

Urządzenia cyfrowe i systemy wbudowane 2 Projekt  
**Implementacja gry „Milionerzy” dla FPGA Spartan3E**

Nikola Meżykowska, Barbara Parzonka

Czerwiec 2024

# Spis treści

<b>1 Sformułowanie tematu projektu</b>	<b>3</b>
<b>2 Opis funkcjonalny projektu</b>	<b>4</b>
2.1 Pytania . . . . .	4
<b>3 Schemat urządzenia</b>	<b>5</b>
3.1 Moduły własne . . . . .	5
3.1.1 Moduł file_selector . . . . .	5
3.1.2 Moduł text_module . . . . .	7
3.1.3 Moduł converter . . . . .	10
3.1.4 Moduł control . . . . .	12
3.2 Wykaz czarnych skrzynek . . . . .	13
3.3 Wymagane urządzenia . . . . .	14
<b>4 Instrukcja gry</b>	<b>15</b>
<b>5 Ocena uzyskanych efektów i analiza możliwości dalszej rozbudowy urządzenia</b>	<b>19</b>
5.1 Porównanie pierwotnych założeń urządzenia z jego finanlnym kształtem . . . . .	19
5.2 Możliwości dalszej rozbudowy . . . . .	19
<b>6 Źródła</b>	<b>21</b>

## 1 Sformułowanie tematu projektu

Tematem projektu jest implementacja gry "Milionerzy" działającej na płytce Spartean-3E FPGA. Gra dotyczy tematów ogólnych (w szczególności zaś dotyczących Wrocławia i Politechniki Wrocławskiej). Składa się z serii pytań, na które użytkownik odpowiada. Za każdą poprawną odpowiedź użytkownik otrzymuje jeden punkt. Gra kończy się, gdy użytkownik odpowie na przewidzianą liczbę pytań (14). Celem gry jest uzyskanie jak największej liczby punktów. Liczba punktów jest wyświetlania na diodach.

Cały projekt znajduje się w publicznym repozytorium: <https://github.com/bParz156/Millionerzy-FPGA-game>

## 2 Opis funkcjonalny projektu

Działanie gry wymaga użycia sprzętu wymienionego w sekcji [3.3](#).

### 2.1 Pytania

Katalog pytań może zostać dostosowany przez użytkownika <- należy zgrać odpowiednie pytania na kartę SD i zmienić listę pytań podanych w ByteArray w module **TextModule**. Konieczne jest jednak pozostawienie pytania o nazwie *m.txt* (kod HEX: *x"6d"*) jako pierwszego - zawiera on instrukcję użytkownika - oraz pytania o nazwie *Z.txt* (kod HEX: *x"5a"*) - podsumowanie gry (szczegóły na temat komunikatów zawartych w tych plikach znajdują się w sekcji [4](#)).

Pytania muszą spełniać poniższe warunki:

- zawierać 5 enterów
- znak znajdujący się po znaku *"!"* jest traktowany jako odpowiedź na dane pytanie, dlatego znak *"!"* powinien znajdować się jedynie po ostatnim enterze
- jakakolwiek treść znajdująca się za znakiem *"!"* nie będzie wyświetlana na ekranie, dlatego nie może się on znajdować w treści pytania ani odpowiedzi
- jeden plik jest odpowiedzialny za jedno pytanie
- nazwą pliku z pytaniem nie może być *Z.txt* ani *m.txt*
- plik pytania ma rozszerzenie *.txt*

