**Universitatea Tehnică Cluj - Napoca**



**Proiectarea sistemelor numerice**

**Proiect: Mașina de spălat**

**Stundenți: Pricopi Andreea, Tocilă Emilia**

**Profesor îndrumător: Dr. Ing. Văcariu Lucia**

**Cuprins**

1. Specificația de proiect............................................................................ 3
   1. Tema proiectului........................................................................... 3
   2. Deciziile luate................................................................................. 4
2. Descrierea schemei bloc. Componente........................................... 5
3. Proiectare și implementare.................................................................. 8
   1. Descrierea schemei de detaliu................................................ 8
   2. Proiectarea componentelor ...................................................10
   3. Proiectarea de ansamblu......................................................... 29
   4. Simularea în VHDL...................................................................... 31
4. Lista componentelor.............................................................................. 35
5. Semnificația notațiilor............................................................................ 37
6. Justificarea soluției alese....................................................................... 39
7. Utilizare și rezultate................................................................................. 40
   1. Descrierea resurselor necesare (software și hardware) 40
   2. Descrierea utilizării proiectului................................................ 42
   3. Rezultatele obținute pe placă.................................................. 46
8. Posibilități de dezvoltare ulterioară.................................................. 49

**Specificația de proiect**

* **Tema proiectului**

Proiectați un automat simplu pentru comanda unei mașini de spălat, cu un mod manual și câteva moduri automate.

Inițial, automatul este într-o stare inactivă, cu ușa mașinii de spălat deschisă. Utilizatorul poate seta parametrii de funcționare manual (modul manual) sau poate selecta unul din modurile pre-programate.

În modul manual, se pot seta: temperatura (30, 40, 60 sau 90 C), viteza (800, 1000, 1200 rpm), selectare/anulare prespălare, clătire suplimentară. Timpul rulării programelor depinde de temperatura selectată (apa vine cu o temperatură de 15 C și se încălzește 1 C în 2 secunde) și de funcția selectată (prespălare- aceeași metodă ca spălarea principală, clătire suplimentară – clătire de două ori; aceste funcții sunt descrise în detaliu mai jos).

Modurile automate:

1. Spălare rapidă – 30 C, 1200 rpm, fără prespălare, fără clătire suplimentară
2. Cămăși – 60 C, 800 rpm, fără prespălare, fără clătire suplimentară
3. Culori închise – 40 C, 1000 rpm, fără prespălare, cu clătire suplimentară
4. Rufe murdare – 40 C, 1000 rpm, cu prespălare, fără clătire suplimentară
5. Antialergic – 90 C, 1200 rpm, fără prespălare, cu clătire suplimentară

Fiecare program conține etapele următoare:

1. Spălare principală
   1. se alimentează mașina cu apă
   2. se încălzește apa
   3. se rotește cu o viteză de 60 rpm timp de 20 de minute
   4. se evacuează apa
2. Clătire
   1. se alimeantează mașina cu apă
   2. se rotește cu o viteză de 120 rpm 10 minute
   3. se evacuează apa
3. Centrifugare
   1. se rotește cu viteza selectată 10 minute

Dacă este selectată prespălarea, se aplică aceeași metodă ca spălarea principală, cu excepția faptului că se rotește timp de 10 minute.   
Ușa se deblochează după primirea programului și se deschide cu un minut după terminarea sa. Mașina nu pornește cu ușa deschisă.  
În timp ce se selectează modul dorit, se afișează durata programului, iar după ce se pornește – timpul rămas (timpul se va afișa utilizând afișoare cu 7 segmente)

* **Decizii luate**

Începând cu realizarea schemei bloc, s-au hotărât următoarele aspecte:

* Timpul va fi recalculat și afișat în secunde, pentru o simulare mai rapidă
* Datele (temperatură, viteză) vor fi stocate (hardcodate) pe MUX-uri
* Etapele (stările – ex: activ, ușă blocată, alimentare, încălzire apă, rotire, presplălare etc) vor fi semnalizate pe FPGA prin leduri
* Selectarea modului, optiunea de prespălare și cea de clătire suplimentară, precum și deschiderea/închiderea ușii se vor face cu ajutorul switch-urilor
* Selectarea programului automat, a vitezei, a temperaturii, dar și activarea se vor face cu ajutorul butoanelor (primele 3 prin apăsări repetate ce vor reprezenta impulsuri de tact pentru numărătoare – detaliile vor fi prezentate ulterior în secțiunile aferente)

**Descrierea schemei bloc**

**Componente**

În continuare vom descrie, pe rând, funcționalitatea fiecărei componente, cu intrările și ieșirile aferente, care vor fi apoi prezentate pe larg în capitolul 5. Ordinea va fi alfabetică.

**Automatul** reprezintă unitatea de comandă. Acesta preia informațiile de la blocul selector (modul – automat/manual, respectiv temperatura, viteza, dacă se dorește prespălare sau clătire suplimentară) și de la senzorul de ușă. Odată cu activarea mașinii de spălat, blochează ușa, transmite informația către automatul de stări și către afișor (timpul rămas).

Intrări (AUTOMAT): CLK, START, SEL\_MODE, USA, BUTON\_AUTO, BUTON\_TEMPERATURA, BUTON\_VITEZA, PRESPALARE\_MANUAL, CLATIRE\_SUPL\_MANUAL, TEMPERATURA, VITEZA, PRESPALARE, CLATIRE\_SUPL, USA\_BLOCATA, TIMP\_TOTAL;

Ieșiri (AUTOMAT): ACTIV, ROTIRE, ALIMENTARE, INCALZIRE, EVACUARE, PRESPALARE\_OUT, CLATIRE\_SUPL\_OUT, A\_TO\_G, AN, TEMPERATURA, VITEZA, PRESPALARE, CLATIRE\_SUPL, USA\_BLOCATA, TIMP\_TOTAL;

**Automatul de stări** reprezintă componenta ce realizează trecerea dintr-o stare în alta, activează componentele electro-tehnice și mecanice (anexele) pentru duratele corespunzătoare de timp și ledurile corespunzătoare stărilor în care se află mașina la un anumit moment de timp. Diagrama de stare va fi prezentată la finalul descrierii componentelor.

Intrări (NEXT\_STATE\_LOGIC): START, PRESPALARE, CLATIRE\_SUPL, USA, CLK, TEMPERATURA;

Ieșiri (NEXT\_STATE\_LOGIC): PRESPALARE\_OUT, CLATIRE\_SUPL\_OUT, ACTIV, ROTIRE, ALIMENTARE, INCALZIRE, EVACUARE, USA\_BLOCATA;

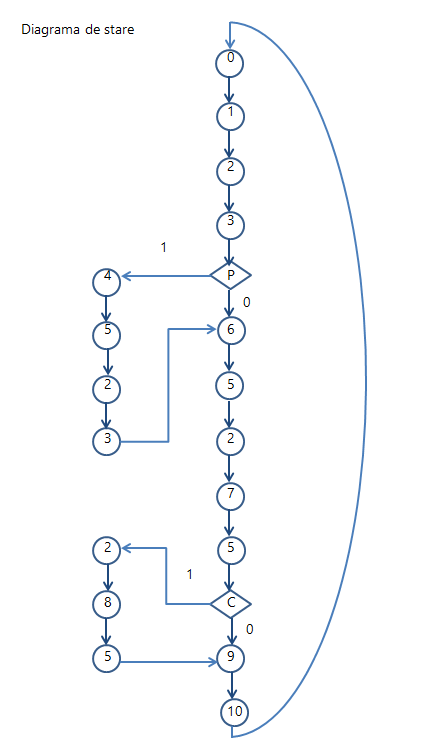
**Blocul afișor** înglobează componentele necesare afișării timpului total, înainte de activarea mașinii, respectiv a timpului rămas ulterior (în continuă scădere, până la 0). Acestea vor fi descrise pe larg în secțiunea 3.b, fiind reprezentate de o componentă ce calculează timpul total în funcție de selecții, un numărător ce numără descrescător, un divizor de frecvență pentru ca timpul să fie vizibil ochiului uman, un convertor de cod din binar în BCD, un decodificator BCD 7 segmente și o componentă cu logica aferentă activării anozilor și a catozilor.

Intrări (AFIȘOR): CLK, TIMP, UNITATI, ZECI, SUTE, MII;

Ieșiri (AFIȘOR): A\_TO\_G, AN;

**Blocul selector** este un ansamblu de componente (respectiv multiplexoare – dintre care unele ”hardcodate” cu informațiile cunoscute (temperatură, viteză), pentru a fi folosite ca memorii statice - și numărătoare) ce vor fi explicate și descrise în secțiunea 3.b. Blocul selector va trimite informațiile selectate de utilizator (modul, temperatura, viteza) automatului (automatul va folosi informațiile selectate până la momentul apăsării butonului de START, adică al activării mașinii de spălat).  
  
Intrări (GET\_DATA): MODE, BUTON\_P, BUTON\_T, BUTON\_V, PRESPALARE\_MANUAL, CLATIRE\_MANUAL;

Ieșiri (GET\_DATA): ON\_T, BUTON\_V, PRESPALARE\_MANUAL, CLATIRE\_MANUAL, VITEZA, TEMPERATURA;



Legendă diagramă

0 = Inactiv

1 = Blocare ușă

2 = Alimentare

3 = Încălzire

4 = Prespălare (rotire)

5 = Evacuare

6 = Spălare (rotire)

7 = Clătire (rotire)

8 = Clătire suplimentar (rotire)

9 = Centrifugare (rotire)

10 = Deblocare ușă

P = Prespălare

C = Clătire suplimentară

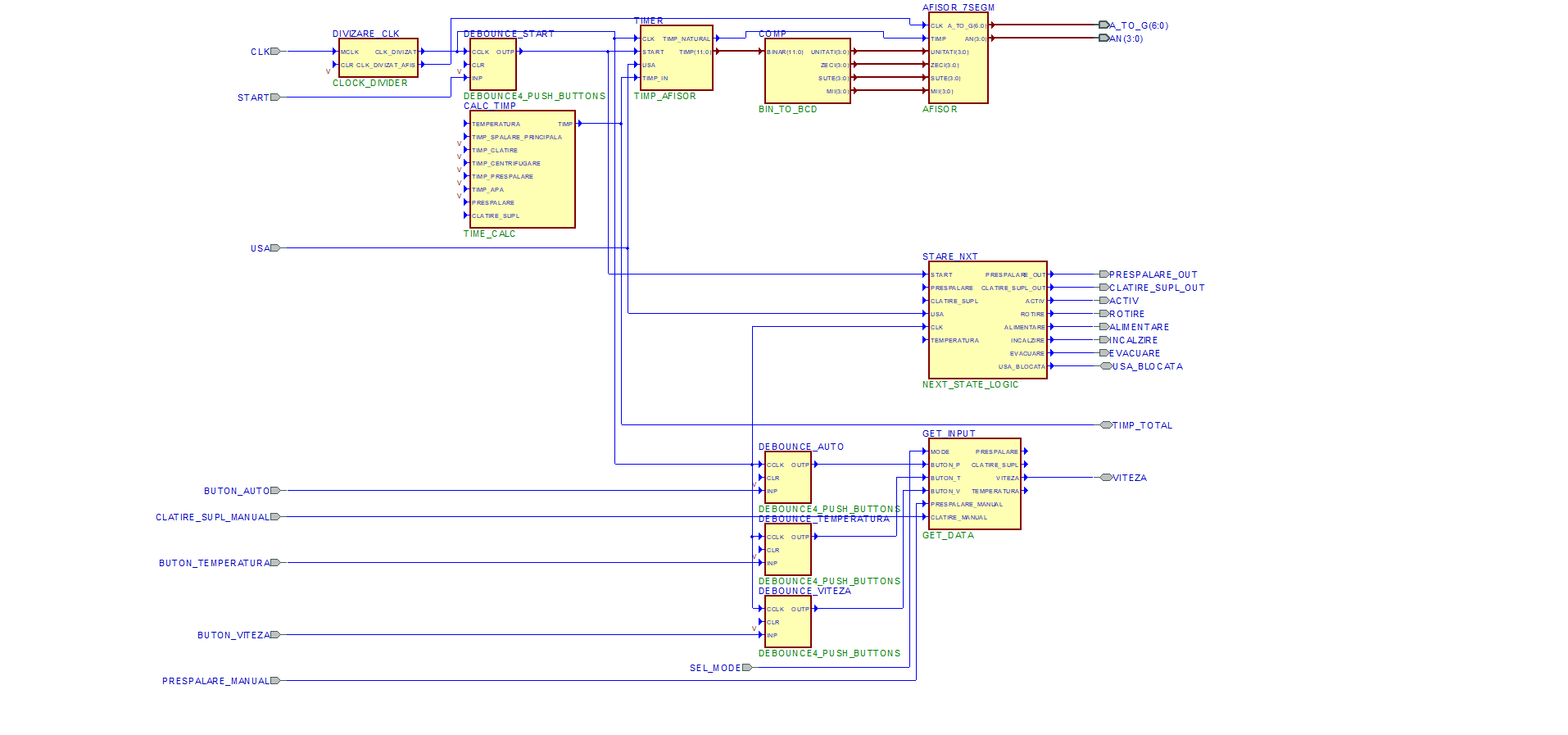
**Proiectare și implementare**

* **Schema de detaliu**

În continuare vom descrie funcționarea în detaliu, pas cu pas, a proiectului, în funcție de componente.

