UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

Structura sistemelor de calcul

Documentație proiect

Universitatea Tehnică din Cluj-Napoca
Catedra de Calculatoare

Titlul proiectului:

Joc de memorie

Autor: Tămașoi Ștefania Maria
Grupa: 30239
Îndrumător: Popa Cristian Mihai

Data: Friday 17th January, 2025

Contents

Rezumat

Proiectul acesta își propune să creeze un joc simplu de memorie care ajută utilizatorii să-și antreneze capacitatea de a memora secvențe de numere. Jocul folosește un sistem hardware bazat pe plăcile NEXYS A7 și modulele PMod KYPD și PMod USBUART. Utilizatorul trebuie să memoreze și să repete secvențele de numere afișate pe un display SSD. Pe măsură ce jucătorul avansează, jocul devine tot mai greu, scurtându-se timpul în care sunt afișate numerele.

Am realizat jocul in VHDL pentru a gestiona atât afișarea, cât și inputul de date. Modulul PMod KYPD este folosit pentru a introduce numerele, iar PMod USBUART trimite scorurile pe un PC pentru a fi salvate și analizate.

1 Introducere

Jocurile de memorie sunt foarte populare și sunt folosite atât pentru învățare, cât și pentru distracție. Ele ajută la antrenarea memoriei și atenției utilizatorului. În ultimii ani, dezvoltarea tehnologiilor hardware, cum ar fi plăcile FPGA, a făcut posibilă crearea unor jocuri interactive mai performante și personalizabile. Aceste tehnologii permit folosirea de module externe, precum tastaturi și ecrane SSD, și permit interacțiunea cu PC-uri pentru a salva și analiza scorurile și progresul utilizatorilor.

Proiectul de față se concentrează pe dezvoltarea unui joc de memorie folosind plăcile NEXYS A7 și modulele PMod KYPD și PMod USBUART. FPGA-urile sunt populare pentru că pot procesa informații rapid și pot implementa algoritmi mai complecși. Folosirea acestora în jocuri oferă o abordare eficientă și rapidă pentru logica jocului. În plus, conectarea la un PC pentru salvarea scorurilor și analiza progresului utilizatorilor îmbunătățește experiența acestora, oferind feedback detaliat despre performanță.

Soluția propusă este diferită de altele, deoarece folosește un sistem FPGA personalizat care combină mai multe module externe, în loc de platforme software pe PC sau soluții standard. De asemenea, jocul se adaptează la performanțele utilizatorului, oferind un feedback dinamic și o dificultate în creștere, ceea ce îmbunătățește experiența de joc.

În continuare, voi discuta despre tehnologiile și platformele folosite în acest proiect pentru a înțelege cum fiecare componentă contribuie la crearea jocului de memorie.

2 Fundamentare teoretică

În această secțiune, voi discuta despre tehnologiile și modelele utilizate pentru implementarea jocului, precum FPGA-urile si modulele PMod.

2.1 Plăcuța FPGA

FPGA NEXYS A7 se distinge prin flexibilitatea și performanțele sale, fiind echipată cu un cip Artix-7 de la Xilinx. În literatura de specialitate, FPGA-urile sunt recunoscute pentru capacitatea lor de a reduce latențele și de a îmbunătăți performanța în aplicațiile care necesită un răspuns rapid, ceea ce le face ideale pentru jocuri interactive în timp real.

2.2 Tehnologia PMod

Modulul PMod KYPD, utilizat în acest proiect, este un keypad care permite utilizatorului să introducă date. Aceste module sunt conectate direct la plăcile FPGA prin interfețe de tipul I/O și permit o integrare ușoară în proiecte care necesită un input din partea utilizatorului.

2.3 Comunicarea cu PC-ul folosind PMod USBUART

Pentru a comunica cu un PC și a analiza scorurile, proiectul utilizează modulul PMod USBUART, care permite trimiterea datelor printr-o conexiune USB. Acest modul este folosit pentru a trimite scorurile și progresul utilizatorilor către un PC, unde datele pot fi analizate și stocate pentru feedback ulterior.

2.4 Soluția propusă - Contribuții și Noutăți

Ceea ce este unic în această lucrare este integrarea unui sistem hardware care permite un feedback dinamic al dificultății jocului.

2.5 Literatura utilizată

1. "Nexys A7 Digilent Reference Manual"
2. PMod KYPD Reference Manual
3. PMod USBuart Reference Manual

3 Proiectare şi implementare

Aceasta este partea principală a raportului proiectului și va conține descrierea detaliată a etapelor parcurse pentru realizarea obiectivelor proiectului. În această secțiune voi aborda următoarele aspecte:

3.1 Metoda experimentală utilizată

Pentru realizarea jocului de memorie, am utilizat o placă FPGA, pe care am implementat logica jocului în VHDL.Pentru validarea scorului și interfața jocului am folosit HTerm.

