

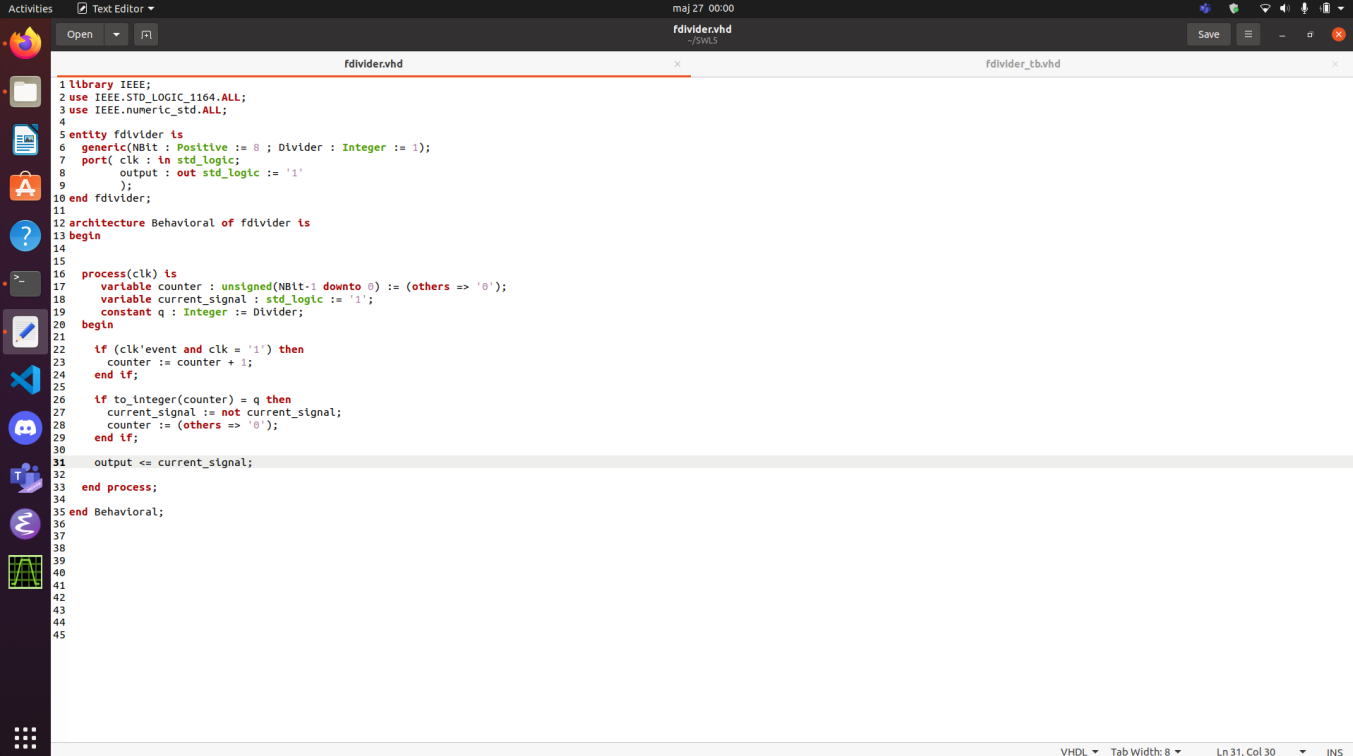
Lista 5 (VHDL) - rozwiązania

Błażej Wróbel, 250070, 3. rok, informatyka, W11

Zadanie 1.

