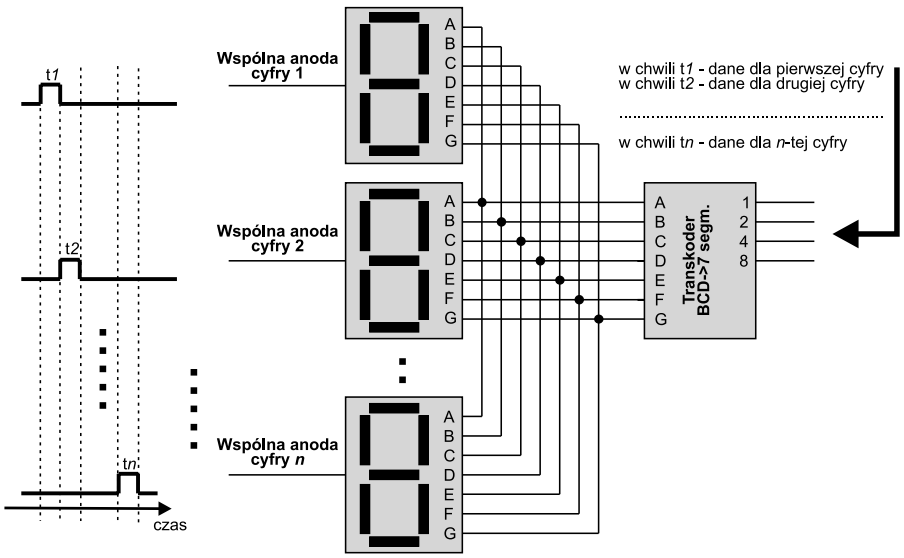
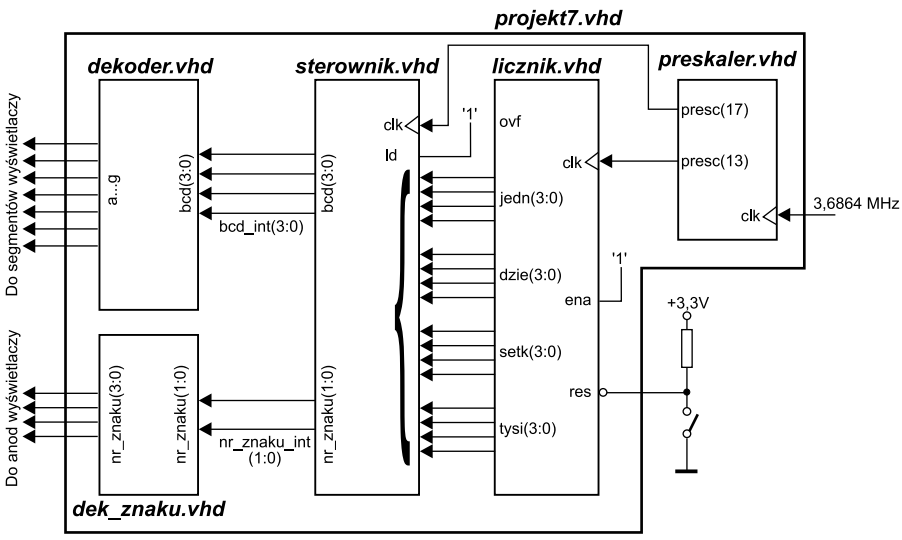
# Układy FPGA w przykładach, część 7
## Projekty przykładowe



*W tym odcinku kursu pokażemy przykładowy projekt sterownika 4–cyfrowego, 7–segmentowego wyświetlacza LED. Jest to kolejny typowy element systemu cyfrowego, często wykorzystywany w praktycznych aplikacjach, w których często stanowi istotny element interfejsu użytkownika.*




