Technika Cyfrowa
Wybór z menu na wyświetlaczach siedmiosegmentowych
Implementacja na układzie FPGA

Radosław Szepielak

5 czerwca 2025

1 Opis zadania

Korzystając z programu Quartus firmy Altera (Intel) (www.altera.com), należy w dowolnym wybranym układzie scalonym FPGA stworzyć działający system sterujący, realizujący bardzo prosty wybór z menu. Mamy do dyspozycji dwa wyświetlacze siedmiosegmentowe (lewy i prawy) oraz dwa przyciski (lewy i prawy). Wciśnięcie lewego przycisku powoduje zwiększanie o jeden wartości wyświetlanej na lewym wyświetlaczu. Po osiągnięciu wartości 9, wartość ta zmienia się na 0 przy ponownym wciśnięciu lewego przycisku. Po już ustawieniu lewym przyciskiem żądanej wartości na lewym wyświetlaczu, wciśnięcie prawego przycisku powinno spowodować zapamiętanie wartości z lewego wyświetlacza na wyświetlaczu prawym, co świadczy o dokonanym wyborze wartości z menu.

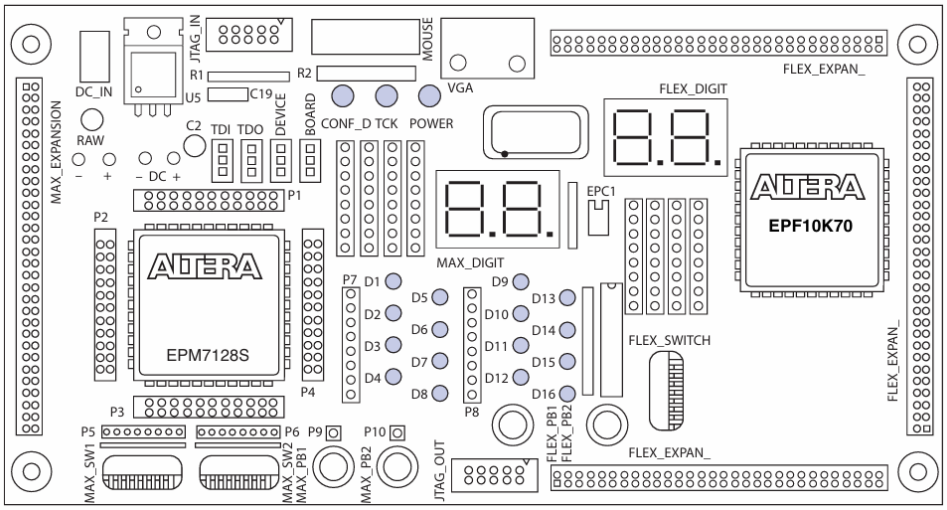
Po uruchomieniu systemu, obydwa wyświetlacze powinny pokazywać wartość zero. Całość powinna działać zgodnie z filmem z poniższego linku:

`http://home.agh.edu.pl/~dlugopol/tc2025/fpga-menu.mp4`

2 Krótka charakterystyka FPGA

2.1 Co to jest?

Field-Programmable Gate Arrays (FPGA) to programowalne układy cyfrowe, które można konfigurować po wyprodukowaniu. Składają się z macierzy bloków logicznych wyposażonych w tablice LUT, przerzutniki i układy sumujące, co umożliwia tworzenie złożonych funkcji cyfrowych.



Rysunek 1: Schemat blokowy płytki UP2 Education

2.2 Zalety FPGA

- **Równoległość działania:** Operacje w FPGA wykonują się jednocześnie
- **Elastyczność:** Możliwość rekonfiguracji logiki w dowolnym momencie
- **Niskie opóźnienia:** Brak systemu operacyjnego i bezpośrednia implementacja algorytmów w sprzęcie zapewnia znacznie większą szybkość wykonania i niskie opóźnienia

2.3 Wady FPGA

- **Wyższy koszt jednostkowy** w porównaniu do ASICów (Application Specific Integration Circuit), szczególnie przy dużej skali produkcji
- **Większe zużycie energii**
- **Mniejsza wydajność** w porównaniu z dedykowanymi układami ASIC