3.2 Arhitectura generală a sistemului

Memory Comparator

- **Wrapper**: Conține generatorul de numere pseudoaleatoare, cele divizări de clock în funcție de valoarea speed, o componentă SSD pentru afișarea numerelor în timpul afișării
- **PmodKYPD**:Dispune de un decodificator, care citește semnalele de la un keypad 4x4, activând fiecare coloană pe rând și verificând starea rândurilor pentru a detecta butonul apăsat. Fiecare coloană este activată pentru 1 ms, iar valorile de pe rânduri sunt citite pentru a determina butonul. Când un buton este apăsat, ieșirea

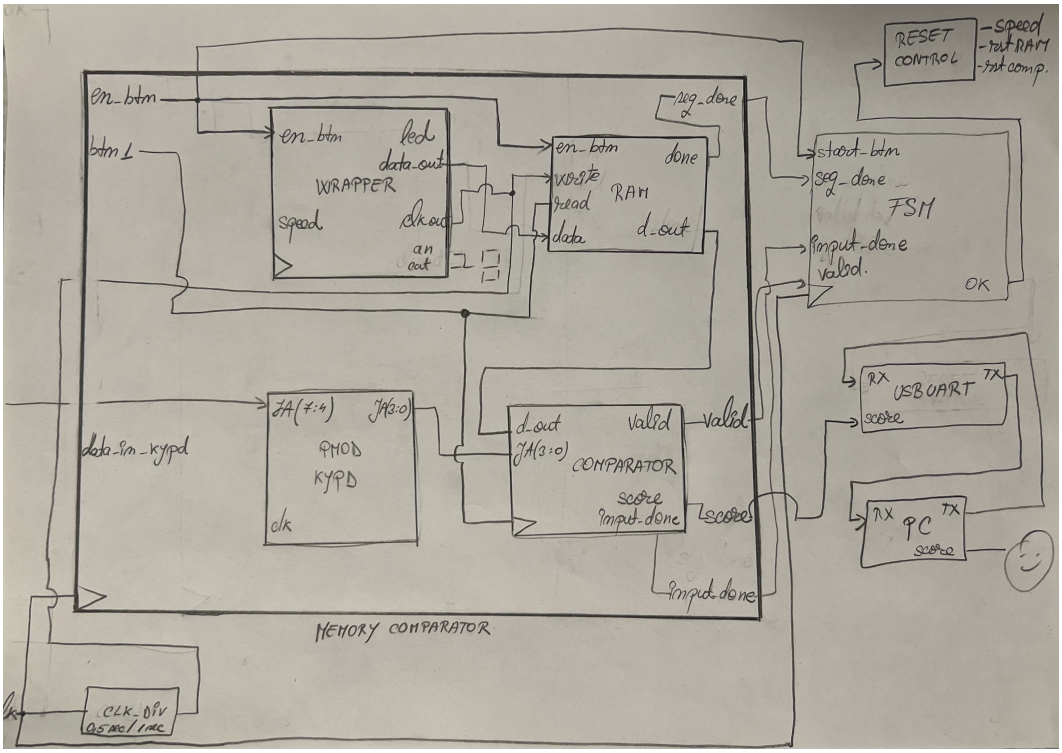

Figure 1: Schema Bloc

DecodeOut emite o valoare corespunzătoare. Procesul de scanare continuă prin schimbarea coloanelor și citirea rândurilor la intervale de 1 ms. Decoderul folosește un contor intern pentru a sincroniza această scanare.

- **RAM**: Memoria este personalizată în asa fel încât butonul de enable al RAM-ului este același buton ca și în Wrapper. Scrierea se face pe semnalul de clock divizat, iar citirea pe semnalul provenit de la butonul 1 de validare a fiecărui număr introdus din KYPD

- **Comparator**: Comparatorul preia datele de ieșire din RAM și datele din KYPD pentru a fi comparate ulterior. Aici se calculează scorul și se trimit semnale de validare în FSM.

PmodUSBUART

Sistemul este împărțit în două părți principale: una pentru transmiterea datelor și alta pentru recepționarea acestora.

- **Transmiterea:** Componenta PmodUSBUART_tx gestionează transmiterea UART. Aceasta trimite date câte un byte folosind o viteză de 9600 baudi. Transmiterea este controlată de semnalul tx_en, iar datele de transmis sunt selectate pe baza unui contor și a stării comutatorului de intrare. Datele sunt convertite în caractere ASCII (precum '0', '1', 'A', etc.) și trimise prin linia tx.

- **Recepția:** Componenta PmodUSBUART_rx gestionează recepția datelor. Aceasta primește date seriale prin linia rx, iar atunci când datele sunt gata, ele sunt sto-

cate în rx_data. Datele primite sunt convertite într-o reprezentare binară de 6 biți (corespunzând valorilor ASCII) și salvate într-un registru.

FSM (Finite State Machine)
Controlează logica jocului, având 5 stări principale:

- **IDLE** – Este starea de așteptare, unde sistemul nu face nimic până când nu primește semnalul de start de la un buton (`start_btn`).
- **SHOW** – Aici se afișează o secvență. După ce secvența este completă (semnalul `sequence_done` este activ), FSM trece la următoarea stare.
- **INPUT** – Utilizatorul introduce o secvență de date. Când intrarea este completă (semnalul `input_done` este activ), FSM avansează la validarea datelor.
- **VALIDATE** – Datele introduse sunt verificate. Dacă sunt valide (semnalul `valid` este activ), se trece la starea de **NEXT_LEVEL**.
- **NEXT_LEVEL** – Se avansează la un nou nivel. Dacă nivelul curent nu este maxim, FSM revine la starea **SHOW** pentru a începe o nouă secvență. Dacă nivelul a atins valoarea maximă, sistemul se întoarce în starea **IDLE**.

