LAB3: Język VHDL (I) – Opis układów kombinacyjnych

I. CEL ĆWICZENIA

Celem ćwiczenia jest zapoznanie z podstawami języka opisu sprzętu VHDL (*VHSIC Hardware Description Language*) oraz jego zastosowaniem do projektowania układów kombinacyjnych. Zapoznanie z metodami projektowania układów w postaci opisu behawioralnego i strukturalnego oraz symulacji kodu w środowisku *Active-HDL* firmy *Aldec*, a także analizą wyników symulacji.

II. WPROWADZENIE

Geneza języków HDL

Tradycyjne projektowanie układów cyfrowych polegało na rysowaniu schematów logicznych z wykorzystaniem bramek i przerzutników. Jednak wraz ze wzrostem złożoności projektów i pojawieniem się programowalnych układów logicznych (CPLD, FPGA), podejście to stało się niewystarczające. Zastąpiły je komputerowe narzędzia projektowe i języki opisu sprzętu (HDL), takie jak VHDL i Verilog.

Użycie języków HDL umożliwia projektowanie złożonych systemów cyfrowych poprzez opis ich zachowania w formie kodu. Taki opis może być następnie użyty do symulacji, syntezy oraz implementacji w układach programowalnych lub ASIC. Coraz powszechniejsze jest także projektowanie z użyciem dedykowanych bloków funkcjonalnych, które automatycznie generują kod HDL.

Język VHDL

Został opracowany w latach osiemdziesiątych XX wieku w ramach programu VHSIC (*Very High Speed Integrated Circuits*), a obecnie jest standardem IEEE 1076. Umożliwia opis działania i struktury układów cyfrowych na różnych poziomach abstrakcji – od poziomu logicznego (bramki, sumatory), po poziom systemowy.

Przy użyciu języka VHDL opisywane są zarówno układy kombinacyjne, jak i układy sekwencyjne, a projekty tworzy się tak, aby mogły zostać poddane symulacji, a następnie syntezie i implementacji np. w układzie FPGA.

Podstawowe fragmenty modułu projektowego w języku VHDL

1. Deklaracja bibliotek i pakietów

Bez uprzedniego zadeklarowania biblioteki IEEE w kodzie języka VHDL nie można skorzystać z wielu typów danych oraz funkcji.

`library IEEE;` – wskazanie na bibliotekę IEEE;
`use` – słowo kluczowe, określające, które pakiety z biblioteki IEEE mają zostać użyte;

Przykład deklaracji

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;      -- pakiet podstawowy
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

2. Entity (jednostka projektowa)

Tutaj opisywany jest interfejs projektowanej jednostki: **nazwa jednostki**, porty **wejściowe** i **wyjściowe**.

Przykład budowy entity

```
entity Nazwa_Jednostki is
    port (
        a  : in  std_logic_vector(2 downto 0);   -- wejscie 3-bitowe
        b  : in  std_logic_vector(7 downto 0);   -- wejscie 8-bitowe
        s  : in  std_logic;                      -- wejscie 1-bitowe
        y  : out std_logic_vector(7 downto 0)    -- wyjscie 8-bitowe
    );
end Nazwa_Jednostki;
```

3. Architecture (architektura)

Tutaj opisywany jest sposób działania jednostki – logika, procesy, **sygnały (*signal*)**. W **architekturze** określamy sposób opisu jednostki behawioralny/strukturalny.

Nazwa jednostki musi być taka sama jak w entity !!!

Przykład budowy architektury

```
architecture Behavioral of Nazwa_Jednostki is
    signal a_s : std_logic_vector(7 downto 0);  -- deklaracja sygnalu a_s
begin
    a_s(2 downto 0)  <= a;       -- przypisanie sygnalu z wejscia a do sygnalu a_s
    a_s(7 downto 3)  <= (others => '0');        -- wyrownanie formatow

      -- przypisanie sygnalu a_s do wyjscia y,
      -- jesli s='0' lub b w pozostalych przypadkach
    y <= a_s when s = '0' else b;

end Behavioral;
```

UWAGA!

Zalecana nazwa pliku powinna być taka sama jak nazwa opisywanej jednostki np. inwerter – inwerter.vhd.

Przykład opisu inwertera

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

entity inwerter is
    port (
        A : in  std_logic;
        Y : out std_logic
    );
end inwerter;

