LABORATORIUM
PROGRAMOWALNYCH UKŁADÓW
LOGICZNYCH

1. WPROWADZENIE

W12IEA-SI0038P

W12EIT-SI0106P

wzn.pwr.edu.pl/materialy-dydaktyczne/PUL

dr inż. Bartosz Świadkowski

dr inż. Piotr Smagowski

1. CEL ĆWICZENIA

Zapoznanie ze środowiskiem Xilinx® Vivado oraz językiem opisu sprzętu VHDL. Program zajęć wprowadzających obejmuje:

- tworzenie projektu w środowisku Vivado® 2032.1,
- tworzenie symulacji
- synteza, fizyczna implementacja kodu w strukturze FPGA

2. ZAGADNIENIA

Język VHDL: podstawowe pojęcia języka (blok entity, blok architecture, zmienne, sygnały, procesy, sekwencyjne i równoległe wykonanie instrukcji), definicje stałych, sygnałów i zmiennych, definicja procesu, instrukcje podstawień. Zasady wykonywania instrukcji opisanych językiem VHDL (równoległe oraz sekwencyjne wykonywanie programu). Umiejętność pisania prostych opisów logiki w sposób behawioralny – tworzenie procesów, tworzenie liczników/dzielników, rejestrów, operacje logiczne oraz arytmetyczne wykonywane na zmiennych i sygnałach, instrukcje warunkowe, instrukcje wyboru.

Zadania do wykonania w ramach laboratorium:

1. Synteza i implementacja układu kombinacyjnego.

2. Synteza i implementacja układu sekwencyjnego (licznika)

3. Synteza i implementacja rejestru przesuwnego.

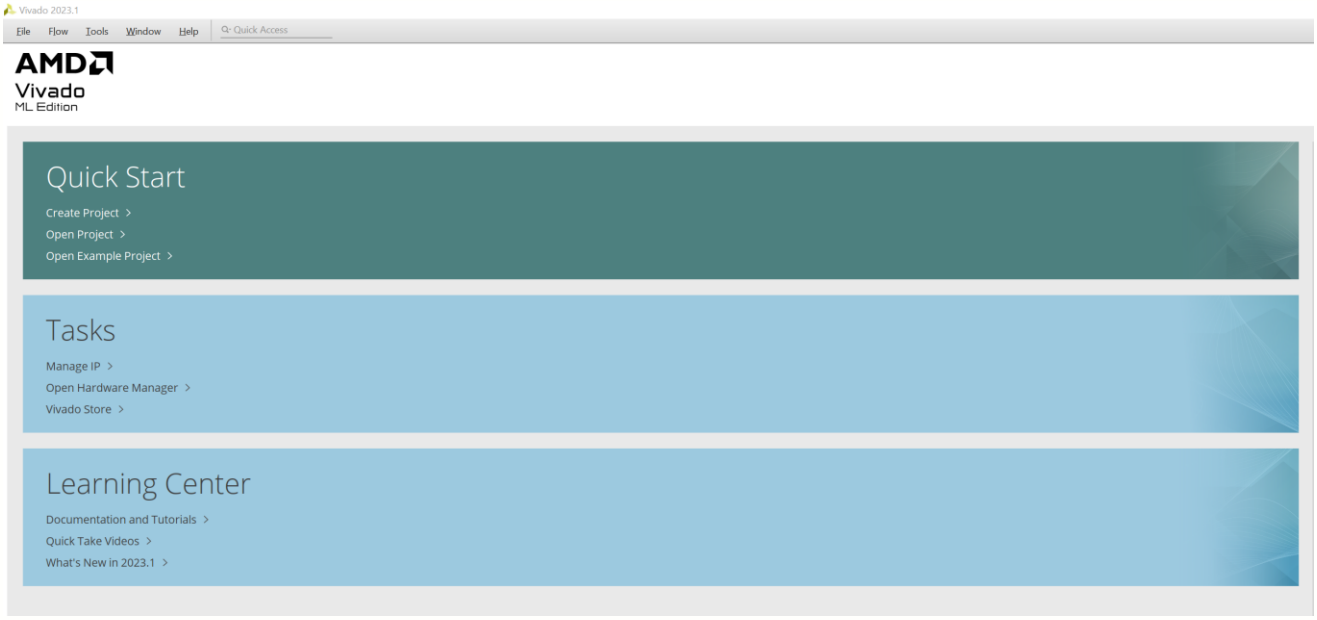
Dygresja.

Materiały pomocnicze

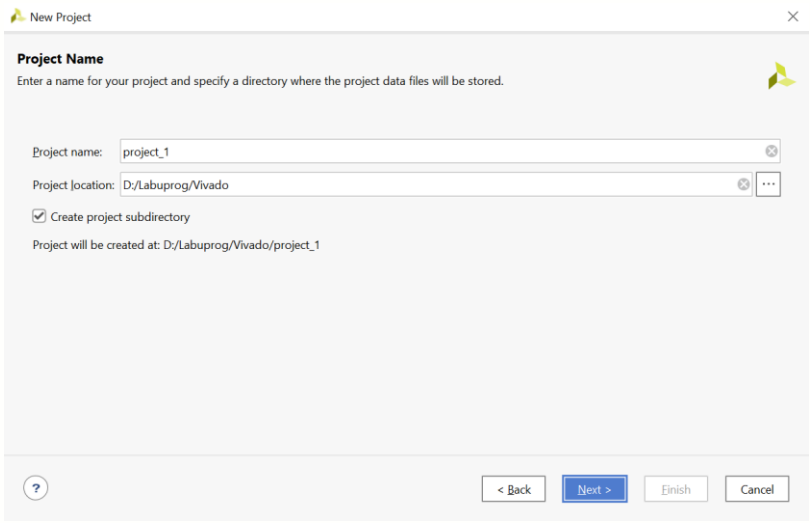
Na stronie laboratorium udostępniliśmy rozmatie materiały, które w naszym rozumieniu powinny być pomocne przy realizacji ćwiczeń. W tekście instrukcji laboratoryjnych odnosimy się do części z tych materiałów. Jeśli jednak odniesienie nie jest jednoznaczne, dajcie znać prowadzącym, co jest niespójne lub wymaga dodatkowego wyjaśnienia.