Semnalul `ok` este activat doar în starea **VALIDATE** atunci când datele introduse sunt valide.

Reset Control

- Resetează Wrapper, Comparator, RAM și trimite un semnal în speed-ul din Wrapper pentru a determina nivelul (clk divizat 1 sec/0.5 sec).

3.3 Algoritmii implementați

Pentru generarea automată a numerelor am folosit un generator de numere pseudoaleatoare (Linear Feedback Shift Register), unde se vor genera numere la interval de 1 secundă sau 0.5 secunde (în funcție de nivel). Toate acestea sunt stocate într-o memorie RAM personalizată care face scrierea pe semnalul de clock divizat. Memoria stochează cele 10 numere pentru a fi comparate ulterior. Prin Pmod KYPD se introduc numerele și se compară pe rând cu numerele din RAM. Se calculeaza scorul (câte numere introduse coincid cu cele din secvența afișată).

3.4 Manual de utilizare

- Înainte de a începe, utilizatorul trebuie să aibă instalat Vivado. "Vivado 2024.1"
- Se încarcă codul sursă și se conectează modulele precum conform fișierului de constrângeri.
- Utilizatorul ține apăsat butonul de sus și se va afișa o secvență de 10 numere, terminarea afișării fiind semnalată de primul led al plăcii.
- Apoi, se introduc numerele, însă după fiecare număr introdus, acesta trebuie validat, apăsând pe butonul din stânga a plăcii.
- După ce au fost introduse cele 10 numere (fie corect, fie greșit) utilizatorul va primi un verdict și anume, scorul.

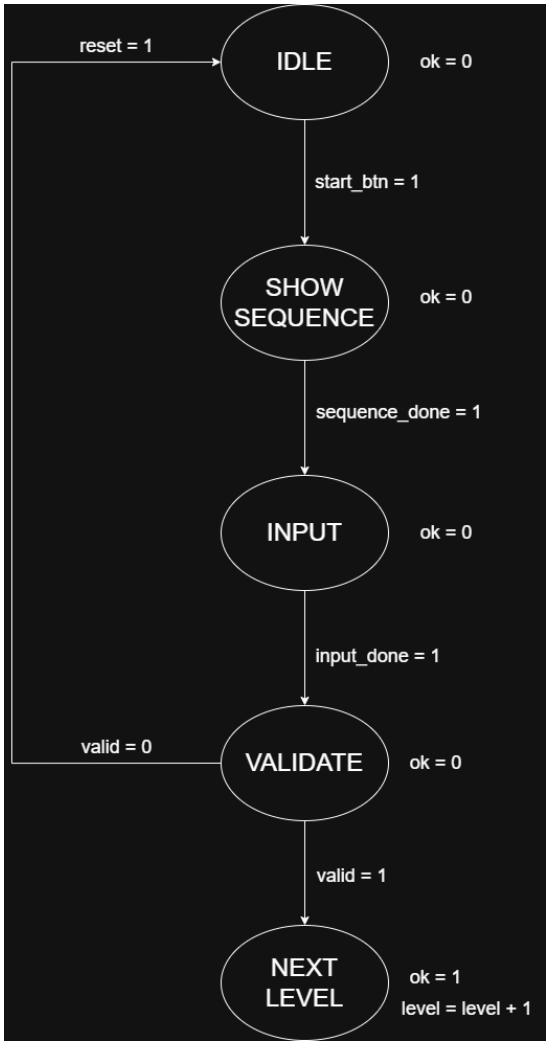

Figure 2: Diagramă stări

3.5 Alternative de proiectare

Mi-aș fi dorit să implementez un joc funcțional care să dispună de mult mai multe niveluri, afișarea ar fi trebuit sa se efectueze fără a fi necesară menținerea apasării butonului. Aceste lucruri ar fi sporit avantajul funcțional și interativ al proiectului.

4 Rezultate experimentale

4.1 Instrumentele de proiectare utilizate

- Limbajul de programare utilizat: [VHDL]
- Mediul software utilizat: [Xilinx Vivado, HTerm]
- Simulatorul utilizat: [Vivado Simulator]
- Platforma hardware utilizată: [NEXYS A7]
- Sistemul de operare: [Windows 10]

- Versiunea Vivado: [Vivado 2024.1]

4.2 Circuitul utilizat pentru implementare

Aici se vor detalia caracteristicile circuitului utilizat pentru implementare. Informațiile vor fi prezentate sub formă tabelară pentru a oferi o comparație clară între diferite metrici de performanţă.

Caracteristică	Valoare
Număr de blocuri logice	[9]
Număr de bistabile	[154]
Frecvenţa maximă de funcţionare	[0.228]
Consumul de energie	[0.091]

Table 1: Informații din rapoartele de implementare. Informațiile sunt extrase din mediul Vivado și analizate doar pe o parte a proiectului (memory comparator).

