Instrukcje do zajęć laboratoryjnych

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

Laboratorium z przedmiotu:

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcje opracował:

dr inż. Marian Gilewski

Białystok 2025

Fundusze Europejskie dla Rozwoju Społecznego

Rzeczpospolita Polska

Dofinansowane przez Unię Europejską

Materiały przygotowane w ramach projektu „PB 5.0 – dostosowanie oferty dydaktycznej Politechniki Białostockiej do potrzeb nowoczesnej gospodarki oraz zielonej i cyfrowej transformacji” (nr umowy FERS.01.05-IP.08-0327/23-00) w ramach programu Fundusze Europejskie dla Rozwoju Społecznego 2021-2027 współfinansowanego ze środków Europejskiego Funduszu Społecznego Plus.

INSTRUKCJA STANOWISKOWA BHP

Dotyczy zasad użytkowania stanowiska do ćwiczeń laboratoryjnych z zakresu Układy cyfrowe i programowalne w sali WE-244

Dotyczy zasad użytkowania następujących urządzeń:
- oscyloskop cyfrowy,
- laboratoryjny zasilacz regulowany,
- komputer stacjonarny,
- komercyjne moduły ewaluacyjne układów FPGA.

W celu uniknięcia obrażeń, porażenia prądem elektrycznym lub uszkodzenia użytkowanych urządzeń, zaleca się przestrzeganie poniższych uwag eksploatacyjnych z zakresu bezpieczeństwa pracy. Użytkowanie urządzenia lub stanowiska powinno być zgodne z jego instrukcją obsługi oraz odrębnymi instrukcjami obsługi elementów składowych, jeżeli takie występują. Wszelkie czynności serwisowe (naprawy, regulacje, wymiany bezpieczników, itp.) powinny być wykonywane jedynie przez odpowiednio wykwalifikowane osoby.

Przed przystąpieniem do pracy na stanowisku należy:
- zapoznać się z instrukcją obsługi użytkowanych urządzeń;
- zapewnić poprawne uziemienie przyrządu, realizowane przez przewód ochronny kabla sieciowego podłączony do sprawnego gniazdka sieciowego z kołkiem uziemiającym;
- używać właściwego kabla sieciowego zaprojektowanego dla urządzenia i spełniający odpowiednie normy krajowe;
- upewnić się, że urządzenie jest prawidłowo uziemione przed wykonaniem jakichkolwiek połączeń wyjść i wejść urządzenia;
- zapewnić wymagane chłodzenie przyrządu poprzez prawidłowy obieg powietrza chłodzącego przyrząd.

Aby uniknąć ryzyka pożaru lub porażenia prądem, należy zwracać uwagę na wszelkie ostrzeżenia na obudowie przyrządu i nie przekraczać podanych w instrukcji oraz na obudowie dopuszczalnych wartości napięć i prądów w jego gniazdach.

Podczas pracy na stanowisku laboratoryjnym należy:
- zachować porządek - przyrządy powinny być bezpiecznie ustawione, aby nie mogły spaść;
- wykonywać połączenia tak, aby nie zagrażały użytkownikom stanowiska;
- należy unikać połączeń przewodami zwisającymi lub leżącymi na podłodze;
- prawidłowo łączyć i rozłączać kable/sondy pomiarowe - nie należy tego wykonywać, gdy znajdują się one pod wysokim napięciem z punktu widzenia ochrony przeciwporażeniowej;
- nie pracować z urządzeniami o zdjętych obudowach lub zdemontowanych panelach,
- nie dotykać przewodzących elementów obwodu (gniazd, styków, podzespołów; nieizolowanych przewodów, itp.) mogących znaleźć się pod napięciem stanowiącym zagrożenie z punktu widzenia ochrony przeciwporażeniowej;
- dbać, aby powierzchnia przyrządu była zawsze czysta i sucha;
- nie spożywać posiłków i/lub pić napojów na stanowiskach laboratoryjnych.

Jeżeli zachodzi podejrzenie uszkodzenia urządzenia lub jego nieprawidłowej pracy, należy wyłączyć zasilanie a fakt ten zgłosić osobie prowadzącej zajęcia. Przed przystąpieniem do dalszej pracy urządzenie takie powinno być sprawdzone przez wykwalifikowaną osobę.

Po skończeniu pracy na stanowisku należy:
- wyłączyć zasilanie przyrządów i układów badanych,
- uporządkować stanowisko.

Czynności zabronione:
- używanie urządzeń niezgodnie z ich przeznaczeniem i instrukcją obsługi,
- łączenie elektrycznych obwodów pomiarowych znajdujących się pod napięciem stanowiącym zagrożenie z punktu widzenia ochrony przeciwporażeniowej,
- samowolne otwieranie obudów i naprawianie urządzeń.

Pierwsza pomoc:
- w przypadku porażenia prądem elektrycznym należy natychmiast wyłączyć napięcia zasilające stanowiska za pomocą przycisku bezpieczeństwa tablicy zasilającej,
- udzielić pomocy przedlekarskiej osobom poszkodowanym,
- w sytuacji nieszczęśliwego wypadku należy wezwać Pogotowie Ratunkowe (tel. 999 / 112).

W przypadku pożaru należy:
- odłączyć zasilanie urządzeń,
- rozpocząć ewakuację ludzi z zagrożonego obszaru.
- gasić urządzenia dostępnymi i zalecanymi środkami ochrony p. poż.,
- w koniecznym przypadku wezwać Straż Pożarną (tel. 998, kom. 112).

Sprawozdanie studenckie z ćwiczeń laboratoryjnych powinno zawierać:
- stronę tytułową sprawozdania obowiązującą na Wydziale Elektrycznym,
- opis zakresu realizowanych zadań wyrażony własnymi słowami,
- charakterystykę techniki rozwiązania zadania w postaci opisanych kodów i/lub schematów,
- wyjaśnienie sposobu weryfikacji/pomiaru wyników badań,
- interpretację/opis uzyskanych wyników z pomiarów lub symulacji,
- wnioski końcowe dotyczące badanych układów.

W przypadku stwierdzenia plagiatu sprawozdania z laboratorium - ocena końcowa z ćwiczenia jest dzielona przez liczbę zespołów zamieszczających te same treści.

W trakcie realizacji ćwiczeń nie należy korzystać z urządzeń technicznych i sieci internetowej w celu nieuprawnionego pozyskiwania lub przekazywania rozwiązań/danych.

W związku z tym, że tematykę kolejnych ćwiczeń cechuje efekt stopniowanej złożoności, przed rozpoczęciem nowego ćwiczenia muszą być zrealizowane założone zadania z poprzedniego. Zadania niezrealizowane w czasie laboratorium z powodu braku czasu, mogą być uzupełnione domowymi badaniami symulacyjnym. Instrukcja zawiera maksymalny zakres ćwiczeń/zadań, rzeczywisty zakres wykonanych badań będzie korygowany na bieżąco, uwzględniając stan techniczny, przypadki losowe i możliwości/sprawność grupy studenckiej.

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 1

Obsługa platformy projektowej Quartus II - definiowanie komponentów VHDL

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcję opracował:

dr inż. Marian Gilewski

Białystok 2025

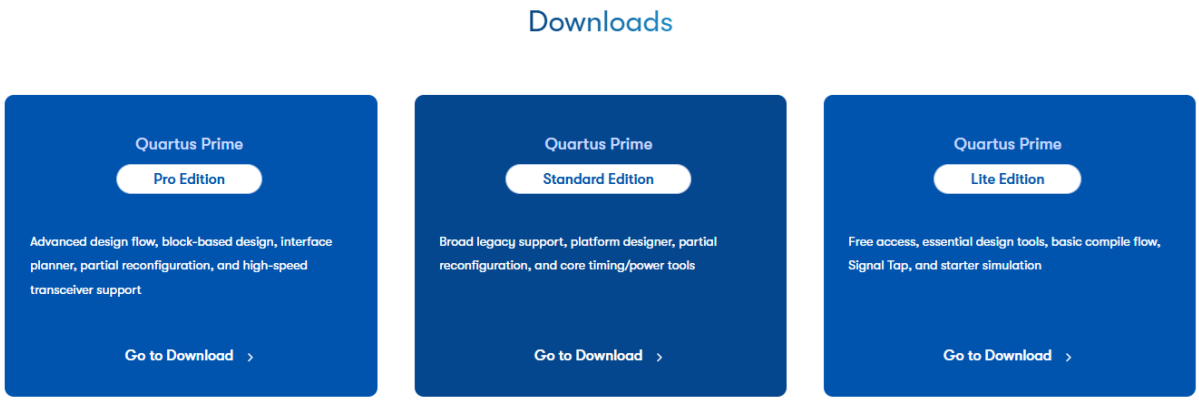
1. Cel ćwiczenia.

Celem ćwiczenia jest nabycie podstawowych umiejętności związanych z obsługą platformy projektowej Quartus II, która będzie wykorzystywana do realizacji dalszych ćwiczeń laboratoryjnych. Jest to oprogramowanie ogólnodostępne, które można wykorzystać w domu do przygotowania zadań na zajęcia laboratoryjne. W ramach zajęć będzie doskonalony pełny cykl projektowy w systemie CAD PLD – od zdefiniowania układu za pomocą schematu/kodu, poprzez weryfikację opisu, badania symulacyjne oraz implementację sprzętową i testowanie,

2. Instalacja oprogramowania na własnym komputerze.

Pakiet Quartus II jest systemem CAD zorientowanym na wspomaganie projektowania i uruchamiania systemów cyfrowych w strukturach programowalnych firmy Intel. Używana w laboratorium wersja umożliwia obsługę dostępnych w laboratorium modułów prototypowych DE2-115. Proces instalacji ***Intel® Quartus® Prime Lite Edition*** jest następujący:

1. Należy otworzyć stronę internetową producenta:
https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime/resource.html
- i pobrać wersję **Lite Edition for Windows** (free, no license required) – widok jak niżej:



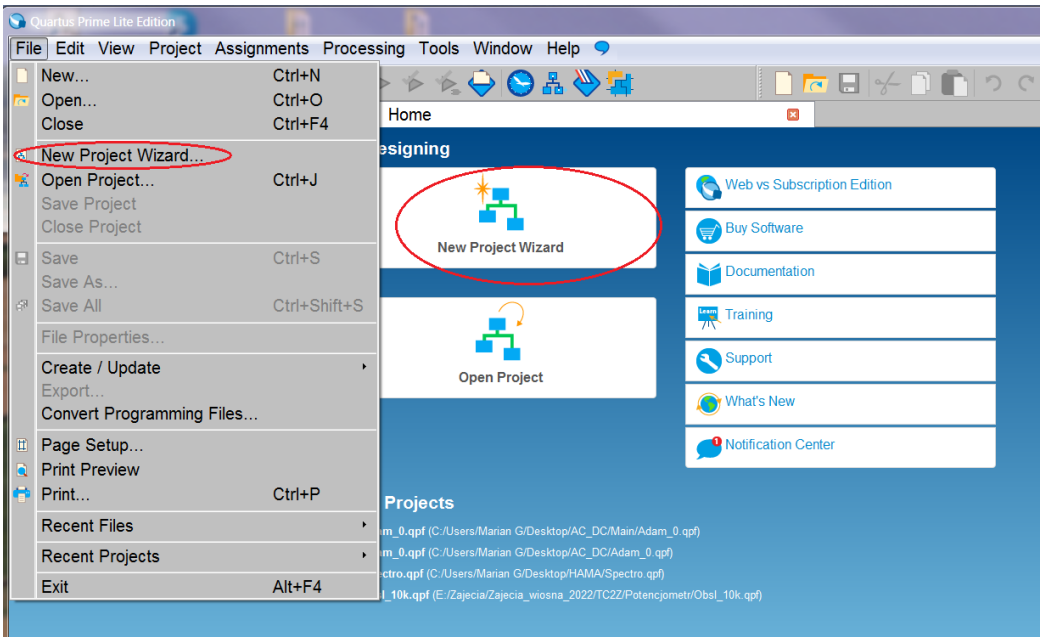
2. Sugerowana jest instalacja wersji 18.1, gdyż obsługuje ona moduły DE2-115 i symuluje układy Cyclone IV, które będą obsługiwane w laboratorium.

Nie jest konieczne pobieranie kompletnego archiwum – niezbędne są 3 pliki instlacyjne:
- plik startowy instalatora: QuartusLiteSetup-18.1.0.625-windows.exe,
- plik symulatora: ModelSimSetup-18.1.0.625-windows.exe,
- biblioteka rodziny FPGA Cyclone IV: cyclone-18.1.0.625.qdz.

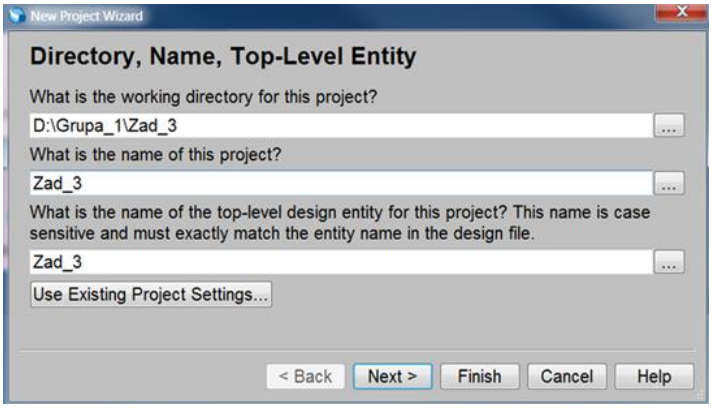
Po pobraniu indywidualnych plików lub rozpakowaniu kompletnego archiwum - należy uruchomić plik instalacyjny QuartusLiteSetup i postępować zgodnie z wbudowaną instrukcją.

3. Obsługa projektów w Quartus II.

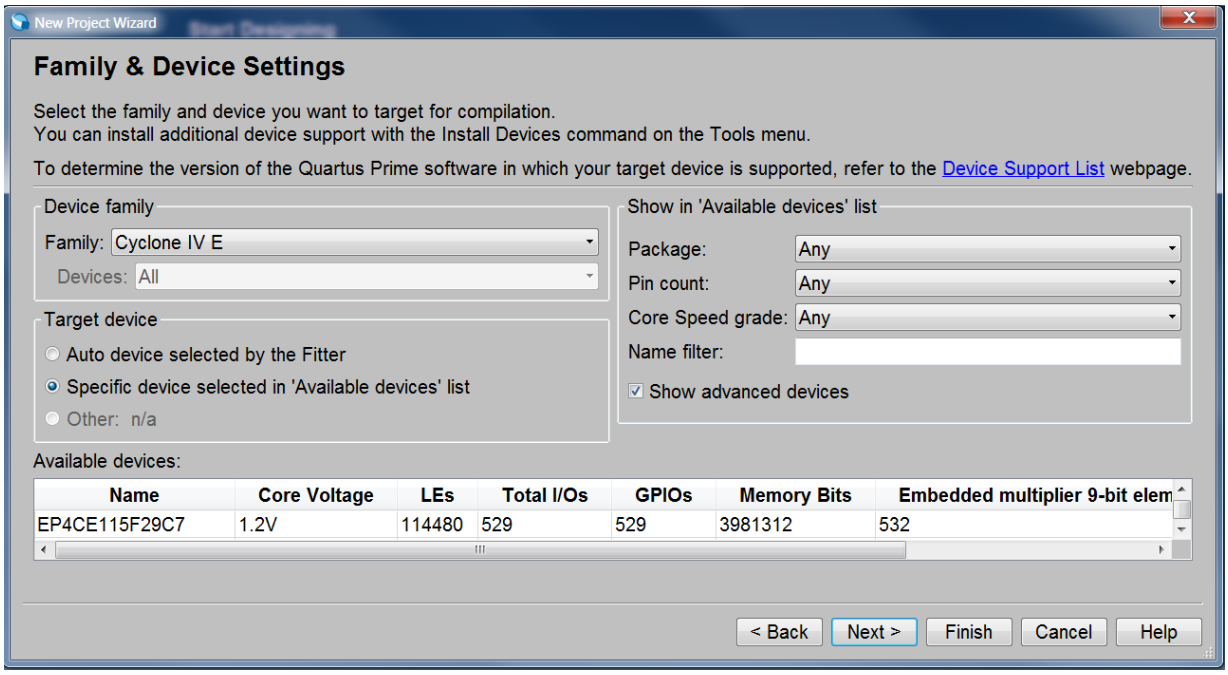
W używanym systemie obowiązuje hierarchiczny układ plików. Najważniejszym plikiem projektu jest plik typu top-level, do którego odnoszą się wszystkie czynności projektowe: kompilacja, symulacja lub programowanie modułu prototypowego. Stąd na początku pracy **zasadą jest** utworzenie nowego projektu lub otwarcie istniejącego - z plikiem top-level.
Niewskazane jest otwieranie poprzez kliknięcie istniejącego lub nowego pliku, bez wcześniejszego zainicjowania projektu – gdyż taka kolejność uniemożliwia syntezę układu.
Inicjowanie nowego projektu rozpoczynamy z menu powitalnego programu lub z grupy funkcji **File** używając opcji **New Project Wizard**:



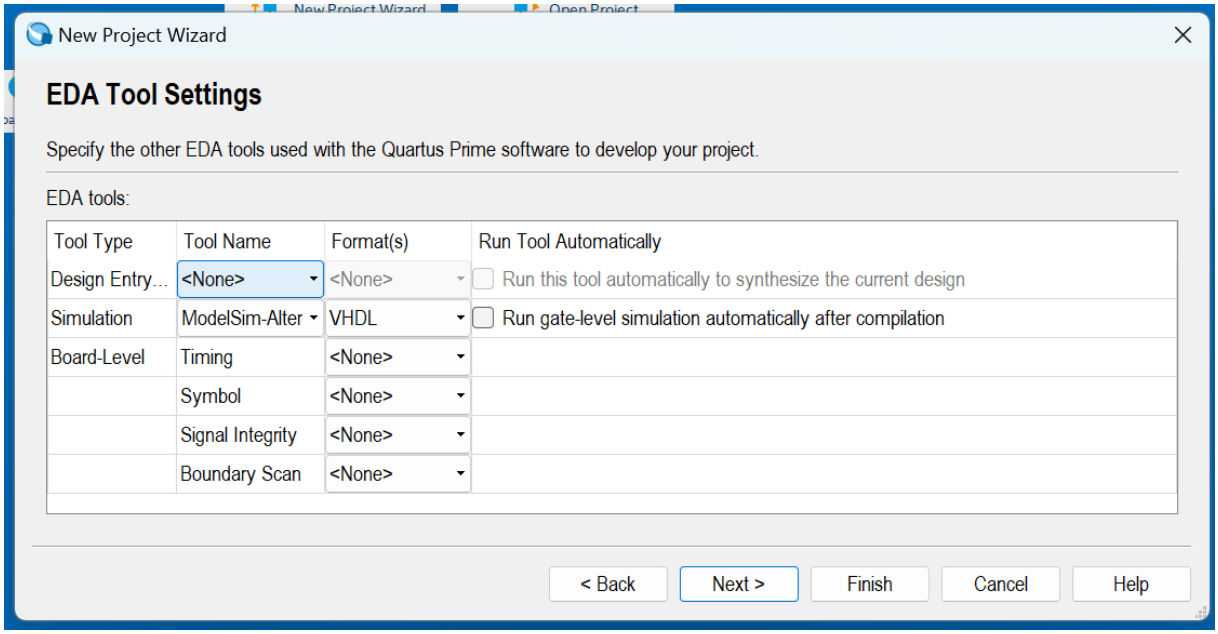
W kolejnym oknie wpisujemy:
- nazwę katalogu/folderu roboczego projektu; nazwę projektu oraz nazwę pliku top-level.
W laboratorium **obowiązuje zasa**da umieszczania folderów projektowych na dysku D. Zabronione jest umieszczanie ich na Pulpicie lub dysku systemowym. Jeżeli nowy folder nie istnieje, zostanie on utworzony w czasie inicjacji projektu zgodnie z podaną nazwą.
Zasadą jest, iż nazwa folderu i nazwa pliku top-level, oraz wszystkie inne nazwy w projekcie **powinny zaczynać się literą, następnie mogą wystąpić litery i/lub cyfry oraz ewentualnie podkreślenia w środku nazwy**.
Nie wolno używać innych znaków lub zaczynać cyfrą nazwy plików i folderów.



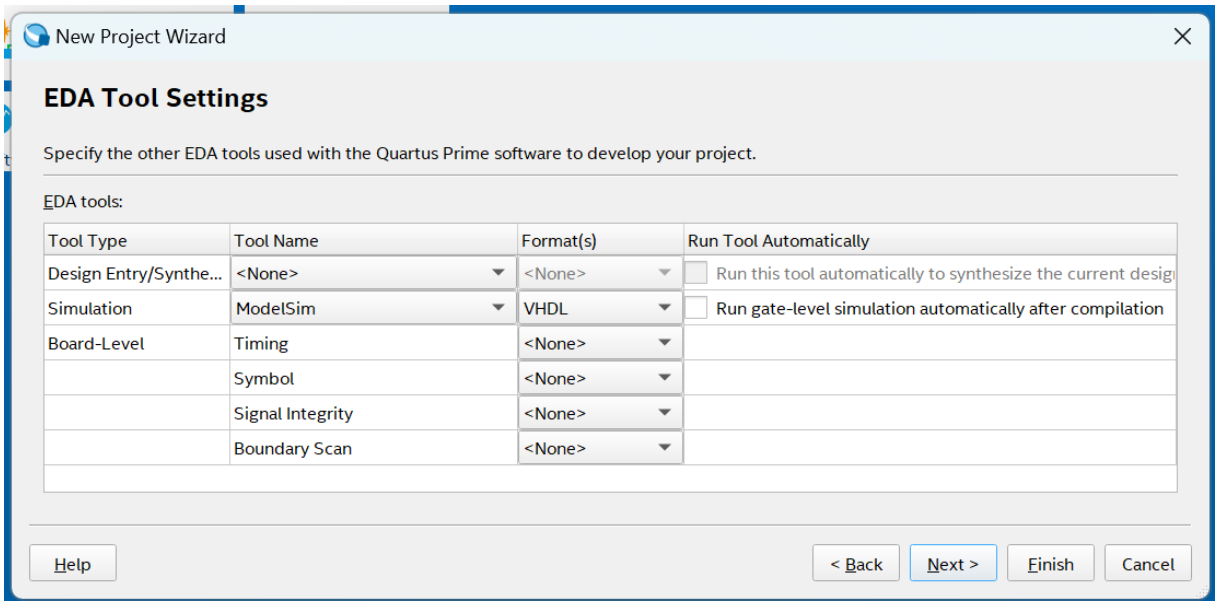
Modyfikacje ustawień w kolejnych oknach: **Project Type** oraz **Add Files** pomijamy (klikając Next). Następnie wskazujemy docelowy układ scalony FPGA modułu laboratoryjnego ustawiając:
Family = **Cyclone IVE**,
Target device = **Specific** device selected in 'Available devices' list,
Available devices: = **EP4CE115F29C7 (**Quartus ver. 18**)** lub **EP4CE115F29C8L** (ver. 22).



Ustawień kolejnego okna **EDA Tool Settings** nie zmieniamy - pod warunkiem, że:
- w wersji 18 programu Quartus II (dokładnie ver. 18.1.0) mamy ustawione **Simulation** = **ModelSim-Altera**, a w **Format(s)** danych mamy **VHDL**,

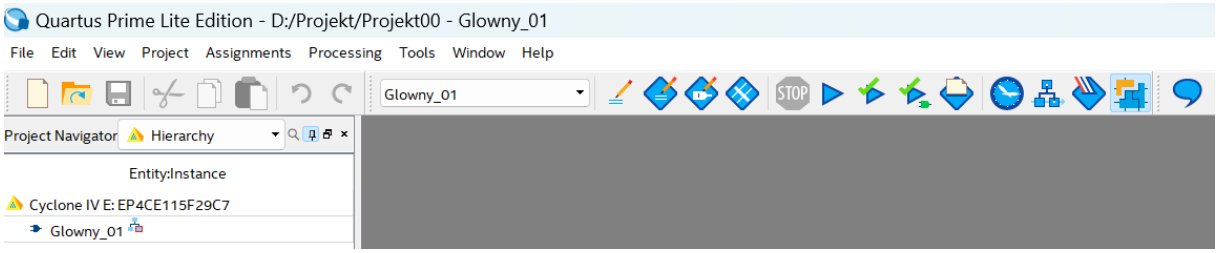


- lub w Quartus II wersja 22 (dokładnie ver. 22.1std.2) mamy: **Simulation** = **ModelSim**, **Format(s)** = **VHDL**

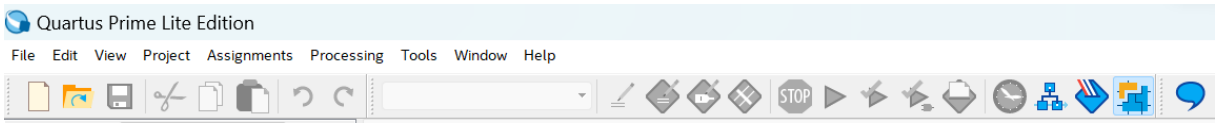


co potwierdza, że w systemie w obu przypadkach mamy zainstalowany ten sam symulator, ModelSim-Altera w wersji 18. W wersji Quartus 22 jest dostępny inny symulator Questa, jest on jednak bardziej skomplikowany w obsłudze, a wyniki symulacji są mniej czytelne. W związku z tym, dla zachowania spójności danych użytkowników pracujących w obu wersjach, w czasie zajęć będziemy korzystać z symulatora ModelSim-Altera.

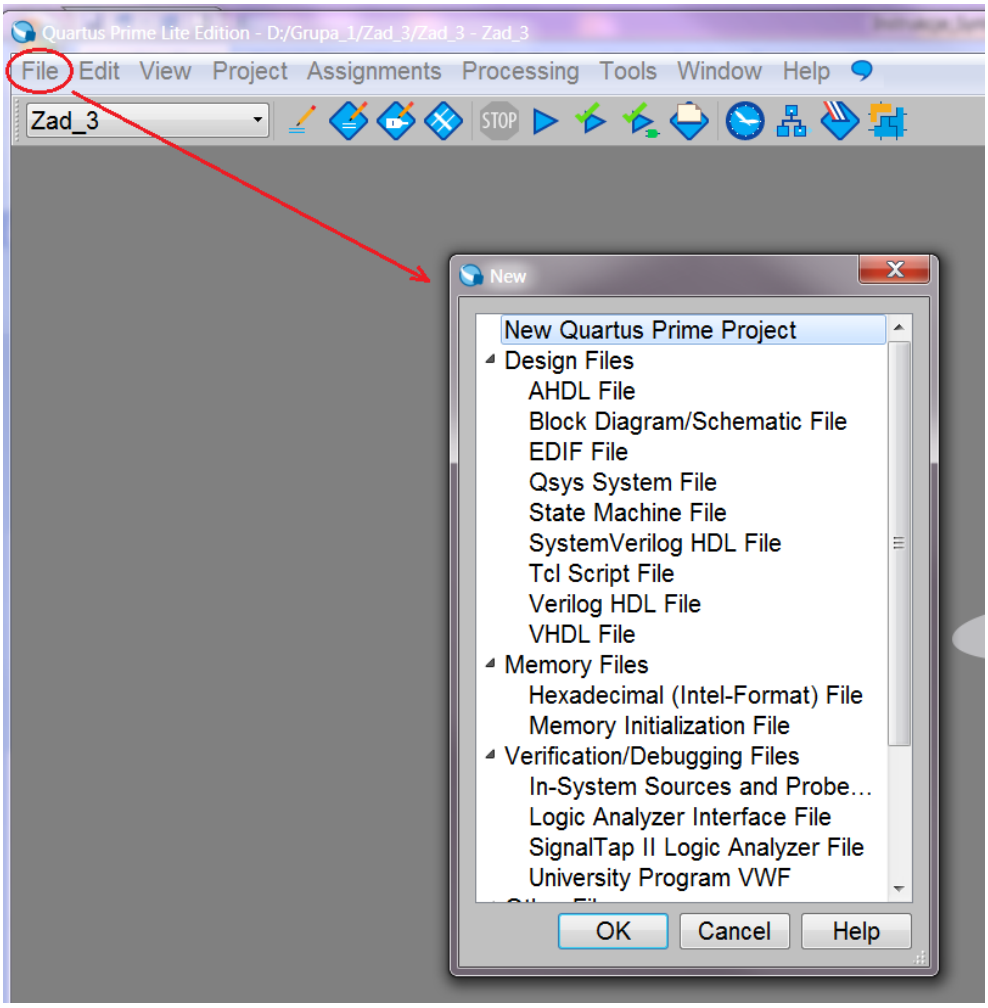
W kolejnym oknie podsumowującym **Summary** akceptujemy dotychczasowe wybierając **Finish**.
O prawidłowym otwarciu/inicjacji projektu świadczy stan pierwszego wiersza widocznego na ekranie, który pokazuje nazwę katalogu roboczego (D:/Projekt), nazwę aktywnego projektu (Projekt00) oraz nazwę pliku top-level (Glowny_01) w tym projekcie.



Poniższy widok świadczy o błędzie - braku projektu, stąd dalsza praca jest niecelowa:

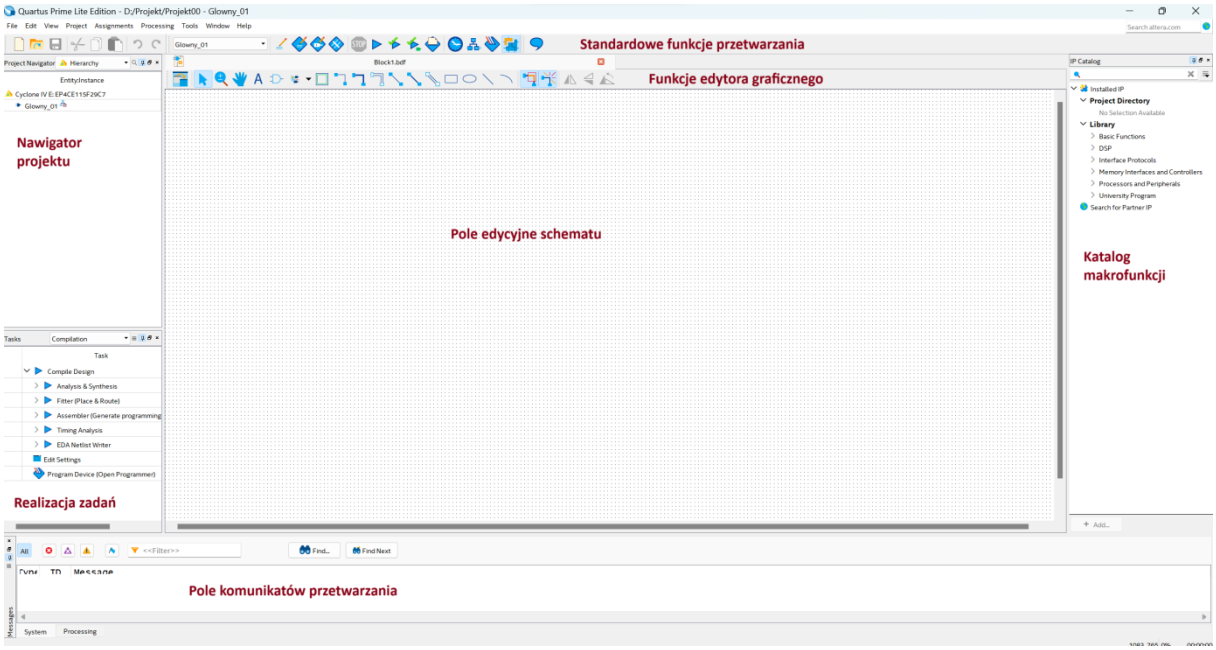


Po skutecznej inicjacji projektu, system jest przygotowany do edycji nowego pliku opisującego badany układ.
Takim plikiem w zależności od planowanej formy opisu, może być:
- schemat, czyli **Block Diagram/Schematic File**;
- kod w języku VHDL, czyli **VHDL File**.

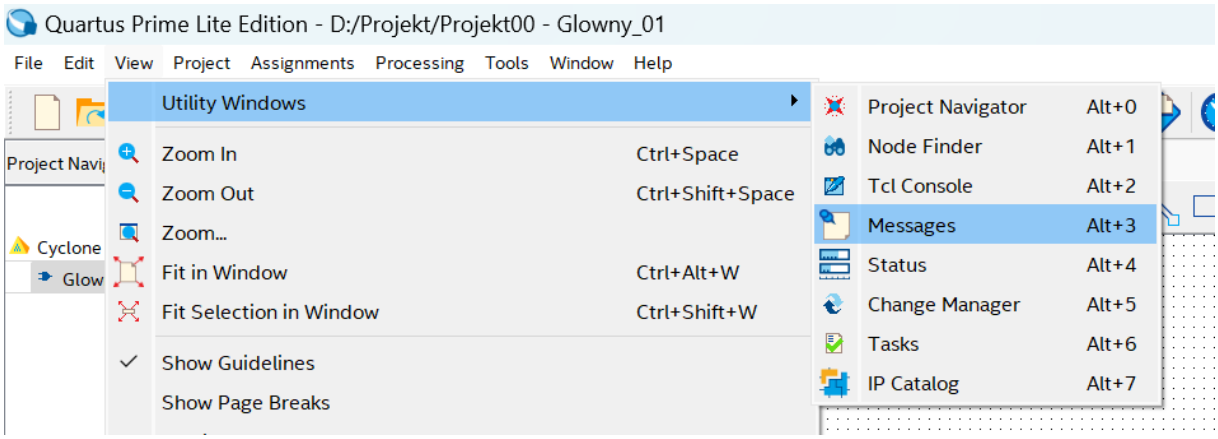


4. Definiowanie układu z użyciem edytora graficznego w Quartus II.

W celu rozpoczęcia rysowania schematów w edytorze należy uruchomić moduł **Block Diagram / Schematic File** – w efekcie czego otworzy się okno edycyjne w typowym ustawieniu:



W zależności od potrzeb, można zmieniać widoczność poszczególnych pól/menu, używając funkcji View → Utility Windows:



W trakcie rysowania schematu, najważniejszymi funkcjonalnościami edytora są:
- pole edycyjne,
- funkcje edytora graficznego,
- katalog makrofunkcji (IP).

W następnych etapach związanych z przetwarzaniem projektu, przydatne są:
- standardowe funkcje przetwarzania,
- pole komunikatów przetwarzania.

Po narysowaniu schematu, ważne jest zapisanie pliku. Jeżeli projekt jest prosty i zawiera jeden plik graficzny, zapisujemy go pod nazwą pliku top-level. W przypadku złożonych projektów hierarchicznych, nazwy plików mogą być dowolne - z zachowaniem zasad nazewnictwa podanych części 3.

Do rysowania schematów użyteczne są poniższe funkcje menu edytora graficznego:

 biblioteka elementów prostych/prymitywnych,

 rysowanie pojedynczej linii ortogonalnie - wire,

 rysowanie magistrali - bus, czyli zestawu linii – np. 8 wires

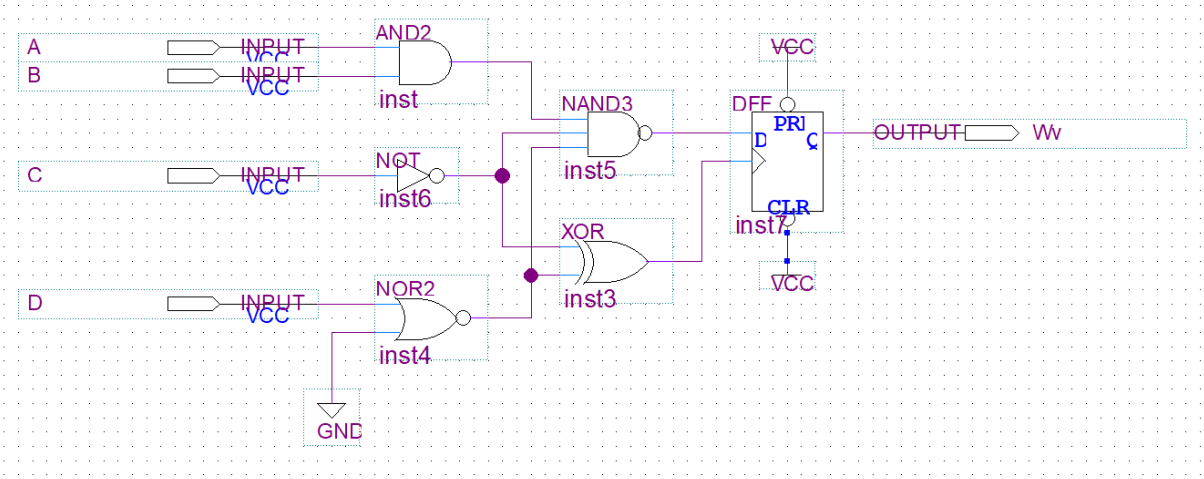
 rysowanie wejść/wyjść układu - czyli portów,

left-click mouse zaznaczanie obiektu/elementu,

right-click mouse obracanie elementów / edycja parametrów / etykietowanie.

Zadanie laboratoryjne 1.1

Uwzględniając wcześniejsze informacje, proszę utworzyć nowy projekt, otworzyć nowy plik graficzny i narysować poniższy schemat. Plik proszę zapisać a projekt zamknąć.

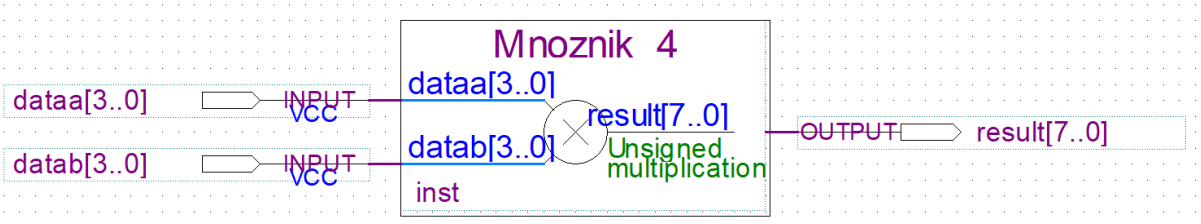


5. Definiowanie układu z użyciem makrofunkcji katalogu IP.

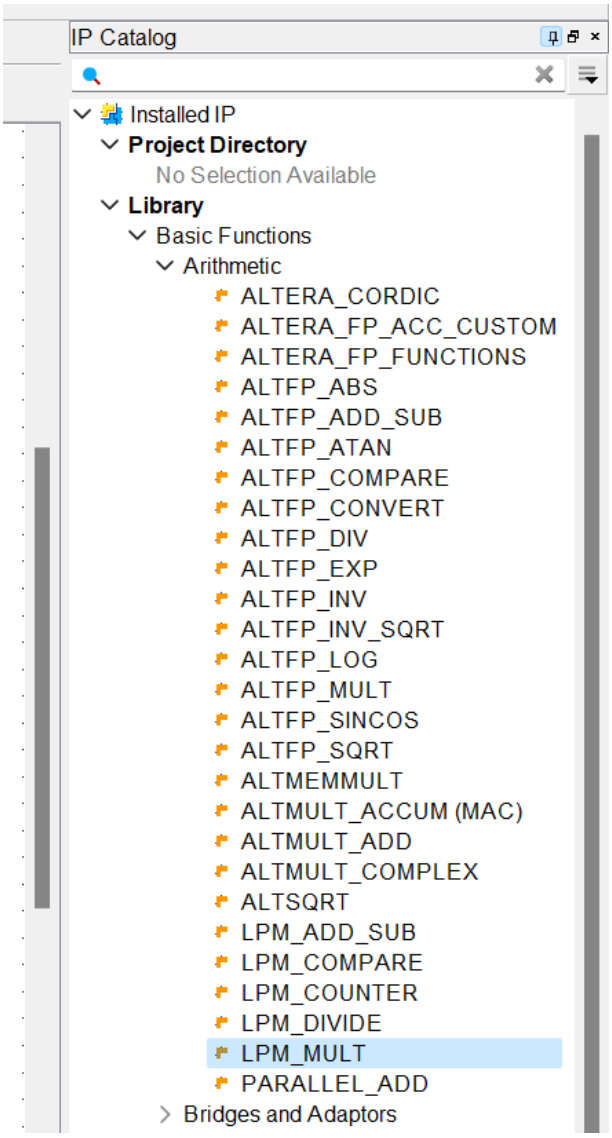
W systemie projektowym znajduje się bogaty katalog IP modeli złożonych elementów, tzw. makrofunkcji. Są to uniwersalne modele opisujące złożone moduły, dla który użytkownik definiuje jedynie parametry zaciskowe, czyli konfiguruje układ „na wymiar”. Sposób wykorzystania tych elementów prześledzimy na przykładzie mnożnika 4-bitowego.

Zadanie laboratoryjne 1.2

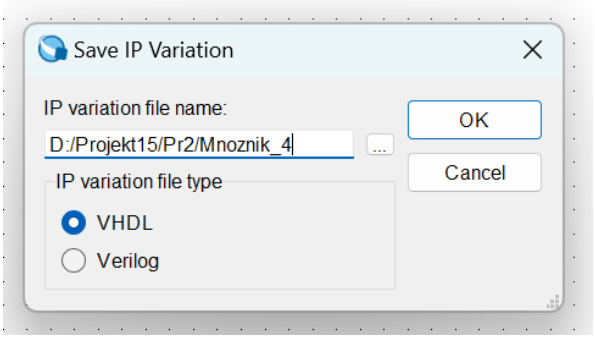
Stosując element LPM_MULT z katalogu makrofunkcji IP proszę zdefiniować moduł mnożnika 4-bitowego i narysować jego schemat przedstawiony poniżej.



