Hexcalcul (2)

Implementacja kalkulatora kodu BCD na Hex w układzie programowalnym

Na przykładzie dosyć złożonego funkcjonalnie kalkulatora przeliczania kodu BCD na Hex, prezentujemy opis w języku VHDL układu do implementacji w strukturach programowalnych CPLD lub FPGA. W tej części opisujemy bloki funkcjonalne, będące komponentami głównego opisu kalkulatora w języku VHDL.

Zespół generatora sygnałów taktujących

Moduł ZL9PLD zawiera generator kwarcowy, którego sygnał jest używany do taktowania pracą całego zestawu. Poszczególne zespoły kalkulatora wymagają impulsów o różnych częstotliwościach. Zostały wyodrębnione trzy sygnały taktujące:

- automatu kalkulatora *SClk*,
- zespołu klawiatury *KClk*,
- zespołu wyświetlacza *DClk*.

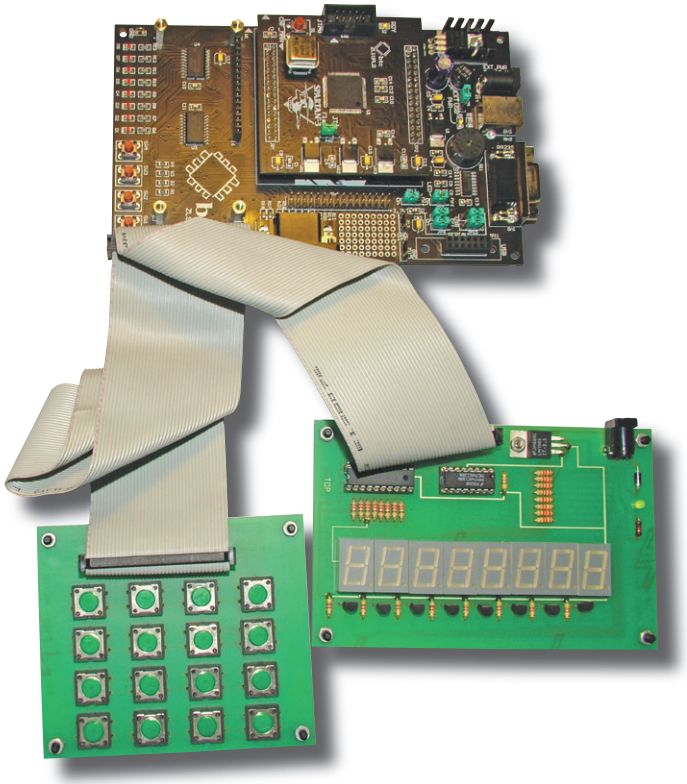
Sygnał zegarowy z zewnętrznego generatora jest doprowadzony do licznika synchronicznego. W wyniku zliczania generowane są sygnały zegarowe o częstotliwościach odpowiednio dobranych do potrzeb poszczególnych zespołów. Implementację generatora poszczególnych sygnałów taktujących w języku VHDL zawiera plik *clockgen.vhd*:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ClockGen is port ( Clk : in std_logic ;
                          Reset : in std_logic ;
                          DClk : out std_logic ;
                          KClk : out std_logic ;
                          SClk : out std_logic ) ;
end ClockGen ;

architecture Behavioral of ClockGen is
  signal ClockDivider : std_logic_vector ( 12 downto 0 ) ;
begin

  process ( Clk , Reset )
  begin
    if Reset = ‚0’ then
      ClockDivider <= „0000000000000” ;
    else
      if Clk’event and Clk = ‚1’ then
        ClockDivider <= ClockDivider + 1 ;
      end if ;
    end if ;
  end process ;

  DClk <= ClockDivider ( 12 ) ;
  KClk <= ClockDivider ( 10 ) ;
  SClk <= ClockDivider ( 4 ) ;

