ENGINE

Teaching online electronics, microcontrollers and programming in Higher Education

Sprzętowa implementacja algorytmów

1. Symulacja i testbench. Dzielnik częstotliwości.

Lider projektu: Politechnika Warszawska

Autor: Łukasz Mik

Akademia Nauk Stosowanych w Tarnowie

GRANT NUMBER: 2020-1-PL01-KA226-HE-095653

Declaration

This laboratory instruction has been prepared in the context of the ENGINE project. Where other published and unpublished source materials have been used, these have been acknowledged.

Copyright

Funding Disclaimer

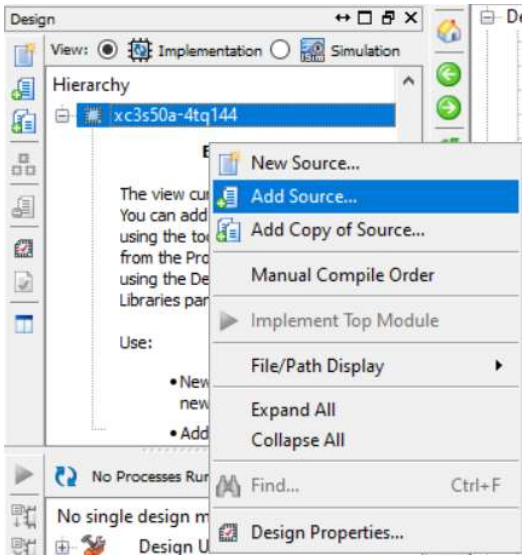
This project has been funded with support from the European Commission. This report reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

I. Testbench

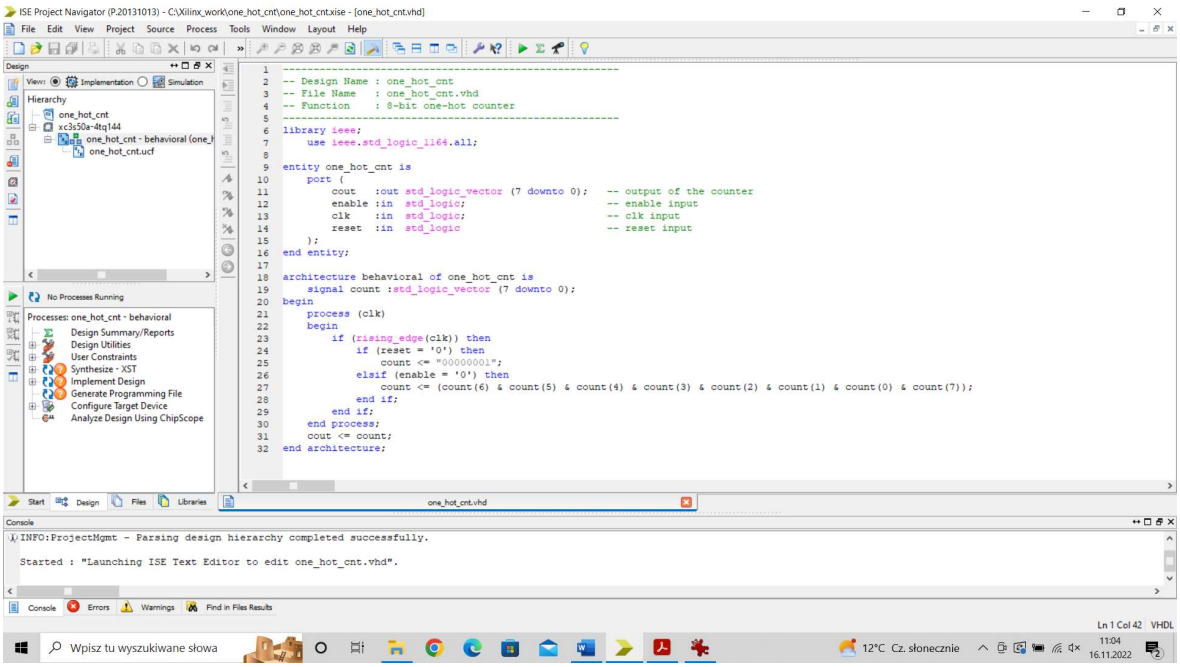
Oprogramowanie ISE Webpack (darmowa wersja pakietu ISE Design Suite) zawiera w sobie narzędzia do pisania procedur testowych tzw. testbench i symulacji projektów. Język VHDL zawiera wiele różnych sposobów definiowania sygnałów wejściowych jednostki testowej. Podczas tych zajęć zostanie przedstawiony podstawowy przykład, który można wykorzystać jako szablon dla bardziej złożonych procedur testowych.

