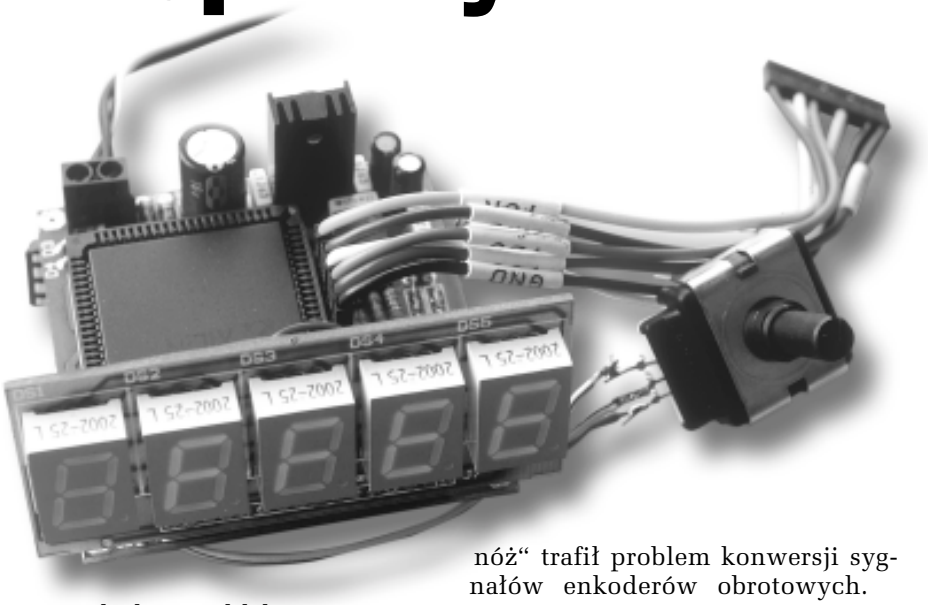
Konwerter sygnałów wyjściowych obrotowych enkoderów opisany w VHDL

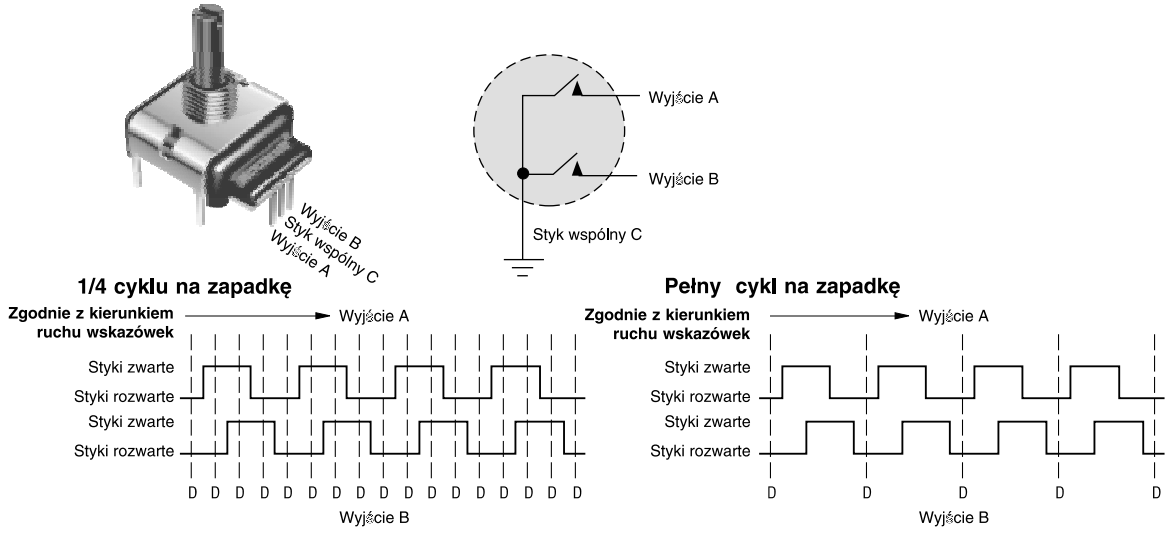
Przedstawiamy kolejny minimoduł, którego opis przygotowano w języku opisu sprzętu - VHDL. W artykule prezentujemy projekt układu przekształcającego sygnały impulsowe z wyjść dwukierunkowych obrotowych enkoderów do postaci pozwalającej na zastosowanie do ich zliczania popularnych liczników z rodziny TTL lub CMOS.
***Rekomendacje**: układ może być stosowany we wszelkiego rodzaju aplikacjach, w których jako elementy regulacyjne są wykorzystywane enkodery obrotowe.*



