

Maciej Byczko Bartosz Matysiak	Prowadzący: dr inż. Jacek Mazurkiewicz	Numer ćwiczenia 7
PN 10:50 TP	Temat ćwiczenia: Licznik synchroniczny sterowany - FPGA	Ocena:
Grupa: B	Data wykonania: 10 Stycznia 2022r.	

Spis treści

1	Polecenie	2
2	Rozwiązanie	2
2.1	Kod VHDL	3
2.2	Kod VHDL TestBench	4
2.3	Symulacja	5
2.4	Schemat układu z wykorzystaniem zaprojektowanego modułu	6
3	Fizyczna implementacja	6
3.1	Kod UCF	6
4	Wnioski	7

1 Polecenie

Licznik synchroniczny rewersyjny 8-bitowy pracujący w kodzie naturalnym binarnym. Wartość inicjująca licznik ma być ładowana z klawiatury komputera PC poprzez uruchomiony na nim terminal. Można także użyć klawiatury PS/2 - uwaga na inne wartości podawane przez klawiaturę - kody skaningowe - oraz inny moduł wejściowy do obsługi portu PS/2.

Oznacza to, że do przystawki dotrze kod naciśniętego klawisza poprzez port szeregowy RS232 lub port PS/2 i ten właśnie kod ma inicjować licznik. Licznik po przyjęciu kodu zaczyna liczyć - grupa wybiera czy będzie zwiększał swój stan - będzie początkowo - pozytywny, czy też zmniejszał swój stan - będzie początkowo - negatywny.

Bieżący stan licznika ma być wyświetlany na wyświetlaczu LCD w dowolnej, ale jednoznacznej i komunikatywnej formie. W dowolnym momencie pracy licznika możemy zmieniać kierunek zliczania wybranym guzikiem z przystawki. Może to być jeden guzik - przełącznik góra/dół, mogą być użyte dwa guziki - jeden włącza zliczanie w górę, drugi - zliczanie w dół.

Realizacja zadania wymaga zatem napisania w VHDL-u własnego modułu, który będzie realizował działanie licznika oraz spięcie tego modułu z gotowymi modułami obsługi urządzeń wejścia/wyjścia przystawki: odbiornika portu RS232 lub portu PS/2 oraz obsługi wyświetlacza LCD stanowiącego integralną część przystawki Spartan FPGA.

2 Rozwiązanie

Projekt licznika się nie zmienia względem poprzednich zajęć, więc dla przypomnienia:

aby wykonać w pełni działający licznik wiemy że potrzebujemy następujące wejścia/wyjścia:

- Wejście na parametry podane z klawiatury
- Zegar na podstawie którego wywołamy kolejny stan
- Reset za pomocą którego będziemy informować licznik że chcemy wprowadzić nową wartość
- Kontrolę w jaki sposób licznik będzie liczyć (do przodu/do tyłu)

Licznik bazuje na 8 bitach więc w momencie przepełnienia ustawiliśmy że licznik wraca do wartości:

- Zerowej gdy liczy do przodu
- Maksymalnej gdy liczy do tyłu

Jedyna różnica w kodzie jest taka, że Spartan posiada zegar ze znacznie większą częstotliwością, więc na podstawie wykonania działania:

$$1MHz = 10^6 Hz \rightarrow 50MHz = 5 * 10^7 Hz$$

$$\log_2(5 * 10^7 Hz) \approx 25.57$$

$$\frac{5 * 10^7 Hz}{2^{25}} \approx 1.49 Hz$$

Dzięki tym obliczeniom wiemy, że musimy zastosować 25 bitowy dzielnik aby uzyskać częstotliwość w przybliżeniu 1Hz.