4.3 Procedura de testare utilizată

Pentru validarea sistemului, au fost utilizate mai multe metode de testare, incluzând simulări și teste pe hardware.

- **Tipul de testare utilizată:**
  - **Testare funcțională:** Aceasta a verificat corectitudinea operațiilor la nivel de comportament logic al fiecărui modul in parte.
- **Testbench:**
  - **Simulare pe platforma de dezvoltare:** Am utilizat Vivado Simulator pentru a valida comportamentul designului.
  - **Testare pe hardware:** Implementarea a fost sintetizată și încărcată pe o placă FPGA (ex: Xilinx Artix-7).
- **Utilitare de testare dezvoltate:**
  - **Scripturi de simulare:** Am creat fișiere de testbench în VHDL care să simuleze diferite scenarii.
  - **Programe de conversie:** Nu am folosit programe de conversie.

Rezultatele simulării

4.4 Dificultăţi întâlnite şi soluţii aplicate

În această subsecţiune se vor discuta dificultăţile întâmpinate pe parcursul implementării şi modul în care acestea au fost depăşite. Se vor analiza problemele legate de optimizarea performanței, sincronizarea modulelor hardware.

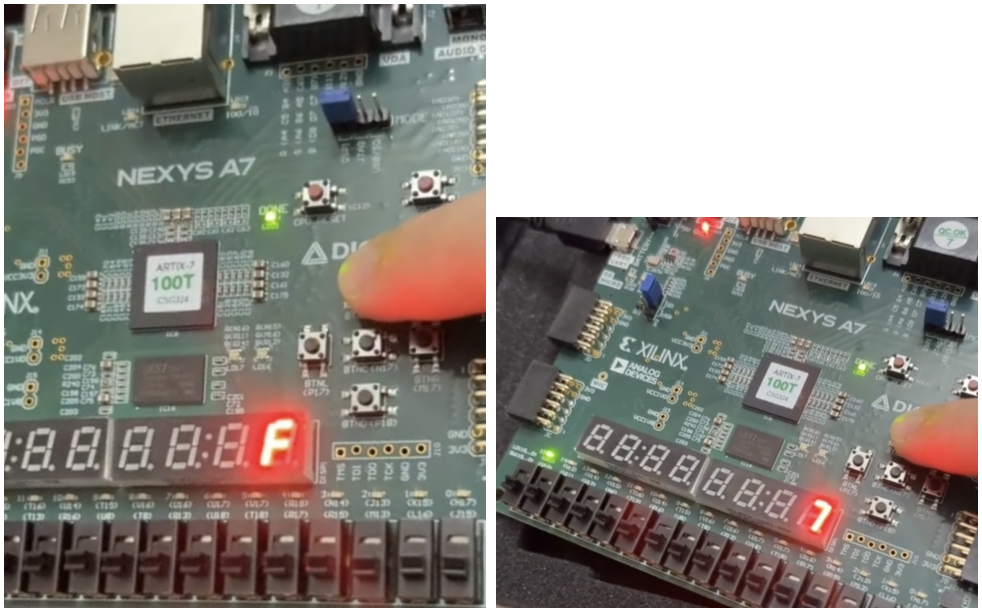

Figure 3: Simularea jocului de memorie

- Dificultăţi de implementare: [Nu am reușit să implementez integral toată funcționalitatea proiectului. Am petrecut foate mult timp încercând să implementez funcționalitatea afișării numerelor pe SSD.]
- Soluţii aplicate: [Fiecare componentă este independent funcțională (PMod KYPD, PMod USBuart, Wrapper, ClkDiv, RAM]

5 Concluzii

Proiectul a fost interesant, mi-a stârnit dorința de învățare în acest domeniu și mi-aș dori să continui pe viitor. Ideea proiectului și modul în care am gândit funcționalitatea întregului proiect a fost plăcută și sunt relativ mulțumită de rezultat. Desigur, pe viitor, voi încerca să îl aduc într-o formă finală.

5.1 Avantaje și dezavantaje ale proiectului

Avantajele proiectului realizat sunt:

- Posibilitatea de a dezvolta jocul pe termen lung, prin introducerea unor noi nivele de dificultate.

Dezavantajele acestui proiect includ:

- Timpul necesar pentru învățarea și configurarea platformei hardware, care poate constitui o barieră pentru utilizatorii mai puțin experimentați.

5.2 Aplicații pentru proiect

Acest proiect poate fi extins și adaptat pentru mai multe aplicații educaționale și de antrenament cognitiv. De exemplu, jocul poate fi folosit pentru:

- Îmbunătățirea abilităților de memorare pentru copii și adulți.
- Dezvoltarea unor aplicații educaționale care să ajute utilizatorii să învețe secvențe numerice, termeni sau concepte complexe prin intermediul unui joc.
- Crearea unor jocuri personalizate pentru diferite tipuri de învățare cognitivă, care să includă exerciții de memorare vizuală sau auditivă.

5.3 Dezvoltări viitoare

Pentru viitor, se propun următoarele dezvoltări:

- Extinderea jocului pentru a include mai multe nivele de dificultate și noi tipuri de secvențe care să ajute la dezvoltarea memoriei pe termen lung.
- Integrarea unui sistem de feedback vizual sau auditiv pentru a oferi utilizatorilor o mai bună experiență de învățare.
- Posibilitatea de a conecta mai multe module hardware pentru a crea un joc multiplayer, care să permită competiții între utilizatori.