**Rys. 1. Schemat ilustrujący zasadę działania sterownika multipleksowego**




**Rys. 2. Schemat blokowy projektu zaimplementowanego w FPGA**

Idea multipleksowego sterowania wyświetlaczy jest stosunkowo stara, bowiem tego typu sposoby sterowania stosowano już w latach ’60 ubiegłego wieku w systemach z wyświetlaczami Nixie. Podstawową zaletą takiego systemu sterowania wyświetlaczy jest minimalizacja liczby wyprowadzeń układu sterującego. Przykładowo, w przypadku pola 4–cyfrowego sterowanego w sposób standardowy, konieczne jest co najmniej 28 linii sterujących (4 cyfry x 7 segmentów). W prezentowanym przykładzie sterownika multipleksowego wystarczy 11 linii sterujących (7 do sterowania segmentami + 4 do sterowania elektrodami wspólnymi), a jeszcze lepsze są osiągane przy zespołach wyświetlaczy o większej liczbie cyfr.

Ideę wyświetlania multipleksowego zilustrowano na **rys. 1** (pokazany sposób sterowania jest charakterystyczny dla wyświetlaczy ze wspólną anodą). Jak widać, konieczne jest zsynchronizowanie „włączania” anod z podaniem odpowiedniej dla wyświetlanego znaku kombinacji stanów na katody diod podświetlających segmenty. Sekwencyjne zapalanie każdej cyfry z odpowiednio dużą częstotliwością (w przykładzie ok. 100 Hz) wywołuje wrażenie ciągłego świecenia wszystkich znaków.

Zapraszamy do współpracy!
Wszystkich Czytelników interesujących się układami FPGA zachęcamy do prezentacji na łamach EP własnych rozwiązań. Odpowiemy także na wszelkie pytania związane z kursem, językiem VHDL i układami FPGA.

# Konfigurowanie FPGA i programowanie konfiguratora Flash

## Przygotowywanie plików do konfiguracji układu FPGA i programowania konfiguratora Flash

Z listów otrzymanych od Czytelników wynika, że wielu z nich ma spore problemy z prawidłowym skonfigurowaniem układu FPGA i zaprogramowaniem pamięci Flash spełniającej w zestawie ZL10PLD rolę konfiguratora.

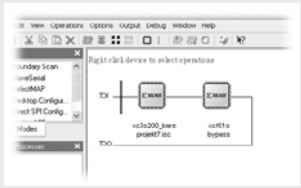
Pokażemy zatem krok po kroku jak sobie poradzić z podstawowymi problemami występującymi na tym etapie implementacji projektów, które w większości przypadków wynikają ze stosunkowo mało zrozumiałego (przynajmniej na początku pracy z FPGA) rozdzielenia programowania (konfiguratora Flash) od konfigurowania (układu FPGA).

## Krótko od podstaw

Jak wspomnieliśmy w pierwszym odcinku kursu (EP10/2006), układy FPGA z rodziny Spartan 3 mają wewnętrzną pamięć konfiguracji typu SRAM, co wymaga każdorazowo po włączeniu zasilania układu odtworzenia jej zawartości z zewnętrznej pamięci nieulotnej (zazwyczaj, także w przypadku kursowego zestawu ZL10PLD – jest to pamięć Flash).

Konieczne jest więc zapewnienie możliwości:
- bezpośredniego modyfikowania zawartości pamięci konfiguracji FPGA (SRAM),
- programowania pamięci Flash po zainstalowaniu jej w systemie,
- skopiowania zawartości pamięci Flash do SRAM (pamięci konfiguracji FPGA).

Firma Xilinx w układach Spartan 3 problem rozwiązała w sposób pokazany



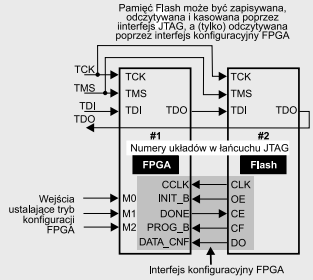
**Automatycznie wykryta przez program iMPACT konfiguracja łańcucha JTAG w zestawie ZL10PLD**

na poniższym rysunku:

Konfigurator XCF01S jest wyposażony w dwa interfejsy umożliwiające wymianę danych z otoczeniem:
- 5–liniowy, służący do przesyłania przechowywanych do FPGA po włączeniu zasilania,
- JTAG, który służy do programowania pamięci Flash za pomocą zewnętrznego programatora.

