

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

AN I, GRUPA 30414

SISTEM DE ILUMINARE VARIABILĂ

Profesor:

Creț Octavian

Student:

Crișan Dan

Cuprins

1. Specificția proiectului.....	3
2. Fenomenul de PWM(pulse-width modulation)	3
3. Proiectare.....	5
4. Schema bloc a aplicației cu componentele principale.....	15
5. Unitatea de comandă și unitatea de execuție.....	16
6. Lista componentelor utilizate.....	17
7. Semnificația notațiilor efectuate și a pinilor interfeței cu exteriorul.....	18
8. Justificarea soluției alese.....	19
9. Instrucțiuni de utilizare și întreținere.....	19
10. Posibilități de dezvoltare ulterioară.....	23

SPECIFICAȚIA PROIECTULUI

Să se proiecteze un **sistem de iluminare variabilă** cu ledurile de pe plăcile cu FPGA. Se cer două moduri de funcționare:

- mod automat – ledurile pulsează la intervalele de timp de 2 secunde și de 5 secunde;
- mod manual – intensitatea luminii ledurilor se programează de pe întrerupătoare.

Pentru varierea intensității luminoase se va folosi PWM (pulse width modulation).

FENOMENUL PULSE-WIDTH MODULATION

Pulse Width Modulation, sau PWM este o tehnică folosită pentru a varia în mod controlat tensiunea dată unui dispozitiv electronic. Această metodă schimbă foarte rapid tensiunea oferită dispozitivului respectiv din ON (1 logic) în OFF (0 logic) și invers. Perioada de timp corespunzătoare valorii ON dintr-un ciclu ON-OFF se numește factor de umplere (duty cycle) și reprezintă, în medie, ce tensiune va primi dispozitivul electronic. Astfel, se pot controla circuitele analogice din domeniul digital. Practic, asta înseamnă că un LED acționat astfel se va putea aprinde / stinge gradual.

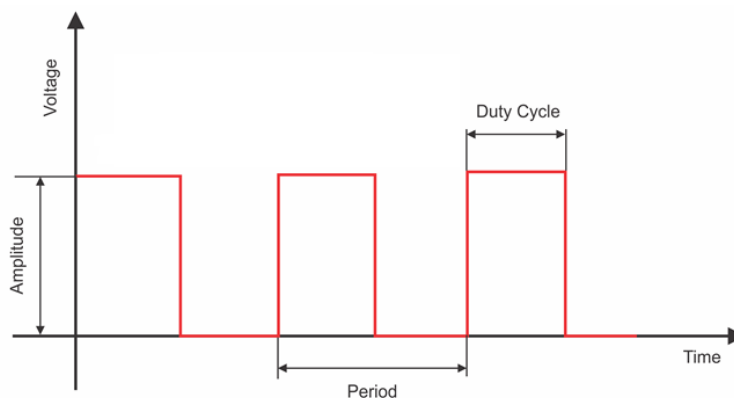


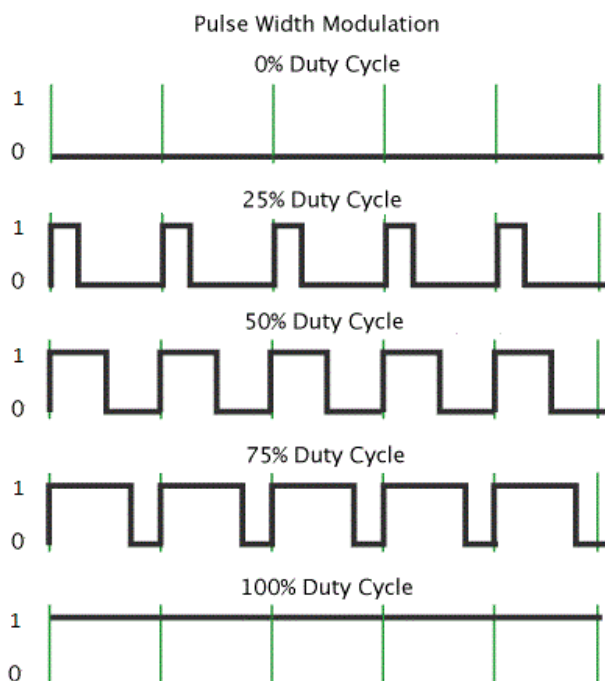
Fig.1. Forma de undă a unui semnal PWM

Factorul de umplere (Duty cycle) se exprimă în procente și reprezintă cât la sută din perioada unui semnal acesta va fi pe nivelul ON (1 logic). În figura 2 se pot observa semnalele

PWM cu factori de umplere diferiți. Astfel, se poate deduce foarte ușor formula pentru a obține valoarea factorului de umplere (D): $D = \frac{\tau}{T}$.

