LED-urile LD1÷LD4. NET "QB" LOC = "P67"; NET "QA" LOC = "P69";</li> </ul>
5	<p>Adăugăm la proiect un nou fișier de tip schemă logică în care descriem schema divizorului de frecvență.</p>	<p>1. Din fereastra <b>Project Navigator</b>, printr-o secvență de comenzi deja cunoscută se introduce un nou fișier de tip schemă logică, denumit <b>div_9</b>.</p> <p>2. În editorul de scheme se verifică lista simbolurilor disponibile pentru a constata prezența simbolului asociat pentru numărătorul <b>n_asincron4</b>.</p> <p>3. Se desenează schema divizorului de frecvență de mai jos. După desenare există posibilitatea de a vizualiza structura internă a</p>



		<p>numărătorului. Pentru aceasta trebuie parcursă următoarea secvență de comenzi: în editorul de scheme se marchează simbolul <b>n_asincron4</b> apoi <b>View → Push Into Symbol</b>. Va apare o schemă identică cu cea din linia 4 a acestui tabel. Chiar și de aici se poate merge pe adâncime, în sensul că se poate vizualiza care este schema logică a blocului <b>div2_jk</b>.</p> 
6	<p>Adăugarea fișierului de constrângeri.</p> <p><i>După introducerea acestui fișier se observă o rearanjare a surselor implicate în proiect, fișierul div_9 devine principal</i></p>	<p>1. Din fereastra <b>Project Navigator</b>, printr-o secvență de comenzi deja cunoscută se adaugă un nou fișier de tip <b>Implementation Constrains File</b>. După declararea numelui, acest fișier trebuie asociat cu schema <b>div_9</b>.</p> <ul style="list-style-type: none"> <li>Pentru comanda intrării se folosește BTN1;</li> <li>Afișarea stării numărătorului se face pe LED-urile LD1÷LD4.</li> </ul> <p>NET "INTRARE" LOC = "P56";  NET "Q3" LOC = "P62";  NET "Q2" LOC = "P65";  NET "Q1" LOC = "P67";  NET "Q0" LOC = "P69";</p>



### 3. Desfășurarea lucrării

#### 3.1. Implementarea divizoarelor de frecvență cu ajutorul numărătoarelor asincrone

- A. Realizați pe macheta de test schemele din figura 9, după care vizualizați cu ajutorul osciloscopului cu două canale formele de undă de la intrarea și ieșirile numărătorului. Desenați corelat în timp aceste semnale.

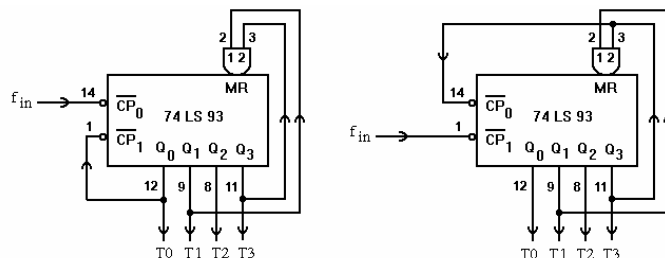


Fig. 9.

#### Întrebări:

- Care sunt stările prin care trece numărătorul pentru cele două cazuri considerate ?
- Ce factor de divizare în frecvență realizează fiecare schemă ? Ce observații puteți face referitor la factorul de umplere al semnalelor de la ieșirile numărătorului ?
- Ce se modifică în funcționarea schemei dacă se întrerupe legătura de la Q1 la MR1?

2. Folosind circuitul 7493 se cere realizarea unor divizoare de frecvență cu următorii factori de divizare ai frecvenței: a) divizare cu 5; b) divizare cu 9; c) divizare cu 10; c) divizare cu 13; Pentru cele patru cazuri se cer schemele electrice și formele de undă de la ieșirile Q3.

3. Care este factorul de divizare în frecvență și factorul de umplere al semnalului de la ieșirea Y a schemei din figura 10?

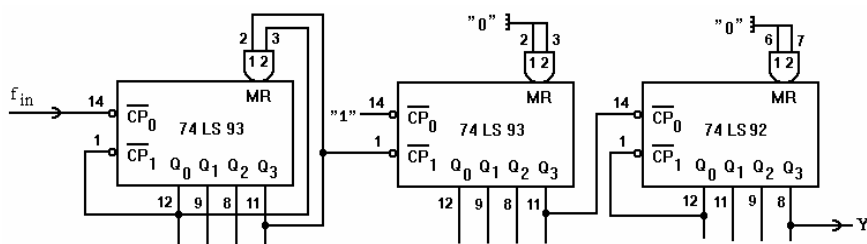


Fig. 10.

4. Analizați funcționarea circuitului din figura 11, circuit ce reprezintă o altă modalitate de obținere a divizoarelor de frecvență cu factori mari de divizare.

După realizarea schemei pe macheta de test, la intrare se aplică un semnal TTL cu o frecvență de aproximativ 100 KHz. Pe un canal al osciloscopului se vizualizează semnalul de intrare, iar pe celălalt canal se aplică succesiv semnalele din punctele A, B, C.

Se desenează corelat în timp semnalul de intrare și semnalele din punctele A, B, C.

#### Întrebări:

- Care este rolul porții AND ?
- Care este factorul de divizare al fiecărui circuit și care este factorul de divizare global obținut la ieșirea C ?
- Care este avantajul acestei metode comparativ cu cea utilizată în schema anterioară ?
- Ce factor de umplere au semnalele din punctele A și B ? Se modifică funcționarea schemei dacă factorul de umplere al celor două semnale (din A și B) este altul ? Exemplificați pe un caz concret.

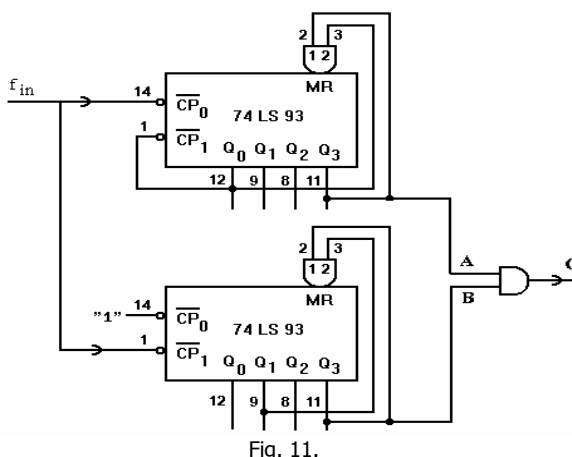


Fig. 11.

### 3.2. Utilizarea ISE WebPack pentru descrierea aplicațiilor sub formă de scheme logice

- Verificați pe macheta de laborator proiectul prezentat ca exemplu în tabelul 2 din prezenta lucrare de laborator. Urmăriți succesiunea stărilor prin care trece numărătorul (pe bareta de LED-uri), atunci când apăsați butonul cu revenire BTN1.
- Adăugați la proiectul din tabelul 2 un decodificator BCD-7segmente pentru afișarea stării numărătorului pe primul digit al machetei de laborator. Notați succesiunea stărilor prin care trece numărătorul.
- Intervenți asupra celei de divizare, *div2\_jkb* și eliminați din schema logică inversorul (pinul de intrare intrarea *f\_in* se conectează direct la intrarea de ceas a bistabilului JK). Refaceți implemetarea și urmăriți succesiunea stărilor prin care trece numărătorul. Ce se constată față de cazul anterior ?
- Folosind modul de lucru prezentat în tabelul 2 și ținând cont de structura internă a numărătorului 74LS93 din figura 5 (atenție, toate intrările J și K trebuie conectate la '1'), realizați câte o implementare pentru fiecare schemă din figura 9.
- Folosind modul de lucru prezentat în tabelul 2 și ținând cont de structura internă a numărătorului 74LS93 din figura 5 (atenție, toate intrările J și K trebuie conectate la '1'), realizați o implementare pentru divizorul de frecvență din figura 11.

### 4. Indicații privind modul de lucru

Pentru fiecare aplicație este necesară deschiderea unui nou proiect după metodologia prezentată într-o lucrare de laborator anterioară.

Toate aplicațiile din această lucrare necesită doar un singur fișier sursă (de tip schemă logică) și un singur fișier de constrângeri.

Referitor la comanda intrării de ceas facem următoarele precizări:

- Intrarea de ceas se poate comanda printr-un buton cu revenire (spre exemplu BTN1). Deoarece există riscul apariției de oscilații la apăsarea butonului, se recomandă urmărirea stărilor prin care trece numărătorul pentru mai multe cicluri complete ale sale;
- O metodă și mai bună de comandă a intrării de ceas constă în folosirea unui semnal digital periodic cu frecvență suficient de mică pentru a putea urmări succesiunea stărilor prin care trece numărătorul. Pentru aceasta, putem folosi semnalul de la pinul P9 al CPLD, semnal ce are o frecvență de 25,175MHz. În schema logică, între pinul P9 și intrarea de ceas a numărătorului testat intercalăm un numărător binar pe 24 de biți. În aceste condiții, semnalul cules de pe ieșirea cea mai semnificativă a numărătorului va avea frecvența de cca. 1,5Hz, valoare ce ne permite vizualizarea stărilor prin care trece numărătorul.

Intrarea de reset se comandă prin BTN7;

Intrarea de încărcare paralelă se comandă printr-un switch;

Afișarea stării numărătorului se face pe LED-uri;

Fișierul de constrângeri este similar celui prezentat în tabelul 2, linia6;



Lucrarea nr. 8: **Studiul numărătoarelor sincrone****1. Scopul lucrării**

Lucrarea are ca scop familiarizarea studentului cu numărătoarelor **sincrone**. Pentru aceasta, se pune accent pe: înțelegerea structurii interne și a modului de funcționare; prezentarea avantajelor și dezavantajelor ce decurg din structura internă; prezentarea numărătoarelor sincrone realizate în structură integrată ce sunt disponibile pe piață (semnificația pinilor, particularitățile în funcționare, modul de cascaderare și modul de operare cu acestea); prezentarea modalităților de extindere a capacității de numărare; prezentarea modalităților de descriere în limbaj VHDL; implementarea în structuri logice programabile de tip CPLD; utilizarea numărătoarelor în realizarea divizoarelor digitale de frecvență.

**2. Considerente teoretice****2.1. Numărătoare binare sincrone****Caracteristici generale:**

- Schema logică a unei structuri de numărător sincron este mult mai complexă față de cazul numărătoarelor asincrone și crește odată cu creșterea capacității numărătorului.
- Semnalul de ceas se aplică simultan tuturor bistabililor din schemă.
- Comutarea unui bistabil se face sincron cu semnalul de ceas numai dacă toți bistabilii anteriori acestuia sunt în unu logic.
- Nu prezintă stări intermediare nedorite.
- Din cauza complexității mai ridicate, de cele mai multe ori numărătoarele sincrone integrate sunt de capacități mici 4, 8 biți.
- Structurile sincrone sunt utilizate, cel mai adesea, pentru realizarea de numărătoare mai complexe, structuri care au facilități suplimentare precum: încărcarea paralelă, controlul numărării, numărare în ambele sensuri, semnalizarea terminării numărării etc.

**Exemplu:** O schemă posibilă de numărător binar sincron unidirecțional pe 4 biți, fără facilități speciale, este prezentată în fig.

1. Semnificația semnalelor din această schemă este următoarea:

- $\Phi$  intrare de numărare, sensibilă (activă) la tranziția negativă;
- $\bar{R}$  intrare de ștergere asincronă a numărătorului, activă pe zero logic;
- Qd, Qb, Qc, Qa - ieșirile numărătorului prin care se indică în exterior, în cod binar, numărul impulsurilor înregistrate. Ieșirea Qd este cel mai semnificativ bit, iar Qa cel mai puțin semnificativ bit.
- CE (Count Enable), intrare activă pe unu logic cu rol de validare a numărării. Pentru CE= 0 numărarea nu este permisă iar pentru CE=1 se desfășoară în ritmul semnalului aplicat intrării  $\Phi$ ;

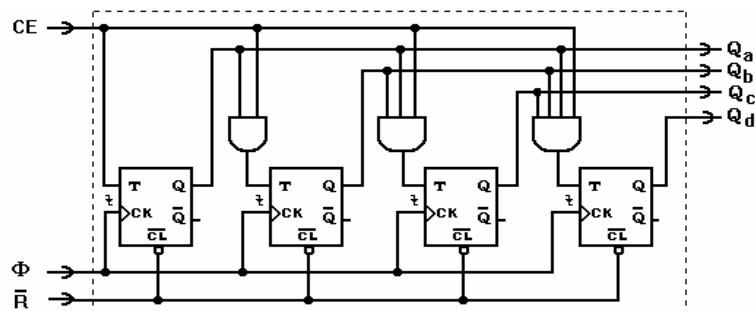


Fig. 1. Structură de numărător binar sincron pe 4 biți

Analizând succesiunea stărilor prin care trece numărătorul în cod binar natural, se observă că schimbarea unui bit se face numai atunci când toți biții anteriori lui, de ponderi inferioare, sunt "unu". Aceste situații sunt identificate cu ajutorul porților AND ce sunt conectate în fața fiecărui bistabil.

În plus, la AND-uri se mai conectează intrarea de CE cu scopul de a introduce o facilitate suplimentară - aceea de a valida sau nu procesul de numărare. Dacă CE= 0 toate porțile AND au ieșirile forțate în "0", fapt ce determină bistabilii T să fie în regim de memorare a stării anterioare. Cu alte cuvinte, atâta timp cât CE= 0 starea numărătorului nu se modifică chiar dacă la intrarea  $\Phi$  se aplică impulsuri. Pentru CE= 1 procesul de numărare este permis.

**2.2. Exemple de numărătoare sincrone realizate în structuri integrate**

În această secțiune se prezintă câteva tipuri de numărătoare asincrone, mai des utilizate, ce sunt disponibile sub formă de circuite integrate. Pentru fiecare circuit se prezintă semnificația pinilor și particularitățile funcționale ale acestuia.

**• Funcția de reversibilitate**

Numărătoarele reversibile pot realiza numărarea impulsurilor de intrare fie în sens crescător, fie în sens descrescător. Aceste circuite se împart în două categorii: a) circuite cu două intrări de ceas; b) circuite cu o intrare de ceas și o intrare de

comandă a sensului de numărare. Pentru primul caz, cele două intrări de ceas sunt denumite COUNT UP respectiv COUNT DOWN. În cazul doi, intrarea de ceas este denumită CLOCK iar cea de comandă UP / DOWN.

• **Funcția de încărcare paralelă (PRESET)**

Numărătoarele ce dispun de această facilități, prezintă avantajul că pot starta numărarea dintr-o stare particulară, ce se încarcă în prealabil în mod paralel. Pentru a permite acest lucru, circuitul este prevăzut cu:

- intrări de date, notate  $P_0, P_1, \dots$ , prin intermediul cărora se specifică constanta binară ce trebuie încărcată paralel;
- o intrare de control prin intermediul căreia se comandă introducerea în bistabili a datelor de pe intrările paralele PI.

Intrarea poate fi întâlnită sub diverse denumiri: LOAD, LOAD ENABLE, PARALLEL LOAD etc.

Facem precizarea că funcția de încărcare paralelă poate fi executată sincron sau asincron față de semnalul de ceas aplicat numărătorului. La numărătoarele cu presetare asincronă, încărcarea se execută odată cu activarea intrării LD și nu ține cont de starea logică a semnalului de ceas. Pentru numărătoarele cu presetare sincronă, încărcarea efectivă se execută la prima tranziție activă a semnalului de ceas care apare după activarea intrării LD.

Trebuie reținut că, pe durata activării funcției de încărcare paralelă, procesul de numărare este inhibat.

• **Funcția de ștergere a numărătorului (RESET)**

Pentru aducerea în starea zero a unui numărător, marea majoritate a numărătoarelor sunt prevăzute cu o intrare denumită RESET. Funcție de circuitul utilizat, funcția de ștergere poate fi executată sincron, sau asincron, față de semnalul de ceas aplicat numărătorului. Pentru numărătoarele cu resetare asincronă, ștergerea se face la momentul activării intrării RESET, deci nesincronizat cu semnalul de ceas. În celălalt caz, numărătoare cu resetare sincronă, ștergerea se face efectiv la prima tranziție activă a semnalului de ceas ce apare după activarea intrării de RESET.

Unele firme, pentru a face distincție între cele două modalități de ștergere, notează prin: MASTER RESET resetul asincron, și prin SYNCHRONOUS RESET pe cel sincron.

• **Funcția de semnalizarea a terminării numărării (TERMINAL COUNT)**

În aplicații precum: extinderea capacității de numărare, realizarea divizoarelor programabile de frecvență etc, este foarte utilă semnalizarea în exterior a momentelor de umplere, sau după caz, de golire a numărătoarelor. Prin umplere se înțelege trecerea numărătorului prin starea sa maximă, iar prin golire trecerea sa prin zero.

De regulă, numărătoarele reversibile au două ieșiri destinate acestui scop: a) TERMINAL COUNT UP sau CARRY, pentru semnalizarea umplerii; b) TERMINAL COUNT DOWN sau BORROW, pentru semnalizarea golirii.

Numărătoarele unidirecționale prezintă o ieșire TERMINAL COUNT prin care semnalizează trecerea prin starea maximă a acestora.

**2.3.1. Numărătoare sincrone presetabile**

- Circuitele: **74LS160A, 74LS162A** - numărătoare BCD sincrone presetabile

**74LS 161A, 74LS163A** - numărătoare binare sincrone presetabile pe 4 biți

Circuitele sunt active pe tranziția pozitivă a semnalului de ceas.

Contorizare dacă:  $CEP \cdot CET \cdot PE = 1$

Ștergerea numărătoarelor se execută:

- în mod asincron pentru LS160A și LS161, dacă  $\overline{MR} = 0$
- în mod sincron pentru LS162A și LS163, dacă  $\overline{SR} = 0$  și tranziție  $\uparrow$  pe CP.

Încărcarea paralelă se face sincron pentru toate circuitele dacă:  $\overline{PE} = 0$  și tranziție  $\uparrow$  pe CP.

Semnalizarea umplerii numărătorului (terminarea numărării):

-  $TC = CET \cdot Q_0 \cdot \overline{Q_1} \cdot \overline{Q_2} \cdot Q_3$  pentru LS160 și LS162

-  $TC = CET \cdot Q_0 \cdot Q_1 \cdot Q_2 \cdot Q_3$  pentru LS160 și LS162

Circuit	Numărare	LOAD	RESET
74 LS 160 A	BCD	sincron	asincron
74 LS 161 A	binar	sincron	asincron
74 LS 162 A	BCD	sincron	sincron
74 LS 163 A	binar	sincron	sincron

CEP - Count Enable Parallel  
 CET - Count Enable Trickle  
 CP - Clock (Active HIGH Going Edge) Input  
 $\overline{PE}$  - Parallel Enable (Active LOW) Input  
 TC - Terminal Count Output  
 $\overline{MR}$  {  $\overline{MR}$  - Master Reset (Active LOW) Input for LS160A and LS161A  
 $\overline{SR}$  - Synchronous Reset (Active LOW) Input for LS162 and LS163A  
 $Q_0 - Q_3$  - Parallel Outputs  
 $P_0 - P_3$  - Parallel Inputs

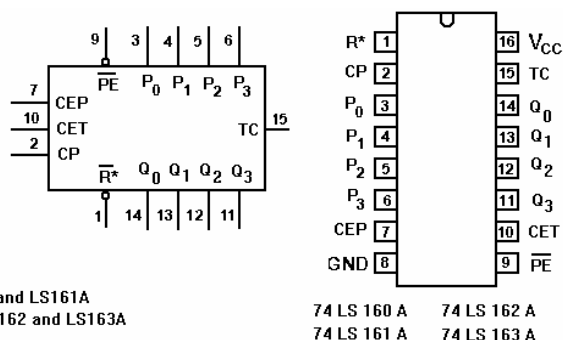


Fig. 2. Circuitele LS160A, LS161A, LS162A, LS163A

Ieșirea TC prezintă spikes-uri datorită decodărilor interne, deci nu se recomandă a fi utilizată pentru: comanda altor numărătoare, resetări asincrone, încărcări paralele asincrone, comanda ceasului la bistabili și registre.

Numărătoarele BCD pot fi aduse într-o stare ilegală prin încărcare paralelă sau la conectarea tensiunii de alimentare. Dintr-o stare ilegală circuitele ajung în una legală în două perioade de ceas.

### 2.3.2. Numărătoare sincrone bidirecționale presetabile

- Circuitele: **74LS168** - numărător BCD bidirecțional sincron presetabil  
**74LS 169** - numărător binar bidirecțional sincron presetabil pe 4 biți  
 Circuitele sunt active pe tranziția pozitivă a semnalului de ceas.  
 Contorizare

- înainte dacă:  $\overline{CEP} \cdot \overline{CET} \cdot PE = 1$  și  $U/\overline{D} = 1$

- înapoi dacă:  $\overline{CEP} \cdot \overline{CET} \cdot PE = 1$  și  $U/\overline{D} = 0$

Încărcarea paralelă se face sincron pentru ambele circuitele dacă:  $\overline{PE} = 0$  și tranziție  $\uparrow$  pe CP.  
 Semnalizarea umplerii numărătorului (terminarea numărării):

-  $\overline{TC} = Q_0 \overline{Q_1} \overline{Q_2} \overline{Q_3} \cdot \overline{CET}$  pentru LS168 la numărarea înainte ( $U/\overline{D} = 1$ );

-  $\overline{TC} = Q_0 Q_1 Q_2 Q_3 \cdot \overline{CET}$  pentru LS169 la numărarea înainte ( $U/\overline{D} = 1$ );

-  $\overline{TC} = \overline{Q_0} \overline{Q_1} \overline{Q_2} \overline{Q_3} \cdot \overline{CET}$  pentru ambele circuite la numărarea înapoi ( $U/\overline{D} = 0$ ).

Pentru circuitul LS168, ieșirea  $\overline{TC}$  se activează ilegal în stările nepermise 11,13 și 15.

Ieșirea  $\overline{TC}$  prezintă spikes-uri datorită decodărilor interne, deci nu se recomandă a fi utilizată pentru: comanda intrării de ceas a altor numărătoare, resetări sau presetări asincrone, comanda ceasului la bistabili sau registre.

Numărătoarele BCD pot fi aduse în stări ilegale, fie la conectarea tensiunii de alimentare, fie prin încărcare paralelă. Dintr-o stare ilegală circuitele ajung în una legală în două perioade de ceas.

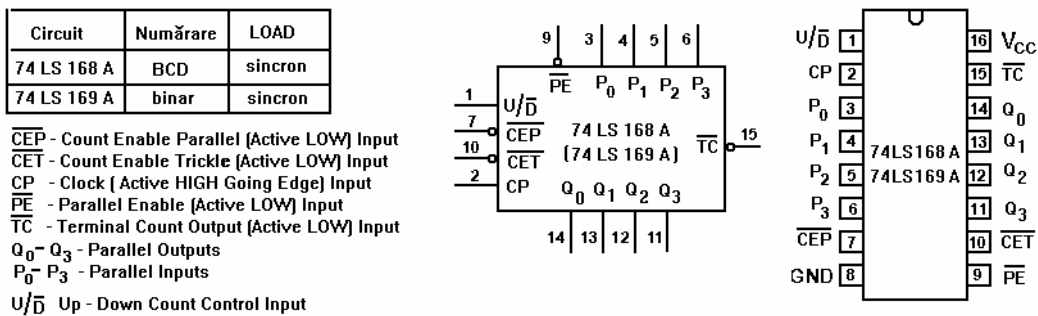


Fig. 3. Circuitele LS168, LS169

- Circuitele: **74LS190** - numărător BCD bidirecțional sincron presetabil  
**74LS 191** - numărător binar bidirecțional sincron presetabil pe 4 biți  
 Circuitele sunt active pe tranziția pozitivă a semnalului de ceas.  
 Contorizare :

- înainte dacă:  $\overline{CE} = 0$  și  $\overline{U}/D = 0$

- înapoi dacă:  $\overline{CE} = 0$  și  $\overline{U}/D = 1$

Încărcarea paralelă se face asincron pentru ambele circuitele dacă  $\overline{PL} = 0$ .