architecture Behavioral of inwerter is
begin
    Y <= not A;
end Behavioral;
```

III. REALIZACJA ĆWICZENIA LABORATORYJNEGO

STANOWISKO KOMPUTEROWE

Zadania projektowe realizowane są z użyciem stanowiska komputerowego z zainstalowanym systemem operacyjnym Windows i środowiskiem projektowo-symulacyjnym *Active-HDL* firmy *Aldec*.

WYKONANIE ĆWICZENIA

Pierwsza część ćwiczenia, zwana projektem wstępnym, ma na celu zapoznanie z obsługą środowiska projektowego *Active-HDL* na przykładzie projektu trójwejściowej bramki OR. Natomiast w drugiej części ćwiczenia realizowane są indywidualne zadania projektowe zadane przez prowadzącego ćwiczenia. Zadania obejmują implementację układów kombinacyjnych (np. bramek logicznych, multiplekserów, dekoderów) w języku VHDL. Opracowane projekty są następnie poddawane syntezie oraz analizie działania poprzez symulacje funkcjonalne.

UWAGA! Wszystkie zadania projektowe należy zapisywać w następującej lokalizacji:

C:\my_designs\<nazwa_grupy>\Lab3

Przed przystąpieniem do ćwiczenia należy utworzyć stosowny folder.

OPRACOWANIE WYNIKÓW

1. Sprawozdanie z ćwiczenia laboratoryjnego powinno zawierać zwięzłe opisy postawionych zadań projektowych wraz z rozwiązaniami (kod VHDL i zrzuty ekranów z wynikami symulacji).
2. W sprawozdaniu należy również zawrzeć rozwiązania dodatkowych problemów projektowych postawionych przez prowadzącego ćwiczenie.

ZALICZENIE ĆWICZENIA

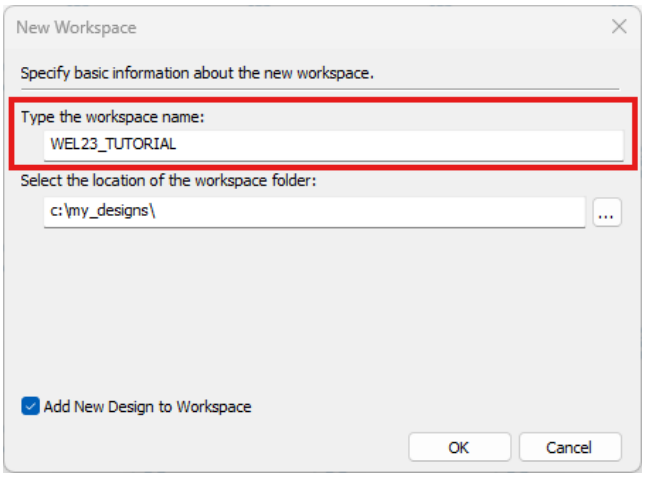
1. Zaliczenie kolokwium wstępnego oraz poprawne wykonanie zadań laboratoryjnych postawionych przez osobę prowadzącą ćwiczenie.
2. Złożenie sprawozdania, zawierającego zwięzły opis wykonanych zadań, wnioski i poprawne odpowiedzi na postawione pytania.

PROJEKT WSTĘPNY

Projekt wstępny to trójwejściowa bramka OR, który wykonano przy użyciu programu Active-HDL firmy ALDEC w wersji **14.0.**

