|  |  |  |
| --- | --- | --- |
| L a b o r a t o r i u m E l e k t r o n i k i C y f r o w e j | | |
| 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.

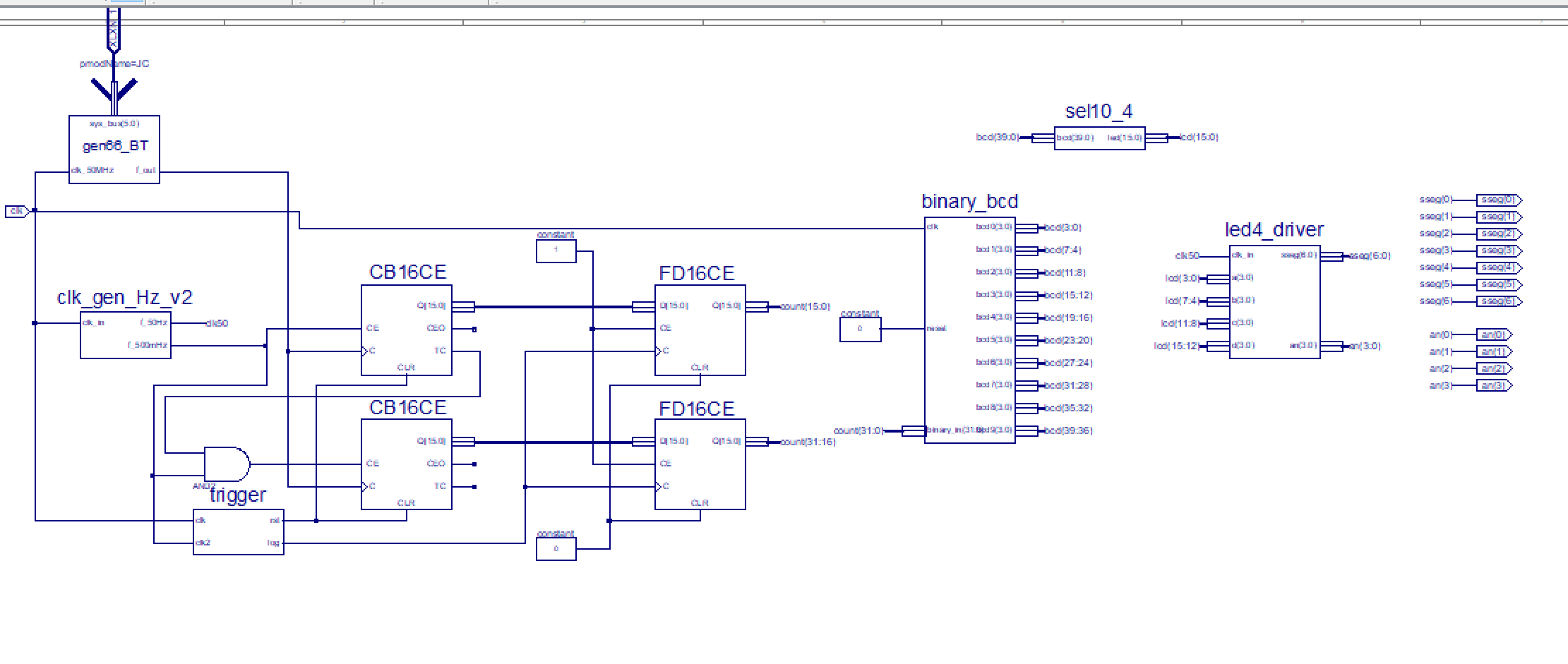
1. ***Idea działania:***

* 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.

1. ***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)

1. ***Schemat zbiorczy:***



1. ***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;