Semnalizarea umplerii numărătorului (terminarea numărării):

-  $\overline{TC} = Q_0 Q_3$  pentru LS190 la numărarea înainte ( $\overline{U}/D = 0$ );

-  $\overline{TC} = Q_0 Q_1 Q_2 Q_3$  pentru LS191 la numărarea înainte ( $\overline{U}/D = 0$ );

-  $\overline{TC} = \overline{Q_0} \overline{Q_1} \overline{Q_2} \overline{Q_3}$  pentru ambele circuite la numărarea înapoi ( $\overline{U}/D = 1$ ).

Ieșirea TC prezintă spikes-uri datorită decodărilor interne, deci nu se recomandă a fi utilizată pentru: comanda intrării de ceas a altor numărătoare, resetări sau presetări asincrone, comanda ceasului la bistabili sau registre.

Ieșirea  $\overline{RC}$  este activă pe zero logic și semnalizează tot terminarea numărării, însă durata de activare este egală cu durata de zero a semnalului de ceas. Dacă  $TC=1$  și  $\overline{CE} = 0$ , ieșirea  $\overline{RC}$  se activează la prima tranziție negativă a semnalului de ceas și durează până la prima tranziție pozitivă a semnalului de ceas. Această ieșire nu prezintă spikes-uri și este utilă pentru cascada numărătoarelor.

Trecerea din zero în unu a intrării  $\overline{CE}$  trebuie făcută numai pe  $CP=1$ .

Modificarea stării intrării  $\overline{U}/D$  trebuie făcută pe  $CP=1$  sau pe  $\overline{CE} = 1$ .

Numărătorul BCD poate fi adus în stări ilegale, fie la conectarea tensiunii de alimentare, fie prin încărcare paralelă. Dintr-o stare ilegală circuitul reintră în secvența legală în două perioade de ceas.

Circuit	Numărare	LOAD
74 LS 190	BCD	sincron
74 LS 191	binar	sincron

$\overline{CE}$  - Count Enable (Active LOW) Input  
 $CP$  - Clock (Active HIGH Going Edge) Input  
 $PL$  - Parallel Load (Active LOW) Input  
 $TC$  - Terminal Count Output (Active HIGH) Output  
 $U/D$  - Up - Down Count Control Input  
 $\overline{RC}$  - Ripple Clock Output  
 $Q_0 - Q_3$  - Parallel Outputs  
 $P_0 - P_3$  - Parallel Inputs

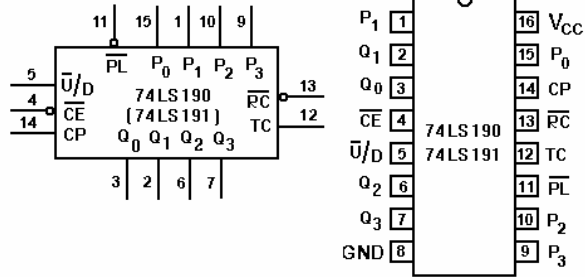


Fig. 4. Circuitele LS190, LS191

- Circuitele: **74LS192** - numărător BCD bidirecțional sincron presetabil  
**74LS 193** - numărător binar bidirecțional sincron presetabil pe 4 biți  
 Circuitele sunt active pe tranziția pozitivă a semnalului de ceas.  
 Contorizare dacă  $PL = 1$  și  $MS = 0$   
 - înainte dacă:  $CP_D = 1$  și  $CP_U$  primește semnalul de numărare.  
 - înapoi dacă:  $CP_U = 1$  și  $CP_D$  primește semnalul de numărare.  
 Încărcarea paralelă se face asincron pentru ambele circuitele dacă  $\overline{PL} = 0$ .

Circuit	Numărare	LOAD	RESET
74 LS 192	BCD	asincron	asincron
74 LS 193	binar	asincron	asincron

$Q_0 - Q_3$  - Parallel Outputs  
 $P_0 - P_3$  - Parallel Inputs  
 $CP_U$  - Count Up Clock Pulse (Active HIGH Going Edge) Input  
 $CP_D$  - Count Down Clock Pulse (Active HIGH Going Edge) Input  
 $MR$  - Asynchronous Master Reset (Active HIGH) Input  
 $PL$  - Asynchronous Parallel Load (Active LOW) Input  
 $TC_U$  - Terminal Count Up (Active LOW) Output  
 $TC_D$  - Terminal Count Down (Active LOW) Output

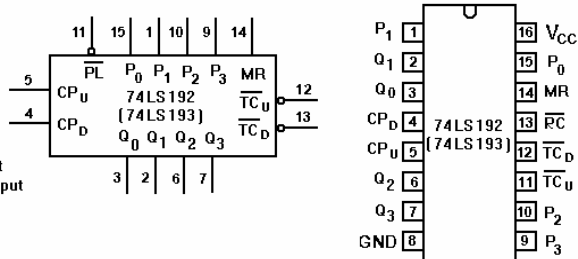


Fig. 5. Circuitele LS192, LS193

- Terminarea numărării este semnalizată prin două ieșiri active pe zero logic:
- $\overline{TC_U}$  pentru umplerea numărătorului (trecerea prin 9 în cazul LS192, respectiv trecerea prin 15 în cazul LS193) dar numai la numărarea înainte;
  - $\overline{TC_D}$  pentru trecerea prin zero a numărătoarelor, numai la numărarea înapoi.
- Ieșirile  $\overline{TC_U}$  și  $\overline{TC_D}$  nu prezintă spikes-uri deci pot fi utilizate pentru cascada circuitelor. Dacă condițiile de activare specificate anterior sunt îndeplinite, activarea efectivă începe cu prima tranziție negativă și se termină la prima tranziție pozitivă a semnalului de ceas.
- Pentru numărătorul BCD, ieșirea dintr-o stare nepermisă și intrarea în secvența legală se face în două perioade de ceas.

## 2.5. Cascadarea numărătoarelor

Extinderea capacității de numărare se face prin conectarea convenabilă a mai multor circuite de capacitate mai mică. Modul de conectare depinde de circuitele utilizate și de performanțele impuse numărătorului mare ce trebuie realizat.

Cascadarea numărătoarelor sincrone poate fi realizată în câteva moduri, dintre care unele pierd caracterul sincron al numărătorului mare.

- **Modul Ripple Clock** este ilustrat în figura 6. Pentru acest mod, se observă că primul circuit numără impulsurile provenite de la semnalul de ceas, iar oricare alt circuit contorizează de câte ori s-a umplut circuitul anterior.

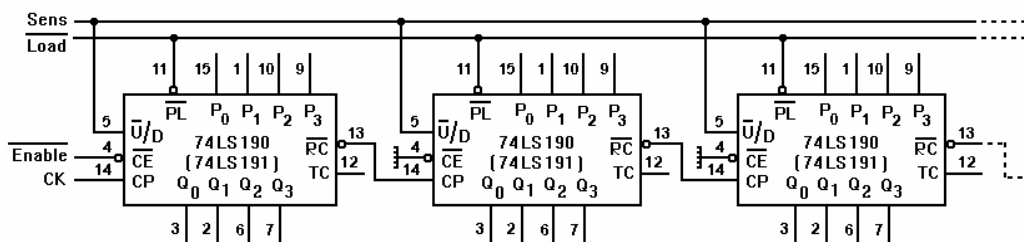


Fig. 6. Cascadarea numărătoarelor sincrone în modul *Ripple Clock*

Referitor la acest mod de lucru se pot face următoarele observații:

- pe ansamblu, schema pierde caracterul de numărare sincronă deși toate numărătoarele din structură sunt sincrone;
- schema este utilă mai ales acolo unde capacitatea de pilotare a semnalului de ceas este redusă, în acest caz semnalul de ceas comandă numai intrarea primului circuit;
- semnalul de validare a numărării,  $\overline{Enable}$ , este suficient să se aplice primului circuit deoarece, pentru  $\overline{CE}=1$  se blochează generarea pusului de zero pe ieșirea  $\overline{RC}$ ;
- schema prezintă dezavantajul unui decalaj temporal între schimbarea stării primului și a ultimului circuit din lanțul de cascaderă.

• Modul **Ripple Carry/Borrow**, prezentat în figura 7 prezintă următoarele proprietăți:

- pe ansamblu, caracterul numărării este sincron deoarece semnalul de ceas se aplică simultan tuturor circuitelor din lanțul de cascaderă;
- durata de zero a semnalului de ceas, CK, trebuie să fie suficient de mare pentru a permite tranziției negative de pe  $\overline{RC}$ , să străbată întreg lanțul de circuite înainte de începerea duratei de unu a semnalului de ceas;
- asupra duratei de unu a semnalului de ceas nu se fac restricții.

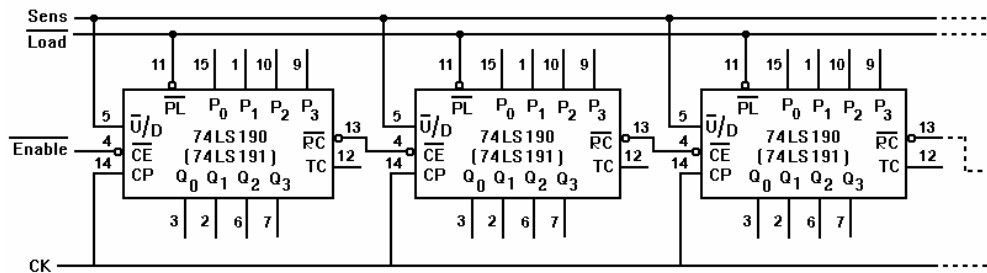


Fig. 7. Cascadarea numărătoarelor sincrone în modul *Ripple Carry/Borrow*

• Modul **Parallel Gated Carry/Borrow**, este prezentat în figura 8 și este caracterizat de următoarele proprietăți:

- schema păstrează caracterul sincron al numărării;
- semnalul de validare a numărării trebuie inclus în fiecare poartă NAND deoarece ieșirea TC a unui circuit nu depinde de intrarea  $\overline{CE}$  proprie.

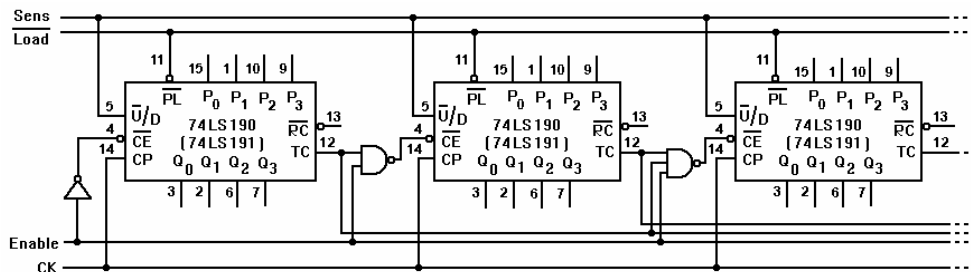


Fig. 8. Cascadarea numărătoarelor sincrone în modul *Parallel Gated Carry/Borrow*

## 2.6. Utilizarea limbajului VHDL pentru descrierea numărătoarelor

În această secțiune prezentăm câteva modalități de descriere în limbaj VHDL a structurilor de numărare.

Din punct de vedere didactic, pentru comanda intrării de ceas a numărătorului studiat se recomandă utilizarea unui semnal digital cu frecvență foarte mică (pentru a putea urmări succesiunea stărilor prin care trece numărătorul) sau utilizarea unui așa zis "ceas manual".

Pentru machetele din laborator, obținerea unui semnal cu frecvență foarte mică necesită o divizare suficient de mare a semnalului cu frecvența de 25,175MHz. Acest semnal este generat de un oscilator de pe machetă și este conectat la pinul P9 al circuitului CPLD. Divizarea acestui semnal se poate face prin introducerea unui proces special destinat acestui scop. Spre exemplu, dacă semnalul de la oscilator este aplicat la intrarea unui numărător asincron de 24 biți, frecvența semnalului cules la cel mai semnificativ bit al numărătorului este de aproximativ 1,5Hz. Modul de intercalare a numărătorului se poate vedea în exemplele de mai jos analizând procesul denumit *div\_ceas*.

"Ceasul manual" este un semnal digital obținut prin închiderea sau deschiderea unui comutator mecanic. Din nefericire, închiderea și deschiderea contactelor mecanice se face cu vibrații – lucru ce face ca semnalul electric generat să aibă scurte oscilații în zona în care se face trecerea de la o stare logică la alta. Din această cauză, există riscul de a genera mai multe tranziții active la o singură apăsare a contactul mecanic. Contactele cu revenire de pe machetă sunt "curățite" de tranzițiile suplimentare prin intermediul unui inversor trigger Schmitt, ceea ce nu reprezintă cea mai bună soluție, de aceea ele trebuie utilizate cu precauție în comanda intrărilor de ceas.

În cele ce urmează prezentăm câteva exemple de descriere în limbaj VHDL a numărătoarelor sincrone.

◆ **Exemplul 1:** Descrierea în limbaj VHDL a unui numărător binar pe 4 biți, cu intrare de ștergere activă pe unu logic și intrarea de numărare sensibilă la tranziția pozitivă.

Observații	Codul VHDL
Secțiune dedicată includerii de librării	<pre>library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_arith.all; use IEEE.std_logic_unsigned.all;</pre>
<p>Secțiune dedicată descrierii entității.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- La intrarea <b>f_in</b> se aplică semnalul de la oscilatorul machetei de test;</li> <li>- intrarea de ștergere: <b>clear</b>;</li> <li>- intrarea de ceas a numărătorului va fi comandată de semnalul de la ieșirea divizorului de frecvență descris prin procesul <b>div_ceas</b>, semnal denumit <b>clk</b>;</li> <li>- ieșirile numărătorului: <b>Q_out</b>;</li> </ul>	<pre>entity nr_4bit is port ( f_in, clear : in std_logic; Q_out: out std_logic_vector(3 downto 0) ); end nr_4bit;</pre>
<p>Secțiune dedicată descrierii arhitecturii.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- Descrierea funcționării numărătorului se face cu procesul <b>pr_A</b> ce este sensibil la intrările <b>clk</b> și <b>clear</b>;</li> <li>- Ștergerea numărătorului se face dacă se activează intrarea <b>clear</b>;</li> <li>- Incrementarea se face pe fiecare tranziție pozitivă a semnalului de ceas;</li> <li>- Procesul <b>div_ceas</b> și declarația <b>clk &lt;= cnt(23)</b> sunt introduse pentru a ușura vizualizarea stărilor prin care trece numărătorului.</li> <li>- Ordinea în care sunt scrise procesele în cadrul arhitecturii nu are importanță.</li> </ul>	<pre>architecture arh_nr4bit of nr_4bit is signal clk: std_logic; signal cnt : std_logic_vector (23 downto 0); begin pr_A: process (clk, clear) begin if (clear = '1') then Q_out &lt;= '0000'; elsif ( clk 'event and clk = '1') then Q_out &lt;= Q_out + "0001"; end if; end process pr_A ;  div_ceas: process (f_in) -- process divizare ceas begin if (f_in 'event and f_in='1') then cnt &lt;= cnt + 1; end if ; end process div_ceas; clk &lt;= cnt(23); end arh_nr4bit ;</pre>

◆ **Exemplul 2:** Descrierea în limbaj VHDL a unui numărător BCD cu două decade, cu intrare de ștergere activă pe unu logic și intrarea de numărare sensibilă la tranziția pozitivă.

Observații	Codul VHDL
Secțiune dedicată includerii de librării	<pre>-- Numarator BCD pe doua decade library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_arith.all; use IEEE.std_logic_unsigned.all;</pre>
<p>Secțiune dedicată descrierii entității.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- intrarea de ștergere: <b>clear</b>;</li> <li>- La intrarea <b>f_in</b> se aplică semnalul de la oscilatorul machetei de test;</li> <li>- intrarea de ceas a numărătorului va fi comandată de semnalul de la ieșirea divizorului de frecvență descris prin procesul <b>div_ceas</b>, semnal denumit <b>clk</b>;</li> <li>- ieșiri: <b>bcd_unit</b>, <b>bcb_zeci</b>;</li> </ul>	<pre>entity nr_2dec is port ( f_in, clear : in std_logic; bcd_unit: out std_logic_vector(3 downto 0); bcd_zeci: out std_logic_vector(3 downto 0) ); end nr_2dec ;</pre>
<p>Secțiune dedicată descrierii arhitecturii.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- Descrierea funcționării numărătorului zecimal cu două decade se face cu două procese: <b>pr_unit</b> și <b>pr_zeci</b> pentru modificarea codului BCD al unităților respectiv al zecilor;</li> <li>- Pentru ambele procese, ștergerea decadei se face dacă se activează intrarea <b>clear</b> sau dacă sa atins starea 1010;</li> <li>- Incrementarea unităților se face pe fiecare tranziție pozitivă a semnalului de ceas;</li> <li>- Semnalul <b>cy</b> are semnificația de semnal de umplere a decadei de unități (trece în starea 0 atunci când numărătorul este în starea maximă);</li> <li>- Incrementarea zecilor se face tot pe tranziția pozitivă a semnalului de ceas numai dacă decada de unități este plină;</li> </ul>	<pre>architecture arh_nr2dec of nr_2dec is signal clk, cy: std_logic; signal cnt : std_logic_vector (23 downto 0); signal unit, zeci : std_logic_vector (3 downto 0); begin pr_unit: process (clear, clk) begin if ((clear = '1') or (unit = "1010")) then unit &lt;= "0000" ; elsif rising_edge(clk) then unit &lt;= unit + "0001" ; end if; if unit = "1001" then cy &lt;= '0'; else cy &lt;= '1'; end if; bcd_unit &lt;= unit; end process pr_unit ; pr_zeci: process (clear, clk) begin if ((clear = '1') or (zeci = "1010")) then zeci &lt;=</pre>



<ul style="list-style-type: none"> <li>- Procesul <b>div_ces</b> și declarația <b>clk &lt;= cnt(23)</b> sunt introduse pentru a ușura vizualizarea stărilor prin care trece numărătorului.</li> <li>- Ordinea în care sunt scrise procesele în cadrul arhitecturii nu are importanță.</li> </ul>	<pre> "0000" ;     elsif (rising_edge(clk) and (cy = '0')) then         zeci &lt;= zeci+ "0001" ;     end if;     bcd_zeci&lt;=zeci; end process pr_zeci; div_ces: process (f_in) -- process divizare ceas begin     if (f_in 'event and f_in='1') then cnt &lt;= cnt + 1; end if; end process div_ces; clk &lt;= cnt(23); end arh_nr2dec ;         </pre>
--	--

♦ **Exemplul 3:** Descrierea în limbaj VHDL a unui numărător sincron presetabil pe 4biți. Intrarea de ștergere și cea de încărcare paralelă sunt active pe unu logic și au o execuție sincronă cu semnalul de ceas. Pentru descriere s-au folosit 2 procese: unul este responsabil de reactualizarea stării numărătorului iar celălalt de sincronizarea cu semnalul de ceas.

Observații	Codul VHDL
<p><b>Package</b> -ul conține elemente ce trebuie recunoscute în mai multe proiecte.</p> <p>Elementele dintr-un pachet sunt recunoscute într-un proiect dacă se face apel la instrucțiunea <b>use</b>.</p> <p>Pentru introducerea pachetului în proiect se adaugă un nou fișier sursă de tip <b>VHDL Packadge</b> cu descrierea de pe coloana alăturată.</p> <p>În exemplul de față, pachetul este folosit doar pentru a specifica formatul <b>bit4</b>.</p>	<pre> -- descrierea pachetului library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_unsigned.all; package count_types is     subtype bit4 is std_logic_vector(3 downto 0); end count_types;         </pre>
<p>Secțiune dedicată includerii de librării.</p> <p><i>Se remarcă faptul că se include și pachetul descris mai sus.</i></p>	<pre> -- Exemplu de numarator sincron library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_unsigned.all; use WORK.count_types.all;         </pre>
<p>Secțiune dedicată descrierii entității.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- intrări de date: <b>din</b>;</li> <li>- ieșiri: <b>dout</b>;</li> </ul>	<pre> entity nr_A is port (clk, clear, load : in std_logic;       din : in bit4;       dout : inout bit4); end nr_A;         </pre>
<p>Secțiune dedicată descrierii arhitecturii.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- procesul <b>pr_1</b> este responsabil de menținerea stării numărătorului;</li> <li>- procesul: <b>pr_2</b> este folosit pentru transferul stării numărătorului la ieșire – transfer ce se execută numai pe tranziția pozitivă a semnalului de ceas;</li> <li>- procesul <b>pr2</b> nu are listă de sensibilități de aceea este absolut necesară folosirea declarației <b>wait until</b>;</li> </ul>	<pre> architecture arh_A of nr_A is signal count_val: bit4; begin pr_1: process (clear, load, din, dout) begin     if load = '1' then count_val &lt;= din ;     elsif clear = '1' then count_val &lt;= "0000" ;     else count_val &lt;= dout + "0001" ;     end if; end process pr_1 ; pr_2: process begin     wait until clk 'event and clk ='1';     dout &lt;= count_val ; end process pr_2 ; end arh_A ;         </pre>

♦ **Exemplul 4:** Descrierea în limbaj VHDL a unui numărător sincron presetabil pe 4 biți. Ștergere este asincronă iar încărcarea paralelă sincronă.

Atenție: În acest exemplu nu este prezentat procesul pentru reducerea frecvenței semnalului dat de oscilatorul de pe macheta de test. Introducerea acestui proces se face similar ca în primul exemplu.

Observații	Codul VHDL
<p>Secțiune dedicată includerii de librării.</p>	<pre> -- Exemplu de numarator sincron library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_unsigned.all;         </pre>
<p>Secțiune dedicată descrierii entității.</p> <p>În cazul de față:</p> <ul style="list-style-type: none"> <li>- comandă încărcare paralelă: <b>load</b>;</li> <li>- intrări de date: <b>Data</b>;</li> <li>- ieșiri: <b>Count</b>;</li> </ul>	<pre> entity num_4bit is port (Clk,Rst,Load: in std_logic;       Data: in std_logic_vector(3 downto 0);       Count: out std_logic_vector(3 downto 0)); end num_4bit;         </pre>
<p>Secțiune dedicată descrierii arhitecturii.</p> <p>În cazul de față:</p>	<pre> architecture arh_num_4bit of num_4bit is begin         </pre>

- procesul este declanșat pentru orice eveniment apărut pe intrările **Rst** și **Clk**;
- Ștergerea se face pe unu logic și se execută independent de semnalul de ceas (execuție asincronă);
- Starea numărătorului este reactualizată printr-o declarație de atribuire condiționată;
- Se observă utilizarea unei bucle **for** pentru copierea intrărilor de date;
- Detecția frontului pozitiv se face folosind **rising\_edge(Clk)** și nu **Clk ' event AND Clk='1'**;
- Proprietatea **rising\_edge(Clk)** întoarce o valoare booleană, deci poate fi folosită în declarațiile ce conțin testarea îndeplinirii unor condiții;

```

process (Rst, Clk)
variable Q: std_logic_vector (3 downto 0);
begin
    if Rst = '1' then Q := "0000";
    elsif rising_edge(Clk) then
        if Load = '1' then
            for i in 3 downto 0 loop
                Q(i) := Data(i);
            end loop;
        elsif Q = "1111" then Q := "0000";
        else Q := Q + "0001";
        end if;
    end if;
    Count <= Q;
end process;
end arh_num_4bit;
    
```



### 3. Desfășurarea lucrării

#### 3.1. Implementarea divizoarelor de frecvență cu ajutorul numărătoarelor sincrone

**A.** Se realizează pe macheta de test schema din figura 9, după care se vizualizează formele de undă în punctele A, B, C, D, E, F, și G pentru situațiile următoare: a) comutatorul K este pe poziția 1; b) comutatorul K este pe poziția 2. Desenați corelat în timp formele de undă pentru fiecare subpunct în parte.