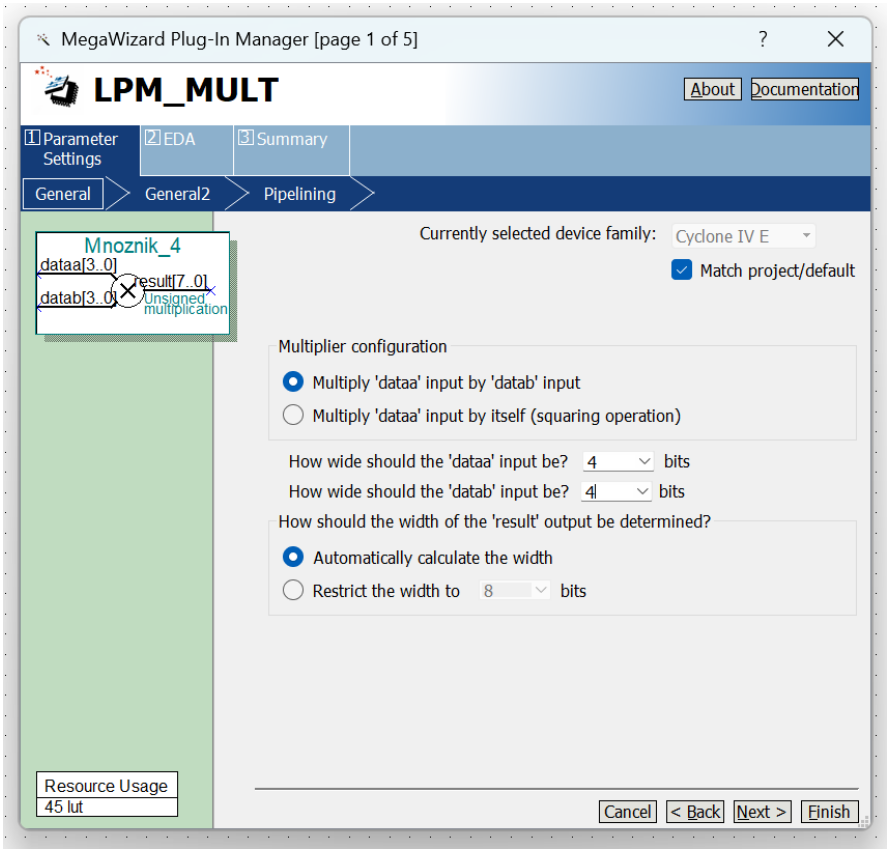
W tym celu:
a) tworzymy w nowym (nie z Zadania 1) folderze nowy projekt i otwieramy plik graficzny,
b) w katalogu makrofunkcji wyszukujemy element LPM_MULT – dwukrotnie nań klikając,



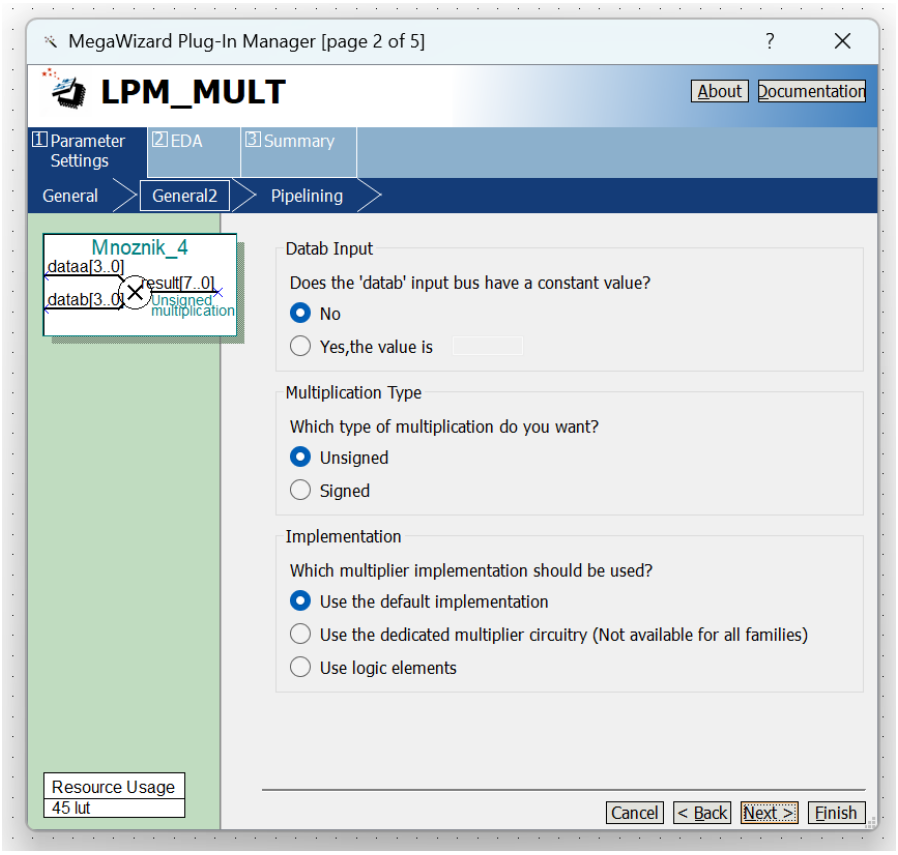
c) następnie w Save IP Variation dopisujemy nazwę tworzonego elementu (np. Mnoznik_4), jako język zaznaczając VHDL - **nie stosujemy nigdy** nazw już użytych w projekcie,



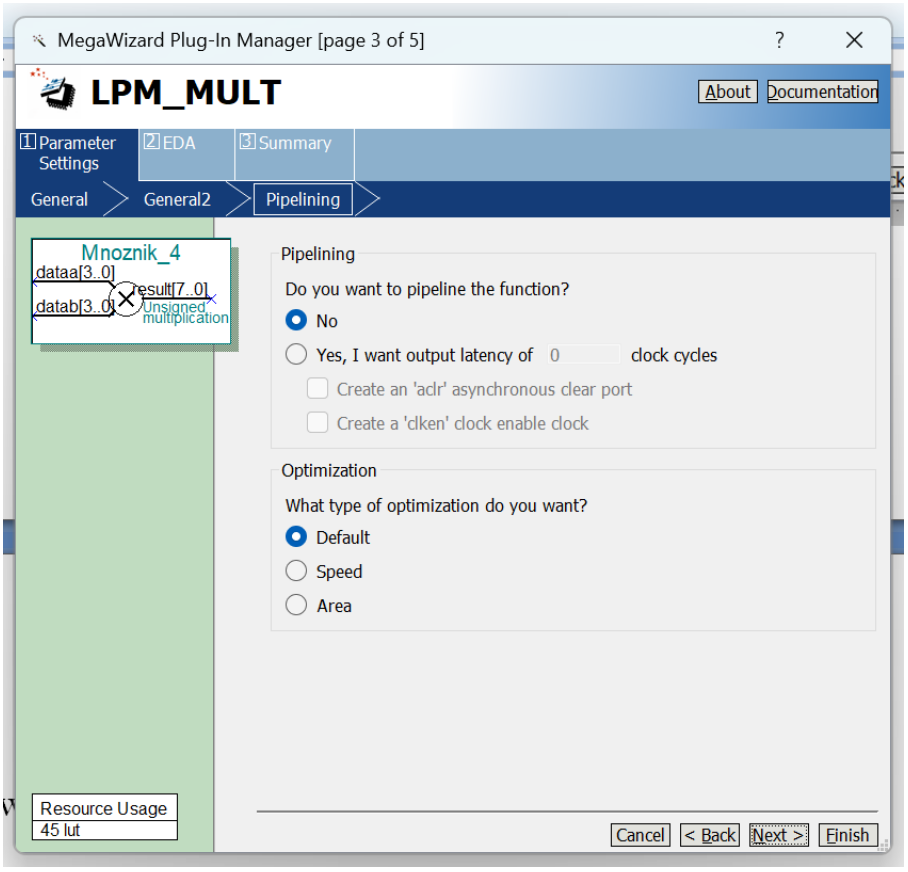
d) w oknie konfiguratora ustawiamy parametry jak na poniższym rysunku:



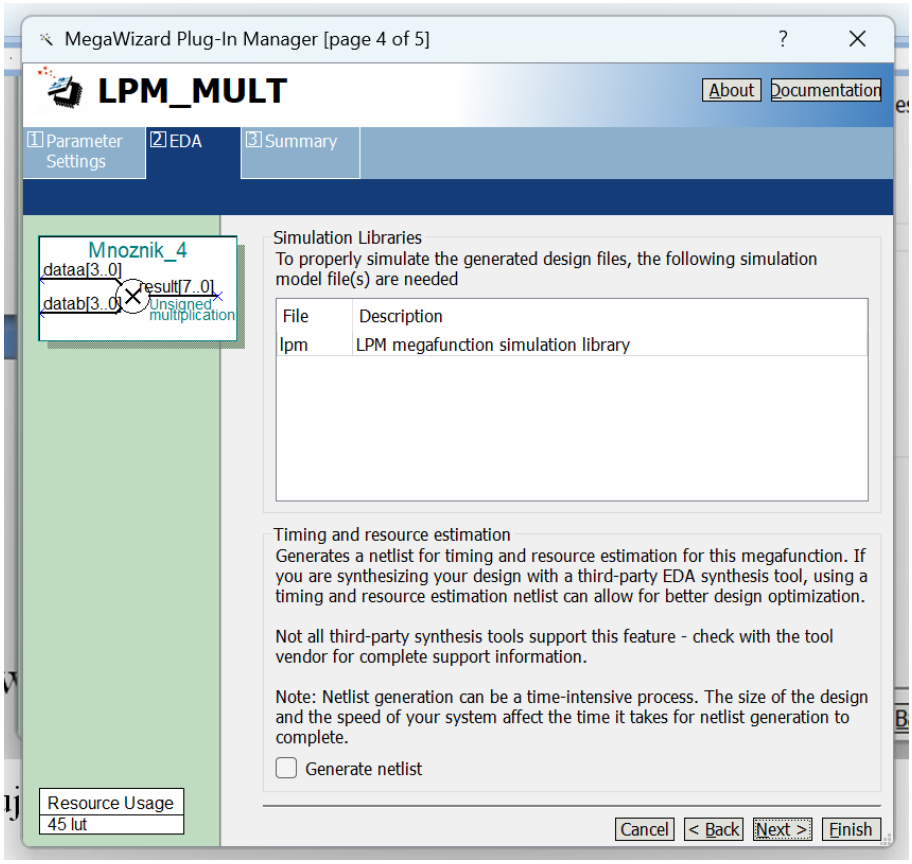
e) w kolejnym oknie nie zmieniamy ustawień domyślnych/sugerowanych:



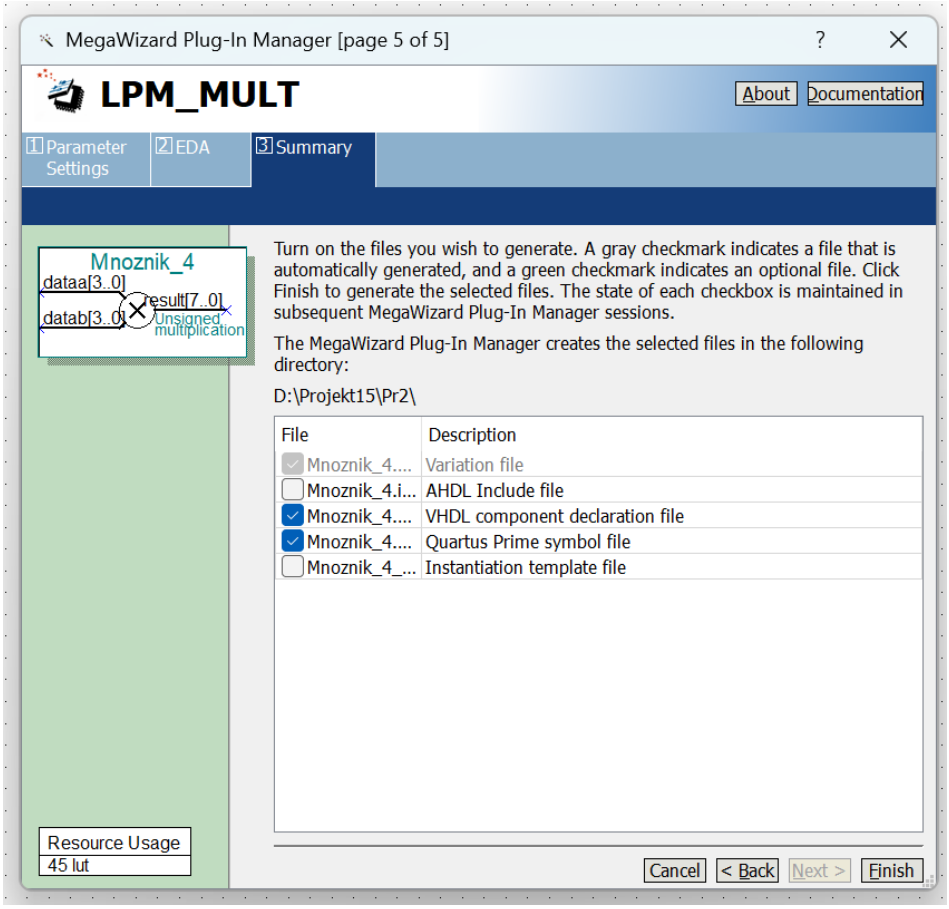
f) pozostawiamy również ustawienia domyślne kolejnego okna,



g) akceptujemy również ustawienia symulatora:

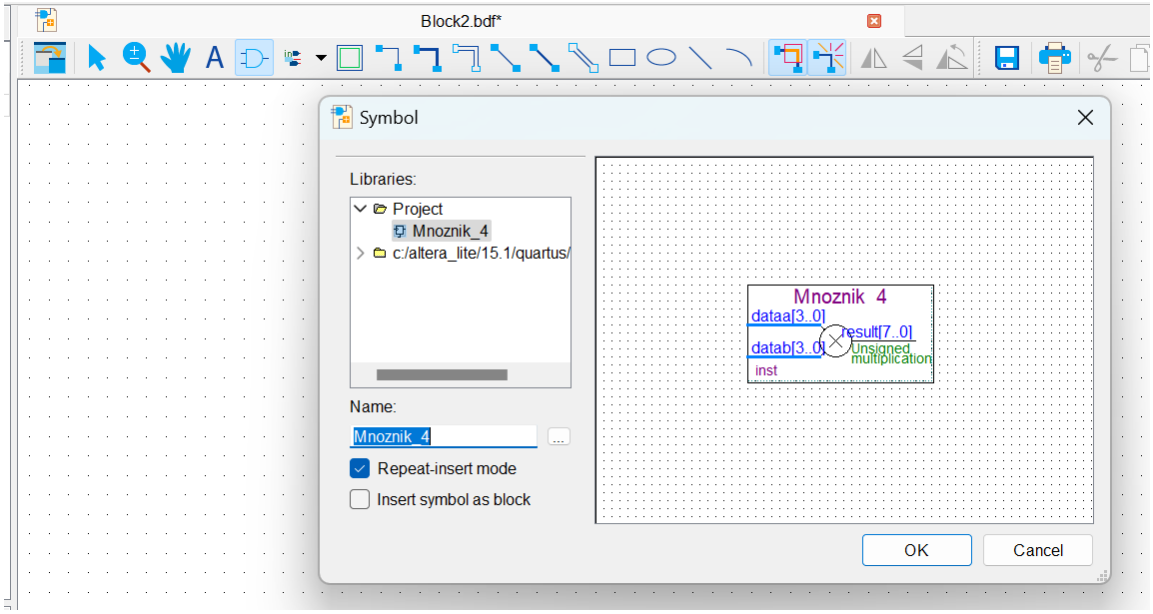


h) **ważne** – w ostatnim oknie **zawsze** zaznaczamy klucz: **Quartus Prime symbol file**,



i) następnie akceptujemy generację makromodelu IP,

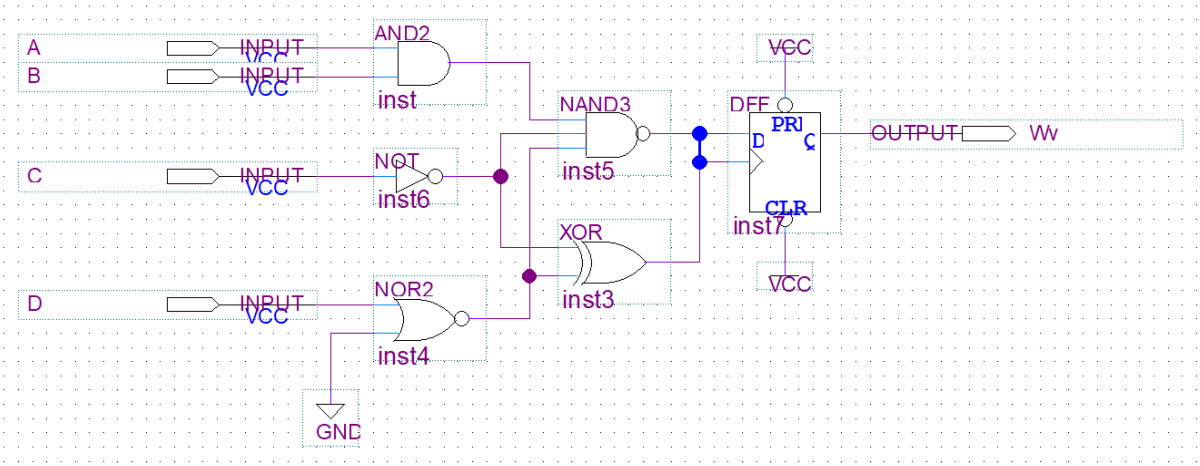
j) po powrocie do edytora projektowego, za pomocą menu graficznego (symbol bramki AND) w podkatalogu Project wskazujemy nasz nowy element i przenosimy go w pole edycyjne schematu,



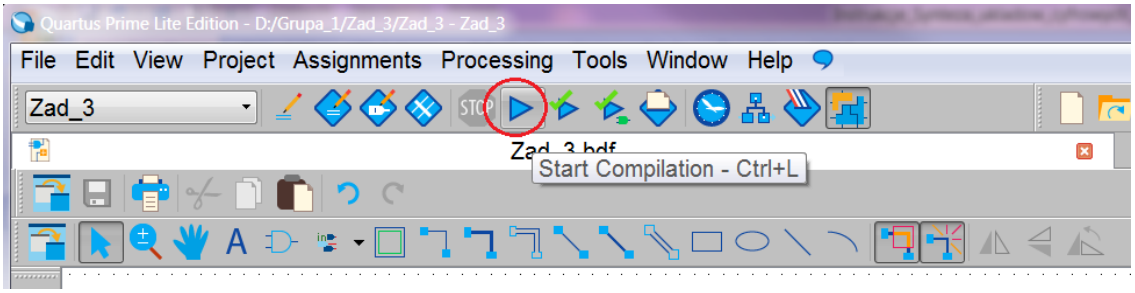
k) dodajemy brakujące porty wejściowe i wyjściowe, zapisujemy plik i zamykamy projekt.

6. Źródłowa kompilacja projektu.

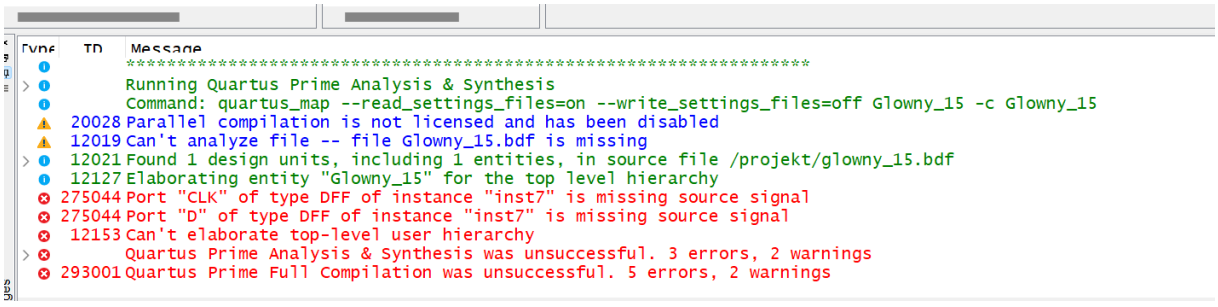
Celem kompilacji jest sprawdzenie poprawności opisu projektu i wygenerowanie wewnętrznych połączeń w strukturze programowalnej. Stąd w procesie projektowym występują dwie kompilacje – źródłowa i końcowa po zdefiniowaniu wyprowadzeń. W tej części dokonamy kompilacji źródłowej, która sprawdzi spójność narysowanego schematu i wykryje ewentualne błędy połączeniowe. W tym celu należy otworzyć projekt z zadania 1 i związany z nim plik graficzny, pokazany na stronie 11. W tym schemacie celowo spowodowano błąd na schemacie – zwierając wejścia przerzutnika DFF.



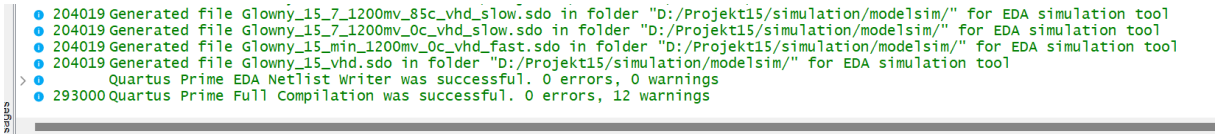
Po uruchomienia kompilatora z menu standardowych funkcji przetwarzania:



W polu komunikatów pojawia się informacja o błędzie występującym na schemacie:

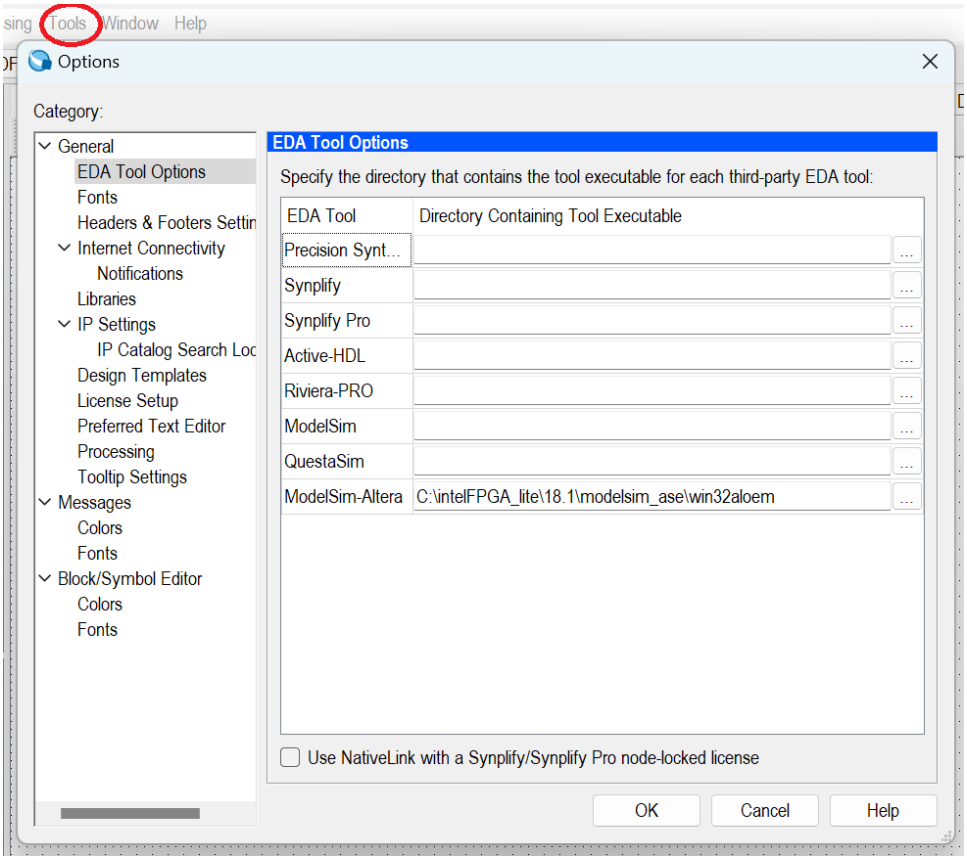


po usunięciu zwarcia i uruchomieniu ponownej kompilacji – zakończy się ona sukcesem, co oznacza, że schemat jest poprawny w zakresie połączeń i można przejść do następnego etapu:

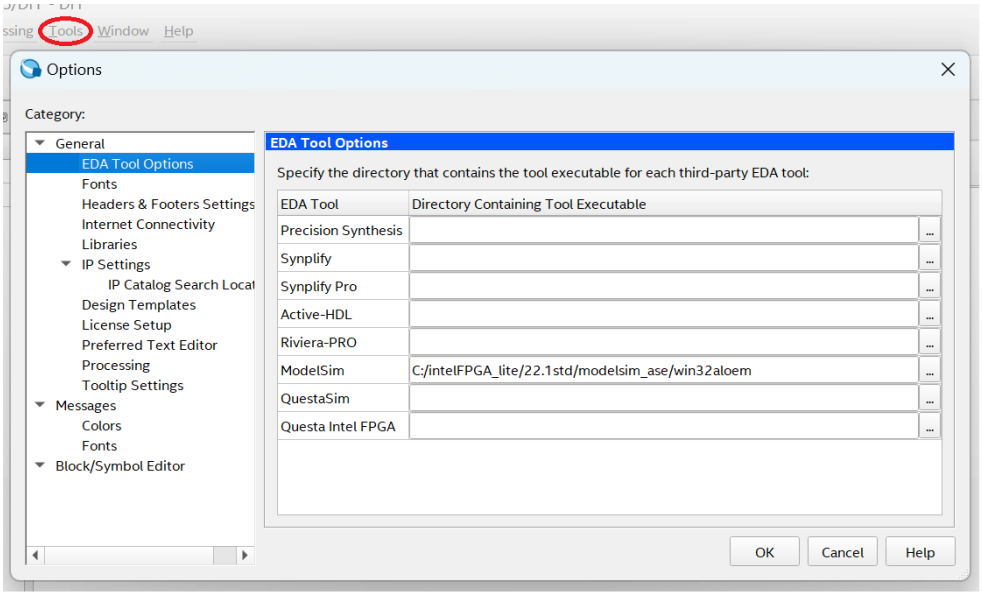


7. Symulacja układu – wstępna weryfikacja poprawności pracy projektowanego układu.

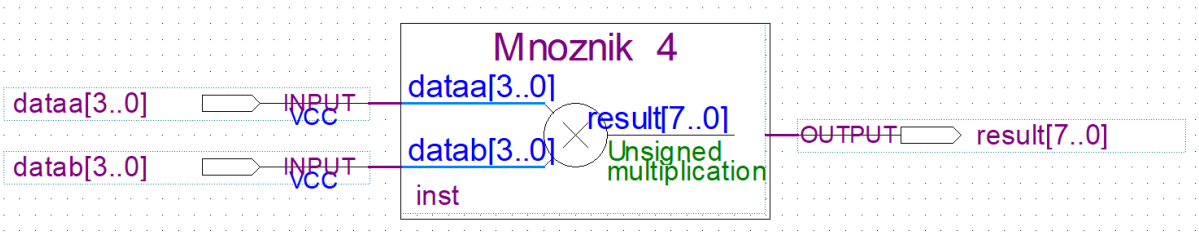
Ważne: żeby uniknąć błędów symulacji **zaleca się** otwieranie projektów w oddzielnych katalogach - co oznacza, że w tym folderze nie powinny znajdować pliki innego projektu.
Drugim wymaganiem jest zainstalowanie wersji symulatora ModelSim-Altera ver. 18.1.0:
- co w wersji Quartus 18 wykonuje się automatycznie i jest widoczne w ustawieniach:
Tools → Options → EDA Tool Options → ModelSim-Altera = ścieżka systemowa (C:\intel …)



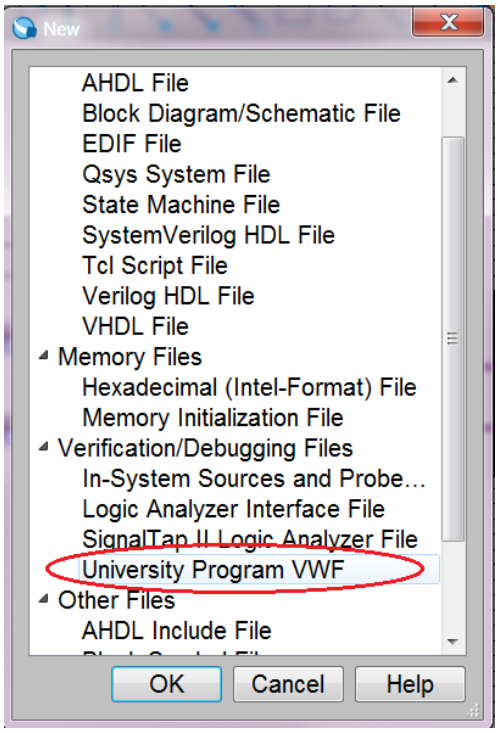
- w wersji Quartus 22 po manualnej instalacji musi być dostępny ModelSim-Altera ver. 18:
Tools → Options → EDA Tool Options → ModelSim = ścieżka systemowa (C:\intelFPGA …)



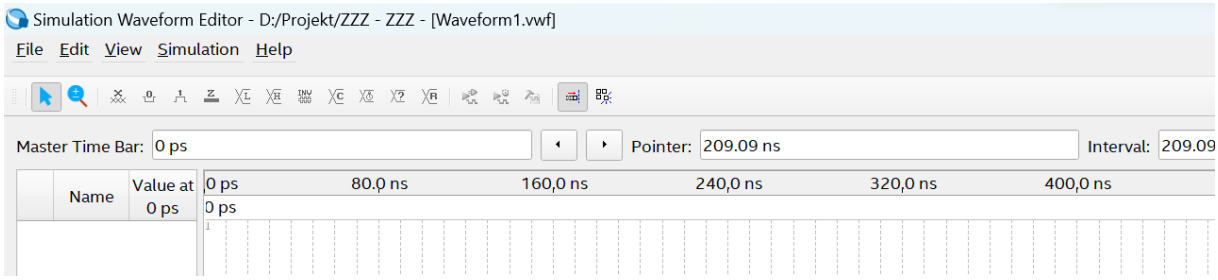
Po zakończonej sukcesem kompilacji, możliwa jest symulacyjna weryfikacja pracy układu. Jest to analityczne sprawdzenie na poziomie sygnałów czy zaprojektowany układ spełnia założenia. Proces symulacji prześledzimy na przykładzie układu z zadania 2, czyli mnożnika.



Przed uruchomieniem symulacji, należy przygotować wektor wymuszeń, czyli zestaw sygnałów wejściowych (w lit. test bench), dla których wyznaczona będzie odpowiedź układu, czyli zostaną wyznaczone wartości sygnałów wyjściowych. W tym celu w aktywnym projekcie utworzymy nowy plik „University Program VWF”:

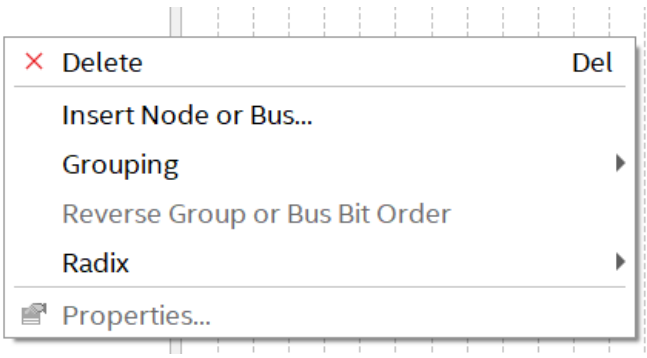


Po otwarciu nowego pliku pojawi się okno edycyjne symulatora, którego fragment pokazuje poniższy rysunek. Symulacje w Quartus II wykonywane są w funkcji czasu, którego zakres i interwał siatki można zmieniać z użyciem górnej funkcji Edit.

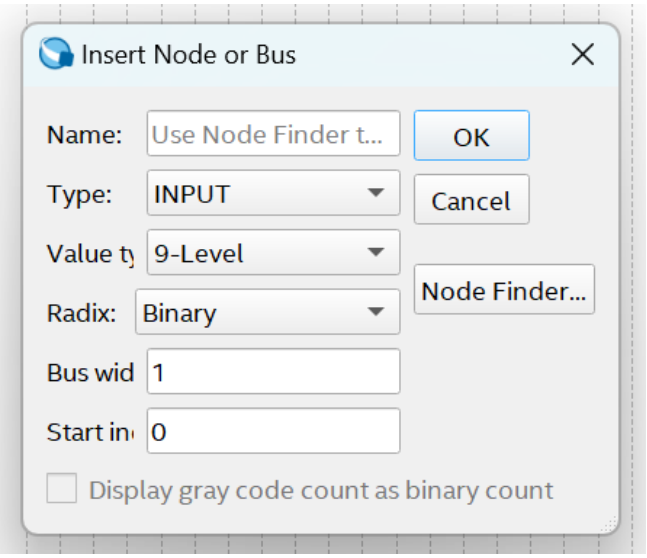


Po ustawieniu lub zaakceptowaniu domyślnego czasu symulacji, należy wczytać zestaw portów wejściowych i wyjściowych badanego projektu. W tym celu należy wykonać następujące czynności:

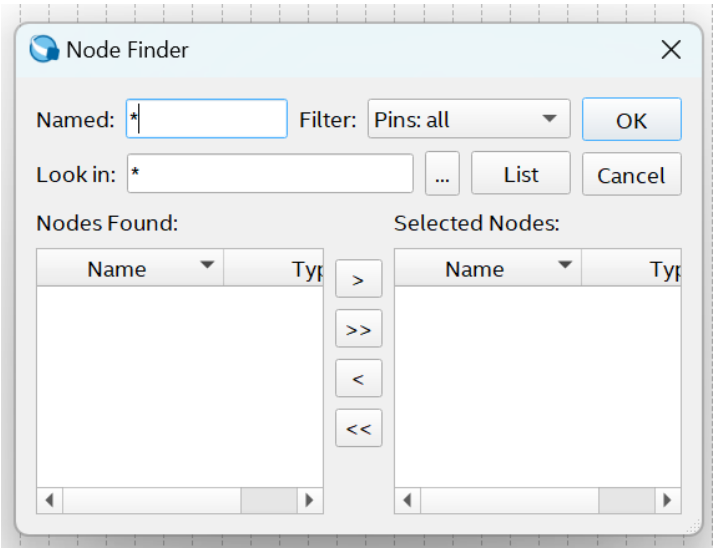
- w polu **Name** należy kliknąć prawy klawisz myszki – otworzy się menu,



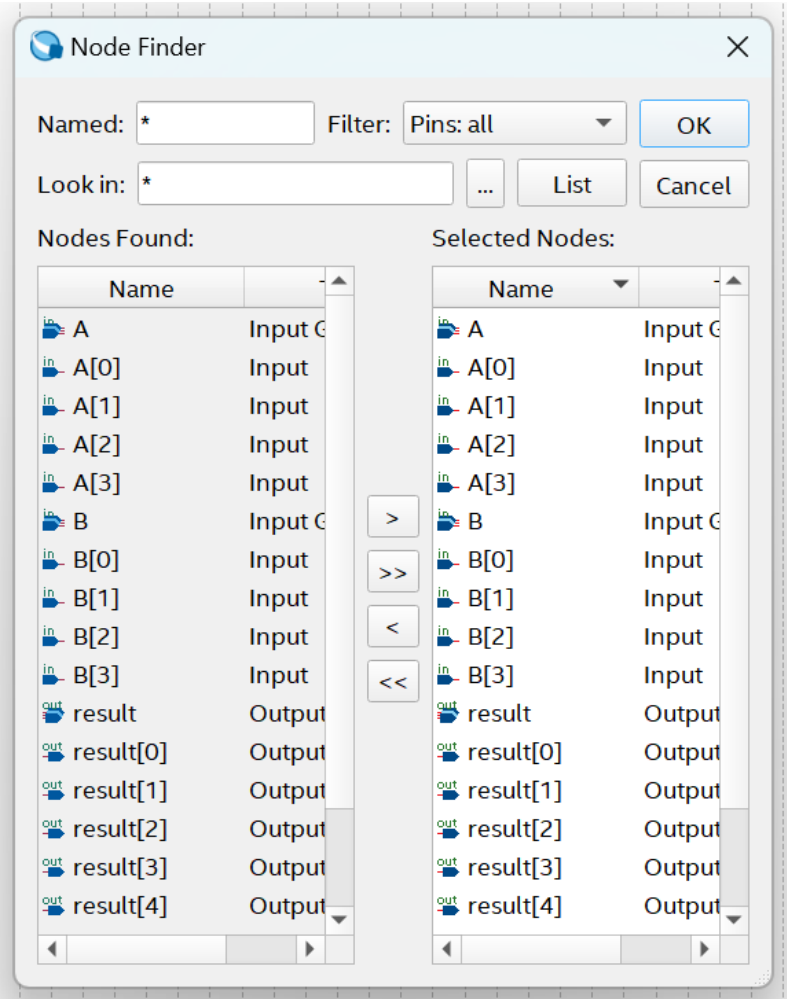
- w którym, należy wybrać **Insert Node Or Bus** – a w kolejnym **Node Finder**,



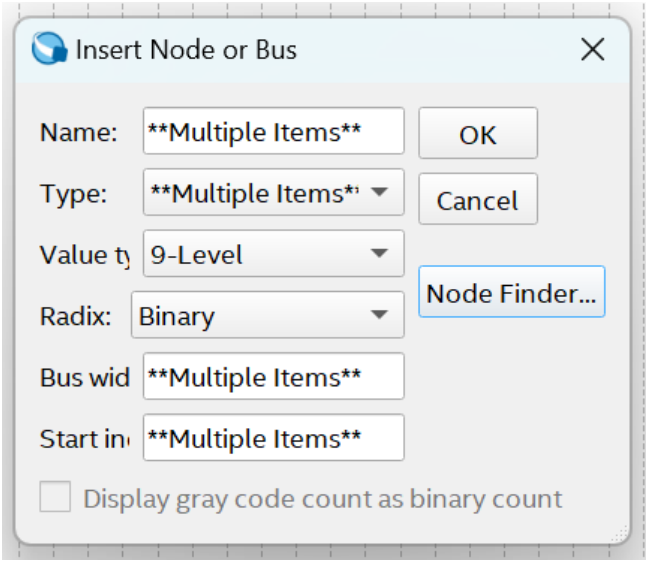
- w kolejnym wybieramy **List**,



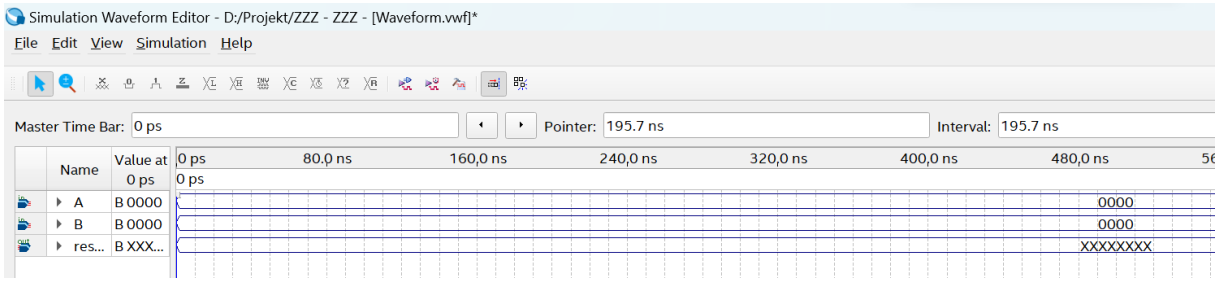
- a następnie przyciskiem >> wybieramy/wskazujemy wszystkie porty w projekcie i akceptujemy **OK**,



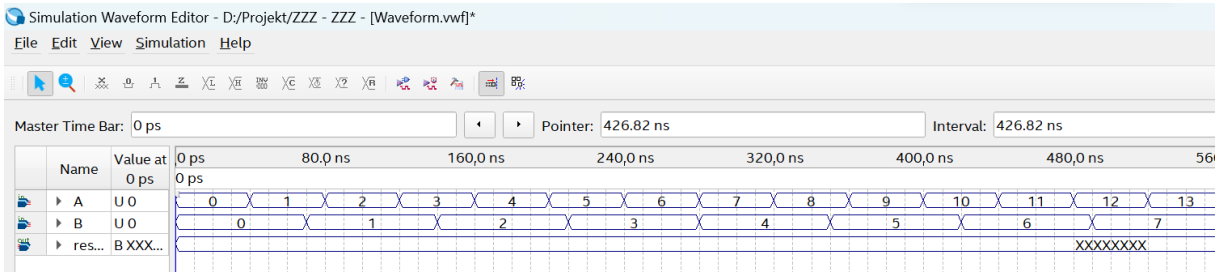
- ustawienia kolejnego okna zatwierdzmy **OK**,



Po wykonaniu ustawień w oknie symulatora pojawią się puste porty wejściowe i wyjściowe:

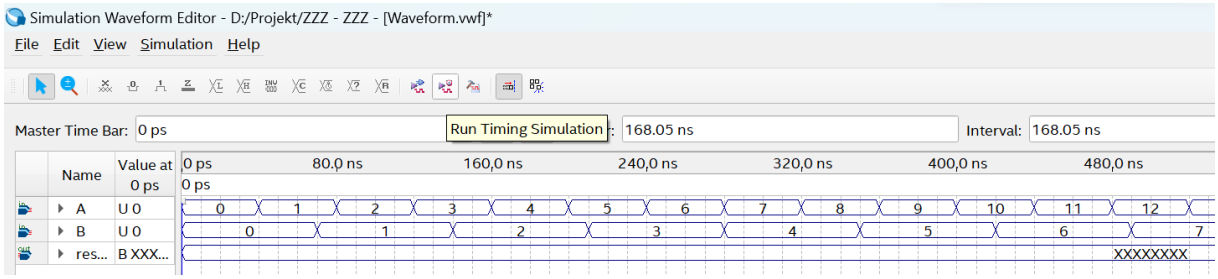


Teraz, używając funkcji menu graficznego należy ustawić wartości portów wejściowych A i B - **szczegóły związane z obsługą funkcji górnego menu, będą podane w trakcie zajęć**:

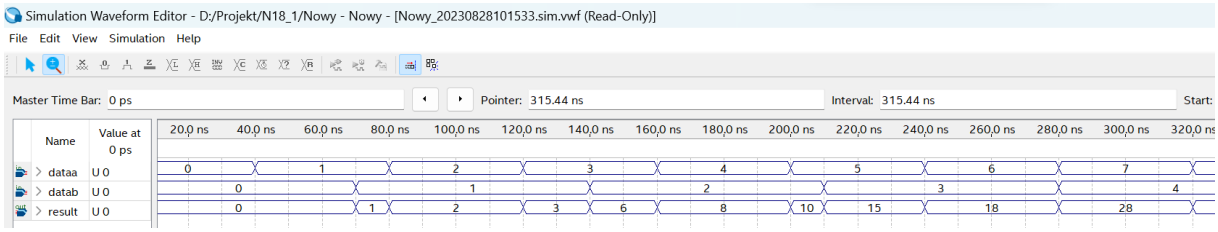


Po ustawieniu sygnałów wejściowych – należy plik zapisać, nie zmieniając nazwy domyślnej. **Ważne** – w celu przeprowadzenia skutecznej symulacji, zaleca się tworzenie projektów badanych układów w oddzielnych folderach dyskowych.

