FACULTATEA: Automatică și Calculatoare
SPECIALIZAREA: Calculatoare și Tehnologia Informației
DISCIPLINA: Proiectarea sistemelor numerice
PROIECT: A13 PWM pe LED-uri RGB

Îndrumător laborator

Șuta Andrei

Realizator

Obadă Alex - Andrei

Cuprins

1. Specificația proiectului

Proiectul propus are ca scop realizarea unui sistem digital de control al iluminării unui LED RGB, implementat pe o placă FPGA. Sistemul funcționează în trei moduri distincte: modul manual, modul de test și modul automat, fiecare având comportamente specifice asupra intensității celor trei componente de culoare: roșu, verde și albastru.

- **Modul Manual**: utilizatorul controlează individual intensitatea fiecărei componente RGB folosind câte 4 biți pentru fiecare culoare, valorile fiind cuprinse între 0 și 15. Intensitatea se aplică printr-un semnal PWM.
- **Modul Test**: fiecare culoare se activează secvențial, pornind de la intensitate minimă (0) până la maximă, apoi se dezactivează. Secvența este următoarea: roșu → verde → albastru. Tranziția este graduală și servește ca metodă de verificare a funcționării corecte a LED-urilor și a circuitului PWM.
- **Modul Automat**: sistemul generează o succesiune continuă de variații ale intensităților celor trei componente de culoare (roșu, verde, albastru), într-un ciclu repetitiv. Tranzițiile între culori sunt realizate gradual, pentru a crea un efect de iluminare dinamică și fluentă, fără intervenția utilizatorului. Acest mod funcționează complet autonom.

Sistemul pornește în starea **Idle**, fără ca LED-urile RGB să fie active. Trecerea într-unul dintre cele trei moduri funcționale se realizează în funcție de poziția comutatoarelor de selecție Mode(1:0) și prin confirmarea utilizatorului cu ajutorul unui buton de validare (OK):

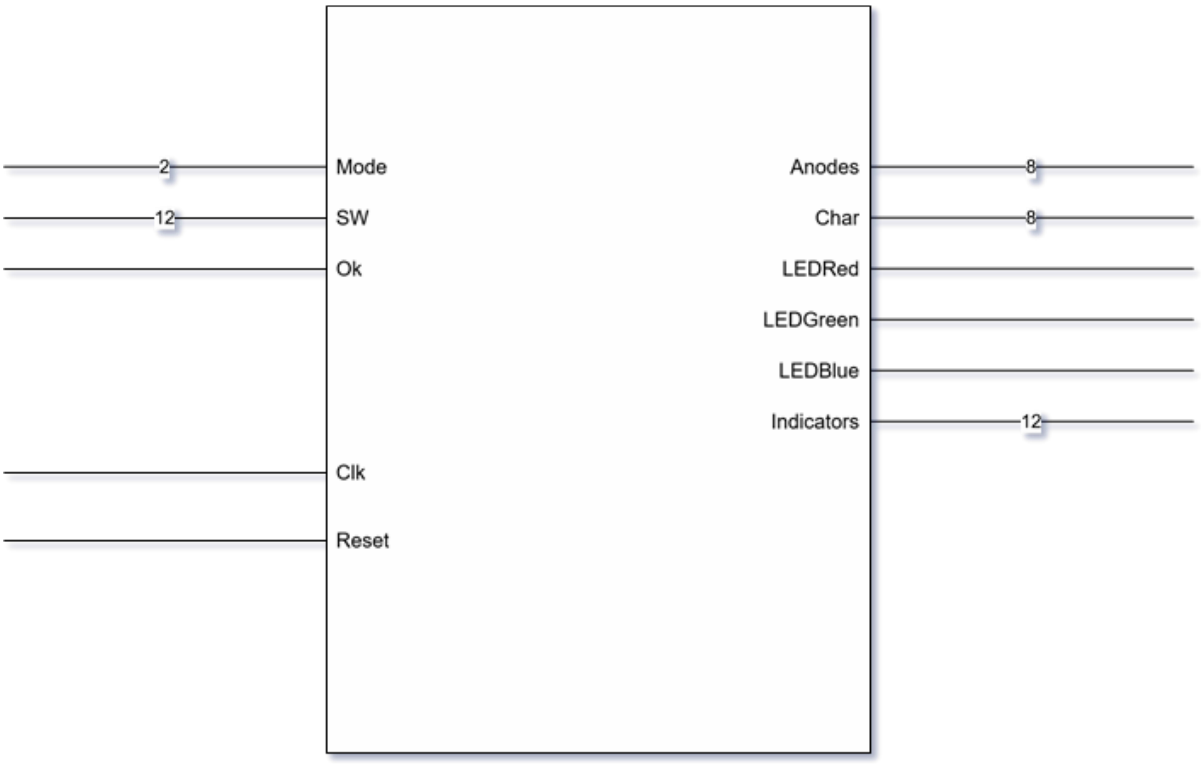
- **00** – Modul Manual
- **01** – Modul Test
- **10** – Modul Automat

După selectarea unei combinații pe comutatoarele Mode(1:0), apăsarea butonului OK determină trecerea în modul respectiv. Indiferent de modul activ, apăsarea butonului **Reset** readuce sistemul în starea inițială **Idle**. Pentru modurile **Manual** și **Automat**, este posibilă revenirea la **Idle** și fără reset, printr-o nouă apăsare a butonului OK.

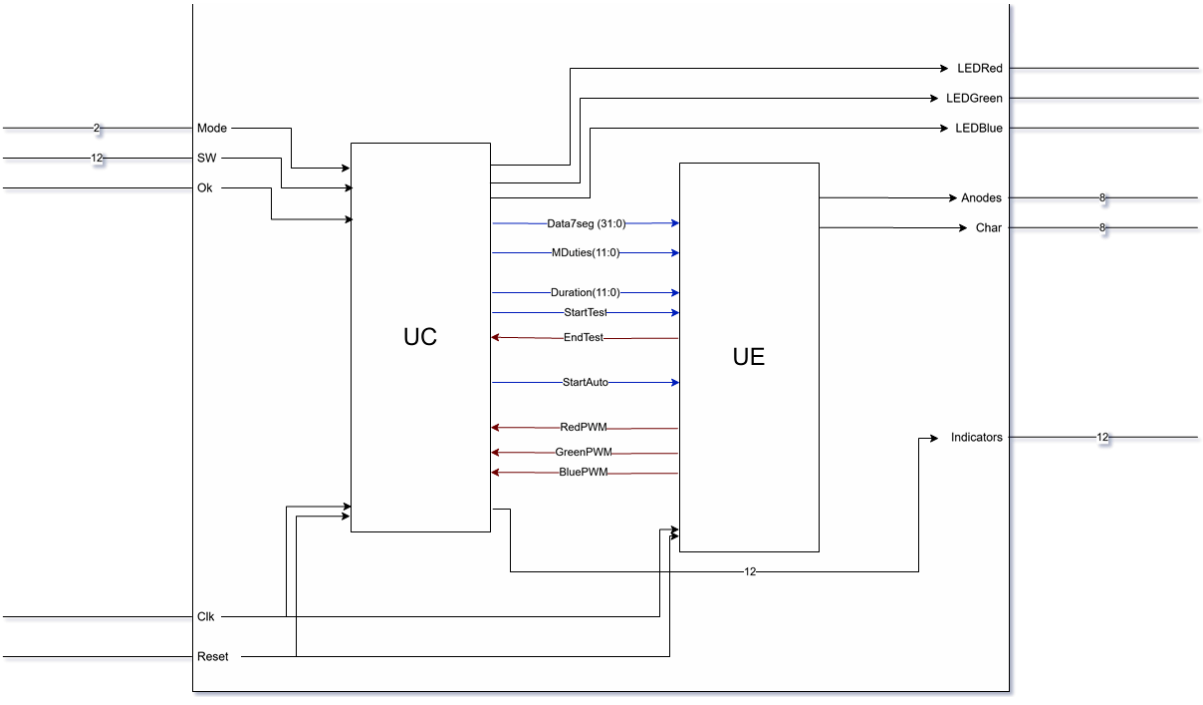
Pentru feedback vizual, proiectul utilizează:

- **LED-uri indicator** (care sunt aprinse dacă switch-ul din fața lor e activat)
- **Afișaj cu 7 segmente** – utilizat pentru afișarea modului curent si a informațiilor relevante ale acestuia.