KROK 1: Tworzenie nowego projektu w programie z wykorzystaniem gotowych modułów *.vhd oraz *.ucf

Uruchom program *ISE Design Suite 14.7*, korzystając ze skrótu na pulpicie lub z menu *Start → Programy → Xilinx Design Tools → 64-bit Project Navigator*. Jeśli są otwarte jakieś projekty to zamknij je wszystkie wybierając opcję *File → Close Project*. Następnie utwórz nowy projekt o nazwie *one_hot_cnt* wg instrukcji lab1.pdf pomijając tworzenie nowego modułu *.vhd, a tylko dodając źródła zamieszczone do ćwiczeń: pliki *one_hot_cnt.vhd* oraz *one_hot_cnt.ucf*. Sposób dodania nowych źródeł został przedstawiony na poniższym rysunku.



Po utworzeniu projektu i zaimportowaniu potrzebnych plików okno programu powinno wyglądać następująco:



W konfiguracji jednostki projektowej znajduje się jedno wyjście w postaci 8-bitowego wektora *cout* i 3 wejścia 1-bitowe: clk, enable i reset.

W opisie architektury licznika „krocząca jedynka” (zwany również licznikiem pierścieniowym) znajduje się jeden proces, który na liście wrażliwości posiada tylko sygnał zegarowy (*clk*). W procesie jest sprawdzane zbocze zegara. Jeśli jest narastające to sprawdzany jest stan wejścia *reset*. Jeśli jest 0 logiczne na tym wejściu to następuje ustawienie jedynki na najmłodszym bicie wektora *count*. W przeciwnym wypadku jest sprawdzany stan wejścia *enable*. Jeśli jest na nim 0 logiczne, to wtedy z każdym zboczem narastającym zegara najstarszy bit (o wadze 7) trafia na miejsce najmłodszego (waga 0), a ten z kolei wędruje o 1 pozycję wyżej.

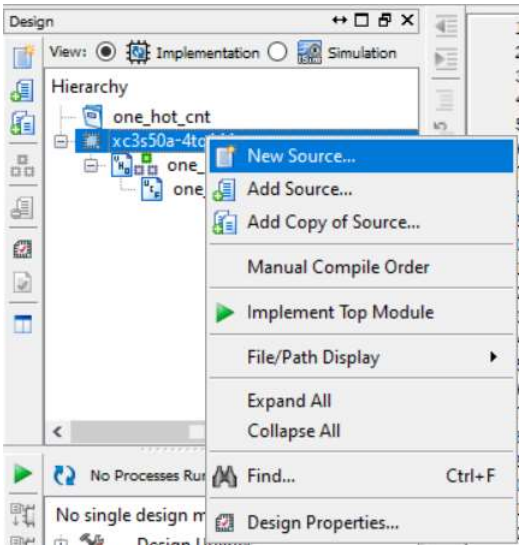
```
process (clk)
begin
if (rising_edge(clk)) then
   if (reset = '0') then
      count <= "00000001";
   elsif (enable = '0') then
       count <= (count(6) & count(5) & count(4) & count(3) & count(2) &
                 count(1) & count(0) & count(7));
   end if;
end if;
end process;
```

Operator ‘&’ odpowiada za sklejanie poszczególnych bitów w wektor *count*. Sygnał *count,* zdefiniowany wewnątrz architektury, jest przypisywany do wyjścia *cout.* Przypisanie `cout <= count` jest operacją współbieżną z procesem (wykonywaną równolegle).

