MINISTERUL EDUCAȚIEI, CERCETĂRII, TINERETULUI ȘI SPORTULUI
UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Facultatea de Automatică și Calculatoare
Calculatoare și Tehnologia Informației

CODUL DE SECURITATE AL UNUI DULAP

Îndrumător laborator:
Maier Noema

Proiect realizat de:
Dănciulescu Bianca &
Iepure Denisa
Grupa:30213

Cuprins:

1.Specificație proiect
2.Proiectare
3. Implementare
4.Utilizare și rezultate
5.Justificarea soluției alese
6.Posibilități de dezvoltare ulterioară

1. Specificație proiect

Cerință:

Să se implementeze o aplicație care permite utilizatorului adăugarea unui cifru din 3 caractere pentru securizarea unui dulap(asemenea dulapurilor folosite la vestiarele de sport, la mall etc).

În realizarea proiectului vom avea nevoie de două leduri, INTRODU_CARACTERE și LIBER_OCUPAT, care vor semnala posibilitatea introducerii unui cod, respectiv finalizarea codificării și de butonul ADAUGĂ_CIFRU cu ajutorul căruia vom face trecerea de la un cifru la altul. Cele trei numere introduse vor fi cuprinse în intervalul [0, F], acestea putând fi modificate folosind butoanele UP și DOWN. Inițial, ledul LIBER_OCUPAT va fi stins, indicând faptul că dulapul nu a fost codificat anterior, iar la prima apăsare a butonului ADAUGĂ_CIFRU se va aprinde ledul INTRODU_CARACTERE și utilizatorul va putea codifica dulapul. După setarea cifrului, la a 4-a apăsare a butonului ADAUGĂ_CIFRU codul va fi salvat, butonul INTRODU_CARACTERE se va stinge și starea blocat a dulapului va fi marcată de aprinderea ledului LIBER_OCUPAT. Se vor repeta pașii anteriori pentru deblocarea dulapului, iar dacă cifrul corespunde cu cel inițial ledurile și afișajul se vor stinge, în caz contrar ledul LIBER_OCUPAT va rămâne aprins, INTRODU_CARACTERE și afișajul se sting.

2.Proiectare

a) SCHEMA BLOC

Pentru început am dorit să stabilim care vor fi intrările și ieșirile asociate proiectului nostru, așadar “blackbox-ul” de mai jos le pune cel mai bine în evidență.

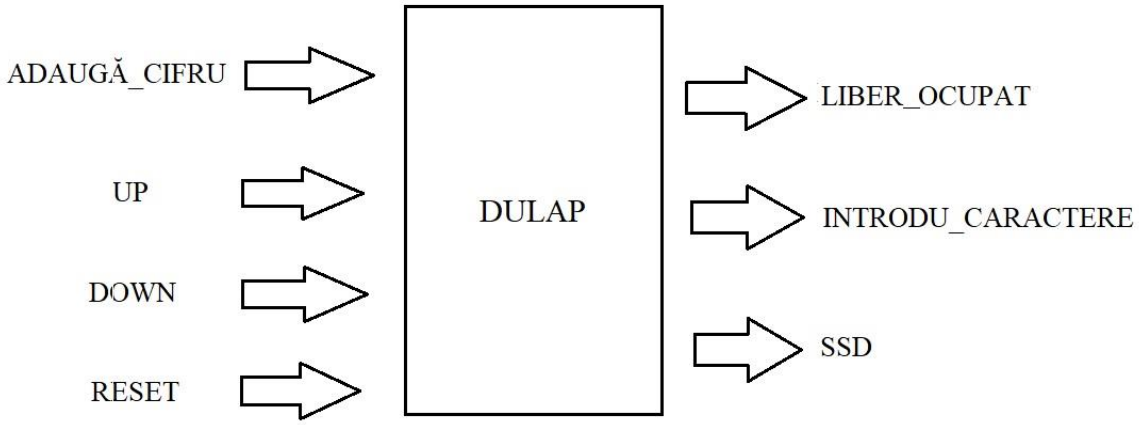


Figura 1- Blackbox

b)UNITATEA DE CONTROL & UNITATEA DE EXECUȚIE

După stabilirea acestor date am ales să mapăm intrările și ieșirile pe cele două componente, UC și UE, după cum se poate observa mai jos:

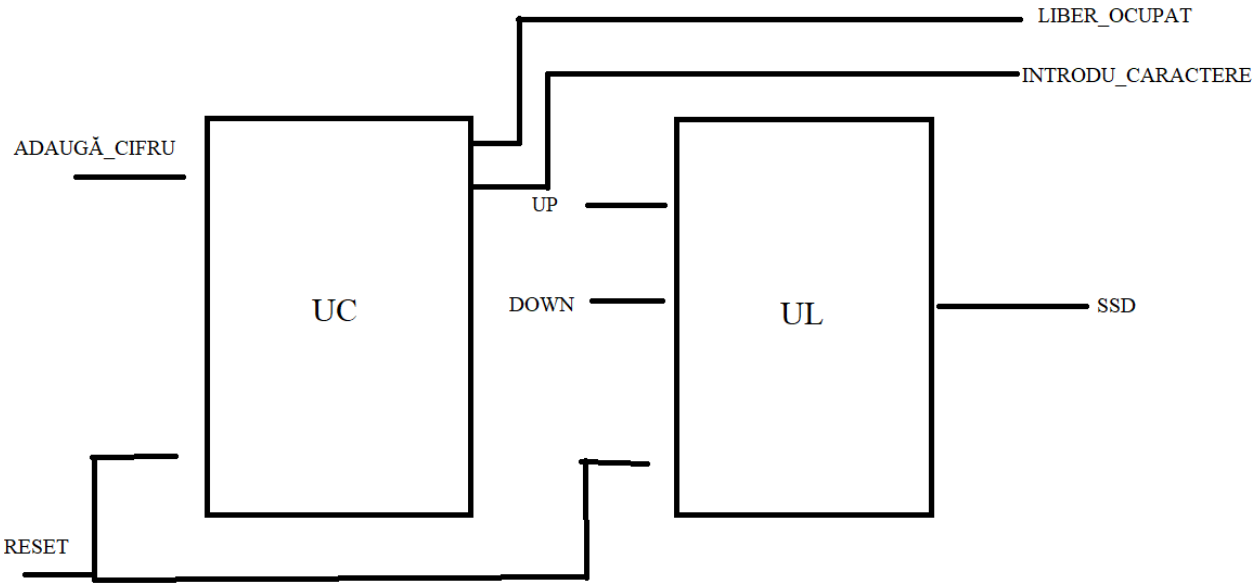


Figura 3-UC&UE

c)Organigrama

Mai departe, am realizat o organigramă de stări care să ne facă mai ușoară înțelegerea mecanismului codificării și, totodată, să ne ajute în alegerea componentelor (vezi ultima pagina).

d)Componente

În continuare vom prezenta resursele necesare codificării cifrului unui dulap.

- 6 numărătoare pe 4 biți;
- un numărător pentru contorizarea numărului de apăsări ale butonului ADAUGĂ_CIFRU;
- 6 regiștrii de memorare ale valorilor introduse;
- 3 comparatoare pe 4 biți formate prin cascadarea comparatoarelor pe 2 biți, respectiv 1 bit;
- o poartă și cu 3 intrări;
- 4 debouncere pentru fiecare buton utilizat;
- 2 afișoare SSD.

3.Implementarea

Pentru setarea fiecărui cod al cifrului ne vom folosi de câte un numărător pe 4 biți, care va avea în componență butoanele UP, DOWN și ADAUGĂ_CIFRU.

```
entity Numaratorr is
port( UP : in std_logic;
DOWN: in std_logic;
RESET: in std_logic;
CLK: in std_logic;
Ad_Cifru: in std_logic;
LED: out std_logic_vector(3 downto 0);
Iesire_Regl: out std_logic_vector ( 3 downto 0);
Nr_Apas: in std_logic_vector(3 downto 0));
end Numaratorr;
```

```
architecture count of Numaratorr is

signal CNT: std_logic_vector(3 downto 0):=  "0000";
signal CNT1: std_logic_vector(3 downto 0):= "0000";
begin
    process(CLK, RESET, UP, DOWN, Ad_Cifru)

    begin
        if(RESET = '1') then CNT <= "0000";

        elsif (CLK'event and CLK='1' ) then
               if (Nr_Apas="0001") then
                 if(UP= '1' and DOWN= '0' AND CNT<"1111") then CNT<= CNT+1;
                                                               CNT1<= CNT1+1;
                 elsif ( UP= '0' and DOWN ='1' and CNT >"0000" ) then CNT<= CNT-1;
                                                              CNT1<= CNT1+1;
                 ELSIF ( UP='1' AND DOWN ='0' AND CNT="1111") THEN CNT<="0000";
                                                                   CNT1<= "0000";
                end if;
              end if;
            end if;

        end process;
        Iesire_Reg1<=CNT1;

        LED<=CNT;
end count ;
```

Acesta prezintă în entitate intrările UP și DOWN, care vor corespunde cu butoanele de pe plăcuță destinate incrementării și decrementării. De asemenea, existența unui buton RESET va reface posibilă numărarea, atunci când va fi activ. Un element nou pe care l-am introdus în structura numărătorului este intrarea Nr_Aps, care face posibilă blocarea numărătorului și setarea caracterului, astfel evitând numărarea în paralel a celor 3 countere. În plus, pentru primele 3 numărătoare avem două contoare ale valorilor. În acest mod la apăsarea butonului de reset , pe ssd se va afișa 000, însă cel de-al doilea contor va face posibila reținerea valorii înainte de reset , lucru ce ne va ajuta la compararea din final . Analog pentru celelalte 5 numărătoare.

