LABORATORIUM – ELEKTRONIKA
Projektowanie liczników asynchronicznych, koderów, transkoderów i dekoderów

1. Cel ćwiczenia

Ćwiczenie składa się z dwóch części.

W pierwszej części student projektuje i implementuje w strukturze układu FPGA (*Field Programmable Gate Array*) licznik asynchroniczny zliczający w zadanym przez prowadzącego zakresie.

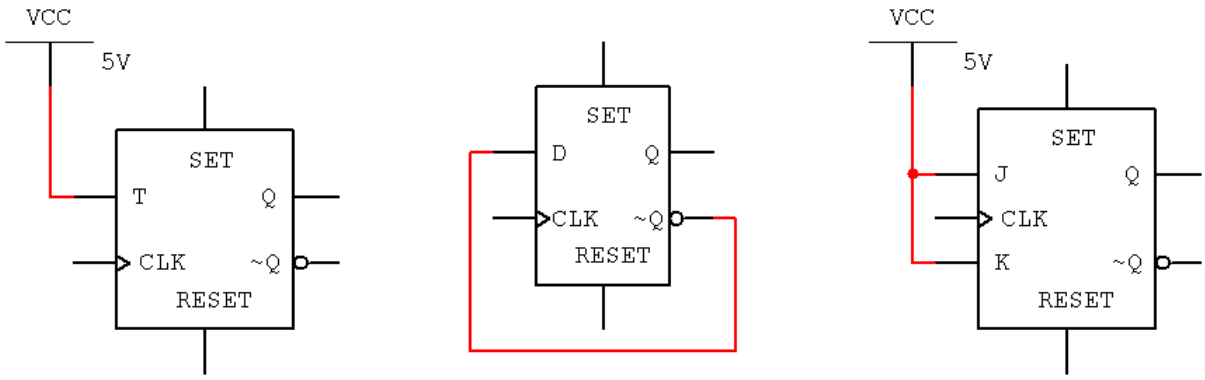
W drugeij części ćwiczenia student projektuje i implementuje w strukturze układu FPGA układ przetwarzający kod binarny na kod wyświetlacza siedmiosegmentowego. W tej części student poznaje też podstawy programowania układów w języku VHDL (*Very High Speed Integrated Circuits Hardware Description Language*).

2. Wstęp

Część 1 - Liczniki asynchroniczne

Fundamentem pozwalającym zbudować licznik asynchroniczny jest tzw. „dwójka licząca", która jest niczym innym jak odpowiednio podłączonym przerzutnikiem.

Dwójkę liczącą zrealizowaną na przerzutnikach typu: T, D i JK pokazano na rysunku poniżej.

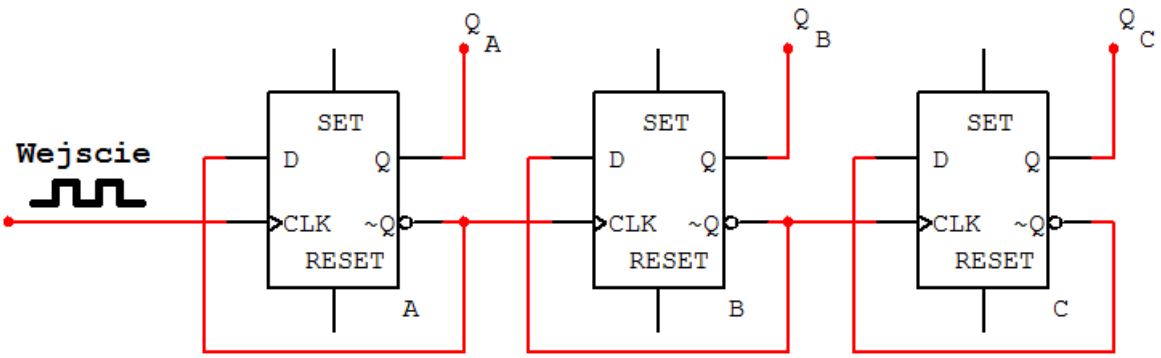


Łącząc tak podłączone przerzutniki w kaskady można realizować liczniki asynchroniczne zliczające od 0 do 2^n-1, gdzie n jest liczbą wykorzystanych przerzutników.

