Sterownik wyświetlacza multipleksowanego w VHDL

W artykule przedstawiamy rozwiązanie sterownika czterech wyświetlaczy LED przeznaczonego do aplikowania w układach programowalnych. Nowością jest zastosowany sposób jego opisu - zastosowano bowiem jeden z najbardziej popularnych obecnie język opisu sprzętu - VHDL. Wykorzystanie uniwersalnego języka HDL pozwala traktować prezentowany projekt jak klasyczny blok IP (Intellectual Property core). Rekomendacje: jest to projekt szczególnie interesujący dla fanów nowoczesnych sposobów projektowania urządzeń elektronicznych. Niebywała szansa poznania od środka IP-core'ów dla układów PLD.

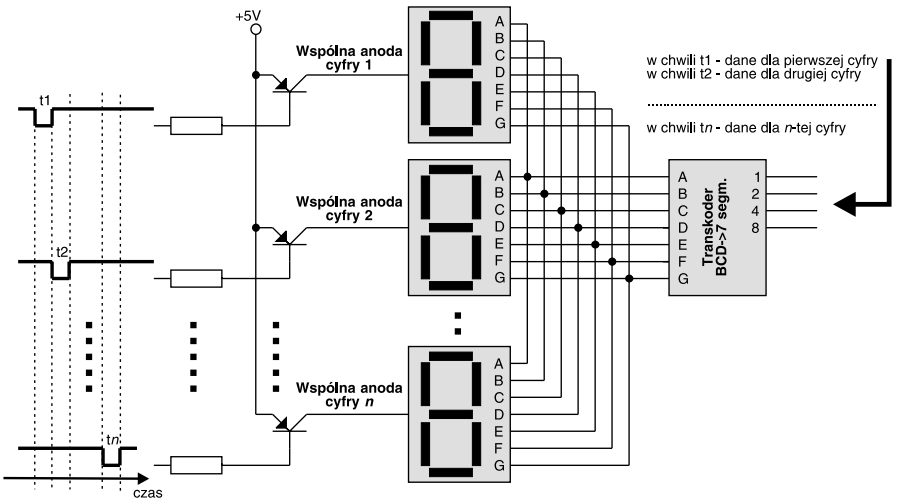
Zacznę od wyjaśnienia, dlaczego sięgnąłem po tak „ciężką" broń jak VHDL. Uchodzi on dość powszechnie, choć niesłusznie, za jeden z bardziej skomplikowanych języków opisu sprzętu (HDL - *Hardware Description Language*). Pomimo dość rygorystycznych reguł formalnych obowiązujących podczas pisania programu w VHDL język ten charakteryzuje się znaczną uniwersalnością, co w praktyce oznacza, że dobrze przygotowany opis bloku funkcjonalnego będzie można „wbudować" zarówno w układ PLD, jak i ASIC pochodzących od różnych producentów - w obydwu przypadkach będą one działały tak samo (za wyjątkiem parametrów czasowych, które są silnie powiązane z technologią i strukturą logiczną układu). Niebagatelne znaczenie dla projektantów systemów cyfrowych ma fakt, że podzbiór języka VHDL, który jest obsługiwany przez programy do syntezy logicznej, jest niewielki i stosunkowo łatwy do nauczenia się.

Walor „uniwersalności" posiada także Verilog, który jakkolwiek bardziej przyjazny użytkownikowi, nie cieszy się aż tak dużą popularnością wśród projektantów.