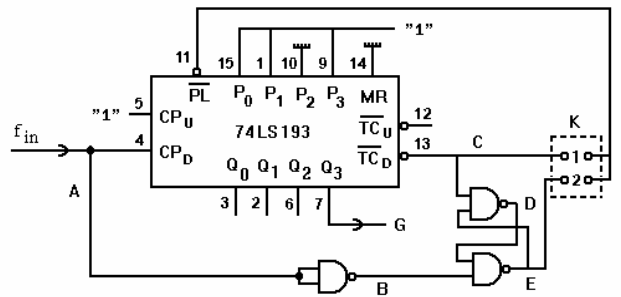


Fig. 9.

#### Întrebări:

- Care este factorul de divizare în frecvență realizat de circuit dacă semnalul de ieșire se culege din punctul G ?
- Explicați funcționarea schemei pentru cele două poziții ale comutatorului K. Ce diferențe apar în funcționare și cum se explică acestea ?
- Care este rolul latch-ului realizat cu porți NAND ?
- Ce observații puteți face în legătură cu durata de zero a semnalului din punctul C pentru cele două poziții ale comutatorului ?
- Care sunt stările prin care trece numărătorul pentru cele două cazuri considerate ?

**B.** Determinați stările prin care trec numărătoarele din schemele prezentate în figura 10.

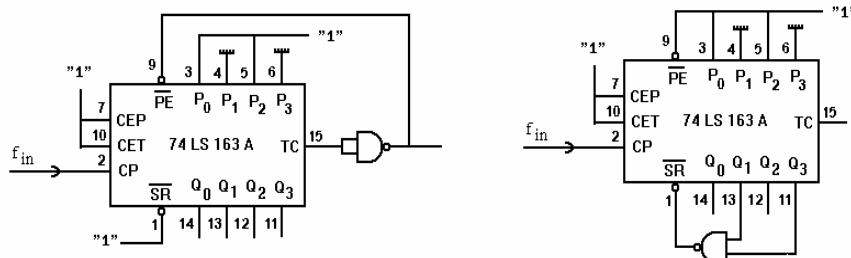


Fig. 10.

#### Întrebări:

- Care este factorul de divizare în frecvență al celor două circuite ? Cum explicați rezultatele obținute ?
- Care va fi noul factor de divizare dacă circuitul 74LS163 se înlocuiește cu 74LS162 ?
- Care este principiul de funcționare al acestor divizoare ?
- Se modifică funcționarea schemelor inițiale dacă circuitul 74LS163 se înlocuiește cu circuitul 74LS161 ?
- Cum se poate modifica factorul de divizare a montajelor anterioare ?
- Care este schema electrică, în cele două variante, pentru realizarea unor divizoare de frecvență cu 13 ?

C. Un alt procedeu de realizare a divizoarelor de frecvență este ilustrat în figura 11. Se aplică la intrarea un semnal TTL cu o frecvență de circa 100 KHz după care se vizualizează cu un osciloscop cu două spoturi semnalul de intrare și cel de ieșire. Desenați corelat în timp aceste semnale.

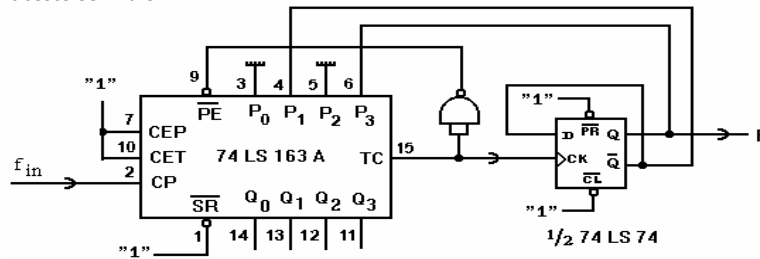


Fig. 11.

Întrebări:

- Cum funcționează această schemă ?
- Care este factorul de divizare în frecvență realizat de această schemă ?
- Care este factorul de umplere al semnalului din punctul F ? Cine impune acest factor ?
- Se modifică divizarea dacă se întrerupe legătura între ieșirea Q a bistabilului și intrarea P3 a numărătorului ? Dacă da, care este noul factor de divizare ?
- Ce se modifică în funcționarea schemei dacă circuitul LS163 este înlocuit cu LS161 ?

D. O generalizarea a schemei de la punctul anterior este prezentată în figura 12.

Întrebări:

- Cum se poate controla factorul de divizare în frecvență al schemei ? Dar factorul de umplere al semnalului de ieșire din punctul F ?
- Depinde funcționarea schemei de starea inițială a bistabilului ? Dar de tranziția activă a acestuia ?
- Cum se transpune această schemă pentru circuite 74LS 191 ?

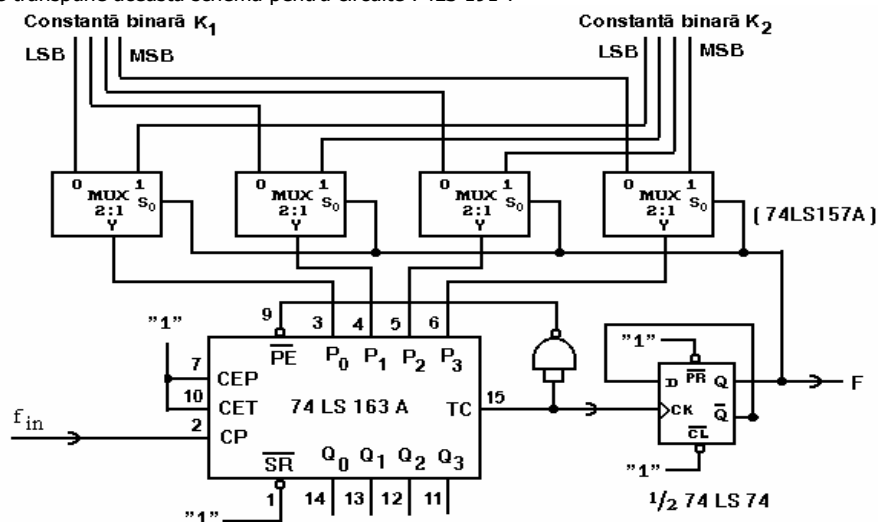


Fig. 12.

### 3.2. Utilizarea ISE WebPack pentru descrierea aplicațiilor sub formă de scheme logice

- Realizați o implementare a schemei din figura 1 și verificați funcționarea acesteia pe macheta de laborator cu CPLD. Urmăriți succesiunea stărilor prin care trece numărătorul (pe bareta de LED-uri).
- Adăugați la proiectul din tabelul 2 un decodificator BCD-7segmente pentru afișarea stării numărătorului pe primul digit al machetei de laborator. Notați succesiunea stărilor prin care trece numărătorul.
- Intervenți asupra schemei și schimbați tranziția activă a bistabililor (prin introducerea de inversoare în fața intrărilor ce ceas). Refaceți implementarea și urmăriți succesiunea stărilor prin care trece numărătorul. Ce se constată față de cazul anterior?
- Realizați câte o implementare pentru fiecare schemă de divizare a frecvenței din figura 10 și verificați funcționarea acestora pe macheta de laborator cu CPLD. Urmăriți succesiunea stărilor prin care trece numărătorul pe primul afișaj cu 7 segmente. Notați succesiunea stărilor și determinați factorul de divizare al frecvenței semnalului de intrare. *Atenție: în simbolul din editorul de scheme există diferențe în denumirea pinilor, față de cele prezentate în figura 2. Astfel: CEP → ENP; CET → ENT; TC → RCO; PL → LOAD; CP → CK;*
- Realizați o implementare pentru schema de divizare a frecvenței din figura 11 și verificați funcționarea acestora pe macheta de laborator cu CPLD. Urmăriți succesiunea stărilor prin care trece numărătorul pe primul afișaj cu 7 segmente. Notați succesiunea stărilor și determinați factorul de divizare al frecvenței semnalului de intrare.

- F. Realizați o implementare pentru schema de divizare a frecvenței din figura 12 și verificați funcționarea acestora pe macheta de laborator cu CPLD. Urmăriți succesiunea stărilor prin care trece numărătorul pe primul afișaj cu 7 segmente. Notați succesiunea stărilor și determinați factorul de divizare al frecvenței semnalului de intrare.

### 3.3. Utilizarea limbajului VHDL

- A. Verificați și implementați pe macheta de laborator descrierea numărătorului BCD pe două decade prezentat în exemplul 2.
- B. Modificați descrierea VHDL astfel încât să nu mai fie nevoie de semnalul **cy**. Verificați și implementați pe machetă noul cod.
- C. Plecând de la descrierea inițială prezentată în exemplul 2, adăugați un semnal de semnalizare a umplerii numărătorului de zeci și unul de semnalizare a umplerii întregului numărător (trecerea sa prin starea 99). Verificați și implementați pe machetă noul cod.
- D. Verificați și implementați pe macheta de laborator descrierea numărătorului sincron pe 4 biți prezentat în exemplul 4. Verificați dacă încărcarea paralelă se poate face fără tranziție pe intrarea de ceas.
- E. Realizați o descriere VHDL, de tip comportamental, pentru un divizor de frecvență cu 25 știind că semnalul de ieșire trebuie să aibă durata de unu logic egală cu 3 perioade ale semnalului de ceas. Verificați această descriere pe macheta de laborator cu CPLD.

## 4. Indicații privind modul de lucru

Pentru fiecare aplicație este necesară deschiderea unui nou proiect după metodologia prezentată într-o lucrare de laborator anterioară.

Toate aplicațiile din această lucrare necesită doar un singur fișier sursă (de tip schemă logică sau VHDL) și un singur fișier de constrângeri.

Referitor la comanda intrării de ceas facem următoarele precizări:

- Intrarea de ceas se poate comanda printr-un buton cu revenire (spre exemplu BTN1). Deoarece există riscul apariției de oscilații la apăsarea butonului, se recomandă urmărirea stărilor prin care trece numărătorul pentru mai multe cicluri complete ale sale;
- O metodă și mai bună de comandă a intrării de ceas constă în folosirea unui semnal digital periodic cu frecvență suficient de mică pentru a putea urmări succesiunea stărilor prin care trece numărătorul. Pentru aceasta, putem folosi semnalul de la pinul P9 al CPLD, semnal ce are o frecvență de 25,175MHz.

În schema logică, între pinul P9 și intrarea de ceas a numărătorului testat intercalăm un numărător binar pe 24 de biți. În aceste condiții, semnalul cules de pe ieșirea cea mai semnificativă a numărătorului va avea frecvența de cca. 1,5Hz, valoare ce ne permite vizualizarea stărilor prin care trece numărătorul.

În fișierul VHDL, divizarea semnalului de intrare de 25,175MHz se face prin introducerea unui proces (cazul procesului **div\_ceas** din exemplele 1 și 2).

Intrarea de reset se comandă prin BTN7;

Intrarea de încărcare paralelă se comandă printr-un switch;

Afișarea stării numărătorului se face pe LED-uri sau pe afișaje cu 7 segmente;

Afișarea stării logice a ieșirilor de semnalizare ale numărătorului se face pe LED-urile rămase nefolosite;

Fișierul de constrângeri este similar celui folosit în lucrarea anterioară;



## Lucrarea nr. 9: **Gestionarea unei matrice de taste cu organizarea 4x4**

### 1. Scopul lucrării

În această lucrare se prezintă funcționarea unui sistem logic ceva mai complex: este vorba de un sistem de gestionare a unei matrice de taste cu organizarea 4x4. Pe acest exemplu concret se pot vedea la lucru mai multe tipuri de circuite logice elementare precum: DCD, MUX, numărătoare binare etc.

În lucrare sunt prezentate două modalități de implementare: una clasică (în sensul că sunt folosite circuite integrate digitale de complexitate mică și medie) și una modernă (circuitul este descris în VHDL și implementat într-un circuit de tip CPLD).

### 2. Considerente teoretice

#### 2.1. Metode de codificare a unei matrice de taste

Tastatura a fost și probabil va rămâne pentru încă mult timp un mijloc facil prin care se pot introduce comenzi într-un sistem digital, sau informații într-un sistem de calcul. De regulă, pentru fiecare tastă, prin convenție, este asociat un cod binar prin intermediul căreia ea poate fi recunoscută de către sistemul de calcul. În funcție de complexitatea și mărimea tastaturii există o mulțime de soluții de implementare a unei astfel de aplicații.

În cele ce urmează sugerăm câteva idei de implementare pentru cazul unei tastaturi alcătuită din 16 taste, considerând doar situațiile de apăsare a unei singure taste.

#### Soluția 1

O primă soluție pentru codificarea celor 16 taste ar fi conceperea unui CLC cu 16 intrări (câte una pentru fiecare tastă) și 5 ieșiri (4 pentru codul binar al tastei apăsate și una pentru semnalizarea evenimentului "tastă apăsată"). Se poate concepe schema electrică astfel neapăsarea tastei să mențină intrarea CLC-ului în unu logic, iar apăsarea să aducă respectiva intrare în zero.

Dacă mergem pe această idee, constatăm destul de repede că tabelul de adevăr este foarte mare: avem de implementat 5 funcții binare ce depinde de 16 variabile. Complexitatea sistemului rezultat este prea mare pentru ca această metodă să fie acceptată din punct de vedere practic.

Această metodă prezintă totuși avantajul că permite, prin extinderea numărului de ieșiri și prin alcătuirea corectă a tabelului de adevăr, generarea de coduri binare distincte și pentru situații în care două sau mai multe taste sunt apăsate.

#### Soluția 2

O altă soluție, ține cont de câteva aspecte practice legate de utilizarea tastaturilor, mai precis de viteza finită de acționare a tastelor:

- durata de apăsare a unei taste este de câteva zeci de milisecunde ( $1\text{m}=10^{-3}\text{s}$ ), chiar dacă ne străduim să facem o apăsare foarte scurtă;
- intervalul de timp dintre două apăsări este de ordinul zecilor de milisecunde chiar și pentru cea mai rapidă secretară;

Ținând cont de aceste aspecte rezultă că nu este nevoie ca circuitul să analizeze în același timp starea logică a tuturor tastelor. Se poate imagina un circuit care să analizeze succesiv starea tastelor: se analizează mai întâi starea primei taste, apoi starea următoarei taste și așa mai departe până se ajunge la ultima tastă după care procesul se repetă la nesfârșit (atâta timp cât circuitul este alimentat). Pentru acest mod de lucru, frecvența de trecere de la o tastă la alta trebuie să fie suficient de mare pentru ca întreaga tastatură să fie verificată cel puțin odată pentru cel mai mic interval de apăsare a unei taste (acesta se poate determina experimental și este dependent și de viteza de reacție a utilizatorului tastaturii).

Schema de principiu a unui astfel de circuit este prezentată în figura 1, pentru cazul unei tastaturi formată din 4 taste. Blocul funcțional denumit "Logică de control", este responsabil de generarea unor coduri binare pe doi biți ( $Q_1 Q_0$ ), cu următoarea succesiune: ... 00 → 01 → 10 → 11 → 00 → 01 ... Aceste coduri binare sunt folosite pentru comanda intrărilor de selecție ale circuitului MUX 4:1. În acest mod, schimbarea codului binar înseamnă de fapt analizarea stării logice pentru o altă tastă.

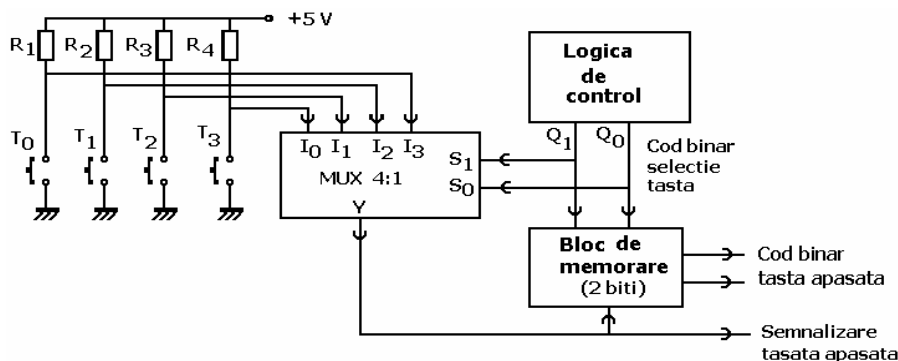


Fig. 1. Schema logică de principiu pentru gestionarea unei tastaturi cu 4 taste

Pentru explicarea funcționării vom considera că  $Q_1 Q_0 = 10$ . În aceste condiții, la ieșirea Y a multiplexorului vom regăsi starea logică de la intrarea de date  $I_2$ . Datorită schemei electrice, starea logică a intrării de date  $I_2$  este dependentă de starea tastei  $T_2$ :

- dacă  $T_2$  nu este apăsată, intrarea de date  $I_2$  se află conectată la unu logic prin intermediul rezistenței  $R_3$ , în consecință  $Y=1$ ;
- dacă tasta este apăsată, intrarea de date  $I_2$  este conectată la masă prin închiderea contactului  $T_2$ , în consecință  $Y=0$ ;

După ce a expirat timpul alocat combinației  $Q_1 Q_0 = 10$ , logica de control schimbă codul (se trece la combinația  $Q_1 Q_0 = 11$ ) și astfel se testează starea logică a tastei următoare (în cazul de față, tasta  $T_3$ ).

Indiferent de codul de selecție generat de logica de control, trecerea în zero logic a ieșirii multiplexorului are semnificația de *tastă apăsată*. În consecință, această ieșire activă pe zero logic, poate fi folosită ca ieșire de semnalizare spre sistemul de calcul. Totodată, trecerea în zero a ieșirii Y mai este folosită și pentru generarea unei comenzi de memorare a codului de selecție al tastei apăsate, în blocul de memorie.

Așa cum este prezentată în figura 1, schema prezintă câteva limitări:

- nu se pot genera coduri pentru apăsarea unor combinații de două sau mai multe taste;
- generează semnalizări multiple pentru o singură apăsare a unei taste;
- mărirea numărului de taste necesită mărirea numărului de intrări ale multiplexorului.

### Soluția 3

Pentru un număr mare de taste se folosește tot principiul prezentat anterior cu deosebirea că tastele sunt organizate într-o matrice de  $m$  linii și  $n$  coloane. Procedând astfel se reduce semnificativ numărul intrărilor de date ale circuitului de multiplexare.

O schema bloc de principiu, pentru cazul unei tastaturi cu 16 taste, se prezintă în figura 2. În primul rând trebuie să remarcăm că 16 taste necesită un cod binar de selecție pe 4 biți. Cei mai semnificativi 2 biți ( $Q_3 Q_2$ ) sunt folosiți pentru comanda intrărilor de selecție ale DCD iar restul de biți pentru comanda intrărilor de selecție ale MUX4:1.

Pentru fiecare tastă este necesar ca un capăt să poată fi conectat la unu logic iar celălalt la masă, cu alte cuvinte un capăt la +Vcc iar celălalt la masă. Organizarea tastelor în matrice este făcută astfel:

- Toate tastele de pe o coloană sunt conectate între ele și sunt conectate printr-o rezistență la +Vcc. Așadar, în orice moment de timp, pe coloane există valoarea logică unu.
- Toate tastele de pe o linie sunt conectate între ele și sunt comandate de către o ieșire (activă pe zero logic) a DCD-ului. Deoarece DCD-ul activează la un moment dat doar o singură ieșire, se poate trage concluzia că există doar o singură linie ce primește zero logic.

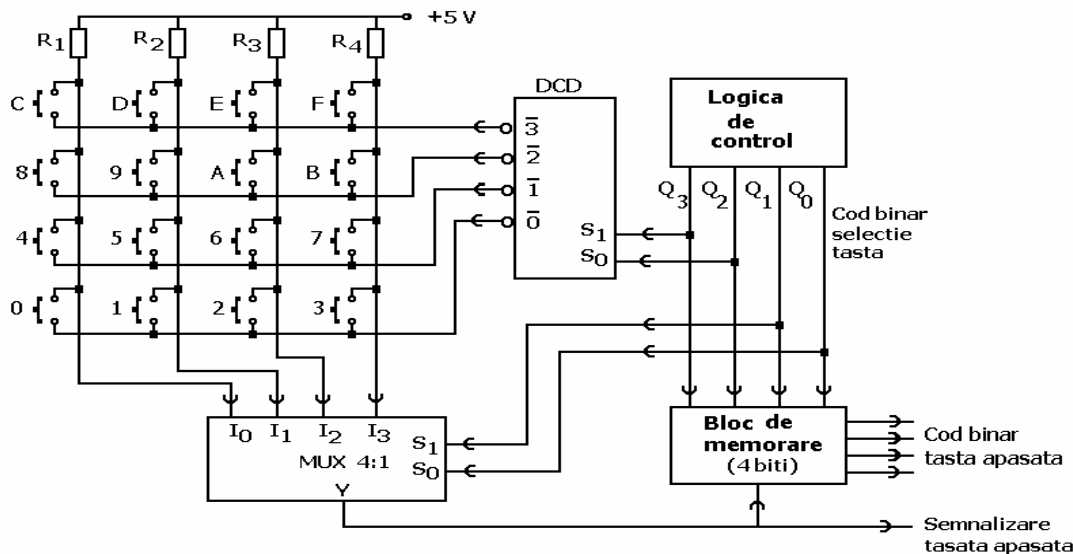


Fig. 2. Schema bloc de principiu pentru gestionarea unei matrice de taste cu organizarea 4 linii x 4 coloane

Pentru explicarea funcționării considerăm situația  $Q_3 Q_2 Q_1 Q_0 = 1001$ . Facem observația că  $Q_3 Q_2$  determină codul liniei (rândului) din matrice iar  $Q_1 Q_0$  codul coloanei din matrice.

- Intrarea de selecție a DCD primește codul binar  $S_1 S_0 = Q_3 Q_2 = 10$ , fapt ce determină activarea ieșirii  $\bar{2}$ . Așadar, se transmite un zero logic pe linia tastelor **8, 9, A, B**. Tasta verificată efectiv se stabilește prin adresa de coloană dată de biții  $Q_1 Q_0$ .
- Intrarea de selecție a MUX4:1 primește codul binar  $S_1 S_0 = Q_1 Q_0 = 01$ , fapt ce determină selecția intrării de date  $I_1$ . Aceasta înseamnă că se analizează starea tastei aflate la intersecția liniei 2 cu coloana 1, este vorba de tasta **9**. Atenție că numerotarea liniilor și a coloanelor începe de la 0!
- Dacă tasta **9** este neapăsată, pe intrarea de date  $I_1$  ajunge un unu logic datorat rezistenței  $R_2$ . Dacă tasta **9** este apăsată, ea face o legătură între ieșirea  $\bar{2}$  a DCD și intrarea  $I_1$ . Deoarece ieșirea  $\bar{2} = 0$ , rezultă că  $I_1 = 0$ .
- Așadar, pentru codul  $Q_3 Q_2 Q_1 Q_0 = 1001$ , se verifică starea tastei **9**.

- După expirarea timpului alocat codului  $Q_3 Q_2 Q_1 Q_0 = 1001$ , logica de control generează codul următor, adică  $Q_3 Q_2 Q_1 Q_0 = 1010$ . Pentru noul cod, funcționarea este similară cu deosebirea că se verifică starea tastei **A**.
- Ieșirea MUX4:1 poate fi privită ca o ieșire activă pe zero logic ce semnalizează evenimentele de tip "tastă apăsată". Totodată, această ieșire este folosită și pentru încărcarea codului de selecție tastă în blocul de memorie deoarece este vorba chiar de codul tastei apăsate.