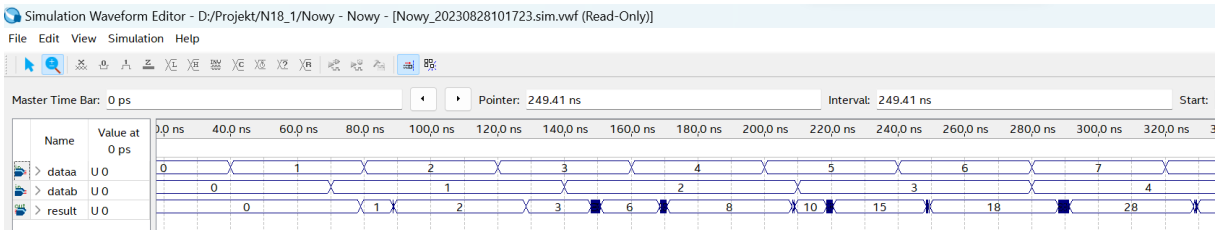
W tym momencie można uruchomić proces symulacji funkcjonalnej lub czasowej – klikając odpowiednią ikonę górnego menu:



Po zakończeniu symulacji np. funkcjonalnej, jej wyniki pojawią się w nowym oknie:



lub, podobnie wyniki symulacji czasowej:



8. Przygotowanie do programowania – definiowanie wyprowadzeń FPGA.

W tym etapie określa się, jakie urządzenia wejściowe płyty prototypowej DE2-115 będą źródłem sygnałów wejściowych oraz, które z nich będą wyprowadzały/wyświetlały dane wyjściowe z projektowanego układu:

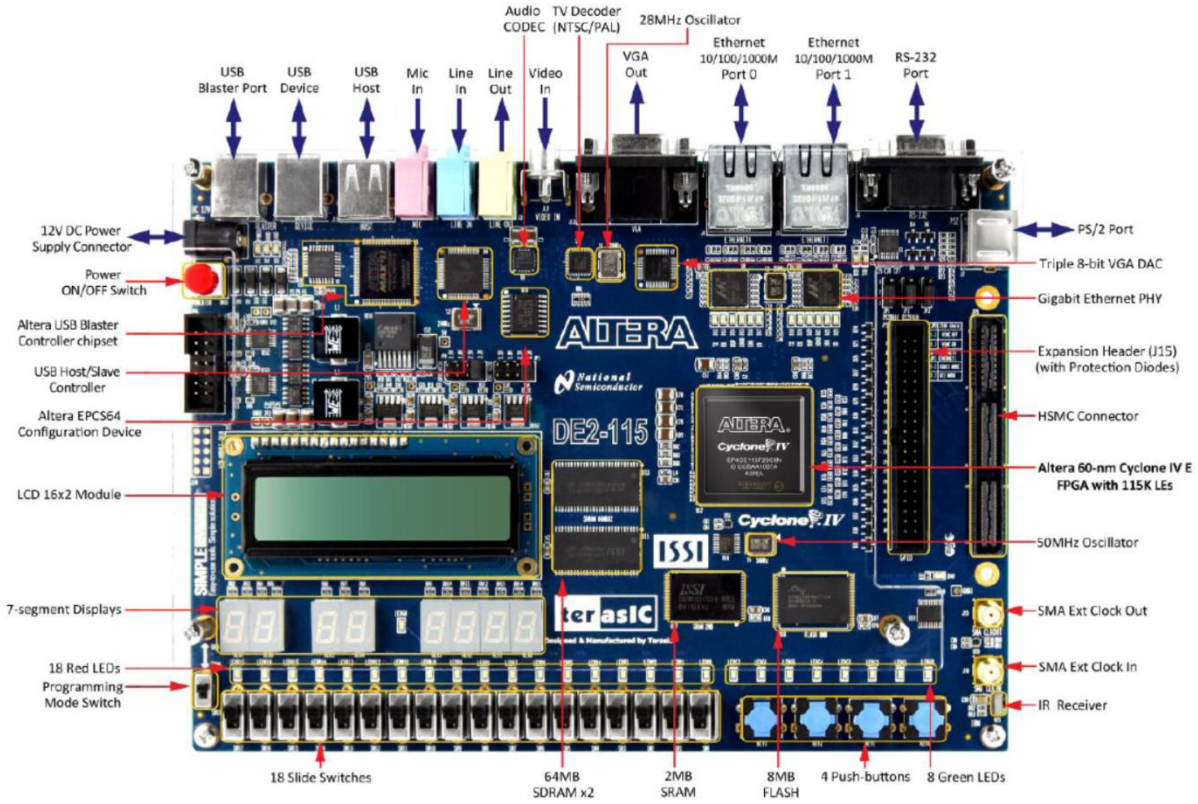


Figure 2-1 The DE2-115 board (top view)

W początkowej fazie laboratorium, najczęstszym źródłem sygnałów wejściowych będą przełączniki (Switches) oraz klucze monostabilne (Push-buttons), zaś wyniki działania układu będą obserwowane na LEDach (Red LEDs i Green LEDs) lub wyświetlaczach 7-segmentowych (7-segment Displays). Położenie:

- przełącznika w dolnej pozycji wytwarza niski poziom (0 logiczne) na jego wyjściu,
- przełącznika w górnej pozycji wytwarza poziom wysoki (1 logiczną),
- wciśnięcie klucza również wytwarza 0 logiczne,
- zwolnienie klucza oznacza stabilny stan wysoki na jego wyjściu (1 logiczną).

Świecenie diody LED oznacza wysoki stan podany na jej wejście, brak świecenia oznacza polaryzację stanem niskim.

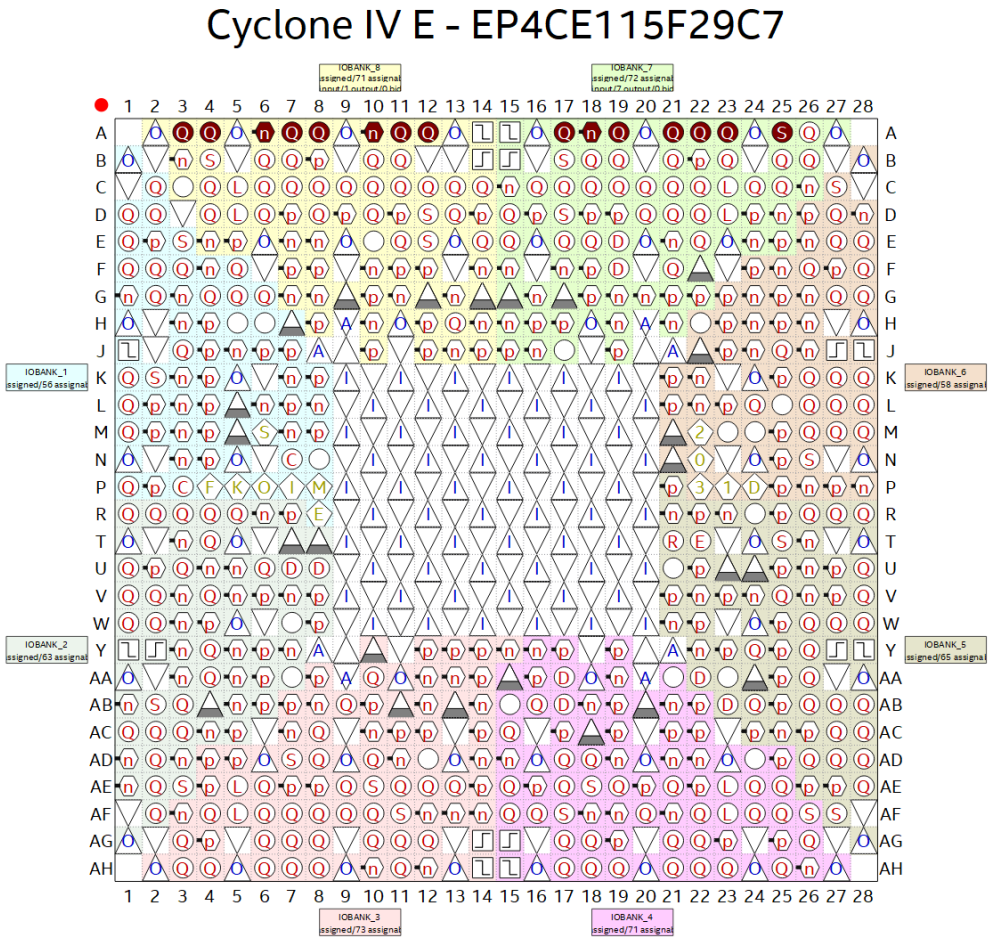
Poszczególne diody wyświetlacza 7-segmentowego, świecą, gdy na ich wejściu występuje stan niski - są to wyświetlacze typu wspólna anoda.

Znaczenie i obsługa innych urządzeń modułu DE2-115 będzie wyjaśniana w zależności od potrzeb kolejnych ćwiczeń laboratoryjnych.

W czasie zajęć laboratoryjnych wszystkie projekty będą uruchamiane z wykorzystaniem modułu FPGA firmy Intel, z rodziny Cyclone IVE – typ: EP4CE115F29C7/…C8L.

W związku z tym, że system Quartus II jest uniwersalnym opracowaniem dla wielu rodzin układów FPGA firmy Intel, projektant musi poinformować system z konkretnie, jakim typem

układu FPGA pracuje oraz do których wyprowadzeń tego układu będą podłączone sygnały wejściowe oraz z których będą wyprowadzane sygnały wyjściowe.
Na poniższym rysunku pokazano topologię wyprowadzeń stosowanego w czasie zajęć układu FPGA EP4CE115F29C7. Odwołując się do konkretnego wyprowadzenia należy podać jego



współrzędne/piny. W przypadku układów scalonych w obudowach FBGA (czyli badanego), nazwę pinu określa nazwa wiersza (nazwy od A do AH) połączona z nr kolumny (od 1 do 28). Np. pin położony w pierwszym wierszu na trzeciej pozycji posiada nazwę A3, zaś pin w ostatnim wierszu na trzeciej pozycji od końca jest oznaczony AH26.
Przypisanie połączeń (wejść/wyjść) urządzeń w module DE2-115 do układu FPGA, jest zawarte w tabelach pliku **DE2_115_User_manual.pdf** od str. 36. Wspomniany plik jest dostępny na pulpitach komputerów w laboratorium. Np. dla przełączników mamy nr pinów:

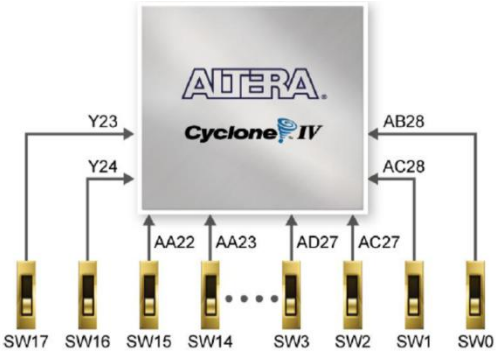
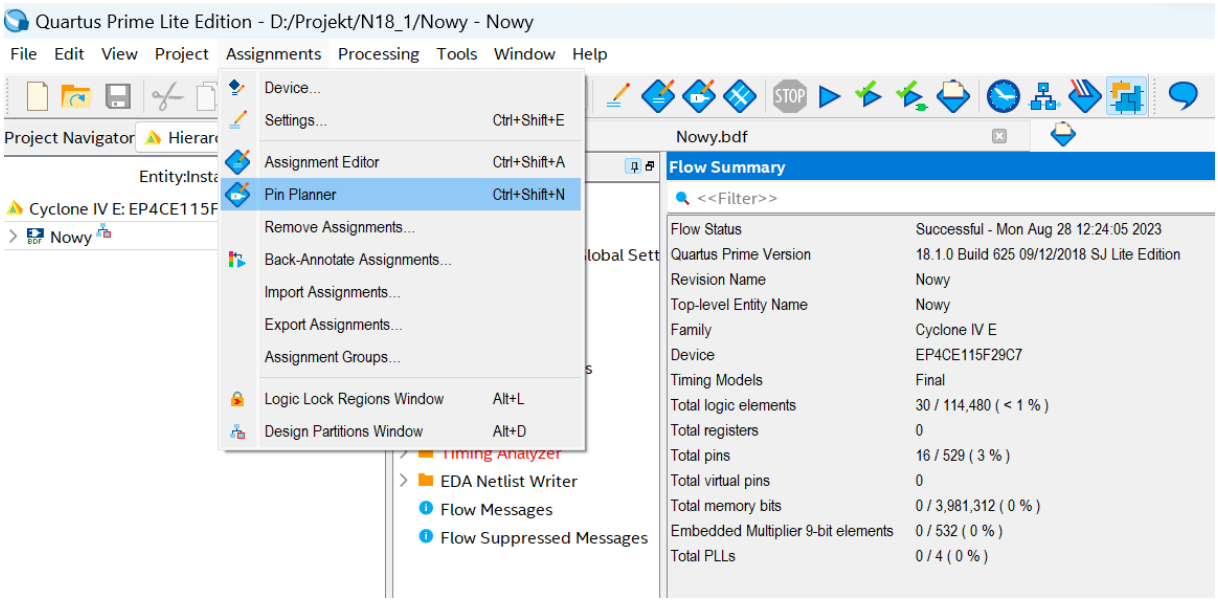


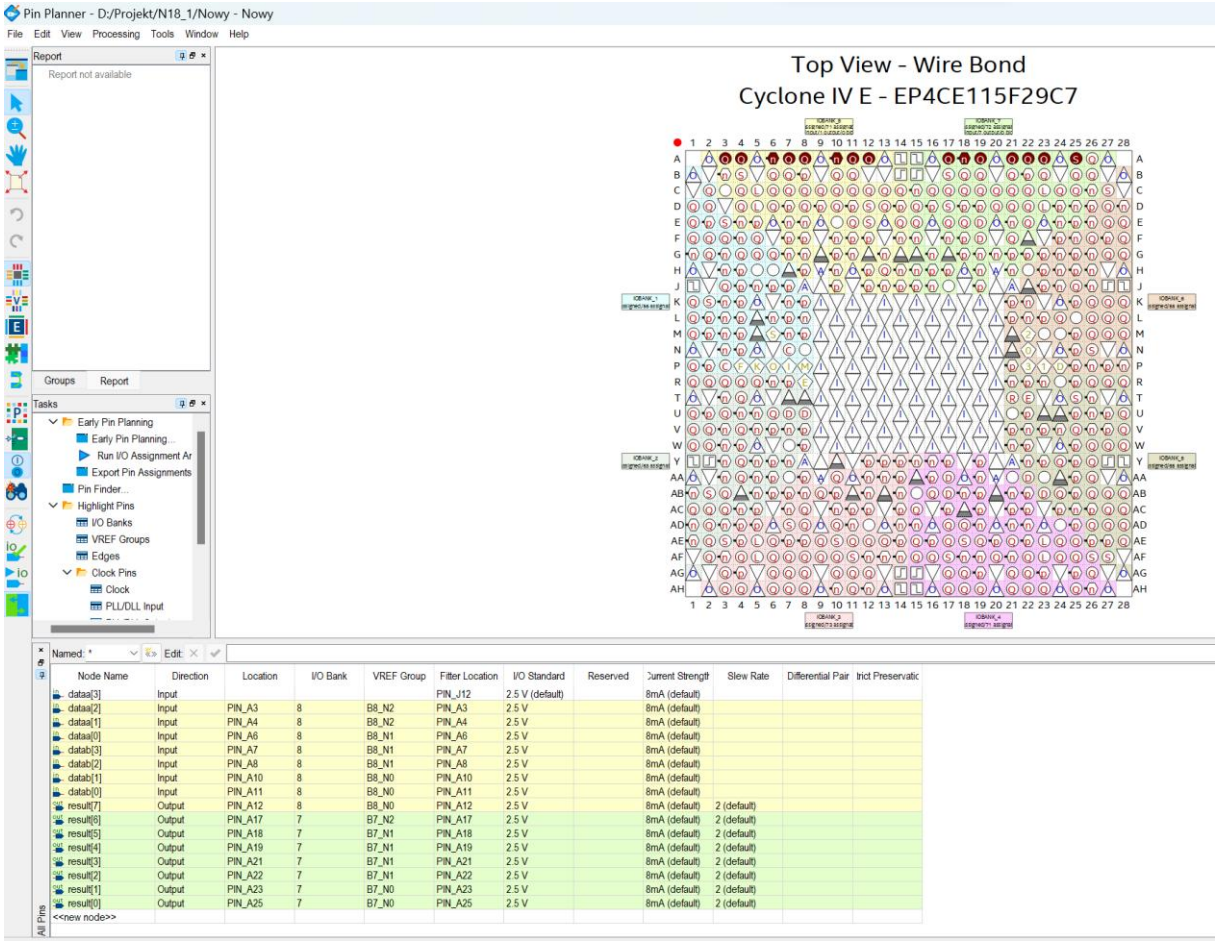
Table 4-1 Switches Pin Assignments

Signal Name	*FPGA Pin No.*
SW[0]	PIN_AB28
SW[1]	PIN_AC28
SW[2]	PIN_AC27
SW[3]	PIN_AD27
SW[4]	PIN_AB27
SW[5]	PIN_AC26
SW[6]	PIN_AD26
SW[7]	PIN_AB26
SW[8]	PIN_AC25

W celu określenia połączeń urządzeń z chipem FPGA – uruchamiamy moduł **Pin Planner**:



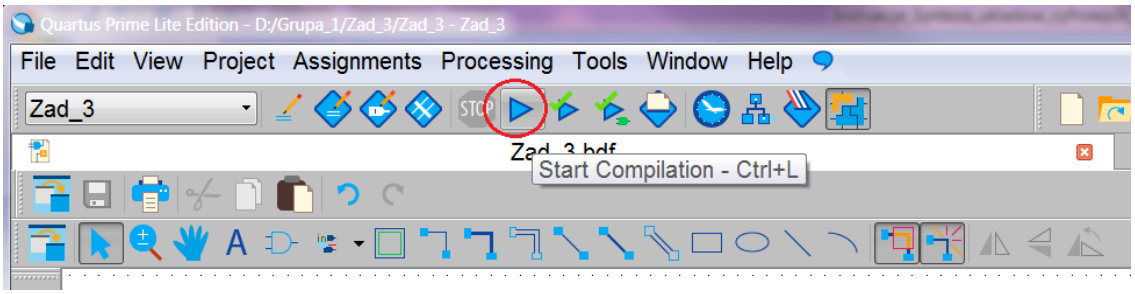
w jego centralnej części widoczne są wyprowadzenia układu FPGA, a w dolnej tabela z wykazem wszystkich portów w projekcie. W tej tabeli wypełniamy odpowiednimi nazwami wyprowadzeń/pinów FPGA kolumnę **Location**.



Po wypełnieniu całej tabeli nie zapisujemy jej, lecz wracamy do programu głównego – w tym momencie projekt jest przygotowany do następnego etapu.

9. Kompilacja końcowa – trasowanie wewnętrznych połączeń FPGA

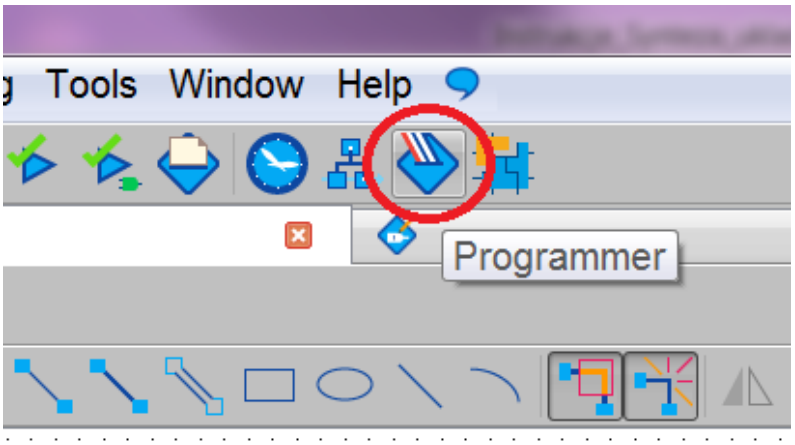
Po zakończeniu obsługi Pin Planner – przeprowadzamy ponowną kompilację projektu.



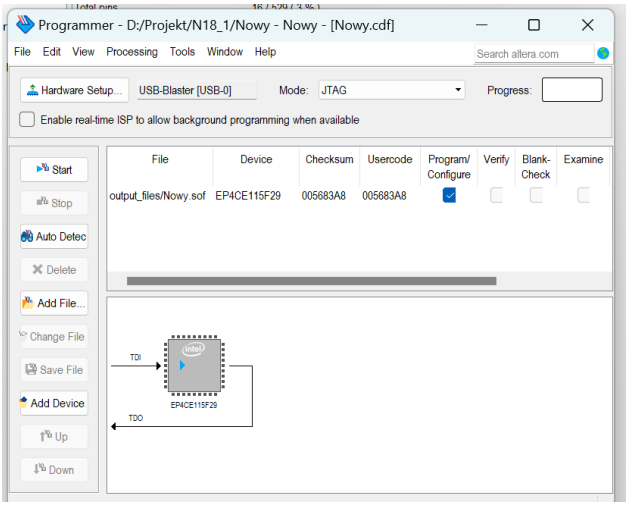
Jej celem jest wygenerowanie wewnętrznych połączeń pomiędzy wewnętrznymi elementami FPGA – wynikiem jest plik .sof programujący chip FPGA - czyli **bit stream**.
Na tym etapie, sporadycznie, mogą pojawić się błędy kompilacji, związane z:
- niewłaściwym przypisaniem pinów,
- przekroczeniem przez projektowany układ wewnętrznych zasobów FPGA,
- niewłaściwym ustawieniem funkcji pinów podwójnego znaczenia.

10. Programowanie i testowanie układu

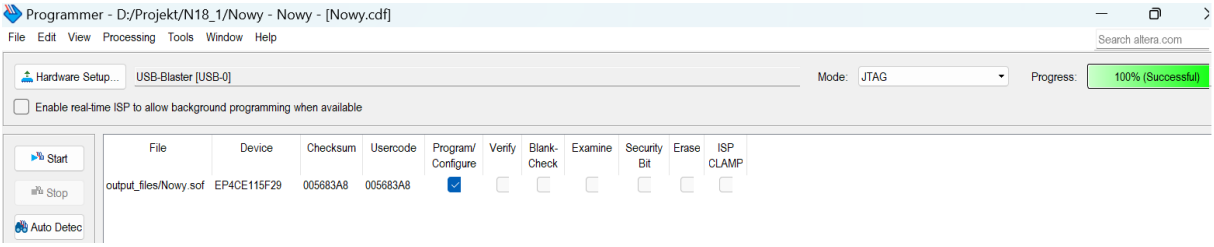
Po zakończonej sukcesem kompilacji końcowej, można zaprogramować moduł prototypowy DE2-115. Funkcję programatora wywołujemy z górnego menu graficznego:



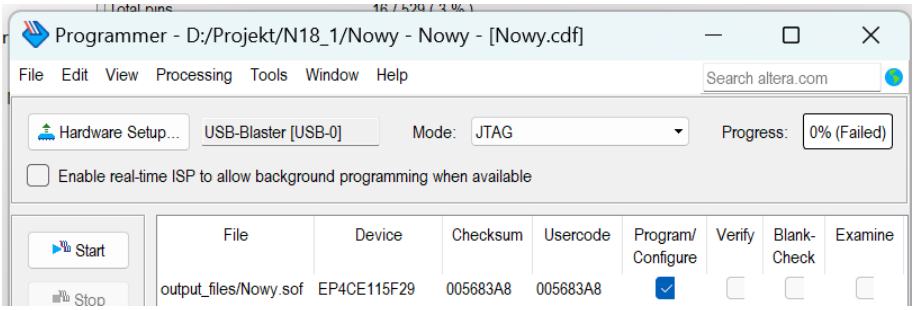
Po jej uruchomieniu pojawi się menu programatora – w którym, należy wcisnąć **Start.**



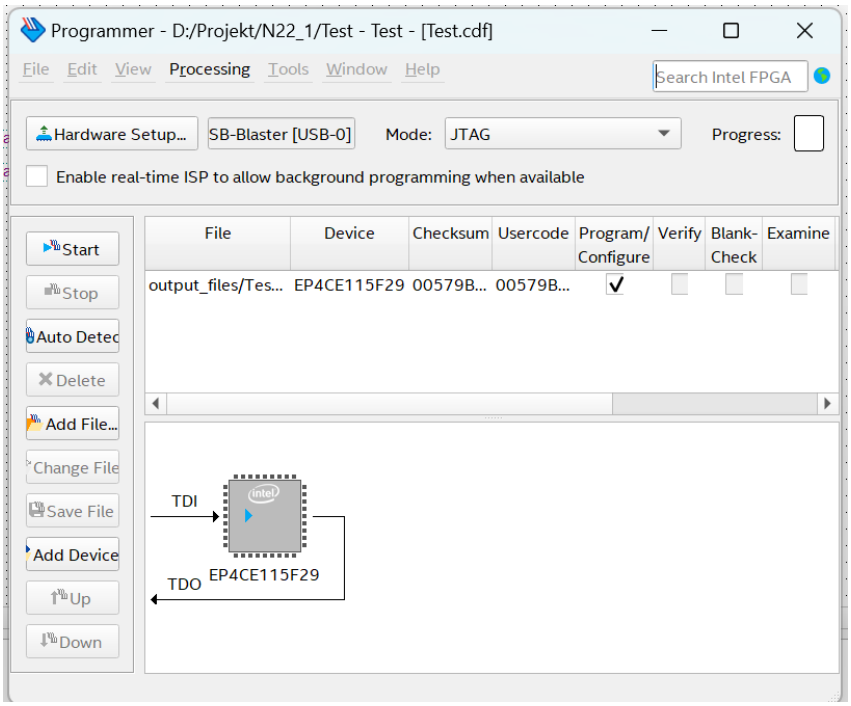
W tym czasie należy obserwować stan paska postępu programowania – Progress:



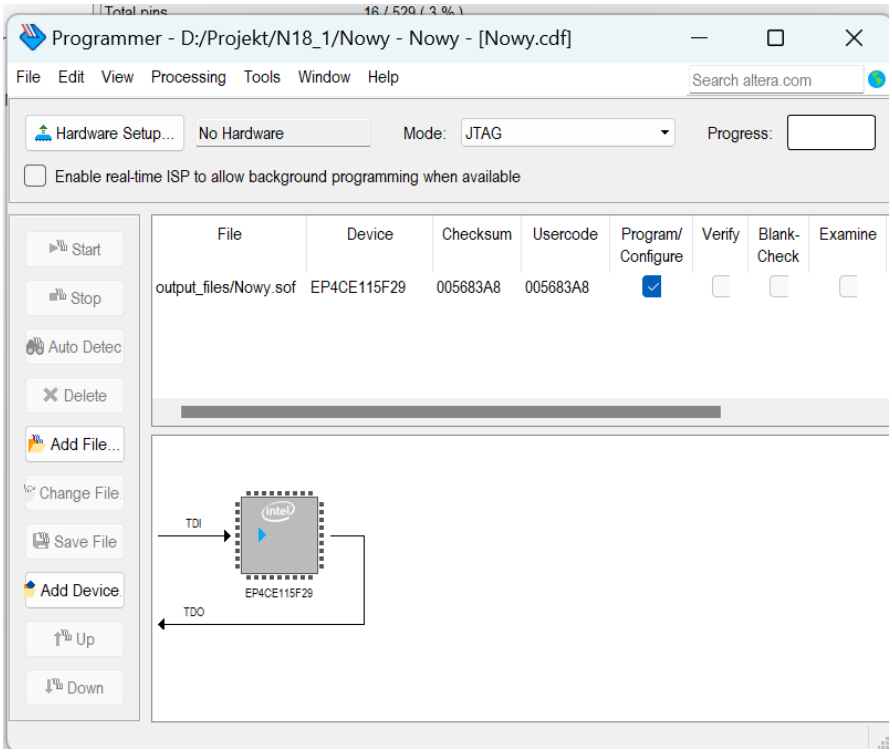
Jeżeli zielone pole w prawym górnym rogu osiągnie 100%, oznacza to poprawne zaprogramowanie FPGA i można przejść do testowania opracowanego układu, zmieniając nastawy kluczy, przełączników oraz obserwując stany LEDów/wyświetlaczy.
Sporadycznie może się zdarzyć, że w trakcie przesyłania bit stream do FPGA wystąpi błąd transmisyjny a w pasku postępu pojawi się komunikat Failed – zazwyczaj oznacza to błąd sumy kontrolnej transmisji bit stream i programowanie należy powtórzyć.



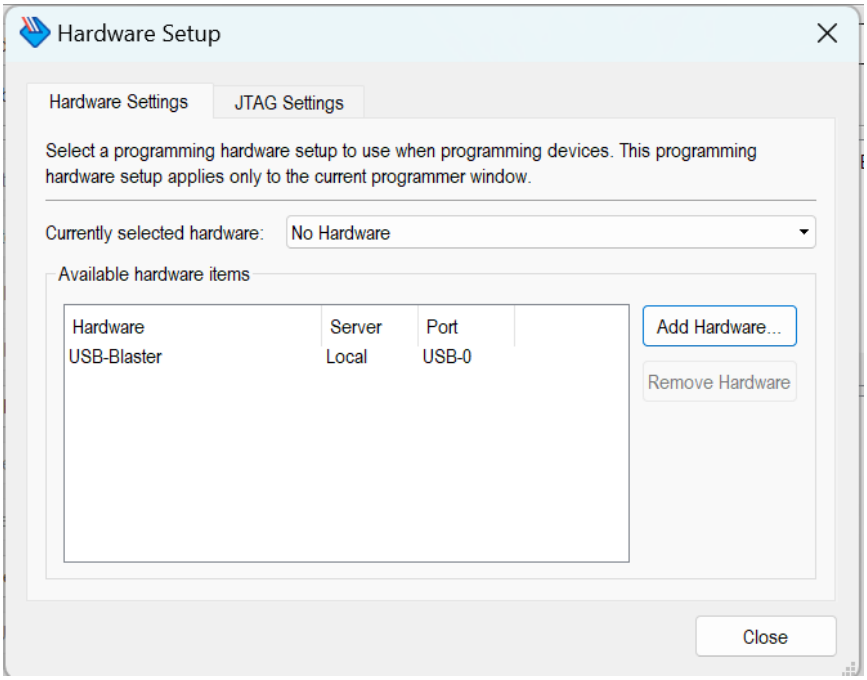
Jeżeli kolejna próba programowania nie zakończy się sukcesem – problem należy zgłosić prowadzącemu zajęcia, gdyż może to oznaczać awarię sprzętu lub błędy ustawienia projektu.
Domyślne ustawienia programatora można zmieniać po jego uruchomieniu. W szczególności można wymienić domyślny plik programujący. W tym celu należy skasować widoczny w centralnej części chip i z użyciem opcji **Add File** wskazać nowy plik programujący. Zazwyczaj pliki programujące znajdują się w podkatalogu **output_files** projektu.



Częstym przypadkiem/błędem jest zbyt późne włączenie zasilania modułu DE2-115, wówczas po uruchomieniu programatora – jest nieaktywny przycisk **Start**. W takiej sytuacji należy odświeżyć sterownik systemowy JTAG.



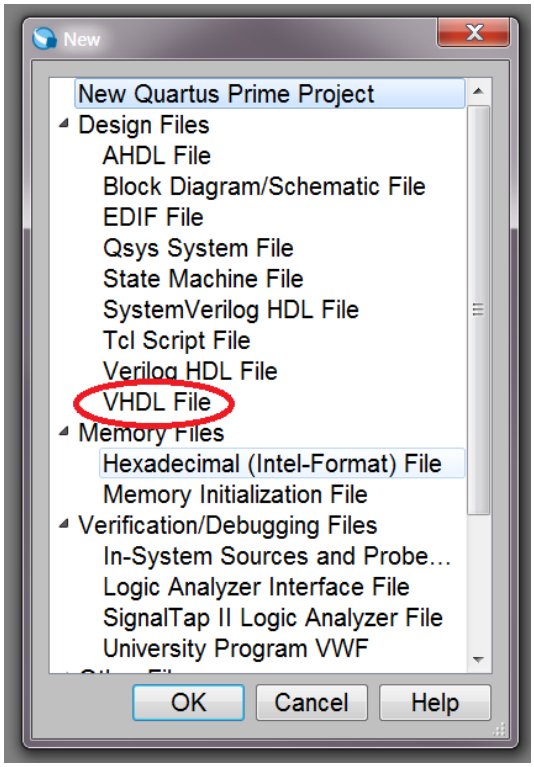
W tym celu należy kliknąć button **Hardware Setup** w lewym górnym rogu. Po uruchomieniu tej funkcji w jej menu należy dwukrotnie kliknąć na USB-Blaster i zamknąć okno. W efekcie przycisk Start powinien znaleźć się w stanie aktywnym. Jeżeli w poniższym menu, nie jest widoczna opcja USB-Blaster, oznacza to błąd połączenia z komputerem lub błąd zasilania.



11. Część końcowa - definiowanie układu z użyciem języka VHDL

Alternatywną, w stosunku do schematów, formą opisów opisu projektów w Quartus II jest użycie języka opisu sprzętowego VHDL. Szczegóły dotyczące składni tego języka, instrukcji, typów danych i metod przetwarzania zostaną podane na wykładzie i przypomniane w laboratorium. W tej części przećwiczymy jedynie sposób włączenia do systemu kodu VHDL.

Po utworzeniu nowego projektu jako plik top-level należy wybrać nowy plik **VHDL File**.



Po jego otwarciu pojawi się typowe okno edytora tekstowego ASCII. Poniżej widać treść kodu w VHDL, który opisuje funkcjonowanie mnożnika 4-bitowego – czyli układu równoważnego badanemu w zadaniu 2.