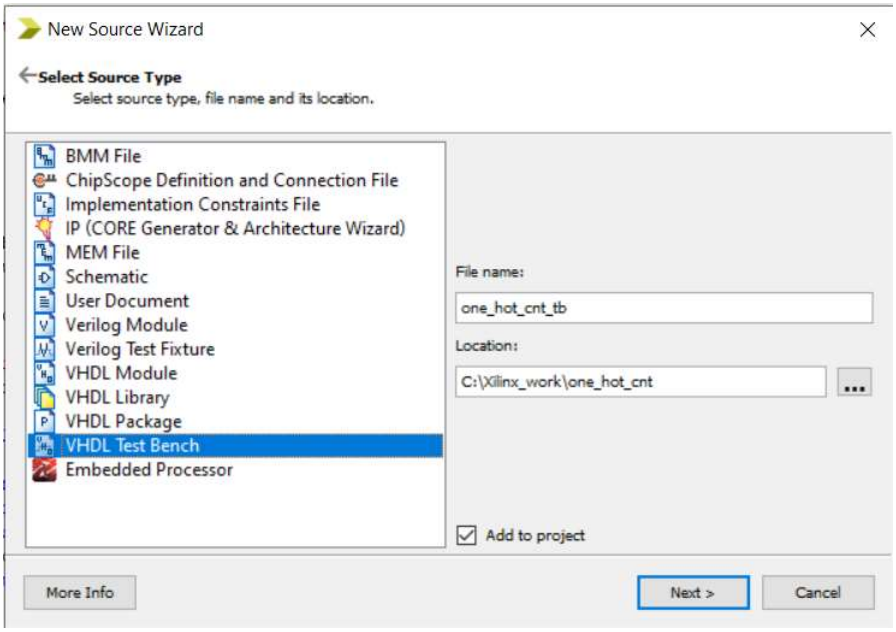
Przed przystąpieniem do generowania wektorów testowych, należy sprawdzić projekt pod względem syntaktycznym (dwukrotnie kliknąć na *Synthesize – XST*) a następnie wygeneruj plik programujący (dwukrotnie kliknąć na *Generate Programming File*). Dwukrotne kliknięcie tylko na ostatnią opcję implikuje użycie wszystkich poprzednich tj. *Synthesize – XST* oraz *Implementing Design.*

KROK 2: Tworzenie procedury testowej (plik VHDL Testbench).

Do projektu należy dodać nowy plik wybierając z menu kontekstowego opcję *New Source.*



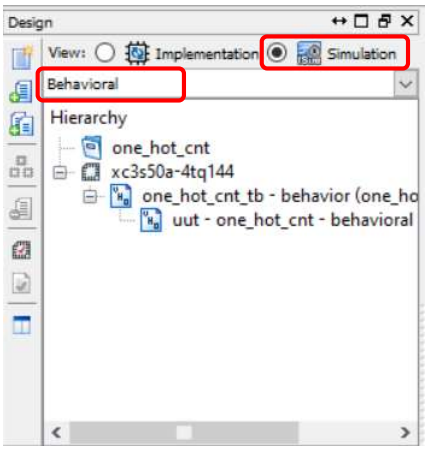
Następnie z listy plików możliwych do utworzenia należy wybrać *VHDL Test Bench.*



W celu łatwej identyfikacji plików w projekcie, najlepiej nazwę pliku z procedurą testową zakończyć ciągiem znaków „*_tb*”. Jest to skrót od słowa testbench, dzięki czemu

łatwo będzie odnaleźć ten plik w przyszłości. W kolejnym oknie dialogowym, które się pojawi należy wybrać „*one_hot_cnt*” jako źródło.