2.4 Zastosowania FPGA

- **Systemy** wizyjne: Szybka analiza obrazu w czasie rzeczywistym
- **Przetwarzanie sygnałów**: Modemy, routery, switche
- **5G i sieci komórkowe**: Przetwarzanie pakietów, szyfrowanie
- **Kryptografia**: algorytmy kryptograficzne, haszowanie
- **Robotyka medyczna**: Precyzyjne sterowanie robotami chirurgicznymi
- **Ultra-szybki trading**: Giełdy i banki inwestycyjne używają FPGA do wykrywania arbitrażu i błyskawicznego składania zleceń, gdzie każda mikrosekunda oznacza miliony dolarów zysku.

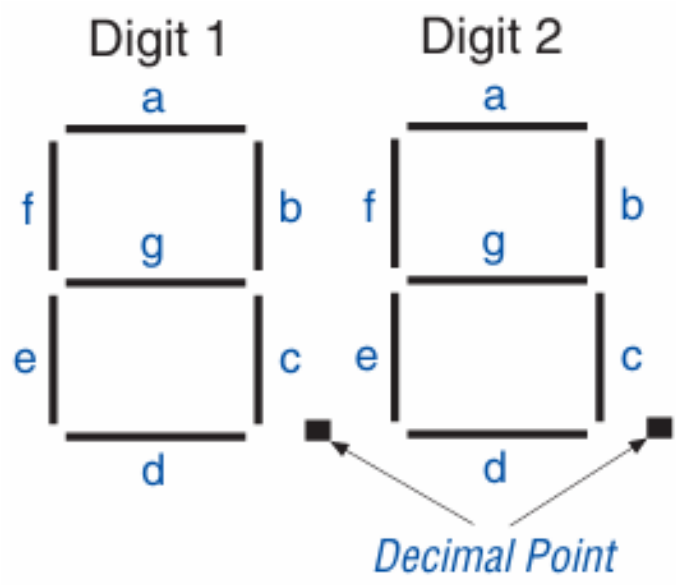
3 Opis rozwiązania zadania

3.1 Użyty układ FPGA i środowisko

- Układ: **Altera FLEX EPF10K70RC240-4 (UP2 Education Kit)**
- Oprogramowanie: **Quartus II ver. 9.0 Web Edition**

3.2 Podstawowa funkcjonalność

Wartości wyświetlane są na dwóch wyświetlaczach siedmiosegmentowych.



Rysunek 2: Wyświetlacz FLEX_DIGIT

Tabela 1: Tabela prawdy układu sterowania wyświetlaczami

Przycisk Lewy (PB1)	**Przycisk Prawy (PB2)**	**Wartość Lewa**	**Wartość Prawa**
0 (naciśnięty)	1	`+1 mod 10`	Bez zmian
1	0 (naciśnięty)	Bez zmian	`= Wartość Lewa`
0	0	`+1 mod 10`	`= Wartość Lewa`
1	1	`Bez zmian`	`Bez zmian`

Stan logiczny przechowywany jest w 7-bitowym wektorze. W implementacji systemu detekcji naciśnięć przycisków zastosowano mechanizm debounce, żeby zapobiec błędnemu odczytowi wielokrotnych naciśnięć spowodowanych drganiami styków.

4 Kod źródłowy projektu w języku VHDL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cw04FPGA is
    port (
        clk     : in  std_logic;
        pb1     : in  std_logic;
        pb2     : in  std_logic;
        hex0    : out std_logic_vector(6 downto 0);  -- wy wietlacz lewy
        hex1    : out std_logic_vector(6 downto 0)   -- wy wietlacz prawy
    );
end entity;

architecture rtl of cw04FPGA is
    signal left_val     : unsigned(3 downto 0) := (others => '0');
    signal selected_val : unsigned(3 downto 0) := (others => '0');


    constant DEBOUNCE_TIME : integer := 100000;
    signal pb1_db_cnt : integer range 0 to DEBOUNCE_TIME := 0;
    signal pb2_db_cnt : integer range 0 to DEBOUNCE_TIME := 0;
    signal pb1_db     : std_logic := '1';
    signal pb2_db     : std_logic := '1';
    signal pb1_prev   : std_logic := '1';
    signal pb2_prev   : std_logic := '1';