2. Schemă bloc, schemă detaliată



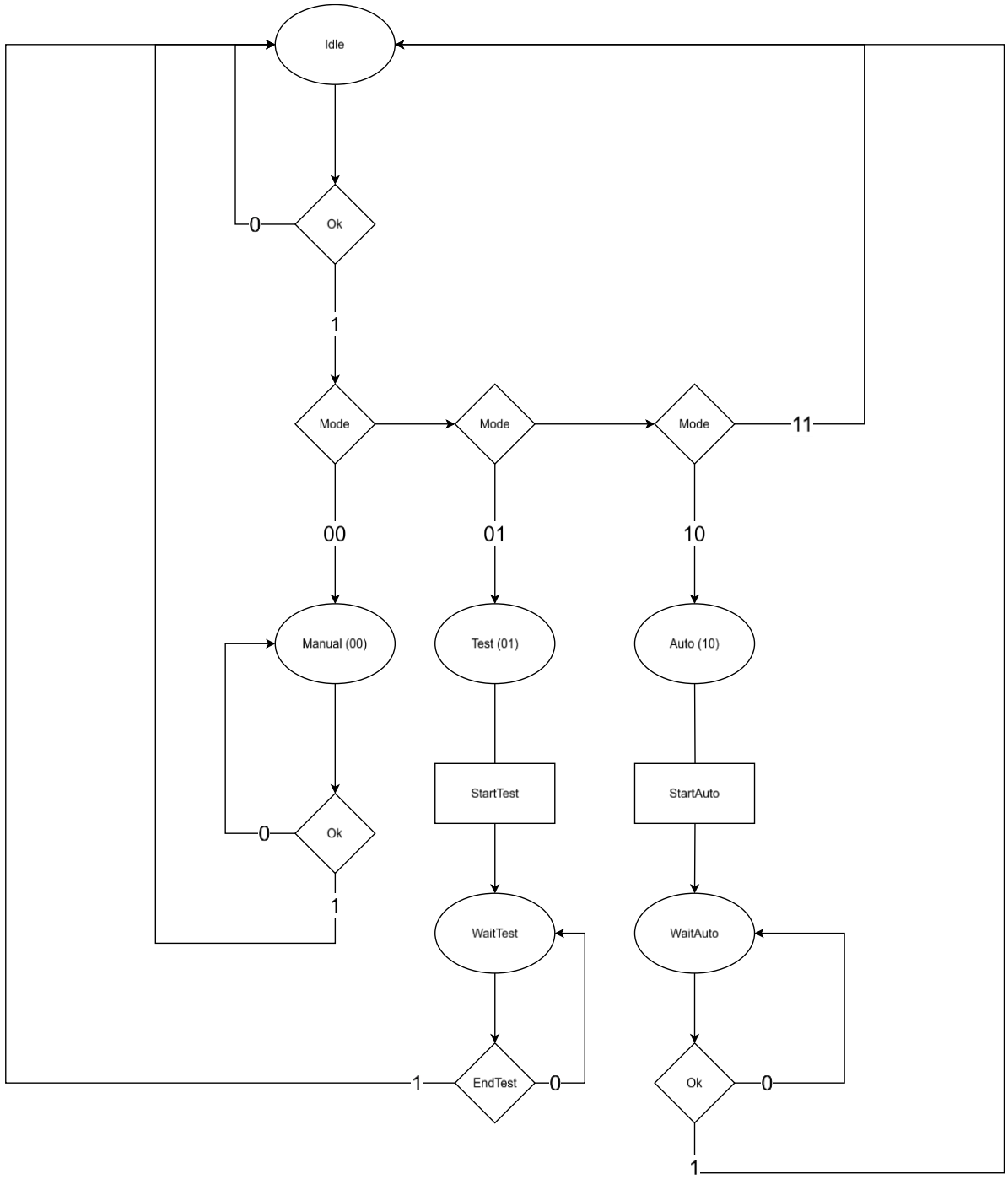
SW reprezintă switch-urile de pe placă, numerotate de la 0 la 11, și sunt reutilizabile în funcție de modul activ al sistemului. În modul **Manual**, aceste switch-uri sunt interpretate ca valori PWM pe 4 biți pentru fiecare componentă de culoare RGB, iar în **Auto** ca un control de viteză a tranzițiilor. SW și Mode sunt considerate **semnale de date**, întrucât transmit efectiv informația utilizată în procesare. În schimb, semnalele OK, Reset și Clk sunt semnale de **control**.



În arhitectura internă a sistemului, semnalul MDuties(11:0) este un semnal de **date** transmis de la Unitatea de Control (UC) către Unitatea de Execuție (UE), care codifică intensitățile componentelor RGB pe câte 4 biți: **roșu** pe cei mai semnificativi biți (11:8), urmat de **verde** (7:4) și **albastru** (3:0). Similar, semnalul Duration(11:0) este tot un semnal de **date** de la UC către UE, utilizat pentru a parametriza durata tranzițiilor de culoare în modul Test. Semnalele StartTest și StartAuto sunt semnale de **control**, folosite de UC pentru a declanșa execuția modurilor respective în UE, în timp ce EndTest este un semnal de **control în sens invers**, de la UE către UC, indicând finalizarea secvenței din modul Test.

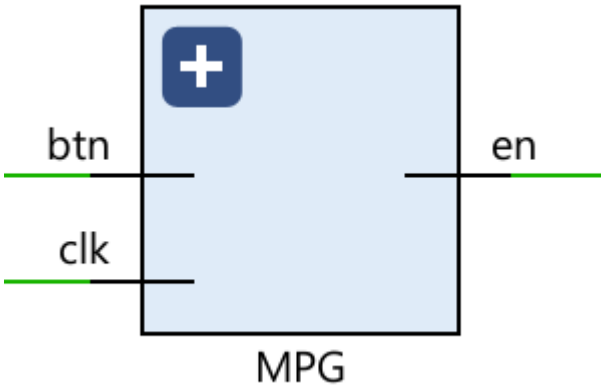
Semnalele RedPWM, GreenPWM și BluePWM sunt considerate **semnale de date** de la UE către UC, reprezentând semnalele PWM generate pentru fiecare componentă RGB, folosite pentru a decide activarea LED-urilor și afișajul stării.

În proiectare, s-a urmărit **abstractizarea fiecărui mod** (Manual, Test, Automat) sub forma unei componente distincte în UE, fiecare componentă având propriile instanțe de generare PWM. Totuși, semnalele RedPWM, GreenPWM și BluePWM sunt partajate în schemă, ele fiind de fapt individuale pentru fiecare mod, iar UC-ul ia decizia care e tripletul folosit la orice moment dat. Aceeași strategie de partajare nu a fost folosită pentru semnalele MDuties și Duration, care au ca sursă comună același vector SW(11:0), dar sunt semnale care se duc la componente diferite în UE. De asemenea, Mduties este o abstractizare a tripletului Red_duty, Green_duty, Blue_duty. Am ales convenția aceasta de reprezentare pentru a nu încărca prea mult schema. Am preferat un design modular, chiar dacă a însemnat un exces de semnale în UC, deoarece a facilitat refolosirea componentelor, a permis detectarea rapidă și delimitarea clară a sursei unui bug logic apărut în timpul testării.