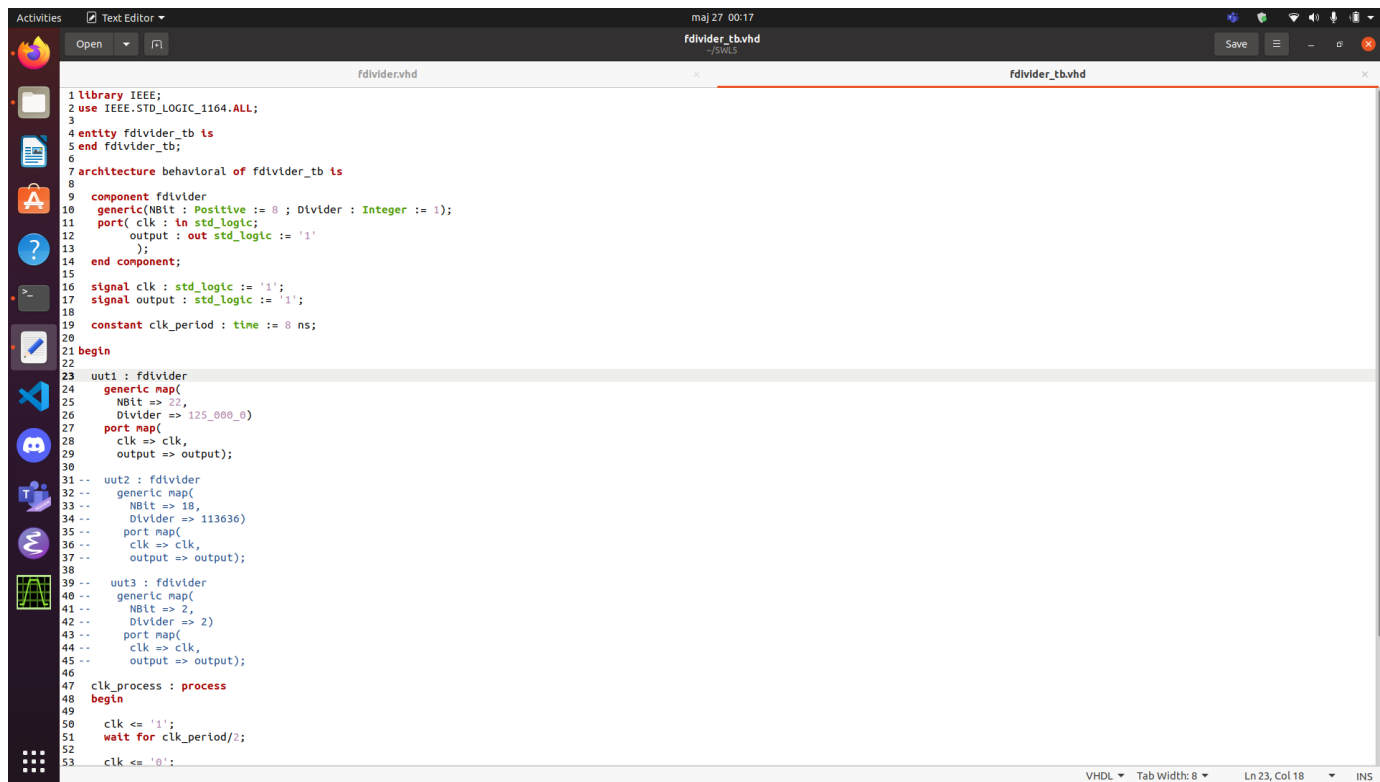
W zadaniu 1 należało napisać kod dzielnika częstotliwości i zbudować test dla tego dzielnika (w którym utworzono trzy instancje generujące przebiegi o następujących częstotliwościach : 100 Hz, 1100 Hz, 50 MHz. Główny zegar ma częstotliwość 125 MHz). Idea implementacji jest następująca : używam N-bitowego licznika binarnego (zmienna typu unsigned. N jest jednym z parametrów w pragmie generic) do liczenia taktów zegara (jako takt zegara przyjmuje sytuację, gdy jego sygnał to logiczne 1). Dodatkowo w pragmie generic daje dodatkowy parametr q (Divider), który mówi ile taktów zegara głównego mam zliczyć (tzn. ile taktów zegara głównego przypada na jeden takt zegara spowolnionego - przez ile mam dzielić częstotliwość). Jeśli wartość licznika będzie równa wartości q (dividera), to wtedy wiem, że zliczyłem odpowiednią liczbę taktów i mogę zanegować wartość logiczną na wyjściu dzielnika ("odwrócić" sygnał), wyzerować licznik i zacząć cały proces od nowa (tu widać, że liczba bitów licznika zależy od wielkości parametru q. Odpowiednia wartość logiczna na wyjściu dzielnika jest "trzymana" dopóki nie doliczymy się właściwej liczby taktów zegara głównego).