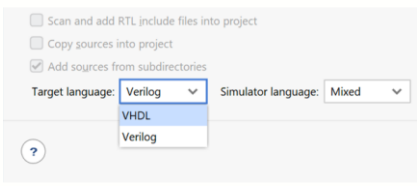
3. WPROWADZENIE DO NARZĘDZI VIVADO

3.1. Uruchomienie programu następuje po wywołaniu ***Vivado 2023.1***, czego wynikiem jest pojawienie się okna powitalnego. ***Dygresja:*** *w materiałach pomocniczych znajduje się plik z podpowiedziami dotyczącymi pakietów, które powinny zostać zainstalowane spośród ogromu dostępnych narzędzi zintegrowanego pakietu Vivado/Vitis.*



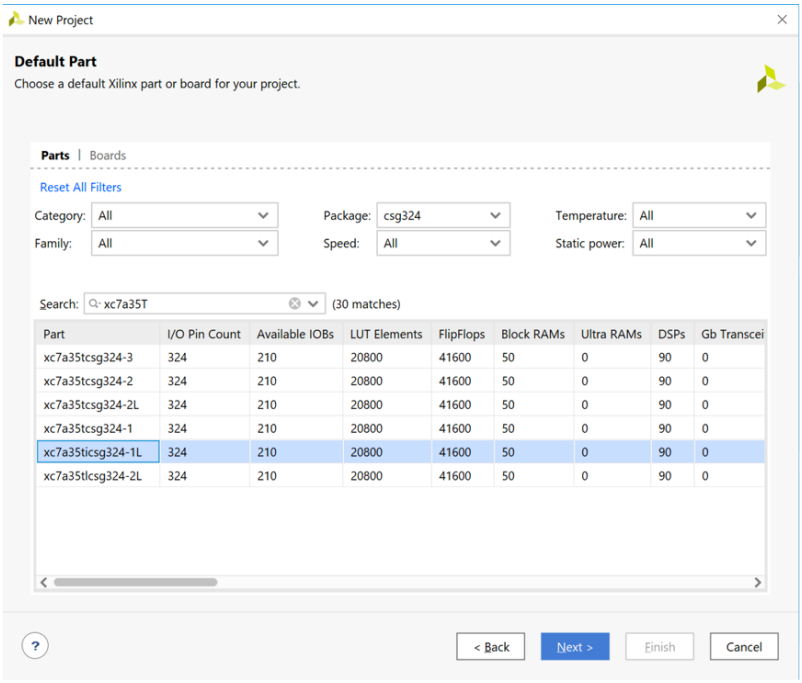
W oknie głównym programu należy stworzyć nowy projekt za pomocą polecenia ***Create Project,*** alternatywnie można to samo zrobić wybierając ***Project -> New*** w menu ***File.*** W oknie projektu należy podać nazwę projektu oraz ścieżkę, w której zostanie zapisany (**należy pamiętać, aby w nazwie projektu nie używać polskich znaków oraz spacji**).





Aby założyć projekt dla używanego na zajęciach zestawu deweloperskiego Digilent Arty*, jako układ projektu należy wybrać układ **XC7A35TICSG324-1L.** XC7A – to oznaczenie serii Artix 7 35T – pojemność/ilość bloków logicznych (ok. 35

tysięcy) CSG324 – rodzaj obudowy Chip Scale Package z 324 wyprowadzeniami 1L – klasa szybkości 1, układ Low Power.

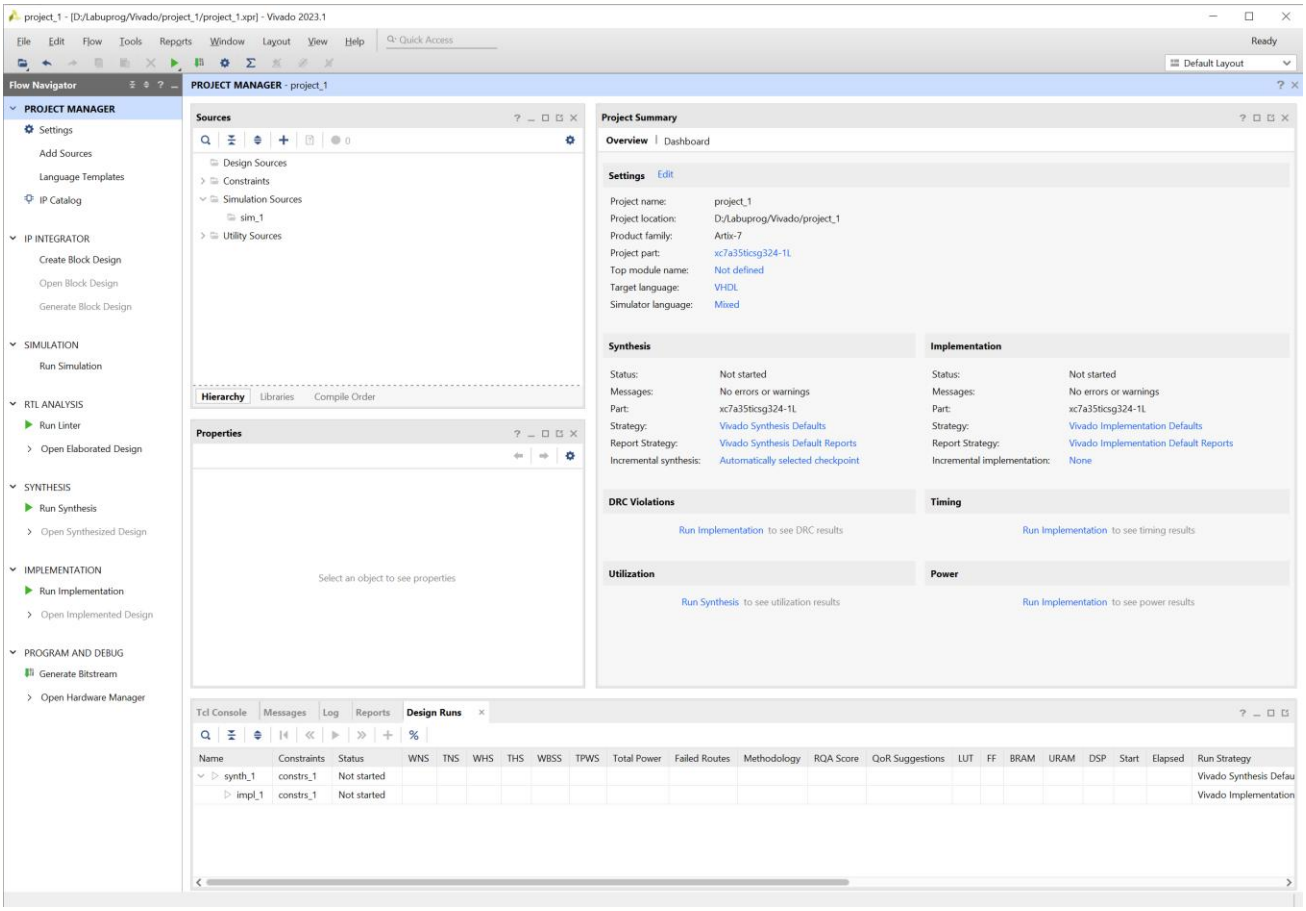


W następnym kroku należy wybrać ***RTL Project*** oraz wybrać język projektu – VHDL. Kroki dodawania źródeł (***Add sources***) oraz założeń projektowych (***Add constraints***) pominąć.