Unde: τ – este durata impulsului, iar T – este perioada semnalului

Fig.2. Exemplu de undă Duty Cycle



PROIECTARE

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity TOP_LEVEL is
    port(CLK, SEL, R: in Std_logic;
         LED: out std_logic_vector(7 downto 0);
         INPUT: in std_logic_vector(6 downto 0));
end entity;

architecture A1 of TOP_LEVEL is
    component Automat is
        port (CLK: in std_logic;
              LED: out std_logic_vector(7 downto 0);
              R: in std_logic);
    end component;

    component manual is
        port (INPUT: in std_logic_vector (6 downto 0);
              CLK: in std_logic;
              LED: out std_logic_vector(7 downto 0);
              R: in std_logic);
    end component;

    component MUX8 is
        port (sel: in std_logic;
              I: out std_logic_vector(7 downto 0);
              A: in std_logic_vector(7 downto 0);
              B: in std_logic_vector(7 downto 0));
    end component;

    component DMUX1_2 is
        port (sel, I: in std_logic;
              A, B: out std_logic);
    end component;

    component MUX2_1 is
        port (sel: in std_logic;
              I: out std_logic;
              A, B: in std_logic);
    end component;

    signal Ledauto, Ledman: std_logic_vector(7 downto 0);
    signal auto, man: std_logic;

begin
    G1: DMUX1_2 port map(SEL, CLK, auto, man);
    G2: automat port map(auto, Ledauto, R);
    G3: manual port map (INPUT,man,Ledman, R);
    G4: MUX8 port map (sel, LED, Ledauto, Ledman);
end architecture;
```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity DMUX1_2 is
    port (sel: in std_logic;
          I: in std_logic;
          A: out std_logic;
          B: out std_logic);
end entity;

architecture DMUX1_2_a of DMUX1_2 is
begin
    P: process (I, sel)
    begin
        casesel is
            when '0' => A<=I;B<='0';
            when '1' => B<=I;A<='0';
            when others => A<='0'; B<='0';
        end case;
    end process;
end architecture;

```

```

libraryieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Automat is
    port (CLK: in std_logic;
          LED: out std_logic_vector(7 downto 0);
          R: in std_logic);
end entity;

architectureAutomat_a of automat is

component counter8 is
    port(CLK:instd_logic;
          RST:instd_logic;
          q:out std_logic_vector(7 downto 0);
          TC: out std_logic);
end component;

component counter9 is
    port(CLK:instd_logic;
          RST:instd_logic;
          q:out std_logic_vector(8 downto 0);
          TC: out std_logic);
end component;

```

```

component Div_1hz is
    port(CLK:instd_logic;
          RST:instd_logic;
          New_CLK:outstd_logic;
          enable: in std_logic);
end component;

component Div0_4 is
    port(CLK:instd_logic;
          RST:instd_logic;
          New_CLK:outstd_logic;
          enable: in std_logic);
end component;

component MUX2_1 is
    port (sel: in std_logic;
          I: out std_logic;
          A: in std_logic;
          B: in std_logic);
end component;

component registru is
    port (LOAD : in std_logic;
          D: in std_logic_vector(6 downto 0);
          Q: out std_logic_vector (6 downto 0);
          CLK: in std_logic);
end component;

component DMUX1_2 is
    port (sel: in std_logic;
          I: in std_logic;
          A: out std_logic;
          B: out std_logic);
end component;

component MUX7 is
    port (sel: in std_logic;
          I: out std_logic_vector(6 downto 0);
          A: in std_logic_vector(6 downto 0);
          B: in std_logic_vector(6 downto 0));
end component;

component comparator is
    port (A: in std_logic_vector (6 downto 0);
          B: in std_logic_vector (6 downto 0);
          AB: out std_logic_vector(7 downto 0);
          BA: out std_logic_vector(7 downto 0));
end component;

component Div_Special is
    port(CLK:instd_logic;
          RST:instd_logic;
          New_CLK:outstd_logic);
end component;

```

```

component MUX8 is
    port (sel: in std_logic;
          I: out std_logic_vector(7 downto 0);
          A: in std_logic_vector(7 downto 0);
          B: in std_logic_vector(7 downto 0));
end component;

signal CE: std_logic := '1';
signal CE1hz: std_logic;
signal CE04hz: std_logic;
signal Clock1hz: std_logic;
signal Clock04hz: std_logic;
signal CLKNUM: std_logic;
signal NUMOUT: std_logic_vector(8 downto 0);
signal terminal :std_logic;
signalterminalTC: std_logic;
signalssel :std_logic;
signal MUXOUT: std_logic_vector(6 downto 0);
signal CLKSPEC: std_logic;
signal REGOUT: std_logic_vector (6 downto 0);
signal NUMOUT1: std_logic_vector(7 downto 0);
signal NUMNEG1: std_logic_vector(6 downto 0);
signal TCNUM: std_logic;
signal MUX7OUT: std_logic_vector(6 downto 0);
signal REGMUX: std_logic_vector(7 downto 0);
signal MUXREG: std_logic_vector(7 downto 0);
signal NUMOUT6:std_logic_vector(6 downto 0);
signal NOTNUMOUT6:std_logic_vector(6 downto 0);
signal NUMOUT16:std_logic_vector(6 downto 0);

begin

    process (CLKNUM)
    begin
        for i in 0 to 6 loop
            NUMOUT6(i)<=NUMOUT(i);
            NOTNUMOUT6(i)<=not NUMOUT6(i);
        end loop;
    end process;

    process (CLKSPEC)
    begin
        for i in 0 to 6 loop
            NUMOUT16(i)<=NUMOUT1(i);
        end loop;
    end process;

    G1: DMUX1_2 port map (NUMOUT(8), CE, CE1hz, CE04hz);
    G2: Div_1hz port map (CLK, R, Clock1hz, CE1hz);
    G3: Div0_4 port map (CLK, R, Clock04hz, CE04hz);
    G4: MUX2_1 port map (NUMOUT(8) , CLKNUM, Clock1hz, Clock04hz);
    G5: counter9 port map (CLKNUM, R, NUMOUT, terminal);
    G6: MUX7 port map(NUMOUT(7),MUXOUT,NUMOUT6,NOTNUMOUT6);
    G7: Div_Special port map (CLK, R, CLKSPEC);
    G8: registru port map (TCNUM, NUMOUT6, REGOUT, CLKSPEC);
    G9: counter8 port map (CLKSPEC, R, NUMOUT1, TCNUM);
    NUMNEG1 <= not NUMOUT16;
    G10:MUX7 port map(NUMOUT(7),MUX7OUT,NUMOUT16, NUMNEG1);
    G11: Comparator port map(REGOUT, MUX7OUT, REGMUX, MUXREG);
    G12: MUX8 port map (NUMOUT(7),LED,REGMUX, MUXREG);

end architecture;