Test.vhd | Compilation Report - Test

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Test is
6      port
7      ( a, b : in  unsigned  (3 downto 0);
8      result : out unsigned (7 downto 0)  );
9  end entity;
10
11 architecture rtl of Test is
12 begin
13
14     result <= a * b;
15
16 end rtl;
17
```

Zadanie laboratoryjne 1.3

Wykorzystując nabyte umiejętności proszę:

- utworzyć nowy projekt w nowym folderze na dysku D,
- nazwę pliku top-level zdefiniować Test,
- otworzyć nowy plik VHDL,
- wpisać do niego kod z poprzedniej strony,
- plik należy zapisać pod nazwą Test.vhdl nie zmieniając folderu,
- dokonać kompilacji wstępnej,
- przeprowadzić symulację pracy układu,
- uzupełnić Pin Planner,
- przeprowadzić kompilację końcową,
- zaprogramować i przetestować układ jak poprzednio.

Literatura:

1. Intel FPGA: *Intel FPGA Software Installation and Licensing*, 11.01.2023, https://www.intel.com/content/www/us/en/docs/programmable/683472/22-4/faq.html
2. *Intel® Quartus® Prime Pro and Standard Software User Guides*, 11.01.2023, https://www.intel.com/content/www/us/en/support/programmable/support-resources/design-software/user-guides.html

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 2

Opis układu cyfrowego w FPGA z użyciem układów logicznych i makromodeli IP

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcję opracował:

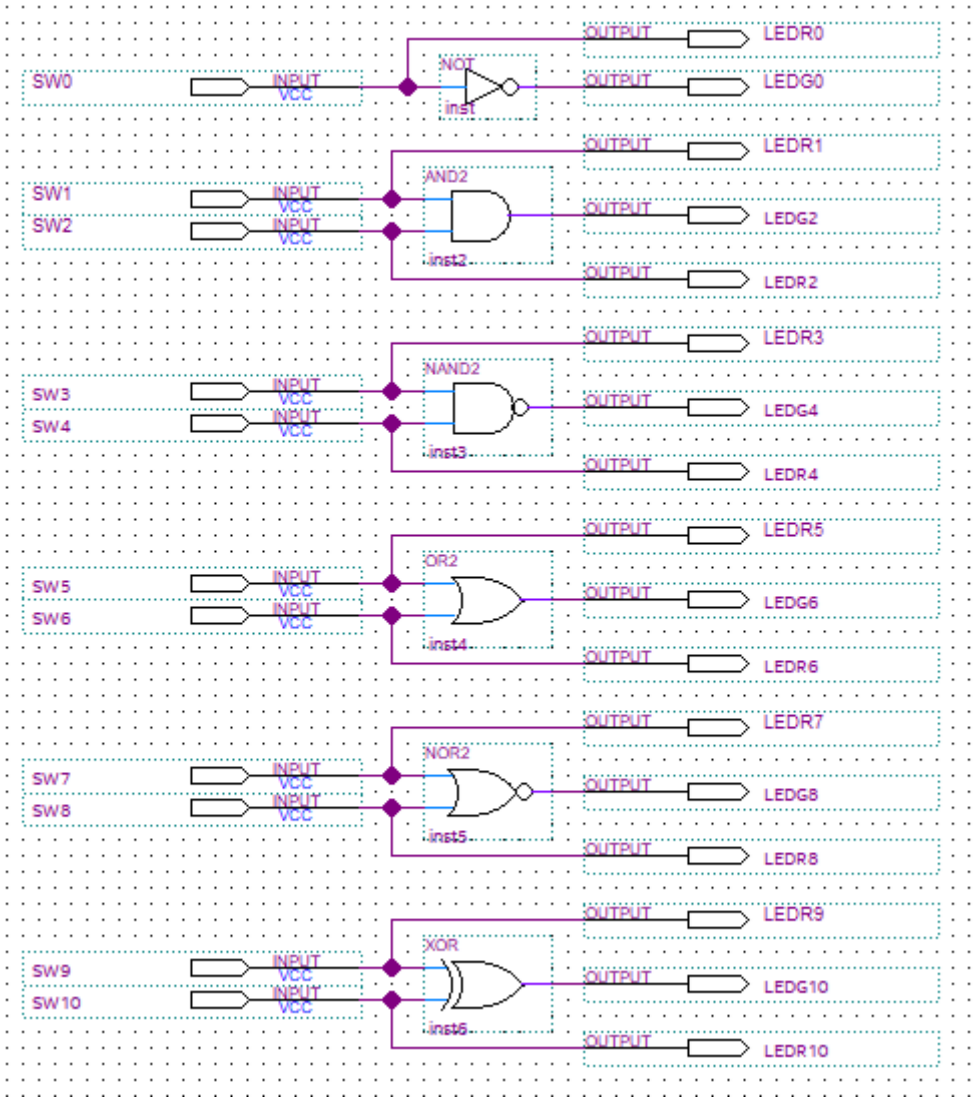
dr inż. Marian Gilewski

Białystok 2025

Cel ćwiczenia.

Celem ćwiczenia jest poznanie cech oraz nabycie umiejętności w zakresie stosowania i obsługi wybranych bramek logicznych, multiplekserów, demultiplekserów, układów cyfrowych. Doskonalone będzie synteza układów kombinacyjnych z użyciem: bramek, modułów IP oraz kodu VHDL.

Zadanie laboratoryjne 2.1 dotyczy badania bramek logicznych różnych typów, jednocześnie pokazuje ono sposób rejestracji wyników pomiarowych w trakcie laboratorium.
W tym celu proszę utworzyć nowy projekt i otworzyć plik top-level typu graficznego (Block Diagram/Schematic) i narysować schemat:



Następnie należy dokonać kompilacji, a w module Pin Planner porty wejściowe połączyć w sposób podany na schemacie (plik DE2_115_User_manual.pdf, str. 37).
Porty wyjściowe proszę połączyć z LEDami LEDG/LEDR (DE2_115_User_manual.pdf, str. 37/38).
Następnie należy dokonać kompilacji końcowej i zaprogramować oraz przetestować układ.

Wyniki badań (stany wyjść) proszę wpisać do poniższej tabeli a tabelę umieścić w sprawozdaniu wraz z interpretacją uzyskanych wyników.

Wejścia		Wyjścia	
SW0	0	LEDG0	
	1		
SW1, SW2	0, 0	LEDG2	
	0, 1		
	1, 0		
	1, 1		
SW3, SW4	0, 0	LEDG4	
	0, 1		
	1, 0		
	1, 1		
SW5, SW6	0, 0	LEDG6	
	0, 1		
	1, 0		
	1, 1		
SW7, SW8	0, 0	LEDG8	
	0, 1		
	1, 0		
	1, 1		
SW9, SW10	0, 0	LEDG10	
	0, 1		
	1, 0		
	1, 1		

Zadanie laboratoryjne 2.2 pokazuje mechanizm syntezy układów cyfrowych z bramek logicznych. Opis działania:
mamy system automatycznej klasyfikacji jakości wyrobów (piwa). Na stanowisku diagnostycznym mierzone są 3 parametry jakości –nazwijmy je:
A, B, C (% alkoholu w piwie, klarowność trunku, objętość) Jeżeli dany parametr spełnia wymagania jakościowe to przyjmuje wartość ‘1’ –w przeciwnym przypadku wartość ‘0’. Wyrób jest zaliczany jako:
- bardzo dobry -jeżeli spełnia wszystkie parametry,
- dobry - jeżeli spełnia 2 parametry,
- przeciętny - gdy spełnia 1 parametr,
- jest wadliwy - gdy nie spełnia żadnego parametru.

Układ cyfrowy nie rozumie języka naturalnego: w tym słów: bardzo dobry, dobry, przeciętny lub wadliwy. Stąd trzeba te terminy zakodować najmniejszą liczbą bitów np.:
bardzo dobry = „11”, dobry = „10”, przeciętny „01” i wadliwy = „00”. Zatem nasz system posiada trzy wejścia: A, B i C oraz dwubitowe wyjście Y1 i Y2 –określające jakość.

Tablica prawdy naszego układu jest następująca:

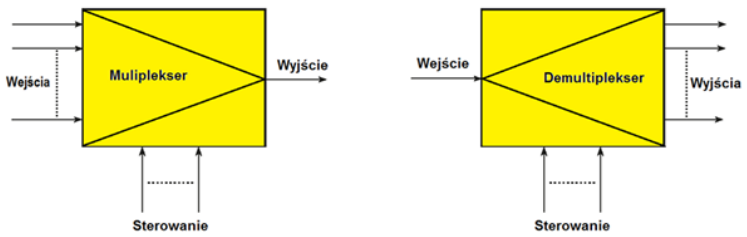
Wejścia			Wyjścia	
A	B	C	Y1	Y0
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Utwórzmy kanoniczną **funkcję sumacyjną** np. dla wyjścia Y1, układu opisanego powyższą tabelą prawdy:
- uwzględniamy tylko te przypadki - gdy funkcja na wyjściu Y1 przyjmuje wartość **1**,
- dla uwzględnianych przypadków tworzymy iloczyn wejść,
- jeżeli argument (A, B, C) przyjmuje wartość 0 - w funkcji wynikowej zapisujemy go w postaci zanegowanej (np. not A),
- jeżeli dla rozpatrywanego przypadku, argument funkcji przyjmuje wartość 1 –zapisujemy go w postaci prostej (np. B),
- tworzymy iloczyny składowe uwzględniając wszystkie argumenty (A, B, C)
- postać końcową funkcji przedstawiamy jako sumę iloczynów składowych:

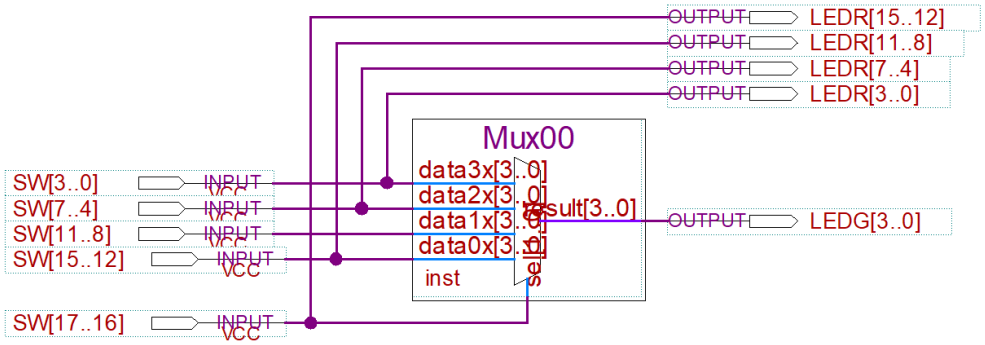
$$Y1 = F(A,B,C) = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$$

Proszę zrealizować powyższą funkcję z bramek dostępnych w systemie, układ skompilować a następnie zaprogramować i przetestować.

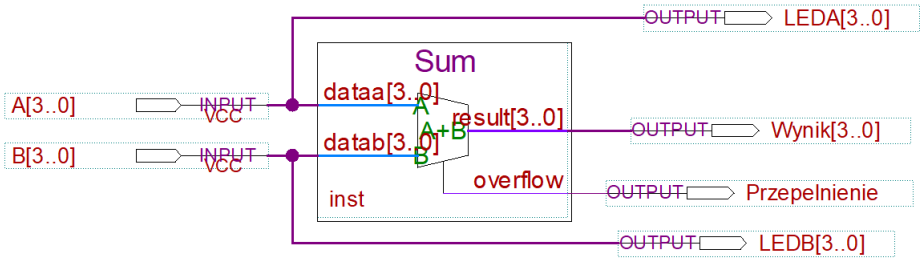
Zadanie laboratoryjne 2.3 obrazuje sposób implementacji multipleksera z wykorzystaniem elementu bibliotecznego IP.
Układy komutacyjne stanowią specyficzną kategorię układów kombinacyjnych, których funkcją, jak sama nazwa mówi, jest przełączanie sygnałów z wielu wejść na jedno wyjście lub rozdzielanie sygnałów z jednego wejścia na wiele wyjść. Historia ich zastosowań sięga początków telekomutacji, gdy za pośrednictwem jednego kanału transmisyjnego (linii przewodowej, łącza radiowego) trzeba było przesyłać dane wielu abonentów, stosując transmisję z podziałem czasu. W tych układach czynność przełączania wejść/wyjść kontrolują sygnały sterujące, ustawiając topologię połączenia w zależności od wartości tych sygnałów.



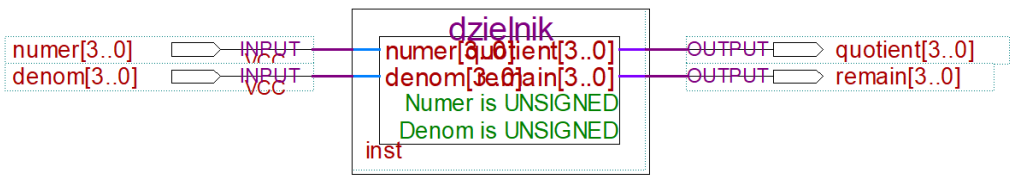
Jeżeli sygnały z wielu wejść są przełączane na jedno wyjście mamy do czynienia z multiplekserem, jeżeli sygnał z jednego wejścia jest rozdzielany na wiele wyjść - układ jest demultiplekserem. Sygnał wejściowy jak i wyjściowy mogą stanowić pojedyńczą linię lub można przełączać magistrale złożone z wielu linii – stąd rozróżnia się multipleksery/demultipleksery sygnałowe lub magistralowe/szynowe. Liczba bitów (n) sygnałów sterujących przełączaniem wynika z zasady 2^n - tzn. układ o n wejściach sterujących może obsłużyć 2^n wejść/wyjść danych.
Współczesne, rzadko się stosuje multipleksery/demultipleksery, zbudowane jako dyskretne układy scalone. Dostępność innych, efektywniejszych technik, w tym języków opisu sprzętowego jak np. VHDL – pozwala syntezować tego typu układy, o dowolnej złożoności.
W tym ćwiczeniu przebadamy bardzo prostą realizację multipleksera z użyciem IP bibliotecznego elementu **lpm_mux**. Proszę zaimplementować i przetestować układ przedstawiony na poniższym rysunku:



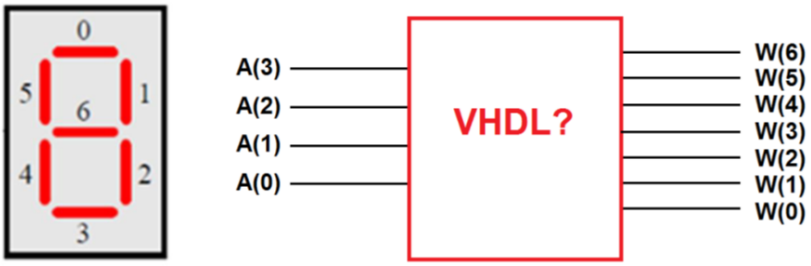
Zadanie laboratoryjne 2.4 pokazuje sposób użycia sumatora parametrycznego IP z zasobów systemu. Wykorzystując makrofunkcję **LPM_ADD_SUB** biblioteki IP w Quartus II proszę zbudować i przetestować 4-bitowy sumator liczb dodatnich pokazany na poniższym rysunku.



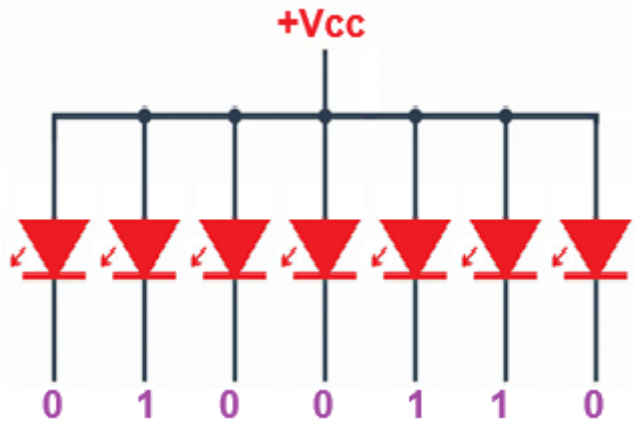
Zadanie laboratoryjne 2.5 obejmuje implementację pokazuje sposób użycia sumatora parametrycznego IP z zasobów systemu. Korzystając z zasobów katalogu IP proszę zdefiniować 4-bitowy dzielnik parametryczny liczb dodatnich makrofunkcji LPM_DIVIDE w przedstawionym układzie.



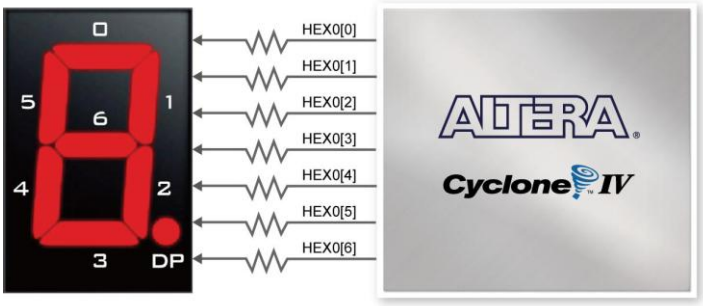
Zadanie laboratoryjne 2.6 obsługa wyświetlacza typu wspólna anoda za pomocą modułu sterującego zdefiniowanego w VHDL.
Wyświetlacz siedmiosegmentowy składa się z 7 LEDów zintegrowanych w jednej strukturze. Są one ponumerowane od 0 do 6. Żeby eksponować na nim cyfry dziesiętne, należy zaświecać odpowiednie diody a inne wygaszać. Stąd, w przykładzie przedstawionym niżej, żeby wyświetlić dziesiętną cyfrę 2 – diody nr 2 i 5 należy wygasić a pozostałe powinny świecić. Podobnie, w przypadku cyfry 8 powinny świecić wszystkie segmenty wyświetlacza. Celem zadania jest opracowanie takiego układu, który będzie przekształcać 4-bitowe dane wejściowe A na 7 sygnałów sterujących poszczególne segmenty wyświetlacza, w taki sposób, żeby dla A równego 0000 świeciła cyfra 0, dla A = 0101 cyfra 5, dla A = 0111 cyfra 7, itd.



W module DE2-115 zastosowano wyświetlacze typu wspólna anoda, tzn. anody diod we wszystkich segmentach są podłączone do dodatniego potencjału zasilania. Stąd, w celu zaświecenia określonego segmentu na jego anodę należy podać poziom niski, czyli zero logiczne.



Zatem chcąc wyświetlić, np. cyfrę 4, na wejścia wyświetlacza należy podać wektor 7-bitowy o wartości: 0011001, w którym bit LSB równy 1 (położony z prawej strony) wygasza segment zerowy, a bit MSB (położony z lewej strony) równy 0 zaświeca segment szósty.



W sprawozdaniu proszę zawrzeć kilka zdjęć pracującego układu, dla wybranych przypadków. Poniżej zawarto kod z tego zadania, który jest niekompletny i należy go uzupełnić.

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3
 4 entity Jakis_tam is
 5 port ( A : in std_logic_vector(3 downto 0);
 6        W : out std_logic_vector(6 downto 0) );
 7 end entity;
 8
 9 architecture sobie_kod of Jakis_tam is
10    begin
11       with A select
12          W <= "1000000" when "0000", --0
13               "1111001" when "0001", --1
14               "......." when "....", --2
15               "......." when "....", --3
16               "......." when "....", --4
17               "......." when "....", --5
18               "......." when "....", --6
19               "......." when "....", --7
20               "0000000" when "1000", --8
21               "0010000" when "1001", --9
22               "1111111" when others;
23    end sobie_kod;
24
```

Literatura:

1. Barski M., Jędruch W.: *Układy cyfrowe, podstawy projektowania i opisu w języku VHDL*, Wydawnictwo Politechniki Gdańskiej, 2007.
2. Dueck R., Digital Design with CPLD Applications and VHDL 2nd Edition, Cengagae Learning Inc., 2020
3. Grodzki L., Owieczko W., Podstawy techniki cyfrowej, Wydawnictwo PB, 2006
4. Hławiczka A., Garbolino T., Laboratorium podstaw techniki cyfrowej, Gliwice: Wydaw. Politechniki Śląskiej, 2010.
5. Tyszer J., Mrugalski G., Pogiel A., Czysz D., Technika cyfrowa: zbiór zadań z rozwiązaniami, Legionowo: Wydaw. BTC, 2010.
6. Intel, Intel FPGA Integer Arithmetic IP Cores User Guide, https://www.intel.com/content/www/us/en/docs/programmable/683490/20-3/lpm-add-sub-adder-subtractor.html, dostępność 8.09.2025.
7. Intel, Introduction to Intel® FPGA IP Cores, https://www.intel.com/programmable/technical-pdfs/683102.pdf, dostępność 8.09.2025.

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 3

Struktury hierarchiczne i wprowadzenie do układów sekwencyjnych PLD

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcję opracował:

dr inż. Marian Gilewski

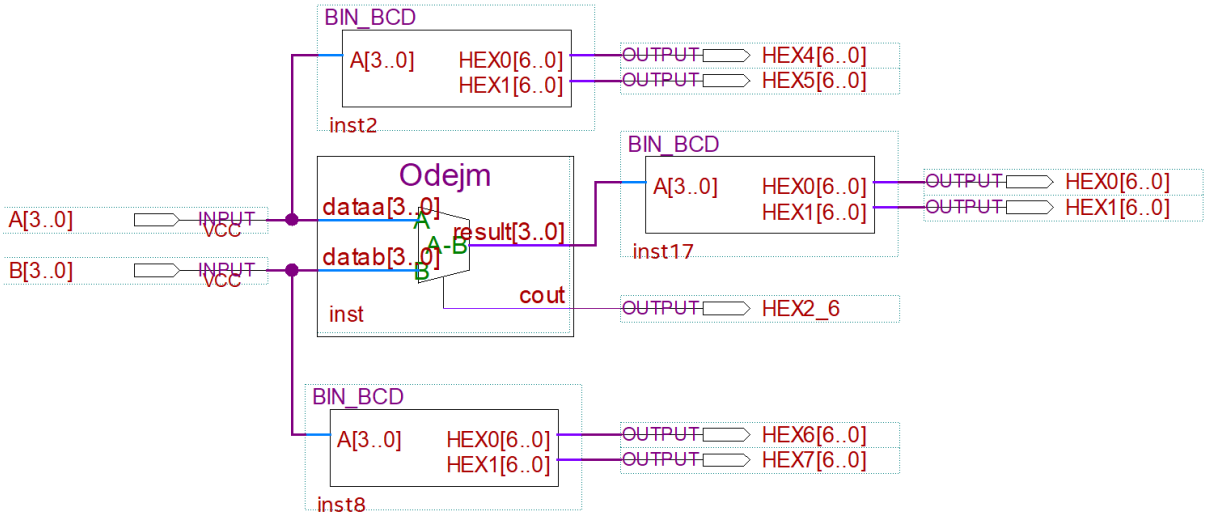
Białystok 2025

Cel ćwiczenia.

Celem ćwiczenia jest poznanie techniki dekompozycji złożonych układów na moduły składowe, syntezy modułów składowych a następnie ich integracji w docelowy system – czyli synteza struktur hierarchicznych. W drugiej części ćwiczenia następuje wprowadzenie do syntezy układów sekwencyjnych, na tym etapie będzie to implementacja przykładowych przerzutników synchronicznych.

Zadanie laboratoryjne 3.1

Zamiast rysować duży skomplikowany schemat lub pisać długi nieczytelny kod – układ można podzielić na mniejsze elementy składowe/komponenty. Każdy komponent można niezależnie od innych przetestować a potem połączyć we wspólnym schemacie pliku top-level. Plik top-level może być strukturą mieszaną, tzn. może łączyć graficzne moduły biblioteczne z komponentami opisanymi przez użytkownika w kodzie VHDL. Jest to unikalna użyteczna cecha projektowania układów w strukturach programowalnych, niedostępna w innych systemach cyfrowych, np. w mikroprocesorach.
Tworzenie takiej struktury prześledźmy na przykładzie układu odejmującego pokazanego na poniższym rysunku:

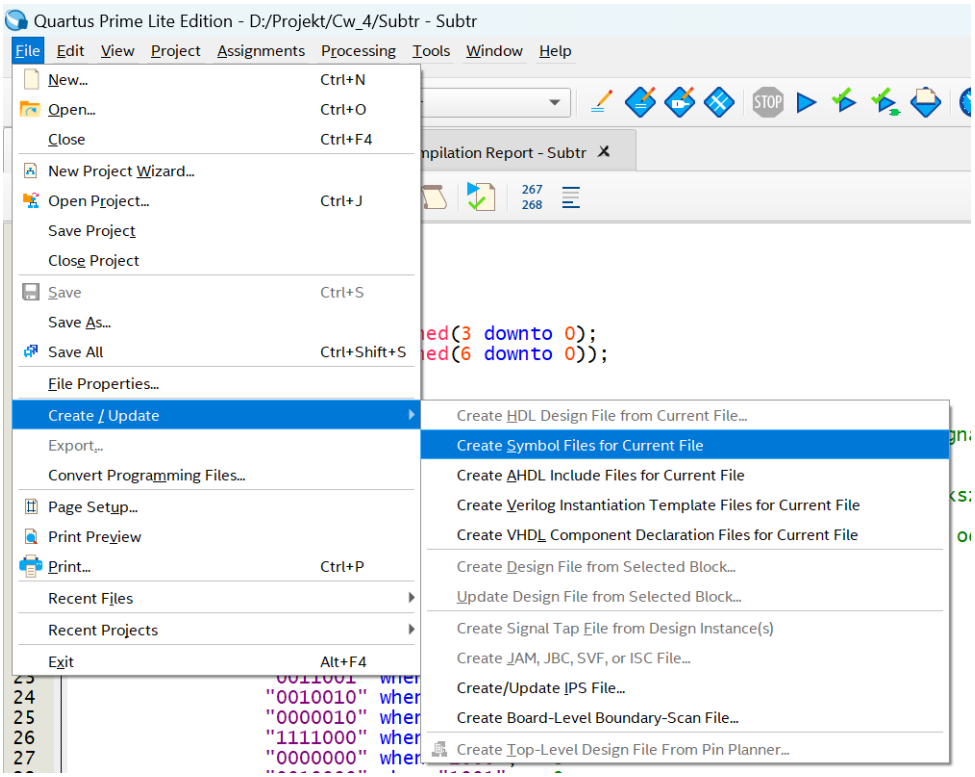


Układ składa się z głównego modułu odejmującego (Odejm) oraz 3 identycznych modułów pomocniczych (BIN_BCD), których rolą jest wyświetlanie dziesiętne 4-bitowych danych binarnych.
Kolejność tworzenia tego projektu jest następująca:
a) należy utworzyć nowy projekt w nowy folderze z plikiem graficznym jako top-level;
b) z katalogu IP zdefiniować makrofunkcję LPM_ADD_SUB jako układ odejmujący;
c) symbol wygenerowanej makrofunkcji należy pobrać z katalogu Project menu edytora;
d) następnie należy otworzyć nowy plik VHDL, do którego wkleić kod zadania 6 instrukcji 2;
e) kod należy uzupełnić w sposób pokazany w poniższym listingu;

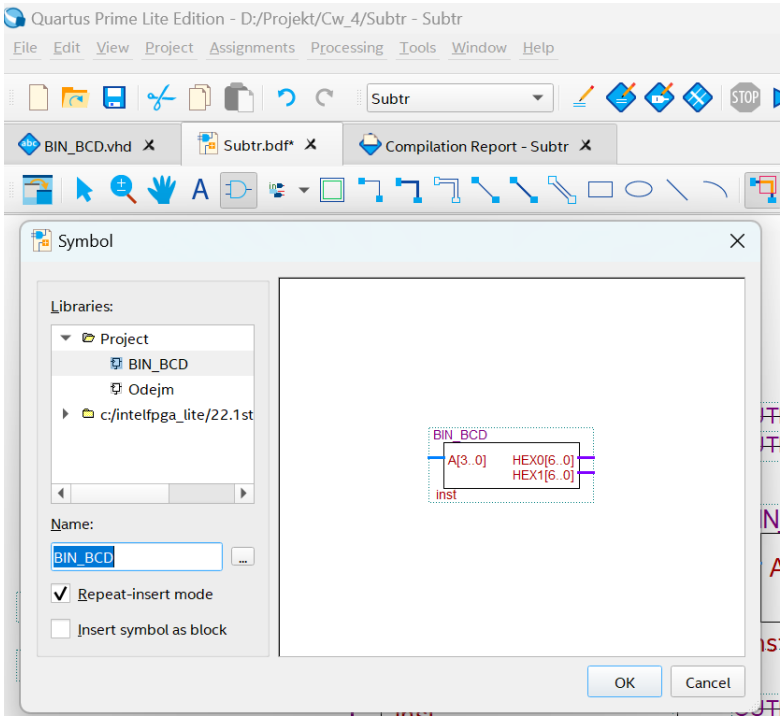
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity BIN_BCD is
port ( A           : in  unsigned(3 downto 0);
       HEX0, HEX1  : out unsigned(6 downto 0));
end entity;

architecture ar00 of BIN_BCD is
   signal roznica, L1, to_H0, to_H1 : unsigned(3 downto 0); -- pomocnicze sygnały wewnętrzne
   begin

      roznica <= A - "1010"; -- reszta po odjęciu wartości 10 dla danych większych od 9
      to_H0   <= A      when A < "1010" else roznica;
      to_H1   <= "1111" when A < "1010" else "0001"; -- jeżeli dane mniejsze od 10 wygaś H1

      with to_H0 select
         HEX0 <= "1000000" when "0000", --0
                 "1111001" when "0001", --1
                 "0100100" when "0010",-- 2
                 "0110000" when "0011",-- 3
                 "0011001" when "0100",-- 4
                 "0010010" when "0101",-- 5
                 "0000010" when "0110",-- 6
                 "1111000" when "0111",-- 7
                 "0000000" when "1000",-- 8
                 "0010000" when "1001",-- 9
                 "1111111" when others;

      with to_H1 select
         HEX1 <= "1111001" when "0001", --1
                 "1111111" when others;

   end ar00;
```

e) pozostając w aktywnym pliku VHDL należy wygenerować ekwiwalent graficzny/symbol kodu - używając funkcji:

File → Create / Update → Create Symbol Files for Current File;



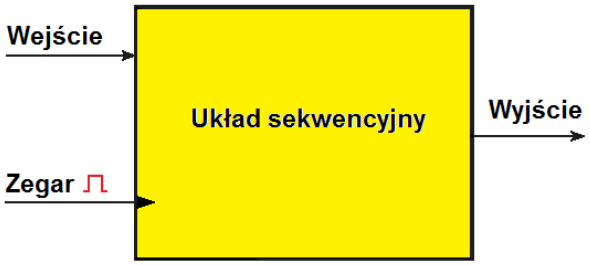
f) następnie z katalogu Project edytora graficznego należy pobrać symbol wygenerowanego komponentu i uzupełnić schemat dodając brakujące elementy:



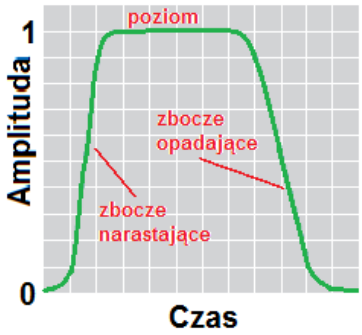
g) projekt należy skompilować, zaprogramować układ i przetestować.
Port A należy połączyć z przełącznikami SW[3] ÷ SW[0], port B z SW[17] ÷ SW[14], porty o nazwach HEXx[6..0] należy połączyć z odpowiadającymi im nazwą wyświetlaczami 7-segmentowymi, port HEX2_6 proszę połączyć z segmentem nr 6 wyświetlacza HEX2.
W sprawozdaniu proszę umieścić kilka zdjęć pracującego układu, dla różnych wartości danych wejściowych.

Zadanie laboratoryjne 3.2

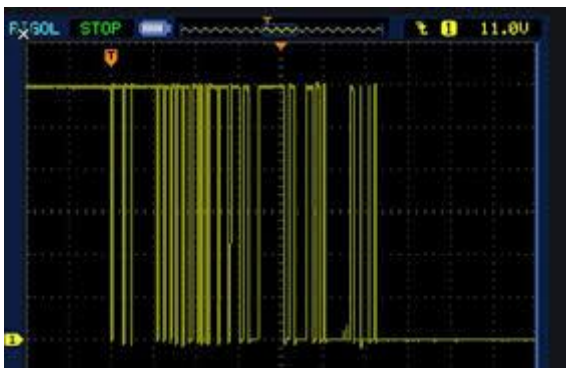
Układy sekwencyjne stanowią dominującą kategorię wszystkich układów cyfrowych. W tych układach, w odróżnieniu od wcześnie poznanych układów kombinacyjnych, sygnał wyjściowy zależy nie tylko od zmian sygnałów wejściowych, ale również od dodatkowego sygnału sterującego. Ten dodatkowy sygnał, zwany sygnałem zegarowym, wyznacza punkt czasowy reakcji układu i zmiany jego zachowania. Unikalną cechą układu sekwencyjnego jest również wewnętrzna pamięć zachowania, czyli zdolność przechowywania informacji o zachowaniu/stanie układu w przeszłości. Na podstawie stanów przeszłych i wartości bieżących sygnałów wejściowych – w aktywnym obszarze sygnału zegarowego, układ wyznacza wartości nowych sygnałów wyjściowych i mówimy, że przechodzi do nowego stanu. Sygnał zegarowy najczęściej ciągiem impulsów prostokątnych, który wytwarzany w zewnętrznym dedykowanym układzie/generatorze sygnału zegarowego.



W układach sekwencyjnych to częstotliwość sygnału zegarowego decyduje o szybkości pracy systemu, a zatem określa jego charakterystyki dynamiczne. Komutacja/przełączanie wyjść układu sekwencyjnego nie następuje w dowolnym punkcie impulsu zegarowego. Taka zmiana może dokonać się w jednym z trzech obszarów czasu trwania impulsu zegarowego:
- w momencie wystąpienia narastającego zbocza sygnału zegarowego,
- w momencie wystąpienia opadającego zbocza sygnału zegarowego,
- lub w czasie trwania wysokiego poziomu impulsu zegarowego.



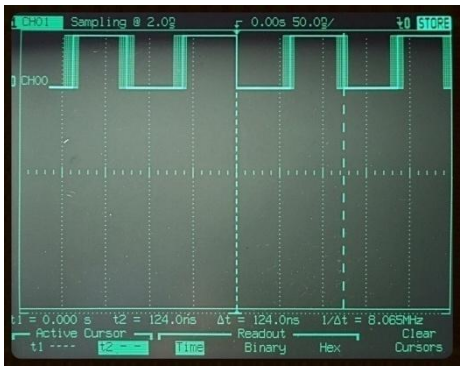
Układy sekwencyjne składają się głównie z przerzutników i prostych elementów kombinacyjnych, które mogą być składnikami bardzo skomplikowanych układów scalonych, co nie zawsze są wprost oczywiste i widoczne dla użytkownika. Użytkownik zazwyczaj postrzega układ zaciskowo, przez pryzmat wyprowadzeń i cech sygnałów wejściowych i wyjściowych.
Podczas uruchamiania wolnozmiennych układów sekwencyjnych, problemem jest ręczne wyzwalanie impulsów zegarowych, gdy nie można zastosować specjalistycznego układu. W przypadku impulsów zegarowych generowanych manualnie, z użyciem mechanicznych przełączników/przycisków/kluczy, obserwuje się zjawisko mechanicznego drgania styków (bouncing). Wynika ono ze skończonej masy elementów przełączających i związanym z nią efektem bezwładności czasowej oraz wchodzeniem styków przełącznika w stan gasnących drgań/oscylacji. W efekcie, w momencie włączenia/wyłączenia przełącznika nie pojawia się czyste/ostre przejście sygnału napięciowego ze stanu wysokiego do niskiego, lecz jest ono poprzedzone ciągiem przypadkowych wytwarzanych impulsów - zanim styki przełącznika ustabilizują się mechanicznie w nowej pozycji. Tę sytuację przy przełączaniu ze stanu wysokiego do niskiego pokazano na poniższym rysunku w formie oscylogramu:



Czas trwania stanu przejściowego oraz liczba niechcianych dodatkowych impulsów są losowe. Zależą one nie tylko od technologii przełącznika, ale i od czasu oraz siły nacisku przełącznika Te niepożądane impulsy przejściowe mogą zakłócać pracę układu sekwencyjnego, gdyż przez

sterowany układ są traktowane, jako kolejne impulsy zegarowe. Z tego powodu z przełącznikami mechanicznymi powinny współpracować układy eliminacji drgań styków (debouncing) sprzętowe lub programowe.

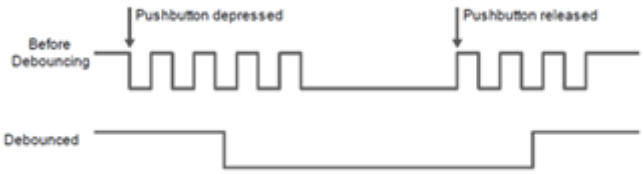
Drugim negatywnym zjawiskiem towarzyszącym sygnałom zegarowym są szumy fazy (jitter), pokazane na kolejnym rysunku:



Tym razem, ich wpływ ujawnia się w scalonych/dedykowanych generatorach sygnału zegarowego wysokiej częstotliwości, w których czas trwania impulsu jest porównywalny z czasem fluktuacji/drgań fazy. Jitter powoduje, że częstotliwość i czas trwania kolejnych impulsów zegarowych nie jest stabilna. Kolejne impulsy, opóźniają się lub trwają krócej, co ma wpływ na pracę reszty układu. Efekt jest trudny do wyeliminowania - gdyż jest zależny od technologii wykonania i techniki montażu generatorów zegarowych. To negatywne zjawisko tylko częściowo eliminują dodatkowe układy pętli synchronizacji fazowej, gdyż ma ono charakter losowy/przypadkowy.

W laboratorium, drgań styków nie są pozbawione również przełączniki SW modułu DE2-115. Stąd nie zaleca się używania przełączników SW do wytwarzania sygnału zegarowego.

Występujące w module klucze Key[0] ÷ Key[3] mimo własnych drgań, współpracują z dodatkowym układem do ograniczania efektu bouncing, co pokazano na poniższym rysunku:

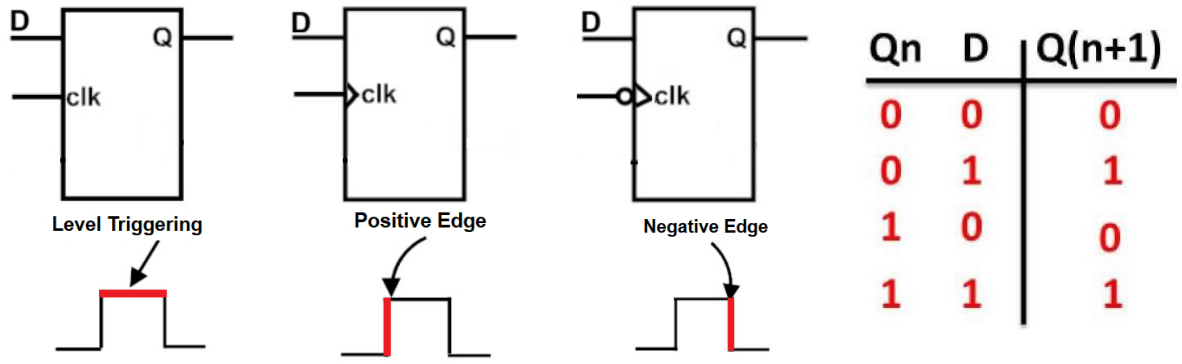


Stąd w czasie zajęć zaleca się korzystanie z kluczy Key[1] ÷ Key[3] do wytwarzania wolnozmiennych/jednostkowych impulsów zegarowych. Negatywną cechą tych kluczy, jest fakt, że ich sygnał wyjściowy jest odwrotnej notacji. T znaczy, nie przyciśnięty klucz wytwarza na wyjściu poziom wysoki (1), po przyciśnięciu przechodzi do poziomu niksiego (0) a po zwolnieniu klucza wraca do poziomu wysokiego. Większość układów jest wyzwalana narastającym zboczem sygnału zegarowego, co wymagana przejścia do poziomu wysokiego,

czyli po użyciu klucza wytworzenia dodatniego impulsu/jedynki logicznej. Z tego względu, zaleca się wyjścia kluczy Key łączyć z badanym układem przez negatory/bramki NOT.

Proszę uruchomić układ przedstawiony na poniższym rysunku, łącząc wejście z Key[2] z wyjściem LEDR[0] przez dodatkowy negator. Należy zaobserwować etapy przełączania sygnału i opisać je w sprawozdaniu.

Zadanie laboratoryjne 3.3 W strukturach programowalnych jednym z najpopularniejszych przerzutników synchronicznych jest przerzutnik typu D. Na poniższym rysunku przedstawiono symbole 3 różnych realizacji przerzutników typu D, które są wyzwalane na trzy różne sposoby.

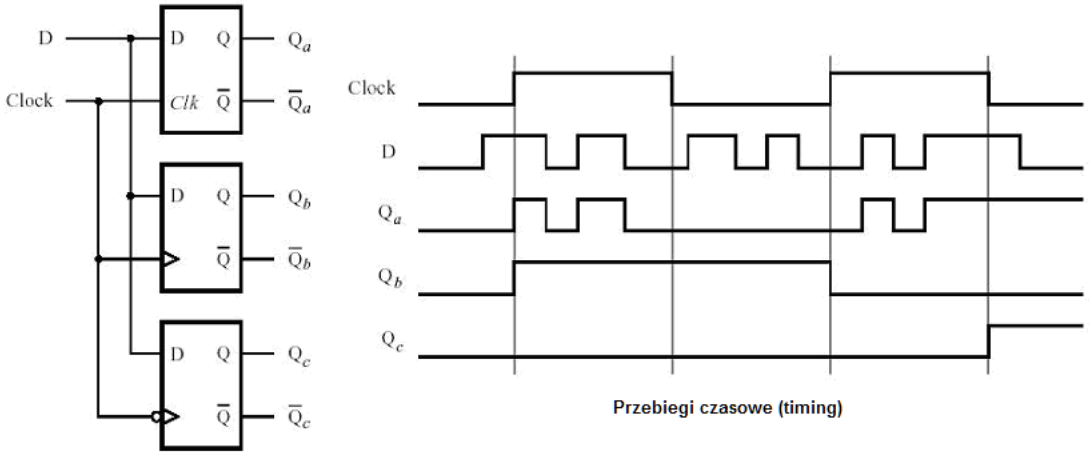


Qn	D	Q(n+1)
0	0	0
0	1	1
1	0	0
1	1	1

W symbolu elementu kształt doprowadzenia sygnału zegarowego **clk** wskazuje na sposób wyzwalania przerzutnika lub wejścia innego układu.
Z prawej strony rysunku znajduje się tabela przejść, która opisuje sposób pracy przerzutnika D. Oprócz sygnału zegarowego **clk**, przerzutnik posiada jednobitowe: wyjście **Q** i wejście danych **D**. W stanie nieaktywnym, przy braku impulsu zegarowego, przerzutnik pamięta zapisany wcześniej bit danych a wartość tego bitu widoczna jest na wyjściu Q - mówimy, że przerzutnik znajduje się w stanie Qn. W momencie pojawienia się impulsu zegarowego do przerzutnika wpisywana jest wartość bitu danych, która aktualnie znajduje się na wejściu D. Ta nowa wartość nadpisuje poprzednią – mówimy, że przerzutnik przechodzi do następnego stanu Q(n+1). Wartość sygnału widziana na wyjściu Q zależy od tego jaka była wartość sygnału wejściowego D w momencie ostatniego impulsu zegarowego. W czasie braku impulsów zegarowych – układ podtrzymuje/pamięta na wyjściu ostatnią zapisaną wartość wejściową.
Dla zrozumienia działania przerzutnika D wystarczy zapamiętanie zasady:
- '1' na wejściu D wpisuje stan '1' na wyjściu Q,
- '0' na wejściu D wpisuje stan '0' i ta wartość jest obserwowana na wyjściu Q.. W języku technicznym często upraszczamy mówiąc, że przerzutnik znajduje się w stanie '1' co oznacza stan wysoki na wyjściu, lub znajduje się w stanie niskim/'0', co oznacza stan '0' na wyjściu.

W ramach niniejszego zadania zdefiniujemy w VHDL i przetestujemy przerzutnik D wyzwalany na 3 różne poznane sposoby. W tym celu w tej samej jednostce projektowej, zgodnie ze schematem pokazanym na kolejnym rysunku, zdefiniujemy w 3 procesach VHDL 3 przerzutniki D wyzwalane na różne sposoby. Każdy z nich będzie obsługiwany przez ten sam sygnał zegarowy oraz to samo wejście danych D, natomiast wyjścia przerzutników będą połączone z 3 różnymi portami wyjściowymi, żeby zaobserwować moment przełączania.

Przebiegi z prawej strony rysunku, przedstawiają zachowania układów dla przykładowych kombinacji sygnału zegarowego i sygnału danych na wejściu D.

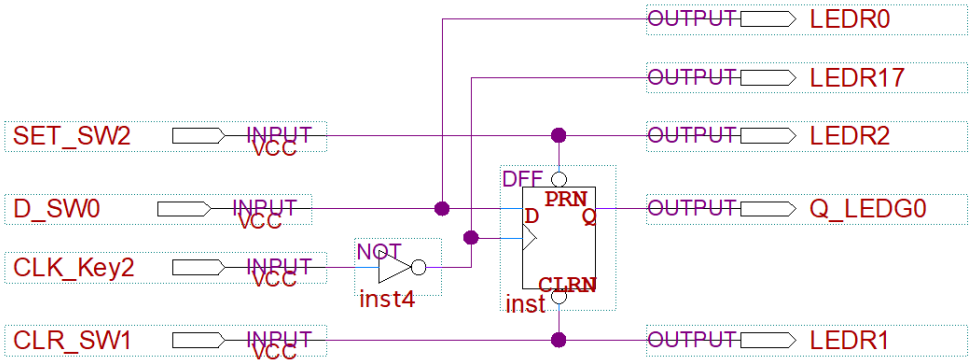


Poniżej przedstawiona jest zasadnicza część kodu opisującego powyższy układ. Jest on niekompletny w pozycji wielokropków. Proszę w ramach pracy domowej uzupełnić kod, a układ skompilować i przetestować symulacyjnie. Wyniki badań należy zamieścić w sprawozdaniu – włączając szczegółowy opis wyników symulacji.

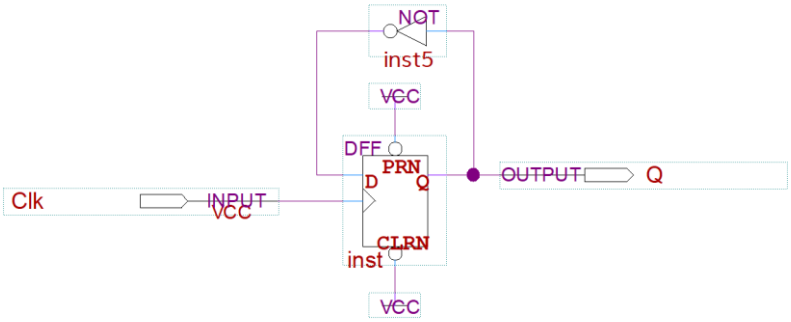
```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity DFFs is
5     port
6     (  Clk, D       : in std_logic;   -- połaczyć z Key[2] i SW[0]
7        LEDClk, LEDD : out std_logic;  -- połączyć z LEDG[4] i LEDR[0]
8        Qa, Qb, Qc   : out std_logic); -- połączyć z LEDR[15], LEDR[16], LEDR[17]
9  end entity;
10
11 architecture rtl of DFFs is
12    signal Int_Clk : std_logic;
13    begin
14       Int_Clk <= not Clk; -- realizacja układu z zadania 1
15       LEDClk  <= Int_Clk; -- podgląd sygnału zegarowego
16       LEDD    <= D;       -- podgląd wejścia D
17       process (Int_Clk)
18          begin
19             if (not (Clk = '1')) then
20                Qa <= D;
21             end if;
22          end process;
23       process (Int_Clk)
24          begin
25             if .....  then   -- (not (Clk = '1')) zastąpić rising_edge(Int_Clk)
26                Qb <= ... ;
27             end if;
28          end process;
29       process (Int_Clk)
30          begin
31             if ........ then  -- (not (Clk = '1')) zastąpić falling_edge(Int_Clk)
32                Qc <= ... ;
33             end if;
34          end process;
35 end rtl;
```

Zadanie laboratoryjne 3.4 Często podstawowe człony przerzutników są uzupełniane o 2 dodatkowe wejścia asynchroniczne S (Set) i R (Reset), służące do wymuszania ustawienia przerzutnika w stan '1' (Set) lub wyzerowanie do poziomu '0' (Reset). Taki przypadek pokazano na poniższym rysunku. Wejścia S i R mają wyższy priorytet niż wejście zegarowe i wejście danych, co oznacza, że w dowolnym momencie mogą one wymuszać zmianę stanu przerzutnika - niezależnie od stanu wejścia zegarowego i wejścia D. Jeżeli wejścia S i R są nieaktywne, przerzutnik pracuje w normalnym trybie synchronicznym – reagując na sygnał zegarowy i dane wejściowe. W rzeczywistości kombinacja pokazana na rysunku jest połączeniem 2 typów przerzutników w jednym układzie:
- przerzutnika asynchronicznego SR,
- przerzutnika synchronicznego D,
przy czym przerzutnik asynchroniczny ma wyższy priorytet niż przerzutnik synchroniczny.

Proszę przetestować poniższy układ, a opisane wyniki badań i zdjęcia podać w sprawozdaniu.



Zadanie laboratoryjne 3.5 Na kolejnym rysunku przedstawiono przykładowe, czasami przydatne zastosowanie przerzutnika D, popularnie nazywane „dwójką liczącą”. W ramach pracy domowej proszę zainicjować projekt oraz dokonać symulacji tego układu podając na jego wejście zegarowe sygnał prostokątny o okresie 100 ps i współczynniku wypełnienia 25%. Na podstawie otrzymanych wyników symulacji proszę określić, jakie funkcje (co najmniej dwie) realizuje badany układ. Wyniki symulacji proszę zamieścić w sprawozdaniu.



Zadanie laboratoryjne 3.6 W układach cyfrowych często jest przydatne/konieczne jest zastosowanie układu, który po włączeniu zasilania całego systemu wygeneruje impuls o określonym czasie trwania a po jego zakończeniu przejdzie w stan zawieszenia. Taki jednokrotny impuls generowany na żądanie może być np. sygnałem resetującym cały system (np. komputer) lub zapewnić rozpoczęcie pracy systemu po zaniku stanów przejściowych w

zasilaczu. Tej kategorii rozwiązania układowe należą do kategorii przerzutników monostabilnych – układów, które po otrzymaniu sygnału startowego przez pewien/określony czas utrzymują wysoki/niski poziom sygnału na wyjściu, a następnie przechodzą do stabilnego/nieskończonego stanu uśpienia, w którym sygnał na wyjściu jest niski/wysoki.

Proszę w ramach pracy domowej symulacyjnie przetestować poniższy kod opisujący przerzutnik monostabilny tak, podając na jego wejście zegarowe sygnał o częstotliwości 50 MHz. W poniższym kodzie zastosowano technikę Maszyną Stanów, którą będziemy doskonalić w ćwiczeniu nr 6.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Mono is
5     port( Clk_50MHz, Reset     : in  std_logic;
6                      output    : out std_logic);
7  end entity;
8
9  architecture rtl of Mono is
10    type state_type is (s0, s1);
11    signal state   : state_type;
12 begin
13    process (Clk_50MHz, Reset)
14       variable tmp : integer := 0;
15       begin
16          if Reset = '1' then
17             state <= s0;
18          elsif rising_edge(Clk_50MHz) then
19             case state is
20                when s0 =>
21                   output <= '1';
22                   tmp := tmp + 1;
23                   if tmp >= 25 then
24                      tmp := 0;
25                      state <= s1;
26                   end if;
27                when s1 =>
28                   output <= '0';
29                   state <= s1;
30             end case;
31          end if;
32       end process;
33 end rtl;
34
```