* Dokumentacja zestawu dostępna pod linkiem:

https://reference.digilentinc.com/reference/programmable-logic/arty/reference-manual

Po zakończeniu pracy z kreatorem ukaże się główne okno programu w perspektywie **Project Manager**:



Perspektywa okna jest zmienna, tzn. zawartość okien dostosowuje się do konkretnego zadania. Project Manager służy edycji kodu źródłowego. Po syntezie projektu pojawiaja się tu również okno podsumowania z informacjami o zajętości zasobów, przewidywanym poborze mocy czy spełnieniu założeń częstotliwości pracy.

3.2. Perspektywy (widok) programu Vivado

Okno programu dzieli się na trzy części:

1. Flow navigator – służy do przemieszczenia się po kolejnych perspektywach na różnych etapach projektu

2. Okno główne – służy do edycji treści

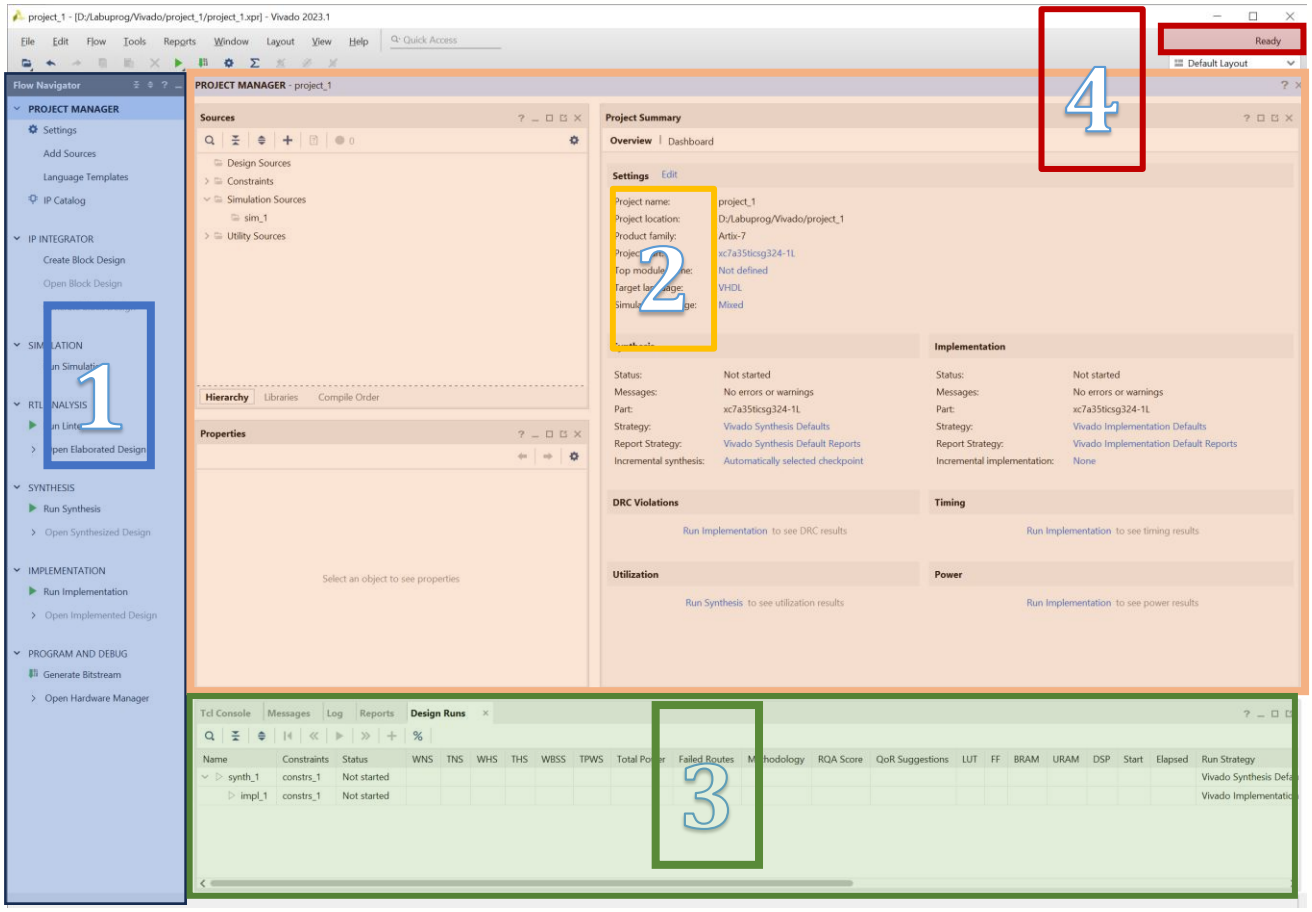
3. Konsola – zawiera aktualne informacje o zachodzących procesach, dziennik zdarzeń, dostęp do raportów czy podglądu ostrzeżeń i błędów

4. Informacja o stanie procesów (syntezy, implementacji etc.)

W perspektywie Project Manager w oknie głównym (2) w lewej części zawarte jest drzewo plików źródłowych, z których do najważniejszych należą:

- Design Sources – pliki źródłowe VHDL,

- Constraints – pliki założeń projektowych, tj. np. pliki z ograniczeniami zakładanej częstotliwości pracy układu FPGA, przypisaniem wyprowadzeń obudowy FPGA do wejść i wyjść jednostki nadrzędnej w VHDL i in.



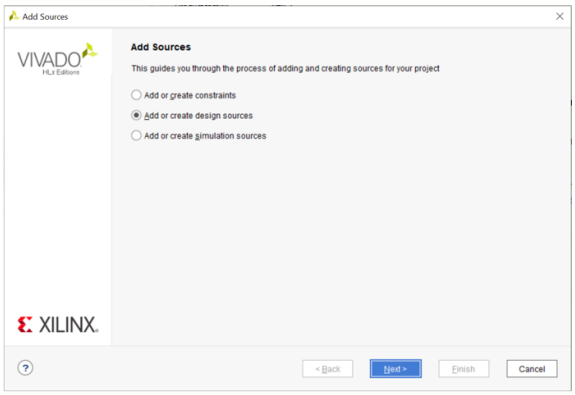
3.3. Pierwszy projekt

W trakcie zajęć laboratoryjnych/projektowych oraz przy realizacji projektów własnych należy posługiwać się językiem VHDL. Aby stworzyć plik VHDL należy wybrać ***File -> Add sources***.

