

# Lift într-un hotel cu P+12 etaje

Universitatea Tehnică Cluj-Napoca  
Facultatea de Automatică și Calculatoare  
Specializare: Calculatoare și Tehnologia Informației

Materie: Proiectarea sistemelor numerice

Anul 2020-2021

Profesor îndrumător: Vlad Cristian Miclea

Realizatori: Alina Aurică, Șerban Gherman

Grupa: 30216 (seria B)

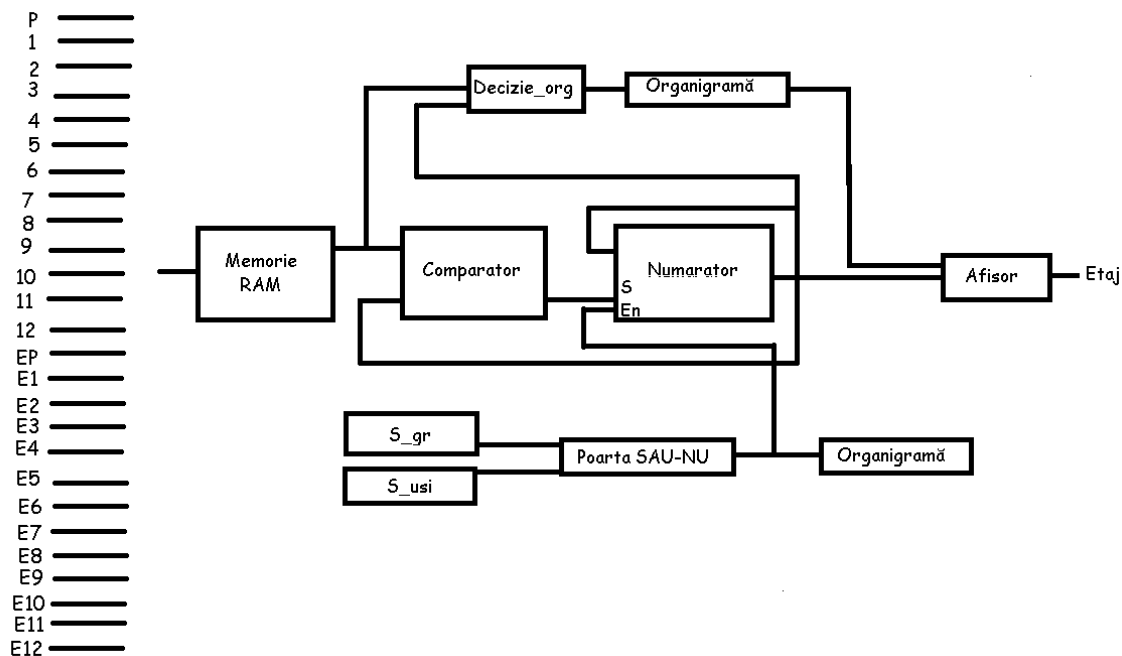
## **Cuprins:**

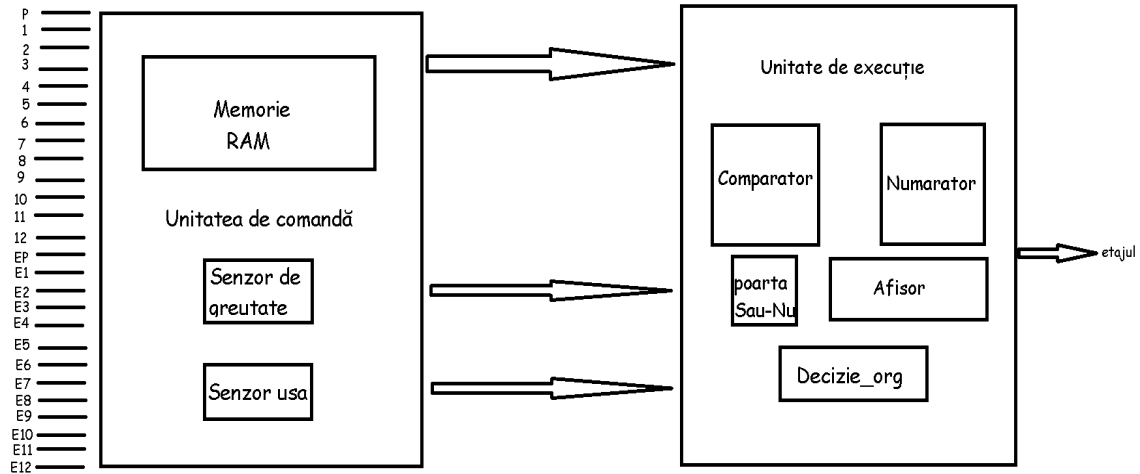
1. Specificații
2. Schema bloc
3. Componente
4. Organigramă
5. Cod componente
6. Semnificația notațiilor efectuate
7. Justificarea soluției alese
8. Instrucțiuni de utilizare și întreținere
9. Posibilități de dezvoltare ulterioare

