UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Proiectarea sistemelor numerice

Proiect:

Lift P+12

Facultatea: Automatică și Calculatoare

Specializarea: Calculatoare și Tehnologia Informației

An universitar: 2019 – 2020

Studenți: Lazea Dragoș – Bogdan

Bozdog Raluca – Delia

Grupa: 30213

Profesor laborator: Pop Diana

Cuprins

1. Specificația proiectului

Să se proiecteze un automat care comandă un lift într-un hotel cu P+12 etaje. Liftul trebuie să răspundă solicitărilor persoanelor aflate în interior și cererilor exterioare (sus, jos) care apar pe parcurs de la ușile aflate la fiecare nivel. Ordinea de onorare a cererilor ține cont de sensul de mers (urcare sau coborâre). Se onorează cererile în ordinea etajelor, indiferent de unde provin ele (lift sau exterior). Liftul are o intrare care sesizează depășirea greutății maxime admise și nu pornește în acest caz. Plecarea nu are loc dacă ușile nu sunt închise. Ușile trebuie să stea deschise un interval de timp programabil. Ușile nu se închid dacă există vreo persoană în ușă. Viteza liftului va fi selectabilă între două valori: 1 sau 3 secunde/ etaj. Se consideră că în momentul inițial liftul se găsește la parter, cu ușile deschise.

2. Descriere schemă bloc cu componente

La fel ca în cazul oricărui automat, schema bloc a automatului ce controlează liftul este formată din două componente: **unitatea de comandă** și **unitatea de execuție**. Unitatea de comandă ia decizii cu privire la următoarea stare în care trebuie să se găsească automatul, și transmite informațiile necesare către unitatea de execuție; aceasta din urmă execută ceea ce trebuie să realizeze automatul, pe baza deciziilor luate de către unitatea de comandă.

Unitatea de comandă conține 3 comparatoare având principii de funcționare și scopuri diferite *(comp_decizie, comp_13, comparator_n2*), un numărător zecimal și un bistabil de tip D care funcționează ca un ansamblu (*n_timp* și *bist_D*), senzorul de greutate și senzorul de prezență.

Unitatea de execuție este alcătuită din două divizoare de frecvență (*div_1* și *div_3*), 4 numărătoare (*numarator_lift, numarator_int, numarator_sus, numarator_jos*), 4 decodificatoare (*decod_lift, decod_int, decod_sus, decod_jos*), 3 registre (*registru_sus, registru_jos, registru_dest*) și componenta de afișare.

Atât în cadrul unității de comandă cât și în cadrul celei de execuție există și porți logice, necesare realizării unor funcții logice a căror utilitate va fi explicată ulterior.

Schema bloc

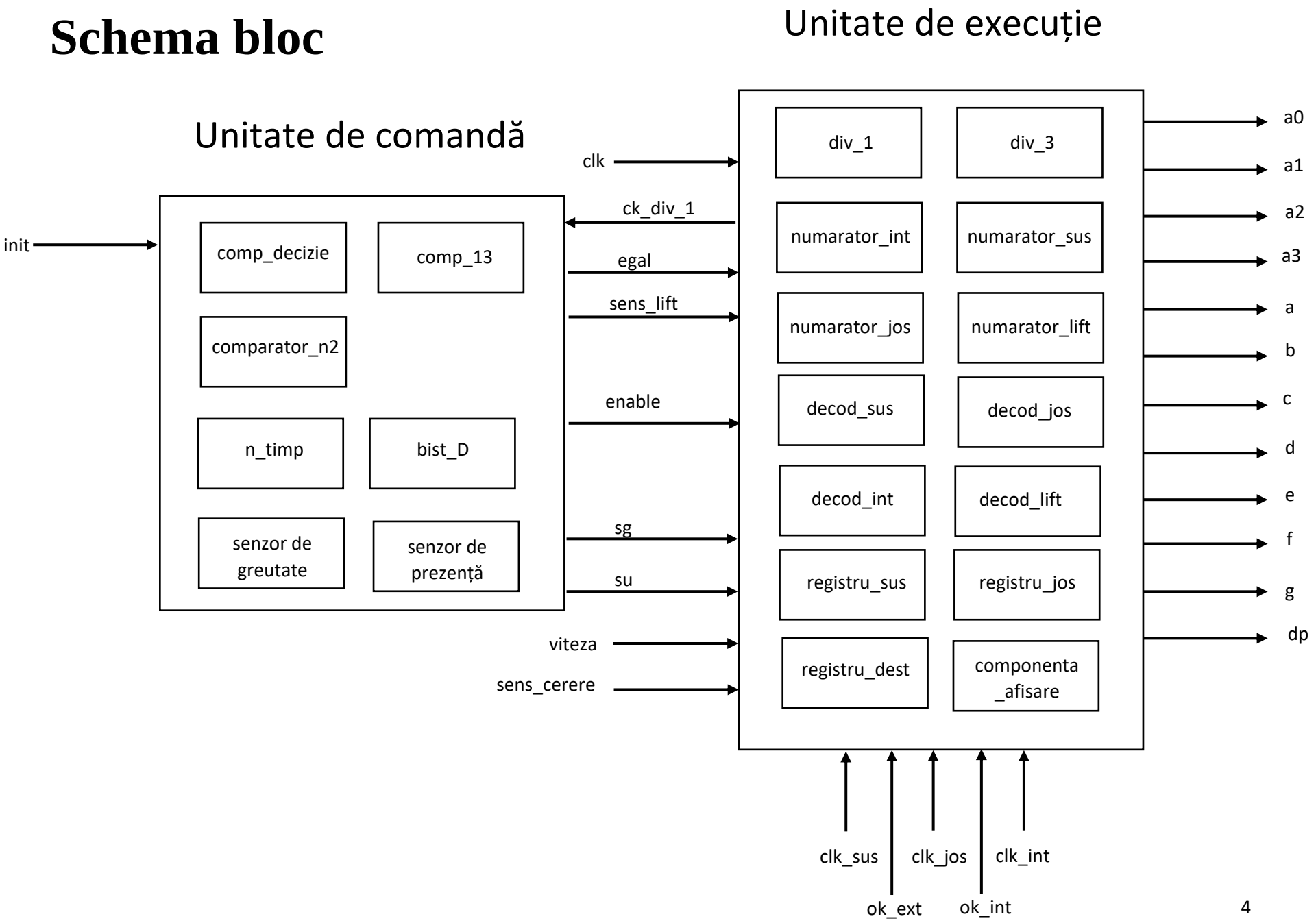
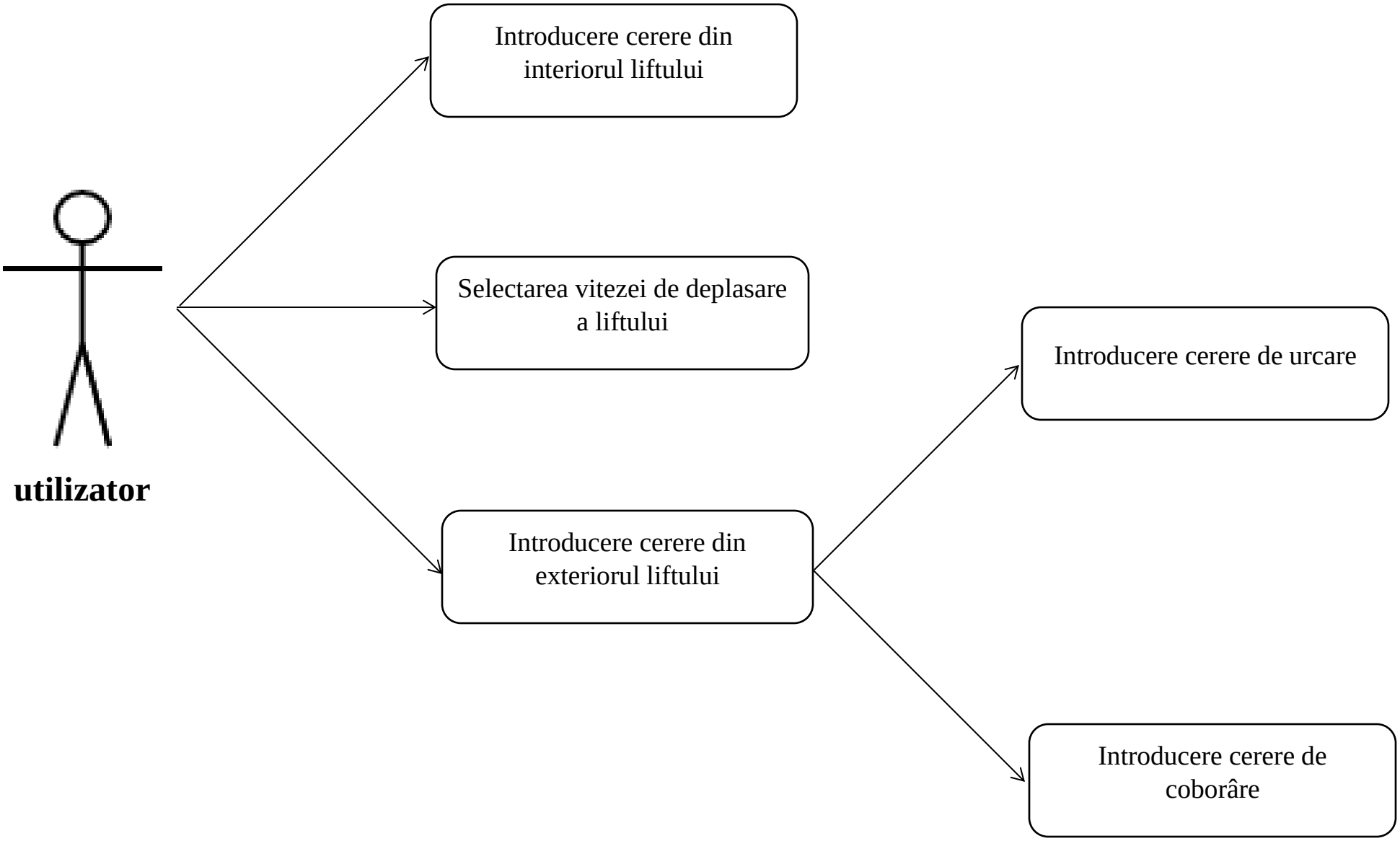
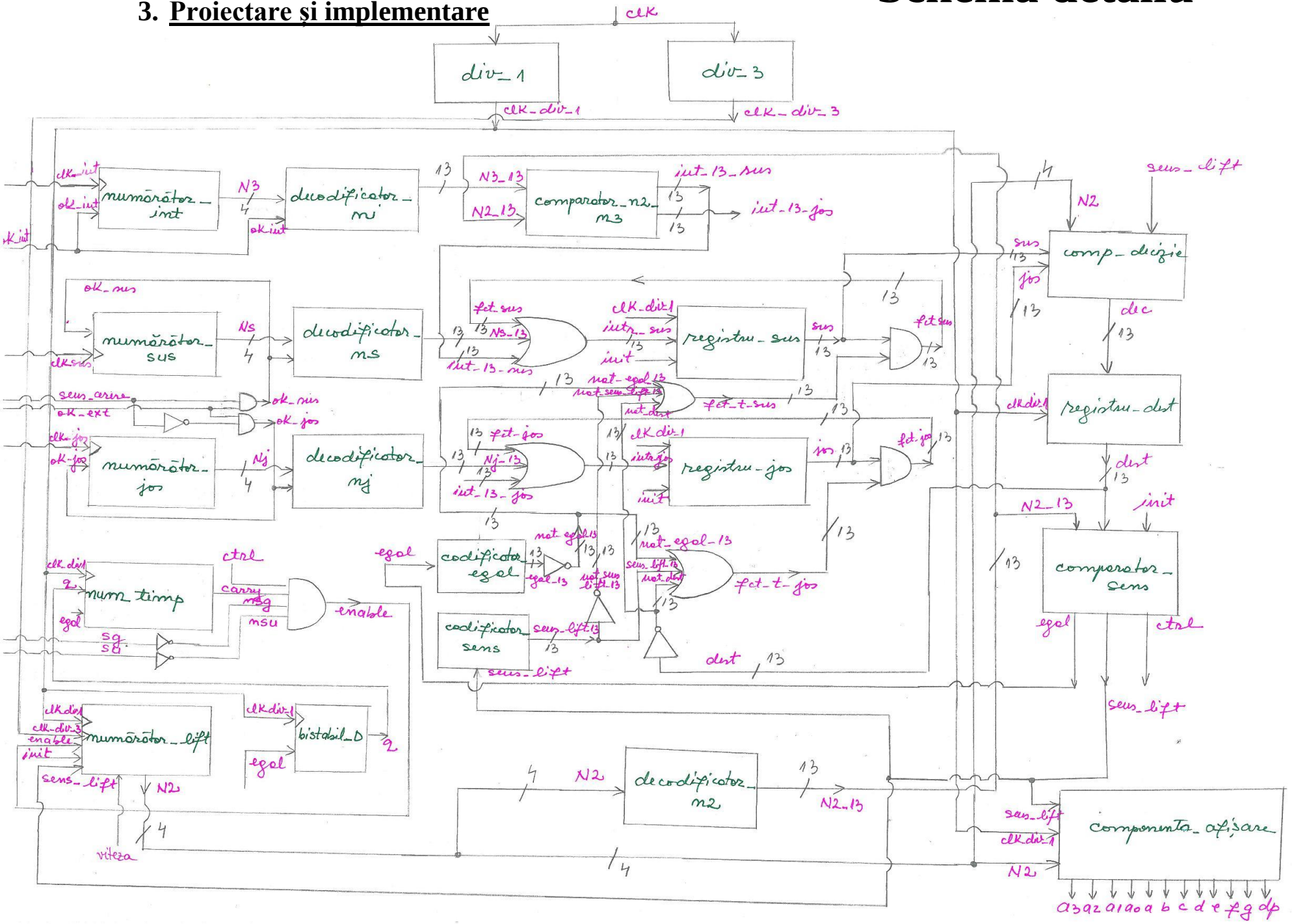

Unitate de execuție
Unitate de comandă
init
comp_decizie
comp_13
comparator_n2
n_timp
bist_D
senzor de greutate
senzor de prezență
clk
ck_div_1
egal
sens_lift
enable
sg
su
viteza
sens_cerere
div_1
div_3
numarator_int
numarator_sus
numarator_jos
numarator_lift
decod_sus
decod_jos
decod_int
decod_lift
registru_sus
registru_jos
registru_dest
componenta _afisare
a0
a1
a2
a3
a
b
c
d
e
f
g
dp
clk_sus
clk_jos
clk_int
ok_ext
ok_int

Diagrama UML



Schema detaliu

3. Proiectare și implementare



3.1 Descriere schema de detaliu

Semnalul de CLOCK primit de la oscilatorul cu cuarț de pe placă este intrare pentru cele două divizoare de frecvență, *div_1* și div_3, care divizează frecvența până la intervalul de timp de o secundă, respectiv de 3 secunde, generând, în mod corespunzător, semnalele **clk_div_1** și **clk_div_3**.

Ambele semnale de CLOCK vor fi intrări pentru numărătorul *numarator_lift*, unde se va alege ce semnal va fi folosit pentru a reda deplasarea liftului între două etaje pe baza semnalului **viteza**, semnal de intrare al automatului. **clk_div_1** va fi semnalul de CLOCK al tuturor componentelor sincrone, cu excepția numărătoarelor folosite la introducerea cererilor din interiorul sau din exteriorul liftului (*num_timp, bistabil_D, registru_sus, registru_jos, registru_dest, componenta afisare*), deoarece frecvența de oscilație a CLOCK-ului de pe placă este prea mare pentru ca aceste componente să funcționeze.

Faptul că semnalul de CLOCK al numărătorului *num_timp* este **clk_div_1**, indică faptul că fiecare impuls de tact reprezintă, de fapt, o secundă scursă în realitate. Fiind un numărător modulo 10, *num_timp* va număra 10 secunde, bazându-se pe anumite comparații între actuala și fosta valoare a semnalului intermediar **egal** pentru a genera 0 sau 1 pe ieșirea **carry**. Fosta valoare a lui **egal** este obținută cu ajutorul bistabilului de întârziere *bistabil_D*, care are ca intrare semnalul **egal**, și ca ieșire semnalul **q**.

Semnalul de **carry**, alături de semnalele **nsg** și **nsu** (rezultate ale trecerii printr_un inversor a intrărilor **sg** și **su** ale automatului), dar și de semnalul **ctrl** (ieșire a comparatorului *comp_sens* care este 0 dacă nu există încă nicio cerere pe care liftul să o onoreze, și 1 în rest), trecute prin poarta și cu 4 intrări, *si_4*, produce semnalul **enable**, care controlează numărătorul ce indică etajul curent al liftului, *numarator_lift*. Liftul se poate deplasa dacă au trecut cel puțin 10 secunde de la ultima staționare (**carry**=1), dacă greutatea maximă admisă nu este depășită (**nsg**=1), dacă închiderea ușilor nu este blocată de prezența cuiva în ușă (**nsu**=1), și dacă există o destinație spre care liftul să se îndrepte (**ctrl**=1), toate acestea simultan.

Numărătorul **numarator_lift** se resetează la inițializarea automatului cu semnalul **init**. Intrarea **sens_lift** este cea care comandă numărarea crescătoare sau descrescătoare a numărătorului: dacă **sens_lift** este 1, liftul urcă, deci numărarea va fi crescătoare de la 0 la 12; dacă **sens_lift** este 0, liftul coboară, deci se va număra descrescător de la 12 la 0. Ieșirea componentei *numarator_lift* este semnalul intermediar **N2**, pe 4 biți, ce reprezintă codul binar al etajului curent.

La introducere unei cereri din interiorul liftului, se va folosi numărătorul *numarator_int*, care are ca semnal de CLOCK semnalul de intrare a la automatului cu denumirea **clk_int**. După ce a fost stabilit etajul spre care se dorește să se facă deplasarea din interiorul liftului, va fi folosit semnalul de intrare **ok_int**, care activează decodificatorul *decodificator_ni*, permițând transpunerea informației din cod binar pe 4 biți, în convenția stabilită pentru reținerea cererilor în registrele pe 13 biți (din cei 13 biți, 12 vor fi 0, iar unul singur va fi 1, și anume cel corespunzător etajului dorit ca destinație).

Cererea din interior se va găsi pe semnalul **N3_13**, care constituie o intrare a comparatorului *comparator_n2_n3*, alături de **N2_13**, ieșirea componentei *decodificator_n2*, cu o funcționare similară cu cea a lui *decodificator_ni*. Comparatorul decide, în funcție de poziția relativă a celor două etaje, dacă solicitarea din interior se va înregistra ca fiind cerere de urcare sau de coborâre.

Atunci când se dorește memorarea unei cereri de urcare din exteriorul liftului, cu ajutorul semnalului **clk_sus**, semnal de intrare al automatului, se va număra pe *numarator_sus* până la etajul de la care se face cererea, moment în care se va introduce valoarea 1 pe semnalul de intrare **ok_ext**. Trebuie menționat că pentru a implementa fizic, pe plăcuță cu FPGA automatul de control al liftului, este nevoie de o intrare suplimentară numită **sens_cerere**, corelată cu intrarea **ok_ext**, pentru obținerea semnalelor **ok_sus** și **ok_jos**, pe post de semnal de restare pentru numărătoarele *numarator_sus* și numarator*_jos*, și **enable** pentru decodificatoarele *decodificator_ns* și *decodificator_nj*.

Pentru reținerea informațiilor despre cereri în registrele *registru_sus* și *registru_jos* se folosește câte o poartă logică SAU cu 3 intrări pentru fiecare dintre ele. Pentru *registru_sus*, intrările acestei porți sunt semnalele intermediare **Ns_13** (ieșire a lui *decodificator_ns*), **int_13_sus** (ieșire a lui *comparator_n2_n3*) și **fct_sus** (funcție de actualizare a fostului conținut din registru). Acestea sunt toate situațiile în care *registru_sus* trebuie să încarce solicitări: cerere de urcare din exterior, cerere de urcare din interior și menținerea cererilor neonorate din *registru_sus*, odată cu ștergerea cererilor onorate. În același mod funcționează și registrul *registru_jos*, care poate reține informații de la *decodificator_nj* (**Nj_13**), *comparator_n2_n3* (**int_13_jos**), și păstrând, totodată cererile care nu au fost finalizate (**fct_jos**). Funcțiile SUS și JOS, ale căror rezultate sunt semnalele **fct_sus** și **fct_jos** sunt explicate pe larg la capitolul de proiectare a componentelor.

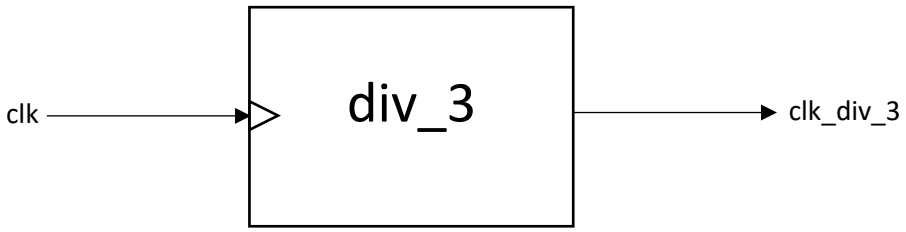
Comparatorul care decide următoarea destinație a liftului, comp*_decizie*, are ca intrări semnalele **sus** și **jos**, ieșiri ale componentelor *registru_sus*, respectiv *registru_jos*, și semnalul **N2**, ieșire a numărătorului *numarator_lift*, ținând cont de valoare intrării **sens_lift**, care indică sensul de deplasare al liftului: 1 pentru urcare, 0 pentru coborâre. Ieșirea acestuia este semnalul intermediar **dec**, ce se încarcă paralel pe *registru_dest*, ce reține la fiecare moment cererea cea mai prioritară din punctul de vedere al sensului deplasării și al etajului la care se află liftul, cu posibilitatea de a se modifica, fără a afecta registrele sus și jos, dacă cererea nu a fost onorată.

Ieșirea acestui registru, semnalul **dest**, ajunge pe *comparator_sens* alături de semnalele **N2_13** și **init** (folosit pentru resetarea componentelor). În cadrul acestei componente are loc stabilirea sensului de deplasare, prezent pe ieșirea **sens_lift**. Tot aici se hotărăște și dacă s-a ajuns sau nu la destinație prin intermediul ieșirii **egal**. Ieșirea **ctrl** are rolul de stabili dacă există sau nu cereri pe *registru_dest*, pentru a asigura rămânerea liftului la etajul curent în cazul în care acesta a onorat toate cererile.

componenta_afisare primește pe intrări semnalele **sens_lift** și **N2**, și este responsabilă de indicarea informațiilor cu privire la sensul de deplasare și a etajului curent pe 3 dintre afișoarele BCD 7 segmente prezente pe placa FPGA aleasă.

3.2 Proiectare componente

I. div_3



Divizorul de frecvență, *div_3*, este, de fapt, un numărător pe 28 de biți de la care singura ieșire utilizată este cel mai semnificativ bit, bitul 27.

