Sebastian PAWLAK
UNIWERSYTET ZIELONOGÓRSKI, INSTYTUT INFORMATYKI I ELEKTRONIKI

Implementacja arytmometru do obliczania sum korelacyjnych w układach FPGA

mgr inż. Sebastian PAWLAK

Autor artykułu ukończył studia na kierunku Elektrotechnika, specjalizacji Inżynieria Systemów Komputerowych na Uniwersytecie Zielonogórskim, gdzie pracuje obecnie na stanowisku asystenta. Przygotowuje się także do otwarcia przewodu doktorskiego. Głównym obszarem zainteresowań badawczych Autora jest wykorzystanie układów FPGA do akceleracji obliczeń w systemach przetwarzania obrazów oraz w analizie sygnałów biomedycznych.

e-mail: S.Pawlak@iie.uz.zgora.pl

Streszczenie

Analiza sygnałów biomedycznych jest szybko rozwijającą się dziedziną nauki. Jedną z obiecujących metod w tej dziedzinie jest wykorzystanie wymiaru korelacyjnego szeregu czasowego interwałów międzyuderzeniowych serca (szeregu RR) do wnioskowania o stanie zdrowia pacjenta. W artykule przedstawiono wyniki implementacji arytmometru do obliczania sum korelacyjnych w układach FPGA.

Abstract

Biomedical signals analysis is one of the fastest growing fields of research. Use of correlation dimension of the R-R intervals time series is a very promising method of patient health assessment. This paper describes a correlation sum calculation unit implementation in Field Programmable Gate Arrays (FPGA).

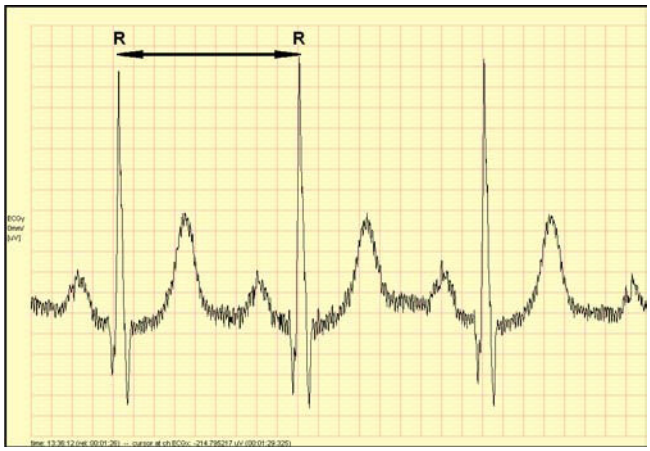
Słowa kluczowe: suma korelacyjna, arytmometr, układy FPGA.

Keywords: correlation sum, calculation unit, FPGA devices.

1. Wstęp

Sumy korelacyjne są elementem wszystkich algorytmów obliczających wymiar korelacyjny (fraktalny) [1, 2] analizowanych sygnałów oraz wielu innych algorytmów stosowanych przy obliczaniu parametrów złożoności, na przykład różnych rodzajów entropii [1, 2, 3]. Algorytmy te stosowane są w rozważaniach teoretycznych dotyczących różnych znanych układów złożonych mających swoje źródło w zagadnieniach dynamiki nieliniowej oraz w problemach praktycznych, na przykład szukania ukrytych cech układów złożonych o nieznanej strukturze [2, 3, 4].

Jednym z takich zagadnień praktycznych jest analiza sygnałów biomedycznych, a w szczególności szeregu czasowego tak zwanych odstępów RR, czyli szeregu odstępów czasowych pomiędzy kolejnymi załamkami R w elektrokardiogramie [2, 4, 5]. Na rys.1. przedstawiono przykładowy elektrokardiogram z zaznaczonym interwałem RR.



Rys. 1. Przykładowy elektrokardiogram z zaznaczonym interwałem RR
Fig. 1. Sample electrocardiogram with RR interval

2. Suma korelacyjna

Suma korelacyjna (1) jest przybliżeniem całki korelacji (2).

$$C_m(\varepsilon) = \frac{1}{[N-(m-1)\tau][N-(m-1)\tau-1]} \times \times \sum_{i=1}^{N-(m-1)\tau} \sum_{j=1}^{N-(m-1)\tau} \Theta\left(\varepsilon - \left\|\vec{x}_i - \vec{x}_j\right\|\right) \quad (1)$$

$$C_m(\varepsilon) = \lim_{N\to\infty} \frac{1}{N^2} \times \sum_{i,j=1}^{N} \Theta\left(\varepsilon - \left\|\vec{x}_i - \vec{x}_j\right\|\right), \vec{x}_i \in R^m \quad (2)$$

m – liczba wymiarów przestrzeni fazowej,
ε – odległość progowa, N – liczba stanów $\vec{x}_i$,
$\| \|$ – norma euklidesowa, Θ – funkcja skokowa Heaviside'a.