## 1. ENUNȚUL PROBLEMEI

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 din exterior (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. SCHEMA BLOC





Comanda dată prin apăsarea butonului va fi reținută într-o memorie RAM, de unde va fi extrasă și introdusă într-un comparator. Acesta va decide dacă din punctul unde se află liftul, acesta va urca sau va coborî. Pe măsură ce numără, dacă ajunge la etajul selectat, liftul se va opri și ușile se vor deschide. Dacă, în schimb, nu e etajul selectat, valoarea aceluia etaj se va întoarce în comparator. Decizia ca numărătorul să pornească va fi dată de starea de decizie UI din organigramă, adică de valoarea care vine de la o poartă SAU-NU. Prin poarta SAU-NU decidem dacă greutatea maximă admisă de lift e depășită sau dacă există vreo persoană care stă în dreptul ușii. Doar dacă ambele condiții nu sunt îndeplinite, numărătorul va începe să funcționeze. Pentru a opri numărătorul, în decizie\_org verificăm dacă etajul unde e acum liftul este egal cu cel la care trebuie să ajungă. Dacă liftul se află în dreptul unui etaj, atunci îl va afișa.

(Șerban & Alina)

### 3. COMPONENTE

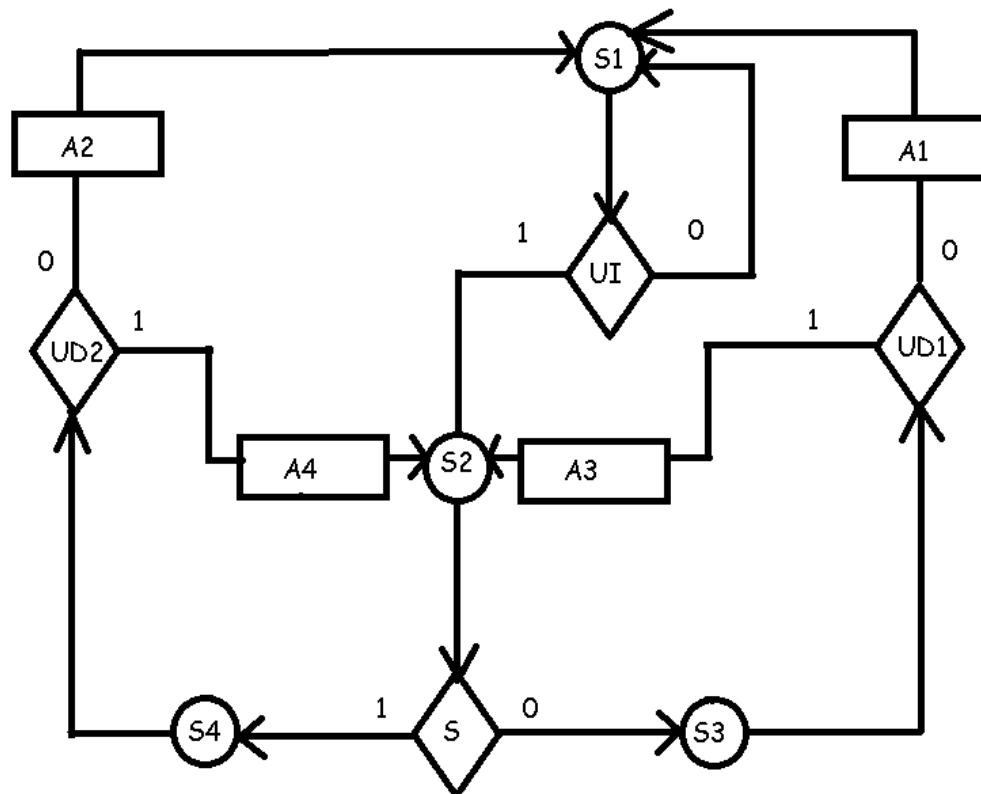
- UC {
1. **Memorie RAM**: reține comenzile din interior și cele din exterior
  2. **Senzor de greutate**: anunță dacă greutatea maximă este depășită
  3. **Senzor ușă**: anunță dacă o persoană este sau nu în dreptul ușii