Także w układzie XC3S200 (FPGA z rodziny Spartan 3, zastosowany w module ZL10PLD) występuje interfejs umożliwiający współpracę z konfiguratorem oraz interfejs JTAG, za pomocą którego jest konfigurowana pamięć SRAM w FPGA.

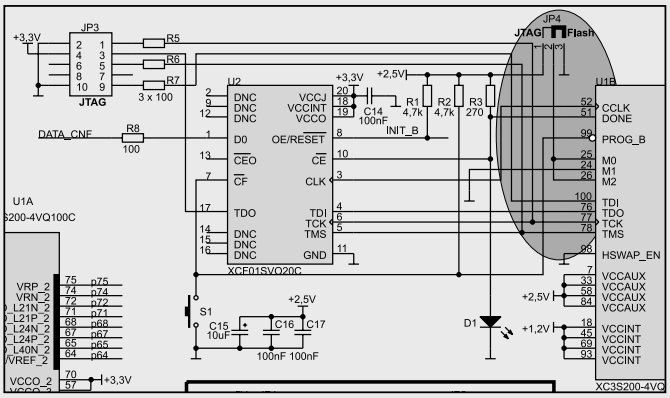
Interfejsy JTAG obydwu układów są połączone w szereg, tworząc tzw. łańcuch JTAG. Dzięki temu dostęp do tych układów jest możliwy za pomocą jednego programatora, dołączonego do pojedynczego złącza JTAG. Na poniższym rysunku pokazano sposób połączenia układu XC3S200 z pamięcią XCF01S, który automatycznie ustalił program iMPACT obsługujący




**Sposób połączenia układu FPGA i konfiguratora Flash w zestawie ZL10PLD**

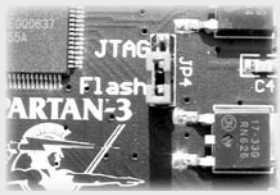
programator.

Pozostaje do omówienia ostatni problem: skąd FPGA ma wiedzieć, z jakiego źródła zostaną przesłane dane konfiguracyjne? Otóż do tego celu służą w układzie FPGA specjalne wejścia M2…M0, które na płytce ZL10PLD połączono w taki sposób, że za pomocą jednego jumpera JP4 (rysunek poniżej) można wybrać jeden z dwóch trybów pracy:
- automatycznego (po włączeniu zasilania) kopiowania zawartości Flash do FPGA,
- oczekiwania na dane konfigurujące z JTAG–a.




**Na fragmencie schematu płytki ZL10PLD zaznaczono na szaro dołączenie zworki JP4, która umożliwia wybranie sposobu konfiguracji FPGA. Pokazano także sposób dołączenia konfiguratora U2 do układu FPGA, przycisku S1 służącego do ręcznego wymuszania rekonfiguracji oraz diody świecącej D1, która służy do sygnalizacji poprawnego zakończenia konfiguracji FPGA**

Na zdjęciu poniżej pokazano lokalizację jumpera JP4 na płytce zestawu ZL10PLD wraz z opisami pozycji, w których konfiguracja FPGA odbywa się poprzez interfejs JTAG (JTAG) i automatycznie po włączeniu zasilania z konfiguratora (Flash).