Funkcja skokowa Heaviside'a wyrażona jest wzorem (3).

$$\Theta(z) = \begin{cases} 0, & z \le 0 \\ 1, & z > 0 \end{cases} \quad (3)$$

Norma euklidesowa wyrażona jest wzorem (4).

$$\left\|\vec{x}_i - \vec{x}_j\right\| = \sqrt{\sum_{k=1}^{m} (x_{i,k} - x_{j,k})^2} \quad (4)$$

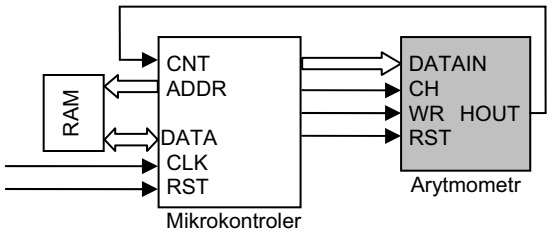
Gdy dostępny jest jedynie szereg czasowy y, wówczas przestrzeń fazowa może być zrekonstruowana za Twierdzeniem Takens'a [6] zgodnie ze wzorem (5).

$$\vec{x}_i = (y_i, y_{i+\tau}, y_{i+2\tau}, \ldots, y_{i+(m-1)\tau}), \quad i = 1,\ldots, N-(m-1)\tau \quad (5)$$

τ – odstęp czasowy, N – liczba elementów szeregu czasowego.

3. Architektura systemu

Na rys. 2. przedstawiono architekturę systemu do obliczania sum korelacyjnych.



Rys. 2. Architektura systemu
Fig. 2. System architecture

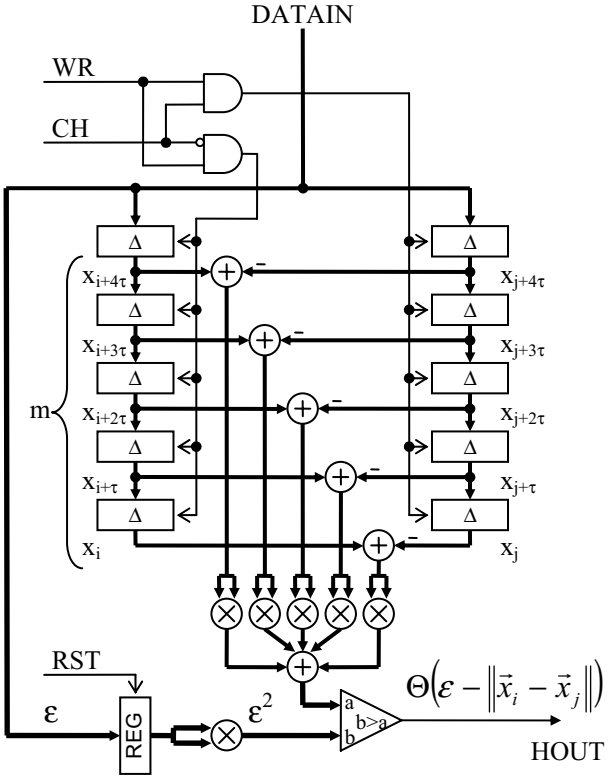
System składa się z mikrokontrolera pełniącego rolę zarządcy i arytmometru. Mikrokontroler inicjalizuje arytmometr poprzez wystawienie stanu niskiego na sygnał *RST* arytmometru. Przy narastającym zboczu sygnału *RST* arytmometr zapamiętuje wartość ε wystawioną przez mikrokontroler na port *DATAIN*. Następnie mi-

krokontroler przy użyciu wejścia *CH* wybiera zestaw rejestrów (dla wektora x_i lub x_j). W kolejnych *m*-cyklach mikrokontroler poprzez port *DATAIN* ładuje do rejestrów wewnętrznych arytmometru wektor x_i. Następnie zmienia stan sygnału *CH* i w kolejnych *m*-cyklach ładuje elementy wektora x_j. Arytmometr wyznacza wartość funkcji (6) i przekazuje ją do mikrokontrolera przy każdym kolejnym cyklu zapisu do arytmometru. Gdy funkcja przyjmie wartość jedynki logicznej, mikrokontroler inkrementuje zawartość licznika.