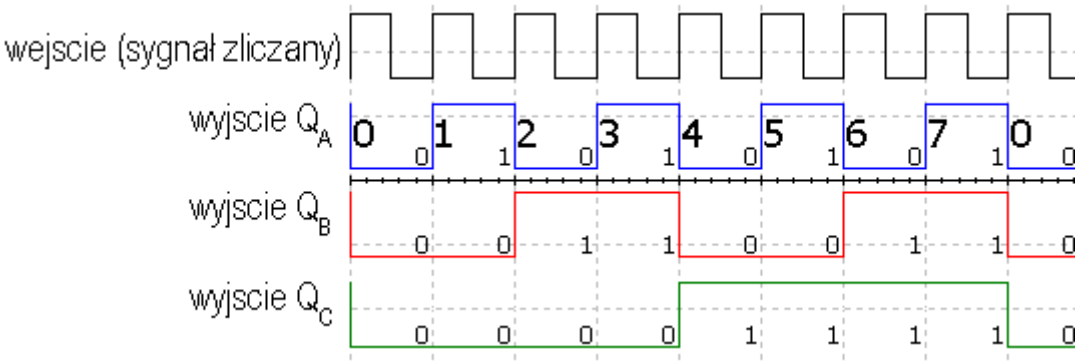
Łączenie w kaskadę sprowadza się do łączenia **wyjścia Q** poprzedzającego przerzutnika z wejściem zegarowym CLK przerzutnika po nim następującego, **w przypadku przerzutników typu JK** (najczęściej stosowanych) i **T**.

Dla przerzutnika typu D z wejściem zegarowym CLK kolejnego przerzutnika należy połączyć **wyjście ~Q** (czyli negacja Q) poprzedniego przerzutnika.

Przykładowo, Pokazany niżej układ jest zbudowanym **na przerzutnikach typu D** licznikiem asynchronicznym liczącym od 0 do 7 (czyli od 0 do 2^3-1, czyli od 000_2 do 111_2).



Gdyby pobrać przebiegi z wyjść przerzutników (będących wyjściami zaprojektowanego licznika), wyglądałyby one jak na rysunku poniżej, gdzie na niebiesko wykreślono przebieg z Q_A (najmłodszy bit licznika), na czerwono z Q_B i na zielono z Q_C (najstarszy bit licznika), na czarno oznaczono sygnał wejściowy, czyli zliczany.



Jak widać na pokazanych przebiegach, licznik po osiągnięciu maksimum (czyli w tym przypadku 7) wraca do stanu początkowego (czyli 0) i zlicza w takiej właśnie pętli.

Pętla ta jest zależna oczywiście od liczby wykorzystanych przerzutników. Gdyby było ich 4 zamiast 3, to zgodnie z informacjami z poprzedniej strony, licznik zamiast zlliczać do 7, zliczałby do 15 (bo $15 = 1111_2 = 2^4-1$) - taki licznik miałby oczywiście jedno wyjście (Q_D) więcej.

Warto też zwrócić uwagę na moment, w którym następuje zmiana na wyjściach licznika (= zmiana stanu licznika), jak widać następuje ona w chwili, gdy na wejściu pojawia się zbocze narastające kolejnej jedynki na wejściu.

Część 2 - Kodery, transkodery i dekodery

Przesyłanie, przetwarzanie i obróbka informacji w systemach cyfrowych jest wykonywana w oparciu o reprezentację odpowiednich zmiennych w postaci kodów liczbowych. Wybór kodu do zastosowania w danej aplikacji zależy od potrzeb, szczególnie chodzi o ułatwienie wykonywania operacji na zmiennych w tym układzie oraz zapewnienie ich prawidłowości przy występowaniu zakłóceń i uszkodzeń w układzie. Przy wyborze kodu umożliwiającego osiągnięcie minimum kosztów układu realizującego daną operację (np. arytmetyczną) należy również uwzględnić problemy konwersji tego kodu na inny/inne stosowane w tym systemie. Do tego celu stosuje się właśnie układy koderów, dekoderów i transkoderów, które to układy mają za zadanie przeliczyć dany kod binarny na inny.

Innym typowym zastosowaniem tego typu układów jest przetworzenie informacji w danym kodzie na kod który jest „czytelny z zewnętrz". Typowym układem tego typu jest US7447, który ma za zadanie przeliczać kod binarny na kod wyświetlacza siedmiosegmentowego. Innym znanym układem scalonym podobnego zastosowania jest US7442 przeliczający

kod binarny na kod 1 z 10. Układ projektowany w ramach laboratorium ma działać podobnie jak US 7447, czyli będzie on przetwarzał 4-bitowy kod binarny na wejściu na 7-bitowy kod, który po podaniu na wejścia wyświetlacza siedmiosegmentowego będzie wyświetlał liczby w pełnym zakresie 4 bitów, czyli od 0 do 15 (tzn. 0, 1, … , E, F).