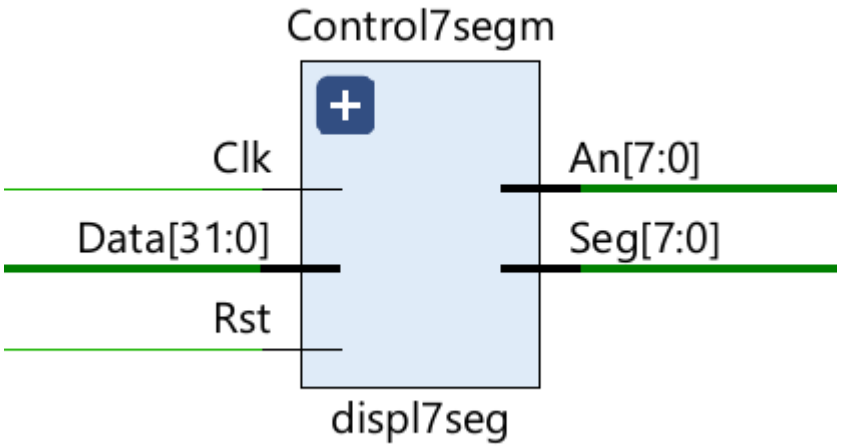


3. Componente

Componenta **MPG** este utilizată pentru a permite folosirea butoanelor de pe placă, reprezentate prin semnalul **btn**. Un Mono Pulse Generator (MPG) generează un singur impuls de durată fixă la detectarea unei tranziții de la 0 la 1 pe un semnal de intrare, fiind util pentru debouncing.



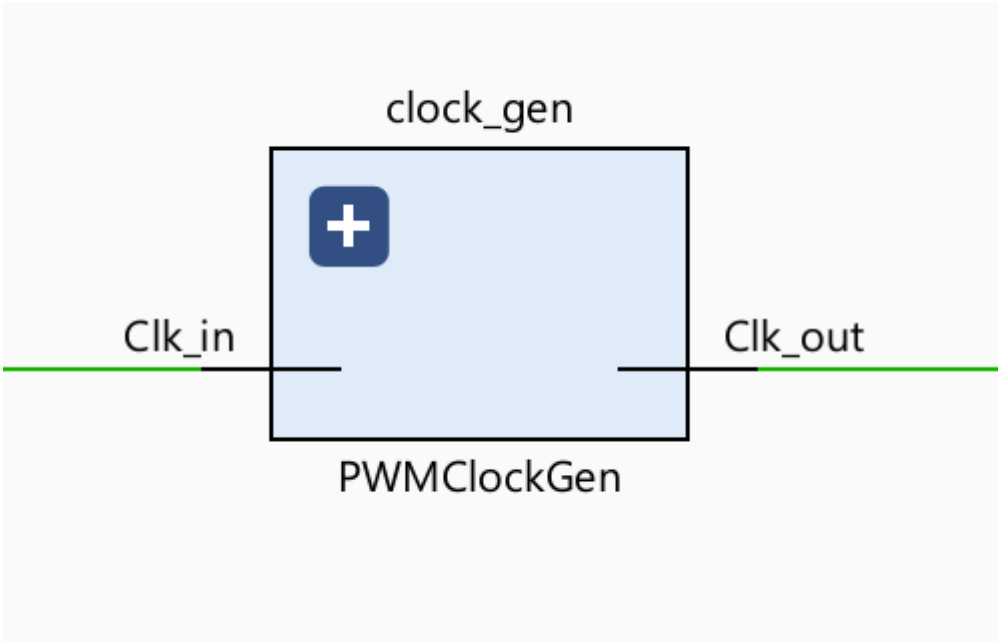
O altă componentă utilă pentru debugging și feedback vizual este **displ7seg**, folosită pentru afișarea modului curent și a informațiilor asociate acestuia. Componenta primește caracterele de afișat sub forma unui vector de 32 de biți (8 cifre hexazecimale, câte 4 biți fiecare) și le afișează pe display-ul cu 7 segmente.



Cele mai importante componente din proiect sunt **PWMClockGen** și **PWMGen**, care reprezintă fundamentul arhitecturii de generare a semnalelor PWM necesare pentru controlul precis al intensității LED-urilor RGB ale plăcii. Aceste două module stau la baza tuturor celorlalte componente funcționale asociate modurilor de operare (Manual, Test și Automat), întrucât fiecare dintre acestea are nevoie de semnale PWM pentru a realiza tranziții graduale sau valori fixe de culoare. Prin abstractizarea generării de semnale PWM într-un mecanism reutilizabil și parametrizabil, PWMClockGen și PWMGen devin esențiale pentru funcționarea întregului sistem, oferind atât flexibilitate, cât și consistență în comportamentul vizual al LED-urilor. Astfel, ele pot fi considerate componente centrale ale proiectului.

Componenta **PWMClockGen** are rolul de a **reduce frecvența semnalului de ceas** primit de la sistem (100 MHz) pentru a genera o frecvență de 5 kHz, potrivită pentru uzul intern al modulului PWM.

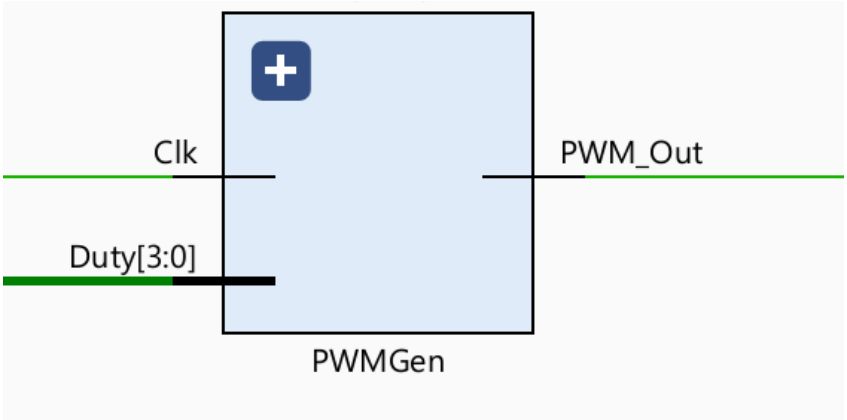
Aceasta funcționează prin incrementarea unui counter la fiecare front crescător al semnalului **Clk_in**. Atunci când contorul atinge valoarea maximă se inversează starea semnalului **Clk_out**.