Pentru ca mașina de spălat să funcționeze, aceasta trebuie să cunoască temperatura și viteza dorite de utilizator, dar și dacă opțiunile de prespălare și clătire suplimentară vor fi utilizate. Pentru aceasta, după cum este specificat în ”Tema proiectului”, utilizatorul poate selecta parametrii manual sau poate alege un mod automat care are în vedere toate aspectele. Automatul se află inițial în ”Mod manual” și primește, dacă nu se face nicio modificare înainte de a se apăsa butonul START, temperatura de 30 C, viteza de 800 rpm, fără prespălare și fără clătire suplimentară.

Așadar, avem un automat în care sunt instanțiate principalele componente- divizorul de clock, componenta de debounce, blocul selector, automatul de stări și blocul afișor.

Scje,a

* **Proiectarea componentelor**

Vom prezenta componentele, în ordinea instanțierii lor în ”AUTOMAT”.

Divizorul de frecvență

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

entity CLOCK\_DIVIDER is

port (MCLK, CLR: in STD\_LOGIC;

CLK\_DIVIZAT, CLK\_DIVIZAT\_AFIS: out STD\_LOGIC);

end CLOCK\_DIVIDER;

architecture CLOCK\_DIVIDER\_ARCH of CLOCK\_DIVIDER is

signal Q: STD\_LOGIC\_VECTOR(27 downto 0);

begin

process(MCLK, CLR)

begin

if (CLR = '1' or Q = "U") then

-- resetare la 0, in hexazecimal

Q <= X"0000000";

elsif (MCLK'EVENT and MCLK = '1' and MCLK'LAST\_VALUE = '0') then

--s-a primit impuls de tact, numaram

Q <= Q + 1;

end if;

end process;

--alegem frecventele dorite

CLK\_DIVIZAT <= Q(23); -- frecventa optima pentru functionarea pe placa, CLK/2^24

CLK\_DIVIZAT\_AFIS <= Q(17); -- vom avea nevoie de aprox. 190 Hz pentru afisor, deci Q(17)

end CLOCK\_DIVIDER\_ARCH;

Placa de dezvoltare FPGA are un oscilator de 50MHz, însă vom folosi o frecvență mai mică pentru parcurgerea stărilor și una diferită pentru afișor. Principiul pe care îl folosim este clasic - un numărător, în cazul nostru pe 28 de biți. Fiecare bit are o frecvență de 2 ori mai mică decât bitul precedent. Am observat și folosit frecvențele optime pentru noi, respectiv f/2^24 și f/2^18, unde f este frecvența inițială, de 50 MHz.

Debounce-ul

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity DEBOUNCE4\_PUSH\_BUTTONS is

port (CCLK, CLR: in STD\_LOGIC; --clk divizat

INP: in STD\_LOGIC;

OUTP: out STD\_LOGIC);

end DEBOUNCE4\_PUSH\_BUTTONS;

architecture DEBOUNCE4\_ARCH of DEBOUNCE4\_PUSH\_BUTTONS is

signal DELAY1, DELAY2, DELAY3: STD\_LOGIC; -- iesirile bistabilelor D

begin

process(CCLK, CLR)

begin

if (CLR = '1') then

--resetare

DELAY1 <= '0';

DELAY2 <= '0';

DELAY3 <= '0';

elsif (CCLK'EVENT and CCLK = '1' and CCLK'LAST\_VALUE = '0') then

--intarzierea

DELAY1 <= INP;

DELAY2 <= DELAY1;

DELAY3 <= DELAY2;

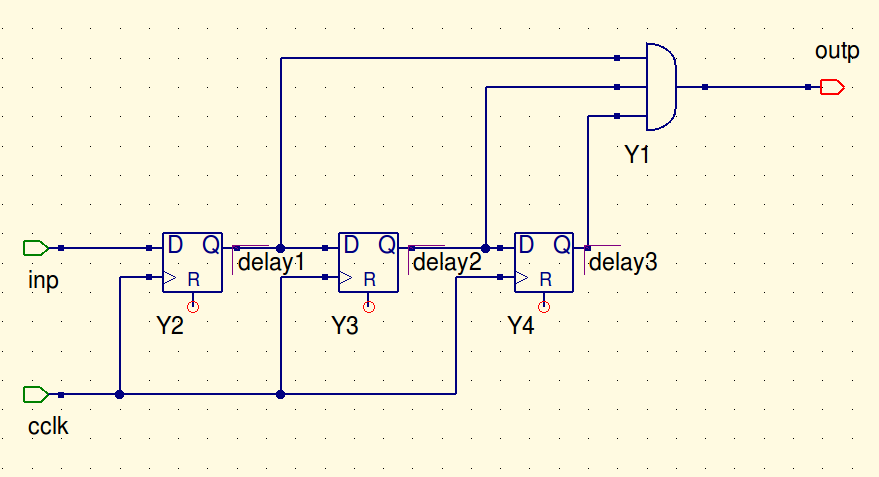
end if;

end process;

OUTP <= DELAY1 and DELAY2 and DELAY3;

end DEBOUNCE4\_ARCH;

Nevoia de debounce se datorează faptului că utilizăm cele 4 butoane de pe placă, care nu trimit semnale perfect dreptunghiulare, ci formate din ”spike-uri”, elementele din interior având nevoie de o anumită perioadă de timp pentru a se stabiliza. Rezolvarea constă în ignorarea a câteva milisecunde din semnal și generarea unuia perfect dreptunghiular pe ieșiri.



GET\_DATA (Bloc selector)

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

entity GET\_DATA is

port (MODE, BUTON\_P, BUTON\_T, BUTON\_V, PRESPALARE\_MANUAL, CLATIRE\_MANUAL: in STD\_LOGIC;  
 --iesirile se vor transmite automatului

PRESPALARE, CLATIRE\_SUPL: out STD\_LOGIC; -- 1 = se dorește, 0 = nu se dorește

VITEZA, TEMPERATURA: out INTEGER);

end GET\_DATA;

architecture GET\_DATA\_ARCH of GET\_DATA is

--programul selectat

signal OUT\_CTR\_AUTO: STD\_LOGIC\_VECTOR(2 downto 0);

--temperatura

signal T\_OUT\_MANUAL, T\_OUT\_AUTO, SEL\_TEMPERATURA: STD\_LOGIC\_VECTOR(1 downto 0);

--viteza

signal V\_OUT\_MANUAL, V\_OUT\_AUTO, SEL\_VITEZA: STD\_LOGIC\_VECTOR(1 downto 0);

--prespalare si clatire intermediare

signal PRESPALARE\_AUTO, CLATIRE\_SUPL\_AUTO: STD\_LOGIC;

component COUNTER\_GET --numaratorul

generic (limita: INTEGER := 4;

N : integer := 1);

port (CLR, CLK: in STD\_LOGIC;

Q: out STD\_LOGIC\_VECTOR((N - 1) downto 0));

end component;

component MUX\_4TO1\_INTEGER –mux-ul final

port (SEL: in STD\_LOGIC\_VECTOR(1 downto 0);

I0, I1, I2, I3: in INTEGER;

Y: out INTEGER);

end component;

component MUX\_PROGRAM –mux-ul cu programele automate pentru viteza si temperatura

port (SEL: in STD\_LOGIC\_VECTOR(2 downto 0);

I0, I1, I2, I3, I4: in STD\_LOGIC\_VECTOR(1 downto 0);

Y: out STD\_LOGIC\_VECTOR(1 downto 0));

end component;

component MUX\_PROGRAM\_PRESP\_CLT –mux-ul cu optiunile automate

port (SEL: in STD\_LOGIC\_VECTOR(2 downto 0);

I0, I1, I2, I3, I4: in STD\_LOGIC;

Y: out STD\_LOGIC);

end component;

component MUX\_2TO1 --

generic (N: integer := 2); --numar de biti

port (SEL: in STD\_LOGIC;

A, B: in STD\_LOGIC\_VECTOR((N - 1) downto 0);

Y: out STD\_LOGIC\_VECTOR((N - 1) downto 0));

end component;

component MUX\_2TO1\_BIT

port (SEL: in STD\_LOGIC;

A, B: in STD\_LOGIC;

Y: out STD\_LOGIC);

end component;

begin

--se alege programul automat, in cazul în care este selectat modul automat

--numaratorul COUNTER\_GET va folosi butonul de program BUTON\_P ca impuls de tact

--OUT\_CTR\_AUTO va reprezenta numarul programului ales

NR\_SEL\_AUTO: COUNTER\_GET generic map (4, 3) port map ('0', BUTON\_P, OUT\_CTR\_AUTO);

--se alege temperatura

--numaratorul COUNTER\_GET va folosi butonul de temperatura BUTON\_T ca impuls de tact

--T\_OUT\_MANUAL va reprezenta numarul temperaturii alese manual (0 = 30, 1 = 40, 2 = 60, 3 = 90)

NR\_SEL\_MANUAL\_TEMP: COUNTER\_GET generic map (3, 2) port map ('0', BUTON\_T, T\_OUT\_MANUAL);

--T\_OUT\_AUTO va reprezenta numarul temperaturii corespunzator programului selectat automat,

--OUT\_CTR\_AUTO

MUX\_AUTO\_TEMP: MUX\_PROGRAM port map (OUT\_CTR\_AUTO, "00", "10", "01", "01", "11", T\_OUT\_AUTO);

--selectia acestui MUX este reprezentata de MOD (0 = manual, 1 = automat)

--intararea de date I0 este numarul temperaturii selectate manual

--intrarea de date I1 este numarul temperaturii selectate prin alegerea unui program automat

--iesirea SEL\_TEMPERATURA va reprezenta selectia pentru urmatorul MUX, respectiv numarul aferent

--temperaturii dorite

MUX\_TEMP: MUX\_2TO1 generic map (2) port map (MODE, T\_OUT\_MANUAL, T\_OUT\_AUTO, SEL\_TEMPERATURA);

--privim temperaturile ca fiind asezate intr-o lista ordonata, pe intrarile de date ale acestui MUX

--SEL\_TEMPERATURA va reprezenta numarul curent al temperaturii

MUX\_MANUAL\_TEMP: MUX\_4TO1\_INTEGER port map (SEL\_TEMPERATURA, 30, 40, 60, 90, TEMPERATURA);

--se alege viteza, conform aceluiasi proces de mai sus

NR\_SEL\_MANUAL\_VIT: COUNTER\_GET generic map (2, 2) port map ('0', BUTON\_V, V\_OUT\_MANUAL);