6 Bibliografie

1. PModKYPD Programmable Logic
2. PModUSBuart Programmable Logic
3. Hennessy, J. L., Patterson, D. A., *Organizarea şi proiectarea calculatoarelor – interfaţa hardware/software*, Editura All Educational, Bucureşti, 2002.

Anexe

Anexa A

Codul sursă al jocului de memorie

Listing 1: Cod sursă Wrapper

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


entity Wrapper is
    Port ( clk      : in  std_logic;
           rst      : in  std_logic;
           en_btn   : in  std_logic;
           speed    : in  std_logic;
           cat      : out std_logic_vector(6 downto 0);
```

```
            an      : out std_logic_vector(7 downto 0);
            led     : out std_logic;            --se aprinde dupa ce se afise
            clk_out : out std_logic
            );
end Wrapper;

architecture Behavioral of Wrapper is

    signal clk_internal : std_logic;
    signal counter : std_logic_vector(3 downto 0) := (others => '0');
    signal completed : std_logic := '0';
    signal reg : std_logic_vector(7 downto 0) := "10001111";
    signal random_number: std_logic_vector(7 downto 0);
    signal clk_1, clk_05: std_logic;

    component ClkDivider_1sec is
        Port ( clk     : in  std_logic;
               rst     : in  std_logic;
               clk_div : out std_logic);
    end component;

    component ClkDivider_05sec is
        Port ( clk     : in  STD_LOGIC;
               rst     : in  STD_LOGIC;
               clk_div : out STD_LOGIC);
    end component;


    component 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);
                digit4 : in std_logic_vector(3 downto 0);
                digit5 : in std_logic_vector(3 downto 0);
                digit6 : in std_logic_vector(3 downto 0);
                digit7 : in std_logic_vector(3 downto 0);
                cat    : out std_logic_vector(6 downto 0);
                an     : out std_logic_vector(7 downto 0);
                clk    : in std_logic);
    end component;

begin


     Clk1: ClkDivider_1sec
           Port map (
               clk     => clk,
```

```
            rst     => rst,
            clk_div => clk_1
        );

Clk05: ClkDivider_05sec
        Port map (
            clk     => clk,
            rst     => rst,
            clk_div => clk_05
        );

clk_out <= clk_1 when speed = '0' else clk_05;
clk_internal <= clk_1 when speed = '0' else clk_05;

SSD_inst : SSD
    Port map (
        digit0 => random_number(3 downto 0),
        digit1 => "0000",
        digit2 => "0000",
        digit3 => "0000",
        digit4 => "0000",
        digit5 => "0000",
        digit6 => "0000",
        digit7 => "0000",
        cat    => cat,
        an     => an,
        clk    => clk_internal
    );

process(clk_internal, rst, en_btn)
begin
    if rst = '1' then
        reg <= "10001111";
        counter <= (others => '0');
        completed <= '0';
        led <= '0';
    elsif rising_edge(clk_internal) then
        if en_btn = '1' and completed = '0' then
            if counter = "1010" then
                completed <= '1';
                led <= '1';
            else
                reg <= reg(6 downto 0) & (reg(7) xor reg(5) xor reg(4)
                counter <= counter + 1;
            end if;
        end if;
    end if;
end process;
```

```
    random_number <= reg;

    clk_out <= clk_internal;