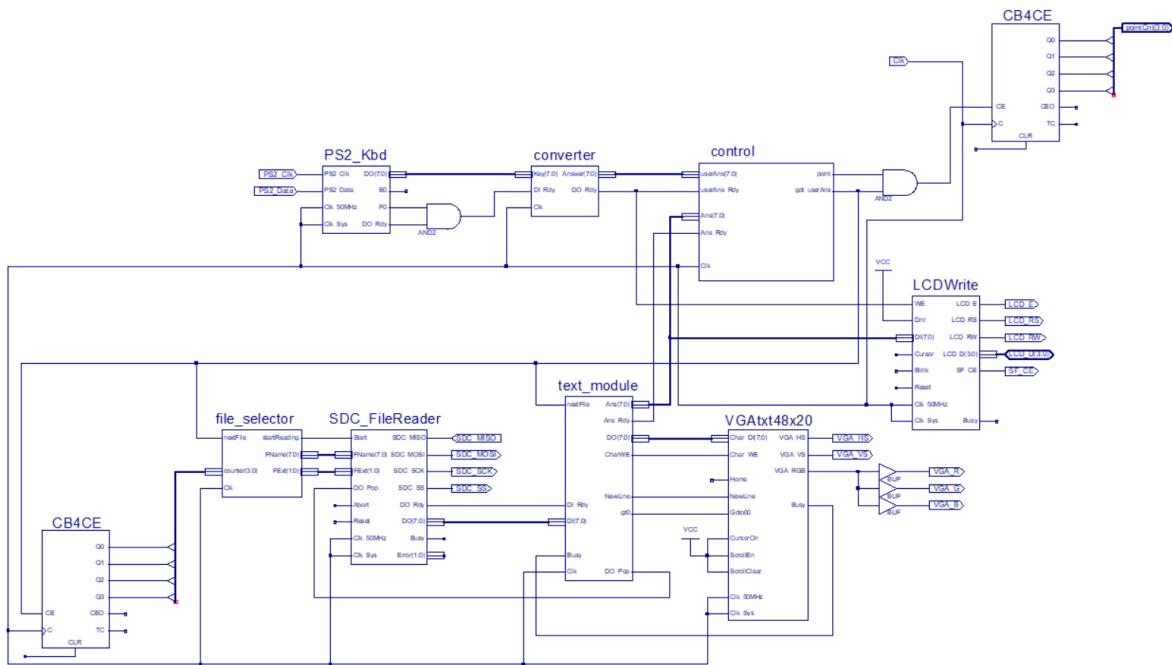
### 3 Schemat urządzenia

W projekcie urządzenia zastosowano zarówno czarne skrzynki (Źródło 2) jak i moduły własne. Działanie układu oparte jest o jeden sygnał zegarowy Clk. Wszystkie zmiany zachodzą przy rosnącym zboczu zegarowym.

Listing 1: Zmiana stanu

```
process1 : process (Clk)
begin
    if rising_edge(Clk) then
        state <= next_state;
    end if;

end process process1;
```



Rysunek 1: Schemat układu

#### 3.1 Moduły własne

##### 3.1.1 Moduł file\_selector

Rola modułu jest kontrola wyboru pytania wyświetlanego na ekranie VGA oraz liczby pytań, które mają zostać zadane użytkownikowi.

Moduł działa przy użyciu portów (Tabela 1). Działanie modułu opiera się na czterech stanach:

- *off* - stan początkowy, bezpośrednio z niego przechodzi się do stanu *starting*

- *starting* - stan, do którego przechodzi moduł, gdy ma zostać odczytany nowy plik. Wówczas sygnał wyjściowy **startReading** (patrz: Tabela 1) jest uaktywniany. Po jednym cyklu zegarowym układ przechodzi w stan *started*
- *started* - w tym stanie na podstawie licznika pytań (sygnał **counter** jest zmieniany na wartość typu **integer** i ustawiany w **questionCnt**) ustawiana jest nazwa kolejnego pytania (sygnał **FName**). Jeśli licznik pytań dojdzie do wartości 15, to moduł przechodzi w stan *finito*
- *finito* - zakończono test, nie ma więcej pytań.

Nazwa portu	Kierunek	typ	Znaczenie
nextFile	in	STD_LOGIC	oznacz, że kolejny plik (pytanie) powinien być czytany
counter	in	STD_LOGIC_VECTOR(3:0)	licznik pytań
Clk	in	STD_LOGIC	sygnał zegarowy
startReading	out	STD_LOGIC	impuls wysyłany na port Start modułu SDC_FileReader, aby rozpocząć odczyt pliku.
FName	out	STD_LOGIC_VECTOR(7:0)	nazwa pliku, który ma zostać odczytany przez moduł SDC_FileReader
FExt	out	STD_LOGIC_VECTOR(1:0)	Format pliku odczytywanego z karty SD. Ustawiony na stałe na "10 patrz ( 3.2 )

Tabela 1: Opis portów modułu file\_selector

```

process2 : process(state)
begin
next_state <= state;
  case state is
    when off =>                      -- kiedyś grę uruchamiało się przyciskiem, stan pozostał
      next_state <= starting;
    when starting =>                  -- stan podający impuls startReading, rozpoczynający odczyt z karty SD
      next_state <= started;
    when started =>
      if questionCnt=15 then           -- koniec gry po 15 pytaniach (właściwie 14, bo pierwsze to menu)
        next_state=<finito>;
      elsif nextFile = '1' then
        next_state <=starting;
      end if;
    when finito=>
      null;
  end case;
end process process2;

```

Rysunek 2: Logika stanów w file\_selector

```

process_change_name : process(state, questionCnt)
begin
    case state is
        when started => FName <= questions(questionCnt); -- wybranie kolejnej nazwy pliku z tablicy
        when others => null;
    end case;
end process process_change_name;

questionCnt<= TO_INTEGER(UNSIGNED(counter));

startReading <= '1' when state = starting else '0';
FExt <= "10";

```

Rysunek 3: Przypisanie wartości sygnałom wyjściowym

### 3.1.2 Moduł `text_module`

Moduł działa przy użyciu portów (Tabela 2). Działanie modułu opiera się na ośmiu stanach:

- *start* - sygnał wewnętrzny *counted* jest zerowany, moduł przechodzi do kolejnego stanu: *reading*, a kurSOR w ekranie VGA przechodzi do pozycji początkowej - lewy górnY róg
- *reading* - trwa odczyt pliku z karty SD i jego wyświetlenie na ekranie VGA. Jeśli sygnał **DI\_Rdy** jest aktywny, a odczytany znak to znak nowej linii (enter = *x"0D"*), to moduł przechodzi w stan *endl*, a jeśli odczytany znak to *"!"*, to znaczy, że osiągnięto koniec pliku - przejście w stan *ansSTBrdy*. Jeśli żaden ze wcześniejszych warunków nie został spełniony, moduł przechodzi w stan *writing*
- *writing* - na wyjście **DO** przekazywane jest wejście **DI** - znak ten jest wypisywany na ekranie. Następuje powrót do stanu *reading*
- *endl* - inkrementowany jest licznik linii (enterów - zgodnie z konwencją przedstawioną w sekcji 2.1 pytanie powinno zawierać 5 enterów), uaktywniane są sygnały **DO\_Pop** oraz **CharWE**, które pozwalają na obsługę przejścia do nowej linii w module **VGAtxt48x20**
- *ansSTBrdy* - wprowadza opóźnienie przed wejściem do stanu *ansRdy*
- *ansRdy* - oznacza, że zakończono odczyt pliku i wyświetlanie go na ekranie VGA oraz, że znak wskazywany przez **DI** jest poprawną odpowiedzią na pytanie. Jeśli sygnał **nextFile** jest aktywny, to na wyjście **Ans** jest ustawiana wartość z **DI** oznaczająca poprawną odpowiedź. Gdy użytkownik poda odpowiedź na pytanie, moduł przechodzi w stan *clr*
- *clr* - czyszczenie ekranu. Gdy sygnał **Busy** jest nieaktywny, przejście do stanu *clrEndl*. W tym stanie sprawdzany jest licznik linii. Ustalono arbitralnie, że jeśli osiągnięty zostanie stan 61 linii, to na pewno poprzednie pytanie zostało wyczyszczone i wówczas można przejść do stanu *start*.
- *clrEndl* - czyszczenie ekranu, przejście do nowej linii

Czyszczenie ekranu stosowane w zaprojektowanym programie polega na przewinięciu ekranu w dół, aż poprzednia zawartość ekranu nie będzie dłużej widoczna. Oprócz sygnałów opisanych w tabeli 2 w module występują sygnały wewnętrzne:

- **lineCnt** : **UNSIGNED (5 downto 0)** - 6-bitowy licznik enterów wypisanych dla danego pytania, pozwala na kontrolę przewinięcia ekranu w dół

- **counted** : STD\_LOGIC - flaga mówiąca o tym, czy odpowiedź została już przekazana.
- **imuplsPop** : STD\_LOGIC - impuls mówiący o konieczności przejścia do nowej linii.

Nazwa portu	Kierunek	typ	Znaczenie
DI	in	STD_LOGIC_VECTOR(7:0)	znak odczytany z pliku podany przez moduł <b>SDC_FileReader</b>
nextFile	in	STD_LOGIC_VECTOR	informuje o tym, czy odczytywany będzie kolejny plik. Gdy zostanie uaktywniony w stanie <i>ansRdy</i> , to moduł przechodzi do stanu <i>clr</i>
Clk	in	STD_LOGIC	sygnał zegarowy
Busy	in	STD_LOGIC	na ten port przekazywany jest sygnał wyjściowy modułu <b>VGAtxt48x20</b> o tej samej nazwie. Jeśli jest on aktywny, tzn. że wszelkie sygnały będą ignorowane przez moduł VGA
DI_Rdy	in	STD_LOGIC	wejście <b>DI</b> jest gotowy do odczytu
DO	out	STD_LOGIC_VECTOR(7:0)	Znak przekazywany na wyjście - powinien zostać wyświetlony na ekranie VGA
Ans	out	STD_LOGIC_VECTOR(7:0)	Znak będący poprawną odpowiedzią na pytanie
NewLine	out	STD_LOGIC_VECTOR	Flaga mówiąca o konieczności przejścia do nowej linii przez moduł <b>VGAtxt48x20</b>
Ans_Rdy	out	STD_LOGIC	Znak podawany na wyjściu <b>Ans</b> rzeczywiście oznacza odpowiedź na pytanie
gt0	out	STD_LOGIC	Flaga mówiąca o konieczności powrotu kurSORA do podstawowej pozycji przez moduł <b>VGA-txt48x20</b>
DO_Pop	out	STD_LOGIC	Flaga przekazywana na wejście modułu <b>SDC_FileReader</b> o tej samej nazwie oznaczająca prośbę o odczytanie kolejnego znaku z pliku.
CharWE	out	STD_LOGIC	Flaga przekazywana na wejście modułu <b>VGAtxt48x20</b> o tej samej nazwie. Służy do wypisania znaku na ekranie. Jest konieczna, gdy następuje przejście do nowej linii.