```



```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Div_1hz is
    port(CLK:instd_logic;
          RST:instd_logic;
          New_CLK:outstd_logic;
          Enable: in std_logic);
end Div_1hz;

architecture Div_1hz_A of Div_1hz is
begin
    process(CLK,RST, enable)
        variable count:std_logic_vector(24 downto 0);
        variable k:std_logic;
        begin
            if(Enable = '1') then
                if (RST='1') then
                    count:=(others=>'0');
                    New_CLK<='0';
                    k:='0';
                else
                    if (CLK'EVENT and CLK='1') then
                        if count=390625 then
                            k:=not(k);
                            New_CLK<=k;
                            count:=(others=>'0');
                        else
                            count:=count+1;
                        end if;
                    end if;
                end if;
            end if;
        end process;
    end Div_1hz_A;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Div0_4 is
    port(CLK:instd_logic;
          RST:instd_logic;
          New_CLK:outstd_logic;
          enable: in std_logic);
end Div0_4;

```

```

architecture Div0_4_A of Div0_4 is
begin
    process(CLK,RST, enable)
        variable count:std_logic_vector(26 downto 0);
        variable k:std_logic;
        begin
            if (enable = '1') then
                if (RST='1') then
                    count:=(others=>'0');
                    New_CLK<='0';
                    k:='0';
                else
                    if (CLK'EVENT and CLK='1') then
                        if count=976563 then
                            k:=not(k);
                            New_CLK<=k;
                            count:=(others=>'0');
                        else
                            count:=count+1;
                        end if;
                    end if;
                end if;
            end if;
        end process;
    end Div0_4_A;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX2_1 is
    port (sel: in std_logic;
          I: out std_logic;
          A: in std_logic;
          B: in std_logic);
end entity;

architecture MUX2_1_a of MUX2_1 is
begin
    P: process (sel, A,B)
    begin
        casesel is
            when '0' => I<=A;
            when '1' => I<=B;
            when others => I<='0';
        end case;
    end process;
end architecture;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter9 is
    port(CLK:instd_logic;
          RST:instd_logic;
          q:out std_logic_vector(8 downto 0);
          TC: out std_logic);
end counter9;

architecture counter9_a of counter9 is
begin
    process(CLK,RST)
        variable c:std_logic_vector(8 downto 0);
    begin
        if(RST='1') then
            c:="000000000";
        else if(CLK'EVENT and CLK='1') then
            if (c="111111111") then c:="000000000"; TC <='1';
            else
                c:=c+1; TC<='0';
            end if;
        end if;
        q<=c;
    end process;
end counter9_a;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX7 is
    port (sel: in std_logic;
          I: out std_logic_vector(6 downto 0);
          A: in std_logic_vector(6 downto 0);
          B: in std_logic_vector(6 downto 0));
end entity;

architecture MUX7_a of MUX7 is
begin
    P: process (sel,A,B)
    begin
        casesel is
            when '0' => I<=A;
            when '1' => I<=B;
            when others => I<="0000000";
        end case;
    end process;
end architecture;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity Div_Special is
    port(CLK:instd_logic;
          RST:instd_logic;
          New_CLK:outstd_logic);
end Div_Special;

architecture Div_Special_A of Div_Special is
begin
    process(CLK,RST)
        variable count:std_logic_vector(24 downto 0);
        variable k:std_logic;
        begin
            if (RST='1') then
                count:=(others=>'0');
                New_CLK<='0';
                k:='0';
            else
                if (CLK'EVENT and CLK='1') then
                    if count=200 then
                        k:=not(k);
                        New_CLK<=k;
                        count:=(others=>'0');
                    else
                        count:=count+1;
                    end if;
                end if;
            end if;
        end process;
    end Div_Special_A;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity registru is
    port (LOAD : in std_logic;
          D: in std_logic_vector(6 downto 0);
          Q: out std_logic_vector (6 downto 0);
          CLK: in std_logic);
end entity;

architecture registru_a of registru is
begin
    P: process (CLK, LOAD, D)
        begin
            if (CLK'EVENT and CLK = '1' and LOAD = '1') then
                Q <= D;
            end if ;
        end process;
    end architecture;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter8 is
    port(CLK:instd_logic;
          RST:instd_logic;
          q:out std_logic_vector(7 downto 0);
          TC: out std_logic);
end counter8;

architecture counter8_a of counter8 is
begin
    process(CLK,RST)
        variable c:std_logic_vector(7 downto 0);
    begin
        if(RST='1') then
            c:"00000000";
        else if(CLK'EVENT and CLK='1') then
            if (c="11111111") then c:"00000000"; TC <='1';
            else
                c:=c+1; TC<='0';
            end if;
        end if;
        q<=c;
    end process;
end counter8_a;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity comparator is
    port (A: in std_logic_vector (6 downto 0);
          B: in std_logic_vector (6 downto 0);
          AB: out std_logic_vector(7 downto 0);
          BA: out std_logic_vector(7 downto 0));
end entity;

architecture comparator_a of comparator is
begin
    P: process (A, B)
    begin
        if (A>B) then
            AB <= "11111111"; BA<="00000000";
        elsif (B>A) then
            BA <= "11111111"; AB<="00000000";
        end if;
    end process;
end architecture;