Oto implementacja zadania 1-szego w VHDL:

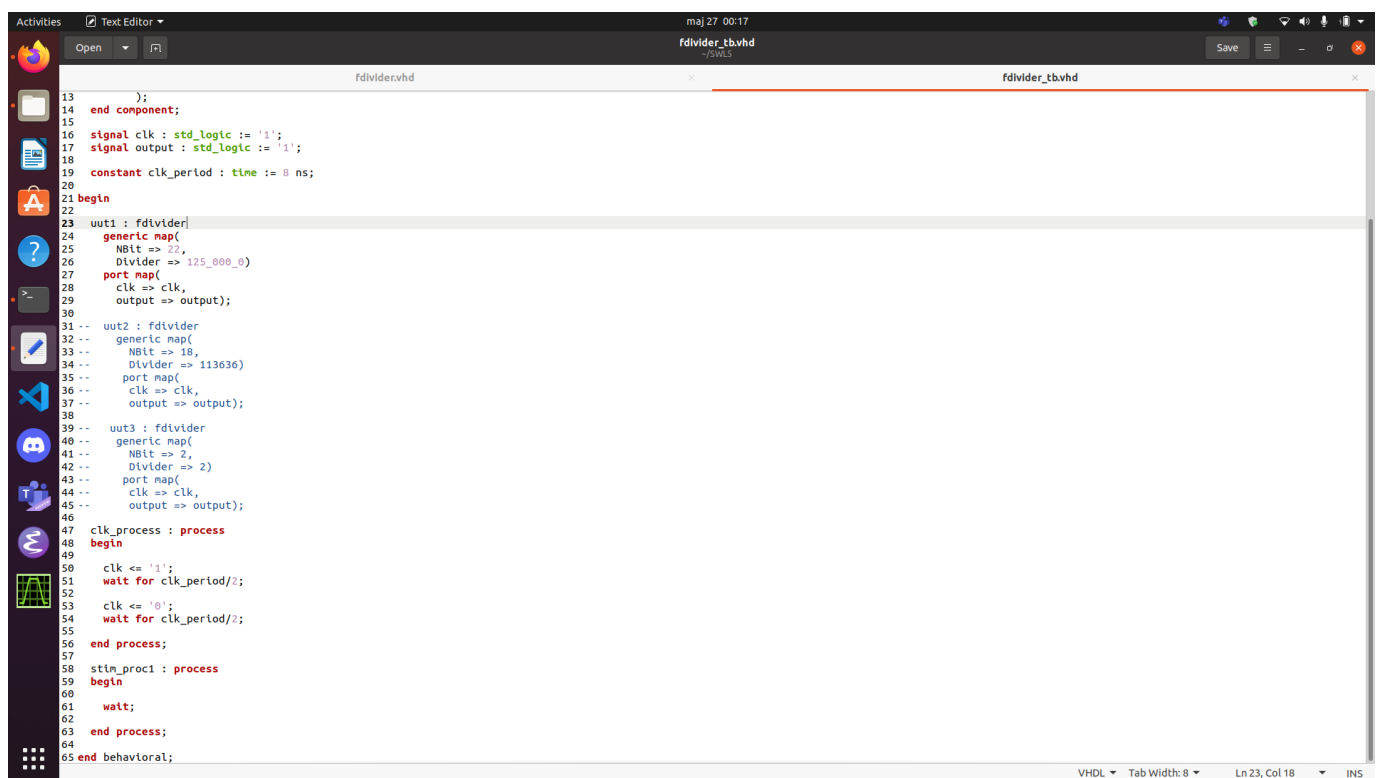


```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.numeric_std.ALL;
4
5 entity fddivider is
6   generic(NBit : Positive := 8 ; Divider : Integer := 1);
7   port( clk : in std_logic;
8         output : out std_logic := '1'
9       );
10 end fddivider;
11
12 architecture Behavioral of fddivider is
13 begin
14
15   process(clk) is
16     variable counter : unsigned(NBit-1 downto 0) := (others => '0');
17     variable current_signal : std_logic := '1';
18     constant q : Integer := Divider;
19   begin
20
21     if (clk'event and clk = '1') then
22       counter := counter + 1;
23     end if;
24
25     if to_integer(counter) = q then
26       current_signal := not current_signal;
27       counter := (others => '0');
28     end if;
29
30     output <= current_signal;
31   end process;
32 end Behavioral;
33
34
35
36
37
38
39
40
41
42
43
44
45
```