MUX\_AUTO\_VIT: MUX\_PROGRAM port map (OUT\_CTR\_AUTO, "10", "00", "01", "01", "10", V\_OUT\_AUTO);

MUX\_VITEZA: MUX\_2TO1 generic map (2) port map (MODE, V\_OUT\_MANUAL, V\_OUT\_AUTO, SEL\_VITEZA);

MUX\_MANUAL\_VIT: MUX\_4TO1\_INTEGER port map (SEL\_VITEZA, 800, 1000, 1200, 0, VITEZA);

--se aleg optiunile de prespalare si clatire suplimentare, conform aceluiasi proces ca selectarea

--temperaturii

MUX\_AUTO\_PRESP: MUX\_PROGRAM\_PRESP\_CLT port map (OUT\_CTR\_AUTO, '0', '0', '0', '1', '0', PRESPALARE\_AUTO);

MUX\_AUTO\_CLT\_SUP: MUX\_PROGRAM\_PRESP\_CLT port map (OUT\_CTR\_AUTO, '0', '0', '1', '0', '1', CLATIRE\_SUPL\_AUTO);

MUX\_MANUAL\_PRESP: MUX\_2TO1\_BIT port map (MODE, PRESPALARE\_MANUAL, PRESPALARE\_AUTO, PRESPALARE);

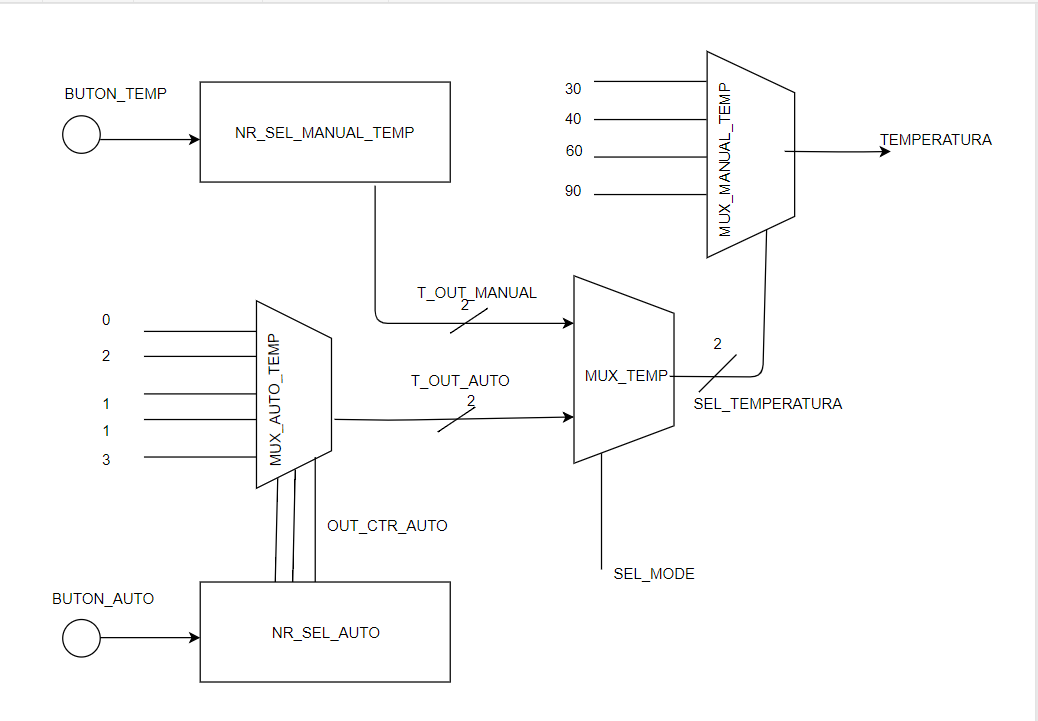
MUX\_MANUAL\_CLT\_SUP: MUX\_2TO1\_BIT port map (MODE, CLATIRE\_MANUAL, CLATIRE\_SUPL\_AUTO, CLATIRE\_SUPL);

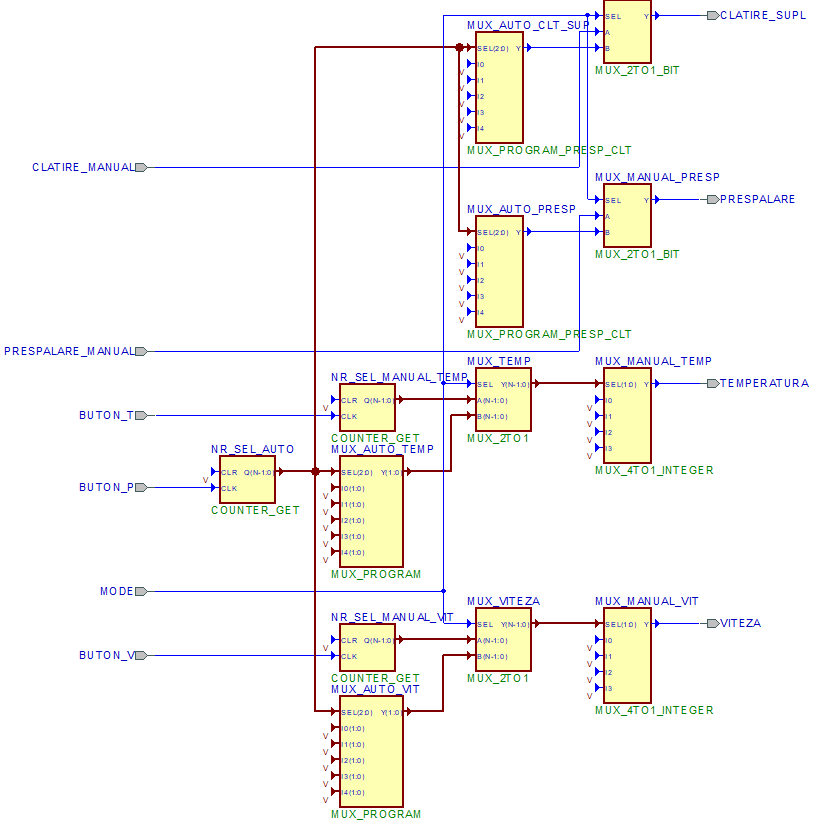
end GET\_DATA\_ARCH;

GET\_INPUT (instanța GET\_DATA) este componenta ce va transmite parametrii TEMPERATURĂ, VITEZĂ, PRESPĂLARE ȘI CLĂTIRE\_SUPL , selectați de utilizator, automatului. În continuare vom descrie principiul de selecție al temperaturii, selecția vitezei și a celor două opțiuni (prespălare și clătire suplimentară) fiind identice.

Pentru o mai bună înțelegere a procesului, îl vom descrie secvențial, amintind, totuși, că execuția este una concurentă și, deci, ordinea instrucțiunilor nu influențează rezultatul final.

Mai întâi, vom așeza cele 4 temperaturi posibile (30, 40, 60, 90) pe intrările de date ale unui MUX 4:1, MUX\_MANUAL\_TEMP, cu intrările și ieșirea (temperatura) de tipul INTEGER. Vom presupune că le alegem după un ”Nr. Crt” (număr curent), care poate fi respectiv 0, 1, 2 sau 3. Acest număr va fi dat de selecția SEL\_TEMPERATURĂ, de tipul STD\_LOGIC\_VECTOR(1 downto 0), deci, respectiv ”00”, ”01”, ”10”, ”11”.   
Știm că temperatura poate fi aleasă fie manual, de către utilizator, fie prin alegerea unui program automat. Așadar, vom folosi un MUX 2:1, cu calea de date pe 2 biți, MUX\_TEMP, cu selecția MODE (0 – manual, 1 – automat). Ieșirea acestui MUX va fi chiar SEL\_TEMPERATURĂ, adică numărul curent pentru temperatura pe care vrem să o selectăm din ”listă”. Intrările vor fi, respectiv, A- numărul introdus manual și B- numărul aferent temperaturii din listă, conform programului selectat (ex: pentru programul ”Cămăși” avem temperatura 60 C, deci numărul 2). Aceste numere din urmă vor fi ”stocate” pe același principiu de listă ca temperatura, în MUX\_AUTO\_TEMP. Ce ne-a rămas acum de făcut este să stabilim cum vom introduce numărul. Am decis că vom aveam două butoane, unul pentru modul automat (care va prelua informația doar dacă modul va fi 1, adică automat) și unul pentru temperatura introdusă manual (care va prelua informația doar dacă modul va fi 0, adică manual). Semnalul generat de apăsarea acestor butoane va reprezenta impulsul de tact pentru două numărătoare, ale căror ieșiri vor fi A și B, intrările de date din multiplexorul MUX\_TEMP. Numărătoarele folosite sunt NR\_SEL\_AUTO și NR\_SEL\_MANUAL\_TEMP. Schemele sunt următoarele:





Bloc selector (GET\_INPUT, instanță a GET\_DATA)

Calculatorul de timp

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

use IEEE.STD\_LOGIC\_UNSIGNED.all;

entity TIME\_CALC is

port (TEMPERATURA, TIMP\_SPALARE\_PRINCIPALA, TIMP\_CLATIRE, TIMP\_CENTRIFUGARE, TIMP\_PRESPALARE, TIMP\_APA: in INTEGER;

PRESPALARE, CLATIRE\_SUPL: in STD\_LOGIC;

TIMP: out NATURAL);

end TIME\_CALC;

architecture TIME\_CALC\_ARCH of TIME\_CALC is

begin

process (TEMPERATURA, PRESPALARE, CLATIRE\_SUPL)

--timpul de incalzire, variabil in functie de temperatura aleasa

variable TIMP\_INCALZIRE: INTEGER := 0;

--timpul final, exprimat in secunde

variable SUMA\_TIMP: INTEGER := 0;

begin

case TEMPERATURA is

--memoram timpul de incalzire in functie de temperatura

--stiind ca apa vine la 15 C si se incalzeste cu 1 C in 2 secunde

--timpul este aici impartit la 10, pentru simulare mai rapida

when 30 => TIMP\_INCALZIRE := 3; --30 de secunde in realitate

when 40 => TIMP\_INCALZIRE := 5; --50 de secunde

when 60 => TIMP\_INCALZIRE := 9; --90 de secunde

when 90 => TIMP\_INCALZIRE := 15; -- 150 de secunde

when others => TIMP\_INCALZIRE := 0; --eroare

end case;

SUMA\_TIMP := TIMP\_INCALZIRE + TIMP\_SPALARE\_PRINCIPALA + TIMP\_CLATIRE + TIMP\_CENTRIFUGARE + 4 \* TIMP\_APA;

if (PRESPALARE = '1') then

--daca se alege prespalare, adaugam timpul de prespalare si dublam timpul de --alimentare si evacuare

SUMA\_TIMP := SUMA\_TIMP + TIMP\_PRESPALARE + 2 \* TIMP\_APA;

end if;

if (CLATIRE\_SUPL = '1') then

--daca se alege clatire suplimentara, adaugam timpul de clatire si

--dublam timpul de alimentare si evacuare

SUMA\_TIMP := SUMA\_TIMP + TIMP\_CLATIRE + 2 \* TIMP\_APA + TIMP\_INCALZIRE;

end if;

--timpul final

TIMP <= SUMA\_TIMP;

end process;

end TIME\_CALC\_ARCH;

Pentru a afișa timpul, avem nevoie de o componentă care să calculeze durata totală a programului. Aceasta practic însumează timpii specifici oricărui program (pentru că orice spălare va conține două alimentări și evacuări, spălare principală, clătire și centrifugare), care sunt declarate ca fiind constante în automat și trimise de acesta ca intrări pentru această componentă, cu mențiunea că minutele sunt, de fapt, secunde aici, pentru o simulare mai rapidă. Știm că timpul variază în funcție de temperatură și că timpul de încălzire are o durată de (TEMPERATURĂ – 15) x 2 secunde. Am decis să împărțim la 10 acest rezultat, tot pentru rapiditate în simulare. Am calculat, deci, timpul de încălzire pentru toate cele 4 temperaturi posibile și am folosit o instrucțiune ”case” pentru a-l adăuga pe cel aferent la timpul total. De asemenea, timpul variază și în funcție de cele două opțiuni, prespălare și clătire suplimentară, cazuri pe care le-am tratat separat, cu instrucțiuni decizionale ”if”.