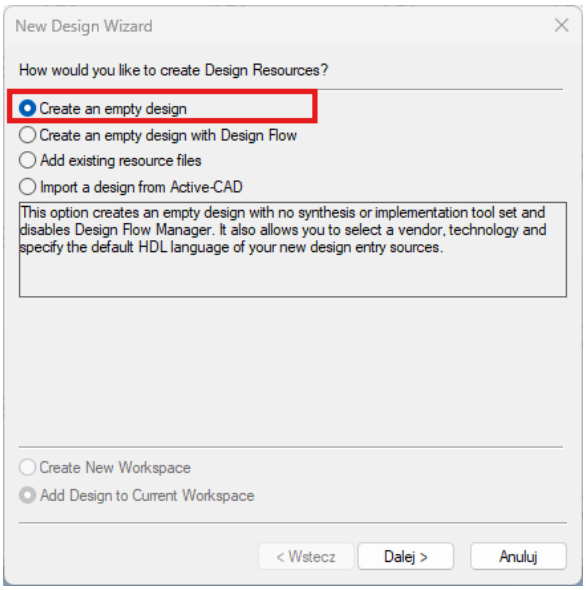
Uruchomienie programu można wykonać poprzez dwukrotne kliknięcie myszką na ikonie Aldec Active-HDL. Po pewnym czasie pojawia się okienko **License Configuration**, w którym naciskamy **OK**. Następnie otwiera się okno programu Active-HDL. Z menu głównego wybieramy **File → New → Workspace….**

Otwiera się okienko **New Workspace**, w którym wpisujemy nazwę tworzonego folderu dla projektu w polu **Type the workspace name**. Należy wpisać symbol grupy studenckiej bez spacji i polskich liter. Pozostawiamy domyślną ścieżkę **C:\my_designs** oraz włączoną opcję **Add New Design to Workspace**. Naciskamy **OK**.



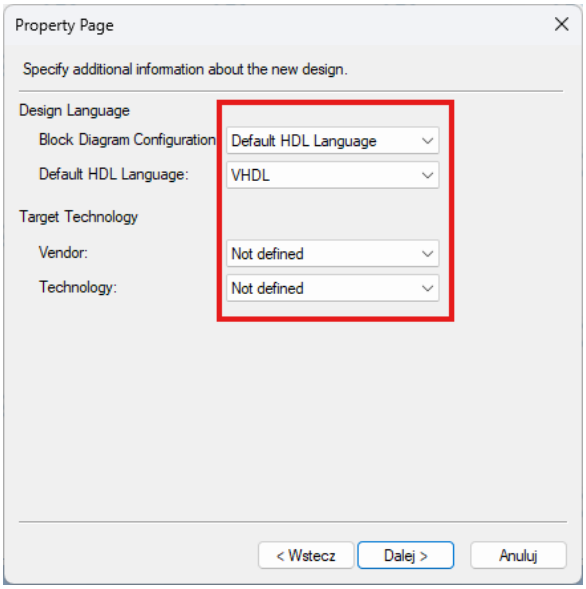
Rys. 1. Widok okna New Workspace

W kolejnym okienku **New Design Wizard** wybieramy opcję **Create an empty design** i naciskamy **Dalej**.



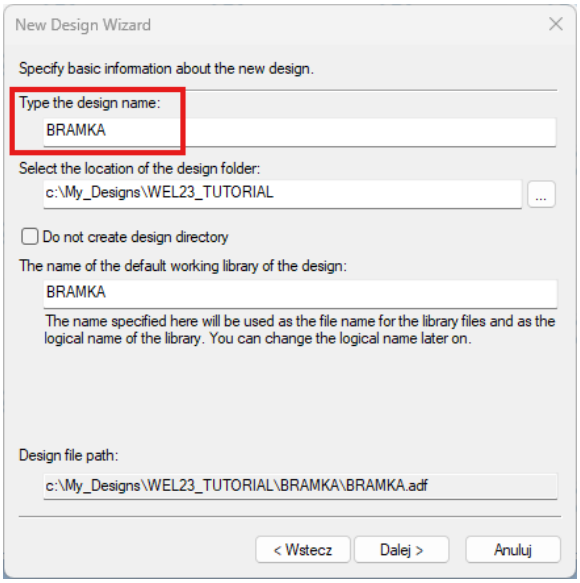
Rys. 2. Widok okna New Design Wizard

W oknie **Property Page** powinny być wybrane opcje języka VHDL bez docelowej technologii implementacji projektu. Naciskamy **Dalej**.



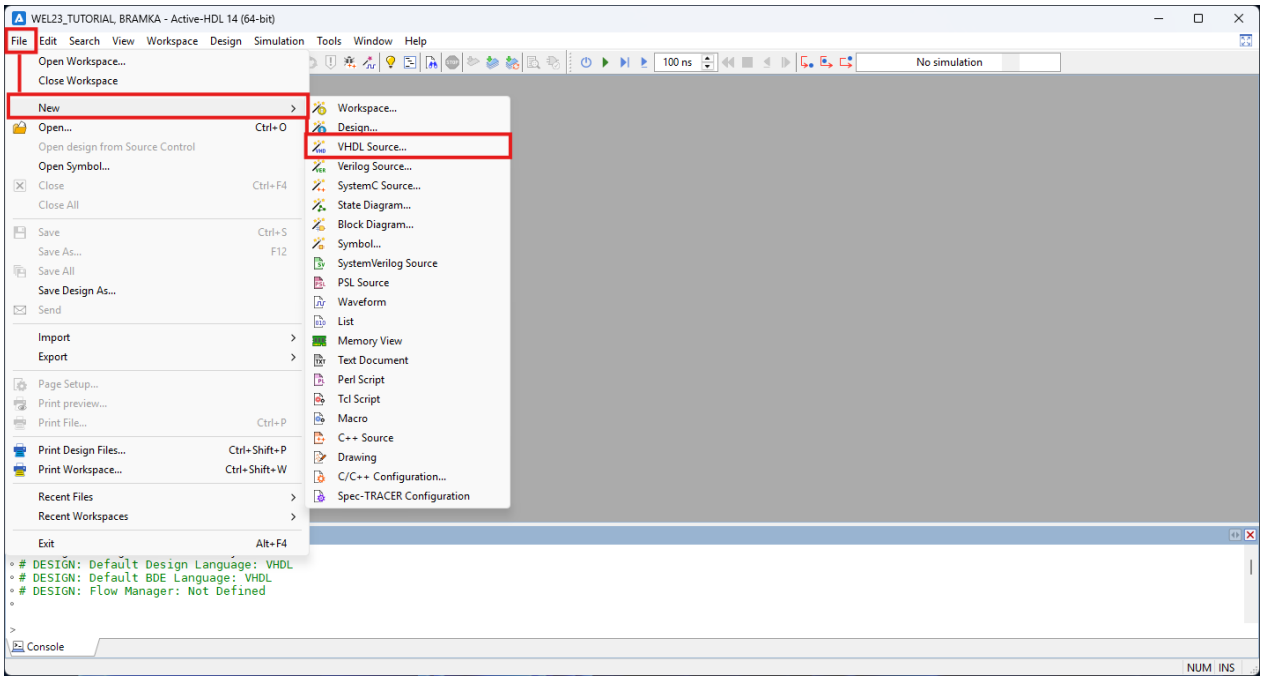
Rys. 3. Widok okna Property Page

W oknie **New Design Wizard**, w polu **Type the design name** wpisujemy nazwę projektu, np. **BRAMKA**, naciskamy **Dalej**. W kolejnym okienku naciskamy **Zakończ**.



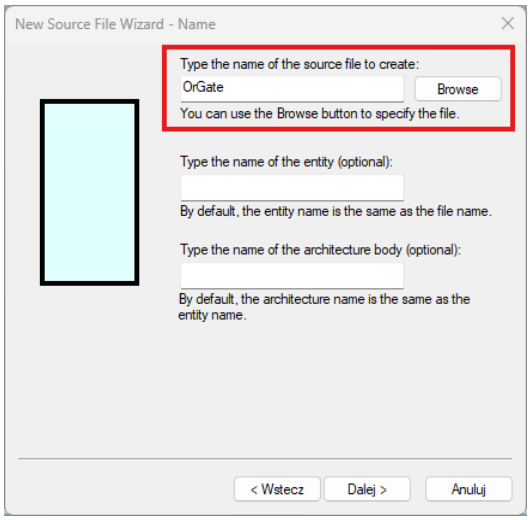
Rys. 4. Widok okna New Design Wizard

Wracamy do okna głównego programu Active-HDL, w którym pojawiło się okienko **Design Browser** zwierające strukturę naszego projektu. Teraz utworzymy i dołączymy do projektu nowy pliku typu VHDL wybierając **File → New → VHDL Source**. Pojawia się okienko **New Source File Wizard**, w którym domyślnie powinna być wybrana opcja **Add the generated file to the project**. Naciskamy **Dalej**.