Tabela 2: Opis portów modułu *text\_module*

```

process2 : process(state, DI_Rdy)
begin
    next_state <= state;
    case state is
        when start =>
            counted<='0';
            next_state <= reading;
        when reading =>                      -- odczytywanie znaku
            imuplsPop<='0';
            if DI_Rdy = '1' then
                if DI = x"0D" then      -- jeśli nowa linia w pliku, przejdź do nowej linii na ekranie
                    next_state <= endl;
                elsif DI = x"21" then   -- jeśli '!' przejdź do stanu answer Soon To Be ready
                    next_state <= ansSTBrdy;
                else
                    next_state <= writing;
                end if;
            end if;
        when writing =>                     --wypisanie odczytanego znaku i powrót do odczytywania
            next_state <= reading;
            imuplsPop<='0';
        when endl =>                      -- przejście do nowej linii
            next_state <= reading;
            imuplsPop<='1';
        when ansSTBrdy =>                  -- odczytano '!', następny znak jest poprawna odpowiedzią
            next_state <= ansRdy;
            imuplsPop<='0';
        when ansRdy =>                     -- odczytano poprawną odpowiedź
            imuplsPop<='0';
            if nextFile ='0' then
                if counted='0' then
                    Ans<=DI;
                    counted<='1';
                end if;
            else
                next_state <= clr;
            end if;
        when clr =>                         -- upewnienie się, że cały ekran zostanie wyczyszczony
            imuplsPop<='0';
            if lineCnt = "111110" then
                next_state <= start;
            elsif Busy ='0' then               -- czekanie na wyczyszczenie linii
                next_state <= clrEndl;
            end if;
        when clrEndl=>                   -- czyszczenie linii
            next_state <= clr;
            imuplsPop<='1';
        when others => null;
    end case;
end process process2;

```

Rysunek 4: Logika stanów modułu text\_module

```

process_counter : process(state, Clk)
begin
  if rising_edge(Clk) then
    case state is
      when start => lineCnt<="000000";
      when endl => lineCnt<=lineCnt+1;
      when clrEndl => lineCnt<=lineCnt+1;
      when others => null;
    end case;
  end if;
end process process_counter;

process_DO: process(state)
begin
  case state is
    when start => DO<=DI;
    when endl => DO<=DI;
    when reading => DO<=DI;
    when writing => DO<=DI;
    when ansRdy => DO<=DI;
    when clr => DO<=x"0D";
    when others => null;
  end case;
end process process_DO;

DO_Pop <= '1' when state = writing or state=ansSTBrdy or imuplsPop = '1' else '0';
CharWE<= '1' when state = writing or state=endl or state=clrEndl else '0';
newLine <= '1' when state = endl or state = clrEndl else '0';
Ans_Rdy <= '1' when state = ansRdy else '0';
gt0 <= '1' when state = start else '0';

```

Rysunek 5: Przypisanie wyjść w module text\_module

### 3.1.3 Moduł converter

Rola modułu jest konwersja wyjścia kodu klawisza podanego przez moduł *PS2\_Kbd* na znak ASCII. Jedyne konwertowane znaki to A, B, C i D.

Moduł działa przy użyciu portów (Tabela 3). Działanie modułu opiera się na trzech stanach:

- *waiting* - stan oczekiwania na znak podany z klawiatury. Po pojawienniu sygnału na porcie **DI\_Rdy** przechodzi do stanu *converting*.
- *converting* - stan, w którym odbywa się konwersja. Trwa jeden takt zegara i przechodzi do stanu *ready*.
- *ready* - trwa jeden takt zegara, podczas którego wyjście **DO\_Rdy** jest ustawiane na '1'. Wraca do stanu *waiting*

Nazwa portu	Kierunek	typ	Znaczenie
Key	in	STD_LOGIC_VECTOR (7 downto 0)	Kod znaku z klawiatury
DI_Rdy	in	STD_LOGIC	Sygnal informujacy o gotowym znaku na porcie Key
Clk	in	STD_LOGIC	sygnal zegarowy
Answer	out	STD_LOGIC_VECTOR (7 downto 0)	Znak ASCII po konwersji
DO_Rdy	out	STD_LOGIC	Sygnal informujacy o gotowym znaku na porcie Answer

Tabela 3: Opis portów modułu converter

```

processStates : process(state, DI_Rdy)
begin
    next_state <= state;
    case state is
        when waiting =>
            if DI_Rdy ='1' then
                next_state <= converting;
            end if;
        when converting =>
            next_state<=ready;
        when ready =>
            next_state<=waiting;
    end case;
end process processStates;

```

Rysunek 6: Logika stanów w module converter

```

processConvert : process(state, Key)
begin
    if state=converting then
        case Key is
            when x"1C"=>
                Answer<=x"41";
            when x"32"=>
                Answer<=x"42";
            when x"21"=>
                Answer<=x"43";
            when x"23"=>
                Answer<=x"44";
            when others=>
                Answer<=x"00";
        end case;
    end if;
end process processConvert;