UE

4. **Comparatorul**: compară etajul unde se află liftul cu etajele de unde primește comandă sau cu etajele la care trebuie să ajungă, pentru a determina dacă liftul urcă sau coboară
5. **Numărător pe 4 biți reversibil, sincron, binar-zecimal**:
  - dacă numără crescător, atunci liftul urcă, dacă numără descrescător, liftul coboară
  - stabilește viteza de deplasare a liftului
  - revine la starea inițială prin reset
6. **Decizie org**: determină dacă liftul continuă să meargă sau se oprește
7. **Poarta SAU-NU**: poarta care imprimă condițiile de funcționare ale număratorului
8. **Afișor pe 7 segmente**: decodifică etajul din binar în zecimal și-l afișează

(Șerban & Alina)

#### 4. **ORGANIGRAMĂ**



Starea 1 – staționare

UI – uși închise: (senzor de greutate, senzor ușă)

0 – ușile nu sunt închise, deci staționează

1 – ușile sunt închise, stabilește dacă urcă sau coboară

Starea 2 – ia decizii (comparator)

S – sens ( urcă/coboară ): (numărător)

0 – urcă

1 – coboară

Starea 3 – coboară (numără descrescător)

Starea 4 – urcă (numără crescător)

UD1, UD2 – uși deschise: (decizie\_org)

1 – ușile nu sunt deschise, liftul continuă să meargă ( revine la starea 3 sau la starea 4)

0 – ușile sunt deschise, liftul staționează

A1, A2, A3, A4 – afișează etajul (afișor)

(Șerban & Alina)

## 5. **COD COMPONENTE**

### **Organigrama:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_arith.all;

use IEEE.std\_logic\_unsigned.all;

entity ORGANIGRAMA is

port(CLK, RST: in STD\_LOGIC;

UI, S, UD1, UD2: in STD\_LOGIC;

A1, A2, A3, A4: out STD\_LOGIC);

end ORGANIGRAMA;

architecture corp of ORGANIGRAMA is

type type\_states is (S1, S2, S3, S4);

signal st, nx\_st: type\_states;

begin

CLS: process(CLK, RST)

begin

if RST = '1' then

st <= S1;

elsif CLK'event and CLK = '1' then

st<=nx\_st;

end if;

end process;

CLC: process(st, UI, S, UD1, UD2)

begin

A1 <= '0';

A2 <= '0';

A3 <= '0';

A4 <= '0';

case st is

when S1 =>

if UI = '0' then nx\_st <= S1;

else nx\_st <= S2;

end if;

when S2 =>

if S = '0' then nx\_st <= S3;

else nx\_st <= S4;

end if;

when S3 =>

if UD1 = '0' then

nx\_st <= S1 after 15000ms;

A1 <= '1';

else



```

        A3 <= '1';

        nx_st <= S2;

    end if;

when S4 =>

    if UD2 = '0' then

        nx_st <= S1 after 15000ms;

        A2 <= '1';

    else nx_st <= S2;

        A4 <= '1';

    end if;

end case;

end process;

end corp;

```

Organigrama are ca intrări: CLK, RST, UI, S, UD1, UD2 și ieșiri: A1, A2, A3, A4 (entitate). Ea funcționează atunci când CLK ia valoarea 1 și revine la starea inițială când RST primește valoarea 1. Stările de decizie sunt S1, S2, S3, S4. În funcție de valoarea lor, liftul staționează, urcă sau coboară. Semnalele intermediare **st** și **nx\_st** ajută la trecerea dintr-o stare în alta în timpul procesului. Tot aici s-a setat timpul pentru care ușile se mențin deschise (15s). *Descrierea este una de tip comportamental.*

(Șerban & Alina)

### **Memoria RAM:**

library IEEE;

use IEEE.std\_logic\_1164.all;

use IEEE.std\_logic\_unsigned.all;

entity MEMORIE\_RAM is

port( A\_RAM: in STD\_LOGIC\_VECTOR (3 downto 0);

CS\_RAM: in STD\_LOGIC;

WE: in STD\_LOGIC; -- 0 - read; 1 - write

D1: in STD\_LOGIC\_VECTOR (3 downto 0);

D2: out STD\_LOGIC\_VECTOR (3 downto 0));

end MEMORIE\_RAM;

architecture MEMORIE\_RAM of MEMORIE\_RAM is

type RAM\_TYPE is array (0 to 12) of STD\_LOGIC\_VECTOR (3 downto 0);

signal RAM: RAM\_TYPE := (0 => "0000", 1 => "0001", 2 => "0010", 3 => "0011", 4  
=> "0100", 5 => "0101", 6 => "0110", 7 => "0111", 8 => "1000", 9 => "1001", 10 =>  
"1010", 11 => "1011", 12 => "1100");

begin

D2 <= RAM(conv\_integer(A\_RAM)) when WE = '0' and CS\_RAM = '1';

RAM(conv\_integer(A\_RAM)) <= D1 when WE = '1' and CS\_RAM = '1';

end MEMORIE\_RAM;

(Entitate:) Memoria RAM are ca intrări: A\_RAM, CS\_RAM, WE și D1. D2 este ieșire. În funcție de valoarea pe care o ia WE, se va face scriere sau citire. Cu alte cuvinte, pentru WE=0, se va introduce în D2 numărul de la adresa A\_RAM, în timp ce pentru WE=1, se va insera la adresa A\_RAM numărul D1 (adică etajul unde trebuie să ajungă). *Descrierea este una de tipul „flux de date”*.