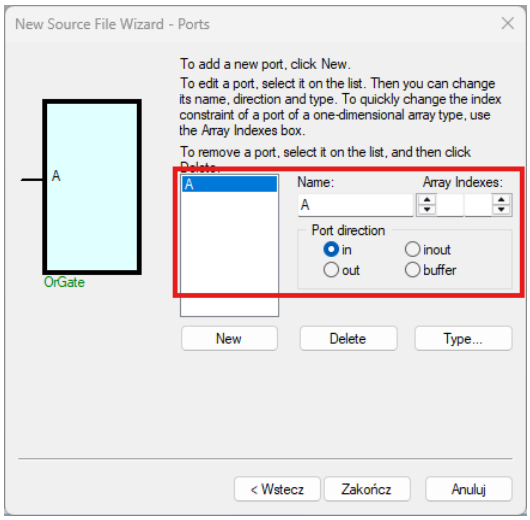
Rys. 5. Widok zakładki File / New / VHDL Source…

W kolejnym okienku **New Source File Wizard – Name** wpisujemy nazwę pliku dla przyszłego kodu VHDL, np. **OrGate**. Naciskamy **Dalej**.



Rys. 6. Widok okna New Source File Wizard - Name

Pojawia się okienko **New Source File Wizard – Ports**, w którym naciskamy **New**. W polu **Name** wpisujemy nazwę dla sygnału wejściowego, np. **A**, przy czym **Port direction** wybieramy typu **in**.

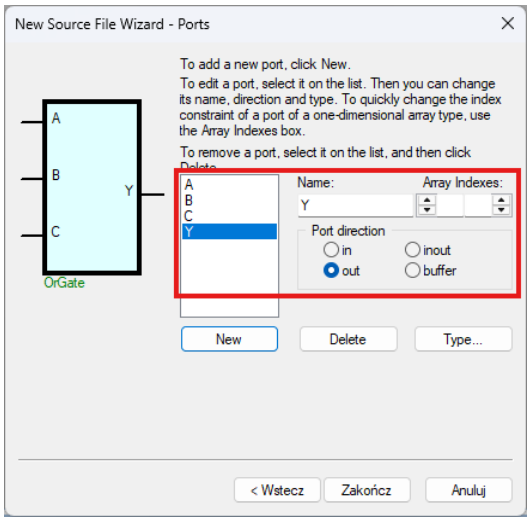


Rys. 7. Widok okna New Source File Wizard - Ports

Ponownie naciskamy **New**. W polu **Name** wpisujemy nazwę dla drugiego sygnału wejściowego, np. **B**.

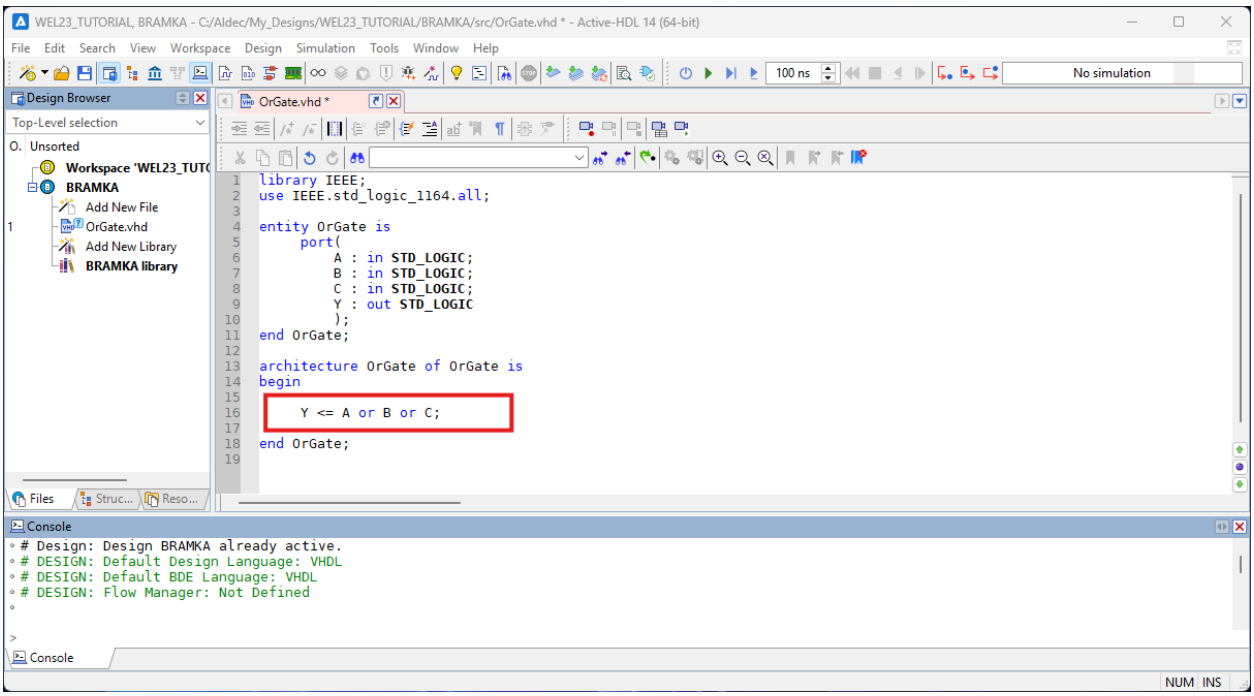
Ponownie naciskamy **New**. W polu **Name** wpisujemy nazwę dla trzeciego sygnału wejściowego, np. **C**.

Kolejny raz naciskamy **New**. W polu **Name** wpisujemy nazwę dla sygnału wyjściowego, np. **Y**, przy czym **Port direction** wybieramy typu **out**. Zamykamy kreatora naciskając **Zakończ**.



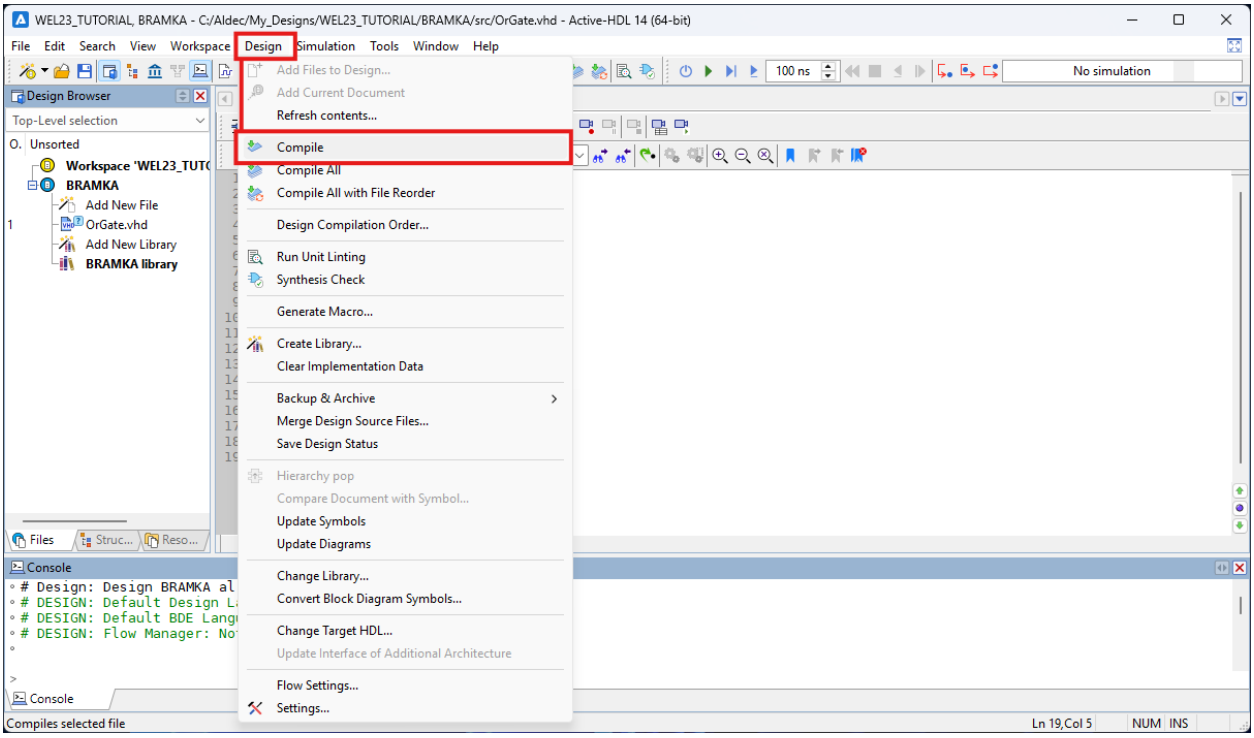
Rys. 8. Widok okna New Source File Wizard - Ports cd.

W oknie głównym programu Active-HDL pojawia się plik „OrGate.vhd”, który uzupełniamy opisem działania trójwejściowej bramki OR, czyli `Y <= A or B or C;`. Zapisujemy plik poprzez **File** → **Save**, albo **Ctrl+S**. Komentarze zaznaczone kolorem zielonym można usunąć, ponieważ nie są istotne dla naszego projektu.



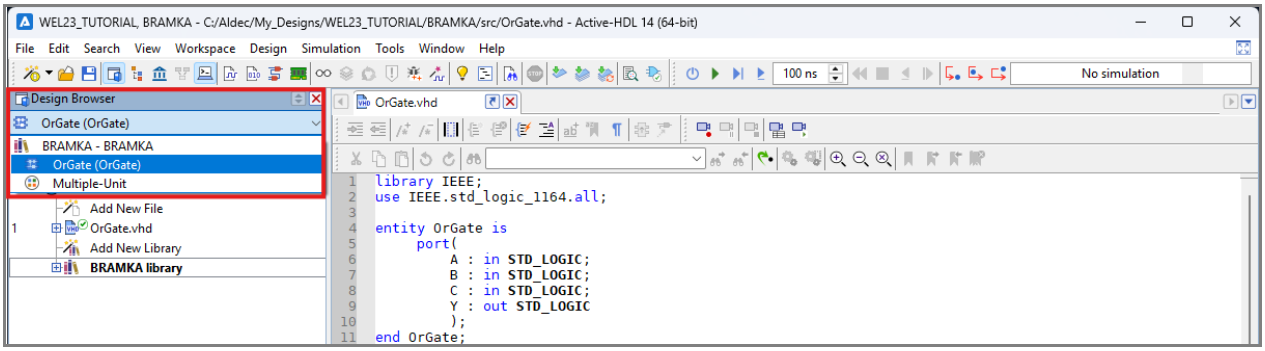
Rys. 9. Widok okna głównego – plik „OrGate.vhd”

Następnie kompilujemy utworzony kod VHDL wybierając **Design** → **Compile** lub klikając ikonę na pasku programu głównego, albo naciskając **F11**. Poprawiamy ewentualne błędy…



Rys. 10. Widok okna głównego – zakładka Design / Compile

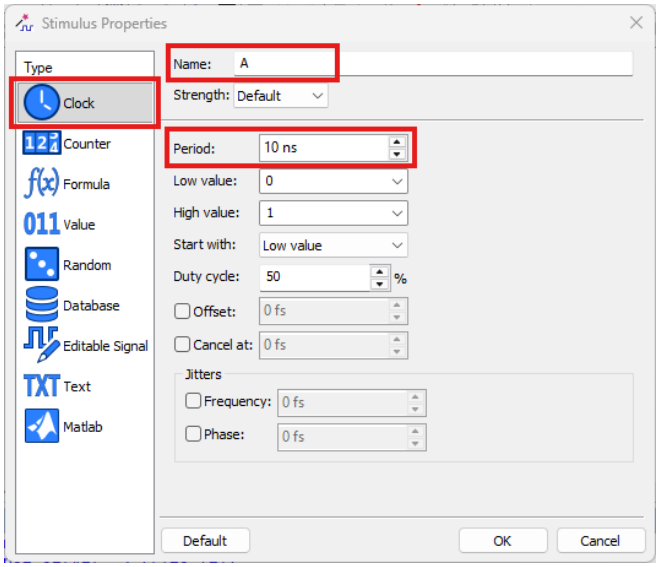
