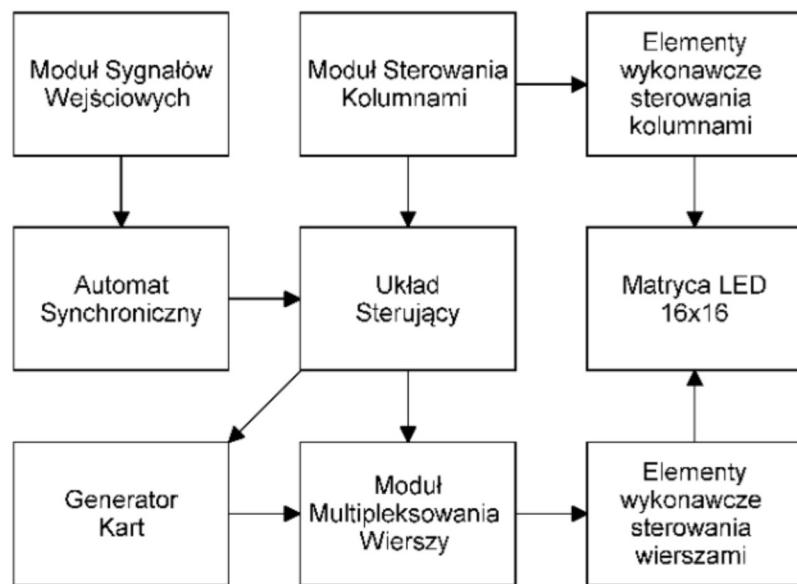
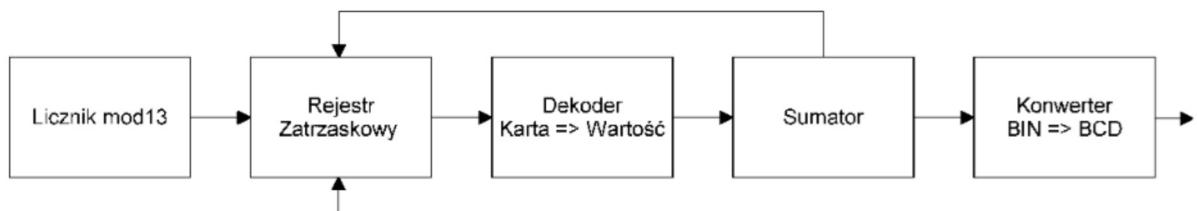


2. Schemat blokowy układu

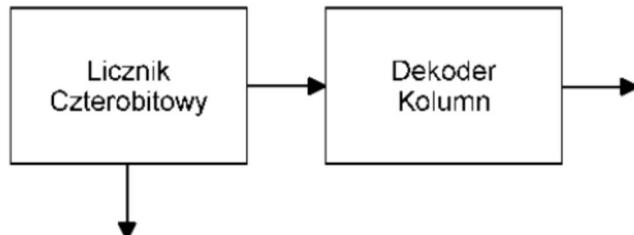
a) Ogólny schemat blokowy



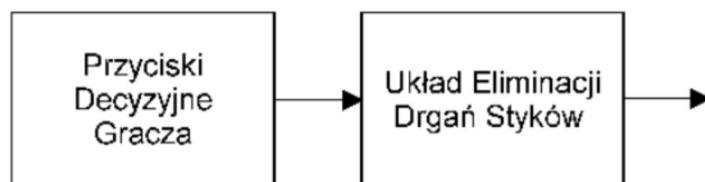
b) Moduł generatora kart



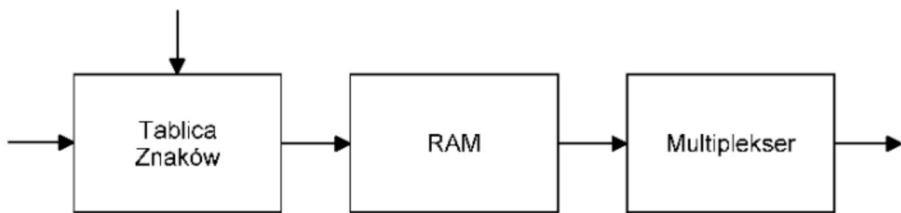
c) Moduł kolumn



d) Moduł interfejsu wejściowego



e) Moduł multipleksowania wierszy



Elementy wykonawcze sterowania kolumnami i wierszami wraz z wyświetlaczem matrycowym są częścią projektu która nie mogła być zaimplementowana w układzie logiki programowalnej. Sterowanie wierszy i kolumn wymaga stosunkowo dużego prądu w porównaniu z maksymalnymi wartościami prądów poszczególnych pinów układu FPGA. Podobnie wyświetlacz matrycowy musiał zostać wykorzystany poza układem ponieważ analiza stanów logicznych na pinach wyjściowych była by zbyt uciążliwa i nie oddawała w pełni przyjemności z gry. Pozostałe elementy mimo iż mogły być zbudowane na zewnątrz układu zostały zaimplementowane wewnętrznie ponieważ zajmuje to znacznie mniej miejsca, pozwala na to struktura oraz było to głównym założeniem przedmiotu.

3. Sposób rozwiązania problemu

Jako jednostkę sterującą wykorzystano układ FPGA firmy Altera: Cyclone II. Wybór został podjęty z powodu wystarczająco dużej ilości logiki wewnętrznej układu jak i mniejszym zaawansowaniem w porównaniu z popularną aktualnie IV generacją tego układu.

Zadanie projektowe (część programowa) zostało podzielone na konkretne etapy, dzięki czemu można było po kolei rozbudowywać działający układ. Plan pracy został podzielony na następujące etapy:

- realizacja modułu multipleksowania kolumn
- realizacja układu wyświetlającego symbol na pojedynczej matrycy 8x8
- rozbudowanie układu o dodatkowe symbole
- rozbudowanie układu o dodatkowe wyświetlacze, realizacja pełnej matrycy 16x16
- realizacja jednostki generującej karty
- realizacja automatu synchronicznego
- dołączenie do układu przycisków umożliwiających sterowanie grą

Opis konkretnych bloków cyfrowych układu:

a) Moduł kolumn

- Licznik czterobitowy – Układ licznika czterobitowego którego zadaniem jest wybór aktualnej kolumny wyświetlanej na matrycy
- Dekoder kolumn – Układ demultipleksera który steruje określoną kolumną przy wyłączeniach pozostałych

b) Moduł wierszy

- Tablica znaków – Jest to pamięć ROM w której są zapisane wszystkie znaki które mogą być wyświetlane na matrycy LED. Na podstawie przychodzących danych określa który znak ma być przekazywany kolumnie po kolumnie do poszczególnych pamięci RAM
- RAM – pamięć której zadaniem jest przechowywanie aktualnie wyświetlonego symbolu. W projekcie są 4 pamięci tego typu w celu podzielenia matrycy na 4 mniejsze wyświetlacze.
- Multiplekser – Układ który scalą dwa wyświetlacze 8x8 do jednego 8x16 który na podstawie globalnego adresu decyduje z której pamięci RAM odczytywać symbol.

c) Moduł interfejsu wejściowego

- Przyciski – W projekcie zostały zastosowane przełączniki micro-switch
- Układ eliminacji drgań styków – Zabezpieczając się przed niepożądanymi drganiami styków układ został wyposażony w debouncer który wyłumia stan nieustalony przycisków.