end Behavioral ;
```

Realizacja generatora sprowadza się więc do utworzenia licznika binarnego. Z wyjść odpowiednich stopni licznika pobierane są sygnały taktujące.

Zespół klawiatury

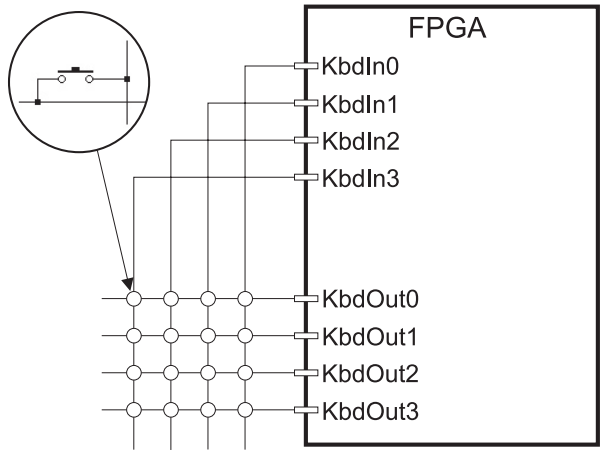
Schemat klawiatury został wcześniej przedstawiony na rys. 2. Klawisze są połączone w formie matrycy o czterech wierszach, po cztery klawisze w każdym. Doprowadzenie sygnałów wierszy i kolumn klawiatury do układu FPGA ilustruje **rys. 6**. W sytuacji, gdy nie jest naciśnięty żaden klawisz, na wejściach układu FPGA (sygnały *KbdInX*) występuje logiczna jedynka 9poziom wysoki). Ten poziom jest uzyskany wskutek odpowiedniego skonfigurowania wyprowadzeń układu (włączony jest rezystor *pull-up* – odpowiednie zapisy zawarte są w zbiorze *hexcalcul.ucf*).



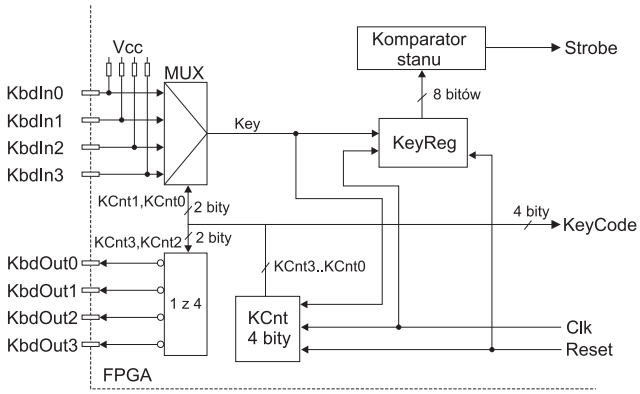
Na wyjściach *KbdOutX* układ FPGA generuje odpowiednie sygnały. W każdej danej tylko na jednym z wyjść występuje stan logicznego zera. Naciśnięcie przycisku powoduje połączenie linii *KbdOutX* z linią *KbnInX*. W wyniku tego połączenia poziom niski może być odczytany na odpowiednim wejściu układu FPGA (*KbdInX*).

Schemat blokowy zespołu obsługującego klawiaturę przedstawiono na **rys. 7**. Doprowadzony sygnał taktujący jest zliczany w 4-bitowym liczniku *Kcnt*. Część jego bitów jest użyta do utworzenia sygnału wyjściowego *KbdOut*. Pozostałe bity licznika służą są do wyselekcjonowania jednej z czterech linii wejściowych z klawiatury. Uzyskany w ten sposób sygnał *Key* zawiera informację, czy naciśnięty jest przycisk włączony pomiędzy wybrane linie *KbdOutX* i *KbdInX*.

W klawiaturach zbudowanych w oparciu o mikroprzyciski występuje niepożądane zjawisko „dzwonienia” (drgania) styków. Za eliminację tego efektu odpowiedzialny jest rejestr *KeyReg*. Pozwala na odczekanie odpowiedniego interwału czasu aż do uzyskania stabilnego



Rys. 6. Przyłączenie poszczególnych klawiszy do układu FPGA



Rys. 7. Obsługa klawiatury

stanu przycisku. Sygnał *Key* jest wprowadzany szeregowo do rejestru przesuwającego. Jeżeli przez osiem taktów zegarowych stan sygnału *Key* nie ulegnie zmianie, to można uznać, że osiągnięty został stan stabilny. Zakładając, że stan licznika *KCnt* (i dalej poprzez odpowiednie sygnały *KbdOutX* i *KbdInX*) adresuje naciśnięty przycisk, to sygnał *Key* przyjmuje stan logicznego zera. Odpowiedni komparator wykrywa moment wyzerowania rejestru przesuwającego i poprzez opadające zbocze sygnału *Strobe* sygnalizuje naciśnięcie przycisku na klawiaturze. Puszczenie przycisku spowoduje, że w rejestrze przesuwającym wystąpią jedynki i w konsekwencji wyjście komparatora (sygnał *Strobe*) przyjmie poziom wysoki.

Opisem w języku VHDL zespołu obsługi klawiatury jest plik *keyb4x4.vhd*.

Deklaracja jednostki projektowej zespołu:

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

entity Keyb4x4 is port ( Clk : in std_logic ;
            Reset : in std_logic ;
            KbdIn : in std_logic_vector ( 3 downto 0 ) ;
            KbdOut : out std_logic_vector ( 3 downto 0 ) ;
            KeyCode : out std_logic_vector ( 3 downto 0 ) ;
            Strobe : out std_logic ) ;
end Keyb4x4 ;
```

W *entity* występują następujące sygnały:

Clk – taktujący,

Reset – zerujący,

KbdIn – wejściowe z klawiatury,

KbdOut – wyjściowe do klawiatury,

KeyCode – stan licznika *KCnt* w momencie wykrycia naciśniętego przycisku,

Strobe – informujący o naciśnięciu klawisza.