Raționamentul de funcționare este următorul: în cadrul unui numărător, pe măsură ce ne îndreptăm de la cel mai nesemnificativ bit înspre cel mai semnificativ, frecvența cu care valorile sunt returnate pe ieșiri este de două ori mai mică de la un bit la succesorul său. Cu alte cuvinte, fiind dată frecvența **ν** cu care oscilează CLOCK-ul numărătorului, bitul 0, cel mai nesemnificativ, va oscila tot cu frecvența **ν**, bitul 1 cu frecvența **ν/2**, bitul 2 cu **ν/4**, și așa mai departe. Se observă că numărătorul de frecvență funcționează ca un divizor de CLOCK cu factor de divizare de 2 la fiecare bit.

Deoarece intervalul de timp la care dorim să ajungem este de 3 secunde, după cum se specifică în cerința impusă, în urma calculelor prezentat mai jos, am ajuns la concluzia că avem nevoie de un numărător pe 28 de biți pentru a ajunge la o perioadă cât mai apropiată de 3 secunde pentru CLOCK-ul rezultat:

$\nu_{cuarț}$ = 100 MHz = 10^8 Hz – frecvența oscilatorului cu cuarț de pe placa Nexys 3

ν_{div_3} – frecvența rezultată din divizorul de frecvența

$t = \frac{1}{\nu} \Rightarrow \quad \nu = \frac{1}{t} \quad \Rightarrow \quad \nu_{div\_3} = \frac{1}{3}$ Hz

n – numărul de biți ai numărătorului necesar

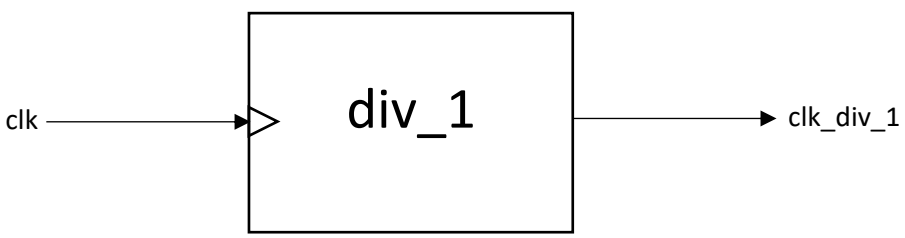
$\nu_{cuarț} \cdot 2^{-n} = \nu_{div\_3}$

$\Rightarrow 2^{-n} = \frac{\nu_{div\_3}}{\nu_{cuarț}} \quad \Rightarrow \quad -n = \log_2 \frac{\nu_{div\_3}}{\nu_{cuarț}} \quad \Rightarrow \quad n = \log_2 \frac{\nu_{cuarț}}{\nu_{div\_3}} \quad \Rightarrow \quad n = \log_2 3 \cdot 10^8$

$\Rightarrow$ n = 28.16038... $\Rightarrow$ prin rotunjire: n = 28

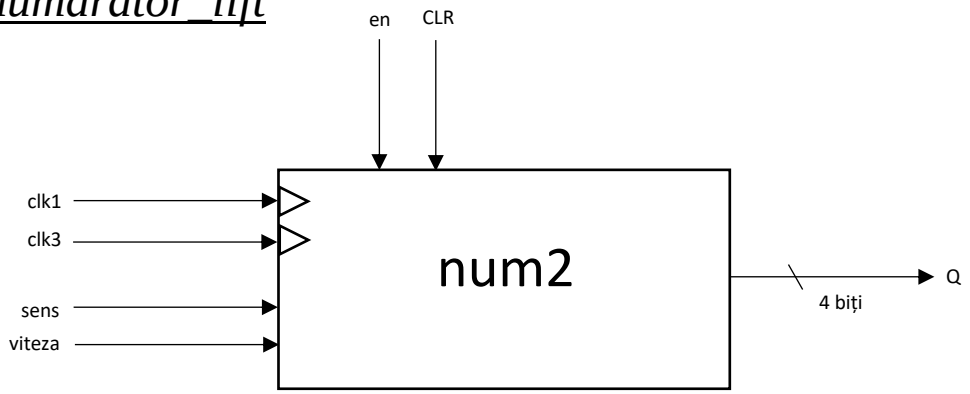
$\Rightarrow$ 28 de biți pentru numărătorul de divizează frecvența pentru a corespunde la 3 secunde

II. div_1



Mergând pe același raționament ca la divizorul de frecvență *div_3* și efectuând în mod similar calculele, dar ținând cont de faptul că v_{div_1} = 1 Hz, deoarece t = 1s, se obține n = 26, numărul de biți pentru numărătorul necesar divizării frecvenței astfel încât aceasta să corespundă timpului de 1s. Semnalul de tact rezultat la ieșirea din acest divizor de frecvență, **clk_div_1**, va funcționa ca intrare de CLOCK și pentru celelalte elemente secvențiale sincrone de circuit.

III. numarator_lift



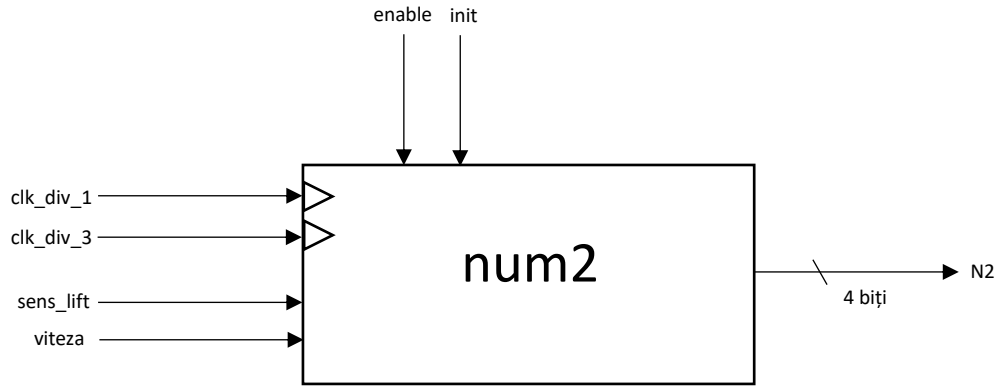
Numărătorul reversibil modulo 13 *num2* este componenta care reține etajul curent al liftului la fiecare impuls de tact.

Are două CLOCK-uri, unul corespunzător intervalului de o secundă între două etaje (**clk1**), iar celălalt corespunzător intervalului de 3 secunde (**clk3**). Alegerea dintre aceste două semnale de CLOCK se face pe baza intrării "**viteza**" a automatului: dacă **viteza** este 0 se va selecta **clk1**, iar dacă este **viteza** este 1, CLOCK-ul folosit va fi **clk3**.

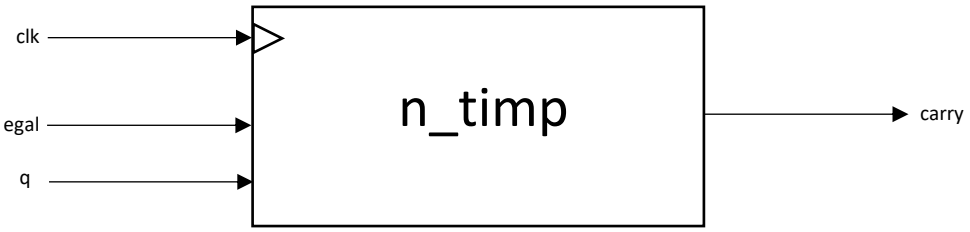
Semnalul de enable "**en**" permite funcționarea numărătorului sau forțează blocarea acestuia în ultima stare pe care avut-o, ceea ce în limbaj natural s-ar traduce astfel: dacă nu sunt îndeplinite condițiile pentru ca liftul să se deplaseze (**en**=1), atunci acesta va staționa până când toate condițiile se îndeplinesc (greutate sub limită, senzor de prezență la ușă inactiv, au trecut 10 secunde de când staționează, are cerere spre care să se deplaseze).

Numărătorul este resetat dacă tocmai s-a făcut inițializarea sistemului (**init**=1). Dacă intrarea asincronă de reset nu este activă, iar intrarea de enable este, atunci numărătorul *num2* va începe să numere pe frontul ascendent al semnalului de tact, ținând cont de sensul liftului: dacă liftul urcă (**sens_lift**=1), numărarea se va face crescător în intervalul 0 – 12; dacă liftul coboară (**sens_lift**=0), n2 va număra descrescător de între 12 și 0. Trebuie menționat faptul că *num2* nu este un numărător modulo 13 reversibil clasic: el nu trece din starea cu indice 12 în starea 0 dacă se află în bucla de numărare crescătoare, și nici nu trece din starea 0 în starea 12 atunci când numără descrescător. Asta deoarece nici liftul în mod fizic nu funcționează așa: dacă a urcat la etajul 12 fie va rămâne acolo până când va primi următoarea cerere (care va implica, în mod evident, ca liftul să coboare), fie are deja cel puțin încă o cerere pe care să o onoreze, deci va coborî după va primi din nou **enable**. Același lucru se întâmplă și la parter: liftul coboară până acolo, după care fie urcă la următoarea cerere ca prioritate, fie staționează până când va primi o nouă cerere. El nu poate, sub nicio formă, să sară peste etaje.

Schema bloc a componentei, având pe intrări semnalele pe care le primește efectiv pe intrări și ieșirea pe care o generează în circuit:



IV. n_timp



n_timp este un numărător zecimal cu ieșire de **carry**, care contorizează cele 10 secunde cât ușile liftului trebuie să rămână deschise la onorarea unei cereri. Evident, în acest interval de timp rezervat pentru ca pasagerii să iasă din lift sau să intre, este natural ca liftul să staționeze.

Fiind un numărător, deci un circuit sincron, *n_timp* primește drept semnal de CLOCK **clk_div_1** – impulsul de tact cu frecvența corespunzătoare intervalului de timp de o secundă, ceea ce înseamnă că fiecare impuls de tact pe numărător corespunde în realitate unei secunde.

Acest numărător trebuie să înceapă să numere de fiecare dată când liftul tocmai a onorat o cerere, blocând liftul la etajul respectiv pe parcursul buclei de numărare, și permițându-i din nou deplasarea după scurgerea intervalului de timp stabilit la 10 secunde. Această funcționare este posibilă prin analiza comparativă a semnalelor **egal** și **q**, unde **q** reprezintă valoarea semnalului **egal** înainte de ultimul CLOCK.

Singura ieșire a numărătorului este ieșirea de **carry**, care este folosită mai apoi pentru a participa la activarea semnalului de **enable**, ce controlează activarea numărătorului *num_2*, și, deci, deplasarea liftului. Dacă ieșirea de **carry** este 0, atunci liftul cu siguranță nu se va deplasa; dacă această ieșire este 1, atunci înseamnă că au trecut cel puțin 10 secunde de la onorarea ultimei cereri, și, deci, liftul s-ar putea deplasa din punctul de vedere al intervalului de timp cât ușile trebuie să rămână deschise. Important de menționat este că pe toată durata deplasării liftului, semnalul de **carry** trebuie să rămână 1, în caz contrar liftul fiind obligat să se oprească.

Situațiile întâlnite în cadrul funcționării lui *n_timp* sunt următoarele:

1. dacă **egal**=0 și **q**=**egal**, atunci fie ne aflăm în situația inițială (liftul nu a onorat încă nicio cerere), fie liftul se află deja în mișcare de cel puțin un clock
2. dacă **egal**=0 și **q**≠**egal**, atunci liftul tocmai a plecat, deoarece acum un clock, semnalul egal a fost 1, deci liftul staționa
3. dacă **egal** =1 și **q**=**egal**, atunci liftul staționează de cel puțin un clock, deoarece etajul curent este egal cu etajul destinație
4. dacă **egal**=1 și **q**≠**egal**, atunci liftul tocmai s-a oprit, deoarece acum un clock, semnalul egal a fost 0, deci liftul se afla în mișcare

Inițial ne aflăm în situația 1, unde *n_timp* va începe să numere 10 secunde (liftul nu poate să plece în mai puțin de 10 secunde, în orice situație de funcționare s-ar afla). După 10 secunde se generează ieșirea de **carry**, deci ipotetic liftul ar putea să plece din punctul de vedere al intervalului de timp de staționare. Dacă, totuși, celelalte condiții pentru ca liftul să plece nu sunt îndeplinite, bucla de numărare se va relua de la 0, dar ieșirea de **carry** va rămâne activă, fiindcă liftul trebuie să fie pregătit să plece în orice moment sunt îndeplinite celelalte cerințe de plecare.

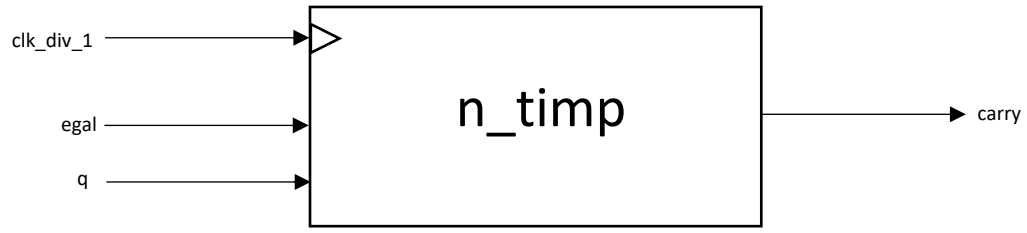
Dacă liftul se pune în mișcare, atunci pe primul interval de CLOCK egal va fi 0, iar fosta valoare a lui **egal**, reținută în **q**, va fi 0, deci situația este tot 1, iar ieșirea de **carry** se păstrează pe 1.

Chiar când liftul ajunge la destinație, **egal** va fi 1, iar **q** va fi 0, deci se va intra în situația 4, liftului i se va opri deplasarea prin transmiterea lui 0 pe semnalul de **carry**, și numărătorul va să fi resetat pentru a fi pregătit să numere în bucla 0 – 9. După un clock, **q** va deveni 1, se va trece în situația 3, iar *n_timp* va începe să numere 10 secunde.

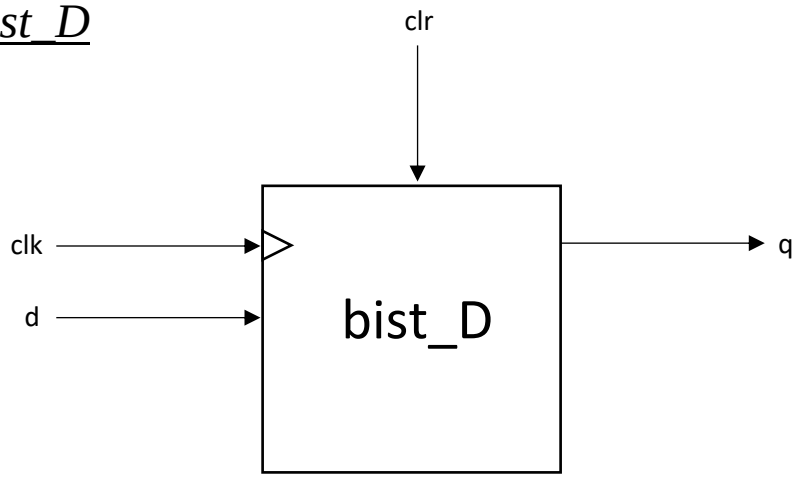
După 10 secunde se generează 1 pe ieșirea **carry**, iar dacă liftul va avea o destinație spre care să se îndrepte, el se va pune în mișcare, trecând în situația 2 de funcționare, când **carry** trebuie să se păstreze 1, iar numărătorul zecimal va număra din nou de la 0 la 9.

După încă un clock, **q** va fi si el 0, la fel ca **egal**, și se va ajunge din nou în situația de funcționare numărul 1, când *n_timp* doar numără, fără a modifica semnalul de **carry** din 1 în 0.

Schema bloc a componentei, având pe intrări semnalele pe care le primește efectiv în implementare și ieșirea pe care o generează în circuit:



V. bist_D

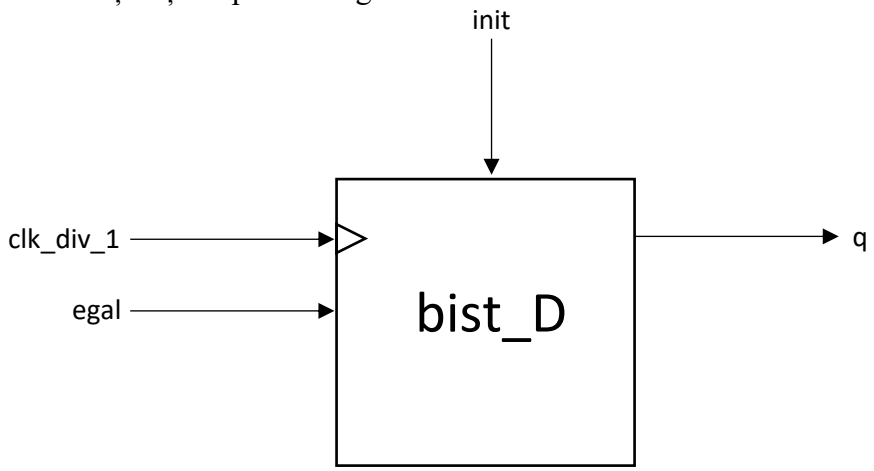


Bistabilul de întârziere *bist_D* este un bistabil clasic de tip delay, cu intrare și ieșire pe un bit, CLOCK, și intrare asincronă de reset.

Acest bistabil este folosit pentru a reține valoarea semnalului "**egal**" cu un impuls de tact în urmă. Logica este următoarea: în orice alt caz în afară de situația inițială, numărătorul *num_timp,* care numără cele 10 secunde în care liftul trebuie să staționeze la etajul la care a onorat cea mai recentă cerere, trebuie să înceapă să numere atunci când liftul a ajuns la destinația dorită (**egal**=1); acesta, însă, trebuie să pornească mereu de la 0, pentru a asigura numărarea corectă a intervalului de timp, motiv pentru care numărătorul *num_timp* trebuie să se reseteze de fiecare data când liftul tocmai s-a oprit la destinație, adică atunci când acum este la etajul destinație, dar cu un CLOCK înainte nu era – moment în care intervine analiza valorii semnalului "**egal**" înainte de ultimul tact. În situația inițială semnalul "**egal**" este nul, caz în care liftul trebuie, totuși să verifice scurgerea a 10 secunde; dar în orice alt caz în care "**egal**" are valoarea 0, liftul nu se află la etajul destinație, cu alte cuvinte, liftul se mișcă, deci numărătorul de timp trebuie să diferențieze din nou între două situații în care un semnal are aceeași valoare, așa că se trece la analiza comparativă a celei mai recente valori înainte de ultimul CLOCK, la fel ca în cazul în care "**egal**" este 1.

Alegerea variantei finale de implementare folosind bistabilul delay *bist_D* a fost determinată de dorința de a elimina o eroare apărută în ultimul stadiu al implementării: chiar dacă în proiectarea VHDL informațiile oferite de acest bistabil erau la fel obținute și prin prin simpla folosire a atributului 'event pentru "**egal**" în arhitectura numărătorului *num_timp,* la sinteza în ISE se genera o eroare la numărător, soluționată prin introducerea circuitului logic secvențial de față.

Schema bloc a componentei, având pe intrări semnalele pe care le primește efectiv în implementare și ieșirea pe care o generează în circuit:



Tabelul de excitație al bistabilului D:

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Tabelul de adevăr al bistabilului D:

D	Q
0	0
1	1

VI. si_4

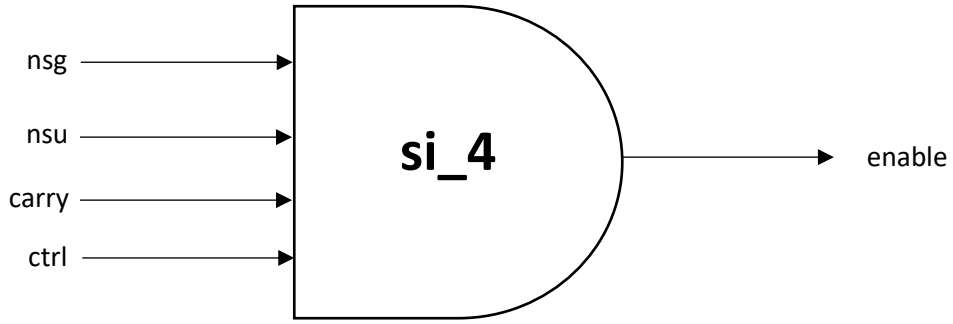
Poarta ȘI cu 4 intrări pe un bit este folosită pentru a controla intrarea de **enable** a numărătorului n2, cel ce reprezintă, în fapt, liftul.

Liftul trebuie să funcționeze dacă limita de greutate nu este depășită (**nsg**=1), dacă senzorul ușilor nu este activ, adică dacă nu există nicio persoană și niciun obiect care să blocheze închiderea ușilor (**nsu**=1), dacă au trecut deja cel puțin 10 secunde de la onorarea ultimei cereri a liftului (**carry**=1) – cu mențiunea că în cazul în care cele 10 secunde s-au scurs, dar liftul nu are nicio cerere încă, acesta va pleca imediat după primirea primei cereri – și, după cum am anticipat, dacă liftul are cel puțin o cerere spre care să se deplaseze (**ctrl**=1, unde **ctrl** este ieșirea comparatorului *comp_13* care este 0 dacă *registru_13_dest* este nul, și 1 în rest).

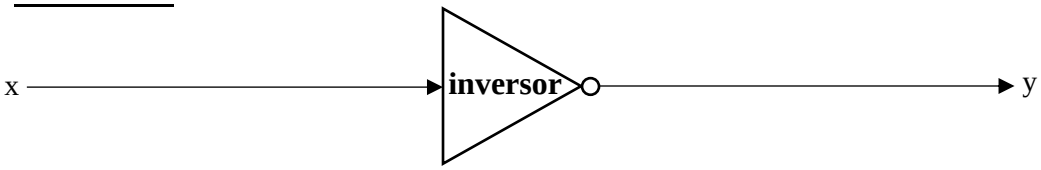
Funcția lui si_4:

carry = nsg **AND** nsu **AND** carry **AND** ctrl

Schema bloc a componentei, având pe intrări semnalele pe care le primește efectiv în implementare și ieșirea pe care o generează în circuit:



VII. inversor



Poarta NU pe un bit realizează funcția de negare a informației primite pe intrare. Astfel, aceasta scoate pe ieșire valoarea negată de pe intrare. Inversorul este folosit de 3 ori în implementare, întrucât este necesară folosirea și a valorilor negate ale semnalelor **sens_cerere**, **sg** (senzor de greutate, are valoarea '1' când greutatea maximă admisă a fost depășită și '0' în caz contrar) și **su** (senzor de prezență, egal cu '1' când o persoană sau un obiect se află în ușa liftului împiedicând închiderea acesteia și '0' în caz contrar).