Prin intermediul unui numărător de click-uri ale butonului ADAUGĂ_CIFRU facem trecerea de la un caracter la altul . Astfel, pentru implementarea primului caracter ADAUGĂ_CIFRU trebuie să fie apăsat o dată, așadar semnalul care iese din Numărător_Adaugă_Cifru va fi intrare pentru fiecare numărător, reprezentând în arhitectura acestora o condiție.

Pentru caracterul 1, numărul de apăsări ale butonului ADAUGĂ_CIFRU trebuie sa fie 1; când numărul de apăsări crește la 2 numărătorul 1 ramane blocat pe valoarea selectata , iar numărătorul 2 va începe să numere , condiția fiind ca numărul de click-uri sa aibă valoarea 2. Analog pentru caracterul 3.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;


entity Nr_Apasari is
   port( CLK, Ad_Cifru: in std_logic;
   Nr_Apasari: out std_logic_vector(3 downto 0));
end Nr_Apasari;

architecture Behavioral of Nr_Apasari is
signal Q: std_logic_vector (3 downto 0);
begin
  process( clk, Ad_Cifru)
  begin
  if rising_edge(clk) and Ad_Cifru='1'     then Q<=Q + 1;
  end if;
  end process;
  Nr_Apasari<=Q;

end Behavioral;
```

Valorile introduse vor fi salvate în 3 regiștrii de memorare pe 4 biți. Regiștrii vor primi pe rând valoarea fiecărui numărător în parte, ajungând ca în final , după setarea fiecărui caracter prin apăsarea butonului ADAUGĂ_CIFRU , aceștia să aibă salvat cifrul dorit de utilizator, până la condiția opririi, lucru care ne va ajuta să facem la sfârșit compararea.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Registru_mem is

    port ( clk: in std_logic;
    intrare : in std_logic_vector(3 downto 0);
    q: out std_logic_vector (3 downto 0));
end Registru_mem;



architecture Registru_mem of Registru_mem is

begin
    process(  clk, intrare)

    begin
        if rising_edge(clk) then

q<=intrare ;
                end if;

        end process;


end Registru_mem;
```

Pentru a verifica dacă cel de-al doilea cod introdus corespunde cu primul vom folosi 3 comparatoare pe 4 biți, realizat cu descriere combinată, prin port maparea unui comparator pe 2 biți.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity COMPARATOR_4BITI is
    PORT ( X, Y: IN STD_LOGIC_VECTOR( 3 DOWNTO 0);
    F1, F2,F3 : out std_logic);
end COMPARATOR_4BITI;



architecture COMPARATOR_4BITI of COMPARATOR_4BITI is

component     COMPARATOR_2BITI is
    port ( A, B: in std_logic_vector(1 downto 0);
    f1, f2, f3: out std_logic);

end component COMPARATOR_2BITI;
SIGNAL mareS, egalS ,micS ,mareN, eganN, micN :  std_logic;
begin
    U1: COMPARATOR_2BITI PORT MAP ( X(3 DOWNTO 2)   , Y(3 DOWNTO 2), mareS, egalS, micS);
    U2: COMPARATOR_2BITI port map( X(1 downto 0), Y( 1 downto 0), mareN, eganN, micN);
    F1<= mareS or( egalS AND mareN);
    F2<= egalS and eganN;
    F3<= micS or ( egalS and micN);


end COMPARATOR_4BITI;
```

De asemenea, pentru a face verificarea egalității între cele 2 coduri introduse va fi nevoie să utilizăm o poartă și cu 3 intrări, în care vom introduce semnalele de egalitate ale primului, celui de-al doilea și al treilea comparator.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;


entity and_3 is
port( a, b, c: in std_logic;
x: out std_logic);
end and_3;



architecture Behavioral of and_3 is

component and2 is
    port(x, y: in std_logic;
z: out std_logic);
 end component;

 signal s: std_logic;
begin
 s<= a and b;
 x<= s and c;



end Behavioral;
```