$$\Theta\left(\varepsilon-\left\|\vec{x}_i-\vec{x}_j\right\|\right)=\left\{\begin{matrix}0,\varepsilon-\left\|\vec{x}_i-\vec{x}_j\right\|\le 0\\1,\varepsilon-\left\|\vec{x}_i-\vec{x}_j\right\|>0\end{matrix}\right\}=$$

$$=\left\{\begin{matrix}0,\varepsilon\le\left\|\vec{x}_i-\vec{x}_j\right\|\\1,\varepsilon>\left\|\vec{x}_i-\vec{x}_j\right\|\end{matrix}\right\}=\left\{\begin{matrix}0,\varepsilon^2\le\sum_{k=1}^{m}\left(x_{i,k}-x_{j,k}\right)^2\\1,\varepsilon^2>\sum_{k=1}^{m}\left(x_{i,k}-x_{j,k}\right)^2\end{matrix}\right. \quad (6)$$

Na rys. 3. przedstawiono schemat blokowy arytmometru.



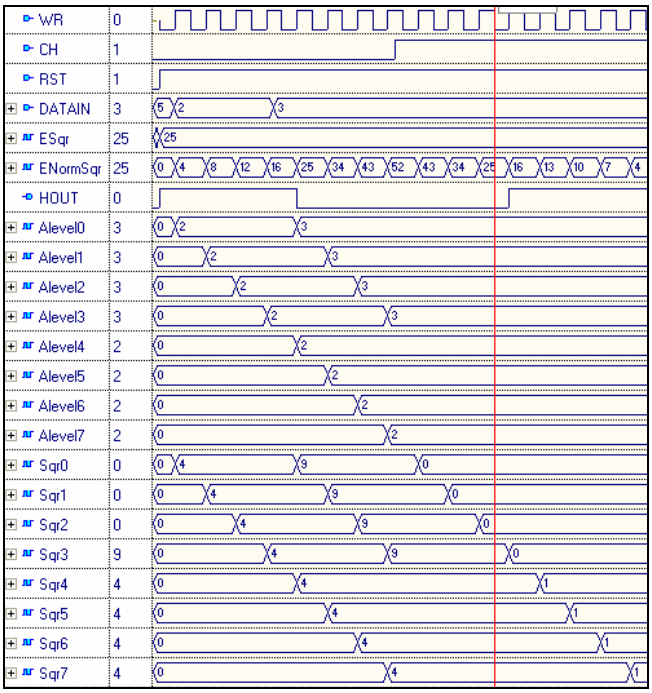
Rys. 3. Schemat blokowy arytmometru
Fig. 3. Block diagram of calculation unit