(Șerban)

**Comparator:**

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
```

```
use IEEE.numeric_std.all;
```

```
use IEEE.std_logic_arith.all;
```

```
use IEEE.std_logic_unsigned.all;
```

```
entity COMPARATOR is
```

```
    port(A, B: inout STD_LOGIC_VECTOR (3 downto 0);
```

```
          S: inout STD_LOGIC);
```

```
end COMPARATOR;
```

```
architecture COMP of COMPARATOR is
```

```
begin
```

```
    process(A, B)
```

```

begin
    if A<B then
        S <= '1';
    end if;
    if A>B then
        S <= '0';
    end if;
end process;
end COMP;

```

Comparatorul are ca intrări numerele A și B, iar ca ieșire S (entitate). *Descrierea este una de tip comportamental.*

(Alina)

### **Numărător:**

```

library IEEE;

use IEEE.std_logic_1164.all;

use IEEE.std_logic_arith.all;

use IEEE.std_logic_unsigned.all;

entity NUMARATOR is

    port(R, CLK, S, ENABLE: in STD_LOGIC;

```

```
Q: inout STD_LOGIC_VECTOR (3 downto 0) := "0000");  
  
end NUMARATOR;
```

architecture num of NUMARATOR is

```
    signal timp: TIME := 3000ms;  
  
begin  
  
    process(R, CLK, S, timp)  
  
        variable val: STD_LOGIC_VECTOR (3 downto 0) := "0000";  
  
    begin  
  
        if R='1' then  
  
            Q <= "0000";  
  
        end if;  
  
  
        if ENABLE = '1' then  
  
            if CLK'event and CLK = '1' then  
  
                if S = '0' then  
  
                    Q <= Q + 1;  
  
                else  
  
                    Q <= Q - 1;  
  
                end if;  
  
            end if;  
  
        end if;  
  
    end if;  
  
end if;
```

```

        end if;

    end process;

end num;

```

Numărătorul are ca intrări pe: R, CLK, S, ENABLE și ca intrare-ieșire pe Q (entitate). R aflat pe 1 resetează numărătorul (îl pune să înnumere din nou de la 0). CLK pus pe 1 activează numărătorul, iar S semnifică selecția, adică dacă înnumără crescător sau descrescător. Tot aici este setată și viteza liftului prin semnalul timp. Numărătorul funcționează numai când ENABLE are valoarea 1. *Descrierea este una de tip comportamental.*

(Alina)

### **Afișor:**

```

library IEEE;

use IEEE.STD_LOGIC_1164.all;

use IEEE.STD_LOGIC_UNSIGNED.all;

entity AFISOR is

    port(input: in STD_LOGIC_VECTOR (3 downto 0);

          output: out STD_LOGIC_VECTOR (6 downto 0));

end AFISOR;

architecture af of AFISOR is

begin

    process(input)

```

```

begin

    case input is

        when "0000" => output <= "1111110"; -- 0 -- 7E

        when "0001" => output <= "1100000"; -- 1 -- 60

        when "0010" => output <= "1011011"; -- 2 -- 5B

        when "0011" => output <= "1110011"; -- 3 -- 73

        when "0100" => output <= "1100101"; -- 4 -- 65

        when "0101" => output <= "0110111"; -- 5 -- 37

        when "0110" => output <= "0111111"; -- 6 -- 3F

        when "0111" => output <= "1100010"; -- 7 -- 62

        when "1000" => output <= "1111111"; -- 8 -- 7F

        when "1001" => output <= "1110111"; -- 9 -- 77

        when "1010" => output <= "1101111"; -- 10 -- 6F

        when "1011" => output <= "0111101"; -- 11 -- 3D

        when "1100" => output <= "0011110"; -- 12 -- 1E

        when others => output <= "0000000"; -- altele -- 00

    end case;

end process;

end af;

```

Afișorul are ca intrare pe input și ca ieșire pe output (entitate). *Descrierea este de tip comportamental.* (Alina)

**Poarta Sau-Nu:**

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
```

```
entity POARTA_SAU_NU is
```

```
    port(S_gr, S_u: in STD_LOGIC;
```

```
          En: out STD_LOGIC);
```

```
end POARTA_SAU_NU;
```

```
architecture SAU_NU of POARTA_SAU_NU is
```

```
begin
```

```
    En<=not(S_gr or S_u);
```

```
end SAU_NU;
```