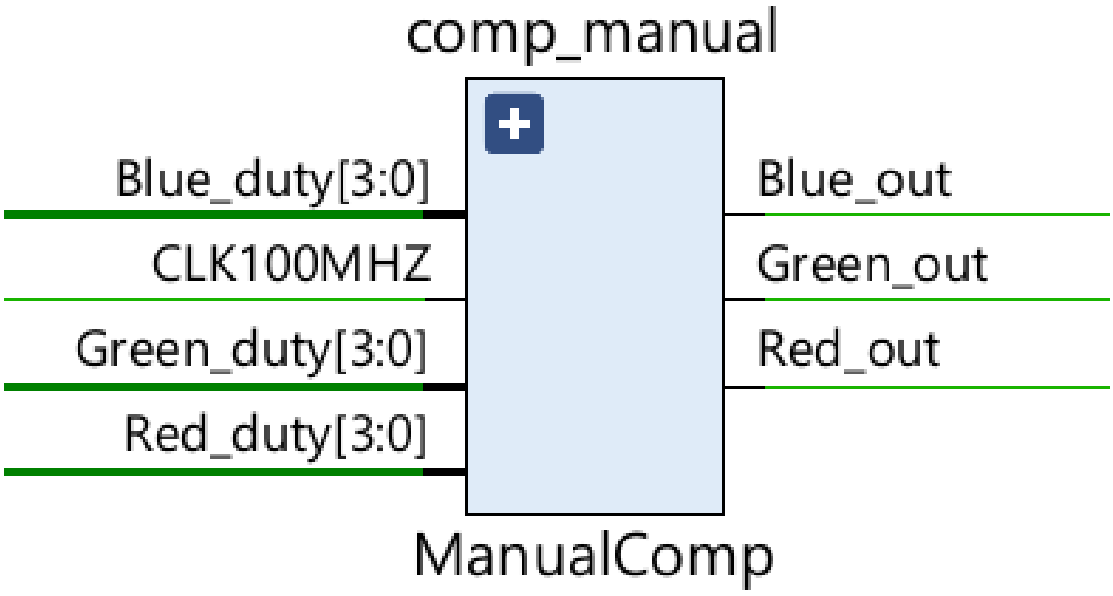
Componenta **PWMGen** implementează logica de **modulație a lățimii impulsului (PWM)** pentru un canal de ieșire.

La fiecare front crescător al semnalului Clk_in, un counter intern este incrementat. Comparând acest contor cu valoarea Duty (reprezentând ciclul de lucru dorit), componenta generează semnalul Clk_Out:

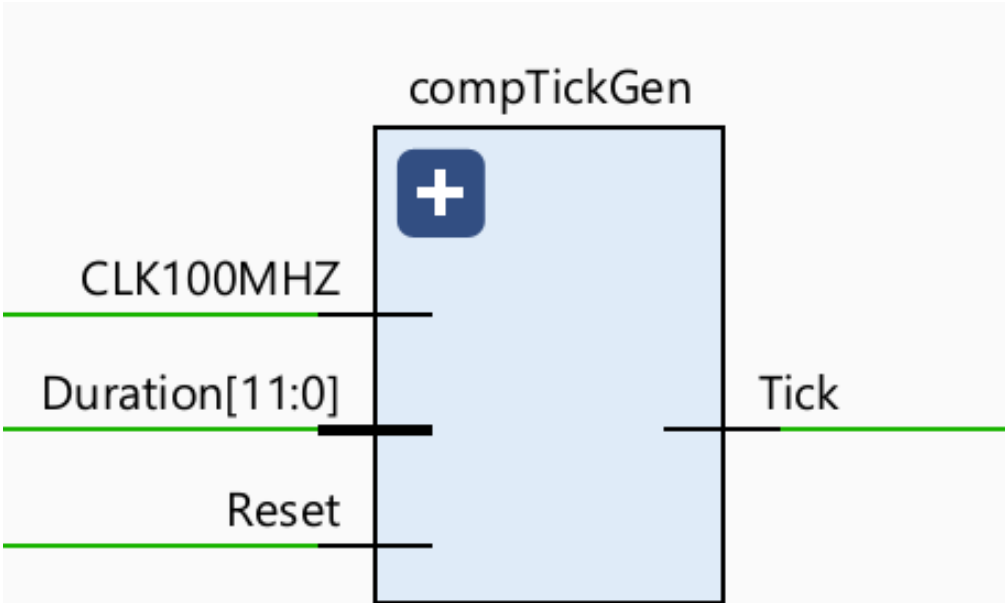
- Dacă count < Duty, semnalul este setat la ‘1’ (activ),
- În rest, este ‘0’ (inactiv).



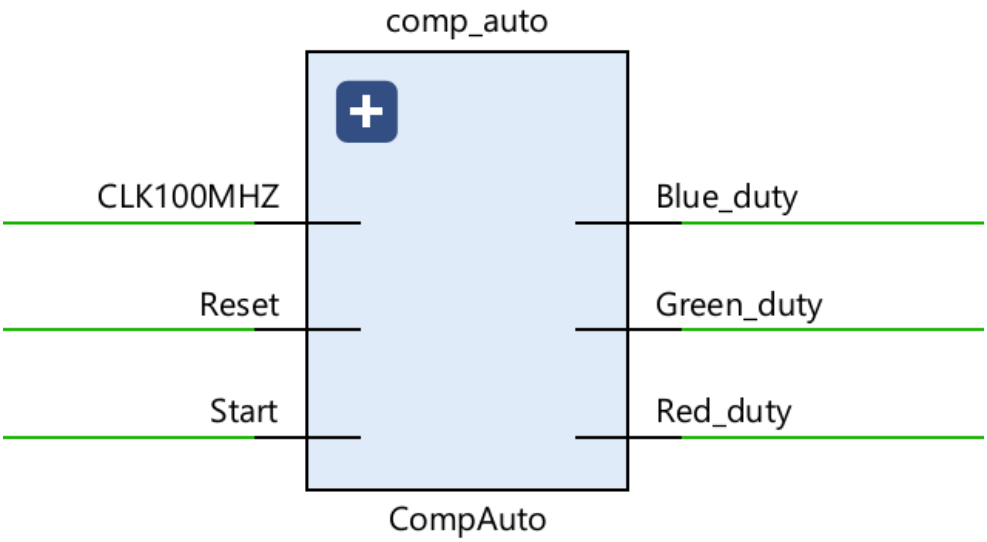
Componenta **ManualComp** implementează logica necesară pentru funcționarea sistemului în **modul Manual**, permițând controlul direct al intensității fiecărei componente de culoare a LED-ului RGB. Fiecare culoare este controlată printr-un semnal de tip Duty pe 4 biți, primit ca intrare, ceea ce permite 16 niveluri discrete de luminozitate, adică 4096 de culori posibile. Intern, componenta utilizează un PWMClockGen pentru a reduce frecvența ceasului de la 100 MHz la un semnal de 5 kHz, potrivit pentru generarea PWM-urilor vizibile. Apoi, pentru fiecare culoare, este instanțiat câte un modul PWMGen, configurat cu o rezoluție de 4 biți, care produce semnalul PWM corespunzător valorii de intensitate primite.



Componenta **TickGen** are rolul de a genera un semnal de tip „tick” cu o frecvență controlabilă, utilizat ca semnal de ceas pentru tranzițiile progresive din modurile **Test** și **Automat**. Această componentă funcționează ca un divizor de frecvență parametrizabil, care generează impulsuri de durată constantă la intervale configurabile de timp, în funcție de valoarea de intrare **Duration**. Semnalul Tick devine activ pentru un singur ciclu de ceas atunci când contorul intern atinge o valoare-țintă, calculată pe baza unei durate unitate și a numărului de evenimente care definesc un ciclu complet de culoare.