Această schemă, așa cum este prezentată în figura 2, prezintă câteva limitări:

- nu poate sesiza apăsarea unor combinații de două sau mai multe taste;
- generează semnalizări multiple pentru o singură apăsare a unei taste;

Problema legată de semnalizarea multiplă a aceleiași taste apăsate este eliminată pe macheta de laborator, (vezi figura 3) printr-o metodă destul de simplă.

## 2.2. Exemplu de implementare în CPLD

Acest exemplu face o descriere a schemei din figura 1, pentru cazul în care avem 8 taste. Pentru schimbarea codurilor de selecție avem nevoie de un semnal cu frecvență între 10kHz ÷ 100kHz. Obținerea acestui semnal se poate face printr-un proces de divizare în frecvență a semnalului dat de oscilatorul de pe macheta de laborator cu CPLD, oscilator ce generează un semnal digital cu frecvența de 25,175MHz. Spre exemplu, dacă alegem o divizare în frecvență cu  $2^{10}$ , se obține un semnal cu frecvența de 24,584kHz.

Observații	Codul VHDL
Secțiune dedicată includerii de librării	-- Exemplu de implementare tastatura library IEEE; use IEEE.std_logic_1164.all;
Secțiune dedicată descrierii entității. În cazul de față: - nume entitate este: <b>tastatura</b> ; - vector cu intrare cu 8 componente ptr. conectare taste: <b>p_taste</b> ; - intrare de ceas: <b>p_clk_in</b> - ieșire ptr. semnalizare cod tastă verificată: <b>p_cod_sel</b> ; - ieșire ptr. semnalizare cod tastă apăsată: <b>p_cod_mem</b> ; - ieșire de semnalizate tastă apăsată: <b>p_signal</b> ;	entity tastatura is port ( p_taste:in std_logic_vector(7 downto 0); p_cod_sel:out std_logic_vector(2 downto 0); p_cod_mem:out std_logic_vector(2 downto 0); p_tap: out std_logic; p_clk_in : in std_logic); end tastatura;
Secțiune dedicată descrierii arhitecturii. În cazul de față: - se folosesc trei procese.  • Procesul <b>divizare</b> : - este folosit pentru divizarea semnalului de intrare cu frecvența de 25,175MHz. - procesul este sensibil doar la <b>p_clk_in</b> ; - constanta de divizare este $2^{10}$ deoarece se folosește un numărător pe 10 biți; - codul de selecție tastă este preluat direct de la cei mai importanți trei biți ai numărătorului <b>cnt</b> ;  • Procesul <b>selectie</b> : - este folosit pentru selecția tastei a cărei stare trebuie verificate; - procesul este sensibil doar la <b>cod_sel</b> ; - starea logică a tastei este copiată în semnalul <b>test</b> ; - fiecare cod de selecție alege o altă tastă de intrare;  • Procesul <b>atrb</b> : - este folosit pentru comanda semnalelor externe (cod selecție, codul ultimei taste apăsate, semnalizare tastă apăsată) ; - procesul este sensibil doar la <b>cod_test</b> ;	architecture arh_tastatura of tastatura is signal cod_sel:std_logic_vector(2 downto 0); signal cod_mem:std_logic_vector(2 downto 0); signal test: std_logic; begin -- proces pentru divizare ceas divizare: process (p_clk_in) variable cnt: std_logic_vector(9 downto 0); begin if p_clk_in'event and p_clk_in='1' cnt := cnt + 1; end if; cod_sel(2 downto 0) <= cnt(9 downto 7); end process divizare; -- proces pentru selectie tasta selectie: process (cod_sel) begin case cod_sel is when "000" => test <= p_taste(0); when "001" => test <= p_taste(1); when "010" => test <= p_taste(2); when "011" => test <= p_taste(3); when "100" => test <= p_taste(4); when "101" => test <= p_taste(5); when "110" => test <= p_taste(6); when "111" => test <= p_taste(7); when others => test <= '0'; end case; end process selectie; atrb: process (cod_sel) begin if test = '1' then cod_mem <= cod_sel; p_tap <= '1'; else

	<pre> p_tap &lt;='0'; end if; pin_cod_mem&lt;=cod_mem; pin_cod_sel&lt;=cod_sel; end process atrb; end arh_tastatura;</pre>
<p>Fișierul de constrângeri</p> <ul style="list-style-type: none"> <li>- tastele sunt implementate cu comutatoarele SW1÷SW8;</li> <li>- codul de selecție se afișează pe LED-urile LD1÷LD3;</li> <li>- codul ultimei taste apăsate se afișează pe LED-urile L6÷LD8;</li> <li>- semnalizarea evenimentului tastă apăsată se face pe LD5;</li> </ul>	<pre> NET "p_taste&lt;0&gt;" LOC = "P37"; NET "p_taste&lt;1&gt;" LOC = "P40"; NET "p_taste&lt;2&gt;" LOC = "P43"; NET "p_taste&lt;3&gt;" LOC = "P45"; NET "p_taste&lt;4&gt;" LOC = "P47"; NET "p_taste&lt;5&gt;" LOC = "P50"; NET "p_taste&lt;6&gt;" LOC = "P52"; NET "p_taste&lt;7&gt;" LOC = "P54"; NET "p_clk_in" LOC = "P9"; NET "p_cod_sel&lt;2&gt;" LOC = "P62"; NET "p_cod_sel&lt;1&gt;" LOC = "P65"; NET "p_cod_sel&lt;0&gt;" LOC = "P67"; NET "p_cod_mem&lt;2&gt;" LOC = "P75"; NET "p_cod_mem&lt;1&gt;" LOC = "P80"; NET "p_cod_mem&lt;0&gt;" LOC = "P82"; NET "p_tap" LOC = "P71";</pre>



### 3. Desfășurarea lucrării

#### 3.1. Studiul machetei cu componente discrete

##### A. Referitor la schema din figura 2, răspundeți la următoarele întrebări:

- Câte perioade de ceas durează un proces de verificare a tuturor tastelor ?
- Câte perioade de ceas este activă ieșirea de semnalizare "tastă apăsată" ?
- Cum se explică fenomenul de semnalizare multiplă a aceleiași taste apăsate ?
- Ce se întâmplă cu procesul de schimbarea a codurilor pe durata de timp în care tasta este apăsată ? Se continuă sau se oprește ? Cum ar fi mai bine ?
- Ce se întâmplă cu semnalul de ieșire dacă apăsarea unei taste este ceva mai lungă ?
- Care este intervalul maxim de timp (exprimat în perioade ale semnalului de ceas) dintre apăsarea propriu-zisă a tastei și activarea ieșirii de semnalizare "tastă apăsată" ?
- Ce cod se generează la apăsarea simultană a două taste ?

##### B. Referitor la schema din figura 3, macheta cu componente discrete din laborator, răspundeți la următoarele întrebări:

- Identificați circuitele și blocurile funcționale;
- Cum s-a rezolvat problema semnalizării multiple pentru apăsarea aceleiași taste ?
- Explicați de ce liniile matricei de taste sunt comandate de ieșirile  $\bar{0}$ ,  $\bar{2}$ ,  $\bar{4}$ ,  $\bar{6}$  ale decodificatorului ? Ce modificări trebuie efectuate în schema pentru a face comanda pe linii cu ieșirile  $\bar{0}$ ,  $\bar{1}$ ,  $\bar{2}$ ,  $\bar{3}$  ?
- Ce modificări trebuie efectuate în schema pentru ca informația de pe coloanele matricei de taste să fie preluate pe intrările D4, D5, D6, D7 ? Dar pentru intrările D0, D1, D6, D7 ?
- Care LED semnalizează bitul cel mai puțin semnificativ al codului binar al tastei verificate ?
- Ce rol au inversoarele ? Se poate concepe un bloc de semnalizare optică fără ele ?
- Ce rol are latch-ul din blocul funcțional denumit "ceas manual" ?
- Poarta NAND din oscilator, cu caracteristică de tip trigger Schmitt, poate fi înlocuită cu una cu caracteristică normală ?
- Poarta NAND de la intrarea număratorului poate fi înlocuită cu o poartă cu caracteristică normală ? Câte intrări sunt necesare pentru această poartă ? Care este starea logică de la ieșirea porții atunci când o tastă este menținută apăsată ?
- Prezentați o modalitate de memorare a codului binar al ultimei taste apăsate.

##### C. Determinări experimentale:

- Plasați comutatorul **K** pe poziția **a** și determinați cu osciloscopul frecvența semnalului de ceas (folosit pentru trecerea de la o tastă la alta); De ce toate LED-urile din blocul de semnalizare luminoasă a codului tastei selectate par în permanență aprinse?
- Plasați comutatorul **K** pe poziția **b** și acționați asupra tastelor și urmăriți secvența de verificare a acestora; Verificați starea logică de la ieșirea porții prin care trec impulsurile de ceas atunci când o tastă este menținută apăsată ?



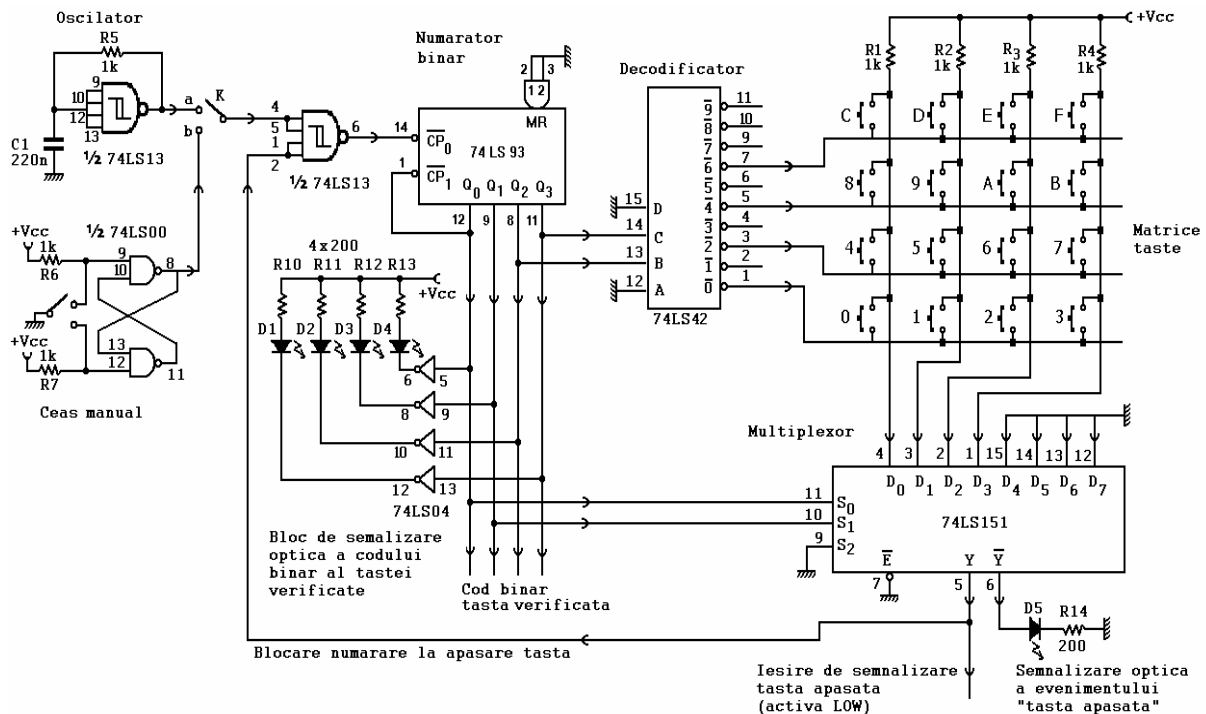


Fig. 3. Schema electrică completă a machetei de laborator reprezentând un sistem logic de gestionare a unei matrice de taste cu organizarea 4x4.

### 3.2. Studiul pe macheta cu CPLD.

- A.** Verificați codul VHDL din exemplul prezentat în secțiunea 2.2. pe macheta de laborator și răspundeți la următoarele întrebări:
- Cum explicați faptul că LED-urile ce afișează codul tastei verificate par mereu aprinse?
  - Mențineți o tastă apăsată și vizualizați cu osciloscopul catodic semnalul ce comandă LED-ul de semnalizare a evenimentului **tastă apăsată**. Ce observații puteți face în legătură cu numărul de semnalizări ale aceleiași taste apăsată?
- B.** Modificați codul VHDL prezentat în secțiunea 2.2. astfel încât procesul de scanare a restului de taste să fie oprit atunci când o tastă este apăsată.
- C.** Adăugați la proiectul anterior tot ce este necesar pentru a obține afișarea în zecimal (pe unul din afișajele cu 7 segmente) a codului tastei apăsată.
- D.** Folosind cunoștințele acumulate până în prezent realizați o aplicație mai amplă care să emuleze funcționarea unui calculator electronic pentru operația de adunare. Pentru aceasta se pleacă de la următoarele ipoteze:
- Gestionarea multiplexată în timp a tastaturii;
  - Semnificația tastelor este: 0 → SW1, 1 → SW2, 2 → SW3, 3 → SW4, + → SW5, = → SW6, CE → SW7 iar tasta SW8 nu are nici o semnificație;
  - Afișarea se face numai pe un singur digit (acest lucru este posibil deoarece cel mai mare număr este 4, deci rezultatul maxim al adunării este 8).
  - Imediat după pornire se afișează 0 iar după apăsarea unei taste se afișează operandul sau operația sau rezultatul operației. Pentru afișarea operației se va aprinde segmentul **f** al afișajului.



## Lucrarea nr. 10: **Sisteme de afișare a informației numerice**

### 1. Scopul lucrării

Lucrarea este destinată prezentării principalelor modalități de afișare a informației numerice, cu referințe concrete la un sistem de afișare cu 4 cifre zecimale ce funcționează pe principiul multiplexării în timp. Studiul acestui sistem de afișaj se face în două variante: o primă variantă se bazează pe implementarea cu circuite logice de complexitate medie iar cealaltă variantă presupune descrierea în limbaj VHDL și implementarea într-un CPLD.

Pentru prima dată în cadrul acestui laborator se folosește conceptul de proiectare ierarhică.

### 2. Considerente teoretice

#### 2.1. Codul BCD

Așa după cum este cunoscut, sistemele digitale lucrează doar cu două valori numerice: zero și unu. Din acest motiv, din punct de vedere tehnic este preferabil ca sistemul digital să lucreze într-un sistem de numerație cu baza 2 (sistemul binar). Pe de altă parte, utilizatorul (operatorul uman) este obișnuit cu sistemul zecimal. Pentru rezolvarea acestui conflict, prima idee care ne vine în minte ar fi introducerea de circuite codificatoare/decodificatoare care să facă conversia între cele două sisteme. Aceste circuite nu fac altceva decât să complice inutil partea hardware.

O soluție de compromis se pare că s-a obținut prin introducerea codului BCD (**B**inary **C**oded **D**ecimal), acesta folosește toate regulile sistemului de numerație **zecimal** dar scrierea (reprezentarea în sistemul de calcul) se face în **binar**. Cu alte cuvinte, în codul BCD, fiecare cifră zecimală este înlocuită de scrierea sa binară pe 4 biți. Codul BCD s-a dovedit atât de util încât s-au conceput circuite de numărare și circuite aritmetice capabile să lucreze în acest cod.

Pentru a pune în evidență diferențele dintre codul binar natural și codul BCD, în tabelul de mai jos sunt prezentate câteva aspecte semnificative:

	Cod BCD	Cod binar natural
Conversia spre sistemul zecimal	$\overbrace{1001}^9 \overbrace{0011}^3 = 93_{10}$	$1001\ 0011_2 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 128 + 16 + 2 + 1 = 147_{10}$
Conversia din sistemul zecimal	$193_{10} = \overbrace{0001}^1 \overbrace{1001}^9 \overbrace{0011}^3_{BCD}$	$193_{10} = 11000010_2$ Rezultatul se obține prin împărțiri succesive la 2 $193:2=96$ rest 1 $96:2=48$ rest 0 $48:2=24$ rest 0 $24:2=12$ rest 0 $12:2=6$ rest 0 $6:2=3$ rest 0 $3:2=1$ rest 1 $1:2=0$ rest 1

#### 2.2. Afișaje statice

Cea mai întâlnită metodă de afișare a unui număr zecimal constă în aprinderea sau stingerea convenabilă a unor segmente așezate după conturul cifrei 8. Cele 7 segmente pot fi realizate cu orice surse luminoase ce pot fi controlate prin mijloace electrice. Cel mai adesea se întâlnesc afișaje cu LED-uri, cu cristale lichide sau cu descărcări în gaze.

Referitor la afișajele cu LED-uri, pentru a reduce numărul de terminale dintr-o capsulă, acestea sunt fabricate în două variante: cu anod comun (AC), sau cu catod comun (KC). Schema electrică a celor două variante constructive de afișaj cu LED-uri, precum și modul de aranjare a segmentelor se prezintă în figura 1.

Comanda afișajelor cu 7 segmente se face cu ajutorul decodificatoarelor BCD - 7 segmente, circuite ce au fost studiate într-o lucrare anterioară. Trebuie precizat că există trei tipuri de decodificatoare BCD - 7 segmente:

a) prima categorie afișează toate cifrele hexazecimale așa cum sunt prezentate în figura 1;

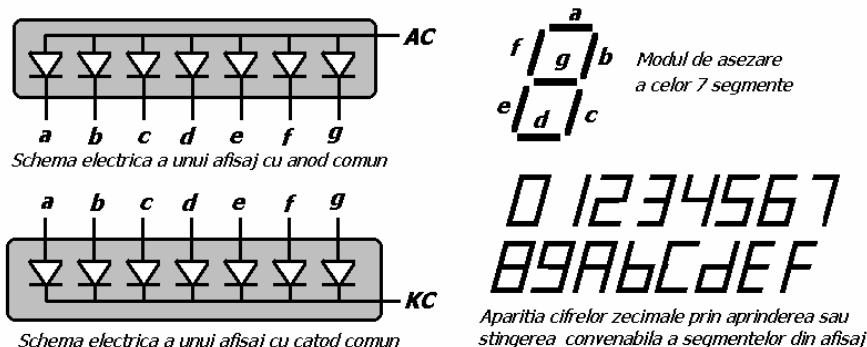


Fig.1: Afișaje cu 7 segmente implementate cu LED-uri

- b) altă categorie afișează doar cifrele zecimale de la 0 la 9, iar pentru restul codurilor mențin afișajul stins;  
 c) ultima categorie afișează corect cifrele zecimale de la 0 la 9 dar, pentru restul codurilor afișează caractere mai ciudate (acest comportament se explică prin faptul că fabricantul s-a folosit de codurile non-BCD pe 4 biți pentru a reduce complexitatea circuitului).

O modalitate de a obține un afișaj cu mai multe cifre, altfel spus cu mai mulți digiți, se arată în figura 2. Se observă că fiecare cifră zecimală are propriul sau decodificator BCD-7segmente, iar informația ce trebuie afișată (spre exemplu rezultatul unei măsurători) trebuie să fie în format BCD.

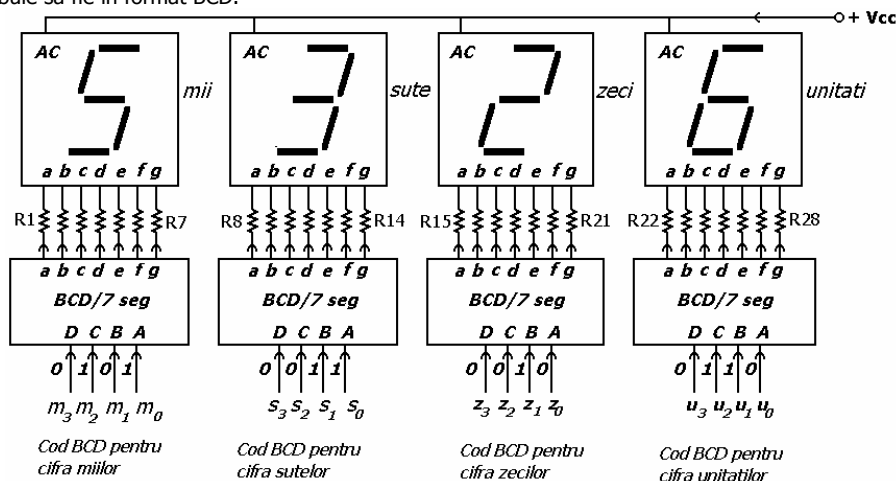


Fig.2. Schema bloc de principiu a unui afișaj static cu 4 digiți

#### Observații:

- decodificatoarele BCD-7segmente trebuie să aibă ieșirile active pe zero logic dacă afișajele au anodul comun;
- decodificatoarele BCD-7segmente trebuie să aibă ieșirile active pe unu logic dacă afișajele au catodul comun;
- rezistențele R1 ÷ R28 au rolul de a limita curentul prin LED-uri, fără aceste rezistențe există riscul distrugerii LED-urilor din afișaje;
- acest mod de lucru este neeconomic din punct de vedere al numărului de componente utilizat.

### 2.3. Afișaje dinamice

Înainte expunerii principiului de funcționare al afișării dinamice trebuie să facem o precizare legată de comportamentul ochiului uman: **dacă frecvența de stingere/aprindere a unei surse luminoase este peste o anumită limită (denumită valoare critică), ochiul percepe respectiva sursă ca fiind aprinsă în permanență.** În urma studiilor efectuate de specialiști, s-a constatat că această valoare limită este aproximativ 46Hz. În tehnică, din motive de siguranță, se consideră o valoare de 50Hz. Această particularitate a ochiului uman este exploatată de multe sisteme tehnice, cele mai cunoscute fiind cinematografia și televiziunea.

În cazul sistemelor cu afișare dinamică, se procedează astfel:

- cifrele nu mai sunt aprinse toate odată;
- în fiecare moment de timp este aprinsă doar o singură cifră;
- fiecare cifră este menținută aprinsă un interval scurt de timp (câteva milisecunde), același pentru toate cifrele;
- cifrele se aprind pe rând: un interval de timp cifra unităților, următorul interval de timp cifra zecilor și așa mai departe până când se ajunge la ultima cifră din afișaj după care procesul se repetă;
- dacă trecerea de la o cifră la alta se face suficient de repede, ochiul percepe întreg afișajul aprins. Spre exemplu, pentru un afișaj cu 4 digiți, frecvența de trece de la o cifră la alta trebuie să fie de cel puțin 200Hz (4 cifre × 50 Hz). Aceasta înseamnă că fiecare cifră este aprinsă un interval de timp egal cu  $(1/200)s = 5ms$ .

Schema bloc de principiu a unui afișaj dinamic cu 4 digiți se prezintă în figura 3. Facem precizarea că acest mod de lucru este adoptat mai ales pentru sistemele de măsură realizate în structuri integrate.

#### Funcționare:

- Oscilatorul este folosit pentru generarea unui semnal digital cu frecvența de 200Hz. Acest semnal, denumit semnal de ceas, indică momentele de timp în care trebuie făcută trecerea de la o cifră la alta;
- Semnalul cu frecvența de 200Hz, este preluat de un numărător binar care generează următoarea secvență ciclică de coduri binare pe 2 biți: 00 → 01 → 10 → 11 → 00 → 01 ..., așa cum se prezintă și în figura 4.
- Fiecare cod binar este menținut neschimbat un interval de timp egal cu o perioadă a semnalului dat de oscilator, în cazul de față 5ms (vezi figura 4).
- Tranzistoarele pnp din figura 3 sunt folosite pe post de comutatoare electronice, ele cuplează sau decuplează anodul comun al digiților la polul pozitiv al tensiunii de alimentare.
- Tranzistoarele lucrează în regim de comutație (regim special în care tranzistorul prezintă doar două stări: tranzistor saturat = contact electric închis, respectiv tranzistor blocat = contact electric deschis).