Automatul de stări (STARE\_NXT)

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity NEXT\_STATE\_LOGIC is

port(START, PRESPALARE, CLATIRE\_SUPL, USA, CLK: in STD\_LOGIC;

TEMPERATURA: in INTEGER;

PRESPALARE\_OUT, CLATIRE\_SUPL\_OUT, ACTIV, ROTIRE, ALIMENTARE, INCALZIRE, EVACUARE, USA\_BLOCATA: out STD\_LOGIC);

end NEXT\_STATE\_LOGIC;

architecture NEXT\_STATE\_LOGIC\_ARCH of NEXT\_STATE\_LOGIC is

--stari

type state\_type is (inactiv, blocare\_usa, alim\_apa, incalzire\_apa, presp, evac, spalare, clt, clatire\_sup, centrifugare, deblocare\_usa);

signal STARE: state\_type := inactiv;

--contor pentru timer (durata fiecarei etape/stari)

signal COUNT: INTEGER := 0;

--timpi

constant TIMP\_SPALARE\_PRINCIPALA: INTEGER := 120;

constant TIMP\_CLATIRE: INTEGER := 60;

constant TIMP\_CENTRIFUGARE: INTEGER := 60;

constant TIMP\_PRESPALARE: INTEGER := 60;

constant TIMP\_APA: INTEGER := 6;

begin

next\_state: process(CLK)

variable TIMP\_INCALZIRE: INTEGER := 0;

variable CONDITII: STD\_LOGIC\_VECTOR(3 downto 0) := "0000"; --descriere in documentatie

begin

case TEMPERATURA is

when 30 => TIMP\_INCALZIRE := 3;

when 40 => TIMP\_INCALZIRE := 5;

when 60 => TIMP\_INCALZIRE := 9;

when 90 => TIMP\_INCALZIRE := 15;

when others => TIMP\_INCALZIRE := 0;

end case;

if (CLK = '1' and CLK'EVENT and CLK'LAST\_VALUE = '0') then

if (STARE = inactiv and START = '1' and USA = '1') then --daca masina a fost --pornita si usa e inchisa

STARE <= blocare\_usa; --blocare usa

end if;

case STARE is

when blocare\_usa =>

STARE <= alim\_apa; --alimentare apa

when alim\_apa =>

if COUNT = TIMP\_APA - 1 then

if CONDITII(1) = '0' then --nu am spalat inca

STARE <= incalzire\_apa; --incalzire apa

elsif CONDITII(2) = '0' then STARE <= clt; --clatire (am spalat, nu am clatit)

elsif (CLATIRE\_SUPL = '1' and CONDITII(3) = '0') then

STARE <= clatire\_sup; --clatire suplimentara (am spalat,   
 --am clatit, n-am clatit suplimentar, avem clatire suplimentara de facut)

end if;

COUNT <= 0;

else COUNT <= COUNT + 1; end if;

when incalzire\_apa =>

if COUNT = TIMP\_INCALZIRE - 1 then

if PRESPALARE = '1' then --am incalzit apa

if CONDITII(0) = '0' then --trebuie sa prespalam

STARE <= presp; --prespalare

else STARE <= spalare; --am prespalat, urmeaza spalare

end if;

else STARE <= spalare; --nu trebuie sa prespalam

end if;

COUNT <= 0;

else COUNT <= COUNT + 1;

end if;

when presp =>

if COUNT = TIMP\_PRESPALARE - 1 then

STARE <= evac; --evacuare

CONDITII(0) := '1';

COUNT <= 0;

else COUNT <= COUNT + 1;

end if;

when evac =>

if COUNT = TIMP\_APA - 1 then

if (CLATIRE\_SUPL = '0' and CONDITII(2) = '1') or (CLATIRE\_SUPL = '1' and CONDITII(3) = '1') then

STARE <= centrifugare; --centrifugare

else STARE <= alim\_apa; --alimentare apa

end if;

COUNT <= 0;

else COUNT <= COUNT + 1;

end if;

when spalare =>

if COUNT = TIMP\_SPALARE\_PRINCIPALA- 1 then

STARE <= evac; --evacuare

CONDITII(1) := '1';

COUNT <= 0;

else COUNT <= COUNT + 1;

end if;

when clt =>

if COUNT = TIMP\_CLATIRE - 1 then

STARE <= evac; --evacuare

CONDITII(2) := '1';

COUNT <= 0;

else COUNT <= COUNT + 1;

end if;

when clatire\_sup =>

if COUNT = TIMP\_CLATIRE - 1 then

STARE <= evac; --evacuare

CONDITII(3) := '1';

COUNT <= 0;

else COUNT <= COUNT + 1;

end if;

when centrifugare =>

if COUNT = TIMP\_CENTRIFUGARE - 1 then

STARE <= deblocare\_usa; --deblocare\_usa

COUNT <= 0;

else COUNT <= COUNT + 1;

end if;

when deblocare\_usa =>

if COUNT = 6 - 1 then

STARE <= inactiv; --inactiv

COUNT <= 0;

else COUNT <= COUNT + 1;

end if;

when others =>

CONDITII := "0000";

end case;

end if;

end process;

outputs: process(STARE)

begin

case STARE is

when inactiv =>

ACTIV <= '0';

ROTIRE <= '0';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0';

INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '0';

when blocare\_usa =>

ACTIV <= '1';

ROTIRE <= '0';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0';

INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '1';

when alim\_apa =>

ACTIV <= '1';

ROTIRE <= '0';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '1';

INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '1';

when incalzire\_apa =>

ACTIV <= '1';

ROTIRE <= '0';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0';

INCALZIRE <= '1';

EVACUARE <= '0';

USA\_BLOCATA <= '1';

when presp =>

ACTIV <= '1';

ROTIRE <= '1';

PRESPALARE\_OUT <= '1';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0';

INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '1';

when evac =>

ACTIV <= '1';

ROTIRE <= '0';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0';

INCALZIRE <= '0';

EVACUARE <= '1';

USA\_BLOCATA <= '1';

when spalare => ACTIV <= '1'; ROTIRE <= '1';

PRESPALARE\_OUT <= '0'; CLATIRE\_SUPL\_OUT <= '0'; ALIMENTARE <= '0'; INCALZIRE <= '0';

EVACUARE <= '0'; USA\_BLOCATA <= '1';

when clt => ACTIV <= '1'; ROTIRE <= '1';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0'; INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '1';

when clatire\_sup =>

ACTIV <= '1';

ROTIRE <= '1';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '1';

ALIMENTARE <= '0';

INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '1';

when centrifugare =>

ACTIV <= '1';

ROTIRE <= '1';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0';

INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '1';

when deblocare\_usa =>

ACTIV <= '1';

ROTIRE <= '0';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0';

INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '1';

when others =>

ACTIV <= '0';

ROTIRE <= '0';

PRESPALARE\_OUT <= '0';

CLATIRE\_SUPL\_OUT <= '0';

ALIMENTARE <= '0';

INCALZIRE <= '0';

EVACUARE <= '0';

USA\_BLOCATA <= '0';

end case;

end process;

end NEXT\_STATE\_LOGIC\_ARCH;

Logica stărilor este cea descrisă în diagrama de stare. Aceasta este completată de durata stărilor, care se realizează cu ajutorul unui numărător ce numără până la o anumită limită, dată de timpul necesar, și anume de timpul exprimat de constanta aferentă stării. Logica ieșirilor va activa leduri diferite pentru acțiunile efectuate de mașină. Ce este important este faptul că prespălarea, spălarea, clătirea și clătirea suplimentară vor avea nevoie de alimentare și evacuare, dar numai unele vor avea nevoie și de încălzirea apei. De aceea vom folosi variabila vectorială CONDITII. Cand ajungem într-o astfel de stare, vom verifica condițiile necesare și o vom modifica pe cea aferentă, astfel:

CONDITII(0) = am prespălat

CONDITII(1) = am spălat

CONDITII(2) = am clătit

CONDITII(3) = am clătit suplimentar

Blocul afișor

AFIȘOR

entity AFISOR is

port (CLK: in STD\_LOGIC;

TIMP: in NATURAL;

UNITATI, ZECI, SUTE, MII: in STD\_LOGIC\_VECTOR(3 downto 0);

A\_TO\_G: out STD\_LOGIC\_VECTOR(6 downto 0);

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

end AFISOR;

architecture AFISOR\_ARH of AFISOR is

signal DIGIT: NATURAL;

signal COUNT: STD\_LOGIC\_VECTOR(1 downto 0) := "00";

signal AEN: STD\_LOGIC\_VECTOR(3 downto 0) := "0001"; --cifra 0 mereu pornita

component BCD\_7\_DECODER

port(inputBCD : in NATURAL;

outputBCD : out STD\_LOGIC\_VECTOR(6 downto 0));

end component;

begin

ACTIVARE\_ANOZI:process(CLK)

begin

if (TIMP <= 9) then

AEN(0) <= '1';

AEN(1) <= '0';

AEN(2) <= '0';

AEN(3) <= '0';

elsif (TIMP < 100) then

AEN(0) <= '1';

AEN(1) <= '1';

AEN(2) <= '0';

AEN(3) <= '0';

elsif (TIMP < 1000) then

AEN(0) <= '1';

AEN(1) <= '1';

AEN(2) <= '1';

AEN(3) <= '0';

elsif (TIMP < 10000) then

AEN(0) <= '1';

AEN(1) <= '1';

AEN(2) <= '1';

AEN(3) <= '1';

end if;

end process ACTIVARE\_ANOZI;

--numarator pe 2 biti

CTR: process(CLK)

begin

if (COUNT = "UU") then

COUNT <= "00";

elsif (CLK'EVENT and CLK = '1' and CLK'LAST\_VALUE = '0') then

COUNT <= COUNT + 1;

end if;

end process;

--MUX4

with COUNT select

DIGIT <= to\_integer(unsigned(unitati)) when "00", to\_integer(unsigned(zeci)) when "01", to\_integer(unsigned(sute)) when "10", to\_integer(unsigned(mii)) when others;

--BCD\_7SEGM

BCD: BCD\_7\_DECODER port map (DIGIT, A\_TO\_G);

--selectare anod

ANODE: process(COUNT)

begin

if (AEN(conv\_integer(COUNT)) = '1') then

AN <= (others => '1');

AN(conv\_integer(COUNT)) <= '0';

else

AN <= "1111";

end if;

end process;

end AFISOR\_ARH;

Convertor de cod binar – BCD

entity BIN\_TO\_BCD is

port(BINAR: in STD\_LOGIC\_VECTOR(11 downto 0);

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

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

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

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

end BIN\_TO\_BCD;

architecture A\_BIN\_TO\_BCD of BIN\_TO\_BCD is

begin

TO\_BCD:process(BINAR)

variable TEMP: STD\_LOGIC\_VECTOR(11 downto 0);

variable BCD: UNSIGNED(15 downto 0) := (others => '0');

--mii: bcd(15 downto 12); sute: bcd(11 donwto 8); zeci: bcd(7 downto 4); unitati: bcd(3 downto 0)

begin

BCD := (others => '0');

TEMP(11 downto 0) := BINAR;

for I in 0 to 11 loop

if BCD(3 downto 0) > 4 then

BCD(3 downto 0) := BCD(3 downto 0) + 3;

end if;

if BCD(7 downto 4) > 4 then

BCD(7 downto 4) := BCD(7 downto 4) + 3;

end if;

if BCD(11 downto 8) > 4 then

BCD(11 downto 8) := BCD(11 downto 8) + 3;

end if;

BCD := BCD(14 downto 0) & TEMP(11);

TEMP := TEMP(10 downto 0) & '0';

end loop;

UNITATI <= STD\_LOGIC\_VECTOR(BCD(3 downto 0));