4. Synteza i implementacja

Do syntezy i implementacji wykorzystano pakiet oprogramowania Xilinx ISE 9.1i. Jako środowisko symulacyjne posłużył pakiet Aldec ActiveHDL 7.3. Syntezę przeprowadzono dla trzech typów układów FPGA: - Xilinx Virtex2 Pro [7], - Xilinx Virtex5 SXT [8], - Xilinx Spartan 3 [9]. Na rys. 4. przedstawiono fragmenty syntezowalnego kodu w języku opisu sprzętu VHDL, opisującego arytmometr. Dla każdego typu układu syntezę przeprowadzono dwukrotnie – z wykorzystaniem wbudowanych układów mnożących i bez ich wykorzystania.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_SIGNED.all;
use IEEE.STD_LOGIC_ARITH.all;
entity arithm is
  generic(
    N : Integer := 4; -- data width
    M : Integer := 8; -- depth
    P : Integer := 11 -- 2N + ]log2(M)[
  );
  port(
    WR     : in STD_LOGIC;
    CH     : in STD_LOGIC;
    RST    : in STD_LOGIC; -- active low
    DATAIN : in STD_LOGIC_VECTOR(N-1 downto 0);
    HOUT   : out STD_LOGIC
  );
end arithm;
architecture arithm of arithm is
signal A0 : STD_LOGIC_VECTOR(N-1 downto 0);
signal B0 : STD_LOGIC_VECTOR(N-1 downto 0);
signal Sum0 : STD_LOGIC_VECTOR(N-1 downto 0);
signal Sqr0 : STD_LOGIC_VECTOR(2*N-1 downto 0);
signal ESqr : STD_LOGIC_VECTOR(2*N-1 downto 0);
signal ENormSqr : STD_LOGIC_VECTOR(P-1 downto 0);
begin
  process (RST, DATAIN) is
  begin
    if RST'event and RST='1' then
      ESqr <= DATAIN * DATAIN;
    end if;
  end process;
  process (WR, CH, RST, DATAIN) is
  begin
    if RST='1' then
      if WR'event and WR='1' then
        if CH='1' then
          Blevel7 <= Blevel6;
          Blevel6 <= Blevel5;
          Blevel5 <= Blevel4;
          Blevel4 <= Blevel3;
          Blevel3 <= Blevel2;
          Blevel2 <= Blevel1;
          Blevel1 <= Blevel0;
          Blevel0 <= DATAIN;
        else
          Alevel7 <= Alevel6;
          Alevel6 <= Alevel5;
          Alevel5 <= Alevel4;
          Alevel4 <= Alevel3;
          Alevel3 <= Alevel2;
          Alevel2 <= Alevel1;
          Alevel1 <= Alevel0;
          Alevel0 <= DATAIN;
        end if;
      end if;
    else
      Blevel7 <= (others => '0');
    end if;
  end process;
  Sum0 <= Alevel0 - Blevel0;
  Sqr0 <= Sum0 * Sum0;
  process (RST, Sqr0, Sqr1, Sqr2, Sqr3, Sqr4, Sqr5,
Sqr6, Sqr7) is
  begin
    if RST = '0' then
      ENormSqr <= (others => '0');
    else
      ENormSqr <= CONV_STD_LOGIC_VECTOR(
        CONV_INTEGER(Sqr0) +
        CONV_INTEGER(Sqr1) +
        CONV_INTEGER(Sqr2) +
        CONV_INTEGER(Sqr3) +
        CONV_INTEGER(Sqr4) +
        CONV_INTEGER(Sqr5) +
        CONV_INTEGER(Sqr6) +
        CONV_INTEGER(Sqr7), P);
    end if;
  end process;
  process (ESqr, ENormSqr) is
  begin
    if ESqr > ENormSqr then
      HOUT <= '1';
    else
      HOUT <= '0';
    end if;
  end process;
end arithm;
```

Rys. 4. Kod VHDL opisujący arytmometr
Fig. 4. VHDL code description of calculation unit



Rys. 5. Działanie arytmometru
Fig. 5. Calculation unit operation

Działanie układu arytmometru ilustrują przebiegi czasowe sygnałów pokazane na rys. 5. Układ rozpoczyna działanie gdy wystąpi zbocze narastające sygnału *RST*. W tym samym momencie zatrzaskiwana jest wartość *ESqr*, czyli kwadrat odległości progowej. Gdy sygnał wyboru grupy rejestrów *CH* przyjmie wartość zera logicznego, do rejestrów *Alevel0*-*Alevel7* w kolejnych ośmiu cyklach zapisu *WR* trafiają z wejścia *DATAIN* elementy wektora x_i. Następnie na wejście *CH* podawana jest jedynka logiczna w celu wybrania grupy rejestrów *Blevel0*-*Blevel7*. W kolejnych cyklach sygnału *WR*, wczytywane są z wejścia *DATAIN* elementy wektora x_j. Jednocześnie obliczane są sumy cząstkowe (niewidoczne na wykresie wielobitowe sygnały *Sum0*-*Sum7*), które są następnie podnoszone do kwadratu (wielobitowe sygnały *Sqr0*-*Sqr7*). *ENormSqr* jest sumą kwadratów sum cząstkowych, czyli normą euklidesową wektorów x_i i x_j podniesioną do kwadratu. Jest ona nieustannie porównywana z kwadratem wartości progowej (wielobitowy sygnał *ESqr*). Jeśli *ESqr* jest większe od *ENormSqr*, sygnał *HOUT* przyjmuje wartość jedynki logicznej, w przeciwnym wypadku zera logicznego. Odpowiada to wartości funkcji Heaviside'a różnicy wartości progowej ε i normy euklidesowej wektorów x_i i x_j.