Realizacja czterobitowego licznika, którego stan pozwala adresować przyciski w klawiaturze:

```
architecture Behavioral of Keyb4x4 is
  signal KCnt : std_logic_vector ( 3 downto 0 ) ;
  signal Key : std_logic ;
  signal KeyReg : std_logic_vector ( 7 downto 0 ) ;
begin

  process ( Clk , Reset , Key )
  begin
    if Reset = ‚0’ then
      KCnt <= „0000” ;
    else
      if Key = ‚1’ then
        if Clk’event and Clk = ‚1’ then
          KCnt <= KCnt + 1 ;
        end if ;
      end if ;
    end if ;
  end process ;

  with KCnt ( 3 downto 2 ) select
    KbdOut <= „0111” when „11” ,
              „1011” when „10” ,
              „1101” when „01” ,
              „1110” when others ;

  with KCnt ( 1 downto 0 ) select
    Key <= KbdIn ( 3 ) when „11” ,
           KbdIn ( 2 ) when „10” ,
           KbdIn ( 1 ) when „01” ,
           KbdIn ( 0 ) when others ;
```

Naciśnięcie klawisza powoduje, że sygnał *Key* przyjmuje stan logicznego zera, co jednocześnie blokuje zliczanie impulsów przez licznik i w konsekwencji stałe zaadresowanie klawisza (co jednocześnie oznacza, że rejestr przesuwający przetwarza informacje pochodzące z tego właśnie klawisza). Puszczenie klawisza uruchamia licznik i następuje oczekiwanie na naciśnięcie kolejnego klawisza.

W oparciu o dwa najbardziej znaczące bity licznika generowany jest na liniach wyjściowych do klawiatury odpowiedni przebieg sygnałów. Najmniej znaczące bity pozwalają, poprzez odpowiedni multiplekser, wybrać pojedynczą linię wejściową z klawiatury. Uzyskany sygnał *Key* określa stan zwarcia pomiędzy wybranymi liniami klawiatury.

Realizacja ośmiobitowego rejestru przesuwającego pozwalającego wykrywać stabilny stan naciśnięcia przycisku:

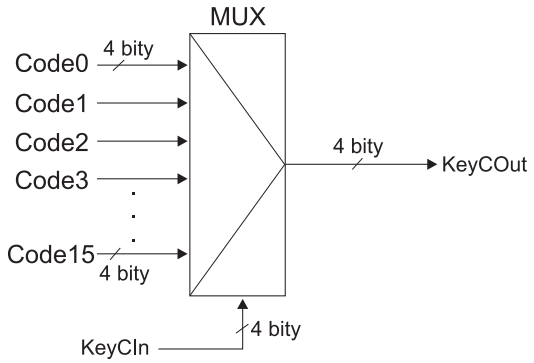
```
  process ( Clk , Reset )
  begin
    if Reset = ‚0’ then
      KeyReg <= „11111111” ;
    else
      if Clk’event and Clk = ‚1’ then
        KeyReg <= KeyReg ( 6 downto 0 ) & Key ;
      end if ;
    end if ;
  end process ;

  Strobe <= ‚0’ when KeyReg = „00000000” else ‚1’ ;
  KeyCode <= KCnt ;

end Behavioral ;
```

Ośmiobitowy komparator wykrywa same zera w rejestrze przesuwającym, zmieniając sygnał *Strobe* z poziomu wysokiego na niski (następuje zbocze opadające). Utrzymuje się on do czasu puszczenia przycisku.

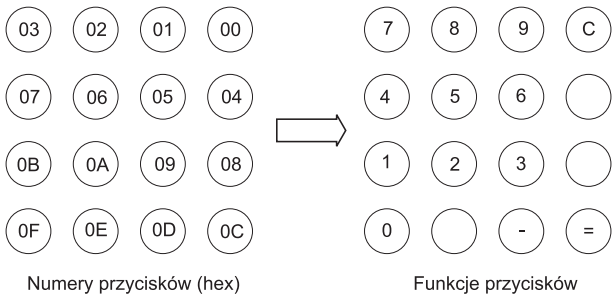
Uzyskany z komponentu (bloku) obsługi klawiatury fizyczny numer naciśniętego przycisku jest przekształcony na kod funkcji przyporządkowanej danemu przyciskowi. Przekodowanie realizowane jest w oparciu o zespół multiplekserowy pokazany na **rys. 8**. Na wejście adresowe doprowadzony jest 4-bitowy sygnał będący numerem przycisku, który powoduje, że na wyjściu multipleksera grupowego (16 multiplekserów 4-bitowych) uzyskany jest przyporządkowany mu kod.

W przedstawionym niżej sposobie przekodowania zrealizowane jest przyporządkowanie zilustrowane na **rys. 9**. Opisem tego komponentu w języku VHDL jest plik *keybencoder.vhd*:

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

entity KeybEncoder is port ( KeyCIn : in std_logic_vector (
3 downto 0 ) ;
                             KeyCOut : out std_logic_vector
```



Rys. 8. Zamiana numeru przycisku na jego kod



Rys. 9. Przyporządkowanie funkcji poszczególnym klawiszom (przyciskom)