ZECI <= STD\_LOGIC\_VECTOR(BCD(7 downto 4));

SUTE <= STD\_LOGIC\_VECTOR(BCD(11 downto 8));

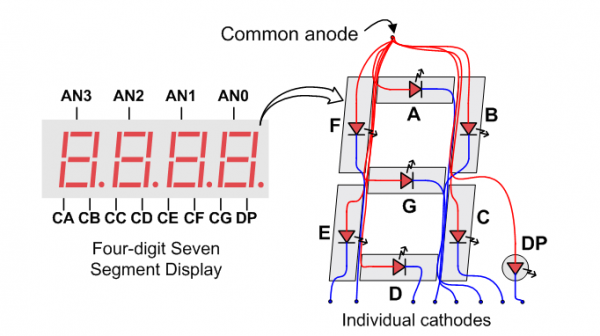
MII <= STD\_LOGIC\_VECTOR(BCD(15 downto 12));

end process TO\_BCD;

end A\_BIN\_TO\_BCD;

Acestea două din urmă vor fi tratate împreună, deoarece fac parte din componenta Bloc afișor. Mai întâi, pentru afișarea unui număr de mai multe cifre, am divizat numărul în cifre – mii, sute, zeci și unități în componenta de conversie. Pentru aceasta, am utilizat principiul ”Double-Dabble”: se ia valoarea binară a numărului pe care dorim să îl convertim, pe 12 biți, și un vector de 16 biți, care se inițializează cu 0. În acest vector, fiecare 4 biți vor reprezenta o cifră astfel : biții 15-12 cifra miilor, 11-8 cifra sutelor, 7-4 cifra zecilor, 3-0 cifra unităților. Se șiftează câte un bit din numărul binar în vector și, de fiecare dată când o cifră (grupare aferentă de 4 biți) este cel puțin egală cu 5, se adaugă 3 la aceasta. Am folosit această abordare, deoarece una matematică, prin împărțiri la 10, nu ar fi fost sintetizabilă.

Afișorul propriu-zis funcționează astfel: o instrucțiune case verifică cifra de afișat (preluată de la convertorul de cod) și aprinde segmentele specifice acesteia. Problema afișorului este că afișează același conținut pe toate cele 4 blocuri și, din această cauză, am avut nevoie de o logică de activare a anozilor.



* **Proiectarea de ansamblu**

Automatul

entity AUTOMAT is

port (CLK, START, SEL\_MODE, USA, BUTON\_AUTO, BUTON\_TEMPERATURA, BUTON\_VITEZA, PRESPALARE\_MANUAL, CLATIRE\_SUPL\_MANUAL: in STD\_LOGIC;

TEMPERATURA, VITEZA: inout INTEGER;

PRESPALARE, CLATIRE\_SUPL, USA\_BLOCATA: inout STD\_LOGIC;

TIMP\_TOTAL: inout NATURAL;

ACTIV: out STD\_LOGIC;

ROTIRE, ALIMENTARE, INCALZIRE, EVACUARE, PRESPALARE\_OUT, CLATIRE\_SUPL\_OUT: out STD\_LOGIC;

A\_TO\_G: out STD\_LOGIC\_VECTOR(6 downto 0);

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

end AUTOMAT;

architecture AUTOMAT\_ARCH of AUTOMAT is

--constante timpi

--minutele sunt exprimare in secunde, pentru o simulare mai rapida

constant TIMP\_SPALARE\_PRINCIPALA: INTEGER := 120;

constant TIMP\_CLATIRE: INTEGER := 60;

constant TIMP\_CENTRIFUGARE: INTEGER := 60;

constant TIMP\_PRESPALARE: INTEGER := 60;

constant TIMP\_APA: INTEGER := 6;

--clock-uri divizate

signal CLK\_DIV: STD\_LOGIC;

signal CLK\_DIV\_AF: STD\_LOGIC;

--semnalele corecte de la butoane (debounced)

signal BUTON\_AUTO\_D, BUTON\_TEMPERATURA\_D, BUTON\_VITEZA\_D, START\_D: STD\_LOGIC;

signal TIMP\_AFIS: STD\_LOGIC\_VECTOR(11 downto 0);

signal TIMP\_NATURAL: NATURAL;

signal UNITATI, ZECI, SUTE, MII: STD\_LOGIC\_VECTOR(3 downto 0);

begin

DIVIZARE\_CLK: CLOCK\_DIVIDER port map (CLK, '0', CLK\_DIV, CLK\_DIV\_AF);

DEBOUNCE\_AUTO: DEBOUNCE4\_PUSH\_BUTTONS port map (CLK\_DIV, '0', BUTON\_AUTO, BUTON\_AUTO\_D);

DEBOUNCE\_TEMPERATURA: DEBOUNCE4\_PUSH\_BUTTONS port map (CLK\_DIV, '0', BUTON\_TEMPERATURA, BUTON\_TEMPERATURA\_D);

DEBOUNCE\_VITEZA: DEBOUNCE4\_PUSH\_BUTTONS port map (CLK\_DIV, '0', BUTON\_VITEZA, BUTON\_VITEZA\_D);

DEBOUNCE\_START: DEBOUNCE4\_PUSH\_BUTTONS port map (CLK\_DIV, '0', START, START\_D);

GET\_INPUT: GET\_DATA port map (SEL\_MODE, BUTON\_AUTO\_D, BUTON\_TEMPERATURA\_D, BUTON\_VITEZA\_D, PRESPALARE\_MANUAL, CLATIRE\_SUPL\_MANUAL, PRESPALARE, CLATIRE\_SUPL, VITEZA, TEMPERATURA);

CALC\_TIMP: TIME\_CALC port map (TEMPERATURA, TIMP\_SPALARE\_PRINCIPALA, TIMP\_CLATIRE, TIMP\_CENTRIFUGARE, TIMP\_PRESPALARE, TIMP\_APA, PRESPALARE, CLATIRE\_SUPL, TIMP\_TOTAL);

STARE\_NXT: NEXT\_STATE\_LOGIC port map (START\_D, PRESPALARE, CLATIRE\_SUPL, USA, CLK\_DIV, TEMPERATURA, PRESPALARE\_OUT, CLATIRE\_SUPL\_OUT, ACTIV, ROTIRE, ALIMENTARE, INCALZIRE, EVACUARE, USA\_BLOCATA);

TIMER: TIMP\_AFISOR port map (CLK\_DIV, START\_D, USA, TIMP\_TOTAL, TIMP\_NATURAL, TIMP\_AFIS);

AFISOR\_7SEGM: AFISOR port map (CLK\_DIV\_AF, TIMP\_NATURAL, UNITATI, ZECI, SUTE, MII, A\_TO\_G, AN);

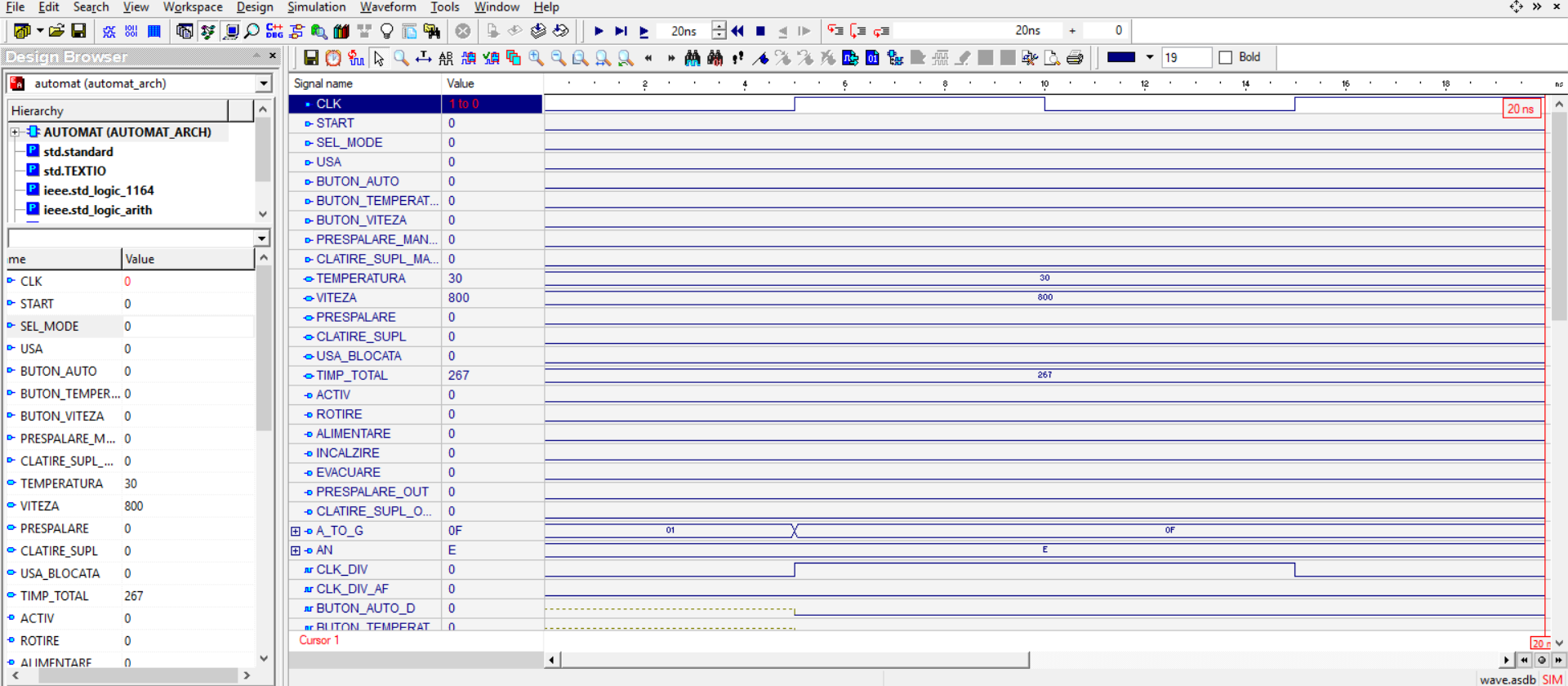
COMP: BIN\_TO\_BCD port map (TIMP\_AFIS, UNITATI, ZECI, SUTE, MII);

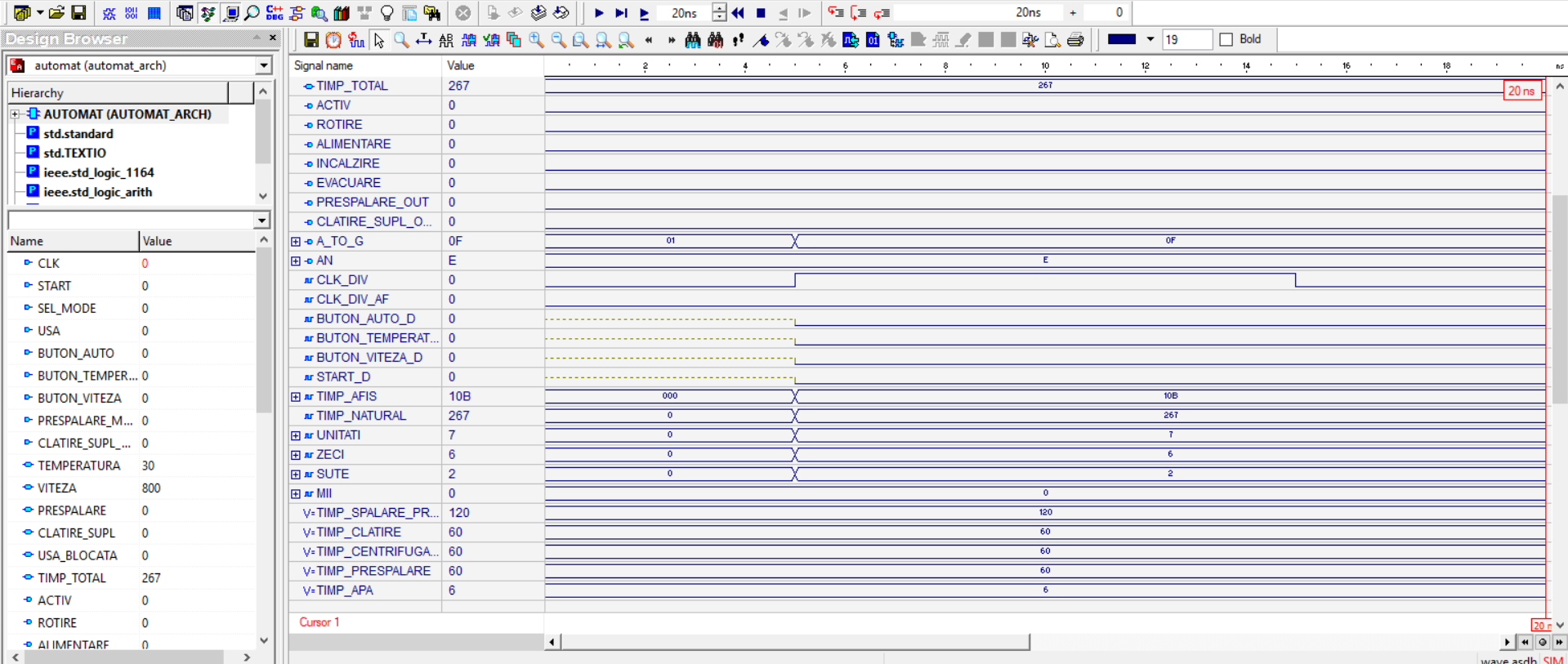
end AUTOMAT\_ARCH;

!Menționăm că din acest cod au fost omise declarațiile componentelor în zona declarativă a arhitecturii pentru sporirea lizibilității.   
  