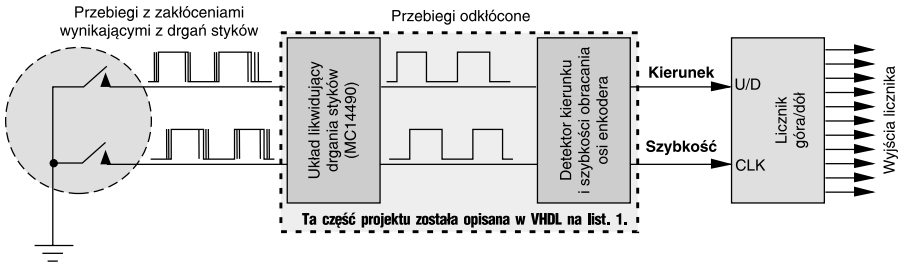
Artykuł opublikowany we wrześniowej EP, poświęcony sterownikowi wyświetlacza multipleksowanego LED opisanemu w VHDL, wzbudził wśród Czytelników spore zainteresowanie. Dostałem także propozycje kilku układów, którymi warto się zająć. Będę je kolejno opracowywał. Opisy rozwiązań przedstawimy w EP, a kody źródłowe udostępnimy na płytach CD-EP dołączanych do EPo/oL oraz na naszej stronie internetowej. Jako pierwszy „pod nóż" trafił problem konwersji sygnałów enkoderów obrotowych.

Jak działa enkoder obrotowy?

Enkodery obrotowe stosowaliśmy kilkukrotnie w projektach publikowanych w EP (m.in. w zasilaczu programowanym cyfrowo - AVT-366 i tunerze FM - AVT-900). Enkodery mogą zastępować - oczywiście po pewnym rozbudowaniu ich części elektronicznej - klasyczne potencjometry stosowane do regulacji, których „obsługa" jest znacznie bliższa ludz-



Rys. 1. Zasada działania enkodera obrotowego



Rys. 2. Schemat blokowy dekodera-enkodera

kim przyzwyczajeniom niż korzystanie z klawiatur dość często stosowanych w różnych urządzeniach.

Zasadę działania enkodera obrotowego zilustrowano na **rys. 1**. Na jego wyjściach A i B pojawiają się impulsy (w wyniku zwarcia ze wspólnym wyprowadzeniem C), których faza niesie informację o kierunku obracania osi enkodera, natomiast ich częstotliwość niesie informację o szybkości jej obracania. Wziąwszy pod uwagę nieuniknione zakłócenia impulsowe powstające podczas zwierania i rozwierania styków, taka para sygnałów nie nadaje się do wykorzystania bezpośrednio w innych układach cyfrowych.

Rozwiązanie tego problemu zalecane przez firmę Bourns (jednego z większych producentów enkoderów obrotowych) pokazano na **rys. 2**. Pierwszym ważnym etapem obróbki sygnałów z wyjść enkodera jest usunięcie zakłóceń wynikających z drgań styków (można np. zastosować specjalizowany układ scalony jak np. MC14490 lub MAX6816/17/18). Następnie należy:

- ustalić na podstawie fazy sygnałów na wyjściach enkodera kierunek obrotu osi i ustalić odpowiedni poziom na wyjściu *Kierunek*,
- generować na wyjściu *Szybkość* sygnał o częstotliwości wynikającej z szybkości obracania osi enkodera.

