Laboratorium Elektroniki Cyfrowej		
Ćwiczenie nr: 6 Temat zajęć: **Pomiar częstotliwości**		Data wykonania: **06.05.2018** Data uruchomienia: **10.05.2018**
Kierunek/semestr: **AiR / 4**	Grupa: **CZW_1145**	
Wykonali: **Katarzyna Kowalska 132079, Eryk Miśkiewicz 132100**		

Zadanie A:

1. Cel zadania / wymagania projektowe

- Wykonanie układu wyznaczającego częstotliwość zadawaną przez moduł zewnętrzny.

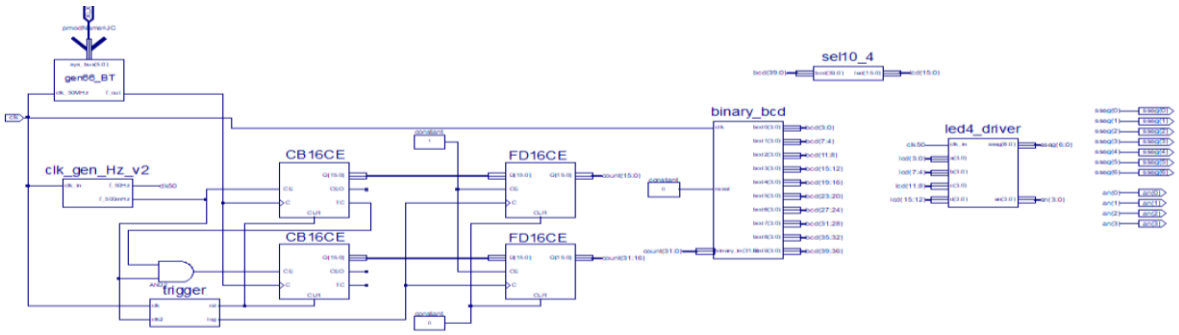
2. Idea działania:

- Na dwa liczniki 16-bitowe połączone kaskadowo podawany jest mierzony sygnał jako sygnał zegarowy. Wykorzystanie licznika 32-bitowego pozwala na zmierzenie częstotliwości 2^32 razy większej od częstotliwości pomiaru.
- Blok clk_gen_Hz_v2 (Kod bloku clk_gen_Hz_v2) generuje sygnał o częstotliwości 50Hz wykorzystywany do sterowania wyświetlaczem 7-seg, oraz sygnał 0,5Hz, który przez dokładnie 1 sekundę podaje stan wysoki na wejście CE liczników. Pozwala to zmierzyć ilość impulsów badanego sygnału w ciągu sekundy.
- Blok trigger (Kod bloku trigger) wykorzystywany jest do sterowania przepisaniem wartości z liczników i resetowaniem liczników.
- Blok binary_bcd (Kod bloku binary_bcd) konwertuje wartość odczytaną z liczników na 10 znaków BCD.
- W module sel10_4(Kod bloku sel10_4) automatycznie wybrany zostaje zakres pomiaru. Na wyjściu przedstawione zostają 4 znaki, 3 zawierające najbardziej znaczące liczby zmierzonej częstotliwości i 4-ty znak informujący o potędze 10, przez którą trzeba pomnożyć odczytany wynik.

3. Dodatkowe informacje:

- Symulacja (Symulacja uwzględniająca moment krytyczny) zawiera symulację złożenia bloków do schematu zbiorczego i reprezentacje zachowania układu w momencie krytycznym.
- Wykorzystanie portów i pinout (Wykorzystywane porty i pinout)

4. Schemat zbiorczy:

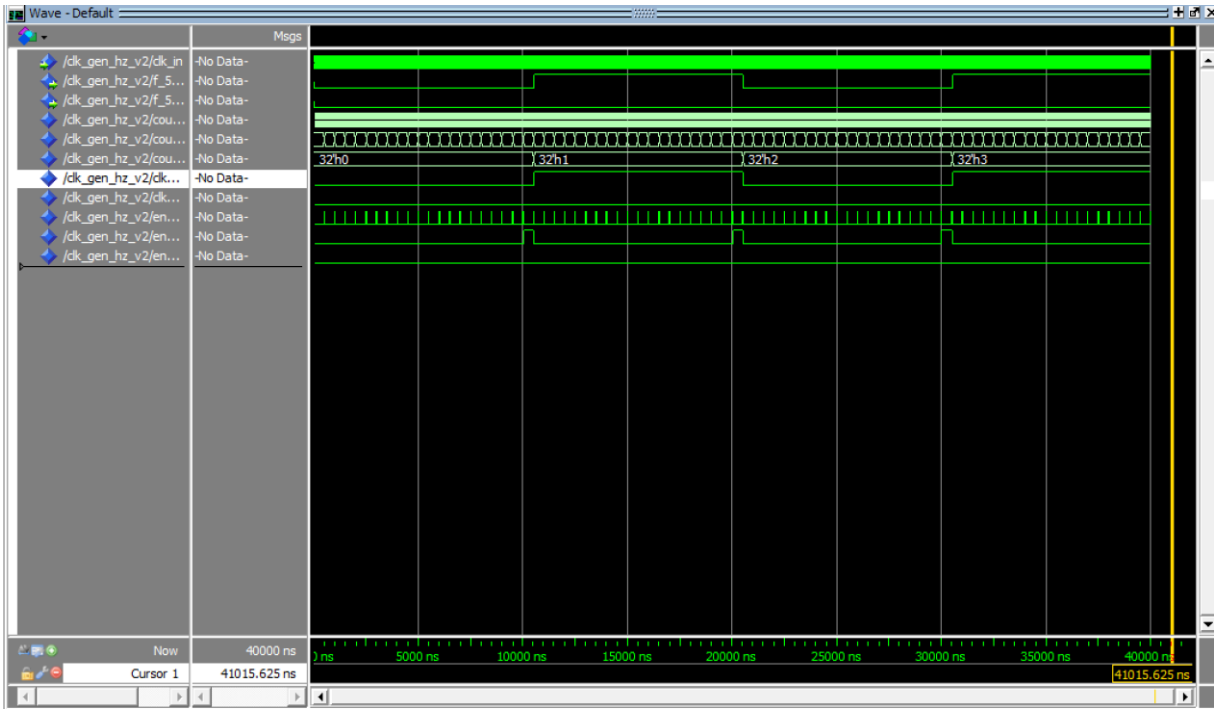


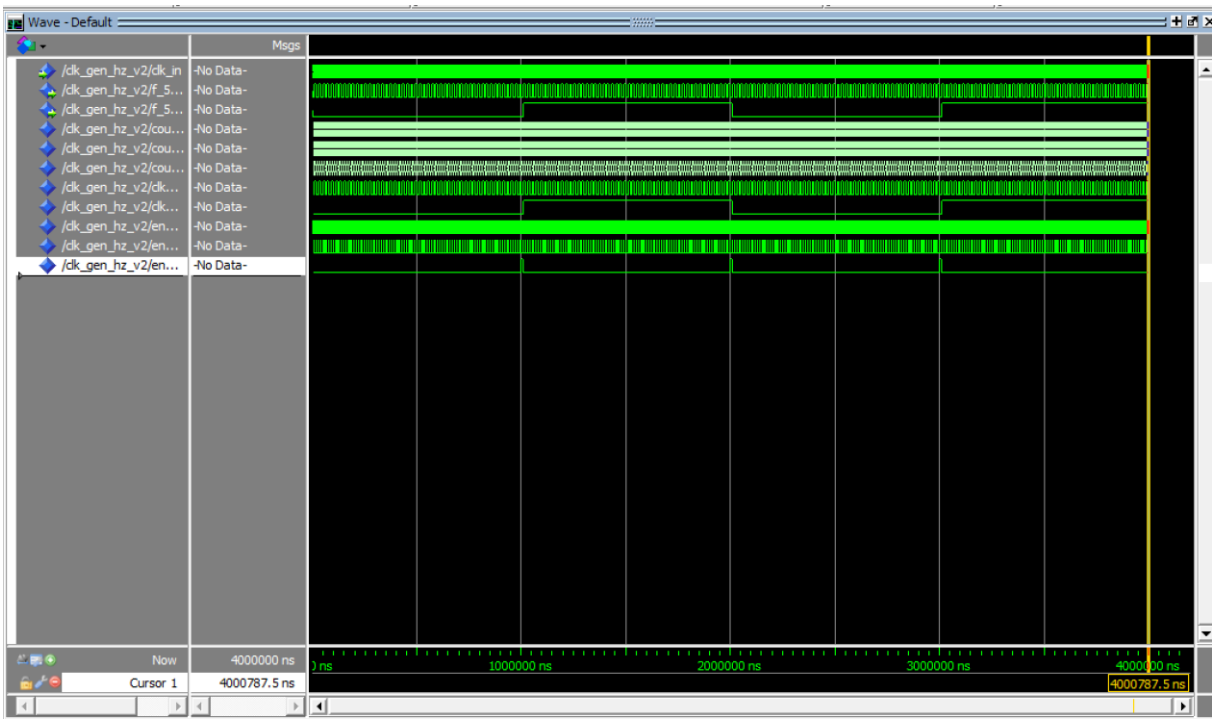
5. Kod bloku clk_gen_Hz_v2

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity clk_gen_Hz_v2 is
    Generic (Fclk : natural := 50); -- in MHz
    Port (  clk_in : in  STD_LOGIC;
          f_50Hz, f_500mHz : out  STD_LOGIC);
end clk_gen_Hz_v2;

architecture Behavioral of clk_gen_Hz_v2 is
constant N_period: natural:=Fclk/2;
signal count_MHz : integer range 0 to N_period:=0;
signal count_50Hz : integer range 0 to 20000 :=0;
signal count_500mHz : integer range 0 to 100 :=0;
signal clk_50Hz, clk_500mHz: std_logic:='0';
signal en_MHz, en_50Hz, en_500mHz: std_logic:='0';

begin

timer_MHz: process(clk_in)
                begin
                        if rising_edge(clk_in) then
                                if count_MHz=(N_period-1) then
                                        count_MHz<= 0;
                                        en_MHz <='1';
                                else
                                        count_MHz<=count_MHz+1;
                                        en_MHz <='0';
                                end if;
                        end if;
                end process timer_MHz;

timer_50Hz: process(clk_in)
  begin
    if rising_edge(clk_in) and en_MHz='1' then
      if count_50Hz=(20000 -1) then
          count_50Hz <= 0;
          en_50Hz <= '1';
      else
          count_50Hz <= count_50Hz + 1;
          en_50Hz <= '0';
      end if;
    end if;
  end process timer_50Hz;

timer_500mHz: process(clk_in)
  begin
    if rising_edge(clk_in) and en_50Hz='1' and en_MHz='1' then
      if count_500mHz=(100-1) then
          count_500mHz <= 0;
          en_500mHz <= '1';
      else
          count_500mHz <= count_500mHz + 1;
          en_500mHz <= '0';
      end if;
    end if;
  end process timer_500mHz;

f_50Hz <=clk_50Hz;
f_500mHz <=clk_500mHz;
x1_50pc: process(clk_in)
  begin
    if clk_in'event and clk_in='1' and en_50Hz='1' and en_MHz='1' then
          clk_50Hz <= not clk_50Hz;
    end if;
  end process;
x2_50pc: process(clk_in)
  begin
    if clk_in'event and clk_in='1' and en_50Hz='1' and en_500mHz='1' and en_MHz='1' then
          clk_500mHz <= not clk_500mHz;
    end if;
  end process;
  end Behavioral;
```

6. *Symulacja bloku clk_gen_Hz_v2*

- Na potrzeby symulacji częstotliwość generowanego sygnału została zmniejszona 1000-krotnie.
- Pomiary zostały wykonane w okresach 40us i 4ms co odpowiada okresom rzeczywistym 40ms i 4s.
- Do symulacji zostały wykorzystane następne wymuszenia:
    - force -freeze sim:/clk_gen_hz_v2/clk_in 1 0, 0 {10000 ps} -r 20ns
- Wyniki symulacji:



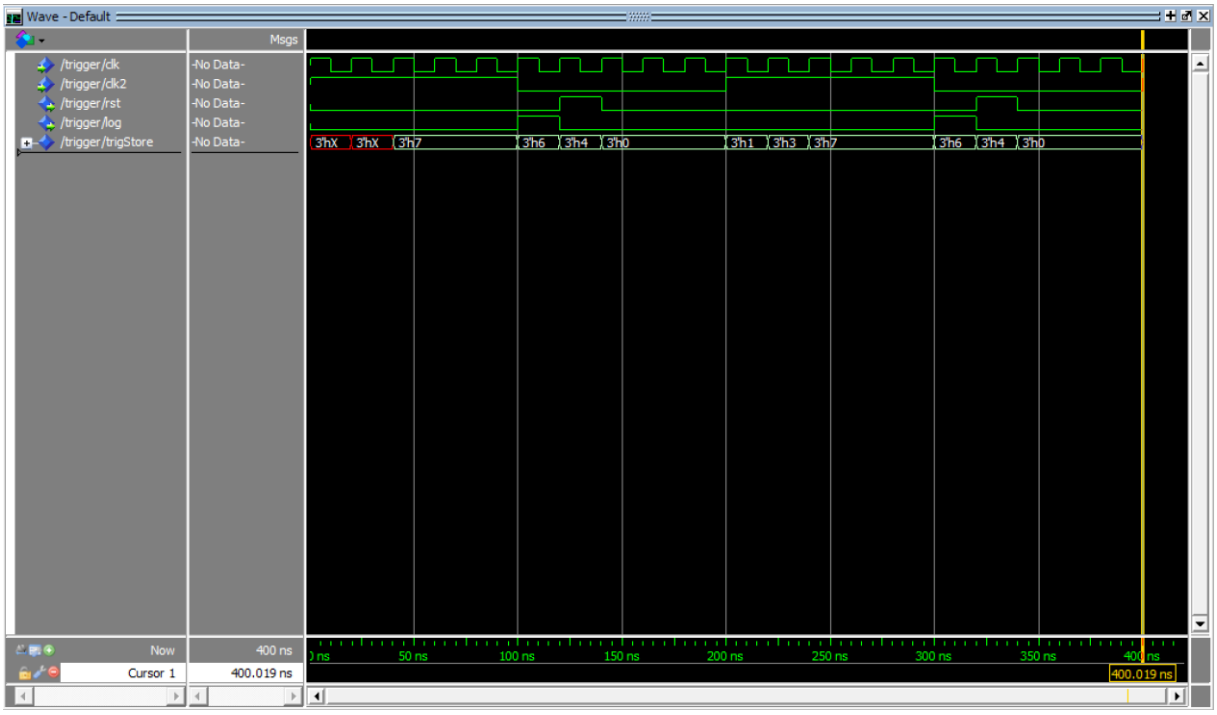


7. *Kod bloku trigger*

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity trigger is
  port(clk, clk2: in std_logic;
       rst,log : out std_logic);
end trigger;

architecture BehavTrigger of trigger is
  signal trigStore: std_logic_vector(2 downto 0);

begin
 process (clk)
 begin
  if clk'event and clk='1' then
    trigStore(0) <= clk2;
    trigStore(1) <= trigStore(0);
    trigStore(2) <= trigStore(1);
  end if;
 end process;

log <= '1' when (trigStore = "110") else '0';
rst <= '1' when (trigStore = "100") else '0';

end BehavTrigger;
```

8. *Symulacja bloku trigger*

- Przedstawiona symulacja obrazuje działanie bloku po wykryciu zbocza opadającego na badanym sygnale.
- Do symulacji zostały wykorzystane następne wymuszenia:
  - force -freeze sim:/trigger/clk 1 0, 0 {10000 ps} -r 20ns
  - force -freeze sim:/trigger/clk2 1 0, 0 {100000 ps} -r 200ns
- Wyniki symulacji:



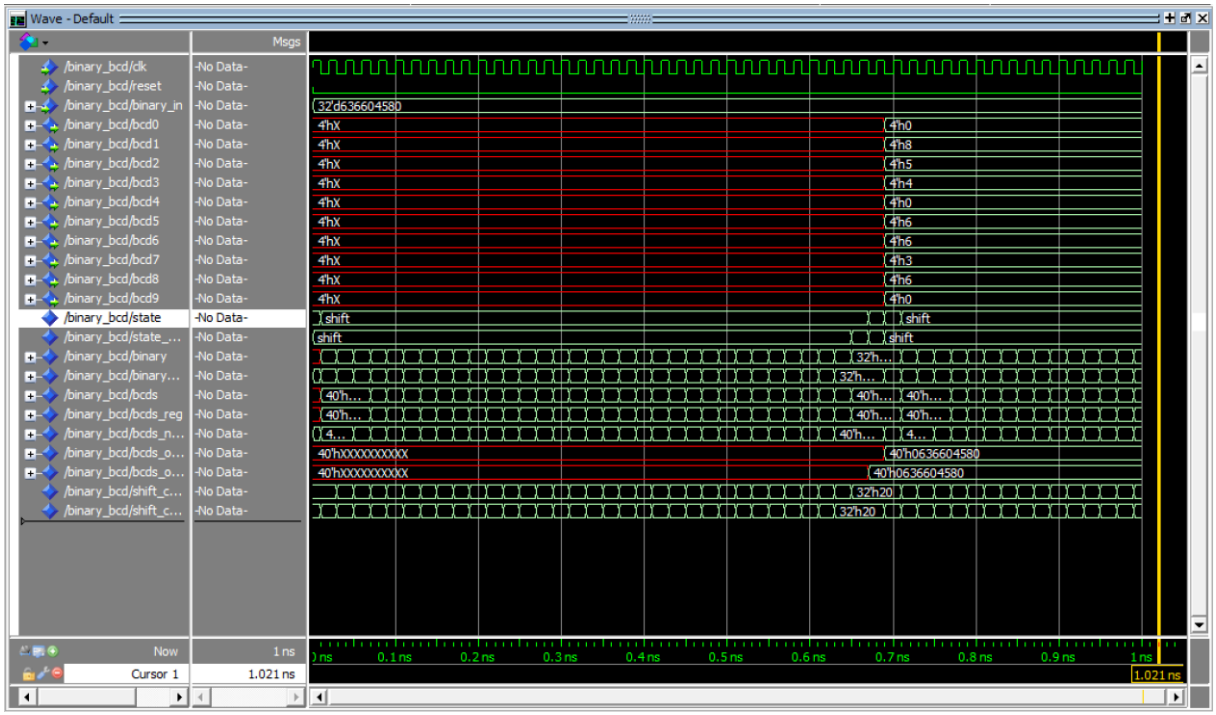
9. Kod bloku binary_bcd

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

entity binary_bcd is
   generic(N: positive := 32);
   port(
      clk, reset: in std_logic;
      binary_in: in std_logic_vector(N-1 downto 0);
      bcd0, bcd1, bcd2, bcd3, bcd4, bcd5, bcd6, bcd7, bcd8, bcd9: out std_logic_vector(3 downto 0)
   );
end binary_bcd ;

architecture behaviour of binary_bcd is
   type states is (start, shift, done);
   signal state, state_next: states;

   signal binary, binary_next: std_logic_vector(N-1 downto 0);
   signal bcds, bcds_reg, bcds_next: std_logic_vector(39 downto 0);
   -- output register keep output constant during conversion
   signal bcds_out_reg, bcds_out_reg_next: std_logic_vector(39 downto 0);
   -- need to keep track of shifts
   signal shift_counter, shift_counter_next: natural range 0 to N;
begin

   process(clk, reset)
   begin
      if reset = '1' then
         binary <= (others => '0');
         bcds <= (others => '0');
         state <= start;
         bcds_out_reg <= (others => '0');
         shift_counter <= 0;
      elsif falling_edge(clk) then
         binary <= binary_next;
         bcds <= bcds_next;
         state <= state_next;
         bcds_out_reg <= bcds_out_reg_next;
         shift_counter <= shift_counter_next;
      end if;
   end process;

   convert:
   process(state, binary, binary_in, bcds, bcds_reg, shift_counter)
   begin
      state_next <= state;
      bcds_next <= bcds;
      binary_next <= binary;
      shift_counter_next <= shift_counter;

      case state is
         when start =>
            state_next <= shift;
            binary_next <= binary_in;
            bcds_next <= (others => '0');
            shift_counter_next <= 0;
         when shift =>
            if shift_counter = N then
               state_next <= done;
            else
               binary_next <= binary(N-2 downto 0) & 'L';
               bcds_next <= bcds_reg(38 downto 0) & binary(N-1);
               shift_counter_next <= shift_counter + 1;
            end if;
         when done =>
```

```
            state_next <= start;
        end case;
    end process;

    bcds_reg(39 downto 36) <= bcds(39 downto 36) + 3 when bcds(39 downto 36) > 4 else
                              bcds(39 downto 36);
    bcds_reg(35 downto 32) <= bcds(35 downto 32) + 3 when bcds(35 downto 32) > 4 else
                              bcds(35 downto 32);
    bcds_reg(31 downto 28) <= bcds(31 downto 28) + 3 when bcds(31 downto 28) > 4 else
                              bcds(31 downto 28);
    bcds_reg(27 downto 24) <= bcds(27 downto 24) + 3 when bcds(27 downto 24) > 4 else
                              bcds(27 downto 24);
    bcds_reg(23 downto 20) <= bcds(23 downto 20) + 3 when bcds(23 downto 20) > 4 else
                              bcds(23 downto 20);
    bcds_reg(19 downto 16) <= bcds(19 downto 16) + 3 when bcds(19 downto 16) > 4 else
                              bcds(19 downto 16);
    bcds_reg(15 downto 12) <= bcds(15 downto 12) + 3 when bcds(15 downto 12) > 4 else
                              bcds(15 downto 12);
    bcds_reg(11 downto 8) <= bcds(11 downto 8) + 3 when bcds(11 downto 8) > 4 else
                             bcds(11 downto 8);
    bcds_reg(7 downto 4) <= bcds(7 downto 4) + 3 when bcds(7 downto 4) > 4 else
                            bcds(7 downto 4);
    bcds_reg(3 downto 0) <= bcds(3 downto 0) + 3 when bcds(3 downto 0) > 4 else
                            bcds(3 downto 0);

    bcds_out_reg_next <= bcds when state = done else
                         bcds_out_reg;

    bcd9 <= bcds_out_reg(39 downto 36);
            bcd8 <= bcds_out_reg(35 downto 32);
            bcd7 <= bcds_out_reg(31 downto 28);
    bcd6 <= bcds_out_reg(27 downto 24);
    bcd5 <= bcds_out_reg(23 downto 20);
    bcd4 <= bcds_out_reg(19 downto 16);
    bcd3 <= bcds_out_reg(15 downto 12);
    bcd2 <= bcds_out_reg(11 downto 8);
    bcd1 <= bcds_out_reg(7 downto 4);
    bcd0 <= bcds_out_reg(3 downto 0);

end behaviour;
```

10. Symulacja bloku binary_bcd

- Przedstawiona symulacja obrazuje działanie bloku dla losowo wybranej liczby binarnej zapisanej na 32-bitach.
- Wyniki symulacji:



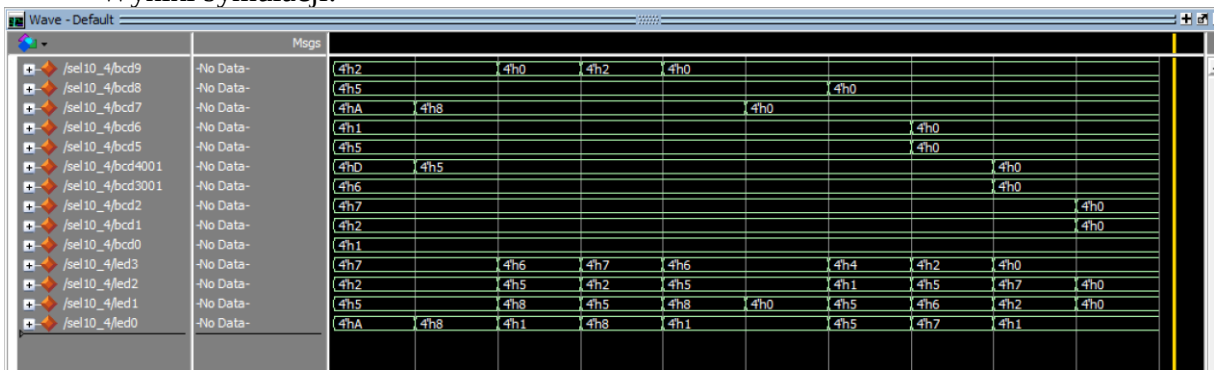
11. Kod bloku sel10_4

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:39:00 05/05/2018
-- Design Name:
-- Module Name:    sel10_4 - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity sel10_4 is
--Port ( bcd0, bcd1, bcd2, bcd3, bcd4, bcd5, bcd6, bcd7, bcd8, bcd9 : in  STD_LOGIC_VECTOR (3 downto 0);       -- 0-lsb
--       led0, led1, led2, led3 : out  STD_LOGIC_VECTOR (3 downto 0)
--        );
Port ( bcd: in  std_logic_vector(39 downto 0);       -- 0-lsb
       led: out  std_logic_vector(15 downto 0));
end sel10_4;

architecture Behavi of sel10_4 is

begin
            process(bcd)
            begin
            if bcd(39 downto 36)="0000" then
                        if bcd(35 downto 32)="0000" then
                                    if bcd(31 downto 28)="0000" then
                                                if bcd(27 downto 24)="0000" then
                                                            if bcd(23 downto 20)="0000" then
                                                                        if bcd(19 downto 16)="0000" then
                                                                                    if bcd(15 downto 12)="0000" then
                                                                                                led(11 downto 0) <= bcd(11 downto 0);
                                                                                                led(15 downto 12)<="0000";
                                                                                    else
                                                                                                led(11 downto 0) <= bcd(15 downto 4);
                                                                                                led(15 downto 12)<="0001";
                                                                                    end if;
                                                                        else
                                                                                    led(11 downto 0) <= bcd(19 downto 8);
                                                                                    led(15 downto 12)<="0010";
                                                                        end if;
                                                            else
                                                                        led(11 downto 0) <= bcd(23 downto 12);
                                                                        led(15 downto 12)<="0011";
                                                            end if;
```

```
                                        else
                                                led(11 downto 0) <= bcd(27 downto 16);
                                                led(15 downto 12)<="0100";
                                        end if;
                                else
                                        led(11 downto 0) <= bcd(31 downto 20);
                                        led(15 downto 12)<="0101";
                                end if;
                        else
                                led(11 downto 0) <= bcd(35 downto 24);
                                led(15 downto 12)<="0110";
                        end if;
                else
                        led(11 downto 0) <= bcd(39 downto 28);
                        led(15 downto 12)<="0111";
                end if;
                end process;
end Behavi;
```