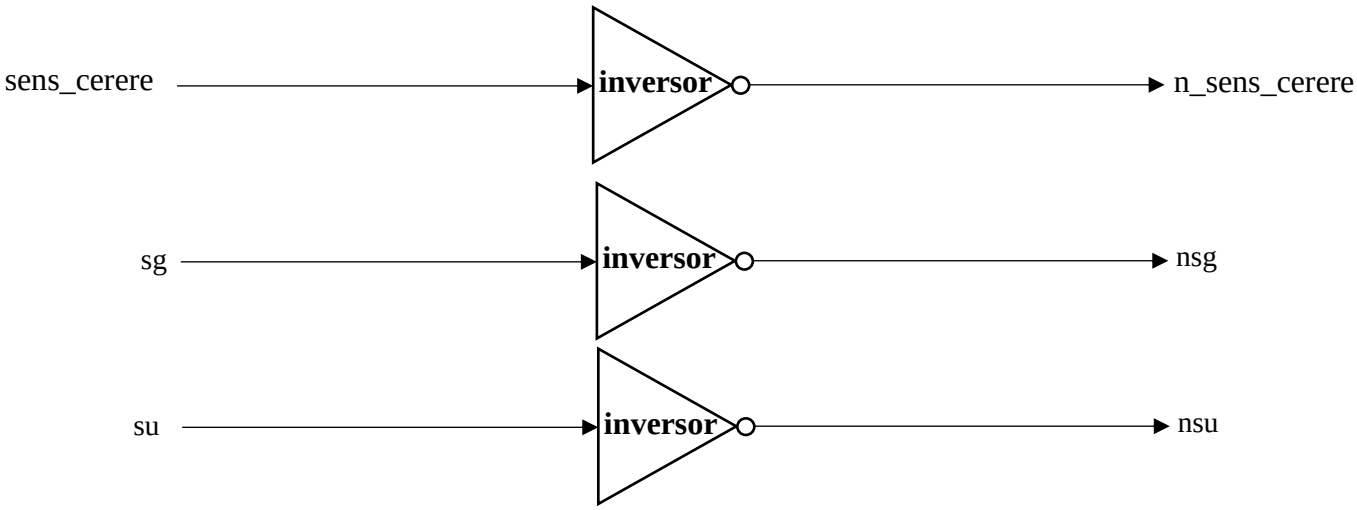
Ecuațiile porților NU pe 1 bit folosite sunt:

n_sens_cerere = **NOT** sens_cerere

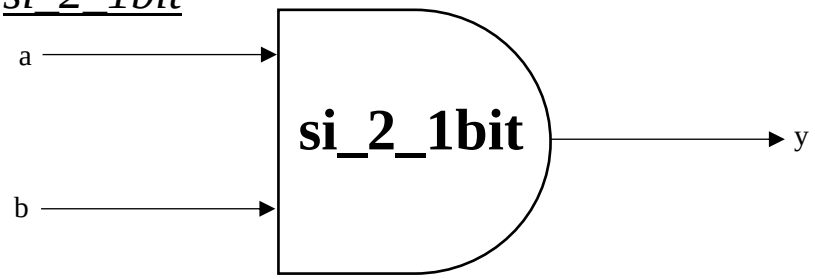
nsg = **NOT** sg

nsu = **NOT** su

Schemele componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:



VIII. si_2_1bit



Poarta ȘI cu două intrări pe 1 bit, *si_2_1bit*, se folosește pentru a obține semnalele de CLEAR ale numărătoarelor *numărător_sus* și *numărător_jos*, dar și semnalele de ENABLE ale decodificatoarelor *decodificator_ns* și *decodificator_nj*.

Prima poartă de acest tip este folosită pentru obținerea semnalului **ok_sus**, care semnalizează momentul finalizării introducerii unei cereri de urcare din exteriorul liftului. Ea primește pe intrări **sens_cerere** (semnal pe un bit, care are valoarea '0' atunci când cererea introdusă este de urcare și '1' când aceasta este de coborâre) și **ok_ext** (semnal egal cu '1' în momentul în care utilizatorul, aflat în exteriorul liftului, termină de introdus o cerere). Așadar, ieșirea acesteia, **ok_sus**, va avea valoarea '1' doar atunci când s-a finalizat introducerea unei cereri de urcare din exteriorul liftului. Ajuns pe numărătorul *numărător_sus*, acest semnal va funcționa pe post de semnal de CLEAR, resetând numărătorul în momentul în care informația cu privire la cererea introdusă a fost transmisă mai departe, spre decodificatorul *decodificator_ns,* și numărătorul este pregătit să preia o nouă cerere de urcare din exteriorul liftului. Pentru *decodificator_ns,* semnalul **ok_sus** are rol de ENABLE, lăsând astfel informația să treacă mai departe doar în cazul în care aceasta reprezintă codificarea unei cereri de urcare introdusă de un utilizator aflat în exteriorul liftului.

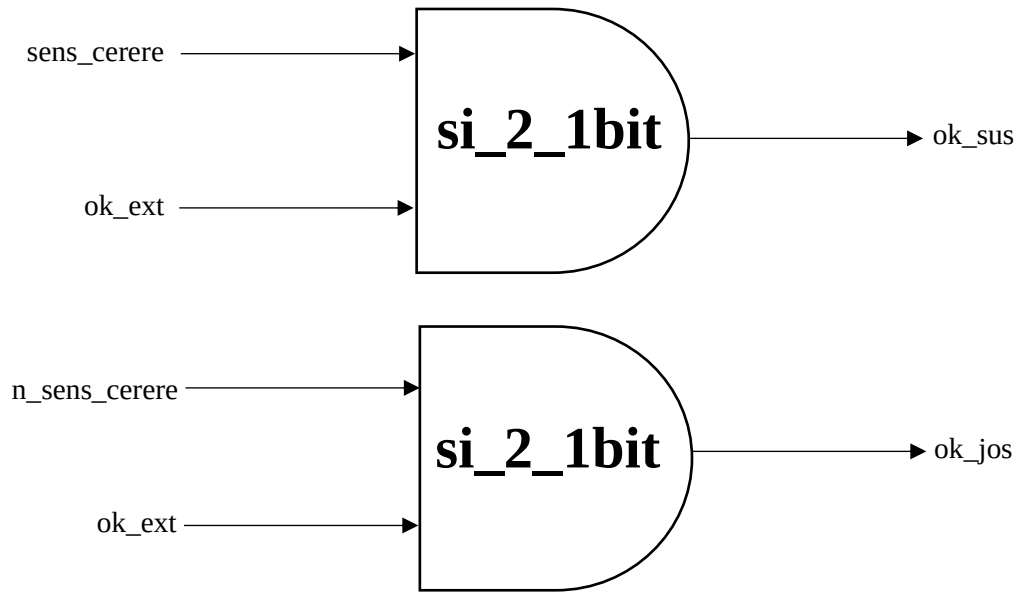
A doua poartă de tip *si_2_1bit* funcționează în mod similar cu prima poartă, descrisă mai sus, aceasta primind pe intrări semnalele **n_sens_cerere** (valoarea negată a semnalului **sens_cerere**) și **ok_ext,** ieșirea sa, **ok_jos**, fiind '1' doar când utilizatorul termină de introdus o cerere de coborâre din exteriorul liftului. Acest semnal, joacă rol de **CLEAR** pentru numărătorul *numărător_jos* și de **ENABLE** pentru decodificatorul *decodificator_nj*, pentru o funcționare similar cu cea descrisă mai sus.

Ecuațiile porților *si_2_1bit* folosite:

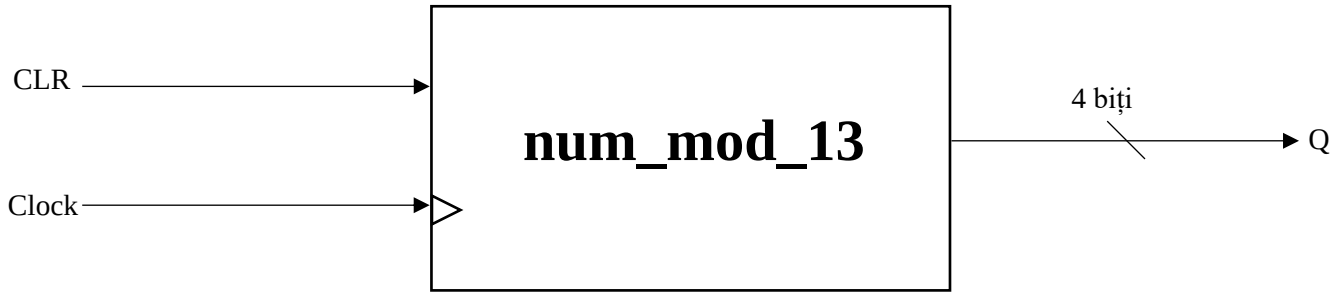
ok_sus = sens_cerere **AND** ok_ext

ok_jos = n_sens_cerere **AND** ok_ext

Schemele componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:



IX. num_mod_13



Numărătorul modulo 13 direct are două moduri de funcționare: resetare și numărare directă în bucla 0-12, funcționarea acestuia putând fi descrisă prin următorul tabel:

CLR	**Operație**	**Descriere**
0	Numărare directă în bucla 0-12	Q = (Q+1) mod 13
1	Resetare	Q = ”0000”

În implementarea automatului au fost necesare 3 numărătoare din acest tip, și anume *numarator_cerere_interior, numarator_sus* și *numarator_jos*.

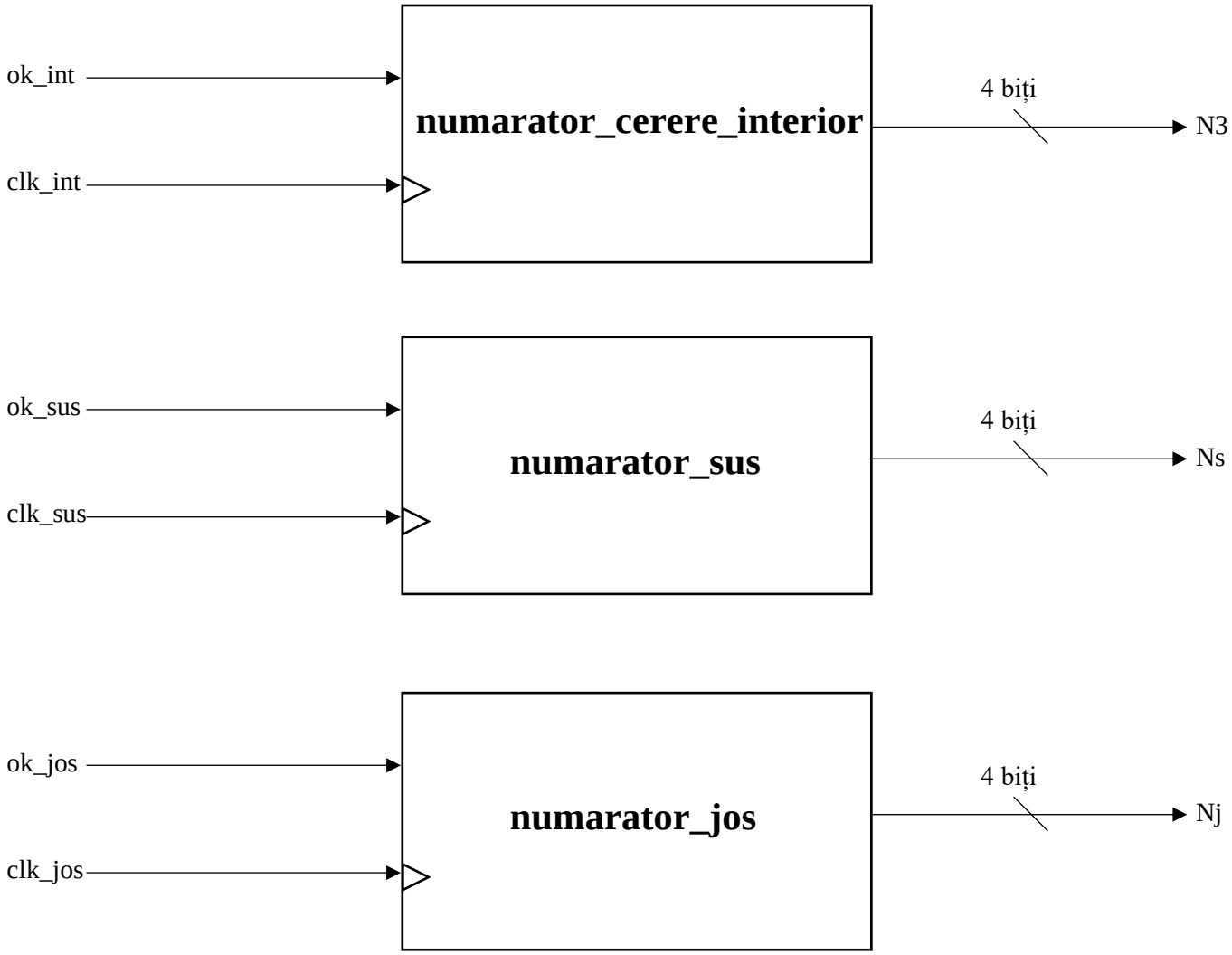
Intrarea de resetare CLR se consideră ca fiind prioritară în fața intrării sincrone de tact. Semnalul de tact este generat prin apăsarea unui buton de pe placă, fiecare dintre cele trei numărătoare folosite în implementare având rezervat câte un buton. Când regimul de funcționare este cel de numărare, numărătorul trece în starea următoare pe frontul ascendent al semnalului de **Clock**.

Numărătorul *numarator_cerere_interior* este folosit pentru introducerea cererilor de către utilizatorii din interiorul liftului, acesta numărând numărul de impulsuri de tact generate prin apăsarea repetată a butonului corespunzător numărătorului. Intrarea de **CLR** a acestui numărător este reprezentată de semnalul **ok_int**, care va fi generat prin apăsarea altui buton rezervat și care indică momentul finalizării introducerii unei cereri din interiorul liftului. Spre exemplu, dacă un utilizator dorește să introducă o cerere din interiorul liftului spre a se deplasa la etajul 8, acesta va apăsa de 8 ori pe butonul corespunzător intrării de **Clock** a numărătorului *numarator_cerere_interior* de 8 ori, apoi va apăsa butonul corespunzător semnalului **ok_int**. În acest moment cererea este transmisă mai departe spre a fi realizată lista priorităților, iar numărătorul este resetat, revenind în starea ”0000”. Excepția de la acest caz o constituie parterul, pentru a introduce o cerere de coborâre spre parter utilizatorul fiind nevoit să apese de 13 ori butonul corespunzător numărătorului și apoi butonul de **ok_int**.

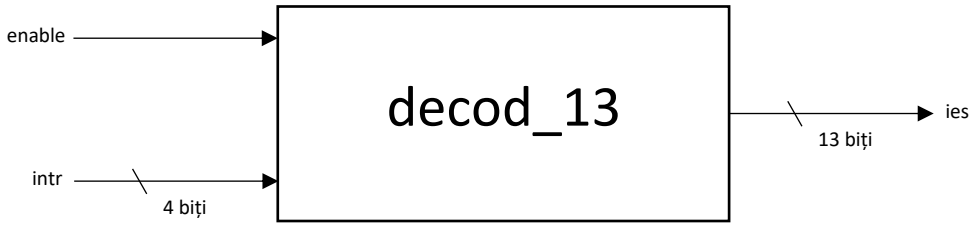
În mod similar funcționează și numărătoarele *numarator_sus* și *numarator_jos*, cu deosebirea că acestea sunt utilizate pentru introducerea cererilor de urcare, respectiv coborâre din exteriorul liftului. Semnalul de **CLR** al numărătorului *numarator_sus* este **ok_sus**, iar cel al numărătorului *numarator_jos* este ok_jos. Pentru a introduce o cerere dine exterior, utilizatorul setează switch-ul corespunzător semnalului **sens_cerere** (1 pentru urcare, 0 pentru coborâre), apasă pe butonul aferent numărătorului *numarator_sus* pentru urcare, *numarator_jos* pentru

coborâre de un număr de ori egal cu etajul de la care se face cererea și apoi apasă butonul corespunzător semnalului **ok_ext**, urmând ca acesta, împreună cu **sens_cerere** să genereze semnalele **ok_sus** și **ok_jos**, în modul specificat în descrierea porții *si_2_1bit*, cu ajutorul căreia acestea sunt obținute.

Schemele bloc ale componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:



X. decod_13



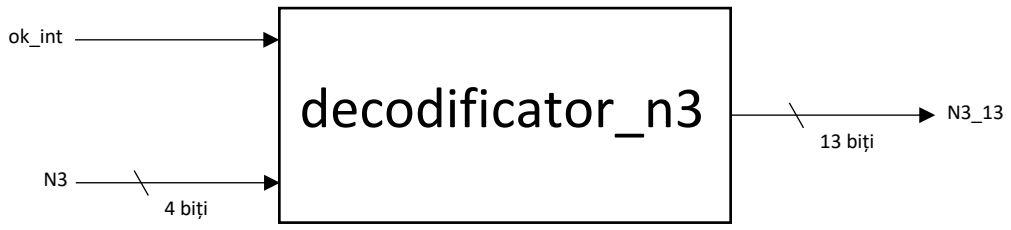
Decodificatorul 4:13 realizează decodificarea informației de pe intrări, reprezentată în binar pe 4 biți, după cum arată tabelul de mai jos:

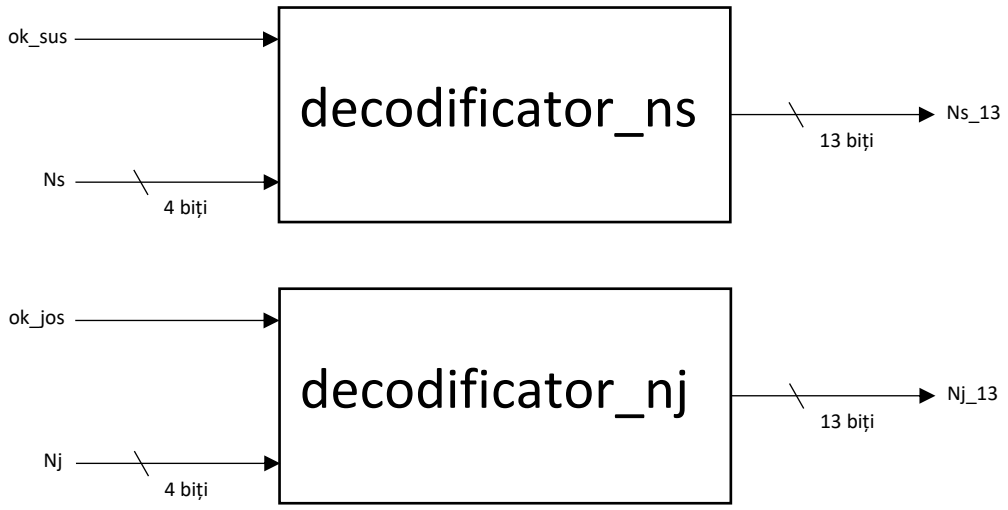
en	**intr**	**ies**
0	X	0000000000000
1	0000	0000000000001
1	0001	0000000000010
1	0010	0000000000100
1	0011	0000000001000
1	0100	0000000010000
1	0101	0000000100000
1	0110	0000001000000
1	0111	0000010000000
1	1000	0000100000000
1	1001	0001000000000
1	1010	0010000000000
1	1011	0100000000000
1	1100	1000000000000
1	1101	0000000000000
1	1110	0000000000000
1	1111	0000000000000

Semnalul **en** activ pe 1 logic are rol de **ENABLE**. Dacă acesta are valoarea 0 logic, reacționează identic pentru orice informație primită pe intrări, ieșirea fiind întotdeauna 0000000000000. Dacă **en** are valoarea 1 logic, și intrarea este mai mică sau egală decât 1100, atunci componenta decodifică intrarea **intr** (4 biți) din binar, punând pe ieșirea **ies** (13 biți) un 1 logic pe poziția corespunzătoare numărului binar primit ca intrare și 0 pe toți ceilalți biți. În cazul în care pe intrări ar ajunge una dintre combinațiile 1101, 1110 sau 1111, ieșirea ar fi constituită doar din biți de 0.

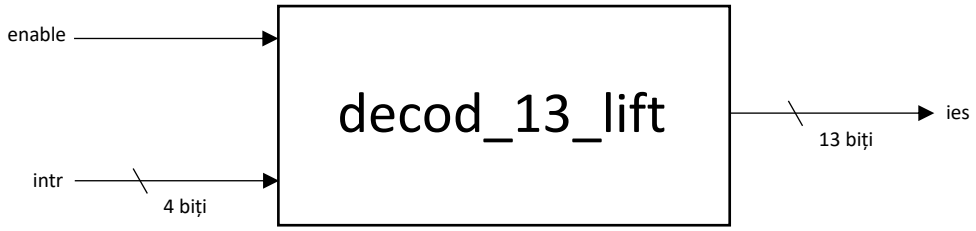
Pentru implementarea acestui automat se vor folosi 3 decodificatoare de tipul ***decod_13***: ***decodificator_n3***, ***decodificator_ns***, ***decodificator_nj***, care primesc pe intrări informația de la numărătoarele ***numarator_cerere_interior***, ***numarator_sus***, respectiv ***numarator_jos*** și o decodifică conform tabelului de mai sus. Intrarea de **en** pentru decodificatorul ***decodificator_n3*** este reprezentată de semnalul **ok_int**, în timp ce ***decodificator_ns*** are drept **ENABLE** semnalul **ok_sus**, iar ***decodifcator_nj*** semnalul **ok_jos**, generate conform descrierii prezentate la ***si_2_1bit***.

Schemele bloc ale componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:





XI. decod_13_lift



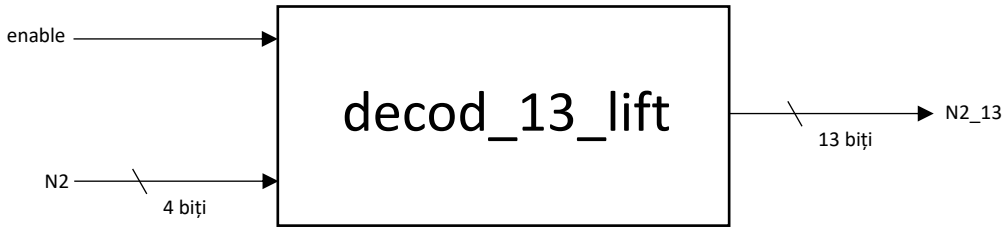
Componenta *decod_13_lift* este un circuit logic combinațional de decodificare a informației binare pe 4 biți primite de la numărătorul *num2*, care dă etajul curent al liftului. Acest decodificator preia codul în BCD al etajului la care se află liftul, și activează doar una dintre cele 13 ieșiri ale sale, și anume cea corespunzătoare codificării în binar. De exemplu, dacă etajul la care se află liftul este etajul 9, acest lucru se regăsește pe ieșirile numărătorului *num2* prin combinația binară "1001", ceea ce înseamnă că pe decodificator trebuie să se activeze ieșirea cu indicele 9, în timp ce toate celelalte ieșiri vor trebui să ia valoarea 0, adică pe ieșire cei 13 biți vor avea configurația "0001000000000".