Literatura:

1. Barski M., Jędruch W., Układy cyfrowe: podstawy projektowania i opis w języku VHDL, Gdańsk: Wydawn. Politechniki Gdańskiej, 2011.
2. Kamionka-Mikuła H., Małysiak H., Pochopień B., Teoria układów cyfrowych. T.2, Układy sekwencyjne, Gliwice: Wydaw. Politechniki Śląskiej, 2013.
3. Intel, Intel FPGA Integer Arithmetic IP Cores User Guide, https://www.intel.com/content/www/us/en/docs/programmable/683490/20-3/lpm-add-sub-adder-subtractor.html, dostępność 30.08.2023.

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 4

Mechanizmy konwersji i przechowywania informacji – rejestry i układy licznikowe

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

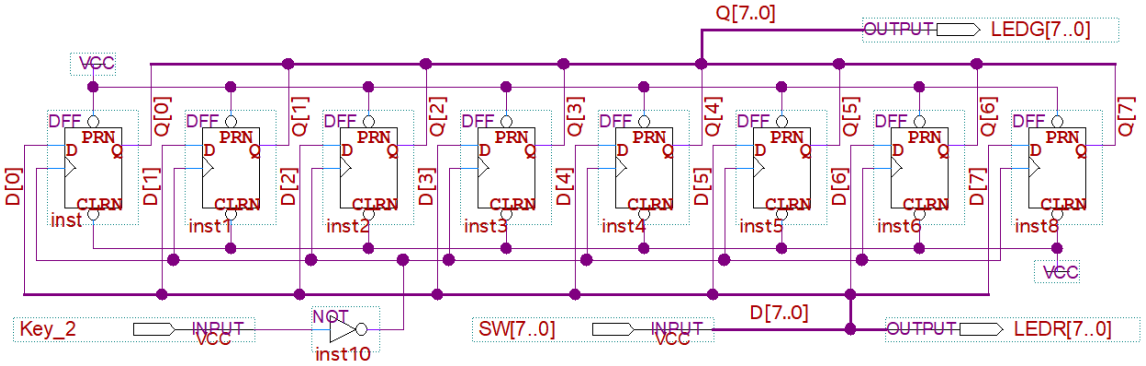
Instrukcję opracował:

dr inż. Marian Gilewski

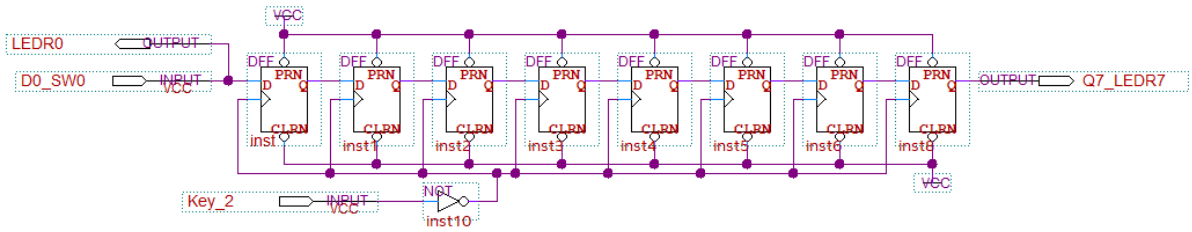
Białystok 2025

Cel ćwiczenia.
Celem ćwiczenia zrozumienie funkcjonowania i nabycie umiejętności w syntezie układów: pamiętających dane cyfrowe dłuższe niż 1 bit. Ponadto przetestowane będą układy przesuwające dane cyfrowe oraz układy serializacji/deserializacji danych i układy zliczania impulsów cyfrowych.

Zadanie laboratoryjne 4.1 W końcowej części poprzedniego ćwiczenia testowany był układ pojedyńczego przerzutnika D, czyli elementu posiadającego właściwość pamiętania 1 bitu danych. Taka pamięć jest niewystarczająca we współczesnych systemach cyfrowych, gdzie przetwarzane są pakiety danych o szerokościach 1 bajta, 16 bitów, 32 czy 64 bitów. Stąd intuicyjnie wyczuwana jest konieczność stosowania większej liczby odpowiednio połączonych przerzutników.
W tym zadaniu będzie testowany układu złożony z 8 przerzutników D, który umożliwia wpisywanie i pamiętanie bajtu danych - jest to podstawowa konfiguracja rejestru równoległego z wpisem równoległym. Jak widać na poniższym rysunku: wszystkie wejścia zegarowe przerzutników są połączone ze sobą, zaś do każdego wejścia danych przerzutnika jest doprowadzony inny bit danych. W momencie pojawienia się impulsu zegarowego dane z wejść D wpisywane są jednocześnie do wszystkich przerzutników. Proszę zaprogramować i przetestować poniższy układ a wyniki badań zamieścić w sprawozdaniu.



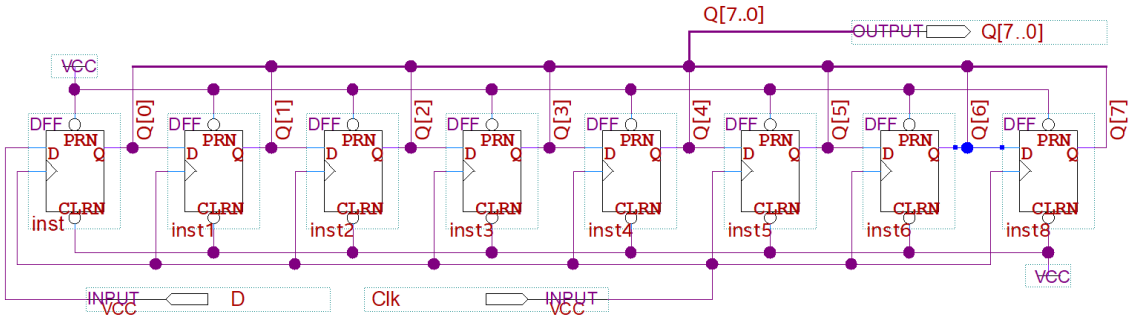
Zadanie laboratoryjne 4.2 Układ połączeń przerzutników z poprzedniego zadania może być inny – np. taki jak pokazano na poniższym rysunku:



co tworzy konfigurację znaną jako rejestrem szeregowy z wyjściem szeregowym. W tym układzie, wejściowy sygnał danych jest podawany szeregowo bit po bicie na wejście D pierwszego przerzutnika. Oznacza to, że najpierw na wejściu D ustawiany jest najmłodszy bit danych (LSB) a następnie podawany jest zegarowy impuls zapisujący, następnie podawany jest kolejny bit danych i kolejny impuls zegarowy – jednocześnie dane z wyjścia pierwszego przerzutnika są przepisywane do kolejnego, itd. aż do wpisania bitu MSB danych. Zatem

wprowadzenie 8 bitów danych wymaga 8 impulsów zapisujących. W kolejnych taktach zegarowych zapisane dane są wyprowadzane wyjście szeregowe Q7_LEDR7, przesuwane dane są konsekwentnie uzupełniane na wejściu D0_SW0. W rejestrze szeregowym, wprowadzana szeregowo informacja propaguje przez wszystkie przerzutniki – od pierwszego do ostatniego. Zatem informacja wyjściowa opóźnia się o czas równy iloczynowi liczby przerzutników w rejestrze i okresu impulsów zegarowych. Jest tu podstawowe zastosowanie tego układu – realizacja opóźnień w systemach cyfrowych.

Konfiguracja pokazana na kolejnym rysunku dotyczy rejestru z szeregowym wejściem danych i wyjściami szeregowym oraz równoległym. Po wczytaniu 8-bitów danych, które są podawane szeregowo na wejście D – możliwy jest odczyt całego słowa równolegle z porty Q[7..0] lub szeregowo z wyjścia Q[7]. Układ taki, w którym szeregowo wprowadza się dane i odczytuje równolegle, nazywa się rejestrem z wejściem szeregowym i wyjściem równoległym – lub w telekomunikacji deserializerem, gdyż zamienia dane szeregowe na równoległe.

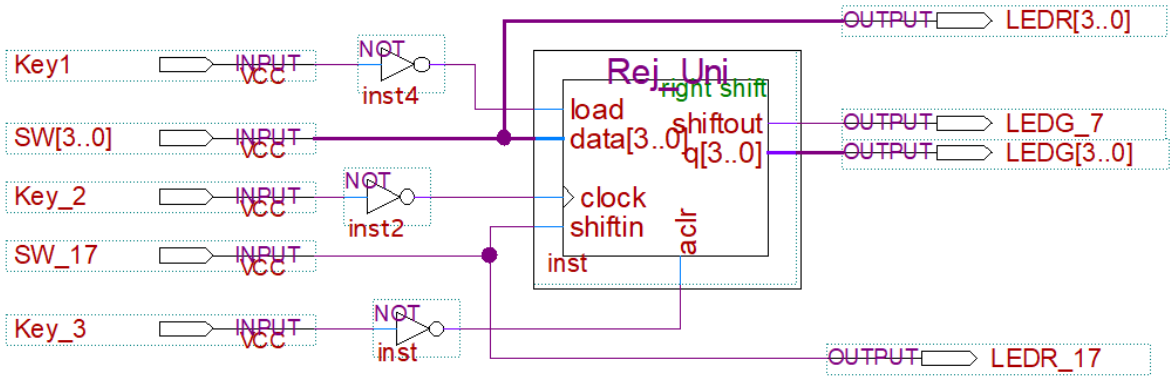


Możliwa jest też konfiguracja układu z zadania 1, w której można zastosować szeregowe wyprowadzenie danych z wyjścia Q[7] w kolejnych 8 taktach zegarowych, po równoległym zapisie danych wejściowych. Tak układ jest rejestrem z wejściem równoległym i wyjściem szeregowym – czyli serializerem.

Ze względu na ograniczony czas trwania laboratorium i malejące znaczenie układów budowanych z dyskretnych przerzutników, wszystkie wyżej omówione układy rejestrów, których działania opisano symbolicznie w poniższej tabeli - nie będą testowane. Zainteresowani

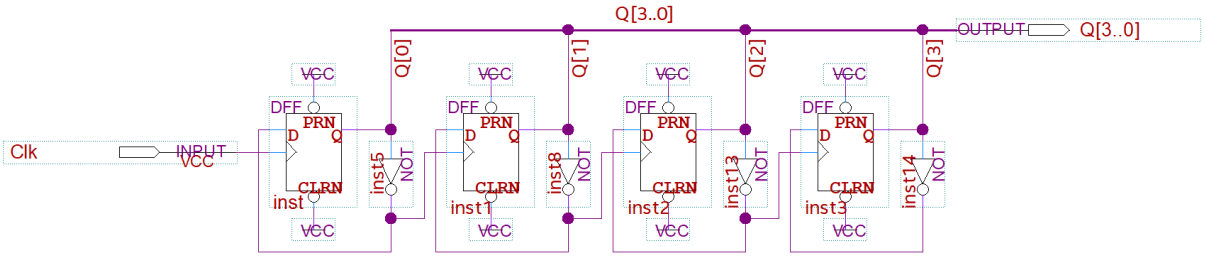
Wejście	Wyjście	Rodzaj	Zachowanie
szeregowe	szeregowe	szeregowy	→ 0 1 1 0 1 1 0 1 →
szeregowe	równoległe	deserializer	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ → 1 1 1 1 0 0 0 0
równoległe	szeregowe	serializer	1 1 0 1 1 0 0 1 → ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑
równoległe	równoległe	równoległy	↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑ 1 0 1 0 1 0 1 0 ↑ ↑ ↑ ↑ ↑ ↑ ↑ ↑

studenci mogą przebadać te układy symulacyjnie w ramach pracy własnej. Natomiast w ramach laboratorium będzie przetestowany model IP rejestru uniwersalnego **LPM_SHIFTREG**, którego układ pomiarowy pokazano na rysunku poniższym:



Proszę zaimplementować, zaprogramować i przetestować konfigurację powyższego rejestru uniwersalnego – badając funkcje: zerowania, szeregowego i równoległego wprowadzania informacji oraz szeregowego i równoległego wyprowadzania zapisanych danych.

Zadanie laboratoryjne 4.3 Oprócz funkcji rejestrowych, układy przerzutnikowe są stosowane do rejestracji/zliczania liczby impulsów wchodzących na wejście układu w określonym przedziale czasu. Ich parametrami są: maksymalna częstotliwość pracy, pojemność wyznaczana przez liczbę bitów/przerzutników oraz tryby pracy. Z każdym impulsem wejściowym licznik liczący w przód zwiększa swój stan o 1, lub zmniejsza o 1 w przypadku liczników odliczających/rewersyjnych. Liczba zarejestrowanych impulsów jest zapamiętywana w sposób cyfrowy – stąd licznik 4-bitowy (złożony z 4 przerzutnikówa), posiada pojemność/może zliczyć 16 kolejnych impulsów a po zapełnieniu zakresu rozpoczyna on zliczanie od początku.



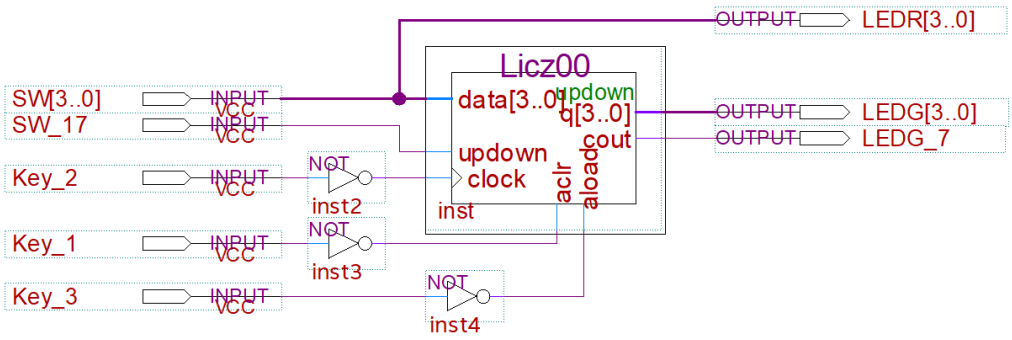
Przykładowe kolejne zliczane wartości licznika 4-bitowego, który zlicza w przód w naturalnym kodzie binarnym pokazano w poniższej tabeli. Wspomniany licznik przyjmuje kolejne wartości binarne, w szczególności układ może zliczać w innym systemie np. Graya. Podobnie licznik nie musi przyjmować wszystkich wartości danego systemu. Często w sposób sprzętowy skraca się cykl pracy licznika – tworząc układ pracujący w systemie modulo. Np. jeżeli 4-bitowy układ opisany poniższą tabelą zlicza tylko w zakresie od 0 do 9 a następnie zeruje się i rozpoczyna pracę od początku – mówimy o liczniku modulo 10. W stosunku do 4-bitowych liczników binarnych modulo 10 używa się też nazw: licznik BCD, dekada licząca.

Nr impulsu zegarowego	Stan bieżący				Stan następny			
	wartości				wartości			
0	0	0	0	0	0	0	0	1
1	0	0	0	1	0	0	1	0
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	1	0	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	1	0
6	0	1	1	0	0	1	1	1
7	0	1	1	1	1	0	0	0
8	1	0	0	0	1	0	0	1
9	1	0	0	1	1	0	1	0
10	1	0	1	0	1	0	1	1
11	1	0	1	1	1	1	0	0
12	1	1	0	0	1	1	0	1
13	1	1	0	1	1	1	1	0
14	1	1	1	0	1	1	1	1
15	1	1	1	1	0	0	0	0

W zależności od sposobu połączenia wejść zegarowych przerzutników w licznikach, historycznie klasyfikowano je, jako liczniki asynchroniczne i liczniki synchroniczne. Jeżeli, sygnały zegarowe kolejnych przerzutników łączą się z wyjściami poprzednich mamy do czynienia z licznikiem asynchronicznym, w którym poszczególne przerzutniki pracują niezależnie.

W licznikach synchronicznych, wejścia zegarowe wszystkich przerzutników są połączone i sterowane tym samym sygnałem wejściowym, co oznacza te same momenty przełączania przerzutników.

Współcześnie stosowane są albo scalone układy liczników, albo komponenty licznikowe struktur programowalnych lub mikrokontrolerów. W związku z tym użytkownik tylko konfiguruje gotowy licznikowy moduł lub też definiuje go w języku HDL. Przykłady takiego uniwersalnego licznika **LPM_COUNTER** jest testowany w niniejszym zadaniu:



Korzystając z zasobów makrofunkcji IP programu Quartus II, proszę skonfigurować, uruchomić i przetestować moduł uniwersalnego licznika LPM_COUNTER, demonstrując jego podstawowe funkcje:
- wybór kierunku zliczania,
- wybór systemu liczbowego,
- zerowanie i ustawianie synchroniczne oraz asynchroniczne,
- wpis równoległy wartości początkowej,
- pracę modulo,
- przepełnienie licznika.

Ustawienia licznika pokazano na powyższym rysunku. Wyniki badań proszę zamieścić w sprawozdaniu, dokładnie opisując.

Literatura:

1. Barski M., Jędruch W., Układy cyfrowe: podstawy projektowania i opis w języku VHDL, Gdańsk: Wydawn. Politechniki Gdańskiej, 2011.
2. Kamionka-Mikuła H., Małysiak H., Pochopień B., Teoria układów cyfrowych. T.2, Układy sekwencyjne, Gliwice: Wydaw. Politechniki Śląskiej, 2013.

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 5

Pamięci wbudowane FPGA - o dostępie swobodnym, pamięci stałe i kolejki danych

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

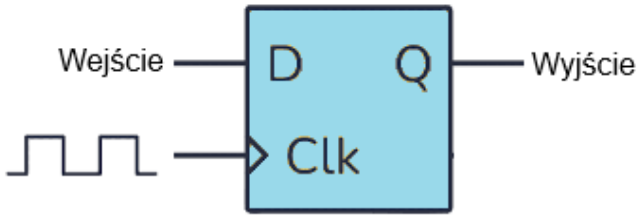
Instrukcję opracował:

dr inż. Marian Gilewski

Białystok 2025

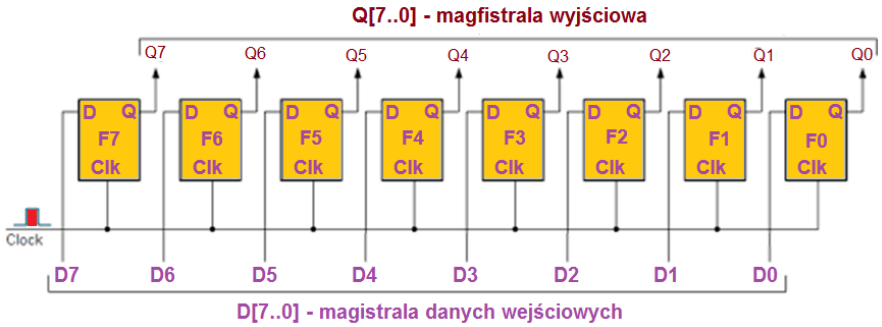
Cel ćwiczenia.
Celem ćwiczenia jest nabycie umiejętności syntezy różnych rodzajów pamięci w strukturach programowalnych. Są one dostępne w postaci makrofunkcji IP w systemie Quartus II. Zakres ćwiczenia obejmuj syntezę pamięci: ROM, RAM i FIFO.

Wprowadzenie.
Pojedyńczy przerzutnik (ang. flip-flop) posiada właściwość pamiętania tylko 1 bitu danych:

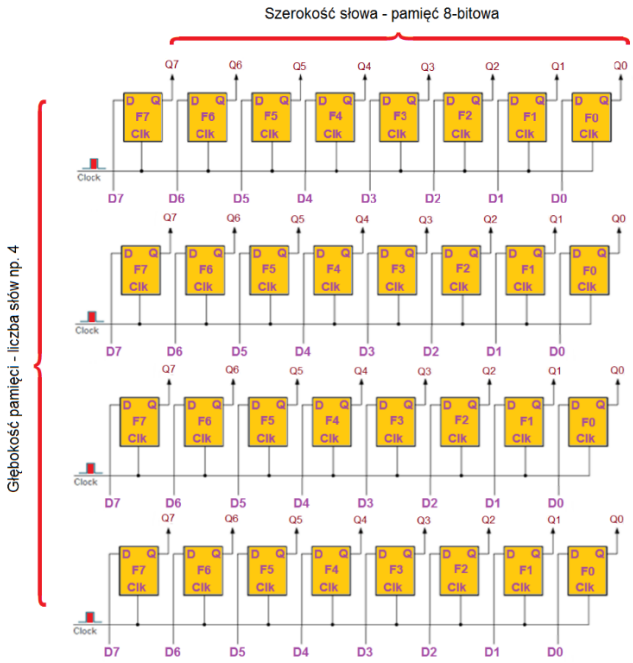


(źr.: iStockphoto.com)

Zespół połączonych przerzutników, tworząc rejestr może zapamiętać słowo – np. 1 bajt:



Z kolei zespół rejestrów może pamiętać większą liczbę słów, stanowiąc matrycę danych pamięci RAM:



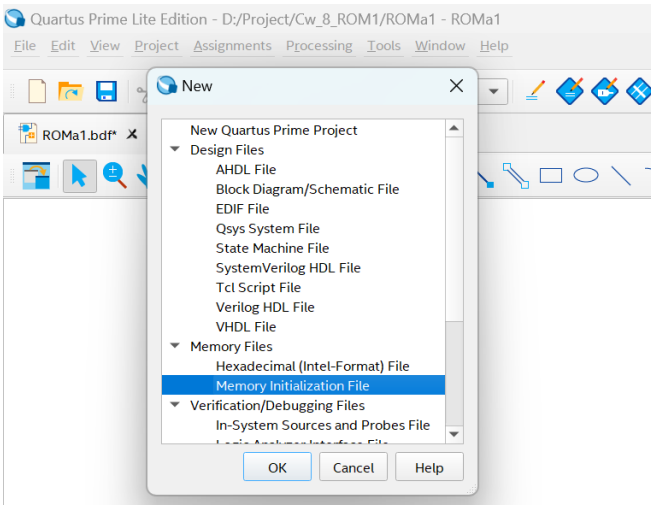
W przypadku pokazanym na powyższym rysunku, jest to pamięć 8-bitowa o pojemności 32 bitów i głębokości 4 słów – czyli o organizacji 4 x 8. W innych rodzajach pamięci przechowywanie informacji realizowane jest w innych niż przerzutnik rozwiązaniach technicznych, takich jak wewnętrzna pojemność lub izolowana bramka tranzystora.
Zakres ćwiczenia obejmuje obsługę i testowanie wewnętrznych pamięci ROM oraz RAM i FIFO, które mogą być implementowane w strukturze FPGA/PLD.

Zadanie laboratoryjne 5.1 – synteza pamięci ROM.
Jest to pamięć, w której zapisane są na stałe wartości o charakterze parametrycznym, tablice znaków, konwertery kodów, współczynniki linearyzacji, itp. Dane przechowywane w tablicy pamięci są zapisywane w postaci słów wielobitowych. Konkretne słowo jest odnajdywane w matrycy pamięci poprzez podanie unikalnego adresu, który jest sygnałem wejściowym. Wszystkie pamięci w strukturach programowalnych są typu synchronicznego, tzn. wymagają doprowadzenia zewnętrznego sygnału zegarowego. Traktując magistralę adresową, jako sygnały wejściowe, a magistralę danych wyjściowych jako odpowiadające im sygnały wyjściowe, za pomocą ROM można budować wszystkie układy kombinacyjne/współbieżne. Taki układ, będzie pracował szybciej niż jego odpowiednik zrealizowany innymi technikami. Jedynym ograniczeniem jest tu pojemność bloków pamięci wewnętrznej FPGA. Metodą odwzorowania tablicy prawdy w ROM, można opisywać skomplikowane zmierzone zależności funkcyjne między wejściami a wyjściami, co często jest trudne w zapisie analitycznym.

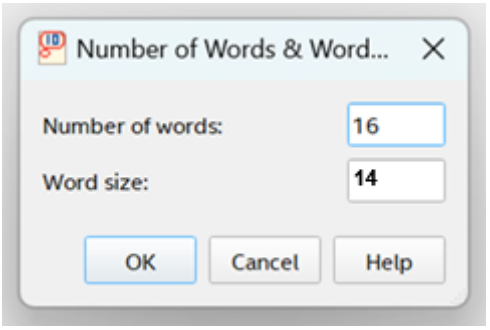
Synteza ROM w Quartus II jest dwuetapowa: najpierw należy zainicjować plik z zawartością danych wgrywanych do pamięci, a następnie skonfigurować model makrofunkcji IP. Proces projektowania prześledzimy to na przykładzie syntezy ROM, która będzie realizować funkcję dekodera 4-bitowych danych wejściowych na kod dwóch wyświetlaczy 7-segmentowych. Skoro wejściowe dane, będące jednocześnie adresem komórki w pamięci, są 4-bitowe, to zgodnie z zasadą 2^n – pamięć powinna posiadać co najmniej głębokość 16 słów. W związku z tym, że wyjściowa szyna danych będzie obsługiwać 2 wyświetlacze 7-segmentowe, jej szerokość będzie wynosić 2x7 = 14. Zatem, należy zaprojektować pamięć o organizacji 16 słów 14-bitowych. Poniższa tabela przedstawia częściowo wypełnioną matrycę pamięci:

Adres wejściowy				Dane wyjściowe – wejścia 2 wyświetlaczy 7-segmentowych													
W3	W2	W1	W0	Segment starszy							Segment młodszy						
				H16	H15	H14	H13	H12	H11	H10	H06	H05	H04	H03	H02	H01	H00
0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
0	0	1	0	1	1	1	1	1	1	1	0	0	1	0	0	1	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1	1	1	0	1	1	1	1	0	0	1	0	0	1	1	0	0	1
1	1	1	1	1	1	1	1	0	0	1	0	0	1	0	0	1	0

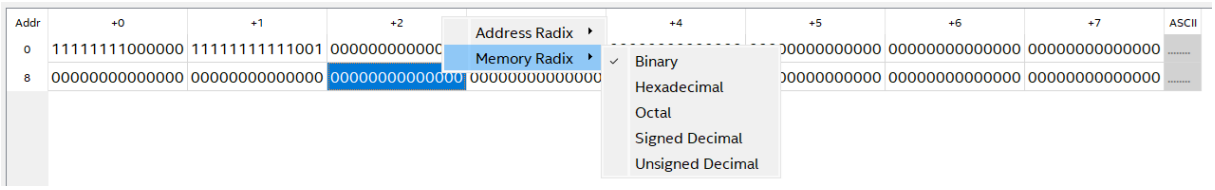
Definiując pamięć ROM, na początku otwieramy nowy projekt z graficznym plikiem top-level, a następnie otwieramy kolejny nowy plik, typu MIF (Memory Initialization File):



Po jego inicjacji określamy liczbę słów w pamięci i szerokość słowa:



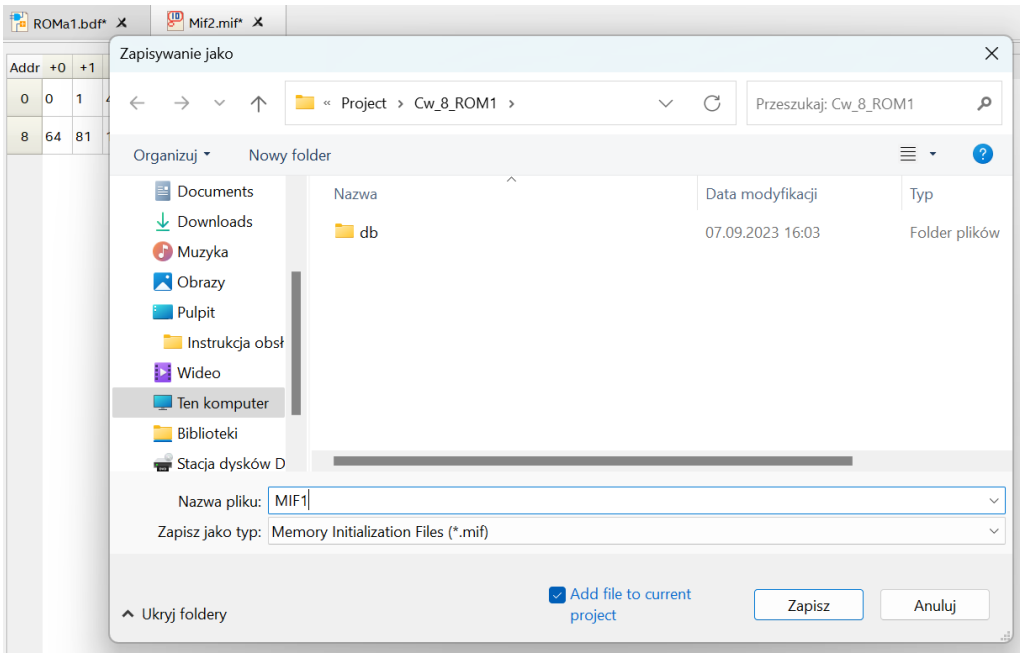
po zatwierdzeniu ustawień pojawi się matryca pamięci, którą należy wypełnić danymi:



Widać w niej adresy komórek w górnym wierszu i lewej kolumnie, natomiast w puste pola danych matrycy wpisuje się zawartość pamięci. Klikając prawym klawiszem myszki z prawej strony matrycy na wysokości pierwszego wiersza, otwieramy menu, które pozwala dostosować format wyświetlanych adresów i danych do potrzeb konkretnego przypadku. W opisanym przykładzie ustawiono zarówno dla adresu jak i danych format Binary – co pozwala wprowadzać dane w sposób binarny. Zgodnie z realizowaną funkcją wartości adresu do komórek wpisano odpowiednio:

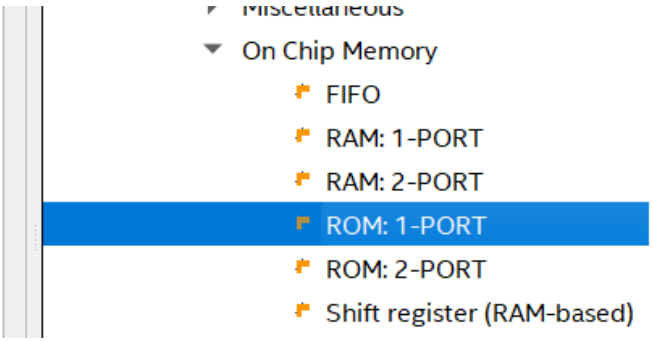
- pod adresem 0 (0000 binarnie) wartość 0 (1111111 1000000),
- pod adresem 1 (0001 binarnie) wartość 1 (1111111 1111001).

Po wprowadzeniu wszystkich wartości plik należy zapisać nie zmieniając folderu/katalogu – oraz zapamiętać jego nazwę, co będzie potrzebne w dalszej części.

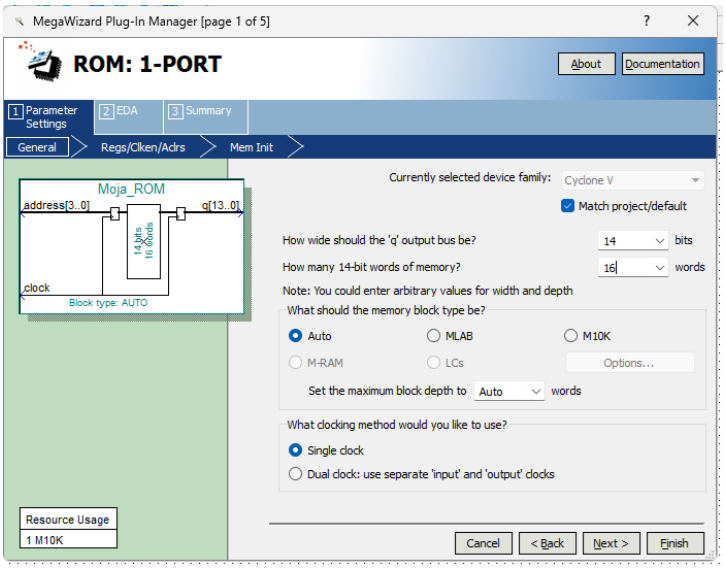


Po zainicjowaniu pliku MIF, można przejść do drugiego etapu, czyli konfigurowania makrofunkcji IP pamięci. Stąd po kolei:

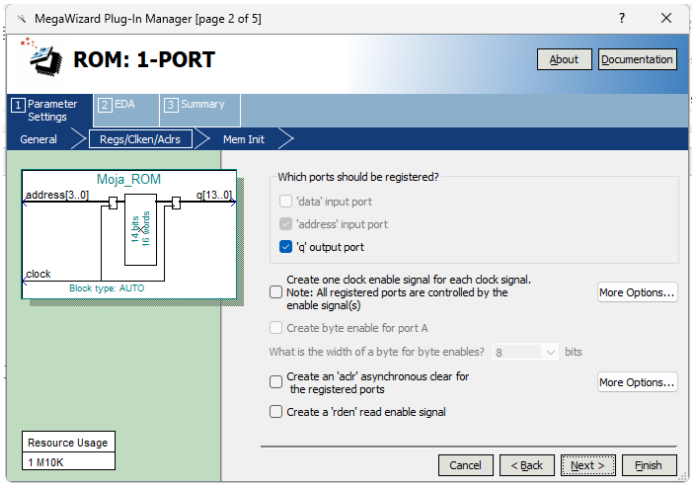
- z katalogu IP wybieramy makrofunkcję ROM: 1-PORT:



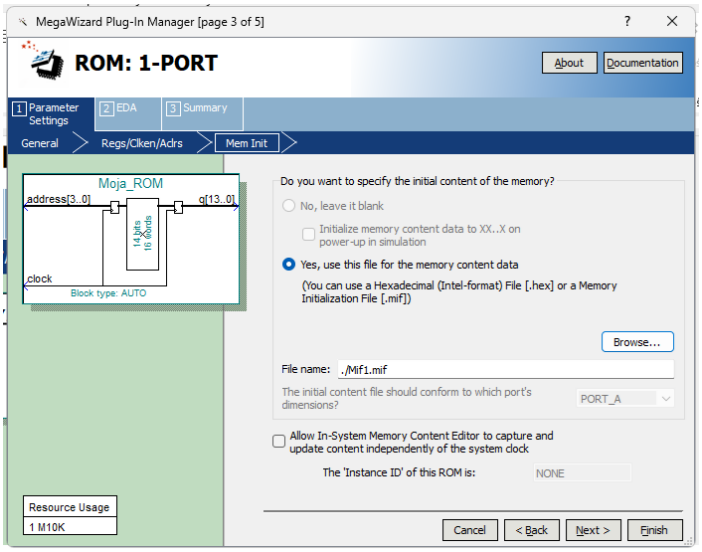
- następnie w poznany wcześniej sposób nadajemy nazwę symbolowi makrofunkcji:



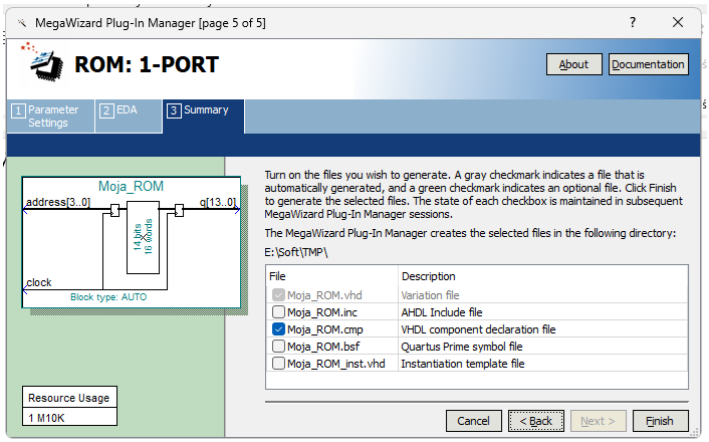
- w kolejnym oknie nie zmieniamy sugerowanych ustawień:



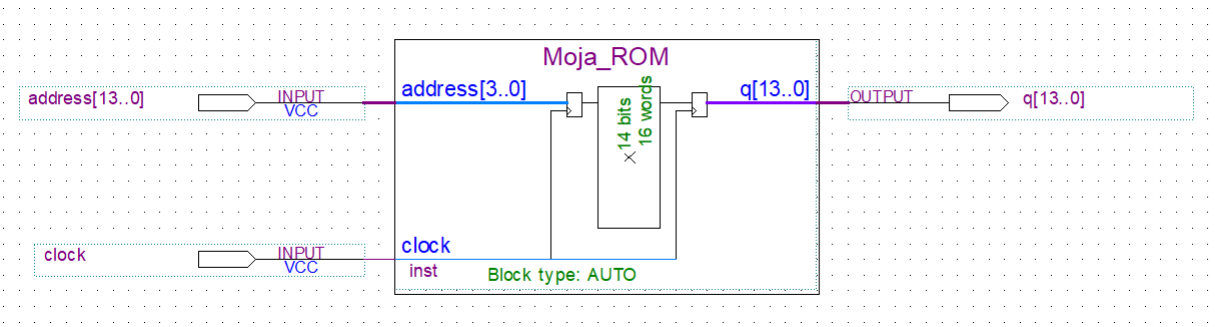
- ważne – w kolejnym oknie wpisujemy/wskazujemy utworzony wcześniej plik .mif



- w oknie nr 4 akceptujemy ustawienia domyślne,
- w oknie nr 5 **zaznaczamy opcję Quartus Prime symbol file** i zatwierdzamy Finish.



