Universitatea Tehnica Cluj-Napoca

Facultatea de Automatica si Calculatoare

Sectia Calculatoare

Anul 2019-2020

Semestrul II



**Proiect: LIFT P+12 etaje**

Profesor coordonator: Diana Irena Pop

Studenti: Tudor Mihai-Nicusor

Varvara Raluca Ana-Maria

Grupa: 30212

**CUPRINS**

1. Specificatie proiect -
2. Organigrama -
3. Descriere schema de detaliu
4. Proiectare si implementare -

- Lista de componente utilizate -

-Date de intrare -

-Date de iesire -

1. Descriere componente
2. Descriere proiectare in ansamblu
3. Fisierul de constrangeri din ISE pentru placa cu FPGA aleasa
4. Justificarea solutiei alese
5. Utilizare si rezultate

- resurse necesare

- descrierea utilizarii

- rezultate obţinute pe plăcile cu FPGA

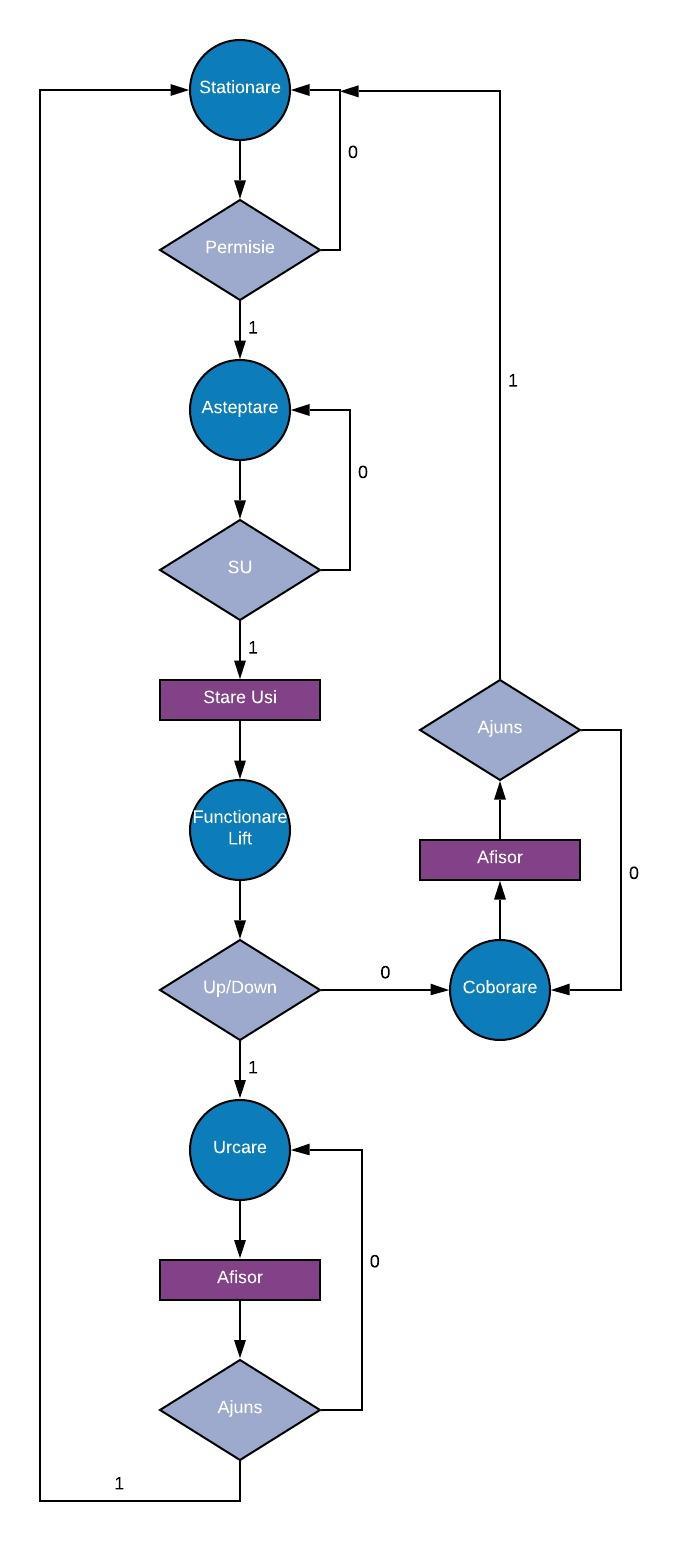
1. Posibilitati de dezvoltare ulterioara
2. **Specificatia proiectului**

Sa se proiecteze un automat care comanda un lift intr-un hotel cu P+12 etaje. Liftul trebuie sa raspunda solicitarilor persoanelor aflate in interior si cererilor exterioare (sus, jos) care apar pe parcurs de la usile aflate la fiecare nivel. Ordinea de onorare a cererilor tine cont de sensul de mers (urcare sau coborare). Se onoreaza cererile in ordinea etajelor, indiferent de unde provin ele (lift sau exterior). Liftul are o intrare care sesizeaza depasirea greutatii maxime admise si nu porneste in acest caz. Plecarea nu are loc daca usile nu sunt inchise. Usile trebuie sa stea deschise un interval de timp programabil. Usile nu se inchid daca exista o persoana in usa. Viteza liftului va fi selectabila intre 2 valori: 1 sau 3 secunde/ etaj. Se considera ca initial liftul de afla la parter, cu usile deschise.

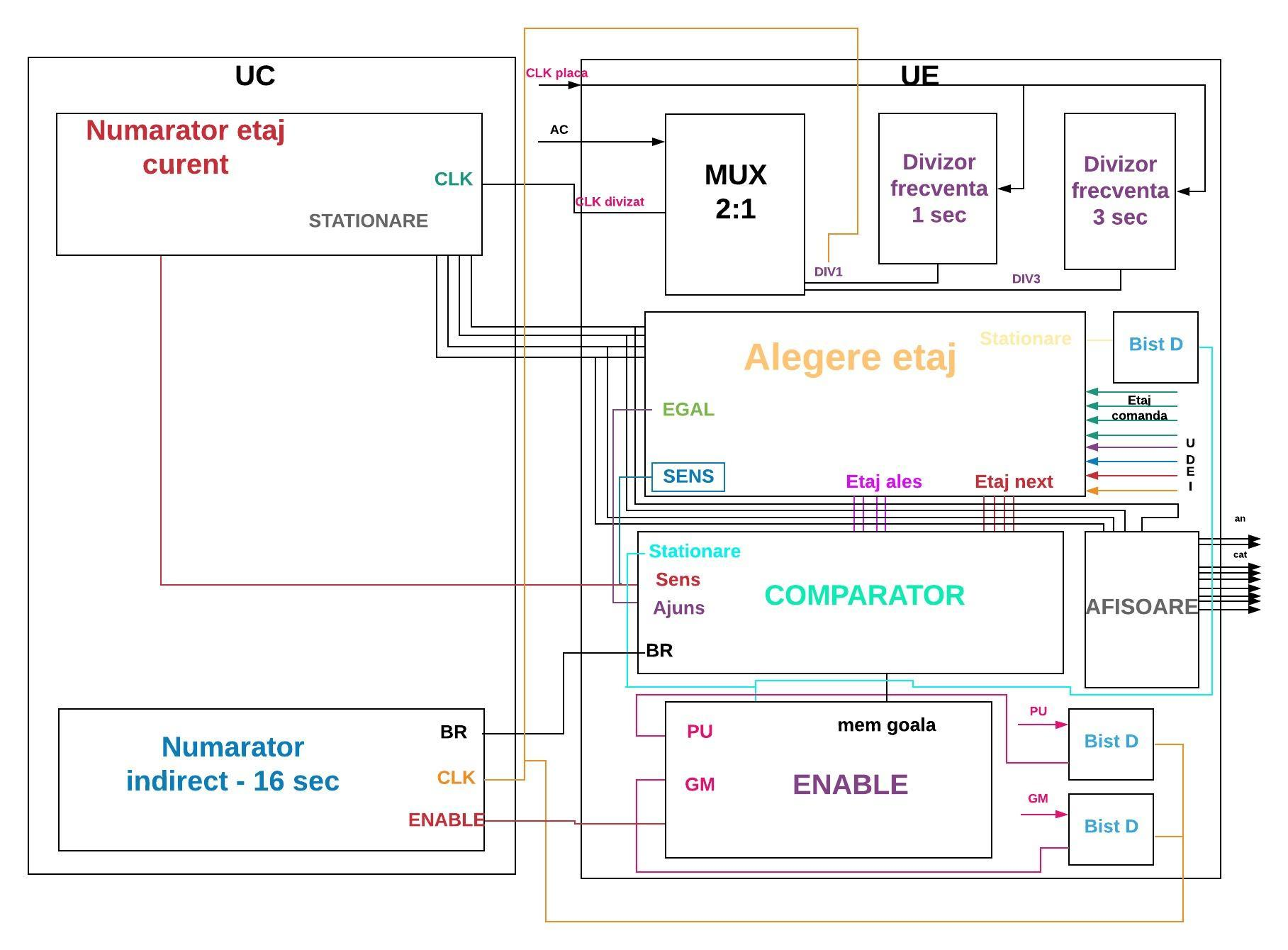
1. **Organigrama**

Starea 1 o sa fie starea de staționare. Semnalul de permisie o sa fie activ atunci când ambii senzori sunt 0, memoria nu este goala, adica are cel putin o comanda, si daca liftul primeste semnal de la comparator ca stationeaza. Starea 2 este starea de așteptare a acelui interval de timp programabil. Testul SU este activ daca s-a așteptat destul (stationare = ’0’, acesta devine 0 daca etajul ales nu este egal cu cel curent si daca semnalul trimis de la numaratorul indirect de stationare este 1, adica s-a asteptat intervalul programat). Starea 3 este starea in care liftul incepe sa functioneze, apoi testul up/down decide dacă liftul urca sau coboară. Acest semnal il generam prin compararea etajelor curent si ales. În starea 4, liftul urca, iar semnalul „ajuns” indica daca s-a ajuns la etajul la care trebuia, daca nu, mai urca încă un etaj. Din nou, acest etaj il primim tot de la comparator. La fel este si starea de coborâre, iar decizia „ajuns” de dupa starea de coborare este identica cu cea de dupa starea de urcare. Apoi se intra din nou in starea de staționare, dupa ce s-a urcat sau coborat destul.

Organigrama reprezinta unitatea de control a automatului si am decis sa o implementam cu numaratoare. Dupa cum se observa, organigrama se poate imparti in 2 parti - partea de asteptare si parte de indeplinire a comenzii (urcare sau coborare). Asa ca am decis sa implementam cate un numarator pentru fiecare parte: numarator „indirect stationare” pentru partea de asteptare, si pentru partea de indeplinire a comenzii un numarator reversibil pentru etajul curent. Cum cele 2 numaratoare nu pot functiona concomitent, ne folosim de enable-uri pentru a face asta. Numaratorul indirect are mai multe conditii, astfel, i-am facut o componenta separata. Numaratorul indirect funcioneaza cand liftul stationeaza, iar cel pentru etaje functioneaza cand liftul nu stationeaza.

****

1. **Schema bloc**



Schema bloc reprezinta proiectul nostru la nivel hardware, astfel impartindu-si sarcinile fiecarei componente separate. La aceasta parte a implementari automatului am lucrat top-down. Prima oara am impartit sarcinile pe care trebuie sa le faca liftul in unitatea de control si unitatea de executie. Intre cele 2 unitati exista comunicare:

Unitatea de comanda, adica organigrama, am implementat-o dupa cum am spus mai sus, cu 2 numaratoare.

Unitatea de executie, se ocupa cu gestionarea resurselor. Avem o componenta care retine comenzile si returneaza care este cel mai avantajos etaj la care sa mearga liftul dupa. Pentru a implementa organigrama avem nevoie de anumite date de la unitatea de executie, de exemplu, de sensul in care va merge liftul. Pentru aceasta avem nevoie de un comparator, care ne da informatii si despre celalalt test pe care il face organigrama: daca stationeaza sau nu liftul. Pentru a aceasta avem nevoie de la unitatea de executie informatii legate de finalizarea asteptarii.

De asemenea pentru a implementa organigrama mai avem nevoie de alegerea impulsului de tact folosit pentru numaratorul etaj curent, impuls primit de la multiplexor.

1. **Proiectare**
   1. **Lista componentelor**

* Numarator reversibil pentru etajul curent
* Numarator indirect de stationare
* Componenta de alegere a etajului
* Comparator etaj curent etaj ales
* Enable pentru numarator indirect
* Afisoare 7 segmente
* MUX 2:1
* Divizor de frecventa 1 secunda
* Divizor de frecventa 3 secunde
* Debouncer pentru butoane
* Bistabil D

1. **Date de intrare**

* **Etaj comanda (switchuri):** intrare pe 4 biti, care reprezinta etajul la care vrem sa mergem fie in cazul in care suntem in interiorul liftului, fie in cazul in care suntem in exteriorul liftului. (Etajele se introduc in cod binar);
* **Int, exte, up, down (switchuri):** fiecare intrare este activa in functie de unde ne aflam cand facem cererea si in functie de sensul in care vrem sa mergem daca suntem in exterior. (EX: intrarea de interior e activa cand suntem in interior, cea din exterior necesita ca si una dintre intrarile sus si jos sa fie activa, deoarece cand facem comanda din exterior trebuie sa precizam si sensul).Am ales acest mod de a introduce datele, deoarece pentru un etaj se pot face mai multe comenzi si e cel mai eficient mod gasit de noi, si, totodata, ne va ajuta pentru alegerea etajului urmator;
* **OK (switch):** intrarea este activa in momentul in care am setat celelalte intrari pentru a introduce comanda;
* **PU (buton)** este activ atunci cand este o persoana in usa;
* **GM (buton)** este activ atunci cand greutatea maxima este depasita;
* **AC (switch)** alegere clock (1sec/etaj, 3 sec/etaj).

1. **Date de iesire**

* **2 Afisoare (afisoare 7 segmente)** afiseaza etajul curent. Afisor 1 este pentru cifra unitatilor, iar Afisor 2 pentru cifra zecilor. Avem ca iesiri propriu-zise 2 biti pentru ce 2 anozi diferiti care vor fi activi pe rand si 7 biti pentru catozii afisorului activ la acel moment;
* **SU (led)** afiseaza starea usilor.

1. **Descrierea componentelor**
2. **Numarator reversibil etaj curent**

Acest numarator reprezinta, efectiv, functionarea liftului. Arata la ce etaj este liftul in momentul actual.

Are 3 intrari si o iesire. „Sens” este intrarea care determina sensul in care va merge liftul (sus, jos). Aceasta este primita de la comparator, care compara etajul curent si etajul la care trebuie sa ajunga liftul, determinand astfel sensul de miscare. Enable primeste semnal tot de la comparator de la iesirea de „stationare”, deoarece acest numarator functioneaza numai atunci cand liftul nu este in starea de stationare. „Output” este iesirea si determina etajul la care este in momentul respectiv. „Clock” este determinat de multiplexor, care determina ce semnal de tact se va folosi: de 1 secunda sau de 3 secunde.

Singura variabila pe care o folosim este variabila etaj care se modifica in interiorul procesului si apoi iesirea primeste valoarea acesteia. Toate aceste comenzi sunt dictate de clock, totul se intampla pe frontul ascendent. Apoi numaratorul functioneaza numai daca primeste enable si variabila etaj creste sau scade in functie de sens. Daca etajul a ajuns la 12 si sensul este 1, adica in sus, el va ramane la 12, deoarece nu mai exista vreun etaj superior.