Debouncerul îl folosim pentru fiecare buton prin intermediul port mapurilor. El face posibilă transmiterea unui singur semnal la apăsarea unui buton, în cazul nostru semnele care ies sunt: En_Up, En_Down, En_Ad.

```
ENTITY Debouncer IS
   PORT ( clk,buton  : IN STD_LOGIC;
            enable : OUT STD_LOGIC);
END;
```

```
architecture butoane of Debouncer is
signal cnt:std_logic_vector(15 downto 0):=x"0000";
signal en, Q0,Q1,Q2:std_logic:='0';
begin
    numar: process(clk)
    begin
        if(rising_edge(clk))
            then cnt<=cnt+1;
        end if;
    end process;
en<='1'when cnt=x"FFFF" else '0';
et1: process(buton,clk,en)
begin
    if(rising_edge(clk))
        then if(en='1') then Q0<=buton;
        end if; end if; end process;
et2: process(buton,clk,en)
begin
    if(rising_edge(clk))
        then Q1<=Q0;
    end if; end process;
et3: process(buton,clk,en)
begin
    if(rising_edge(clk))
        then Q2<=Q1;
    end if; end process;

enable<=(not (Q2)) and Q1;

    end;
```

Nu în ultimul rând mai avem nevoie de o componentă de afișare, compusă dintr-un decodificator BCD-7 segmente. Acesta realizează afișarea unor cifre pe afișoarele plăcuței FPGA.

```
entity SSD is
    Port ( digit0 : in STD_LOGIC_VECTOR (3 downto 0);
           digit1 : in STD_LOGIC_VECTOR (3 downto 0);
           digit2 : in STD_LOGIC_VECTOR (3 downto 0);
           digit3 : in STD_LOGIC_VECTOR (3 downto 0);
           clk ,reset: in STD_LOGIC;
           Nr_Apas: in std_logic_vector(3 downto 0);
           cat : out STD_LOGIC_VECTOR (6 downto 0);
           an : out STD_LOGIC_VECTOR (3 downto 0));
end SSD;

architecture Behavioral of SSD is
signal aux : STD_LOGIC_VECTOR(15 downto 0):="0000000000000000";
signal digit:STD_LOGIC_VECTOR (3 downto 0);
begin

process(clk)
begin
if reset='1' then aux<=(others=>'0');
elsif  rising_edge(clk) then
aux <= aux + 1;
end if;
end process;
```

```
process(clk)
begin
case aux(15 downto 14) is
when "00" => an <= "1110";
when "01" => an <= "1101";
when "10" => an <= "1011";
when "11" => an <= "0111";
end case;
END PROCESS;
```

```
   with digit SELect
  cat<= "1111001" when "0001",   --1
        "0100100" when "0010",   --2
        "0110000" when "0011",   --3
        "0011001" when "0100",   --4
        "0010010" when "0101",   --5
        "0000010" when "0110",   --6
        "1111000" when "0111",   --7
        "0000000" when "1000",   --8
        "0010000" when "1001",   --9
        "0001000" when "1010",   --A
        "0000011" when "1011",   --b
        "1000110" when "1100",   --C
        "0100001" when "1101",   --d
        "0000110" when "1110",   --E
        "0001110" when "1111",   --F
        "1000000" when others;   --0
end Behavioral;
```