    signal pb1_sync   : std_logic_vector(1 downto 0) := "11";
    signal pb2_sync   : std_logic_vector(1 downto 0) := "11";

    signal blink_clock_cycles_cnt : integer range 0 to 25175000 := 0;
    signal blink_toggle : std_logic := '1';
    signal blink_count   : integer range 0 to 5 := 0;
    signal blink_active  : std_logic := '0';

    function to_7seg(n : unsigned(3 downto 0)) return std_logic_vector is
        variable seg : std_logic_vector(6 downto 0);
    begin
        case n is
            when "0000" => seg := "1000000"; -- 0
            when "0001" => seg := "1111001"; -- 1
            when "0010" => seg := "0100100"; -- 2
            when "0011" => seg := "0110000"; -- 3
            when "0100" => seg := "0011001"; -- 4
            when "0101" => seg := "0010010"; -- 5
            when "0110" => seg := "0000010"; -- 6
            when "0111" => seg := "1111000"; -- 7
            when "1000" => seg := "0000000"; -- 8
            when "1001" => seg := "0010000"; -- 9
            when others => seg := "1111111"; -- blank
        end case;
        return seg;
    end function;

begin
```

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

        pb1_sync <= pb1_sync(0) & pb1;
        pb2_sync <= pb2_sync(0) & pb2;


        if pb1_sync(1) = '0' then
            if pb1_db_cnt < DEBOUNCE_TIME then
                pb1_db_cnt <= pb1_db_cnt + 1;
            else
                pb1_db <= '0';
            end if;
        else
            pb1_db_cnt <= 0;
            pb1_db <= '1';
        end if;


        if pb2_sync(1) = '0' then
            if pb2_db_cnt < DEBOUNCE_TIME then
                pb2_db_cnt <= pb2_db_cnt + 1;
            else
                pb2_db <= '0';
            end if;
        else
            pb2_db_cnt <= 0;
            pb2_db <= '1';
        end if;


        pb1_prev <= pb1_db;
        pb2_prev <= pb2_db;


        if pb1_db = '0' and pb1_prev = '1' then
            if left_val = "1001" then
                left_val <= (others => '0');
            else
                left_val <= left_val + 1;
            end if;
        end if;


        if pb2_db = '0' and pb2_prev = '1' then
            selected_val <= left_val;
            blink_active <= '1';
            blink_count <= 0; --liczba mrugni
            blink_clock_cycles_cnt <= 0;
        end if;


        if blink_active = '1' then
                            if blink_clock_cycles_cnt < 8000000 then
                                    blink_clock_cycles_cnt <= blink_clock_cycles
                            else
                                    blink_clock_cycles_cnt <= 0;
```

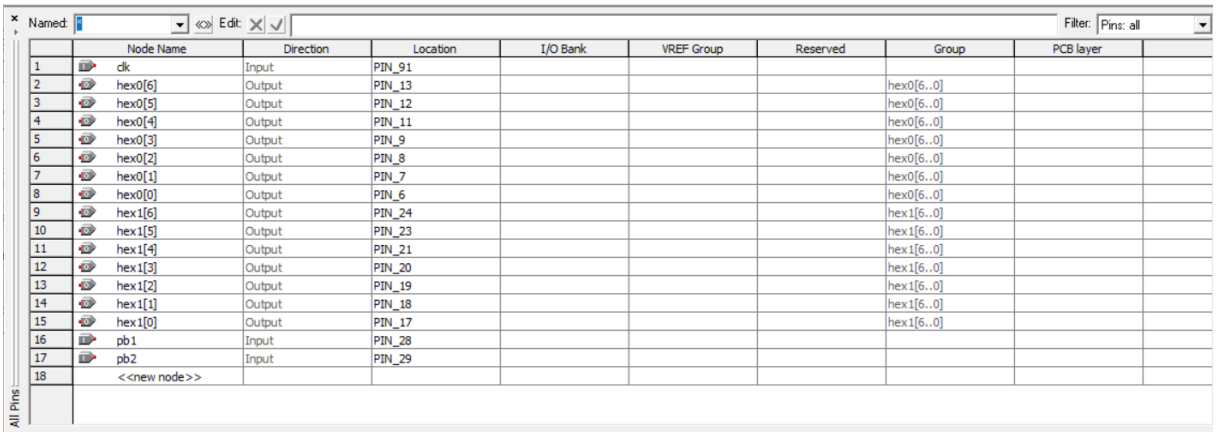
```
                                        blink_toggle <= not blink_toggle;
                                        blink_count <= blink_count + 1;

                                        if blink_count = 5 then
                                                blink_active <= '0';
                                                blink_toggle <= '1';
                                        end if;
                                end if;
                        end if;
        end if;
    end process;

    hex0 <= to_7seg(left_val);
    hex1 <= to_7seg(selected_val) when blink_toggle = '1' else "1111111";
end rtl;
```