```

Rysunek 7: Konwersja w module converter

### 3.1.4 Moduł control

Rolą modułu jest sprawdzanie poprawności odpowiedzi podanej przez użytkownika i przyznawanie punktów. Moduł działa przy użyciu portów (Tabela 4). Działanie modułu opiera się na czterech stanach:

- *waitAns* - stan oczekiwania na znak będący poprawną odpowiedzią przekazywaną przez moduł **text\_module**. Po otrzymaniu jedynki na porcie **Ans\_Rdy** przechodzi do stanu *waitIn*.
- *waitIn* - stan oczekiwania na znak będący odpowiedzią użytkownika przekazywaną przez moduł **converter**. Po otrzymaniu jedynki na porcie **userAns\_Rdy** przechodzi do stanu *gotIn*.
- *gotIn* - trwa jeden takt zegara, przechodzi do stanu *validAns*.
- *validAns* - trwa jeden takt zegara, podczas którego przekazywana jest informacja o otrzymaniu odpowiedzi użytkownika oraz podejmowana jest decyzja o przyznaniu punktu. Wraca do stanu *validAns*.

Nazwa portu	Kierunek	typ	Znaczenie
userAns	in	STD_LOGIC_VECTOR (7 downto 0)	Znak ASCII będący odpowiedzią użytkownika na pytanie, podany z modułu <b>converter</b>
userAns_Rdy	in	STD_LOGIC	Sygnal informujący o gotowym znaku na porcie <b>userAns</b>
Clk	in	STD_LOGIC	sygnał zegarowy
Ans	in	STD_LOGIC_VECTOR (7 downto 0)	Znak ASCII będący poprawną odpowiedzią na pytanie, podany z modułu <b>text_module</b>
Ans_Rdy	in	STD_LOGIC	Sygnal informujący o gotowym znaku na porcie <b>Ans</b>
point	out	STD_LOGIC	Sygnal informujący o tym, czy odpowiedź użytkownika zgadza się z poprawną. Używany do przydzielania punktów.
got_userAns	out	STD_LOGIC	Sygnal informujący o tym, że użytkownik udzielił odpowiedzi. Używany do przydzielania punktów oraz przechodzenia do kolejnego pytania.

Tabela 4: Opis portów modułu control

```

process2 : process(state, Ans_Rdy, userAns_Rdy)
begin
    next_state<=state;
    case state is
        when waitAns =>                      -- czekanie na podanie poprawnej odpowiedzi z text_module
            if Ans_Rdy = '1' then
                next_state <= waitIn;
            end if;
        when waitIn =>                        -- czekanie na odpowiedź użytkownika
            if userAns_Rdy = '1' then
                next_state <= gotIn;
            end if;
        when gotIn =>
            next_state <= validAns;
            when validAns=>                  -- przekazanie informacji o otrzymaniu odpowiedzi użytkownika
                next_state<=waitAns;           -- (można naliczyć punkt i przejść do następnego pytania)
            end case;
            if Ans_Rdy = '0' then
                next_state <= waitAns;
            end if;
    end process process2;

point<='1' when Ans=userAns else '0';
got_userAns <= '1' when state = validAns else '0';

```

Rysunek 8: Logika stanów w module control

### 3.2 Wykaz czarnych skrzynek

- **Moduł PS2\_Kbd**

Pozwala na pracę z klawiaturą - pdczyt klawisza odpowiedzi wciśniętego przez użytkownika. Aby zapewnić, że klawisz został wciśnięty i puszczyony, za gotowość odpowiedzi od użytkownika uznaje się moment, w którym zarówno sygnał *F0* (zwolnienie klawisza) jak i sygnał *DO\_Rdy* są aktywne. Na

wyjściu moduł podaje kod klawisza, dlatego przed porównaniem odpowiedzi użytkownika z poprawną odpowiedzią konieczna jest konwersja kodu na znak ASCII (sekcja [3.1.3](#))

- **Moduł LCDWrite**

Jest odpowiedzialny za wyświetlenie na ekranie LCD poprawnej odpowiedzi na poprzednie pytania.

- **Moduł VGAtxt48x20**

Zadaniem tego modułu jest wyświetlanie pytania na ekranie VGA, a także przygotowanie ekranu do nowego pytania (scrollowanie ekranu, zapewniające "wyczyszczenie" ekranu z poprzedniego pytania i przejście do lewego górnego rogu ekranu przed wyświetleniem nowego).

- **Moduł SDC \_ FileReader**

Jest wykorzystywany do odczytania pytania o zadanej nazwie z karty SD podpiętej do urządzenia. Sygnał *DO\_Pop* jest żądaniem przejścia do kolejnego znaku, a sygnał *Start* jest impusem zmuszającym kartę do odczytu plików o nazwie *FName*. Definicja *FName* jako wektora 8-bitowego sprawia, że nazwy plików z pytania muszą być jednoznakowe. Sygnał *FExt*, oznaczający format odczytywanego pliku, jest stale ustawiony w module **FileSelector** na "10", co oznacza plik o rozszerzeniu *.txt*.