Rozpoczynamy symulację działania projektowanego układu (czyli bramki OR opisanej kodem VHDL). Jeżeli domyślnie nie została wybrana architektura do symulacji, to należy w obszarze **Design Browser** rozwinąć odpowiednie pole i wybrać **OrGate (OrGate)**. W sytuacji wielu plików zawartych w projekcie, należy w tym polu wskazać architekturę wybraną do symulacji.



Rys. 11. Widok okna głównego – obszar Design Browser

Teraz przygotujemy zestaw sygnałów testowych, które podłączymy do wejść projektu (bramki OR). W tym celu otwieramy okienko wymuszeń **Stimuli** wybierając z menu programu głównego **View** → **Stimuli**. Następnie wskazując w tym okienku na puste pole, naciskamy prawy klawisz myszki. Pojawia się menu, w którym wybieramy **Create Stimuli Set**. Zostaje utworzona grupa **StimuliSet1**, której możemy nadać nazwę **ORGATE_SET**, zamiast proponowanej. W tej grupie zadeklarujemy sygnały testowe i ich formę.

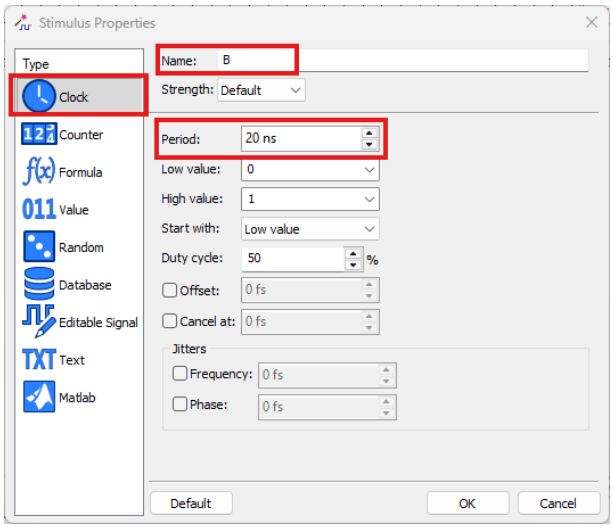
Ponownie wskazując na puste pole w okienku **Stimuli**, naciskamy prawy klawisz myszki. Pojawia się menu, w którym wybieramy **Create Stimulus…**. Otwiera się okienko deklaracyjne **Stimulus Properties**. W polu **Name** wpisujemy nazwę sygnału, czyli **A** (aby kojarzyła się z wejściem bramki OR w kodzie VHDL). Wybieramy typ zegarowy sygnału klikając na **Clock**. W polu **Period** ustawmy okres sygnału równy **10 ns**. Pozostałe opcje pozostawiamy bez zmian. Zamykamy okienko naciskając **OK**.



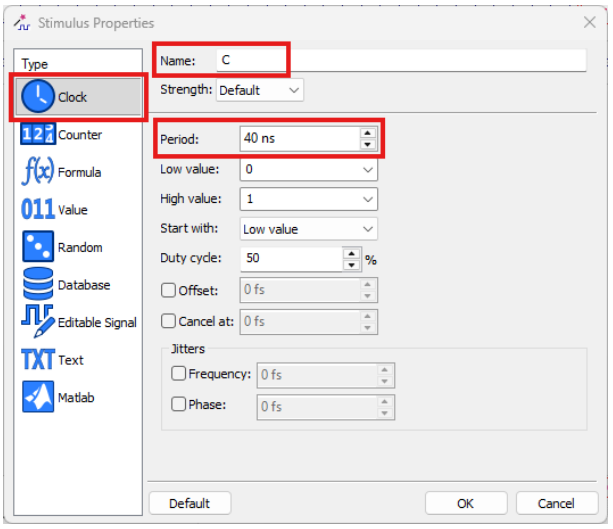
Rys. 12. Widok okna Stimulus Properties – dodanie sygnału A

Wstawiamy jeszcze dwa sygnały testowe **B** i **C**, dla pozostałych wejść bramki OR. Postępujemy podobnie jak dla sygnału A, jednak deklarując okresy sygnałów o wartościach **20 ns** i **40 ns**, aby uzyskać zmiany sygnałów testowych zgodnie z naturalnym kodem binarnym.