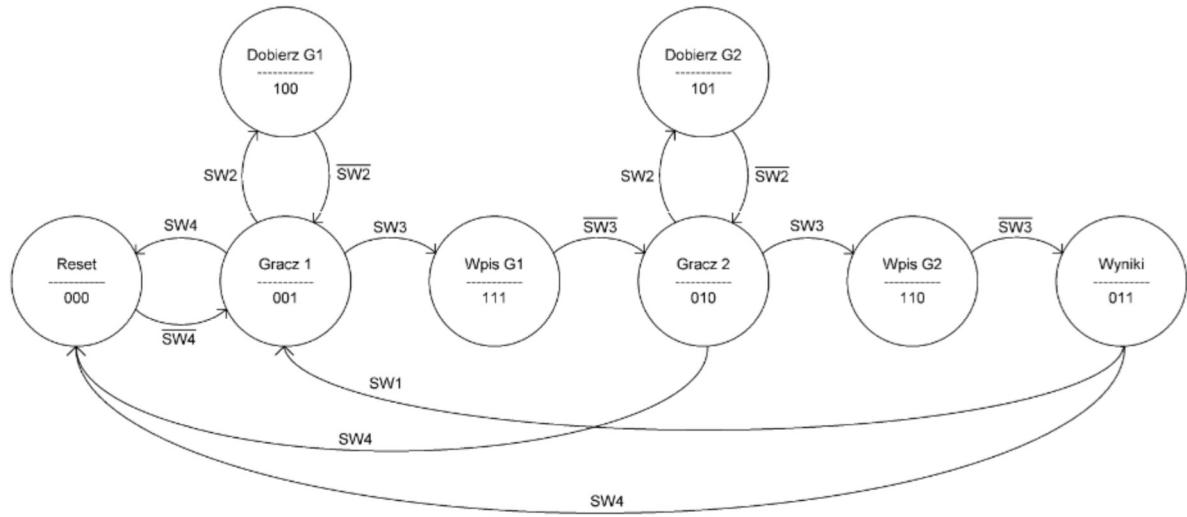
d) Moduł generatora kart

- Licznik mod13 - W celu generacji liczb pseudolosowych autor zdecydował się na wykorzystanie zwykłego licznika modulo 13 taktowanego szybkim zegarem. Mała dynamika człowieka oraz szybkość taktowania dają wystarczającą pseudolosowość.
- Rejestr zatrzaskowy – Układ rejestru równoległo-równoległe którego przepisanie jest równoznaczne z dobraniem karty ponieważ po jego zadziałaniu zostają wprowadzone na sumator nowe dane o poprzedniej wartości sumy i nowej karty.
- Dekoder Karta=> Wartość – Konwerter kodu który wylosowaną wartość liczbową „tłumaczy” na wartości poszczególnych kart.
- Sumator – Podstawowy model układu sumującego liczby
- Konwerter BIN=>BCD – Układ który konwertuje liczby zrozumiałe dla układu na system dziesiętny łatwiej rozumiany dla człowieka.

e) Automat – Układ sterujący przebiegiem działania programu na podstawie sygnałów wejściowych pochodzących z przycisków i aktualnego stanu gry.

f) Układ sterujący – Odpowiedzialny jest za generację i odpowiednie sterowanie przepływem danych na podstawie aktualnego stanu automatu synchronicznego.

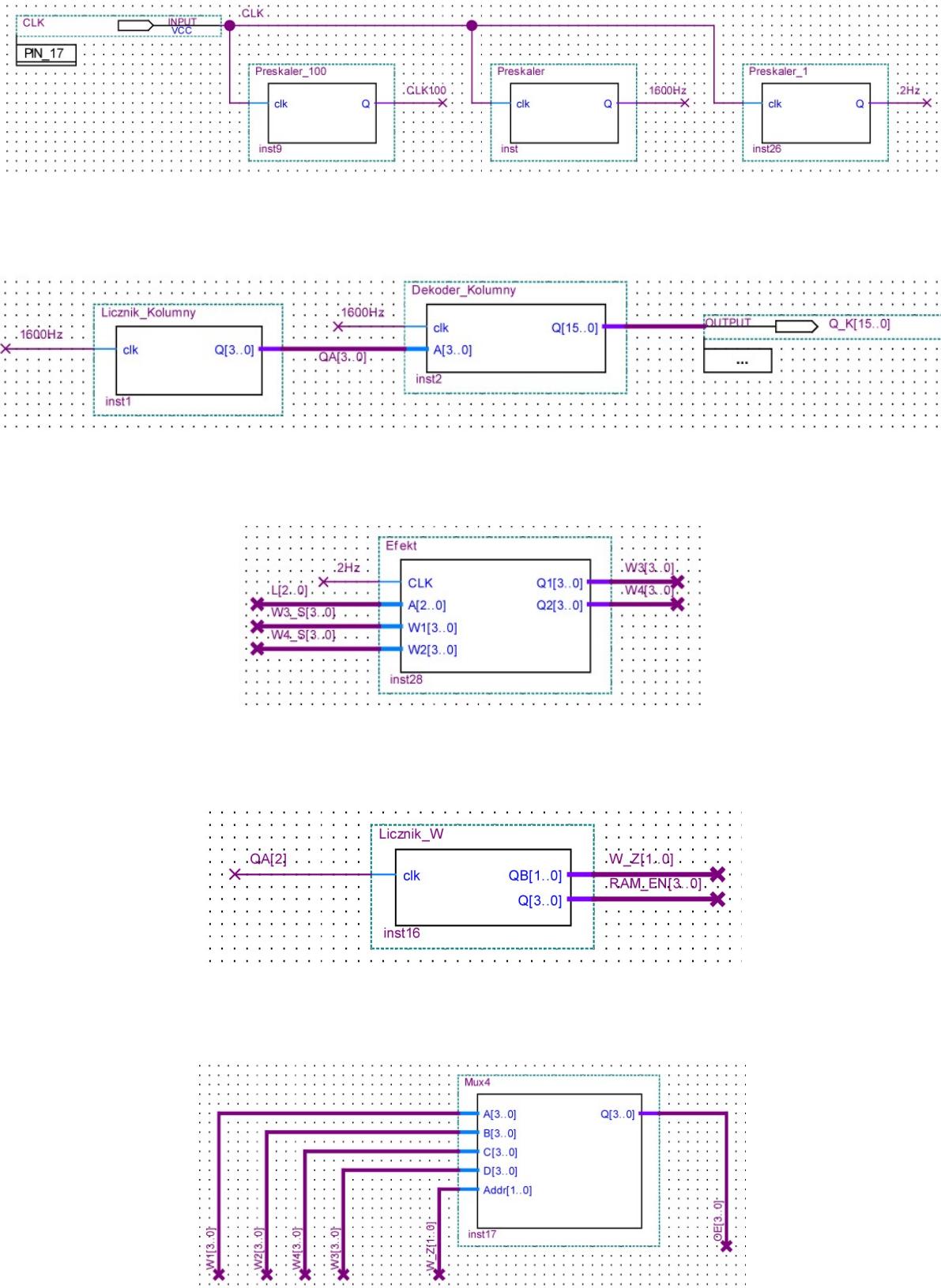
4. Graf przejść automatu synchronicznego