Zadanie nie jest zbyt łatwe do zrealizowania w oparciu o układy cyfrowe TTL lub CMOS. Znacznie łatwiej można je rozwiązać posługując się mikrokontrolerem (dobry przykład można znaleźć w opisie zasilacza AVT-366 w EP1/98).

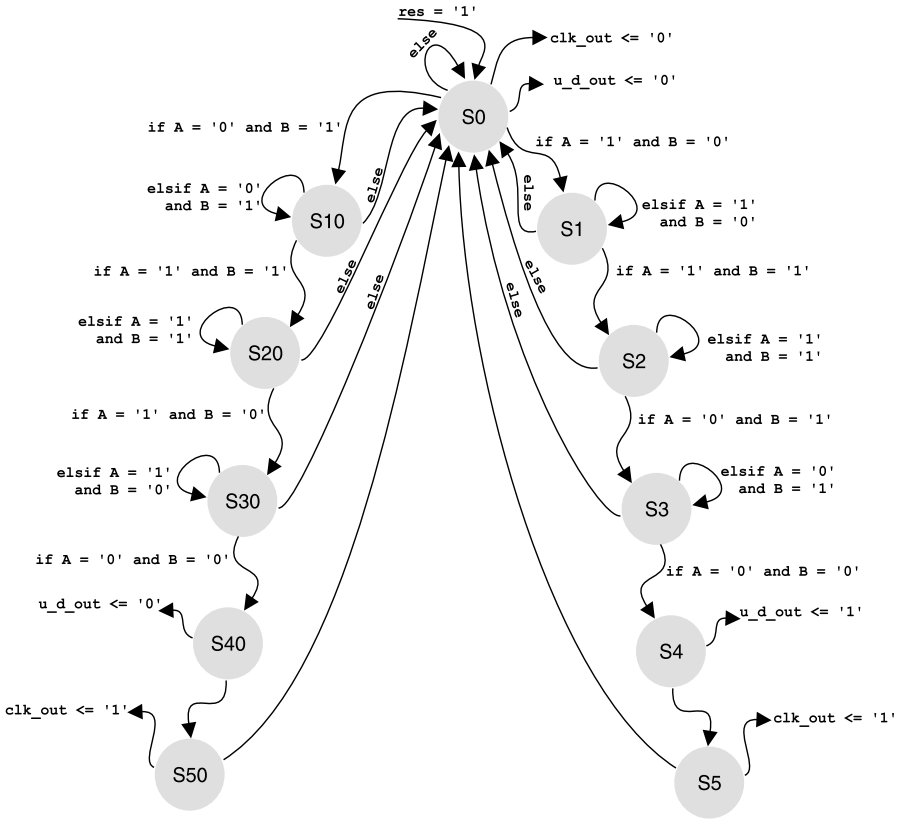
Jak to opisać w VHDL-u?

Postawiliśmy sobie za cel wykonanie kompletnego „dekodera" współpracującego z enkoderem w układzie programowalnym. Ze względu na łatwą przenośność pomiędzy systemami narzędziowymi różnych producentów i wynikającą z niej uniwersalność projektów opisanych w VHDL, ten właśnie język posłużył do przygotowania opisu modułu funkcjonalnego, którego uproszczony schemat blokowy zaznaczono grubą linią przerywaną na rys. 2. Tak zaprojektowany „dekoder" nadaje się do bezpośredniej współpracy z licznikami wyposażonymi w wejścia: sterujące kierunkiem zliczania UP/DOWN i zegarowe (mogą to być np. CMOS - 4029, TTL - 74190 lub 74191).

Ponieważ do likwidacji drgań styków potrzebne są układy odmierzające czas, to konieczne było wbudowanie w strukturę PLD timera. Najprostszym sposobem wykonania go w układzie PLD jest zbudowanie automatu taktowanego zewnętrznym sygnałem zegarowym i ustalenie mu takiego cyklu pracy, który zminimalizuje wpływ drgań styków na jakość dekodowania.