2.1 Kod VHDL

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use ieee.std_logic_unsigned.all;
4 use IEEE.STD_LOGIC_ARITH.ALL;
5 use ieee.numeric_std.all;
6
7 entity mainModule is
8     Port ( REVERSE : in  STD_LOGIC;
9           CLK_LF   : in  STD_LOGIC;
10          RESET    : in  STD_LOGIC;
11          WEJ      : in  STD_LOGIC_VECTOR (7 downto 0);
12          WYJ      : out STD_LOGIC_VECTOR (7 downto 0));
13 end mainModule;
14
15 architecture Behavioral of mainModule is
16
17     signal current_state: STD_LOGIC_VECTOR(7 downto 0);
18     signal splitter: STD_LOGIC_VECTOR(24 downto 0);
19
20     begin
21
22         process1: process(CLK_LF, RESET, REVERSE, WEJ)
23             begin
24                 — gdy wcisniety reset, pobranie wartosci z klawiatury
25                 if RESET = '1' then
26                     current_state <= WEJ;
27                 elsif rising_edge(CLK_LF) then
28                     splitter <= splitter + 1;
29                     if splitter = "10000000000000000000000000" then
30                         splitter <= "00000000000000000000000000";
31                     if REVERSE = '0' then
32                         if current_state = "11111111" then
33                             current_state <= "00000000";
34                         else
35                             current_state <= current_state + 1;
36                         end if;
37                     else
38                         if current_state = "00000000" then
39                             current_state <= "11111111";
40                         else
41                             current_state <= current_state - 1;
42                         end if;
43                     end if;
44                 end if;
45             end if;
46
47         end process process1;
48
49         WYJ <= current_state;
```

```

50
51 end Behavioral;

```

2.2 Kod VHDL TestBench

```

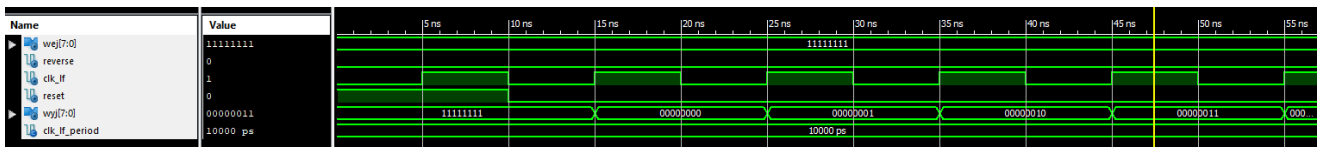
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3
4  ENTITY counter_testbench IS
5  END counter_testbench;
6
7  ARCHITECTURE behavior OF counter_testbench IS
8
9      — Component Declaration for the Unit Under Test (UUT)
10     COMPONENT counter_mod
11     PORT(
12         WEJ : IN  std_logic_vector(7 downto 0);
13         REVERSE : IN  std_logic;
14         CLK_LF : IN  std_logic;
15         RESET : IN  std_logic;
16         WYJ : OUT std_logic_vector(7 downto 0)
17     );
18     END COMPONENT;
19
20
21     —Inputs
22     signal WEJ : std_logic_vector(7 downto 0) := (others => '0');
23     signal REVERSE : std_logic := '0';
24     signal CLK_LF : std_logic := '0';
25     signal RESET : std_logic := '0';
26
27     —Outputs
28     signal WYJ : std_logic_vector(7 downto 0);
29
30     — Clock period definitions
31     constant CLK_LF_period : time := 10 ns;
32
33 BEGIN
34
35     — Instantiate the Unit Under Test (UUT)
36     uut: counter_mod PORT MAP (
37         WEJ => WEJ,
38         REVERSE => REVERSE,
39         CLK_LF => CLK_LF,
40         RESET => RESET,
41         WYJ => WYJ
42     );
43
44     — Clock process definitions
45     CLK_LF_process : process
46     begin
47         CLK_LF <= '0';

```

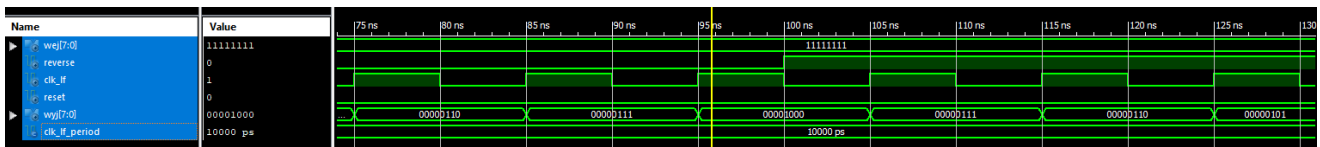
```
48     wait for CLK_LF_period/2;
49     CLK_LF <= '1';
50     wait for CLK_LF_period/2;
51 end process;
52
53 — Stimulus process
54 stim_proc: process
55 begin
56
57     WEJ <= "11111111";
58     RESET <= '1', '0' after 10 ns;
59     REVERSE <= '0', '1' after 100 ns;
60
61     wait;
62 end process;
63
64 END;
```

2.3 Symulacja

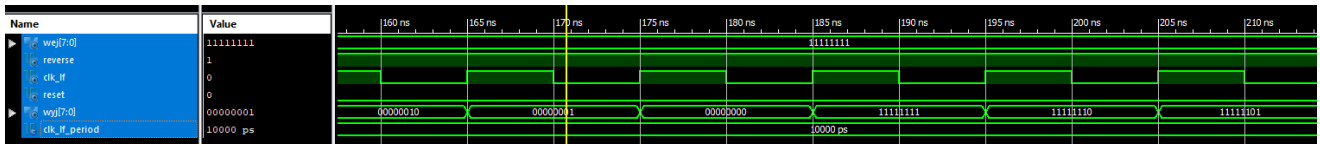
Rysunek 1: Początek symulacji



Rysunek 2: Wprowadzenie nowej wartości

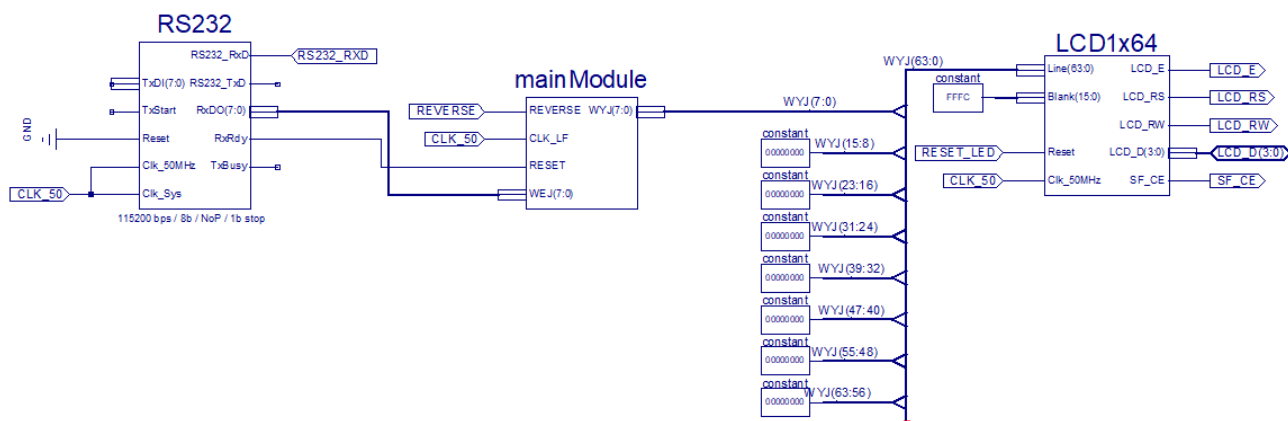


Rysunek 3: Odwrócenie kolejności odliczania



2.4 Schemat układu z wykorzystaniem zaprojektowanego modułu

Rysunek 4: Schemat z podłączoną klawiaturą oraz wyświetlaczem LED



3 Fizyczna implementacja

3.1 Kod UCF

Normalnie Kod byłby w dwóch plikach: GenIO.ucf oraz LDC.ucf lecz w celu poprawienia czytelności kody zostały umieszczone w jednym bloku

