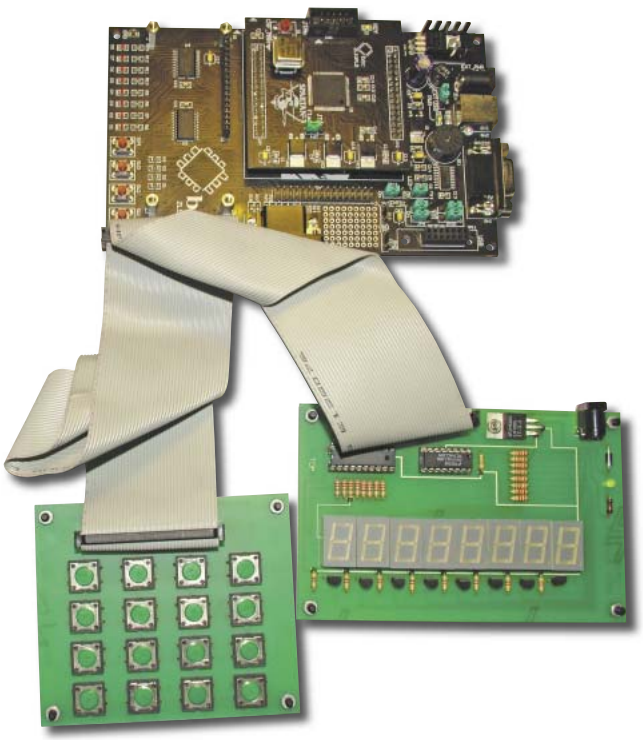
Hexcalcul (1)

Implementacja kalkulatora kodu BCD na Hex w układzie programowalnym

Na przykładzie dosyć złożonego funkcjonalnie kalkulatora, chcemy zaprezentować sposób opisu strukturalnego w języku VHDL układu do implementacji w strukturach programowalnych CPLD lub FPGA. Mamy nadzieję, że po uważnym przestudiowaniu VHDL'owego opisu projektu, Czytelnicy zainteresowani tą tematyką pogłębią swoją znajomość języka VHDL.

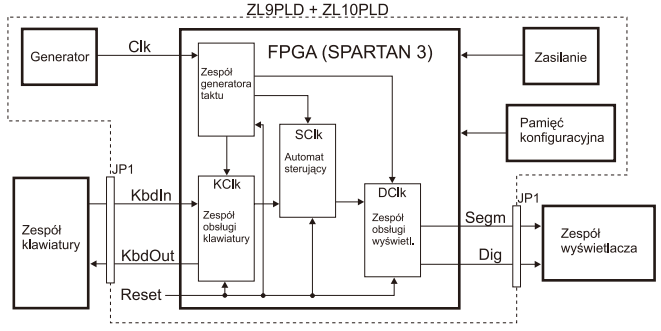


We współczesnych układach programowalnych jest możliwa realizacja bardzo złożonych projektów, które jeszcze nie tak dawno wymagałyby użycia dużej liczby układów scalonych. Przykładem ilustrującym możliwości układów FPGA jest projekt specjalizowanego kalkulatora, którego zadaniem jest przeliczanie liczb z zapisu dziesiętnego na zapis szesnastkowy. Może być bardzo pomocny podczas opracowywania oprogramowania dla mikrokontrolerów, których wewnętrzny świat jest oparty o system dwójkowy. Jednak w praktyce stosowany jest zapis liczb w systemie szesnastkowym. Ważną cechą kalkulatora jest możliwość przeliczania liczb ujemnych. W systemach mikroprocesorowych zazwyczaj jest stosowana notacja w kodzie U2 (uzupełnienie do 2), dlatego w kalkulatorze również ją zastosowano.

Budowa kalkulatora

Do budowy kalkulatora użyto modułu ZL9PLD z układem FPGA (SPARTAN 3 firmy Xilinx) oraz płyty bazowej ZL10PLD, w której jest instalowany. Dokumentacja tego układu oraz oprogramowanie do syntezy własnych projektów są dostępne na stronie *www.xilinx.com*.

Zastosowanie gotowych modułów do eksperymentowania pozwala uniknąć problemów technologicznych, które mogą okazać się trudne do pokonania w warunkach domowych (jak przykładowo wykonanie obwodów drukowanych zawierających układy o gęstym rastrze wyprowadzeń). Ponadto zestaw ZL9PLD + ZL10PLD zawiera wiele elementów wchodzących w skład zaprojektowanego kalkulatora. Można tu wymienić przykładowo: obwody zasilania (zastosowany układ FPGA wymaga kilku napięć zasilających), pamięć konfiguracyjną, złącze JTAG do programowania. Niestety wyposażenie zestawu ZL9PLD nie jest wystarczające do budowy kalkulatora bez dodatkowych elementów. Oznacza to konieczność dobudowywania dwóch zespołów: klawiatury i wyświetlacza, które są przyłączane poprzez złącze JP1 znajdujące się w zestawie ZL9PLD (**rys. 1**).