Uwaga!
Dla sygnałów wielobitowych typu std_logic_vector(…) można zastosować wymuszenie **Counter**. Wówczas poszczególnym bitom sygnału przypisane są odpowiednie bity wirtualnego licznika binarnego zgodnie z ich indeksami.

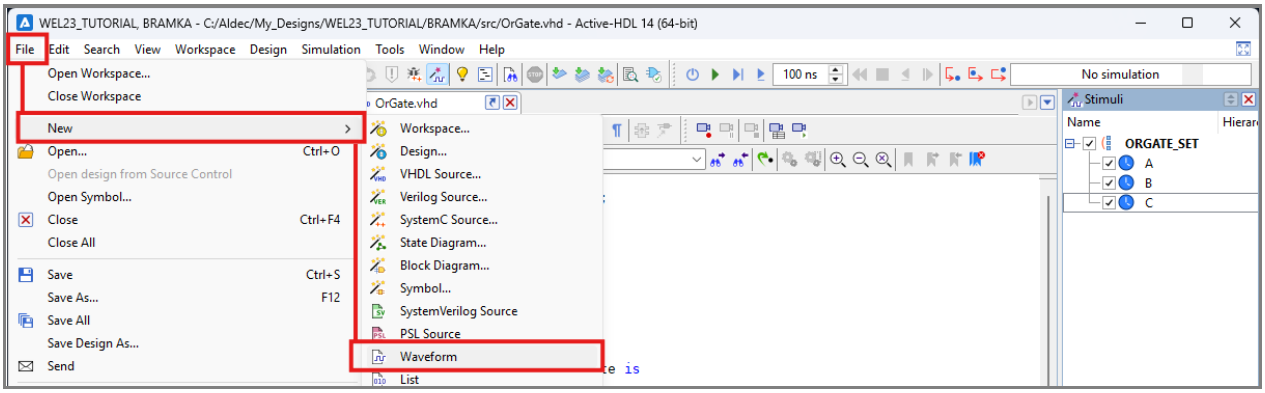


Rys. 13. Widok okna Stimulus Properties – dodanie sygnału B



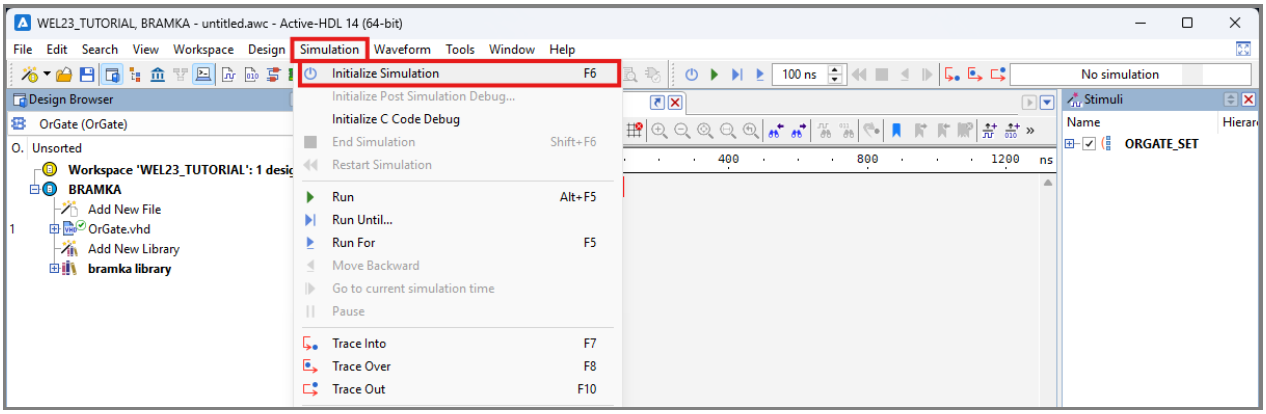
Rys. 14. Widok okna Stimulus Properties – dodanie sygnału C

W kolejnym kroku tworzymy nowy plik dla graficznego zobrazowania wyników symulacji w postaci przebiegów czasowych, wybierając **File** → **New** → **Waveform**. W głównym oknie programu Active-HDL pojawia się plik **untitled.awc**.



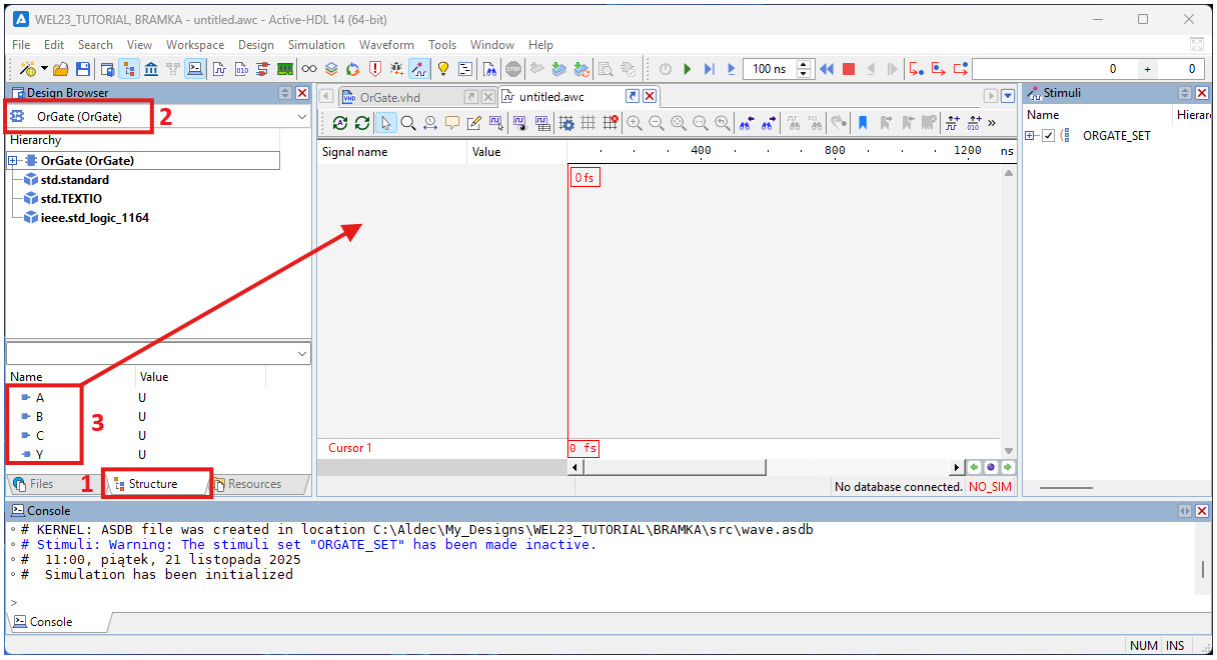
Rys. 15. Widok zakładki File / New / Waveform

Następnie inicjalizujemy symulację poprzez **Simulation** → **Initialize Simulation**, albo naciskając **F6**.



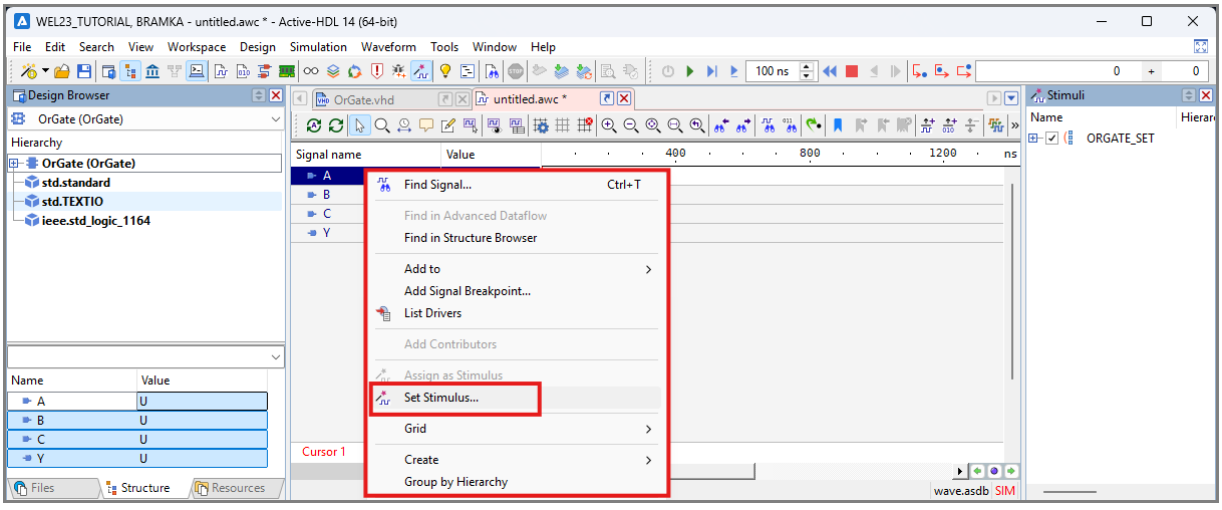
Rys. 16. Widok zakładki Simulation / Initialize Simulation

Na dole obszaru **Design Browser** wybieramy zakładkę **Structure** (1). Jeżeli wcześniej nie wskazano skompilowanego kodu VHDL do symulacji, to można zrobić to teraz wybierając **OrGate (OrGate)** (2). Następnie myszką przeciągamy sygnały do lewego obszaru pliku **untitled.awc**, jak pokazano poniżej (3).



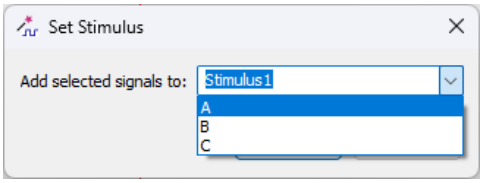
Rys. 17. Widok okna głównego - zakładka Structure

W celu nadania wartości sygnałom wejściowym zaznaczamy w pliku **untitled.awc** sygnał, np. **A**. Naciskamy prawy klawisz myszki i z rozwijanego menu wybieramy **Set Stimulus…**.



Rys. 18. Widok rozwijanego menu

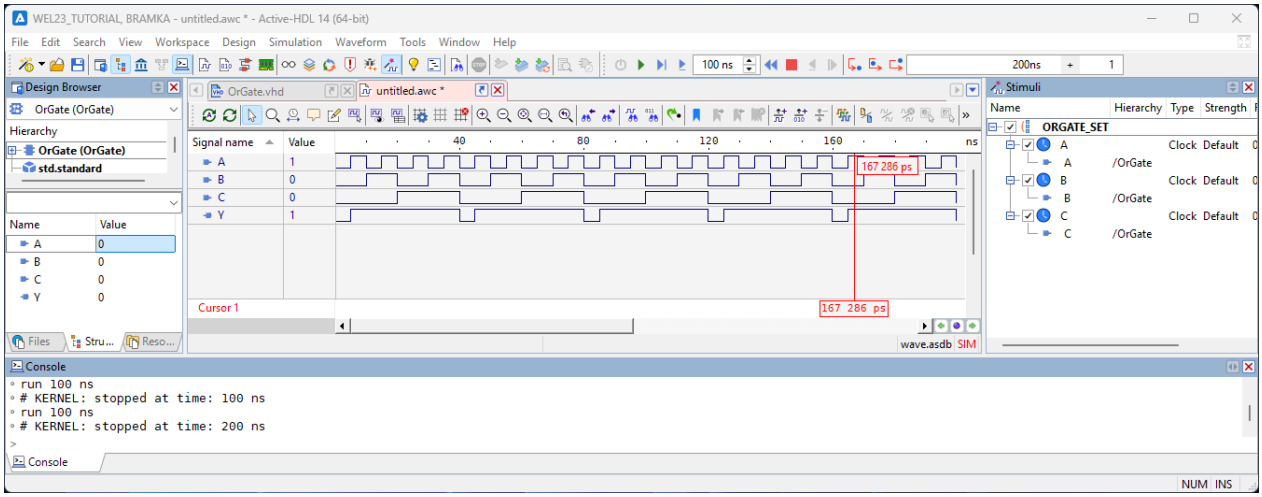