Drugą, często przytaczaną cechą VHDL-a jest możliwość opisywania projektowanego układu na wiele sposobów m.in.: przepływowy (opis ścieżek przesyłania danych), behawioralny (opis zachowania się bloku w zależności od sygnałów zewnętrznych i wewnętrznych), czy też strukturalny (najczęściej spotykaną formą ta-

Wojsko napędza rozwój technologii

Opracowanie języka VHDL zostało zainicjowane w 1981 roku przez Departament Obrony USA w celu ujednolicenia opisu elementów systemów elektronicznych. Przyjęto, że ma to być język z szerokim zakresem możliwości opisowych, który byłby akceptowalny przez dowolny symulator i był niezależny od technologii i sposobu projektowania. Pierwotnie VHDL miał służyć jednoznacznemu i precyzyjnemu dokumentowaniu dużych systemów cyfrowych. W 1987 roku VHDL uzyskał normę IEEE (nowelizowaną w 1993 roku). Dzięki standaryzacji i różnorodnemu zastosowaniu (do celów dokumentacyjnych, symulacji, adaptacji do układów programowalnych) powstały efektywne kompilatory tego języka, które przyczyniły się do jego spopularyzowania. Chociaż powstał z myślą o opisywaniu dużych systemów, to szybko został zaadoptowany do opisywania układów programowalnych (szczególnie układów ASIC i dużych systemów tworzonych w FPGA).



Rys. 1. Ilustracja zasady wyświetlania multipleksowanego

kiego opisu są równania logiczne). Szczerze mówiąc większość języków HDL, także starszej generacji jak np. CUPL czy ABEL, oferują podobne możliwości, ale poddajmy się modzie...

Przed laty wielką estymą wśród elektroników cieszyli się ci, którzy potrafili programować mikroprocesory w asemblerze. Dziś taka umiejętność jest również ważna, pozwala bowiem na najbardziej efektywne wykorzystanie możliwości mikroprocesora (krótki kod i większa szybkość realizowania procedur), ale nie jest już niezbędna. Z czasem pojawiły się bowiem narzędzia programowe, które pozwalają na programowanie mikroprocesorów w językach wyższego poziomu, jak np. w C czy nawet w BASIC-u. Długość kodu wynikowego też nie jest już parametrem tak krytycznym, gdyż pamięci o dużej pojemności są powszechnie dostępne. Wydaje się, że podobnie jest z opisem układów do realizacji w strukturach programowalnych. Dobra znajomość języków opisu sprzętu jest cenną umiejętnością projektanta, szczególnie wtedy gdy projektuje duże układy i chciałby efektywnie wykorzystać zasoby docelowego układu programowalnego.

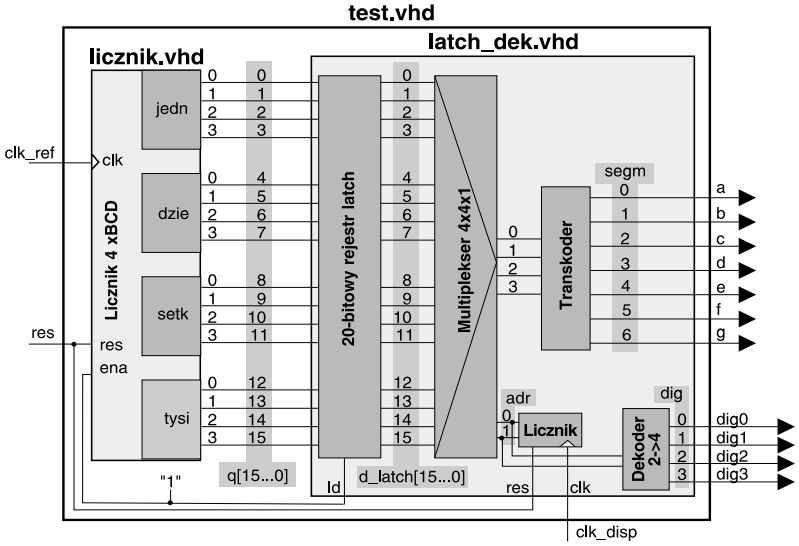
Opis projektu

Ze względu na możliwość ograniczenia liczby niezbędnych wyprowadzeń układu, do sterowania wyświetlaczy LED bardzo często są stosowane systemy z multipleksowaniem. Cieszą się one powodzeniem zarówno wśród projektantów systemów mikroprocesorowych, jak wśród projektantów urządzeń budowanych z układów dyskretnych (TTL lub CMOS), jak i w sterownikach wyświetlaczy wbudowywanych w układy PLD.