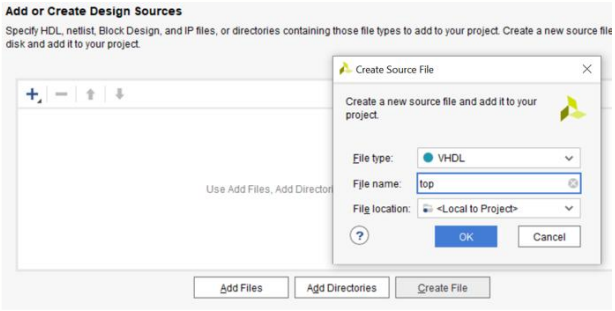
- Constraints – pliki założeń (ograniczeń)

- Design sources – pliki źródłowe VHDL

- Simulation sources – pliki tylko do symulacji



Klikamy ***Create File***, nadajemy nazwę i upewniamy się, że jest on typu VHDL.



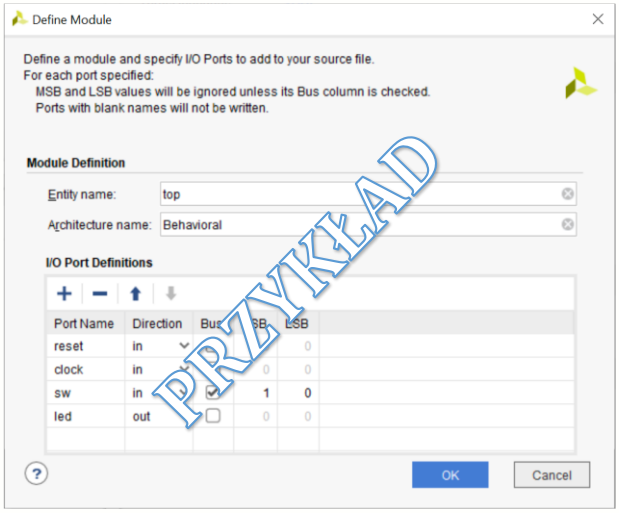
W kolejnym kroku można wypełnić pola jednostki, tj. ***entity***:

- nazwę modułu,

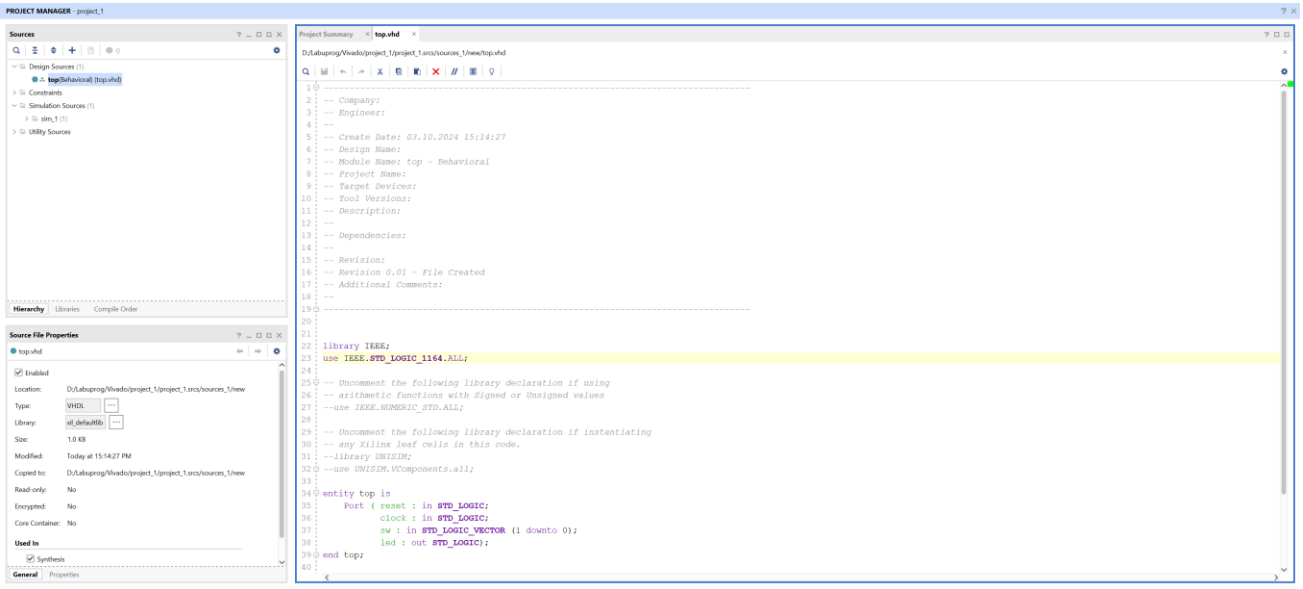
- rodzaj opisu architektury,

- porty (wejścia i wyjścia ***entity***):

nazwę, kierunek (wejście lub wyjście), czy to bit pojedynczy, czy magistrala oraz jej szerokość. Na tym etapie nie uzupełniamy danych portów. Szczegóły te można uzupełnić ręcznie w edycji pliku *.vhd.

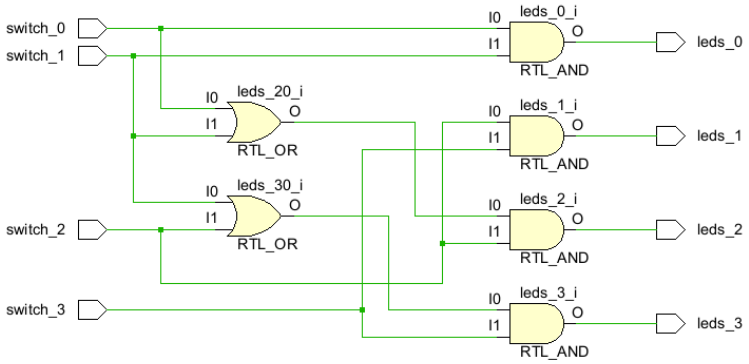


Możemy teraz przystąpić do edycji wygenerowanego pliku źródłowego *.vhd z zupełnionymi wcześniej danymi. Dodana zostały również definicja i polecenie użycia biblioteki standardowej STD_LOGIC_1164, która zawiera definicję typu STD_LOGIC oraz jej pochodnych.