```
( 3 downto 0 ) ) ;
end KeybEncoder ;

architecture Behavioral of KeybEncoder is
begin

  with KeyCIn select
    KeyCOut <= „1011” when „0000” ,   -- ,C’
               „1001” when „0001” ,   -- ,9’
               „1000” when „0010” ,   -- ,8’
               „0111” when „0011” ,   -- ,7’
               „1111” when „0100” ,   --
               „0110” when „0101” ,   -- ,6’
               „0101” when „0110” ,   -- ,5’
               „0100” when „0111” ,   -- ,4’
               „1111” when „1000” ,   --
               „0011” when „1001” ,   -- ,3’
               „0010” when „1010” ,   -- ,2’
               „0001” when „1011” ,   -- ,1’
               „1010” when „1100” ,   -- ,=’
               „1100” when „1101” ,   -- ,-’
               „1111” when „1110” ,   --
               „0000” when others ;   -- ,0’

end Behavioral ;
```

Można zauważyć na rys. 9, że niektóre klawisze nie mają przyporządkowanej funkcji. W tych przypadkach następuje przekodowanie na binarną wartość 1111, która jako nierozpoznana przez automat sterujący kalkulatorem zostanie zignorowana.

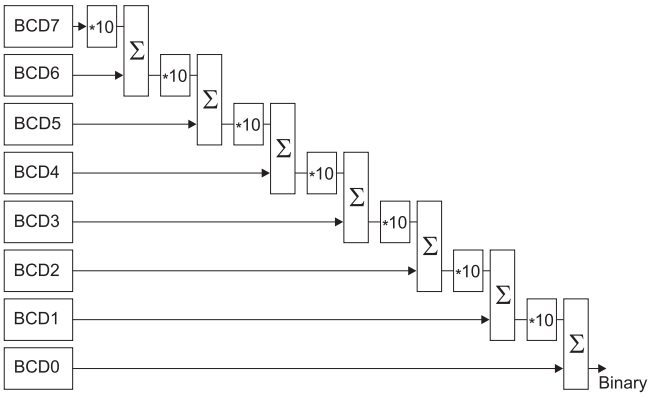
Zespół arytmetyczny

Zapisana w poszczególnych rejestrach liczba w kodzie BCD musi być przekształcona na liczbę w kodzie dwójkowym (NB) oraz ewentualnie jako liczba ujemna w kodzie uzupełnieniowym (U2). Przetworzenie jest zrealizowane w oparciu o operację mnożenia i sumowania. Schemat blokowy zespołu przedstawia **rys. 10**.

Poczynając od cyfry najbardziej znaczącej (zapisanej w rejestrze BCD7), jej wartość jest pomnożona przez 10. Do otrzymanego wyniku jest dodana zawartość kolejnego rejestru zawierającego następną mniej znaczącą cyfrę liczby. Powielając ten schemat działania na kolejne cyfry, aż do najmniej znaczącej, otrzymuje się algorytm przetwarzania zbioru ośmiu cyfr zapisanych w kodzie BCD (w ośmiu rejestrach) na liczbę w kodzie binarnym.

Występująca tu operacja mnożenia zrealizowana jest poprzez sumowanie. Korzystając z tożsamości:

$$10 * X = 8 * X + 2 * X$$



Rys. 10. Konwersja cyfr w kodzie BCD na liczbę binarną

Należy zsumować liczbę X przesuniętą o 3 pozycje bitowe z tą samą liczbą przesuniętą o jeden bit, jak pokazano na **rys. 11**.

Opisem w języku VHDL tego zespołu przetwarzającego jest plik *bcdtobin.vhd*:

Deklaracja jednostki projektowej konwertera kodu BCD na NB:

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

entity BcdToBin is port ( BCD0 : in std_logic_vector ( 3
downto 0 ) ;
                          BCD1 : in std_logic_vector ( 3
downto 0 ) ;
                          BCD2 : in std_logic_vector ( 3
downto 0 ) ;
                          BCD3 : in std_logic_vector ( 3
downto 0 ) ;
                          BCD4 : in std_logic_vector ( 3
downto 0 ) ;
                          BCD5 : in std_logic_vector ( 3
downto 0 ) ;
                          BCD6 : in std_logic_vector ( 3
downto 0 ) ;
                          BCD7 : in std_logic_vector ( 3
downto 0 ) ;
                          Binary : out std_logic_vector (
26 downto 0 ) ) ;
end BcdToBin ;
```

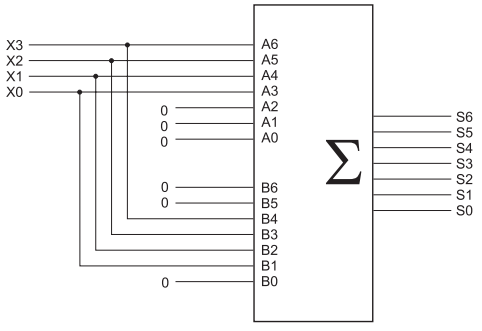
Deklarowane sygnały maja następujące znaczenie:

BCD0...BCD7 – wejściowe sygnały określające poszczególne cyfry w kodzie BCD jako 4-bitowe wektory,

Binary – 27-bitowy sygnał wyjściowy.