- **Liczniki CB4CE**

4-bitowe liczniki NBC. Zliczają one, jeśli ich wejście CE jest aktywne. Licznik znajdujący się w lewym dolnym rogu schematu (Rysunek [1](#)) jest odpowiedzialny za zliczanie pytań, dlatego na jego wejście CE doprowadzone jest wyprowadzenie **got\_userAns** z modułu **control** (sekcja [3.1.4](#)).

Drugi licznik występujący w projekcie jest odpowiedzialny za zliczanie poprawnych odpowiedzi, dlatego na jego wejście CE doprowadzony jest iloczyn logiczny sygnałów **point** i **got\_userAns** z modułu **control** (sekcja [3.1.4](#)).

### 3.3 Wymagane urządzenia

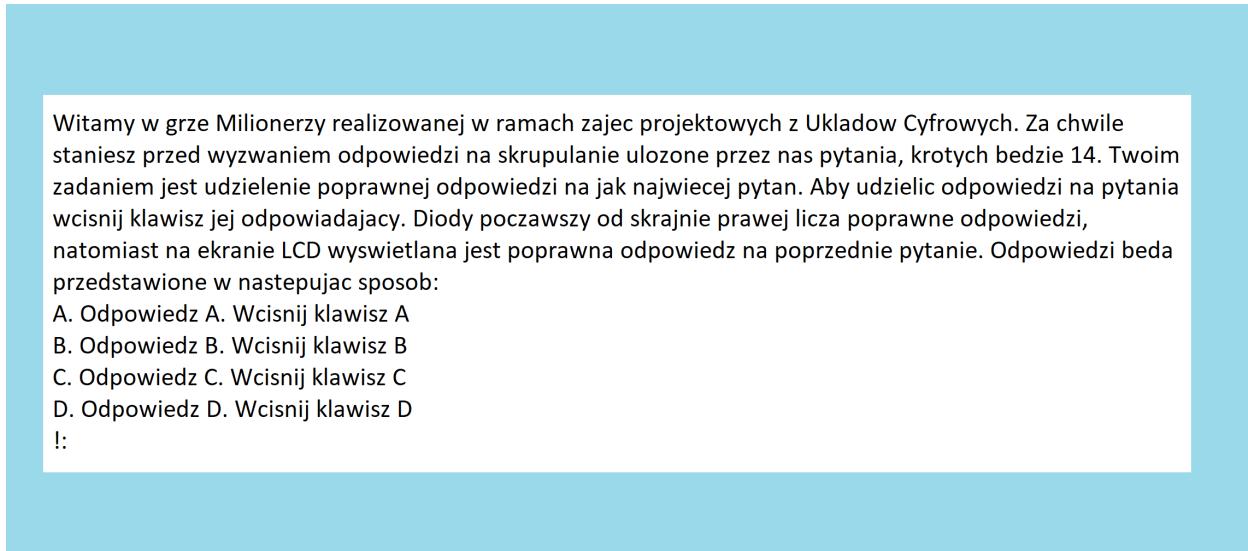
Gra działa w opraciu o obecność dodatkowych urządzeń. Są to:

- płytka Spartan-3E FPGA z ekranem LCD i diodami LED
- karta SD
- ekran z połączeniem VGA
- klawiatura działająca w protokole PS2.