Do zrobienia.

Układ kombinacyjny.



Pierwszym projektem zaimplementowanym fizycznie w układzie programowalnym w ramach wprowadzenia do zajęć będzie układ kombinacyjny. Za pomocą języka opisu sprzętu VHDL należy utworzyć układ, który połączy porty jednostki projektowej w sposób zaprezentowany na rysunku. Uwaga. W celu zwiększenia czytelności sygnałów, zastosowano porty o nazewnictwie odpowiadającym typowi **std_logic**. W celu przyspieszenia opisu można posłużyć się opisem indeksowanym oraz typem **std_logic_vector**, tj. zamiast switch_0, switch_1, itp. będzie switch(0), switch(1)…. Zmieni się wówczas wygląd schematu RTL (*register-transfer level*) implementowanego układu. Plik tak przygotowanej jednostki projektowej (*entity*), będzie miał następującą strukturę przedstawioną obok. Docelowo, układ logiczny zaimplementowany w układzie programowalnym, będzie reagował zmianą świecenia diod na makiecie, przy zmianie stanów wejściowych przełączników suwakowych (switch). To będzie nasz pierwszy układ typu *blink*, czyli „Hello world" w dziedzinie układów mikroprocesorowych i programowalnych! Co prawda diody nie będą migać okresowo, ale użytkownik będzie miał wpływ na ich stan w każdej chwili.

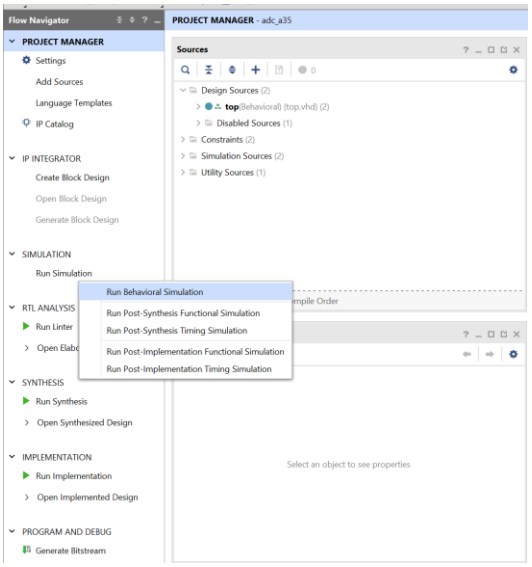
```
library IEEE;
use IEEE.std_logic_1164.all;

entity Top is
  Port ( switch: in std_logic_vector(3 downto 0);
         leds: out std_logic_vector(3 downto 0));
end Top;

architecture Behavioral of Top is

begin
-- tutaj wpiszemy
-- nasz syntezowalny
-- kod ukladu kombinacyjnego
end Behavioral;

end arcitecture
```

Ważne.

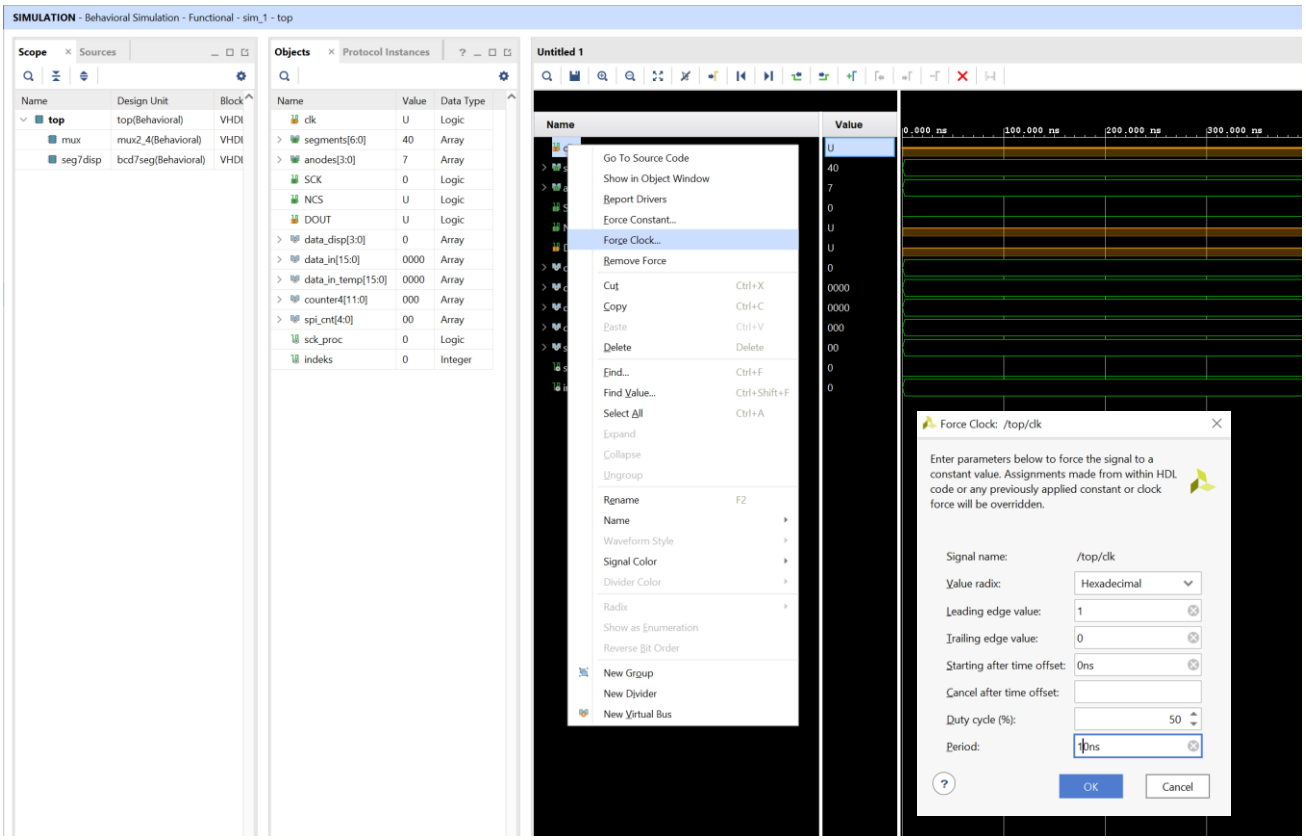
Na następnych stronach znajdą się dalsze instrukcje zmierzające do implementacji opracowanego układu. Na tym etapie nie ma raczej potrzeby symulacji przykładu, zatem powinniśmy przejść do uruchomienia syntezy (pkt. 3.5), przypisania portów i utworzenia i wgrania na makietę pliku konfiguracyjnego połączeń (tzw. bitstream).