W oknie projektu należy zmienić widok źródeł z *Implementation* na *Simulation.* Edytor przeniesie nas do widoku plików przygotowanych do symulacji.



Z rozwijanej listy w tym samym oknie powinna być wybrana opcja *Behavioral*, co oznacza, że nasz moduł projektowy na tym etapie będzie symulowany pod względem funkcjonalności.

Po dwukrotnym kliknięciu na nazwę pliku z procedurą testową (*one_hot_cnt_tb*) w oknie edytora otworzy się jego zawartość, wygenerowana automatycznie.

```
ARCHITECTURE behavior OF one_hot_cnt_tb IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT one_hot_cnt
    PORT(
         cout : OUT  std_logic_vector(7 downto 0);
         enable : IN  std_logic;
         clk : IN  std_logic;
         reset : IN  std_logic
        );
    END COMPONENT;
```

W opisie architektury jednostki testowej został dołączony komponent, który jest naszym projektem licznika typu krocząca jedynka. Jest ona opisana komentarzem *Unit Under Test*, czyli jednostka w trakcie testu. Wewnątrz jednostki testowej znajdują się sygnału podłączone do licznika oraz okres zegara (*clk_period*), zdefiniowany jako stała. Wartość tego parametru jest automatycznie ustawiana na 10 ns (częstotliwość 100 MHz). W naszym przypadku częstotliwość zegara wynosi 12 MHz więc okres będzie odwrotnością tej liczby i wynosi w przybliżeniu 83,3 ns. Trzeba więc zamienić domyślną wartość na tą, która wynika z częstotliwości rzeczywistego zegara w układzie.

```
--Inputs
signal enable : std_logic := '0';
signal clk : std_logic := '0';
signal reset : std_logic := '0';

--Outputs
signal cout : std_logic_vector(7 downto 0);

-- Clock period definitions
constant clk_period : time := 83.3 ns;
```

Program sam automatycznie generuje sygnał zegarowy na podstawie okresu zdefiniowanego przez użytkownika. W architekturze jednostki testowej znajduje się też mapa połączeń jej sygnałów z testowanym komponentem.

```
-- Instantiate the Unit Under Test (UUT)
uut: one_hot_cnt PORT MAP (
       cout => cout,
       enable => enable,
       clk => clk,
       reset => reset
     );

-- Clock process definitions
clk_process :process
begin
   clk <= '0';
   wait for clk_period/2;
   clk <= '1';
   wait for clk_period/2;
end process;
```

W ostatniej części szablonu procedury testowej jest utworzony proces o nazwie *stim_proc*, którego zadaniem jest generowanie wejściowych sygnałów wymuszających testowany układ. Generowanie sygnałów testowych rozpoczyna się z domyślnym, 100 – sekundowym opóźnieniem. Przed tym opóźnieniem możemy ustawić ‘0’ logiczne na sygnale reset. Po tym czasie do sygnałów reset i enable przypisujemy odpowiednio stany logiczne ‘1’ i ‘0’. Po tym następuje odczekanie 10 okresów zegara. W miejscu komentarza „- - *insert stimulus here*” wstawiamy własne sygnały testowe. Na tym etapie nie ma potrzeby generowania większej liczby sygnałów wymuszających. Kod procesu *stim_proc* został przedstawiony poniżej.

```
-- Stimulus process
stim_proc: process
begin
   -- hold reset state for 100 ns.
   reset <= '0';
   wait for 100 ns;
   reset <= '1';
   enable <= '0';
   wait for clk_period*10;

   -- insert stimulus here

   wait;
end process;
```