## 4 Instrukcja gry

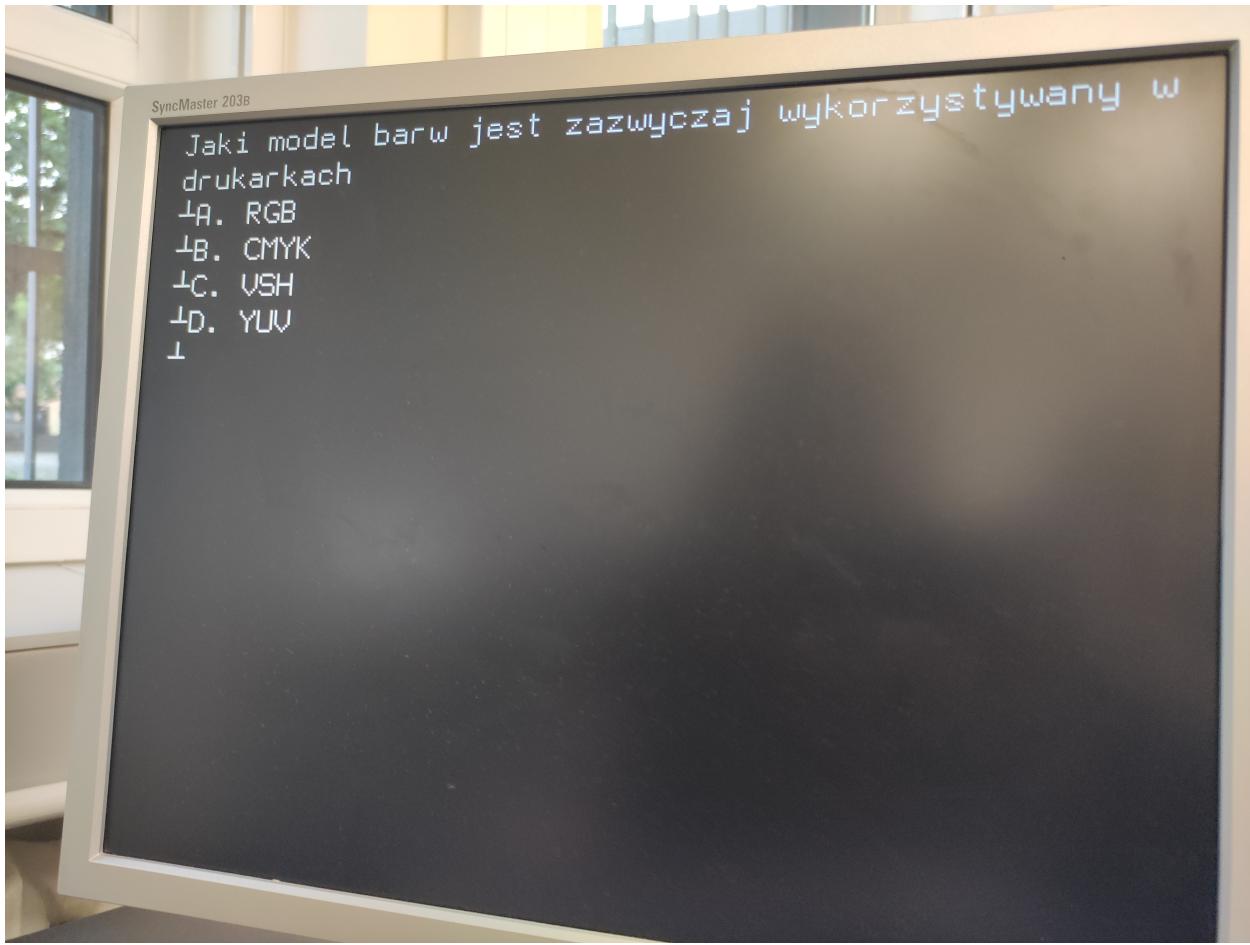
Przed rozpoczęciem należy się upewnić, że w urządzeniu znajduje się karta SD z pytaniami! (Na temat pytań można dowiedzieć się więcej w sekcji [2.1](#))

Na początku gry, na ekranie VGA, na którym w czasie trwania gry będą pojawiały się pytania, wyświetlana jest instrukcja gry (Patrz: Rysunek [9](#)).



Rysunek 9: Instrukcja gry

Gracz odpowiada na pytania, wciskając klawisz klawiatury PS2 odpowiadający wybranej odpowiedzi. Możliwe odpowiedzi na pytania to "A", "B", "C" i "D". Podanie jako odpowiedzi dowolnego innego klawisza skutkuje niezaliczeniem odpowiedzi.(Rysunek [10](#))



Rysunek 10: Sposób wyświetlania pytania

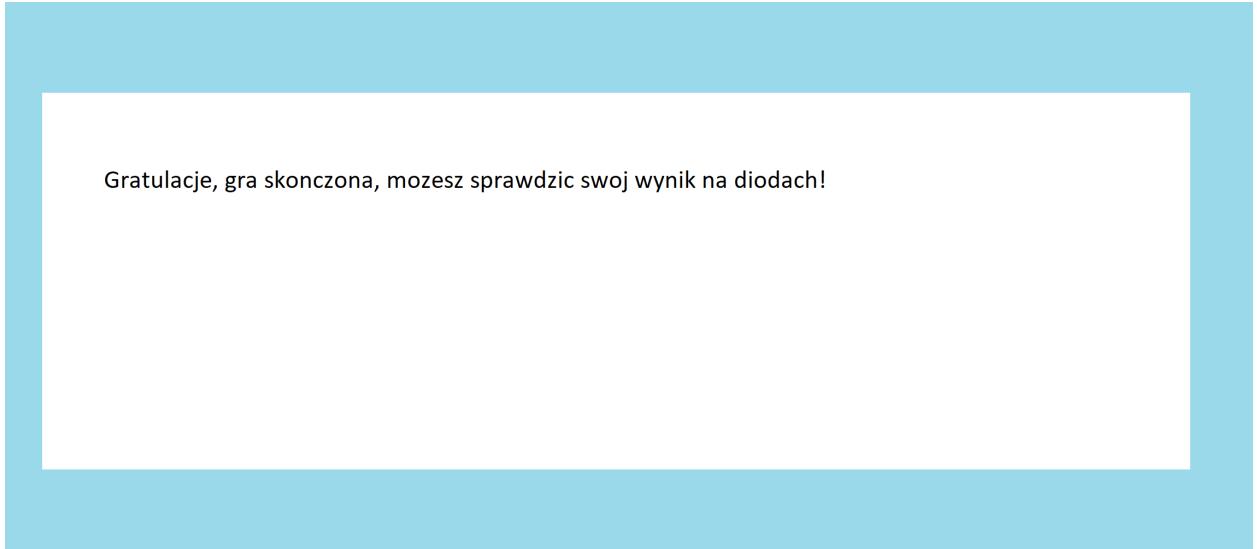
Użytkownik może sprawdzać poprawność udzielonej odpowiedzi, analizując stan licznika punktów wyświetlanego na czterech skrajnie prawych diodach urządzenia. Na ekranie LCD wyświetlane są poprawne odpowiedzi na poprzednie pytania. (Rysunek 11).