Oprócz tego, należało jeszcze zaimplementować testbench. Zaimplementowałem go w taki sposób jaki należy, czyli : podałem parametry do pragmy generic (liczba bitów w liczniku binarnym (logarytm o podstawie 2 z wartości parametru Divider plus 1) i wartość, do której ma doliczyć), stworzyłem trzy jednostki UUT, uruchomiłem proces zegara i symulacji. Oto kod testbenchu:



```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 entity fdivider_tb is
5 end fdivider_tb;
6
7 architecture behavioral of fdivider_tb is
8
9     component fdivider
10         generic(NBlt : Positive := 8 ; Divider : Integer := 1);
11         port( clk : in std_logic;
12               output : out std_logic := '1'
13             );
14     end component;
15
16     signal clk : std_logic := '1';
17     signal output : std_logic := '1';
18
19     constant clk_period : time := 8 ns;
20
21 begin
22
23     uut1 : fdivider
24         generic map(
25             NBlt => 22,
26             Divider => 125_000_0)
27         port map(
28             clk => clk,
29             output => output);
30
31 -- uut2 : fdivider
32 --     generic map(
33 --         NBlt => 18,
34 --         Divider => 113636)
35 --     port map(
36 --         clk => clk,
37 --         output => output);
38
39 -- uut3 : fdivider
40 --     generic map(
41 --         NBlt => 2,
42 --         Divider => 2)
43 --     port map(
44 --         clk => clk,
45 --         output => output);
46
47     clk_process : process
48     begin
49         clk <= '1';
50         wait for clk_period/2;
51         clk <= '0';
52         wait for clk_period/2;
53     end process;
```



```
54
55     clk_process : process
56     begin
57         clk <= '1';
58         wait for clk_period/2;
59         clk <= '0';
60         wait for clk_period/2;
61     end process;
62
63     stin_proc1 : process
64     begin
65         wait;
66     end process;
67
68 end behavioral;
```

Niedokładności generowanych przebiegów:

- 100 Hz - dzielnik (równy $125\text{MHz}/100\text{Hz} = 1250000$) jest liczbą całkowitą (nie ma żadnego zaokrąglania), więc niedokładność generowanego przebiegu jest równa 0 %.

- 1.1 kHz - dzielnik zlicza 113636 taktów zegara zamiast ($125\text{MHz}/1.1\text{kHz} = 113636.3636$). Zatem niedokładność generowanego przebiegu to $113636/113636.3636 = 0.9999$, czyli 99.99 %.
- 50 MHz - dzielnik zlicza 2 takty zegara, zamiast 2.5 taktów ($125\text{MHz}/50\text{MHz} = 2.5$). Zatem niedokładność przebiegu wynosi $2/2.5 = 0.8$, czyli 80 %

Wygenerowany przebieg znajduje się w pliku *fdiv.vcd* (można go obejrzeć za pomocą programu gtkwave).

Zadanie 2.