Deklaracja sygnałów wewnętrznych komponentu do zapamiętywania wyniku sumowania na kolejnych stopniach:

```
architecture Behavioral of BcdToBin is

  signal Sum6 : std_logic_vector ( 6 downto 0 ) ;     -- 99
dec = 63 hex -> 7 bit
  signal Sum5 : std_logic_vector ( 9 downto 0 ) ;     --
999 dec = 3E7 hex -> 10 bit
  signal Sum4 : std_logic_vector ( 13 downto 0 ) ;    --
9999 dec = 270F -> 14 bit
  signal Sum3 : std_logic_vector ( 16 downto 0 ) ;    --
99999 dec = 1869F hex -> 17 bit
  signal Sum2 : std_logic_vector ( 19 downto 0 ) ;    --
999999 dec = F423F hex -> 20 bit
```



Rys. 11. Mnożenie liczby 4-bitowej przez 10

```
  signal Sum1 : std_logic_vector ( 23 downto 0 ) ;     --
9999999 dec = 98967F hex -> 24 bit
  signal Sum0 : std_logic_vector ( 26 downto 0 ) ;     --
99999999 dec = 5F5E0FF hex -> 27 bit
```

Każdy kolejny stopień, pamiętający o jedną cyfrę dziesiętną więcej, wymaga większej liczby bitów. Końcowa suma, będąca wynikiem działania komponentu, potrzebuje do zapisu 27 bitów:

```
begin
  Sum6 <= ( BCD7 & „000” ) + ( „00” & BCD7 & „0” ) + (
„000” & BCD6 ) ;
-- 7 bit <=   7 bit         +          7 bit          +     7
bit

  Sum5 <= ( Sum6 & „000” ) + ( „00” & Sum6 & „0” ) + (
„000000” & BCD5 ) ;
-- 10 bit <=  10 bit        +         10 bit          +     10
bit
  Sum4 <= ( „0” & Sum5 & „000” ) + ( „000” & Sum5 & „0” ) +
( „0000000000” & BCD4 ) ;
-- 14 bit <=    14 bit           +         14 bit           +
14 bit
  Sum3 <= ( Sum4 & „000” ) + ( „00” & Sum4 & „0” ) + (
„0000000000000” & BCD3 ) ;
-- 17 bit <=      17 bit         +         17 bit           +
17 bit
  Sum2 <= ( Sum3 & „000” ) + ( „00” & Sum3 & „0” ) + (
„0000000000000000” & BCD2 ) ;
-- 20 bit <=     20 bit     +        20 bit           +
20 bit
  Sum1 <= ( „0” & Sum2 & „000” ) + ( „000” & Sum2 & „0” ) +
( „00000000000000000000” & BCD1 ) ;
-- 24 bit <=     24 bit           +         24 bit           +
24 bit
  Sum0 <= ( Sum1 & „000” ) + ( „00” & Sum1 & „0” ) + (
„000000000000000000000000” & BCD0 ) ;
-- 27 bit <=    27 bit      +       27 bit           +
27 bit

  Binary <= Sum0 ;
end Behavioral ;
```

W pierwszym stopniu z najbardziej znaczącymi dwiema cyframi operacja jest przeprowadzana na liczbach 7-bitowych. Liczba reprezentowana przez 4-bitowy sygnał BCD7 jest pomnożona przez 10 poprzez arytmetyczne zsumowanie jej z uzupełnieniem po młodszej stronie w postaci 3 bitów o wartości 0 (przesunięcie o 3 bity) z tą samą liczbą z uzupełnieniem jednobitowym po młodszej stronie (przesunięcie o 1 bit). Do powstałej sumy jest dodana liczba (z wypełnieniem zerami po starszej stronie do wymaganej liczby bitów) reprezentowana przez sygnał BCD6.

W ten sam sposób zrealizowane są kolejne stopnie z uwzględnieniem rosnącej liczby bitów, na jakiej wykonywane są operacje arytmetyczne. Końcowy wynik, jako sygnał *Binary*, jest dostępny na zewnątrz komponentu (bloku).

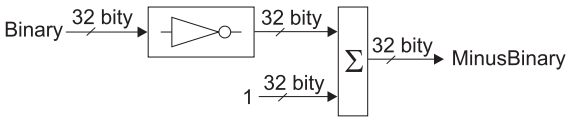
W przypadku konieczności przetwarzania liczb ujemnych występuje operacja uzupełnienia do podstawy (do 2 – kod U2). Realizując operację przekształcania do kodu U2 liczby zapisanej w kodzie naturalnym binarnym (NB), należy wykonać następujące działania:

- zanegować każdy bit liczby,
- do tak powstałej liczby dodać arytmetycznie 1.

W zapisie liczb w kodzie U2, bit na najbardziej znaczącej pozycji jest bitem znaku (liczby są dodatnie jeżeli bit znaku ma wartość zero, a ujemne w przeciwnym przypadku). Zauważmy, że maksymalna wartość ośmiocyfrowej, ujemnej liczby dziesiętnej wymaga do jej zapisu 28 bitów. Realizując operacje arytmetyczne dla liczb 32-bitowych, wyniki działań będą zawsze poprawne i nie zaistnieje przypadek błędnego wyniku spowodowanego przepełnieniem. Schemat blokowy układu konwersji przedstawiono na a **rys. 12**.