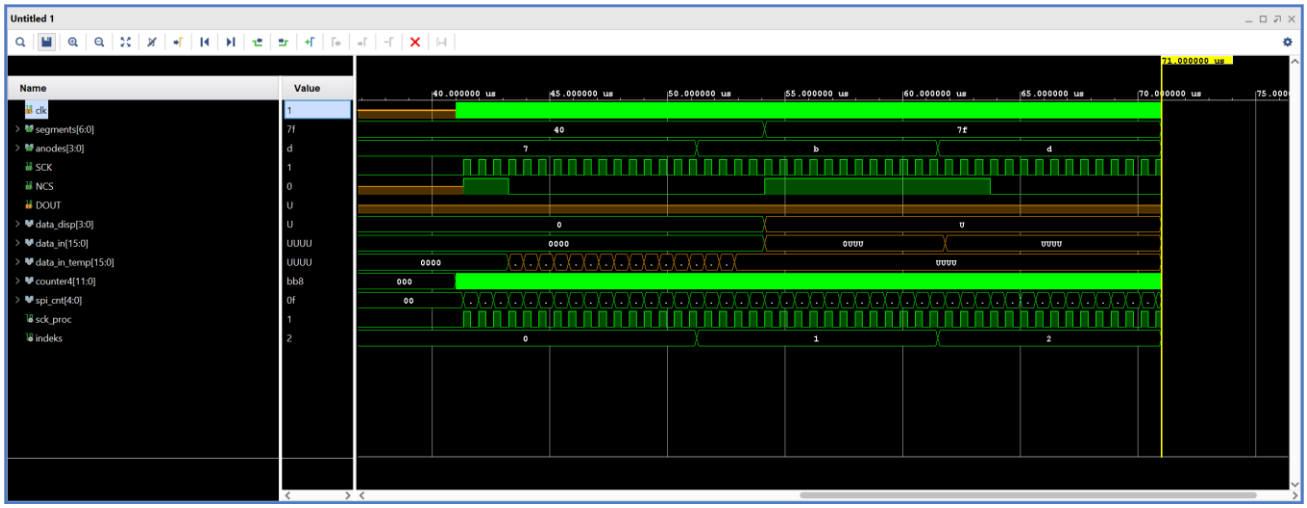
3.4 Symulacja

Aby wykonać symulację projektu, należy albo dodać plik symulacji (tzw. ***testbench***, dobrze znany z innych kursów!), albo bezpośrednio uruchomić symulację z poziomu nawigatora projektu.



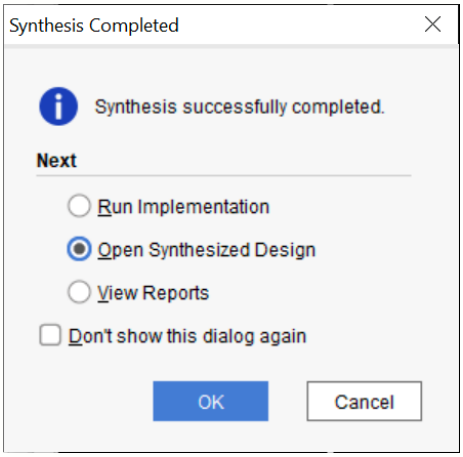
Uruchomi się wówczas okno przebiegów czasowych sygnałów. Na tym etapie można wprowadzać wymuszenia sygnałów, np. przebiegu zegara.

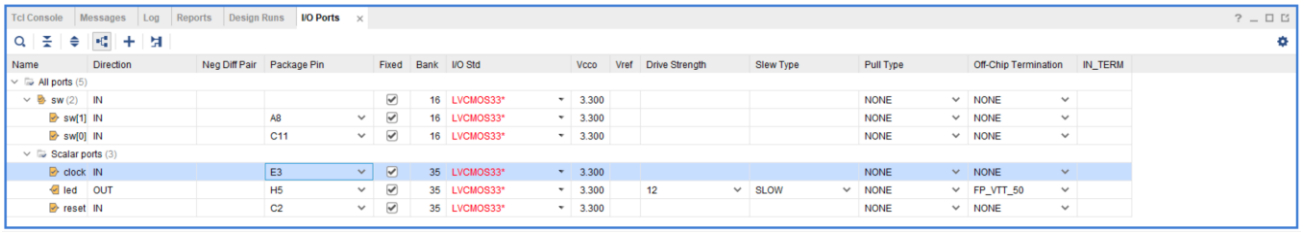




3.5. Synteza, implementacja, debugowanie

Po poprawnie wykonanej symulacji, projekt należy zsyntezować wybierając we Flow Navigatorze ***Synthesis -> Run Synthesis***. Po ukończeniu wybieramy ***Open Synthesized Design***, by przypisać sygnały do wyprowadzeń układu FPGA, tj. uzupełnić założenia projektowe. W tym celu wybieramy menu ***Window -> I/O Ports***. W oknie konsoli powinna pojawić się lista sygnałów z modułu nadrzędnego. Wybieramy odpowiednie wyprowadzenie w polu ***Package Pin*** oraz ***I/O Std*** ustawiamy na ***LVCMOS33***, co odpowiada standardowi 3,3 V.





Plik ograniczeń (*constraints*) zapisujemy nadając mu wybraną nazwę.