Rysunek 11: Widok ekranu LCD i diod

Nie jest możliwy powrót do poprzedniego pytania ani opuszczenie pytania. Gra nie pozwala na resetowanie (nie posiada żadnego guzika umożliwiającego taką funkcję) - jeśli użytkownik zechce zagrać od początku, konieczne jest ponowne zgranie programu gry na Spartanie.

Gdy użytkownik odpowie na wszystkie pytania wyświetlany jest stosowny komunikat (Rysunek 12).



Gratulacje, gra skonczona, mozesz sprawdzic swój wynik na diodach!

Rysunek 12: Kouminkat na koniec gry

## 5 Ocena uzyskanych efektów i analiza możliwości dalszej rozbudowy urządzenia

### 5.1 Porównanie pierwotnych założeń urządzenia z jego finanlnym kształtem

Pomyślnie zrealizowano odczyt pytań z karty SD oraz połączenie funkcjonalności odpowiedzi na pytania z klawiaturą PS2. Nie udało się zrealizować generatora losowego, który pozwalałby na losowy wybór pytań ze zbioru, dlatego kolejność pytań jest stała. Osiągnięto natomiast nową funkcjonalność - na ekranie LCD wyświetlane są poprawne odpowiedzi na poprzednie pytania. Dzięki temu osiągnięto wymiar edukacyjny gry - "bawi i uczy". Uzyskane efekty pozwoliły na prawie pełną zakładaną funkcjonalność gry: możliwa jest gra, znakowany jest koniec gry, następuje weryfikacja punktów, a użytkownik jest informowany o swoim wyniku. Pytania i odpowiedzi są wyświetlane na ekranie w sposób czytelny dla użytkownika, a odpowiedź na pytanie dzięki użyciu klawiatury jest intuicyjna.

Zaletą gry jest duża czytelność i przejrzystość pytań uzyskana dzięki wyświetlaniu pojedynczego pytania na ekranie. Gra może zostać w łatwy sposób spersonalizowana poprzez wymianę pytań - muszą one jednak spełniać wymagania opisane w sekcji [2.1](#) oraz nazywać się w określony sposób (znaki: x"45", x"46", x"47", x"48", x"49", x"4A", x"4E", x"4F", x"50", x"51", x"52", x"53", x"54", x"55").

### 5.2 Możliwości dalszej rozbudowy

Możliwości dalszej rozbudowy obejmują:

- wprowadzenie losowej kolejności pytań
- wyświetlanie poprawnej odpowiedzi na pytanie na ekranie VGA zamiast na ekranie LCD (przepisanie pytania jeszcze raz z wyświetlaniem poprawnej odpowiedzi) - problem z przełączeniem ekranu na kolejne pytanie, konieczność modyfikacji **modułu TextModule**. (Rysunek [13](#))
- możliwość ponownego uruchomienia - resetu gry - bez konieczności zgrywania programu na płytke, np. poprzez naciśnięcie klawisza

SyncMaster 203B

Która z wymienionych cech nie jest paradigmatem programowania obiektowego

- ↑A. abstrakcja
- ↑B. współbieżność
- ↑C. polimorfizm
- ↑D. heretyzacja

↑ B

Rysunek 13: Propozycja wyświetlania pytania z odpowiedzią

## 6 Źródła

1. Czarne skrzynki: [https://indyk.ict.pwr.wroc.pl/ucyfr/fpga/\\_Toc99974103](https://indyk.ict.pwr.wroc.pl/ucyfr/fpga/_Toc99974103)
2. Programowanie w VHDL, instrukcja środowiska ISE: wykłady dr. Jarosława Sugiera z kursu Układy Cyfrowe i Systemy Wbudowane
3. Inspiracje do pytań
  - Pytania o Politechnice Wrocławskiej <https://pwr.edu.pl/uczelnia/informacje-ogolne/historia-i-rozwoj/historia-wroclawia>
  - Pytania o Wrocławiu: <https://www.wroclaw.pl/dla-mieszkanca/jak-dobrze-znasz-wroclaw-rozwiaz-nasz-quiz>