Na **rys. 3** przedstawiono graf prostego automatu, którego opis w VHDL zamieszczono na **list. 1**. Jak można zauważyć, zaprojektowany automat jest pełnym odpowiednikiem bloku zakreślonego na rys. 2 przerywaną grubą linią, czyli odpowiada za dekodowanie stanu wyjść enkodera, ich obróbkę i generację sygnałów sterujących zewnętrznym licznikiem. Jak w każdym automacie synchronicznym, także praca tego jest synchronizowana przebiegiem zegarowym, który jest podawany na wejście *clk*.

We współczesnych językach HDL, także w VHDL, opisanie pracy automatu jest możliwe na wiele sposobów. Aby zapewnić przejrzystość opisu działania, wybrałem opis za pomocą grafu, którego poszczególne stany są jawnie predefiniowane za pomocą deklaracji *constant Sx*, gdzie x



Rys. 3. Uproszczony graf ilustrujący zasadę działania układu opisanego na list. 1

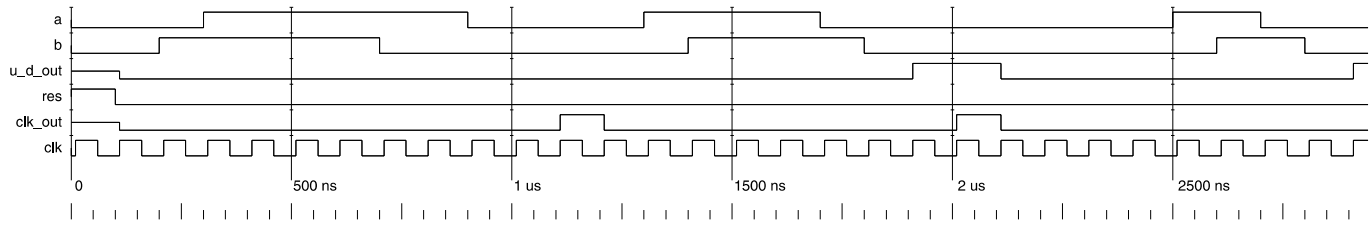
List. 1. Opis w języku VHDL działania „dekodera" enkoderów obrotowych

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

entity enkoder is
   Port ( A : in std_logic;  - wejście kanalu A enkodera
          B : in std_logic;  - wejście kanalu B enkodera
          u_d_out: out std_logic;   - wyjscie kierunku
          res: in std_logic; - wejscie zerujace
          clk_out: out std_logic;   - wyjscie sygnalu taktujacego licznik
          clk: in std_logic  - wejscie sygnalu zegarowego taktujacego automat
      );
end enkoder;

architecture arch of enkoder is
signal stan: std_logic_vector(3 downto 0);
constant S0: std_logic_vector(3 downto 0):= "0000";
constant S1: std_logic_vector(3 downto 0):= "0001";
constant S2: std_logic_vector(3 downto 0):= "0010";
constant S3: std_logic_vector(3 downto 0):= "0011";
constant S4: std_logic_vector(3 downto 0):= "0100";
constant S5: std_logic_vector(3 downto 0):= "0101";
constant S10: std_logic_vector(3 downto 0):= "1001";
constant S20: std_logic_vector(3 downto 0):= "1010";
constant S30: std_logic_vector(3 downto 0):= "1011";
constant S40: std_logic_vector(3 downto 0):= "1100";
constant S50: std_logic_vector(3 downto 0):= "1101";

begin
  process (clk, res)
  begin
     if res = '1' then
        stan <= "0000";
        elsif clk'event and clk = '1' then
           case stan is
           when S0 =>
              clk_out <= '0';
              u_d_out <= '0';
           if A = '1' and B = '0' then
           stan <= S1;
           elsif A = '0' and B = '1' then
              stan <= S10;
           else stan <= S0;
           end if;