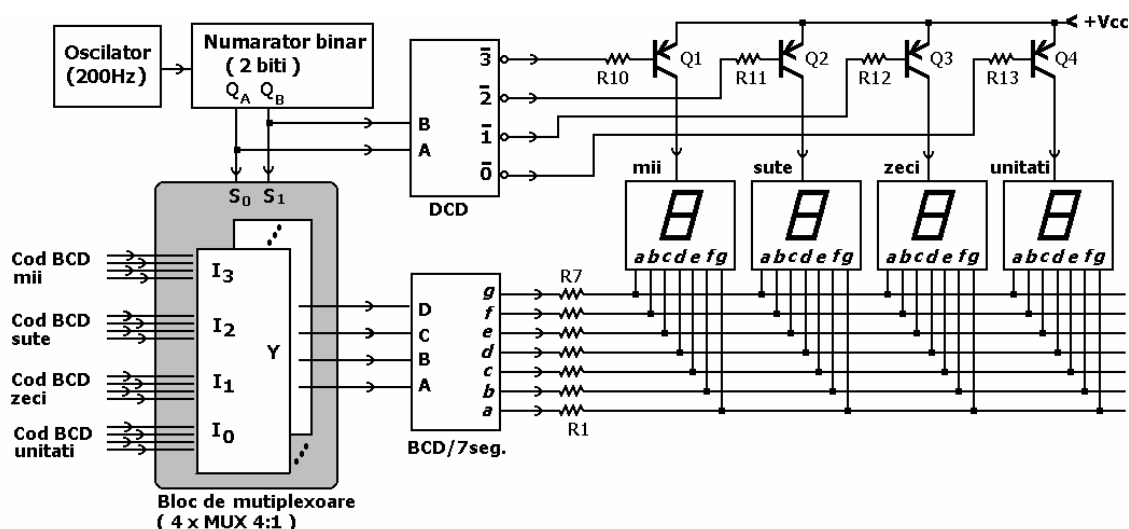


Fig. 3. Schema bloc de principiu a unui afișaj cu 4 digiți folosind multiplexarea în timp

- Pentru modul de conectare al tranzistoarelor din figura 3, blocarea se face prin aplicarea unui unu logic în bază iar saturarea prin aplicarea unui zero logic în bază.
- Deoarece comanda în bază pentru tranzistoare se face de către un DCD cu ieșirile active pe zero logic, rezultă că, în fiecare moment de timp, vom avea doar un singur tranzistor saturat iar restul vor fi blocate. Aceasta înseamnă că numai un singur digit din afișaj este conectat la polul pozitiv al tensiunii de alimentare.

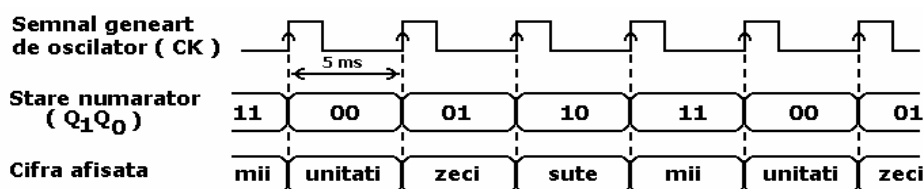


Fig. 4. Succesiunea de afișare a cifrelor zecimale pentru afișajul din figura 3.

- **Continuăm explicarea funcționării schemei din figura 3 considerând că starea număratorului este  $Q_1Q_0 = 01$**
- Decodificatorul binar primește pe intrările de selecție codul BA=01, situație în care starea ieșirilor devine:  $\overline{0} = 1, \overline{1} = 0, \overline{2} = 1, \overline{3} = 1$ . Ținând cont de explicațiile anterioare, această stare a ieșirilor DCD, conduce la saturarea tranzistorului Q3 și la blocarea celorlalte. Așadar, la polul pozitiv al sursei de alimentare este conectat doar digitul de zeci, deci numai el se poate aprinde, supunem că digitul de zeci este activat.
- Blocul de multiplexoare primește pe intrările de date patru coduri BCD (pentru mii, sute, zeci și unități) și scoate pe ieșirile un singur cod BCD, cel care corespunde digitului activat.
- Trebuie să existe o corespondență între digitul activat și codul BCD selectat de blocul de multiplexoare: este necesar ca digitul de unități să primească codul BCD al unităților, digitul de zeci să primească codul BCD al zecilor și așa mai departe. Acest lucru se obține folosind pentru comanda intrărilor de selecție tot starea  $Q_1Q_0$  a număratorului.
- În exemplul considerat, deoarece am presupus  $Q_1Q_0 = 01$ , rezultă că  $S_1S_0 = 01$ , deci codul BCD conectat la intrările  $I_2$  ale multiplexoarelor vor avea cale liberă să ajungă la decodificatorul BCD/7segmente.
- După decodificare, informația despre zeci este aplicată în același timp celor patru afișaje. La prima vedere s-ar părea că informația de zeci va fi afișată pe toate cele patru afișaje. Acest lucru nu se întâmplă deoarece, așa după cum am arătat anterior, numai digitul de zeci are anodul comun conectat la +Vcc, deci numai acesta se poate aprinde.
- Digitul de zeci este menținut aprins 5ms, atâta timp cât durează o stare a număratorului. La următoarea tranziție pozitivă a semnalului de ceas, starea număratorului se schimbă și devine  $Q_1Q_0 = 10$ , ceea ce înseamnă că se activează digitul de sute. Așa cum se arată și în figura 4, urmează activarea miilor, a unităților, și procesul se reia.
- Modul de conectare a blocului de multiplexoare este prezentat în figura 5. Se observă că toate multiplexoarele primesc aceeași informație de selecție. Fiecare multiplexor extrage bitul cu aceeași pondere din codul de mii, de sute, de zeci respectiv de unități.
- Afișarea dinamică, denumită uneori și **afișare multiplexată**, prezintă avantajul unui consum energetic mai mic și necesită un singur decodificator BCD-7segmente.
- Rezistențele R10 ÷ R13 au rol de protecție a joncțiunii BE a tranzistoarelor, ele trebuie calculate astfel încât să permită intrarea în saturație a tranzistoarelor.
- Rezistențele R1 ÷ R7 au rol de limitare a curentului prin LED-urile afișajului.

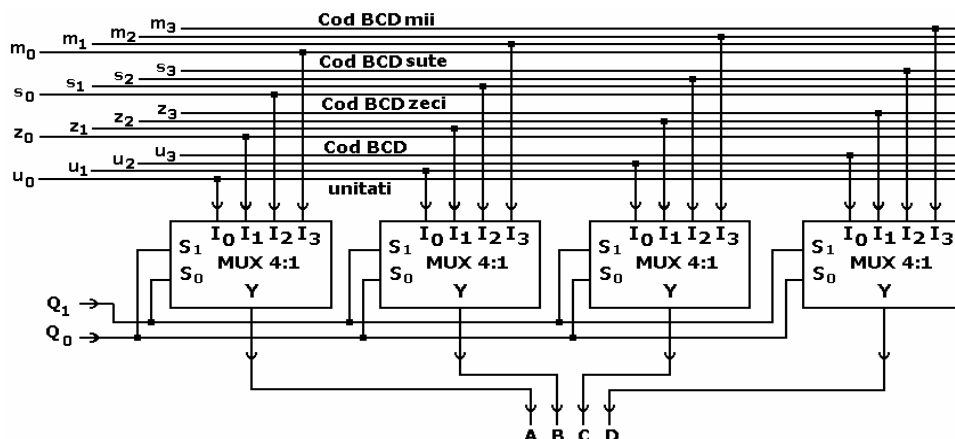


Fig. 5. Schema de conexiuni a blocului de multiplexoare

În final, dacă facem o comparație între numărul de componente necesare afișării statice (figura 3) și numărul de componente necesare afișării dinamice (figurile 4 și 5), se observă un necesar mai mare de componente pentru afișarea dinamică. Din simpla contorizare a numărului de componente se poate trage concluzia greșită că afișarea dinamică este neeconomică. În realitate, complexitatea a 4 decodificatoare BCD-7segmente este mult mai mare decât complexitatea restului de componente ce intervin în afișarea dinamică. Așadar, necesarul de arie dintr-un circuit integrat este mai mic pentru afișarea dinamică decât pentru afișarea statică, în plus mai apare și avantajul unui consum energetic mai redus atunci când discutăm de afișarea dinamică.

#### 2.4. Descrierea unui sistem de afișare cu ajutorul VHDL – conceptul de descriere ierarhică

Din experiența proiectanților de sisteme logice complexe rezultă că proiectarea acestora se face mai ușor folosind conceptul de proiectare ierarhică. Într-o astfel de abordare, se concepe mai întâi o schema bloc a sistemului iar mai apoi se trece la proiectarea în detaliu a fiecărui bloc în parte.

În cazul utilizării limbajelor de descriere hardware, cum este și cazul VHDL, proiectarea ierarhică presupune utilizarea unui proiect în care sunt incluse mai multe fișiere :

- Un fișier de nivel înalt (*top level*), care specifică modul de interconectare a blocurilor componente din structura sistemului. Pentru această scop, se poate folosi un fișier de tip VHDL sau o descriere grafică a schemei bloc, prin intermediul editorului de scheme.
- Mai multe componente de nivel redus (low level). Fiecare bloc funcțional din componența schemei bloc este descris fie de un fișier VHDL, fie de o schemă logică.
- Cel puțin un fișier de constrângeri prin care sunt specificate date referitoare la implementarea

Pentru a înțelege mai bine modul de lucru vom considera ca exemplu cazul sistemului de afișare a informației numerice din figura 3. Pentru a putea fi implementată în CPLD, schema bloc trebuie puțin modificată pentru a ține cont de particularitățile hardware ale machetei de laborator. Printre aceste particularități amintim:

- oscilatorul de pe macheta de laborator are frecvența de 25,175MHz  $\Rightarrow$  necesitatea introducerii unui divizor de frecvență care să primească la intrare un semnal digital cu frecvența de 25,175MHz și să furnizeze la ieșire un semnal cu frecvența de 200Hz, sau puțin mai mare;
- comanda activării afișajelor se face prin unu logic  $\Rightarrow$  ieșirile decodificatorului binar DCD trebuie să fie active pe unu logic;
- comanda segmentelor se face prin zero logic  $\Rightarrow$  ieșirile decodificatorului BCD/7 segmente trebuie să fie active pe zero logic ;
- macheta dispune numai de 8 switch-uri  $\Rightarrow$  putem asigura informație BCD doar pentru doi digiți.

Dacă dorim ca informația afișată să fie introdusă manual avem nevoie de 16 switch-uri însă macheta nu are decât 8. Pentru a rezolva această problemă admitem ca fiecare pachet de 4 switch-uri să comande doi digiți: SW1÷SW4 vor furniza informație pentru digitul de mii și cel de sute iar SW5÷SW8 vor furniza informație pentru digitul de zeci și cel de unități.

♦ **Fișierul de nivel înalt**, să-l denumim **afisaj\_4dig.vhd** este folosit pentru descrierea schemei bloc a sistemului de afișare. Din exemplul de descriere de mai jos se observă că fiecare bloc funcțional este introdus prin intermediul unei declarații de tip **component**.

Observații	Codul VHDL pentru schema bloc
Secțiune dedicată includerii de librării	<pre>library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_arith.all; use IEEE.std_logic_unsigned.all;</pre>
Secțiune dedicată descrierii entității. În cazul de față:	<pre>entity afisaj_4dig is port ( pin_clk : in std_logic; pin_act_dig:out std_logic_vector(3 downto 0); pin_act_seg:out std_logic_vector(6 downto 0); pin_u, pin_s : in std_logic_vector(3 downto 0)); end afisaj_4dig;</pre>

Secțiune dedicată descrierii arhitecturii.

- Fiecare bloc funcțional este introdus printr-o declarație de tip **component**;
- Pentru fiecare componentă trebuie introdus un fișier separat în care se descrie funcția logică realizată (se poate face apel la orice metodă de descriere admisă);
- Denumirea intrărilor și ieșirilor trebuie să fie aceeași în declararea componentei și în descrierea funcționării acesteia;
- Legăturile dintre componente (blocuri funcționale) se face prin intermediul **semnalelor**;
- Pentru intrările sau ieșirile blocurilor funcționale (componentelor) ce sunt conectate direct la intrările sau ieșirile arhitecturii de nivel înalt, definirea de semnale nu mai este necesară;
- Deși nu este recomandată, mai ales pentru începători, este permisă și declararea de **componente** ce conțin două sau mai multe blocuri funcționale;
- Declararea componentelor și a semnalelor trebuie făcută înainte de **begin** –ul arhitecturii;
- Pentru fiecare instanță a unei componente se declară modul de conectare a intrărilor și a ieșirilor prin **port map**;

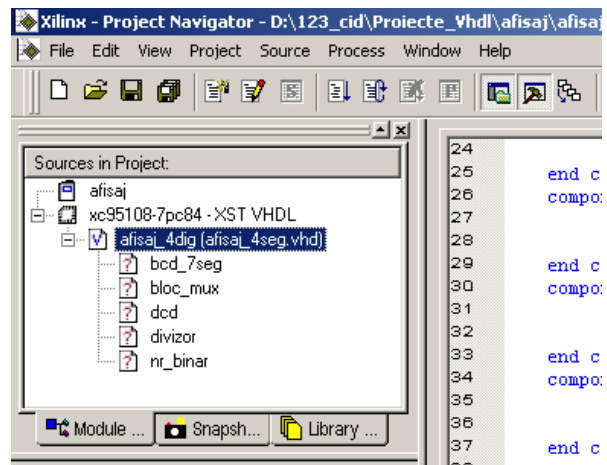
```
architecture arh_afisaj_4dig of afisaj_4dig is
  component divizor
    port ( f_in : in std_logic;
          f_out : out std_logic );
  end component;
  component nr_binar
    port ( nr_clk : in std_logic;
          nr_out : out std_logic_vector(1 downto 0));
  end component;
  component dcd
    port ( sel_dcd : in std_logic_vector(1 downto 0);
          dcd_out : out std_logic_vector(3 downto 0));
  end component;
  component bcd_7seg
    port ( bcd_in : in std_logic_vector(3 downto 0);
          out_7seg_out : out std_logic_vector(6 downto 0));
  end component;
  component bloc_mux
    port ( u, z, s, m : in std_logic_vector(3 downto 0);
          sel_mux : in std_logic_vector(1 downto 0);
          mux_out : out std_logic_vector(3 downto 0));
  end component;
  signal s_div, s_bcd : std_logic;
  signal s_num : std_logic_vector(1 downto 0);
  signal s_bcd : std_logic_vector(3 downto 0);
begin
  bloc1: divizor
    port map ( f_in => pin_clk, f_out => s_div);
  bloc2: nr_binar
    port map ( nr_clk => s_div, nr_out => s_num);
  bloc3: dcd
    port map ( sel_dcd => s_num, dcd_out => pin_act_dig);
  bloc4: bloc_mux
    port map ( sel_mux => s_num, mux_out => s_bcd,
              u => pin_u, z => pin_u, s => pin_s, m => pin_s);
  bloc5: bcd_7seg
    port map ( bcd_in => s_bcd, out_7seg_out
              => pin_act_seg);
end arh_afisaj_4dig;
```

După introducerea și salvarea acestui fișier, fereastra mediului WebPack ce indică sursele implicate în proiect arată ca în figura alăturată.

Se observă că mediul de dezvoltare a aplicației a sesizat prezența a 5 componente și acum așteaptă introducerea unei modalități acceptate pentru descrierea funcționării fiecărei componente.

Pentru a introduce codul VHDL pentru blocul de multiplexoare, este necesar să facem dublu clic pe **bloc\_mux**, iar din fereastra apărută să alegem opțiunea **VHDL Module**. În continuare mediul WebPack ne cere să introducem denumirea și tipul intrărilor și ieșirilor din modul. După aceasta, se generează un template în care noi nu mai trebuie decât să descriem funcționarea respectivului circuit sau bloc logic.

♦ **Fișierul bloc\_mux.vhd**, este folosit pentru descrierea funcționării blocului de multiplexoare din figura 3. O descriere posibilă este prezentată în tabelul de mai jos.



Observații	Codul VHDL pentru blocul de multiplexoare
Secțiune dedicată includerii de librării	<pre>library IEEE; use IEEE.std_logic_1164.all; use IEEE.std_logic_arith.all; use IEEE.std_logic_unsigned.all;</pre>
Secțiune dedicată descrierii entității.	<pre>entity bloc_mux is   port (u, z, s, m : in std_logic_vector(3 downto 0);         sel_mux : in std_logic_vector(1 downto 0);         mux_out : out std_logic_vector(3 downto 0)); end bloc_mux;</pre>
Secțiune dedicată descrierii arhitecturii.	<pre>architecture arh_mux of bloc_mux is begin   with sel_mux select</pre>
- Se utilizează o descriere comportamentală a blocului de	

multiplexoare; - Pentru combinația de selecție 00 se alege codul unităților, pentru combinația 01 codul yecilor, etc.;	<pre> mux_out&lt;= u when "00",               z when "01",               s when "10",               m when "11",               u when others;  end arh_mux; </pre>
---	--

♦ **Fișierele divizor.vhd , nr\_binar.vhd și dcd.vhd** se introduc într-o manieră similară cu **bloc\_mux.vhd** .

♦ **Fișierul de constrângeri** are extensia **.ucf** și este în principal folosit pentru a specifica pinii circuitului CPLD la care sunt conectate intrările, respectiv ieșirile afișajului.

Pentru a introduce fișierul de constrângeri se alege: **Project → New Source → Implementation Constraints File** → se alege un nume pentru fișier, spre exemplu *abc* → se face asocierea cu fișierul principal **afisaj\_4dig** → se alege **Finish**. În urma acestor operații în fereastra surselor implicate în proiect apare o nouă componentă **abc.ucf**.

Dacă se face dublu clic pe **abc.ucf** se lansează un utilitar care ne ajută să edităm fișierul de constrângeri. Din fereastra apărută se selectează tabul **Ports**. După această alegere, pe ecran apare lista porturilor de intrare și de ieșire ale sistemului digital proiectat de noi. Pentru fiecare port trebuie să specificăm pinul CPLD-ului unde dorim conectarea respectivului port, aceasta presupune completarea coloanei denumită **Location**. După completare se face o salvare a fișierului de constrângeri.

**Trebuie avut grijă ca numărul pinului completat în coloana Location să corespundă cu schema hardware a machetei de laborator. Pentru acesta este absolut obligatoriu să consultați tabelul de conexiuni prezentat în anexe.**

Ținând cont de schema electrică a machetei de laborator, pentru afișajul pe 4 digiți descris anterior, conținutul fișierului de constrângeri este prezentat pe coloana alăturată.

```

NET "pin_act_dig<0>" LOC = "P70";
NET "pin_act_dig<1>" LOC = "P68";
NET "pin_act_dig<2>" LOC = "P66";
NET "pin_act_dig<3>" LOC = "P63";
NET "pin_act_seg<0>" LOC = "P39";
NET "pin_act_seg<1>" LOC = "P41";
NET "pin_act_seg<2>" LOC = "P44";
NET "pin_act_seg<3>" LOC = "P46";
NET "pin_act_seg<4>" LOC = "P48";
NET "pin_act_seg<5>" LOC = "P51";
NET "pin_act_seg<6>" LOC = "P53";
NET "pin_s<3>" LOC = "P37";
NET "pin_s<2>" LOC = "P40";
NET "pin_s<1>" LOC = "P43";
NET "pin_s<0>" LOC = "P45";
NET "pin_u<3>" LOC = "P47";
NET "pin_u<2>" LOC = "P50";
NET "pin_u<1>" LOC = "P52";
NET "pin_u<0>" LOC = "P54";

```

♦ **Implementarea circuitului în CPLD.** După introducerea surselor în proiect (etapă denumită *Design Entry*), urmează etapa de sinteză (*Synthesis*) - în care se generează o schemă logică pe baza descrierilor din etapa anterioară, apoi o etapă de implementare (*Implement Design*) - în care circuitul logic deja sintetizat este amplasat în CPLD ținând cont de resursele acestuia și de cerințele utilizatorului. Ultima etapă constă în generarea fișierelor de configurare a circuitului CPLD, etapă denumită incorect de generare a fișierelor de programare (*Generate Programming Files*).

Cu excepția etapei de *Design Entry*, restul etapelor se fac în mod automat de către mediul WebPack (nu necesită nici o intervenție din partea utilizatorului).

Pentru generarea fișierului de configurare, în fereastra resurselor se alege fișierul principal (**afisaj\_4dig**) iar în fereastra proceselor disponibile pentru acest fișier se face dublu clic pe opțiunea **Configure Device (iMPACT)**. În ferestre ce urmează se aleg următoarele opțiuni: **Configure Devices → Next → Boundary Scan Mode → Automatically connect to cable ... → Finish → OK** → se alege fișierul **afisaj\_4dig.jed** → **Open** → clic dreapta pe icoana circuitului XC95108 → **Program...** → se alege opțiunea **Erase Before Programming → OK**. Dacă totul este în regulă, după câteva secunde va apare mesajul **Programming Succeeded** și se poate trece la verificarea funcționării circuitului.



### 3. Desfășurarea lucrării

#### 3.1. Afișarea statică.

A. Referitor la schema din figura 2, răspundeți la următoarele întrebări:

- Care este căderea de tensiune pe un LED aflat în conducție ?
- O reducere a necesarului de rezistențe de limitare se poate obține eliminând rezistențele de limitare a curentului dintre decodificatoare și afișaje și montarea altora între Vcc și anodul comun. Este posibilă o astfel de abordare? Care ar fi inconvenientele ?

#### 3.2. Afișarea dinamică.

A. Referitor la schema din figura 3, răspundeți la următoarele întrebări:

- Ce se întâmplă dacă frecvența oscilatorului este de 400Hz ? Dar dacă este de 150Hz ?
- Ce modificări trebuie făcute în schemă pentru a obține un afișaj cu 6 digiți ?
- Dacă avem la dispoziție un decodificator BCD-7segmente cu ieșiri active pe unu logic și afișaje cu anodul comun, ce modificări trebuie efectuate în schemă pentru a deveni funcțională ?
- Este strict necesară "aprinderea" cifrelor în ordinea: unități, zeci, sute, mii ?
- Are importanță starea inițială de la care pleacă numărătorul ?
- Care este schema pentru un afișaj cu 4 digiți cu catod comun ?

- După verificarea unui montaj electronic s-a constatat că există o eroare de execuție: ieșirea  $\bar{2}$  comandă tranzistorul Q3 iar ieșirea  $\bar{1}$  comandă tranzistorul Q2. Cum trebuie conectate codurile BCD de la intrarea blocului de multiplexoare pentru ca informația de intrare să fie afișată corect ?
- Care este schema logică realizată cu porți NAND ce poate înlocui decodificatorul binar ce comandă baza tranzistoarelor ?
- Care este intervalul maxim de timp (în cazul cel mai defavorabil) dintre modificarea informației de intrare și afișarea efectivă a modificării ?
- Care este efectul vizibil dacă se întrerupe legătura dintre ieșirea Q<sub>B</sub> a numărătorului și intrarea de selecție B a decodificatorului binar ?