Rys. 1. Schemat blokowy kalkulatora

Schemat zespołu klawiatury przedstawiono na **rys. 2**. Składa się ona z 16 przycisków połączonych w matrycę oraz złącza P101 przeznaczonego do przyłączenia jej do zestawu bazowego FPGA.

Układ FPGA generując odpowiednie sygnały na wyprowadzeniach wierszy matrycy klawiatury (wyprowadzenia 8, 20, 22 i 24 złącza P101 na rys. 2), a testując sygnały z kolumn (wyprowadzenia 26, 28, 30 i 32 złącza P101), może wykryć naciśnięcie każdego przycisku z matrycy.

Drugim zespołem jest ośmiocyfrowy 7-segmentowy wyświetlacz LED, który również jest przyłączony do modułu bazowego za pomocą złącza P1. Jego schemat przedstawiono na **rys. 3**.

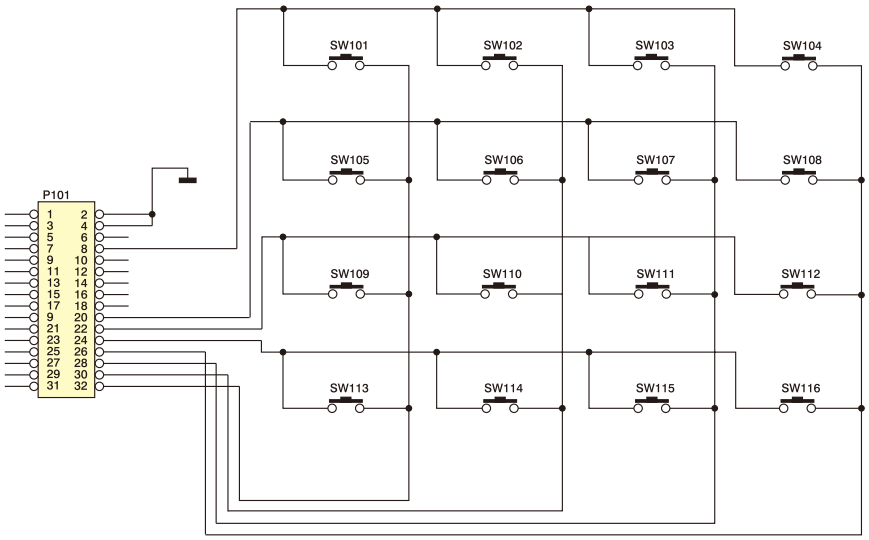
Jest to typowe rozwiązanie stosowane w wielocyfrowych wyświetlaczach pracujących w trybie multipleksowym. Poprzez złącze P1 doprowadzone są odpowiednie sygnały do układu U1 określające świecące segmenty oraz sygnały do układu U2 sterujące włączaniem odpowiedniej cyfry w wyświetlaczu.

Wewnątrz układu FPGA zaimplementowano następujące zespoły funkcjonalne (rys. 1):

– generatora impulsów taktujących,

– obsługi klawiatury matrycowej,
– obsługi wyświetlacza,
– realizacji wymaganych operacji wraz z automatem sterującym.

Najistotniejszym blokiem kalkulatora jest właśnie automat sterujący. Jego zadaniem jest, w oparciu o sygnały z klawiatury, odpowiednie sterowanie 7-segmentowym wyświetlaczem LED. Zespół obsługi wyświetlacza generuje sygnały sterujące w celu właściwego zobrazowania informacji na wyświetlaczach. Współpraca automatu sterującego kalkulatora z zespołem klawiatury zilustrowano na **rys. 4**. Operator może nacisnąć przycisk w dowolnym momencie, w wyniku czego zostanie wygenerowany sygnał *Strobe* (rys. 4) związany z tym zdarzeniem. Sygnał ten służy do wpisania flagi sygnalizującej zdarzenie (sygnał *KbdStatus*) do odpowiedniego przerzutnika, który w odpowiednim czasie, wynikającym z pracy automatu sterującego, będzie testowany. Po wykonaniu operacji wynikającej z naciśniętego przycisku, automat sygnałem *Clear* zeruje wskaźnik *KbdStatus*. Takie rozwiązanie uniezależnia działanie automatu od czasu trzymania naciśniętego przycisku oraz zapobiega wielokrotnej interpretacji jednorazowo naciśniętego przycisku.