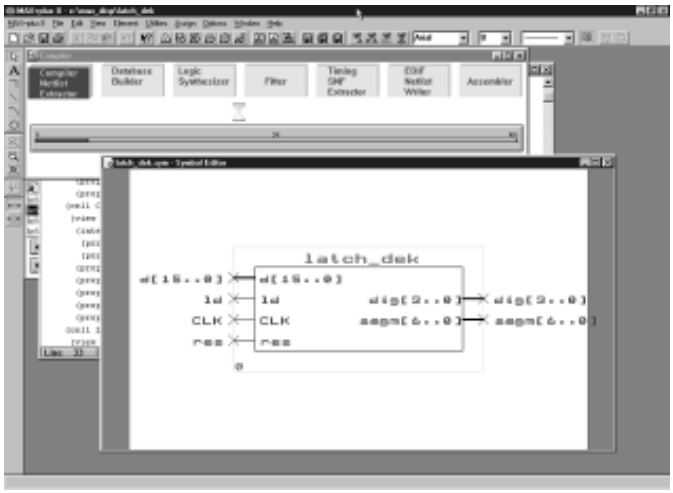
Zasada działania wyświetlania multipleksowego jest dość prosta: segmenty wszystkich wyświetlaczy są połączone ze sobą równolegle i sterowane z wyjść jednego transkodera kodu (przykładowo) BCD na kod wyświetlacza 7-segmentowego (**rys. 1**). Wspólne elektrody wyświetlaczy (anody lub katody) są sterowane niezależnie w taki sposób, że w danej chwili zasilana jest tylko jedna z nich. Jednocześnie na wejścia dekodera jest podawany kod znaku, który ma być wyświetlony na wybranej pozycji. W ten sposób wyświetlane są po kolei wszystkie znaki i cały proces jest powtarzany z częstotliwością większą niż 50 Hz (licząc częstotliwość przypadającą na każdy wyświetlany znak), dzięki czemu osoba patrząca na wyświetlacz widzi ciągle wyświetlane cyfry.

Schemat blokowy proponowanego rozwiązania sterownika 4-cyfrowego wyświetlacza pokazano na **rys. 2**. Do jego wejść został dołączony blok czterech liczników BCD, dzięki któremu można przetestować działanie sterownika. Zaczniemy od omówienia zasadniczej części projektu, którego opis w języku VHDL pokazano na **list. 1**. Każdy z fragmentów funkcjonalnych sterownika opisano osobno (łatwo je zauważyć dzięki komentarzom umieszczonym na listingu), przy czym opisy fragmentów synchronicznych (jak np. licznik wyświetlanej cyfry) umieściłem w niezależnych procesach.

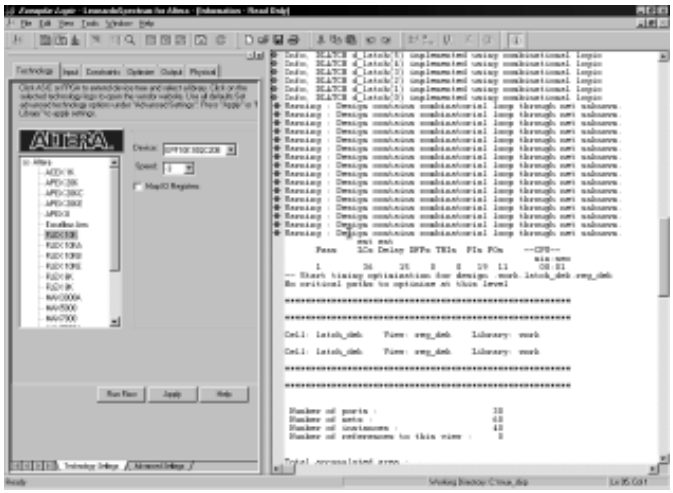
Ponadstandardowym wyposażeniem sterownika jest 16-bitowy rejestr wejściowy typu *latch*, który może pracować w trybie przeźroczystym (gdy LD = 1) lub jako standardowy przerzutnik zatrzaskowy (dane są w nim zatrzaskiwane poprzez zmianę stanu wejścia LD z 1 na 0). W zależności od wymagań aplikacji z rejestru tego można skorzystać lub nie. W tym drugim przypadku można się pokusić o usunięcie z list. 1 fragmentu za-