(Entitate:) Intrările S\_gr și S\_u reprezintă senzorul de greutate, respectiv senzorul de ușă. Dacă unul dintre ei este setat pe 1, adică este activat, En (ieșirea) va fi setat pe 0. *Descrierea este de tip „flux de date”.*

(Șerban)



**Decizie org:**

```
library IEEE;
```

```
use IEEE.std_logic_1164.all;
```

```
entity DECIZIE_ORG is
```

```
    port(A, Q: in STD_LOGIC_VECTOR (3 downto 0);
```

```
          U: out STD_LOGIC);
```

```
end DECIZIE_ORG;
```

```
architecture DECIZIE_ORG of DECIZIE_ORG is
```

```
begin
```

```
    process(A, Q)
```

```
    begin
```

```
        if A = Q then
```

```
            U <= '0';
```

```
        else
```

```
            U <= '1';
```

```
        end if;
```

```
    end process;
```

```
end DECIZIE_ORG;
```

Decizie\_org are ca intrări pe A și B și ca ieșire pe U (entitate). *Descrierea este una de tip comportamental.* (Alina)

**Lift (legarea tuturor componentelor):**

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity LIFT is

port(INTRARE: in STD\_LOGIC\_VECTOR (3 downto 0); --intrari lift

CERERE: in STD\_LOGIC\_VECTOR (3 downto 0); --adresa de memorie

S\_gr, S\_u: in STD\_LOGIC; --senzori de greutate

CLK, R: in STD\_LOGIC;

ETAJ: out STD\_LOGIC\_VECTOR (6 downto 0)); --iesire

end LIFT;

architecture LIFT of LIFT is

component ORGANIGRAMA

port(CLK, RST: in STD\_LOGIC;

UI, S, UD1, UD2: in STD\_LOGIC;

A1, A2, A3, A4: out STD\_LOGIC);

end component;

component POARTA\_SAU\_NU

port(S\_gr, S\_u: in STD\_LOGIC;

En: out STD\_LOGIC);

end component;

