LABORATORIUM – OPTOELEKTRONIKA

Oprogramowanie bariery podczerwieni w układzie CPLD

1. Wstęp i cel ćwiczenia

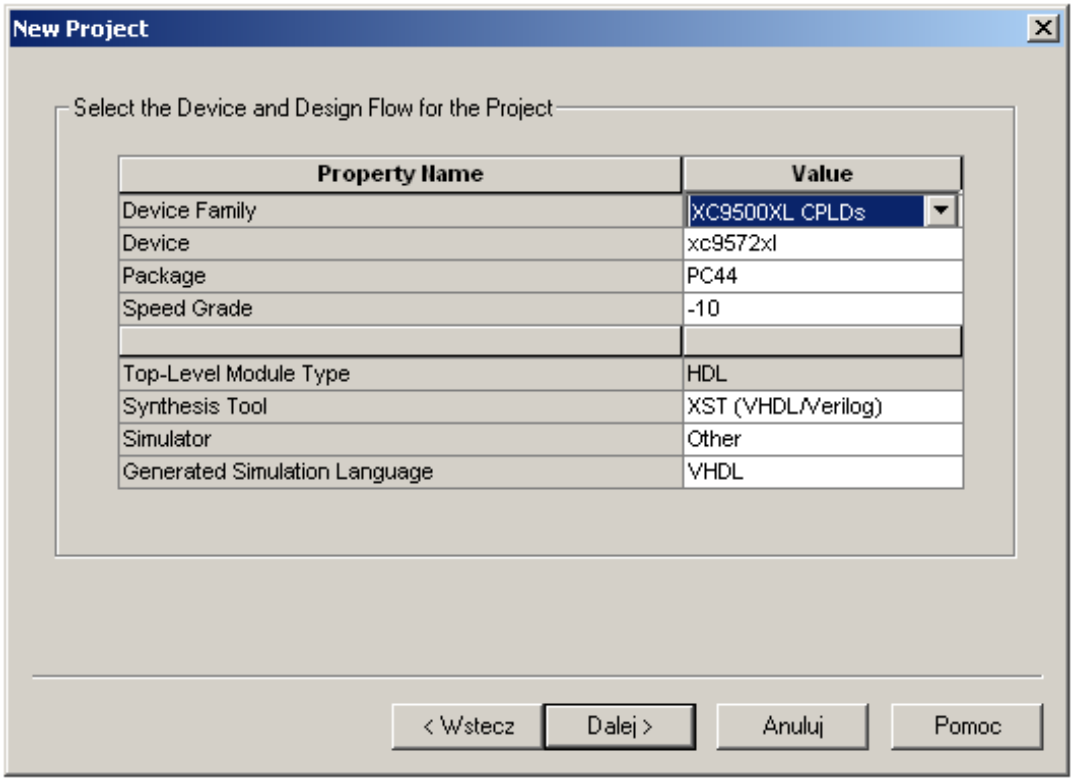
W ćwiczeniu student tworzy barierę podczerwieni złożoną z diody nadawczej IR (*Infra Red*) i odbiornika podczerwieni, w ten sposób poznając zasadę działania tego typu układów. Sterowanie diodą nadawczą, odbiór informacji z odbiornika i sygnalizacja stanu bariery opiera się o układ CPLD (*Complex Programmable Logic Device*), dzięki czemu w trakcie ćwiczenia student uczy się też podstaw programowania układów programowalnych w języku VHDL (*Very High Speed Integrated Circuits Hardware Description Language*).

2. Wykonanie ćwiczenia.

2.1. Uruchomić aplikację **Xilinx - Project Navigator**. Jest to aplikacja firmy Xilinx umożliwiająca projektowanie, symulację i implementację układów logicznych w strukturach układów CPLD i FPGA.

2.2. 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***) – **HDL**.

2.3. 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 CPLD):



2.4. W kolejnym oknie stworzyć nowe źródło (***New Source***) do projektu. Nadać mu nazwę (***File Name***) i wybrać typ – **VHDL Module**.