3. Wykonanie ćwiczenia

3.1. Część 1 - Liczniki asynchroniczne

CZĘŚĆ A – projekt układu.

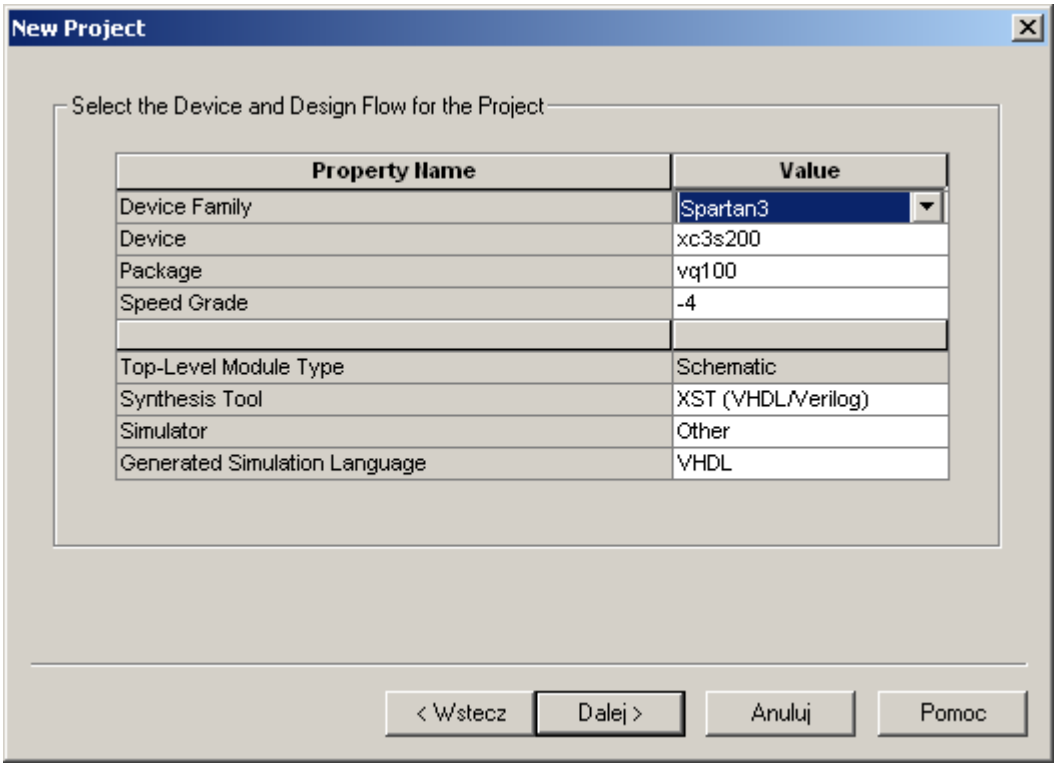
Zaprojektować (na papierze) zadany przez prowadzącego (Tabela 1 w protokole) licznik asynchroniczny na zadanym typie przerzutników, zgodnie z metodą opisaną we wstępie do instrukcji, i zamieścić go w protokole (na odwrocie).

CZĘŚĆ B – implementacja projektu w strukturze FPGA

(UWAGA: kolejne podpunkty należy realizować bardzo uważnie, ponieważ błędy popełnione na tym etapie mogą być później nie do cofnięcia i trzeba będzie zaczynać od początku).