component COMPARATOR

port(A, B: in STD\_LOGIC\_VECTOR (3 downto 0);

```
        S: out STD_LOGIC);  
end component;
```

```
component NUMARATOR
```

```
    port(R, CLK, S, ENABLE: in STD_LOGIC;  
          Q: inout STD_LOGIC_VECTOR (3 downto 0));  
end component;
```

```
component AFISOR
```

```
    port(input: in STD_LOGIC_VECTOR (3 downto 0);  
          output: out std_logic_vector (6 downto 0));  
end component;
```

```
component MEMORIE_RAM
```

```
    port(A_RAM: in STD_LOGIC_VECTOR (3 downto 0); -- Adresele  
          CS_RAM: in STD_LOGIC; -- Semnal selectie cip, „Chip Select” (clock)  
          WE: in STD_LOGIC; -- 0 - read; 1 - write  
          D1: in STD_LOGIC_VECTOR (3 downto 0);  
          D2: out STD_LOGIC_VECTOR (3 downto 0)); -- Valoare  
end component;
```

```
component DECIZIE_ORG
```

```
    port(A, Q: in STD_LOGIC_VECTOR (3 downto 0);  
          U: out STD_LOGIC);  
end component;
```

```
signal A: STD_LOGIC_VECTOR (3 downto 0);
```

```

signal S_nr: STD_LOGIC;
signal En_nr: STD_LOGIC;
signal B: STD_LOGIC_VECTOR (3 downto 0);
signal UD: STD_LOGIC;
signal A1: STD_LOGIC;
signal A2: STD_LOGIC;
signal A3: STD_LOGIC;
signal A4: STD_LOGIC;
signal WE: STD_LOGIC:= '0';

begin

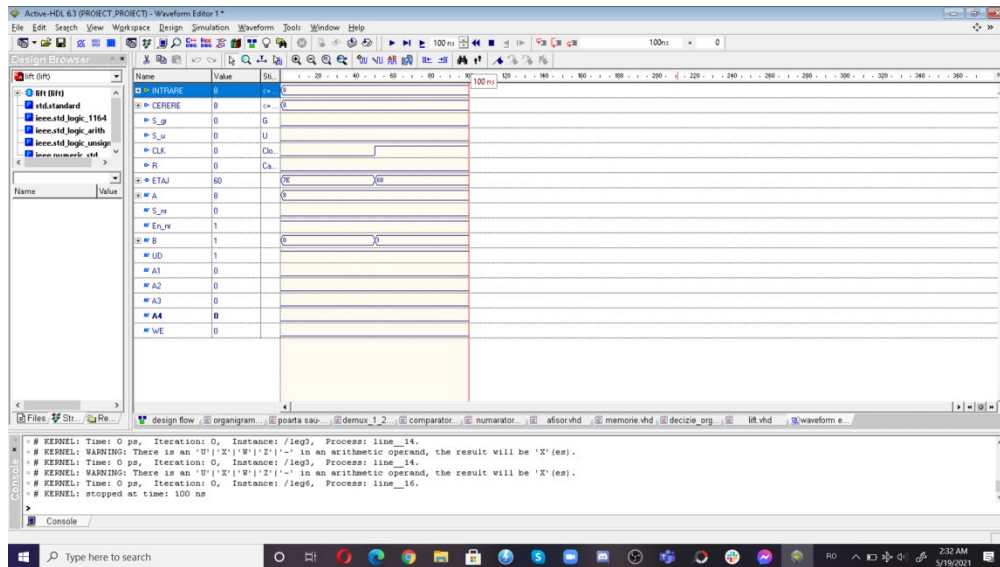
    leg1: MEMORIE_RAM port map (CERERE, '1', WE, INTRARE, A);
    leg3: COMPARATOR port map (A, B, S_nr);
    leg4: POARTA_SAU_NU port map (S_gr, S_u, En_nr);
    leg5: NUMARATOR port map (R, CLK, S_nr, En_nr, B);
    leg6: DECIZIE_ORG port map (A, B, UD);
    leg7: ORGANIGRAMA port map (CLK, R, En_nr, S_nr, UD, UD, A1, A2, A3, A4);
    leg8: AFISOR port map (B, ETAJ);

end LIFT;

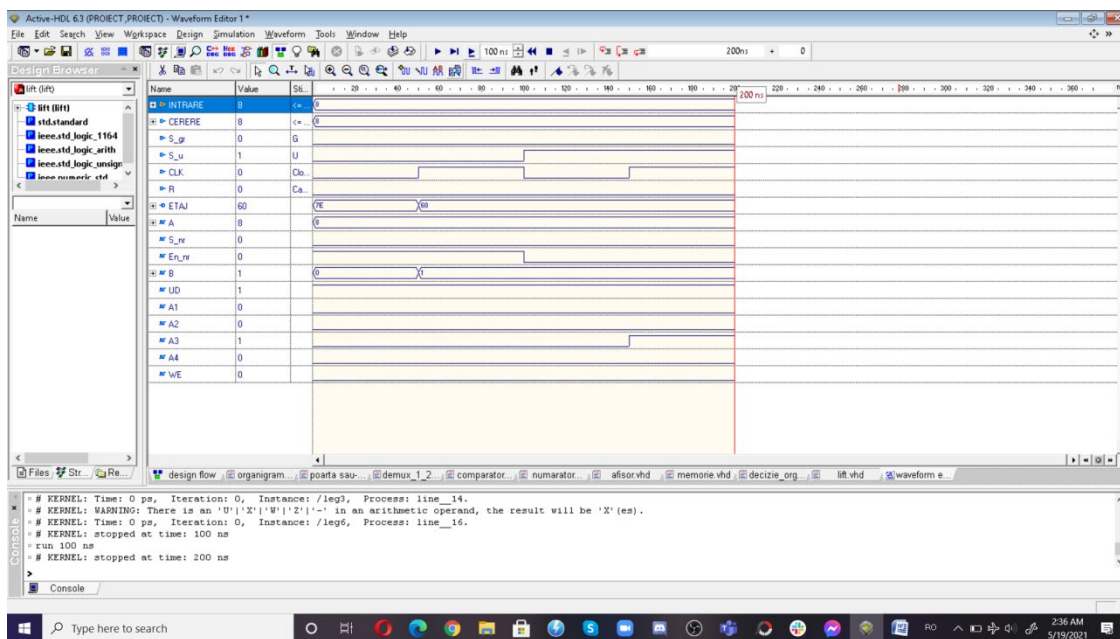
```