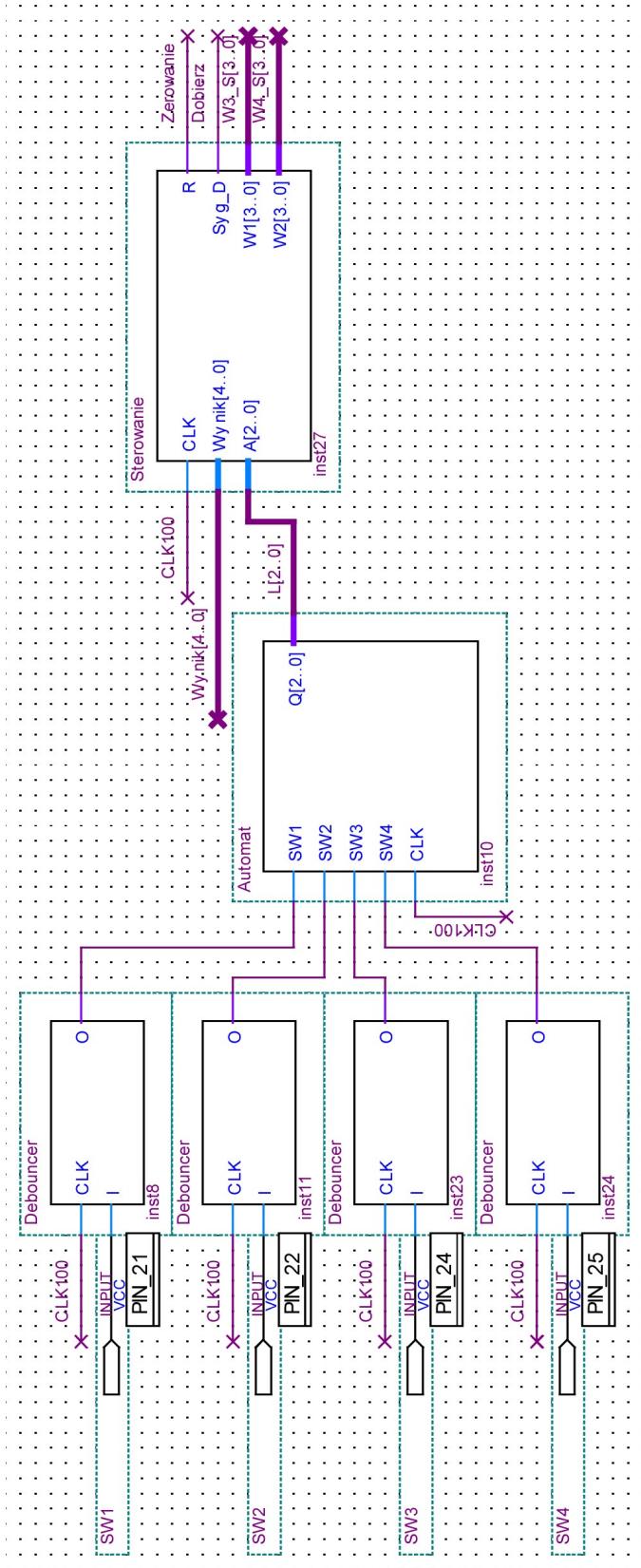


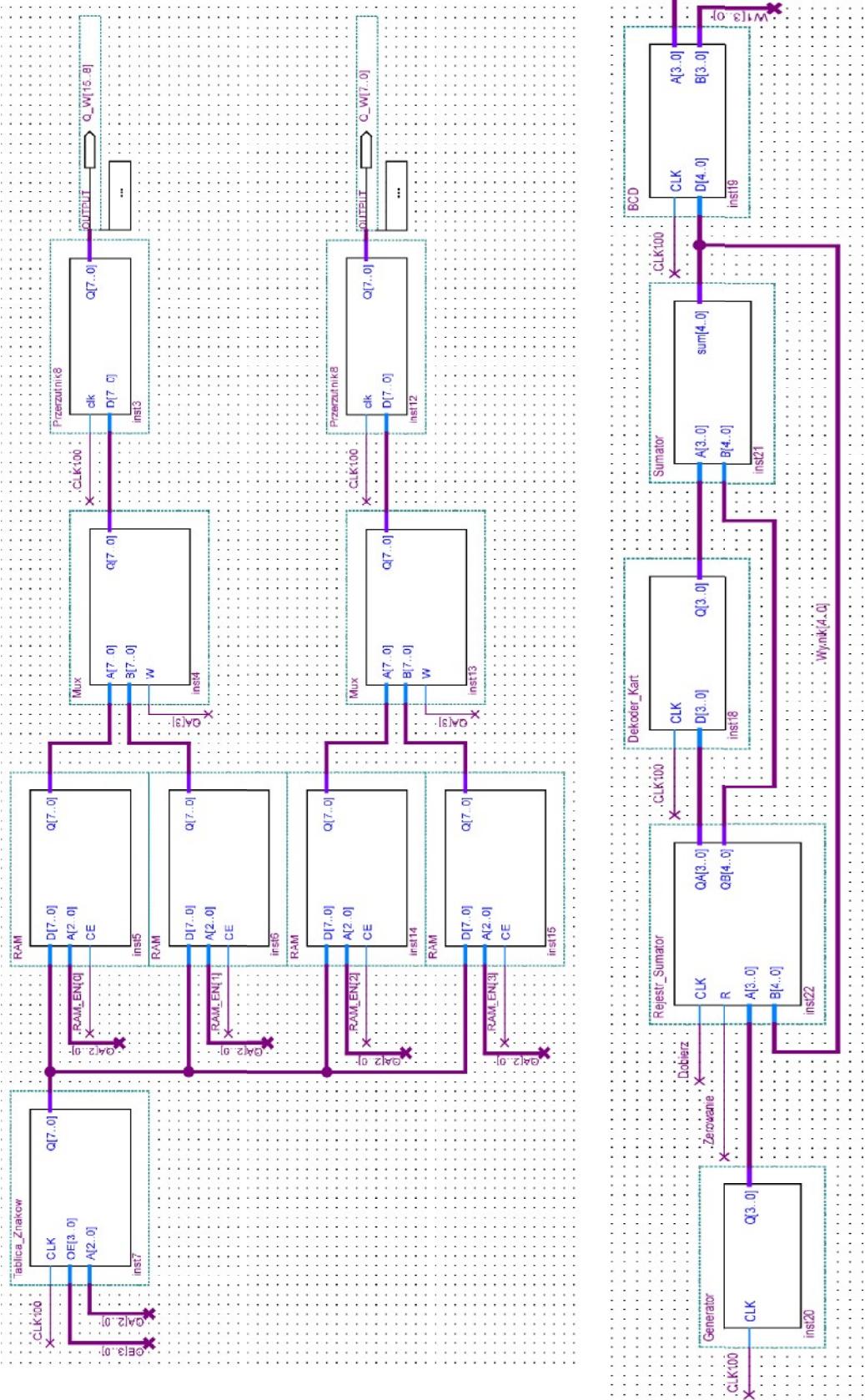
Układ opiera się tylko na jednym automacie synchronicznym którego graf jest przedstawiony powyżej. Sygnały sterujące pochodzą z przycisków na płytce. Uznaje się, że układ można zresetować w dowolnym momencie gry co jest jednak błędne ponieważ jak widać na grafie do stanu „000” możemy dojść tylko z trzech innych stanów. W rzeczywistości ten kruczek można zauważać tylko podczas naciśnięcia więcej niż jednego przycisku. Podczas normalnego użytkowania nie jesteśmy w stanie tego zauważać ponieważ stany dobierania i wpisu są realizowane za jednym wciśnięciem przycisku (zbocze narastające i opadające) co jest jak najbardziej naturalne. W układzie nie został zaimplementowany moduł zabezpieczający przed sklejeniem przycisku z którymkolwiek z stanów.