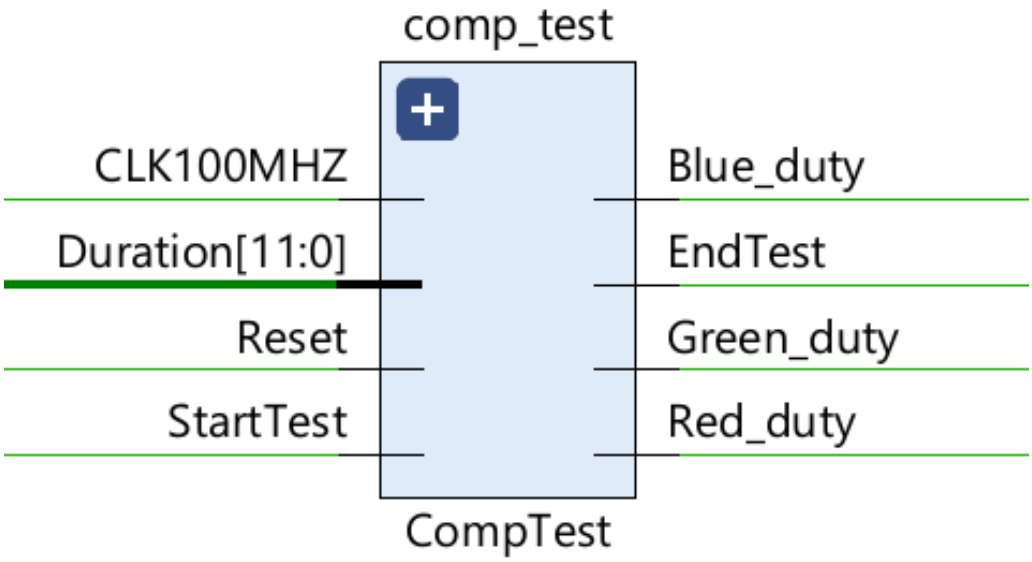


Componenta **CompAuto** controlează comportamentul sistemului în modul automat. Comportamentul acestei componente este determinat de un FSM ce controlează valorile componentelor de culoare (Red_comp, Green_comp, Blue_comp) pe baza unui contor și a unei succesiuni de stări logice (Red, G_Up, R_Down, B_Up, G_Down, R_Up, B_Down). Tranzițiile între stări sunt temporizate de semnalul Tick, generat de componenta **TickGen**. Secvența componentei este activată de semnalul Start sau mai exact **StartAuto** din UE.



Componenta **CompTest** implementează comportamentul sistemului în **modul de test**, în care LED-urile RGB sunt activate succesiv. Această secvență este realizată din nou printr-un FSM cu stările Idle, Red, Green, Blue și Done. Tranzițiile între stări sunt sincronizate prin semnalul Tick, generat de componenta **TickGen**. De asemenea, este parametrizată de

variabila **Duration** și secvența de test este inițiată de activarea semnalului **StartTest**.



4. Cod VHDL

Componenta top reprezintă entitatea principală și este responsabilă de coordonarea generală a sistemului în funcție de modul selectat de utilizator. Aceasta înglobează toate subcomponentele dezvoltate anterior – ManualComp, CompTest, CompAuto, TickGen, PWMGen, PWMClockGen, precum și module auxiliare precum displ7seg (pentru afișaj) și MPG (pentru debouncing-ul butoanelor).

Funcționarea generală se bazează pe un FSM, care gestionează modurile sistemului: Idle, Manual, Test, Auto, WaitTest, WaitAuto. La pornire sau după reset, sistemul este în Idle. Utilizatorul selectează modul dorit prin semnalul Mode(1:0) (din switch-uri) și îl validează prin butonul OK. În funcție de selecție, componenta top activează unul dintre cele trei module de execuție.

Ieșirile corespunzătoare fiecărui modul (Red_duty, Green_duty, Blue_duty) sunt comutate în funcție de starea activă, rezultând semnalele LEDRed, LEDGreen, LEDBlue. Butoanele fizice sunt conectate prin MPG pentru a genera semnale de tip puls (Ok, Reset) stabilite pe fronturi.

Comunicarea între UC și UE este reflectată în semnalele interne:

- StartTest, StartAuto – semnale de control pentru activarea modurilor Test și Automat.
- EndTest – semnal de finalizare de la CompTest către UC.

- Indicators – LED-uri simple pentru afișarea poziției switch-urilor.
- Data7seg, Char, Anodes – semnale pentru controlul afișajului cu 7 segmente.

Comportamentul sistemului este reflectat vizual și pe display-ul cu 7 segmente, în funcție de modul activ:

- În modul Manual – se afișează valorile RGB în format hex.
- În modul Test – se afișează codul de modul și durata.
- În modul Auto – se afișează codul „10”.
- În Idle – se afișează „FF”.

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

entity top is
    Port ( Mode : in STD_LOGIC_VECTOR (1 downto 0);
           CLK100MHZ : in STD_LOGIC;
           SW : in STD_LOGIC_VECTOR (11 downto 0);
           btnOk : in STD_LOGIC;
           btnReset : in STD_LOGIC;

           Anodes : out STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
           Char : out STD_LOGIC_VECTOR (7 downto 0) := (others => '0');

           LEDRed : out STD_LOGIC := '0';
           LEDGreen : out STD_LOGIC := '0';
           LEDBlue : out STD_LOGIC := '0';

           Indicators: out STD_LOGIC_VECTOR(13 downto 0) := (others =>
'0'));
end top;

architecture Behavioral of top is

    component displ7seg is
    Port ( Clk  : in  STD_LOGIC;
           Rst  : in  STD_LOGIC;
           Data : in  STD_LOGIC_VECTOR (31 downto 0);   -- datele pentru 8
cifre (cifra 1 din stanga: biti 31..28)
           An   : out STD_LOGIC_VECTOR (7 downto 0);    -- selectia
anodului activ
```

```
           Seg   : out STD_LOGIC_VECTOR (7 downto 0));    -- selectia
catozilor (segmentelor) cifrei active
    end component;


    component CompAuto is
        generic(NoBits: integer := 10);
        Port ( Red_duty : out STD_LOGIC;
               Green_duty : out STD_LOGIC;
               Blue_duty : out STD_LOGIC;
               Start : in STD_LOGIC;
               Reset : in STD_LOGIC;
               CLK100MHZ: in std_logic);
    end component;

    component CompTest is
        generic(NoBits: integer := 8);
        Port ( Red_duty : out STD_LOGIC;
               Green_duty : out STD_LOGIC;
               Blue_duty : out STD_LOGIC;
               StartTest : in STD_LOGIC;
               EndTest : out STD_LOGIC;
               Duration : in STD_LOGIC_VECTOR (11 downto 0);
               Reset : in STD_LOGIC;
               CLK100MHZ: in std_logic);
    end component;

    component MPG is
    Port ( btn : in STD_LOGIC;
           clk : in STD_LOGIC;
           en : out STD_LOGIC);
    end component;

    component ManualComp is
        Port ( CLK100MHZ : in STD_LOGIC;
               Red_duty : in STD_LOGIC_VECTOR (3 downto 0);
               Green_duty : in STD_LOGIC_VECTOR (3 downto 0);
               Blue_duty : in STD_LOGIC_VECTOR (3 downto 0);
               Red_out : out STD_LOGIC;
               Green_out : out STD_LOGIC;
               Blue_out : out STD_LOGIC);
    end component;

    type STATE_TYPE is (Idle, Manual, Test, Auto, WaitTest, WaitAuto);
    signal state : STATE_TYPE := Idle;

    signal Ok : STD_LOGIC := '0';
    signal Reset : STD_LOGIC := '0';
```

```
    signal Manual_Red_duty : STD_LOGIC := '0';
    signal Manual_Green_duty : STD_LOGIC := '0';
    signal Manual_Blue_duty : STD_LOGIC := '0';

    signal Test_Red_duty : STD_LOGIC := '0';
    signal Test_Green_duty : STD_LOGIC := '0';
    signal Test_Blue_duty : STD_LOGIC := '0';

    signal Auto_Red_duty : STD_LOGIC := '0';
    signal Auto_Green_duty : STD_LOGIC := '0';
    signal Auto_Blue_duty : STD_LOGIC := '0';

    signal StartTest, EndTest, StartAuto: STD_LOGIC := '0';

    signal Data7seg : STD_LOGIC_VECTOR (31 downto 0) := (others => '0');

begin
    Indicators(11 downto 0) <= SW(11 downto 0);
    Indicators(13 downto 12) <= Mode;

    proc_update_state: process(Reset, Clk100MHZ)
    begin
        if Reset = '1' then
            state <= Idle;
            StartAuto <= '0';
            StartTest <= '0';
        elsif rising_edge(Clk100MHZ) then
            case state is
                when Idle =>
                    StartTest <= '0';
                    if Ok = '1' then
                        case Mode is
                            when "00" => state <= Manual;
                            when "01" =>
                                state <= Test;
                                StartTest <= '1';