Następnie, w poznany wcześniej sposób pobieramy z katalogu Project zdefiniowany symbol modułu pamięci i uzupełniamy schemat w pliku top-level do widoku:

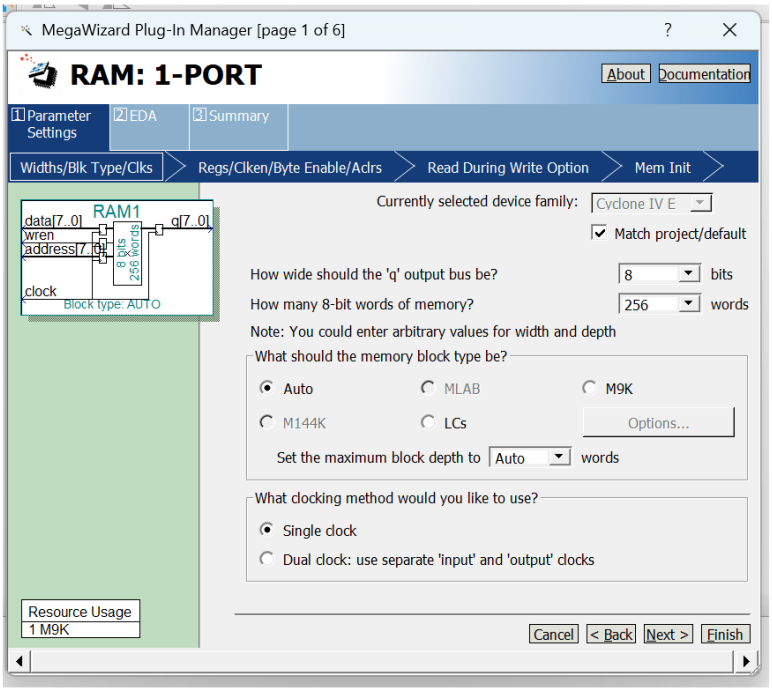


Proszę do końca uzupełnić plik MIF, zaimplementować i przetestować układ pamięci łącząc jej wyjścia z wejściami dwóch dowolny wyświetlaczy 7-segmentowych. Na wejście sygnału zegarowego proszę podać sygnał prostokątny 50 MHz z generatora płyty DE2-115.

Zadanie laboratoryjne 5.2

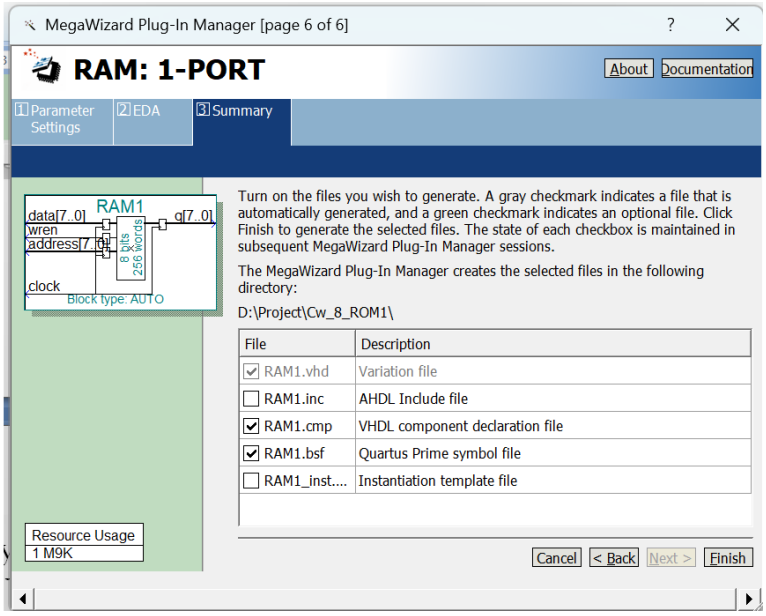
W FPGA możliwa jest synteza różnych konfiguracji RAM wewnętrznych: jednoportowych, dwuportowych a nawet 4-portowych, co skraca czas dostępu do danych matrycy pamięci. Jednoportowa RAM, jest klasycznym, historycznie najstarszą koncepcją pamięci o dostępie swobodnym. Jej implementacja przebiega w następujących krokach:

- z katalogu IP Library → Basic Functions → On Chip Memory wybieramy **RAM 1-PORT**,
- po pojawieniu się okna konfiguratora ustalamy organizację pamięci:

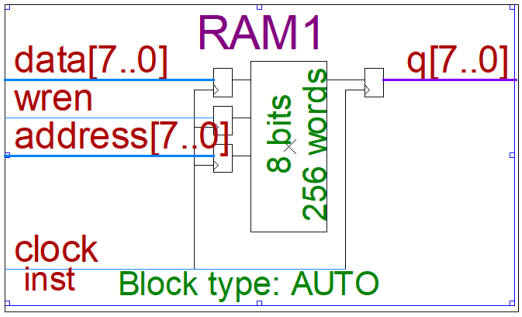


- w kolejnym oknie konfiguratora akceptujemy ustawienia domyślne,
- również pozostawiamy ustawienia domyślne w oknie nr 3,
- w oknie nr 4 rezygnujemy z wstępnej inicjacji zawartości – też akceptując sugerowane,
- podobnie nie zmieniamy ustawień symulacyjnych w oknie nr 5,

- w oknie podsumowującym zatwierdzamy konfigurację jak na rysunku:



Symbol skonfigurowanego modułu 1-portowej RAM zawiera np. następujące sygnały:
- magistralę danych wejściowych data[7..0],
- magistralę danych wyjściowych q[7..0],
- magistralę adresową address[7..0],
- sygnał zegarowy clock,
- sygnał sterujący zapisem i odczytem wren, w którym 1 oznacza zapis, 0 odczyt danych.



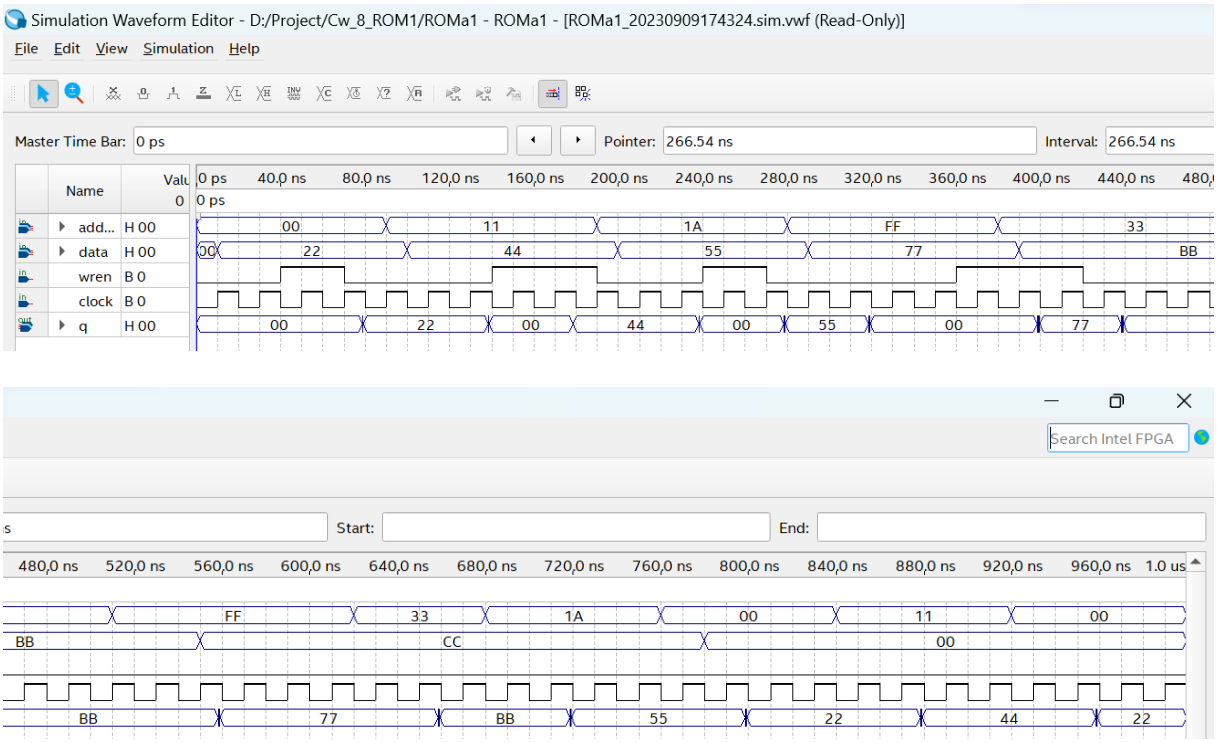
W czasie korzystania z RAM występują dwa charakterystyczne ciągi ustawień sygnałów związane z:
- zapisem danych do pamięci - tzw. cykl zapisu,
- oraz odczytem z pamięci, czyli cykl odczytu.

Obsługa każdego z powyższych cykli wiąże się z koniecznością zachowania odpowiedniej sekwencji/kolejności sygnałów wejściowych, stąd zrozumienie kolejności podawania tych sygnałów jest kluczowe dla poprawnej obsługi pamięci. Zarówno cykl zapisu jak i lub odczytu dotyczą tylko jednej, unikalnej i wskazanej przez adres komórki pamięci. Stąd np. chcąc zapełnić całą pamięć danymi, cykl zapisu należy powtórzyć dla każdego z jej adresów. Po zainicjowaniu/włączeniu pamięć RAM jest „pusta”. W jednym ciągu zapis danych nie musi dotyczyć wszystkich komórek, może obejmować jedną lub kilku wybranych. Podobnie odczyt

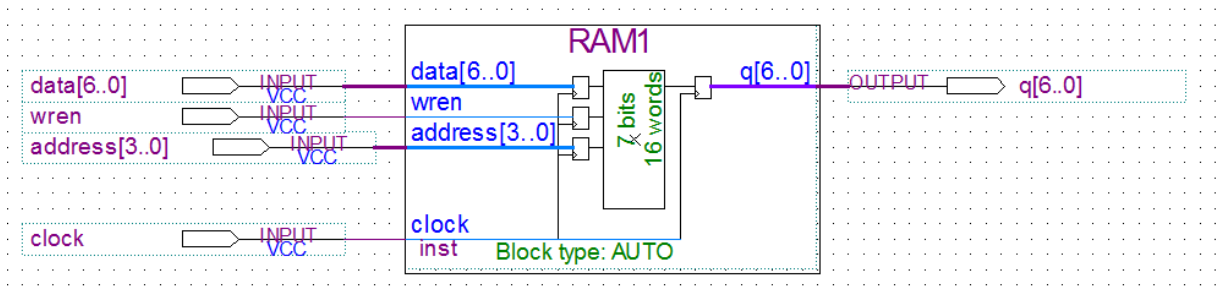
może dotyczyć sekwencji wybranych komórek. Ponadto, cykle zapisu i odczytu mogą przeplatać się wzajemnie.
Cykl odczytu z RAM jest bardzo podobny jak w przypadku poznanej wcześniej ROM. Polega on na podawaniu adresów odczytanych komórek oraz sygnału zegarowego. W tym przypadku, warunkiem odczytu jest ustawienie sygnału zezwolenia wren w stan niski, czyli wren = 0.

Zapis do RAM wymaga dłuższej i określonej sekwencji sygnałów wejściowych:
- na początku sygnał **wren** należy ustawić w tryb odczytu - wren = 0;
- następnie, na szynie adresowej **address** ustawiamy adres komórki, do której nastąpi zapis;
- dalej, na szynie danych **data** ustawiamy wartość zapisywaną do wybranej komórki;
- teraz przełączamy sygnał **wren** do trybu zapisu - wren = 1, na co najmniej 1 impuls clock,
- po upływie co najmniej 1 impulsu zegarowego, przełączamy **wren** w stan niski, wren = 0.

Po wykonaniu tych czynności, zapisana jest tylko jedna komórka pamięci. Żeby zapisać kolejne, cykl zapisu należy powtórzyć dla innych adresów. Najczęściej cykle obsługi pamięci wykonuje automatycznie mikroprocesor albo układ programowalny, z częstością kilkuset tysięcy razy na sekundę. Jednak dla ich lepszego zrozumienia proces przećwiczymy ręcznie. Przykładowe cykle odczytu/zapisu pokazuje poniższa symulacja:



Proszę zaimplementować i przetestować 1-portową pamięć RAM w poniższej konfiguracji:



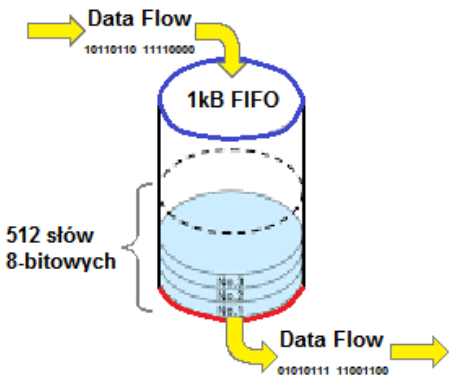
Przetestujemy ją w funkcji dekodera danych 4-bitowych, które będą adresami komórek w pamięci. Będzie to dekoder wskaźnika 7-segmentowego, który dal danych wejściowych w przedziale 0 ÷ 9 będzie wyświetlał cyfry dziesiętne, a powyżej 9 inne znaki alfanumeryczne – zgodnie z konfiguracją podaną w poniżej tabeli. Proszę wypełnić całą pamięć danymi z tabeli, a następnie sprawdzić pracę układu dokonując jej odczytu dla wszystkich wartości danych wejściowych/adresów. Komórki zapisane błędnie proszę poprawić, dokonując selektywnie ponownych zapisów.

Adres komórki pamięci:	Znak widoczny na wyświetlaczu:
0000	A
0001	B
0010	C
0011	E
0100	F
0101	H
0110	I
0111	L
1000	O
1001	P
1010	S
1011	U
1100	b
1101	d
1110	h
1111	u

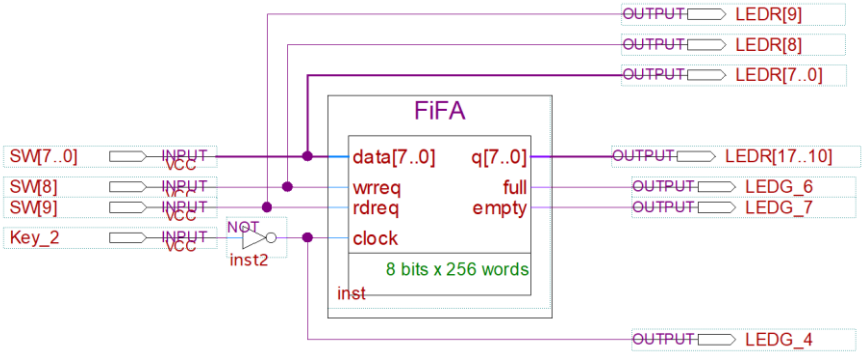
Porty pamięci proszę połączyć w sposób następujący:
- szynę danych połączyć z przełącznikami SW[6..0] i jednocześnie diodami LEDR[6..0],
- szynę adresową z przełącznikami SW[11..8] i LEDR[11..8],
- sygnał wren z przełącznikiem S[17] i diodą LEDR[17],
- sygnał clock proszę połączyć wewnętrznym generatorem 50 MHz czyli pinem Y2,
- wyjście q[6..0] proszę połączyć z dowolnym wyświetlaczem 7-segmentowym.

Zadanie laboratoryjne 5.3 – synteza FIFO

Pamięć FIFO jest rodzajem pamięci stosowanym do szybkiej transmisji dużych bloków danych. W takich zastosowaniach konieczne jest szybkie ich zapamiętanie, bez spowalniającego generowania kolejnych adresów i sygnałów sterujących. Po zapisaniu danych możliwy jest ich odczyt w dostępnym czasie i ze znacznie mniejszą prędkością/częstotliwością. Mechanizm obsługi FIFO poglądowo pokazuje rysunek (wykorzystano źródło: www2.advantech.com):



Pamięć ta nie posiada szyny adresowej, funkcjonuje ona na zasadzie złożonego rejestru przesuwnego. Na jej wejście, synchronicznie z zegarem, podawane jest całe słowo. Kolejne zapisywane słowa danych „spychają poprzednie w dół", aż do zapełnienia całej przestrzeni danych. Po zapełnieniu pamięci, pierwsze wczytane słowo jest możliwe do odczytu na wyjściu - stąd nazwa pamięci First-In First-Out. Po odczytaniu pierwszego słowa, zwalnia ono przestrzeń danych a pozostałe słowa w pamięci „przesuwają się w dół".
Struktura konfiguracyjna FIFO zawiera ona szereg sygnałów pomocniczych i ostrzegawczych a obsługa pamięci posiada pewne ograniczenia. Musi być zapewniona kontrola przed ryzykiem zapisu/nadpisania do pamięci całkowicie zapełnionej, co grozi utratą danych. Podobnie blokowana jest próba odczytu z pamięci pustej, co mogłoby generować dane nieprawdziwe. W ćwiczeniu przetestujemy uproszczoną konfigurację FIFO z tym samy sygnałem zegarowym dla wejścia i wyjścia. Poniżej przedstawiono implementację przykładowej FIFO, w której:
- data[7..0] – jest szyną danych wejściowych,
- q[7..0] – magistralą danych wyjściowych,
- clock – wejściem sygnału zegarowego,
- wrreq – żądaniem/umożliwieniem/zezwoleniem zapisu,
- rdreq – żądaniem/umożliwieniem/zezwoleniem odczytu,
- full – flagą informująca o całkowitym zapełnieniu pamięci,
- empty – flagą informująca o całkowicie pustej pamięci.



Proszę skonfigurować oraz przetestować FIFO o organizacji 8 słów 4-bitowych pokazaną na powyższym rysunku. W przypadku braku czasu w laboratorium, należy wykonać symulacje.

Literatura:

1. Salauyou V., Klimowicz A., Projektowanie systemów wbudowanych w układach FPGA, Białystok, Oficyna Wydawnicza Politechniki Białostockiej, 2022.

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 6

Wprowadzenie do Maszyny Stanów w VHDL - mechanizmy kontroli sterowania

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcję opracował:

dr inż. Marian Gilewski

Białystok 2025

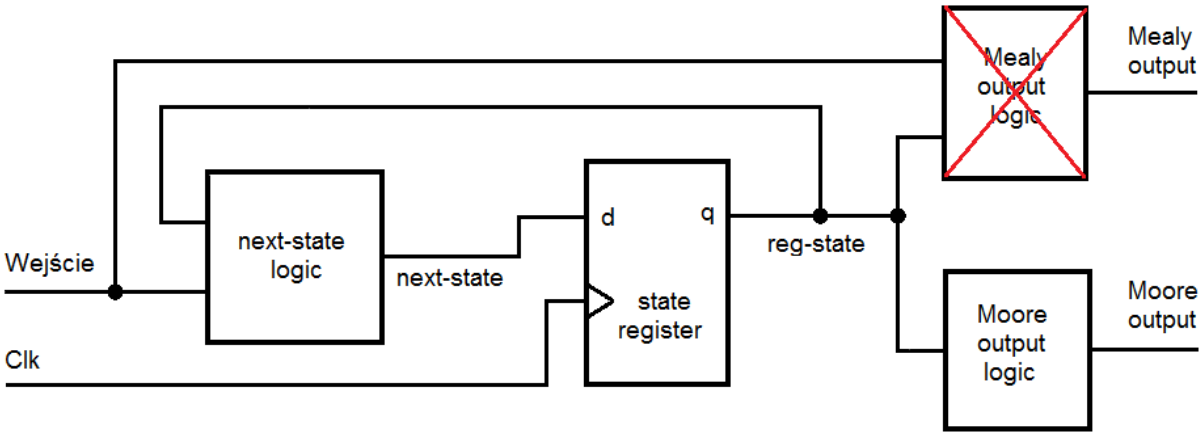
Cel ćwiczenia.

Maszyna Stanów lub Automat Stanów (Finite State Machine) jest jedną z najczęściej stosowanych technik opisu układów sekwencyjnych w VHDL. Stąd nabycie umiejętności jej implementacji w strukturach programowalnych, warunkuje efektywne zastosowanie układów CPLD i FPGA w różnych obszarach, w tym w układach sterowania. W zakresie ćwiczenia planowane jest poznanie techniki i specyfiki implementacji FSM w syntezie układów cyfrowych.

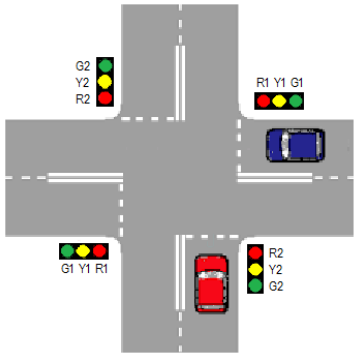
Wprowadzenie.

W teorii układów cyfrowych, układy FSM przedstawia się, jako złożone z wejść sygnałowych, wejścia zegarowego oraz wyjść, często dodawane jest też wejście zerujące. Zgodnie z tą koncepcją wewnętrzna struktura FSM podzielona jest na 3 części (https://climber.uml.edu.ni):

- next-state logic – części kombinacyjnej, która wyznacza stan następny;
- state register – rodzaju pamięci do przechowywania informacji o stanie bieżącym,
- output logic – część w wersji Moore'a / Mealy'ego do generacji sygnałów wyjściowych.

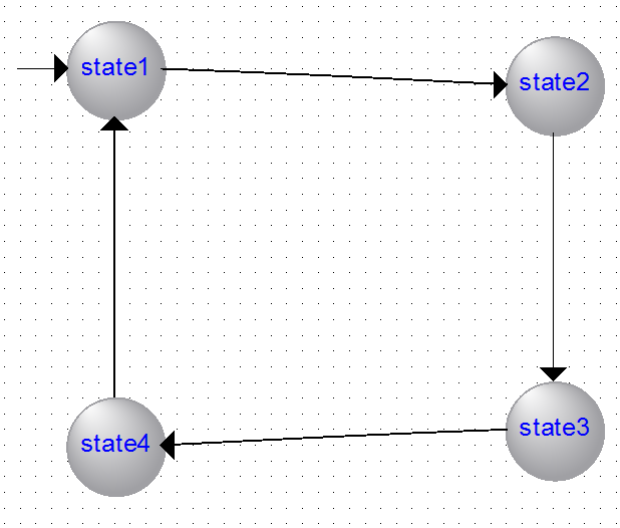


W czasie zajęć ograniczmy się do badania układów w konfiguracji Moore, jako prostszych w syntezie. Z implementacją FSM nierozerwalnie jest związane pojęcie stanu. Stan oznacza aktualne zachowanie układu, z którym związany jest określony zestaw wartości sygnałów wejściowych i wyjściowych. Rozważmy proste równorzędne skrzyżowanie uliczne, w którym zaimplementowano układ sterowania światłami, stąd mamy zestawy sygnałów wyjściowych w postaci 3 lamp RGB. Jest to bardzo uproszczony model stanu rzeczywistego, w celu zrozumienia mechanizmu syntezy FSM.



Zidentyfikujemy sytuacje/stany układu - patrząc w układzie pionowego kierunku jazdy mamy:
- w stanie state1 świecą zielone światła G2 - następnie układ przechodzi do stanu state2,
- w state2 świecą światła żółte Y2 a następnie układ przechodzi do state3,
- w state3 świecą światła czerwone R2 a układ przechodzi do state4,
- w stanie state4 świecą światła czerwone R2 i żółte Y2 a układ wraca do state1.

Opisaną sytuację można przedstawić w postaci grafu w przejść, w którym węzłami są nazwy stanów a gałęzie reprezentują przejścia między stanami, którym towarzyszą określone wartości sygnałów wejściowych. W omawianym układzie nie występują inne sygnały wejściowe niż sygnał zegarowy, stąd tylko zegar – a właściwie kolejne jego impulsy będą decydować o zmianie stanu.



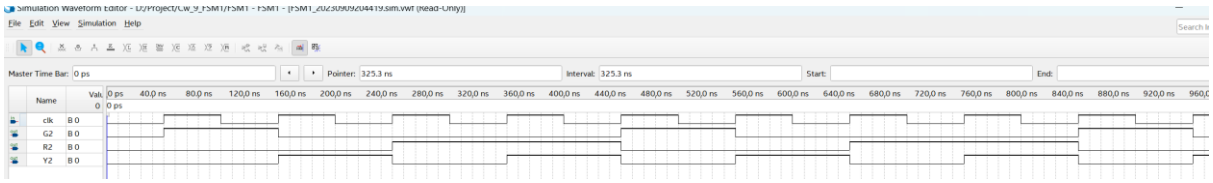
Opiszemy powyższy układ w sposób uproszczony, niż formalny opis FSM:

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity FSM1 is
5     port( clk       : in  std_logic;
6           R2,Y2,G2  : out std_logic);
7  end entity;
8
9  architecture rtl of FSM1 is
10    type state_type is (state1, state2, state3, state4); -- definiujemy nowy typ danych - lista stanów
11    signal state   : state_type;                         -- oraz sygnał pomocniczy nowego typu
12 begin
13    process (clk)
14       begin
15          if rising_edge(clk) then
16             case state is
17                when state1 =>
18                   G2 <= '1'; Y2 <= '0'; R2 <= '0'; -- świeci lampa G2
19                   state <= state2;                 -- przejście do następnego stanu
20                when state2 =>
21                   G2 <= '0'; Y2 <= '1'; R2 <= '0';
22                   state <= state3;
23                when state3 =>
24                   G2 <= '0'; Y2 <= '0'; R2 <= '1';
25                   state <= state4;
26                when state4 =>
27                   G2 <= '0'; Y2 <= '1'; R2 <= '1';
28                   state <= state1;
29             end case;
30          end if;
31       end process;
32 end rtl;
33
```

w którym:

- zdefiniowano nowy wyliczeniowy typ danych, będący faktycznie listą nazw stanów;
- stany w układzie opisuje instrukcja case – wartości wyjść i nazwę stanu następnego,
- układ rozpoczyna pracę od pierwszego stanu na liście,
- przejście do następnego stanu następuje na narastającym zboczu kolejnego impulsu clk.

Powyższy układ można zweryfikować symulacyjnie – uzyskując wyniki:

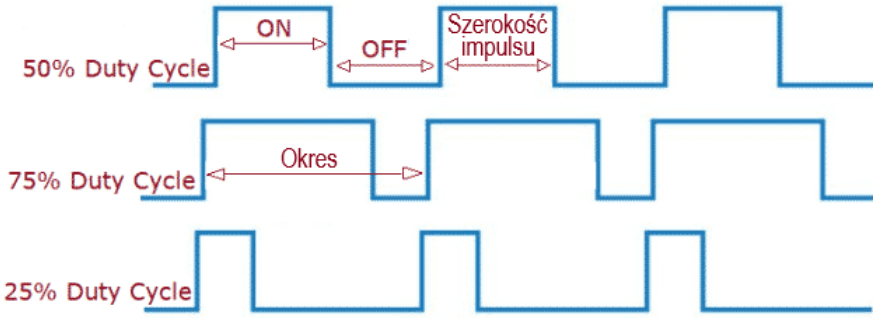


Zadanie laboratoryjne 6.1

Proszę uzupełnić powyższy kod źródłowy, dodając wyjścia pozostałych lamp (G1, R1, Y1) oraz negację sygnału zegarowego z klucza Key[2]. Wyjścia proszę połączyć z diodami LED, układ zaprogramować i przetestować.

Zadanie laboratoryjne 6.2 - cyfrowa synteza sygnału PWM.

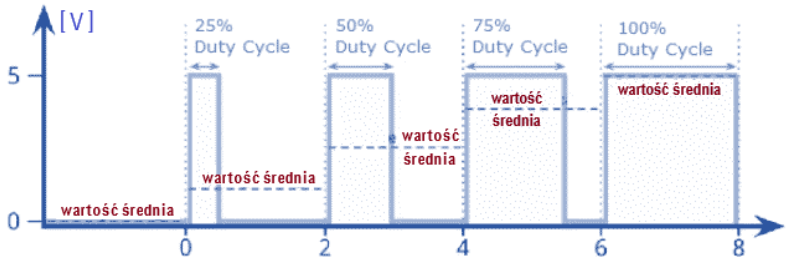
W wielu układach energoelektroniki, automatyki, robotyki czy elektroakustyki wykorzystuje się sygnał okresowy, w którym regulowany jest czas trwania impulsu przy niezmiennym okresie (czasie powtarzania). Taką zmienność czasu trwania impulsu nazywana jest modulacją szerokości impulsu (PWM - Pulse Width Modulation). Parametrem sygnału PWM jest



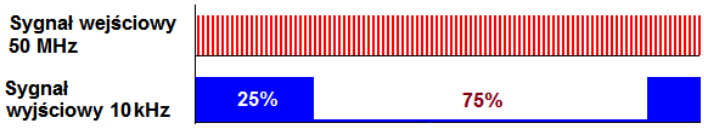
współczynnik wypełnienia impulsu (Duty Cycle) wyrażony w % . Zdefiniowany jest on jako stosunek czasu trwania impulsu **TON** w stanie wysokim do czasu powtarzania, czyli sumy czasu **TON** i czasu trwania stanu niskiego **TOFF**.

$$\text{Duty Cycle} = \frac{T_{ON}}{T_{ON} + T_{OFF}} * 100$$

Modulacja PWM wykorzystywana do sterowania tych urządzeń lub przekształtników różnych rodzajów energii, które reagują liniowo na wartość średnią energii impulsu zasilającego lub sterującego. Stąd ten rodzaj modulacji jest stosowany do regulacji subiektywnego wrażenia wartości strumienia świetlnego lamp LEDowych, prędkości obrotowej silnika prądu stałego, regulacji temperatury rezystancyjnej nagrzewnicy, itp. Tego typu odbiornikiem jest również

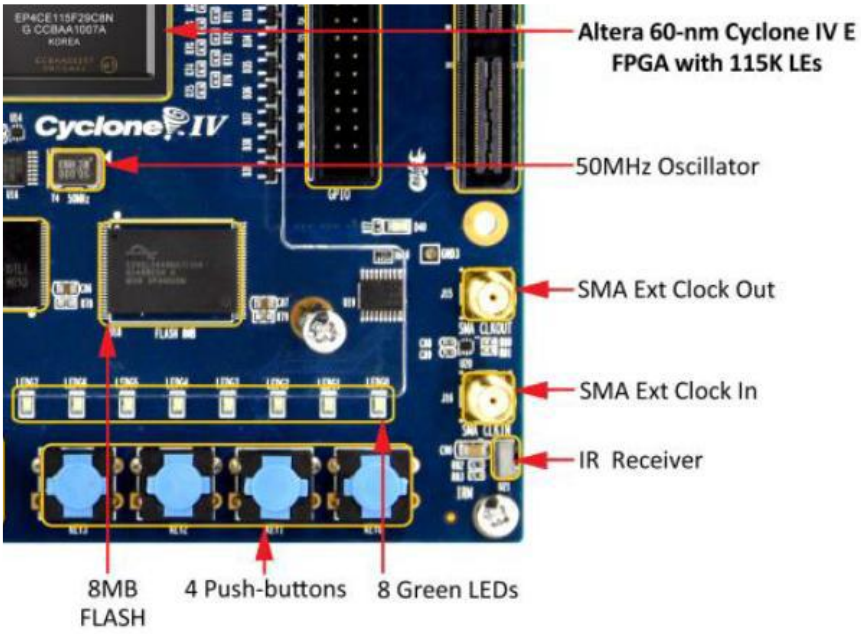


organ wzroku. Stąd modulację PWM stosuje się do regulacji subiektywnej luminacji źródeł światła, gdyż impulsy świetlne powyżej częstotliwości fuzji wskutek bezwładności oka postrzegane są jako stały strumień, którego wartość można regulować zmieniając współczynnik wypełnienia impulsów. Istnieje kilka metod wytwarzania sygnału PWM. W energoelektronice dominują metody analogowe. Natomiast technika oparta na syntezie częstotliwości, jest w pełni cyfrową metodą, łatwą w syntezie i bardziej stabilną oraz mniej wrażliwą na zakłócenia. Jej idea polega na wytwarzaniu sygnału wyjściowego w wyniku zliczania wzorcowych impulsów wysokiej częstotliwości w stanie wysokim i niskim na wyjściu. W tym zadaniu wytwarzany będzie sygnał wyjściowy o częstotliwości 10 kHz metodą zliczania impulsów z generatora wzorcowego 50 MHz. Kod opisujący tak działający układ jest oparty na technice FSM, gdzie zlicza się 25% impulsów wejściowych w stanie wysokim i pozostałe 75% w stanie niskim.




```
1   library ieee;
2   use ieee.std_logic_1164.all;
3
4   entity inst10 is
5      port(
6         clk       : in std_logic;
7         output    : out   std_logic
8      );
9   end entity;
10  architecture rtl of inst10 is
11     type state_type is (s0, s1);
12     signal state   : state_type;
13  begin
14
15     process (clk)
16     variable TT : integer := 0;
17     begin
18        if rising_edge(clk) then
19           case state is
20              when s0=>
21              output <= '1';
22              TT:= TT + 1;
23                 if TT = 1250 then
24                    TT := 0;
25                    state <= s1;
26                 end if;
27              when s1=>
28              output <= '0';
29              TT:= TT + 1;
30                 if TT = 3750 then
31                    TT := 0;
32                    state <= s0;
33                 end if;
34           end case;
35        end if;
36     end process;
37  end rtl;
38
```

Tym razem, ze względu na wyższą częstotliwość, sygnał wyjściowy będziemy obserwować oscyloskopem a nie diodą LED. W związku z tym, należy go wyprowadzić do gniazda SMA Ext Clock Out modułu DE2-115:

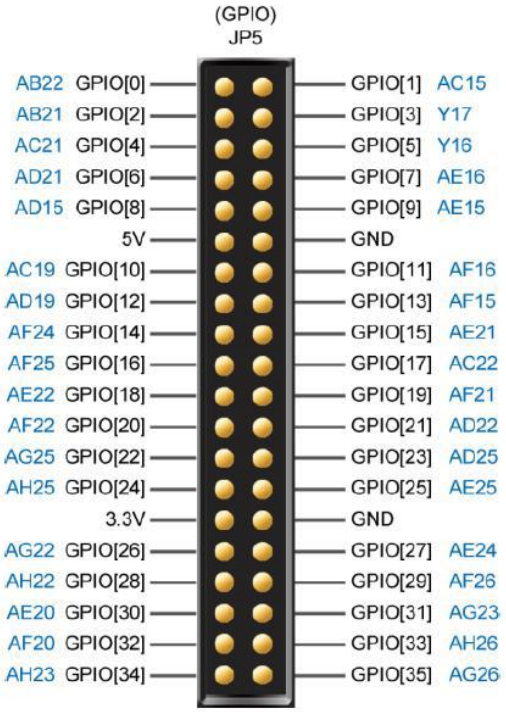


a następnie kablem bnc połączyć z wejściem analogowym A oscyloskopu. Instrukcja obsługi oscyloskopu, jest udostępnionym składnikiem dokumentacji laboratoryjnej. Przed przystąpieniem do zajęć proszę zapoznać się z zawartością instrukcji. Niezbędne informacje wprowadzające w obsługę oscyloskopu będą przekazane w laboratorium podczas wprowadzenia do ćwiczenia.

Zadanie laboratoryjne 6.3 – wielokanałowa PWM.
Proszę zaprojektować i przetestować w module DE2 używając oscyloskopu, układ generatora o przełączanej częstotliwości i zmiennym współczynniku wypełnienia sygnału.
Układ powinien:
- wykorzystać wewnętrzny generator 50 MHz modułu DE2,
- posiadać 4 wyjścia,
- na każdym z wyjść wytwarzać sygnał o częstotliwościach: 1 kHz lub 10 kHz lub 100 kHz lub 1 MHz – wybierany parą przełączników SW;

- współczynnik wypełnienia impulsu dla każdego wyjścia i każdej częstotliwości powinien przyjmować wartości: 20% lub 40% lub 60% lub 80%;
- nastawy współczynnika wypełnienia należy realizować odpowiednimi parami przełączników SW.

Cztery sygnały wyjściowe należy wyprowadzić na złącze GPIO modułu DE2-115, a następnie zarejestrować te sygnały używając cyfrowych sond oscyloskopu.



W złączu tym, **6 pin od góry i 6 pin od dołu po prawej** stronie są wyprowadzeniami masy (GND), które należy **połączyć z czarnymi skrajnymi i krótszymi przewodami** sondy cyfrowej. Proszę **unikać** połączeń masy i przewodów sygnałowych sondy z pinami zasilania po lewej stronie (5V i 3.3V), gdyż **grozi to uszkodzeniem modułu sond cyfrowych** oscyloskopu. Pozostałe piny złącza można wykorzystać dowolnie, jako wyjścia badanego układu, kojarząc ich połączenia z numerami pinów FPGA podano na powyższym rysunku.

Literatura:

1. Salauyou V., Klimowicz A., Projektowanie systemów wbudowanych w układach FPGA, Białystok, Oficyna Wydawnicza Politechniki Białostockiej, 2022.
2. Barkalov A., Titarenko L., Kolopienczyk M., Mielcarek K., Bazydlo G., Logic Synthesis for FPGA-Based Finite State Machines, Springer International Publishing, 2016.
3. MIT, Finite State Machines, https://web.mit.edu/6.111/www/f2017/handouts/L06.pdf, 2017, dostępność 7.09.2025.

Politechnika Białostocka
Wydział Elektryczny
Katedra Automatyki i Robotyki

ĆWICZENIE Nr 7

Układy uzależnień czasowych

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcję opracował:

dr inż. Marian Gilewski

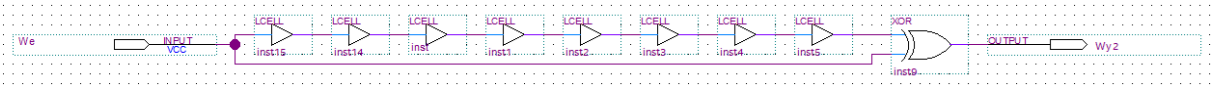
Białystok 2025

1. Cel ćwiczenia.

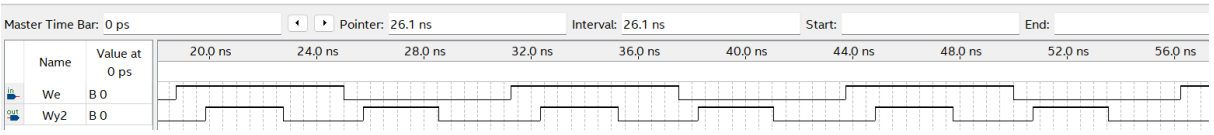
Celem ćwiczenia jest nabycie umiejętności w zakresie syntezy częstotliwości, w tym kontroli podstawowych parametrów sygnału zegarowego takich jak: wartość częstotliwości oraz kontrola opóźnień. Oprócz technik programowych i strukturalnych, w drugiej części studenci poznają sposób implementacji dedykowanego wbudowanego modułu pętli fazowej.

Zadanie laboratoryjne 7.1 – sprzętowa kontrola opóźnień oraz powielanie częstotliwości.

Na poniższym schemacie pokazano układ, w którym wykorzystano mechanizm różniczkowania impulsów cyfrowych – wykorzystany w całym układzie do powielania częstotliwości sygnału wejściowego. Sygnał wejściowy dociera do wejścia bramki XOR dwiema drogami: bezpośrednim połączeniem oraz przez linię opóźniającą złożoną z elementów LCELL (Logic Cell). Użycie elementu LCELL nie zmienia poziomu przenoszonego sygnału



a jedynie wymusza przejście sygnału przez dodatkową makrokomórkę. Czas propagacji sygnału przez pojedyńczą makrokomórkę FPGA jest rzędu kilku ns i zależy on od technologii. Stąd suma modulo 2 wykonana na sygnale bezpośrednim i opóźnionym powoduje wygenerowanie na każdym zboczu przetwarzanego sygnału krótkiego impulsu („szpilki"). Jeżeli w szeregowym łańcuch znajduje się więcej niż jedna makrokomórka, czas opóźnienia wydłuża się odpowiednio. W przypadku odpowiedniego dopasowania liczby makrokomórek do częstotliwości sygnału wejściowego - można uzyskać efekt podwojenia/powielenia częstotliwości sygnału wejściowego. Oznacza to, że na każdy okresowy impuls wejściowy na wyjściu pojawiają się dwa impulsy, co oznacza dwukrotnie wyższą częstotliwość niż wejściowa. Pokazują to wyniki symulacji układu na poniższym rysunku:



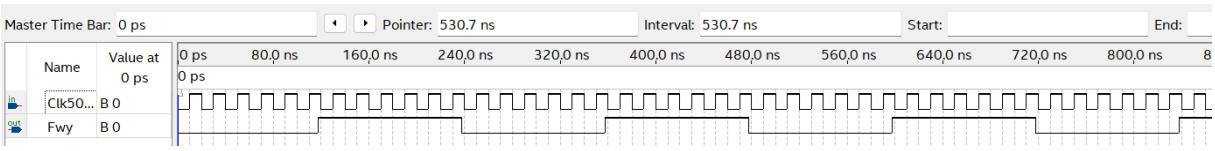
Proszę zmodyfikować i przetestować symulacyjnie układ, w którym efekt powielania częstotliwości uzyskano dla częstotliwości sygnału wejściowego około 50 MHz. Proszę wyjaśnić różnice wyników symulacji – w przypadku symulacji funkcjonalnej i czasowej.

Zadanie laboratoryjne 7.2 – programowany podział częstotliwości.

Poznany w poprzednim ćwiczeniu sposób FSM dzielenia częstotliwości nie jest optymalny ze względu na długość kodu. Poniżej pokazano prostszy sposób realizacji dzielników częstotliwości. Przedstawiony kod dzieli częstotliwość sygnału wejściowego przez 10, stąd w przypadku sygnału wejściowego o częstotliwości 50 MHz, częstotliwość sygnału wyjściowego wyniesie 5 MHz. Opisany sposób można stosować do wytwarzania sygnału zegarowego w zakresie niskich częstotliwości, z użyciem stabilnego źródła kwarcowego - gdzie nie można zastosować technik podziału z użyciem pętli synchronizacji fazowej.

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4   entity cw7 is
5      port(
6         Clk50MHz      : in std_logic;
7         Fwy           : buffer std_logic);
8   end entity;
9   architecture rtl of cw7 is
10     signal tmp      : integer := 0;
11  begin
12     process (Clk50MHz)
13        begin
14           if rising_edge(Clk50MHz) then
15              tmp <= tmp + 1;
16              if tmp = 5 then
17                 tmp <= 0;
18                 Fwy <= not Fwy;
19              end if;
20           end if;
21        end process;
22  end rtl;
23
```

Układ opisany powyższym kodem wytwarza symetryczny sygnał wyjściowy, co pokazują wyniki symulacji przedstawione na kolejnym rysunku.