SW1	SW2	SW3	SW4
Start	Dobierz kartę	Stop	Reset

5. Schemat ideowy układu

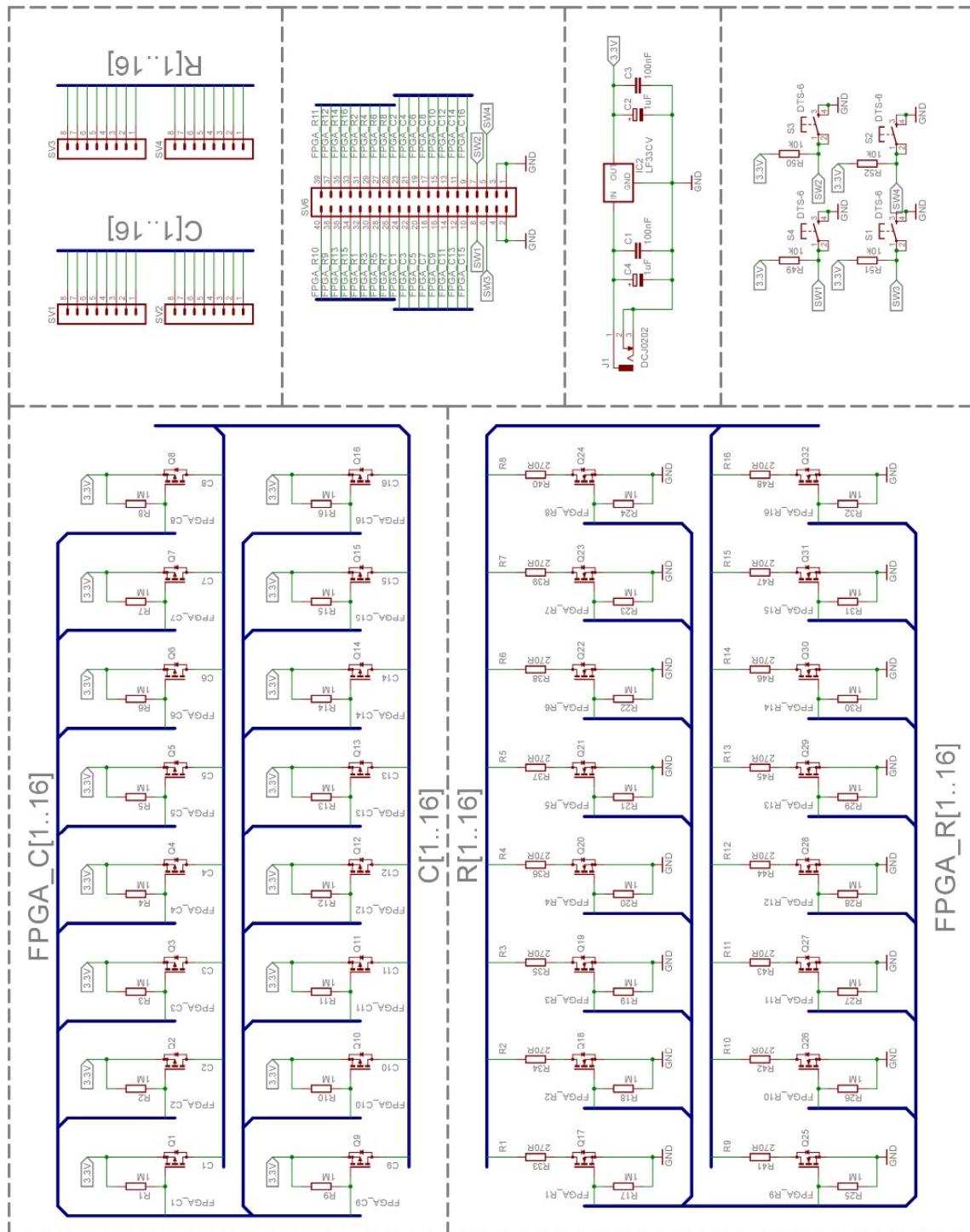




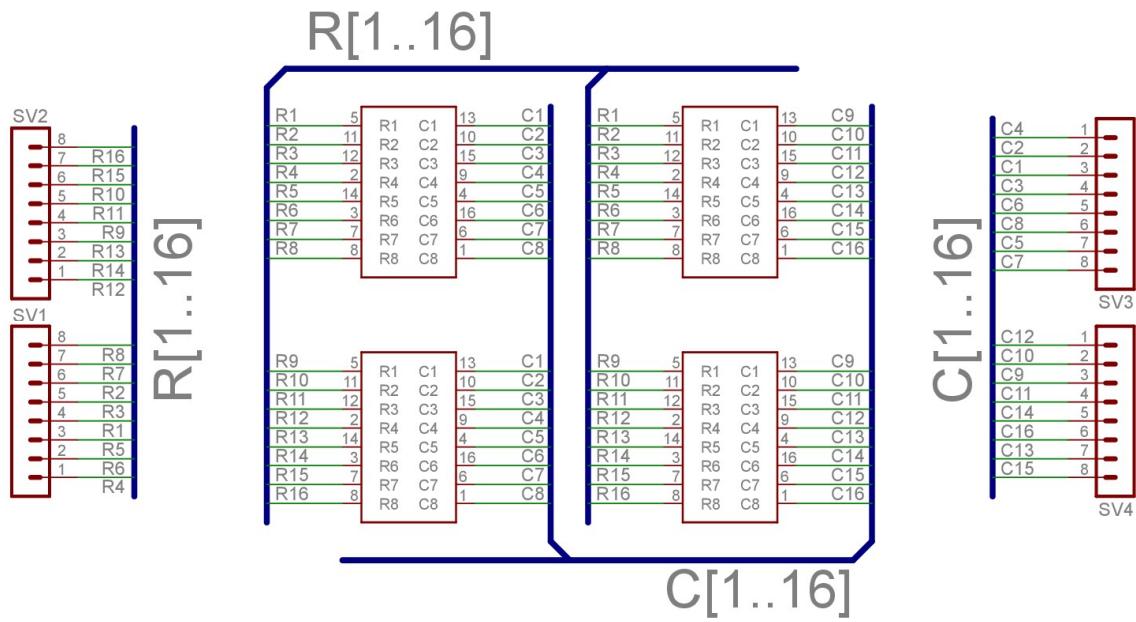


6. Projekt elektryczny modułu wyświetlającego

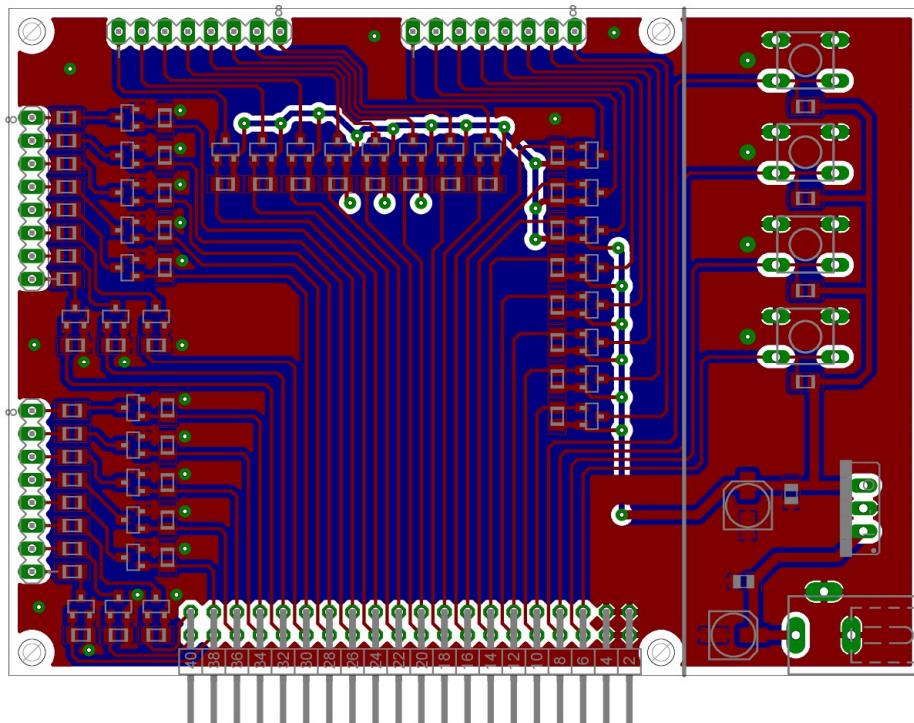
- Układ wykonawczy



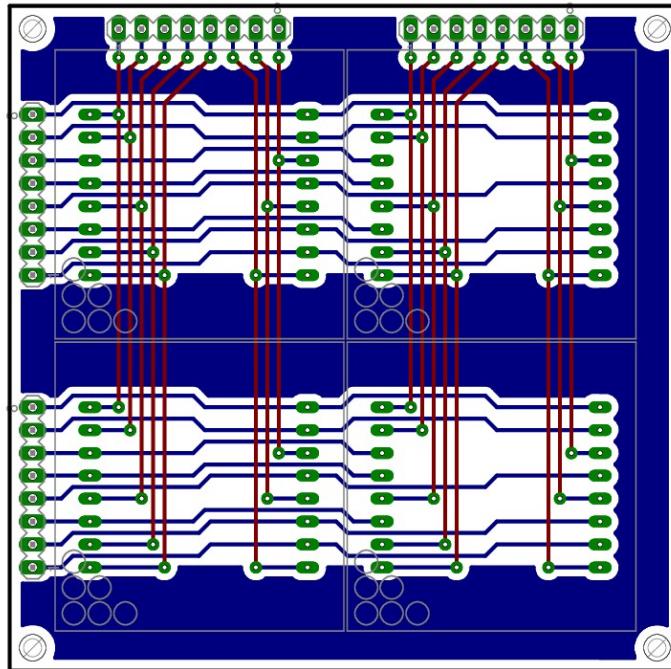
- Moduł wyświetlacza matrycowego 16x16



- Projekt PCB układu wykonawczego

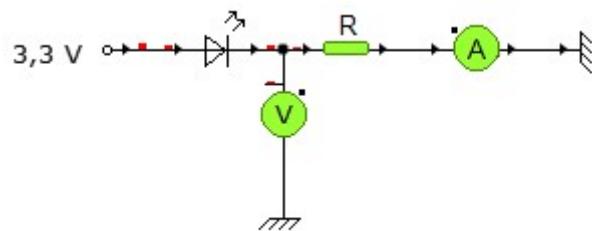


- Projekt PCB modułu wyświetlacza matrycowego 16x16



- Niezbędne obliczenia:

W celu określenia wartości progowych prądów należało ustalić minimalną wartość potrzebą do zadowalającego zaświecenia pojedynczą kropką na matrycy LED. W tym celu został zmontowany układ:



Dobierając różne wartości rezystora R określona została minimalna wartość prądu potrzebna do zadowalającej jasności świecenia.

$$I_{F_{min}} = 700\mu A, U_F = 1.8V$$