Vom detalia în cele ce urmează funcționarea automatului, adică a întregului ansamblu, acesta reprezentând componenta principală (de ansamblu) a proiectului. Așadar, semnalele de la butoane vor trece prin circuitul de debounce și va fi divizat impulsul de tact primit de la placa FPGA. Totodată, va fi calculat timpul total și transmis afișorului. Odată cu apăsarea butonului de START, vor fi preluate datele de la blocul selector (temperatura, viteza, prespălarea, clătirea suplimentară) , așa cum a fost descris în secțiunea 3.b, automatul de stare își va începe funcționarea, respectiv trecerea dintr-o stare în alta, marcată vizual prin aprinderea și stingerea ledurilor corespunzătoare fiecărei etape, iar timpul va începe să scadă. În final, mașina de spălat ajunge în starea inactivă inițială, iar timpul afișat este cel pentru parametrii ”default”. Mașina a terminat, deci, programul de spălare și așteaptă instrucțiunile următoare.

* **Simularea în VHDL**

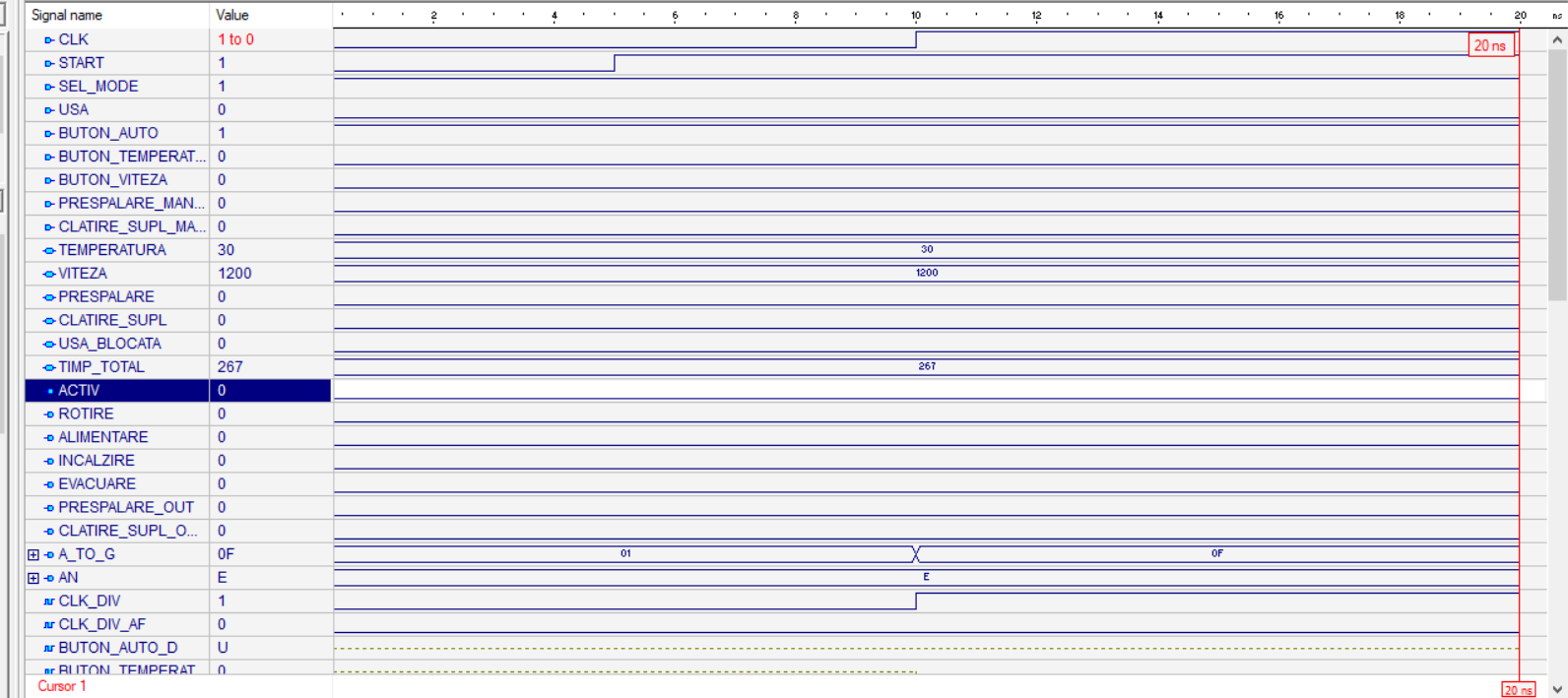
**Faza inițială**





Pentru această simulare s-a folosit un semnal de clock cu frecvența de 50 MHz, semnalul divizat de 25 MHz și un timp de simulare de 20 ns. Se poate observa cum în această etapă, mașina de spălat nu a fost activată (START = 0, ACTIV = 0), ușa este deschisă (USA = 0), iar parametrii sunt ”default” – modul este manual (SEL\_MODE = 0), temperatura este de 30 C, viteza de 800 RPM, nu sunt selectate prespălarea și clătirea, timpul total este 267.

**Selectarea parametrilor în mod automat și încercarea de pornire cu ușa deschisă**



În această etapă se poate observa cum selecția este automată (SEL\_MODE = 1), iar parametrii aferenți primului program (temperatura = 30, viteza = 1200). Butonul de start a fost apăsat (START = 1), dar mașina nu a fost activată, deoarece ușa nu este închisă. (USA = 0).