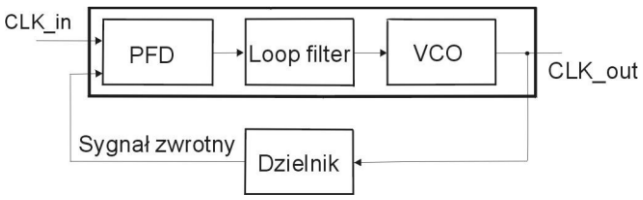


Stosując wyżej opisaną technikę, proszę zaprojektować i przetestować (mierząc oscyloskopem sygnał wyjściowy) układ wytwarzający z sygnału o częstotliwości 50 MHz – sygnał wyjściowy 20 kHz.

Zadanie laboratoryjne 7.3 – zastosowanie PLL

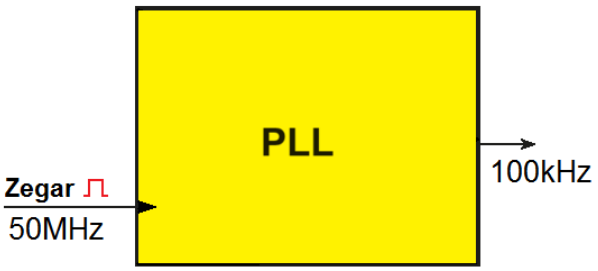
Do obsługi sygnałów zegarowych w układach FPGA przeznaczone są sprzętowe pętle fazowe (PLL). Funkcją PLL jest:

- mnożenie częstotliwości wejściowego sygnału zegarowego,
- dzielenie częstotliwości wejściowego sygnału zegarowego,
- realizacja przesunięcia fazowego w sygnale zegarowym,
- nastawa wypełnienia sygnału zegarowego.
- regeneracja sygnału zegarowego, minimalizację opóźnień, regulację czasów propagacji.

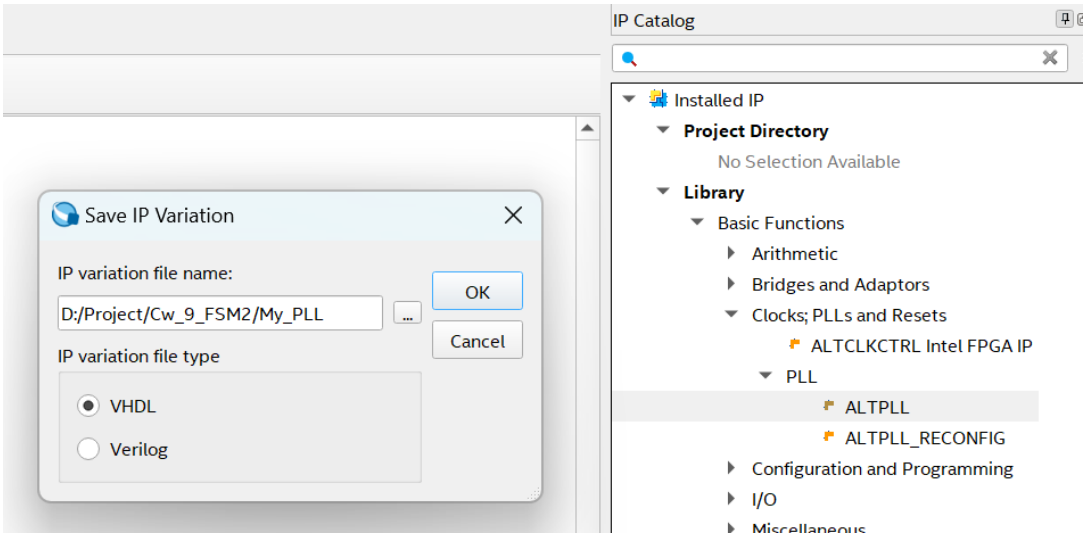


Pętla PLL zawiera: detektor fazy, filtr, generator przestrajany napięciem i dzielnik częstotliwości. Sprzężenie zwrotne dopasowuje fazy i częstotliwości sygnału wyjściowego i wejściowego. Występujący w sprzężeniu zwrotnym i/lub na wyjściu dzielnik/układ powoduje, że częstotliwość wyjściowa jest wyższa od wejściowej. Umieszczając dodatkowy dzielnik na wyjściu oraz moduły opóźniające w pętli można regulować częstotliwość, fazę i opóźnienie sygnału wejściowego.

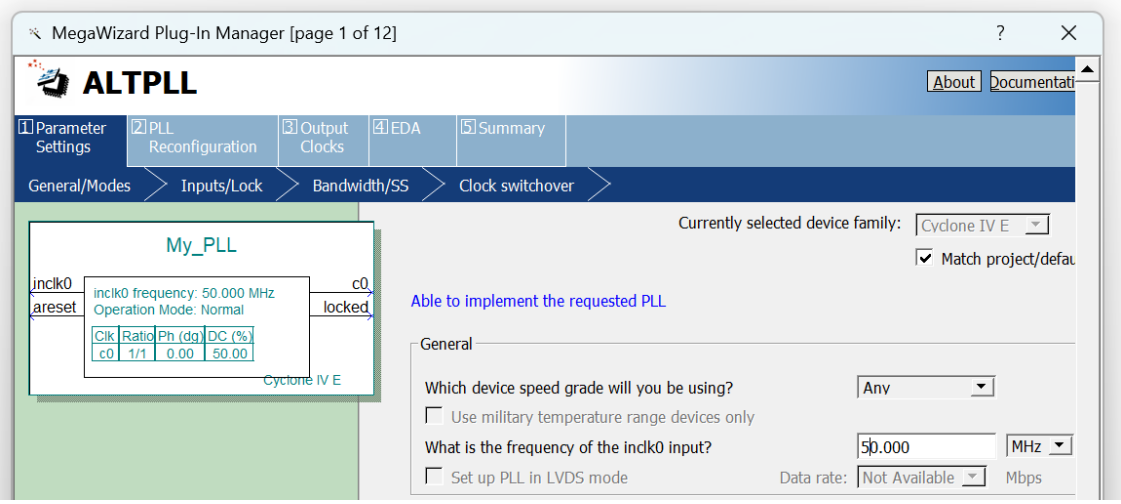
Korzystając z makrofunkcji ALTPLL proszę zbudować układ wytwarzający sygnał o częstotliwości 100 kHz, poglądowo pokazany na poniższym rysunku:



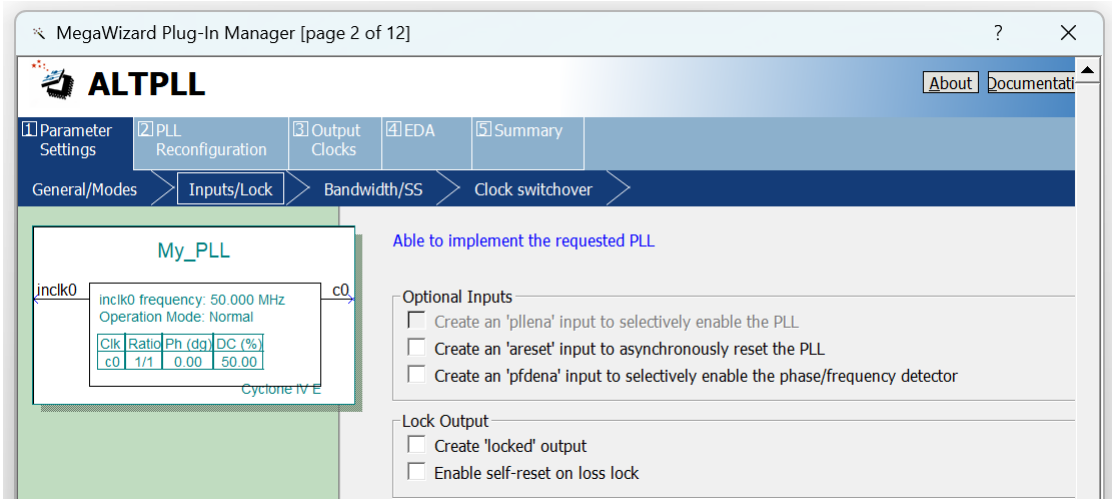
W tym celu proszę skonfigurować biblioteczną makrofunkcję ALTPLL w sposób następujący, na początku w znany sposób należy zainicjować nazwę nowego modułu:



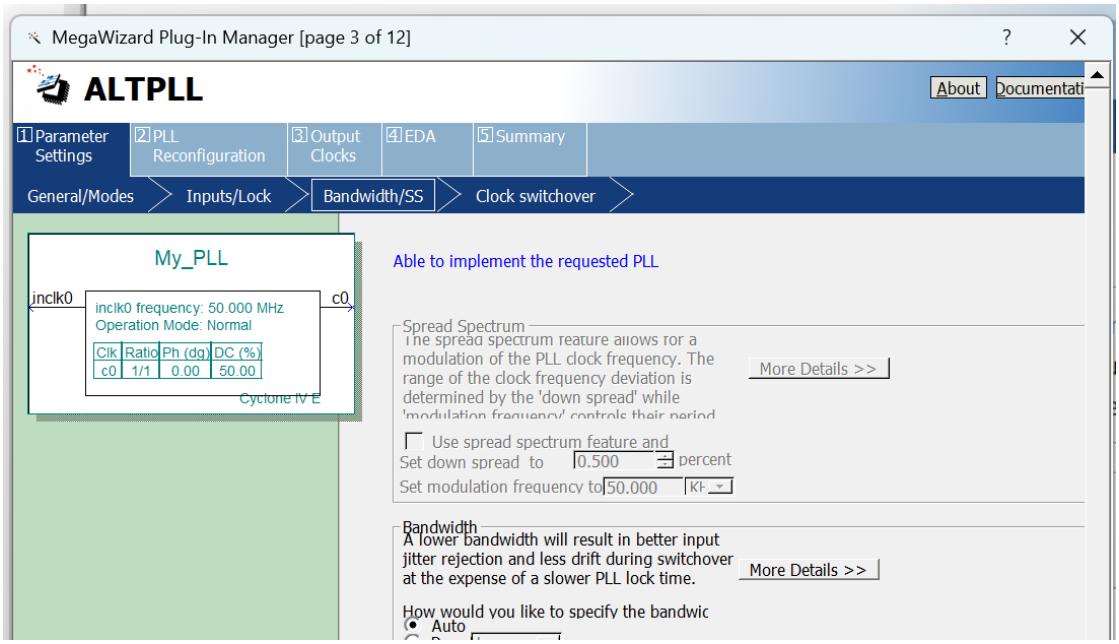
- następnie ustawiamy częstotliwość sygnału wejściowego na 50 MHz, czyli taką, jaka jest dostępna w module prototypowym DE2-115;



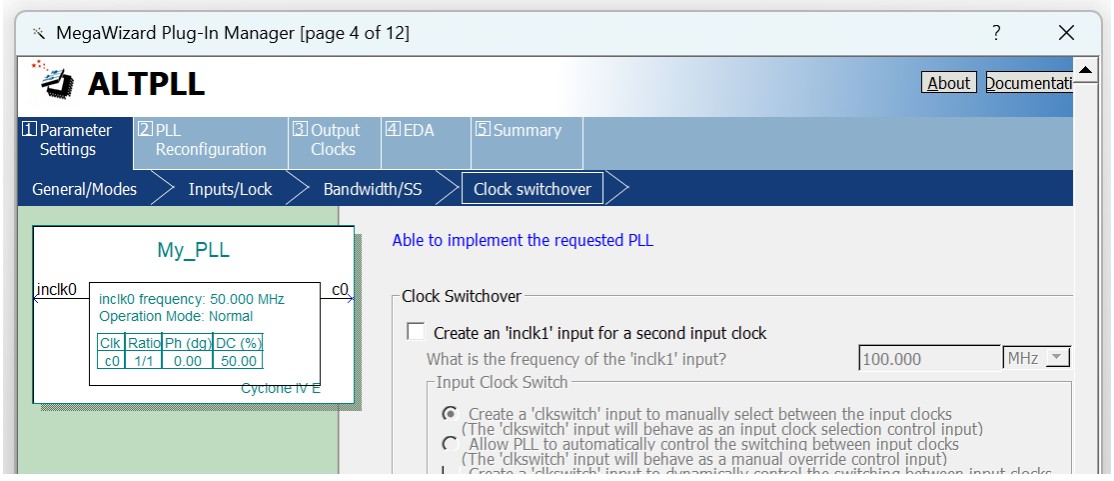
- w kolejnym kroku wyłączamy nieużywane sygnały pomocnicze;



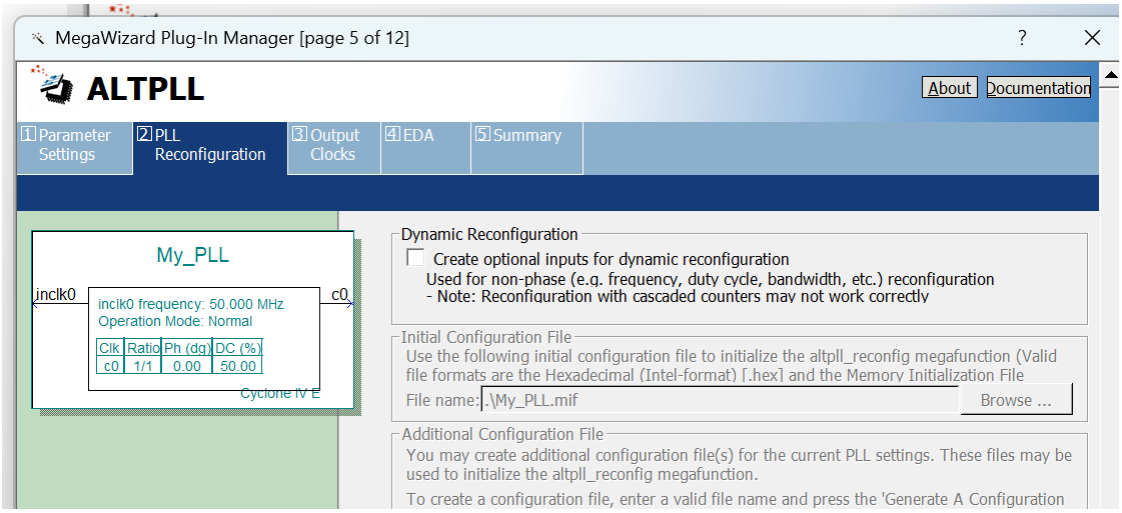
- następnie nie zmieniamy domyślnego pasma częstotliwościowego PLL;



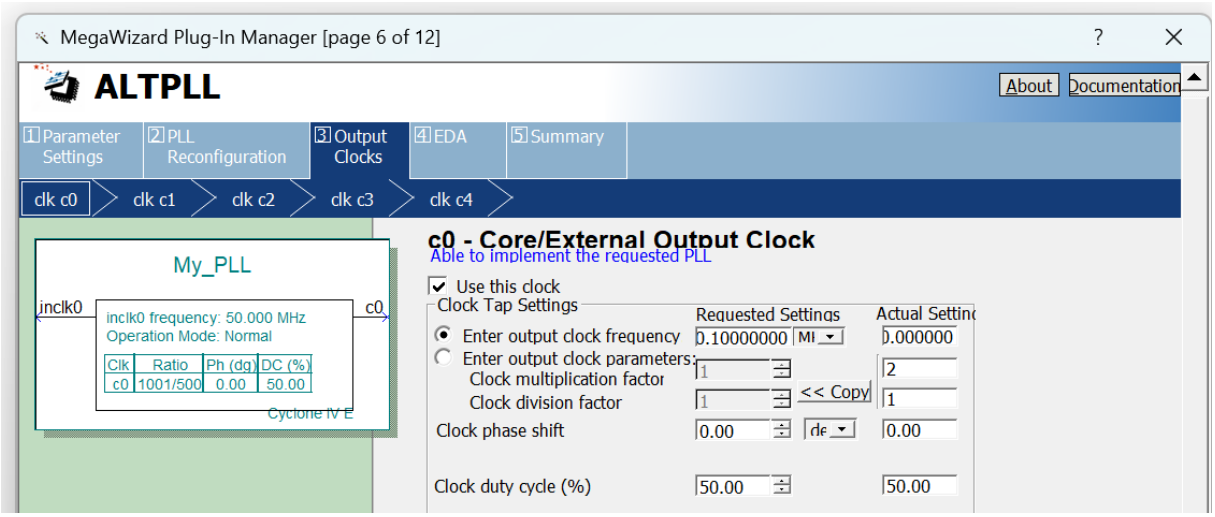
- w kolejnym oknie nie włączamy alternatywnego wejścia sygnału zegarowego,



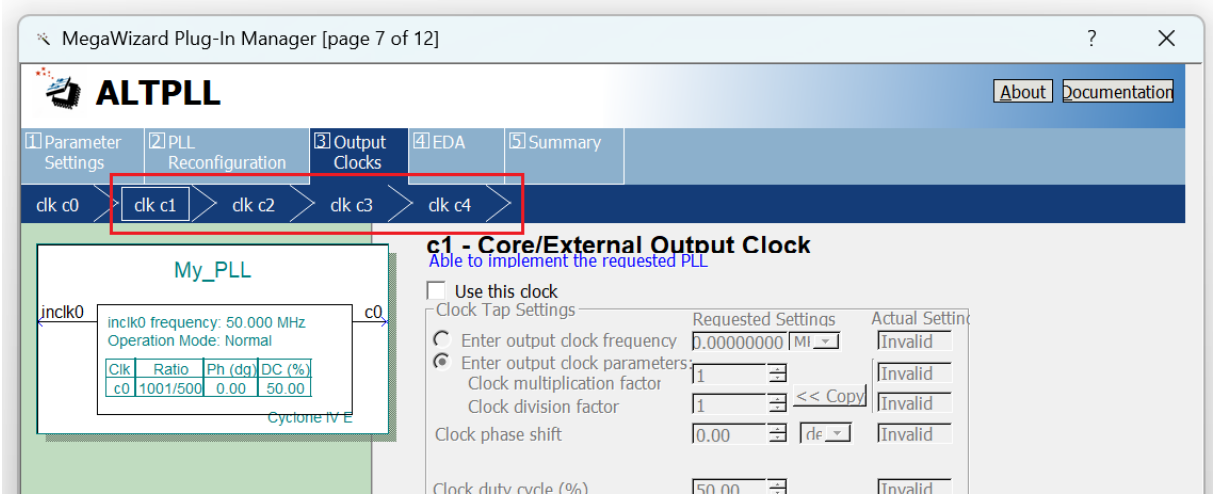
- nie ustawiamy również mechanizmu dynamicznej rekonfiguracji PLL;



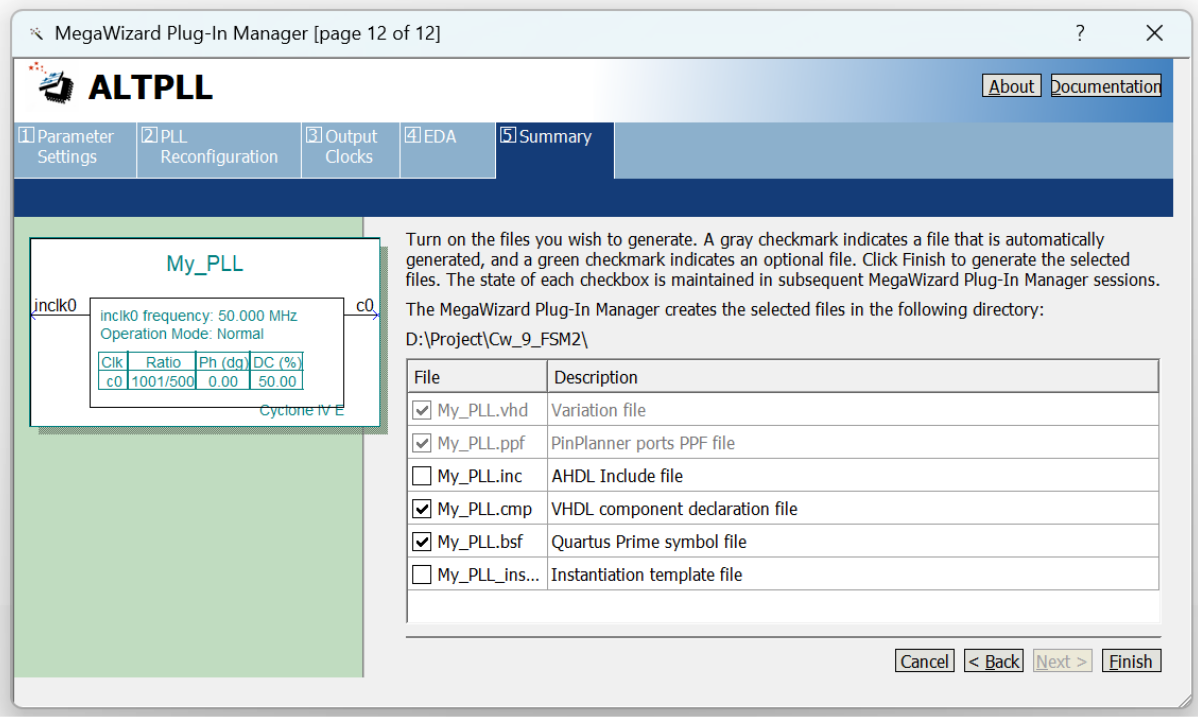
- w kolejnym oknie ustawiamy parametry syntezy częstotliwości, ustawiając częstotliwość wyjściową na np. 100 kHz (0,1 MHz) i nie zmieniając fazy ani współczynnika wypełnienia;



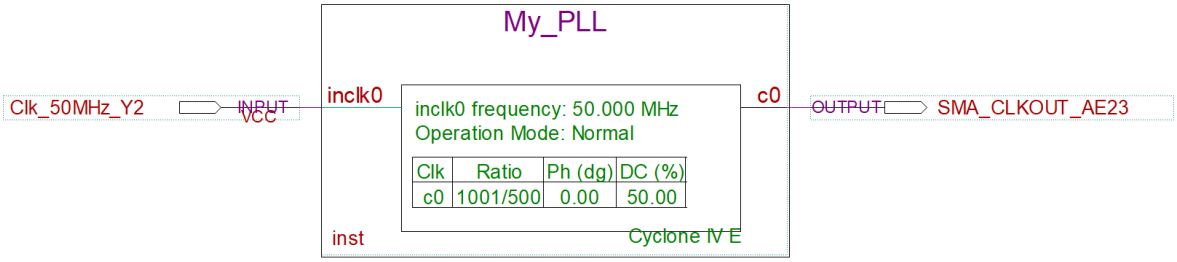
- pomijamy konfigurację wyjść clk c1 ÷ clk c4 – przechodząc do ostatniego okna ustawień;



- w którym, zaznaczamy klucz Quartus Prime symbol file.



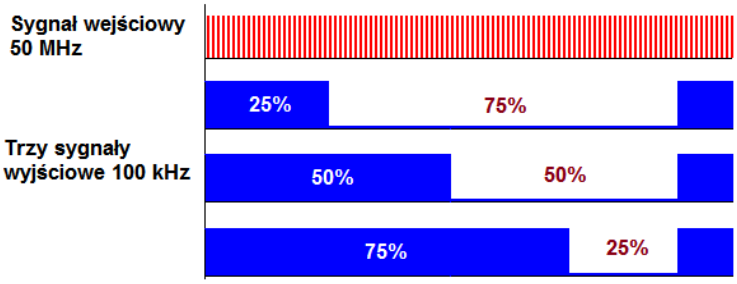
Po skonfigurowaniu modułu, pobierają symbol PLL z folderu Project, kompletujemy plik top-level, w sposób pokazany na poniższym rysunku:



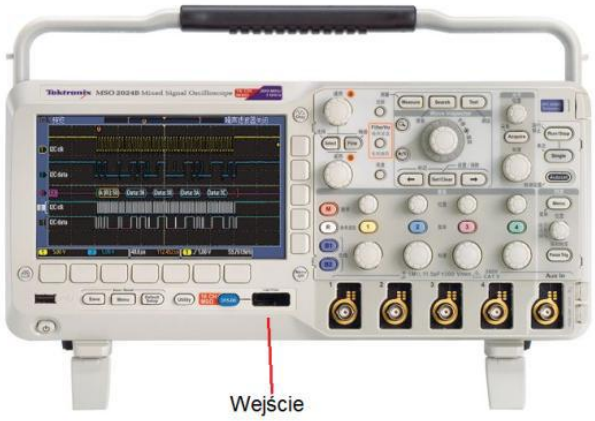
Układ kompilujemy, uruchamiamy i testujemy z użyciem oscyloskopu podobnie jak w zadaniu poprzednim.

Zadanie laboratoryjne 7.4 – wielowyjściowa PLL.
Stosując moduł PLL proszę zaprojektować układ o 1 wejściu i 3 wyjściach, który z sygnału zegarowego 50 MHz wytworzy 3 sygnały wyjściowe 100 kHz lub 1 MHz:

Każdy z sygnałów wyjściowych powinien posiadać tę samą częstotliwość, ale inną wartość współczynnika wypełnienia impulsu - np. 25%, 50% i 75%:



Układ należy skompilować i przetestować z użyciem oscyloskopu podobnie jak w zadaniu poprzednim – mierząc wszystkie sygnały wyjściowe jednocześnie.
W związku z tym, że sygnałów wyjściowych jest 3 a wyjściowych złączy SMA tylko jedno, do pomiarów wykorzystamy wejścia cyfrowe oscyloskopu:



oraz 16-kanałową sondę cyfrową oscyloskopu. W sondzie cyfrowej tego oscyloskopu, skrajne czarne przewody służą do podłączenia masy układu, pozostałe przewody oznaczone barwnymi opaskami są połączeniami mierzonych sygnałów.

Literatura:

1. Salauyou V., Klimowicz A., Projektowanie systemów wbudowanych w układach FPGA, Białystok, Oficyna Wydawnicza Politechniki Białostockiej, 2022.
2. Encinas J., Phase Locked Loops, Springer, 2012.

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 8

Implementacja interfejsu SPI w FPGA

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcję opracował:

dr inż. Marian Gilewski

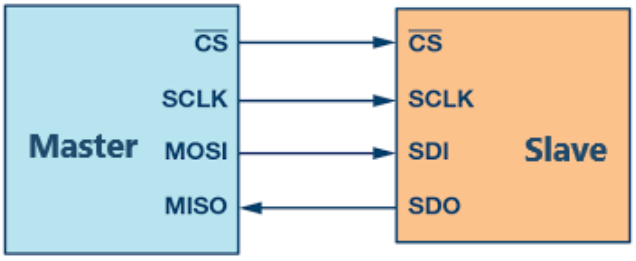
Białystok 2025

Cel ćwiczenia.

Celem ćwiczenia jest poznanie techniki syntezy w strukturach programowalnych popularnego szeregowego interfejsu sprzętowego typu SPI , stosowanych do wymiany danych pomiędzy układami scalonymi cyfrowymi, analogowo-cyfrowymi lub analogowymi z interfejsem cyfrowym. Ćwiczenie będzie obejmować implementację konkretnej wersji interfejsu, służącego obsługi scalonego potencjometru cyfrowego przez układ Master, będącym w tym przypadku układem FPGA. Studenci po zrealizowaniu tego ćwiczenia nabędą umiejętność syntezy dowolnego interfejsu tej rodziny oraz wiedzę, umożliwiającą implementację interfejsów szeregowych innego typu.

Wprowadzenie.

Interfejs SPI (Serial Peryfheral Interface) jest synchronicznym standardem komunikacyjnym, który został opracowany przez firmę Motorola w latach 80 ubiegłego wieku. Umożliwia ona szybką wymianę informacji pomiędzy dwoma urządzeniami w układzie Master-Slave, przy czym układ Master inicjuje transmisję wybierając układ/układy Slave oraz synchronizuje jej przebieg za pomocą sygnału zegarowego SCLK. Urządzenie Slave nie może manipulować sygnałem zegarowym, w czasie transmisji jeden bit danych przypada na okres sygnału zegarowego. Żadne dane nie są przesyłane, jeśli nie jest obecny sygnał zegarowy. Współczesne magistrale SPI umożliwiają transmisję z prędkością nawet 37.5 Mb/s. Interfejs SPI może być 3-przewodowy lub 4-przewodowy. Poniższy rysunek pokazuje układ 4-przewodowego interfejsu SPI. W układzie 4-przewodowym SPI jest protokołem dwukierunkowej wymiany



(źródło: https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html)

danych (full-duplex), gdy nowe dane są wysyłane, również nowe dane są wczytywane. Nie stosuje się potwierdzenia transmisji SPI – aczkolwiek w konfiguracji 4-przewodowej układ, który wysyła dane musi je ponownie odczytać przed ponowną próbą transmisji, to pozwala porównać dane wychodzące z przychodzącymi. Na tej podstawie układ wysyłający może określić poprawność transmisji a w przypadku błędów zdecydować o powtórzeniu transmisji lub jej przerwaniu. Każde z urządzeń (Master lub Slave) w układzie transmisyjnym może być zarówno nadajnikiem jak i odbiornikiem danych. Długość transmitowanych danych może być dowolna i nie musi zamykać się w jednym bajcie lub jego wielokrotności – stąd nieprawdziwe jest stwierdzenie wyznawców techniki mikroprocesorowej, iż istnieją uniwersalne insterfejsy. Znaczenie sygnałów magistrali SPI jest następujące:

- CS/SS (Chip Select lub Slave Select) - sygnał przechodząc w stan niski rozpoczyna transmisję, przejście tego sygnału w stan wysoki kończy lub zrywa transmisję;
- SCK (Serial Clock) - jest to sygnał zegara szeregowego generowany przez urządzenie Master, który kontroluje, kiedy dane są wysyłane i kiedy są odczytywane;
- SDO/MOSI/SO (Serial Data Output) - wyjście danych szeregowych, ten sygnał

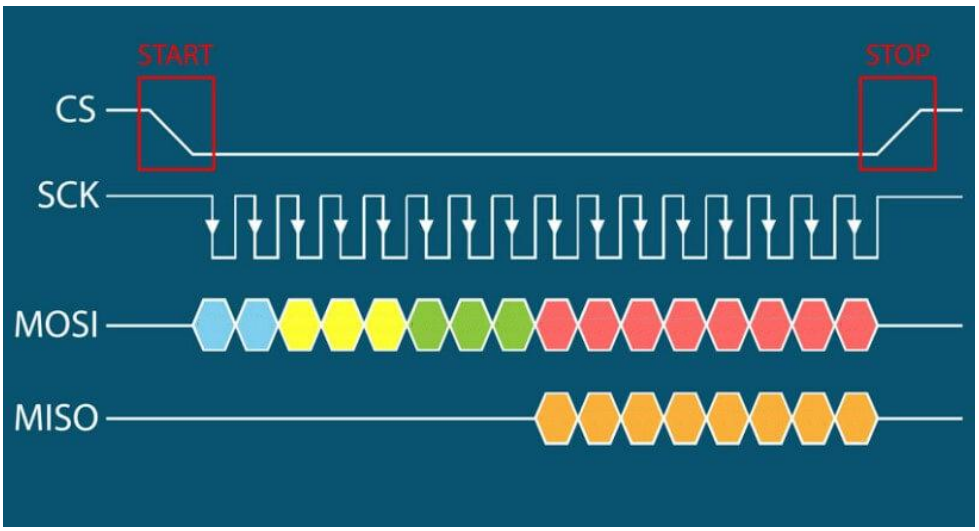
przekazuje dane wysyłane z urządzenia;

- SDI/MISO/SI (Serial Data Input) - wejście danych szeregowych, ten sygnał doprowadza dane przesyłane do urządzenia.

Konstrukcja interfejsu SPI jest często oparta na rejestrze przesuwnym, aczkolwiek w FPGA będziemy ją syntezować stosując technikę FSM.

SPI tworzy pętlę danych pomiędzy dwoma urządzeniami. Dane opuszczające urządzenie Master wychodzą na SDO, dane przychodzące do urządzenia Master wchodzą na linię danych szeregowych SDI. Zegar SCK generowany przez Master kontroluje, kiedy i jak szybko dane są wymieniane między dwoma urządzeniami. Sygnał CS/SS, pozwala Master kontrolować, kiedy wybierany jest konkretny Slave do transmisji.

Aby rozpocząć komunikację, czyli ustawić sygnał Start urządzenie Master musi wskazać urządzenie Slave i następnie wysłać sygnał zegarowy. Urządzenie Slave jest wskazywane poprzez włączenie sygnału CS. Zazwyczaj CS jest sygnałem aktywnym w stanie niskim; dlatego Master musi ustawić logiczne 0 na tym sygnale, aby wybrać konkretny układ Slave. Następnie w takt impulsów zegarowych wysyłane lub wczytywane są kolejne bity danych. Transmisję kończy sygnał Stop, czyli powrót sygnału CS do poziomu wysokiego.



(źródło: https://www.ariat-tech.pl/blog/what-is-spi-serial-peripheral-interface-explained.html)

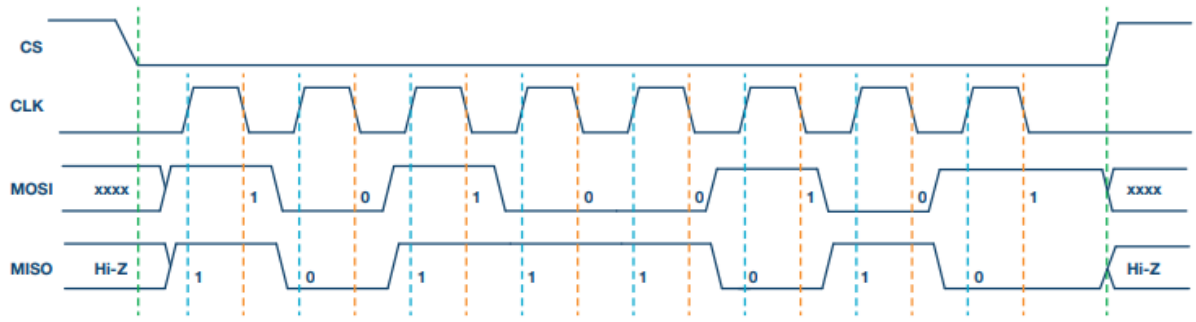
Zatrzaśnięcie bitu danych w SPI, może następować zarówno podczas narastającego jak i opadającego zbocza SCK, zależy to od konstrukcji konkretnego układu scalonego, zatem te informacje znajdują się w dokumentacji technicznej każdego urządzenia z magistralą SPI. Z tego powodu, w niektórych przypadkach, przed obsługą rozpoczęciem transmisji danych Master musi wysłać do rejestru sterującego Slave tzw. bity konfigurujące tryb pracy - czyli ustawić bit CPOL polaryzacji sygnału zegarowego, oraz ustawić bit CPHA fazy sygnału zegarowego. Poniższa tabela zawiera zestawienie możliwych 4 trybów pracy SPI, które określają, które zbocze sygnału SCK pojawia się pierwsze (CPOL, 0 - narastające, 1 – opadające), oraz kiedy pojawiają się dane na MOSI/MISO (CPHA, 0/1 – przed/po pierwszym zboczem/zboczu SCK).

Table 1. SPI Modes with CPOL and CPHA

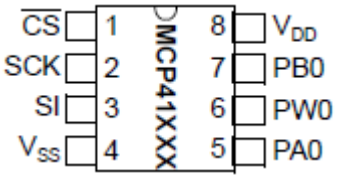
SPI Mode	CPOL	CPHA	Clock Polarity in Idle State	Clock Phase Used to Sample and/or Shift the Data
0	0	0	Logic low	Data sampled on rising edge and shifted out on the falling edge
1	0	1	Logic low	Data sampled on the falling edge and shifted out on the rising edge
2	1	0	Logic high	Data sampled on the falling edge and shifted out on the rising edge
3	1	1	Logic high	Data sampled on the rising edge and shifted out on the falling edge

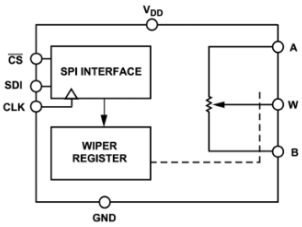
(źródło: https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html)

Poniższy rysunek (źródło j.w.) pokazuje tryb przypadek trybu 1 gdzie CPOL = 0, CPHA = 1, w stanie oczekiwania sygnał zegarowy przyjmuje poziom niski, pierwszym zboczem sygnału zegarowego jest zbocze narastające, dane wejściowe ustawiają po pierwszym zboczu SCK, dane są zapisywane/próbkowane na opadającym zboczu SCK (linia żółta), dane są przesuwane w czasie narastającego zbocza SCK (linia niebieska).