Idąc dalej, do otrzymania pełnej jasności świecenia należy dostarczyć 16 razy większy prąd z powodu multipleksowania kolumn.

$$I_W = 16 * 700\mu A = 11.2mA$$

Tranzystor kluczujący wiersze powinien zostać dobrany aby był w stanie wytrzymać prąd o wartości I_W . Na podstawie tej wartości można dobrać wartość rezystorów ustalających jasność świecenia matrycy.

$$R = \frac{3.3V - U_F}{I_W} = \frac{3.3V - 1.8V}{11.2mA} \cong 134\Omega$$

Obliczonej wartości nie ma w szeregu, więc została wykorzystany rezystor o wartości 100Ω . Tego typu zaokrąglenie spowodowało zmianę w minimalnej wartości prądu na wiersz.

$$I_{W_{min}} = \frac{1.5V}{100\Omega} = 15mA$$

Prąd płynący przez diodę przy braku multipleksowania:

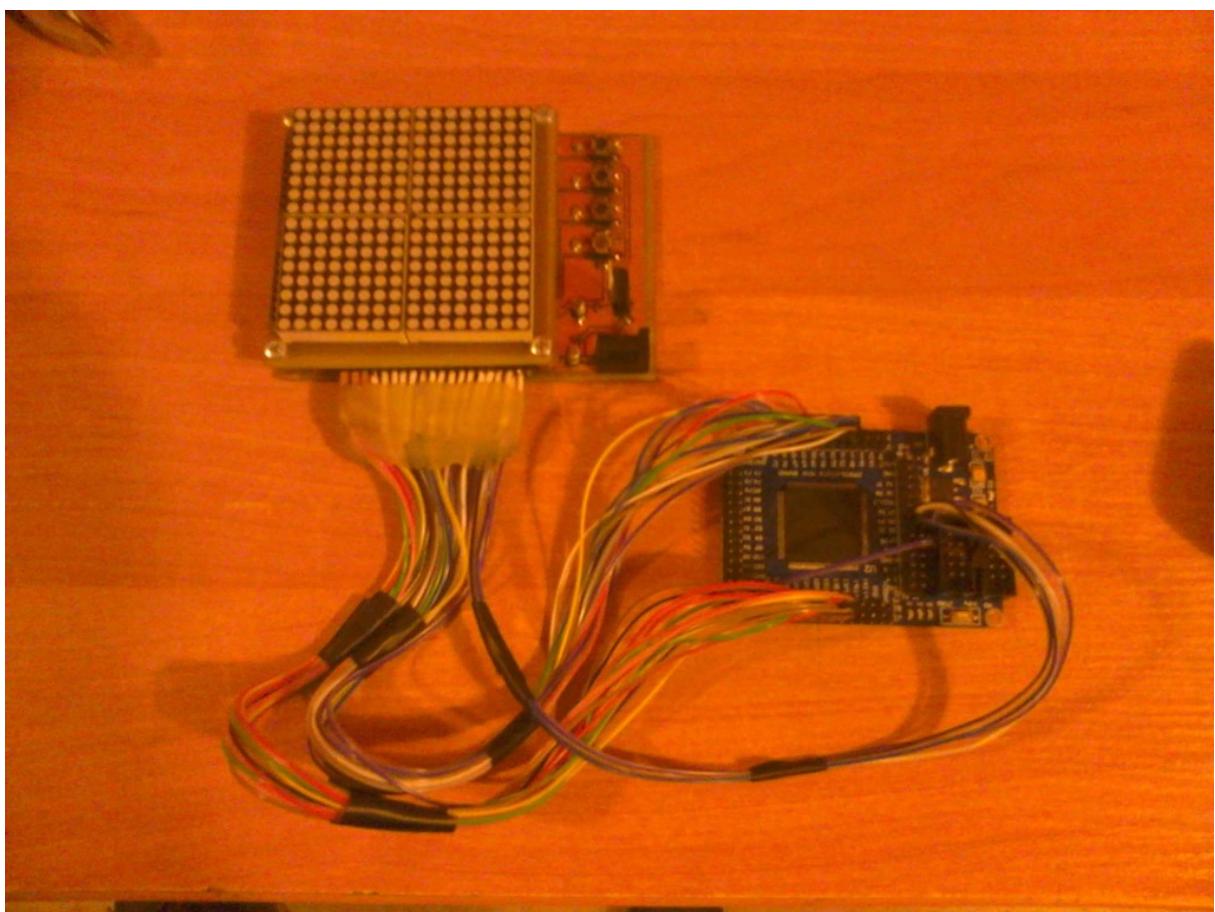
$$I_F = \frac{I_{W_{min}}}{16} = \frac{15mA}{16} = 937.5\mu A$$

Dzięki temu zaokrągleniu diody nie świecą z minimalną dopuszczalną jasnością.