Początkowo, taki sposób przypisywania sygnałów wydaje się być wygodniejszym, ponieważ wykorzystuje graficzny interfejs użytkownika. Jednakże warto obrać z pozoru trudniejszą drogę przypisywania atrybutów portom wyjściowym, za pomocą pliku ograniczeń (.xdc), przygotowanego dla konkretnych płytek (rozwojowych lub też specjalizowanych). Taki plik dla płytek rozwojowych stosowanych w laboratorium znajduje się w materiałach do zajęć (Arty-A7-35-Master.xdc oraz PUL_Arty_Shield.xdc). Aby go użyć w projekcie, należy go dodać za pomocą ikony „Add sources" w „Project Manager". Przykład sposobu przypisania atrybutów portom za pomocą pliku ograniczeń zaprezentowano na poniższym rysunku.

```
## Clock signal
set_property -dict {PACKAGE_PIN E3 IOSTANDARD LVCMOS33} [get_ports clk]
create_clock -period 10.000 -name sys_clk_pin -waveform {0.000 5.000} -add [get_ports clk]

## Switches
set_property -dict {PACKAGE_PIN A8 IOSTANDARD LVCMOS33} [get_ports reset]
#set_property -dict { PACKAGE_PIN C11    IOSTANDARD LVCMOS33 } [get_ports { sw[1] }]; #IO_L13P_T2_MRCC_16 Sch=sw[1]
#set_property -dict { PACKAGE_PIN C1     IOSTANDARD LVCMOS33 } [get_ports { sw[2] }]; #IO_L13N_T2_MRCC_16 Sch=sw[2]
#set_property -dict { PACKAGE_PIN A10    IOSTANDARD LVCMOS33 } [get_ports { sw[3] }]; #IO_L14P_T2_SRCC_16 Sch=sw[3]

## RGB LEDs
#set_property -dict { PACKAGE_PIN E1     IOSTANDARD LVCMOS33 } [get_ports { led0_b }]; #IO_L18N_T2_35 Sch=led0_b
#set_property -dict { PACKAGE_PIN F6     IOSTANDARD LVCMOS33 } [get_ports { led_rgb[1]}]; #IO_L19N_T3_VREF_35 Sch=led0_g
#set_property -dict { PACKAGE_PIN G6     IOSTANDARD LVCMOS33 } [get_ports { led0_r }]; #IO_L19P_T3_35 Sch=led0_r
#set_property -dict { PACKAGE_PIN G4     IOSTANDARD LVCMOS33 } [get_ports { led1_b }]; #IO_L20P_T3_35 Sch=led1_b
```

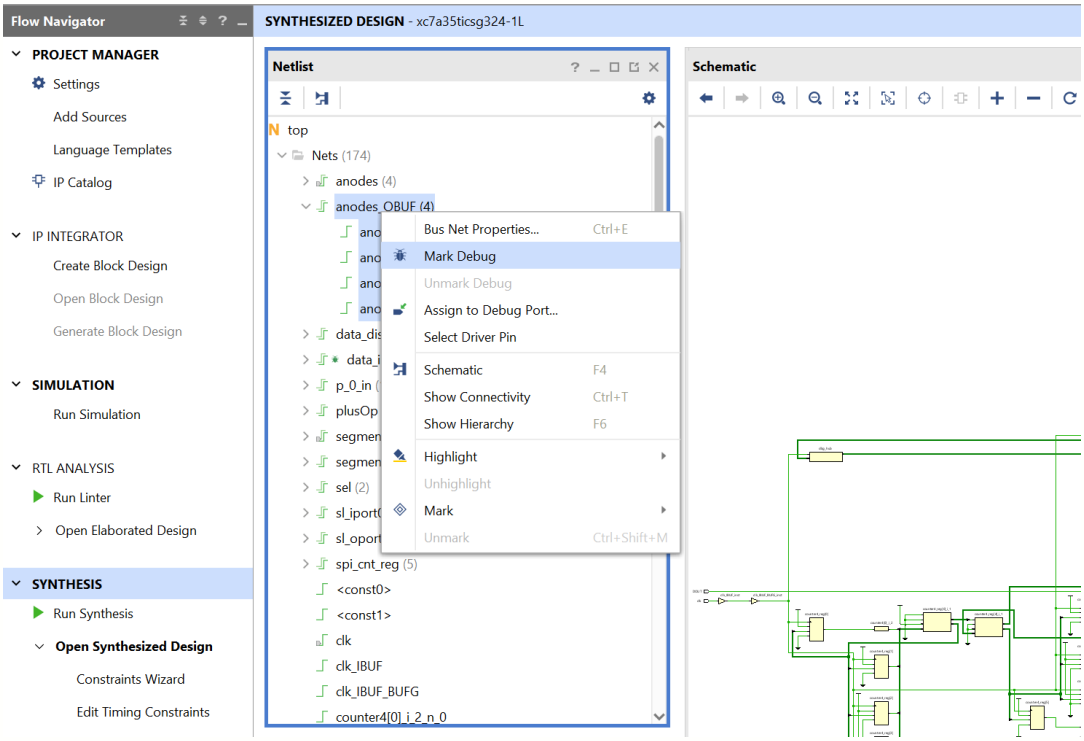
Lista atrybutów

Nazwa portu
(rozróżniana wielkość liter!)

Następnie przechodzimy do implementacji i generowania bitstreamu. W tym celu można bezpośrednio wybrać we Flow Navigatorze ***Generate Bitstream***, a implementacja zostanie wykonana samoczynnie.

Dygresja.

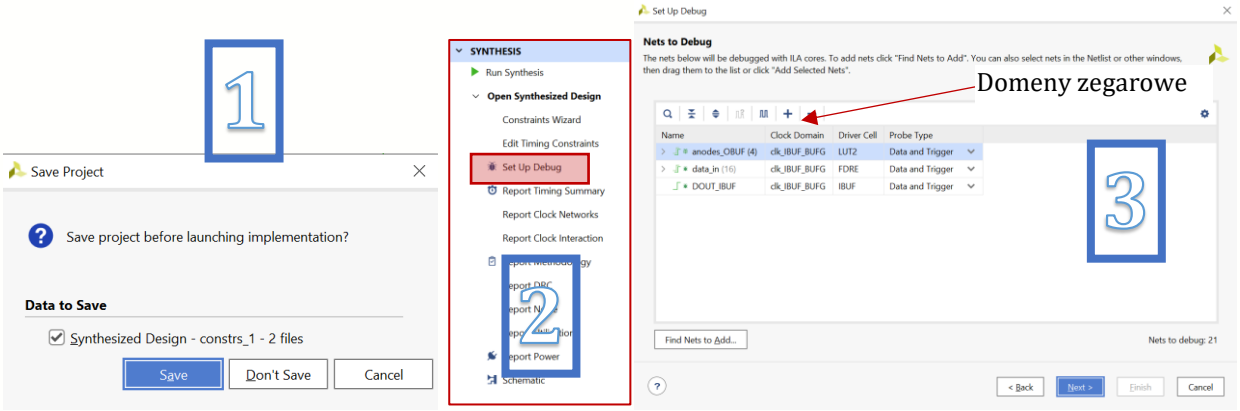
***Alternatywnie**, można również wybrać sygnały do debugowania. W tym celu należy wybrać w polu widoku **Open Synthesized Design**, z okna Netlist sygnały, które chcemy debugować.*