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**use IEEE.STD\_LOGIC\_ARITH.ALL;**

**use IEEE.STD\_LOGIC\_UNSIGNED.ALL;**

**entity counter4bit is**

**port (clock: in std\_logic;**

**enable: in std\_logic := '1';**

**sens: in std\_logic := '0'; -- sus:0 jos:1**

**output: out std\_logic\_vector (3 downto 0));**

**end entity;**

**architecture comportamental of counter4bit is**

**begin**

**process (clock, sens, enable)**

**variable etaj: std\_logic\_vector(3 downto 0) := "0000";**

**begin**

**if (clock'event and clock = '1') then**

**if enable = '0' then**

**if sens = '1' then**

**if etaj = "1100" then**

**etaj := "1100";**

**else**

**etaj := etaj + '1';**

**end if;**

**else**

**if etaj = "0000" then**

**etaj := "0000";**

**else**

**etaj := etaj - '1';**

**end if;**

**end if;**

**else**

**etaj := etaj;**

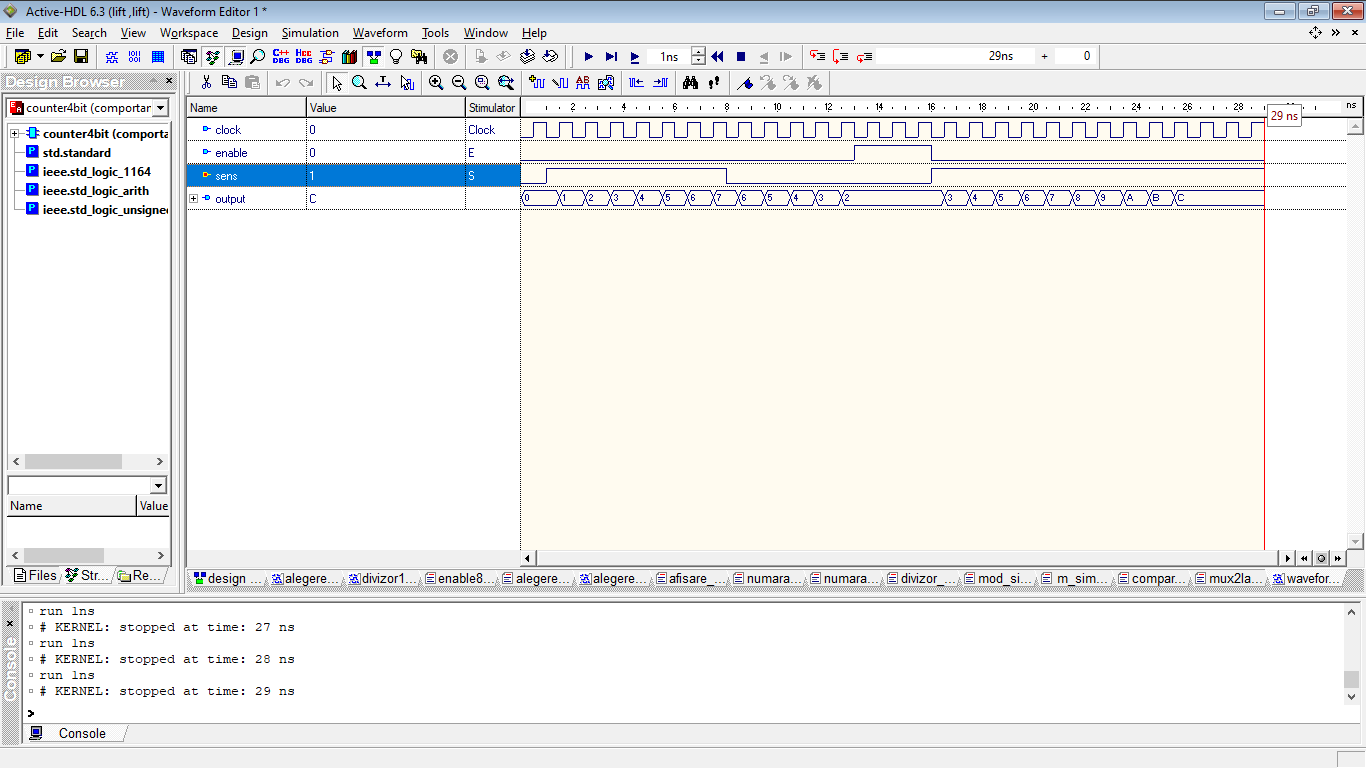
**end if;**

**end if;**

**output <= etaj;**

**end process;**

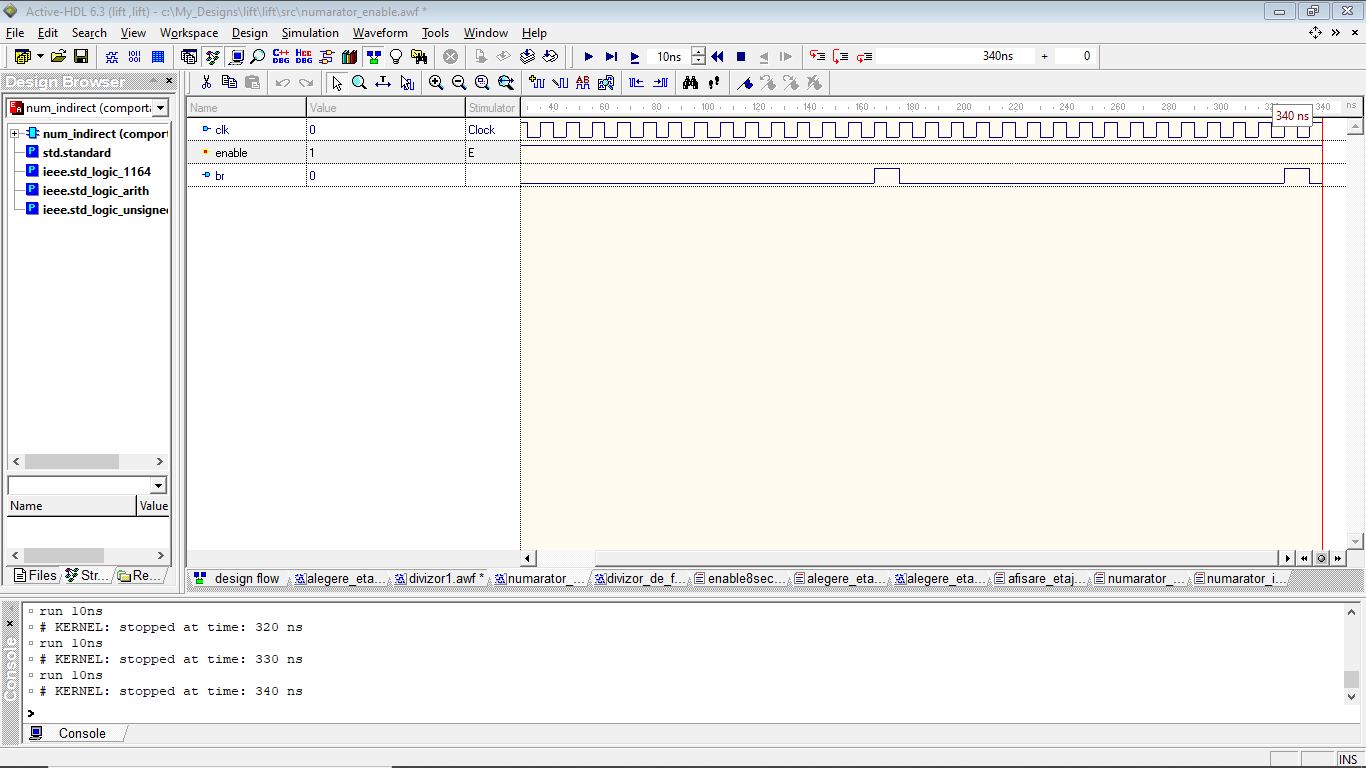
**end architecture;**



1. **Numarator indirect stationare**

Acest numarator este ca un temporizator – comtorizeaza numarul de secunde programat, secunde pe care trebuie sa le astepte liftul inainte sa porneasca. Noi am ales ca timpul pe care sa il astepte sa fie de 16 secunde, deoarece in acest timp de stationare se va parcurge memoria in componenta de alegere etaj, iar 16 secunde sunt necesare pentru a parcurge toata memoria (valorile de la o adresa pe secunda ). Enable primeste de la o componenta separata care verifica mai multe conditii pentru care numaratorul sa poata functiona (daca memoria e goala, senzorii, daca stationeaza). In momentul in care se termina numrarea secundelor, semnalul din interiorul arhitecturii care retine secundele se reinitializeaza, de la 0 la 15, iar semnalul de output „br” se activeaza. Atunci cand se reincepe numararea, iesirea „br” revine la valoarea 0. Aceasta iesire este folosita in comparator pentru a determina daca numaratorul mai stationeaza sau nu.

Singura variabila pe care o folosim este „sec”, care reprezinta numarul de secunde pe care il mai are de asteptat numaratorul.



**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**entity num\_indirect is**

**port (clk: in std\_logic;**

**enable: in std\_logic;**

**br : out std\_logic := '0');**

**end entity;**

**architecture comportamental of num\_indirect is**

**begin**

**process(clk, enable)**

**variable sec: std\_logic\_vector (3 downto 0) := "1111";**

**begin**

**if (clk'event and clk = '1') then**

**if enable = '1' then**

**if sec = "0000" then**

**sec := "1111";**

**br <= '1';**

**else**

**sec := sec - '1';**

**br <= '0';**

**end if;**

**end if;**

**end if;**

**end process;**

**end architecture;**

1. **Componenta de alegere a etajului**

Componenta de alegere etaj este cea raspunzatoare de alegerea etajului urmator. Acest algoritm este impartit in 2 parti de alegere: cea din timpul stationarii si cea din timpul functionarii liftului, in partea de citire a comenzilor, si in partea de stergere a comenzilor. Daca memoria nu are nici o comanda atunci nu intra in niciunul dintre aceste parti ale algoritmului. De asemenea aceasta componenta mai returneaza urmatorul etaj la care urma sa mearga liftul inainte de stationare, o informatie foarte pretioasa pentru comparator.

**Citirea comenzilor**

Se introduc in memorie, numai pe impuls de clock si cu semnalul de citire OK, datele primite de la intrarea: Int, Exte, Up, Down, la adresa determinata de „etaj\_comanda”.

**Stergerea comenzilor**

Se sterg comenzile din memorie, adica toate valorile de la adresa respectiva devin 0, numai pe impuls de clock si la semnal de stergere (egal), pe care il primeste de la comparator, adica atunci cand etajul curent este egal cu etajul ales. Se sterg toate valorile, deoarece daca o comanda din exterior a fost facuta in sus si in jos de la acelasi etaj, iar liftul se duce in sus, utlizatorul care vrea sa se duca in jos nu va stii asta, si atunci presupunem ca, in mod firesc si natural, va intra si el.

**Verificare existenta comenzi**

Iesirea „mem\_goala” este initializat cu 1, deoarece in starea initiala nu are nici o coamanda, iar in momentul in care primeste o comanda se face 0. De asemenea, devine inapoi 1 atunci cand nu mai exista nici o comanda in memorie. Este nevoie de asemenea informatie despre starea memoriei si pentru celelalte procese din cadrul arhitecturii asa ca avem nevoie si de un semnal „s\_goala” pe care sa il putem folosi (nu se pot citi date de pe iesiri si de aceea avem nevoie de el).

**Partea algoritmului care se executa in timpul stationarii**

In timpul stationarii, pe fiecare impuls de tact parcurgem cate o adresa din memoria in care retinem comenzile . Avem o variabila i, care se duce la valorile mai mari decat cea a etajului curent (de ex daca suntem la etajul 6, pe primul tact verifica comenzile de la etajul 7, la urmatorul impuls de tact, de la etajul 8 si asa mai departe), si cu o variabila j parcurgem la valorile mai mici decat valoare etajului curent. Cele 2 variabile se opresc in momentul in care au ajuns la 12 respectiv 0.

Avem 4 semnale care o sa retine etajul la care este cea mai apropiata comanda de deasupra etajului care e facuta din interior sau din exterior si se duce in sus, de deasupra etajului din exterior si vrea sa se duca in jos, de dedesubt si comanda e facuta din interior sau din exterior si se duce in jos, si din exterior si vrea sa se duca in sus(etaj\_i\_sus\_int, etaj\_i\_jos, etaj\_j\_jos\_int, etaj\_j\_sus, i se refera la etaje de deasupta etajului curent, iar j la etaje de dedesubtul lui).

Apoi in functie de sens se alege cea mai buna varianta dintre acestea 4. Daca nu exista o comanda pentru unul din semnal, acesta va ramane 15. Avem nevoie de comenzile de acest tip care sunt cele mai apropiate de etajul curent asa ca imediat ce s-a gasit un etaj favorabil, valoarea semnalului corespunzator cu acea comanda se schimba (de exemplu daca suntem la 5 si este comanda la 8 int sus, etaj\_i\_sus\_int primeste 8). Daca un astfel de semnal, de exemplu etaj\_i\_sus\_int nu este 15, nu isi mai schimba valoarea intrucat el deja a gasit cea mai apropiata comanda.

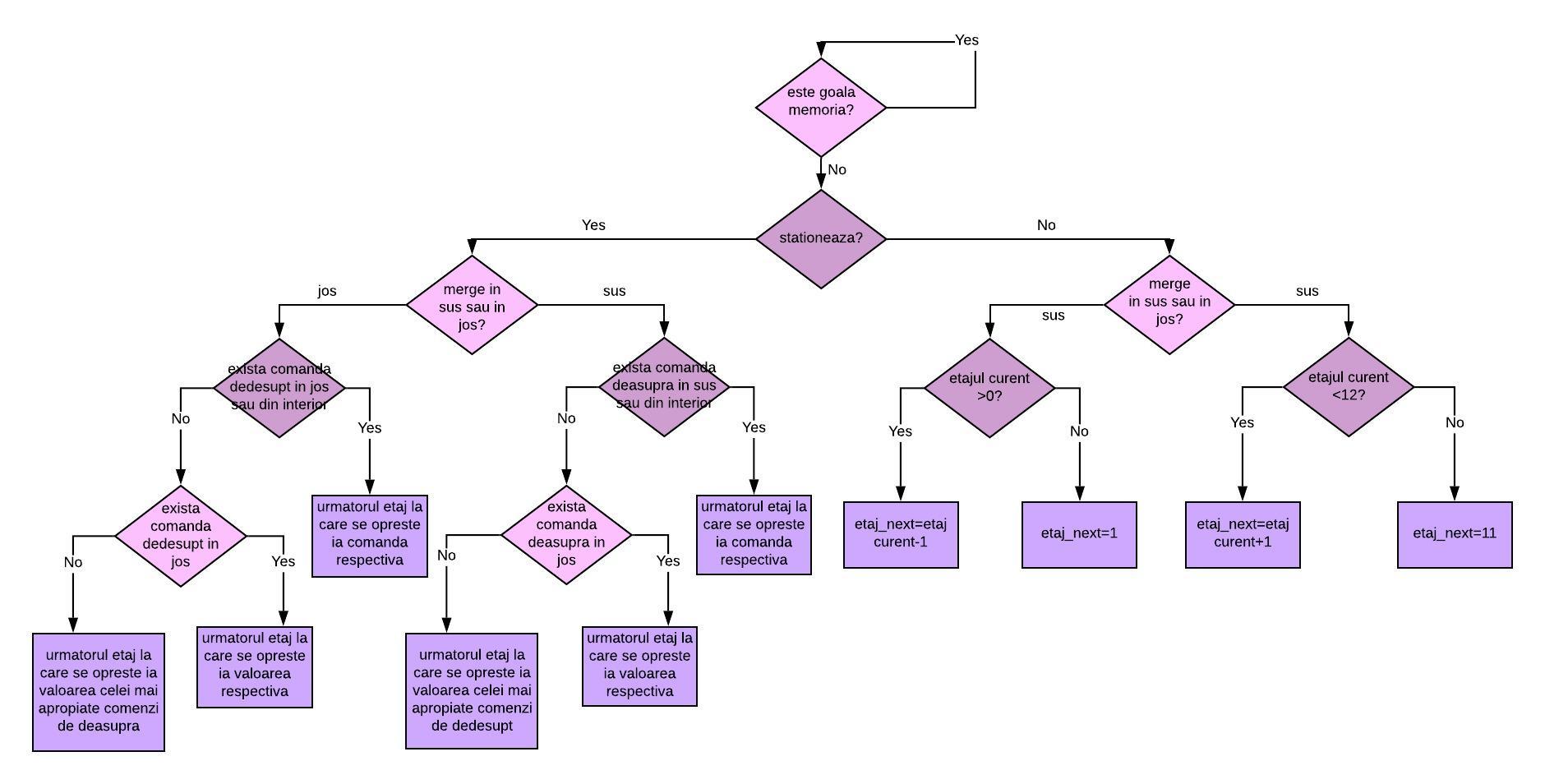
In cazul in care liftul se duce in sus cea mai buna varianta ar fi cea mai apropiata comanda de la un etaj mai mare decat etajul curent care sa fie facuta in sus sau din interior. Daca nu exista o astfel de comanda se verifica daca este vreo comanda tot de deasupra etajului curent dar care vrea sa mearga in jos, iar daca nu se gaseste acest caz se ia cea mai apropiata comanda de la un etaj mai mic decat etajul curent. La fel este si alegerea daca sensul este in jos, numai ca schimbam si sensurile in care se cheama etajele.

**Partea algoritmului care se executa cand nu stationeaza**

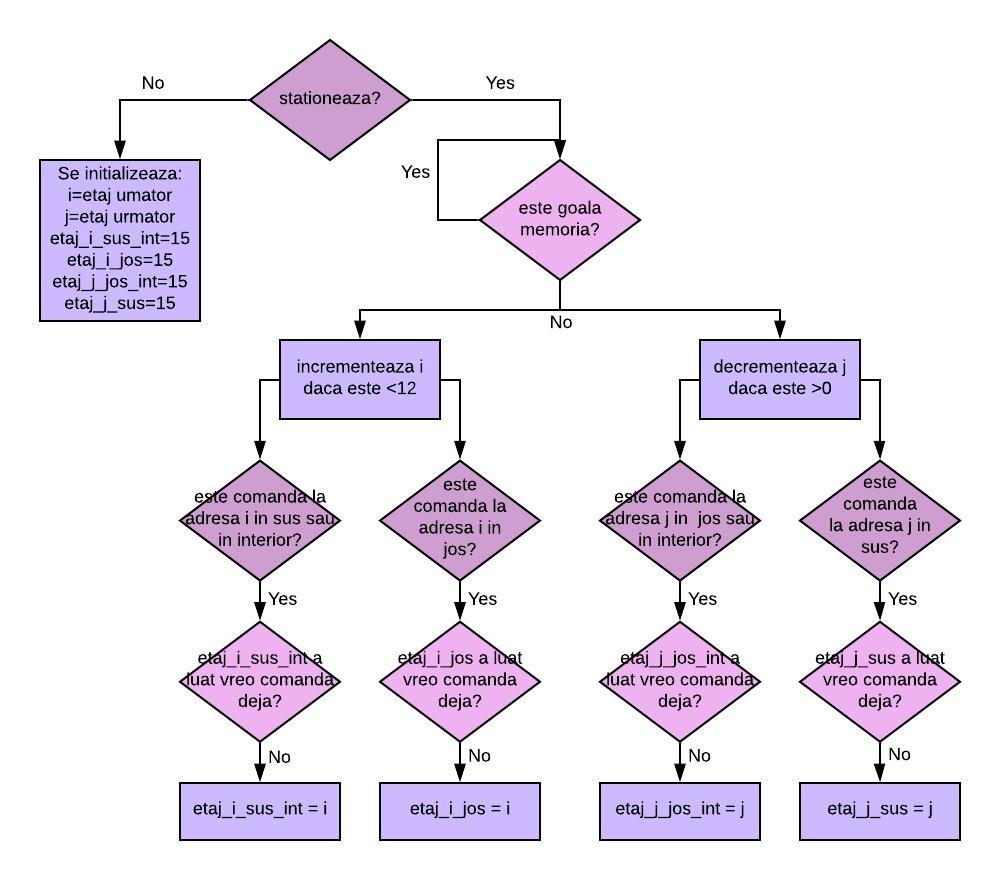
Exista unele cazuri in care se mai introduc comenzi care ar fi mai avantajoase in timpul functionarii liftului, de exemplu daca ne ducem de la etajul 2 spre 8 si primim comanda de la 5 din ext in sus, trebuie sa se opreasca mai intai la 5. De aceea in aceasta parte a algoritmul verificam etajul urmator (cel cu o pozitie mai sus daca liftul merge in sus, si cel cu o pozitie mai jos daca liftul merge in jos). Daca acesta are o comanda avantajoasa „etaj\_ales” ia valoarea acelui etaj.

