

Technika Cyfrowa – Sprawozdanie z ćwiczenia 4

Praktyczna implementacja projektów cyfrowych w rzeczywistym programowalnym układzie scalonym FPGA

Autorzy: Robert Raniszewski, Kacper Feliks, Paweł Czajczyk, Mateusz Pawliczek

1. Treść 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:

<https://home.agh.edu.pl/~dlugopol/tc2025/fpga-menu.mp4>

2. Układy FPGA

FPGA to programowalne układy cyfrowe, które umożliwiają tworzenie własnych struktur logicznych bez potrzeby projektowania dedykowanego układu scalonego.

Układ FPGA składa się z tysięcy konfigurowalnych bloków logicznych, układów dodających i przerzutników typu D. Programowanie FPGA odbywa się najczęściej w językach opisu sprzętu, takich jak VHDL lub Verilog.

W przeciwieństwie do tradycyjnych mikroprocesorów czy mikrokontrolerów, FPGA nie wykonują kodu sekwencyjnie i posiadają większą swobodę działania w architekturze.

3. Praktyczne zastosowania FPGA

Układy FPGA znajdują szerokie zastosowanie w wielu dziedzinach, m.in.:

- **Elektronika przemysłowa:** sterowniki maszyn,
- **Telekomunikacja:** kodowanie i dekodowanie sygnałów, szybkie przetwarzanie strumieni danych,
- **Motoryzacja:** systemy wspomagania kierowcy (ADAS), czujniki i układy przetwarzające sygnały w czasie rzeczywistym,
- **Medycyna:** obrazowanie medyczne, szybkie przetwarzanie sygnałów z czujników,
- **Obrona i lotnictwo:** systemy radarowe, komunikacja szyfrowana, analiza sygnałów w czasie rzeczywistym,

4. Rozwiązanie zadania

Do rozwiązania zadania wykorzystaliśmy układ FPGA Altera FLEX EPF10K70RC240-4 oraz program Altera Quartus II ver. 9.0.

4.1 Wersja podstawowa

System został zrealizowany w środowisku Quartus na układzie FPGA i wykorzystuje dwa przyciski oraz dwa wyświetlacze siedmiosegmentowe.

Po uruchomieniu oba wyświetlacze pokazują 0. Wciśnięcie **lewego przycisku** zwiększa wartość na **lewym wyświetlaczu** (od 0 do 9, potem od nowa). Wciśnięcie **prawego przycisku** powoduje **przepisanie** tej wartości na **prawy wyświetlacz**, co oznacza zatwierdzenie wyboru.

4.2 Dodatkowe modyfikacje

Do podstawowego działania systemu dodaliśmy kilka funkcji:

1. **Auto-repeat** – długie przytrzymanie przycisku powoduje automatyczne, powtarzające się zwiększanie (lewy przycisk) lub zmniejszanie (prawy przycisk) wartości. Częstotliwość powtórzeń rośnie z czasem – po około 2 sekundach przytrzymania interwał zmniejsza się aż do 0,5 sekundy.
2. **Odejmovanie przy prawym przycisku** – przyciśnięcie **prawego przycisku** powoduje zmniejszenie wartości na lewym wyświetlaczu. Jeśli osiągnie 0, przechodzi na 9 (cyklicznie w dół).
3. **Zapis obu przyciskami** – jeśli oba przyciski zostaną wciśnięte jednocześnie, wartość z lewego wyświetlacza zostaje zapisana na prawy, niezależnie od wcześniejszego działania.

Wszystkie przyciski są objęte **mechanizmem debouncingu**, aby wyeliminować fałszywe przełączenia wywołane drganiami styków.

5. Kod projektu

W osobnych plikach zdefiniowaliśmy obsługę debouncingu oraz konwersję liczb na wyświetlacz siedmiosegmentowy.

Debouncing:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.NUMERIC_STD.ALL;
4
5  entity debounce is
6  Port (
7      clk      : in  STD_LOGIC;
8      btn_in   : in  STD_LOGIC;
9      btn_out  : out STD_LOGIC
10 );
11 end debounce;
12
13 architecture Behavioral of debounce is
14     -- 20 ms przy clk 25 MHz = 20ms / 40ns = 500000 cykli
15     constant DEBOUNCE_LIMIT : unsigned(18 downto 0) := to_unsigned(500000, 19);
16     signal counter : unsigned(18 downto 0) := (others => '0');
17     signal btn_sync : STD_LOGIC := '0';
18 begin
19     process(clk)
20     begin
21         if rising_edge(clk) then
22             if btn_in = '1' then
23                 if counter < DEBOUNCE_LIMIT then
24                     counter <= counter + 1;
25                 end if;
26
27                 if counter = DEBOUNCE_LIMIT then
28                     btn_sync <= '1';
29                 end if;
30             else
31                 counter <= (others => '0');
32                 btn_sync <= '0';
33             end if;
34         end if;
35     end process;
36
37     btn_out <= btn_sync;
38 end Behavioral;
```

seg7_decoder:

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity seg7_decoder is
5  Port (
6      bcd : in  STD_LOGIC_VECTOR(3 downto 0);
7      seg : out STD_LOGIC_VECTOR(6 downto 0)
8  );
9  end seg7_decoder;
10
11 architecture Behavioral of seg7_decoder is
12 begin
13     process(bcd)
14     begin
15         case bcd is
16             when "0000" => seg <= "1000000"; -- 0
17             when "0001" => seg <= "1111001"; -- 1
18             when "0010" => seg <= "0100100"; -- 2
19             when "0011" => seg <= "0110000"; -- 3
20             when "0100" => seg <= "0011001"; -- 4
21             when "0101" => seg <= "0010010"; -- 5
22             when "0110" => seg <= "0000010"; -- 6
23             when "0111" => seg <= "1111000"; -- 7
24             when "1000" => seg <= "0000000"; -- 8
25             when "1001" => seg <= "0010000"; -- 9
26             when others => seg <= "1111111";
27         end case;
28     end process;
29 end Behavioral;
30
31
32
```

Wersja podstawowa projektu:

```
1  -- menu_top.vhd (bez debounce dla uproszczenia)
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  entity menu_top is
8  Port (
9      clk      : in  STD_LOGIC;
10     btn_l     : in  STD_LOGIC;
11     btn_r     : in  STD_LOGIC;
12     seg_l     : out STD_LOGIC_VECTOR (6 downto 0);
13     seg_r     : out STD_LOGIC_VECTOR (6 downto 0)
14 );
15 end menu_top;
16
17 architecture Behavioral of menu_top is
18     signal cnt_l  : STD_LOGIC_VECTOR(3 downto 0) := "0000";
19     signal cnt_r  : STD_LOGIC_VECTOR(3 downto 0) := "0000";
20     signal btn_l_d, btn_r_d : STD_LOGIC;
21     component seg7_decoder
22     Port (
23         bcd : in  STD_LOGIC_VECTOR(3 downto 0);
24         seg : out STD_LOGIC_VECTOR(6 downto 0)
25     );
26 end component;
27 component debounce is
28 Port (
29     clk      : in  STD_LOGIC;
30     btn_in   : in  STD_LOGIC;
31     btn_out  : out STD_LOGIC
32 );
33 end component;
34 begin
35     -- debounce dla przycisków
36     debounce_l: debounce port map(clk, btn_l, btn_l_d);
37     debounce_r: debounce port map(clk, btn_r, btn_r_d);
38     process(clk)
39     begin
40         if rising_edge(clk) then
41
42             if btn_l_d = '1' then
43                 if cnt_l = "1001" then
44                     cnt_l <= "0000";
45                 else
46                     cnt_l <= cnt_l + 1;
47                 end if;
48             end if;
49
50             if btn_r_d = '1' then
51                 cnt_r <= cnt_l;
52             end if;
53         end process;
54
55         u1: seg7_decoder port map(cnt_l, seg_l);
56         u2: seg7_decoder port map(cnt_r, seg_r);
57     end Behavioral;
```

Wersja zmodyfikowana:

```
1  -- menu_top.vhd
2
3  library IEEE;
4  use IEEE.STD_LOGIC_1164.ALL;
5  use IEEE.STD_LOGIC_UNSIGNED.ALL;
6
7  entity menu_top is
8  Port (
9      clk      : in  STD_LOGIC;
10     btn_l    : in  STD_LOGIC;
11     btn_r    : in  STD_LOGIC;
12     seg_l    : out STD_LOGIC_VECTOR (6 downto 0);
13     seg_r    : out STD_LOGIC_VECTOR (6 downto 0)
14 );
15 end menu_top;
16
17 architecture Behavioral of menu_top is
18     signal cnt_l    : STD_LOGIC_VECTOR(3 downto 0) := "0000";
19     signal cnt_r    : STD_LOGIC_VECTOR(3 downto 0) := "0000";
20     signal l_counter : INTEGER := 0;
21     signal r_counter : INTEGER := 0;
22     signal l_repeat_interval : INTEGER := 1000000;
23     signal r_repeat_interval : INTEGER := 1000000;
24     signal btn_l_d, btn_r_d : STD_LOGIC;
25     signal btn_l_prev, btn_r_prev : STD_LOGIC := '0';
26
27     constant HOLD_THRESHOLD : INTEGER := 50000000;    -- 2 s
28     constant MIN_INTERVAL   : INTEGER := 12500000;    -- 0.5 s
29     constant ACCELERATION_STEP : INTEGER := 6250000;  -- 0.25 s
30
31
32     component seg7_decoder
33     Port (
34         bcd : in  STD_LOGIC_VECTOR(3 downto 0);
35         seg : out STD_LOGIC_VECTOR(6 downto 0)
36     );
37 end component;
38
39     component debounce is
40     Port (
41
42         clk      : in  STD_LOGIC;
43         btn_in   : in  STD_LOGIC;
44         btn_out  : out STD_LOGIC
45     );
46 end component;
47
48 begin
49     debounce_l: debounce port map(clk => clk, btn_in => btn_l, btn_out => btn_l_d);
50     debounce_r: debounce port map(clk => clk, btn_in => btn_r, btn_out => btn_r_d);
51
52     process(clk)
53     begin
54         if rising_edge(clk) then
55
56             -- oba przyciski wcisniete
57             if btn_l_d = '1' and btn_r_d = '1' then
58                 cnt_r <= cnt_l;
59
60             -- LEWY przycisk
61             elsif btn_l_d = '1' then
62                 -- klikni?cie
63                 if btn_l_prev = '0' then
64                     if cnt_l = "1001" then
65                         cnt_l <= "0000";
66                     else
67                         cnt_l <= cnt_l + 1;
68                     end if;
69                 else
70                     -- auto-repeat
71                     l_counter <= l_counter + 1;
72                     if l_counter > l_repeat_interval then
73                         if cnt_l = "1001" then
74                             cnt_l <= "0000";
75                         else
76                             cnt_l <= cnt_l + 1;
77                         end if;
78                     if l_repeat_interval > MIN_INTERVAL then
79                         l_repeat_interval <= l_repeat_interval - ACCELERATION_STEP;
80                     end if;

```




















```

81
82         l_counter <= 0;
83     end if;
84 end if;
85
86 -- PRAWY przycisk
87 elsif btn_r_d = '1' then
88     if btn_r_prev = '0' then
89         if cnt_l = "0000" then
90             cnt_l <= "1001";
91         else
92             cnt_l <= cnt_l - 1;
93         end if;
94     else
95         r_counter <= r_counter + 1;
96         if r_counter > r_repeat_interval then
97             if cnt_l = "0000" then
98                 cnt_l <= "1001";
99             else
100                 cnt_l <= cnt_l - 1;
101             end if;
102
103             if r_repeat_interval > MIN_INTERVAL then
104                 r_repeat_interval <= r_repeat_interval - ACCELERATION_STEP;
105             end if;
106
107             r_counter <= 0;
108         end if;
109     end if;
110
111 -- reset licznikow
112 else
113     r_counter <= 0;
114     r_repeat_interval <= HOLD_THRESHOLD;
115     l_counter <= 0;
116     l_repeat_interval <= HOLD_THRESHOLD;
117 end if;
118 btn_l_prev <= btn_l_d;
119 btn_r_prev <= btn_r_d;
120 end if;
121
122 end process;
123
124 u1: seg7_decoder port map(bcd => cnt_l, seg => seg_l);
125 u2: seg7_decoder port map(bcd => cnt_r, seg => seg_r);
126 end Behavioral;
127
128

```

6. Przypisanie pinów

Node Name	Direction	Location
 btn_l	Input	PIN_28
 btn_r	Input	PIN_29
 clk	Input	PIN_91
 seg_l[6]	Output	PIN_13
 seg_l[5]	Output	PIN_12
 seg_l[4]	Output	PIN_11
 seg_l[3]	Output	PIN_9
 seg_l[2]	Output	PIN_8
 seg_l[1]	Output	PIN_7
 seg_l[0]	Output	PIN_6
 seg_r[6]	Output	PIN_24
 seg_r[5]	Output	PIN_23
 seg_r[4]	Output	PIN_21
 seg_r[3]	Output	PIN_20
 seg_r[2]	Output	PIN_19
 seg_r[1]	Output	PIN_18
 seg_r[0]	Output	PIN_17