*Descrierea este una de tip structural.*

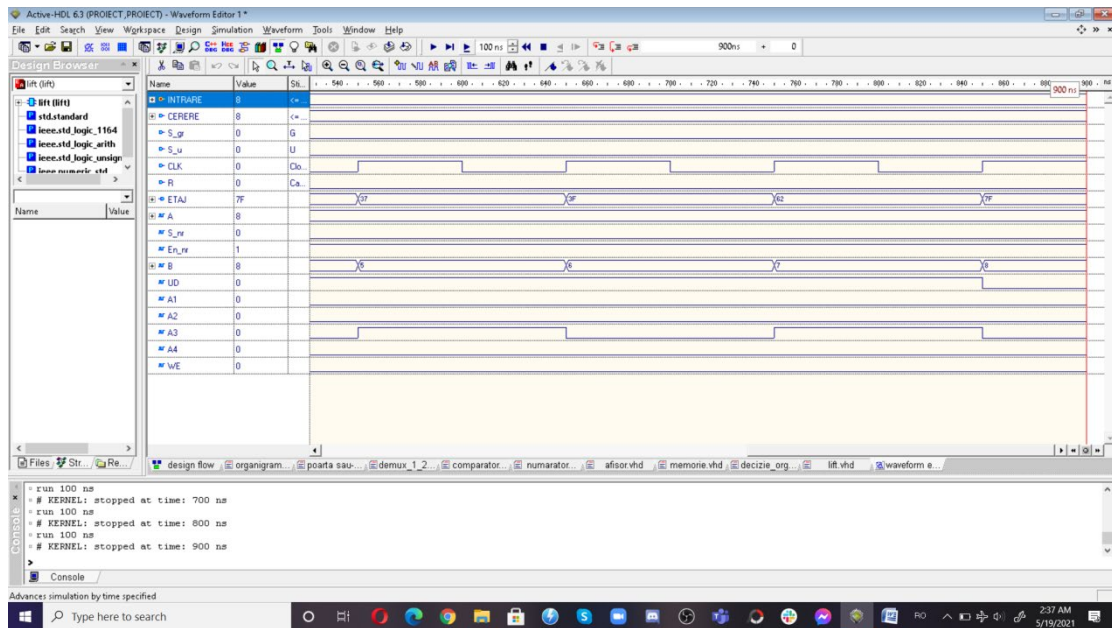
(Șerban & Alina)



Inițial introducem un etaj în memorie și apelăm pentru prima dată „Run for”. Numărătorul urcă de la parter la etajul 1 și afișează etajul în dreptul căruia a ajuns. Sensorii de greutate și de uși sunt pe 0, adică nu se depășește greutatea maximă și nici nu se află cineva în dreptul ușilor liftului.



Dacă unul dintre senzori se află activat, atunci liftul staționează.



Prin apelări multiple de „Run for” liftul ajunge la etajul dorit. Dacă nu mai primește alt semnal, se resetează (coboară la parter). Dacă mai primește, atunci se îndreaptă în acea direcție.

## 6. SEMNIFICATIA NOTATIILOR EFECTUATE

### Intrările liftului:

- ✚ CERERE - reprezintă adresa de memorie unde se salvează etajul
- ✚ INTRARE - reprezintă etajul apăsat din interior/exterior
- ✚ S\_gr - senzorul care determină dacă greutatea maximă este depășită
- ✚ S\_u - senzorul care determină dacă cineva e sau nu în ușă
- ✚ CLK - clock-ul liftului
- ✚ R - semnalul de reset al liftului

### Ieșirile liftului:

- ✚ ETAJ – reprezintă etajul unde a ajuns liftul

### Semnale intermediare:

- ✚ A - reprezintă numărul care vine de la memorie și intră în numărător și în decizie\_org

- ✚ S\_nr - este semnalul care stabilește sensul de deplasare al numărătorului; este rezultatul comparatorului
- ✚ En\_nr - stabilește dacă ușile sunt închise pentru a trece la următoarea stare și a începe să numere
- ✚ B - reprezintă numărul care vine de la numărător și este etajul unde se află liftul; joacă rol de intrare pentru comparator și decizie\_org
- ✚ UD - este semnalul care stabilește dacă ușile se deschid sau nu; provine de la decizie\_org
- ✚ A1 - este un semnal declanșator pentru afișor; vine de la organigramă
- ✚ A2 - este un semnal declanșator pentru afișor; vine de la organigramă
- ✚ A3 - este un semnal declanșator pentru afișor; vine de la organigramă
- ✚ A4 - este un semnal declanșator pentru afișor; vine de la organigramă
- ✚ WE - stabilește când se face scrierea și când se face citirea în memorie

(Șerban & Alina)

## 7. JUSTIFICAREA SOLUȚIEI ALESE


Noi am ales această soluție, pentru că am considerat că este cea mai ușoară metodă de a implementa un lift. Ea constă în împărțirea proiectului pe mai multe componente principale, fiecare având funcții diferite, putând, astfel, să-și îndeplinească rolul în proiect. Acestea sunt legate în modulul principal, componenta numită „lift”, printr-o descriere structurală, reprezentând rezultatul cerinței. Am folosit organigrama pentru a putea pune în funcțiune automatul, adică pentru a putea trece dintr-o stare în alta.