W tabeli 1 przedstawiono wykorzystanie zasobów sprzętowych podczas implementacji arytmometru z wykorzystaniem sprzętowych układów mnożących. Można zauważyć, że arytmometr zajmuje jednakową liczbę zasobów w rodzinach układów Virtex2Pro i Spartan3, co świadczyć może o tym, że elementy LUT, Slice i mnożarki sprzętowe mają w tych układach zbliżoną, jeśli nie jednakową strukturę.

Tab. 1. Wykorzystanie zasobów sprzętowych – sprzętowe układy mnożące
Tab. 1. Device resources utilization – hardware multipliers

	xc2vp30	xc5vsx50t	xc3s400
Slices	331	626	331
Flip Flops	256	256	256
LUTs	402	404	402
MULT18X18s	9	0	9
DSP48E	0	9	0

W tabeli 2 przedstawiono wykorzystanie zasobów sprzętowych układu FPGA z układami mnożącymi implementowanymi w logice rozproszonej. Wyniki dla układów Virtex2Pro i Spartan3 ponownie są identyczne.

Tab. 2. Wykorzystanie zasobów sprzętowych – układy mnożące w LUT
Tab. 2. Device resources utilization – multipliers in distributed logic

	xc2vp30	xc5vsx50t	xc3s400
Slices	1304	2918	1304
Flip Flops	288	288	288
LUTs	2238	2695	2238
MULT18X18s	0	0	0
DSP48	0	0	0

Zaskakującym jest fakt, że pomimo iż układy Virtex5SXT mają elementy LUT sześciowejściowe (LUT6), to do implementacji arytmometru niezbędna jest znacznie większa liczba tych elementów niż w przypadku układów Xilinx Virtex2Pro i Spartan3, które przecież wyposażone są w czterowejściowe elementy LUT (LUT4).

Być może jest to spowodowane strategią trasowania połączeń pomiędzy elementami LUT, w celu zapewnienia możliwie najwyższej wydajności.

5. Podsumowanie

Zastosowanie układu FPGA do implementacji arytmometru wspomagającego obliczanie sum korelacyjnych powinno prowadzić do przyspieszenia wykonywania algorytmu obliczeniowego. Zaproponowana przez Autora struktura ma niewątpliwą zaletę, jaką jest wykonywanie obliczeń potokowo. Po zapełnieniu kolejki rejestrów elementami szeregu czasowego, co jeden cykl zapisu, na wyjściu układu pojawia się składowa sumy korelacyjnej (wartość funkcji Heaviside'a). Systemy obliczeniowe wykorzystujące komputery osobiste do obliczania sum korelacyjnych na obliczenie wartości tej funkcji potrzebowały by znacznej liczby cykli, bo wykonują operacje algorytmu obliczeniowego w sposób sekwencyjny. Istotną wadą zaproponowanej metody jest brak mechanizmu zarządzania dostępem do pamięci. Polega ona na tym, że dostęp do pamięci realizuje mikrokontroler, sekwencyjnie pobierając dane, zatem tzw. „wąskim gardłem" będzie mikrokontroler. Opracowanie alternatywnego rozwiązania problemu dostępu do pamięci będzie stanowiło dalszy ciąg badań nad akceleracją obliczeń sum korelacyjnych prowadzonych przez Autora.

6. Literatura

[1] Grassberger P., Procaccia I.: Measuring the strangeness of strange attractors, Physica D: Nonlinear Phenomena, Volume 9, Issue 1-2, Elsevier 1983, 189-208.
[2] Kantz H., Schreiber T.: Nonlinear Time Series Analysis. Cabridge University Press, Cambridge (2003).
[3] Pincus S. M.: Approximate entropy as a measure of system complexity. Proc Natl Accad Sci USA 88 (1991) 2297-2301.
[4] Carvajal R., Wessel N., Vallverdu M., Caminal P., Voss A.: Correlation dimension analysis of heart rate variability in patients with dilated cardiomyopathy, Computer Methods and Programs in Biomedicine, Issue 78 2003, 133-140.
[5] Macklem P. T., Seely A. J. E.: Complex systems and technology of variability analysis, Critical Care 8 (2004) R367-R384.
[6] Takens F.: Detecting strange attractors in turbulence. Dynamical Systems and Turbulence, Lecture Notes in Mathematics 898, Springer-Verlag, Warwick 1980, 366-381.
[7] Xilinx: Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet, Xilinx, DS083 (v.4.6) March 5, 2007.
[8] Xilinx: Virtex-5 Family Overview – LX, LXT and SXT Platforms, Xilinx, DS100 (v.3.0) February 2, 2007.
[9] Xilinx: Spartan-3 FPGA Family: Complete Data Sheet, Xilinx, DS099 (v.2.3) November 30, 2007.

Title: Implementation of correlation sum calculation unit in FPGA devices.

Artykuł recenzowany