Toate aceste sarcini ale algoritmului le-am impartit in 4 procese concurente: procesul de citire si de stergere, procesul de deteminare daca memoria este goala, procesul de cautare a comenzilor, si procesul de actualizare al etajului ales.

**Flowchart : update (actualizarea „etaj\_ales”)**



**Flowchart: cautam\_stationand**

****

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**use IEEE.STD\_LOGIC\_ARITH.ALL;**

**use IEEE.STD\_LOGIC\_UNSIGNED.ALL;**

**entity alegere\_etaj\_nou is**

**port (clk : in std\_logic;**

**etaj\_curent : in std\_logic\_vector (3 downto 0) := "0000";**

**etaj\_comanda : in std\_logic\_vector (3 downto 0);**

**int, exte, up, down : in std\_logic; --- up down ext int**

**ok : in std\_logic;**

**egal : in std\_logic;**

**stationare : in std\_logic := '1';**

**sens : in std\_logic := '1';**

**mem\_goala : out std\_logic := '1';**

**etaj\_urmator : out std\_logic\_vector (3 downto 0) := "0000";**

**p\_comp : out std\_logic\_vector (3 downto 0) := "0001");**

**end entity;**

**architecture comportamental of alegere\_etaj\_nou is**

**type mem is array(0 to 12) of std\_logic\_vector(3 downto 0);**

**signal RAM: mem :=**

**(x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0",x"0");**

**signal s\_goala : std\_logic := '1';**

**signal etaj\_i\_sus\_int : std\_logic\_vector (3 downto 0) := "1111";**

**signal etaj\_i\_jos : std\_logic\_vector (3 downto 0) := "1111";**

**signal etaj\_j\_sus : std\_logic\_vector (3 downto 0) := "1111";**

**signal etaj\_j\_jos\_int : std\_logic\_vector (3 downto 0) := "1111";**

**signal etaj\_next : std\_logic\_vector(3 downto 0) := "0001";**

**begin**

**citire\_si\_stergere\_mem : process(clk, egal, ok, etaj\_curent, etaj\_comanda, RAM)**

**begin**

**if (clk'event and clk = '1') then**

**--- stergere din memorie daca s-a ajuns la etaj**

**if (egal = '1') then**

**RAM (conv\_integer(etaj\_curent)) <= "0000";**

**end if;**

**--- introducere comanda in memorie**

**if (ok = '1') then**

**if (up = '1' and exte = '1') then**

**RAM (conv\_integer(etaj\_comanda))(3) <= '1';**

**RAM (conv\_integer(etaj\_comanda))(1) <= '1';**

**end if;**

**if (down = '1' and exte = '1') then**

**RAM (conv\_integer(etaj\_comanda))(2) <= '1';**

**RAM (conv\_integer(etaj\_comanda))(1) <= '1';**

**end if;**

**if (int = '1') then**

**RAM (conv\_integer(etaj\_comanda))(0) <= '1';**

**end if;**

**end if;**

**end if;**

**end process;**

**verif\_mem\_goala : process (clk, RAM)**

**variable goala: std\_logic\_vector (3 downto 0) := "0001";**

**begin**

**if (clk'event and clk = '1') then**

**goala := RAM(0) or RAM(1) or RAM(2) or RAM(3) or RAM(4) or RAM(5) or RAM(6) or RAM(7) or RAM(8) or RAM(9) or RAM(10) or RAM(11) or RAM(12);**

**if (goala = "0000") then**

**mem\_goala <= '1';**

**s\_goala <= '1';**

**else**

**mem\_goala <= '0';**

**s\_goala <= '0';**

**end if;**

**end if;**

**end process;**

**update : process (clk, RAM, s\_goala, stationare, sens, etaj\_i\_sus\_int, etaj\_i\_jos, etaj\_j\_sus, etaj\_j\_jos\_int, etaj\_curent)**

**variable lvl : std\_logic\_vector (3 downto 0) := "0000";**

**variable nextt: std\_logic\_vector (3 downto 0) := "0001";**

**begin**

**if (clk'event and clk = '1') then**

**if (s\_goala = '0') then**

**if (stationare = '0') then**

**if (sens = '1') then**

**if etaj\_curent < "1100" then**

**etaj\_next <= etaj\_curent + '1';**

**nextt:=etaj\_curent + '1';**

**else**

**etaj\_next <= "1011";**

**nextt := "1011";**

**end if;**

**if (RAM(conv\_integer(nextt))(3) = '1' or RAM(conv\_integer(nextt))(0) = '1') then**

**lvl := nextt;**

**end if;**

**else**

**if (etaj\_curent > "0000") then**

**etaj\_next <= etaj\_curent - '1';**

**nextt := etaj\_curent - '1';**

**else**

**etaj\_next <= "0001";**

**nextt := "0001";**

**end if;**

**if (RAM(conv\_integer(nextt))(2) = '1' or RAM(conv\_integer(nextt))(0) = '1') then**

**lvl := nextt;**

**end if;**

**end if;**

**else**

**if (sens = '1') then**

**if (etaj\_i\_sus\_int /= "1111") then**

**lvl := etaj\_i\_sus\_int;**

**elsif (etaj\_i\_jos /= "1111") then**

**lvl := etaj\_i\_jos;**

**else**

**if (etaj\_j\_jos\_int /= "1111" and etaj\_j\_sus = "1111") then**

**lvl := etaj\_j\_jos\_int;**

**elsif (etaj\_j\_jos\_int = "1111" and etaj\_j\_sus /= "1111") then**

**lvl := etaj\_j\_sus;**

**elsif (etaj\_j\_jos\_int /= "1111" and etaj\_j\_sus /= "1111") then**

**if (etaj\_j\_jos\_int > etaj\_j\_sus) then**

**lvl := etaj\_j\_jos\_int;**

**elsif (etaj\_j\_jos\_int < etaj\_j\_sus) then**

**lvl := etaj\_j\_sus;**

**else**

**lvl := etaj\_j\_sus;**

**end if;**

**end if;**

**end if;**

**else**

**if (etaj\_j\_jos\_int /= "1111") then**

**lvl := etaj\_j\_jos\_int;**

**elsif (etaj\_j\_sus /= "1111") then**

**lvl := etaj\_j\_sus;**

**else**

**if (etaj\_i\_sus\_int /= "1111" and etaj\_i\_jos = "1111") then**

**lvl := etaj\_i\_sus\_int;**

**elsif (etaj\_i\_sus\_int = "1111" and etaj\_i\_jos /= "1111") then**

**lvl := etaj\_i\_jos;**

**elsif (etaj\_i\_sus\_int /= "1111" and etaj\_i\_jos /= "1111") then**

**if (etaj\_i\_sus\_int < etaj\_i\_jos) then**

**lvl := etaj\_i\_sus\_int;**

**elsif (etaj\_i\_sus\_int > etaj\_i\_jos) then**

**lvl := etaj\_i\_jos;**

**else**

**lvl := etaj\_i\_jos;**

**end if;**

**end if;**

**end if;**

**end if;**

**end if;**

**etaj\_urmator <= lvl;**

**else**

**etaj\_urmator <= etaj\_curent;**

**end if;**

**end if;**

**p\_comp <= nextt;**

**end process;**

**cautam\_stationand : process (clk, RAM, s\_goala, stationare, etaj\_i\_sus\_int, etaj\_i\_jos, etaj\_j\_sus, etaj\_j\_jos\_int, etaj\_curent)**