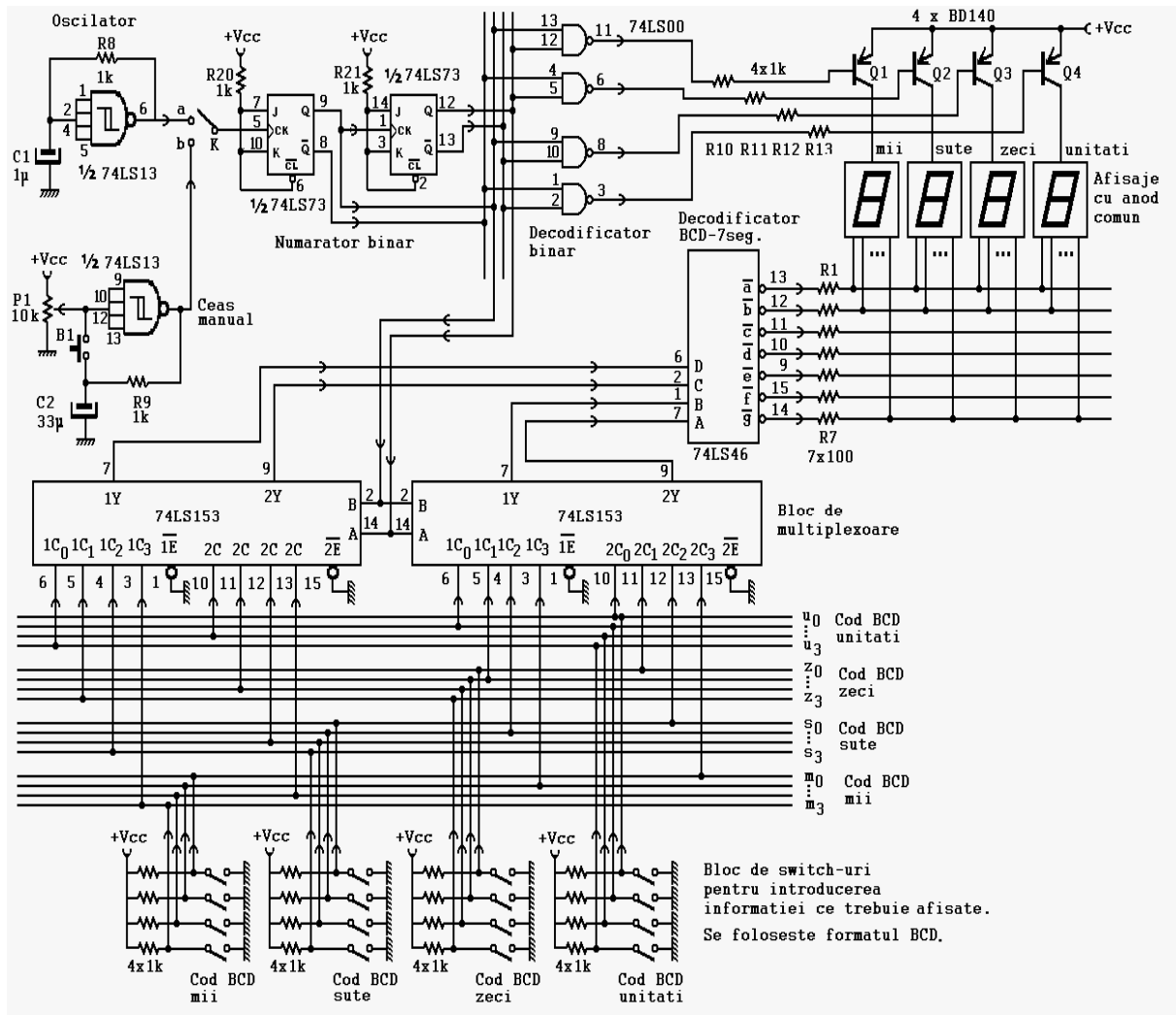


Fig.6. Schema electrică a machetei de laborator reprezentând un afișaj multiplexat cu 4 digiți

**B.** Referitor la schema electrică a machetei de laborator prezentată în figura 6, răspundeți la următoarele întrebări:

- Care este corespondența dintre schema bloc din figura 3 și schema detaliată din figura 6 ?
- Cum se poate modifica frecvența oscilatorului ?
- Poarta NAND cu caracteristică de transfer de tip trigger Schmitt poate fi înlocuită cu o poartă NAND cu caracteristică normală ? Dar cu un inversor cu caracteristică trigger Schmitt ?
- Explicați funcționarea ceasului manual (P1, B1, C2, R9, poarta NAND cu caracteristică de tip trigger Schmitt) ?
- Cum a fost implementat numărătorul pe 2 biți ? Ce rol au rezistențele R20 și R21 ?
- Se poate înlocui numărătorul din schema 6 cu un circuit specializat (spre exemplu 7493) fără a modifica restul schemei ?
- Care este modul de realizare a decodificatorului binar necesar pentru comanda tranzistoarelor ?
- Se poate înlocui decodificatorul binar cu un registru de deplasare pe 4 biți ?
- Consultați foile de catalog și precizați care sunt particularitățile circuitului 74LS153 ?

**C.** Determinări experimentale pe macheta de laborator cu componente discrete:

- Plasați comutatorul **K** pe poziția **a** și determinați cu osciloscopul frecvența semnalului de ceas (folosit pentru trecerea de la o cifră la alta);
- Plasați comutatorul **K** pe poziția **b** și acționați asupra butonului **B1**, urmăriți secvența de aprinderea a cifrelor de pe afișaj;



- În ambele situații acționați asupra switch-urilor de intrare și urmăriți efectul pe afișaj.

### 3.3. Implementarea unui sistem de afișaj folosind limbajul VHDL.

- A.** Folosind modul de lucru descris în secțiunea 2.4. a lucrării, se cere:
- Codul VHDL pentru restul blocurilor funcționale (divizor, numărător binar, decodificator binar, decodificator BCD-7segmente;
  - Implementarea proiectului pe macheta de laborator cu CPLD;
  - Verificarea funcționării proiectului;
- B.** Modificați proiectul anterior (prin adăugare de noi blocuri funcționale) astfel încât pe afișaj să apară, unul după altul în ordine crescătoare, numerele zecimale de la 0 la 9999. Numărătoarea va fi ciclică, după numărul 9999 se reîncepe de la zero.
- Indicații:*
- *Este nevoie de introducerea unui numărător zecimal cu patru decade;*
  - *Intrarea de ceas a numărătorului zecimal trebuie comandată de un semnal cu frecvență mică pentru ca observatorul să aibă timpul necesar să observe (pe afișaj) succesiunea stărilor. Pentru aceasta, o sugestie ar fi să micșorăm frecvența semnalului de 200Hz, printr-un bloc suplimentar de divizare a frecvenței.*
  - *Ieșirile numărătorului se vor conecta la intrările de date ale blocului de multiplexoare. Semnalul de*
- C.** Modificați proiectul de la subpunctul **B** astfel încât numărătoarea să se facă invers (de la 9999 spre 0).
- D.** Modificați proiectul de la subpunctul **B** astfel încât numărătoarea să se facă astfel: în sens crescător de la 133 la 777, după care în sens descrescător de la 777 la 133.



## Lucrarea nr. 11: **Studiul și implementarea automatelor FSM** (FSM - Finite State Machine)

### 1. Scopul lucrării

Prezentarea modalităților de analiză și sinteză logică a automatelor de tip FSM și implementarea acestora în structuri de tip CPLD sau FPGA. Totodată se continuă seria de prezentarea prin exemple a limbajului de descriere hardware VHDL.

### 2. Considerente teoretice

Automatele elementare sunt circuite logice secvențiale alcătuite dintr-un registru de memorie și un circuit logic combinațional.

**Registrul de memorie**, denumit și registru de stare, este format dintr-un număr de bistabili, de regulă de același tip, ale căror intrări de ceas sunt conectate împreună formând astfel intrarea de ceas a automatului (intrarea CK). *Starea unui automat este dată de valorile logice ale bistabililor din registrul de stare și, în consecință, este susceptibilă de modificare după fiecare tranziție activă a semnalului aplicat intrărilor de ceas.*

**Circuitul logic combinațional** are dublu rol: calculează ieșirile automatului și starea următoare a acestuia. *Starea următoare* a unui automat este calculată în funcție de starea prezentă (indicată de registrul de stare) și în funcție de starea semnalelor de intrare. Starea următoare înlocuiește stare prezentă imediat după ce apare prima tranziție activă a semnalului de ceas. În calculul semnalelor de ieșire, pe lângă starea prezentă a automatului, în funcție de tipul automatului, mai poate interveni starea semnalelor de intrare.

#### 2. 1. Clasificarea automatelor sincrone

##### a) După modul de calcul al ieșirii:

- **Automate de tip Medvedev** – ieșirea automatului este identică cu starea automatului:

$$\text{ieșire} = \text{stare prezentă}$$

$$\text{stare viitoare} = G(\text{intrare}, \text{stare prezentă})$$

- **Automate de tip Moore** - ieșirea este dependentă numai de starea prezentă a automatului:

$$\text{ieșire} = F(\text{stare prezentă})$$

$$\text{stare viitoare} = G(\text{intrare}, \text{stare prezentă})$$

- **Automate de tip Mealy** - ieșirea este dependentă de intrare și de starea prezentă a automatului:

$$\text{ieșire} = F(\text{intrare}, \text{stare prezentă})$$

$$\text{stare viitoare} = G(\text{intrare}, \text{stare prezentă})$$

##### b) După modul de utilizare a ieșirii calculate:

- Automate cu ieșire imediată;
- Automate cu ieșire întârziată;

Schemele bloc de principiu ale automatelor de tip Mealy și Moore sunt prezentate în fig. 1.

##### Observații:

- Automatele cu utilizarea imediată a ieșirilor prezintă dezavantajul că fenomenele tranzitorii (de scurtă durată) ce apar imediat după tranziția activă a semnalului de ceas, sunt transmise direct la ieșirile automatului.
- Automatele cu întârziere prezintă în plus față de cele imediate un registru de memorie în care se stochează ieșirile calculate.
- Pentru automatele Mealy imediate, modificarea intrărilor are efect imediat asupra ieșirilor ceea ce înseamnă că modificarea ieșirilor are loc în același tact cu modificarea intrărilor.
- Pentru automatele Moore imediate, modificarea ieșirilor se face cu o întârziere de un tact în raport cu modificarea intrărilor. Intrările ce se modifică în tactul  $t$ , afectează starea curentă a automatului din tactul  $t+1$ , care mai departe determină schimbarea ieșirilor tot în tactul  $t+1$ . Deci, la ieșire, efectul intrărilor este resimțit cu o întârziere de o perioadă a semnalului de ceas, cu toate că automatul este referit ca un automat imediat.
- La automatele cu întârziere, ieșirile calculate sunt înscrise într-un registru de memorie suplimentar, altul decât cel de stare. Înscriserea se face la tactul următor al ceasului atunci când se presupune că regimul tranzitoriu este stins. În acest mod, față de cazul automatelor imediate, la cele cu întârziere, se mai adaugă o întârziere suplimentară de un tact. Astfel că răspunsul unui automat Mealy cu întârziere apare după un tact, iar pentru un automat Moore cu întârziere după două.

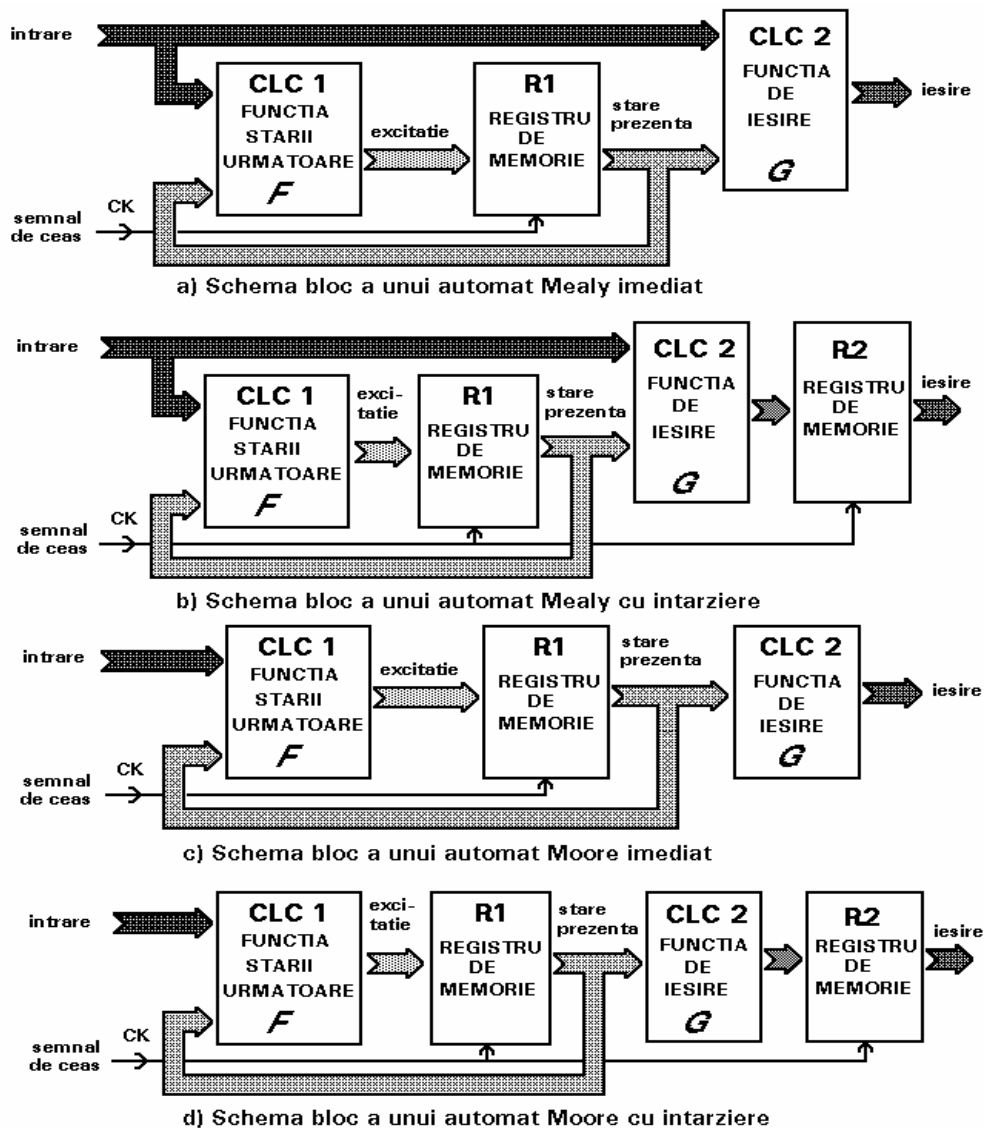


Fig. 1. Scheme bloc de principiu pentru automate de tip Mealy, respectiv Moore

## 2. 2. Modalități de reprezentare a automatelor

### A.) Graful de tranziție a stărilor

Este o metodă intuitivă ce permite descrierea și verificarea rapidă a funcționării unui automat. Pentru fiecare nod al grafului, reprezentat printr-un cerculeț, se asociază o stare a automatului, iar fiecare arc orientat între două noduri corespunde tranziției între două stări.

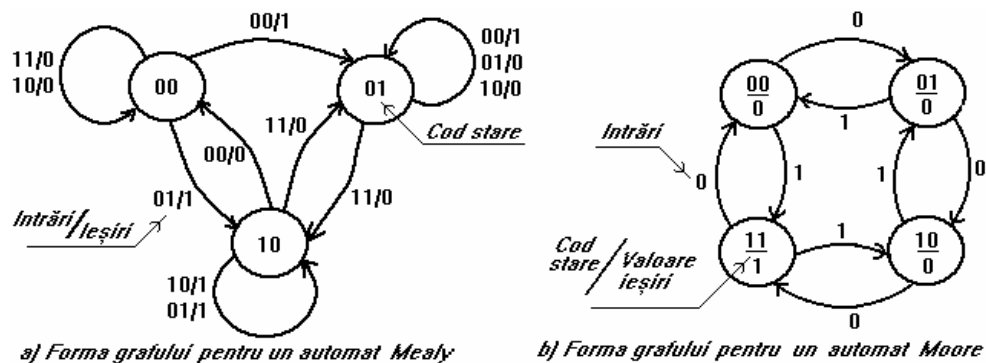


Fig. 2. Exemple de garfuri de tranziție pentru cele două tipuri de automate

**Observații:**

- În cazul automatelor de tip Mealy, în cerculeț se trece codul stării, iar pe arc se trec variabilele de intrare ce provoacă tranziția respectivă precum și ieșirea determinată de acestea.
- La automatele Moore, ieșirea nu depinde de stare motiv pentru care în cerculeț se trece codul stării și valoarea ieșirii, iar pe arc se trece combinația intrării ce generează tranziția.
- Pentru exemplificare, în fig. 2 se prezintă câte un graf pentru fiecare caz.

**B.) Tabelul de tranziție a stărilor**

Informația conținută în diagrama de tranziție poate fi transpusă într-o formă mai utilă pentru sinteza automatului, această formă poartă denumirea de tabel de tranziție. Transpunerea celor două grafuri de tranziție a stărilor din figura 2, în tabele de tranziție se arată în tabelele ce urmează.

*Tabelul de tranziție a stărilor corespunzător grafului din fig. 2. b)*

Stare prezentă $Q_1 \ Q_0$	Intrări $X_1 \ X_0$	Stare următoare $Q_1 \ Q_0$	Ieșiri $Y$
0 0	0 0	0 1	1
0 0	0 1	1 0	1
0 0	1 0	0 0	0
0 0	1 1	0 0	0
0 1	0 0	0 1	1
0 1	0 1	0 1	0
0 1	1 0	0 1	0
0 1	1 1	1 0	0
1 0	0 0	0 0	0
1 0	0 1	1 0	1
1 0	1 0	1 0	1
1 0	1 1	0 1	0
1 1	x x	x x	x

*Tabelul de tranziție a stărilor corespunzător grafului din fig. 2. b)*

Stare prezentă $Q_1 \ Q_0$	Intrare $X$	Stare viitoare $Q_1 \ Q_0$	Ieșire $Y$
0 0	0	0 1	0
0 0	1	1 1	0
0 1	0	1 0	0
0 1	1	0 0	0
1 0	0	1 1	0
1 0	1	0 1	0
1 1	0	0 0	1
1 1	1	1 0	1

**2.3. Analiza automatelor elementare**

Studiul funcționării unui automat elementar dat se numește **analiză**. Analiza unui automat are drept scop determinarea funcțiilor F și G astfel încât funcționarea automatului să poată fi prezisă pe baza schemei electrice a acestuia. Etapele parcurse în analiza unui automat sunt reliefate în următoarele două exemple:

**Exemplul:** Să se analizeze funcționarea automatului sincron din fig. 3:

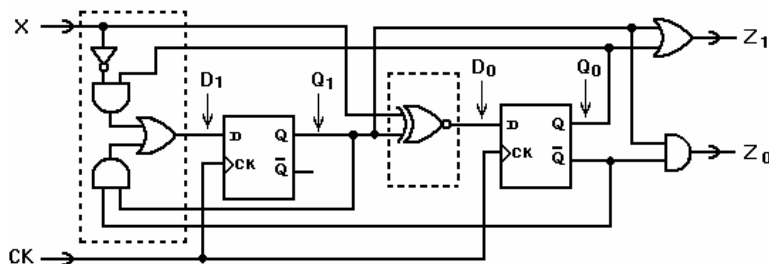


Fig. 3. Exemplu de automat Moore

Din analiza schemei electrice se observă că este vorba despre un automat Moore imediat caracterizat de: o intrare notată X; două ieșiri notate Z1, Z0 și un registru de stare format din doi bistabili D, ceea ce înseamnă că automatul are cel mult 4 stări distincte.

Etapă	Observații
<b>Pasul 1: Determinarea semnalelor de excitație a bistabililor</b> Prin semnale de excitație sunt referite semnalele aplicate intrărilor pasive ale bistabililor (D pentru bistabili D, JK pentru bistabili JK, etc.). Un semnal de excitație este sintetizat cu ajutorul unui CLC ce are drept intrări starea curentă și intrările automatului. Odată delimitat acest CLC se trece la descrierea sa matematică.	Pentru automatul din fig. 3, semnale de excitație sunt calculate de cele două circuite combinatoriale delimitate cu linie punctată și au următoarele expresii logice: $D_1 = \left  \begin{array}{c} \bar{x} \ Q_0 \\ Q_1 \ Q_0 \end{array} \right , \quad D_0 = \left  \begin{array}{c} \bar{x} \ Q_1 \\ x \ Q_1 \end{array} \right $
<b>Pasul 2: Scrierea ecuației funcționale a bistabilului utilizat în registrul de stare al automatului.</b>	Ecuația funcțională a unui bistabil D este dată de relația: $Q^+ = D$

	<p>unde prin <math>Q^+</math> s-a notat starea viitoare a bistabilului.</p> <p>În cazul bistabililor JK, ecuația funcțională este :</p> $Q^+ = \left  \begin{array}{c} J \quad \overline{Q} \\ \hline \overline{K} \quad Q \end{array} \right $																																																																						
<p><b>Pasul 3: Determinarea ecuațiilor de tranziție a stărilor</b></p> <p>Ecuațiile de tranziție a stărilor se obțin prin înlocuirea semnalelor de excitație, determinate la pasul 1, în ecuația funcțională a bistabilului scrisă în pasul 2.</p>	<p>Pentru exemplul considerat obținem:</p> $Q_1^+ = D_1 = \left  \begin{array}{c} \overline{x} \quad Q_0 \\ \hline Q_1 \quad Q_0 \end{array} \right  \quad Q_0^+ = D_0 = \left  \begin{array}{c} \overline{x} \quad Q_1 \\ \hline x \quad Q_1 \end{array} \right $																																																																						
<p><b>Pasul 4: Alcătuirea tabelului de tranziție</b></p> <p>Pentru fiecare stare prezentă a automatului, se determină starea următoare în care v-a ajunge automatul luând în calcul totalitatea combinațiilor variabilelor de intrare.</p> <p>Pentru automatul luat ca exemplu, există o singură variabilă de intrare ceea ce înseamnă că tabelul de tranziție va conține 4x2 linii.</p>	<table><tr><th colspan="2">Stare prezentă</th><th>Intrare</th><th colspan="2">Stare viitoare</th><th colspan="2">Ieșiri</th></tr><tr><th><math>Q_1</math></th><th><math>Q_0</math></th><th><math>X</math></th><th><math>Q_1^+</math></th><th><math>Q_0^+</math></th><th><math>Z_1</math></th><th><math>Z_0</math></th></tr><tr><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td></tr><tr><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr></table>	Stare prezentă		Intrare	Stare viitoare		Ieșiri		$Q_1$	$Q_0$	$X$	$Q_1^+$	$Q_0^+$	$Z_1$	$Z_0$	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	1	0	1	0	1	1	0	0	0	1	1	0	0	1	0	1	1	1	0	1	1	1	1	1	1	1	0	1	0	0	1	1	1	1	0	1	0	1
Stare prezentă		Intrare	Stare viitoare		Ieșiri																																																																		
$Q_1$	$Q_0$	$X$	$Q_1^+$	$Q_0^+$	$Z_1$	$Z_0$																																																																	
0	0	0	0	1	0	0																																																																	
0	0	1	0	0	0	0																																																																	
0	1	0	1	1	0	1																																																																	
0	1	1	0	0	0	1																																																																	
1	0	0	1	0	1	1																																																																	
1	0	1	1	1	1	1																																																																	
1	1	0	1	0	0	1																																																																	
1	1	1	0	1	0	1																																																																	
<p><b>Pasul 5: Trasarea grafului de tranziție</b></p> <p>Graful de tranziție extras din tabelul anterior ne arată că automatul din fig. 3 este un numărător binar în cod Gray care, numără înainte pentru <math>x=0</math> și invers pentru <math>x=1</math>. În plus, odată ajuns la cap de scală procesul de numărare se oprește.</p> <p>Ieșirea <math>z_1</math> semnalizează prin "unu logic" atingerea stării minime, iar <math>z_0</math> atingerea stării maxime a numărătorului în cod Gray pe 2 biți.</p>																																																																							

## 2. 4. Sinteza automatelor elementare sincrone

Proiectarea unui automat elementar, pornind de la o descriere a funcționării acestuia în cuvinte (cazul cel mai uzual și totodată cel mai dificil), sau de la o diagramă de tranziție se numește sinteză. Procesul de sinteză necesită etapelor prezentate mai jos.

### • Descrierea formală a funcționării automatului

Această etapă este cea mai importantă deoarece trebuie să facă trecerea de la descrierea vagă, uneori ambiguă, a limbajului natural spre o specificație clară a funcționării automatului.