```

1 # soldered 50MHz Clock.
2 NET "Clk_50" LOC = "C9" | IOSTANDARD = LVTTTL;
3 NET "Clk_50" PERIOD = 20.0ns HIGH 50%;
4
5
6 # Slide switches (Up = Hi)
7 NET "REVERSE" LOC = "L13" | IOSTANDARD = LVTTTL | PULLUP;
8
9 # RS-232 Serial Port: DCE
10 NET "RS232_RXD" LOC = "R7" | IOSTANDARD = LVTTTL ;
11
12
13 # ===== Character LCD (LCD) =====
14 NET "LCD_E" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW =
    SLOW ;
15 NET "LCD_RS" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW =
    SLOW ;
16 NET "LCD_RW" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW =
    SLOW ;
17 # LCD data connections are shared with StrataFlash connections SF_D
    <11:8>
18 NET "LCD_D<0>" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW
    = SLOW ;
19 NET "LCD_D<1>" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW
    = SLOW ;
20 NET "LCD_D<2>" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW
    = SLOW ;

```

```
21 NET "LCD_D<3>" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW  
    = SLOW ;  
22 NET "SF_CE" LOC = "D16" | IOSTANDARD = LVCMOS33 | DRIVE = 4 | SLEW =  
    SLOW ;
```

4 Wnioski

Niestety przez zajęcia zdalne nie mieliśmy możliwości przetestowania zaprojektowanego układu.