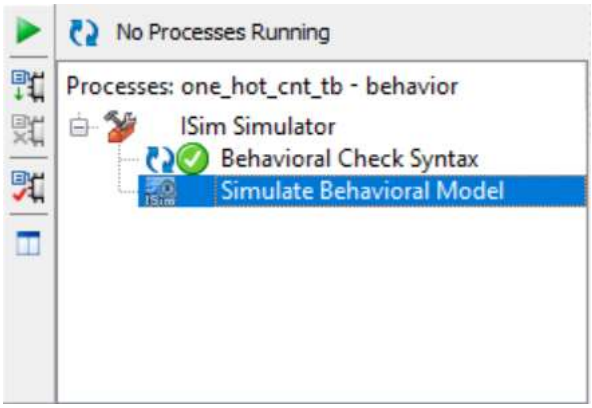
II. Symulacja funkcjonalna i czasowa

W dalszej części laboratorium zostaną omówione sposoby symulacji projektu po syntezie nazywana symulacją funkcjonalną lub behawioralną oraz po implementacji w układzie docelowym nazywana symulacją czasową, bo uwzględnia opóźnienia w układzie docelowym.

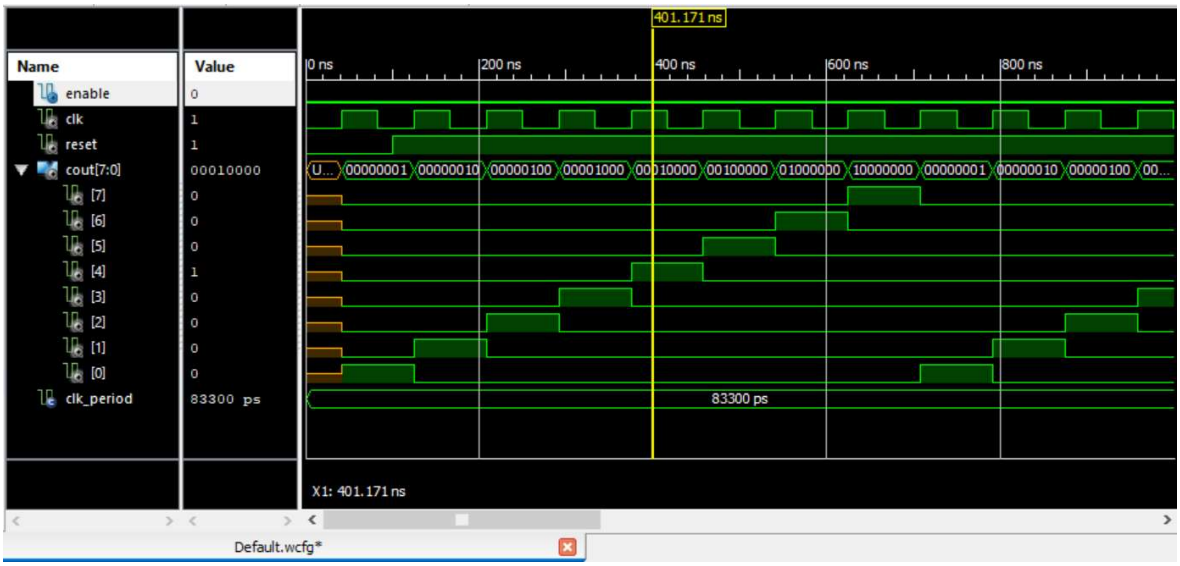
KROK 1: Symulacja funkcjonalna (behawioralna).

W pierwszej kolejności przeprowadzimy symulację projektu pod względem funkcjonalnym. Do tego jest potrzebne tylko wykonanie syntezy (wybranie opcji Synthesize – XST) w trybie implementacji projektu.