Na podstawie stworzonego w punkcie 1 projektu:

- Uruchomić aplikację **Xilinx - Project Navigator**. Jest to aplikacja firmy Xilinx umożliwiająca projektowanie, symulację i implementację układów logicznych w strukturach układów FPGA.
- Stworzyć nowy projekt (***File -> New Project***). Nadać mu nazwę (**w nazwach NIE używać spacji**) (***Project Name***), wskazać lokalizację (można użyć domyślnej) i wybrać sposób realizacji projektu (***Top-Level Module Type***) – **Schematic**.
- W następnym oknie wybrać rodzaj programowanego układu logicznego. Powinien być to układ wykorzystywany na ćwiczeniu, zgodnie ze screenem (wszystkie widoczne informacje są dostępne na obudowie układu FPGA):



- W kolejnym oknie stworzyć nowe źródło (***New Source***) do projektu. Nadać mu nazwę (***File Name***) i wybrać typ – **Schematic**.
- Dwa kolejne okna można przeklikać bez zmian. Na tym kończy się wstępna konfiguracja projektu. Na ekranie powinno pojawić się okno projektowania układu. W tym oknie budujemy swój projekt.
- Najpierw należy wstawić potrzebne przerzutniki, z menu u góry wybrać pozycję **Add Symbol**, po lewej pojawi się lista dostępnych kategorii, stamtąd wybrać pozycję **Flip_Flop**, a wtedy na dole pojawi się lista dostępnych przerzutników, z których należy wybrać i wstawić do schematu te potrzebne do projektu.

  Będą to:

  - w przypadku przerzutników typu T – oznaczenie **FTC**
  - w przypadku przerzutników typu D – oznaczenie **FDC**
  - w przypadku przerzutników typu JK – oznaczenie **FJKC**

  Logiczne zero i jedynkę znajdziemy w podmenu **General** – jako **GND** (poziom zera) i **VCC** (poziom jedynki).

  Jeżeli dodatkowo potrzebne będą bramki, to można je znaleźć w podmenu **Logic** (najprawdopodobniej potrzebna będzie co najmniej jedna negacja (bramka NOT) oznaczona jako **INV**).
- Po wstawieniu wszystkich potrzebnych przerzutników, poziomu zera i jedynki oraz ewentualnie bramek, z menu należy wybrać pozycję **Add I/O Marker** i wstawić wejścia i wyjścia układu (w przypadku przykładowego projektu będą to 2 wejścia (**Add an input marker**) i trzy wyjścia (**Add an output marker**)).

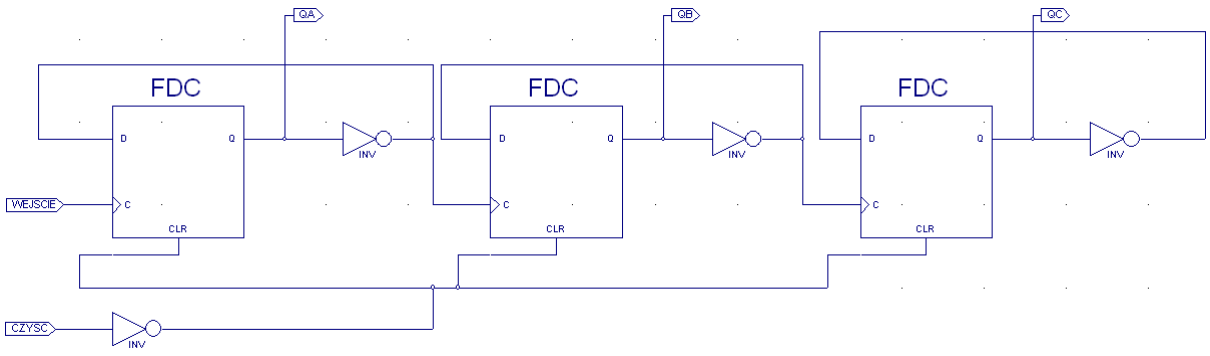
  UWAGA: Kolejność jest ważna, NAJPIERW należy wstawiać wejścia/wyjścia w odpowiedniej ilości, a dopiero potem wykonywać połączenia. Wejścia/wyjścia da się wstawiać tylko przy samych wejściach/wyjściach układów, dopiero potem można je przesuwać.

  W przypadku projektu realizowanego na laboratorium wejścia też będą dwa, liczba wyjść zależy od liczby bitów licznika (tożsamej z liczbą wykorzystywanych przerzutników).

  Nazwy wejść/wyjść można zmieniać dwuklikiem.

- Gdy wstawione będą już elementy, wejścia i wyjścia, można wykonać wszystkie potrzebne połączenia (**Add Wire**). Gotowy układ warto jeszcze sprawdzić pod względem formalnym (**Check Schematic**).

  Ostateczny wygląd projektu realizującego pokazany wcześniej licznik do 7 na przerzutnikach typu D pokazany jest poniżej. Jak łatwo zauważyć, w oprogramowaniu firmy Xilinx nie występują wyjście negowane z przerzutników i w związku z tym należy je samemu negować bramkami NOT.