**variable i : integer range 0 to 12 := 0;**

**variable j : integer range 0 to 12 := 0;**

**begin**

**if (clk'event and clk = '1') then**

**if (stationare = '0') then**

**i := conv\_integer(etaj\_next);**

**j := conv\_integer(etaj\_next);**

**etaj\_i\_sus\_int <= "1111";**

**etaj\_i\_jos <= "1111";**

**etaj\_j\_sus <= "1111";**

**etaj\_j\_jos\_int <= "1111";**

**else**

**if (s\_goala = '0') then**

**if (i < 12) then**

**i := i + 1;**

**else**

**i := 12;**

**end if;**

**if (j > 0) then**

**j := j - 1;**

**else**

**j := 0;**

**end if;**

**if (i < 13) then**

**if (RAM(i)(3) = '1' or RAM(i)(0) = '1') then**

**if etaj\_i\_sus\_int = "1111" then**

**etaj\_i\_sus\_int <= conv\_std\_logic\_vector(i, 4);**

**end if;**

**end if;**

**if (RAM(i)(2) = '1') then**

**if etaj\_i\_jos = "1111" then**

**etaj\_i\_jos <= conv\_std\_logic\_vector(i, 4);**

**end if;**

**end if;**

**end if;**

**if (j > - 1) then**

**if (RAM(j)(3) = '1' or RAM(j)(0) = '1') then**

**if etaj\_j\_jos\_int = "1111" then**

**etaj\_j\_jos\_int <= conv\_std\_logic\_vector(j, 4);**

**end if;**

**end if;**

**if (RAM(j)(2) = '1') then**

**if etaj\_j\_sus = "1111" then**

**etaj\_j\_sus <= conv\_std\_logic\_vector(j, 4);**

**end if;**

**end if;**

**end if;**

**end if;**

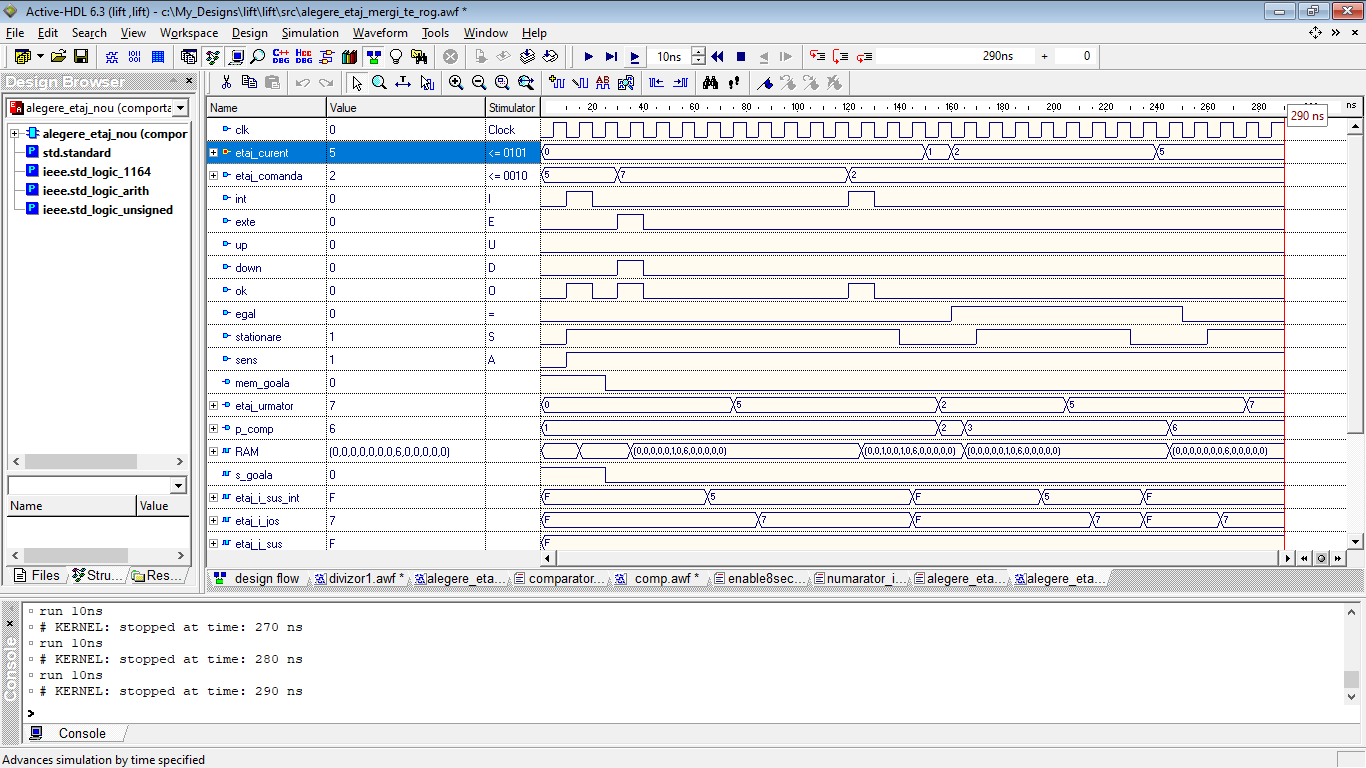
**end if;**

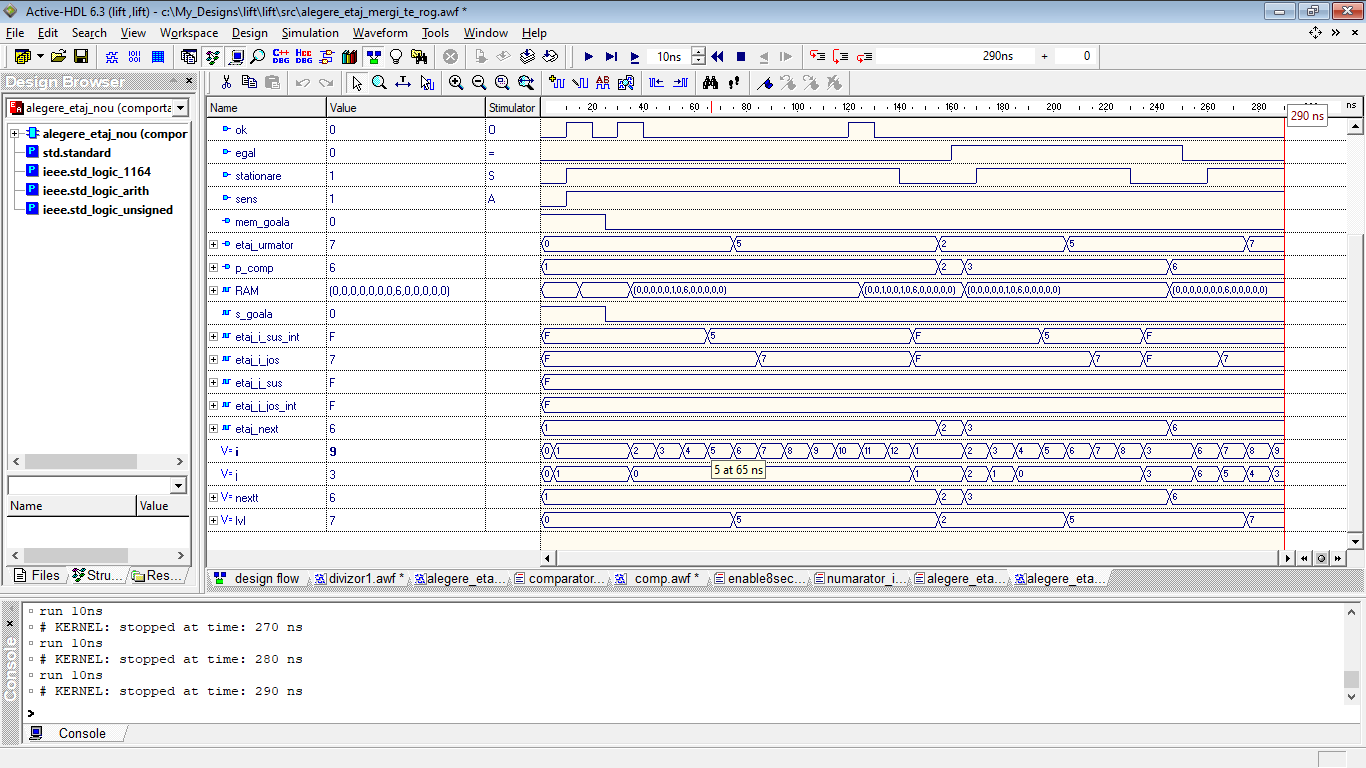
**end if;**

**end process;**

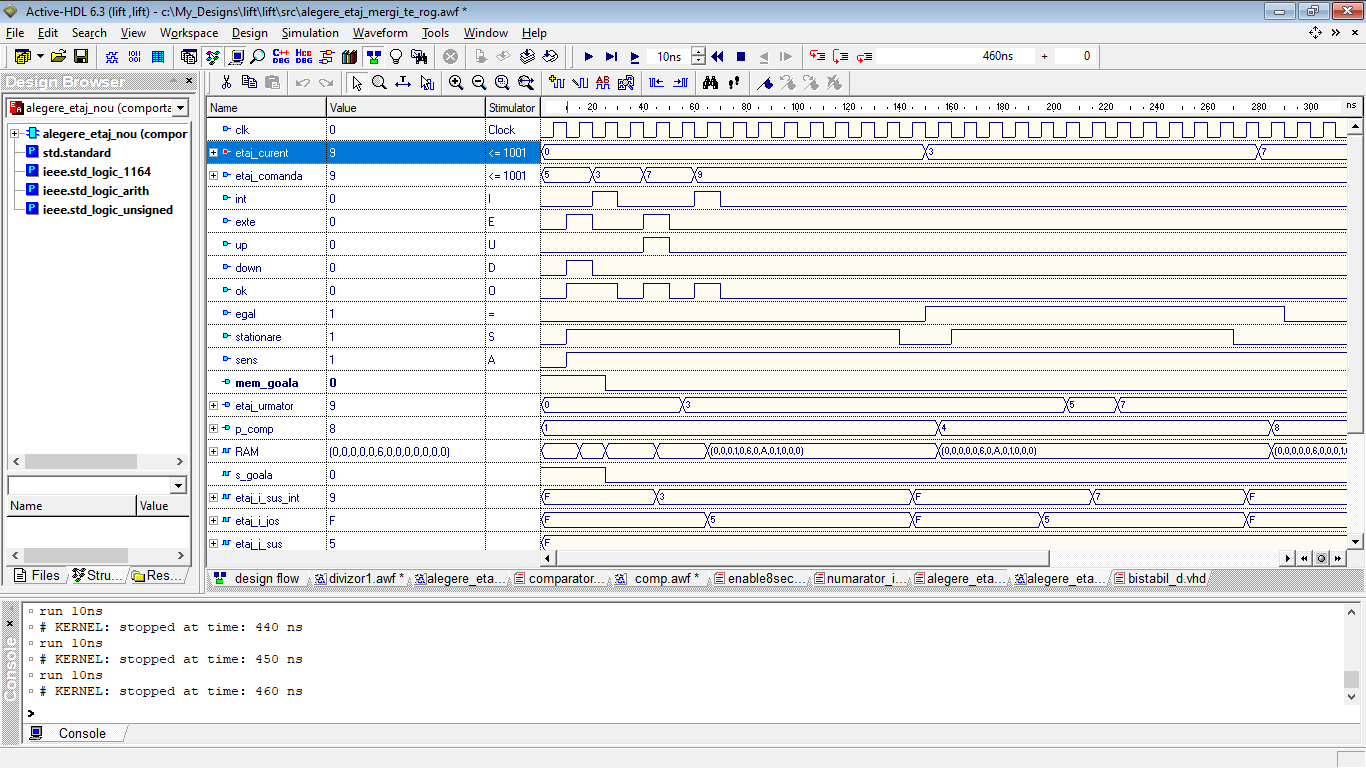
**end architecture;**

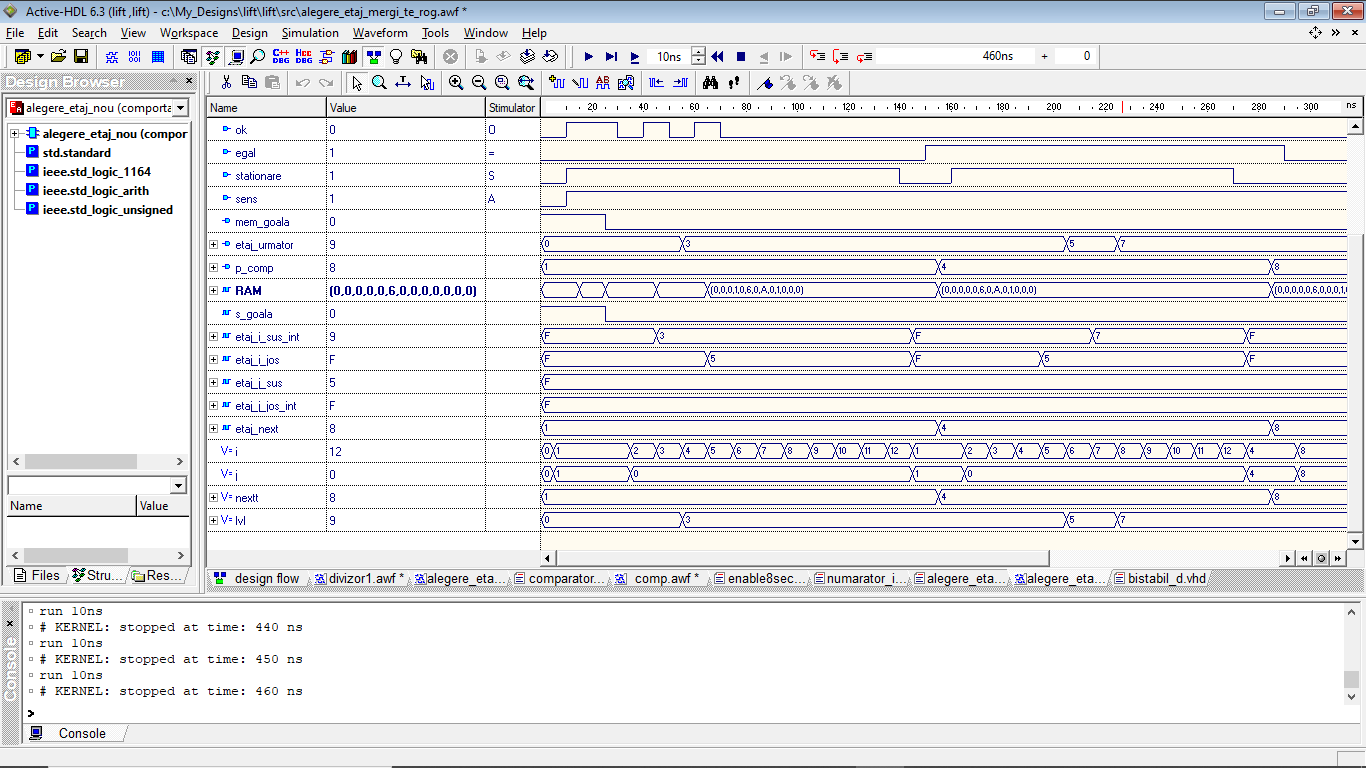
**Simulare I (introducem o comanda la un etaj dupa ce variabila i a trecut de valoarea acelui etaj – alege acel etaj in timp ce nu stationeaza)**





**Simulare II (sensul este in sus si nu ia prima oara comenzile care au sensul in jos)**



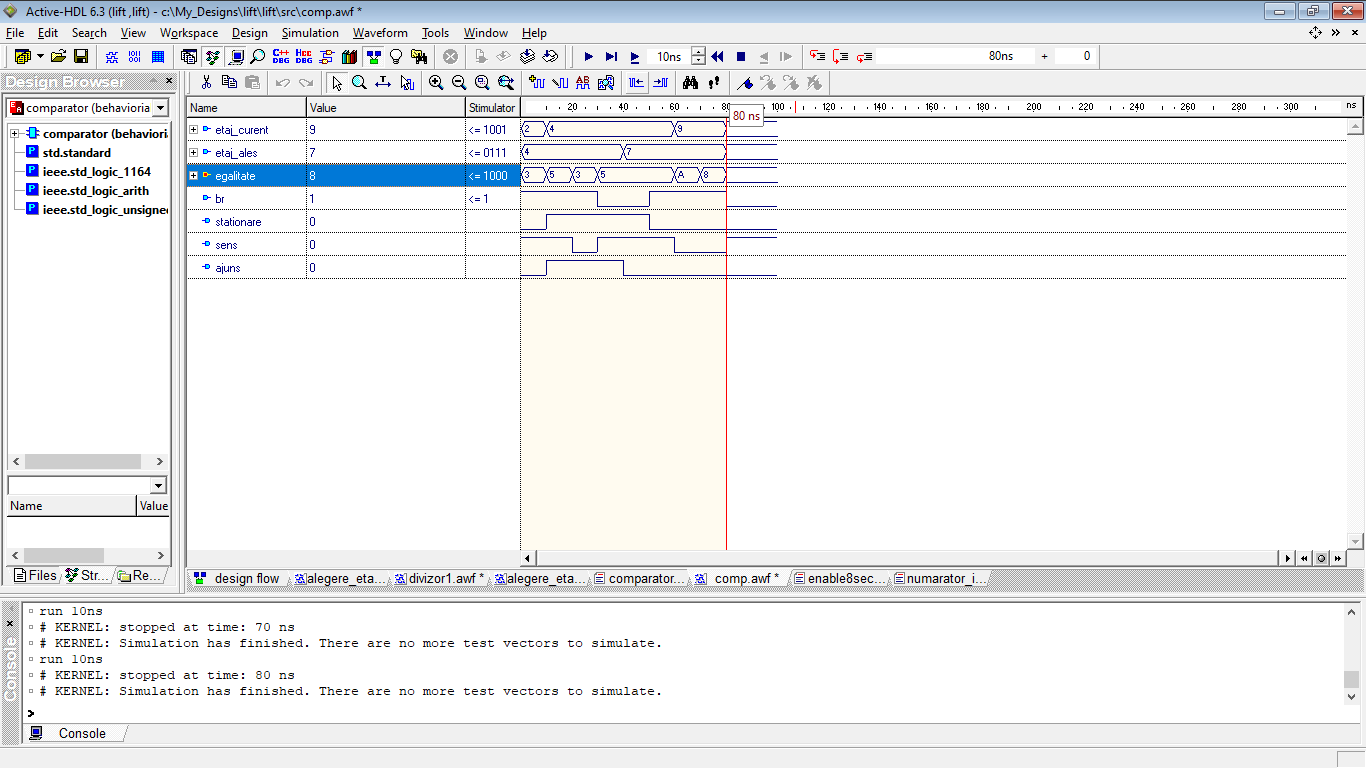


1. **Comparator etaj curent-etaj ales**

Aceasta componenta compara „etajul curent” cu „etajul ales”. Acesta determina sensul, in sus sau in jos (0 daca se duce in jos, si 1 daca se duce in sus) , daca a ajuns la „etajul ales” („egal” ia valoarea 1) si daca stationeaza sau nu. In momentul in care a ajuns la etajul ales „stationare” devine 1, iar in momentul in care primeste semnal de la numaratorul indirect ca s-a terminat numararea secundelor („br” este semnalul care trimite aceasta informatie) „stationare” devine inapoi 0, si va putea sa isi inceapa functionarea si numaratorul pentru etajul curent.

In momentul in care etajul ales este egal cu etajul curent, nu se poate alege un sens exact, sus sau jos, asa ca mai avem nevoie de o intrare care provine de la componenta de alegere a etajului, care ne indica urmatorul etaj la care urma sa se duca liftul inainte sa inceapa stationarea.

Iesirea ajuns ia valoarea 1 cand etajul curent si etajul ales au aceeasi valoarea pentru a trimite aceasta informatie catre componenta de alegere etaj pentru a sterge din memorie comanda la etajul curent.



**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**use IEEE.STD\_LOGIC\_ARITH.ALL;**

**use IEEE.STD\_LOGIC\_UNSIGNED.ALL;**

**entity comparator is**

**port(etaj\_curent: in std\_logic\_vector(3 downto 0) := "0000";**

**etaj\_ales: in std\_logic\_vector(3 downto 0);**

**br: in std\_logic;**

**egalitate : in std\_logic\_vector (3 downto 0) := "0001";**

**-- este etajul urmator pentru a next da seama de sens cand etajele sunt egale**

**stationare: out std\_logic := '1';**

**sens: out std\_logic := '1';**

**ajuns: out std\_logic := '1');**

**end entity;**

**architecture behaviorial of comparator is**

**begin**

**process(etaj\_curent, etaj\_ales, br, egalitate)**

**variable s: std\_logic := '1';**

**begin**

**if (etaj\_curent = etaj\_ales) then**

**stationare <= '1';**

**elsif ( etaj\_curent /= etaj\_ales and br = '0') then**

**stationare <= '1';**

**else**

**stationare <= '0';**

**end if;**

**if (etaj\_curent = etaj\_ales) then**

**ajuns <= '1';**

**else**

**ajuns <= '0';**

**end if;**

**if (etaj\_curent > etaj\_ales) then**

**sens <= '0';**

**elsif (etaj\_curent < etaj\_ales) then**

**sens <= '1';**

**else**

**if (egalitate < etaj\_curent) then**

**sens <= '0';**

**else**

**sens <= '1';**

**end if;**

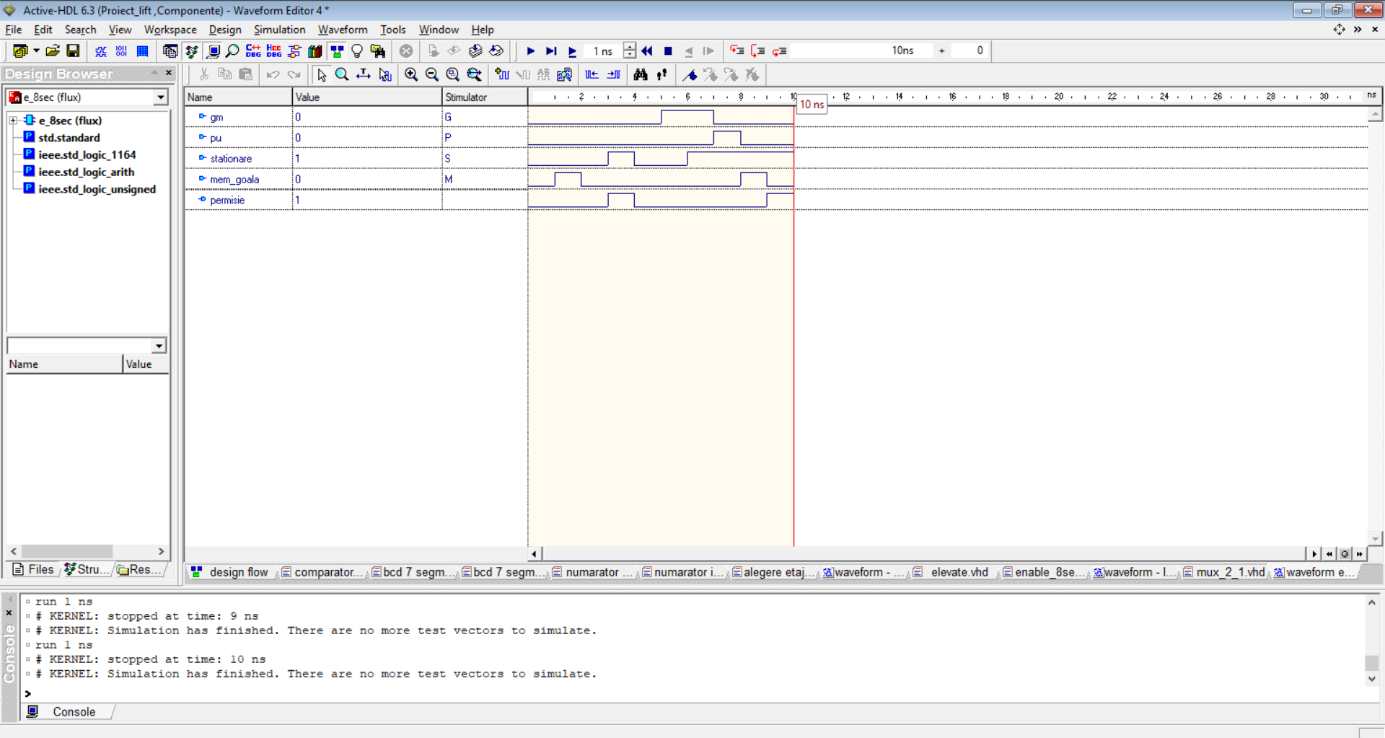
**end if;**

**end process;**

**end architecture;**

1. **Enable numarator indirect**

Pentru ca tot procesul sa inceapa, adica functionarea numaratorului indirect, si, automat apoi si numaratorul pentru etaj curent, trebuie indeplinite niste conditii. In primul rand, trebuie ca memoria sa nu fie goala, semnal pe care il preia de la componenta de alegere a urmatorului etaj. De asemenea, pentru fuctionarea numaratorului trebuie ca cei 2 senzori sa nu fie activi, adica sa nu se depaseasca greutatea si sa nu fie o persoana in usa. „Pu” reprezinta senzorul care arata daca este vreo persoana in usa si este activ pe 1, „GM” este senzorul care arata daca s-a depasit greutate maxima, „meme\_goala” este activ cand memoria este goala, iar stationare este activ cand liftul stationeaza.



**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**entity e\_8sec is**

**port (gm : in std\_logic;**

**pu : in std\_logic;**

**stationare : in std\_logic;**

**mem\_goala : in std\_logic;**

**permisie : out std\_logic);**

**end entity;**

**architecture flux of e\_8sec is**

**begin**

**permisie <= not gm and not pu and not mem\_goala and stationare;**

**end architecture;**

1. **Afisoare 7 segmente**

Cele 2 afisoare primesc date de la numaratorul etaj curent si afisor I, afiseaza cifra unitatilor a etajului, iar II cifra zecilor, pt etajele sub 10, afisor 2 nu va afisa nimic, nu va fi activ nici un segment. Deoarece nu pot fi activi 2 anozi in acelasi timp trebuie sa schimbam la o frecventa mare anodul care o sa fie activ si valoarea pe care o iau catozii corespunzatoare cu anodul care este activ (de zeci sau unitati). Pentru a schimba anodul activ vom folosi un fel de registru inel pe doi biti, care primeste impuls de tact de la frecventa divizata (de la 100MHz la 150Hz) pentru a crea o iluzie optica, astfel incat sa para ca sunt ambele afisoare aprinse. Daca se schimba destul de repede, ochiul uman nu poate sesiza asta. Semnalul „unitati” reprezinta cum vor fi aprinsi catozii pentru cifra unitatilor a etajului, si „zeci” pentru cifra zecilor.

Semnalul „count” ne va ajuta sa divizam frecventa pe care o primim de la placa si de fiecare data cand se schimba bitul cel mai semnificativ vom aprinde celalalt anod. Iesirea „an” reprezinta anozii, iesirea „cat” reprezinta catozii.