Rys. 2. Schemat blokowy prezentowanego projektu w VHDL



Rys. 3. Widok okna edytora elementów bibliotecznych w pakiecie Max+Plus II (bezpłatna wersja Baseline)



Rys. 4. Widok okna programu do syntezy VHDL Leonardo Spectrum (także dostępny bezpłatnie)

wierającego opis tego rejestru, dzięki czemu zmniejszy się liczba (o 16) makrokomórek niezbędnych do implementacji projektu.

Jak wspomniano wcześniej, do testowania sterownika zastosowano cztery liczniki BCD, których opis pokazano na **list. 2**. Są to liczniki pracujące synchronicznie, których opis podzieliłem na cztery procesy niezależne dla każdej pozycji (cyfry). Liczniki są zerowane asynchronicznie i wyposażone w wejście zezwolenia na zliczanie (*ena*), które uaktywnia licznik gdy na to wejście jest podana logiczna 1.

Tak przygotowane opisy bloków wymagają połączenia ze sobą, dzięki czemu powstanie urządzenie, którego schemat blokowy pokazano na rys. 2. Można je połączyć ze sobą na wiele różnych sposobów, z których niewątpliwie najwygodniejszy jest sposób graficzny (za pomocą specjalizowanego edytora schematów). Ze względu na walory dydaktyczne w artykule przedstawimy nieco mniej czytelny sposób, który zilustruje zasadę tworzenia projektów hierarchicznych w języku VHDL. Hierarchiczny opis zestawu: sterownik wyświetlaczy multipleksowanych (plik *latch_dek.vhd* - list. 1) z dołączonym 4-cyfrowym licznikiem BCD (plik *licznik.vhd* - list. 2) pokazano na **list. 3**. Pomocny w analizie tego opisu będzie rys. 2, na którym zostały wyraźnie zaznaczone nazwy wszystkich sygnałów wykorzystanych w projekcie.

Implementacja

Dzięki zastosowaniu opisu w języku VHDL udało się uzyskać łatwą jego przenośność pomiędzy systemami projektowymi różnych producentów. Ponieważ prezentowany przykład jest stosunkowo prosty, zastosowane w nim mechanizmy

List. 1. Opis w języku VHDL bloku sterownika 4-cyfrowego wyświetlacza multipleksowanego (plik *latch_dek.vhd*)

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

entity latch_dek is Port (
   d: in std_logic_vector(15 downto 0);
   ld, clk, res: in std_logic;
   dig: out std_logic_vector(3 downto 0);
   segm: out std_logic_vector(6 downto 0)
    );
end latch_dek;

architecture reg_dek of latch_dek is
signal adr: std_logic_vector(1 downto 0);        - linie adresowe MUX-a
signal bcd: std_logic_vector(3 downto 0);        - wyj. MUX-a/wej. dekodera
signal d_latch: std_logic_vector(15 downto 0);   - wyjscie rejestru latch
begin
- dekoder wyswietlanych cyfr
with adr select
   dig <= "1110" when "00",    - cyfra 0 (jedn.)
          "1101" when "01",    - cyfra 1 (dzies.)
          "1011" when "10",    - cyfra 2 (setki)
          "0111" when "11",    - cyfra 3 (tys.)
          "1111" when others;    - wygaszenie cyfr

- MUX danych do wyswietlania
with adr 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, res) begin
   if (res='1') then adr  <= "00";
      elsif (clk='1' and clk'event) then
         adr <= adr + 1;
   end if;
end process licz_adr;