Legarea tuturor componentelor o realizăm în fișierul intitulat “Main”. Ne folosim de port mapare pentru numărătoare, regiștrii, comparatoare și debouncere și folosim 2 procese pentru clock și En_Ad pentru afișarea corectă pe plăcuță.

```
entity Main is
    port ( clk , reset: in std_logic;
     Ad_Cifru , Buton_Up, Buton_Down: in std_logic ;
    Anozi : out std_logic_vector(3 downto 0);
    Catozi: out std_logic_vector ( 6 downto 0);
    Introdu_Caractere: out std_logic:='1';
    Liber_Ocupat : out std_logic :='0');

end Main;
```

```
signal En_Up, En_Down ,En_Ad ,En_Reset: std_logic;
signal cresc: std_logic_vector(3 downto 0):="0000";
signal Val1, Val2, Val3, Val4,VALinter,Iesire_Reg1,Iesire_Reg2,Iesire_Reg3,iesire_r1,iesire_r2,iesire_r3: std_logic_vector ( 3 downto 0):="0000";
signal Val12, Val13, Val14, Val21, Val23,Val24, Val31, Val32, Val34:std_logic_vector( 3 downto 0);
signal  Anozis: std_logic_vector ( 3 downto 0);
signal Catozis: std_logic_vector ( 6 downto 0);
signal  Anozis1: std_logic_vector ( 3 downto 0);
signal Catozis1: std_logic_vector ( 6 downto 0);
signal  Anozis2: std_logic_vector ( 3 downto 0);
signal Catozis2: std_logic_vector ( 6 downto 0);
signal  Anozis3: std_logic_vector ( 3 downto 0);
signal Catozis3: std_logic_vector ( 6 downto 0);
signal Nr_apas,Nr_apasari_3: std_logic_vector( 3 downto 0):="0000";
signal Val5, Val6, Val7, Val8: std_logic_vector ( 3 downto 0):="0000";
signal Reset_Numarator: std_logic;
signal reset1: std_logic;
signal semnal : std_logic;
signal f_mic_c1, f_egal_c1, f_mare_c1,  f_mic_c2, f_egal_c2, f_mare_c2,  f_mic_c3, f_egal_c3, f_mare_c3, semnal_egalitate: std_logic;
signal iesire_reg4, iesire_reg5, iesire_reg6 : std_logic_vector(3 downto 0);
```

```
bt1: Debouncer port map ( clk , Buton_Up , En_Up); -- Debouncer buton Up
bt2: Debouncer port map ( clk , Buton_Down , En_Down );  -- Debouncer pt Down
bt3: Debouncer port map ( clk , Ad_Cifru, En_Ad); -- Debouncer pt Ad_Cifru
bt4: Debouncer port map (clk, reset, En_Reset);

Apas_Ad: Nr_Apasari port map ( clk, En_Ad,Nr_Apas);

Numar1: Numaratorr port map ( En_Up , En_Down , En_Reset, clk  ,En_Ad, Val1,Iesire_Reg1, Nr_apas);
Numar2: Numaratorr2 port map ( En_Up, En_Down ,En_Reset, clk,En_Ad, Val2,Iesire_Reg2, Nr_Apas);
Numar3: Numaratorr3 port map ( En_Up, En_Down ,En_Reset, clk, semnal, En_Ad, Val3, Iesire_Reg3,Nr_Apas);


Numar4: Numaratorr4 port map ( En_Up , En_Down , En_Reset , clk  ,En_Ad, Val5, Nr_Apas);
Numar5: Numaratorr5 port map ( En_Up, En_Down , En_Reset, clk,En_Ad, Val6, Nr_Apas);
Numar6: Numaratorr6 port map ( En_Up, En_Down ,En_Reset, clk,En_Ad, Val7, Nr_Apas);


Registru1: Registru_mem port map(clk, Iesire_Reg1, iesire_r1);
Registru2: Registru_mem2 port map(clk, Iesire_Reg2, iesire_r2);
Registru3: Registru_mem3 port map(clk,Iesire_Reg3 , iesire_r3);

Registru4: Registru_mem4 port map(clk, Val5, iesire_reg4);
Registru5: Registru_mem5 port map(clk, Val6, iesire_reg5);
Registru6: Registru_mem6 port map(clk, Val7, iesire_reg6);

C1: COMPARATOR_4BITI port map (iesire_reg1, Val5, f_mic_c1, f_egal_c1, f_mare_c1);
C2: COMPARATOR_4BITI port map (iesire_reg2, Val6, f_mic_c2, f_egal_c2, f_mare_c2);
C3: COMPARATOR_4BITI port map (iesire_reg3, Val7, f_mic_c3, f_egal_c3, f_mare_c3);
```