Minimalna wartość prądu jaką powinien wytrzymać tranzystor kluczujący kolumny:

$$I_K = I_{W_{min}} * 16 = 15 * 16 = 240mA$$

- Realizacja praktyczna – zmontowany układ



7. Listing programu

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

entity Przerzutnik8 is
    port(      clk: in std_logic;
                D:  in std_logic_vector(7 downto 0);
                Q: out std_logic_vector(7 downto 0));
end Przerzutnik8;

architecture Behavioral of Przerzutnik8 is
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            Q(0) <= D(0);
            Q(1) <= D(1);
            Q(2) <= D(2);
            Q(3) <= D(3);
            Q(4) <= D(4);
            Q(5) <= D(5);
            Q(6) <= D(6);
            Q(7) <= D(7);
        end if;
    end process;
end Behavioral;
```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Preskaler is
    port(      clk: in std_logic;
                Q: buffer std_logic);
end Preskaler;

architecture Behavioral of Preskaler is
begin
    process(clk)
        variable Licznik : integer range 0 to 3_906;
    begin
        if (clk'event and clk = '0') then
            if Licznik > 0 then
                Licznik := Licznik - 1;
            else
                Licznik := 3_906;
                Q <= Q xor '1';
            end if;
        end if;
    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Licznik_Kolumny is
    port(      clk: in std_logic;
                Q: buffer std_logic_vector(3 downto 0) );
end Licznik_Kolumny;

architecture Behavioral of Licznik_Kolumny is
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            Q <= Q + 1;
        end if;
    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Dekoder_Kolumny is
    port(      clk: in std_logic;
                A: in std_logic_vector(3 downto 0);
                Q: buffer std_logic_vector(15 downto 0) );
end Dekoder_Kolumny;

architecture Behavioral of Dekoder_Kolumny is
begin
    process(A)
    begin
        --if clk'event and clk = '1' then
        case A is
            when "0000" => Q <= "1111111111111110";
            when "0001" => Q <= "1111111111111101";
            when "0010" => Q <= "11111111111111011";
            when "0011" => Q <= "111111111111110111";
            when "0100" => Q <= "11111111111101111";
            when "0101" => Q <= "1111111111011111";
            when "0110" => Q <= "11111111110111111";
            when "0111" => Q <= "1111111101111111";
            when "1000" => Q <= "1111111011111111";
            when "1001" => Q <= "11111110111111111";
            when "1010" => Q <= "1111101111111111";
            when "1011" => Q <= "1111011111111111";
            when "1100" => Q <= "1110111111111111";
            when "1101" => Q <= "1101111111111111";
            when "1110" => Q <= "1011111111111111";
            when "1111" => Q <= "0111111111111111";
            when others => Q <= "1111111111111110";
        end case;
        --end if;
    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Mux is
    port(
        A:    in std_logic_vector(7 downto 0);
        B:    in std_logic_vector(7 downto 0);
        W:    in std_logic;
        Q:    out std_logic_vector(7 downto 0));
end Mux;

architecture Behavioral of Mux is
begin
    process(W)
    begin
        if W = '1' then
            Q <= A;
        else
            Q <= B;
        end if;
    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity RAM is
    port(
        D: in std_logic_vector(7 downto 0);
        A: in std_logic_vector(2 downto 0);
        CE: in std_logic;
        Q: out std_logic_vector(7 downto 0));
end RAM;

architecture Behavioral of RAM is
begin
    process(A, CE, D)
    begin
        if CE = '1' then
            case A is
                when "000" => Q <= R0;
                when "001" => Q <= R1;
                when "010" => Q <= R2;
                when "011" => Q <= R3;
                when "100" => Q <= R4;
                when "101" => Q <= R5;
                when "110" => Q <= R6;
                when "111" => Q <= R7;
            end case;
        else
            case A is
                when "000" => R0 <= D;
                when "001" => R1 <= D;
                when "010" => R2 <= D;
                when "011" => R3 <= D;
                when "100" => R4 <= D;
                when "101" => R5 <= D;
                when "110" => R6 <= D;
                when "111" => R7 <= D;
            end case;
        end if;
    end process;
end Behavioral;

```

```

library ieee;
use ieee.std_logic_1164.all;

entity Tablica_Znakow is
  port ( CLK: in std_logic;
         OE : in std_logic_vector(3 downto 0);
         A : in std_logic_vector(2 downto 0);
         Q : out std_logic_vector(7 downto 0));
end entity Tablica_Znakow;

architecture Behavioral of Tablica_Znakow is
  signal Q0: std_logic_vector(7 downto 0);
  signal Q1: std_logic_vector(7 downto 0);
  signal Q2: std_logic_vector(7 downto 0);
  signal Q3: std_logic_vector(7 downto 0);
  signal Q4: std_logic_vector(7 downto 0);
  signal Q5: std_logic_vector(7 downto 0);
  signal Q6: std_logic_vector(7 downto 0);
  signal Q7: std_logic_vector(7 downto 0);
  signal Q8: std_logic_vector(7 downto 0);
  signal Q9: std_logic_vector(7 downto 0);
  signal Q10: std_logic_vector(7 downto 0);
  signal Q11: std_logic_vector(7 downto 0);
  signal Q12: std_logic_vector(7 downto 0);
  signal Q13: std_logic_vector(7 downto 0);
  signal Q14: std_logic_vector(7 downto 0);
  signal Q15: std_logic_vector(7 downto 0);
begin
  IC0: entity work.ROM_0 port map (A => A, Q => Q0);
  IC1: entity work.ROM_1 port map (A => A, Q => Q1);
  IC2: entity work.ROM_2 port map (A => A, Q => Q2);
  IC3: entity work.ROM_3 port map (A => A, Q => Q3);
  IC4: entity work.ROM_4 port map (A => A, Q => Q4);
  IC5: entity work.ROM_5 port map (A => A, Q => Q5);
  IC6: entity work.ROM_6 port map (A => A, Q => Q6);
  IC7: entity work.ROM_7 port map (A => A, Q => Q7);
  IC8: entity work.ROM_8 port map (A => A, Q => Q8);
  IC9: entity work.ROM_9 port map (A => A, Q => Q9);
  IC10: entity work.ROM_G port map (A => A, Q => Q10);