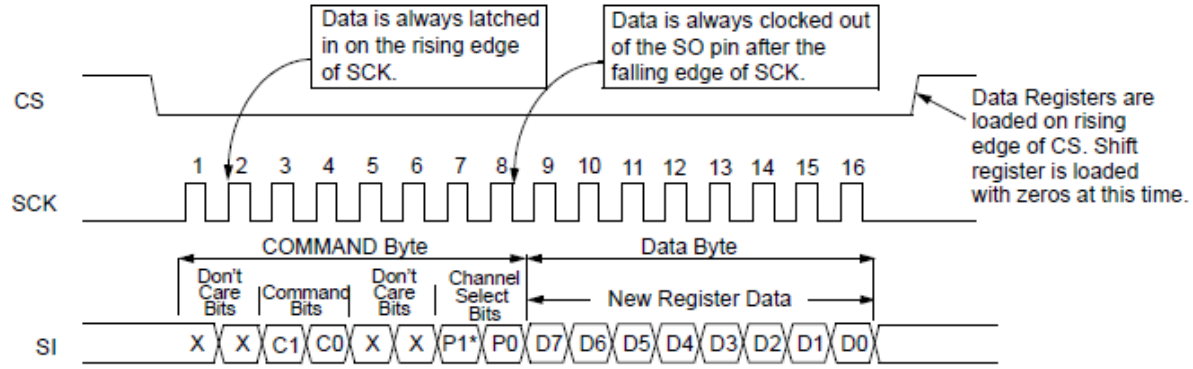


W czasie tego ćwiczenia będzie testowana 3-przewodowa wersja SPI, czyli bez pętli sprzężenia zwrotnego sygnału danych. Nie będzie też potrzeby modyfikacji bitów konfigurujących tryb pracy, gdyż relacja CPOL/CPHA będzie narzucona przez konstrukcję układu scalonego. W zależności od typu układu scalonego z interfejsem SPI, konfiguracja rejestru Slave może obejmować modyfikację innych parametrów, np. wybór kanału, sposób kodowania danych, itd. W systemie Quartus II zaprojektować moduł obsługi magistrali SPI potencjometru cyfrowego, moduł należy zaimplementować w płycie prototypowej DE2-115 łącząc ją z układem scalonym potencjometru zamontowanym na płytce stukowej. Przed przystąpieniem do prac przygotowawczych proszę zapoznać się z dokumentacją techniczną potencjometru. Układ potencjometru zawiera 3-przewodową magistralę SPI, obejmującą sygnały: **CS**, **SCK** i **SI**, nie zawiera on wyprowadzenia sygnału zwrotnego SO. Ponadto zawiera on dwa piny zasilające: $\mathbf{V_{DD}}$ – plus zasilania, $\mathbf{V_{SS}}$ – wspólna masa zasilania i sygnałowa. Część wykonawcza potencjometru zawiera wyprowadzenia: terminal PA0/A – sugerowane łączenie do wyższych potencjałów, terminal PB0/B – preferowane łączenie do niższych potencjałów lub masy, PW0/W (Wiper) – wyprowadzenie „suwaka” potencjometru.

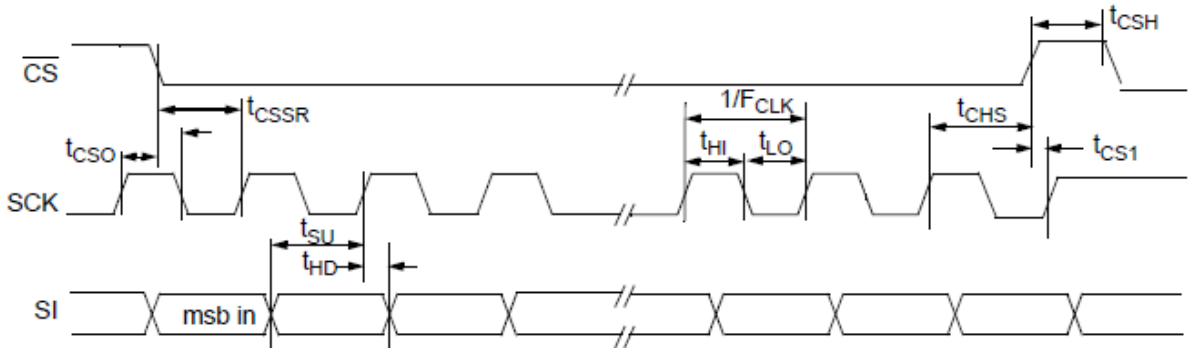




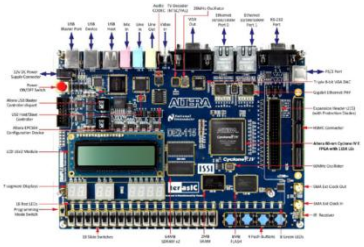
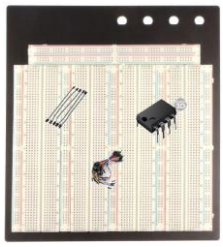
Obsługa potencjometru polega na cyklicznym wysyłaniu 16 bitowych danych szeregowych z FPGA za pośrednictwem linii SI synchronicznie z sygnałami CS i SCK. FPGA, będąc urządzeniem typu Master musi wysyłać właściwe poziomy sygnałów na liniach CS i SCK zgodnie z poniższym timingiem, oraz w tym przypadku również dane szeregowe. Badany potencjometr może przyjmować 256 różnych pozycji suwaka, stąd do kontroli jego położenia będą wykorzystywane 8-bitowe dane D7÷D0 (Data Byte). Wysyłane na początku transmisji słowo sterujące (COMMAND Byte) zawiera:
- bity C1C0 ustalające tryb pracy układu – w czasie testów będzie stosowane ustawienie 01,
- bity wyboru kanału P1P0 – układ zawiera tylko potencjometr P0 z ustawieniami 01.



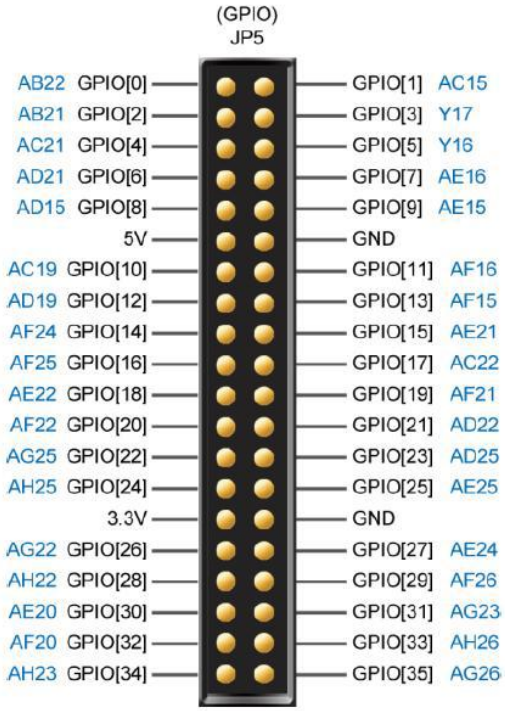
Projektując sygnały magistrali w module sterującym FPGA, należy zwrócić uwagę na zależności czasowe pomiędzy aktywnym zboczem sygnału zegarowego a zmianą danych na linii SI, tak żeby zmiana sygnału na linii danych i sygnału zegarowego nie następowały w tym samym czasie. W związku z tym należy zapewnić katalogowe wartości czasów t_{SU} i t_{HD}. Praktycznie, wystarczy zapewnić położenie narastającego zbocza sygnału zegarowego w środku okna czasowego bitu danych, czyli odpowiednio przesunąć sygnał zegarowy względem sygnału danych.

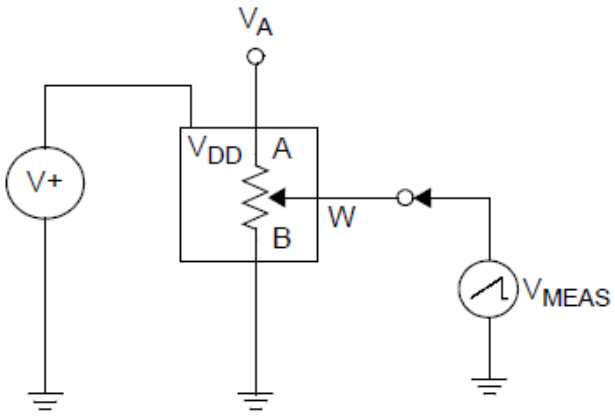


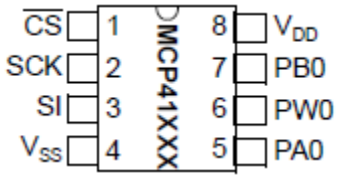
Badania potencjometru będą prowadzone na stanowisku zawierającym: moduł prototypowy FPGA DE2-115, oscyloskop, płytkę stykową oraz potencjometr i zestaw kabli łączących.


Układ MCP41010 będzie testowany w konfiguracji potencjometrycznej dzielnika napięciowego prądu stałego oraz okresowego sygnału prostokątnego wytwarzanego przez FPGA. W roli źródła sygnałów sterujących oraz zasilających należy wykorzystać złącze GPIO w module DE2-115. Z tego powodu, wszystkie punkty masy w układzie należy połączyć z pinem/pinami **GND** złącza GPIO modułu FPGA. Jako doprowadzenia sygnałów CS, SCK i SI można zastosować dowolne piny sygnałowe złącza GPIO, które są oznaczone na niebiesko na rysunku poniższym. Zasilanie $\mathbf{V_{DD}}$ potencjometru należy doprowadzić z pinu oznaczonego jako **5V** w złączu GPIO.







W przypadku testowania potencjometru w układzie dzielnika napięciowego prądu stałego, terminal $\mathbf{V_A}$ potencjometru należy zasilić z pinu 3.3V w złączu GPIO a sygnał wyjściowy na terminalu suwaka W potencjometru należy mierzyć oscyloskopem z użyciem sondy napięciowej.

Podczas testowania potencjometru w układzie dzielnika napięciowego sygnału okresowego, terminal $\mathbf{V_A}$ potencjometru należy połączyć z dowolnym pinem sygnałowym GPIO, do którego doprowadzony będzie cyfrowy sygnał prostokątny o regulowanej częstotliwości.

Moduł sterujący potencjometrem proszę opracować w formie komponentu zapisanego w bibliotece użytkownika. Będzie on wywoływany z kodu głównego VHDL z parametrami:

sygnału zegarowego 50MHz oraz bajtu danych nastaw potencjometru, natomiast będzie on zwracać sygnały CS oraz SCK i SI.

Zadanie laboratoryjne

Realizacja ćwiczenia będzie przebiegać wg następującego schematu:

- w ramach przygotowania do zajęć studenci opracowują projekt modułu obsługi magistrali SPI potencjometru, który weryfikują symulacyjnie;
- studenci opracowują również pomocniczy moduł przestrajanego generatora cyfrowego sygnału prostokątnego, który również sprawdzają za pomocą symulatora;
- ponadto studenci opracowują kompletny schemat połączeń układu na płytce stykowej;
- po rozpoczęciu zajęć zostaną skompilowane oba moduły a ich plikiem bit stream zostanie zaprogramowany moduł prototypowy FPGA;
- pracę modułów należy sprawdzić mierząc oscyloskopem z sondą cyfrową ich sygnały wyjściowe na złączu GPIO – w przypadku stwierdzenia rozbieżności przebiegów czasowych w stosunku do założeń, projekt należy poprawiać aż do skutku;
- po poprawnej weryfikacji funkcjonowania modułów w FPGA, należy połączyć układ potencjometru na płytce stykowej ze złączem GPIO i przetestować jego pracę.

Pomiary należy przeprowadzić w następujących zakresach:

- zmierzyć i wykreślić charakterystykę przejściową trybu stałoprądowego dzielnika, tzn. zmierzyć wartość napięcia stałego na suwaku w funkcji nastaw cyfrowych potencjometru, pomiary należy przeprowadzić, w co najmniej 25 punktach w przedziale nastaw bajtu sterującego od wartości 00000000 do wartości 11111111;
- zmierzyć i zarejestrować charakterystykę częstotliwościową potencjometru, podając w punkcie VA przestrajany częstotliwościowo sygnał prostokątny i rejestrując kształt impulsów wyjściowych z FPGA wraz z kształtem impulsów na wyjściu suwaka potencjometru, pomiar i rejestrację oscylogramów przeprowadzić, dla co najmniej 4 różnych częstotliwości.

Literatura:

1. Microchip Technology: Single/Dual Digital Potentiometer with SPI™ Interface.
2 Wykład przedmiotu Układy cyfrowe i programowalne.

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 9

Implementacja interfejsu I^2C w FPGA

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcję opracował:

dr inż. Marian Gilewski

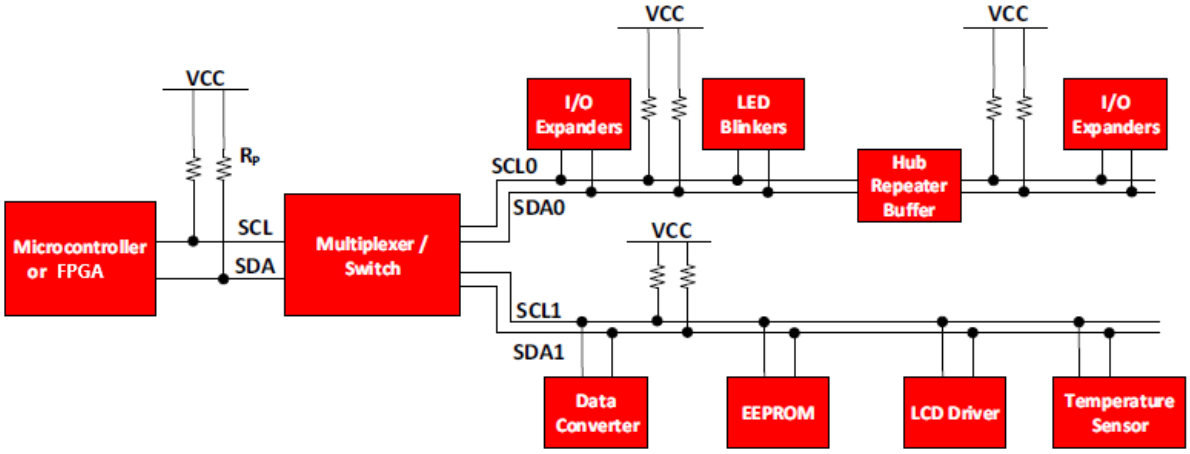
Białystok 2025

Cel ćwiczenia.

Celem ćwiczenia jest poznanie techniki syntezy w strukturach programowalnych popularnego szeregowego interfejsu sprzętowego typu I^2C, stosowanych do wymiany danych pomiędzy układami scalonymi cyfrowymi, analogowo-cyfrowymi lub analogowymi z interfejsem cyfrowym. Ćwiczenie będzie obejmować implementację konkretnej wersji interfejsu, służącego obsługi scalonego potencjometru cyfrowego przez układ Master, będącym w tym przypadku układem FPGA. Studenci po zrealizowaniu tego ćwiczenia nabędą umiejętność syntezy dowolnego interfejsu tej rodziny oraz wiedzę, umożliwiającą implementację interfejsów szeregowych innego typu.

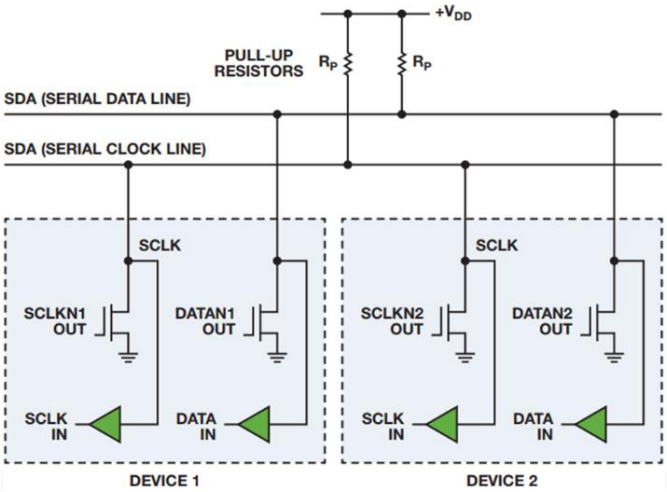
Wprowadzenie.

Magistrala I2C (Inter-Integrated Circuit – pośrednik pomiędzy układami scalonymi) jest wydajną 2-przewodowaą magistralą używaną do komunikacji między urządzeniem Master (lub wieloma urządzeniami Master) a jednym lub wieloma urządzeniami Slave. Magistrala została opracowana przez firmę Philips na początku lat 80 ubiegłego wieku. Maksymalna przepustowość tej magistrali jest rzędu kilku Mb/s, nie istnieje dolna graniczna szybkość transmisji. I²C jest magistralą zorientowaną bajtowo, co oznacza, że przesyłane dane i bity sterujące są grupowane w słowa będą wielokrotności 1 bajtu. W tej magistrali wyraźnie wyróżnia się nadajnik i odbiornik, a przesyłane bajty są potwierdzane przez odbiornik. W magistrali I2C do tej samej szyny może być dołączone wiele urządzeń, wybór konkretnego odbiornika danych następuje na podstawie unikalnego adresu. Ten rodzaj magistrali obsługuje



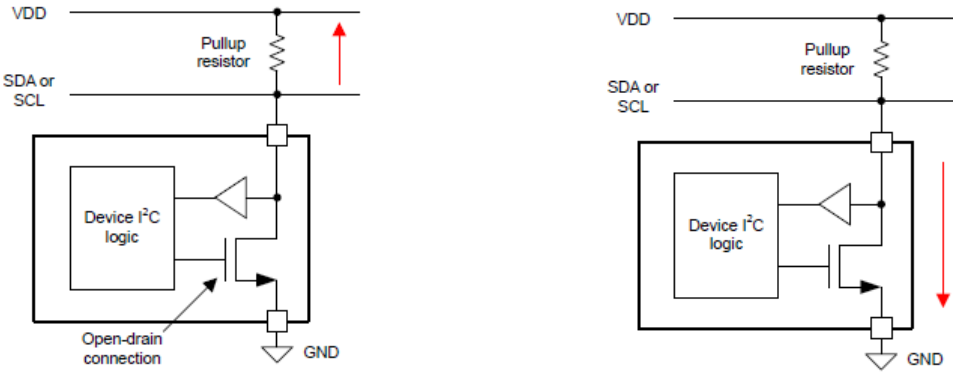
(źródło: https://diglab.technion.ac.il/projects/i2c-master-design-for-zynq-7000/)

wiele typów układów scalonych. Przewagą magistrali I2C są tylko dwie linie sygnałowe: jedna to SCL przesyłająca sygnał zegarowy i druga SDA służąca do dwukierunkowej transmisji danych pomiędzy urządzeniami. W odróżnieniu do SPI, I2C to komunikacja pół-dupleksowa, w której tylko jeden Master lub jeden Slave wysyła dane na magistralę w danym momencie. Charakterystyczną cechą sprzętową wszystkich urządzeń dołączonych do magistrali są wyjścia typu otwarty dren, co oznacza realizację iloczynu montażowego ma liniach sygnałowych i konieczność stosowania rezystorów polaryzujących RP (Pull-Up).



(źródło: https://www.analog.com/en/resources/technical-articles/i2c-cabling.html)

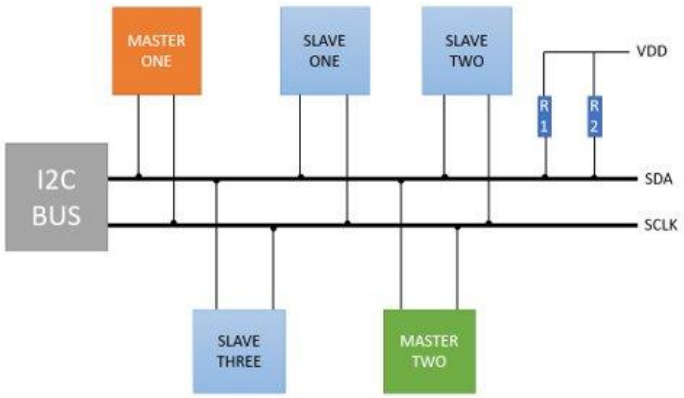
W stanie ustalonym przy braku transmisji, na obu liniach występuje poziom wysoki wymuszony przez zasilanie VDD, włączenie tranzystora wyjściowego jednego z urządzeń powoduje zwarcie do masy przez rezystancję kanału i wymuszenie na linii poziomu niskiego.



(źródło: https://electrosome.com/i2c/)

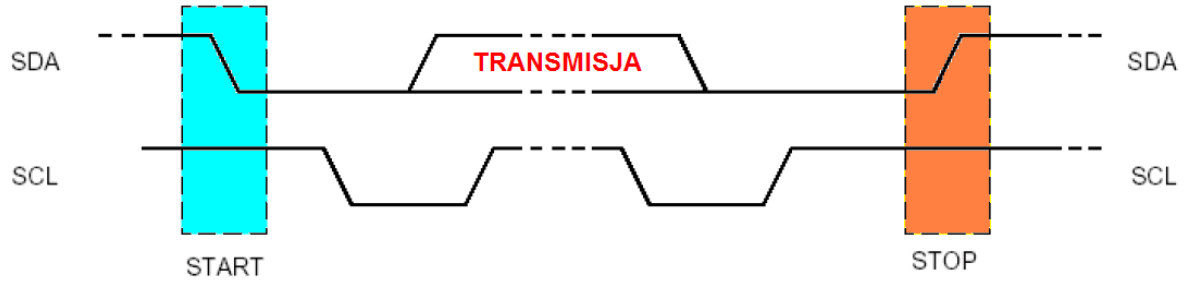
Układy podłączone do magistrali pracują jako urządzenia:

- Master – rozpoczyna i kończy cykl transmisji, wytwarza sygnał zegarowy SCL, wysyła adres urządzenia Slave, wysyła bit kierunku transmisji i czasami ustawia bit potwierdzenia,
- Slave – przyjmuje lub wysyła dane na polecenie Master synchronizowane sygnałem zegarowym SCL, czasami wysyła bity potwierdzenia.



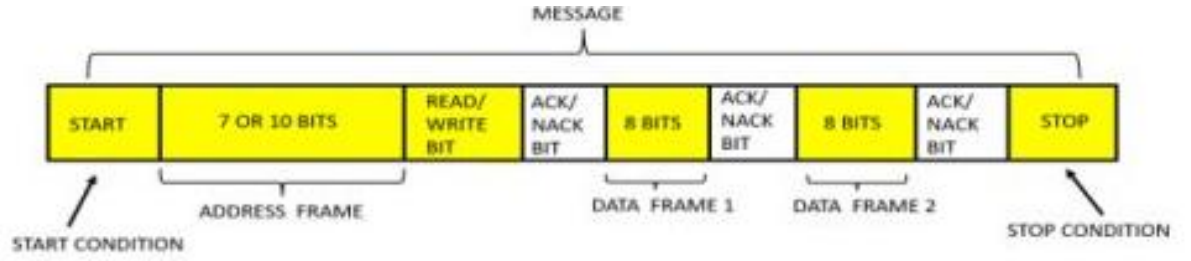
(źródło: https://www.design-reuse.com/article/61458-i2c-interface-timing-specifications-and-constraints/)

Urządzenie Master rozpoczynając transmisję wytwarza sygnał Startu a po jej zakończeniu sygnał Stopu lub kolejny sygnał startu. Transmisja odbywa się tylko pomiędzy Master a zaadresowanym urządzeniem/urządzeniami Slave, pozostałe urządzenia czekają na zwolnienie linii danych, czyli na sygnał końca transmisji. Urządzenia Slave śledzą stan magistrali w oczekiwaniu na sygnał startu i ich własny adres, i po jego wykryciu wczytują dane lub wysyłają je do urządzenia Master.



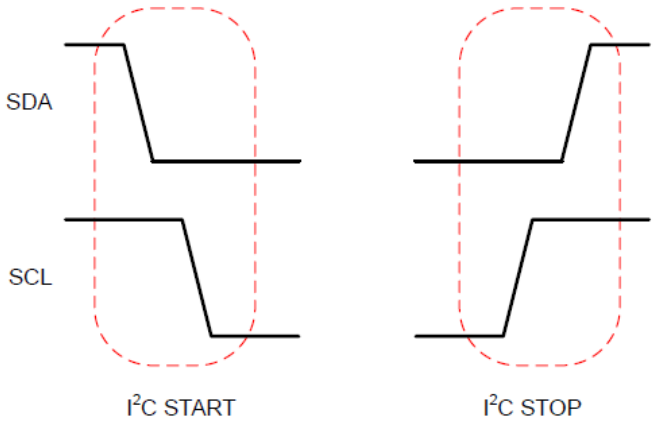
(opracowano na podstawie: https://www.design-reuse.com/article/61458-i2c-interface-timing-specifications-and-constraints/)

Typowa ramka protokołu transmisyjnego składa się z: sygnału Startu, po nim występuje 7 lub 10 bitowy adres urządzenia Slave, następnie wysyłany jest bit kierunku transmisji (READ / WRITE), po bicie kierunku występuje bit potwierdzenia adresu, następnie wysyłane są 8 bitowe dane, bajty danych kończą się bitami potwierdzenia odbioru, cykl transmisji kończy sygnał Stopu. Ilość przesyłanych bajtów danych nie jest ograniczona.

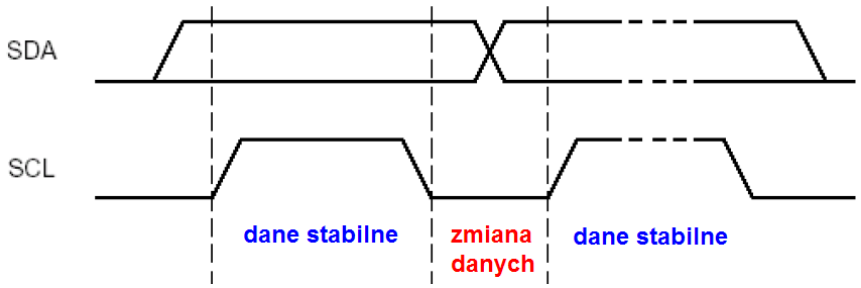


(źródło: https://www.design-reuse.com/article/61458-i2c-interface-timing-specifications-and-constraints/)

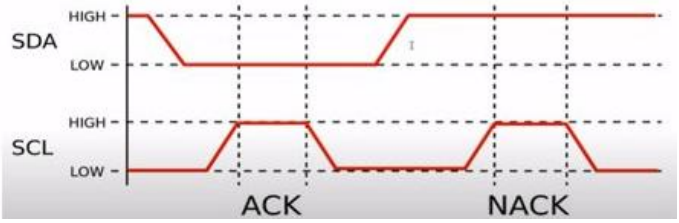
Dane są wysyłane w kolejności od najbardziej znaczącego bitu do najmniej znaczącego. Znaczenie poszczególnych danych nie jest zdefiniowane dla całego standardu, gdyż określają je szczegółowe dane producenta układu scalonego. Przed rozpoczęciem transmisji obie linie



magistrali I2C znajdują się w stanie wysokim. Master wymusza na linii SDA stan niski przy wysokim stanie na SCL, a następnie wymusza stan niski na linii SCL – taka sekwencja sygnałów oznacza sygnał Startu dla urządzeń Slave. Zwolnienie linii SDA, czyli jej przejście do stanu wysokiego w czasie wysokiego poziomu sygnału zegarowego – oznacza sygnał Stopu, czyli zakończenie lub przerwanie transmisji. Dlatego wszelkie zmiany sygnału (danych) na linii SDA mogą występować tylko w czasie niskiego stanu linii, gdyż w przeciwnym przypadku będą one rozumiane, jako sygnał Stopu. W czasie narastającego zbocza sygnału SCL dane na linii SDA powinny być stabilne.



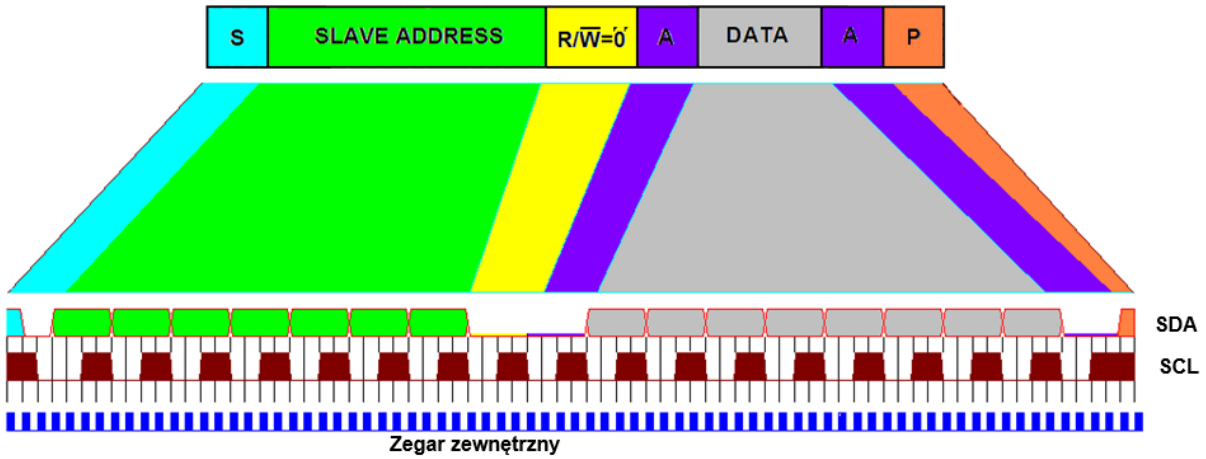
W czasie transmisji jest ściśle określone, który z układów dla danego impulsu SCL odpowiada za stan linii SDA, drugi z układów musi w tym czasie zwolnić linię SDA. 8-bitowa transmisja danych uzupełniana jest przez 9 bit potwierdzenia. Ten bit na linii SDA jest wysyłany w przeciwnym kierunku do nadajnika, niż bity bajtu danych.



(źródło: https://www.design-reuse.com/article/61458-i2c-interface-timing-specifications-and-constraints/)

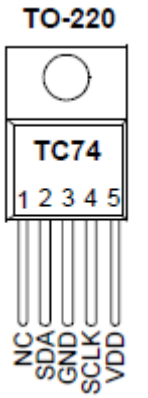
Potwierdzenie (ACK) oznacza wymuszenie przez odbiornik stanu niskiego na SDA w 9 takcie zegarowym, stan wysoki SDA w tym czasie oznacza brak potwierdzenia (NACK). Brak potwierdzenia informuje nadajnik o odłączonym/uszkodzonym/zajętym odbiorniku. Pierwszy bajt po sygnale Startu jest zawsze wysyłany przez układ Master, 7 bitów danych jest adresem urządzenia slave zaś 8 bit oznacza kierunek transmisji: 0 wymusza kierunek write z Master do Slave, 1 wymusza kierunek read od Slave do Master. Adres zawsze musi być potwierdzony przez Slave. Nie wolno zmieniać kierunku danych do zakończenia cyklu transmisji.
Sterowniki magistrali I2C w FPGA można projektować stosując różne techniki. Żmudną, aczkolwiek bardzo skuteczną jest metoda FSM. Na początku należy precyzyjnie w skali rozrysować wzajemne położenie sygnałów SDA i SCL w ramce transmisyjnej. Każdy impuls zegarowy należy syntezować w czasie 4 lub 3 impulsów zewnętrznego sygnału zegarowego FSM o wyższej częstotliwości. W ten sposób zapewnione będzie spełnienie katalogowych czasów ustawiania i podtrzymania danych SDA względem narastającego zbocza sygnału zegarowego SCL magistrali, gdyż w każdej trójce/czwórce impulsów zewnętrznego sygnału zegarowego wystąpią stany: ustawiania bitu SDA przy niskim poziomie SCL, wystąpienie impulsu SCL w czasie stabilnej wartości SDA, zmiana wartości bitu SDA na nową, gdy sygnał

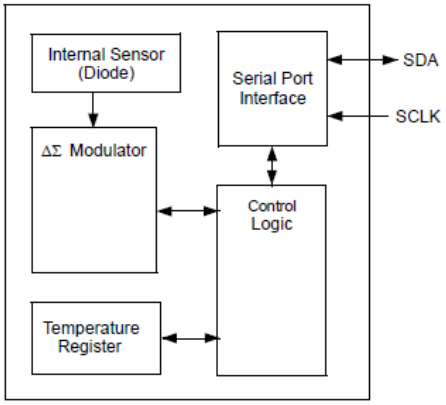
SCL nie jest aktywny. Tak rozrysowane kombinacje sygnałów SDA i SCL dla każdego impulsu zegara zewnętrznego należy nazwać kolejnymi stanami, a następnie utworzyć kod FSM dowolną metodą.



Zadanie laboratoryjne

W systemie Quartus II zaprojektować moduł obsługi magistrali I2C termometru cyfrowego, moduł należy zaimplementować w płycie prototypowej DE2-115 łącząc ją z układem scalonym potencjometru zamontowanym na płytce stukowej. Przed przystąpieniem do prac przygotowawczych proszę zapoznać się z dokumentacją techniczną potencjometru. Układ termometru zawiera 2-przewodową magistralę I2C, zawierającą sygnały SDA i SCLK. Ponadto zawiera on dwa piny zasilające: VDD – plus zasilania, GND – wspólną masę zasilania i sygnałową (katalog TC74).





Obsługa termometru polega na cyklicznym odczycie 8 bitowych danych wartości temperatury z linii SDA synchronizowanej sygnałem zegarowym SCLK. FPGA, jako urządzenie Master musi wysyłać właściwe poziomy sygnałów na liniach SDA i SCLK oraz odczytywać wartości temperatury z linii SDA zgodnie z poniższym timingiem (katalog TC74).

Receive Byte Format

S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits		

Należy opracować sterownik w FPGA dla najprostszej ramki transmisyjnej Receive Byte Format. W laboratorium będzie badany model termometru TC74A0-5.0VAT, którego adres należy odczytać z dokumentacji technicznej. Obsługa potencjometru będzie polegać na cyklicznym: wygenerowaniu sygnału Startu (S), wysłaniu 7-bitowego adresu, wysłaniu bitu kierunku w trybie Read, zignorowaniu bitu potwierdzenia adresu przez Slave, odczycie 8 bitowych danych i zakończeniu transmisji sygnałem Stop (P).
Moduł DE2-115 powinien wyświetlać zmierzoną wartość temperatury, wykorzystując komponent wyświetlacza z ćwiczenia nr 1 zapisany w pakiecie/bibliotece użytkownika. Moduł obsługujący termometr należy również zdefiniować w postaci procedury w bibliotece użytkownika. Termometr TC74 zwraca wartość zmierzonej temperatury w systemie zapisu danych binarnych U2. Dlatego należy również opracować moduł konwersji 8-bitowych danych w kodzie U2 na 8-bitowe dane w naturalnym kodzie binarnym, gdyż temperatury ujemne nie będą mierzone. Moduł konwersji należy zdefiniować, jako funkcję, która jest zapisana w bibliotece użytkownika (katalog TC74).

Actual Temperature	Registered Temperature	Binary Hex
+130.00°C	+127°C	0111 1111
+127.00°C	+127°C	0111 1111
+126.50°C	+126°C	0111 1110
+25.25°C	+25°C	0001 1001
+0.50°C	0°C	0000 0000
+0.25°C	0°C	0000 0000
0.00°C	0°C	0000 0000
-0.25°C	-1°C	1111 1111
-0.50°C	-1°C	1111 1111
-0.75°C	-1°C	1111 1111
-1.00°C	-1°C	1111 1111
-25.00°C	-25°C	1110 0111
-25.25°C	-26°C	1110 0110
-54.75°C	-55°C	1100 1001
-55.00°C	-55°C	1100 1001
-65.00°C	-65°C	1011 1111

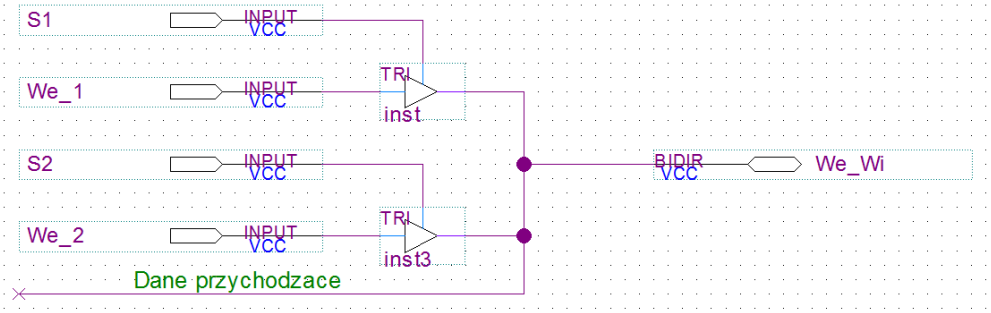
Badania termometru będą prowadzone na identycznym stanowisku jak w przypadku zadania nr 1, zawierającym: moduł prototypowy FPGA DE2-115, oscyloskop, płytkę stykową oraz układ termometru, zestaw kabli łączących i rezystorów Pull-Up. Zasilanie VDD potencjometru należy doprowadzić z pinu oznaczonego, jako 5V w złączu GPIO. Linie sygnałowe SDA i SCLK należy połączyć z pinami I/O złącza GPIO. Ustawienie pinów sygnałowych FPGA w tryb open drain można zrealizować w dwojaki sposób. Pierwszy sposób polega na ustawieniu parametru Auto Open-Drain Pins na On w Assignment Editor systemu Quartus II.

Reklama.bdf — Assignment Editor*

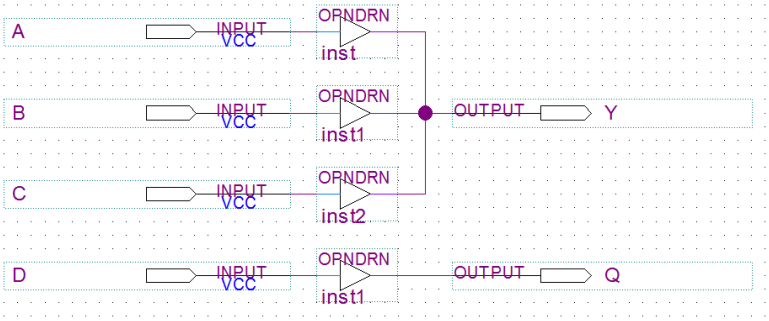
<<new>> Filter on node names: *

	Status	From	To	Assignment Name	Value	Enabled	Entity	Comment	Tag
41	Ok		K_Green[4]	Current Strength	16mA	Yes	Reklama		
42	Ok		K_Green[3]	Current Strength	Max...ent	Yes	Reklama		
43	Ok		K_Green[2]	Current Strength	Mini...rent	Yes	Reklama		
44	Ok		W_Green[7]	Auto Open-Drain Pins	On	Yes	Reklama		
45		<<new>>	<<new>>	<<new>>					

Drugi sposób ustawienia open drain dla portów dwukierunkowych można zrealizować na poziomie schematu:



dodając szeregowy element (dyrektywę kompilatora) OPNDRN lub TRI.



Realizacja ćwiczenia będzie przebiegać wg następującego schematu:

- w ramach przygotowania do zajęć studenci opracowują projekty modułów: obsługi magistrali I2C oraz konwersji U2 na BIN, który weryfikują symulacyjnie;
- ponadto studenci opracowują kompletny schemat połączeń układu na płytce stykowej;
- po rozpoczęciu zajęć zostaną skompilowane oba moduły a ich plikiem bit stream zostanie zaprogramowany moduł prototypowy FPGA;
- pracę modułu I2C należy sprawdzić mierząc oscyloskopem z sondą cyfrową sygnały SDA i SCLK na złączu GPIO - w przypadku stwierdzenia rozbieżności przebiegów czasowych w stosunku do założeń, projekt należy poprawiać aż do skutku;
- po poprawnej weryfikacji funkcjonowania modułów w FPGA, należy połączyć układ termometru na płytce stykowej ze złączem GPIO i przetestować jego pracę.

Pomiary należy przeprowadzić w następujących zakresach:

- zmierzyć kilka wartości temperatury nagrzewając obudowę termometru,
- zarejestrować oscyloskopem przykładowe sygnały magistrali I2C.

Literatura:

1. Microchip Technology Inc.: TC74 - Tiny Serial Digital Thermal Sensor.

Politechnika Białostocka

Wydział Elektryczny

Katedra Automatyki i Robotyki

ĆWICZENIE Nr 10

Poprawa i zaliczenie ćwiczeń laboratoryjnych

Układy cyfrowe i programowalne

studia stacjonarne pierwszego stopnia, kierunek: Elektromobilność, semestr 1

Kod przedmiotu: **EMS1A1009**

Instrukcję opracował:

dr inż. Marian Gilewski

Białystok 2025

Cel ćwiczenia.

Będą to zajęcia uzupełniające i podsumowujący kurs laboratorium. W czasie zajęć zostaną ustalane oceny końcowe. Każdy student ma prawo poprawić o 0,5 pkt ocenę końcową - realizując samodzielnie dodatkowe zadanie związane z zakresem laboratorium. Ponadto studenci posiadający z różnych przyczyn (losowych, zwolnień lekarskich, itp.) zaległości w zakresie zrealizowanych zajęć laboratoryjnych, mogą je uzupełnić w ramach niniejszego ćwiczenia.

Literatura:

1. Barski M., Jędruch W.: Układy cyfrowe, podstawy projektowania i opisu w języku VHDL, Wydawnictwo Politechniki Gdańskiej, 2007.
2. Dueck R., Digital Design with CPLD Applications and VHDL 2nd Edition, Cengagae Learning Inc., 2020.
3. Grodzki L., Owieczko W., Podstawy techniki cyfrowej, Wydawnictwo PB, 2006.
4. Tyszer J., Mrugalski G., Pogiel A., Czysz D., Technika cyfrowa: zbiór zadań z rozwiązaniami, Legionowo: Wydaw. BTC, 2010.
5. Kamionka-Mikuła H., Małysiak H., Pochopień B., Teoria układów cyfrowych. T.2, Układy sekwencyjne, Gliwice: Wydaw. Politechniki Śląskiej, 2013.
6. Salauyou V., Klimowicz A., Projektowanie systemów wbudowanych w układach FPGA, Białystok, Oficyna Wydawnicza Politechniki Białostockiej, 2022.
7. Barkalov A., Titarenko L., Kolopienczyk M., Mielcarek K., Bazydlo G., Logic Synthesis for FPGA-Based Finite State Machines, Springer International Publishing, 2016.
8. MIT, Finite State Machines, https://web.mit.edu/6.111/www/f2017/handouts/L06.pdf, 2017, dostępność 7.09.2025.
9. Encinas J., Phase Locked Loops, Springer, 2012.