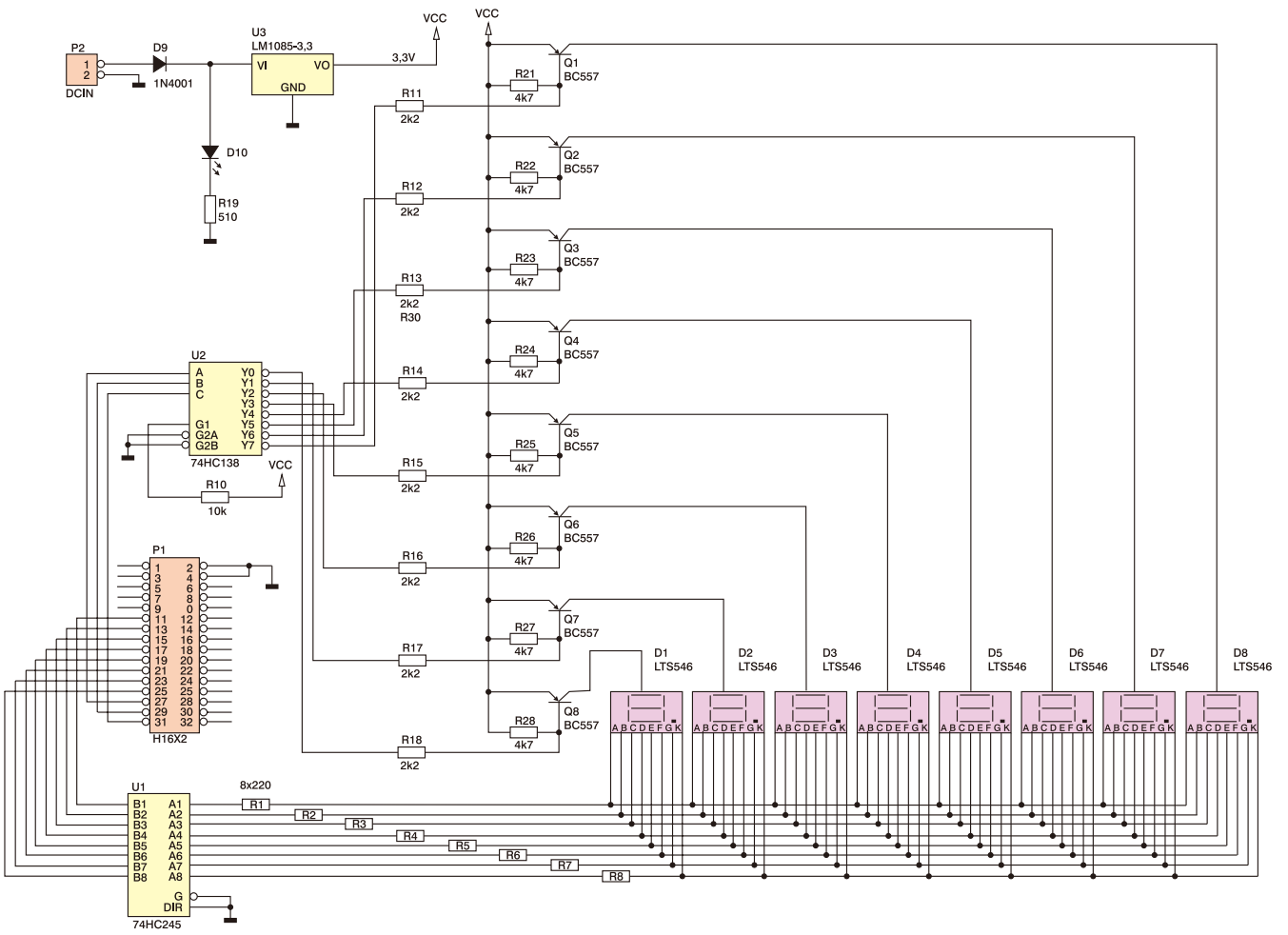


Rys. 2. Schemat elektryczny klawiatury

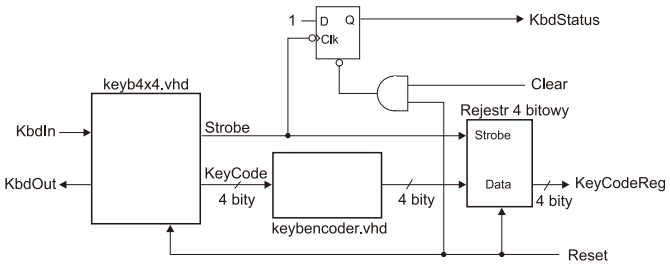
Automat sterujący kalkulatorem, w wyniku wykrycia zdarzenia naciśnięcia klawisza, wykonuje odpowiednie czynności w oparciu o 4-bitowy kod uzyskany z rejestru *KeyCodeReg*. Realizowane polecenia mają następujące kody:

– od 0 do 9 – oznacza naciśnięcie przycisku cyfry dziesiętnej,
– A hex – oznacza naciśnięcie przycisku „=”,
– B hex – oznacza naciśnięcie przycisku „C”,
– C hex – oznacza naciśnięcie przycisku „–”.