Odpowiadający schematowi opis w języku VHDL jest następujący: (plik *arthneg.vhd*):

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

entity ArthNeg is port ( InBinary : in std_logic_vector (
31 downto 0 ) ;
                         OutBinary : out std_logic_vector (
31 downto 0 ) ) ;
end ArthNeg ;

architecture Behavioral of ArthNeg is
```



Rys. 12. Zespół realizujący operację negacji uzupełnienia do 2 (U2)

```
  signal Tmp : std_logic_vector ( 31 downto 0 ) ;
begin

  Tmp <= not InBinary ;

  OutBinary <= Tmp + „00000000000000000000000000000001” ;

end Behavioral ;
```

Zespół wyświetlacza

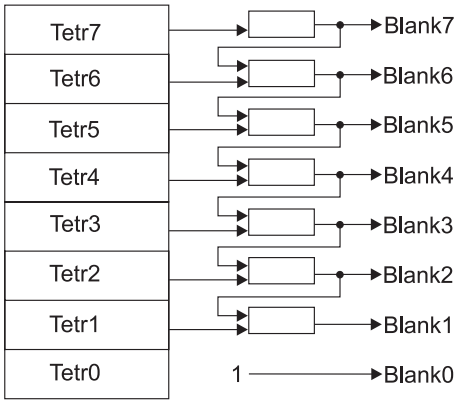
Do prezentacji przetwarzanych przez kalkulator danych (wprowadzanych z klawiatury oraz wynikowych) jest użyty ośmiocyfrowy wyświetlacz LED (rys. 3). Liczby są wyświetlane w systemie multipleksowym.

Zespół obsługi wyświetlacza wygasza wszystkie nieznaczące cyfry zer. Zespół odpowiedzialny za sterowanie wyświetlaczem, obok informacji dotyczącej wyświetlanej treści, generuje 8-bitowy sygnał *Blank* (po jednym bicie na każdą cyfrę wyświetlacza) określający, czy dana cyfra ma być wygaszona. Sygnał ten jest utworzony przez komponent *SetBlanks*. Schemat blokowy ilustrujący działanie komponentu przedstawiono na **rys. 13**. Zaczynając od tetrady najbardziej znaczącej, sygnał wygaszenia przyjmuje poziom niski (cyfra nie jest wyświetlana) w przypadku gdy wartość cyfry wynosi 0. Każda następna cyfra będzie wygaszana, jeżeli ma ona wartość zero, a sąsiednia bardziej znacząca cyfra jest również wygaszona. Najmniej znacząca cyfra jest zawsze wyświetlana bez względu na wartość.

Opis tego komponentu w języku VHDL jest w pliku *setblanks.vhd*.

Deklaracja jednostki projektowej komponentu wygaszania cyfr:

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

entity SetBlanks is port ( Tetr1 : in std_logic_vector ( 3
downto 0 ) ;
                           Tetr2 : in std_logic_vector ( 3
downto 0 ) ;
                           Tetr3 : in std_logic_vector ( 3
downto 0 ) ;
                           Tetr4 : in std_logic_vector ( 3
downto 0 ) ;
                           Tetr5 : in std_logic_vector ( 3
downto 0 ) ;
                           Tetr6 : in std_logic_vector ( 3
downto 0 ) ;
                           Tetr7 : in std_logic_vector ( 3
downto 0 ) ;
                           Blank : out std_logic_vector ( 7
downto 0 ) ) ;
end SetBlanks ;
```



Rys. 13. Sposób generowania sygnałów wygaszających cyfry na wyświetlaczu

W *entity* zadeklarowano następujące sygnały:

Tetr1...Tetr7 – sygnały ośmiu 4-bitowych liczb do przeanalizowania,

Blank – sygnał 8-bitowy wskazujący cyfry na wyświetlaczu, które mają być wygaszone.

```
architecture Behavioral of SetBlanks is
  signal TmpBlank : std_logic_vector ( 7 downto 0 ) ;

begin
  TmpBlank ( 7 ) <= ,0' when Tetr7 = „0000" else ,1' ;
  TmpBlank ( 6 ) <= ,0' when ( Tetr6 = „0000" and TmpBlank
( 7 ) = ,0' ) else ,1' ;
  TmpBlank ( 5 ) <= ,0' when ( Tetr5 = „0000" and TmpBlank
( 6 ) = ,0' ) else ,1' ;
  TmpBlank ( 4 ) <= ,0' when ( Tetr4 = „0000" and TmpBlank
( 5 ) = ,0' ) else ,1' ;
  TmpBlank ( 3 ) <= ,0' when ( Tetr3 = „0000" and TmpBlank
( 4 ) = ,0' ) else ,1' ;
  TmpBlank ( 2 ) <= ,0' when ( Tetr2 = „0000" and TmpBlank
( 3 ) = ,0' ) else ,1' ;
  TmpBlank ( 1 ) <= ,0' when ( Tetr1 = „0000" and TmpBlank
( 2 ) = ,0' ) else ,1' ;
  TmpBlank ( 0 ) <= ,1' ;

  Blank <= TmpBlank ;
end Behavioral
```

Najbardziej znacząca cyfra nie jest wyświetlana jeżeli jej wartość wynosi zero. Każda kolejna nie jest wyświetlana jeżeli jej wartość wynosi zero oraz sąsiednia starsza cyfra również jest wygaszona. Najmniej znacząca cyfra jest wyświetlana zawsze.

Schemat blokowy zespołu wyświetlacza przedstawiono na **rys. 14**. Wyświetlanie informacji odbywa się w trybie multipleksowym. Impulsy zegarowe (*DClk*, rys. 1) są zliczane w 3-bitowym liczniku tworząc sygnał *Cnt*, który określa jedną z ośmiu tertad, która ma być zobrazowana na wyświetlaczu.