- ********************************************************
- sciezka automatu dla detekcji ruchu w kierunku "zwieksz"
           when S1 =>
           if A = '1' and B = '1' then
              stan <= S2;
           elsif A = '1' and B = '0' then
              stan <= S1;
           else stan <= S0;
           end if;

           when S2 =>
           if A = '0' and B = '1' then
              stan <= S3;
           elsif A = '1' and B = '1' then
              stan <= S2;
           else stan <= S0;
           end if;

           when S3 =>
           if A = '0' and B = '0' then
              stan <= S4;
           elsif A = '0' and B = '1' then
              stan <= S3;
           else stan <= S0;
           end if;

           when S4 =>
              u_d_out <= '1';
              stan <= S5;

           when S5 =>
              clk_out <= '1';
              stan <= S0;

- ********************************************************
- sciezka automatu dla detekcji ruchu w kierunku "zmniejsz"
           when S10 =>
        if A = '1' and B = '1' then
              stan <= S20;
           elsif A = '0' and B = '1' then
              stan <= S10;
           else stan <= S0;
           end if;

           when S20 =>
           if A = '1' and B = '0' then
              stan <= S30;
           elsif A = '1' and B = '1' then
              stan <= S20;
           else stan <= S0;
           end if;

           when S30 =>
           if A = '0' and B = '0' then
              stan <= S40;
           elsif A = '1' and B = '0' then
              stan <= S30;
           else stan <= S0;
           end if;

           when S40 =>
              u_d_out <= '0';
              stan <= S50;