Ținând cont de faptul că numărătorul 2 este un numărător modulo 13 se poate desprinde concluzia că stările codificate în cod BCD cu "1101", "1110" și "1111" nu vor ajunge niciodată pe intrările lui *decod_13_lift*, așa că strict pentru completitudinea codului am considerat că în aceste cazuri ieșirea pe 13 biți va fi nulă, aidcă "0000000000000".

Decodificatorul *decod_13_lift* trebuie să poată să își schimbe valoarea în orice moment în care numărătorul ce dă etajul curent al liftului poate să se modifice pe ieșiri, ceea ce înseamnă că oricând liftul se deplasează (**enable**=1 pentru lift), *decod_13_lift* trebuie să poată prelua informația de pe *num2* și să o transmită mai departe pe căile de date. Dacă în schimb liftul staționează, pe ieșirea decodificatorului nu trebuie să intervină modificări, ci trebuie să se păstreze vechea valoare, deci intrarea **enable** trebuie să fie 0. În cazul în care această intrare s-ar elimina din structura decodificatorului, el ar trebui să realizeze decodificarea în orice moment, ceea ce ar presupune

efectuarea unor operații suplimentare fără vreo finalitate, fapt care nu ar face altceva decât să reducă performanța automatului.

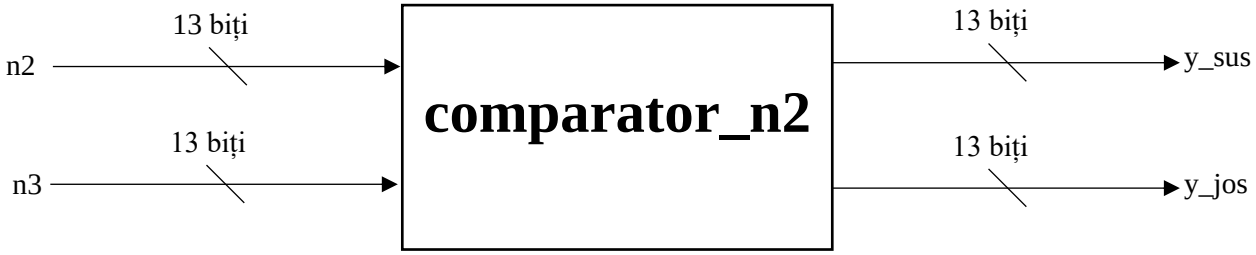
Schema bloc a componentei, având pe intrări semnalele pe care le primește efectiv în implementare și ieșirea pe care le generează în circuit:



Tabel de adevăr pentru decod_13_lift:

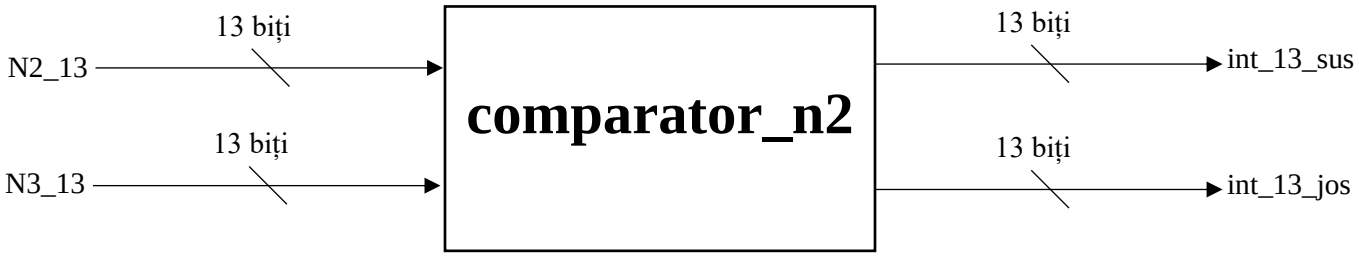
intr (3)	intr (2)	intr (1)	intr (0)	ies(12)	ies(11)	ies(10)	ies (9)	ies (8)	ies (7)	ies (6)	ies (5)	ies (4)	ies (3)	ies (2)	ies (1)	ies (0)
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

XII. comparator_n2



Comparatorul pe 13 biți, *comparator_n2*, este un circuit logic combinațional care primește pe intrarea **n3** informația de la numărătorul *numarator_cerere_interior* și pe intrarea **n2** informația de la *numarator_lift* decodificată pe 13 biți și are rolul de a determina poziția etajului pentru care a fost introdusă o cerere din interiorul liftului (**N3_13**, ieșirea decodificatorului *decodificator_n3*), relativă la etajul curent la care se află liftul în acest moment (**N2_13,** ieșirea decodificatorului *decod_13_lift*). Acesta are, așadar, scopul de a determina dacă cererea introdusă de un utilizator din interiorul liftului este una de urcare sau de coborâre, relativ la poziția curentă a liftului, pentru ca aceasta sa poată fi plasată în registrul corespunzător ce reține cererile de același tip (*registru_sus* pentru cererile de urcare și *registru_jos* pentru cele de coborâre). Astfel, dacă informația de pe intrarea **n3** reprezintă un număr mai mic decât cea de pe intrarea **n2**, adică dacă etajul spre care se dorește deplasarea liftului este mai mic decât etajul curent, cererea va fi una de coborâre și, prin urmare, aceasta va fi regăsită pe ieșirea **y_jos** pentru a putea fi încărcată în registrul *registru_jos,* iar ieșirea **y_sus** va primi valoarea ”0000000000000” pentru a nu influența conținutul registrului *registru_sus,* care reține cererile de urcare, funcția SUS de încărcare a acestui registru fiind astfel realizată încât valoarea ”0000000000000” pe semnalul **int_13_sus** (rezultat din ieșirea **y_sus** a comparatorului *comparator_n2*) să nu influențeze cererile de urcare deja existente. În mod similar, când informația de pe intrarea **n3** este un număr mai mare decât cea de pe intrarea **n2** a comparatorului, cererea este una de urcare, deci ea se va regăsi pe ieșirea **y_sus**, fiind încărcată în *registru_sus* prin intermediul funcției SUS, în timp ce ieșirea **y_jos** va avea valoarea ”0000000000000”, din aceleași considerente: semnalul **int_13_jos** (rezultat din **y_jos**), având valoarea ”0000000000000”, nu va altera conținutul registrului ce conține cererile de coborâre.

Schema bloc a componentei, având pe intrări semnalele pe care le primește efectiv în implementare și ieșirea pe care le generează în circuit:



XIII. Funcțiile SUS și JOS

Alegerea de a reține cererile de urcare și de coborâre ale liftului în registre pe 13 biți, în care fiecare bit să reprezinte un etaj (parter – bitul 0, etajul 1 – bitul 1, etajul 2 – bitul 2, etc.), implică automat faptul că pentru a actualiza conținutul registrelor la onorarea unei cereri, este necesar să se țină cont și de celelalte valori din registre, deoarece nu se poate lucra individual pe biți pentru încărcare în registru; toți biții trebuie să se încarce simultan, însă dacă o cerere a existat într-un registru și tocmai a fost onorată, ea trebuie să fie ștearsă din registrul corespunzător, totuși fără a afecta conținutul registrului pe orice altă poziție în afară de cea a etajului la care s-a soluționat cererea.

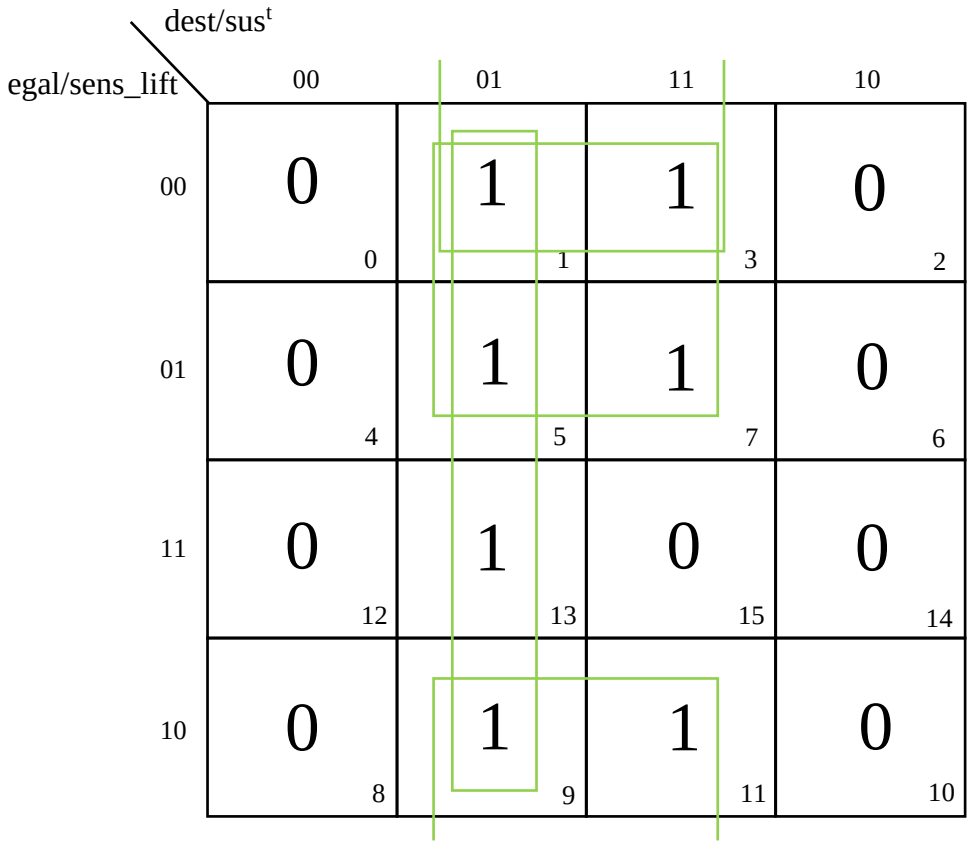
Analizând acest aspect, am ajuns la concluzia că este nevoie de elemente suplimentare de logică pentru a realiza funcționarea corespunzătoare a registrelor *registru_sus* și *registru_jos,* fiind necesară analiza pe tabele de adevăr a comportării dorite de noi în cazul fiecăruia dintre cele două registre.

În cele ce urmează vom descrie raționamentul abordat pentru obținerea funcției ce actualizează conținutul registrului *registru_sus* la onorarea unei cereri de urcare, denumită de noi funcția SUS, modul de gândire pentru funcția JOS – corespunzătoare registrului *registru_jos* fiind absolut similar, analiza pentru tabelul de adevăr și mai apoi concluziile în urma realizării diagramelor Karnaugh făcându-se exact în același mod.

Primele patru coloane din tabelul de adevăr reprezintă variabilele de care depinde funcția ce urmează a fi determinată: **egal**, **sens_lift**, **dest** și fost valoare din *registru_sus*. Important de menționat este că tabelul de adevăr este întocmit pe un bit, deoarece perechile omoloage de biți din registrele sus și dest se vor analiza individual, în corelație cu **egal** și **sens_lift** după cum urmează:

- Dacă **egal**=0, atunci înseamnă că nu s-a ajuns încă la destinație, deci conținutul din *registru_sus* nu trebuie actualizat pentru a elimina ceva, ci trebuie să se păstreze pe *registru_sus* exact valorile precedente, fără a șterge vreun semnal de 1 (cu posibilitatea de a mai încărca, eventual, alte noi valori de 1, prin intermediul porții sau cu 3 intrări ce determină valoarea finală încărcată pe registru, prin însumarea booleană a valorilor venite de la codificatoarele numărătoarelor *numarator_int* și *numarator_sus*, cu valoarea rezultată în urma efectuării funcției SUS);
- Dacă **egal**=1 și **sens_lift**=0, atunci, din nou, conținutul nu se va actualiza în modul mai sus amintit, deoarece chiar dacă s-a ajuns la o destinație, aceasta este corespunzătoare lui **sens_lift**=0, adică sensului de coborâre; la fel ca în cazul precedent, valorile din *registru_sus* nu suferă modificări din cauza lui **dest**;
- Dacă **egal**=1 și **sens_lift**=1, atunci:
    - dacă **dest** =0, atunci înseamnă că etajul corespunzător bitului respectiv nu a fost etaj destinație, deci valoarea de pe bitul de același rang din *registru_sus* nu se va schimba în momentul imediat următor;
    - dacă **dest**=1 și $\mathbf{sus^t}$=0 (valoarea de pe bitul analizat din *registru_sus* la momentul de timp t), atunci înseamnă că etajul curent a fost destinație, dar din celălalt sens de deplasare a liftului, deci va fi șters din *registru_jos*, fără a avea vreo influență asupra lui *registru_sus*;
    - dacă **dest**=1 și $\mathbf{sus^t}$=1, atunci destinația la care tocmai s-a ajuns a fost primită de pe *registru_sus*, deci conținutul registrului *registru_sus* trebuie actualizat prin "încărcarea" lui 0, deci $\mathbf{sus^{t+1}}$=0;

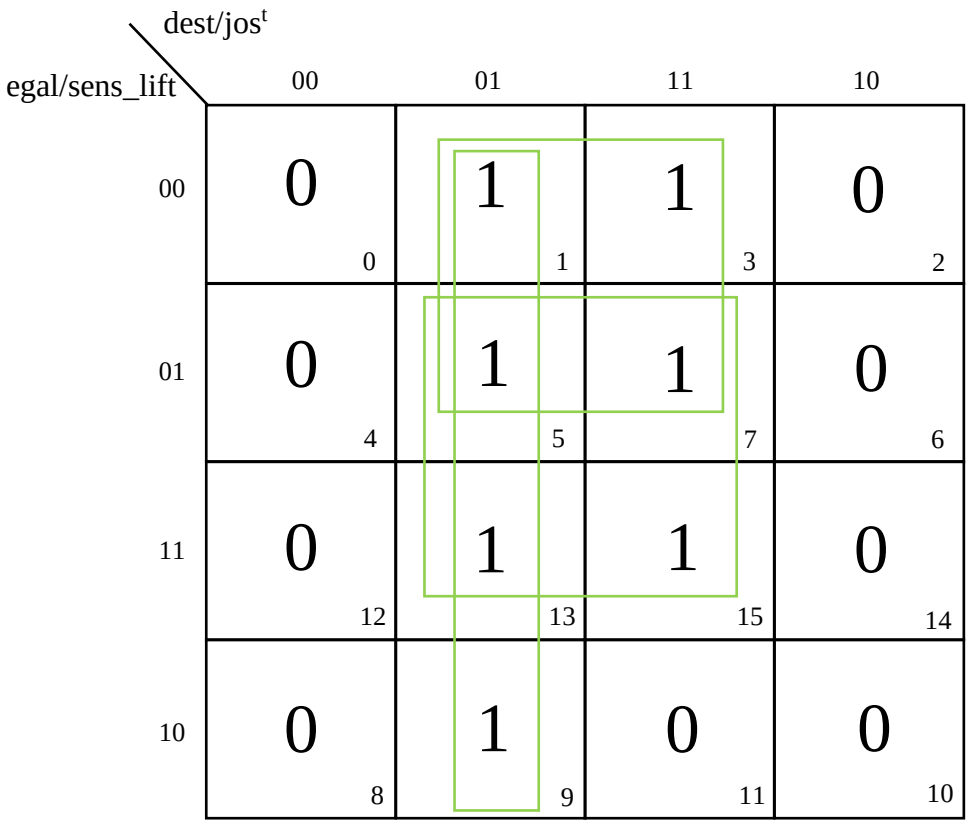
egal	sens_lift	dest	sus^t	sus^{t+1}
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0



$$sus^{t+1} = \overline{dest} \cdot sus^t + \overline{egal} \cdot sus^t + \overline{sens\_lift} \cdot sus^t$$

$$\Rightarrow sus^{t+1} = sus^t \cdot (\overline{dest} + \overline{egal} + \overline{sens\_lift})$$

egal	sens_lift	dest	jos^t	jos^{t+1}
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1



$$jos^{t+1} = \overline{dest} \cdot jos^t + \overline{egal} \cdot jos^t + sens\_lift \cdot sus^t$$

$$\Rightarrow jos^{t+1} = jos^t \cdot (\overline{dest} + \overline{egal} + sens\_lift)$$

XIV. sau_3_13

Poarta SAU cu trei intrări pe 13 biți, sau_3_13biti, se folosește pentru în implementarea funcțiilor SUS și JOS, care controlează funcționarea registrelor *registru_sus* și *registru_jos*, după cum reiese din tabelul de adevăr al acestor funcții și în urma simplificărilor făcute prin diagrame Karnaugh.

Această poartă, folosită de două ori în cadrul implementării, realizează funcția de SAU-LOGIC între biții de pe pozițiile corespunzătoare ale intrărilor, ieșirea fiind, de asemenea, reprezentată pe 13 biți.

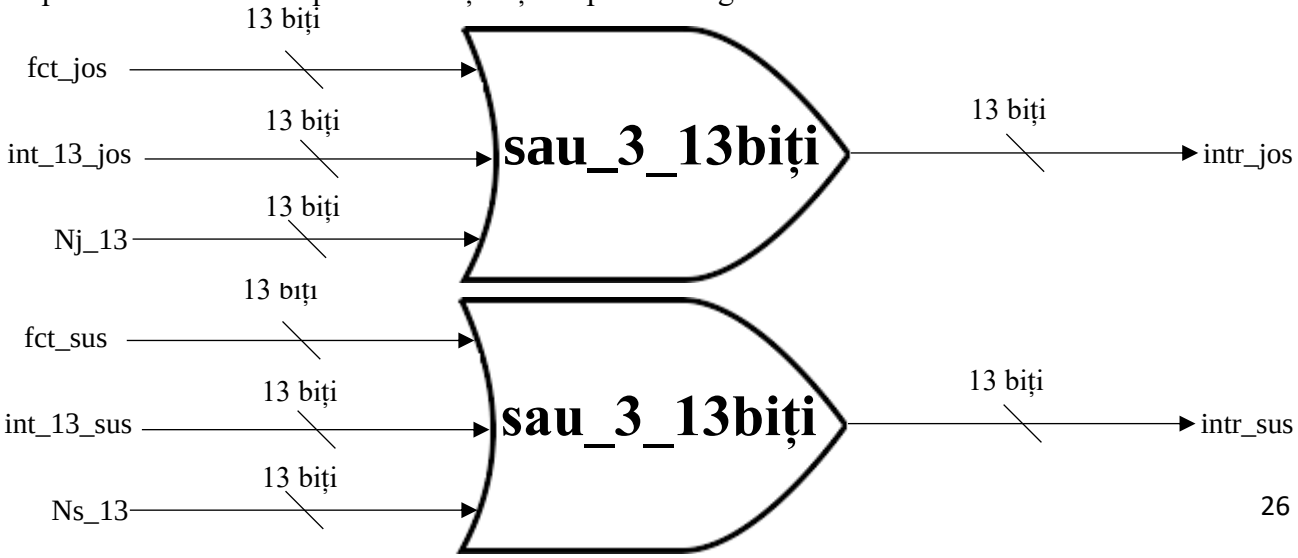
Alegerea acestui tip de poartă și funcționarea explicită va fi detaliată și justificată în cadrul descrierilor funcțiilor SUS și JOS.

Ecuațiile celor două porți *sau_3_13biti* folosite:

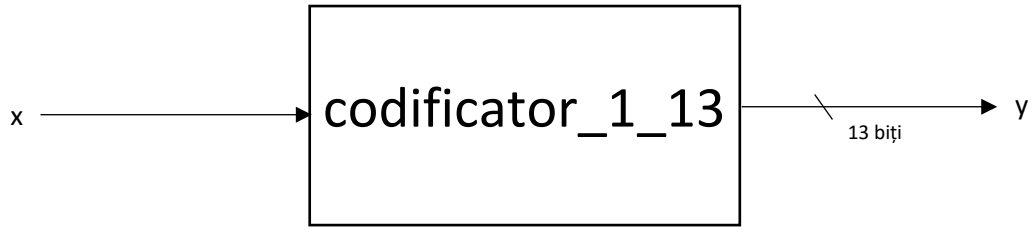
intr_sus = fct_sus **OR** int_13_sus **OR** Ns_13

intr_jos = fct_jos **OR** int_13_jos **OR** Nj_13

Schemele componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:

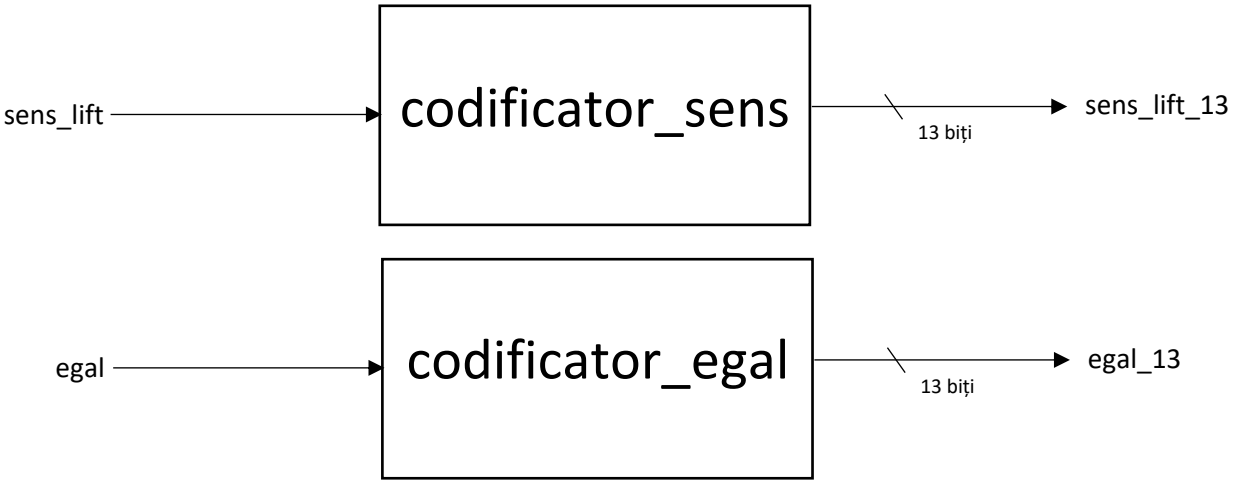


XV. codificator_1_13



Circuitul logic combinațional *codificator_1_13* este folosit în realizarea funcțiilor SUS și JOS, care au ca scop actualizarea constantă a registrelor ce reține cererile de urcare, respectiv cele de coborâre. Funcțiile operează pe 13 biți, iar cum semnalele "**sens_lift**" și "**egal**" se regăsesc drept termeni ai celor două funcții, se impunea posibilitatea de translatare a fiecăruia dintre acestea de pe un bit pe 13 biți, de unde și necesitatea codificatorului de față.

Schemele componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:



XVI. nu_13

Poarta NU pe 13 biți, *nu_13*, își justifică existența prin faptul că în cadrul funcțiilor SUS și JOS rezultate în urma minimizărilor Karnaugh este nevoie de termenul "**dest**", ieșirea registrului destinație, pe 13 biți, care să fie negat. Această poartă este, de asemenea, folosită și pentru obținerea lui NOT"**sens_lift**", respectiv a lui NOT"**egal**", pentru implementarea funcției SUS.