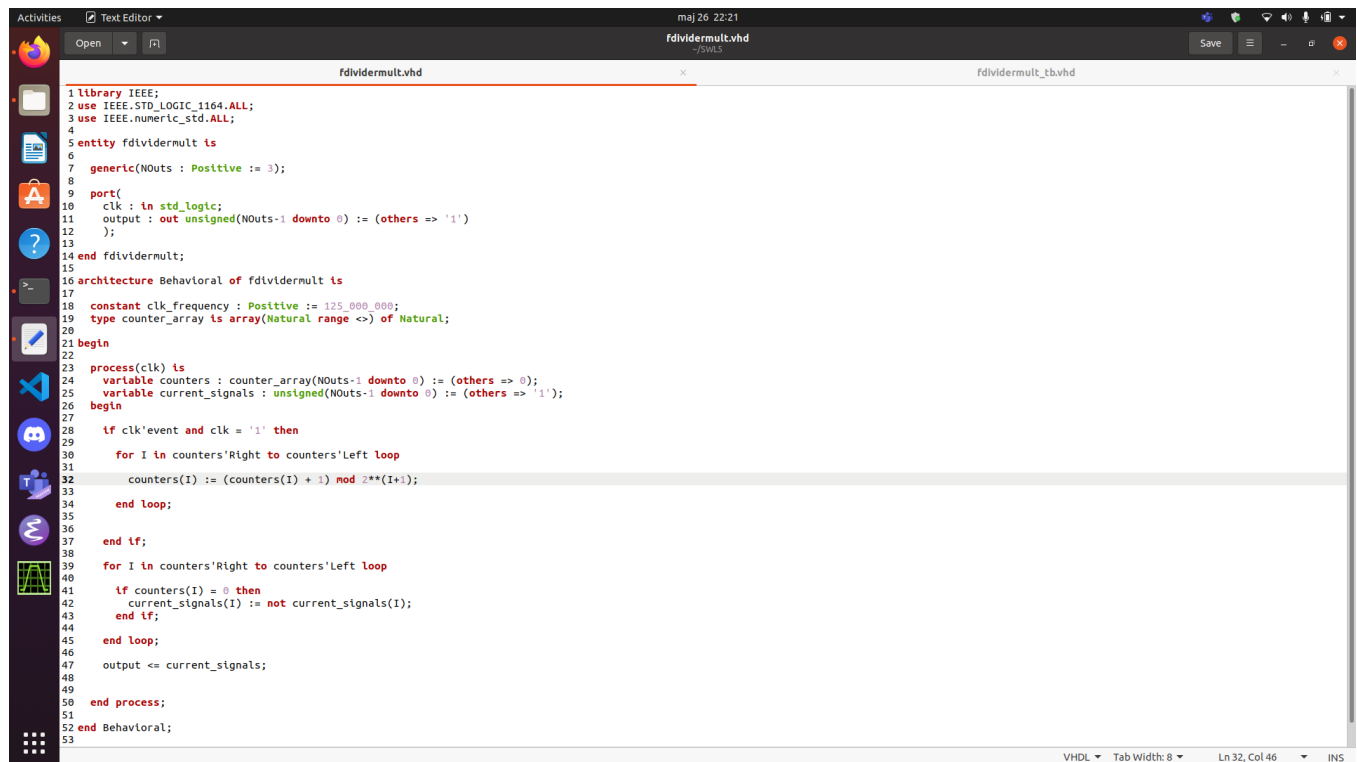
W zadaniu 2 należało zaimplementować kod dzielnika częstotliwości, który będzie miał N wyjść i j -te wyjście będzie zegarem o okresie $2^{(j+1)}$ razy dłuższym niż okres zegara głównego (liczba wyjść jest zmienna tj. trzeba użyć pragmy generic).

Implementacja tego zadania jest podobna do implementacji zadania 1-szego, ale są pewne różnice:

1. Zamiast używania licznika (który był typu unsigned), korzystam z tablicy liczb naturalnych. Każda pozycja (j) w tablicy, podczas taktu zegara będzie zwiększana o 1 i dzielona modulo $2^{(j+1)}$ (dzięki temu po np. 16 taktach na pozycji $j = 3$ będziemy mieli wartość 0).
2. Aby zasymulować fakt, że układ ma N wyjść używam N - bitowego typu *unsinged* (przy czym N jest określone pragmą generic).