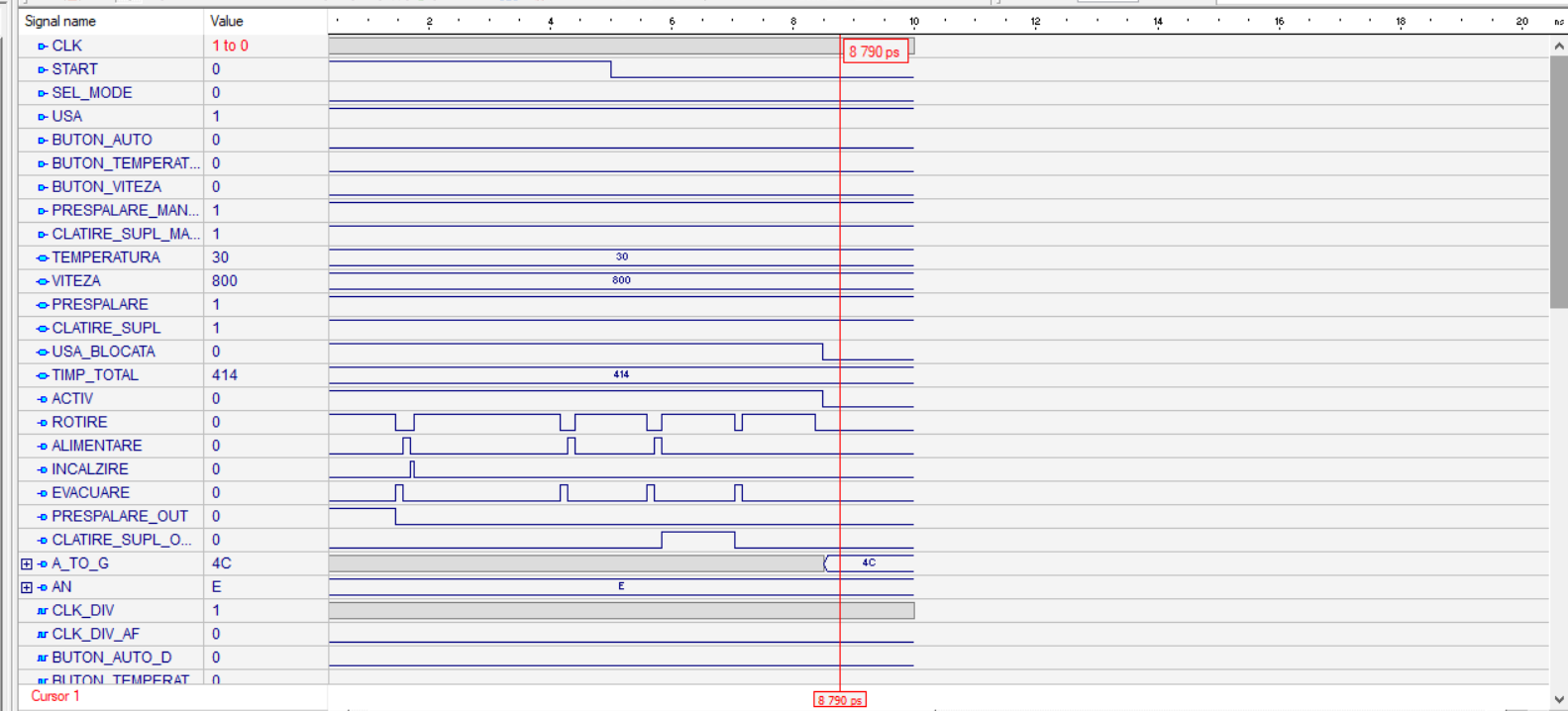
**Simularea unei spălări complete**

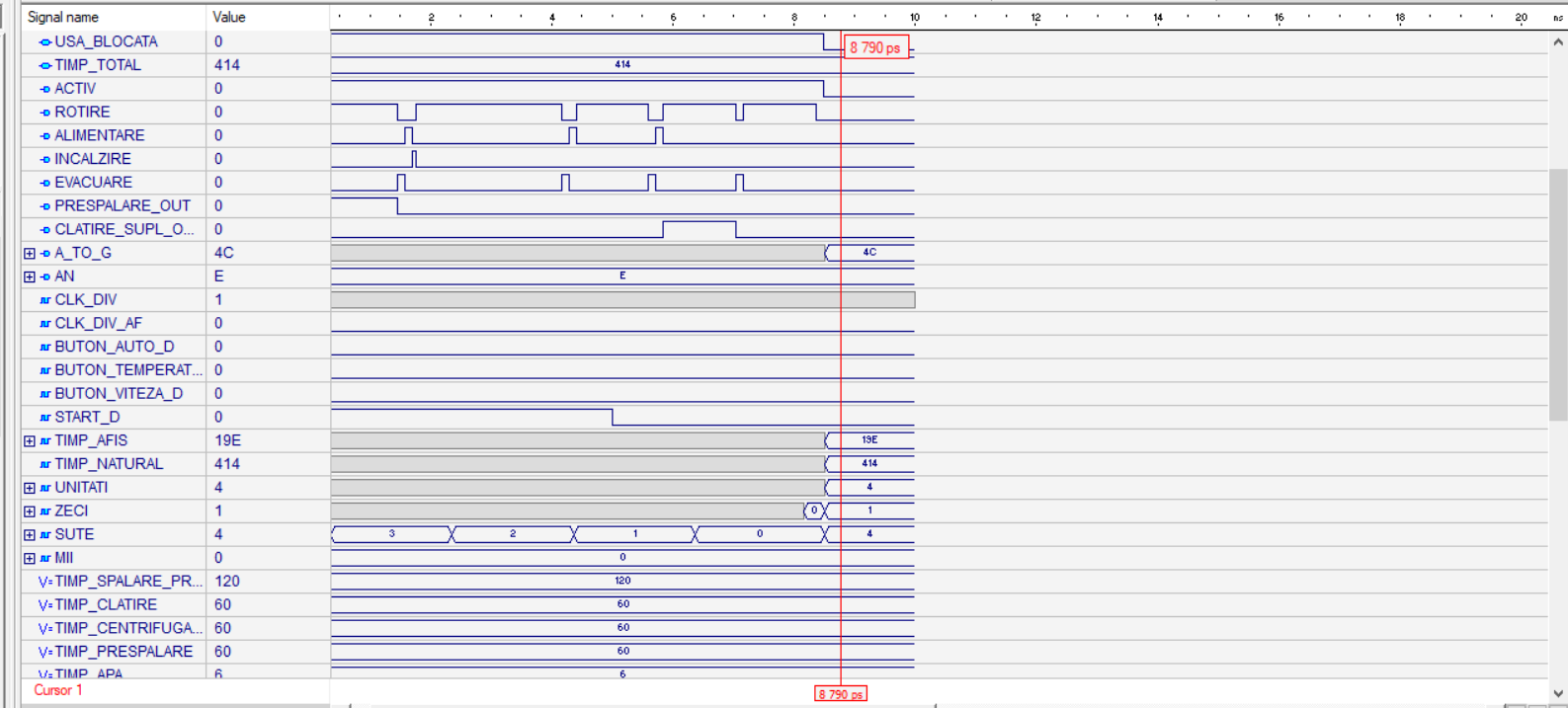
****

****

Pentru această simulare s-a folosit un semnal de tact de 1 GHz, respectiv un semnal divizat de 500 MHz, pentru a putea cuprinde un ciclu complet în 10 ns. Simularea are loc pentru temperatura de 30 C, viteza de 800 rpm, prespălare și clătire. Pe formele de undă se poate observa ciclul complet de activitate, trecerea dintr-o stare în alta și durata fiecărei stări. Se poate vedea și timpul total (414), precum și cel rămas (68) în momentul marcat de linia roșie.

**Sfârșitul unui ciclu complet**





Aici putem observa cum, la finalul ciclului de spălare, mașina de spălat rămâne în starea inactiva (ACTIV = 0), până la următorul start.

**Lista componentelor**

Componentele folosite, unele fiind enumerate și în secțiunea 3.b, sunt:

* AFISOR
  + aprinde pe rând anozii pentru a evita afișarea aceleiași cifre, astfel afișând cifre diferite, corespunzătoare pozițiilor unităților, zecilor, sutelor și miilor
* BCD\_7\_DECODER
  + decodifică informația primită ca un număr natural într-un STD\_LOGIC\_VECTOR de dimensiune 7, pentru a activa catozii corespunzători afișării acelei cifre
* BIN\_TO\_BCD
  + face posibilă extragerea unităților, zecilor și miilor dintr-un STD\_LOGIC\_VECTOR folosind principiul Double-Dabble
* COUNTER\_GET
  + numărător care numără până la o anumită limită transmisă ca parametru generic, pe un anume număr de biți transmis tot ca parametru generic
* CLOCK\_DIVIDER
  + micșorează frecvența clock-ului intern pentru ca afișarea să fie percepută normal de ochiul uman și pentru a adapta timpul programelor, etapelor spălării etc, la un timp dorit
* DEBOUNCE\_4PUSH\_BUTTONS
  + stabilizează semnalele transmise de butoane
* GET\_DATA
  + preia toate informațiile necesare pe care utilizatorul le introduce: modul (manual/automat), programul, temperatura, rotatiile, opțiunea pentru prespălare sau clătire suplimentară
* MUX\_PROGRAM
  + pe intrările acestui multiplexor se află un cod pe 2 biți care reprezintă care temperatură/viteză este prezentă în programul respectiv (de exemplu, dacă am ales programul pentru spălare rapidă, codul pentru temperatură va fi ”00” – 30 C; pentru viteză, ”10” – 1200 rpm); în funcție de numărul apăsărilor butonului auto, se alege codul, acest număr fiind selecție
* MUX\_2TO1
  + acest multiplexor cu calea de date pe 2 biți alege, în funcție de modul selectat, dacă să transmită de câte ori am apăsat pe butonul de temperatură/viteză, sau codul temperaturii/vitezei din program
* MUX\_4TO1\_INTEGER
  + alege din 4 valori de tip INTEGER în funcție de selecție. În cazul automatului nostru, se aleg temperatura și viteza, iar selecția pe 2 biți este, în cazul în care ne aflăm în modul manual, de câte ori am apăsat pe butoane (butonul de temperatură, respectiv viteză), sau, în cazul în care ne aflăm in modul automat, codul care corespunde temperaturii/vitezei în funcție de programul ales
* MUX\_PROGRAM\_PRESP\_CLT
  + la fel ca și pentru temperatură și viteză, prespălarea și clătirea au și ele un cod în funcție de programul presabilit ales (de exemplu, dacă am ales programul pentru rufe murdare, codul pentru prespălare ar fi ”1”, iar pentru clătire suplimentară ar fi ”0”); acest multiplexor are calea de date de 1 bit
* MUX\_2TO1\_BIT
  + acest multiplexor este important pentru a alege de unde să preluam informația referitoare la prespălare și clătire suplimentară; acest lucru se face în funcție de selecție, care este modul (manual = 0/automat = 1); dacă suntem în modul manual, se preia informația de pe switch-uri, iar dacă suntem în modul automat, informația se preia în funcție de apăsarea butonului auto de alegere al programului, care generează codul pentru prespălare sau clătire suplimentară
* NEXT\_STATE\_LOGIC
  + asigură trecerea cu succes dintr-o stare în alta, în funcție de timpul necesar fiecărei stări și ordinea lor
* TIMP\_AFISOR
  + această componentă asigură afișarea timpului programului ales, schimbându-se la fiecare alegere care ar influența durata; scăderea timpului odată cu începerea programului; menținerea timer-ului pe 0 pentru puțin timp când se termină programul (în enunț se specifică faptul că ușa se deblochează un minut după terminarea programului); reinițializarea timer-ului când se deblochează ușa
* TIME\_CALC
  + calculează timpul total al programului pe care urmează sa îl efectueze mașina de spălat în funcție de temperatură și în caz că am optat pentru prespălare sau clătire suplimentară

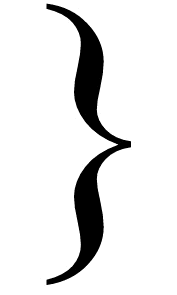
**Semnificația notațiilor**

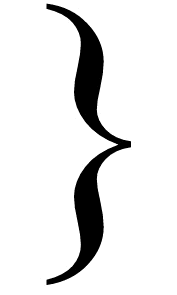
**Intrările automatului sunt:**

* CLK - clock-ul intern al plăcuței
* START - buton pentru începerea programului
* SEL\_MODE - switch pentru selectarea modului: 0 = manual; 1 = automat
* USA - switch pentru a deschide/închide ușa: 0 = ușă deschisă; 1 = ușă închisă
* BUTON\_AUTO - buton pentru alegerea unuia dintre programele prestabilite
* BUTON\_TEMPERATURA - buton pentru alegerea temperaturii când ne aflăm in modul manual
* BUTON\_VITEZA - buton pentru alegerea vitezei când ne aflăm in modul manual
* PRESPALARE\_MANUAL - switch pentru alegerea prespălării când ne aflăm in modul manual
* CLATIRE\_SUPL\_MANUAL - switch pentru alegerea clătirii suplimentare când ne aflăm in modul manual

**Semnale de tip inout:**

* TEMPERATURA – automatul îl primește după ce am ales programul/am apăsat butonul de temperatură; îl transmite către celelalte componente (next\_state\_logic, time\_calc etc)
* VITEZA – automatul îl primește după ce am ales programul/am apăsat butonul de viteză; îl transmite către celelalte componente (time\_calc etc)
* PRESPALARE – automatul îl primește după ce am ales programul/am activat switch-ul pentru prespălare; îl transmite către celelalte componente (next\_state\_logic, time\_calc etc)
* CLATIRE\_SUPL – automatul îl primește după ce am ales programul/am activat switch-ul pentru clătire suplimentară; îl transmite către celelalte componente (next\_state\_logic, time\_calc etc)
* USA\_BLOCATA – automatul îl primește după ce s-a semnalizat blocarea ușii; îl transmite către celelalte componente (next\_state\_logic, time\_calc etc)
* TIMP\_TOTAL – automatul îl primește după ce s-a ales programul; îl transmite către celelalte componente (time\_calc etc)

**Ieșirile automatului sunt:**

* ACTIV
* USA\_BLOCATA
* ALIMENTARE
* INCALZIRE leduri care semnalează starea prezentă
* ROTIRE
* EVACUARE
* PRESPALARE\_OUT
* CLATIRE\_SUPL\_OUT
* A\_TO\_G – catozii activați
* AN – anozii activați
* TEMPERATURA
* VITEZA
* PRESPALARE semnale vizibile doar în simulator
* CLATIRE\_SUPL
* TIMP\_TOTAL

**Semnalele interne intermediare din automat și din componente sunt:**

Clock-uri divizate:

* signal CLK\_DIV: STD\_LOGIC;
* signal CLK\_DIV\_AF: STD\_LOGIC;

Semnalele corecte de la butoane (debounced):

* signal BUTON\_AUTO\_D, BUTON\_TEMPERATURA\_D, BUTON\_VITEZA\_D, START\_D: STD\_LOGIC;

Timpul:

* signal TIMP\_AFIS: STD\_LOGIC\_VECTOR(11 downto 0);
* signal TIMP\_NATURAL: NATURAL;
* signal UNITATI, ZECI, SUTE, MII: STD\_LOGIC\_VECTOR(3 downto 0);

Ieșirile bistabilelor D din circuitul de debounce:

* signal DELAY1, DELAY2, DELAY3: STD\_LOGIC;
* signal Q: STD\_LOGIC\_VECTOR(27 downto 0);

Semnalele care activează pe rând anozii:

* signal COUNT: STD\_LOGIC\_VECTOR(1 downto 0) := "00";
* signal AEN: STD\_LOGIC\_VECTOR(3 downto 0) := "0001"; --cifra 0 mereu afișată

Stările automatului:

* type state\_type is (inactiv, blocare\_usa, alim\_apa, incalzire\_apa, presp, evac, spalare, clt, clatire\_sup, centrifugare, deblocare\_usa);
* signal STARE: state\_type := inactiv;

Contor pentru timer (durata fiecarei etape/stari):

* signal COUNT: INTEGER := 0;

**Justificarea soluției alese**

Principalul factor decisiv în momentul proiectării a fost eficiența. De aceea, codul nostru este structurat în componente simple și reutilizabile de-a lungul proiectului, dar și pentru proiecte viitoare. Această abordare este eficientă din punct de vedere al timpului. Totodată, am considerat esențial ca produsul final să fie ușor de utilizat, cerință pe care considerăm că am îndeplinit-o: pe de o parte, din punct de vedere al introducerii datelor, utilizatorul trebuie să folosească butoane și switch-uri individuale, cu scop bine-definit, iar pe de cealaltă parte, afișajul este clar și sugestiv pentru toate etapele. Din punct de vedere al memoriei și al costului, abordarea noastră, respectiv găsirea unei alternative pentru memoriile ROM, care ar fi fost costisitoare și ar fi făcut proiectarea mai dificilă, corespunde scopului pe care l-am urmărit, și anume eficiență pentru un cost scăzut.