Aby uniezależnić funkcję przypisaną klawiszowi od jego położenia na płytce klawiatury, realizowane jest odpowiednie przekodowanie kodu klawisza. Przed wpisaniem kodu do rejestru *KeyCodeReg*, stan rejestru *KeyCode* zawierający fizyczny numer naciśniętego przycisku na klawiaturze jest przetworzony w bloku *keybencoder* (rys. 4), w którym do rejestru *KeyCodeReg* zamiast fizycznego numeru przycisku zostaje wpisany przyporządkowany mu kod. Takie rozwiązanie pozwala w elastyczny sposób przyporządkować poszczególnym przyciskom klawiatury odpowiednie funkcje.



Rys. 3. Zespół wyświetlacza



Rys. 4. Współpraca automatu kalkulatora z zespołem klawiatury

Opis kalkulatora w VHDL

Głównym blokiem kalkulatora jest automat sterujący. Jest on odpowiedzialny na realizację wszystkich wykonywanych operacji, sterowanie przepływem danych ,oraz synchronizację pracy poszczególnych zespołów. Dla automatu sterującego zostały określone następujące stany:

IdleState – stan oczekiwania automatu na polecenie,

DecodeKeyState – stan, w którym dokonywane jest dekodowanie informacji uzyskanej z klawiatury,

DigitStrState – stan, w którym wykonywane są czynności związane z wprowadzeniem kolejnej cyfry dziesiętnej,

ClrDispState – stan, w którym czyszczony jest wyświetlacz i kalkulator jest przygotowywany do wprowadzenia kolejnej liczby dziesiętnej z klawiatury,

MinusState – stan, w którym realizowane są operacje w wyniku naciśnięcia znaku minus,

RunState – stan, w którym kalkulator przechodzi do wyświetlenia wyniku operacji,

AccKeyState – stan, w którym automat zeruje flagę sygnalizującą wykrycie naciśnięcia przycisku.

Działanie automatu jest zgodne z grafem przejść pokazanym na **rys. 5**. Zewnętrzny sygnał zerowania ustawia automat w stan początkowy *IdleState*. W tym stanie testowany jest wskaźnik sygnalizujący naciśnięcie jakiegokolwiek przycisku na klawiaturze (sygnał *KbdStatus*). W zależności od wartości tego sygnału, automat pozostaje nadal w tym stanie lub przechodzi do następnego (*DecodeKeyState*), którego zadaniem jest rozpoznanie funkcji przypisanej naciśniętemu klawiszowi oraz w dalszej kolejności wykonanie niezbędnych operacji związanych z tą funkcją. W trakcie dekodowania funkcji, oprócz kodu uzyskanego z klawiatury, brany jest pod uwagę wskaźnik związany z aktualnym kontekstem. Jeżeli nie został naciśnięty przycisk „=" (*RunContext*=0), to automat znajduje się w kontekście wprowadzania polecenia (podobnie jak w typowych kalkulatorach można użyć klawiszy cyfr, „–", „=" itp.). W przeciwnym przypadku (*RunContext*=1), na wyświetlaczu prezentowana jest informacja wynikowa oraz automat może wykonać jedynie operację związaną z klawiszem „C" (wyzerować wszystkie rejestry i przejść do kontekstu związanego z wprowadzaniem nowego polecenia). Poziom sygnału *RunContext* jest zmieniany po naciśnięciu klawisza „=" albo „C'. W wyniku zdekodowania każdej dopuszczalnej kombinacji zawartej w rejestrze *KeyCodeReg*, automat przechodzi do odpowiedniego stanu, w którym wykonane są właściwe operacje i w dalszej kolejności do stanu *AccKeyState* w celu wyzerowania wskaźnika sygnalizującego wykrycie kolejnego naciśnięcia klawisza. Po bezwarunkowym przejściu automatu do stanu *IdleState*, następuje oczekiwanie na naciśnięcie klawisza.

Poniżej omówimy szczegółowo opis kalkulatora w języku VHDL – poszczególne fragmenty listingu stanowią jeden plik *hexcalcul.vhd* (płyta CD):