Cu toate că pentru obținerea ultimelor 2 valori se putea folosi mai întâi un inversor pe un bit și abia apoi un codificator pe 13 biți, am optat pentru această variantă pentru a sugera apartenența la aceleași funcții cu NOT"**dest**", acesta din urmă fiind exprimabil doar cu o poartă nu pe 13 biți.

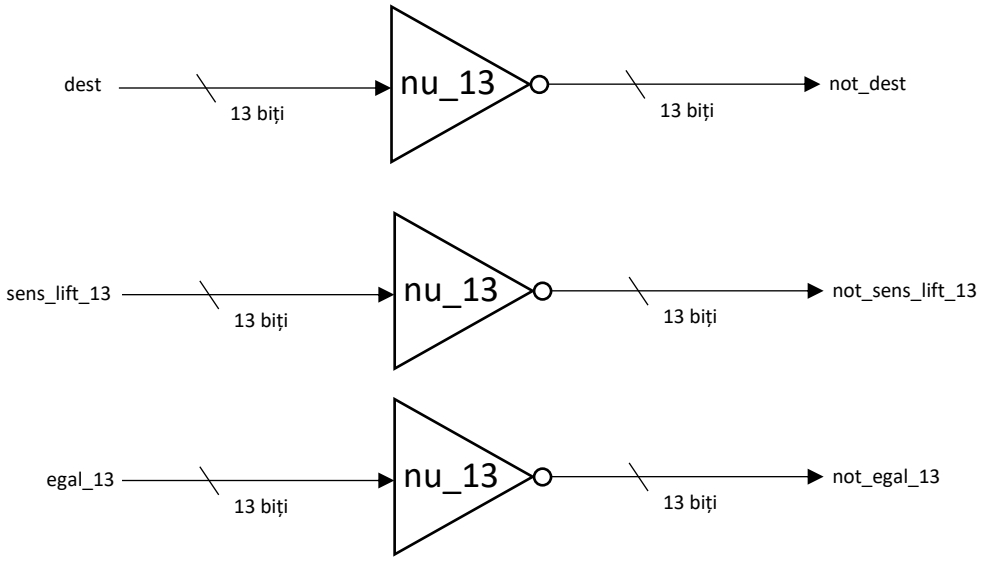
Funcții *nu_13*:

not_dest = **NOT** dest

not_sens_lift_13 = **NOT** sens_lift_13

not_egal_13 = **NOT** egal_13

Schemele componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:



XVII. si_2_13

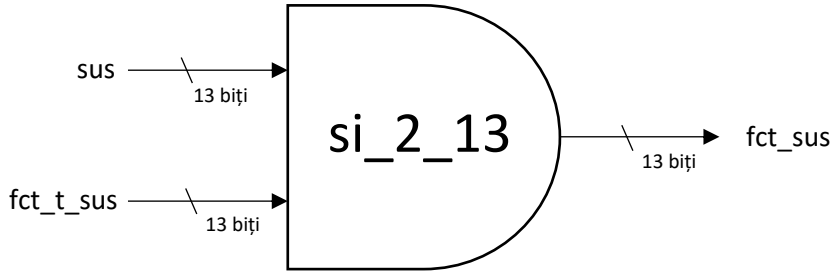
Poarta ȘI cu două intrări pe 13 biți, *si_2_13*, este folosită pentru a realiza funcțiile SUS și JOS, care controlează funcționarea dorită a registrelor sus și jos, după cum reiese din tabelul de adevăr și în urma simplificărilor făcute prin diagrame Karnaugh.

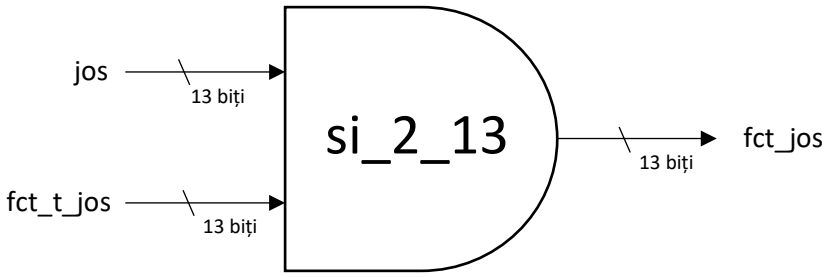
Acest tip de poartă este folosit de două ori în implementare, o dată pentru fiecare dintre cele două funcții mai sus amintite, având următoarele funcții:

fct_sus = sus **AND** fct_t_sus

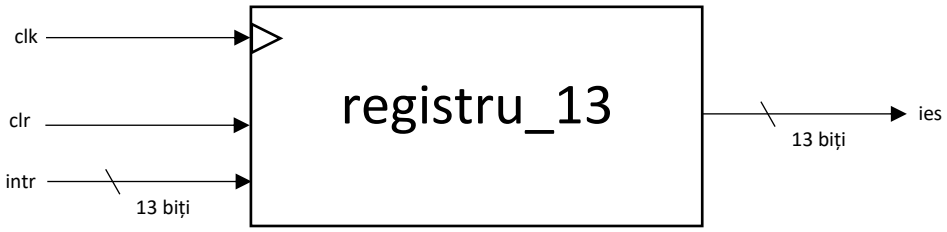
fct_jos = jos **AND** fct_t_jos

Schemele componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:





XVIII. registru_13



Registrul sincron pe 13 biți, *registru_13*, are două moduri de funcționare: resetare și încărcare paralelă, controlabile prin intrarea asincronă de control clr, astfel:

clr = 1 – resetare

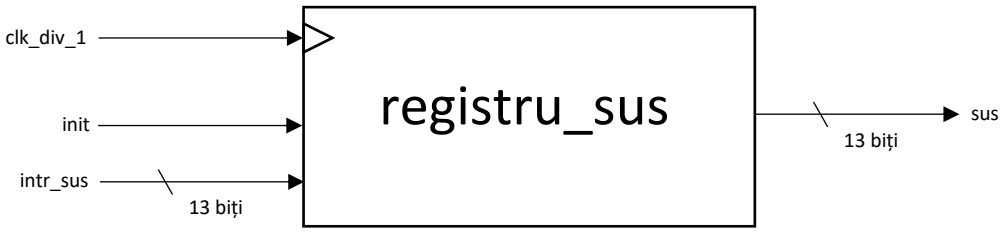
clr = 0 – încărare paralelă

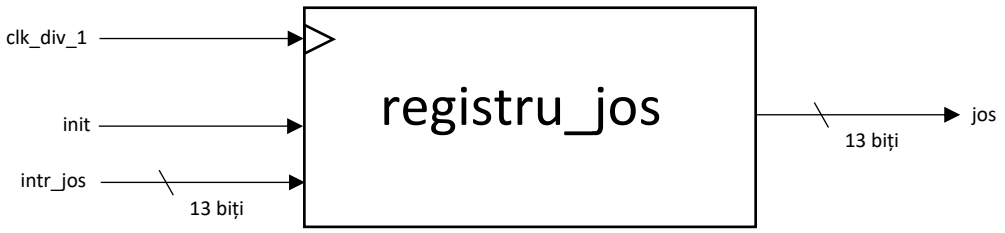
Acest registru este folosit de două ori în implementare: pentru registrul ce reține cererile de urcare, respectiv pentru cel ce reține cererile de coborâre. De menționat este faptul că aceste registre se vor reseta doar la semnalul de "**init**", care inițializează automatul. Eliminarea cererilor onorate din aceste registre se va face în modul de încărcare paralelă cu ajutorul funcțiilor SUS și JOS, pentru ca toate cererile din registru care nu au fost încă onorate să rămână înregistrate, lucru care nu ar fi fost posibil dacă s-ar fi optat pentru resetare, deoarece registrul este resetat în întregimea sa, fără a exista varianta de a elimina câte un singur bit.

În *registru_sus*, informațiile trebuie să fie preluate atât din codificatorul lui *numarator_int* (pentru a acoperi cazul în care se face o cerere de urcare din interiorul liftului), din codificatorul lui *numarator_sus* (pentru a înregistra cererile de urcare din exteriorul liftului), și informația anterioară din *registru_sus*, prelucrată pe baza funcției SUS. Același raționament se aplică și pentru *registru_jos*.

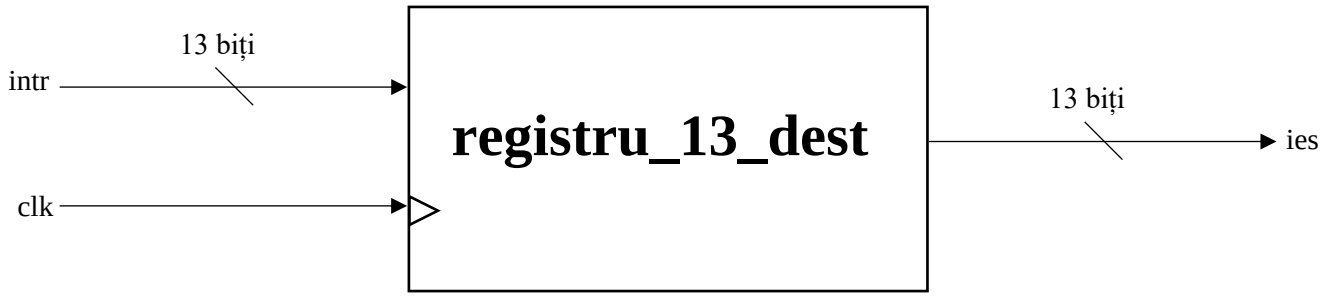
Semnalul de CLOCK al ambelor registre de tip *registru_13* este tactul divizat ce rezultă în urma trecerii oscilatorului cu cuarț de pe placă printr-un divizor de frecvență (**div_1**) ca să se asigure funcționarea corespunzătoare a componentelor, întrucât frecvența de 100 MHz de pe placă este prea mare.

Schemele componentelor de acest tip utilizate, având pe intrări semnalele pe care le primesc efectiv în implementare și ieșirile pe care le generează în circuit:





XIX. registru_13_dest

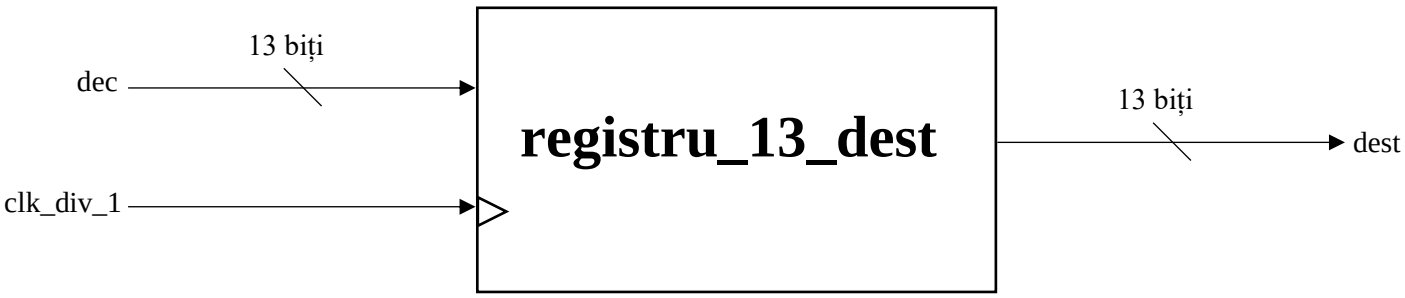


Componenta *registru_13_dest* este un registru pe 13 biți care realizează funcția de încărcare paralelă a datelor primite pe intrări, acesta fiind singurul mod de funcționare al registrului. În registrul *registru_13_dest* se reține etajul spre care se deplasează liftul în fiecare moment de timp, caz în care pe ieșirile acestuia se va afla 1 pe poziția corespunzătoare etajului destinație și 0 în rest, sau numai valori de 0 pe cei 13 biți de la ieșire, atunci când liftul a onorat deja toate cererile. Atât intrarea, **dec**, cât și ieșirea acestui registru, **dest**, sunt reprezentate pe 13 biți.

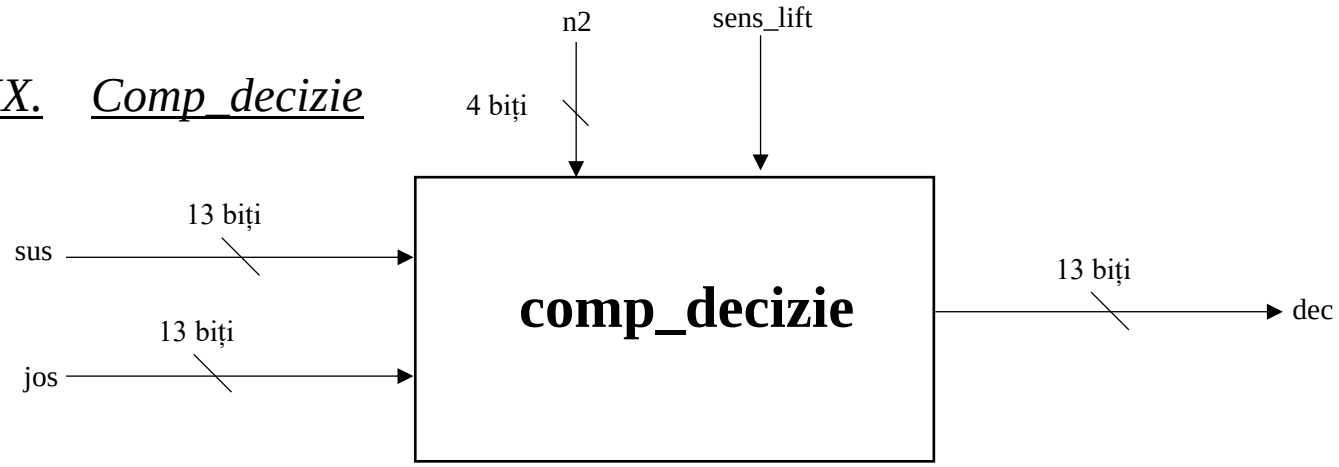
Funcționarea registrului este simplă, acesta încărcând paralel informația primită pe intrări la fiecare impuls de ceas. Operația se realizează pe frontul crescător al semnalului de tact. Semnalul de CLOCK pe care registrul îl primește reprezintă un semnal de tact divizat, **clk_div_1**, rezultat în urma trecerii impulsurilor generate de oscilatorul de cuarț de pe placă prin divizorul de frecvență *div_1*, întrucât frecvența acestui oscilator este prea mare pentru a asigura funcționarea corespunzătoare a componentelor automatului.

Registrul nu necesită intrări de comandă, acesta trebuind în orice moment de timp să încarce paralel pe ieșiri informația primită pe intrări. De actualizarea etajului destinație se va ocupa componenta *comp_decizie*, a cărei ieșire, **dec**, va fi dusă pe intrările registrului *registru_13_dest*.

Schema componentei, având pe intrări semnalele pe care le primește efectiv în implementare și ieșirea pe care o generează în circuit:



XX. Comp_decizie



Comparatorul *comp_decizie* vine drept răspuns pentru necesitatea de a stabili prioritățile cererilor introduse și de a alege dintre acestea următoarea destinație a liftului. În mod general, mecanismul său de funcționare poate fi descris astfel: cunoscând etajul curent și cererile deja introduse și încărcate în registrele *registru_sus* și *registru_jos*, comparatorul stabilește care dintre acestea se află cel mai aproape de etajul curent conform sensului de mers. În cazul în care o astfel de cerere nu există, se va determina cererea cea mai apropiată de etajul curent, dar corespunzătoare sensului contrar celui de deplasare al liftului.

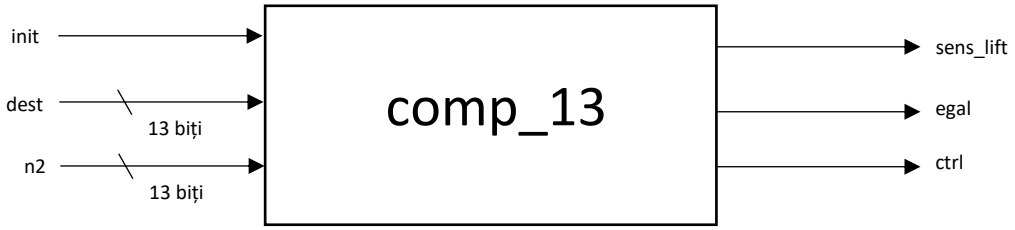
Așadar, comparatorul primește pe intrarea **n2**, informația de la ieșirea numărătorului *numarator_lift*, reprezentând etajul curent, iar pe intrarea **sens_lift** sensul de deplasare. Intrările sus și jos sunt reprezentate de informațiile din registrele *registru_sus* și *registru_jos*, care contorizează cererile de urcare, respectiv coborâre. Dacă sensul de deplasare are valoarea 1 logic (liftul urcă), atunci se verifică în primul rând dacă există cereri de urcare de la un etaj superior etajului curent dintre care se va alege cea care se află la etajul cel mai apropiat de etajul curent (practic, se alege cererea reprezentată prin bitul de 1 aflat în *registru_sus* pe cea mai mică poziție strict mai mare decât poziția etajului curent), ieșirea **dec** va avea valoarea 1 pe poziția corespunzătoare acestei cereri și 0 pe toate celelate 12 poziții. Dacă nu există nicio astfel de cerere, se va căuta în *registru_jos* cererea de coborâre de la un etaj superior etajului curent, dintre care se va regăsi pe ieșire cea de la etajul cel mai înalt. În cazul în care nu poate fi găsită nicio astfel de cerere, se trece la următorul caz: căutarea cererii de urcare de la cel mai mic etaj aflat la o poziție inferioară poziției etajului curent. Dacă nici o astfel de cerere nu a putut fi găsită, se trece la un ultim nivel al priorității de căutare: se încearcă găsirea cererii de coborâre de la cel mai înalt etaj, a cărui poziție este inferioară poziției etajului curent. În final, inexistența unei cereri din acest tip confirmă faptul că liftul a onorat deja toate cererile introduse anterior și, prin urmare, acesta va rămâne la etajul curent până la apariția unei noi cereri, indiferent dacă aceasta a fost introdusă din interiorul sau din exteriorul liftului. În mod similar se va proceda și dacă sensul liftului este 0 (liftul coboară), cu deosebirea că prioritatea de căutare este următoarea: cereri de coborâre aflate la nivele inferioare nivelului curent (se alege cea de la nivelul cel mai înalt, mai mic decât etajul curent), cereri de urcare aflate la nivele superioare (prioritară fiind cea de la cel mai mic etaj aflat sub etajul curent), cereri de coborâre aflate deasupra etajului curent (se alege cea de la cel mai mare etaj superiori etajul curent) și cereri de urcare aflate deasupra etajului curent (se alege cea de la cel mai mic etaj superior etajului curent).

Ieșirea acestuia, **dec**, va ajunge pe *registru_dest* unde se va reține următoarea oprire a liftului, în ordinea priorității cererilor.

Tabelul ce descrie, în mod formal, funcționarea comparatorului *comp_decizie*:

sens_lift	**Operațiile de căutare, în ordinea descrescătoare a priorităților**
1	sus-deasupra, jos-deasupra, sus-dedesubt, jos-dedesubt
0	jos-dedesubt, sus-dedesubt, jos-deasupra, sus-deasupra

XXI. comp_13



Circuitul *comp_13* este un comparator pe 13 biți, care primind pe intrări decodificarea pe 13 biți a etajului curent al liftului și ieșirea tot pe 13 biți a registrului *registru_dest* (ce reține destinația imediat următoare), furnizează pe ieșiri sensul de deplasare al liftului ("**sens_lift**"), precum și dacă s-a ajuns sau nu la destinație ("**egal**").

sens_lift=1 corespunde sensului de urcare, în timp ce **sens_lift**=0 se traduce drept sens de coborâre; **egal**=1 transmite informația că s-a ajuns la etajul destinație, în timp ce **egal**=0 indică faptul că încă nu s-a onorat cerea imediat următoare ca prioritate.

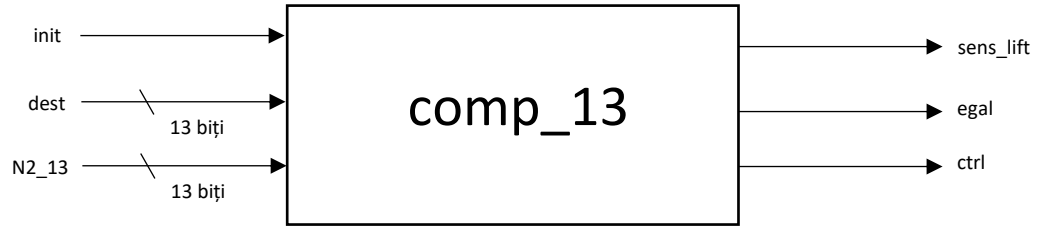
Totuși, în situația inițială, atunci când prin acționarea semnalului "**init**" (**init**=1) se transmite automatului informația că liftul trebuie să se afle la parter, pregătit pentru a înregistra prima cerere, care, în mod evident, va fi de urcare, se stabilește prin convenție că "**egal**" este 0 (nu s-a onorat nicio cerere până în momentul de față, chiar dacă liftul staționează), iar "**sens_lift**" este 1 (fie că prima cerere va fi din interiorul liftului, în cazul în care cineva se va intra în lift de la parter, fie că liftul va fi solicitat din exterior de la un anumit etaj, el nu mai are unde să coboare, ci va putea doar să urce).

În realizarea comparațiilor dintre **N2_13** și **dest** se desprinde următorul caz special, care era imperativ să fie soluționat: în cazul în care liftul a onorat toate cererile avute, registrele *registru_sus* și *registru_jos* ajung să fie nule, determinând ca și *registru_dest* să aibă pe ieșiri combinația "0000000000000". La orice etaj s-ar afla în acest moment liftul, în cazul în care s-ar fi efectuat o comparație între **dest** și **N2_13**, ramura logică pe care ar fi ajuns execuția codului ar fi fost cazul **dest**<**n2**, ceea ce ar fi determinat ca "**sens_lift**" să fie 0, iar "**egal**" să fie, de asemenea, 0. Altfel spus, liftul ar fi înțeles că trebuie să coboare, fiindcă are o cerere neonorată. El ar fi coborât la parter de la orice etaj ar fi fost mai înainte, în loc să rămână la etajul de la care a avut ultima cerere, așa cum ar fi de dorit.