**Zworka JP4 umożliwia wybór trybu konfiguracji FPGA (poprzez JTAG lub automatyczne odtworzenie konfiguracji z pamięci Flash)**

## Konfigurowanie układu FPGA

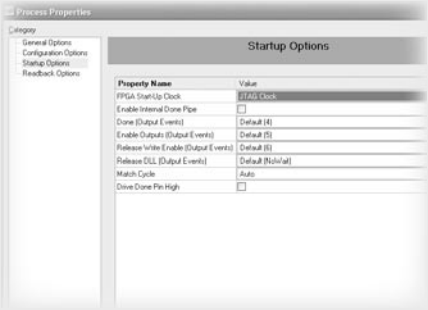
Konfigurowanie układu FPGA za pomocą interfejsu JTAG jest zalecane przede wszystkim podczas prac projektowych, bowiem liczba konfiguracji (czyli wpisów do pamięci konfiguracji typu SRAM) nie jest niczym ograniczona (w przeciwieństwie do pamięci Flash, którą można kasować i programować skończoną, choć niemałą, liczbę razy).

Do skonfigurowania układu FPGA jest konieczny plik o rozszerzeniu **.isc*, który jest generowany przez proces *Generate Programming File* pakietu WebPack ISE. Ponieważ ustawienia domyślne nie powodują utworzenia takiego pliku, trzeba we właściwościach (*Process Properties>General Options*) procesu *Generate Programming File* uaktywnić opcję *Create IEEE1532 Configuration File* (jak na rysunku poniżej).




**W zakładce *General Options* okna *Process Properties* (dla procesu *Generate Programming File*) trzeba uaktywnić generację pliku **.isc*, który posłuży nam do konfigurowania FPGA**

Żeby konfigurowany za pomocą interfejsu JTAG układ FPGA poprawnie rozpoczął swoją pracę (po zakończeniu konfigurowania), trzeba zmienić domyślne ustawienie w oknie *Process Properties>Startup Options* w polu *FPGA Start–up Clock* (jak na rysunku poniżej) z *CCLK* na *JTAG Clock*. Zaniechanie tej czynności przed skompilowaniem projektu spowoduje występujące losowo „zawieszanie" się FPGA. >>




**W przypadku konfigurowania FPGA za pomocą interfejsu JTAG trzeba zmienić ustawienie *FPGA Start–up Lock* na *JTAG Clock*, w przeciwnym przypadku układ może nie rozpocząć pracy po zakończeniu konfiguracji**

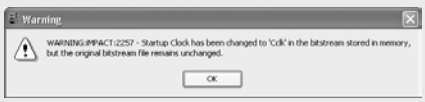
### Programowanie konfiguratora (Flash)

Jak już wielokrotnie wspominaliśmy, konfigurator (czyli pamięć Flash z interfejsem szeregowym i – w przypadku układu XCF01S – także JTAG) służy do przechowywania danych opisujących konfigurację FPGA. Do jego zaprogramowania niezbędny jest plik **.mcs*, który trzeba „ręcznie" utworzyć za pomocą programu iMPACT, wchodzącego w skład standardowej instalacji pakietu WebPack ISE. iMPACT musi być uruchomiony w trybie *Prepare a PROM File* (rysunek poniżej), kolejne kroki narzuca użytkownikowi kreator.

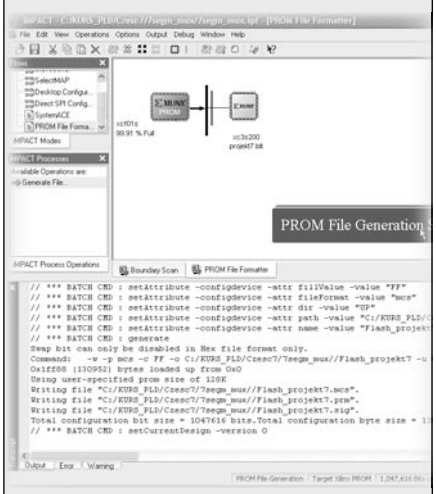



**Utworzenie pliku **.mcs* umożliwia program iMPACT, który należy uruchomić w trybie *Prepare a PROM File***

Żeby zapewnić poprawny restart układu FPGA po jego skonfigurowaniu z pamięci Flash, ustawienie *FPGA Start–up Lock* musi być *CCLK*. Jeżeli ostatnia kompilacja została przeprowadzona z innym ustawieniem, program iMPACT „zauważy" błąd użytkownika, ostrzeże go (rysunek poniżej) i samoczynnie zmieni fragment pliku **.mcs* w taki sposób, żeby układ FPGA po skonfigurowaniu działał poprawnie.