Ponadto, wejścia zerujące przerzutników również zostały wyprowadzone na zewnątrz (przez negację). W ten sposób użytkownik będzie miał możliwość np. ręcznego (przyciskiem) wyzerowania układu.

- Następnym krokiem jest przypisanie wejść i wyjść wykonanego układu do nóżek układu scalonego. W tym celu należy wrócić do okna **Xilinx - Project Nawigator** i w oknie po lewej stronie dwukliknąć w opcję **Assign Package Pins**. Powinno otworzyć się okno Xilinx PACE i topologia wykorzystywanego układu FPGA. Po lewej stronie (**Design Object List**) widać wcześniej zaprojektowane wejścia/wyjścia układu.

  Nóżki układu przypisać zgodnie z załączoną notą wykorzystywanego w ćwiczeniu zestawu ZL6PLD.
  Jako wyjścia ustawić wybrane diody LED (strona 5 noty) – najlepiej po kolei, czyli niech dioda D1 będzie QA, dioda D2 - QB itd.
  Jako wejście zerujące (CZYSC) wykorzystać wybrany przycisk klawiatury (strona 6 noty).
  Jako wejście zegarowe (WEJSCIE) wykorzystać pin 14.

  Piny najłatwiej przypisać wpisując tam bezpośrednio nóżkę układu scalonego, której chcemy przypisać daną funkcję, przykładowo, chcąc przypisać nóżce 14 układu scalonego funkcję wejścia zegarowego należy w kolumnie **Loc** wpisać (z klawiatury) P14 w odpowiednim wierszu (jak na rysunku poniżej). Tak samo należy postąpić przypisując pozostałe sygnały do odpowiednich pinów.

	I/O Name	I/O Direction	Loc	Bank	I/O Std.
	WEJSCIE	Input	P14	BANK	
	QC	Output			
	QB	Output			
	QA	Output			
	CZYSC	Input			

- Po przypisaniu pinów można przystąpić do implementacji projektu w strukturze FPGA.
  W tym celu należy kolejno:
  - w oknie „Project Navigator” kliknąć w plik nadrzędny projektu;
  - w menu poniżej, na liście „Implement Design” znajduje się pozycja „Configure Device”, dwukliknąć w nią LPM;
  - kolejne okna przeklikać z domyślnymi ustawieniami;
  - jeśli automatycznie nie otworzy się okno wyboru pliku, dwukliknąć LPM w symbol układu scalonego i wybrać plik z rozszerzeniem *.jed lub *.bit, który powinien mieć nazwę jak plik nadrzędny projektu;
  - następnie kliknąć PPM w symbol układu scalonego i wybrać „Program…”,
  - po odczekaniu na implementację pliku w strukturze programowalnej, należy zgłosić prowadzącemu gotowość do testów.

- **Jeżeli licznik działa, należy przejść do drugiej części ćwiczenia.**

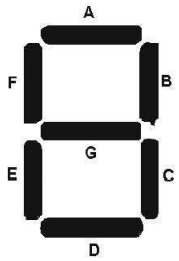
3.2. Część 2 - Kodery, transkodery i dekodery

CZĘŚĆ A – TABELA STANÓW

- Wypełnić Tabelę 2 w protokole. Jest to tabela stanów projektowanego transkodera, czyli tabela zawierająca wszystkie możliwe kombinacje wejść (od 0000 do 1111, innymi słowy binarnie policzone od 0 do 15) i odpowiadające im stany wyjść. Wykorzystywany na laboratorium wyświetlacz działa w taki sposób, że zapala segment na który podana jest logiczna jedynka.

  A zatem, przykładowo:

  - pierwszy wiersz tabeli odpowiada kombinacji wejść 0000, czyli po prostu dziesiętnemu 0,
  - by wyświetlić na wyświetlaczu 0, należy zaświecić wszystkie segmenty poza G,
  - zatem na wyjścia należy podać kombinację: 0111111 (G=0, F=1, E=1, itd.).



Analogicznie należy wypełnić pozostałe wiersze tabeli stanów – z uwagi na fakt, że układ 4-bitowy ma 16 kombinacji, tabeli nie kończy wyświetlenie 9, po 9 możnal zliczać w kodzie szesnastkowym, czyli 8, 9, A, b, itd.

Po wykonaniu tej czynności można przejść do CZĘŚCI B ćwiczenia.