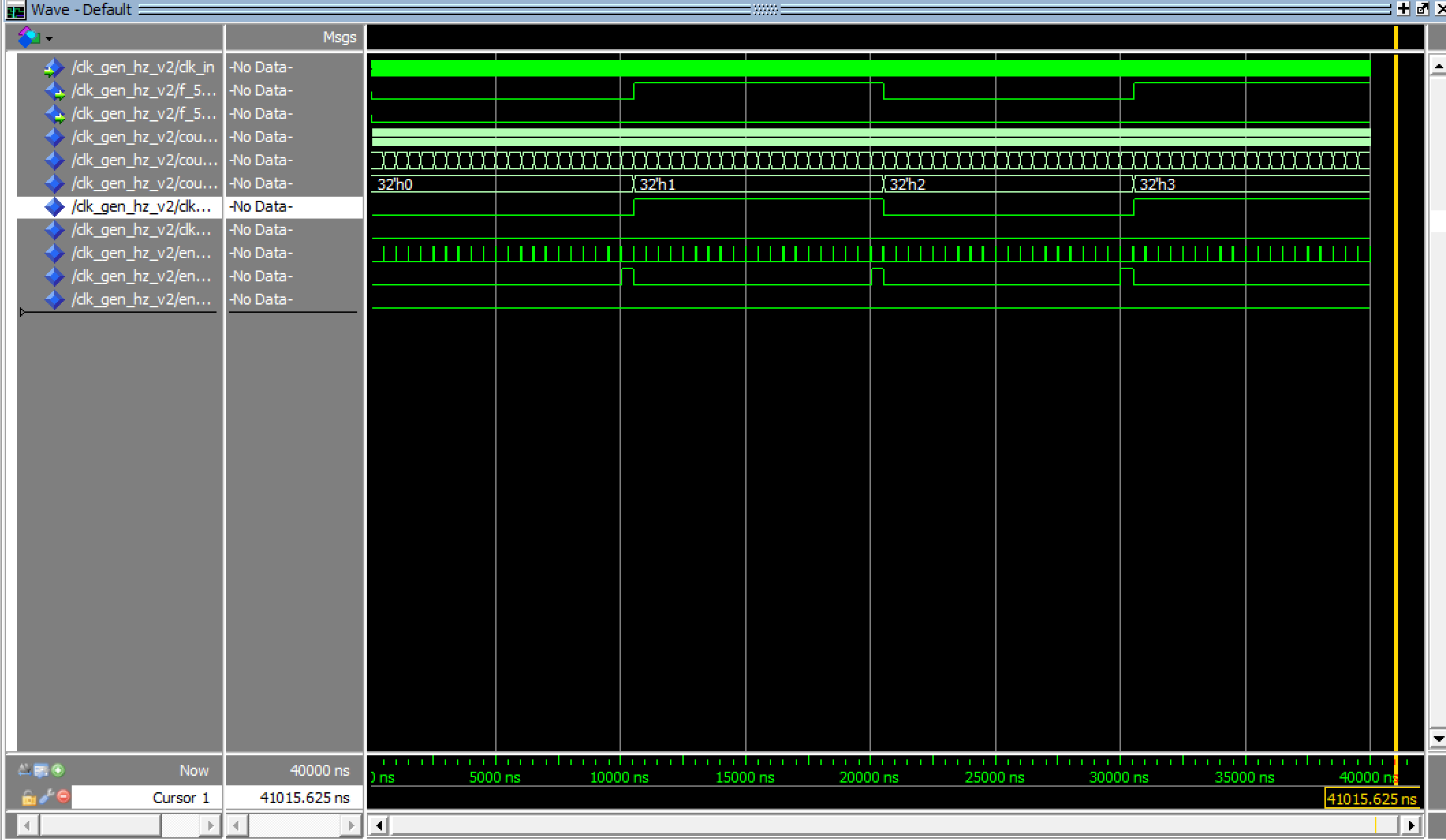
end if;