```

```

process(CLK, OE)
begin
    if clk'event and clk = '1' then
        case OE is
            when "0000" => Q <= Q0;
            when "0001" => Q <= Q1;
            when "0010" => Q <= Q2;
            when "0011" => Q <= Q3;
            when "0100" => Q <= Q4;
            when "0101" => Q <= Q5;
            when "0110" => Q <= Q6;
            when "0111" => Q <= Q7;
            when "1000" => Q <= Q8;
            when "1001" => Q <= Q9;
            when "1010" => Q <= Q10;
            when others => Q <= Q0;
        end case;
    end if;
end process;

end architecture Behavioral;

=====
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Preskaler_100 is
    port(      clk: in std_logic;
               Q: buffer std_logic);
end Preskaler_100;

architecture Behavioral of Preskaler_100 is
begin
    process(clk)
        variable Licznik : integer range 0 to 15;
    begin
        if (clk'event and clk = '0') then
            if Licznik > 0 then
                Licznik := Licznik - 1;
            else
                Licznik := 15;
                Q <= Q xor '1';
            end if;
        end if;
    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Automat is
    port(
        -- Reset
        SW1: in std_logic;
        -- Start / Nowa gra
        SW2: in std_logic;
        -- Stop dobierania
        SW3: in std_logic;
        -- Dobierz
        SW4: in std_logic;
        CLK: in std_logic;
        Q: out std_logic_vector(2 downto 0));
end Automat;

architecture Behavioral of Automat is
signal Stan: std_logic_vector(2 downto 0):= "011";
begin
    process(CLK)
    begin
        if CLK'event and CLK = '1' then
-- =====
-- STAN RESET
-- =====
            if Stan = "000" then
                -- Guzik reset
                if SW1 = '1' then
                    Stan <= "000";
                else
                    Stan <= "001";
                end if;
            end if;
-- =====
-- STAN GRACZ 1
-- =====
            if Stan = "001" then
                -- Guzik reset
                if SW1 = '1' then
                    Stan <= "000";
                end if;

                -- Guzik stop dobierania
                if SW3 = '1' then
                    Stan <= "111";
                end if;

                -- Guzik dobierz karte
                if SW4 = '1' then

```

```

                Stan <= "100";
            end if;
        end if;
-- =====
--          STAN DOBIERZ G1
-- =====

        if Stan = "100" then
            if SW4 = '0' then
                Stan <= "001";
            end if;
        end if;
-- =====
--          STAN WPIS 1
-- =====

        if Stan = "111" then
            if SW3 = '0' then
                Stan <= "010";
            end if;
        end if;
-- =====
--          STAN GRACZ 2
-- =====

        if Stan = "010" then
            -- Guzik reset
            if SW1 = '1' then
                Stan <= "000";
            end if;

            -- Guzik stop dobierania
            if SW3 = '1' then
                Stan <= "110";
            end if;

            -- Guzik dobierz karte
            if SW4 = '1' then
                Stan <= "101";
            end if;
        end if;
-- =====
--          STAN DOBIERZ G2
-- =====

        if Stan = "101" then
            if SW4 = '0' then
                Stan <= "010";
            end if;
        end if;
-- =====
--          STAN WPIS 2
-- =====

        if Stan = "110" then
            if SW4 = '0' then

```

```

                Stan <= "011";
            end if;
        end if;
-- =====
--          STAN WYNIKI
-- =====
        if Stan = "011" then
            -- Guzik reset
            if SW1 = '1' then
                Stan <= "000";
            end if;

            -- Guzik start gry
            if SW2 = '1' then
                Stan <= "001";
            end if;
        end if;
    end process;

    Q <= Stan;
end Behavioral;

=====
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Licznik_W is
    port(      clk: in std_logic;
               QB: buffer std_logic_vector(1 downto 0);
               Q: buffer std_logic_vector(3 downto 0) );
end Licznik_W;

architecture Behavioral of Licznik_W is
begin
    process(clk)
    begin
        if clk'event and clk = '0' then
            case Q is
                when "1110" => Q <= "1101"; QB <= "01";
                when "1101" => Q <= "1011"; QB <= "10";
                when "1011" => Q <= "0111"; QB <= "11";
                when "0111" => Q <= "1110"; QB <= "00";
                when others => Q <= "1110"; QB <= "00";
            end case;
        end if;
    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Mux4 is
    port(
        A:    in std_logic_vector(3 downto 0);
        B:    in std_logic_vector(3 downto 0);
        C:    in std_logic_vector(3 downto 0);
        D:    in std_logic_vector(3 downto 0);
        Addr: in std_logic_vector(1 downto 0);
        Q:    out std_logic_vector(3 downto 0));
end Mux4;

architecture Behavioral of Mux4 is
begin
    process(Addr)
    begin
        case Addr is
            when "00" => Q <= A;
            when "01" => Q <= B;
            when "10" => Q <= C;
            when "11" => Q <= D;
        end case;
    end process;
end Behavioral;
=====
```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Dekoder_Kart is
    port(      CLK: in std_logic;
                D: in std_logic_vector(3 downto 0);
                Q: buffer std_logic_vector(3 downto 0) );
end Dekoder_Kart;

architecture Behavioral of Dekoder_Kart is
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            case D is
                when "0000" => Q <="0010";
                when "0001" => Q <="0011";
                when "0010" => Q <="0100";
                when "0011" => Q <="0101";
                when "0100" => Q <="0110";
                when "0101" => Q <="0111";
                when "0110" => Q <="1000";
                when "0111" => Q <="1001";
                when "1000" => Q <="1010";
                when "1001" => Q <="0010";
                when "1010" => Q <="0011";
                when "1011" => Q <="0100";
                when "1100" => Q <="1011";
                when others => Q <="0000";
            end case;
        end if;
    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;
entity BCD is
    port(    CLK: in std_logic;
              D: in std_logic_vector(4 downto 0);
              A: buffer std_logic_vector(3 downto 0);
              B: buffer std_logic_vector(3 downto 0));
end BCD;
architecture Behavioral of BCD is
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            case D is
                when "00000" => A <= "0000"; B <= "0000";
                when "00001" => A <= "0000"; B <= "0001";
                when "00010" => A <= "0000"; B <= "0010";
                when "00011" => A <= "0000"; B <= "0011";
                when "00100" => A <= "0000"; B <= "0100";
                when "00101" => A <= "0000"; B <= "0101";
                when "00110" => A <= "0000"; B <= "0110";
                when "00111" => A <= "0000"; B <= "0111";
                when "01000" => A <= "0000"; B <= "1000";
                when "01001" => A <= "0000"; B <= "1001";
                when "01010" => A <= "0001"; B <= "0000";
                when "01011" => A <= "0001"; B <= "0001";
                when "01100" => A <= "0001"; B <= "0010";
                when "01101" => A <= "0001"; B <= "0011";
                when "01110" => A <= "0001"; B <= "0100";
                when "01111" => A <= "0001"; B <= "0101";
                when "10000" => A <= "0001"; B <= "0110";
                when "10001" => A <= "0001"; B <= "0111";
                when "10010" => A <= "0001"; B <= "1000";
                when "10011" => A <= "0001"; B <= "1001";
                when "10100" => A <= "0010"; B <= "0000";
                when "10101" => A <= "0010"; B <= "0001";
                when "10110" => A <= "0010"; B <= "0010";
                when "10111" => A <= "0010"; B <= "0011";
                when "11000" => A <= "0010"; B <= "0100";
                when "11001" => A <= "0010"; B <= "0101";
                when "11010" => A <= "0010"; B <= "0110";
                when "11011" => A <= "0010"; B <= "0111";
                when "11100" => A <= "0010"; B <= "1000";
                when "11101" => A <= "0010"; B <= "1001";
                when "11110" => A <= "0011"; B <= "0000";
                when "11111" => A <= "0011"; B <= "0001";
            end case;
        end if;
    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Generator is
    port(      CLK: in std_logic;
                Q: buffer std_logic_vector(3 downto 0) );
end Generator;

architecture Behavioral of Generator is
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if Q < "1101" then
                Q <= Q + 1;
            else
                Q <= "0000";
            end if;
        end if;
    end process;
end Behavioral;

```