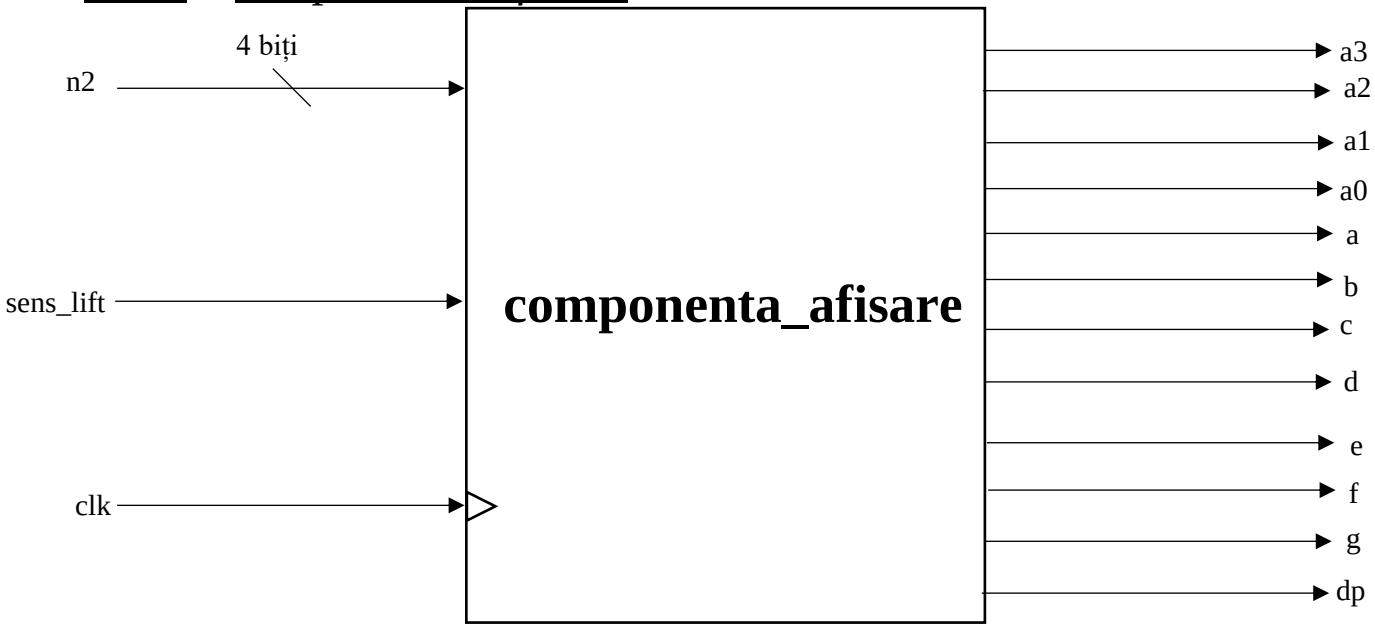
Pentru a evita comparația dintre etajul curent și destinație atunci când, în fapt, nu mai există nicio nouă destinație, s-ar realizat mai întâi comparația dintre **dest** și "0000000000000". În caz de

egalitate devine nulă ieșirea pe un bit "**ctrl**"; dacă **dest** are cel puțin un bit de 1, deci dacă **dest** indică, într-adevăr, o destinație validă, "**ctrl**" va fi 1. Semanlul tocmai menționat, "**ctrl**", participă la luarea deciziei ca liftul să staționeze sau să se deplaseze, fiind unul dintre cele patru semnale care intră în poarta *si_4* pentru a genera semnalul de **enable** al numărătorului ce dă poziția liftului. În mod natural, dacă liftul nu are nicio destinație spre care să se deplaseze, numărătorului corespunzător lui i se va sista posibilitatea de a număra.

Schema componentei, având pe intrări semnalele pe care le primește efectiv în implementare și ieșirea pe care o generează în circuit:



XXII. componenta_afisare



Această componentă își justifică existența prin necesitatea controlării celor 4 afișoare de tipul 7 segmente pentru a indica la fiecare moment de timp sensul de deplasare al liftului și etajul curent la care acesta se află. Ținând cont de faptul că toate cele 4 afișoare au în comun cei 8 catozi (CA, CB, CD, CE, CF, CG, DP) și sunt controlate fiecare de către un anod separat (AN3, AN2, AN1, AN0) și de faptul că pentru a face ca un LED să se aprindă, trebuie ca atât semnalul care controlează segmentul individual (CA pentru segmentul a, CB pentru segmentul b, ș.a.m.d.), cât și semnalul de control al anodului să aibă valoarea '0' logic, mecanismul de funcționare este următorul: se setează catozii corespunzători segmentelor ce alcătuiesc cifra sau simbolul dorit pe '0' logic, apoi se setează anodul corespunzător pe '0' logic, urmând ca acesta să fie dezactivat, să fie setați catozii corespunzători următorului simbol ce se dorește a fi afișat pe valoarea '0' logic și

să se activeze anodul corespunzător. Acest procedeu se repetă pentru fiecare afișor în fiecare perioadă de ceas pentru a obține fiecare simbol individual. Datorită persistenței privirii, creierul uman percepe toate cele 4 simboluri ca apărând simultan, desi, fizic, mecanismul de afișare este altul.

Pentru afișarea informațiilor despre sensul liftului și etajul curent sunt necesare 3 afișoare de tip 7 segmente. Astfel, pentru a afișa sensul de deplasare se va apinde segmental a (pentru urcare) sau d (pentru coborâre) al unuia dintre afișoare, celelalte două fiind utilzate pentru a indica cifra zecilor și respectiv cifra unităților etajului la care se află liftul.

Luând în considerare faptul că frecvența oscilatorului de cuarț de pe placă este prea mare, ochiul uman neputând sesiza modificările apărute pe afișoare, *componenta_afisare* are o intrare de tact, **clk**, pe care primește semnalul **clk_div_1**, rezultat prin divizarea clock-ului de pe placă cu ajutorul divizorului ***div_1***, astfel încât informația de pe afișoare să fie actualizată în fiecare perioadă de tact, fiind sesizabilă pentru utilizator. De asemenea, această componentă va primi informația de la ieșirea numărătorul *numarator_lift* pe intrarea de date **n2**, reprezentând etajul curent al liftului, iar pe intrarea **sens_lift** va ajunge ieșirea **sens_lift** a componentei *comparator_13*. La fiecare impuls de tact ajuns pe intrarea **clk**, cicuitul va seta ieșirile a, b, c, d, e, f, g, dp la valorile corespunzătoare pentru a afișa informația primită pe intrarea de **sens_lift**, va seta pe 0 ieșirea **a2**, corespunzătoare anodului ce controlează afișorul aferent sensului, apoi o va dezactiva, actualizând ieșirile cu informația referitoare la cifra zecilor etajului curent, va activa ieșirea **a1**, urmând să procedeze în același mod și ieșirea **a0**, rezervată controlării afișorului pentru cifra unităților. Ieșirile de control, a, b, c, d, e, f, g, dp, ale segmentelor individuale vor fi '1' logic la momentul activării ieșirii **a3**, corespunzând celui de-al treilea afișor, care trebuie să fie dezactivat pe tot parcursul funcționării automatului.

Tabelul de funcționare al afișorului controlat de ieșirea **a0** (valori posbile pentru afișorul ce indică cifra unităților etajului curent):

sens_lift	**n2**	**a**	**b**	**c**	**d**	**e**	**f**	**g**	**dp**
X	0000	0	0	1	1	0	0	0	1
X	0001	1	0	0	1	1	1	1	1
X	0010	0	0	1	0	0	1	0	1
X	0011	0	0	0	0	1	1	0	1
X	0100	1	0	0	1	1	0	0	1
X	0101	0	1	0	0	1	0	0	1
X	0110	0	1	0	0	0	0	0	1
X	0111	0	0	0	1	1	1	1	1
X	1000	0	0	0	0	0	0	0	1
X	1001	0	0	0	0	1	0	0	1
X	1010	0	0	0	0	0	0	1	1
X	1011	1	0	0	1	1	1	1	1
X	1100	0	0	1	0	0	1	0	1
X	1101	1	1	1	1	1	1	1	1
X	1110	1	1	1	1	1	1	1	1
X	1111	1	1	1	1	1	1	1	1

Observație:

În cazul în care liftul se află la parter, nu se va afișa cifra 0, ci litera P.

Tabelul de funcționare al afișorului controlat de ieșirea **a1** (valori posbile pentru afișorul ce indică cifra zecilor etajului curent):

sens_lift	n2	a	b	c	d	e	f	g	dp
X	0000	1	1	1	1	1	1	1	1
X	0001	1	1	1	1	1	1	1	1
X	0010	1	1	1	1	1	1	1	1
X	0011	1	1	1	1	1	1	1	1
X	0100	1	1	1	1	1	1	1	1
X	0101	1	1	1	1	1	1	1	1
X	0110	1	1	1	1	1	1	1	1
X	0111	1	1	1	1	1	1	1	1
X	1000	1	1	1	1	1	1	1	1
X	1001	1	1	1	1	1	1	1	1
X	1010	1	0	0	1	1	1	1	1
X	1011	1	0	0	1	1	1	1	1
X	1100	1	0	0	1	1	1	1	1
X	1101	1	1	1	1	1	1	1	1
X	1110	1	1	1	1	1	1	1	1
X	1111	1	1	1	1	1	1	1	1

Observație:

Ieșirile de control ale segmentelor individuale vor fi ‘1’ logic dacă pe intrarea n2 a ajuns un număr mai mic decât 1010, întrucât etajul curent nu are decât o cifră.

Tabelul de funcționare al afișorului controlat de ieșirea **a2** (valori posbile pentru afișorul ce sensul de deplasare):

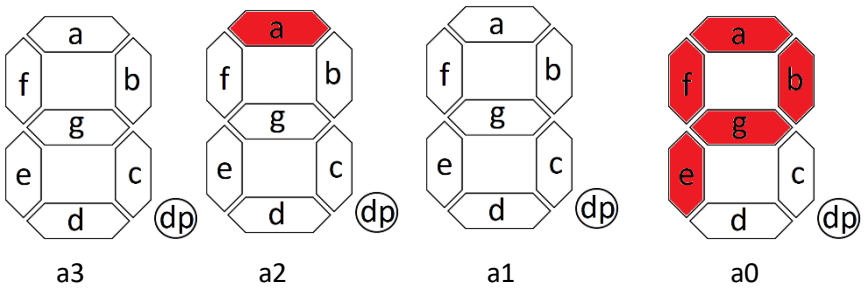
sens_lift	n2	a	b	c	d	e	f	g	dp
0	X	0	1	1	1	1	1	1	1
1	X	1	1	1	0	1	1	1	1

Tabelul de funcționare al afișorului controlat de ieșirea **a2** (valori posbile pentru afișorul ce sensul de deplasare):

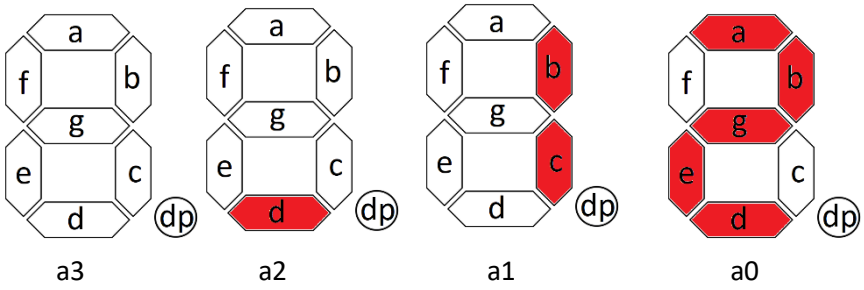
sens_lift	n2	a	b	c	d	e	f	g	dp
X	X	1	1	1	1	1	1	1	1

Exemple de afișare:

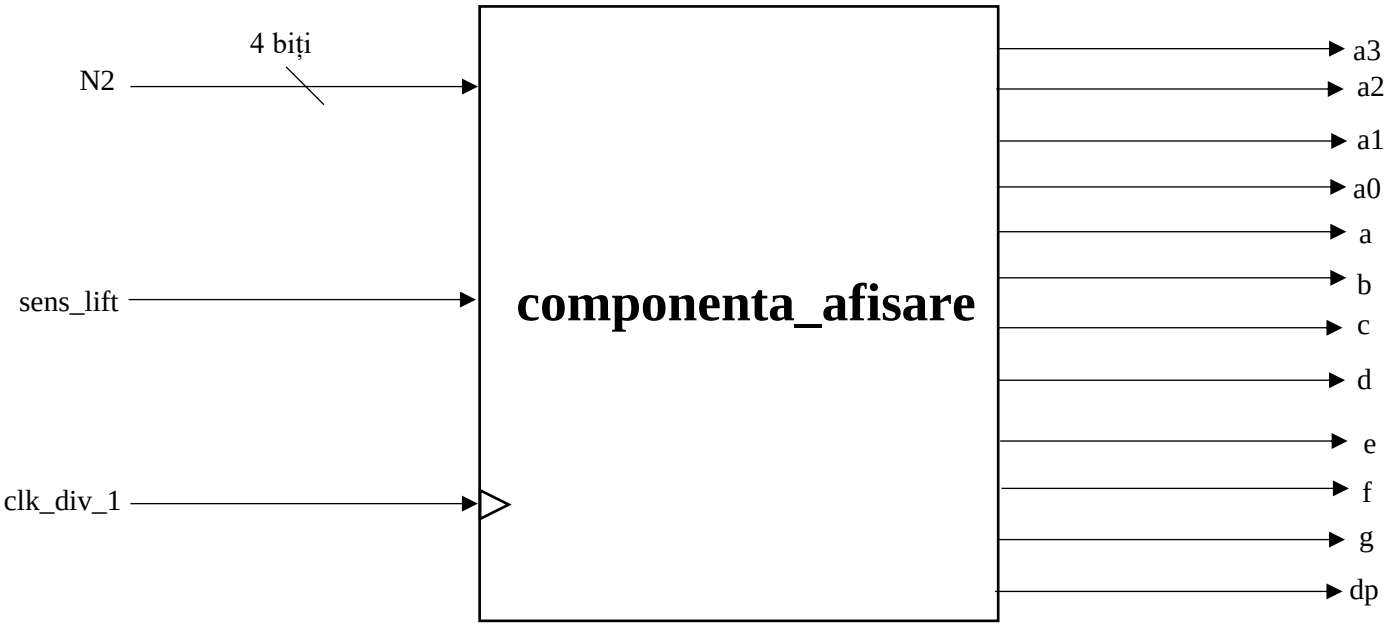
- Liftul se află la parter, iar sensul de deplasare a acestuia este de urcare



- Liftul se află la etajul 12, având sensul de coborâre



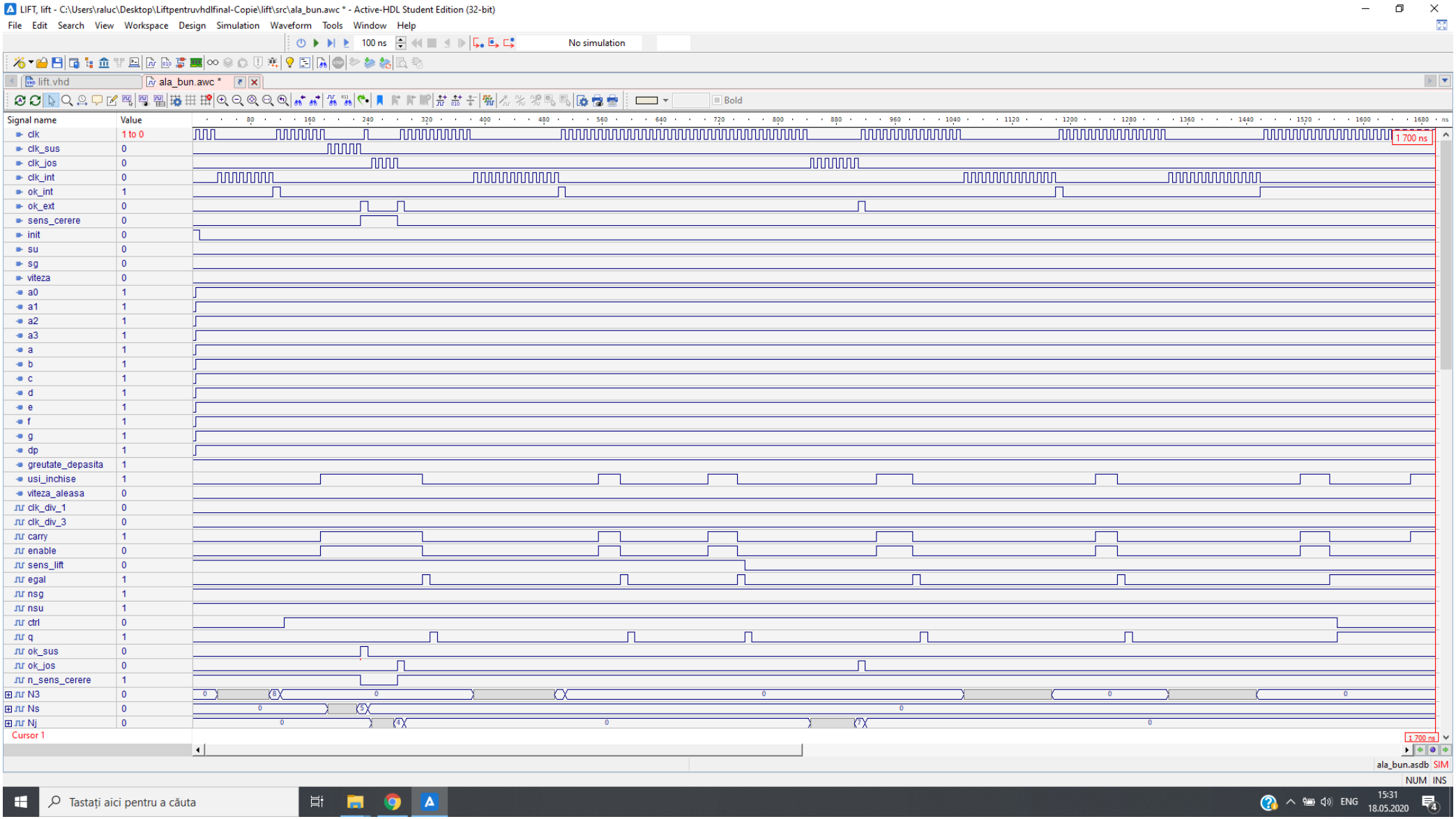
Schema componentei, având pe intrări semnalele pe care le primește efectiv în implementare și ieșirea pe care o generează în circut:

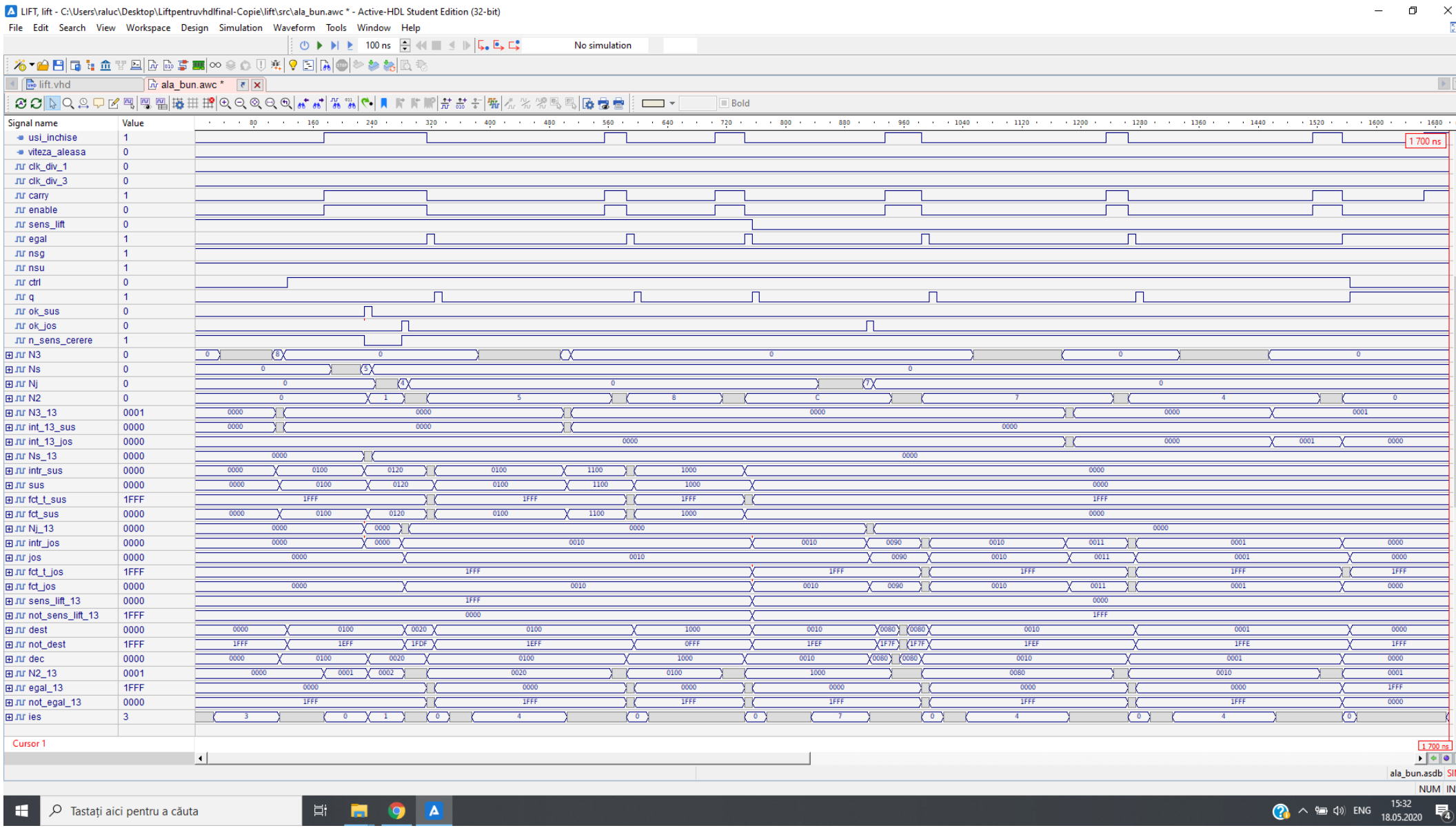


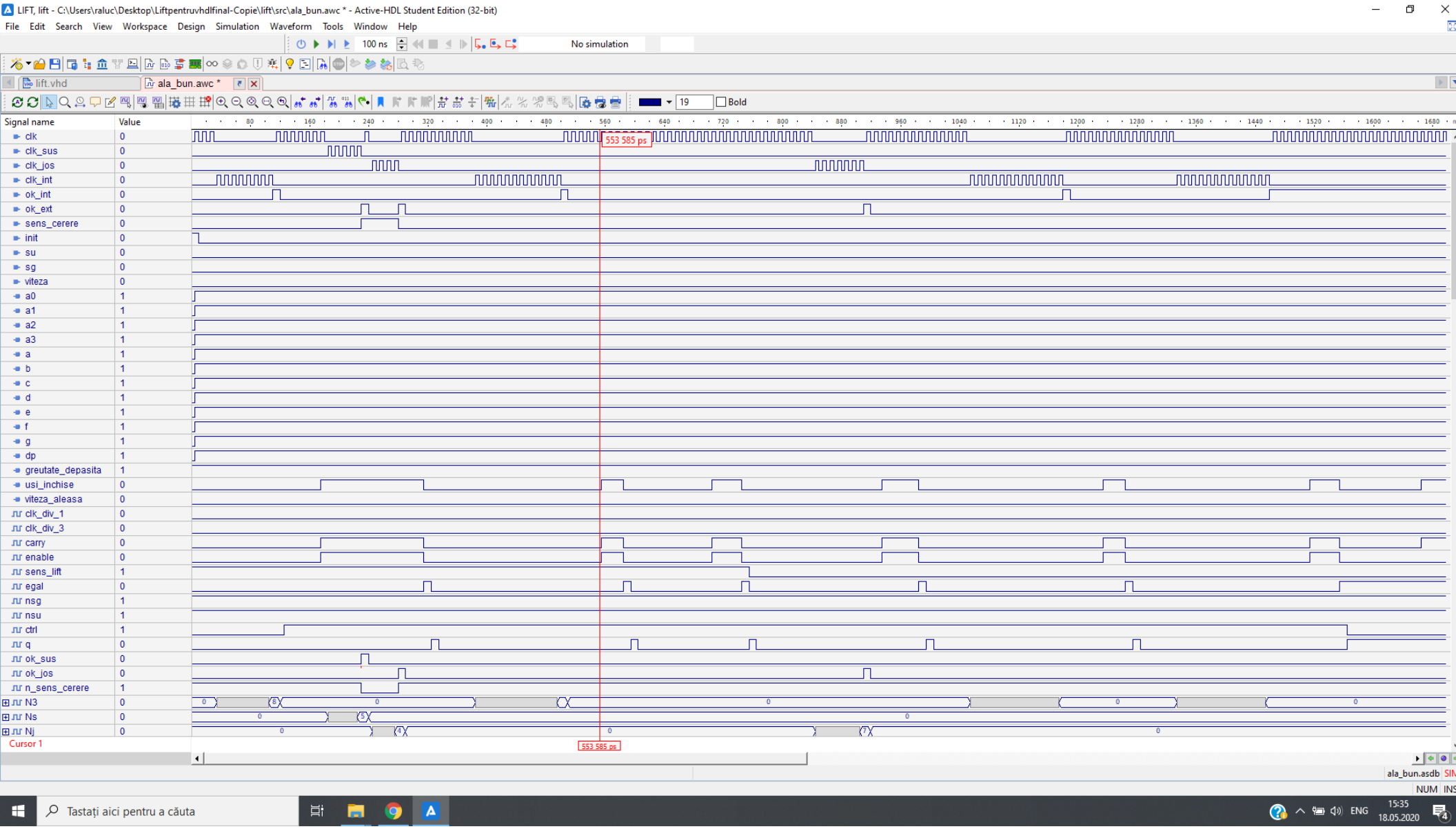
3.3 Proiectare ansamblu

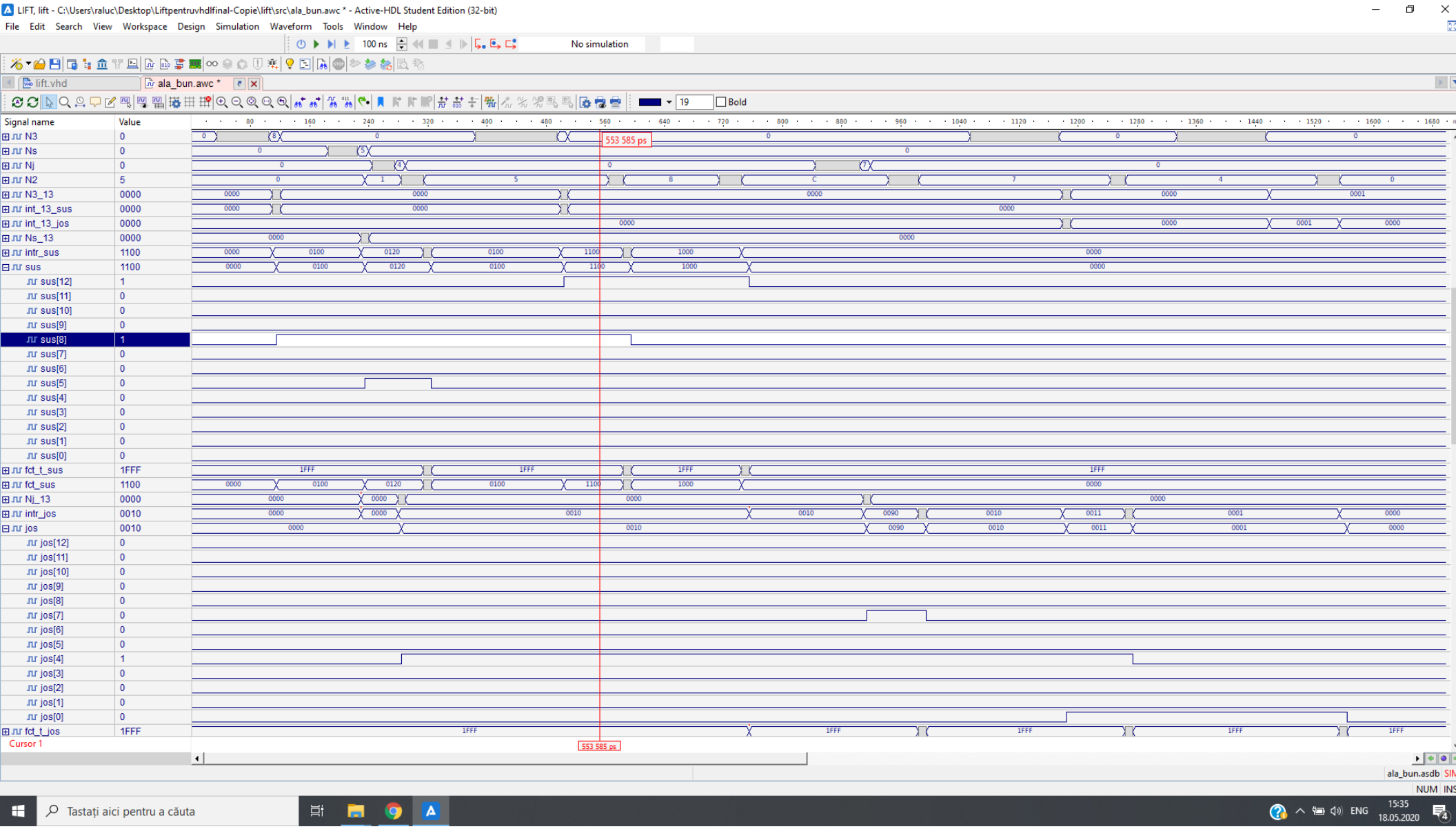
În codul scris în VDHL, fișierul lift.vhd este cel care descrie funcționarea ansamblului reprezentat de lift. Aici s-a declarat entitatea lift, ale cărei intrări și ieșiri sunt chiar intrările și ieșirile automatului. Arhitectura entității lift este pur structurală, conținând declarările tuturor componentelor folosite, urmate de declarările semnalelor intermediare (pe un bit, pe 4 biți și pe 13 biți), iar mai apoi, maparea componentelor strict pe baza schemei de detaliu (inclusiv denumirea semnalelor), care a fost descrisă anterior. În finalul arhitecturii se atribuie semnalelor de ieșire ale automatului, cele trei semnale ce indică dacă a fost depășită greutatea maximă admisă a liftului, dacă ușile sunt închise și care este treapta de viteză aleasă.

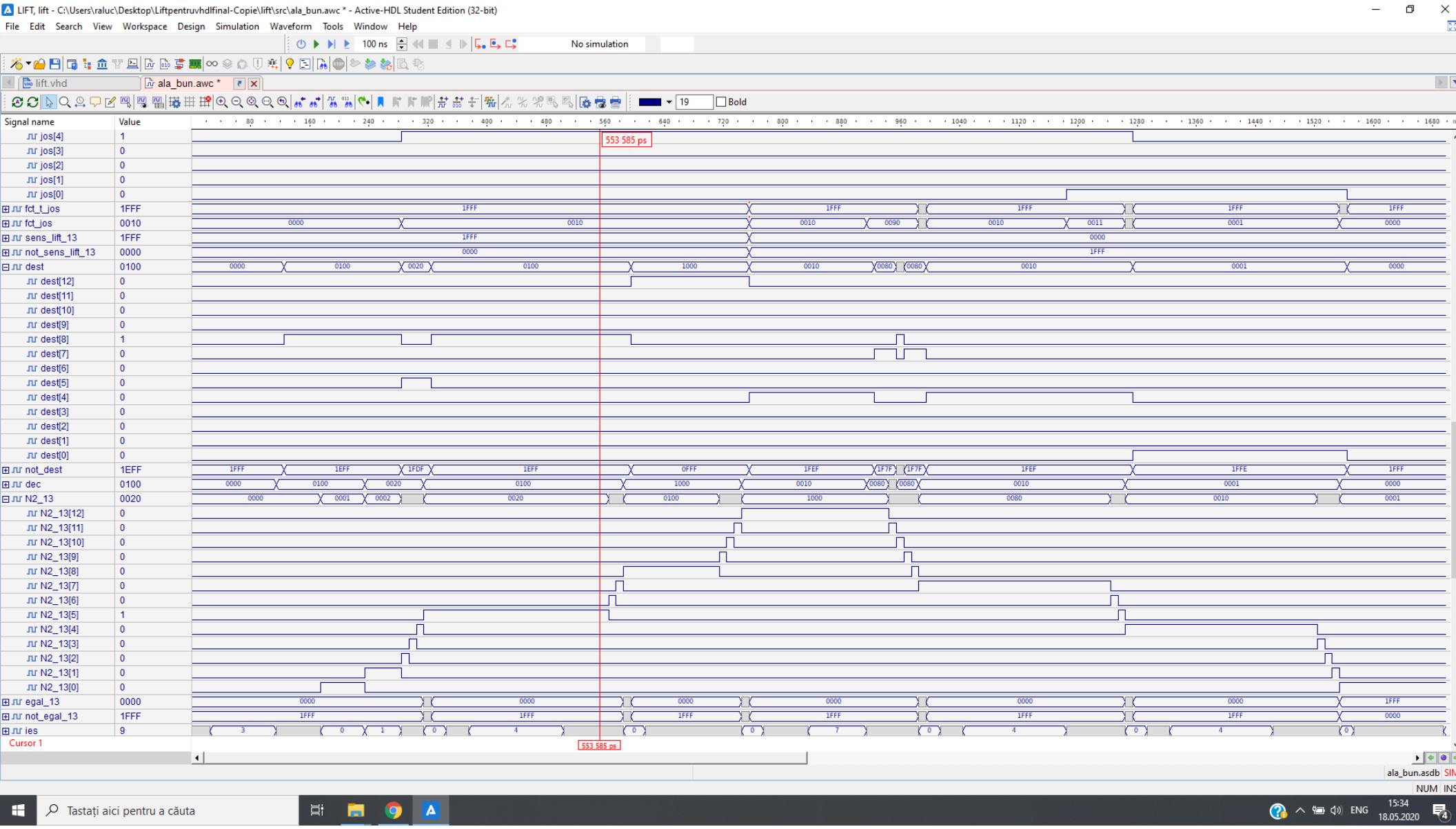
3.4 Simulare în Active – HDL




LIFT, lift - C:\Users\raluc\Desktop\Liftpentruvhdlfinal-Copie\lift\src\ala_bun.awc * - Active-HDL Student Edition (32-bit)
File Edit Search View Workspace Design Simulation Waveform Tools Window Help
No simulation
lift.vhd
ala_bun.awc *
Signal name
Value
usi_inchise
viteza_aleasa
clk_div_1
clk_div_3
carry
enable
sens_lift
egal
nsg
nsu
ctrl
q
ok_sus
ok_jos
n_sens_cerere
N3
Ns
Nj
N2
N3_13
int_13_sus
int_13_jos
Ns_13
intr_sus
sus
fct_t_sus
fct_sus
Nj_13
intr_jos
jos
fct_t_jos
fct_jos
sens_lift_13
not_sens_lift_13
dest
not_dest
dec
N2_13
egal_13
not_egal_13
ies
1 700 ns
Cursor 1
ala_bun.asdb
NUM
Tastați aici pentru a căuta
ENG
15:32
18.05.2020


LIFT, lift - C:\Users\raluc\Desktop\Liftpentruvhdlfinal-Copie\lift\src\ala_bun.awc * - Active-HDL Student Edition (32-bit)
File Edit Search View Workspace Design Simulation Waveform Tools Window Help
100 ns
No simulation
lift.vhd
ala_bun.awc *
Bold
Signal name
Value
clk
clk_sus
clk_jos
clk_int
ok_int
ok_ext
sens_cerere
init
su
sg
viteza
a0
a1
a2
a3
a
b
c
d
e
f
g
dp
greutate_depasita
usi_inchise
viteza_aleasa
clk_div_1
clk_div_3
carry
enable
sens_lift
egal
nsg
nsu
ctrl
q
ok_sus
ok_jos
n_sens_cerere
N3
Ns
Nj
553 585 ps
Cursor 1
ala_bun.asdb
NUM
Tastați aici pentru a căuta
ENG
15:35
18.05.2020


LIFT, lift - C:\Users\raluc\Desktop\Liftpentruvhdlfinal-Copie\lift\src\ala_bun.awc * - Active-HDL Student Edition (32-bit)
File Edit Search View Workspace Design Simulation Waveform Tools Window Help
100 ns
No simulation
lift.vhd
ala_bun.awc *
Bold
Signal name
Value
N3
Ns
Nj
N2
N3_13
int_13_sus
int_13_jos
Ns_13
intr_sus
sus
sus[12]
sus[11]
sus[10]
sus[9]
sus[8]
sus[7]
sus[6]
sus[5]
sus[4]
sus[3]
sus[2]
sus[1]
sus[0]
fct_t_sus
fct_sus
Nj_13
intr_jos
jos
jos[12]
jos[11]
jos[10]
jos[9]
jos[8]
jos[7]
jos[6]
jos[5]
jos[4]
jos[3]
jos[2]
jos[1]
jos[0]
fct_t_jos
553 585 ps
Cursor 1
ala_bun.asdb
NUM
Tastați aici pentru a căuta
ENG
15:35
18.05.2020


LIFT, lift - C:\Users\raluc\Desktop\Liftpentruvhdlfinal-Copie\lift\src\ala_bun.awc * - Active-HDL Student Edition (32-bit)
File Edit Search View Workspace Design Simulation Waveform Tools Window Help
100 ns
No simulation
lift.vhd
ala_bun.awc *
Bold
Signal name
Value
553 585 ps
jos[4] 1
jos[3] 0
jos[2] 0
jos[1] 0
jos[0] 0
fct_t_jos 1FFF
fct_jos 0010
sens_lift_13 1FFF
not_sens_lift_13 0000
dest 0100
dest[12] 0
dest[11] 0
dest[10] 0
dest[9] 0
dest[8] 1
dest[7] 0
dest[6] 0
dest[5] 0
dest[4] 0
dest[3] 0
dest[2] 0
dest[1] 0
dest[0] 0
not_dest 1EFF
dec 0100
N2_13 0020
N2_13[12] 0
N2_13[11] 0
N2_13[10] 0
N2_13[9] 0
N2_13[8] 0
N2_13[7] 0
N2_13[6] 0
N2_13[5] 1
N2_13[4] 0
N2_13[3] 0
N2_13[2] 0
N2_13[1] 0
N2_13[0] 0
egal_13 0000
not_egal_13 1FFF
ies 9
Cursor 1
553 585 ps
ala_bun.asdb
NUM INS
Tastați aici pentru a căuta
ENG
15:34
18.05.2020

3.5 Fișierul de constrângeri din ISE pentru placa cu FPGA aleasă

Conținutul fișierului *lift.ucf*, pe baza căruia se va explica ulterior poziționarea pinilor, a butoanelor, a întrerupătoarelor și a led-urilor:

```
NET "clk" LOC=V10;
NET "clk_sus" LOC=A8;
NET "clk_jos" LOC=C9;
NET "clk_int" LOC=B8;
NET "ok_int" LOC=C4;
NET "ok_ext" LOC=D9;
NET "init" LOC=T10;
NET "su" LOC=T9;
NET "sg" LOC=V9;
NET "viteza" LOC=M8;
NET "sens_cerere" LOC=N8;
NET "greutate_depasita" LOC=U16;
NET "usi_inchise" LOC=V16;
NET "viteza_aleasa" LOC=U15;
NET "a0" LOC=N16;
NET "a1" LOC=N15;
NET "a2" LOC=P18;
NET "a3" LOC=P17;
NET "a" LOC=T17;
NET "b" LOC=T18;
NET "c" LOC=U17;
NET "d" LOC=U18;
NET "e" LOC=M14;
NET "f" LOC=N14;
NET "g" LOC=L14;
```

NET "dp" LOC=M13;

4. Lista de componente utilizate

- *div_1*
- *num_mod_13*
- *div_3*
- *si_4*
- *num2*
- *decod_13*
- *si_2_13*
- *si_2_1bit*
- *sau_3_13biti*
- *registru_13*
- *nu_13*
- *comp_decizie*
- *comparator_13*
- *codificator_1_13*
- *inversor*
- *decod_13_lift*
- *registru_13_dest*
- *comparator_n2*
- *n_timp*
- *bist_D*
- *componenta_afisare*

5. Semnificația semnalelor I/O și a semnalelor interne

- Semnale de intrare:
  - **clk** (oscilatorul cu cuart de pe placă)
  - **clk_int** (clock-ul numărătorului ce înregistrează cererile din interiorul liftului)
  - **clk_sus** (clock-ul numărătorului ce înregistrează cererile de urcare, din exteriorul liftului)
  - **clk_jos** (clock-ul numărătorului ce înregistrează cererile de coborâre, din exteriorul liftului)
  - **ok_int** (intrarea ce trebuie acționată la finalizarea introducerii unei cereri din interiorul liftului)
  - **ok_ext** (intrarea ce trebuie acționată la finalizarea introducerii unei cereri din exteriorul liftului)

- **sens_cerere** (sensul cererii făcute din exteriorul liftului)
- **su** (senzor de prezență al ușilor)
- **sg** (senzor de greutate)
- **viteza** (viteza liftului)

- Semnale de ieșire:
  - **a0**
  - **a1**
  - **a3**
  - **a3**

  } Anozii afișajului BCD 7 segmente

  - **a**
  - **b**
  - **c**
  - **d**
  - **e**
  - **f**
  - **g**
  - **dp**

  } Catozii afișajului BCD 7 segmente

  - **greutate_depasita**
  - **usi_inchise**
  - **viteza_aleasa**
- Semnale intermediare:

  (explicate în descrierea schemei de detaliu)

  - *Pe un bit:*
    - **clk_div_1**
    - **clk_div_3**
    - **carry**
    - **enable**
    - **sens_lift**
    - **egal**
    - **nsg**
    - **nsu**
    - **ctrl**
    - **q**
    - **ok_sus**
    - **ok_jos**
    - **n_sens_cerere**

- *Pe 4 biți:*
  - **N3**
  - **Ns**
  - **Nj**
  - **N2**

- *Pe 13 biți:*
  - **N3_13**
  - **int_13_sus**
  - **int_13_jos**
  - **Ns_13**
  - **intr_sus, sus**
  - **fct_t_sus**
  - **fct_sus**
  - **Nj_13**
  - **intr_jos**
  - **jos**
  - **fct_t_jos**
  - **fct_jos**
  - **sens_lift_13**
  - **not_sens_lift_13**
  - **dest**
  - **not_dest**
  - **dec**
  - **N2_13**
  - **egal_13**
  - **not_egal_13**

6. Justificarea soluției alese

În stadiile inițiale de proiectare am stabilit că pentru a transmite cereri automatului de control al liftului, dar și pentru a reda poziția curentă a liftului va fi nevoie de numărătoare.

Necesitatea divizoarelor de frecvență a fost evidentă datorită posibilității liftului de a se deplasa cu două viteze distincte; pentru a reda acest lucru pe ieșirile numărătorului ce reprezintă, de fapt, liftul, a fost nevoie ca semnalul de CLOCK al acestuia să poată funcționa după două frecvențe diferite.

Pentru a reține cererile liftului și etajul curent am ales decodificarea informației binare primite de la numărătoare în cod binar, prin transpunerea acesteia pe 13 biți, fiecare bit reprezentând, de fapt, un etaj al hotelului pentru care se proiectează liftul.

Cererile am ales să le reținem folosind registre, pe motiv că scrierea acestora în cod VHDL este una facilă și intuitivă în contextul proiectului nostru. Este nevoie de două registre pentru cereri, deoarece sensul liftului este prioritatea 0 în alegerea următoarei destinații, așa că trebuie să se facă distincția între cererile de pe *registru_sus* și *registru_jos*. S-a impus folosirea unui nou registru pentru a reține următoarea destinație, întrucât cererea care este în acest moment destinație poate fi înlocuită de o alta înainte de onorare, iar fosta informație trebuie să nu se piardă, fiind folosită mai apoi de automat.

Pentru a afișa pe placă etajul curent al liftului și sensul de deplasare pe afișajul BCD 7 segmente a fost nevoie de conceperea unei componente speciale de afișare.

7. Utilizare

Pentru implementarea proiectului pe placa FPGA am ales tipul de placă Nexys 3, deoarece acest tip de placă are suficiente afișoare, întrerupătoare și led-uri pentru a corespunde cerințelor proiectului elaborat de noi, fiind în același timp un model de plăcuță cu care suntem, totuși, mai familiarizați de la laboratoare.