**Ostrzeżenie o nieprawidłowo wybranym źródle sygnału zegarowego inicjującego start FPGA po konfiguracji**




**Okno programu iMPACT po wygenerowaniu pliku **.mcs* służącego do programowania pamięci Flash konfiguratora**

## Realizacja

Schemat blokowy proponowanego rozwiązania pokazano na **rys. 2**. Jest to typowe rozwiązanie sterownika multipleksowego, w którym rozdzielono (opisano w osobnych plikach **.vhd*) dekodery wyjściowe (cyfr i segmentów) od sterownika, w którym zintegrowano (**rys. 3**):

– 16–bitowy rejestr latch, który służy do przechowywania kodów 4 liczb BCD wyświetlanych na wyświetlaczu,
– 2–bitowy licznik numeru wyświetlanej cyfry,
– multiplekser 16–>4, który podaje na zewnętrzny dekoder (sterujący segmentami wyświetlaczy) 4–bitowe słowo odpowiadające wyświetlanej pozycji.

Tak więc, budowa prezentowanego sterownika jest bardzo podobna do klasycznych realizacji, wykonywanych na układach TTL/CMOS. Wykorzystanie opisu HDL zwiększa elastyczność projektu, bowiem jakiekolwiek jego modyfikacje (zmiana polaryzacji wyświetlaczy z WA na WK, zmiana liczby obsługiwanych wyświetlaczy, zmiana kształtów wyświetlanych znaków

### Plan kursu

1. Wprowadzenie
   - Budowa zestawu uruchomieniowego
   - Programowanie i konfiguracja układu XC3S200
   - Tryby konfiguracji układu XC3S200
   - Zasilanie układu XC3S200
   - Linie I/O w układzie XC3S200
   - JTAG jako uniwersalny interfejs do programowania i konfigurowania
2. Budowa, cechy funkcjonalne i parametry układów FPGA z rodziny Spartan 3
   - CLB
   - IOB
   - Globalne sygnały zegarowe
   - DCM
   - Sprzętowe multiplikatory
   - Pamięć BlockRAM
3. **Projekty przykładowe**
   - Debouncer
   - Klawiatura matrycowa
   - **Obsługa wyświetlacza multipleksowego LED**
   - Obsługa wyświetlacza LCD
   - Sterownik LCD 2x16 (prosty)
   - Sterownik LCD 2x16 (zaawansowany)
   - Komunikacja via RS232 i USB
   - Sterownik VGA
   - Implementacja mikrokontrolera PicoBlaze

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

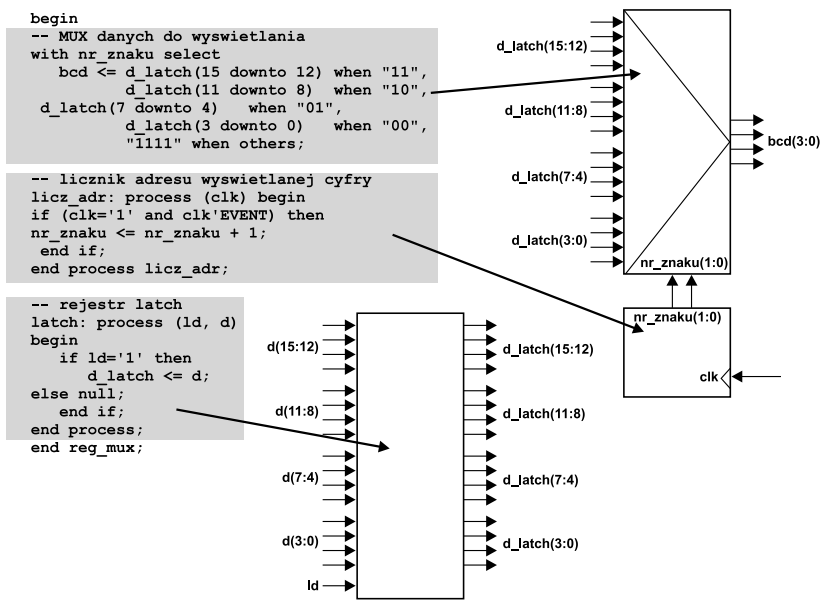
entity sterownik is port (
   d: in std_logic_vector(15 downto 0);
   ld, clk: in std_logic;
   bcd: out std_logic_vector(3 downto 0);
   nr_znaku: inout std_logic_vector(1 downto 0)
 );
end sterownik;