end Behavioral;
```

Listing 2: Cod sursă PMod USBuart

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


entity PmodUSBUART_topModule is
Port (
      clk: in std_logic;
      led: out std_logic_vector(3 downto 0);
      sw: in std_logic_vector(3 downto 0);
      btn: in std_logic_vector(3 downto 0);
      rx: in std_logic;
      tx: out std_logic
      );
end PmodUSBUART_topModule;

architecture Behavioral of PmodUSBUART_topModule is

component MPG is
    Port ( btn : in std_logic;
           clk : in std_logic;
           enable : out std_logic);
end component;

component PmodUSBUART_tx is
Port (
        clk: in std_logic;
        rst: in std_logic;
        baud_en: in std_logic;
        tx_en: in std_logic;
        tx_data: in std_logic_vector(7 downto 0);
        tx: out std_logic;
        tx_rdy: out std_logic
      );
end component;

component PmodUSBUART_rx is
Port(
     clk: in std_logic;
```

```
        rst: in std_logic;
        baud_en: in std_logic;
        rx: in std_logic;
        rd_data: out std_logic_vector(7 downto 0);
        rx_rdy: out std_logic
        );
end component;

signal baud_en, baud_en_x16: std_logic;
signal tx_start, tx_en, tx_rdy, tx_rdy1: std_logic;
signal rx_rdy, rx_rdy1:std_logic;
signal cnt: std_logic_vector(13 downto 0) := (others => '0');
signal cnt_x16: std_logic_vector(9 downto 0) := (others => '0');
signal tx_reg, rx_reg: std_logic_vector(23 downto 0);
signal tx_digit, rx_digit: std_logic_vector(5 downto 0);
signal tx_data: std_logic_vector(7 downto 0);
signal rx_data: std_logic_vector(7 downto 0) := (others => '0');
signal tx_digit_cnt: std_logic_vector(1 downto 0);
signal rx_digit_cnt: std_logic_vector(1 downto 0) := (others => '0');
signal en: std_logic;
signal transmitionData: std_logic_vector(23 downto 0);

begin

    monopulse: MPG port map(btn(0),clk, en);

    --UART TX
    process(clk)
    begin
        if rising_edge(clk) then
            tx_start <= en;
        end if;
    end process;

    process(clk)
    begin
        --100.000.000 / 9600
        if rising_edge(clk) then
            if cnt = 10416  then
                baud_en <= '1';
                cnt <= (others => '0');
            else
                baud_en <= '0';
                cnt <= cnt + 1;
            end if;
        end if;
     end process;
```

```
process(clk)
begin
    if rising_edge(clk) then
        if tx_start = '1' then
            tx_en <= '1';
        elsif baud_en = '1' and tx_digit_cnt = 3 then
            tx_en <= '0';
        end if;
    end if;
end process;

process(clk)
begin
    if rising_edge(clk) then
        tx_rdy1 <= tx_rdy;
        if tx_start = '1' then
            tx_digit_Cnt <= (others => '0');
        elsif tx_rdy = '1' and tx_rdy1 = '0' then
            tx_digit_cnt <= tx_digit_cnt + 1;
        end if;
    end if;
end process;

process(clk)
begin
    if rising_edge(clk) then
        if tx_start = '1' then
            tx_reg <= transmitionData;
        end if;
   end if;
end process;

with tx_digit_cnt select
    tx_digit <= tx_reg(23 downto 18) when "00",
                tx_reg(17 downto 12) when "01",
                tx_reg(11 downto 6) when "10",
                tx_reg(5 downto 0) when "11",
                (others => 'X') when others;

with tx_digit select
    tx_data <= x"30" when "000000", -- '0'
               x"31" when "000001", -- '1'
               x"32" when "000010", -- '2'
               x"33" when "000011", -- '3'
               x"34" when "000100", -- '4'
               x"35" when "000101", -- '5'
               x"36" when "000110", -- '6'
               x"37" when "000111", -- '7'
```

```
                x"38" when "001000", -- '8'
                x"39" when "001001", -- '9'
                x"41" when "001010", -- 'A'
                x"42" when "001011", -- 'B'
                x"43" when "001100", -- 'C'
                x"44" when "001101", -- 'D'
                x"45" when "001110", -- 'E'
                x"46" when "001111", -- 'F'
                x"47" when "010000", -- 'G'
                x"48" when "010001", -- 'H'
                x"49" when "010010", -- 'I'
                x"4A" when "010011", -- 'J'
                x"4B" when "010100", -- 'K'
                x"4C" when "010101", -- 'L'
                x"4D" when "010110", -- 'M'
                x"4E" when "010111", -- 'N'
                x"4F" when "011000", -- 'O'
                x"50" when "011001", -- 'P'
                x"51" when "011010", -- 'Q'
                x"52" when "011011", -- 'R'
                x"53" when "011100", -- 'S'
                x"54" when "011101", -- 'T'
                x"55" when "011110", -- 'U'
                x"56" when "011111", -- 'V'
                x"57" when "100000", -- 'W'
                x"58" when "100001", -- 'X'
                x"59" when "100010", -- 'Y'
                x"5A" when "100011", -- 'Z'
                (others => 'X') when others;


transmit: PmodUSBUART_tx port map(clk, '0', baud_en, tx_en, tx_data, tx



 --UART RX

receive: PmodUSBUART_rx port map(clk, '0', baud_en_x16, rx, rx_data, rx
process(clk)
begin
    if rising_edge(clk) then
        --100.000.000 / (16 * 9600)
        if cnt_x16 = 651 then
            baud_en_x16 <= '1';
            cnt_x16 <= (others => '0');
        else
            baud_en_x16 <= '0';
            cnt_x16 <= cnt_x16 + 1;
```

```
        end if;
    end if;
end process;

with rx_data select
    rx_digit <= "000000" when x"30", -- '0'
                "000001" when x"31", -- '1'
                "000010" when x"32", -- '2'
                "000011" when x"33", -- '3'
                "000100" when x"34", -- '4'
                "000101" when x"35", -- '5'
                "000110" when x"36", -- '6'
                "000111" when x"37", -- '7'
                "001000" when x"38", -- '8'
                "001001" when x"39", -- '9'
                "001010" when x"41", -- 'A'
                "001011" when x"42", -- 'B'
                "001100" when x"43", -- 'C'
                "001101" when x"44", -- 'D'
                "001110" when x"45", -- 'E'
                "001111" when x"46", -- 'F'
                "010000" when x"47", -- 'G'
                "010001" when x"48", -- 'H'
                "010010" when x"49", -- 'I'
                "010011" when x"4A", -- 'J'
                "010100" when x"4B", -- 'K'
                "010101" when x"4C", -- 'L'
                "010110" when x"4D", -- 'M'
                "010111" when x"4E", -- 'N'
                "011000" when x"4F", -- 'O'
                "011001" when x"50", -- 'P'
                "011010" when x"51", -- 'Q'
                "011011" when x"52", -- 'R'
                "011100" when x"53", -- 'S'
                "011101" when x"54", -- 'T'
                "011110" when x"55", -- 'U'
                "011111" when x"56", -- 'V'
                "100000" when x"57", -- 'W'
                "100001" when x"58", -- 'X'
                "100010" when x"59", -- 'Y'
                "100011" when x"5A", -- 'Z'
                (others => 'X') when others;

transmitionData <= B"010110_010110_011001_000001" when sw(1) = '1' else

process(clk)
begin
    if rising_edge(clk) then
```

```
            rx_rdy1 <= rx_rdy;
            if rx_rdy = '1' and rx_rdy1 = '0' then
               case rx_digit_cnt is
                    when "00" => rx_reg(23 downto 18) <= rx_digit;
                    when "01" => rx_reg(17 downto 12) <= rx_digit;
                    when "10" => rx_reg(11 downto 6)  <= rx_digit;
                    when "11" => rx_reg(5 downto 0)   <= rx_digit;
                    when others => rx_reg(5 downto 0) <= (others => 'X');
                end case;

                      rx_digit_cnt <= rx_digit_cnt + 1;
            end if;
        end if;
    end process;
end Behavioral;
```