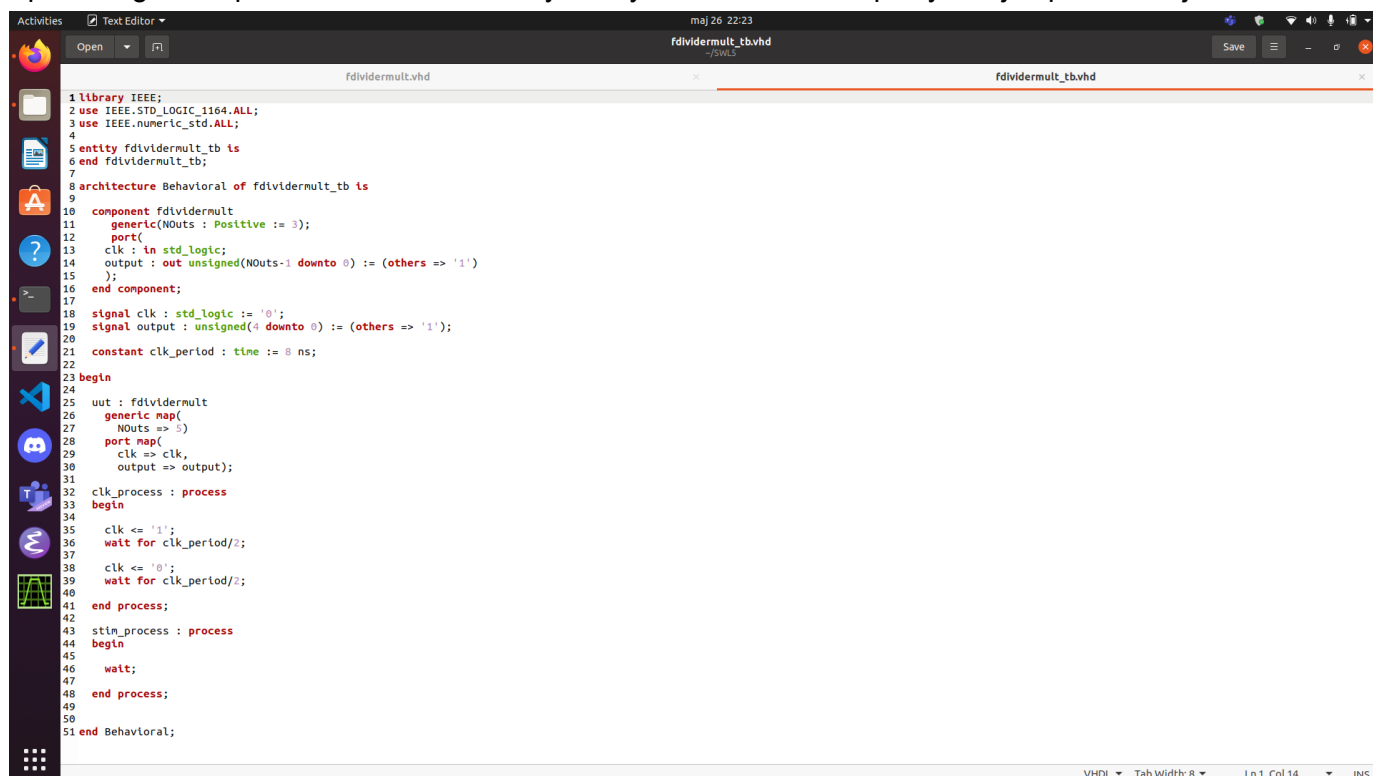
Jeśli na j -tej pozycji pojawi się wartość 0, to oznacza, że zliczyliśmy odpowiednią ilość taktów zegara głównego i musimy zanegować wartość logiczną na j -tym wyjściu (konkretnie na j -tej pozycji w typie unsigned). Potem ta wartość jest "trzymana" na tym wyjściu dopóki znowu nie zliczymy odpowiedniej liczby taktów zegara głównego (jak to już nastąpi to znowu negujemy wartość logiczną na j -tym wyjściu układu i powtarzamy powyższy proces). Dzięki temu na j -tym wyjściu uzyskujemy zegar o okresie $2^{(j+1)}$ razy dłuższym niż okres zegara głównego.

Implementacja w VHDL:



```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.numeric_std.ALL;
4
5 entity fddivermult is
6
7     generic(NOuts : Positive := 3);
8
9     port(
10         clk : in std_logic;
11         output : out unsigned(NOuts-1 downto 0) := (others => '1')
12     );
13
14 end fddivermult;
15
16 architecture Behavioral of fddivermult is
17
18     constant clk_frequency : Positive := 125_000_000;
19     type counter_array is array(Natural range <>) of Natural;
20
21 begin
22
23     process(clk) is
24         variable counters : counter_array(NOuts-1 downto 0) := (others => 0);
25         variable current_signals : unsigned(NOuts-1 downto 0) := (others => '1');
26     begin
27
28         if clk'event and clk = '1' then
29             for I in counters'Right to counters'Left loop
30                 counters(I) := (counters(I) + 1) mod 2**(I+1);
31             end loop;
32
33             end if;
34
35             for I in counters'Right to counters'Left loop
36                 if counters(I) = 0 then
37                     current_signals(I) := not current_signals(I);
38                 end if;
39             end loop;
40
41             output <= current_signals;
42
43         end process;
44     end Behavioral;
45
```