CZĘŚĆ B – implementacja projektu w strukturze FPGA

- W aplikacji **Xilinx - Project Navigator** stworzyć nowy projekt (***File -> New Project***). Nadać mu nazwę (**w nazwach NIE używać spacji**), wskazać lokalizację (można użyć domyślnej) i wybrać sposób realizacji projektu (***Top-Level Module Type***) – **HDL**.
- W następnym oknie wybrać rodzaj programowanego układu logicznego, taki sam jak w pierwszej części ćwiczenia
- W kolejnym oknie stworzyć nowe źródło (***New Source***) do projektu. Nadać mu nazwę i wybrać typ – **VHDL Module**.
- Kolejne okno służy do definiowania wejść i wyjść układu. Projektowany układ powinien zawierać jeden 4 bitowy wektor wejść (bo kod wejściowy naszego transkodera jest 4-bitowym kodem binarnym), deklarowany następująco:

Port Name	Direction	MSB	LSB
wejscia	in	3	0

Operacja jak widać sprowadza się do wpisania nazwy sygnału w kolumnie ***Port Name***, wskazania, czy definiujemy wejście (***in***) czy wyjście (***out***) i zadeklarowania wielkości wektora (w tym przypadku od 0 do 3, więc razem 4).

Analogicznie do wektora wejść należy zadeklarować 7 bitowy wektor wyjść (bo wyświetlacz ma 7 segmentów) i jeden bit (także wyjściowy) do zasilenia wyświetlacza. Przy deklarowaniu tego ostatniego, jako, że nie jest to wektor ale sygnał, kolumny MSB (*Most Significant Bit*) i LSB (*Least Significant Bit*) powinny pozostać puste.

- Kolejne okna przeklikujemy, zgadzając się na ewentualne pytania programu. Ostatecznie powinien się nam ukazać automatycznie utworzony szkielet programu, wyglądający mniej więcej tak:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--  Uncomment the following lines to use the declarations that are
--  provided for instantiating Xilinx primitive components.
--library UNISIM;
--use UNISIM.VComponents.all;

entity transkoder is
   Port ( wejscia : in std_logic_vector(3 downto 0);
          wyjscia : out std_logic_vector(6 downto 0);
          zasil : out std_logic);
end transkoder;

architecture Behavioral of transkoder is
-- (1) TUTAJ DEKLARUJE SIĘ SYGNAŁY WEWNĘTRZNE
begin
-- (2) TUTAJ TWORZY SIĘ WŁAŚCIWY PROGRAM
end Behavioral;
```

Jak widać, w programie pojawiły się (zaznaczone na niebiesko) deklarowane wcześniej wejścia i wyjścia układu. We wskazanych na czerwono miejscach tworzy się właściwy program, czyli **(1)** deklaruje ewentualne sygnały działające jedynie wewnątrz programu (tzn. niewyprowadzane na zewnątrz) oraz **(2)** określa jego działanie.

Jeżeli o czymś zapomnieliśmy na wcześniejszym etapie i w niebieskim fragmencie są braki lub błędy, można je poprawić ręcznie – nie trzeba wracać do konfiguracji projektu.

- Przystąpmy więc do tworzenia właściwego programu. W naszym transkoderze nie będzie sygnałów działających jedynie wewnątrz programu, więc nasze działania będą się toczyć jedynie we fragmencie kodu **(2)**.

Opis działania dekodera, możliwy do wykonania na kilka sposobów, sprowadza się do wskazania mu jaką kombinację wyjść ma wystawiać w zależności od kombinacji wejść. Takie zachowanie można mu narzucić na przykład przypisaniem rozpoczynającym się następująco:

```
wyjscia <=
"0111111" when (wejscia = "0000") else
```

która to kombinacja, zgodnie z wcześniejszym przykładem, powinna na wyświetlaczu siedmiosegmentowym pokazywać 0 (tj. wygaszać jedynie segment G) dla kombinacji wejść 0000.

W kolejnych wierszach wpisujemy pozostałe 15 zależności wyjść od wejść.

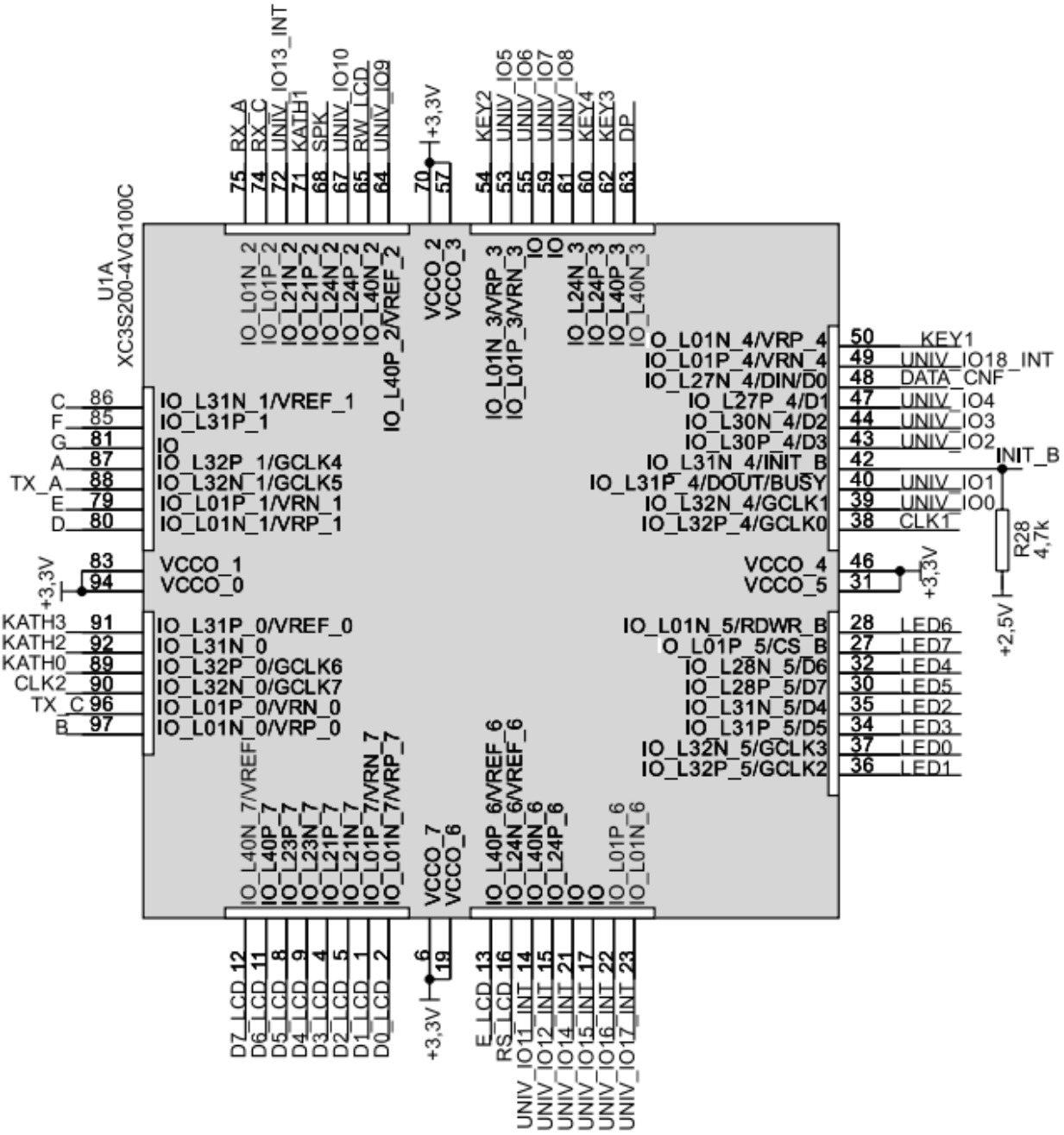
Ostatniego wiersza nie kończymy słowem „else” lecz średnikiem.

Ostatnim co pozostało do zrobienia, jest zasilenie wyświetlacza. W tym celu należy wystawić stan wysoki (logiczną jedynkę) na bit zasilający. Wystarczy do tego prosta instrukcja (słowo „zasil” oczywiście jest nadawaną wcześniej nazwą bitu zasilającego wyświetlacz):