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**use IEEE.STD\_LOGIC\_UNSIGNED.ALL;**

**use IEEE.STD\_LOGIC\_ARITH.ALL;**

**entity afisoare is**

**port(clock: in std\_logic;**

**etaj: in std\_logic\_vector (3 downto 0);**

**an: out std\_logic\_vector (1 downto 0);**

**cat: out std\_logic\_vector (6 downto 0));**

**end entity;**

**architecture comportamental of afisoare is**

**signal zeci: std\_logic\_vector (6 downto 0);**

**signal unitati: std\_logic\_vector (6 downto 0);**

**signal count: std\_logic\_vector (15 downto 0) := "0000000000000000";**

**signal c: std\_logic:='0';**

**begin**

**process(etaj)**

**begin**

**case etaj is**

**when "0000"=> unitati<="0011000";---parter**

**zeci<="1111111";**

**when "0001"=> unitati<="1001111";---etaj 1**

**zeci<="1111111";**

**when "0010"=> unitati<="0010010";---etaj 2**

**zeci<="1111111";**

**when "0011"=> unitati<="0000110";---etaj 3**

**zeci<="1111111";**

**when "0100"=> unitati<="1001100";---etaj 4**

**zeci<="1111111";**

**when "0101"=> unitati<="0100100";---etaj 5**

**zeci<="1111111";**

**when "0110"=> unitati<="0100000";---etaj 6**

**zeci<="1111111";**

**when "0111"=> unitati<="0001111";---etaj 7**

**zeci<="1111111";**

**when "1000"=> unitati<="0000000";---etaj 8**

**zeci<="1111111";**

**when "1001"=> unitati<="0000100";---etaj 9**

**zeci<="1111111";**

**when "1010"=> unitati<="0000001";---etaj 10**

**zeci<="1001111";**

**when "1011"=> unitati<="1001111";---etaj 11**

**zeci<="1001111";**

**when "1100"=> unitati<="0010010";---etaj 12**

**zeci<="1001111";**

**when others=> unitati<="1111111";---null**

**zeci<="1111111";**

**end case;**

**end process;**

**divizare:process(clock)**

**begin**

**if(clock’event and clock = ’1’) then**

**count <= count+1;**

**end if;**

**end process;**

**c <= count(15);**

**anozi:process(c)**

**begin**

**if c = '0' then**

**an <= "01";**

**cat <= zeci;**

**else**

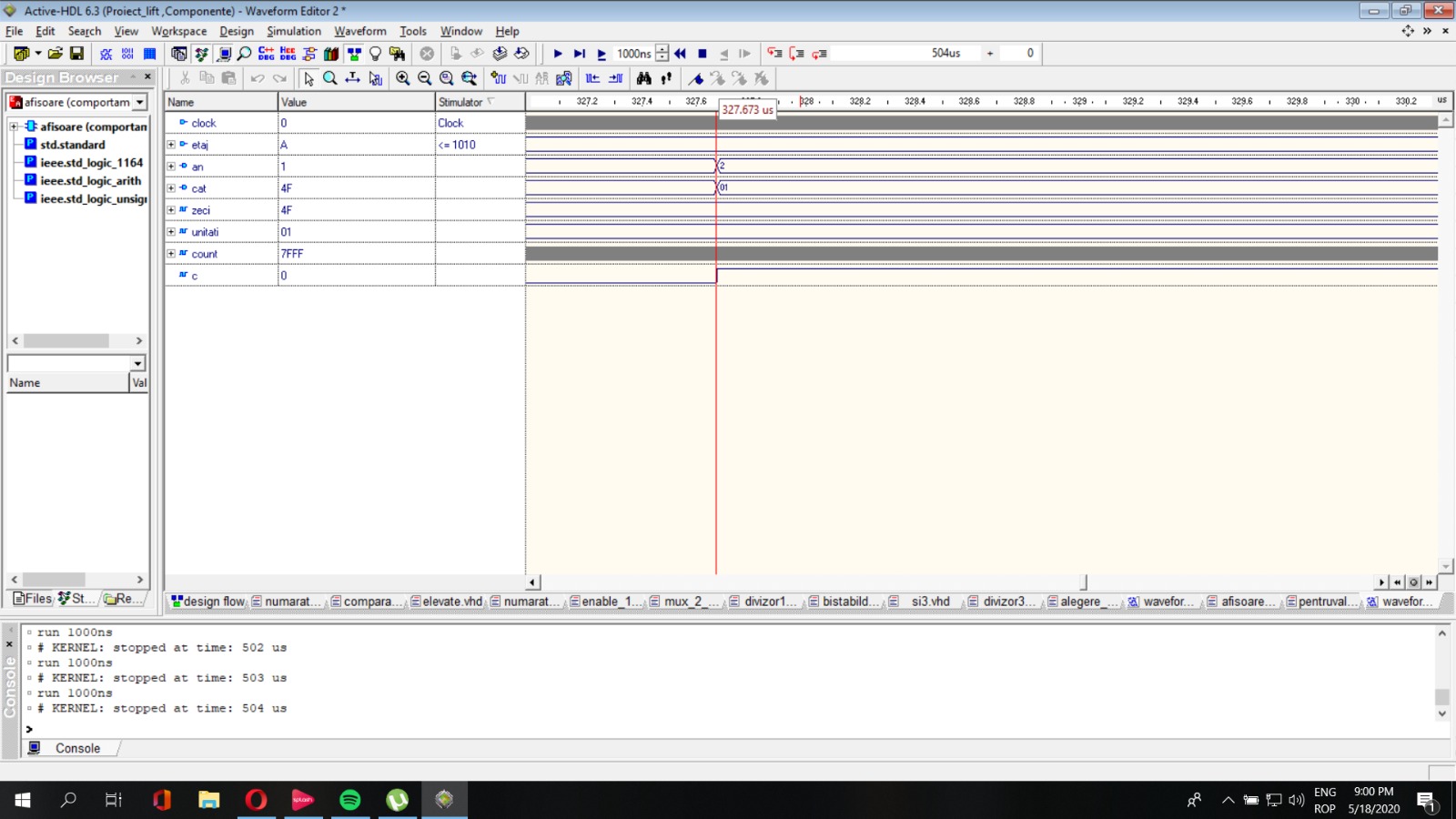
**an <= "10";**

**cat <= unitati;**

**end if;**

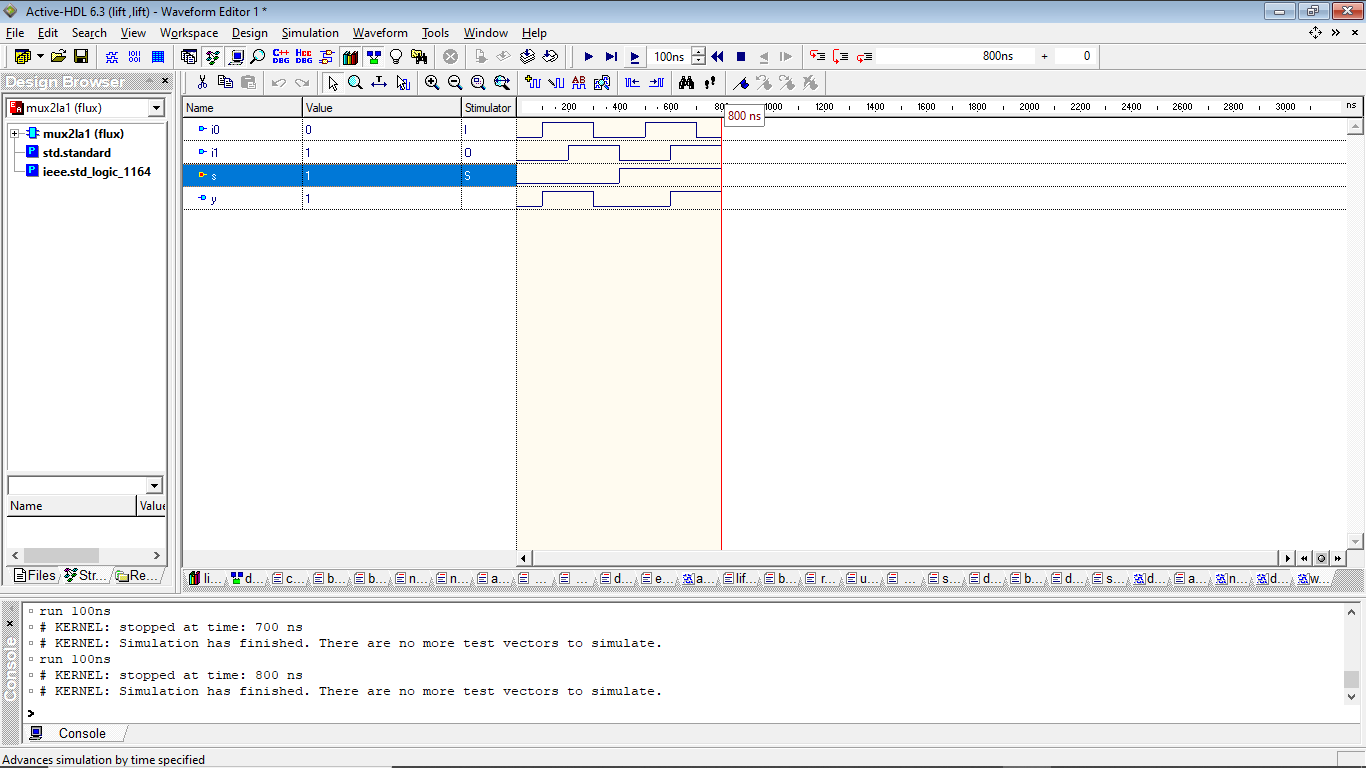
**end process;**

**end architecture;**



1. **Mux 2:1**

Acest multiplexor se ocupa cu alegerea semnalului de tact, de o secunda sau 3 secunde, in functie de alegerea utilizatorului, adica intrarea AC (alegere clock), iesirea este trimisa numai la numaratorul etaj curent. Cand Ac este 0 se va alege semnalul de tact de o secunda si cand este 1 se va alege semnalul de tact de 3 secunde.



**library ieee;**

**use ieee.std\_logic\_1164.all;**

**entity mux2la1 is**

**port(i0, i1, s:in std\_logic;**

**y: out std\_logic);**

**end entity;**

**architecture flux of mux2la1 is**

**begin**

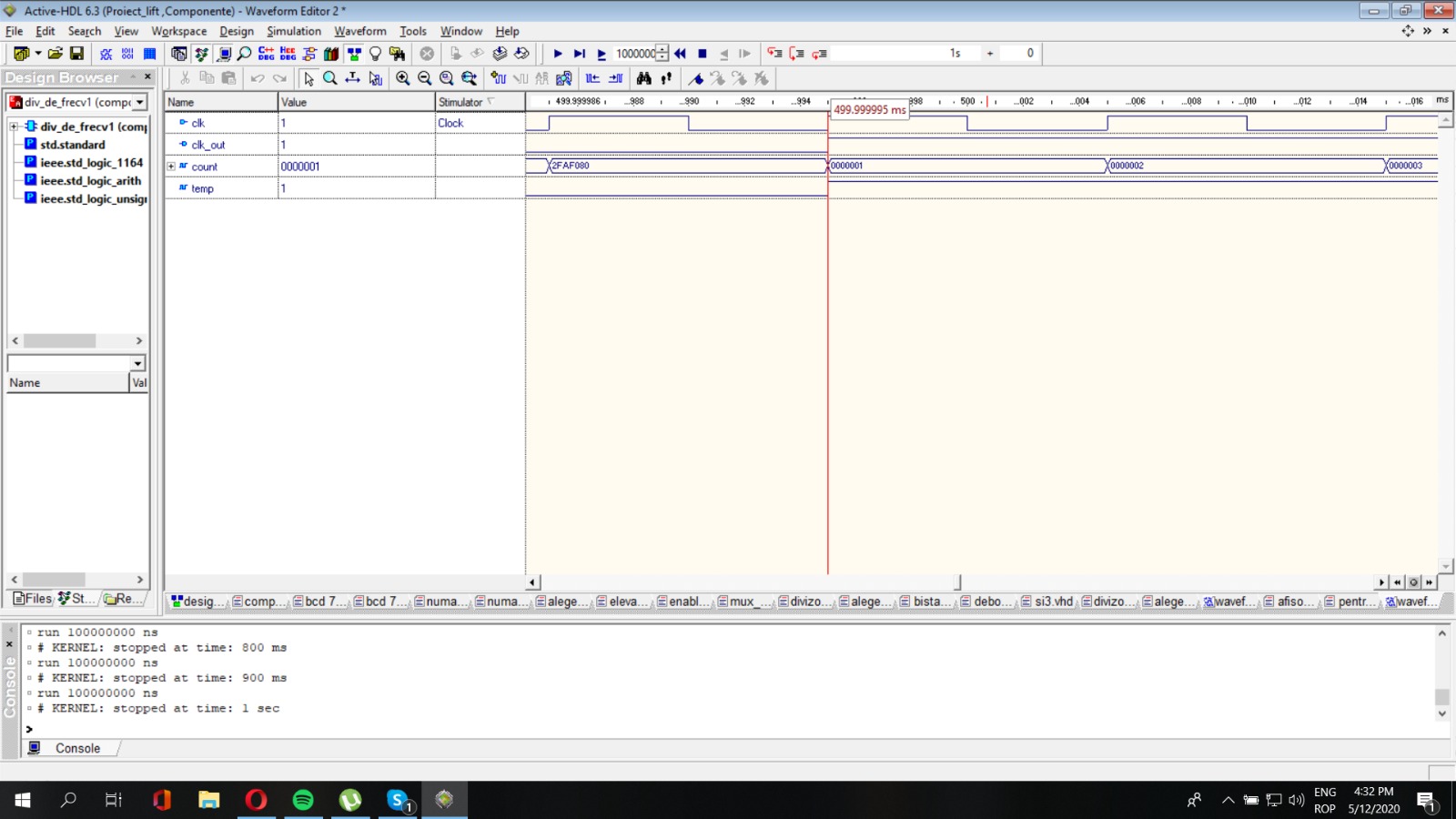
**y <= (i0 and (not s)) or (i1 and s);**

**end architecture;**

1. **Divizor de frecventa 1 secunda**

Pentru functionarea liftului avem nevoie sa generam un semnal de tact de o secunda, care va fi folosit pentru numaratorul indirect sau pentru numaratorul reversibil etaj curent atunci cand este aleasa posibilitatea de a merge cu o secunda pe etaj. Frecventa placii este de 100MHz si pentru a genera un semnal de tact de o secunda trebuie sa divizam frecventa la 1 HZ. Pentru a genera semnalul de tact mai trebuie sa divizam o data pentru ca jumatate din timp semnalul sa ia valoarea 0, iar cealalta jumtate sa ia valoarea 1.

Semnalul „count” numara pana la 50 000 000, iar apoi se face inapoi 1. „Temp” este semnalul care isi schimba valoarea in momentul in care „count” ajunge la 50 000 000. De aceea, el este 0 timp de 0.5 secunde si 1, timp de alte 0.5 secunde. Astfel el creeaza o forma de unda care descrie un semnal de tact de o secunda. Iesirea „clk\_out” ia valoarea lui „temp”.