architecture reg_mux of sterownik is
signal d_latch: std_logic_vector(15 downto 0);

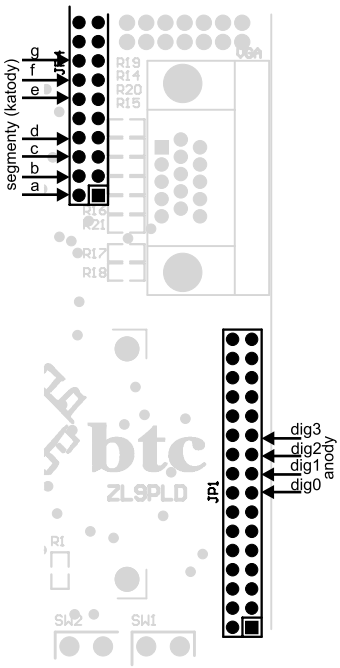
begin
-- MUX danych do wyswietlania
with nr_znaku select
   bcd <= d_latch(15 downto 12) when "11",
          d_latch(11 downto 8)  when "10",
 d_latch(7 downto 4)   when "01",
          d_latch(3 downto 0)   when "00",
          "1111" when others;

-- licznik adresu wyswietlanej cyfry
licz_adr: process (clk) begin
if (clk='1' and clk'EVENT) then
nr_znaku <= nr_znaku + 1;
 end if;
end process licz_adr;

-- rejestr latch
latch: process (ld, d)
begin
   if ld='1' then
      d_latch <= d;
else null;
   end if;
end process;
end reg_mux;
```

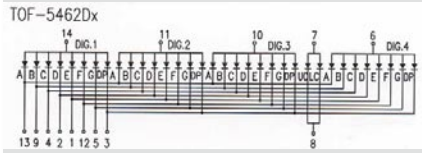



**Rys. 3. Funkcje spełniane przez fragmenty opisu w pliku sterownik.vhd**




**Rys. 4. Sposób dołączenia wyświetlacza do układu FPGA**

Zastosowany w projekcie wyświetlacz LED (TOF–5462DB–B) jest przystosowany do pracy w systemach ze sterowaniem multipleksowym, niezbędne połączenia segmentów są wykonane w jego wnętrzu. Na rysunku poniżej pokazano schemat połączeń wewnętrznych.




Dodatkowe informacje: *http://www.artronic.com.pl/o_produkcie.php?id=656?*, dystrybutor oferuje wersje świecące na różne kolory.

itp.) wiąże się z wprowadzeniem niewielkich modyfikacji w opisie (przykład pokażemy w dalszej części artykułu). Możliwe jest także przygotowanie uniwersalnego opisu z parametryzacją, co – jeśli Czytelnicy wykażą zainteresowanie tematem – pokażemy w jednym z kolejnych odcinków kursu.

W przykładowym projekcie wykorzystano bardzo wygodny w praktyce wyświetlacz TOF–5462DB–B, który jest przystosowany do sterowania multipleksowego, bowiem segmenty poszczególnych znaków są ze sobą wewnętrznie połączone. Zastosowano wyświetlacz świecący na kolor niebieski, bowiem pozwoliło to uniknąć konieczności stosowania rezystorów ograniczających prąd segmentów. Wynika to z faktu, że napięcie progowe struktur świecących na niebiesko wynosi ok. 3...3,2 V, co przy zasilaniu segmentów napięciem ok. 3,3 V zapewnia przepływ prądu o natężeniu nie przekraczającym (w egzemplarzu modelowym) 3,5 mA. Elektrody wyświetlaczy (anody i katody) są sterowane bezpośrednio z wyjść FPGA (bez jakichkolwiek buforów, **rys. 4**), co upraszcza budowę urządzenia i minimalizuje liczbę niezbędnych elementów.

**List. 1. Modyfikacja dekodera sterującego wyświetlacz LED (ze wspólną anodą): po lewej stronie pokazano opis dekodera znaków 0...9, po prawej 0...F**