Un aspect decisiv în hotărârea implementării a fost necesitatea de a avea cod sintetizabil pentru plăcile FPGA. În acest sens, ne-am documentat despre posibilele probleme ce ar putea apărea în scrierea codului corect din punct de vedere al simulatorului și astfel, soluția noastră nu conține instrucțiuni de tipul wait. Totodată, în acest sens, am avut grijă în abordarea noastră la folosirea loop-urilor, a for-urilor și la inițializarea semnalelor.

**Utilizare și rezultate**

* **Descrierea resurselor necesare (software și hardware)**

Pentru implementarea și ultilizarea acestui proiect, avem nevoie de anumite resurse software și hardware.

Active-HDL este un instrument software care permite scrierea codului pentru proiectarea unui automat, dar care permite și simularea codului scris de noi, observând rezultatele.

Xilinx ISE Design Suite este un instrument software care sintetizează codul scris, permițând mai departe înscrierea programului pe plăcuță.

ADEPT este un instrument software care programează plăcuța FPGA.

Placa NEXYS 2 este un instrument hardware, un dispozitiv programabil de tip FPGA (field-programmable gate array). Modul său de programare se realizează pe portul USB. Fiind produsă de compania Digilent Inc., placa NEXYS 2 are în principiu componente principale similare cu cele ale plăcii Spartan3 Starter Kit: cele 4 afișoare cu 7 segmente, cele 8 comutatoare cu 2 stări, cele 4 comutătoare de tip push-button și LED-urile.



|  |  |
| --- | --- |
| Segment | Pin |
| CA | L18 |
| CB | D17 |
| CD | D16 |
| CE | G14 |
| CF | J17 |
| CG | H14 |
| DP | C17 |

Conexiunile dintre dispozitivul FPGA şi afişajul cu 7 segmente (active pe 0) și semnalele de control al anodului (active pe 0) sunt:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Anode Control | AN3 | AN2 | AN1 | AN0 |
| Pin | F15 | C18 | H17 | F17 |

Conexiunile comutatoarelor la pinii dispozitivului FPGA sunt:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Comutator | SW7 | SW6 | SW5 | SW4 | SW3 | SW2 | SW1 | SW0 |
| Pin | R17 | N17 | L13 | L14 | K17 | K18 | H18 | G18 |

Conexiunile comutatoarelor de tip „push button” la pinii dispozitivului FPGA sunt:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Push button | BTN3 | BTN2 | BTN1 | BTN0 |
| Pin | B18 | D18 | E18 | H13 |

Conexiunile LED-urilor la pinii dispozitivului FPGA Spartan3 sunt:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LED | LD7 | LD6 | LD5 | LD4 | LD3 | LD2 | LD1 | LD0 |
| Pin | P4 | E4 | P16 | E16 | K14 | K15 | J15 | J14 |

Placa NEXYS-2 include un oscilator de quartz interior de 50MHz și o cutie pentru un al doilea oscilator. Semnalele de tact de la oscilatorii se conectează la pinii de tact global. Astfel se poate realiza divizarea de frecvență precum și multiplicarea acestuia pentru semnale de tact de frecvență variabilă.





* **Descrierea utilizării proiectului**

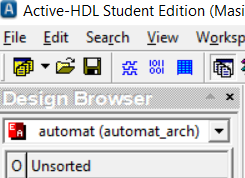
**Active-HDL**

În Active HDL proiectul se folosește astfel:

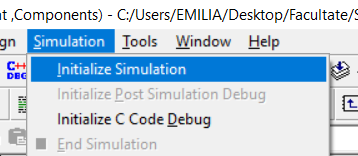
1. Se deschide programul Active HDL

2. Se compilează cu ajutorul comenzii “Compile” 

3. Se alege ca Top-Level: automat



4. Se apasă ”Initialize Simulation”



3. Se simulează cu “New waveform” 

a) Se face click dreapta pe fereastra nou deschisă

b) Se selectează “Add signals…” 

c) Adăugăm din listă toate intrările, ieșirile și semnalele intermediare

d) Se dau valori intrărilor facând click dreapta pe lista aparută și selectând “Simulators” (de acolo se alege “Clock” pentru clock-ul din proiect si “Hotkey” pentru butoane, switch-uri)

e) Dupa ce toate acestea au fost alese se apasa butonul “Run for” din meniu 

f) Următorul pas este selectarea valorilor pentru funcționare și se apasă din nou butonul “Run for” 

g) Se urmăresc rezultatele obținute. (A se vedea la pagina 31)

**Xilinx și implementarea pe placa FPGA**

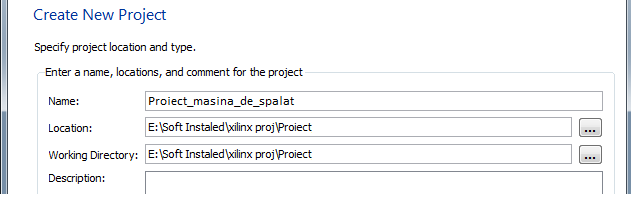
Proiectul dat presupune utilizarea plăcii FPGA Nexys 2 și respectiv utilitarul soft Xilinx ISE Design Suite. Pentru început e necesară sintetizarea codului, asignarea pinilor și generarea fișierului cu extensia ”\*.bit”, după care e posibilă încărcarea proiectului pe placa FPGA. Pentru aceasta realizăm următorii pași:

1. Lansăm aplicația Xilinx ISE Design Suite și creăm un proiect nou:

File > New Project



2. Se asociază un nume



3. Se configurează setările corespunzătoare pentru placa Nexys 2 după cum urmează:

Family: SPARTAN 3E

Device: XC3S500E

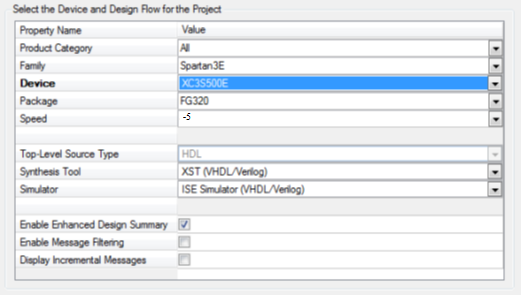
Package: FG320

Synthesis Tool: XST(VHDL/Verilog)

Simulator: ISE Simulator (VHDL/Verilog)

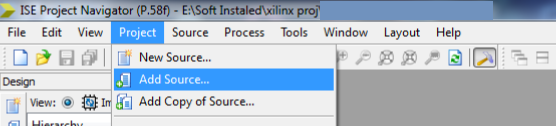
Preferred Language: VHDL

Speed: -5



4. Se adaugă fișierul-sursă: Project > Add Source ...

Se găsește sursa fișierului \*.vhdl și se adaugă în program. Se adaugă la fel și celelalte componentele necesare.



5. Se asignează pinii pentru intrările și ieșirile proiectului



Dublu click pe Create Timing Constraints. Se va genera un fișier \*.ucf de care avem nevoie.

Deschidem fișierul respectiv, introducem următorul cod și salvăm fișierul.

net "SEL\_MODE" loc = G18;

net "PRESPALARE\_MANUAL" loc = H18;

net "CLATIRE\_SUPL\_MANUAL" loc = K18;

net "USA" loc = K17;

net "START" loc = B18;

net "BUTON\_AUTO" loc = D18;

net "BUTON\_TEMPERATURA" loc = E18;

net "BUTON\_VITEZA" loc = H13;

net "ACTIV" loc = J14;

net "USA\_BLOCATA" loc = J15;

net "ALIMENTARE" loc = K15;

net "INCALZIRE" loc = K14;

net "ROTIRE" loc = E16;

net "EVACUARE" loc = P16;

net "PRESPALARE\_OUT" loc = E4 ;

net "CLATIRE\_SUPL\_OUT" loc = P4 ;

net "AN<0>" loc = F17;

net "AN<1>" loc = H17;

net "AN<2>" loc = C18;

net "AN<3>" loc = F15;

net "A\_TO\_G<0>" loc = H14;

net "A\_TO\_G<1>" loc = J17;

net "A\_TO\_G<2>" loc = G14;

net "A\_TO\_G<3>" loc = D16;

net "A\_TO\_G<4>" loc = D17;

net "A\_TO\_G<5>" loc = F18;

net "A\_TO\_G<6>" loc = L18;

net "CLK" loc = B8;

6. Verificăm proiectul și generăm fișierul de execuție



Facem dublu click pe: Synthesize – XST, Implement Design, Generate Programming File. Când în stânga acestor trei procese va fi o bifă alba pe fundal verde (ca în imagine), sau simbolul de warning, putem trece la următorul pas.

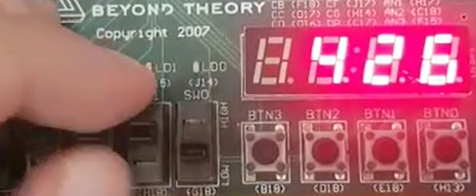
Configure Target Device – se lansează programul Adept (utilitarul folosit pentru programarea plăcii), se conectează placa FPGA Nexys 2 prin cablu USB la calculator, se apasă pe Boundary Scan, se accesează comanda Initialize Chain, din utilitarul Adept, se introduce fișierul cu extensia ”\*.bit” generat de Xilinx, apoi se accesează Program.

* **Rezultatele obținute pe placă**

După ce s-a programat plăcuța, trebuie ca imediat să se observe pe afișor secvența ”267”, anume timpul inițial, standard (program manual cu 30 C, 800 rpm, fără prespălare și clătire suplimentară).

Apoi utilizatorul alege, după cum dorește, orice combinație din modul manual sau orice program din modul automat apăsând butoanele corespunzătoare.

În același timp se observă modificarea timpului pe afișor.



După acest pas, se închide ușa (switch-ul K17) și se apasă butonul START (BTN0), care începe programul mașinii.



Imediat după apăsare, se observă cum timer-ul scade și mașina de spălat trece dintr-o stare în alta.

În exemplu este un program manual cu 60 C, 800 rpm.



 Automatul este activ (LD0), iar ușa este blocată (LD1). Aceste leduri vor rămâne aprinse pe tot parcursul programului.

Se alimentează cu apă (LD2).

Se încălzește apa (LD3).



Spală/Rotește (LD4).

Evacuează (LD5).



Alimentează cu apă.



Clătește/Rotește.

Evacuează.

Centrifughează/Rotește.

Programul s-a terminat, se așteaptă deblocarea ușii.

S-a deblocat ușa, iar automatul a revenit în starea inițială, inactivă.

**Posibilități de dezvoltare ulterioară**

Cum oricărui aparat i se pot aduce îmbunătățiri, la fel se poate face și în cazul mașinii noastre de spălat.

Un aspect care ar putea fi îmbunătățit ar fi introducerea unei opțiuni de anulare și oprire a programului. În cazul în care automatul s-ar afla într-o stare în care există apă în mașină, automatul evacuează apa, iar apoi deblochează ușa.

Alt aspect care ar putea fi îmbunătățit ar fi adăugarea de programe noi, mai diverse și potrivite oricărei cerințe a utilizatorului.

În al treilea rând, existența unui senzor de greutate ar putea aduce un plus. Programele ar putea fi adaptate în funcție de greutate (de exemplu: la programul de cămăși greutatea maximă este de 2 kg).

În al patrulea rând, funcția de pauză și continuare ar putea fi de ajutor. Programul s-ar opri și ar continua de unde a rămas, oricând sunt apăsate aceste butoane.

În plus, programarea spălarilor la anumite ore ar fi o îmbunătățire excelentă. Utilizatorul ar încarca mașina în prealabil cu haine, închide ușa și setează la ce oră să înceapă programul.