           when S50 =>
              clk_out <= '1';
              stan <= S0;
           when others =>
              stan <= S0;
        end case;
     end if;
  end process;
end arch;
```

identyfikuje stan. Jakkolwiek jawne przypisywanie wartości poszczególnym stanom wymaga nieco pracy, to łatwiejsze jest sprawdzenie poprawności opisu logicznego (weryfikacja opisu), ponieważ na podstawie wartości wektora *stan* łatwo określić, w którym stanie (w jakim miejscu grafu) znajduje się w danej chwili automat. Ze względu na zasadę działania, projektowany automat wyposażono w wejście zerowania (asynchroniczne) *res* - każdorazowo po włączeniu zasilania należy podać na nie krótki impuls o poziomie wysokim. Sprzętowe zerowanie wymusza przejście automatu do stanu S0, który jest stanem początkowym - automat pozostaje w nim do czasu zmiany poziomu na jednym z wejść: A lub B. W zależności od tego, na którym z nich poziom wysoki wystąpi jako pierwszy (co określa kierunek obracania osi enkodera) automat przechodzi do stanu S1 (kierunek zliczania „w górę") lub S10 (kierunek zliczania „w dół"). Począwszy od tych stanów rozpoczyna się kontrola drgań styków (stany inne od oczekiwanych powodują przejście automatu do stanu początkowego S0), a także generację sygnałów *u_d_out* (stany S4 lub S40) i *clk_out* (stany S5 lub S50). Przyjęty sposób opisu działania wyjść *u_d_out* i *clk_out* wymusza na syntezerze logicznym przypisanie im rejestrów - są one bowiem generowane synchronicznie z sygnałem zegarowym *clk*. Jest to rozwiązanie pozornie nadmiarowe, bo przecież obydwa sygnały można wytwarzać kombinacyjnie na podstawie wartości wektora *stan*, a co gorsza zajmowane są dwa dość cenne przerzutniki. Jak pokazały doświadczenia, w tym przypadku oszczędność się zupełnie nie opłaca, ponieważ przy rozwiązaniu kombinacyjnym trudno jest zlikwidować efekty hazardów, z powodu których na wyjściach mogą pojawiać się trudne do zlikwidowania zakłócenia szpilkowe. Wybrałem więc rozwiązanie nieco droższe, ale gwarantujące stabilną pracę układu.

Implementacja

Dzięki zastosowaniu jako języka opisu sprzętu VHDL-a, z prezentowanego projektu mogą sko-



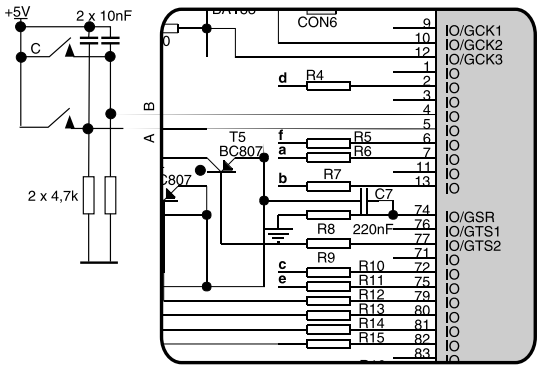
Rys. 4. Wyniki symulacji opisu z list. 1 uzyskane w programie ModelSim

rzystać fani dowolnej rodziny układów PLD. Do poprawnego zaimplementowania automatu niezbędne jest 6 przerzutników D lub JK, co w przypadku układów CPLD (jak np. MAX7000, czy XC9500) przekłada się „wprost" na liczbę zajętych makrokomórek.

Plik z opisem projektu (list. 1) był kompilowany za pomocą bezpłatnych pakietów: Max+Plus II Student Edition (obsługuje VHDL), Quartus II 2.1 Web Edition - obydwa firmy Altera i WebPack ISE 4.2 firmy Xilinx (zamieszczamy go także na płycie CD-EP11/2002B).

Na **rys. 4** pokazano wyniki symulacji funkcjonalnej projektu, wykonanej za pomocą programu ModelSim firmy Mentor Graphics. Skala czasu zaznaczona na tym rysunku jest umowna i nie odzwierciedla szybkości działania, w związku z czym nie należy się nią sugerować.

Weryfikację poprawności działania projektu przeprowadziłem na układzie CPLD XC95108 firmy Xilinx, który wchodzi w skład minizestawu ewaluacyjnego, który opisaliśmy w EP9/2002 (przy okazji opisu bloku IP - sterownika wyświetlacza multipleksowanego). Sposób dołączenia enkodera do wyprowadzeń układu U2 (zgodnie ze schematem z rys. 6, str. 30, EP9/2002) pokazano na **rys. 5**. Źródłem sygnału zegarowego dla automatu jest generator z układem U3 (j.w.), który jednocześnie spełnia rolę wzorca czasu dla wyświetlacza multipleksowanego. Sposób zrealizowania „dekodera" zapewnia małą zależność jakości jego pracy od wartości częstotliwości taktującej.



Rys. 5. Zalecany sposób dołączenia enkodera do układu XC95108

Działanie egzemplarza prototypowego sprawdziłem, wykorzystując dwa enkodery (ECW1J-B24-BE0012 - bez zapadek, ECW1J-B24-BC0006 - z zapadkami, obydwa firmy Bourns oraz CI-11CT-V1N31-HFACF firmy Piher) o różnej rozdzielczości. Dekoder działał bez zarzutu dla częstotliwości zegarowych z przedziału 400 Hz...1 kHz. Aby ułatwić przeprowadzenie testów, w układ PLD wbudowany został także 16-bitowy licznik góra-dół oraz sterownik wyświetlacza multipleksowanego, który został szczegółowo opisany we wrześniowym wydaniu EP.

Opisy w postaci źródłowej wszystkich niezbędnych modułów, pliki z testami funkcjonalnymi dla automatu oraz modułu licznika, a także kompletny system projektowy WebPack ISE 4.2 udostępniamy na płycie CD-EP11/2002B, a na naszej stronie WWW (*Download>Dokumentacje*) znajdą się pliki źródłowe, w tym także komplet plików tworzących projekt hierarchiczny w systemie WebPack ISE - gorąco polecam to narzędzie do nauki języka VHDL.

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

Uwaga! System WebPack ISE, znajdujący się na płycie CD-EP11/2002B, przed instalacją trzeba zarejestrować na stronie http://www.btc.pl/cd.htm.