Listing 3: Cod sursă PMod KYPD

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


entity PmodKYPD is
    Port (
        clk : in  STD_LOGIC;
        JA : inout  STD_LOGIC_VECTOR (7 downto 0);
        an : out  STD_LOGIC_VECTOR (7 downto 0);
        cat : out  STD_LOGIC_VECTOR (6 downto 0)
    );
end PmodKYPD;

architecture Behavioral of PmodKYPD is

    component PmodKYPD_Decoder is
        Port (
            clk : in  STD_LOGIC;
            Row : in  STD_LOGIC_VECTOR (3 downto 0);
            Col : out  STD_LOGIC_VECTOR (3 downto 0);
            DecodeOut : out  STD_LOGIC_VECTOR (3 downto 0)
        );
    end component;

    component PmodKYPD_display is
        Port (
            DispVal : in  STD_LOGIC_VECTOR (3 downto 0);
            an : out STD_LOGIC_VECTOR (7 downto 0);
            segOut : out  STD_LOGIC_VECTOR (6 downto 0)
        );
    end component;
```

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

begin

    decoder: PmodKYPD_Decoder
        port map (
            clk => clk,
            Row => JA(7 downto 4),
            Col => JA(3 downto 0),
            DecodeOut => Decode
        );


    display: PmodKYPD_display
        port map (
            DispVal => Decode,
            an => an,
            segOut => cat
        );


end Behavioral;
```

Listing 4: Cod sursă FSM

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

entity Logic_FSM is
Port ( clk: in std_logic;
       reset: in std_logic;
       start_btn: in std_logic;
       input_done: in std_logic;
       valid: in std_logic;
       sequence_done: in std_logic;
       ok: out std_logic
       );
end Logic_FSM;

architecture Behavioral of Logic_FSM is

type state_type is (IDLE, SHOW, INPUT, VALIDATE, NEXT_LEVEL);

signal current_state, next_state: state_type;
signal current_level: INTEGER range 0 to 1;
```

```
begin

process(clk, reset)
begin
    if reset = '1' then
        current_state <= IDLE;
        current_level <= 0;
    elsif rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;

process(current_state, start_btn, input_done, valid, sequence_done)
begin
    case current_state is
        when IDLE =>
            if start_btn = '1' then
                next_state <= SHOW;
            end if;

        when SHOW =>
            if sequence_done = '1' then
                next_state <= INPUT;
            end if;

        when INPUT =>
            if input_done = '1' then
                next_state <= VALIDATE;
            end if;

        when VALIDATE =>
            if valid = '1' then
                next_state <= NEXT_LEVEL;
            end if;

        when NEXT_LEVEL =>
            if current_level < 1 then
                next_state <= SHOW;
            else
                next_state <= IDLE;
            end if;

        when others =>
            next_state <= IDLE;

    end case;
end process;
```

```
process(current_state)
begin
    ok <= '0';

    case current_state is
        when IDLE =>
            ok <= '0';

        when SHOW =>
            ok <= '0';

        when INPUT =>
            ok <= '0';

        when VALIDATE =>
            if valid = '1' then
                ok <= '1';
            else
                ok <= '0';
            end if;

        when NEXT_LEVEL =>
            current_level <= current_level + 1;

        when others => null;

    end case;


end process;



end Behavioral;
```

Listing 5: Cod sursă MPG

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

entity MPG is
    Port ( btn : in std_logic;
           clk : in std_logic;
           enable : out std_logic);
end MPG;

architecture Behavioral of MPG is
```

```
signal count:std_logic_vector(15 downto 0):=(others=>'0');
signal q1:std_logic:='0';
signal q2:std_logic:='0';
signal q3:std_logic:='0';
signal en:std_logic:='0';
begin

process(clk)
begin
if (rising_edge(clk)) then
    count <= count + 1;
end if;
end process;

process(clk)
begin
if (en='1') then
    if (rising_edge(clk)) then
        q1 <= btn;
    end if;
end if;
end process;

process(clk)
begin
if rising_edge(clk) then
    q2 <= q1;
end if;
end process;

process(clk)
begin
if rising_edge(clk) then
    q3 <= q2;
end if;
end process;

en <= '1' when count = x"FFFF" else '0';
enable <= q2 and not(q3);

end Behavioral;
```