                            when "10" =>
                                StartAuto <= '1';
                                state <= Auto;
                            when others => state <= Idle;
                        end case;
                    end if;

                when Manual =>
                    if Ok = '1' then
                        state <= Idle;
                    end if;
```

```
            when Test =>

                if EndTest = '1' then
                    state <= Idle;
                    StartTest <= '0';
                end if;

            when Auto =>

                if Ok = '1' then
                    StartAuto <= '0';
                    state <= Idle;
                end if;

            when others => state <= Idle;

        end case;

    end if;
end process;

proc_change_state: process(state)
begin
    case state is
        when Idle =>
            LEDRed <= '0';
            LEDGreen <= '0';
            LEDBlue <= '0';

        when Manual =>
            LEDRed <= Manual_Red_duty;
            LEDGreen <= Manual_Green_duty;
            LEDBlue <= Manual_Blue_duty;

        when Test =>
            LEDRed <= Test_Red_duty;
            LEDGreen <= Test_Green_duty;
            LEDBlue <= Test_Blue_duty;

        when Auto =>
            LEDRed <= Auto_Red_duty;
            LEDGreen <= Auto_Green_duty;
            LEDBlue <= Auto_Blue_duty;

        when others =>
            LEDRed <= '0';
            LEDGreen <= '0';
```

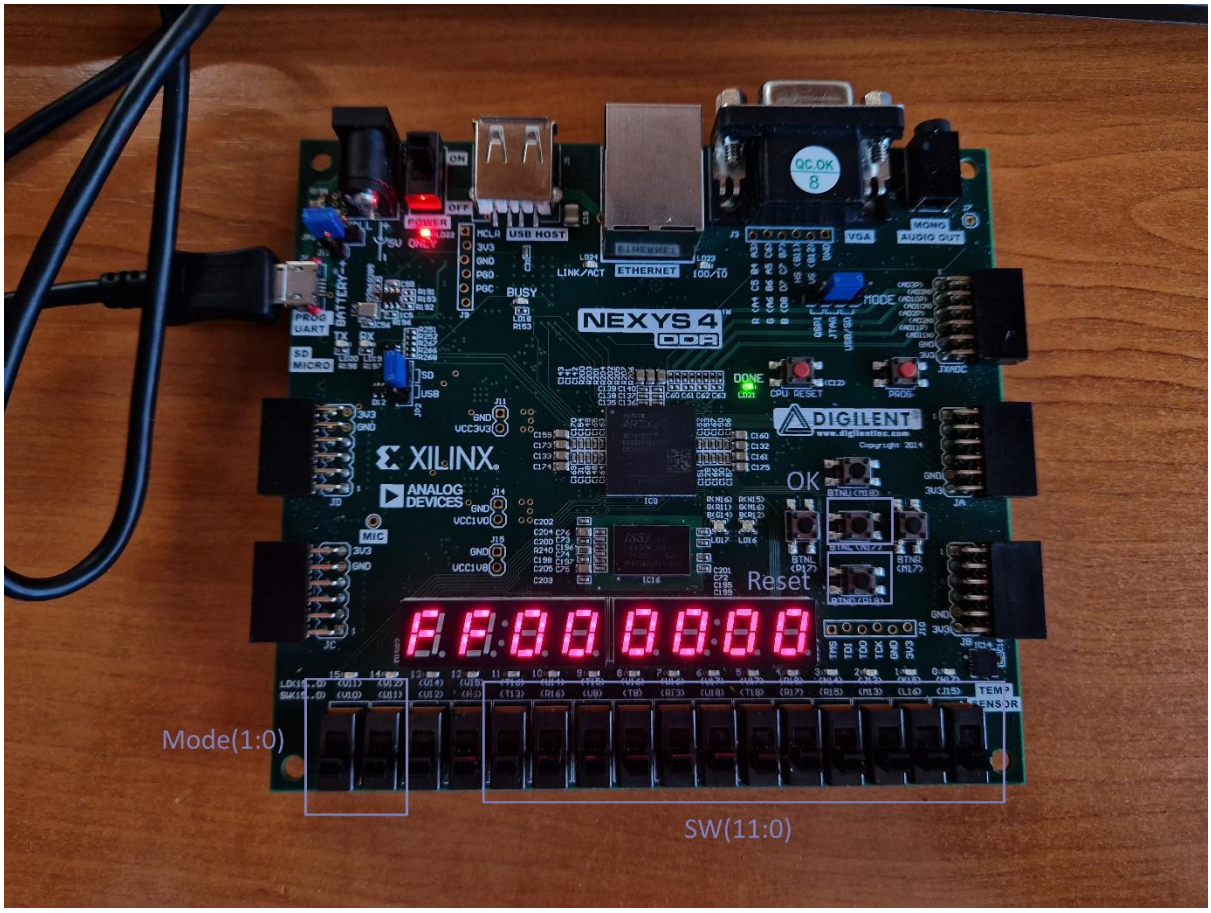
```
            LEDBlue <= '0';

        end case;
    end process;

    mpg_ok: MPG port map(btnOk, CLK100MHZ, Ok);
    mpg_reset: MPG port map(btnReset, CLK100MHZ, Reset);

    comp_manual: ManualComp port map(CLK100MHZ,
                                     SW(11 downto 8),    -- red
                                     SW(7 downto 4),     -- green
                                     SW(3 downto 0),     -- blue
                                     Manual_Red_duty,
                                     Manual_Green_duty,
                                     Manual_Blue_duty);

    comp_test: CompTest generic map(8)
                        port map(Test_Red_duty,
                                 Test_Green_duty,
                                 Test_Blue_duty,
                                 StartTest,
                                 EndTest,
                                 SW(11 downto 0),
                                 Reset,
                                 CLK100MHZ);

    comp_auto: CompAuto generic map(4)
                        port map(Auto_Red_duty,
                                 Auto_Green_duty,
                                 Auto_Blue_duty,
                                 StartAuto,
                                 Reset,
                                 CLK100MHZ);


    proc7seg: process(state)
    begin
        case state is
                when Manual =>
                    Data7seg(31 downto 28) <= x"0";
                    Data7seg(27 downto 24) <= x"0";

                    -- red
                    Data7seg(23 downto 20) <= x"0";
                    Data7seg(19 downto 16) <= SW(11 downto 8);

                    -- green
                    Data7seg(15 downto 12) <= x"0";
                    Data7seg(11 downto 8) <= SW(7 downto 4);
```

```
                    -- blue
                    Data7seg(7 downto 4) <= x"0";
                    Data7seg(3 downto 0) <= SW(3 downto 0);

                when Test =>
                    Data7seg(31 downto 28) <= x"0";
                    Data7seg(27 downto 24) <= x"1";