Am avut nevoie de toate cele 5 butoane de tip push-button ale plăcii, după cum urmează:

- butonul de sus (A8): se va folosi pentru a stabili etajul de la care se face cererea de urcare din exteriorul liftului; se va apăsa acest buton de n ori pentru a stabili etajul n, cu mențiunea că pentru a selecta parterul se va apăsa de 13 ori; corespunde semnalului **clk_sus** din codul VHDL
- butonul de jos (C9): se va folosi pentru a stabili etajul de la care se face cererea de coborâre din exteriorul liftului; se va apăsa acest buton de n ori pentru a stabili etajul n, cu mențiunea că pentru a selecta parterul se va apăsa de 13 ori; corespunde semnalului **clk_jos** din codul VHDL
- butonul din centru (B8): se va folosi pentru a stabili etajul de la care se face cererea din interiorul liftului; se va apăsa acest buton de n ori pentru a stabili etajul n, cu

mențiunea că pentru a selecta parterul se va apăsa de 13 ori; corespunde semnalului **clk_int** din codul VHDL

- butonul din dreapta (D9): se va apăsa la terminarea introducerii etajului de la care se face o cerere din exteriorul liftului; corespunde semnalului **ok_ext** din codul VHDL
- butonul din stânga (C4): se va apăsa la terminarea introducerii etajului de la care se face o cerere din interiorul liftului; corespunde semnalului **ok_int** din codul VHDL

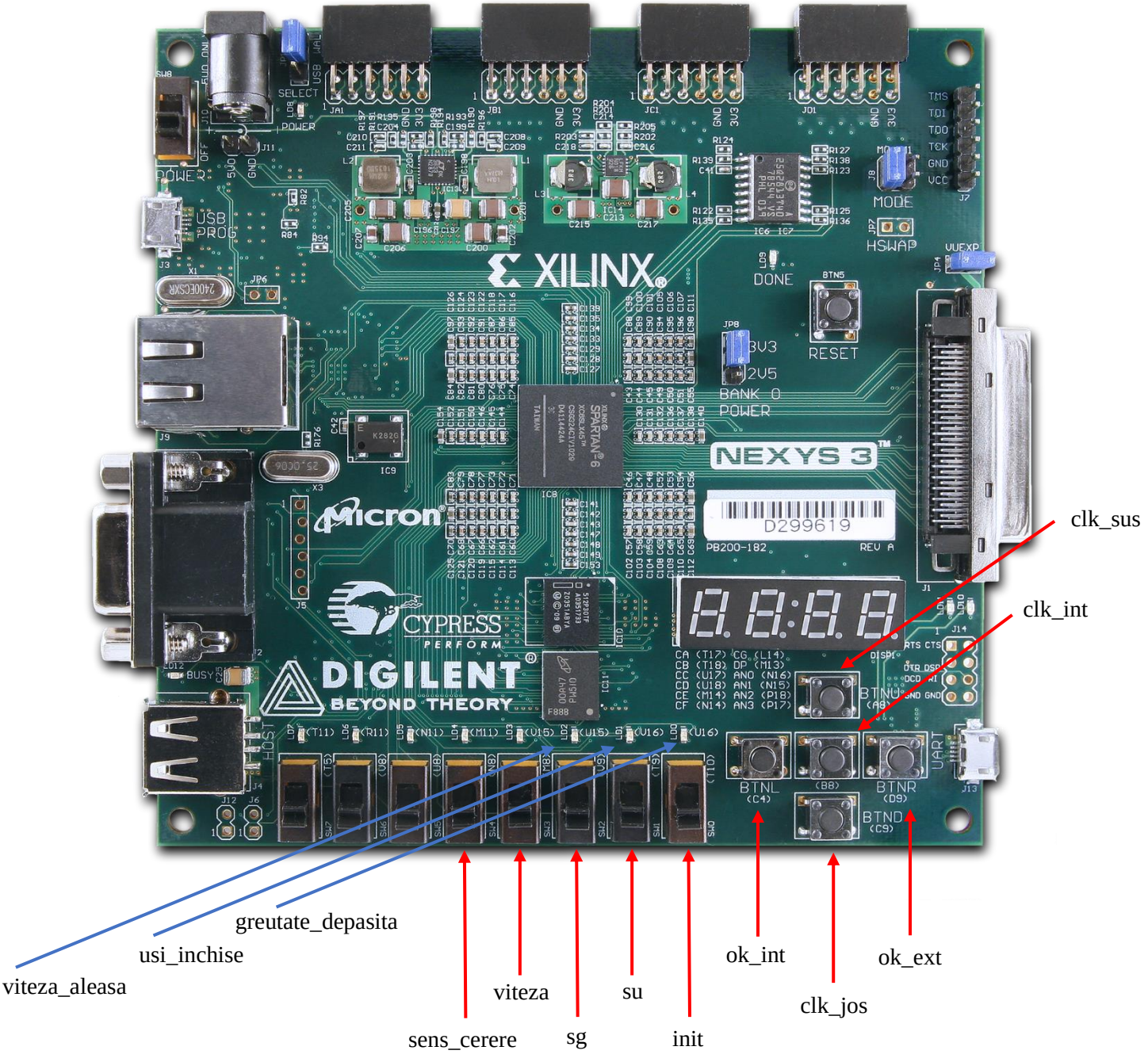
Pentru introducerea semnalelor de intrare a fost nevoie de 5 întrerupătoare, astfel:

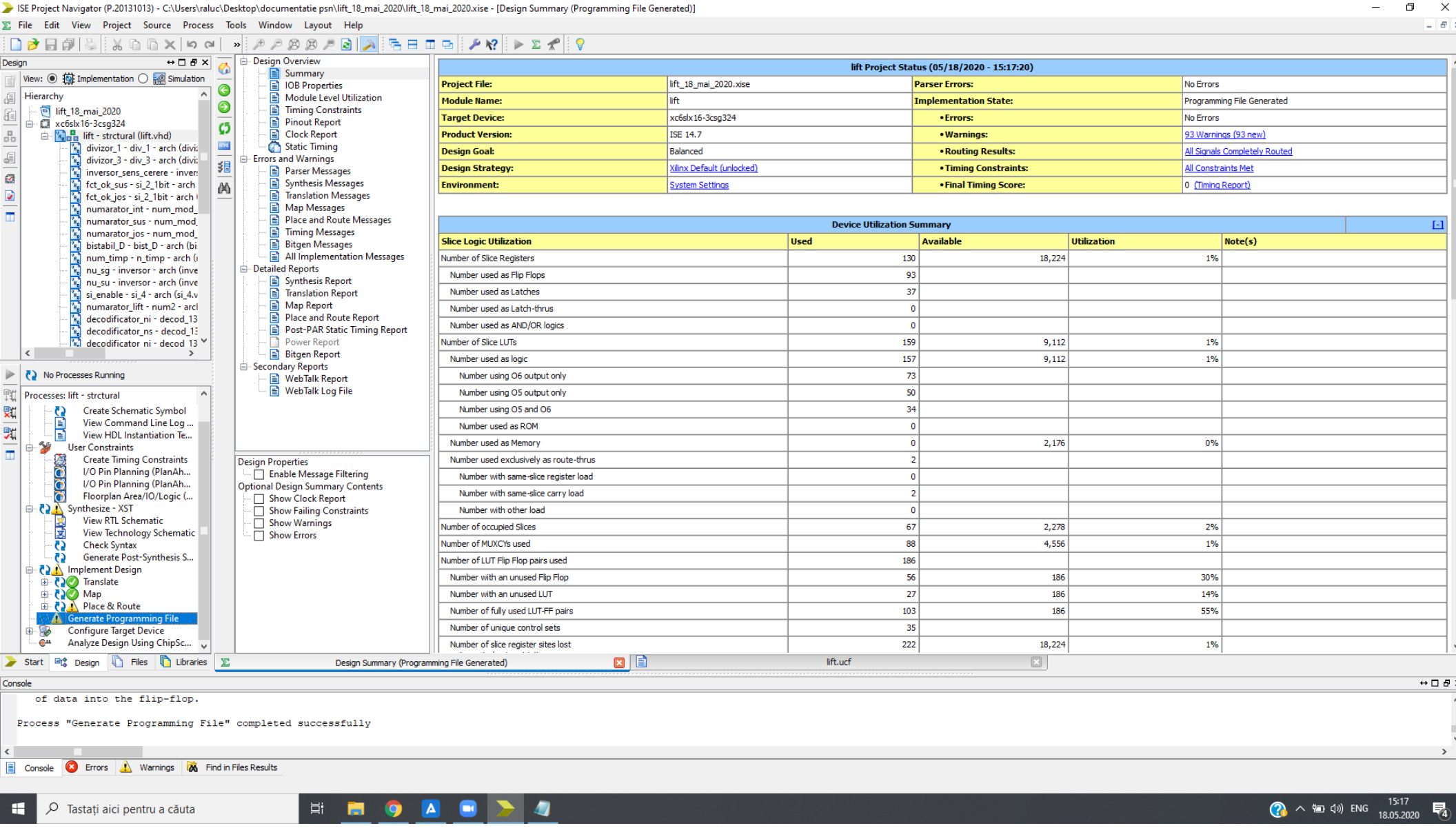
- întrerupătorul SW0 (T10): semnalul **init** – va fi trecut pe poziția 1, iar apoi, din nou pe 0 când se dorește inițializarea automatului
- întrerupătorul SW1 (T9): semnalul **su** – va fi trecut pe poziția 1 dacă se consideră că greutatea maximă admisă pentru lift a fost depășită, și pe poziția 0 în caz contrar
- întrerupătorul SW2 (V8): semnalul **sg** – va fi trecut pe poziția 1 dacă se consideră că există în calea ușii liftului ceva ce îi împiedică închiderea, și pe poziția 0 în caz contrar
- întrerupătorul SW3 (M8): **viteza** – va fi trecut pe poziția 1 dacă se dorește ca liftul să se deplaseze cu viteza de 3 secunde/ etaj, și pe poziția 0 dacă viteza aleasă este de o secundă/ etaj
- întrerupătorul SW4 (N8): **sens_cerere** – va fi trecut pe poziția 1 dacă din exteriorul liftului se face o cerere de urcare, și pe poziția 0 dacă din exteriorul liftului se dorește să se coboare

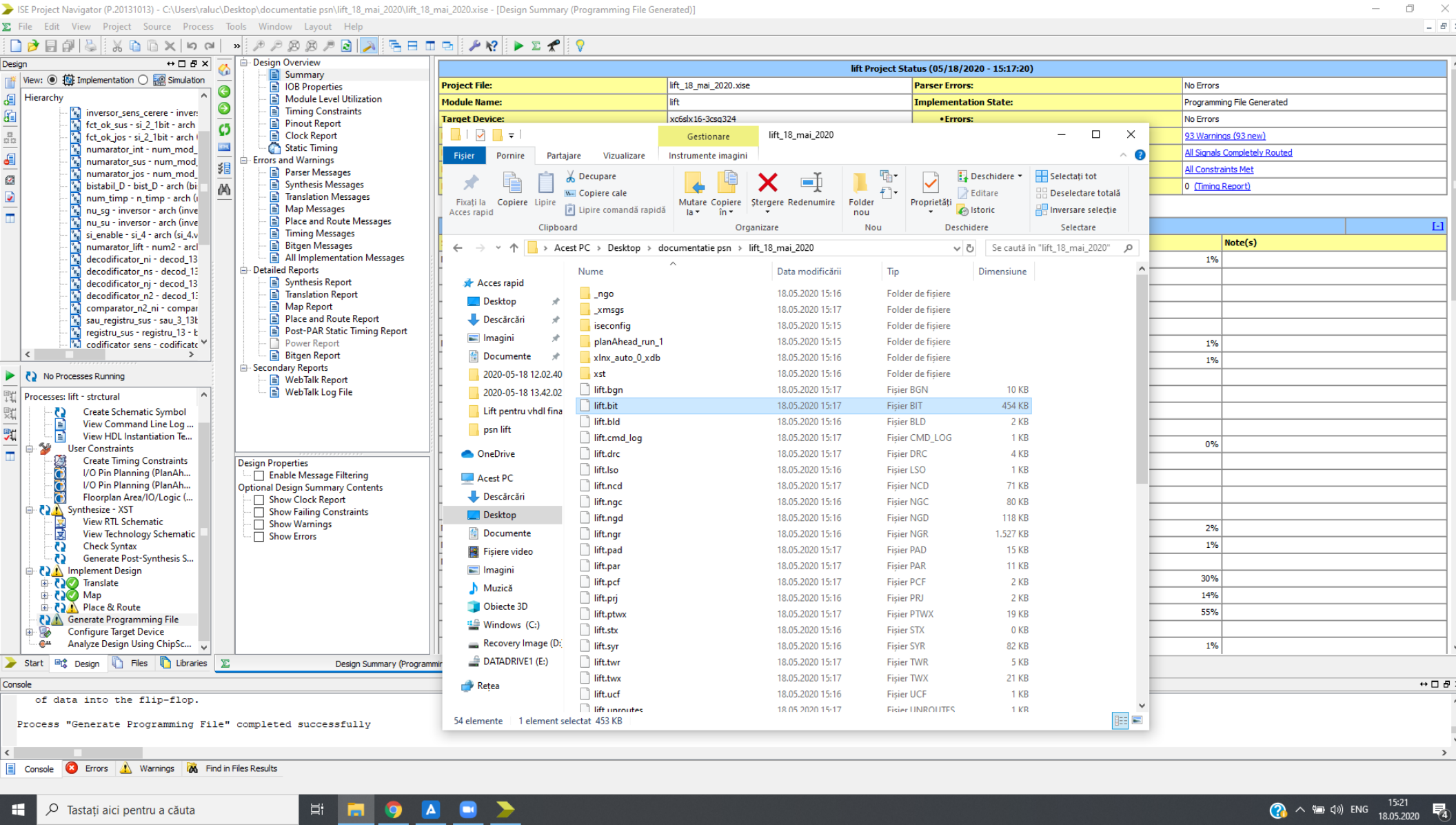
Pentru afișare am avut nevoie de 3 led-uri și de afișajul BCD 7 segmente:

- Led-uri:
  - led-ul LD0 (U16): **greutate_depasita** – se aprinde dacă limita de greutate a fost depășită
  - led-ul LD1 (V16): **usi_inchise** – se aprinde dacă ușile liftului sunt închise
  - led-ul LD2 (U15): **viteza_aleasa** – se aprinde dacă viteza aleasă este cea de 3 secunde/etaj, și rămâne stins dacă viteza este de o secundă/etaj

- Afișajul BCD 7 segmente: se vor folosi 3 din cele 4 afișoare, cel mai din dreapta pentru afișarea cifrei unităților etajului curent, al doilea din dreapta pentru cifra zecilor etajului curent, iar al treilea din dreapta pentru a reda sensul de deplasare al liftului, în timp ce al patrulea afișor din dreapta va rămâne mereu stins. Descrierea detaliată a funcționării afișajului BCD este descrisă pe larg în cadrul prezentării componentei de afișare.


clk_sus
clk_int
viteza_aleasa
usi_inchise
greutate_depasita
sens_cerere
viteza
sg
su
init
ok_int
clk_jos
ok_ext


ISE Project Navigator (P.20131013) - C:\Users\raluc\Desktop\documentatie psn\lift_18_mai_2020\lift_18_mai_2020.xise - [Design Summary (Programming File Generated)]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
lift_18_mai_2020
xc6slx16-3csg324
lift - strctural (lift.vhd)
No Processes Running
Processes: lift - strctural
Create Schematic Symbol
View Command Line Log ...
View HDL Instantiation Te...
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAh...
I/O Pin Planning (PlanAh...
Floorplan Area/IO/Logic (...
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis S...
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Analyze Design Using ChipSc...
Design Overview
Summary
IOB Properties
Module Level Utilization
Timing Constraints
Pinout Report
Clock Report
Static Timing
Errors and Warnings
Parser Messages
Synthesis Messages
Translation Messages
Map Messages
Place and Route Messages
Timing Messages
Bitgen Messages
All Implementation Messages
Detailed Reports
Synthesis Report
Translation Report
Map Report
Place and Route Report
Post-PAR Static Timing Report
Power Report
Bitgen Report
Secondary Reports
WebTalk Report
WebTalk Log File
Design Properties
Enable Message Filtering
Optional Design Summary Contents
Show Clock Report
Show Failing Constraints
Show Warnings
Show Errors
lift Project Status (05/18/2020 - 15:17:20)
Project File: lift_18_mai_2020.xise
Parser Errors: No Errors
Module Name: lift
Implementation State: Programming File Generated
Target Device: xc6slx16-3csg324
•Errors: No Errors
Product Version: ISE 14.7
•Warnings: 93 Warnings (93 new)
Design Goal: Balanced
•Routing Results: All Signals Completely Routed
Design Strategy: Xilinx Default (unlocked)
•Timing Constraints: All Constraints Met
Environment: System Settings
•Final Timing Score: 0 (Timing Report)
Device Utilization Summary
Slice Logic Utilization | Used | Available | Utilization | Note(s)
Number of Slice Registers 130 18,224 1%
Number used as Flip Flops 93
Number used as Latches 37
Number used as Latch-thrus 0
Number used as AND/OR logics 0
Number of Slice LUTs 159 9,112 1%
Number used as logic 157 9,112 1%
Number using O6 output only 73
Number using O5 output only 50
Number using O5 and O6 34
Number used as ROM 0
Number used as Memory 0 2,176 0%
Number used exclusively as route-thrus 2
Number with same-slice register load 0
Number with same-slice carry load 2
Number with other load 0
Number of occupied Slices 67 2,278 2%
Number of MUXCYs used 88 4,556 1%
Number of LUT Flip Flop pairs used 186
Number with an unused Flip Flop 56 186 30%
Number with an unused LUT 27 186 14%
Number of fully used LUT-FF pairs 103 186 55%
Number of unique control sets 35
Number of slice register sites lost 222 18,224 1%
Start Design Files Libraries
Design Summary (Programming File Generated)
lift.ucf
Console
of data into the flip-flop.
Process "Generate Programming File" completed successfully
Console Errors Warnings Find in Files Results
Tastați aici pentru a căuta
ENG 15:17 18.05.2020


ISE Project Navigator (P.20131013) - C:\Users\raluc\Desktop\documentatie psn\lift_18_mai_2020\lift_18_mai_2020.xise - [Design Summary (Programming File Generated)]
File Edit View Project Source Process Tools Window Layout Help
Design
View: Implementation Simulation
Hierarchy
inversor_sens_cerere - inver
fct_ok_sus - si_2_1bit - arch
fct_ok_jos - si_2_1bit - arch
numarator_int - num_mod_
numarator_sus - num_mod_
numarator_jos - num_mod_
bistabil_D - bist_D - arch (bi
num_timp - n_timp - arch (
nu_sg - inversor - arch (inve
nu_su - inversor - arch (inve
si_enable - si_4 - arch (si_4.v
numarator_lift - num2 - arcl
decodificator_ni - decod_13
decodificator_ns - decod_1
decodificator_nj - decod_13
decodificator_n2 - decod_1
comparator_n2_ni - compar
sau_registru_sus - sau_3_13
registru_sus - registru_13 - b
codificator sens - codificato
No Processes Running
Processes: lift - strctural
Create Schematic Symbol
View Command Line Log ...
View HDL Instantiation Te...
User Constraints
Create Timing Constraints
I/O Pin Planning (PlanAh...
I/O Pin Planning (PlanAh...
Floorplan Area/IO/Logic (...
Synthesize - XST
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis S...
Implement Design
Translate
Map
Place & Route
Generate Programming File
Configure Target Device
Analyze Design Using ChipSc...
Start Design Files Libraries
Design Overview
Summary
IOB Properties
Module Level Utilization
Timing Constraints
Pinout Report
Clock Report
Static Timing
Errors and Warnings
Parser Messages
Synthesis Messages
Translation Messages
Map Messages
Place and Route Messages
Timing Messages
Bitgen Messages
All Implementation Messages
Detailed Reports
Synthesis Report
Translation Report
Map Report
Place and Route Report
Post-PAR Static Timing Report
Power Report
Bitgen Report
Secondary Reports
WebTalk Report
WebTalk Log File
Design Properties
Enable Message Filtering
Optional Design Summary Contents
Show Clock Report
Show Failing Constraints
Show Warnings
Show Errors
Design Summary (Programmi
lift Project Status (05/18/2020 - 15:17:20)
Project File: lift_18_mai_2020.xise
Module Name: lift
Target Device: xc6slx16-3csg324
Parser Errors: No Errors
Implementation State: Programming File Generated
•Errors: No Errors
93 Warnings (93 new)
All Signals Completely Routed
All Constraints Met
0 (Timing Report)
Note(s)
Gestionare
lift_18_mai_2020
Fișier Pornire Partajare Vizualizare Instrumente imagini
Fixați la Acces rapid Copiere Lipire Decupare Copiere cale Lipire comandă rapidă Clipboard
Mutare la Copiere în Ștergere Redenumire Organizare
Folder nou Nou
Proprietăți Deschidere Editare Istoric Deschidere
Selectați tot Deselectare totală Inversare selecție Selectare
Acest PC › Desktop › documentatie psn › lift_18_mai_2020
Se caută în "lift_18_mai_2020"
Acces rapid
Desktop
Descărcări
Imagini
Documente
2020-05-18 12.02.40
2020-05-18 13.42.02
Lift pentru vhdl fina
psn lift
OneDrive
Acest PC
Descărcări
Desktop
Documente
Fișiere video
Imagini
Muzică
Obiecte 3D
Windows (C:)
Recovery Image (D:
DATADRIVE1 (E:)
Rețea
Nume Data modificării Tip Dimensiune
_ngo 18.05.2020 15:16 Folder de fișiere
_xmsgs 18.05.2020 15:17 Folder de fișiere
iseconfig 18.05.2020 15:15 Folder de fișiere
planAhead_run_1 18.05.2020 15:15 Folder de fișiere
xlnx_auto_0_xdb 18.05.2020 15:16 Folder de fișiere
xst 18.05.2020 15:16 Folder de fișiere
lift.bgn 18.05.2020 15:17 Fișier BGN 10 KB
lift.bit 18.05.2020 15:17 Fișier BIT 454 KB
lift.bld 18.05.2020 15:16 Fișier BLD 2 KB
lift.cmd_log 18.05.2020 15:17 Fișier CMD_LOG 1 KB
lift.drc 18.05.2020 15:17 Fișier DRC 4 KB
lift.lso 18.05.2020 15:16 Fișier LSO 1 KB
lift.ncd 18.05.2020 15:17 Fișier NCD 71 KB
lift.ngc 18.05.2020 15:16 Fișier NGC 80 KB
lift.ngd 18.05.2020 15:16 Fișier NGD 118 KB
lift.ngr 18.05.2020 15:16 Fișier NGR 1.527 KB
lift.pad 18.05.2020 15:17 Fișier PAD 15 KB
lift.par 18.05.2020 15:17 Fișier PAR 11 KB
lift.pcf 18.05.2020 15:17 Fișier PCF 2 KB
lift.prj 18.05.2020 15:16 Fișier PRJ 2 KB
lift.ptwx 18.05.2020 15:17 Fișier PTWX 19 KB
lift.stx 18.05.2020 15:16 Fișier STX 0 KB
lift.syr 18.05.2020 15:16 Fișier SYR 82 KB
lift.twr 18.05.2020 15:17 Fișier TWR 5 KB
lift.twx 18.05.2020 15:17 Fișier TWX 21 KB
lift.ucf 18.05.2020 15:16 Fișier UCF 1 KB
54 elemente 1 element selectat 453 KB
Console
of data into the flip-flop.
Process "Generate Programming File" completed successfully
Console Errors Warnings Find in Files Results
Tastați aici pentru a căuta
ENG 15:21 18.05.2020

8. Posibilități de dezvoltare ulterioară

- Implementarea cu bistabile în loc de registre pentru a avea acces la biții ce rețin informația aflată actualmente în registrele *registru_sus, registru_jos, registru_dest*, în mod individual, și fără a face referire de fiecare dată la toți cei 13 biți
- Introducerea unui afișaj ce arată cât de plin este liftul din punctul de vedere al greutății, în mod gradual, stabilind anumite trepte de referință

- Dotarea liftului cu un buton de panică, prin acționarea căruia să se poată transmite un semnal de alarmă, în cazul în care, spre exemplu, liftul se blochează.
- Introducerea posibilității de alegere între mai multe caracteristici de funcționare (temperatură, intensitate) a unui aparat încorporat de climatizare
- Posibilitatea de selecție a muzicii din lift în funcție de preferințe