```
zasil <= '1';
```

- I to jest koniec właściwego programu. Kolejnym krokiem jest przypisanie wejść i wyjść wykonanego układu do nóżek układu scalonego. W tym celu należy w oknie po lewej stronie dwukliknąć w opcję **Assign Package Pins**. Jeżeli w programie nie ma błędów, to powinno otworzyć się okno Xilinx PACE i topologia wykorzystywanego układu FPGA. Po lewej stronie (**Design Object List**) widać wcześniej deklarowane wejścia/wyjścia układu.

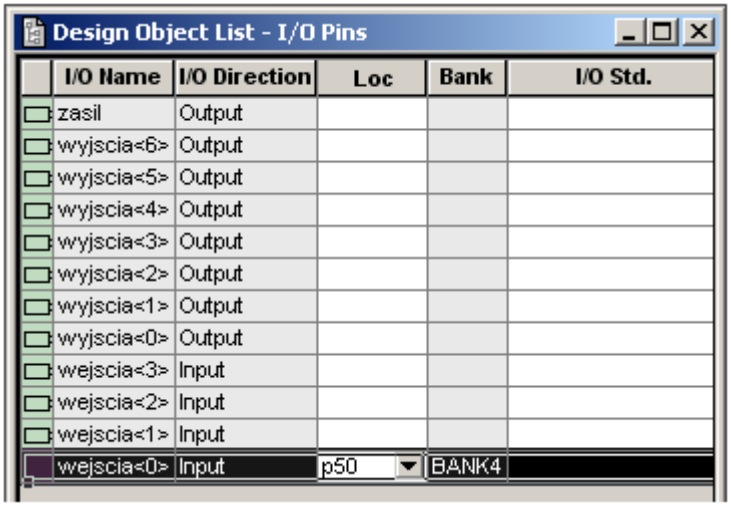
  Wykorzystywane piny układu przypisać zgodnie z rysunkiem poniżej i opisem na następnej stronie:



Oznaczenia są następujące:

- wejścia - przyciski – KEY4, KEY3, KEY 2, KEY1 (gdzie: wejścia<3> to KEY4, wejścia<2> to KEY3 itd.),
- wyjścia – G, F, E, D, C, B, A (gdzie: wyjscia<6> to G, wyjscia<5> to F itd.),
- pin zasilający wyświetlacz – dowolny z pinów KATH0, KATH 2 lub KATH3.

Piny przypisuje się wpisując bezpośrednio nóżkę układu scalonego, której chcemy przypisać daną funkcję. Przykładowo: chcąc przypisać najmniej znaczący bit wejściowy (wejścia<0>) do przycisku KEY1, na rysunku powyżej sprawdzamy, że KEY1 jest podpięty do nóżki (pinu) 50 układu FPGA, więc w odpowiednim miejscu wpisujemy p50, jak na rysunku poniżej, następnie analogicznie przypisujemy wszystkie pozostałe piny.



3.3. Po przypisaniu pinów można przystąpić do implementacji projektu w strukturze FPGA i sprawdzenia poprawności działania transkodera.

3.4. W tym celu należy kolejno:

- w oknie „Project Navigator" kliknąć w plik nadrzędny projektu;
- w menu poniżej, na liście „Implement Design" znajduje się pozycja „Configure Device", dwukliknąć w nią LPM;
- kolejne okna przeklikać z domyślnymi ustawieniami;
- jeśli automatycznie nie otworzy się okno wyboru pliku, dwukliknąć LPM w symbol układu scalonego i wybrać plik z rozszerzeniem *.jed lub *.bit, który powinien mieć nazwę jak plik nadrzędny projektu;
- następnie kliknąć PPM w symbol układu scalonego i wybrać „Program…",
- po odczekaniu na implementację pliku w strukturze programowalnej, należy sprawdzić działanie układu, czyli…
- **wciskając przyciski S1, S2, S3 i S4 sprawdzać jak zachowuje się wyświetlacz, tj. czy wyświetla on znaki zgodnie z tabelą stanów zaprojektowaną na początku ćwiczenia.**

4. Literatura

[1] Borzdyński J.: *Kurs CPLD*, Elektronika dla Wszystkich 02/2009-04/2009

[2] Głocki W., Grabowski L., „Pracownia podstaw techniki cyfrowej", WSiP, Warszawa 1998

[3] Pawluczuk A.: *Układy programowalne dla początkujących*, btc, Legionowo 2010.

[4] Skahill K.: *Język VHDL - projektowanie programowalnych układów logicznych*, WN-T, Warszawa 2004

[5] Zwoliński M.: *Projektowanie układów cyfrowych z wykorzystaniem języka VHDL*, WKiŁ, Warszawa 2007

Opracowanie ćwiczenia: Seweryn Lipiński