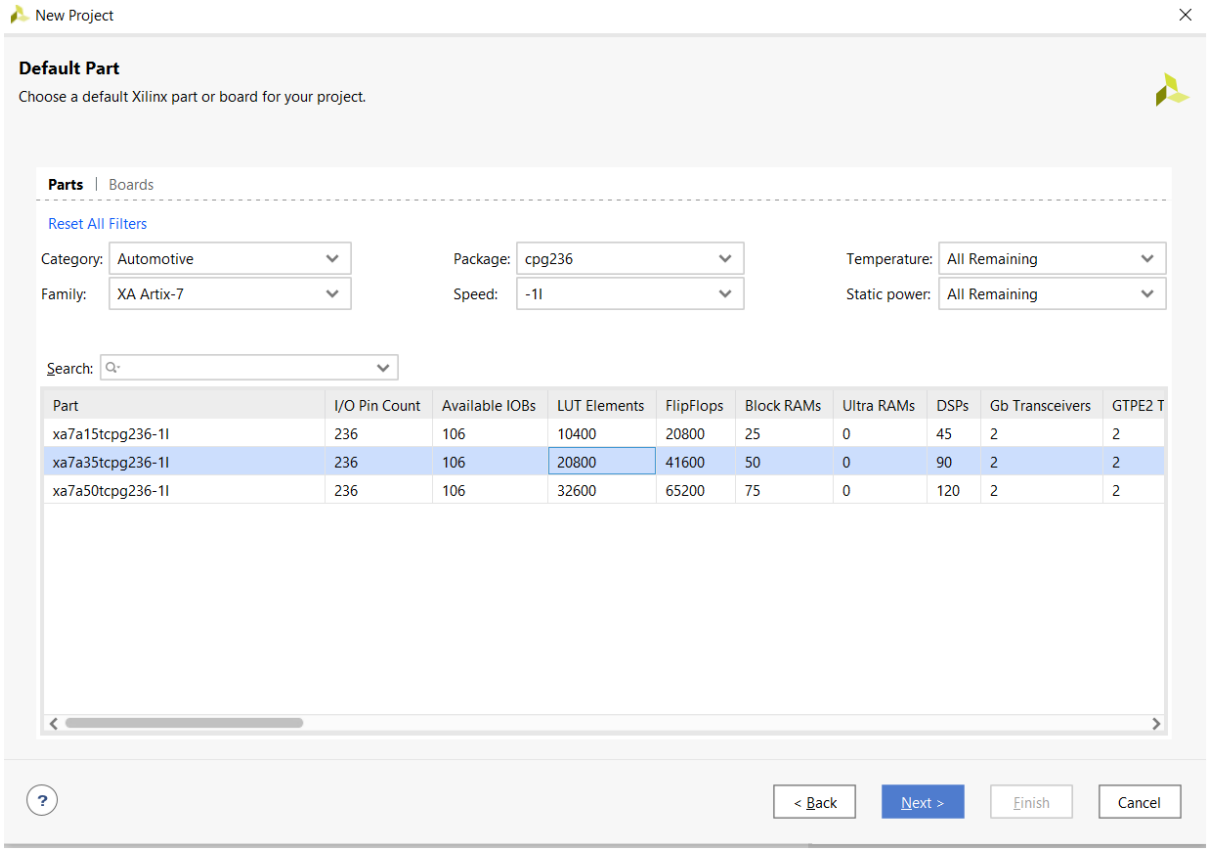
```
    process (clk, En_Ad)
begin
 if (rising_edge(clk) and En_Ad='1') then
 cresc<=cresc+1;
 end if;
 end process;
```

```
process (En_Ad, En_Reset)
begin
              Anozi(2 downto 0)<= Anozis1( 2 downto 0);
              Catozi<=Catozis1(6 downto 0);
   if cresc="0100" then
                Anozi(2 downto 0)<="111";
                Introdu_Caractere<='0';
                Liber_Ocupat<='1';

  elsif (cresc="0110") then
           Liber_Ocupat<= '0';
           Introdu_Caractere<='1';
           Anozi(2 downto 0)<= Anozis2( 2 downto 0);
           Catozi<=Catozis2(6 downto 0);
   elsif(cresc ="0111") then
              Liber_Ocupat<= '0';
              Introdu_Caractere<='1';
              Anozi(2 downto 0)<= Anozis2( 2 downto 0);
              Catozi<=Catozis2(6 downto 0);

    elsif cresc ="1000" then
               Liber_Ocupat<= '0';
              Introdu_Caractere<='1';
              Anozi(2 downto 0)<= Anozis2( 2 downto 0);
              Catozi<=Catozis2(6 downto 0);

    elsif cresc ="1001" then
           if semnal_egalitate = '1' then

                       Liber_Ocupat <= '0';
```