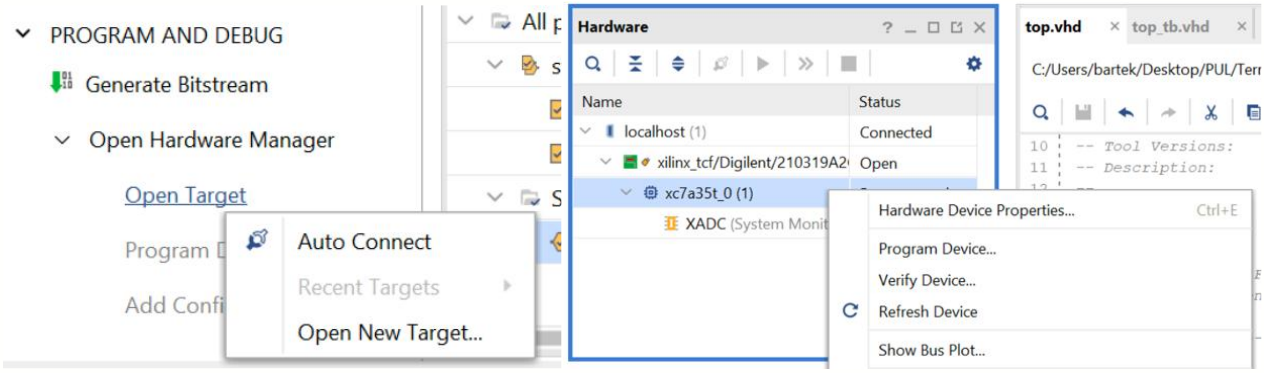
*Po wybraniu sygnałów do debugowania, przy próbie kontynuacji implementacji narzędzia Vivado wyświetlą monit (1) o zapisanie zmienionych plików (jako nowego pliku lub w uzupełnieniu istniejących plików constraints). Następnie, należy uruchomić opcję **Set Up Debug** (2). W oknie ustawień debugowania należy ustawić domeny zegarowe, do których należą sygnały (3).*

Koniec dygresji

Można teraz uruchomić implementację i generowanie pliku konfiguracyjnego (*bitstream*).



Zestaw prototypowy należy podłączyć do komputera za pomocą przewodu USB. Interfejs USB przeznaczony jest zarówno do komunikacji z wbudowanym na PCB programatorem JTAG oraz konwerterem USB - UART. Aby zaprogramować układ FPGA wygenerowanym bitstreamem przechodzimy do sekcji ***Program and debug -> Open Hardware Manager -> Open Target -> Auto Connect***. Na liście urządzeń w oknie głównym powinien po chwili pojawić się układ xc7a35t_0. Należy na niego kliknąć prawym przyciskiem myszy i wybrać ***Program Device***.



Uwaga.

Spora część kodu opisu sprzętu znaleziona w Internecie lub podpowiedziana przez SI (AI), dotyczy wyłącznie symulacji, nie kodu syntezowalnego! Ponadto, dotychczas korzystaliście z takiego właśnie (niesyntezowalnego) kodu podczas symulacji układów cyfrowych. Dobrze znana instrukcja `wait for` należy do tej grupy! Odmierzanie czasu realizujemy licznikami.

Do zrobienia.

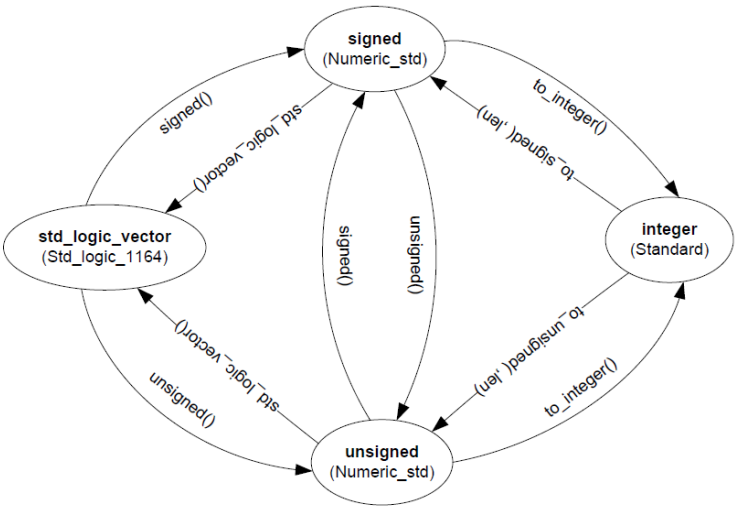
Układ sekwencyjny.

Po zapoznaniu się ze sposobem implementacji prostego układu kombinacyjnego w układzie programowalnym, pora spróbować czegoś więcej – podstawowych układów niezbędnych do prawidłowego funkcjonowania układów sekwencyjnych: liczników. Te proste urządzenia w naszych systemach służą do odmierzania czasu, wyznaczając pożądane jego odstępy pomiędzy etapami działania systemu (np. przejścia pomiędzy stanami w maszynach stanu).

Poniżej przedsatwiono szkielet projektu dla prostego licznika. Zadanie polega na napisaniu kodu licznika, którego efekt działania będzie można zaobserwować na diodach LED (miganie diod w takt zmian bitów licznika). Uwaga: na tym etapie nie należy stosować żadnych dodatkowych sygnałów sterujących typu ustawienie (*preset*), liczenie góra/dół.

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  -- Uncomment the following library declaration if using
5  -- arithmetic functions with Signed or Unsigned values
6  use IEEE.NUMERIC_STD.ALL;
7  -- Uncomment the following library declaration if instantiating
8  -- any Xilinx leaf cells in this code.
9  --library UNISIM;
10 --use UNISIM.VComponents.all;
11
12 entity licznik is
13     Port ( clk : in STD_LOGIC;
14            LED : out STD_LOGIC_VECTOR (3 downto 0));
15 end licznik;
16
17 architecture Behavioral of licznik is
18 signal licz_val : unsigned(3 downto 0):= (others => '0');
19 begin
20
21 process(clk)
22 begin
23 if rising_edge(clk) then
24     licz_val <= licz_val + 1;
25 end if;
26 end process;
27 LEDS <= std_logic_vector(licz_val);
28 end Behavioral;
```

Zwróćcie uwagę na typy danych. W definicji sygnału licznika mamy typ **unsigned**, dla którego dodawanie jest wspierane przez bibliotekę NUMERIC_STD. Jest to jedyne podejście „właściwe" według wymagań języka. Stosowanie innych blibliotek, które wspierają dodawanie dla typu **std_logic_vector** (np. *std_logic_arith*, *std_logic_unsigned*) może powodować przeładowanie operatora w nieprzewidywany sposób; biblioteki te są obecnie niewspierane (*legacy*). Stąd w przedostatniej linijce (27) konwersja sygnału na typ **std_logic_vector**, aby móc przypisać bity licznika na diody. Konwersja pomiędzy typami odbywa się za pomocą następującego grafu:



Udanej pracy. Powodzenia!