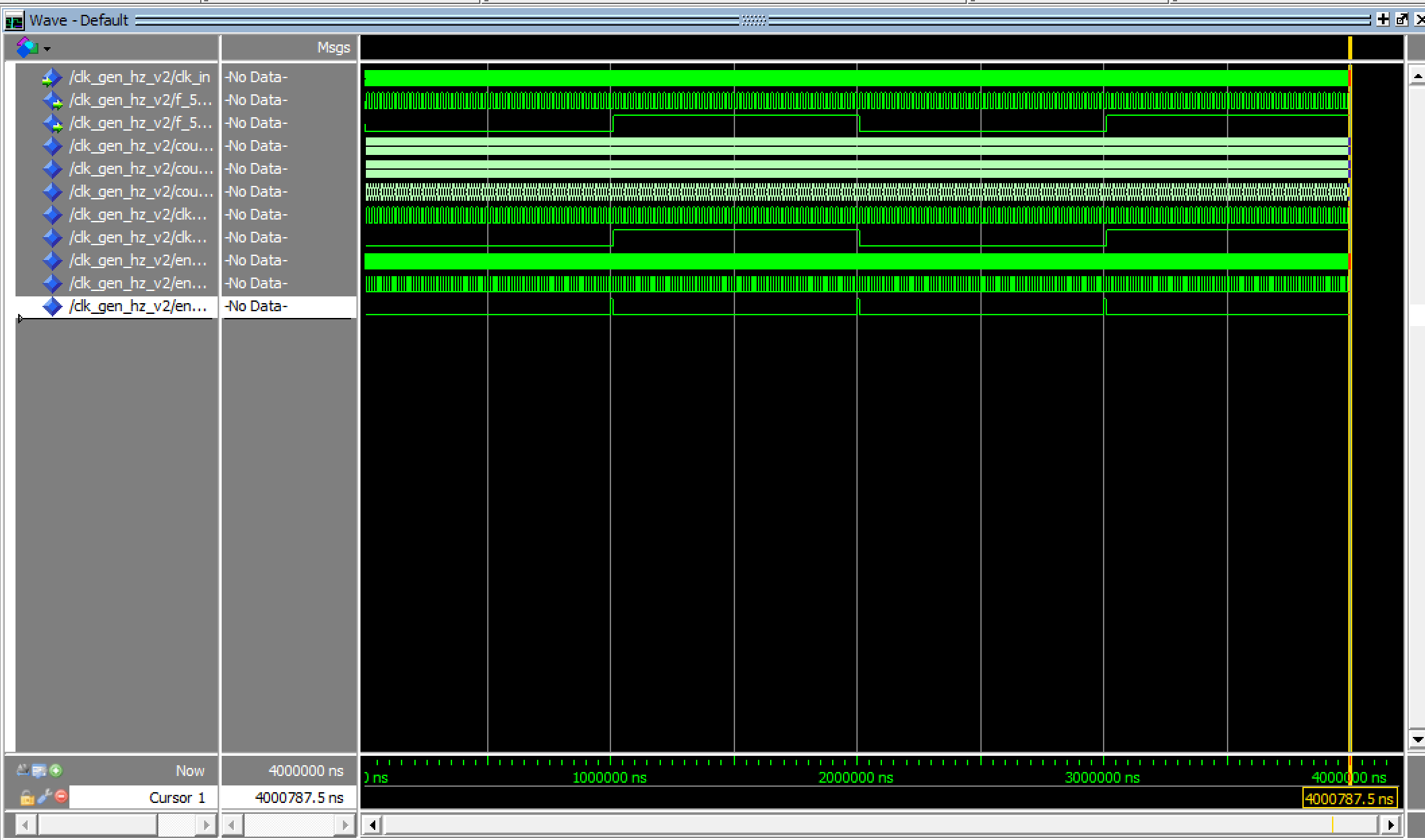
end process;

end Behavioral;

1. ***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:





1. ***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;