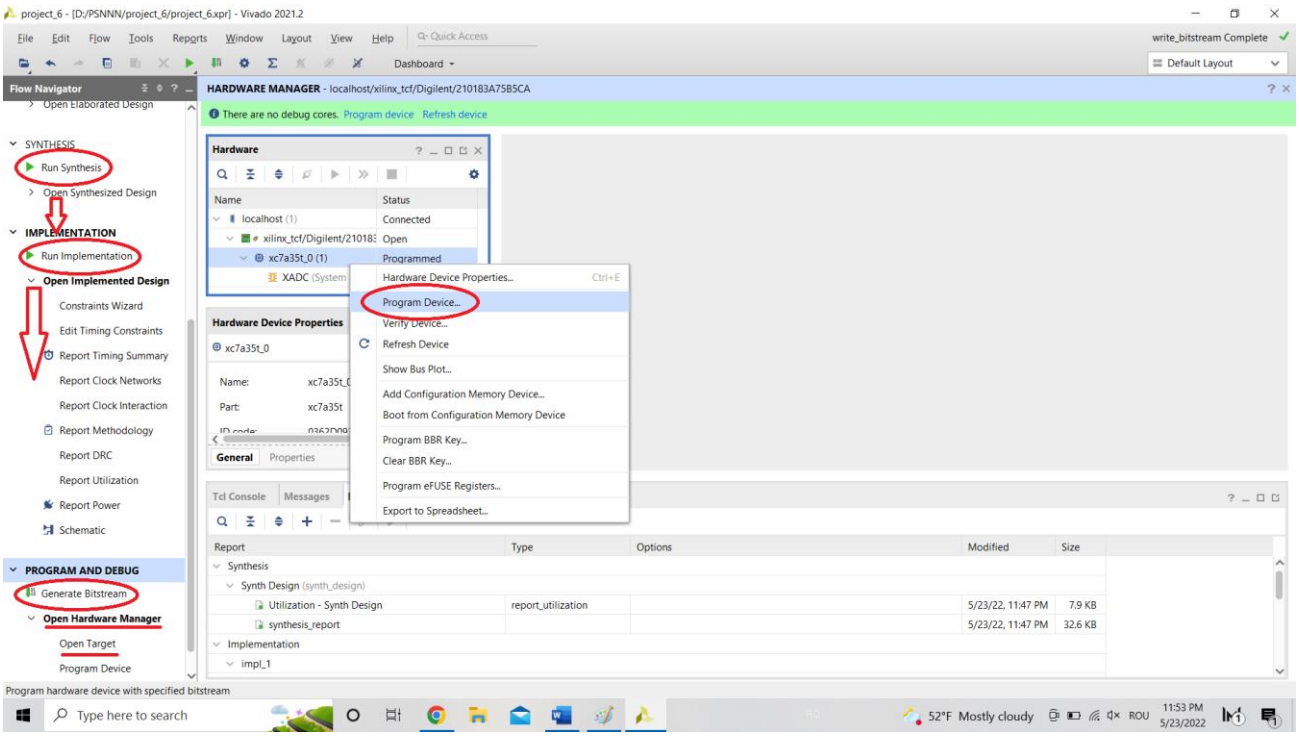
4.Utilizare și rezultate

Proiectul a fost realizat în Design Suite-ul Vivado, care se poate instala de pe site-ul Xilinx, iar pentru simularea rezultatelor se folosește placa Basys 3.

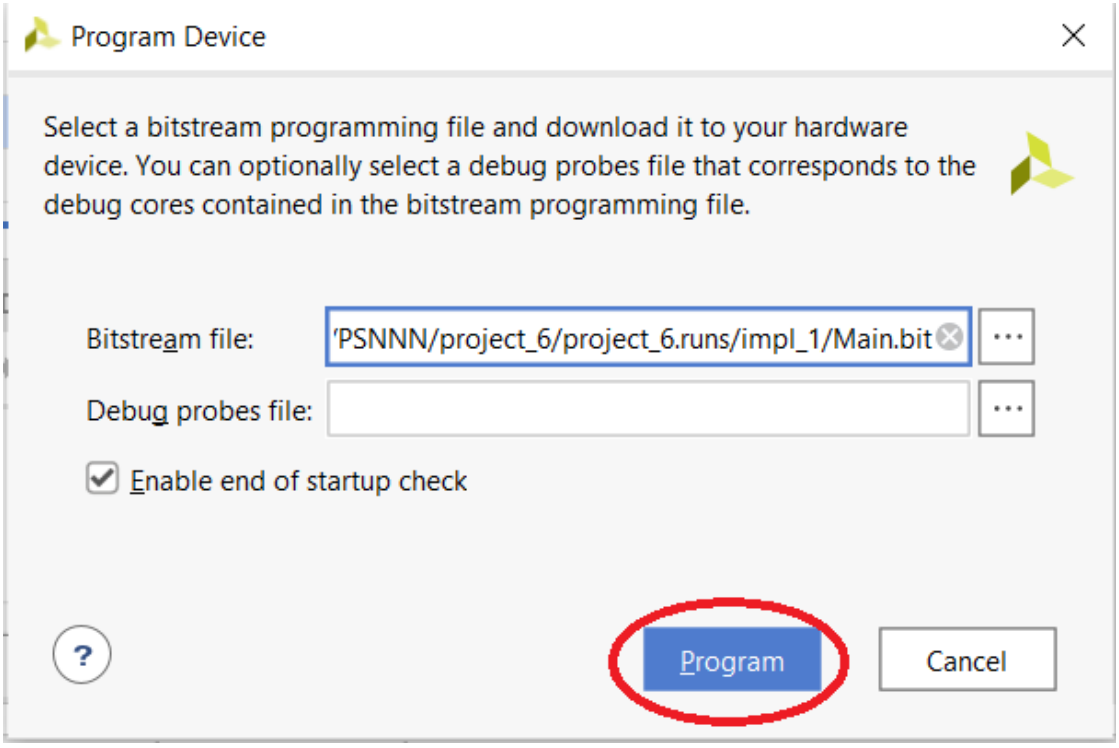
La crearea unui proiect prin selectarea *File-> New Project* se va deschide o fereastră în care trebuie selectate datele cu privire la tipul de placă utilizată. În cazul nostru trebuie să fie introduse următoarele caracteristici:



După deschiderea proiectului propriu-zis, în folder-ul *Design Sources*, aflat în caseta *Sources* vor fi inserate fișierele necesare rulării programului și în folder-ul *Constraints* constrângerile conform plăcuței Basys 3. Se va verifica corectitudinea codului prin selectarea opțiunii *Run Synthesize* din partea stângă a paginii. În caz de succes se selectează caseta *Run Implementation*, apoi *Generate Bitstream.* După efectuarea acestui pas va apărea opțiunea *Open Hardware Manager* , se va da dublu click pe *Open Target*. Ulterior se va aprinde plăcuța folosind switch-ul aflat în partea stângă sus, iar ultimul pas constă în realizarea legăturii dintre aceasta și program, după cum se poate observa în imaginea de mai jos.