```

```

Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity counter7 is
    port(CLK:instd_logic;
          RST:instd_logic;
          q:out std_logic_vector(6 downto 0);
          TC: out std_logic);
end counter7;

architecture counter7_a of counter7 is
begin
    process(CLK,RST)
        variable c:std_logic_vector(6 downto 0);
    begin
        if(RST='1') then
            c:="0000000";
        else if(CLK'EVENT and CLK='1') then
            if (c="1111111") then c:="0000000"; TC <='1';
            else c:=c+1; TC<='0';
            end if;
        end if;
        q<=c;
    end process;
end counter7_a;

```

```

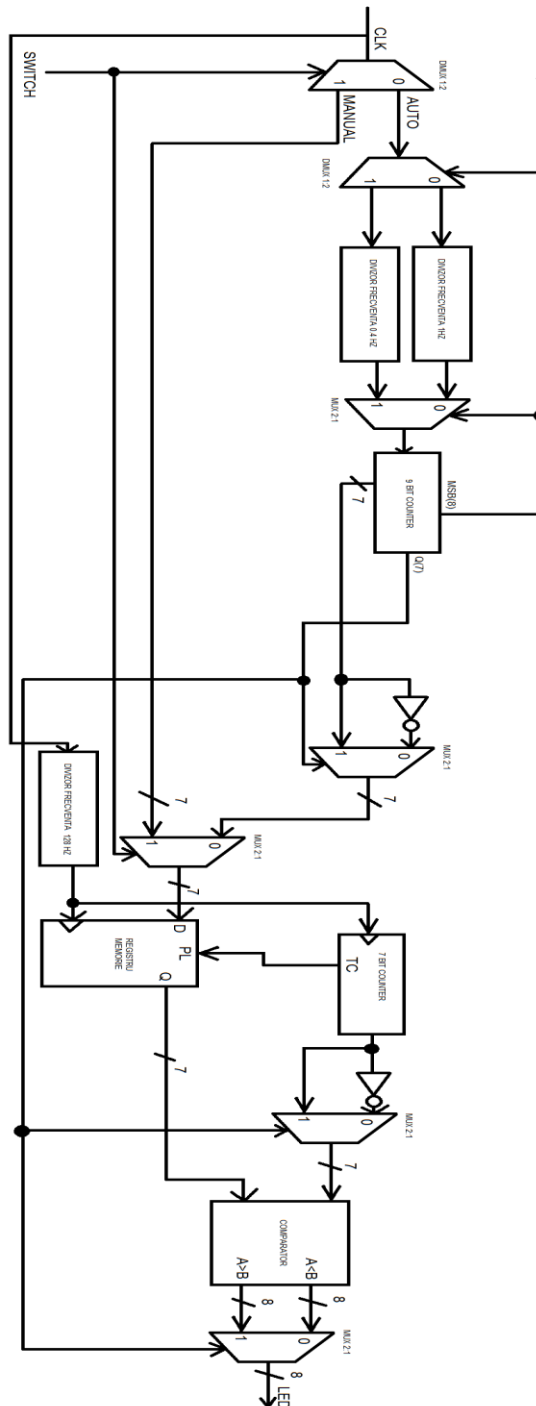
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity MUX8 is
    port (sel: in std_logic;
          I: out std_logic_vector(7 downto 0);
          A: in std_logic_vector(7 downto 0);
          B: in std_logic_vector(7 downto 0));
end entity;

architecture MUX8_a of MUX8 is
begin
    P: process (sel,A,B)
    begin
        casesel is
            when '0' => I<=A;
            when '1' => I<=B;
            when others => I<="00000000";
        end case;
    end process;
end architecture;

```

SCHEMA BLOC A APLICATIEI



UNITATEA DE COMANDĂ ȘI CEA DE EXECUȚIE

• UNITATEA DE COMANDĂ

Unitatea de comandă are, ca primă componentă, un switch care permite selectarea modului (automat/ manual) în care va funcționa sistemul. Astfel, când switch=0, sistemul intră în modul automat, iar când switch=1, sistemul intră în modul manual.