- transkoder
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

- rejestr latch
latch: process (ld, d, res)
begin
  if res='1' then
     d_latch <= "0000000000000000";
  elsif ld='1' then
     d_latch <= d;
  end if;
end process;
end reg_dek;
```

Rys. 5. Wygląd okna systemu projektowego WebPack ISE (dostępny bezpłatnie w Internecie)

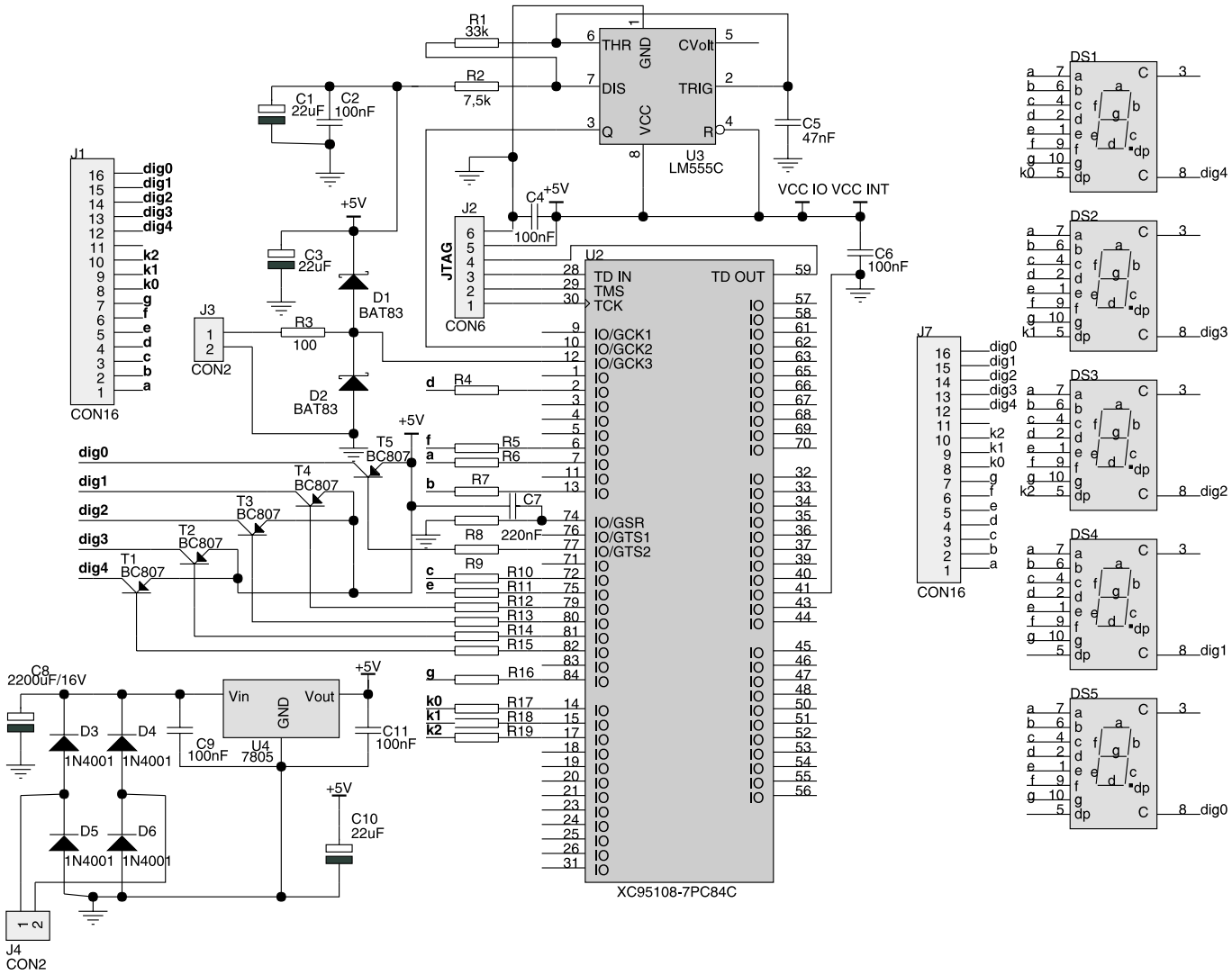
opisu są elementarne i możliwe do przyswojenia przez zdecydowaną większość systemów projektowych. Ostateczną wersję prezentowanego projektu poddano kompilacji przez systemy (wszystkie dostępne bezpłatnie): Max+Plus II (wersje: SE oraz Baseline - **rys. 3** - z Leonardo Spectrum - **rys. 4**) i Quartus II firmy Altera, WebPack ISE firmy Xilinx (**rys. 5**) oraz Warp firmy Cypress (wersja bezpłatna tego systemu dostępna wyłącznie z książką „Język VHDL" Kevina Skahilla, która ukazała się nakładem WNT). Warto tutaj przypomnieć, że kurs posługiwania się pakietem WebPack ISE publikowaliśmy w EPo/oL 4...7/2002.

Projekt był kompilowany na układy CPLD (z serii MAX7000S, XC9500 i Flash370i) i zajmuje w nich 30 makrokomórek.

Testowanie

Nic tak dobrze nie robi projektowi, jak praktyczna weryfikacja jego działania. Testy przeprowadziłem na prostym zestawie, którego schemat znajduje się na **rys. 6**. Ponieważ zestaw był projektowany do zupełnie innych zadań, jego wyposażenie jest nieco nadmiarowe (zastosowano w nim m.in. 5 wyświetlaczy, z których tylko 4 wykorzystujemy w przykładzie). Zastosowany w nim układ XC95018 jest wyposażony w interfejsach JTAG, za pomocą którego można programować jego nieulotną pamięć konfiguracji. Opisy programatorów ISP (odpowiednik DLC5, który jest obsługiwany przez system WebPack ISE) można znaleźć m.in. w EP4/2001 (AVT-1303).

Tranzystory T1...T5 (w przykładzie wykorzystane T2...T5) spełniają rolę wzmacniaczy prądowych sterujących wspólne anody wyświetlaczy. Segmenty wyświetlaczy są sterowane bezpośrednio z wyjść układu CPLD (ich



Rys. 6. Schemat elektryczny modułu ewaluacyjnego

List. 2. Opis 4-cyfrowego licznika BCD (plik *licznik.vhd*)

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

entity licznik is
   Port ( res, clk, ena: in std_logic;
  ovf: out std_logic;
  q: out std_logic_vector(15downto 0)
   );
end licznik;

architecture a of licznik is
signal jedn: std_logic_vector(3 downto 0);
signal dzie: std_logic_vector(3 downto 0);
signal setk: std_logic_vector(3 downto 0);
signal tysi: std_logic_vector(3 downto 0);

begin
jedn_cnt: PROCESS (clk, res) BEGIN
   if (res='1') then jedn  <= "0000";