5 Kompilacja na płytkę z FPGA

Proponowany sposób mapowania portów na płytkę UP2.



Named: Edit: Filter: Pins: all

	Node Name	Direction	Location	I/O Bank	VREF Group	Reserved	Group	PCB layer
1	clk	Input	PIN_91					
2	hex0[6]	Output	PIN_13				hex0[6..0]	
3	hex0[5]	Output	PIN_12				hex0[6..0]	
4	hex0[4]	Output	PIN_11				hex0[6..0]	
5	hex0[3]	Output	PIN_9				hex0[6..0]	
6	hex0[2]	Output	PIN_8				hex0[6..0]	
7	hex0[1]	Output	PIN_7				hex0[6..0]	
8	hex0[0]	Output	PIN_6				hex0[6..0]	
9	hex1[6]	Output	PIN_24				hex1[6..0]	
10	hex1[5]	Output	PIN_23				hex1[6..0]	
11	hex1[4]	Output	PIN_21				hex1[6..0]	
12	hex1[3]	Output	PIN_20				hex1[6..0]	
13	hex1[2]	Output	PIN_19				hex1[6..0]	
14	hex1[1]	Output	PIN_18				hex1[6..0]	
15	hex1[0]	Output	PIN_17				hex1[6..0]	
16	pb1	Input	PIN_28					
17	pb2	Input	PIN_29					
18	<<new node>>							

All Pins

Rysunek 3: Mapowanie portów w programie Quartus

6 Możliwe ulepszenia

- **Dodanie funkcji zmniejszania wartości przy długim naciśnięciu**
  Obecna implementacja pozwala tylko na zwiększanie wartości. Dodanie możliwości zmniejszania wartości poprzez długie naciśnięcie przycisku poprawi ergonomię użytkowania. Wymaga to modyfikacji stanów i dodania timera wykrywającego długie naciśnięcie (np. $>$1s).

- **Rozszerzenie zakresu wyświetlanych wartości o (A-F)**
  Pozwoliłoby to na wyświetlanie wartości szesnastkowych. Wymagałoby to rozszerzenia funkcji `to_7seg` o nowe znaki, jak i również dodania logiki konwersji. Jeśli chcielibyśmy rozróżnić '0' od 'D' (13) moglibyśmy użyć **Decimal Point** przykładowo dla **'0.'**

7 Zastosowania tego konkretnego układu

- **Zegar cyfrowy z ręcznym ustawianiem czasu:**
  Lewy przycisk służy do ustawiania godziny lub minuty, a prawy przycisk do zatwierdzenia wybranej wartości.

- **System kodu PIN:**
  Naciśnięcia przycisków traktowane są jako wprowadzanie kolejnych cyfr PIN-u. Zatwierdzenie odbywa się za pomocą prawego przycisku.

- **Interfejs użytkownika w sprzęcie AGD:**
  Przykładowo: wybór programu w pralce, mikrofalówce lub suszarce. Realizacja interfejsu bez potrzeby stosowania skomplikowanego GUI.

- **Licznik rowerowy** Przywracanie liczby przejechanych kilometrów po twardym resecie lub obwodu koła poprzez zatwierdzanie kolejnych cyfr.