Otwiera się okienko **Set Stimulus**. Rozwijamy menu **Add selsected signals to** i wybieramy sygnał **A**. Naciskamy **OK**. Jeżeli rozwijane menu nie będzie zawierało zadeklarowanych sygnałów testowych, to należy w okienku **Stimuli** kliknąć na **ORGATE_SET**. W przypadku wielu zestawów sygnałów testowych, dodatkowo należy nacisnąć prawy klawisz myszki i wybrać **Set as Active**.



Rys. 19. Widok okna Set Stimulus

Podobnie postępujemy z sygnałami **B** i **C,** przypisując im wymuszenia testowe.

Ustawiamy krok symulacji **100 ns** w odpowiednim polu na pasku programu głównego Active-HDL. Wykonujemy krokową symulację naciskając lewy klawisz myszki na ikonie , albo naciskając F5. Ponadto restart symulacji to ikona , a jej zakończenie to . Należy zaobserwować działanie układu na podstawie uzyskanych przebiegów czasowych.



Rys. 20. Widok okna głównego – symulacja – plik „untitled.awc"

PRZYKŁADOWE ZADANIA PROJEKTOWE

Poniższe zadania należy opisać przy użyciu instrukcji: przypisania do sygnału (<=), przypisania warunkowego (when-else, if-then-else) i przypisania selektywnego (with-select, case).

1. Zaprojektować bramkę AND / NAND / OR / NOR / XOR o liczbie wejść wskazanej przez prowadzącego ćwiczenie, a następnie sprawdzić działanie z użyciem symulatora.
2. Zaprojektować układ kombinacyjny opisany zbiorem T_4={…} określonym przez prowadzącego ćwiczenie, a następnie sprawdzić działanie z użyciem symulatora.
3. Zaprojektować komparator o dwóch wejściach *n*-bitowych – liczba bitów wskazana przez prowadzącego ćwiczenie, a następnie sprawdzić działanie z użyciem symulatora.
4. Zaprojektować multiplekser o liczbie wejść danych wskazanej przez prowadzącego ćwiczenie, a następnie sprawdzić działanie z użyciem symulatora.
5. Zaprojektować demultiplekser o liczbie wyjść danych wskazanej przez prowadzącego ćwiczenie, a następnie sprawdzić działanie z użyciem symulatora.