Odpowiednio wysterowany multiplekser (wyjście MUX1, rys. 14) dostarcza 4-bitową liczbę do dekodera, którego zadaniem jest przetworzenie jej na kod wyświetlacza 7-segmentowego. Jednocześnie zostaje wygenerowany sygnał *Blank* (wyjście MUX2) określający czy aktualnie wyświetlana cyfra ma być wygaszona na wyświetlaczu. Sygnał *Blank* sterując pracą multipleksera MUX3 powoduje, że na jego wyjściu występują same jedynki (wszystkie segmenty wyświetlanej cyfry są wygaszone) lub wartość określająca, które segmenty mają być włączone do zobrazowania aktualnie wyświetlanej cyfry. W torze znajduje się dodatkowo blok, którego zadaniem jest generowanie sygnałów pozwalających na wyświetlenie znaku minus.

Poniżej omówimy opis zespołu wyświetlacza w języku VHDL (plik *hexdispl.vhd*).

Deklaracja jednostki projektowej zespołu wyświetlacza:

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

entity LedDispl is port ( Clk : in std_logic ;
                          Reset : in std_logic ;
                          Tetr0 : in std_logic_vector ( 3
downto 0 ) ;
                          Tetr1 : in std_logic_vector ( 3
downto 0 ) ;
                          Tetr2 : in std_logic_vector ( 3
downto 0 ) ;
                          Tetr3 : in std_logic_vector ( 3
downto 0 ) ;
                          Tetr4 : in std_logic_vector ( 3
downto 0 ) ;
                          Tetr5 : in std_logic_vector ( 3
downto 0 ) ;
                          Tetr6 : in std_logic_vector ( 3
downto 0 ) ;
                          Tetr7 : in std_logic_vector ( 3
downto 0 ) ;
                          Blank : in std_logic_vector ( 7
downto 0 ) ;
                          Signum : in std_logic ;
                          Segm : out std_logic_vector ( 7
downto 0 ) ;
                          Dig : out std_logic_vector ( 2
downto 0 ) ) ;
end LedDispl ;
```

W *entity* zadeklarowano następujące sygnały:

Clk – zegarowy sterujący procesem wyświetlania,

Reset – zerowania,

Tert0...Tert7 – ośmiu czterobitowych liczb do wyświetlenia,

Blank – ośmiobitowy, określający cyfry, które mają być wygaszone na wyświetlaczu,

Signum – określający, czy wyświetlana liczba jest ujemna,

Segm – wyjściowy do sterowania poszczególnymi segmentami wyświetlacza 7-segmentowego,

Dig – wyjściowy określający w kodzie binarnym aktualnie wyświetlaną cyfrę.

Deklaracja sygnałów wewnętrznych zespołu wyświetlacza:

```
architecture Behavioral of LedDispl is

  signal Cnt : std_logic_vector ( 2 downto 0 ) ;
  signal TmpTetr : std_logic_vector ( 3 downto 0 ) ;
  signal CurrBlank : std_logic ;
  signal TmpSegm : std_logic_vector ( 6 downto 0 ) ;
  signal SegData : std_logic_vector ( 7 downto 0 ) ;
  signal MSBSegData : std_logic_vector ( 7 downto 0 ) ;
  signal MSBDigit : std_logic ;
  signal DotSegm : std_logic ;
```

Najistotniejsze z zadeklarowanych są następujące sygnały:

Cnt – wyjściowy licznika, którego stan określa aktualnie wyświetlaną tetradę,

CurrBlank – określając, czy aktualnie wyświetlana cyfra ma być wygaszona,

SegData – 8-bitowy sygnał sterujący wyświetlaczem 7-segmentowym (7 segmentów i kropka).

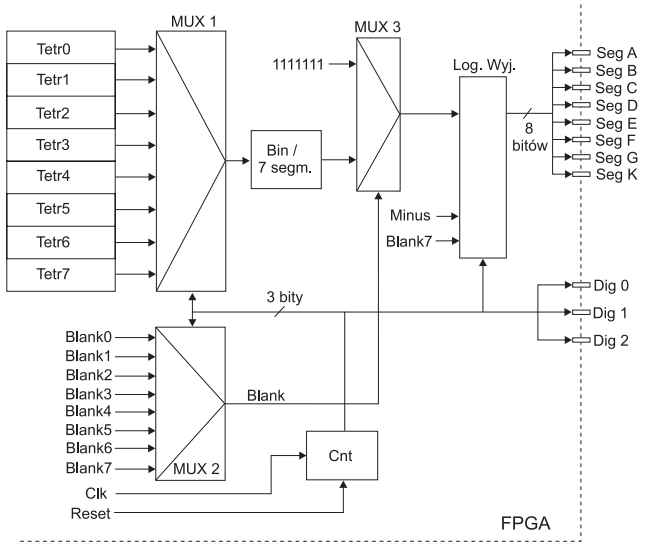
Implementacja 3-bitowego licznika zliczającego impulsy *Clk* doprowadzone do zespołu wyświetlacza:

```
begin
  process ( Clk , Reset )
  begin
    if Reset = ,0' then
      Cnt <= „000" ;
    else
      if Clk'event and Clk = ,1' then
        Cnt <= Cnt + 1 ;
      end if ;
    end if ;
  end process ;
```

Opis multipleksera (MUX1, rys. 14), którego zadaniem jest wybranie na podstawie stanu licznika (sygnał *Cnt*) kodu dwójkowego (tetrady) aktualnie wyświetlanej cyfry:

```
  with Cnt select
    TmpTetr <= Tetr7 when „111" ,
               Tetr6 when „110" ,
               Tetr5 when „101" ,
               Tetr4 when „100" ,
               Tetr3 when „011" ,
               Tetr2 when „010" ,
               Tetr1 when „001" ,
               Tetr0 when others ;
```

Następnie kod tej tertrady zostaje przekształcony na 7-bitowy sygnał sterujący poszczególnymi segmentami wyświetlacza. Ponieważ stoso-



Rys. 14. Zespół wyświetlacza

wane są wyświetlacze ze wspólną anodą, to świecenie odpowiedniego segmentu dekodera następuje przy wartości zero (poziom niski) na wyjściu segmentowym dekodera, a przy wartości jeden dany segment jest wygaszony.

```
  with TmpTetr select
    TmpSegm <= „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 „1010” , -- A
               „0000011” when „1011” , -- b
               „1000110” when „1100” , -- C
               „0100001” when „1101” , -- d
               „0000110” when „1110” , -- E
               „0001110” when „1111” , -- F
               „1000000” when others ; -- 0
```

Opis multipleksera (MUX2, rys. 14), którego zadaniem jest określenie, czy aktualnie wyświetlana cyfra ma być wygaszona:

```
  with Cnt select
    CurrBlank <= Blank ( 7 ) when „111” ,
                 Blank ( 6 ) when „110” ,
                 Blank ( 5 ) when „101” ,
                 Blank ( 4 ) when „100” ,
                 Blank ( 3 ) when „011” ,
                 Blank ( 2 ) when „010” ,
                 Blank ( 1 ) when „001” ,
                 Blank ( 0 ) when others ;

  with CurrBlank select
    SegData <= Point & TmpSegm when ,1’ ,
               „11111111” when others ;
```

Uzyskany sygnał *CurrBlank* steruje kolejnym multiplekserem grupowym MUX3. Na jego wyjściu mogą wystąpić same jedynki (co oznacza wygaszenie wszystkich segmentów) lub odpowiedni kod 7-segmentowy, będący wynikiem przetworzenia kodu binarnego wyświetlanej cyfry.

Ustalenie sygnału wyświetlania znaku minus:

```
  MSBDigit <= ,1’ when ( Cnt = „111” ) else ,0’ ;
  DotSegm <= not Signum when ( Cnt = „111” ) else ,1’ ;
  MSBSegData <= „10111111” when ( Blank ( 7 ) = ,0’ and
Signum = ,1’ ) else
                DotSegm & TmpSegm when ( CurrBlank = ,1’ )
else „11111111” ;
  with MSBDigit select
    Segm <= MSBSegData when ,1’ ,
            SegData when others ;

  Dig <= Cnt ;

end Behavioral ;
```

Zespół, którego zadaniem jest zobrazowanie na wyświetlaczu znaku minus, modyfikuje sygnały sterujące poszczególnymi segmentami wyświetlacza tylko w sytuacji, gdy aktualne operacje dotyczą najbardziej znaczącej cyfry (sygnał *MSBDigit* przyjmuje poziom wysoki, gdy stan licznika *Cnt* jest równy “111”). Ewentualne zobrazowanie znaku minus na wyświetlaczu nastąpi w sytuacji, gdy wyświetlana jest najbardziej znacząca cyfra oraz żądane jest wyświetlenie znaku (doprowadzony do komponentu sygnał *Signum* przyjmuje wartość 1). W przypadku, gdy wyświetlana liczba nie zajmuje ośmiu pozycji znakowych, znak minus jest zobrazowany jako włączony segment G. W przeciwnym wypadku, gdy wyświetlana liczba jest ośmiocyfrowa, znak minus jest zobrazowany przez segment kropki.

Stan licznika określającego aktualnie wyświetlaną cyfrę zostaje przekazany na zewnątrz komponentu i dalszej kolejności poprzez wyprowadzenia układu FPGA dostarczony do zespołu wyświetlacza LED.

Przedstawiony projekt kalkulatora może być rozwijany przez Czytelników. Można przykładowo pokusić się o implementację funkcji kasowania omyłkowo wprowadzonej ostatniej cyfry dziesiętnej. Wymagałoby to określenia dodatkowego stanu dla automatu sterującego, w którym wykonywana byłaby operacja odpowiedniego przepisywania danych w zbiorze rejestrów pamiętających wprowadzone cyfry dziesiętne.

Mam nadzieję, ze uważne przestudiowanie opisu VHDL’owego tego projektu przyczyni się do pogłębienia przez Czytelników znajomości języka VHDL i jest dobrą ilustracją metodyki tworzenia złożonych projektów, z zastosowaniem opisu strukturalnego, czyli dekompozycji złożonego układu na elementarne bloki funkcjonalne.

Andrzej Pawluczuk
apawluczuk@vp.pl