       elsif (clk='1' and clk'event) then
           if (ena='1') then
               if (jedn="1001") then jedn <= "0000";
                   else jedn <= jedn + 1;
               end if;
       end if;
       end if;
   end process jedn_cnt;

dz_cnt: process (clk, res) begin
   if (res='1') then dzie  <= "0000";
       elsif (clk='1' and clk'event) then
            if (ena='1') then
                if (dzie="1001" and jedn="1001") then dzie<="0000";
               elsif (jedn="1001") then dzie <= dzie + 1;
                    end if;
           end if;
   end if;
end process dz_cnt;

set_cnt: process (clk, res) begin
   if (res='1') then setk  <= "0000";
       elsif (clk='1' and clk'event) then
           if (ena='1') then
               if (setk="1001" and dzie="1001" and jedn="1001") then setk <= "0000";
                  elsif (dzie="1001" and jedn="1001") then setk <= setk + 1;
               end if;
          end if;
   end if;
end process set_cnt;

tys_cnt: process (clk, res) begin
   if (res='1') then tysi  <= "0000";
       elsif (clk='1' and clk'event) then
           if (ena='1') then
                if (tysi="1001" and setk="1001" and dzie="1001" and jedn="1001" ) then
tysi<="0000";
                   elsif (setk="1001" and dzie="1001" and jedn="1001") then tysi <= tysi + 1;
                end if;
           end if;
   end if;
end process tys_cnt;

q <= tysi & setk & dzie & jedn; -
end a;


test.vhd
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;

entity test is Port (
   clk_ref, res, clk_disp: in std_logic;
   a, b, c, d, e, f, g, dig4: out std_logic;
   dig: out std_logic_vector(3 downto 0)
   );
end test;

architecture cplx of test is
component latch_dek port (
   d: in std_logic_vector(15 downto 0);
   ld, clk, res: in std_logic;
   dig: out std_logic_vector(3 downto 0);
   segm: out std_logic_vector(6 downto 0)
  );
end component latch_dek;

component licznik port (
   res, clk, ena: in std_logic;
   q: out std_logic_vector(15 downto 0)
  );
end component licznik;

signal q_int: std_logic_vector(15 downto 0);

begin
licznik_kpl: licznik port map (
   clk => clk_ref,
   res => res,
   q   => q_int,
   ena => '1'
   );

reg_dek_mux: latch_dek port map (
   d   => q_int,
   clk => clk_disp,
   res => res,
   dig => dig,
   ld  => '1',
   segm(0)  => a,
   segm(1)  => b,
   segm(2)  => c,
   segm(3)  => d,
   segm(4)  => e,
   segm(5)  => f,
   segm(6)  => g
   );

dig4 <= '1'; - wynika z budowy zestawu eval - nie wplywa na dzialanie bloku

end cplx;
```