                    -- duration
                    Data7seg(11 downto 8) <= SW(11 downto 8);
                    Data7seg(7 downto 4) <= SW(7 downto 4);
                    Data7seg(3 downto 0) <= SW(3 downto 0);

                when Auto =>
                    Data7seg(31 downto 28) <= x"1";
                    Data7seg(27 downto 24) <= x"0";

                    Data7seg(23 downto 0) <= x"000000";

                when others =>
                    Data7seg(31 downto 28) <= x"F";
                    Data7seg(27 downto 24) <= x"F";
                    Data7seg(23 downto 0) <= x"000000";
            end case;
    end process;

    Control7segm: displ7seg port map(CLK100MHZ, Reset, Data7seg, Anodes,
Char);


end Behavioral;
```

5. Utilizare și semnificația notațiilor

Stare inițială - modul Idle

Când sistemul este pornit sau butonul Reset este apăsat, sistemul intră în starea Idle. LED-urile RGB sunt stinse, iar afișajul cu 7 segmente afișează codul "FF", indicând absența unui mod activ.



Modul Manual

Pentru a accesa modul Manual:

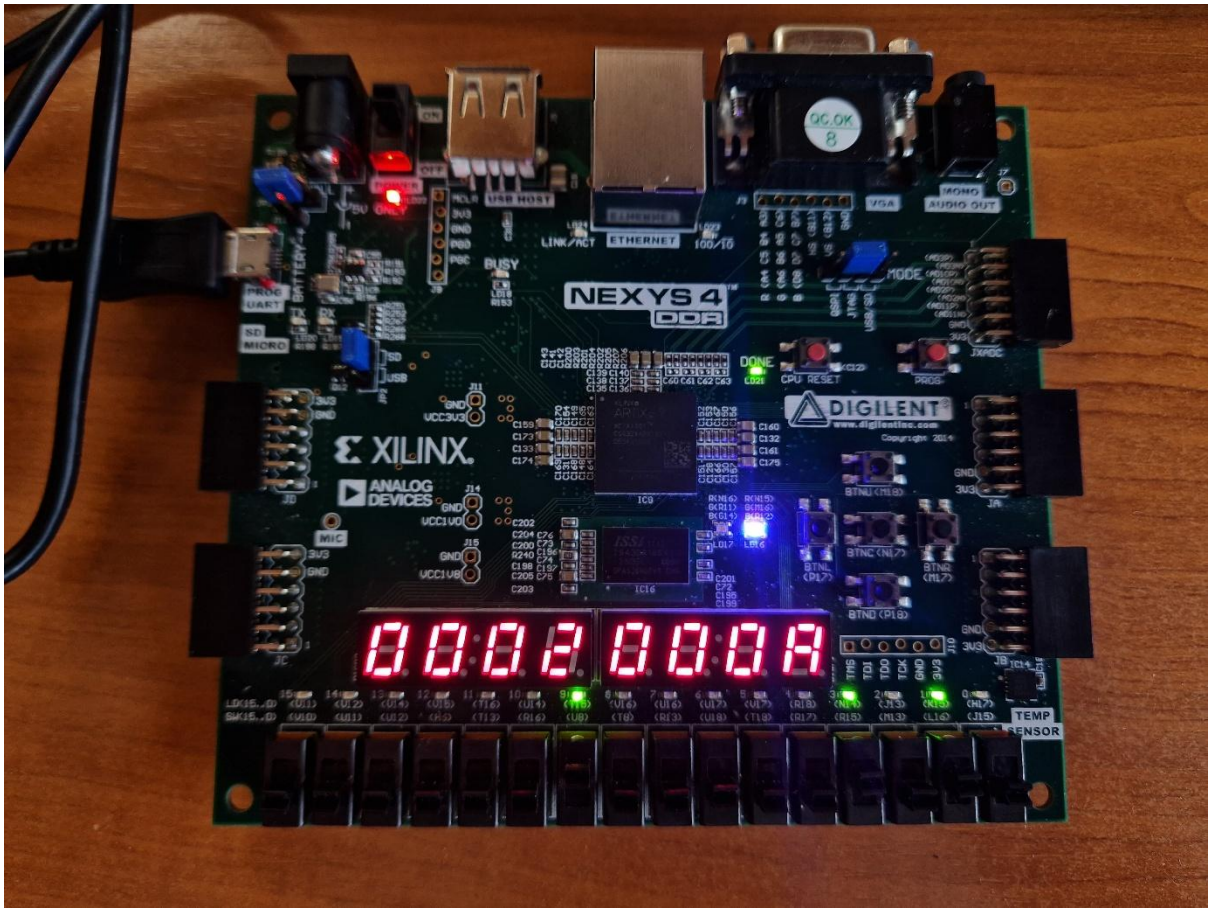
- Comutatoarele Mode(1:0) se setează pe 00.
- Se apasă butonul OK.

Sistemul trece în modul Manual, iar pe afișaj apar valorile pentru fiecare componentă RGB în format hexazecimal. Utilizatorul poate controla intensitatea fiecărei componente:

- SW(11:8) pentru roșu
- SW(7:4) pentru verde
- SW(3:0) pentru albastru

Fiecare componentă are 16 niveluri de intensitate (0-15), iar combinațiile lor generează până la 4096 de culori posibile.

Pe afișaj apare codul "00" și intensitatea fiecărei componente.



Ieșirea din modul Manual se face prin apăsarea din nou a butonului OK sau prin Reset.

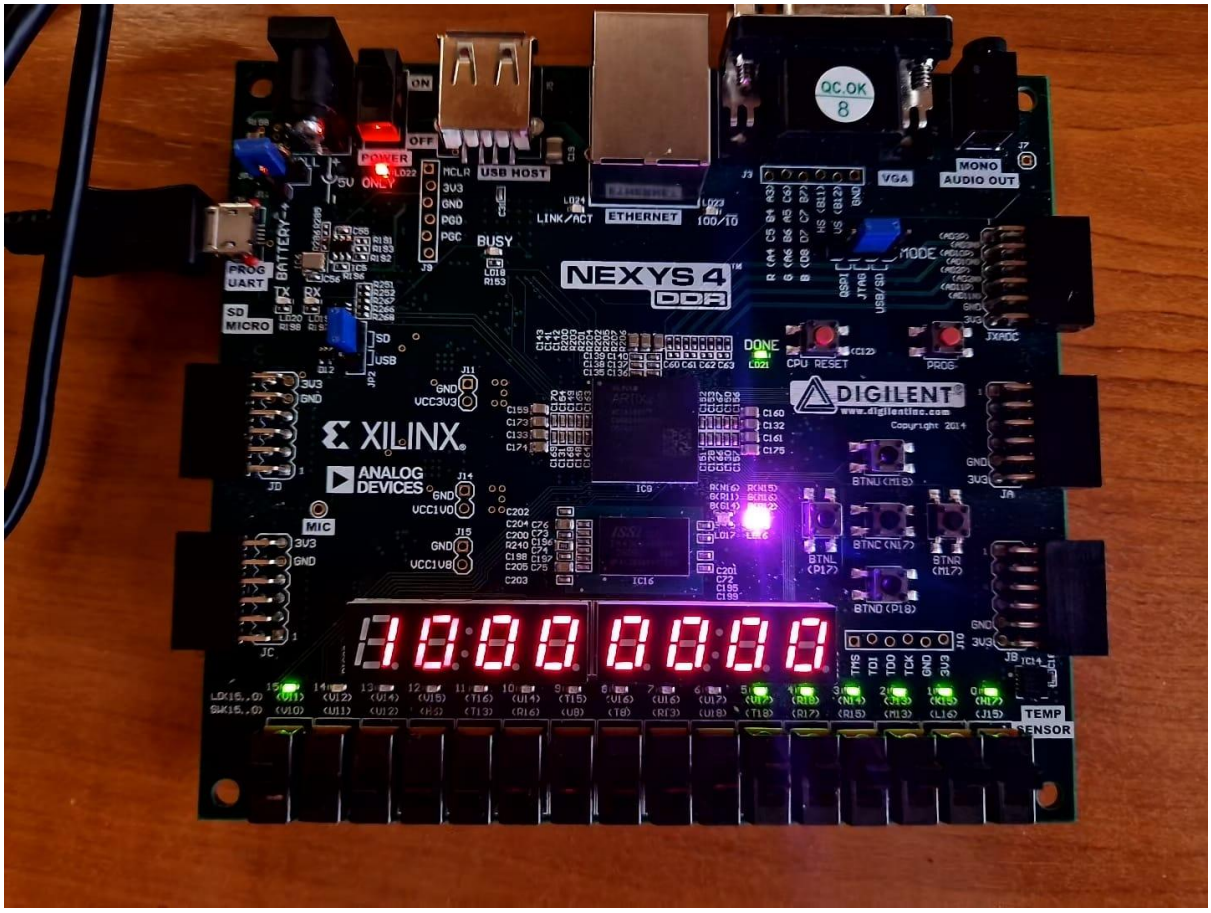
Modul Automat

Pentru a accesa modul Automat:

- Comutatoarele Mode(1:0) se setează pe 10.
- Se apasă butonul OK.

Sistemul generează automat tranziții line între culorile RGB, fără intervenția utilizatorului. Tranzițiile sunt controlate de semnalul Tick, al cărui ritm este influențat de o durată configurabilă.

Pe afișajul cu 7 segmente apare codul "10".