6. Zaprojektować sumator *n*-bitowy z przeniesieniem – liczba bitów wskazana przez prowadzącego ćwiczenie, a następnie sprawdzić działanie z użyciem symulatora.
7. Zaprojektować konwerter kodu *1-z-n* na BIN o liczbie *n* wskazanej przez prowadzącego ćwiczenie, a następnie sprawdzić działanie z użyciem symulatora.
8. Zaprojektować układ realizujący podstawowe operacje logiczne (AND, NAND, OR, NOR, XOR i XNOR) w zależności od wartości sygnału sterującego, a następnie sprawdzić działanie z użyciem symulatora.
9. Z wykorzystaniem stylu strukturalnego zaprojektować układ zawierający kilka bramek logicznych (struktura wskazana przez prowadzącego ćwiczenie), a następnie sprawdzić działanie z użyciem symulatora.

IV. ZAGADNIENIA DO OPRACOWANIA PRZED PRZYSTĄPIENIEM DO ĆWICZENIA

1. Podać definicję układu kombinacyjnego.
2. Z użyciem języka VHDL opisać dwuwejściową bramkę logiczną AND/NAND/OR/NOR/XOR.
3. Wskazać różnicę pomiędzy opisem behawioralnym a strukturalnym w języku VHDL.
4. Na czym polega symulacja funkcjonalna?
5. Wyjaśnić do czego służą sekcje *entity* i *architecture* w kodzie VHDL.
6. Wymienić typy danych w języku VHDL i krótko je scharakteryzować.
7. Wymienić rodzaje obiektów w języku VHDL i krótko je scharakteryzować.

8. Wymienić rodzaje portów w języku VHDL i krótko je scharakteryzować.

V. LITERATURA I MATERIAŁY POMOCNICZE

1. Materiały wykładowe.
2. J. Kalisz: Podstawy elektroniki cyfrowej, WKŁ, Warszawa 2007.
3. J. Kalisz: Język VHDL w praktyce, WKŁ, Warszawa 2002.
4. J.F. Wakerly: Digital Design, Principles and Practices, 4th Edition, Pearson/Prentice Hall, 2005.
5. R.E. Haskell, D.M. Hanna: Introduction to Digital Design: Using Digilent FPGA Boards. LBE Books, 2009.
6. B.J. LaMeres, Quick Start Guide to VHDL, Springer Cham, 2019.
7. B.J. LaMeres, Introduction to Logic Circuits & Logic Design with VHDL, Springer Cham, 2019.