```

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

entity Sumator is
    port(
        A: in unsigned(3 downto 0);
        B: in unsigned(4 downto 0);
        sum: out unsigned(4 downto 0));
end entity Sumator;

architecture Behavioral of Sumator is
signal temp : unsigned(4 downto 0);
begin
    temp <= ("0" & A) + B;
    sum      <= temp(4 downto 0);
end architecture Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Rejestr_Sumator is
    port(      CLK: in std_logic;
               R: in std_logic;
               A: in std_logic_vector(3 downto 0);
               B: in std_logic_vector(4 downto 0);
               QA: buffer std_logic_vector(3 downto 0);
               QB: buffer std_logic_vector(4 downto 0));
end Rejestr_Sumator;

architecture Behavioral of Rejestr_Sumator is
begin
    process(clk, R)
    begin

        if R = '1' then
            QA <= "1111";
            QB <= "00000";
        else
            if clk'event and clk = '1' then
                QA <= A;
                QB <= B;
            end if;
        end if;

    end process;
end Behavioral;

```

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Preskaler_1 is
    port(      clk: in std_logic;
                Q: buffer std_logic);
end Preskaler_1;

architecture Behavioral of Preskaler_1 is
begin
    process(clk)
        variable Licznik : integer range 0 to 24_999_999;
    begin
        if (clk'event and clk = '0') then
            if Licznik > 0 then
                Licznik := Licznik - 1;
            else
                Licznik := 24_999_999;
                Q <= Q xor '1';
            end if;
        end if;
    end process;
end Behavioral;

```

```

=====
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Sterowanie is
    port(      CLK: in std_logic;
                Wynik: in std_logic_vector(4 downto 0);
                A: in std_logic_vector(2 downto 0);
                R: out std_logic;
                Syg_D: buffer std_logic;
                W1: buffer std_logic_vector(3 downto 0);
                W2: buffer std_logic_vector(3 downto 0)
            );
end Sterowanie;

architecture Behavioral of Sterowanie is
signal Gracz1: std_logic_vector(4 downto 0) := "00000";
signal Gracz2: std_logic_vector(4 downto 0) := "00000";
begin
    process(A, CLK)
    begin
        if clk'event and clk = '1' then
            case A is

```

```

--                      STAN RESET
-- =====
when "000" =>
    R <= '1';
-- =====
--                      STAN GRACZ 1
-- =====
when "001" =>
    Syg_D <= '0';
    W1 <= "1010";
    W2 <= "0001";
    R <= '0';
    Gracz1 <= Wynik;
-- =====
--                      STAN DOBIERZ G1
-- =====
when "100" =>
    if Wynik < 21 then
        Syg_D <= '1';
    end if;
-- =====
--                      STAN WPIS 1
-- =====
when "111" =>
    if Gracz1 > 21 then
        Gracz1 <= "00000";
    end if;
    R <= '1';
-- =====
--                      STAN GRACZ 2
-- =====
when "010" =>
    Syg_D <= '0';
    W1 <= "1010";
    W2 <= "0010";
    R <= '0';
    Gracz2 <= Wynik;
-- =====
--                      STAN DOBIERZ G2
-- =====
when "101" =>
    if Wynik < 21 then
        Syg_D <= '1';
    end if;
-- =====
--                      STAN WPIS 2
-- =====
when "110" =>
    if Gracz2 > 21 then
        Gracz2 <= "00000";
    end if;

```

```

                    R <= '1';
-- =====
--          STAN WYNIKI
-- =====
        when "011" =>
            if Gracz1 > Gracz2 then
                W1 <= "1010";
                W2 <= "0001";
            end if;
            if Gracz1 < Gracz2 then
                W1 <= "1010";
                W2 <= "0010";
            end if;
            if Gracz1 = Gracz2 then
                W1 <= "1010";
                W2 <= "1010";
            end if;
        end case;
    end if;
end process;
end Behavioral;

=====
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Efekt is
    port(      CLK: in std_logic;
               A: in std_logic_vector(2 downto 0);
               W1: in std_logic_vector(3 downto 0);
               W2: in std_logic_vector(3 downto 0);
               Q1: buffer std_logic_vector(3 downto 0);
               Q2: buffer std_logic_vector(3 downto 0));
end Efekt;

architecture Behavioral of Efekt is
begin
    process(CLK, A)
    begin
        case A is
            when "011" =>
                if CLK = '1' then
                    Q1 <= W1;
                    Q2 <= W2;
                else
                    Q1 <= "1100";
                    Q2 <= "1100";
                end if;
            when others =>

```

```

        Q1 <= W1;
        Q2 <= W2;
    end case;
end process;
end Behavioral;

=====
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Debouncer is
    port(      CLK: in std_logic;
               I: in std_logic;
               O: buffer std_logic);
end Debouncer;

architecture Behavioral of Debouncer is
signal R: std_logic := '1';
signal Delay: std_logic := '1';
begin

    IC0: entity work.RS port map
    (
        CLK => CLK,
        S => I,
        R => R,
        Q => O
    );

    IC1: entity work.Licznik_Delay port map
    (
        CLK => CLK,
        Start => I,
        Q => Delay
    );

    process(I, Delay)
    begin
        R <= Delay nand I;
    end process;
end Behavioral;

library IEEE;

```

```

use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity RS is
    port(      CLK: in std_logic;
               S: in std_logic;
               R: in std_logic;
               Q: buffer std_logic);
end RS;

architecture Behavioral of RS is
begin
    process(clk)
    begin
        if S = '1' and R = '1' then
            Q <= Q;
        end if;
        if S = '1' and R = '0' then
            Q <= '0';
        end if;
        if S = '0' and R = '1' then
            Q <= '1';
        end if;
    end process;
end Behavioral;

```

=====

```

library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.std_logic_unsigned.all;

entity Licznik_Delay is
    port(      clk: in std_logic;
               Start: in std_logic;
               Q: out std_logic);
end Licznik_Delay;

architecture Behavioral of Licznik_Delay is
signal Licznik: std_logic_vector (17 downto 0) :=
"00000000000000000000";
begin
    process(clk)
    begin
        if clk'event and clk = '1' then
            if Licznik = 0 then
                Licznik <= Licznik;
                Q <= '1';
            else
                Licznik <= Licznik + 1;
                Q <= '0';
            end if;
        end if;
    end process;
end Behavioral;

```

```

        end if;

        if Start = '0' and Licznik = 0 then
            Licznik <= Licznik + 1;
            Q <= '0';
        end if;
    end if;
end process;
end Behavioral;
=====
```

- Moduły ROM mają dokładnie taką samą strukturę, różnią się tylko danymi zapisanymi w swej strukturze dlatego zostanie przedstawiony tylko jeden z modułów, reszta jest analogiczna do przedstawionego.

```

library ieee;
use ieee.std_logic_1164.all;

entity ROM_0 is
    port ( A : in std_logic_vector(2 downto 0);
           Q : out std_logic_vector(7 downto 0));
end entity ROM_0;

architecture Behavioral of ROM_0 is
type M is array ( 0 to 7) of std_logic_vector(7 downto 0);
constant ROM : M := (    0 => "00000000",
                        1 => "01111100",
                        2 => "11111110",
                        3 => "10000010",
                        4 => "10000010",
                        5 => "11111110",
                        6 => "01111100",
                        7 => "00000000");
begin
    process (A)
    begin
        case A is
            when "000" => Q <= ROM(0);
            when "001" => Q <= ROM(1);
            when "010" => Q <= ROM(2);
            when "011" => Q <= ROM(3);
            when "100" => Q <= ROM(4);
            when "101" => Q <= ROM(5);
            when "110" => Q <= ROM(6);
            when "111" => Q <= ROM(7);
        end case;
    end process;
end architecture Behavioral;
```