**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.all;**

**entity div\_de\_frecv1 is**

**port(clk: in std\_logic;**

**clk\_out: out std\_logic);**

**end entity;**

**architecture comportamental of div\_de\_frecv1 is**

**signal count : std\_logic\_vector (25 downto 0) := "00000000000000000000000001";**

**signal temp : std\_logic := '0';**

**begin**

**process(clk, count, temp)**

**begin**

**if(clk'event and clk = '1') then**

**if(count = "10111110101111000010000000") then**

**temp <= not temp;**

**count <= "00000000000000000000000001";**

**else**

**count <= count + '1';**

**temp <= temp;**

**end if;**

**end if;**

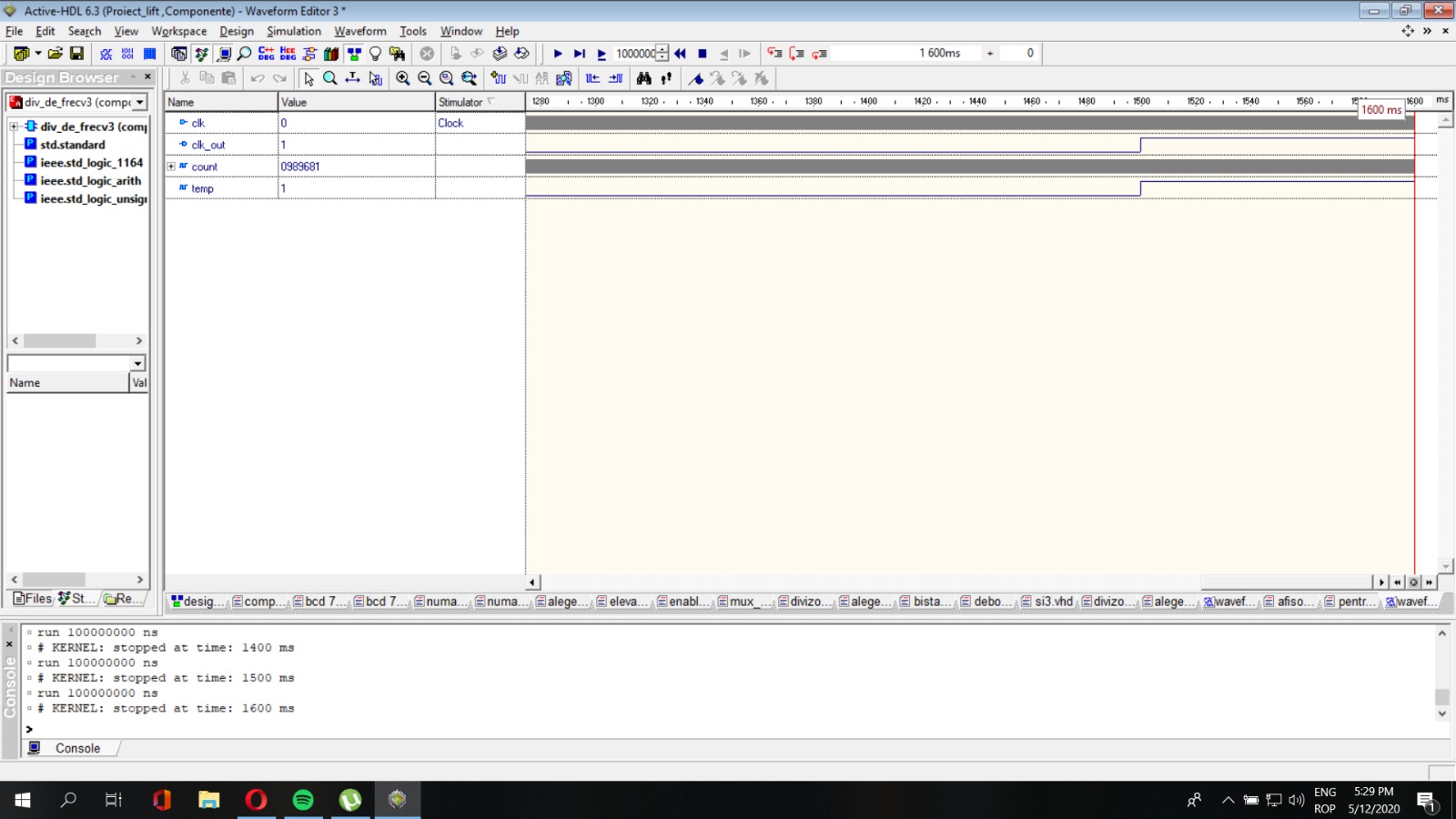
**clk\_out <= temp;**

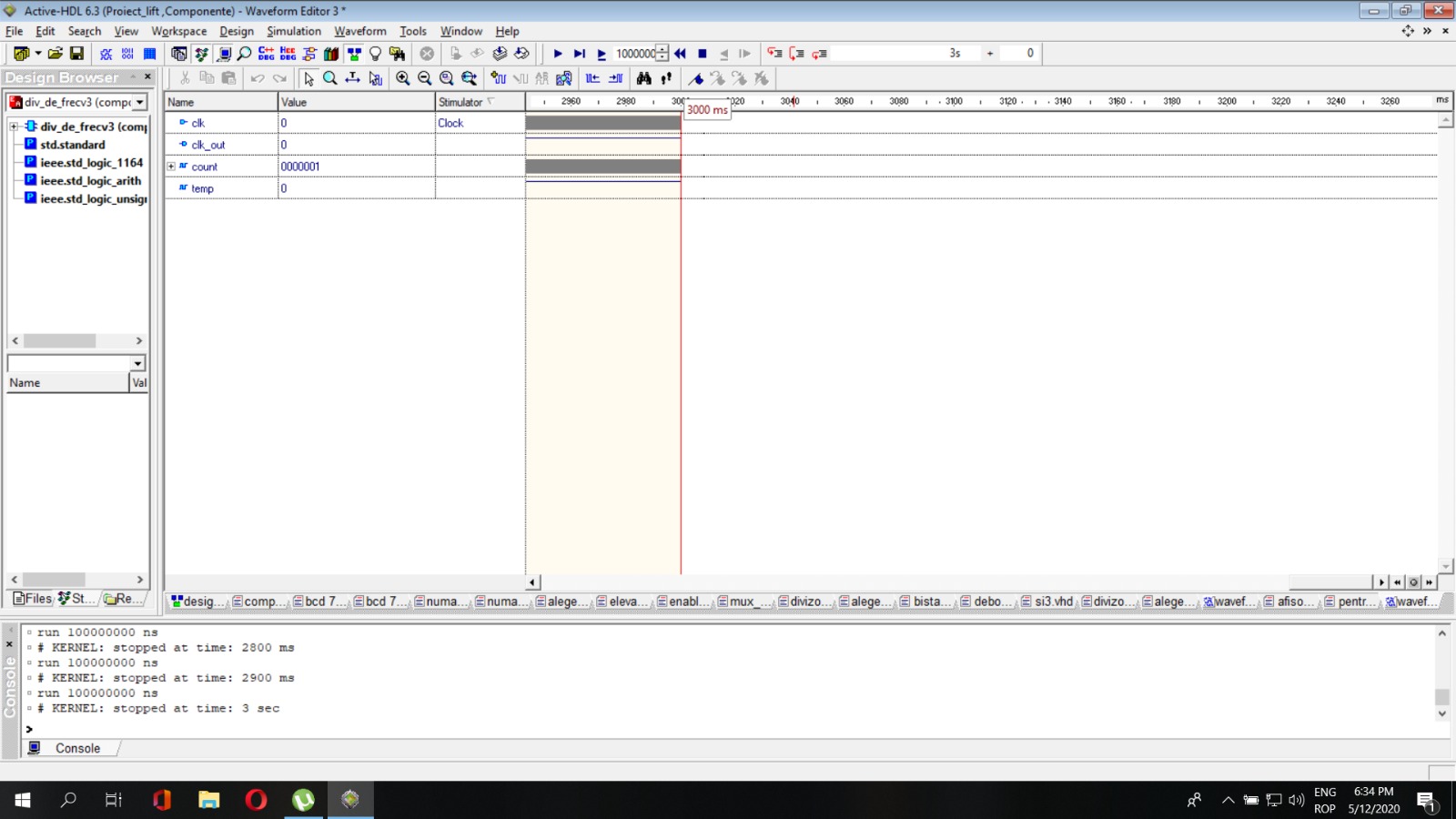
**end process;**

**end architecture;**

1. **Divizor de frecventa 3 secunde**

Acest divizor de frecventa se comporta exact la fel ca cel de o secunda, singura diferenta fiind faptul ca imparte frecventa la 1/6 Hz, pentru ca semnalul de tact sa aiba in total 3 secunde. 3/2 secunde semnalul ia valoarea 1 si 3/2 secunde ia valoarea 0. Semnalele sunt identice cu cele de la divizorul de frecventa de o secunda. Singura diferenta este ca pentru a diviza frecventa cum am explicat mai sus, „count” nu va creste numai pana la 50 000 000, ci pana la 150 000 000. „Temp” se va comporta exact la fel. Iesirea „clk\_out” ia valoarea lui „temp”.





**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.all;**

**entity div\_de\_frecv3 is**

**port(clk: in std\_logic;**

**clk\_out: out std\_logic);**

**end entity;**

**architecture comportamental of div\_de\_frecv3 is**

**signal count : std\_logic\_vector (27 downto 0) := "0000000000000000000000000001";**

**signal temp : std\_logic := '0';**

**begin**

**process(clk, count, temp)**

**begin**

**if(clk'event and clk = '1') then**

**if(count = "** **1000111100001101000110000000‬") then**

**temp <= not temp;**

**count <= "0000000000000000000000000001";**

**else**

**count <= count + '1';**

**temp <= temp;**

**end if;**

**end if;**

**clk\_out <= temp;**

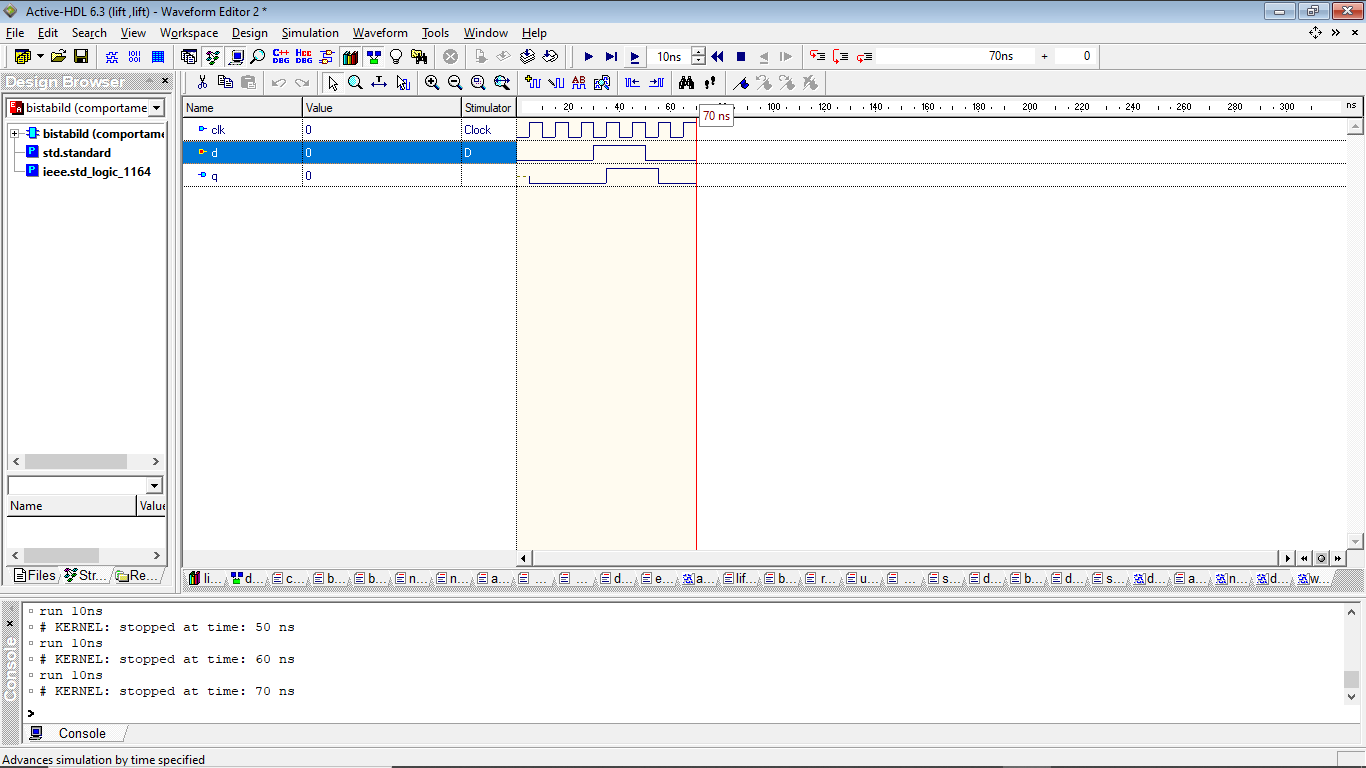
**end process;**

**end architecture;**

1. **Bistabil D (debouncer)**

Acesta este un bistabil delay, iesirea ia valoarea intrarii „D” dupa o secunda. Aceste primeste semnal de tact de la divizor de frecventa de o secunda. Acest bistabil d functioneaza ca un debouncer pentru butoane, adica verifica daca butonul este apasat timp de o secunda.

Acest bistabil mai are si o alta intrebuintare: pentru a functiona corect componenta de alegere etaj; pentru ca liftul a ajuns la etajul ales prima oara trebuie sa se stearga din memorie comanda de la el si de abia apoi sa primeasca semnalul ca liftul stationeaza pentru a incepe cautarea.



**library ieee;**

**use ieee.std\_logic\_1164.all;**

**entity bistabild is**

**port( clk: in std\_logic;**

**d: in std\_logic;**

**q: out std\_logic);**

**end entity;**

**architecture comportamental of bistabild is**

**begin**

**process(clk,d)**

**variable qq: std\_logic;**

**begin**

**if(clk’event and clk = ’1’) then**

**qq := d;**

**end if;**

**q <= qq;**

**end process;**

**end architecture;**

1. **Descriere proiect in ansamblu**

Arhitectura proiectului intreg este de tip structural, iar componentele sunt instantiate dupa schema bloc a proiectului.

Liftul primeste comenzi din interiorul liftului, sau din exterior de la fiecare etaj, in sus sau in jos. Acestea sunt introduse intr-o memorie, si sterse atunci cand cererea a fost indeplinita. Memoria retine la adresa etajului daca comanda a fost facuta din interior sau din exterior, si daca din exterior daca persoana doreste sa mearga in sus sau in jos. Liftul functioneaza dupa un algoritm pentru a se opri la cel mai avantajos etaj. Acesta onoreaza cererile in functie de sensul de mers.

Liftul nu porneste decat daca exista cel putin o comanda, greutatea pasagerilor nu depaseste greutatea maxima si daca nu se afla nicio persoana in usa. Dupa ce sunt indeplinite aceste conditii, exista un timp de asteptare, dupa care liftul isi inchide usile si porneste spre etajul destinatie.

Toate comenzile pe care le primeste liftul sunt retinute intr-o memorie, iar cand liftul se opreste la un etaj sunt sterse toate comenzile de la acel etaj. Memoria are la fiecare adresa 4 biti. Bitul cel mai putin semnificativ retine daca comanda data de la/ la etajul (etajul reprezinta adresa) respectiv este facuta din interiorul liftului. Bitul de pe pozitia 1 retine daca comanda a fost facuta din exterior, cel de pe pozitia 2 retine daca comanda facuta de la etajul respectiv si se vrea sa se mearga in jos si de pe pozitia 3 in sus.

Fiecare componenta comunica cu cealalta astfel exista semnale care sunt transimsa de la o componenta la cealalta.

Semnalul **pornire** reprezinta iesirea componentei de enable care este intrare pentru numaratorul de stationare. Acesta este la inceput 0 deoarece chiar la inceputul functionarii liftului nu exista vreo comanda in memorie. **Caut** este semnalul care desemneaza sensul in care merge liftul, la inceput fiind 1 deoarece de la parter se poate duce numai in sus. Acesta este iesire a comparatorului si este intrare pentru componenta de alegere a etajului urmator si a numaratorului pentru etajul curent.O alta iesire a comparatorului este  **sosire**. Semnalul **sosire** este semnalul care arata daca liftul stationeaza sau nu ( la inceput este 1 deoarece stationeaza). Acesta este necesar pentru alegerea etajului, pentru numaratorul etaj curent si pentru componenta de enable. Ultima iesire a comparatorului este **egale** care arata daca etajul curent si cel ales sunt egale. Acesta este necesar doar pentru alegerea etajului, adica pentru stergerea din memorie a comenzii indeplinite.

**Curr\_lvl** este semnalul care reprezinta etajul curent. Acesta este iesire a numaratorului pentru etaj curent si este transmis catre componenta de alegere a etajului si catre componenta de afisoare. Acesta este la inceput „0000” pentru ca, dupa cum spune si cerinta, automatul se „trezeste” la parter. **Next\_lvl** reprezinta etajul urmator care este iesirea cea mai importanta a componentei de alegere a etajului. Aceasta este necesaa numai pentru comparator. O alta iesire a componentei alegere etaj necesara pentru comparator este **pt\_comp** care reprezinta urmatorul etaj la care urma sa mearga liftul. Ultima iesire a componentei de alegere a etajului este **nimic** care este activa daca memoria din interiorul acestei componente este goala. Acest semnal este o intrare pentru componenta de enable numarator stationare.

Pentru componenta de enable mai avem nevoie de informatii despre cei 2 senzori. Acestia sunt butoane si de aceea trebuie inainte sa treaca printr-un debouncer asa ca semnalele care ies din debouncer sunt **gm\_v**  si **pu\_v**.

**Pt\_alegere** este semnalul de stationare intarziat cu o secunda care iese dintr-un bistabil d necesar pentru componenta de alegere a etajului.

De la numaratorul de stationare singura iesire care este transmisa comparatorului ( necesar pentru a determina daca liftul stationeaza sau nu) este **terminat** (adica daca s-a terminat timpul de stationare).

Singurele semnale care au mai ramas sunt semnalele de clk: **clk1** este semnalul care iese din divizorul de frecventa de o secunda si este necesar pentru toate componentele care necesita clock in afara de numaratorul etaj curent, **clk3**  este semnalul care iese din divizorul de frecventa de 3 secunde, iar **clk\_div**