Oprócz tego zaimplementowałem test, aby zweryfikować działanie powyższej implementacji:



```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3 use IEEE.numeric_std.ALL;
4
5 entity fddivermult_tb is
6 end fddivermult_tb;
7
8 architecture Behavioral of fddivermult_tb is
9
10     component fddivermult
11         generic(NOuts : Positive := 3);
12         port(
13             clk : in std_logic;
14             output : out unsigned(NOuts-1 downto 0) := (others => '1')
15         );
16     end component;
17
18     signal clk : std_logic := '0';
19     signal output : unsigned(4 downto 0) := (others => '1');
20
21     constant clk_period : time := 8 ns;
22
23 begin
24
25     uut : fddivermult
26         generic map(
27             NOuts => 5)
28         port map(
29             clk => clk,
30             output => output);
31
32     clk_process : process
33     begin
34
35         clk <= '1';
36         wait for clk_period/2;
37
38         clk <= '0';
39         wait for clk_period/2;
40
41     end process;
42
43     stim_process : process
44     begin
45
46         wait;
47
48     end process;
49
50 end Behavioral;
51
```

Jak widać test jest standardowy - po prostu tworzę UUT, podaje parametry (których potrzebuje pragma generic), deklaruje odpowiednie sygnały i tworzę procesy zegara i symulacji. Za pomocą tego testu, wygenerowałem przebiegi, które są w pliku *fddivmult.vcd* i można je obejrzeć za pomocą programu gtkwave.

Zadanie dodatkowe:

A. Po podwojeniu częstotliwości zegara głównego mamy częstotliwość 250 MHz.

Zatem:

- 100 Hz - dzielnik wynosi 2500000 i jest liczbą całkowitą, więc niedokładność jest nadal równa 0 %.
- 1100 Hz - dzielnik wynosi 227272.7273 i nie jest liczbą całkowitą, ale również w tym przypadku niedokładność wynosi 99.99 % (bo $227272.7273/227272 = 0.9999$).
- 50 MHz - dzielnik wynosi 5 i jest liczbą całkowitą, więc niedokładność w tym przypadku wynosi 0 %.

Zatem widać, że podwojenie częstotliwości zegara głównego spowodowało poprawę niedokładności generowanych przebiegów lub nie spowodowało jej zmiany (niedokładność była podobna do tej sprzed podwojenia częstotliwości). Zatem można "wysnuć" hipotezę, że podwojenie częstotliwości zegara głównego, nie pogorszy niedokładności generowanych przebiegów.

- B. Niech a będzie liczbą rzeczywistą i będzie okresem T głównego zegara. Zatem okres przyspieszonego zegara będzie równy $T' = a - 0.5 \cdot a = a/2$. Niech f i f' będą częstotliwościami, odpowiednio zegara głównego i przyspieszonego. Z wartości T i T' otrzymujemy związek $f/f' = 1/2$. Po rozrysowaniu przebiegów dla obu zegarów można zauważyć, że jeden takt zegara głównego odpowiada dwóm taktom zegara przyspieszonego i okres zegara głównego można podzielić na 4 części - 2 gdy na wyjściu zegara przyspieszonego jest logiczna 1 oraz 2 gdy na wyjściu zegara przyspieszonego jest logiczne 0. Zatem można (np. w VHDL) policzyć okres głównego zegara (T), a potem stworzyć proces zegara przyspieszonego, który co $T/4$ s (sekundy) będzie "odwracał" (negował) wartość logiczną aktualnie znajdującą się na wyjściu zegara przyspieszonego.
- C. Można spróbować podwoić częstotliwość zegara głównego (wtedy będziemy mieli 2 MHz) i podzielić ją przez 5 (liczba całkowita, więc niedokładność generowanego przebiegu będzie równa 0%), bo $(2 \cdot 10^6 / 4 \cdot 10^5 = 5)$.