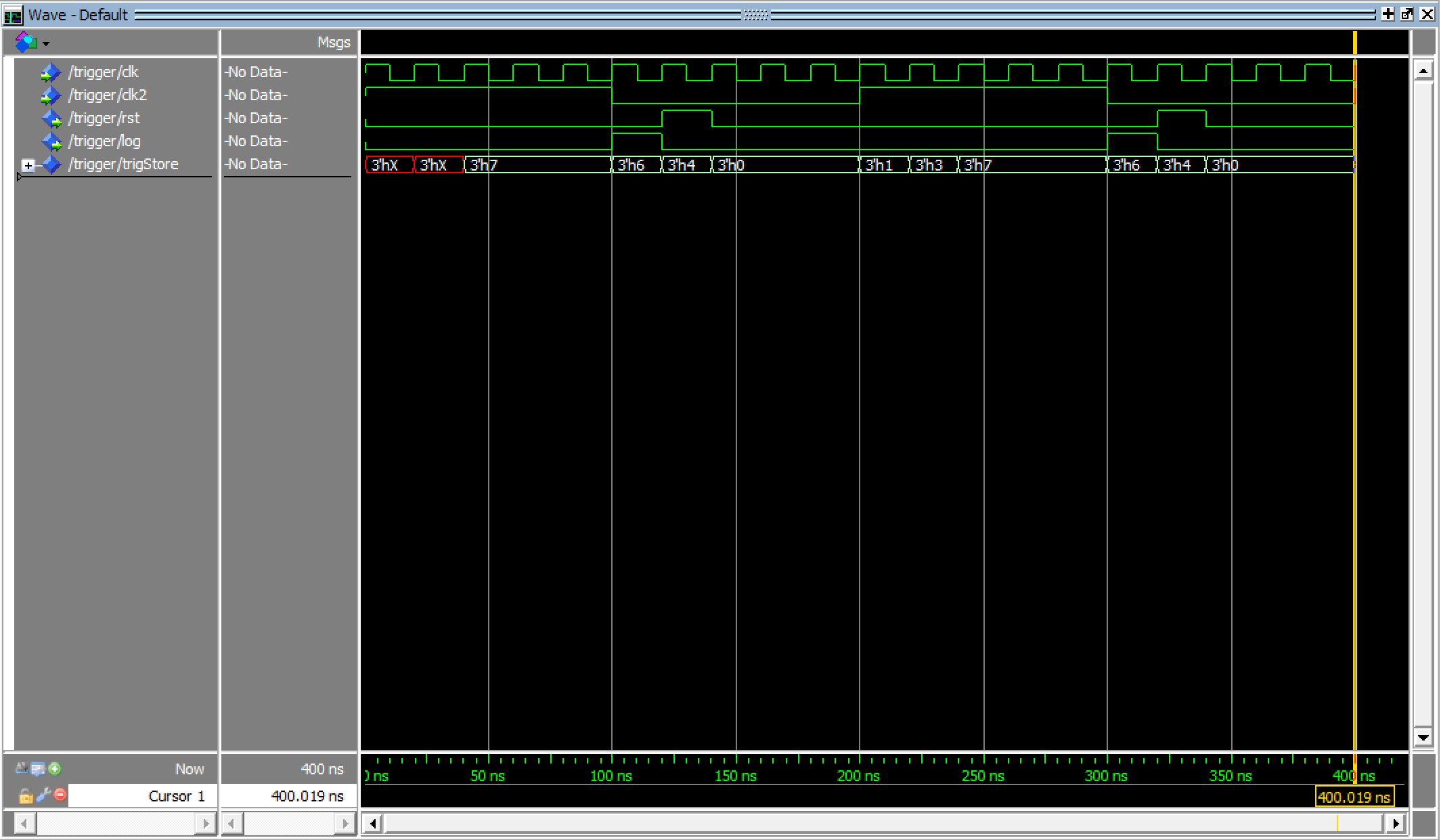
log <= '1' when (trigStore = "110") else '0';

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

end BehavTrigger;

1. ***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:



1. ***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);*

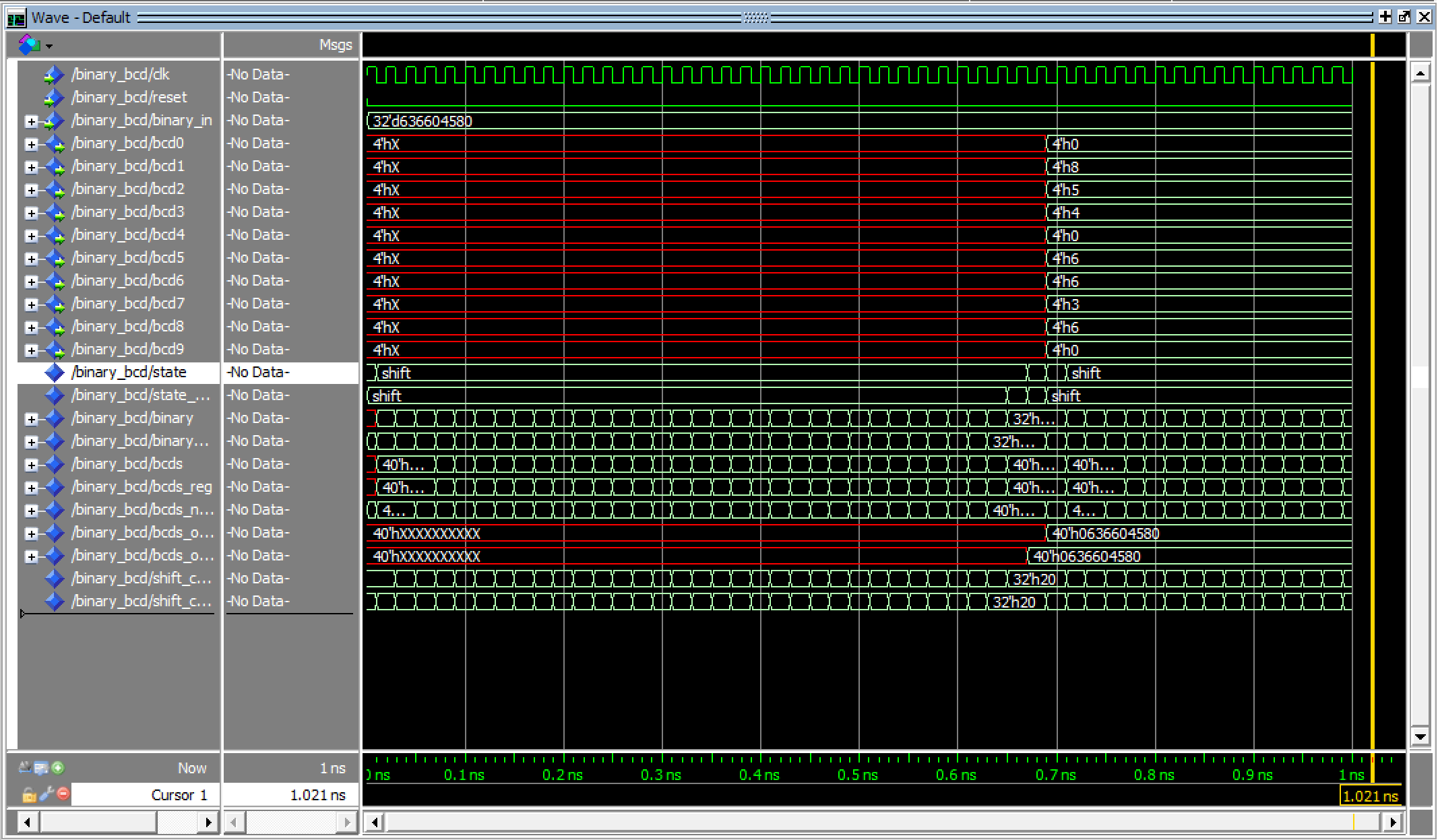
*bcd1 <= bcds\_out\_reg(7 downto 4);*

*bcd0 <= bcds\_out\_reg(3 downto 0);*

*end behaviour;*

1. ***Symulacja bloku binary\_bcd***

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



1. ***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";

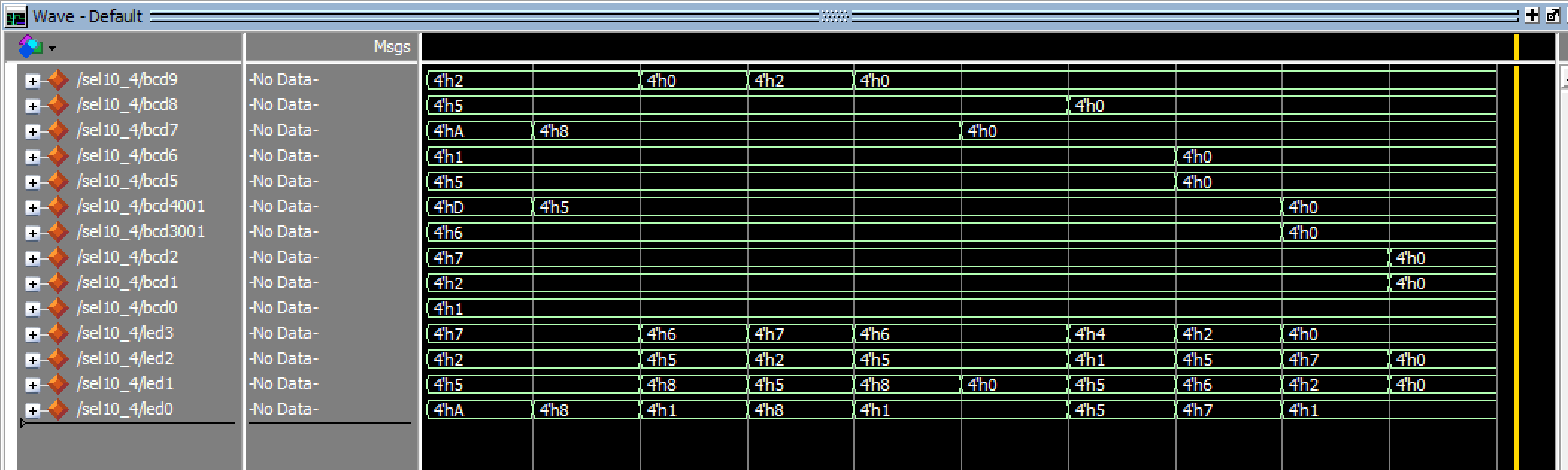
end if;

end process;

end Behavi;