dopuszczalna obciążalność prądowa „do masy" zasilania wynosi 24 mA). Taki sposób sterowania wymusił konieczność zakodowania tablicy prawdy transkodera (list. 1) w logice ujemnej (czyli 0 jest aktywne). Także tranzystory-drivery (typu PNP) są aktywowane przez 0 logiczne, w związku z czym w opisie dekodera wyświetlanych cyfr (list. 1) sygnałem aktywnym jest także 0.

Układ U3 spełnia rolę zewnętrznego generatora taktującego sterownik wyświetlaczy. Wartości elementów R1, R2 i C5 ustalają częstotliwość jego pracy na ok. 400 Hz, co w zupełności wystarcza do poprawnego wysterowania 4 wyświetlaczy.

Zastosowanie

Prezentowany w artykule moduł sterownika wyświetlaczy jest jednym z elementów uniwersalnego projektu częstościomierza z automatyczną zmianą zakresów, który w całości opisano za pomocą języka VHDL (szczegółowo opiszemy go w jednym z najbliższych wydań EP). Nie jest to oczywiście jego jedyne możliwe zastosowanie. Ponieważ komplet plików źródłowych udostępniamy w Internecie (na *www.ep.com.pl* w dziale *Download>Dokumentacje*), każdy projektant będzie mógł wykorzystać sterownik w dowolnych własnych opracowaniach. Pomimo niedoskonałości współczesnych narzędzi do syntezy VHDL, dzięki zastosowaniu tego języka opisu udało się uzyskać 100% przenośność pomiędzy wcześniej wymienionymi systemami projektowymi. Istnieje duża szansa na kompatybilność opisu także w z innymi systemami projektowymi (pośród bezpłatnych dostępne są jeszcze m.in.: QuickWorks firmy QuickLogic oraz pakiety Libero i Designer firmy Actel), dzięki czemu można łatwo wykorzystać udostępnione opisy dla dowolnego układu PLD (także FPGA).

Piotr Zbysiński, AVT
piotr.zbysinski@ep.com.pl

Wzory płytek drukowanych w formacie PDF są dostępne w Internecie pod adresem: ***http://www.ep.com.pl/?pdf/wrzesien02.htm****.*