2.5. Kolejne okno służy do definiowania wejść i wyjść układu. Projektowany układ powinien zawierać 2 wejścia (wejście z odbiornika podczerwieni i wejście z generatora częstotliwości, który posłuży do synchronizacji działania układu, w tym dopasowanie częstotliwości diody nadawczej do częstotliwości, na którą czuły jest odbiornik) i dwa wyjścia (wyjście do sterowania diodą nadawczą i wyjście, które posłuży do sygnalizacji stanu bariery). Operacja sprowadza się do wpisania nazwy sygnału w kolumnie ***Port Name*** i wskazania, czy definiujemy wejście (***in***) czy wyjście (***out***).

2.6. Kolejne okna przeklikujemy, zgadzając się na ewentualne pytania programu. Ostatecznie powinien się nam ukazać automatycznie utworzony szkielet programu, wyglądający na przykład 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 barieraIR is
   Port ( ODBIORNIK : in std_logic;
          GENERATOR : in std_logic;
          DIODA_IR : out std_logic;
          DIODA_LED : out std_logic);
end barieraIR;

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

2.6.1. 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 zewnętrz) oraz (2) określa jego działanie.

2.6.2. Przystąpmy więc do tworzenia właściwego programu. Na początek we fragmencie kodu oznaczonym jako **(1)** zadeklarujmy dwa sygnały o różnych częstotliwościach, które za chwilę utworzymy:

- ✓ signal SYGNAL36kHz : std_logic ;
- ✓ signal SYGNAL500Hz : std_logic ;

Do czego one nam posłużą?

Sygnał pierwszy ma częstotliwość 36 kHz, ponieważ na taką częstotliwość reaguje odbiornik podczerwieni.

Drugi sygnał (o częstotliwości 500 Hz) posłuży do modulacji tego sygnału.

Sygnał nadawany z diody IR musi być zmodulowany inną (mniejszą) częstotliwością, by odbiornik na niego reagował. Ciągłe nadawanie sygnału o częstotliwości 36 kHz uniewrażliwiłoby odbiornik. Wybór 500 Hz jest arbitralny, równie dobrze mogłaby być to inna częstotliwość.

2.7. Przejdźmy teraz do fragmentu kodu **(2)**. Na początek należy wygenerować sygnały o wyżej deklarowanych częstotliwościach. Działanie to zależy od dostępnego generatora częstotliwości wzorcowej. Generator podłączony do układu CPLD ma częstotliwość 24MHz (kto nie wierzy, może zweryfikować tę informację na płytce zestawu uruchomieniowego). A zatem, by utworzyć sygnał o częstotliwości 36 kHz należy podzielić tę częstotliwość przez ~333, bo 24MHz/36kHz = 666,(6), którą to wartość dzielimy jeszcze przez 2, tak, by dwa półokresy (zero i jeden) przypadły na jeden takt zegara. Cały proces generujący sygnał 36 kHz w VHDL będzie wyglądał tak:

```
-- generator 36kHz
 generator1: process (GENERATOR) is
  variable temp1 :  unsigned (8 downto 0);
 begin
  if rising_edge (GENERATOR) then
   temp1 := temp1+1 ;
   if temp1 = 333 then
    temp1 := "000000000";
    SYGNAL36kHz <= not SYGNAL36kHz;
   end if;
  end if;
 end process generator1;
```

Nie wnikając głębiej w składnię języka VHDL widać, że program kolejno:

- tworzy 9 bitową (8 downto 0) zmienną o nazwie temp1,
- jeśli wykrywa zbocze narastające na generatorze zwiększa tę zmienną o 1,
- jeśli wykrywa, że zmienna temp 1 jest równa 333, neguje ją.

W ten sposób co 333 takty zegara zmienia się logiczna wartość sygnału o nazwie SYGNAL36kHz, a zatem co 666 taktów jego okres, więc uzyskaliśmy sygnał o częstotliwości bliskiej 36 kHz.