Înainte să fie posibilă simularea pe plăcuță, mai rămâne doar selectarea fișierului cu terminația ".bit" din folderul proiectului și apăsarea butonului *Program*, după cum se poate observa în imagine.



Pașii în simularea proiectului:

1. Apăsarea butonului ADAUGĂ_CIFRU
2. Setarea primului cod prin manipularea butoanelor UP și DOWN, urmată apăsarea butonului ADAUGĂ_CIFRU
3. Setarea celui de-al doilea cod prin manipularea butoanelor UP și DOWN, urmată apăsarea butonului ADAUGĂ_CIFRU
4. Setarea celui de-al treilea cod prin manipularea butoanelor UP și DOWN, urmată apăsarea butonului ADAUGĂ_CIFRU
5. În acest moment afișajul și ledul Introdu_Caractere se vor stinge, dar se va aprinde ledul Liber_Ocupat
6. Pentru resetare se va apăsa butonul RESET, urmat de dubla apăsare a butonului ADAUGĂ_CIFRU și reluarea pașilor 2-4
7. În caz de egalitate a celor două coduri introduse, se vor stinge atât ledurile, cât și afișajul, iar în caz contrar doar ledul Liber_Ocupat va fi aprins

Placa Basys 3

Basys 3 este o placă de dezvoltare FPGA entry-level concepută exclusiv pentru Vivado Design Suite, cu arhitectura Xilinx Artix-7-FPGA.

Am folosit această plăcuță pentru simularea rezultatelor proiectului, utilizând butoane, leduri și afișorul, după cum urmează:

- Butoane
  - Up: T18
  - Down: U17
  - Reset: W19
  - Adaugă_Cifru: U18

- Leduri
  - Introdu_Carcatere: U16
  - Liber_Ocupat: L1

- SSD(primele 3 caractere de la dreapta la stânga)

5. Justificarea soluției alese

Titlul proiectului nostru, "Codul de securitate al unui dulap", îi evidențiază cel mai bine utilitatea, anume posibilitatea de codificare a unui dulap, utilizând trei caractere de la 0 la F. În urma setării cifrului, utilizatorul va avea o singura încercare de deschidere a dulapului, introducând codul corect, altfel acesta va rămâne închis.

Până să ajungem la implementarea propriu-zisă, am urmat o serie de pași. Pentru început am stabilit care vor fi intrările și ieșirile utilizate, construind blackbox-ul din *figura 1*. Ulterior, am realizat legăturile dintre Unitatea de Control și cea de Execuție (*figura 2*) pe baza unei organigrame(*figura 3)* de stări care ne-a ajutat să vizualizăm mai ușor condițiile de îndeplinire a unor evenimente. Am stabilit resursele necesare respectării cerinței, iar singurul lucru care a mai rămas a fost să punem componentele cap la cap. În final, am urmat pașii de la punctul 4 pentru a vedea rezultatele pe plăcuță.

6. Posibilități de dezvoltare ulterioară

Codificarea unui dulap apare în distincte împrejurări, precum la mall, la sala de fitness, la bibliotecă ș. a., motiv pentru care importanța acesteia este una ridicată.

În ciuda faptului că proiectul prezintă o funcționalitate normală, fiind posibilă închiderea și deschiderea unui dulap, acesta ar putea suferi o serie de îmbunătățiri. În primul rând, o modificare necesară ar fi permiterea utilizatorului de a introduce un cod de deschidere a dulapului de 3 ori, înainte de ca acesta să rămână definitiv închis. Un led aprins la a doua încercare a introducerii ar putea semnala faptul că a mai rămas o șansă de deschidere a dulapului. În altă ordine de idei, pentru a mări securitatea, codul introdus ar putea fi extins la 4 caractere. Nu în ultimul rând, pentru a amplifica ușurința modului de utilizare, în locul apăsării succesive a butonului ADAUGĂ_CIFRU pentru a introduce toate caracterele, s-ar putea seta un timp scurt de salvare al fiecărui cifru. Prin urmare, codificarea dulapului va fi mai accesibilă și sigură.