W oknie procesów symulatora najpierw uruchamiamy sprawdzanie składni pliku testowego, a następnie uruchamiamy symulację behawioralną testowanej jednostki przez wybranie opcji *Simulate Behavioral Model.*



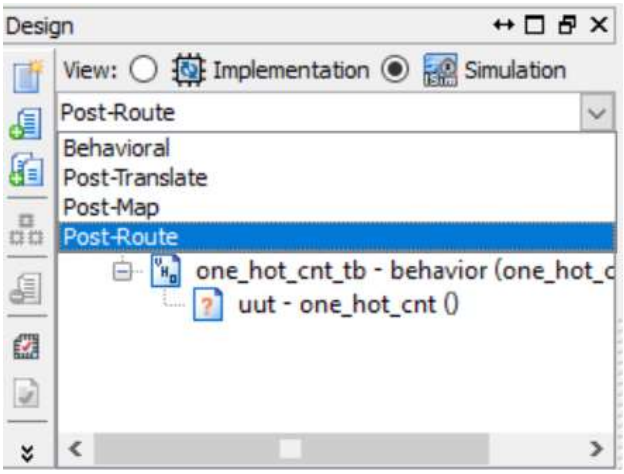
Po wybraniu tej opcji uruchomi się okno symulatora z wszystkimi sygnałami, będącymi portami jednostki testowej.



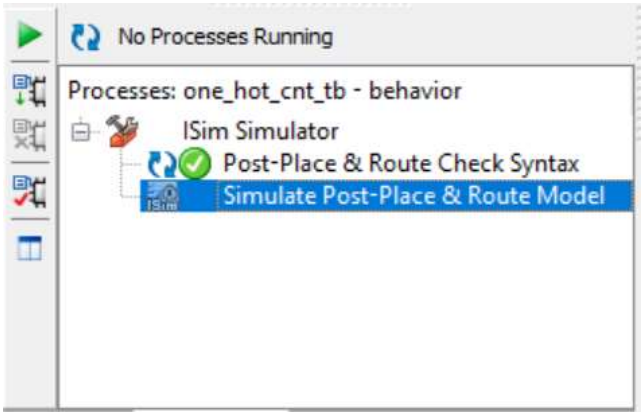
Domyślnie plik z wygenerowanymi przebiegami czasowymi ma przypisaną nazwę *Default.wcfg*. Można ją zmienić w zależności od swoich potrzeb.

KROK 2: Symulacja czasowa, uwzględniająca opóźnienia w układzie docelowym.

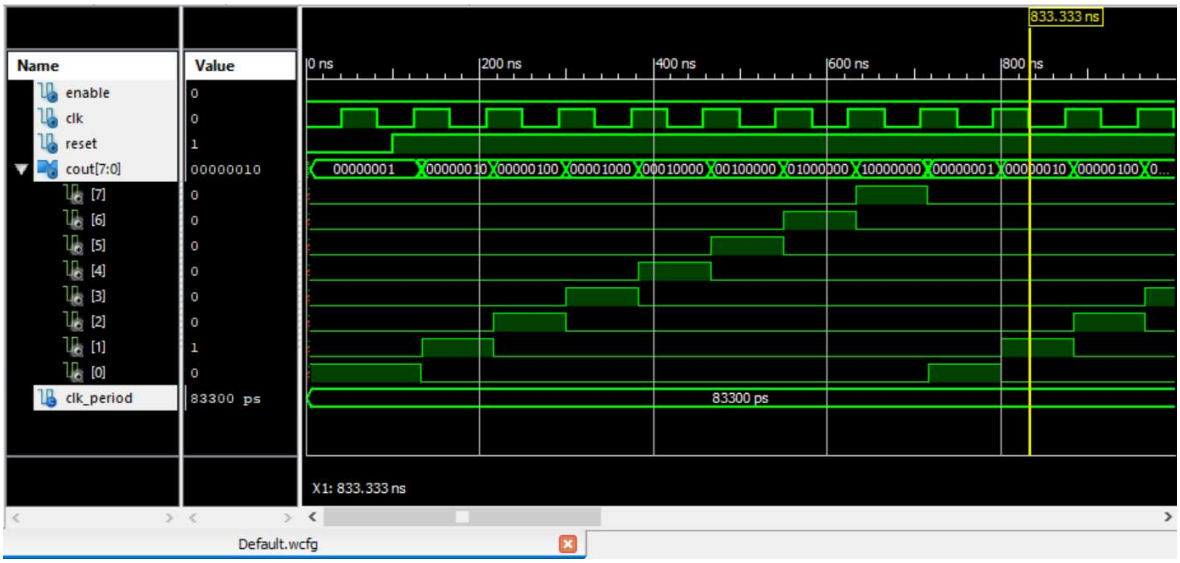
W oknie wyboru rodzaju symulacji wybieramy opcję Post-Route.