De aici, fiecare mod are propria sa unitate de comandă. Astfel, în modul automat apare semnalul SEL ca select pentru demultiplexor, care selectează care dintre cele două numărătoare de frecvență este utilizat, precum și semnalul NUMOUT(7) (MSB-ul numărătorului pe 8 biți) care selectează direcția (MSB=0 pentru direcție crescătoare și 1 pentru direcție descrescătoare) în care numără cele două numărătoare (este de precizat faptul că nu am folosit un numărător reversibil obișnuit, ci am optat pentru un numărător direct ai cărui biți de ieșire – toți cu excepția MSB-ului – sunt introduși într-un multiplexor 2:1 pe 7 biți, care primește pentru cealaltă intrare aceeași biți, dar negați) prin intermediul unui multiplexor pe 7 biți și, totodată, ce funcție (ieșirea registrului <ieșirea numărătorului în cazul în care acesta numără crescător sau ieșirea registrului >ieșirea numărătorului în cazul în care acesta numără descrescător) va fi afișată la ieșirea din comparator.

În modul manual, unitatea de comandă este reprezentată de input-ul pe 7 biți prin care setăm intensitatea pe care dorim să o aibă LED-ul. Semnalele de selecție lipsesc din acest mod, deoarece nu este necesară nicio schimbare a direcției de numărare în cadrul acestui mod.

• UNITATEA DE EXECUȚIE

Unitatea de execuție cuprinde generatorul de PWM, care preia semnalul corespunzător intensității LED-ului atât în modul automat, cât și în modul manual, cu specificarea că, în cazul modului automat, numărătorul din cadrul generatorului de PWM este urmat de un multiplexor ce permite selectarea direcției în care să numere acesta în funcție de direcția dată de semnalul NUMOUT(7).

Pentru modul automat, unitatea de execuție se mai concretizează în două divizoare de frecvență, unul care generează un clock de frecvență un 1 Hz, iar celălalt – frecvența de 0.4 Hz,

precum și într-un numărător reversibil (mai concret, un numărător crescător, urmat de un multiplexor ce selectează ca output-ul să fie negat sau nu).

LISTA COMPONENTELOR UTILIZATE

- ***Componenta PWM***

Componenta PWM este alcătuită dintr-un numărător pe 7 biți, un registru, un comparator, un multiplexor și un divizor de frecvență. Divizorul de frecvență are rolul de a diviza semnalul de ceas, astfel încât numărătorul componentei PWM să poată parcurge ciclul 0-127 înainte ca registrul să fie încărcat cu următorul număr. Astfel, acesta își găsește întrebuințare numai în modul automat. Registrul memorează numerele pe 7 biți introduse de către utilizator în modul manual sau generate de numărătorul componentei automate în modul automat. Acesta primește un semnal de la Terminal Count-ul numărătorului componentei PWM pentru a se încărca cu următorul număr. Numărătorul generează numere din intervalul 0-127 pentru a putea fi comparate cu numărul din registru. Ieșirile registrului și a numărătorului intră în comparator, unde se verifică care dintre numerele generate este mai mare. Acesta are două ieșiri, una pentru cazul în care numărul memorat de registru este mai mare și un caz contrar. Duty Cycle-ul semnalului generat ne dă intensitatea la care va funcționa LED-ul.

- ***Componenta manuală***

În cazul componentei manuale, registrul din componenta PWM primește un număr selectat de către utilizator, acesta fiind menținut până ce va fi introdus un nou număr. În continuare componenta manuală este identică cu componenta PWM, iar la ieșire va fi afișat numai cazul în care numărul din registru este mai mare decât cel din numărător.

- ***Componenta automată***

Componenta automată conține două divizoare de frecvență, pentru generarea unei frecvențe de 1Hz, respectiv 0.4Hz. Divizoarele funcționează alternativ, cu ajutorul unui demultiplexor, în funcție de un semnal ce este generat după următorii pași: ieșirile de la ambele divizoare intră într-un multiplexor ce are ca și select același semnal ce determină divizorul folosit. Ieșirea acestuia reprezintă clock-ul numărătorului pe 9 biți. Primii doi cei mai semnificativi biți de ieșire ai numărătorului au următoarele roluri: primul este semnal de selecție pentru demultiplexorul și multiplexorul mai sus menționate, iar cel de-al doilea este semnal de selecție pentru un multiplexor 2 la 1, pe 7 biți, care primește ca intrări numărul generat de către numărător și același număr negat cu ajutorul unei porți NOT, pentru un alt multiplexor 2 la 1 pe 7 biți aflat în componenta PWM, precum și pentru un multiplexor 2 la 1 final, conectat la

comparatorul din componenta PWM. Primele două multiplexoare care au acest select generează sensul de numărare (crescător sau descrescător). Spre deosebire de modul manual, comparatorul ia în calcul ambele cazuri. Ieșirea acestuia va genera semnalul ce va da intensitatea cu care va lumina LED-ul.

Alegerea modului de utilizare (manual sau automat) se va face cu un demultiplexor ce are ca select un semnal dat de către utilizator.

SEMNIFICAȚIA NOTAȚIILOR EFECTUATE ȘI A PINILOR INTERFEȚEI CU EXTERIORUL

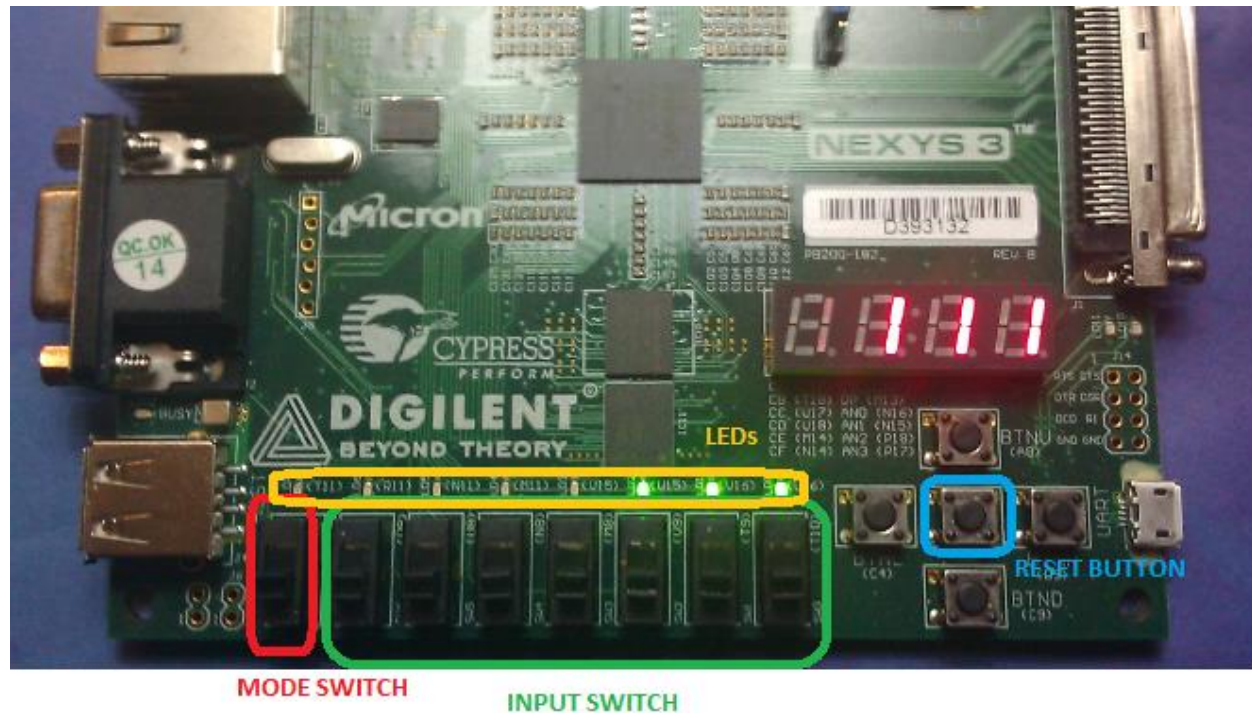


Fig.3. Semnificația pinilor interfeței cu exteriorul

- **SEL** (switch T5) - Selectează modul de funcționare al sistemului: pentru Sel='0' acesta va funcționa în modul automat, iar pentru Sel='1' va funcționa în modul manual.
- **INPUT(6 downto 0)** (switch V8 - T10) – Cât timp Sel='1', adică suntem în modul automat, Inputurile au rolul de a seta intensitatea pe care dorim să o primească LED-ul, între 0 și 127. Valoarea implicită (pentru toate întrerupătoarele deschise) este "0000000" care reprezintă 0% (LED stins). Switch-ul **INPUT(6)** (V8) reprezintă MSB(most significant bit), adică

pentru "1000000" valoarea PWM va fi 64, iar **INPUT(0)** (T10) reprezintă LSB(least significant bit), adică pentru "0000001" valoarea PWM va fi 1.

- **R** (B8) – Butonul de reset. Este activ pe '1', astfel încât nu afectează sistemul cât timp nu este apăsat. Odată apăsat acesta resetează întregul sistem.
- **LED** (T11 - U16) – Reprezintă ieșirea sistemului. În modul manual, acesta va lumina la intensitatea dată de INPUT, iar în modul automat afișează creșterea intensității de la 0 la 127, după care revenirea acesteia în 0, în timp de 2 secunde, respectiv 5 secunde.

JUSTIFICAREA SOLUȚIEI ALESE

În vederea realizării acestui proiect, am ales utilizarea unor componente simple, descrise în stilul comportamental, accesibile din punct de vedere al înțelegerii lor de către utilizator, care mai apoi au fost conectate între ele într-o formă clară și logică, prin intermediul stilului structural, având nume semnificative pentru semnale, ceea ce conferă lizibilitate codului. Datorită flexibilității descrierii în VHDL, am avut posibilitatea să adaptăm numărul de biți utilizați în cadrul unora dintre componente în funcție de nevoile noastre. Acest dispozitiv este ușor de utilizat, deoarece nu necesită transmiterea unui număr mare de comenzi din partea utilizatorului către unitatea de comandă. Astfel, soluția aleasă este una eficientă și accesibilă utilizatorilor.

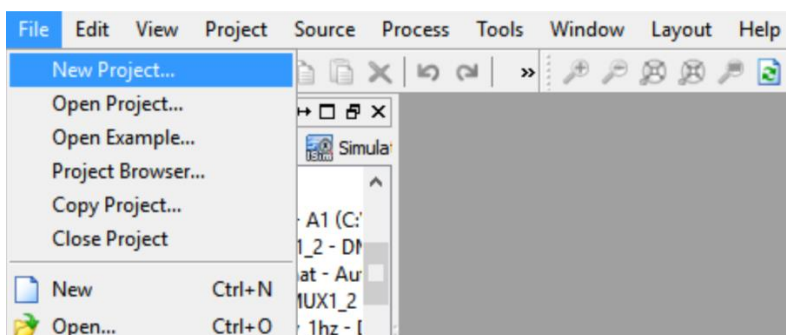
INSTRUCȚIUNI DE UTILIZARE ȘI ÎNTREȚINERE

Proiectul dat presupune utilizarea plăcii FPGA *Nexys 3* și respectiv utilitarul soft *Xilinx ISE Design Suite*.