```
LEWA STRONA
with bcd select
--        g f e d c b a
   segm <= „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;   -- wygaszenie

PRAWA STRONA
with bcd select
--        g f e d c b a
   segm <= „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
           „0001000” when „1001”,   -- 10 “A”
           „0000011” when „1001”,   -- 11 “B”
           „0000110” when „1001”,   -- 12 “C”
           „0100001” when „1001”,   -- 13 “D”
           „0000110” when „1001”,   -- 14 “E”
           „0001110” when „1001”,   -- 15 “F”
           „1111111” when others;   -- wygaszenie
```

## Implementacja

Projekt zaimplementowany w testowym zestawie, poza sterownikiem wyświetlaczy, zawiera także 4–dekadowy licznik BCD (zliczający impulsy z wyjścia preskalera, jego stany wyjściowe podglądamy na wyświetlaczach) oraz 20–bitowy preskaler (w projekcie wykorzystano 18 bitów), który służy do wytworzenia sygnałów zegarowych dla zespołu liczników i sterownika wyświetlacza.

Strukturę projektu (w plikach) pokazano na **rys. 5**. Widoczny na tym rysunku plik *projekt7.ucf* zawiera przypisania sygnałów do fizycznych wyprowadzeń układu FPGA.

## Podsumowanie

Wielokrotnie podkreślana duża elastyczność opisów HDL dotyczy także tego projektu. Na **list. 1** pokazano modyfikację dekodera 7–segmentowego, który w wersji pokazanej po lewej stronie wyświetla znaki wyłącznie z zakresu 0...9 (dla wartości 10...15 wyświetlacz jest wygaszony), po prawej stronie pokazano zmodyfikowana wersję wyświetlającą także znaki A...F (w formie możliwej na wyświetlaczu 7–segmentowym). Inną możliwą modyfikacją jest dostosowanie sterownika do współpracy z wyświetlaczami ze wspólną katodą, co wymaga zanegowania wyjść dekodera znaków i segmentów, np. (odnosi się do pliku *dek_znaku.vhd*) opis:

```
with nr_znaku select
    dig <= „0001” when „00”,   --
cyfra 0 (jedn.)
.....
```

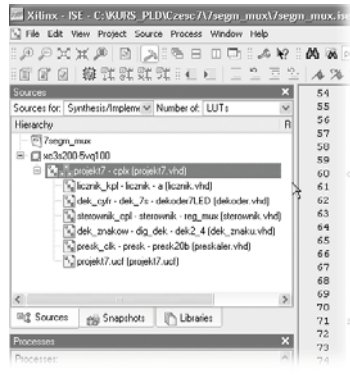
można zastąpić opisem:

```
with nr_znaku select
    dig <= „1110” when „00”,   --
cyfra 0 (jedn.)
.....
```

można także zanegować stany wyjściowe za pomocą operacji *not*.

Podobnych modyfikacji można wykonać wiele: jeżeli któryś z Czytelników chciałby pochwalić się własnymi rozwiązaniami – zapraszamy!

**Jacek Majewski**
**jacek.majewski@pwr.wroc.pl**
**Piotr Zbysiński, EP**
**piotr.zbysinski@ep.com.pl**




**Rys. 5. Budowa prezentowanego projektu z podziałem na pliki z opisami VHDL**