#### Observații:

- de regulă, sunt cunoscute numărul variabilelor de intrare și al celor de ieșire;
- stărilor automatului le sunt asociate mnemonici cât mai sugestive;
- pe parcursul descrierii pot apare probleme ce nu au fost cuprinse în formularea inițială;
- este posibil ca descrierea corectă să nu rezulte din prima iterație și atunci procesul trebuie reluat;
- rezultatul final al acestei etape este o diagramă de stare sau un tabel de stare ieșire.

### • Reducerea numărului de stări ale automatului

Din etapa anterioară este posibil ca descrierea formală să conțină un număr mai mare de stări decât cel strict necesar. Reducerea numărului de stări se bazează pe identificarea stărilor echivalente. Două stări ale automatului sunt echivalente, și deci pot fi înlocuite cu una singură, dacă ele nu se pot deosebi prin inspectarea stării curente și a celei următoare. Trebuie remarcat că în procesul de comparare nu participă variabilele de stare.

Se poate demonstra că stările A și B, sunt echivalente dacă sunt îndeplinite următoarele condiții:

- A și B trebuie să conducă la aceleași valori ale variabilelor de ieșire ale automatului, pentru automatele Mealy această condiție trebuie îndeplinită pentru toate variabilele de intrare;
- pentru fiecare combinație de intrare în parte, din A și B trebuie să se ajungă în aceeași stare următoare.

#### Observații:

- Foarte adesea, în practică, sunt aplicații în care anumite combinații ale variabilelor de intrare nu apar niciodată, caz în care atât ieșirile cât și stările următoare ale automatului nu au nici o semnificație pentru acesta. În tabelul tranzițiilor aceste cazuri se notează cu *don't care* și sunt tratate ca în cazul simplificărilor de funcții binare.
- Două stări binare, A și B, pot fi echivalente chiar dacă nu conduc spre aceleași stări următoare numai cu condiția realizării de echivalențe între stările următoare, "șintite", de stările inițiale A și B.

- Există metode automate de reducere a stărilor însă sunt mai puțin utilizate de proiectanți, deoarece în baza experienței acumulate, aceștia pot face o descriere minimală a funcționării unui automat.
- Nu întotdeauna, un număr redus de stări înseamnă și o implementare mai simplă, există și situații în care creșterea numărului de stări poate simplifica proiectul.

#### • **Asignarea stărilor automatului**

În această etapă, pentru fiecare stare a automatului, stare până acum referită printr-o mnemonică, i se asociază un **cod binar unic**. Asocierea de coduri binare, pentru fiecare stare a automatului, se poate face în mod arbitrar sau urmărind un anumit criteriu. Deoarece există mai multe posibilități de codare a stărilor, vor exista tot atâtea variante de scheme logice de realizare, echivalente funcțional dar deosebite ca mod de interconectare. Este greu de estimat care schemă logică, dintre cele posibile, este cea mai redusă în complexitate.

Uzual, codarea stărilor se face cel mai adesea arbitrar, sau folosind un criteriu de optimizare cum ar fi: codarea cu variație minimă sau codarea cu dependență redusă față de intrări.

**Codificarea cu variație minimă** constă în asignarea unor coduri astfel încât trecerea între două stări succesive ale automatului să se facă prin modificarea unui număr cât mai redus de biți ai cuvântului de stare. Această codificare prezintă avantajul că generează expresii simple pentru funcțiile ce calculează stările următoare.

**Codificarea cu dependență redusă față de intrări.** Funcțiile de tranziție și cele de ieșire depind atât de starea curentă cât și de intrări. Pe căile de tranziție necondiționate nu se testează nici o variabilă de intrare, în schimb pe cele condiționate se poate ajunge la testarea tuturor variabilelor de intrare. Codificare care conduce la expresii ale funcțiilor de tranziție cu o dependență cât mai redusă față de intrări este denumită **codificare cu dependență redusă**.

#### **Observații:**

- numărul maxim al stărilor distincte ale unui automat cu  $n$  bistabili este egal cu:  $2^n$ .
- pentru stări apropiate din punct de vedere funcțional se recomandă alocarea unor cuvinte de cod care să difere printr-un singur bit (codare cu variație minimă);
- dacă există stări nefolosite se alege cea mai convenabilă codare pentru stările utilizate, nu trebuie să ne limităm strict la codarea binară naturală;
- schimbarea codului unei stări provoacă modificarea întregii scheme logice a automatului;
- acolo unde este posibil, se recomandă separarea din setul variabilelor de stare a biților, sau a grupurilor de biți, cu semnificație precisă referitoare la schimbarea ieșirilor automatului;

#### • **Alcătuirea tabelului de tranziție / ieșire**

După ce s-au efectuat etapele anterioare, se optează pentru un tip de bistabil, după care se trece la alcătuirea tabelului de tranziție/ieșire. În acest tabel se trec și semnalele de excitație ai bistabililor automatului. După rezolvarea acestei etape, problema se reduce la implementarea unor funcții binare, funcțiile de excitație a bistabililor și funcțiile de ieșire ale automatului.

#### • **Deducerea ecuațiilor funcțiilor de excitație și a celor de ieșire**

Din tabelul rezultat la pasul anterior se extrage forma analitică a funcțiilor de excitație și a celor de ieșire, după care, prin diverse procedee se minimizează aceste expresii în vederea implementării automatului cu un hardware minimal.

#### • **Implementarea automatului**

Rezultatul final al acestei etape este o schemă logică care trebuie să realizeze o funcționare conformă cu diagrama automatului. Spunem o schemă logică deoarece pot fi o multitudine de realizări logice ale aceluiași automat.

## **2. 4. Utilizarea limbajului VHDL pentru implementarea automatelor sincrone**

În mod evident, cea mai avantajoasă metodă de descriere a automatelor sincrone este cea comportamentală.

Am arătat anterior că în structura internă a unui automat putem distinge un registru de stare și două circuite logice combinatoriale: unul pentru calculul stării viitoare și altul pentru calculul ieșirii. Referitor la descrierea acestor blocuri funcționale cu ajutorul VHDL sunt valabile următoarele reguli:

- Circuitul logic de calcul al ieșirii poate fi descris fie printr-un proces special destinat acestui scop, fie prin declarații concurente.
- Registrul de stare trebuie să fie descris în mod obligatoriu printr-un proces declanșat de semnalul de ceas și eventual (dacă există) de semnalul de reset.
- Logica de calcul a stării viitoare poate fi descrisă printr-un proces separat de celelalte sau poate fi inclusă în procesul destinat registrului de stare.

Se observă așadar că există posibilitatea de a descrie automatul cu două sau cu trei procese. Din punct de vedere al ușurinței de proiectare, mai ales pentru începători, se recomandă descrierea cu trei procese. Din punct de vedere al performanțelor circuitului rezultat în urma sintezei, în literatura de specialitate se arată că este mai avantajoasă descrierea cu două procese.

Referitor la codificarea stărilor automatului, există două posibilități:

- a) **Codificarea este făcută de compilator** după niște reguli interne (de regulă, metoda implicită de codificare este cea binară iar dacă se dorește optimizarea din punct de vedere al vitezei de răspuns se folosește codificarea de tip "one hot").

Pentru declararea stărilor automatului trebuie introduse declarații similare celor de mai jos:

```
type st_val is (st0, st1, st2);
signal stare_prez, stare_viit: st_val;
```

- b) **Codificarea stărilor poate fi forțată** după dorința noastră prin intermediul unor declarații similare celor de mai jos:

```
subtype stare_Automat is std_logic_vector(1 downto 0);
signal stare_prezentă, stare_viitoare :stare_Automat;
constant st0:stare_Automat := '01';
constant st1:stare_Automat := '11';
constant st2:stare_Automat := '11';
```

În cele ce urmează prezentăm câteva exemple de automate de tip Mealy și Moore descrise prin două sau trei procese cu sau fără impunerea codificării stărilor automatului.

♦ **Exemplul 1:** Descrierea în limbaj VHDL a unui automat de tip Moore (Varianta 1 - descriere folosind trei procese)

Observații	Codul VHDL
Secțiune dedicată includerii de librării	-- Exemplu de automat Moore cu 5 stari <b>library</b> ieee; <b>use</b> ieee.std_logic_1164.all;
Secțiune dedicată descrierii entității. În cazul de față: - nume entitate este: <b>moore</b> ; - intrarea de ceas : <b>clk</b> ; - intrarea de ștergere: <b>reset</b> ; - intrările automatului: <b>data_in</b> ; - ieșirea automatului: <b>data_out</b> ; - afișare stare automat: <b>afis_stare</b> ;	<b>entity</b> moore <b>is</b> <b>port</b> (clk, reset: <b>in</b> std_logic; data_out: <b>out</b> std_logic; afis_stare: <b>out</b> std_logic_vector (2 <b>downto</b> 0); data_in: <b>in</b> std_logic_vector (1 <b>downto</b> 0)); <b>end</b> moore;
Secțiune dedicată descrierii arhitecturii. În cazul de față: - numele arhitecturii este: <b>arh_comport</b> ; - arhitectura este asociată cu entitatea: <b>mealy</b> ; - stările automatului sunt introduse printr-o enumerare, codificarea prpriu-zisă este lăsată pe seama compilatorului; - sunt folosite 3 procese: <b>reg_stare</b> pentru descrierea registrului de stare, <b>st_viit</b> pentru determinare stare viitoare și <b>iesire</b> pentru determinarea ieșirilor automatului; - graful de funcționare al automatului este:	<b>architecture</b> arh_comport <b>of</b> moore <b>is</b> <b>type</b> st_val <b>is</b> (st0, st1, st2, st3, st4); <b>signal</b> stare_prez, stare_viit: st_val; <b>begin</b> -- proces ptr. registrul de stare reg_stare: <b>process</b> (clk, reset) <b>begin</b> <b>if</b> (reset = '1') <b>then</b> stare_prez <= st0; <b>elsif</b> (clk = '1' <b>and</b> clk'event) <b>then</b> stare_prez <= stare_viit; <b>end if</b> ; <b>end process</b> reg_stare; -- proces ptr. calcul stare viitoare st_viit: <b>process</b> (stare_prez, data_in) <b>begin</b> <b>case</b> stare_prez <b>is</b> <b>when</b> st0 => <b>case</b> data_in <b>is</b> <b>when</b> "00" => stare_viit <= st0; <b>when</b> "01" => stare_viit <= st4; <b>when</b> "10" => stare_viit <= st1; <b>when</b> "11" => stare_viit <= st2; <b>when others</b> => null; <b>end case</b> ; <b>when</b> st1 => <b>case</b> data_in <b>is</b> <b>when</b> "00" => stare_viit <= st0; <b>when</b> "10" => stare_viit <= st2; <b>when others</b> => stare_viit <= st1; <b>end case</b> ; <b>when</b> st2 => <b>case</b> data_in <b>is</b> <b>when</b> "00" => stare_viit <= st1; <b>when</b> "01" => stare_viit <= st1; <b>when</b> "10" => stare_viit <= st3; <b>when</b> "11" => stare_viit <= st3; <b>when others</b> => null; <b>end case</b> ; <b>when</b> st3 => <b>case</b> data_in <b>is</b> <b>when</b> "01" => stare_viit <= st4; <b>when</b> "11" => stare_viit <= st4; <b>when others</b> => stare_viit <= st3; <b>end case</b> ; <b>when</b> st4 => <b>case</b> data_in <b>is</b> <b>when</b> "01" => stare_viit <= st4; <b>when</b> "11" => stare_viit <= st4; <b>when others</b> => stare_viit <= st3; <b>end case</b> ; <b>end case</b> ; <b>end process</b> st_viit;

- Observații referitoare la procesul **reg\_stare** :
- este folosit pentru a descrie funcționarea registrului de stare al automatului;
  - lista de senzitivități cuprinde intrarea de ceas și intrarea de reset;
  - intrarea de reset este activă pe unu logic;
  - intrarea de ceas este activă pe tranziția pozitivă;

- Observații referitoare la procesul **st\_viit** :
- este folosit pentru a determina starea viitoare a automatului;
  - lista de senzitivități cuprinde starea prezentă și intrările automatului;
  - starea viitoare a automatului depinde de starea prezentă și de intrările automatului;
  - selecția după starea prezentă se face prin declarații

<p><b>when</b>, iar selecția în funcție de intrările automatului se face cu declarații de tip <b>case</b>;</p> <p>Observații referitoare la procesul <b>iesire</b> :</p> <ul style="list-style-type: none"> <li>- este folosit pentru determinarea ieșirii automatului pe baza stării prezente a automatului;</li> <li>- lista de senzitivități cuprinde doar starea prezentă a automatului;</li> </ul> <p>Observații referitoare la afișarea stării automatului:</p> <ul style="list-style-type: none"> <li>- afișarea stării este necesară doar din considerente didactice;</li> <li>- în acest exemplu, codificarea stărilor este făcută de compilator – în consecință nu știm ce asocieri a făcut acesta;</li> <li>- propunem următoarea combinație pentru a vizualiza schimbarea stărilor (este ca și cum am calcula un nou set de ieșiri din automat): <ul style="list-style-type: none"> <li>- st0 → 111;</li> <li>- st1 → 001;</li> <li>- st2 → 010;</li> <li>- st3 → 011;</li> <li>- st4 → 100;</li> </ul> </li> </ul>	<pre> when st4 =&gt;     case data_in is         when "11" =&gt; stare_viit &lt;= st4;         when others =&gt; stare_viit &lt;= st0;     end case; when others =&gt; stare_viit &lt;= st0; end case; end process st_viit; -- proces ptr. calcul iesiri -- depind doar de starea prezenta iesire: process (stare_prez) begin     case stare_prez is         when st0 =&gt; data_out &lt;= '1';                         afis_stare &lt;= "111";         when st1 =&gt; data_out &lt;= '0';                         afis_stare &lt;= "001";         when st2 =&gt; data_out &lt;= '1';                         afis_stare &lt;= "010";         when st3 =&gt; data_out &lt;= '0';                         afis_stare &lt;= "011";         when st4 =&gt; data_out &lt;= '1';                         afis_stare &lt;= "100";         when others =&gt; data_out &lt;= '0';                         afis_stare &lt;= "110";     end case; end process iesire; end arh_comport; </pre>
Fișierul de constrângeri	<pre> NET "clk" LOC = "P56"; NET "reset" LOC = "P83"; NET "data_in&lt;1&gt;" LOC = "P37"; NET "data_in&lt;0&gt;" LOC = "P40"; NET "afis_stare&lt;2&gt;" LOC = "P62"; NET "afis_stare&lt;1&gt;" LOC = "P65"; NET "afis_stare&lt;0&gt;" LOC = "P67"; NET "data_out" LOC = "P82"; </pre>

◆ **Exemplul 2:** Descrierea în limbaj VHDL a unui automat de tip Moore (Varianta 2 - descriere folosind două procese)

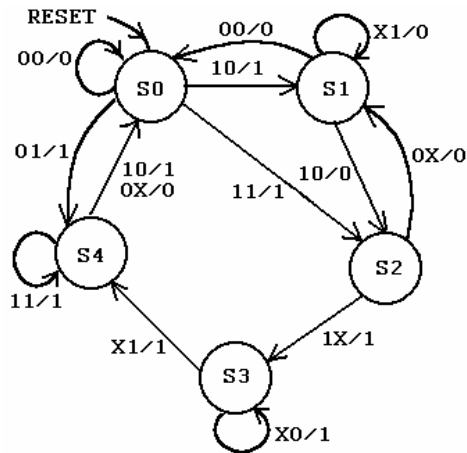
Observații	Codul VHDL
Secțiune dedicată includerii de librării	<pre> -- Exemplu de automat Moore cu 5 stari library ieee; use ieee.std_logic_1164.all; </pre>
<p>Secțiune dedicată descrierii entității. În cazul de față:</p> <ul style="list-style-type: none"> <li>- nume entitate este: <b>moore</b>;</li> <li>- intrarea de ceas : <b>clk</b>;</li> <li>- intrarea de ștergere: <b>reset</b>;</li> <li>- intrările automatului: <b>data_in</b>;</li> <li>- ieșirea automatului: <b>data_out</b>;</li> <li>- afișare stare automat: <b>afis_stare</b>;</li> </ul>	<pre> entity moore is port (clk, reset: in std_logic;       data_out: out std_logic(1 downto 0);       afis_stare: out std_logic(1 downto 0);       data_in: in std_logic_vector (1 downto 0)); end moore; </pre>
<p>Secțiune dedicată descrierii arhitecturii. În cazul de față:</p> <ul style="list-style-type: none"> <li>- numele arhitecturii este: <b>arh_comport</b>;</li> <li>- arhitectura este asociată cu entitatea: <b>moore</b>;</li> <li>- sunt folosite 2 procese;</li> </ul> <p>Observații referitoare la procesul <b>reg_stare</b> :</p> <ul style="list-style-type: none"> <li>- este folosit pentru a descrie funcționarea registrului de stare al automatului;</li> <li>- lista de senzitivități cuprinde intrarea de ceas și intrarea de reset;</li> <li>- intrarea de reset este activă pe zero logic;</li> <li>- intrarea de ceas este activă pe tranziția pozitivă;</li> </ul> <p>Graful de funcționare al automatului este:</p>	<pre> architecture arh_comport of moore is     subtype st_val is std_logic_vector(1 downto 0);     signal stare_prez, stare_viit: st_val;     constant stA: st_val := '00';     constant stB: st_val := '01';     constant stC: st_val := '10'; begin     -- proces ptr. registrul de stare     reg_stare: process (clk, reset)     begin         if (reset = '1') then             stare_prez &lt;= stA;         elsif (clk = '1' and clk'event) then             stare_prez &lt;= stare_viit;         end if;     end process reg_stare;      -- proces ptr. CLC     comb: process (stare_prez, data_in)     begin         stare_viit &lt;= stare_prez;     end process comb; end architecture arh_comport; </pre>



<p>Legenda:</p> <p>Observații referitoare la procesul <b>comb</b> :</p> <ul style="list-style-type: none"> <li>- este folosit pentru a determina starea viitoare a automatului;</li> <li>- lista de sensibilități cuprinde starea prezentă și intrările automatului;</li> <li>- starea viitoare a automatului depinde de starea prezentă și de intrările automatului;</li> </ul> <p>Observații referitoare la calculul ieșirilor :</p> <ul style="list-style-type: none"> <li>- se folosesc două asignări concurente, acestea se desfășoară în paralel cu procesele.</li> </ul> <p>Observații referitoare la afișarea stării automatului:</p> <ul style="list-style-type: none"> <li>- afișarea stării este necesară doar din considerente didactice;</li> <li>- în acest exemplu, codificarea stărilor este impusă de noi, deci este cunoscută;</li> </ul>	<pre> when stA =&gt;     if data_in="00" then         stare_viit &lt;= stB;     end if; when stB =&gt;     if data_in="11" then         stare_viit &lt;= stC;     end if; when stC =&gt;     if data_in="01" or data_in="10" then         stare_viit &lt;= stA;     end if; when others =&gt;     stare_viit &lt;= stA; end case; end process comb; -- asignari cocurente ptr. iesiri data_out(1)&lt;='1' when stare_prez =stB else '0'; data_out(0)&lt;='0' when stare_prez =stA else '1'; afis_stare&lt;= stare_prez; -- afisare stare end arh_comport;         </pre>
Fișierul de constrângeri:	<pre> NET "clk" LOC = "P56"; NET "reset" LOC = "P83"; NET "data_in&lt;1&gt;" LOC = "P37"; NET "data_in&lt;0&gt;" LOC = "P40"; NET "afis_stare&lt;1&gt;" LOC = "P65"; NET "afis_stare&lt;0&gt;" LOC = "P67"; NET "data_out&lt;1&gt;" LOC = "P80"; NET "data_out&lt;0&gt;" LOC = "P82";         </pre>

◆ **Exemplul 3:** Descrierea în limbaj VHDL a unui automat de tip Mealy (descriere folosind trei procese)

Observații	Codul VHDL
Secțiune dedicată includerii de librării	<pre> -- Example of a 5-state Mealy FSM library ieee; use ieee.std_logic_1164.all;         </pre>
Secțiune dedicată descrierii entităților. În cazul de față: <ul style="list-style-type: none"> <li>- nume entitate este: <b>mealy</b>;</li> <li>- intrarea de ceas : <b>clk</b>;</li> <li>- intrarea de ștergere: <b>reset</b>;</li> <li>- intrările automatului: <b>data_in</b>;</li> <li>- ieșirea automatului: <b>data_out</b>;</li> </ul>	<pre> entity mealy is port (     clk, reset: in std_logic;     data_out: out std_logic;     afis_stare:out std_logic_vector(1 downto 0);     data_in: in std_logic_vector (1 downto 0)); end mealy;         </pre>
Secțiune dedicată descrierii arhitecturii. În cazul de față: <ul style="list-style-type: none"> <li>- sunt folosite 3 procese: <b>reg_stare</b> pentru descrierea registrului de stare, <b>st_viit</b> pentru determinare stare viitoare și <b>iesire</b> pentru determinarea ieșirilor automatului;</li> </ul> Observații referitoare la procesul <b>reg_stare</b> : <ul style="list-style-type: none"> <li>- este folosit pentru a descrie funcționarea registrului de stare al automatului;</li> <li>- lista de sensibilități cuprinde intrarea de ceas și intrarea de reset;</li> <li>- intrarea de reset este activă pe zero logic;</li> <li>- intrarea de ceas este activă pe tranziția pozitivă;</li> </ul> Graful de funcționare al automatului este:	<pre> architecture arh_comport of mealy is type st_val is (st0, st1, st2, st3, st4); signal stare_prez, stare_viit: st_val; begin  -- proces ptr. registrul de stare reg_stare: process (clk, reset) begin     if (reset = '1') then         stare_prez &lt;= st0;     elsif (clk'event and clk ='1') then         stare_prez &lt;= stare_viit;     end if; end process reg_stare;  -- proces ptr. determinare stare viitoare st_viit: process (stare_prez, data_in)         </pre>



Observații referitoare la procesul **st\_viit**:

- este folosit pentru a determina starea viitoare a automatului;
- lista de senzitivități cuprinde starea prezentă și intrările automatului;
- starea viitoare a automatului depinde de starea prezentă și de intrările automatului;
- selecția după starea prezentă se face prin declarații **when**, iar selecția în funcție de intrările automatului se face cu declarații de tip **case**;

Observații referitoare la procesul **iesire**:

- este folosit pentru determinarea ieșirii automatului pe baza stării prezente și a intrărilor automatului;
- lista de senzitivități cuprinde starea prezentă și intrările automatului;

Observații referitoare la afișarea stării automatului:

- afișarea stării este necesară doar din considerente didactice;
- codificarea binară a stărilor **S0, S1, S2, S3, S4** este lăsată pe seama compilatorului de VHDL;
- propunem următoarea combinație pentru a vizualiza schimbarea stărilor (este ca și cum am calcula un nou set de ieșiri din automat):
  - S0 → 111; S1 → 001; S2 → 010;
  - S3 → 011; S4 → 100;

```

begin
case stare_prez is
when st0 =>
    case data_in is
        when "00" => stare_viit<= st0;
        when "01" => stare_viit<= st4;
        when "10" => stare_viit<= st1;
        when "11" => stare_viit<= st2;
        when others => null;
    end case;
when st1 =>
    case data_in is
        when "00" => stare_viit<= st0;
        when "10" => stare_viit<= st2;
        when others => stare_viit<= st1;
    end case;
when st2 =>
    case data_in is
        when "00" => stare_viit<= st1;
        when "01" => stare_viit<= st1;
        when "10" => stare_viit<= st3;
        when "11" => stare_viit<= st3;
        when others => null;
    end case;
when st3 =>
    case data_in is
        when "01" => stare_viit<= st4;
        when "11" => stare_viit<= st4;
        when others => stare_viit<= st3;
    end case;
when st4 =>
    case data_in is
        when "11" => stare_viit<= st4;
        when others => stare_viit<= st0;
    end case;
when others => stare_viit<= st0;
end case;
end process st_viit;

```

```

-- proces ptr. calcul iesire din automat
iesire: process (stare_prez, data_in)
begin
case stare_prez is
when st0 =>
    afis_stare<= "000";
    case data_in is
        when "00" => data_out <= '0';
        when others => data_out <= '1';
    end case;
when st1 => data_out <= '0';
    afis_stare<= "001";
when st2 =>
    afis_stare<= "010";
    case data_in is
        when "00" => data_out <= '0';
        when "01" => data_out <= '0';
        when others => data_out <= '1';
    end case;
when st3 => data_out <= '1';
    afis_stare<= "011";
when st4 =>
    afis_stare<= "100";
    case data_in is
        when "10" => data_out <= '1';
        when "11" => data_out <= '1';
        when others => data_out <= '0';
    end case;
when others => data_out <= '0';
    afis_stare<= "110";
end case;
end process iesire;
end arh_comport;

```

Fișierul de constrângeri:	<p>NET "clk" LOC = "P56";  NET "reset" LOC = "P83";  NET "data_in&lt;1&gt;" LOC = "P37";  NET "data_in&lt;0&gt;" LOC = "P40";  NET "afis_stare&lt;2&gt;" LOC = "P62";  NET "afis_stare&lt;1&gt;" LOC = "P65";  NET "afis_stare&lt;0&gt;" LOC = "P67";  NET "data_out" LOC = "P82";</p>
---------------------------	--



### 3. Desfășurarea lucrării

#### 3.1. Analiza automatelor sincrone.

- A. Folosind metode similare celor prezentate în partea teoretică, determinați grafurile de tranziție pentru automatele din figura 6.