**library IEEE;**

**use IEEE.STD\_LOGIC\_1164.ALL;**

**use IEEE.STD\_LOGIC\_ARITH.ALL;**

**use IEEE.STD\_LOGIC\_UNSIGNED.ALL;**

**entity lift is**

**port (clk : in std\_logic;**

**etaj\_comanda : in std\_logic\_vector (3 downto 0);**

**int : in std\_logic;**

**exte : in std\_logic;**

**up : in std\_logic;**

**down : in std\_logic;**

**ok : in std\_logic;**

**pu : in std\_logic;**

**gm : in std\_logic;**

**ac : in std\_logic;**

**su : out std\_logic;**

**anozi : out std\_logic\_vector (1 downto 0);**

**catozi : out std\_logic\_vector (6 downto 0));**

**end entity;**

**architecture structural of lift is**

**component e\_16sec is**

**port (gm : in std\_logic;**

**pu : in std\_logic;**

**stationare : in std\_logic;**

**mem\_goala : in std\_logic;**

**permisie : out std\_logic);**

**end component;**

**component alegere\_etaj\_nou is**

**port (clk : in std\_logic;**

**etaj\_curent : in std\_logic\_vector (3 downto 0) := "0000";**

**etaj\_comanda : in std\_logic\_vector (3 downto 0);**

**int, exte, up, down : in std\_logic; --- up down ext int**

**ok : in std\_logic;**

**egal : in std\_logic;**

**stationare : in std\_logic := '1';**

**sens : in std\_logic := '1';**

**mem\_goala : out std\_logic := '1';**

**etaj\_urmator : out std\_logic\_vector (3 downto 0) := "0000";**

**p\_comp : out std\_logic\_vector (3 downto 0) := "0001");**

**end component;**

**component num\_indirect is**

**port (clk: in std\_logic;**

**enable: in std\_logic; --merge pe 1**

**br : out std\_logic := '0');**

**end component;**

**component counter4bit is**

**port (clock: in std\_logic;**

**enable: in std\_logic := '1'; --merge pe 0**

**sens: in std\_logic := '0'; -- sus:0 jos:1**

**output: out std\_logic\_vector (3 downto 0));**

**end component;**

**component afisoare is**

**port(clock: in std\_logic;**

**etaj: in std\_logic\_vector (3 downto 0);**

**an: out std\_logic\_vector (1 downto 0);**

**cat: out std\_logic\_vector (6 downto 0));**

**end component;**

**component comparator is**

**port(etaj\_curent: in std\_logic\_vector(3 downto 0);**

**etaj\_ales: in std\_logic\_vector(3 downto 0);**

**br: in std\_logic;**

**egalitate : in std\_logic\_vector (3 downto 0) := "0001";**

**stationare: out std\_logic := '1';**

**sens: out std\_logic := '1'; -- sus = 1, jos = 0**

**ajuns : out std\_logic);**

**end component;**

**component div\_de\_frecv1 is**

**port(clk: in std\_logic;**

**clk\_out: out std\_logic);**

**end component;**

**component div\_de\_frecv3 is**

**port(clk: in std\_logic;**

**clk\_out: out std\_logic);**

**end component;**

**component bistabild is**

**port (clk : in std\_logic;**

**d : in std\_logic;**

**q : out std\_logic);**

**end component;**

**component mux2la1 is**

**port (i0, i1, s : in std\_logic;**

**y : out std\_logic);**

**end component;**

**signal pornire : std\_logic := '0'; --enable num.16sec = permisie**

**signal caut: std\_logic := '1'; --sens**

**signal sosire : std\_logic := '1'; --enable num. etaj = stationare**

**signal next\_lvl : std\_logic\_vector (3 downto 0) := "0000"; --etaj urmator**

**signal curr\_lvl : std\_logic\_vector (3 downto 0) := "0000"; --etaj curent**

**signal terminat : std\_logic := '0'; --br**

**signal nimic : std\_logic := '1'; --memorie goala**

**signal egale : std\_logic := '1'; --ajuns**

**signal pt\_comp : std\_logic\_vector (3 downto 0) := "0001"; --egalitate**

**signal pt\_alegere : std\_logic := '1'; --intarziere stationare**

**signal clk1 : std\_logic;**

**signal clk3 : std\_logic;**

**signal clk\_div : std\_logic;**

**signal gm\_v : std\_logic;**

**signal pu\_v : std\_logic;**

**begin**

**F1 : div\_de\_frecv1 port map (clk, clk1);**

**F3 : div\_de\_frecv3 port map (clk, clk3);**

**MUX : mux2la1 port map (clk1, clk3, ac, clk\_div);**

**DEB\_GM : bistabild port map (clk1, gm, gm\_v);**

**DEB\_PU : bistabild port map (clk1, pu, pu\_v);**

**INTARZIERE : bistabild port map (clk1, sosire, pt\_alegere);**

**NI : num\_indirect port map (clk\_div, pornire, terminat);**

**NE : counter4bit port map (clk\_div, sosire, caut, curr\_lvl); --- trebuie schimbata logica**

**AF : afisoare port map (clk\_div, curr\_lvl, anozi, catozi);**

**E13S : e\_16sec port map (gm\_v, pu\_v, sosire, nimic, pornire);**

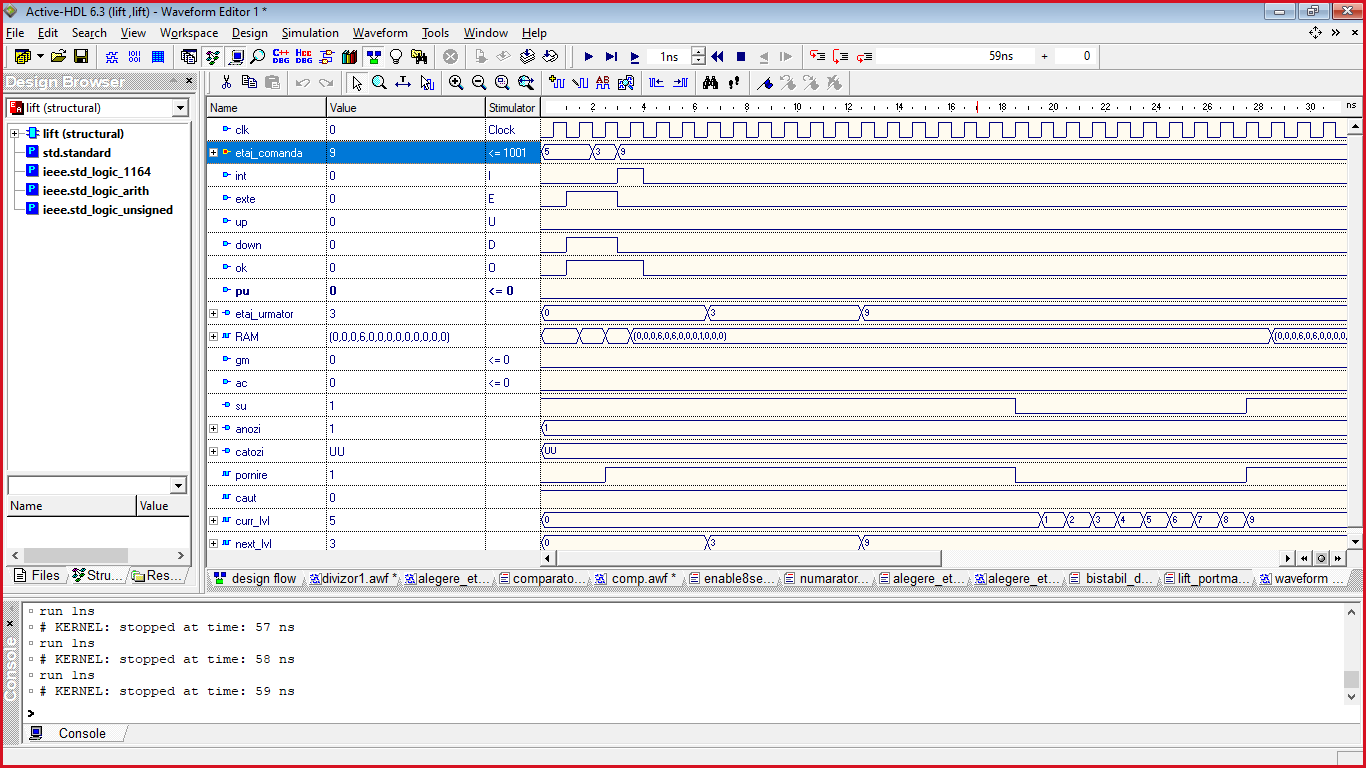
**COM : comparator port map (curr\_lvl, next\_lvl, terminat, pt\_comp, sosire, caut, egale);**

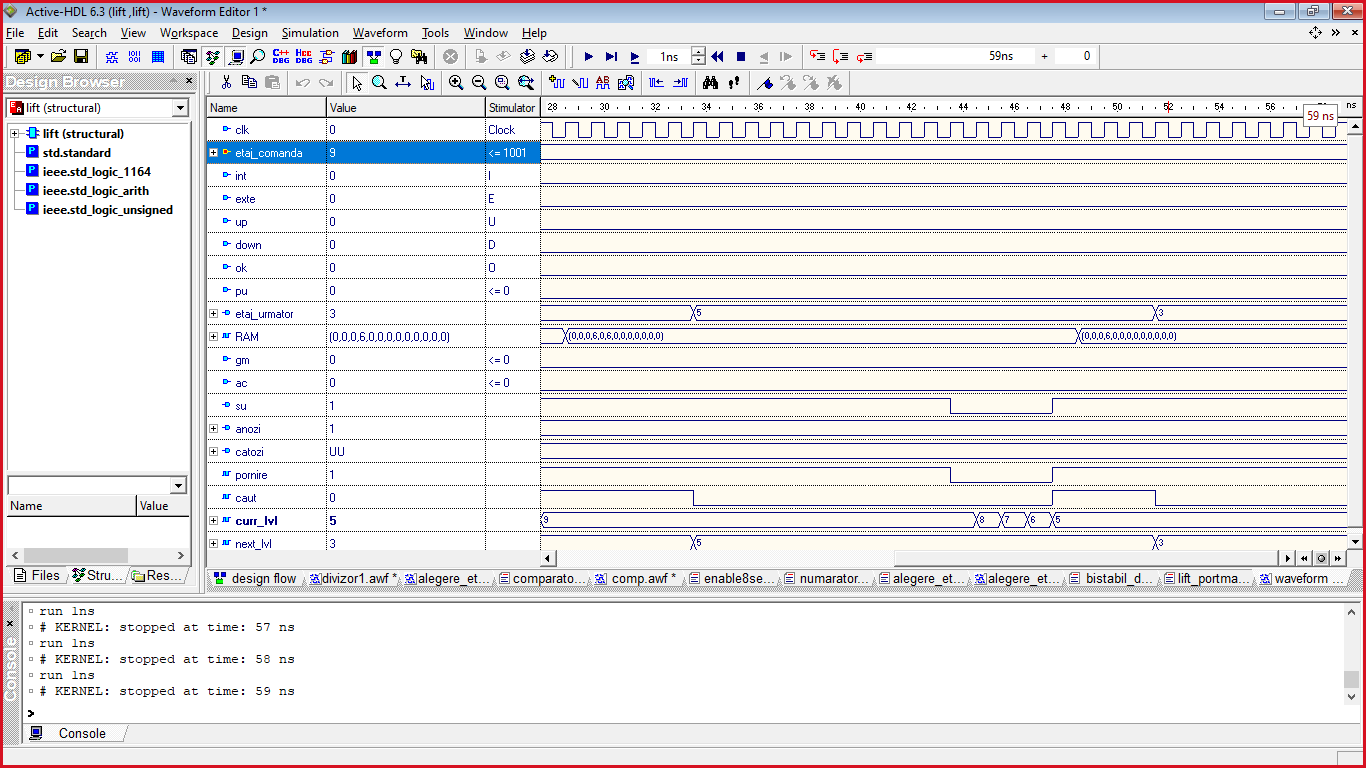
**NLVL : alegere\_etaj\_nou port map (clk\_div, curr\_lvl, etaj\_comanda, int, exte, up, down, ok, egale, pt\_alegere, caut, nimic, next\_lvl, pt\_comp);**

**su <= sosire;**

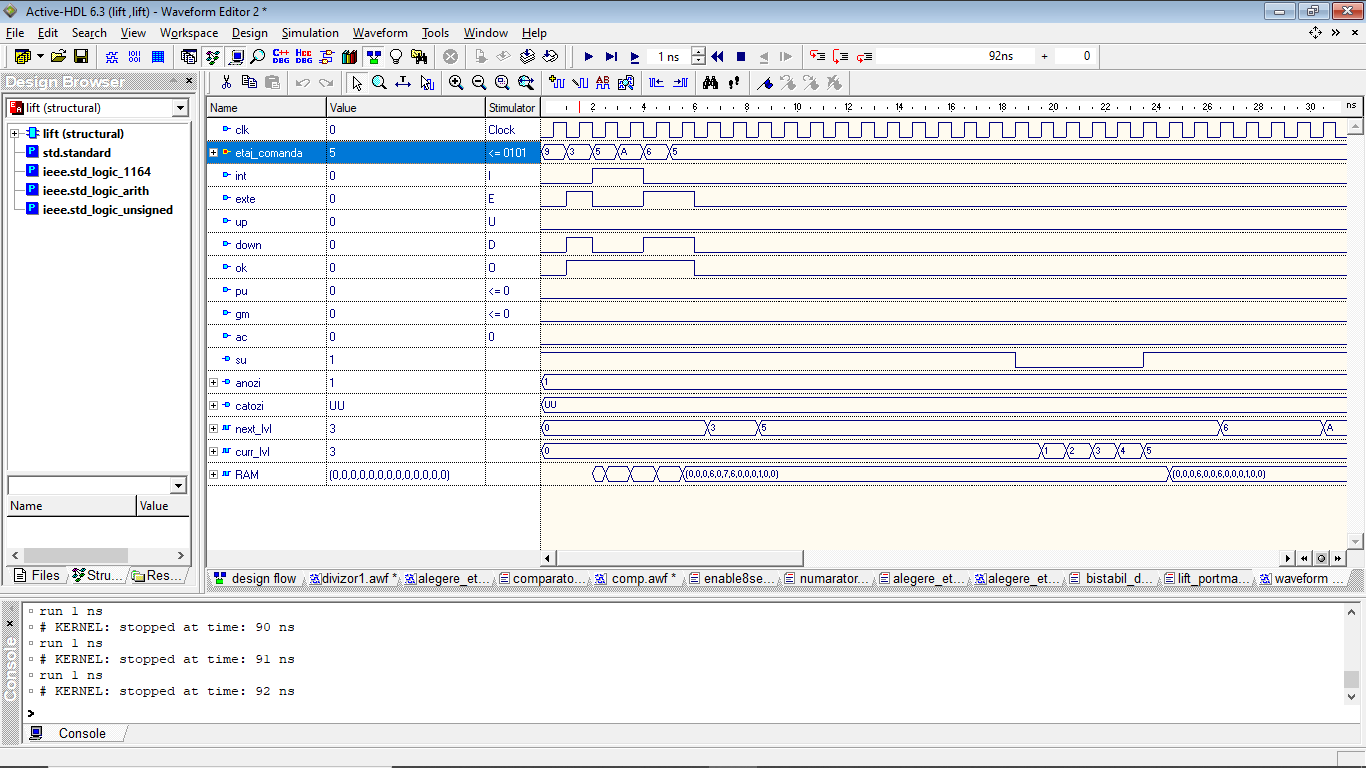
**end architecture structural;**

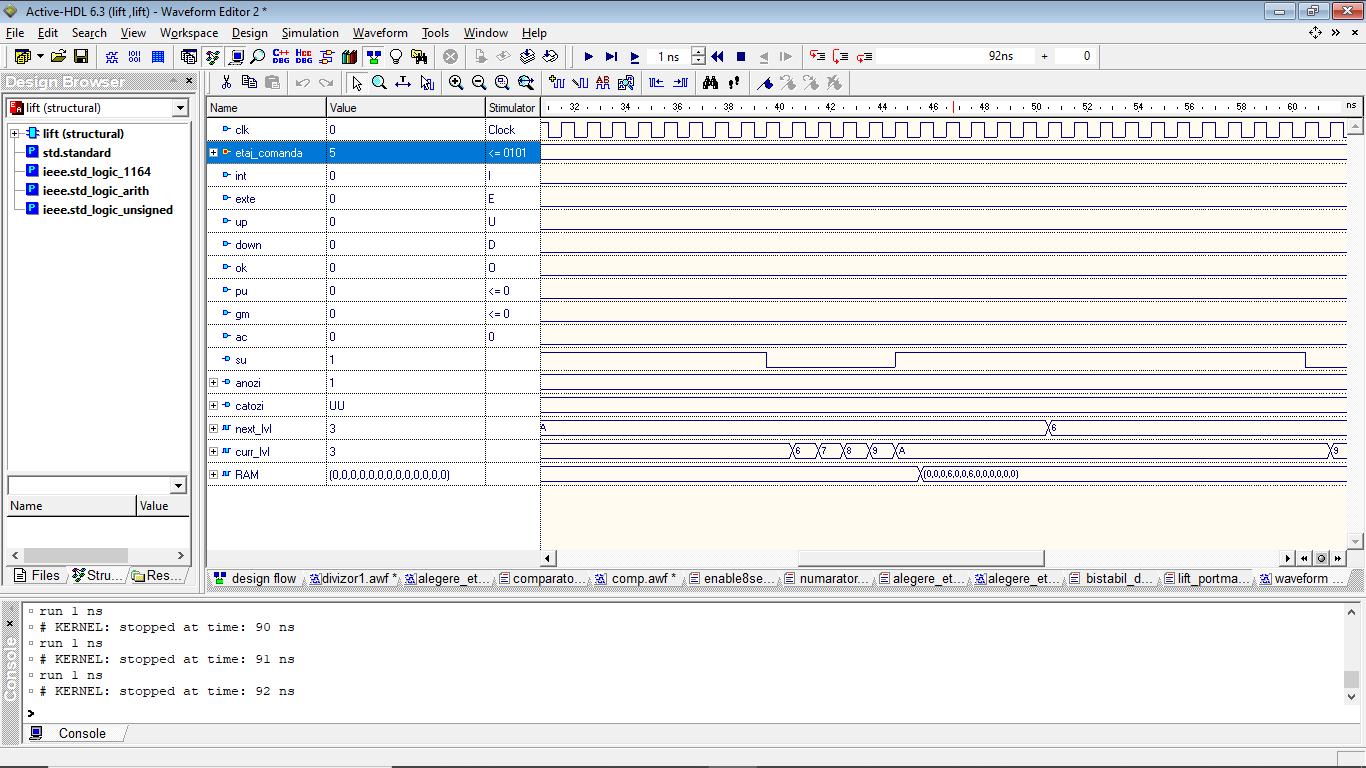
**Simulare I**

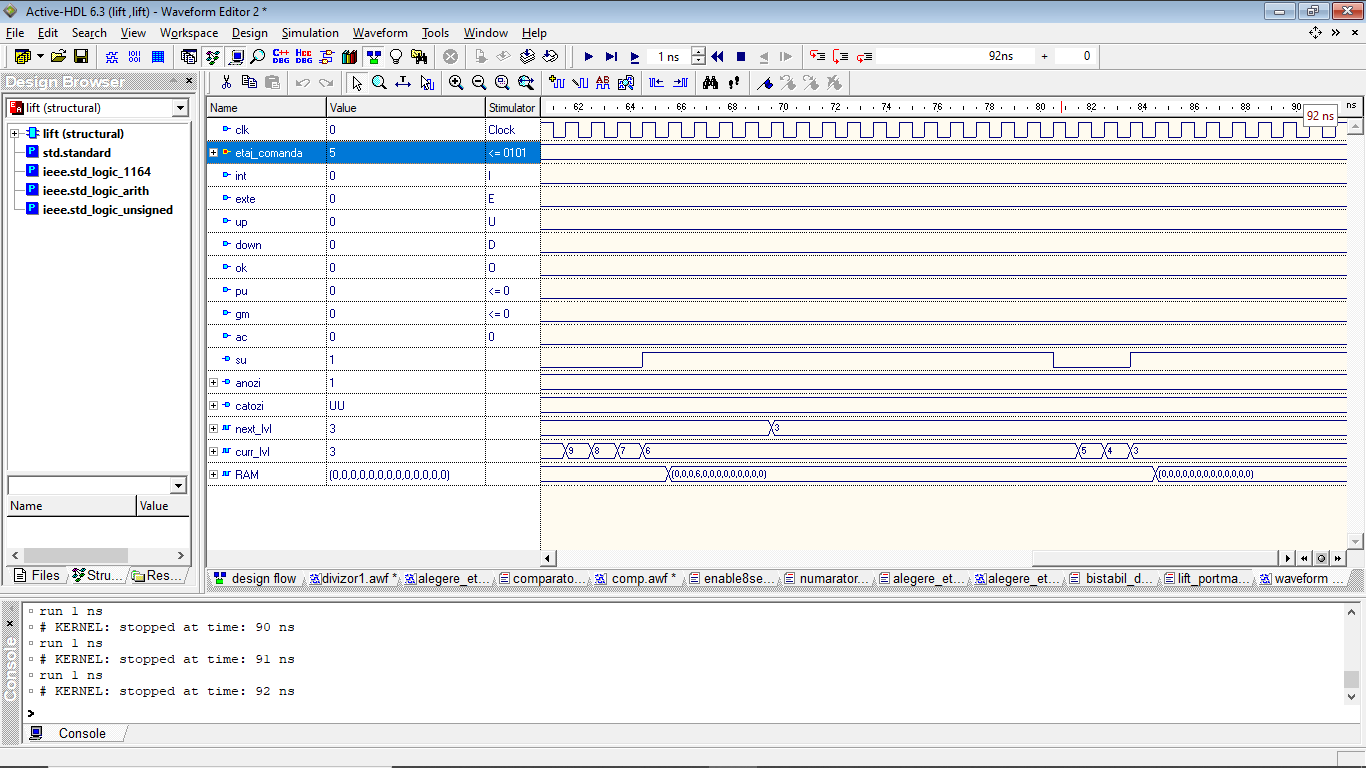




**Simulare II**







Cum in simularea din Active – HDL, nu am folosit divizoarele de frecventa, iar liftul se misca la frecventa aleasa de noi, anozii si catozii nu is schimba valorile initiale.