Deklaracja jednostki projektowej (*entity*) projektu:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity HexCalcul is port ( Clk : in std_logic;
        Reset : in std_logic;
        Segm : out std_logic_vector ( 7 downto 0);
        KbdIn : in std_logic_vector ( 3 downto 0);
        KbdOut : out std_logic_vector ( 3 downto 0);
        Dig : out std_logic_vector ( 2 downto 0));
end HexCalcul;
```

Sygnały wymienione w *entity* są przyporządkowane do wyprowadzeń układu FPGA (szczegółowe informacje są zawarte w pliku *hexcalcul.ucf* – płyta CD). Deklarowane nazwy sygnałów mają następujące znaczenie:

Clk – wejściowy sygnał taktujący,

Reset – wejściowy sygnał zerujący, ustawia wszystkie liczniki i rejestry w stan początkowy,

KbdIn – wektor wejściowych sygnałów z klawiatury,

KbdOut – wektor wyjściowych sygnałów do klawiatury,

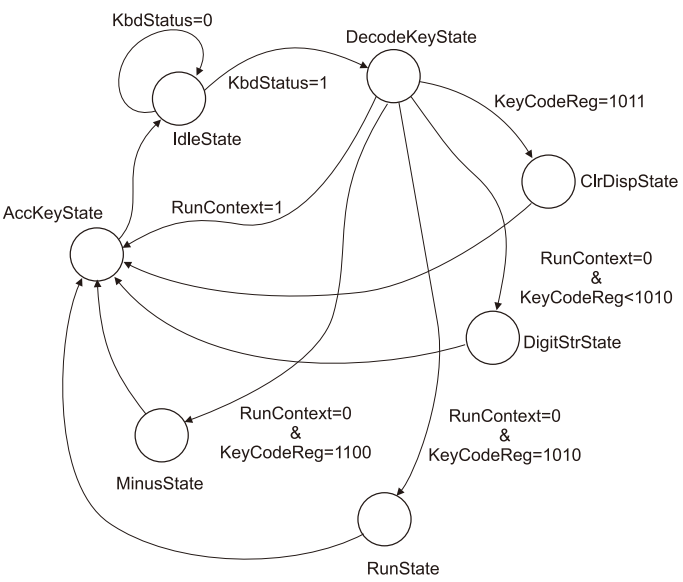
Segm – wektor sygnałów do sterowania poszczególnymi segmentami w wyświetlaczu LED,

Dig – wektor sygnałów sterujących włączaniem poszczególnych cyfr w wyświetlaczu.

Deklaracja w architekturze komponentów, które są używane w projekcie:

```
architecture Behavioral of HexCalcul is
  component ClockGen port (Clk : in std_logic;
        Reset : in std_logic;
        DClk : out std_logic;
        KClk : out std_logic;
        SClk : out std_logic);
  end component;
  component Keyb4x4 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 component;

  component BcdToBin 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 component;
  component LedDispl 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 component;
  component KeybEncoder port (KeyCIn : in std_logic_vector
(3 downto 0);
        KeyCOut : out std_logic_vector (3 downto 0));
  end component;
```



Rys. 5. Graf przejść automatu sterującego

```
  component SetBlanks 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 component;
  component ArthNeg port ( InBinary : in std_logic_vector
(31 downto 0);
OutBinary : out std_logic_vector (31 downto 0));
  end component;
```

Poszczególne komponenty mają następujące zadania:

ClockGen – wygenerowanie sygnałów zegarowych dla zespołów kalkulatora,

Keyb4x4 – obsługa klawiatury,

KeybEncoder – przyporządkowanie poszczególnym przyciskom klawiatury ich kodów,

BcdToBin – konwersja zbioru cyfr zapisanych w kodzie BCD na liczbę w kodzie binarnym,

ArthNeg – realizacja negacji kodu dwójkowego liczby,

LedDispl – obsługa 7-segmentowego wyświetlacza LED,

SetBlanks – realizacja wygaszania nieznaczących zer.

Definicja typu *CalculStateType* do opisu działania automatu kalkulatora:

```
  type CalculStateType is ( IdleState ,
        DecodeKeyState ,
        DigitStrState ,
        ClrDispState ,
        MinusState ,
        RunState ,
        AccKeyState ) ;
```

Automat taktowany sygnałem zegarowym *SysClk*, przyjmuje kolejne stany zgodnie z grafem pokazanym na rys. 5.

Deklaracja sygnałów wewnętrznych użytych w opisie kalkulatora:

```
  signal CalculState : CalculStateType ;
  signal KbdClk : std_logic ;
  signal DisplClk : std_logic ;
  signal SysClk : std_logic ;
  signal DispBus : std_logic_vector ( 31 downto 0 ) ;
  signal MpxBinary : std_logic_vector ( 31 downto 0 ) ;
  signal Binary : std_logic_vector ( 31 downto 0 ) ;
  signal MinusBinary : std_logic_vector ( 31 downto 0 ) ;
  signal KeyNum : std_logic_vector ( 3 downto 0 ) ;
  signal KeyCodeReg : std_logic_vector ( 3 downto 0 ) ;
  signal KeyCode : std_logic_vector ( 3 downto 0 ) ;
  signal Strobe : std_logic ;
  signal Clear : std_logic ;
  signal Minus : std_logic ;
  signal DispSign : std_logic ;
  signal RunContext : std_logic ;
  signal KbdStatus : std_logic ;
  signal BCD0 : std_logic_vector ( 3 downto 0 ) ;
  signal BCD1 : std_logic_vector ( 3 downto 0 ) ;
  signal BCD2 : std_logic_vector ( 3 downto 0 ) ;
  signal BCD3 : std_logic_vector ( 3 downto 0 ) ;
  signal BCD4 : std_logic_vector ( 3 downto 0 ) ;
  signal BCD5 : std_logic_vector ( 3 downto 0 ) ;
  signal BCD6 : std_logic_vector ( 3 downto 0 ) ;
  signal BCD7 : std_logic_vector ( 3 downto 0 ) ;
  signal Blank : std_logic_vector ( 7 downto 0 ) ;
```

Użycie komponentu generującego sygnały zegarowe dla poszczególnych zespołów:

```
begin
  ClkGenInstance : ClockGen port map ( Clk => Clk ,
        Reset => Reset ,
        DClk => DisplClk ,
        KClk => KbdClk ,
        SClk => SysClk ) ;
```

Sygnał *DispClk* jest sygnałem taktującym układu sterującego wyświetlaczem LED, sygnał *KbdClk* steruje obsługą klawiatury, a sygnał *SysClk* jest użyty do taktowania automatu sterującego.

Użycie komponentów obsługi klawiatury:

```
  KeybInstance : Keyb4x4 port map ( Clk => KbdClk ,
        Reset => Reset ,
        KbdIn => KbdIn ,
        KbdOut => KbdOut ,
        KeyCode => KeyNum ,
        Strobe => Strobe ) ;
  KeybStatInstance : KeybEncoder port map ( KeyCIn =>
KeyNum ,
        KeyCOut => KeyCode ) ;
  KeybRegInstance : process ( Strobe , Reset )
  begin
    if Reset = '0' then
      KeyCodeReg <= "0000" ;
    else
      if Strobe'event and Strobe = '0' then
        KeyCodeReg <= KeyCode ;
      end if ;
    end if ;
  end process ;

  KeybStatusInstance : process ( Strobe , Reset , Clear )
  begin
    if Reset = '0' or Clear = '0' then
      KbdStatus <= '0' ;
    else
      if Strobe'event and Strobe = '0' then
        KbdStatus <= '1' ;
      end if ;
    end if ;
  end process ;
```

W komponencie *Keyb4x4* sygnały *KbdIn* i *KbdOut* są doprowadzone do wyprowadzeń układu FPGA. Sygnał *KeyNum* określa numer przyciśniętego klawisza przetwarzany dalej na odpowiedni kod (w *KeybEncoder.vdh*) zwracając wynik w postaci sygnału *KeyCode*). Sygnał *Strobe* informuje o wykryciu naciśniętego klawisza.

Uzyskany z komponentów obsługujących klawiaturę kod naciśniętego przycisku jest zapisany do rejestru *KeyCodeReg*. Wpis następuje opadającym zboczem sygnału *Strobe*. Jednocześnie ustawiany jest jednobitowy rejestr *KeybStatus*, który sygnalizuje dla automatu sterującego użycie klawiatury. Automat po wykonaniu akcji związanej z naciśniętym przyciskiem ma możliwość zerowania tego rejestru sygnałem *Clear*.

Użycie komponentów do wykonywania operacji arytmetycznych:

```
  ConvInstance : BcdToBin port map ( BCD0 => BCD0 ,
        BCD1 => BCD1 ,
        BCD2 => BCD2 ,
        BCD3 => BCD3 ,
        BCD4 => BCD4 ,
        BCD5 => BCD5 ,
        BCD6 => BCD6 ,
        BCD7 => BCD7 ,
        Binary => Binary ( 26 downto 0 ) ) ;
  Binary ( 31 downto 27 ) <= „00000" ;
  NegInstance : ArthNeg port map ( InBinary => Binary ,
        OutBinary => MinusBinary ) ;
  with Minus select
   MpxBinary <= MinusBinary when ,1' ,
                Binary when others ;
  with RunContext select
    DispSign <= Minus when ,0' ,
                ,0' when others ;
  with RunContext select
    DispBus <= MpxBinary when ,1' ,
               BCD7 & BCD6 & BCD5 & BCD4 & BCD3 & BCD2 &
BCD1 & BCD0 when others ;
```

Aktualna liczba w kodzie BCD, pamiętana w ośmiu rejestrach o symbolach od *BCD0* do *BCD7*, jest przetwarzana na liczbę dwójkową. Maksymalna wartość liczby dziesiętnej, możliwa do zapamiętania (99999999), jest przetwarzana na 27-bitową liczbę dwójkową. Pozostałe bity uzupełnienia do liczby 32-bitowej są wyzerowane. Wynik przetworzenia, jako sygnał *Binary,* jest jednocześnie zanegowany i reprezentowany przez sygnał *MinusBinary*.

W zależności od stanu przerzutnika *Minus*, który jest modyfikowany przez automat na podstawie informacji uzyskanych z klawiatury, binarnym sygnałem wynikowym jest *Binary* lub *MinusBinary* (multipleksowany w zależności od wartości sygnału *Minus*). Blok multipleksera grupowego (32 multipleksery 2-bitowe), w zależności od wartości sygnału *RunContext*, przesyła na szynę sterującą wyświetlaczem (sygnał *DispBus*) dane wejściowe (jako złożenie w odpowiedniej kolejności wszystkich sygnałów z rejestrów przechowujących poszczególne cyfry w kodzie BCD) lub wynik przetworzenia binarnego (jako sygnał *Binary* lub *MinusBinary*). Jednocześnie zostaje określony sygnał *DispSign* do sygnalizacji znaku dla liczb ujemnych.

Użycie komponentów i przygotowanie danych do ich wyświetlenia na wyświetlaczu LED:

```
  BlankCheckInstance : SetBlanks port map ( Tetr1 =>
DispBus ( 7 downto 4 ) ,
   Tetr2 => DispBus ( 11 downto 8 ) ,
   Tetr3 => DispBus ( 15 downto 12 ) ,
   Tetr4 => DispBus ( 19 downto 16 ) ,
   Tetr5 => DispBus ( 23 downto 20 ) ,
   Tetr6 => DispBus ( 27 downto 24 ) ,
   Tetr7 => DispBus ( 31 downto 28 ) ,
```

```
 Blank => Blank ) ;
HexDisplInstace : LedDispl port map ( Clk => DisplClk ,
 Reset => Reset ,
 Tetr0 => DispBus ( 3 downto 0 ) ,
 Tetr1 => DispBus ( 7 downto 4 ) ,
 Tetr2 => DispBus ( 11 downto 8 ) ,
 Tetr3 => DispBus ( 15 downto 12 ) ,
 Tetr4 => DispBus ( 19 downto 16 ) ,
 Tetr5 => DispBus ( 23 downto 20 ) ,
 Tetr6 => DispBus ( 27 downto 24 ) ,
 Tetr7 => DispBus ( 31 downto 28 ) ,
 Blank => Blank ,
 Signum => DispSign ,
 Segm => Segm ,
 Dig => Dig ) ;
```

Uzyskany z odpowiednich muliplekserów sygnał *DispBus* jest doprowadzony do komponentu (bloku) określającego, które pozycje cyfr wyświetlacza mają być wygaszone oraz do komponentu samego wyświetlacza (łącznie z uzyskaną informacja o wygaszeniach). Każdej tetradzie odpowiada cyfra w kodzie BCD lub kolejne 4 bity (jako cyfra szesnastkowa) z binarnego wyniku.

Implementacja automatu sterującego pracą kalkulatora:

```
CalcAutInstance : process ( SysClk , Reset )
begin
  if Reset = ,0’ then
    CalculState <= IdleState ;
    Minus <= ‘0’ ;
    BCD7 <= “0000” ;
    BCD6 <= “0000” ;
    BCD5 <= “0000” ;
    BCD4 <= “0000” ;
    BCD3 <= “0000” ;
    BCD2 <= “0000” ;
    BCD1 <= ”0000” ;
    BCD0 <= ”0000” ;
    RunContext <= ’0’ ;
  else
    if SysClk’event and SysClk = ’1’ then
      case CalculState is
```

Zewnętrzny sygnał zerujący ustawia stan początkowy automatu oraz zeruje wybrane sygnały wewnętrzne komponentu. Po operacji zerowania, przerzutniki związane z sygnałami *Minus* i *RunContext* są wyzerowane, co odpowiada informacji, że wprowadzana liczba jest dodatnia, a na wyświetlaczu są wyświetlane wprowadzane z klawiatury cyfry (kalkulator znajduje się w trybie wprowadzania danych). Wyzerowane są również rejestry pamiętające wprowadzone cyfry dziesiętne.

Po narastającym zboczu sygnału zegarowego *SysClk* rozpatrywane są przypadki, jakie mogą wystąpić dla automatu sterującego:

```
when IdleState =>
  Clear <= ,1’ ;
  if KeybStatus = ,1’ then
    CalculState <= DecodeKeyState ;
  else
    CalculState <= IdleState ;
  end if ;
when DecodeKeyState =>
  Clear <= ,1’ ;
  if KeyCodeReg = „1011” then
   CalculState <= ClrDispState ;
  else
   if RunContext = ,1’ then
    CalculState <= AccKeyState ;
    elsif KeyCodeReg < „1010” then
      CalculState <= DigitStrState ;
    elsif KeyCodeReg = „1010” then
      CalculState <= RunState ;
    elsif KeyCodeReg = „1100” then
      CalculState <= MinusState ;
    else
      CalculState <= AccKeyState ;
    end if ;
  end if ;
```

W stanie *IdleState* sprawdzana jest flaga sygnalizująca wykrycie naciśniętego klawisza (sygnał *KeybStatus*) i w zależności od wyniku sprawdzenia automat pozostaje w stanie *IdleState* lub przechodzi do zdekodowania informacji związanej z użytym klawiszem (stan *DecodeKeyState*).

W stanie *DecodeKeyState*, w zależności od kodu znaku zapisanego w rejestrze *KeyCodeReg*, zostanie wykonane jedno z możliwych poleceń. Jeżeli naciśniętemu klawiszowi odpowiada kod związany z operacją zerowania rejestrów wyświetlacza, to automat przechodzi do stanu *ClrDispState*. W przeciwnym przypadku, jeżeli aktualnie na wyświetlaczu prezentowany jest wynik, to automat przechodzi do stanu związanego z kasowaniem flagi sygnalizującej użycie klawiatury (nie wykona żadnej operacji). Oznacza to, że w trybie prezentacji na wyświetlaczu wyniku działania, jedynym możliwym do użycia klawiszem jest „C”, którego naciśnięci oznacza przejście do trybu wprowadzania danych wejściowych. Jeżeli nie zaistniał żaden z powyższych przypadków, to aktualny tryb umożliwia wprowadzanie liczb z klawiatury. Dopuszczalne klawisze są związane z cyframi dziesiętnymi (automat przechodzi do stanu *DigitStrState*), klawiszem „=” (automat przechodzi do stanu *RunState*) lub klawiszem „–” (automat przechodzi do stanu *MinusState*). W przypadku, gdy został naciśnięty każdy inny przycisk, automat przechodzi do gaszenia flagi związanej z klawiaturą (nie wszystkie klawisze mają przyporządkowaną funkcję i ich użycie jako nierozpoznane zostaje zignorowane).

```
when DigitStrState =>
  Clear <= ,1’ ;
  BCD7 <= BCD6 ;
  BCD6 <= BCD5 ;
  BCD5 <= BCD4 ;
  BCD4 <= BCD3 ;
  BCD3 <= BCD2 ;
  BCD2 <= BCD1 ;
  BCD1 <= BCD0 ;
  BCD0 <= KeyCodeReg ;
  CalculState <= AccKeyState ;
```

W stanie *DigitStrState*, do rejestru związanego z najmniej znaczącą cyfrą dziesiętną zostaje wpisana nowa wprowadzona z klawiatury. Wprowadzone dotychczas cyfry zostają przepisane na sąsiednią bardziej znaczącą pozycję, a cyfra najbardziej znacząca zostaje utracona. Po wykonaniu odpowiednich wpisów w rejestrach, automat przechodzi do stanu gaszenia wskaźnika z klawiatury i oczekiwania na kolejne polecenie.

```
when MinusState =>
  Clear <= ,1’ ;
  CalculState <= AccKeyState ;
  Minus <= not Minus ;
```

W stanie *MinusState*, zostaje zmieniony stan przerzutnika *Minus* na przeciwny i automat przechodzi do stanu, w którym jest zerowany wskaźnik z klawiatury.

```
when RunState =>
  RunContext <= ,1’ ;
  Clear <= ,1’ ;
  CalculState <= AccKeyState ;
```

W stanie *RunState*, zostaje zmieniona wartość sygnału *RunContext*. Od tej pory na wyświetlaczu prezentowana jest liczba wynikowa. Po wyzerowaniu flagi z klawiatury, automat będzie oczekiwał na kolejne polecenie.

```
when ClrDispState =>
  RunContext <= ,0’ ;
  Clear <= ,1’ ;
  BCD7 <= „0000” ;
  BCD6 <= „0000” ;
  BCD5 <= „0000” ;
  BCD4 <= „0000” ;
  BCD3 <= „0000” ;
  BCD2 <= „0000” ;
  BCD1 <= „0000” ;
  BCD0 <= „0000” ;
  Minus <= ,0’ ;
  CalculState <= AccKeyState ;
```

W stanie *ClrDispState*, zostają wyzerowane rejestry pamiętające wprowadzone cyfry dziesiętne, ustawiony jest tryb umożliwiający wprowadzanie nowych danych oraz ustalenie, że nowe dane będą dotyczyć liczby dodatniej. Po wyzerowaniu flagi z klawiatury automat będzie oczekiwał na kolejne polecenia.

```
    when AccKeyState =>
      Clear <= ,0’ ;
      CalculState <= IdleState ;
    when others =>
      Clear <= ,1’ ;
      CalculState <= IdleState ;
  end case ;
 end if ;
end if ;
end process ;
end Behavioral ;
```

W stanie *AccKeyState* wyzerowany jest przerzutnik, którego stan sygnalizuje naciśnięcie klawisza. Po wykonaniu tej operacji, automat bezwarunkowo przechodzi do stanu, w którym oczekuje na kolejne polecenie (*IdleState*).

W części 2. opiszemy poszczególne komponenty kalkulatora, to jest bloki: sygnałów taktujących, obsługi klawiatury, arytmetycznego i obsługi wyświetlacza.

Andrzej Pawluczuk
apawluczuk@vp.pl