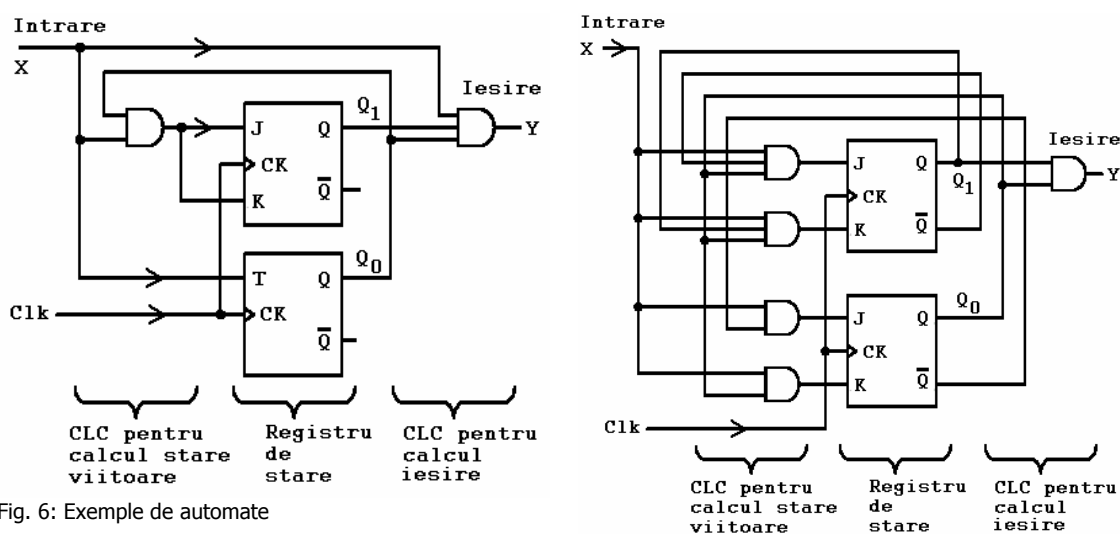


Fig. 6: Exemple de automate

#### 3.2. Sinteza automatelor sincrone.

- A. Folosind metodologia de proiectare a automatelor sincrone determinați schema logică a unui numărator sincron bidirecțional pe 3 biți.  
B. Pentru automatele descrise în exemplele VHDL, determinați câte o schemă logică folosind bistabili la alegere.

#### 3.3. Aplicații cu VHDL.

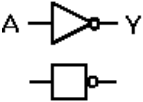
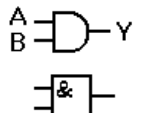
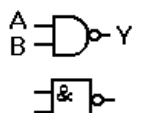
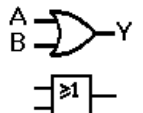
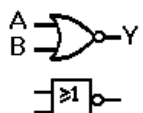
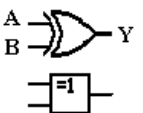
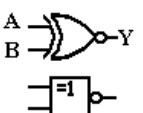
- A. Verificați pe macheta de laborator cu CPLD, descrierile VHDL prezentate ca exemple.  
B. Descrieți în limbaj VHDL automatele sincrone ce funcționează după grafurile de tranziție din figura 2.  
C. Folosind facilitatea ISE WebPack prin care se permite introducerea schemelor logice ale sistemelor digitale, implementați schemele din figura 6 și comparați rezultatele obținute cu cele de la subpunctul 3.1. din desfășurarea lucrării.

#### Mode de lucru:

- variabilele de intrare vor fi implementate cu switch-uri de pe macheta de laborator;
- starea automatului se afișează în cod binar pe două sau trei LED-uri existente pe macheta de laborator;
- starea semnalelor de ieșire se afișează în cod binar pe alte LED-uri (rămase libere) de pe macheta de laborator;
- semnalul de ceas al automatului se preia de la oscilatorul de 25,175MHz de pe macheta de laborator, eventual printr-o divizare prealabilă a frecvenței.



**Anexa 1. Porți logice: simboluri și tabele de adevăr**

NR. CRT.	DENUMIRE	SIMBOL	TABEL DE ADEVĂR	OBSERVAȚII															
1	<b>INVERSOR</b>  (NOT)		<table><tr><td>A</td><td><math>Y = \overline{A}</math></td></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	$Y = \overline{A}$	0	1	1	0	<ul style="list-style-type: none"><li>Realizează negarea variabilei de intrare;</li></ul>									
A	$Y = \overline{A}$																		
0	1																		
1	0																		
2	ȘI  (AND)		<table><tr><td>A</td><td>B</td><td><math>Y = AB</math></td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	$Y = AB$	0	0	0	0	1	0	1	0	0	1	1	1	<ul style="list-style-type: none"><li>Realizează funcția de minim a variabilelor de intrare.</li><li>Funcționare:<ul style="list-style-type: none"><li>- dacă A=0 atunci Y=0 (poartă blocată);</li><li>- dacă A=1 atunci Y=B (poartă deschisă);</li></ul></li><li>Există porți AND cu 2,3,4 sau 8 intrări.</li></ul>
A	B	$Y = AB$																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
3	ȘI- NU  (NAND)		<table><tr><td>A</td><td>B</td><td><math>Y = \overline{AB}</math></td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$Y = \overline{AB}$	0	0	1	0	1	1	1	0	1	1	1	0	<ul style="list-style-type: none"><li>Realizează inversul funcției logice de la punctul 2.</li><li>Există porți NAND cu 2,3,4 sau 8 intrări.</li></ul>
A	B	$Y = \overline{AB}$																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
4	SAU  (OR)		<table><tr><td>A</td><td>B</td><td><math>Y = A + B</math></td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	$Y = A + B$	0	0	0	0	1	1	1	0	1	1	1	1	<ul style="list-style-type: none"><li>Realizează funcția de maxim al variabilelor de intrare.</li><li>Funcționare:<ul style="list-style-type: none"><li>- dacă A=1 atunci Y=1 (poartă blocată);</li><li>- dacă A=0 atunci Y=B (poartă deschisă);</li></ul></li><li>Există porți OR cu 2,3,4 sau 8 intrări.</li></ul>
A	B	$Y = A + B$																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
5	SAU - NU  (NOR)		<table><tr><td>A</td><td>B</td><td><math>Y = \overline{A + B}</math></td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$Y = \overline{A + B}$	0	0	1	0	1	0	1	0	0	1	1	0	<ul style="list-style-type: none"><li>Realizează inversul funcției logice de la punctul 4.</li><li>Există porți NOR cu 2,3,4 sau 8 intrări.</li></ul>
A	B	$Y = \overline{A + B}$																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
6	SAU EXCLUSIV (XOR)		<table><tr><td>A</td><td>B</td><td><math>Y = A \oplus B</math></td></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	$Y = A \oplus B$	0	0	0	0	1	1	1	0	1	1	1	0	<ul style="list-style-type: none"><li>Moduri de interpretare:<ul style="list-style-type: none"><li>- realizează adunarea modulo doi;</li><li>- funcție de anticoincidență;</li><li>- funcție de complementare comandată:<ul style="list-style-type: none"><li>- dacă A=0 atunci Y=B</li><li>- dacă A=1 atunci Y= <math>\overline{B}</math></li></ul></li></ul></li></ul>
A	B	$Y = A \oplus B$																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	
7	SAU EXCLUSIV NEGAT (XNOR)		<table><tr><td>A</td><td>B</td><td><math>Y = \overline{A \oplus B}</math></td></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	$Y = \overline{A \oplus B}$	0	0	1	0	1	0	1	0	0	1	1	1	<ul style="list-style-type: none"><li>Realizează inversul funcției logice de la punctul 6.</li></ul>
A	B	$Y = \overline{A \oplus B}$																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	1																	

## Anexa 2 : Formele canonice ale funcțiilor binare

Forma canonică constituie o altă modalitate de reprezentare a funcțiilor binare în care, descrierea funcționării se face analitic. Scrierea canonică a unei funcții binare se poate face în două moduri:

- prin utilizarea unei sume a tuturor **mintermenilor** pentru care funcția binară prezintă valoarea "1" - cazul primei forme canonice, denumită și **formă canonică disjunctivă**;
- prin utilizarea unui produs al tuturor **maxtermenilor** pentru care funcția binară prezintă valoarea "0" - cazul celei de-a doua forme canonice denumită și **formă canonică conjunctivă**.

### Definiții:

- *Mintermenul este un produs logic la care participă fiecare variabilă de intrare o singură dată: sub formă directă - dacă variabila de intrare este unu logic, sau sub formă negată - dacă variabila de intrare este zero logic.*
- *Maxtermenul este o sumă logică, (un produal), la care participă fiecare variabilă de intrare luată o singură dată: sub formă directă - dacă variabila de intrare este zero logic, sau sub formă negată - dacă variabila de intrare este unu logic.*

**Exemplu:** Pentru funcția de transfer prezentată în figura 2 obținem:

- **Modul de definire al min/max - termenilor** se prezintă în tabelul de mai jos:

Linie	Vector de intrare <b>X</b>	<b>MINTERMEN</b>	<b>MAXTERMEN</b>	Vector de ieșire <b>Y</b>
	$x_3 \ x_2 \ x_1$			$y_2 \ y_1$
0	0 0 0	$m_0 = \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1$	$M_0 = x_3 + x_2 + x_1$	0 1
1	0 0 1	$m_1 = \bar{x}_3 \cdot \bar{x}_2 \cdot x_1$	$M_1 = x_3 + x_2 + \bar{x}_1$	0 1
2	0 1 0	$m_2 = \bar{x}_3 \cdot x_2 \cdot \bar{x}_1$	$M_2 = x_3 + \bar{x}_2 + x_1$	1 0
3	0 1 1	$m_3 = \bar{x}_3 \cdot x_2 \cdot x_1$	$M_3 = x_3 + \bar{x}_2 + \bar{x}_1$	1 1
4	1 0 0	$m_4 = x_3 \cdot \bar{x}_2 \cdot \bar{x}_1$	$M_4 = \bar{x}_3 + x_2 + x_1$	0 0
5	1 0 1	$m_5 = x_3 \cdot \bar{x}_2 \cdot x_1$	$M_5 = \bar{x}_3 + x_2 + \bar{x}_1$	0 1
6	1 1 0	$m_6 = x_3 \cdot x_2 \cdot \bar{x}_1$	$M_6 = \bar{x}_3 + \bar{x}_2 + x_1$	0 0
7	1 1 1	$m_7 = x_3 \cdot x_2 \cdot x_1$	$M_7 = \bar{x}_3 + \bar{x}_2 + \bar{x}_1$	1 1

- **Prima formă canonică** (forma disjunctivă) a celor două funcții binare:

$$y_2 = m_2 + m_3 + m_7 = \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 + \bar{x}_3 \cdot x_2 \cdot x_1 + x_3 \cdot x_2 \cdot x_1 = \left| \begin{array}{c} \bar{x}_3 \cdot x_2 \cdot \bar{x}_1 \\ \bar{x}_3 \cdot x_2 \cdot x_1 \\ x_3 \cdot x_2 \cdot x_1 \end{array} \right|$$

$$y_1 = m_0 + m_1 + m_3 + m_5 + m_7 = \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1 + \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 + \bar{x}_3 \cdot x_2 \cdot x_1 + x_3 \cdot \bar{x}_2 \cdot x_1 + x_3 \cdot x_2 \cdot x_1 =$$

$$= \left| \begin{array}{c} \bar{x}_3 \cdot \bar{x}_2 \cdot \bar{x}_1 \\ \bar{x}_3 \cdot \bar{x}_2 \cdot x_1 \\ \bar{x}_3 \cdot x_2 \cdot x_1 \\ x_3 \cdot \bar{x}_2 \cdot x_1 \\ x_3 \cdot x_2 \cdot x_1 \end{array} \right|$$

- **A doua formă canonică** (forma conjunctivă) a celor două funcții binare:

$$y_2 = M_0 \cdot M_1 \cdot M_4 \cdot M_5 \cdot M_6 = (x_3 + x_2 + x_1)(x_3 + x_2 + \bar{x}_1)(\bar{x}_3 + x_2 + x_1)(\bar{x}_3 + x_2 + \bar{x}_1)(\bar{x}_3 + \bar{x}_2 + x_1) =$$

$$= \left| \begin{array}{c} x_3 \\ x_2 \\ x_1 \end{array} \right| \left| \begin{array}{c} x_3 \\ x_2 \\ \bar{x}_1 \end{array} \right| \left| \begin{array}{c} \bar{x}_3 \\ x_2 \\ x_1 \end{array} \right| \left| \begin{array}{c} \bar{x}_3 \\ x_2 \\ \bar{x}_1 \end{array} \right| \left| \begin{array}{c} \bar{x}_3 \\ \bar{x}_2 \\ x_1 \end{array} \right|$$

$$y_1 = M_2 \cdot M_4 \cdot M_6 = (x_3 + \bar{x}_2 + x_1)(\bar{x}_3 + x_2 + x_1)(\bar{x}_3 + \bar{x}_2 + x_1) = \left| \begin{array}{c} x_3 \\ \bar{x}_2 \\ x_1 \end{array} \right| \left| \begin{array}{c} \bar{x}_3 \\ x_2 \\ x_1 \end{array} \right| \left| \begin{array}{c} \bar{x}_3 \\ \bar{x}_2 \\ x_1 \end{array} \right|$$

### Observații:

- Se remarcă faptul că atât mintermenii, cât și maxtermenii, sunt dependenți numai de starea logică a variabilelor de intrare și nu depind de valorile de ieșire ale funcției logice.
- Din analiza expresiilor celor două forme canonice se observă că, deși descriu aceleași funcție logică, cu cât o formă canonică este mai restrânsă cu atât cealaltă este mai amplă.
- În practică se recomandă utilizarea primei forme canonice pentru funcțiile binare care au un număr mai redus de valori "adevărat", printre valorile funcției, și a formei a doua în caz contrar.
- Pentru exemplul considerat, se obțin expresii mai simple dacă se utilizează prima formă canonică pentru  $y_2$ , respectiv cea de-a doua formă pentru  $y_1$ .

### Anexa 3 : Bistabili

O deosebire fundamentală între un latch și un bistabil constă în modul în care are loc modificarea ieșirilor. Astfel, pentru latch-uri ieșirea se poate modifica oriunde pe nivelul activ al semnalului de ceas, pe când, pentru bistabili modificarea ieșirii se face întotdeauna sincron cu o anumită tranziție a semnalului de ceas, denumită tranziție de basculare.

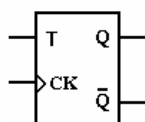
#### Definiții:

- **Tranziția de basculare**, sau **tranziția activă**, a semnalului de ceas este acea tranziție, negativă sau după caz pozitivă, care poate modifica ieșirea bistabilului.
- **Tranziția neutră** a semnalului de ceas este opusul tranziției de basculare. Această tranziție nu poate modifica ieșirea bistabilului în nici o situație.

#### Tipuri de bistabili

##### 1. Bistabilul de tip T

SIMBOL



FORME ALE TABELULUI DE ADEVĂR

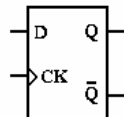
T	$Q_n$	$Q_{n+1}$
0	0	0
0	1	1
1	0	1
1	1	0

T	$Q_n$	$Q_{n+1}$
0	Q	Q
1	Q	$\bar{Q}$

T	Q
0	memorare
1	basculare

##### 2. Bistabilul de tip D

SIMBOL



FORME ALE TABELULUI DE ADEVĂR

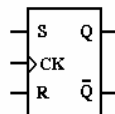
D	$Q_n$	$Q_{n+1}$
0	0	0
0	1	0
1	0	1
1	1	1

D	$Q_n$	$Q_{n+1}$
0	Q	0
1	Q	1

D	Q
0	0
1	1

##### 3. Bistabilul de tip SR

SIMBOL



FORME ALE TABELULUI DE ADEVĂR

S	R	$Q_n$	$Q_{n+1}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	?
1	1	1	?

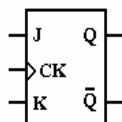
S	R	$Q_n$	$Q_{n+1}$
0	0	Q	Q
0	1	*	0
1	0	*	1
1	1	*	?

S	R	Q
0	0	memorare
0	1	0
1	0	1
1	1	?

\* don't care (poate fi zero logic sau unu logic)

##### 4. Bistabilul de tip JK

SIMBOL



FORME ALE TABELULUI DE ADEVĂR

J	K	$Q_n$	$Q_{n+1}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

J	K	$Q_n$	$Q_{n+1}$
0	0	Q	Q
0	1	*	0
1	0	*	1
1	1	Q	$\bar{Q}$

J	K	Q
0	0	memorare
0	1	0
1	0	1
1	1	basculare

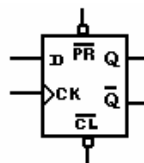
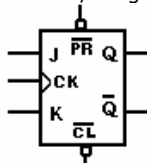
\* don't care (poate fi zero logic sau unu logic)

Bistabilii realizați în structuri integrate prezintă, pe lângă intrările specifice, una sau două intrări ce sunt tratate prioritar. Aceste intrări, când sunt activate, permit aducerea forțată a bistabilului într-o stare logică în mod independent de semnalul de ceas și indiferent de starea logică a intrărilor pasive.

Intrarea de aducere forțată a bistabilului în starea zero, este de regulă activă pe nivelul Low al semnalului aplicat, și este denumită *RESET* sau *CLEAR*; notațiile uzuale pentru această intrare sunt  $\bar{R}$  respectiv  $\bar{CL}$ .

Intrarea de aducere forțată a bistabilului în starea unu, este de regulă activă pe nivelul Low al semnalului aplicat, și este denumită *SET* sau *PRESET*; notațiile uzuale pentru această intrare sunt  $\bar{S}$  respectiv  $\bar{PR}$ .

Ținând cont de observațiile anterioare, în figura de mai jos se prezintă două simboluri complete de bistabili.



## Lista conexiunilor resurselor I/O la circuitul CPLD de tip XC95108

### 1. Resurse de pe placa de bază

Resursa	Pin CPLD XC95108
OSC (25,175 MHz)	P9
LD 2	P1
BTN1	P2

### 2. Resurse de pe placa DIO1

Butoane cu revenire	Pin CPLD XC95108
BTN1	P56
BTN 2	P57
BTN 3	P58
BTN 4	P61
BTN 5	P83

Activare digiți afișaj (Anod comun )	Pin CPLD XC95108
A1	P63
A2	P66
A3	P68
A4	P70

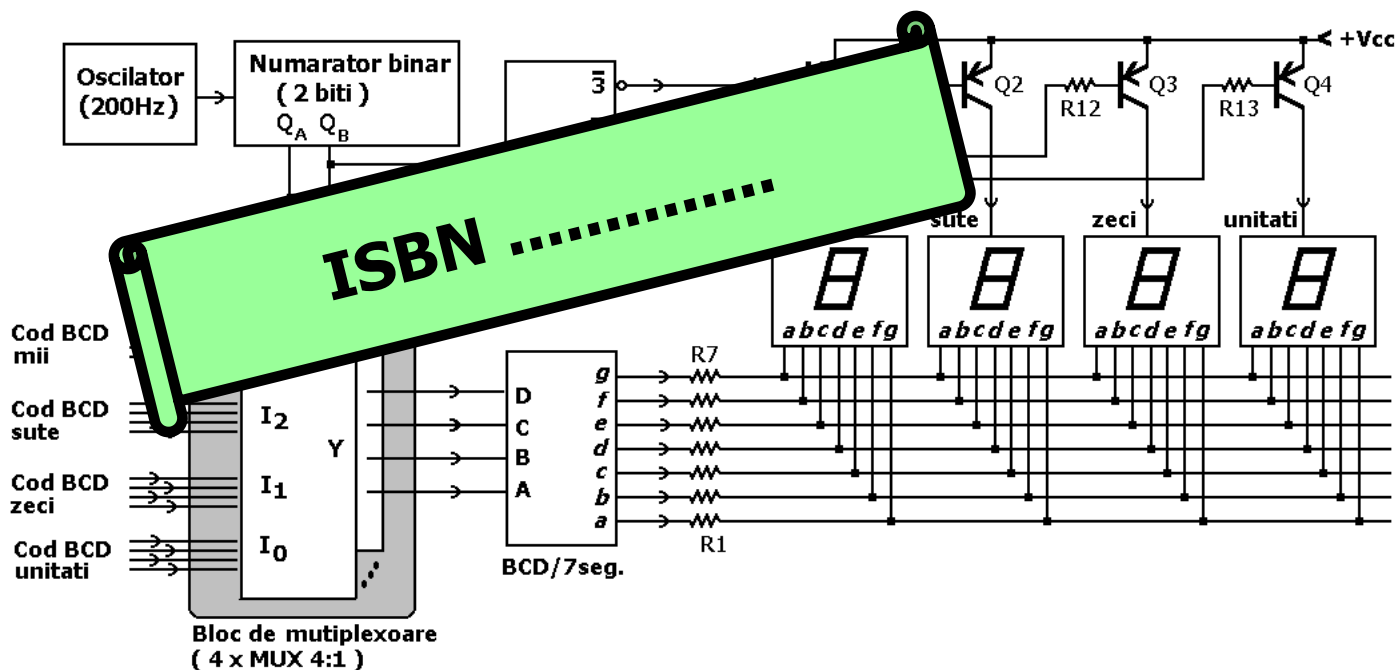
Switch-uri	Pin CPLD XC95108
SW1	P37
SW2	P40
SW3	P43
SW4	P45
SW5	P47
SW6	P50
SW7	P52
SW8	P54

Activare segmente afișaj (comanda pe catod)	Pin CPLD XC95108
CA	P39
CB	P41
CC	P44
CD	P46
CE	P48
CF	P51
CG	P53

Bareta cu LED-uri	Pin CPLD XC95108
LD1	P62
LD 2	P65
LD 3	P67
LD 4	P69
LD 5	P71
LD 6	P75
LD 7	P80
LD 8	P82

Semnale PS2	Pin CPLD XC95108
KCLK	P35
KDAT	P33

Semnale VGA	Pin CPLD XC95108
HS	P34
HV	P36
R	P24
G	P26
B	P32



Semnal generat de oscilator (CK)

Stare numarator ( $Q_1 Q_0$ )

Cifra afisata