Ostatnią wątpliwością może być: dlaczego zmienna temp1 jest akurat 9 bitowa?
Jest tak dlatego, że by binarnie zapisać maksymalną wartość przewidzianą dla niej, potrzebujemy 9 bitów, bowiem 333 dziesiątkowo to binarnie 101001101.

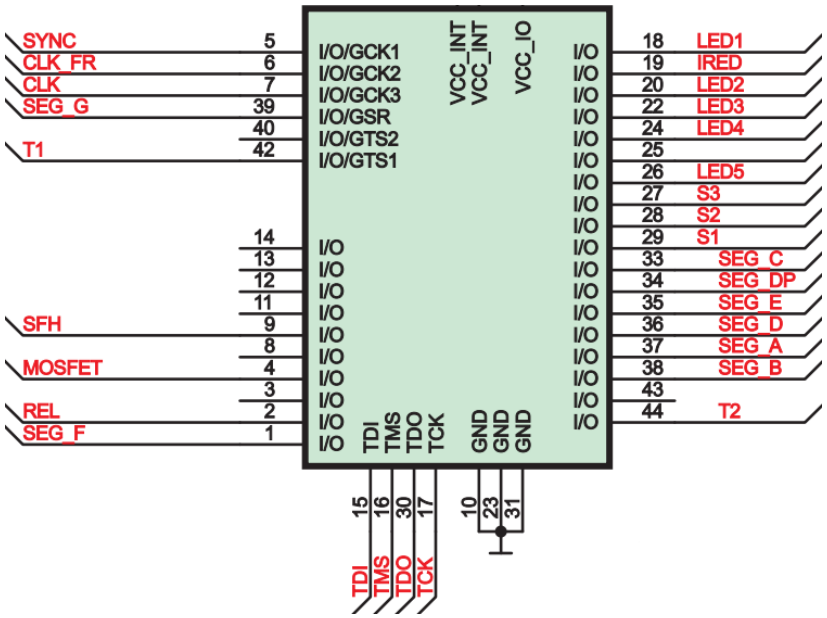
2.8. Kolejnym krokiem jest (na własną rękę) stworzenie generatora 500 Hz, analogicznie jak w poprzednim podpunkcie.

2.9. Mając dostępne dwa generatory pozostało tylko zdefiniować zachowanie bariery, czyli…

- Wygenerować zmodulowany sygnał na diodzie IR, do czego wystarczy prosta instrukcja:
  DIODA_IR <= SYGNAL36kHz and SYGNAL500Hz ;
  sprowadzająca się do wykonania operacji AND na obu wcześniej wygenerowanych sygnałach.
- Zaświecić diodę LED jeśli bariera nie jest przerwana, jeszcze prostszą instrukcją:
  DIODA_LED <= ODBIORNIK ;
  sprowadzającą się do powtórzenia sygnału z odbiornika na diodzie LED.

2.9.1. I to jest koniec właściwego programu. Kolejnym krokiem jest przypisanie wejść i wyjść wykonanego układu do nóżek układu scalonego. W tym celu, najlepiej uprzednio zapisawszy projekt, należy w oknie po lewej stronie dwukliknąć w opcję **Assign Package Pins**. Jeżeli projekt nie zawiera błędów, to otworzy się okno Xilinx PACE wraz z topologią wykorzystywanego układu CPLD. Po lewej stronie (**Design Object List**) widać wcześniej zaprojektowane wejścia/wyjścia układu.

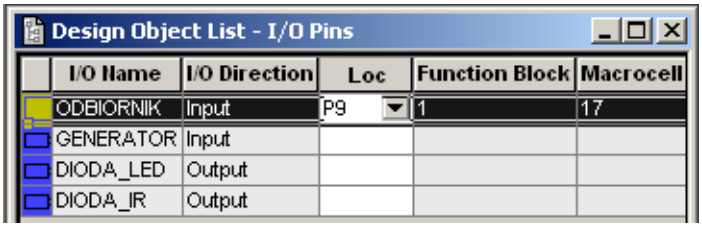
Cztery wykorzystywane piny układu przypisać zgodnie z rysunkiem poniżej:



Oznaczenia są następujące:

- odbiornik podczerwieni – SFH,
- generator 24MHz – SYNC,
- dioda nadawcza podczerwieni – IRED,
- dioda LED – dowolna z diod LED1 – LED 5.