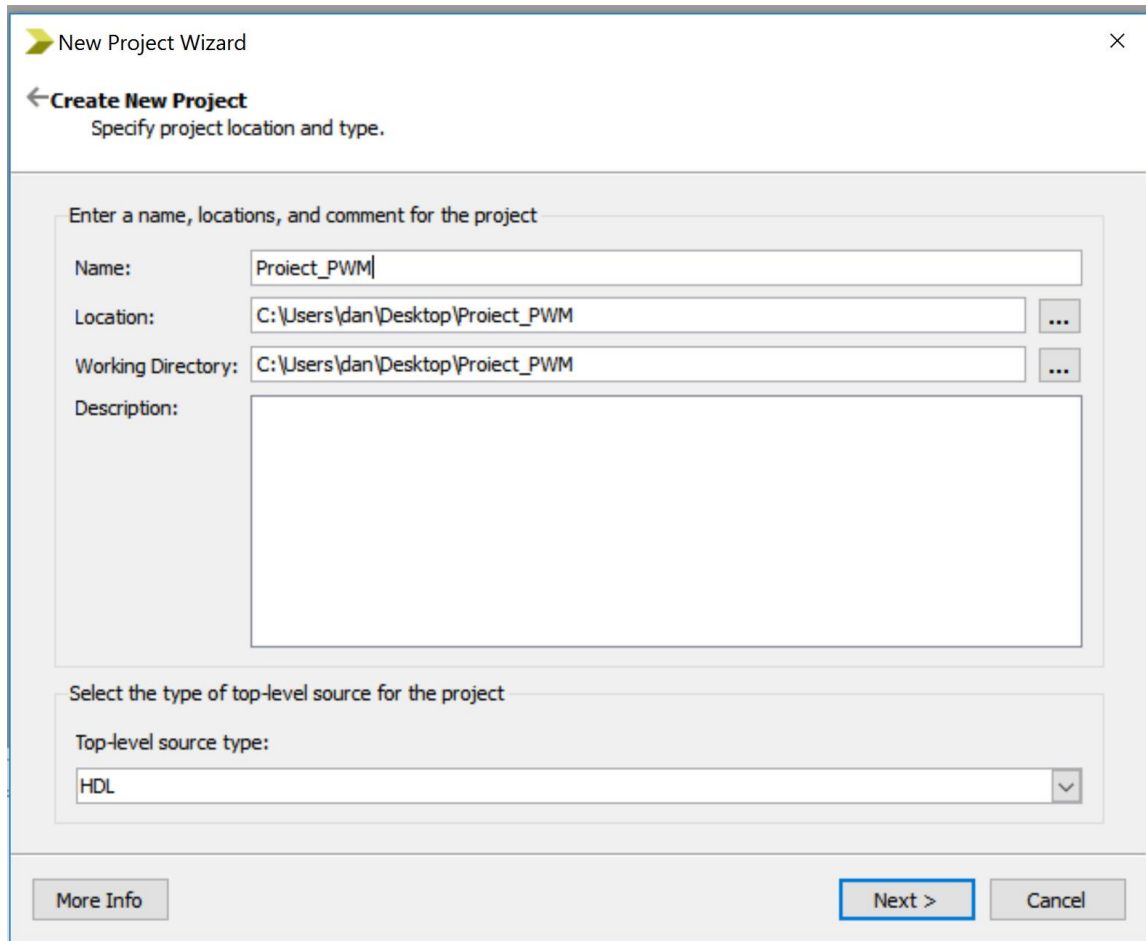
Pentru început e necesară sintetizarea codului, asignarea pinilor și generarea fișierului cu extensia **"*.bit"**, după care e posibilă încărcarea proiectului pe placa FPGA.

Pentru aceasta realizăm următorii pași:

1. Lansăm aplicația *Xilinx ISE Design Suite* și creăm un proiect nou:
File > NewProject



2. Se asociază un nume

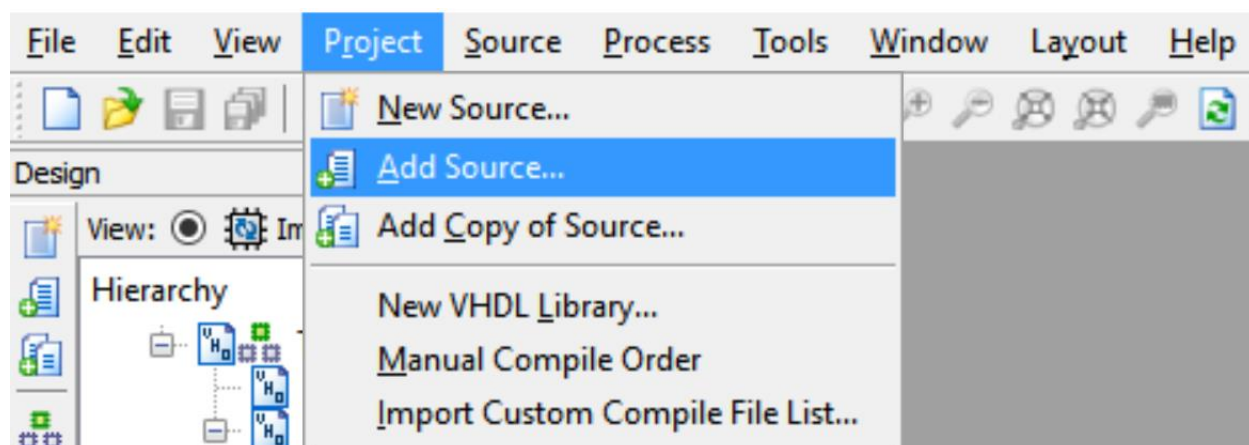


3. Se configurează setările corespunzătoare pentru placa Nexys 3 după cum urmează:

Family:	SPARTAN 6
Device:	XC3S500E
Package:	CSG324
Synthesis Tool:	XST(VHDL/Verilog)
Simulator:	ISim (VHDL/Verilog)
Preferred Language:	Verilog
Speed:	-3

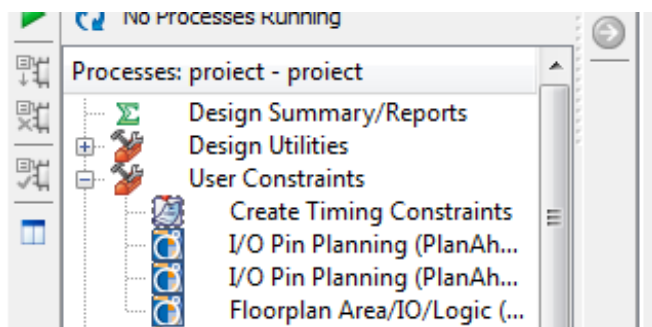
Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan6
Device	XC6SLX16
Package	CSG324
Speed	-3
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISim (VHDL/Verilog)
Preferred Language	Verilog
Property Specification in Project File	Store all values
Manual Compile Order	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

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



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

5. Se asignează pinii pentru intrările și ieșirile proiectului.



```

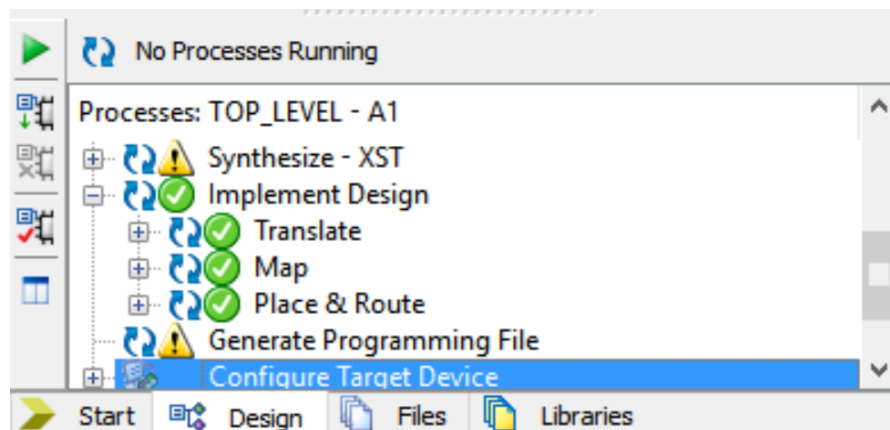
1  NET "CLK" loc=V10;
2  NET "SEL" loc=T5;
3  NET "R" loc=B8;
4
5  NET "LED (7)" loc=T11;
6  NET "LED (6)" loc=R11;
7  NET "LED (5)" loc=N11;
8  NET "LED (4)" loc=M11;
9  NET "LED (3)" loc=V15;
10 NET "LED (2)" loc=U15;
11 NET "LED (1)" loc=V16;
12 NET "LED (0)" loc=U16;
13
14 NET "INPUT (6)" loc=V8;
15 NET "INPUT (5)" loc=U8;
16 NET "INPUT (4)" loc=N8;
17 NET "INPUT (3)" loc=M8;
18 NET "INPUT (2)" loc=V9;
19 NET "INPUT (1)" loc=T9;
20 NET "INPUT (0)" loc=T10;

```

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 (Ctrl+s).

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 albă pe fundal verde (ca în imagine) putem trece la următorul pas.

Configure Target Device – se lansează programul **Adept** (utilitarul folosit pentru programarea plăcii), se conectează placa FPGA Nexys 3 prin cablu USB la calculator, se accesează comanda

Initialize Chain, din utilitarul Adept, se introduce fișierul cu extensia **“.bit”** generat de Xilinx, apoi se accesează **Program**.

POSSIBILITĂȚI DE DEZVOLTARE ULTERIOARĂ

Proiectul dat poate fi dezvoltat prin adăugarea de noi funcționalități, precum:

1. Dezvoltarea componentei manual prin adăugarea a două butoane: Count up și Count down, care vor crește, respective descrește valoarea introdusă în modul manual.
2. Crearea de varietăți pentru componenta automată.
3. Afișarea intensității introduce în modul manual pe ecranul plăcii.
4. Realizarea de reclame cu ajutorul sistemului.
5. Sisteme precum cel descrise își găsesc o utilitate mare în trafic.