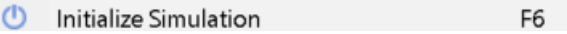
(Șerban & Alina)

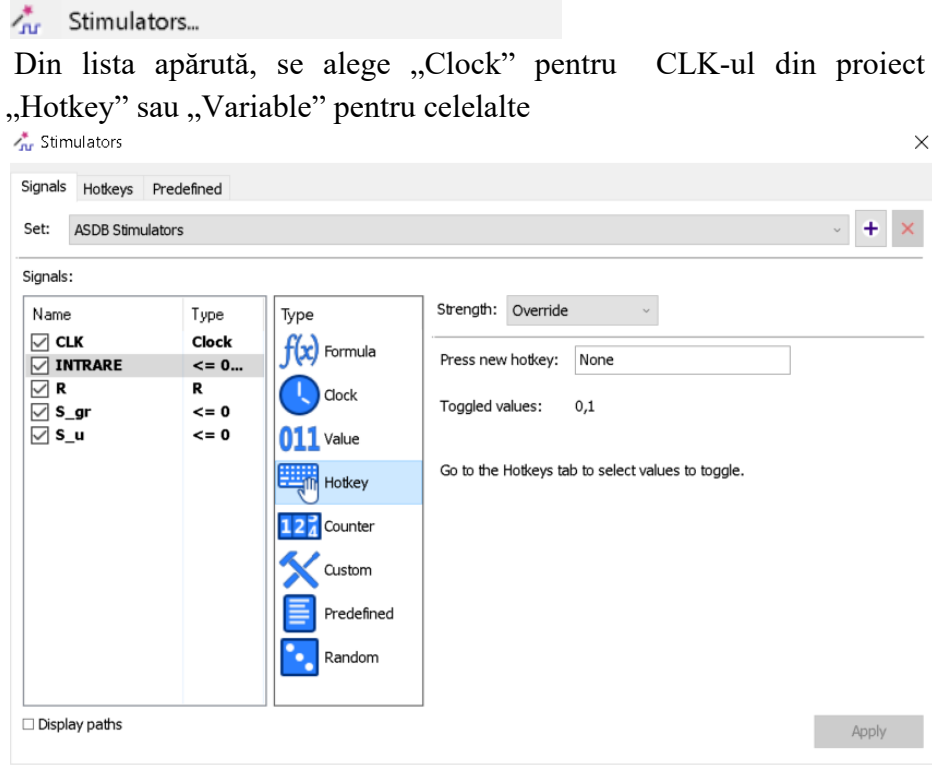
## 8. INSTRUCȚIUNI DE UTILIZARE ȘI ÎNTREȚINERE




### **Pentru Active-HDL:**

În Active-HDL proiectul se folosește astfel:

1. Se deschide programul Active-HDL
2. Se compilează cu ajutorul comenzii „Compile” din meniul „Design”
3. Se simulează cu „New waveform” (  ):
  - Se deschide meniul „Structure”, iar din bara de sus se selectează componenta pe care dorim s-o simulăm (componenta LIFT)

- Se inițializează simularea prin comanda „Initialize simulation” din meniul „Simulation” 
- Se selectează toate semnalele componente și se trag în Waveform
- Se adaugă fiecărui semnal de intrare câte un stimulator, dând click dreapta pe acesta și selectând din meniul deschis „Stimulators...”



- După ce toate acestea au fost alese, se apasă butonul „Run for” (  ) din meniu
- Începe selectarea valorilor pentru funcționare și se apasă din nou butonul „Run for” (  )
- Se urmăresc rezultatele obținute
- Se oprește apoi simularea prin apăsarea butonului „End” (  )

Din când în când programul mai necesită compilări și simulări pentru a ne asigura că nu s-au făcut modificări care pot da erori.

(Șerban & Alina)

## 9. POSSIBILITĂȚI DE DEZVOLTARE ULTERIOARE

Liftul este un automat foarte des folosit în zilele noastre, ușurând deplasarea persoanelor în clădirile cu multe etaje printr-o simplă apăsare a unui buton. În raport cu noile descoperiri tehnologice, proiectul nostru s-ar putea digitaliza mai mult, utilizând, spre exemplu, comenzi vocale pentru a selecta etajul sau pentru a chema liftul. Acest lucru ar facilita accesul mai ușor



pentru persoanele cu dizabilități (persoanele nevăzătoare) la utilizarea liftului. De asemenea, rapiditatea cu care liftul urcă ar putea crește, ajungând de la câteva secunde la câteva nanosecunde.

(Șerban & Alina)