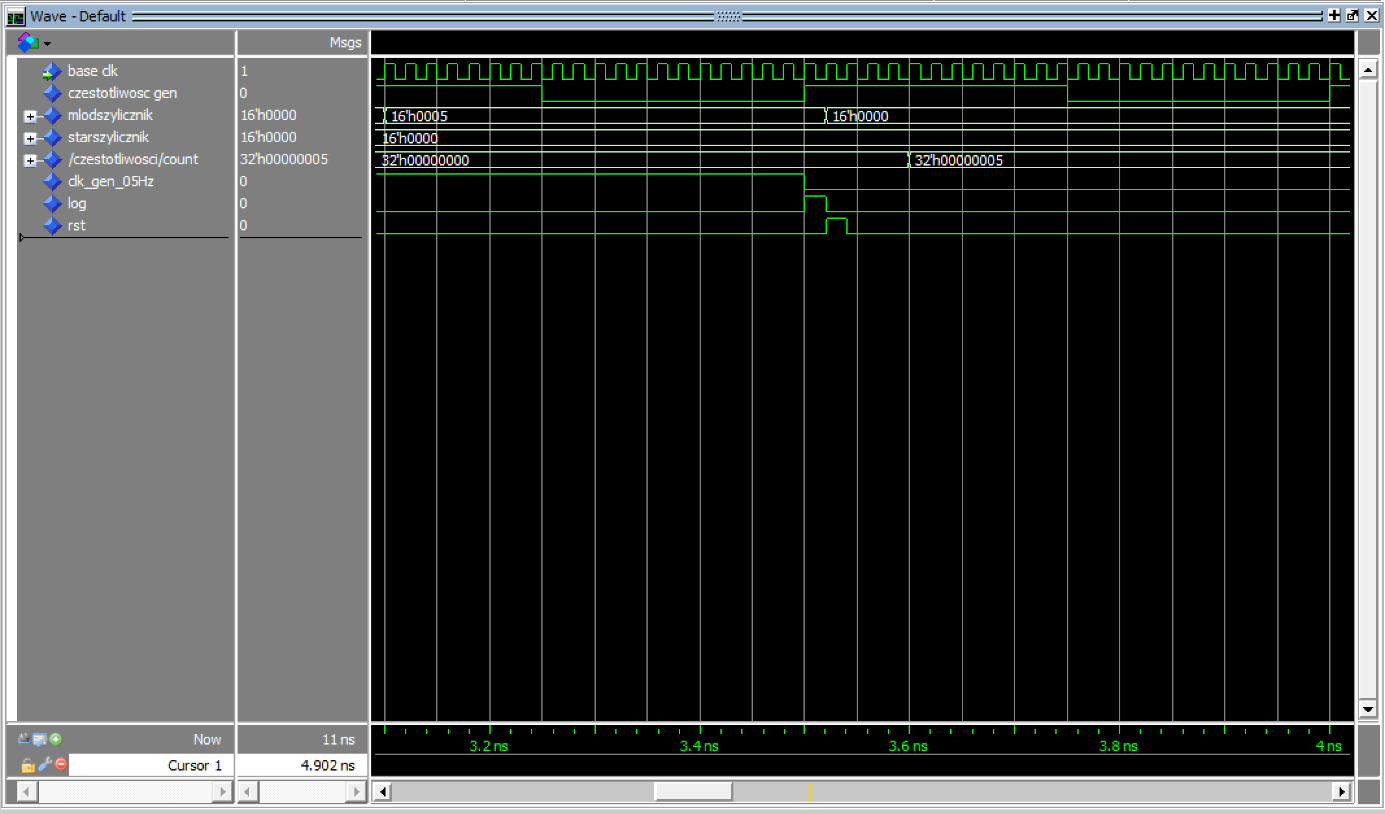
1. ***Symulacja bloku sel10\_4***

* 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:



1. ***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.



1. ***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:

