Technika cyfrowa – sprawozdanie nr 4

Autorzy: Rozmus Dariusz, Fabia Jakub, Smołka Tomasz, Czyż Bartosz, Zając Kacper

1. TREŚĆ ZADANIA W SKRÓCIE

Korzystając z programu **Quartus** firmy **Altera**, 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. KILKA SŁÓW O FPGA

FPGA to programowalne układy cyfrowe, które użytkownik może konfigurować, aby realizowały konkretne funkcje logiczne. Składają się z matrycy bloków logicznych i programowalnych połączeń, co pozwala na tworzenie złożonych systemów cyfrowych bez konieczności projektowania dodatkowego, specjalistycznego układu scalonego.

Programowanie układów **FPGA** najczęściej odbywa się przy pomocy języków takich jak **VHDL** lub **Verilog**.

W odróżnieniu do tradycyjnych mikroprocesorów, **FPGA** nie wykonuje instrukcji sekwencyjnie, lecz realizuje je równolegle, co pozwala na wykonywanie wielu operacji jednocześnie w szybkim czasie.

3. ZASTOSOWANIA FPGA

- **Telekomunikacja** – szybkie przetwarzanie strumieni danych, przetwarzanie obrazu i wideo w czasie rzeczywistym
- **Lotnictwo** – radary, analiza sygnałów w czasie rzeczywistym
- **Medycyna** - szybkie przetwarzanie sygnałów w sprzęcie medycznym (np. aparaty USG)
- **Motoryzacja** - systemy wizyjne w samochodach (np. asystent pasa ruchu)
- **Automatyka i elektronika** – sterowniki do robotów przemysłowych

4. ROZWIĄZANIE ZADANIA

Rozwiązanie zostało napisane w języku **VHDL**.

4.1 Użyty układ i wersja programu

- Zestaw FPGA Altera UP2 z układem FLEX EPF10K70RC240-4
- Program Altera Quartus II ver. 9.0.

4.2 Wejścia i wyjścia układu

Wejścia:

- Przyciski: FLEX_PB1 i FLEX_PB2
- Przełączniki: FLEX_SWITCH-1 i FLEX_SWITCH-2
- Clock

Wyjścia:

- Wyświetlacze: FLEX_DIGIT1 i FLEX_DIGIT2

4.3 Funkcjonalność z treści zadania

Wciśnięcie lewego przycisku powoduje zwiększenie wartości wskazywanej na lewym wyświetlaczu o jeden (wartości od 0 do 9 z zawijaniem). Wciśnięcie prawego przycisku powoduje zapisanie wartości z lewego wyświetlacza na prawym wyświetlaczu.

4.4 Funkcjonalności dodatkowe

- Przełączenie FLEX_SWITCH-1 powoduje zmniejszanie wartości na lewym wyświetlaczu po wciśnięciu lewego przycisku.
- Przełączenie FLEX_SWITCH-2 pokazuje animację "węża" przesuwającego się po zewnętrznych segmentach wyświetlaczy.

4.5 Pliki projektu

- **counter.vhd**: licznik jednocyfrowy (0–9) z obsługą zawijania wartości
- **int_to_digit.vhd:** mapuje wartość liczbową (0–9) na odpowiednią cyfrę dla wyświetlacza siedmiosegmentowego
- **select.vhd:** odpowiada za przepisanie wartości z lewego wyświetlacza na prawy po wciśnięciu prawego przycisku
- **snake_anim.vhd:** animacja węża

Pomocnicze moduły:

- **one_pulse.vhd:** odpowiada za generowanie pojedynczego impulsu po naciśnięciu przycisku. Używany do obsługi sygnału z przycisku, aby przy każdym naciśnięciu zliczyć tylko jedno zdarzenie
- **debounce.vhd:** filtruje zakłócenia sygnału z przycisku, eliminując fałszywe zliczenia wynikające z drgań styków

Całość projektu zintegrowano w pliku **main.vhd**

5. KOD ŹRÓDŁOWY

Moduł main.vhd

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

entity main is
    port (
        clk             : in  STD_LOGIC;  -- 50 MHz
        btn_counter     : in  STD_LOGIC;  -- FLEX_PB1  (pin 28)
        btn_select      : in  STD_LOGIC;  -- FLEX_PB2  (pin 29)

        switch_down     : in  STD_LOGIC;  -- FLEX_SWITCH-1 (pin 41)
        switch_animate  : in  STD_LOGIC;  -- FLEX_SWITCH-2 (pin 40)

        menu_digit      : out STD_LOGIC_VECTOR(6 downto 0); -- lewy wyswietlacz
        selected_digit  : out STD_LOGIC_VECTOR(6 downto 0); -- prawy wyswietlacz
        menu_dp         : out STD_LOGIC;                    -- kropka (DP)
        selected_dp     : out STD_LOGIC                     -- kropka (DP)
    );
end entity;

architecture rtl of main is
    signal deb_counter   : STD_LOGIC;
    signal deb_select    : STD_LOGIC;
    signal pulse_counter : STD_LOGIC;
    signal pulse_select  : STD_LOGIC;

    signal count    : INTEGER range 0 to 9;
    signal selected : INTEGER range 0 to 9;

    signal menu_norm : STD_LOGIC_VECTOR(6 downto 0);
    signal sel_norm  : STD_LOGIC_VECTOR(6 downto 0);

    signal anim_left  : STD_LOGIC_VECTOR(6 downto 0);
    signal anim_right : STD_LOGIC_VECTOR(6 downto 0);

    signal dir_count : STD_LOGIC;
    signal anim_en   : STD_LOGIC;
begin
    deb1: entity work.debounce
        generic map (CNT_MAX => 250_000)          -- ~5 ms przy 50 MHz
        port map (
            clk     => clk,
            btn_in  => btn_counter,
            btn_out => deb_counter
        );

    deb2: entity work.debounce
        generic map (CNT_MAX => 250_000)
        port map (
            clk     => clk,
            btn_in  => btn_select,
            btn_out => deb_select
        );

    op1: entity work.one_pulse
        port map (
            clk       => clk,
            btn_in    => deb_counter,
            pulse_out => pulse_counter
        );
```

```
    op2: entity work.one_pulse
        port map (
            clk       => clk,
            btn_in    => deb_select,
            pulse_out => pulse_select
        );

    dir_count <= not switch_down;
    anim_en   <= not switch_animate;

    cnt0: entity work.counter
        port map (
            clk => clk,
            en  => pulse_counter,
            dir => dir_count,
            num => count
        );

    sel0: entity work.selector
        port map (
            clk     => pulse_select,
            num_in  => count,
            num_out => selected
        );

    to7seg1: entity work.int_to_digit
        port map (
            num   => count,
            digit => menu_norm
        );

    to7seg2: entity work.int_to_digit
        port map (
            num   => selected,
            digit => sel_norm
        );

    anim0: entity work.snake_anim
        generic map (DIV_MAX => 5_000_000)
        port map (
            clk         => clk,
            enable      => anim_en,
            left_digit  => anim_left,
            right_digit => anim_right
        );

    menu_digit     <= anim_left  when anim_en = '1' else menu_norm;
    selected_digit <= anim_right when anim_en = '1' else sel_norm;

    menu_dp     <= '1';   -- aktywne-LOW: '1' = zgaszona
    selected_dp <= '1';   -- aktywne-LOW: '1' = zgaszona
end rtl;
```

Moduł counter.vhd

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

entity counter is
    port (
        clk  : in  STD_LOGIC;
        en   : in  STD_LOGIC;
        dir  : in  STD_LOGIC;
        num  : out INTEGER range 0 to 9
    );
end counter;

architecture rtl of counter is
    signal value : INTEGER range 0 to 9 := 0;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if en = '1' then
                if dir = '1' then
                    if value = 0 then
                        value <= 9;
                    else
                        value <= value - 1;
                    end if;
                else
                    if value = 9 then
                        value <= 0;
                    else
                        value <= value + 1;
                    end if;
                end if;
            end if;
        end if;
    end process;

    num <= value;
end rtl;
```

Moduł int_to_digit.vhd

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

entity int_to_digit is
    port(
        num : in integer range 0 to 9;
        digit : out STD_LOGIC_VECTOR(6 downto 0)
    );
end int_to_digit;

architecture a of int_to_digit is
begin
    -- Assuming common cathode 7-segment display
    with num select
        digit <=
            "1000000" when 0,  -- 0
            "1111001" when 1,  -- 1
            "0100100" when 2,  -- 2
            "0110000" when 3,  -- 3
            "0011001" when 4,  -- 4
            "0010010" when 5,  -- 5
            "0000010" when 6,  -- 6
            "1111000" when 7,  -- 7
            "0000000" when 8,  -- 8
            "0010000" when 9,  -- 9
            "1111111" when others;
end a;
```

Moduł select.vhd

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

entity selector is
    port(
        clk : in STD_LOGIC;
        num_in: in integer range 0 to 9;
        num_out : out integer range 0 to 9
    );
end selector;

architecture a of selector is
    signal curr : integer range 0 to 9 := 0;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            curr <= num_in;
        end if;
    end process;

    num_out <= curr;
end a;
```

Moduł one_pulse.vhd

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

entity one_pulse is
    port (
        clk      : in  STD_LOGIC;
        btn_in   : in  STD_LOGIC;
        pulse_out: out STD_LOGIC
    );
end entity;

architecture rtl of one_pulse is
    signal btn_reg : STD_LOGIC := '0';
begin
    process(clk)
    begin
        if rising_edge(clk) then
            pulse_out <= btn_in and not btn_reg;
            btn_reg   <= btn_in;
        end if;
    end process;
end architecture;
```

Moduł debounce.vhd

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

entity debounce is
    generic (
        CNT_MAX : integer := 50000
    );
    port (
        clk     : in  STD_LOGIC;
        btn_in  : in  STD_LOGIC;
        btn_out : out STD_LOGIC
    );
end entity;

architecture rtl of debounce is
    signal btn_sync  : STD_LOGIC_VECTOR(1 downto 0) := (others => '0');
    signal cnt       : integer := 0;
    signal btn_state : STD_LOGIC := '0';
begin
    -- Synchronize button to clk domain
    process(clk)
    begin
        if rising_edge(clk) then
            btn_sync(0) <= btn_in;
            btn_sync(1) <= btn_sync(0);
        end if;
    end process;

    process(clk)
    begin
        if rising_edge(clk) then
            if btn_sync(1) /= btn_state then
                cnt <= cnt + 1;
                if cnt > CNT_MAX then
                    btn_state <= btn_sync(1);
                    cnt <= 0;
                end if;
            else
                cnt <= 0;
            end if;
        end if;
    end process;

    btn_out <= btn_state;
end architecture;
```

Moduł snake_anim.vhd

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

entity snake_anim is
    generic (
        DIV_MAX : natural := 5_000_000
    );
    port (
        clk         : in  STD_LOGIC;
        enable      : in  STD_LOGIC;             -- switch_animate
        left_digit  : out STD_LOGIC_VECTOR(6 downto 0);  -- g f e d c b a, active-low
        right_digit : out STD_LOGIC_VECTOR(6 downto 0)
    );
end snake_anim;

architecture rtl of snake_anim is
    constant OFF : STD_LOGIC_VECTOR(6 downto 0) := (others => '1');


    constant SEG_A : STD_LOGIC_VECTOR(6 downto 0) := "1111110";
    constant SEG_B : STD_LOGIC_VECTOR(6 downto 0) := "1111101";
    constant SEG_C : STD_LOGIC_VECTOR(6 downto 0) := "1111011";
    constant SEG_D : STD_LOGIC_VECTOR(6 downto 0) := "1110111";
    constant SEG_E : STD_LOGIC_VECTOR(6 downto 0) := "1101111";
    constant SEG_F : STD_LOGIC_VECTOR(6 downto 0) := "1011111";

    type pat_arr is array (0 to 7) of STD_LOGIC_VECTOR(6 downto 0);

    constant L_PAT : pat_arr := (
        SEG_D,  -- 0
        SEG_E,  -- 1
        SEG_F,  -- 2
        SEG_A,  -- 3
        OFF,    -- 4
        OFF,    -- 5
        OFF,    -- 6
        OFF     -- 7
    );

    constant R_PAT : pat_arr := (
        OFF,    -- 0
        OFF,    -- 1
        OFF,    -- 2
        OFF,    -- 3
        SEG_A,  -- 4
        SEG_B,  -- 5
        SEG_C,  -- 6
        SEG_D   -- 7
    );

    signal frame   : integer range 0 to 7 := 0;
    signal div_cnt : natural range 0 to DIV_MAX-1 := 0;
begin
    process(clk)
    begin
        if rising_edge(clk) then
            if enable = '1' then
                if div_cnt = DIV_MAX-1 then
                    div_cnt <= 0;
                    frame   <= (frame + 1) mod 8;
                else
                    div_cnt <= div_cnt + 1;
                end if;
            else
                div_cnt <= 0;
                frame   <= 0;
            end if;
        end if;
    end process;

    left_digit  <= L_PAT(frame);
    right_digit <= R_PAT(frame);
end rtl;
```