1. **Fisierul de constrangeri din ISE pentru placa cu FPGA aleasa (Nexys 4)**

**///clk**

**NET "clk" LOC = "E3" | IOSTANDARD = "LVCMOS33";**

**NET "clk" TNM\_NET = sys\_clk\_pin;**

**TIMESPEC TS\_sys\_clk\_pin = PERIOD sys\_clk\_pin 100 MHz HIGH 50%;**

**///intrari**

**NET "etaj\_comanda[3]" LOC = P4 | IOSTANDARD = "LVCMOS33";**

**NET "etaj\_comanda[2]" LOC = P3 | IOSTANDARD = "LVCMOS33";**

**NET "etaj\_comanda[1]" LOC = R3 | IOSTANDARD = "LVCMOS33";**

**NET "etaj\_comanda[0]" LOC = T1 | IOSTANDARD = "LVCMOS33";**

**NET "int" LOC = T3 | IOSTANDARD = "LVCMOS33";**

**NET "exte" LOC = U2 | IOSTANDARD = "LVCMOS33";**

**NET "up" LOC = V2 | IOSTANDARD = "LVCMOS33";**

**NET "down" LOC = U4 | IOSTANDARD = "LVCMOS33";**

**NET "ok" LOC = V5 | IOSTANDARD = "LVCMOS33";**

**NET "ac" LOC = V6 | IOSTANDARD = "LVCMOS33";**

**NET "pu" LOC = T16 | IOSTANDARD = "LVCMOS33";**

**NET "gm" LOC = F15 | IOSTANDARD = "LVCMOS33";**

**///iesiri**

**NET "su" LOC = P2 | IOSTANDARD = "LVCMOS33";**

**NET "anozi[1]" LOC = M6 | IOSTANDARD = "LVCMOS33";**

**NET "anozi[0]" LOC = N6 | IOSTANDARD = "LVCMOS33";**

**NET "catozi[6]" LOC = L3 | IOSTANDARD = "LVCMOS33";**

**NET "catozi[5]" LOC = N1 | IOSTANDARD = "LVCMOS33";**

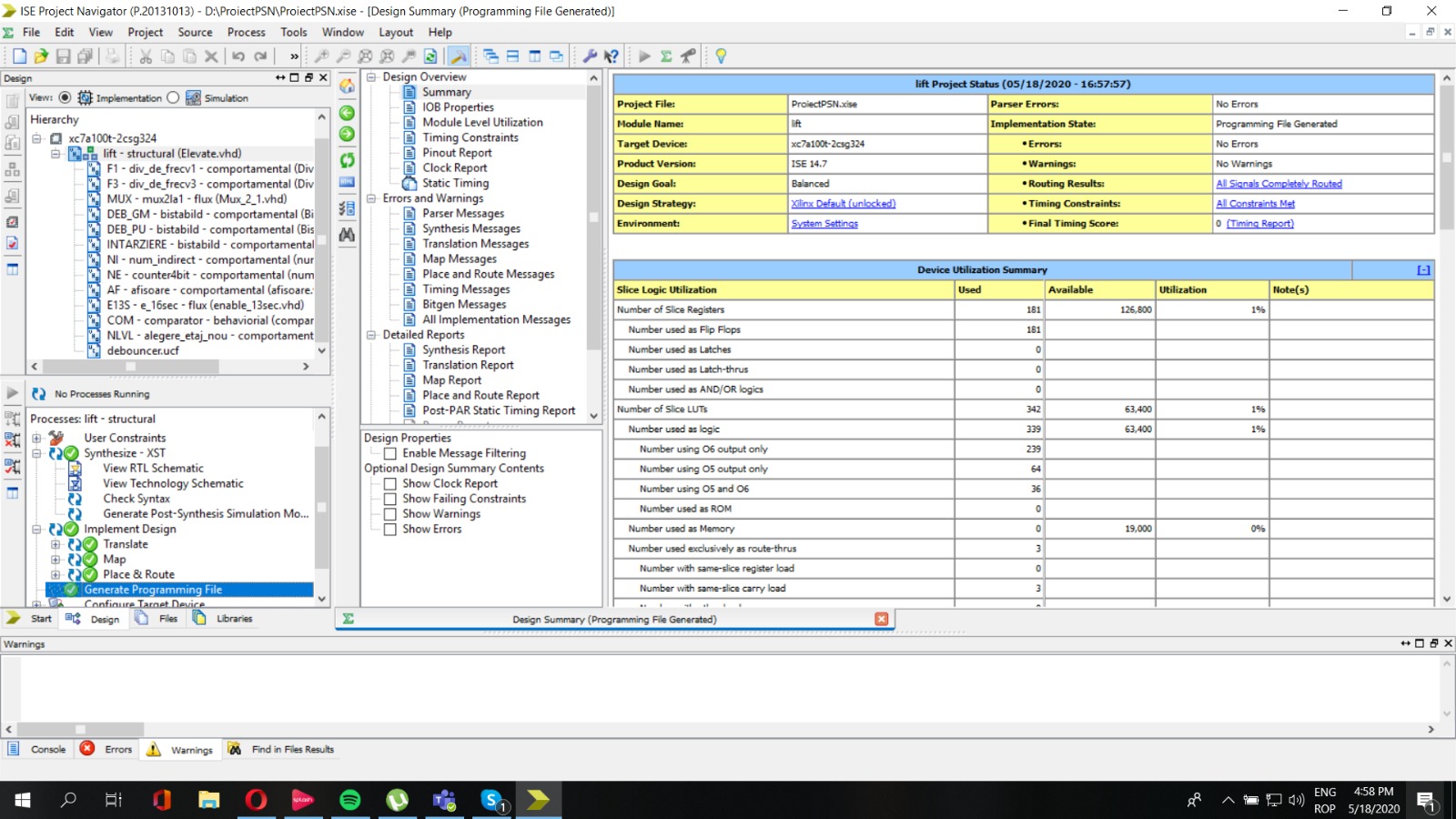
**NET "catozi[4]" LOC = L5 | IOSTANDARD = "LVCMOS33";**

**NET "catozi[3]" LOC = L4 | IOSTANDARD = "LVCMOS33";**

**NET "catozi[2]" LOC = K3 | IOSTANDARD = "LVCMOS33";**

**NET "catozi[1]" LOC = M2 | IOSTANDARD = "LVCMOS33";**

**NET "catozi[0]" LOC = L6 | IOSTANDARD = "LVCMOS33";**

****

1. **Justificarea solutiei alese**

Prima faza de proiectare a fost sa ne gandim de ce intrari avem nevoie si ce iesiri vom avea. La partea de iesiri a fost simplu, deoarece se intelege din cerinta problemei ca vom avea 2 afisoare 7 segmente care sa ne afiseze etajul curent, si un led care sa indice starea usilor.

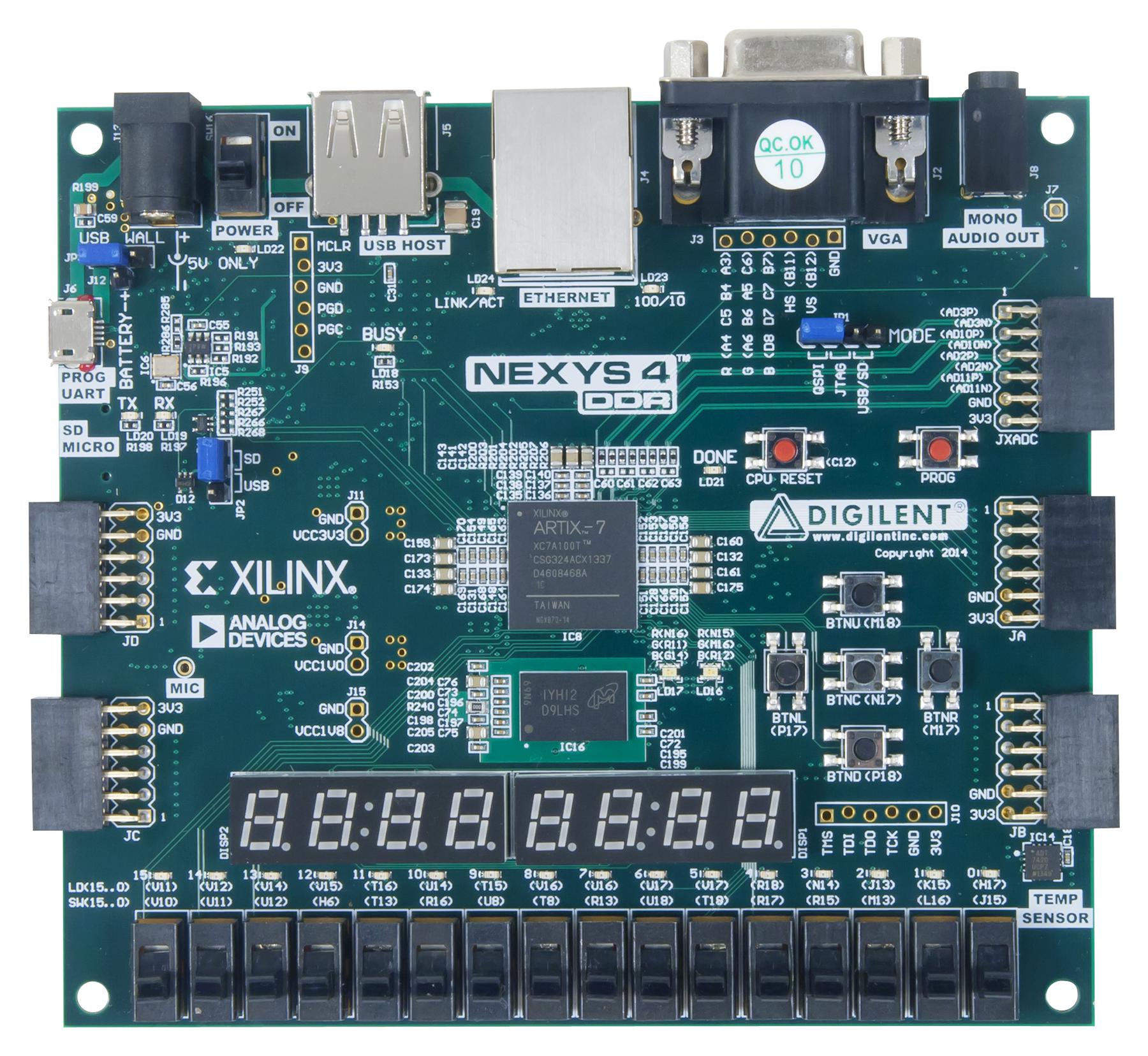
La intrari trebuia sa rezolvam prima oara problema comenzilor, sa gasim o solutie prin care sa introducem de unde este facuta comanda, la ce etaj si in ce sens. Nu puteam avea un buton pentru fiecare etaj si din exterior si din interior, asa ca am decis sa luam 4 switchuri care sa reprezinte etajul la care este facuta comanda, un switch care, daca este activat, sa reprezinte faptul ca aceasta este facuta din interior, unul pentru exterior, iar daca este facuta din exterior trebuie sa stim in ce sens vrea sa se duca utilizatorul, in sus sau in jos, asa ca am mai avut nevoie de inca un switch pentru sus si unul pentru jos. Pe langa aceste switchuri pentru a introduce o comanda am mai ajuns la concluzia ca ne mai trebuie un buton, care sa reprezinte un fel de verificare comanda. Numai cand este apasat pe acest buton se face comanda pentru a nu o face de repetate ori pana sunt inchise switchurile. Am mai avut nevoie de un switch care sa ne aleaga viteza cu care se deplaseaza liftul si de butoane pentru cei 2 senzori.

Apoi pentru organigrama am decis ca cel mai usor este sa avem 2 numaratoare, intrucat organigrama se poate imparti in 2: partea de asteptare si deplasarea la etajul ales, astfel ca un numarator se va ocupa de partea de asteptare, iar dupa el va fi partea numaratorul care va arata etajul curent la care este liftul (partea de deplasare la etaj). Aceste numaratoare nu pot functiona in acelasi timp asa ca pentru a implementa organigrama ne-am mai folosit de enableuri. Numaratorul de asteptare (numaratorul indirect) are mai multe conditii pentru a functiona asa ca am facut componenta enable numarator indirect explicata mai sus. Pentru ca Numaratorul pentru etaj curent sa functioneze trebuie numai ca liftul sa stationeze asa ca am primit aceasta intrare de la comparator.

Pentru unitatea de executie am decis ca avem nevoie, in primul rand, de o componenta mai complexa care sa retina comenzile, si daca retine comenzile sa si faca alegerea etajului cel mai potrivit. Pentru aceasta are nevoie, pe langa comenzi, sa stie sensul in care merge liftul si etajul curent. Sensul am zis ca il vom face prin implemetarea unui comparator. Pentru comparator nu avem nevoie numai de etajul ales si etajul curent dar si de inca o data, deoarece in momentul in care sunt etajele egale, adica atunci cand ajunge la etajul ales, nu poate decide daca se duce in sus sau in jos. In componenta de alegere a etajului am mai facut o iesire care sa ne indice care era etajul urmator pana in momentul in care a inceput liftul sa stationeze.

1. **Utilizare si rezultate**
2. **Resurse necesare**

Pentru creearea acestui proiect am avut nevoie de platforma ActiveHDL, din care am folosit editorul schematic si simulatorul. De asemenea am mai avut nevoie de ISE Design Suite pentru implementarea pe placuta FPGA. Noi am folosit o placuta Nexys 4, care face parte din familia Artix 7. Are 15,850 de sliceuri, fiecare cu 4 6-input LUTs si 8 bistabile flip-flop, 4,860 Kbits de memorie RAM, 240 sliceuri DSP. Aceasta placuta are 16 switch-uri, 16 leduri, 5 butoane si 8 afisoare.



1. **Descrierea utilizarii**

Pentru a se introduce o comanda trebuie sa se introduca pe switchurile P4, P3, R3, T1, numarul etajului in binare catre care se doreste deplasarea sau etajul la care este utilizatorul in functie, daca acesta este in interior sau in exteriorul liftului (daca este in exterior se introduce etajul la care este, iar daca se afla in interior, etajul la care vrea sa mearga). De asemenea, trebuie sa se specifice daca comanda este facuta din interior, pe switch-ul T3, sau din exterior pe switch-ul U2. Daca este facuta din exterior trebuie neaparat sa se specifice in ce sens vrea sa mearga utilizatorul (sus pe switch-ul V2, jos pe switch-ul U4). Dupa ce au fost introduse toate aceste date trebuie sa se aprinda switch-ul ok (V5) pentru ca aceasta sa fie introdusa in sistem. Aceste switch-uri trebuie tinute in aceasta pozitie pentru cel putin o secunda, iar cand se termina de introdus comanda trebuie sa se inchida.

Din interior se mai poate alege viteza cu care sa se deplaseze liftul de e switch-ul V6, cand este 0 se deplaseaza cu 1 secunda pe etaj si cand este 1 cu 3 secunde pe etaj. Butonul T16 va fi pornit cand este o persoana in usa, iar butonul F15 daca s-a depasit greutatea maxima.

1. **Rezultate obtinute pe placile FPGA**

Pe placuta vom avea active numai 2 afisoare 7 segment si un led care va determina starea usilor. Ledul va fi aprins cand usile sunt deschise, adica cand stationeaza, si o sa fie aprins timp de 16 secunde daca senzorii nu sunt aprinsi ( daca sunt aprinsi pot sa fie si mai multe secunde). Ledul pe care l-am ales pentru acesta este P2.

Afisoarele 7 segment vor arata constant la ce etaj este liftul, cand stationeaza va ramane acelasi etaj timp de 16 secunde (sau mai mult dupa cum am spus mai sus), iar daca nu se vor schimba cu etajul urmator la fiecare secunda, sau 3 secunde in functie de care frecventa este aleasa de multiplexor.

1. **Posibilitati de dezvoltare ulterioara**

In ceea ce priveste proiectul s-ar putea aduce mai mult imbunatatiri, decat modificari, deoarece am incercat sa alegem etajele in cel mai economic mod posibil. Un sistem de preluare al comenzilor vocale ar usura mult folosirea liftului.

O mare problema a lifturilor este greutatea folosirii lor de catre persoanele cu anumite handicapuri, precum cele cu deficiente de vaz. In afara de sistemul de preluare al comenzilor vocal care ar fi foarte util pentru acestea, un sistem care sa anunte vocal cand se deschid usile, sau sa anunte din 5 in 5 secunde cat mai este pana la inchiderea usilor ar fi util. Sau un sistem care sa anunte in interior la ce etaj se opreste.

De multe ori, un om poate gresi cand introduce comanda, sau in cazul copiilor mici care se joaca cu butoanele si introduc mai multe comenzi, ar fi foarte util un buton pentru a sterge o comanda, pentru ca liftul sa nu functioneze fara a fi utilizat (sa nu functioneze fara pasageri).

Un buton de „comanda urgenta” ar mai putea fi implementat. Aceste tipuri de comenzi ar fi indeplinite primele inaintea oricarei alte comenzi (de exemplu, folosirea liftului in cadrul unui spital).

Un buton care sa emite semnale sonore in cazul defectarii liftului.

Un buton care sa opreasca functionarea liftului in cazul unei urgente.