Piny przypisuje się wpisując bezpośrednio nóżkę układu scalonego, której chcemy przypisać daną funkcję. Przykładowo: chcąc odpowiednio przypisać sygnał z odbiornika podczerwieni, na rysunku sprawdzamy, że sygnał SFH jest podpięty do nóżki (pinu) 9 układu CPLD, więc w odpowiednim miejscu wpisujemy:



2.10. Po przypisaniu pinów można przystąpić do implementacji projektu w strukturze CPLD i sprawdzenia poprawności działania bariery. W tym celu należy zgłosić prowadzącemu gotowość do zaprogramowania układu.

2.11. Jeżeli projekt działa (tj. stan bariery IR przekłada się bezpośrednio na stan diody LED), to rozbudować projekt następująco:

- niech stan bariery wyświetla się nie tylko na diodzie, ale też na wyświetlaczu siedmiosegmentowym, jako I/U (interrupted/uninterrupted) – **wykonanie obowiązkowe**;
- niech stan bariery po przerwaniu zostaje zachowany i wraca do stanu nieprzerwania dopiero po naciśnięciu przycisku S1 (przyciski na płytce po naciśnięciu dają stan niski) – **wykonanie nieobowiązkowe – na plusa**;

3. Wskazówki

1. Przy rozbudowywaniu układu o kolejne wejścia wyjścia można je deklarować "z palca", rozszerzając program w "niebieskim" fragmencie kodu. Zadeklarowanie kolejnej diody jako wyjścia lub przycisku jako wejścia będzie miało taką samą składnię jak np. deklaracja już istniejącej diody, trzeba tylko pamiętać o tym, czy deklaruje się wejście czy wyjście.

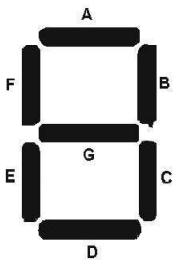
Różnica pojawia się przy wyświetlaczu siedmiosegmentowym, który można oczywiście deklarować segment po segmencie bit po bicie, ale wygodniej jest zrobić to hurtem, tzn. jako wektor, np.:

```
wysw7seg : out std_logic_vector (6 downto 0);
```

Uwaga! Przy wyświetlaczu należy zadeklarować nie tylko wyświetlacz jako taki, ale też bit go zasilający.

Oznaczenia dodatkowych wejść/wyjść (do etapu Assign Package Pins) są następujące:

- zasilanie wyświetlacza 7-segmentowego – T1 lub T2 (zasilenie następuje przez podanie 0),
- wyświetlacz 7-segmentowy – zgodnie z nazwami poszczególnych segmentów, tzn. SEG_A, SEG_B, itd.,



- przyciski – S1, S2 i S3 (wciśnięcie przycisku zadaje stan niski, tj. 0).

2. Przykładowa składnia "when" w VHDL:

```
wyjscie <= "00" when (wejscie = '0') else "11" when (wejscie = '1');
```

W powyższym przykładzie warto zwrócić uwagę na fakt, że jeżeli odwołujemy się do bitu, to jego wartość wstawia się między apostrofy, a jeżeli do wektora, to jego wartość wstawia się w cudzysłów.

Dodatkowo przydać może się wiedza, że warunki wielokrotne buduje się słowami kluczowymi odpowiadającymi nazwom bramek logicznych, np.

```
wyjscie <= "00" when (wejscie1 = '0') or (wejscie2='0') else "11" when (wejscie1 = '1');
```

Kolejnych warunków ("else") może być oczywiście dowolnie dużo. W przypadku gdy ostatnie else obejmuje wszystkie pozostałe sytuacje, składnię można zakończyć sformułowaniem "... when others".

4. Literatura

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

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

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

Opracowanie ćwiczenia: Seweryn Lipiński