Listing 6: Cod sursă Clk Divider

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

```
entity ClkDivider_1sec is
    Port ( clk      : in  STD_LOGIC;
           rst      : in  STD_LOGIC;
           clk_div  : out STD_LOGIC);
end ClkDivider_1sec;

architecture Behavioral of ClkDivider_1sec is
    constant constantNumber : integer := 50000000; --1 sec
    signal count            : integer range 0 to constantNumber-1 := 0;
    signal clk_div_reg      : STD_LOGIC := '0'; -- internal signal to hold
begin

    process(clk, rst)
    begin
        if rst = '1' then
            count <= 0;
        elsif rising_edge(clk) then
            if count = constantNumber - 1 then
                count <= 0;
            else
                count <= count + 1;
            end if;
        end if;
    end process;

    process(clk, rst)
    begin
        if rst = '1' then
            clk_div_reg <= '0';  -- Reset the internal signal when rst is
        elsif rising_edge(clk) then
            if count = constantNumber - 1 then
                clk_div_reg <= not clk_div_reg;  -- Toggle the internal sig
            end if;
        end if;
    end process;

    clk_div <= clk_div_reg;  -- Assign the internal signal to the output p

end Behavioral;
```

Listing 7: Memory Comparator

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

entity memory_comparator is
Port (
        clk: in std_logic;
```

```
        reset: in std_logic;
        en_btn: in std_logic;                          --up button (pe care ti
        btn1: in std_logic;                            --left button (validez
        data_in_kypd: in std_logic_vector(3 downto 0);
        valid: out std_logic;                          --semnal care valideaza
        score: out std_logic_vector(3 downto 0);       --semnal care calculeaz
        seq_done: out std_logic;                       --semnal care arata ca
        input_done: out std_logic                      --semnal care arata ca
    );
end memory_comparator;

architecture Behavioral of memory_comparator is

type matrix is array(0 to 9) of std_logic_vector(3 downto 0);
signal mem: matrix := (x"0", x"1", x"2", x"3", x"4", x"5", x"6", x"7", x"8"

signal address: integer := -1;
signal RAM_data: std_logic_vector(3 downto 0) := (others => '0');
signal data_out_wrapper: std_logic_vector(3 downto 0) := (others => '0');

signal led_wrapper: std_logic := '0';
signal clk_div: std_logic := '0';

--bidirectional signal for JA
signal JA_signal: std_logic_vector(7 downto 0);

signal counter: integer range 0 to 10 := 0;
signal score_signal: integer range 0 to 15 := 0;
signal speed_signal: std_logic := '0';
signal an: std_logic_vector(7 downto 0);
signal cat: std_logic_vector(6 downto 0);


component Wrapper is
    Port ( clk      : in  std_logic;
           rst      : in  std_logic;
           en_btn   : in  std_logic;
           speed    : in  std_logic;
           cat      : out std_logic_vector(6 downto 0);
           an       : out std_logic_vector(7 downto 0);
           led      : out std_logic;            --se aprinde dupa ce se afise
           clk_out  : out std_logic
           );
end component;

component PmodKYPD is
    Port (
        clk : in  STD_LOGIC;
```

```
        JA : inout  STD_LOGIC_VECTOR (7 downto 0);
        an : out  STD_LOGIC_VECTOR (7 downto 0);
        cat : out  STD_LOGIC_VECTOR (6 downto 0)
    );
end component;

begin

    Wrapper_inst: Wrapper
        Port map(
            clk     => clk,
            rst     => reset,
            en_btn  => en_btn,
            speed   => speed_signal,
            cat     => cat,
            an      => an,
            led     => led_wrapper,
            clk_out => clk_div
        );

    PmodKYPD_inst: PmodKYPD
        Port map(
            clk => clk,
            JA => JA_signal,
            an => an,
            cat => cat
        );

    JA_signal(7 downto 4) <= data_in_kypd;

    scriere: process(clk_div)
    begin
        if reset = '1' then
            address <= 0;
        elsif rising_edge(clk_div) and en_btn = '1' then
            address <= address + 1;
            mem(address) <= data_out_wrapper;
        end if;
    end process;

--ram_data <= mem(address) when led_wrapper = '1';     --dupa ce se afiseaz

    seq_done <= led_wrapper;

    comparator: process(btn1, reset)
    begin
        if reset = '1' then
            input_done <= '0';
```

```
            counter <= 0;
            score_signal <= 0;
        elsif rising_edge(btn1) then
            if counter < 10 then
                if mem(counter) = JA_signal(3 downto 0) then
                    score_signal <= score_signal + 1;
                end if;
                counter <= counter + 1;
                if counter = 10 then
                    input_done <= '1';
                end if;
            end if;
            if score_signal = 10 then
                valid <= '1';
            end if;
        end if;
    end process;

score <= std_logic_vector(to_unsigned(score_signal, 4));

end Behavioral;
```