W oknie procesów symulatora najpierw uruchamiamy sprawdzanie składni pliku testowego, a następnie uruchamiamy symulację typu Post-Place and Route Model, która bazuje na rzeczywistym modelu układu scalonego, dla którego tworzymy projekt.



Wynik symulacji sygnałów wchodzących w skład architektury jednostki projektowej możemy obejrzeć w zaktualizowanym oknie przebiegów czasowych symulatora.


Name
Value
enable
clk
reset
cout[7:0]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
clk_period
0
0
1
00000010
0
0
0
0
0
0
1
0
83300 ps
0 ns
200 ns
400 ns
600 ns
800 ns
833.333 ns
00000001
00000010
00000100
00001000
00010000
00100000
01000000
10000000
00000001
00000010
00000100
83300 ps
X1: 833.333 ns
Default.wcfg

III. Dzielnik częstotliwości

Ze względu na to, że oko ludzkie nie jest w stanie reagować na zmiany szybciej niż 20 razy na sekundę, w związku z tym konieczne jest obniżenie częstotliwości zegara taktującego licznik pierścieniowy, aby użytkownik mógł dostrzec zmiany na linijce diodowej.

Należy utworzyć nowy projekt i dodać do niego pliki źródłowe o nazwach: *one_hot_cnt_divider.vhd* oraz *one_hot_cnt_divider.ucf*.

W stosunku do poprzedniego projektu, na liście deklaracji wewnątrz architektury zdefiniowany został sygnał o nazwie *clk_1Hz*, któremu przypisana jest początkowa wartość 0. W opisie architektury utworzony jest nowy proces, którego zadaniem będzie generowanie sygnału o częstotliwości 1 Hz, na podstawie wartości zmiennej „*puls_cnt*", która będzie inkrementowana w takt zegara *clk*. Stan zegara *clk_1Hz* będzie zmieniany na przeciwny, po każdym osiągnięciu przez zmienną „pulse_cnt" wartości 6000000, co daje pół okresu zegara pracującego z częstotliwością 1 Hz. W procesie dotyczącym licznika pierścieniowego zostaje zamieniony zegar *clk* na *clk_1Hz*. Zmiana też nastąpiła w nazwie jednostki projektowej – teraz nazywa się *one_hot_cnt_divider*.

Wprowadzenie tych modyfikacji do pierwotnego projektu spowodowało, że w liczniku pierścieniowym zmiana bitu następuje z częstotliwością 1 Hz.

Skompiluj projekt i zaprogramuj układ plikiem konfiguracyjnym *.bit. Sprawdź czy wszystko działa zgodnie z opisem architektury, tj. przycisk *reset* ustawia '1' logiczną na początku linijki LED (D8) a przycisk *enable* powoduje uruchomienie licznika.

ZADANIE:

Na podstawie omówionych w trakcie zajęć projektów licznika pierścieniowego i dzielnika częstotliwości zbuduj własny układ, który na wyjściu audio (złącze J2) będzie generował sygnał z różnymi częstotliwościami, wybieranymi za pomocą przycisków SW1 – SW6. Poniższa tabela przedstawia częstotliwości przypisane do określonych przycisków.

Przycisk	Częstotliwość sygnału
SW1	396 Hz
SW2	417 Hz
SW3	528 Hz
SW4	639 Hz
SW5	741 Hz
SW6	852 Hz