Ieșirea din acest mod se face cu butonul OK sau Reset.

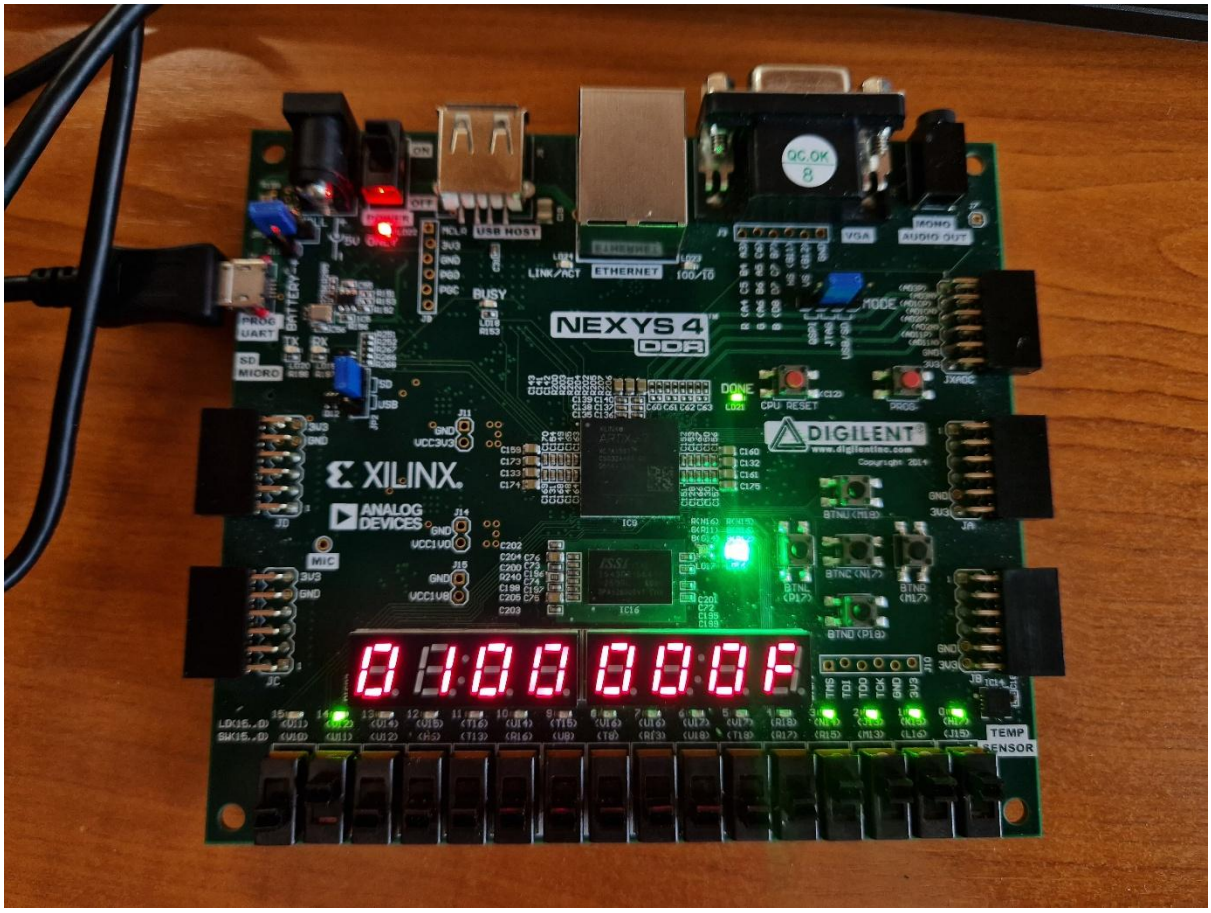
Modul Test

Pentru a accesa modul Test:

- Comutatoarele Mode(1:0) se setează pe 01.
- Se apasă butonul OK.

Sistemul activează succesiv LED-urile pentru roșu, verde și albastru, crescând și apoi scăzând intensitatea fiecărei componente. Secvența este controlată de un semnal Tick, influențat de valoarea introdusă prin SW(11:0).

Pe afișaj apare codul "01" și durata de execuție.



La finalul secvenței, sistemul revine automat în Idle sau se poate întrerupe prin apasărea butonului de Reset.

6. Justificarea soluției alese

Am implementat proiectul într-un mod modular, separând fiecare funcționalitate (Manual, Test, Automat) în componente dedicate, pentru a obține o structură clară, organizată și ușor de extins. Fiecare modul gestionează propriul comportament și generează semnalele PWM necesare, folosind aceleași componente de bază (PWMGen, PWMClockGen). Această abordare permite testarea și modificarea independentă a fiecărui mod, fără a afecta restul sistemului. De asemenea, logica de comutare între moduri este centralizată în componenta principală, ceea ce face sistemul ușor de controlat și de integrat cu interfața utilizatorului (buton OK, Reset, switch-uri, LED-uri, display). Principalul dezavantaj este creșterea numărului de semnale și a complexității interconectărilor interne, însă acest aspect este compensat de claritate.

7. Oportunități de dezvoltare

În timpul testării pe placă, am observat un efect de flicker pe LED-ul RGB în timpul tranzițiilor din modul Test. Acest comportament nedorit apare, cel mai probabil, din cauza discrepanței dintre frecvența cu care sunt actualizate valorile PWM și frecvența semnalului de ceas care controlează tranzițiile între culori. Pe viitor, mi-aș dori să elimin acest efect pentru a obține tranziții mai fluide și un comportament vizual mai plăcut, apropiat de o estompare continuă a culorilor.

8. Bibliografie

- https://digilent.com/reference/_media/nexys:nexys4:nexys4_rm.pdf
- Materialele de curs și laborator