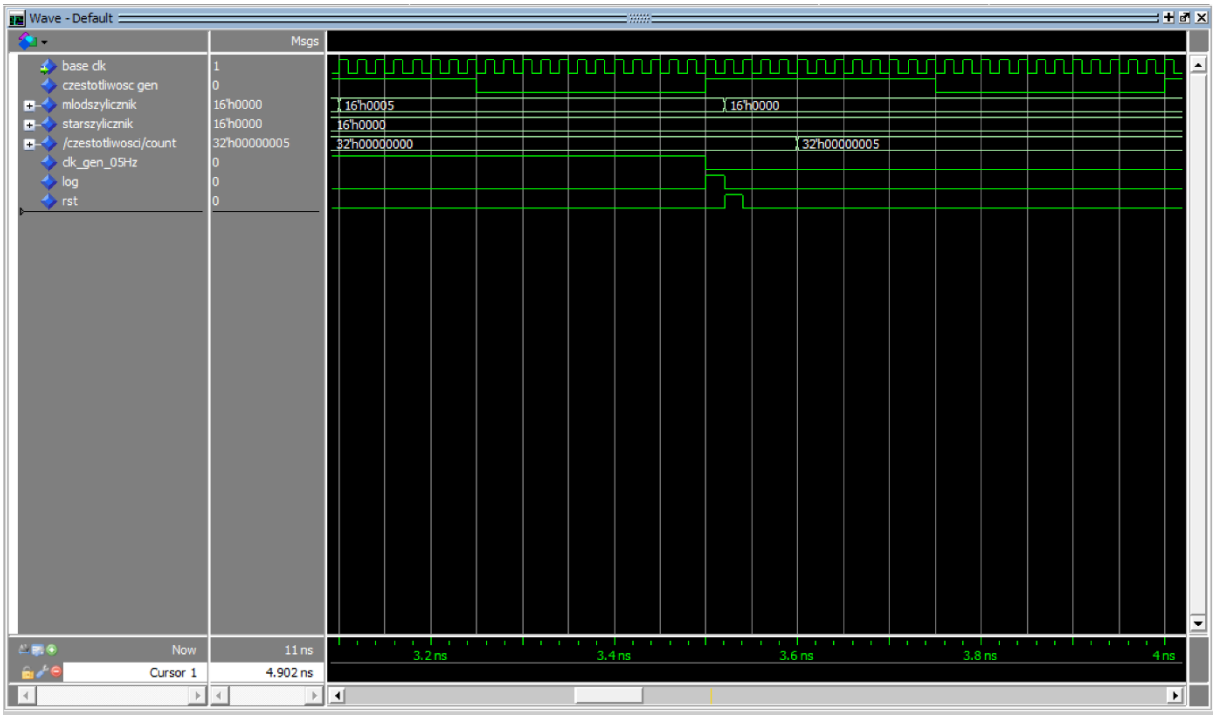
12. Symulacja bloku sel10_4

- Przedstawiona symulacja obrazuje działanie bloku.
- Na wyjściach led2, led1, led0 przepisywane są 3 najbardziej znaczące wartości
- Na wyjściu led3 zapisywana jest skala przepisanych wartości.
- Wyniki symulacji:



13. Symulacja uwzględniająca moment krytyczny

- Dla wykonanego układu przeprowadzona została symulacja krytycznego momentu tj. zerowanie liczników i nadpisanie pamięci.
- Poprzez wykorzystanie układów synchronicznych gwarantowane jest zachowane kolejności wykonywanych akcji.



14. Wykorzystywane porty i pinout

- Interfejs testowanego urządzenia (wg schematu):

Port urządzenia testowanego	Sygnał płyty prototypowej
Sygnał bluetooth	PmodJC
CLK	Zegar 50MHz
sseg(0)	CA
sseg(1)	CB
sseg(2)	CC
sseg(3)	CD
sseg(4)	CE
sseg(5)	CF
sseg(6)	CG
an(0)	AN0
an(1)	AN1
an(2)	AN2
an(3)	AN3

- Testowanie polega na zadawaniu częstotliwości przez terminal bluetooth i odczytywaniu wyników z wyświetlacza 7-seg.
- Pinout Report:

	Pin Number	Signal Name	Pin Usage	Pin Name	Direction	IO Standard	IO Bank Number	Drive (mA)	Slew Rate	Termination	IOB Delay	Voltage	Constraint	IO Register	S Int
1	J12	XLXN_117<0>	IOB	IO_L15P_1/A2	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
2	G16	XLXN_117<1>	IBUF	IO_L18N_1	INPUT	LVCMOS...	1				NONE		LOCATED	NO	NO
3	H15	XLXN_117<2>	IOB	IO_L17N_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
4	F14	XLXN_117<3>	IOB	IO_L21N_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
5	G13	XLXN_117<4>	IBUF	IO_L20N_1	INPUT	LVCMOS...	1				NONE		LOCATED	NO	NO
6	H16	XLXN_117<5>	IOB	IO_L16P_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
7	F17	an<0>	IOB	IO_L19N_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
8	H17	an<1>	IOB	IO_L16N_1/A0	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
9	C18	an<2>	IOB	IO_L24P_1/LDC1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
10	F15	an<3>	IOB	IO_L21P_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
11	B8	clk	IBUF	IP_L13P_0/GCLK8	INPUT	LVCMOS...	0				NONE		LOCATED	NO	NO
12	L18	sseg<0>	IOB	IO_L10P_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
13	F18	sseg<1>	IOB	IO_L19P_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
14	D17	sseg<2>	IOB	IO_L23P_1/HDC	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
15	D16	sseg<3>	IOB	IO_L23N_1/LDC0	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
16	G14	sseg<4>	IOB	IO_L20P_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
17	J17	sseg<5>	IOB	IO_L13P_1/A6/RHCLK4/IR...	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO
18	H14	sseg<6>	IOB	IO_L17P_1	OUTPUT	LVCMOS...	1	12	SL...	NONE**			LOCATED	NO	NO