Paweł WUJEK, Ryszard PEŁKA
WOJSKOWA AKADEMIA TECHNICZNA, WYDZIAŁ ELEKTRONIKI, INSTYTUT TELEKOMUNIKACJI, ZAKŁAD TECHNIKI CYFROWEJ
ul. gen. Sylwestra Kaliskiego 2, 00-908 Warszawa 49

Układ SoC – FPGA do detekcji twarzy w obrazach cyfrowych

Mgr inż. Paweł WUJEK

Ukończył studia magisterskie na Wydziale Elektroniki i Technik Informacyjnych Politechniki Warszawskiej w 2009 roku. Obecnie jest asystentem w Zakładzie Techniki Cyfrowej w Instytucie Telekomunikacji w WAT. Jego zainteresowania naukowe dotyczą projektowania systemów cyfrowych z układami FPGA i SoC.

e-mail: pwujek@wat.edu.pl

Prof. dr hab. inż. Ryszard PEŁKA

Ukończył studia na Wydziale Elektroniki Politechniki Warszawskiej, obronił pracę doktorską w 1984 roku w Wojskowej Akademii Technicznej. W roku 2004 otrzymał tytuł profesora. Obecnie pełni funkcję kierownika Zakładu Techniki Cyfrowej w Instytucie Telekomunikacji w WAT. Jego zainteresowania naukowe dotyczą metod projektowania, optymalizacji i testowania systemów cyfrowych z układami FPGA i SoC.

e-mail: rpelka@wat.edu.pl

Streszczenie

W artykule przedstawiono wyniki badań dotyczących sprzętowej implementacji algorytmu detekcji twarzy w obrazach cyfrowych z wykorzystaniem układów programowalnych FPGA (Xilinx). Przeprowadzono symulację algorytmu w środowisku PC – Matlab. Przebadany wstępnie algorytm zaimplementowano w układzie FPGA Virtex-4. Wykonano badania eksperymentalne, w których porównano szybkość działania algorytmu w wersji programowej i sprzętowej oraz określono zajętość zasobów układu FPGA.

Słowa kluczowe: detekcja twarzy, FPGA, SoC.

A SoC – FPGA for face detection in digital images

Abstract

In this paper there are presented recent results of the authors' work on implementation of face detection algorithms in digital images based on FPGA technology from *Xilinx*. There was considered a number of existing face detection methods, described in papers [1-3] to find out which one is the best for implementation in a single FPGA device. Then the authors proposed a modified algorithm for face detection that was tested using PC – MATLAB environment. The results of software simulations were used for appropriate adjusting of some essential parameters, according to the requirements of FPGA implementation (the basic limitation is a total number of FPGA resources). The main results of simulations are shown in Tab. 1. The final version of the algorithm was implemented in a Virtex-4 FPGA device and tested using a set of example digital images. An important advantage of the proposed SoC for face detection is its speed (2-4 times higher than that for software implementation, as it is shown in Tab. 2). Furthermore, this speed does not depend on the window size used in image analysis. There was also reported the final utilization of FPGA resources (Tab. 3). The experimental results obtained from laboratory tests of the proposed face detection algorithm implemented in a single FPGA device show that the hardware approach to face detection problem has important advantages: high speed, flexibility and relatively low requirements on the total number of FPGA resources.

Keywords: face detection, FPGA, SoC.

1. Wstęp

Detekcja twarzy w obrazie (FD, ang. *face detection*) jest intensywnie badanym zagadnieniem współczesnej cyfrowej analizy obrazów. Znajduje zastosowanie m.in. w biometrycznych systemach bezpieczeństwa (autoryzacja dostępu, systemy nadzoru), jak również w elektronicznym sprzęcie powszechnego użytku, np. w aparatach cyfrowych do automatyzacji procesu ekspozycji. Przegląd współczesnych prac związanych z tą dziedziną można znaleźć np. w [1]. Większość praktycznych aplikacji FD opiera się na fundamentalnym algorytmie Viola-Jones [2-3]. Jego implementacja w układach SoC, zwłaszcza opartych na technologii FPGA napotyka jednak na problemy związane z ograniczoną liczbą zasobów logicznych. Dlatego opisane dotąd implementacje sprzętowych układów FD dotyczą albo technologii ASIC, albo ograniczonej realizacji w postaci akceleratorów FD [4-5]. Wybrane aspekty FD przedstawiono w [6-7].

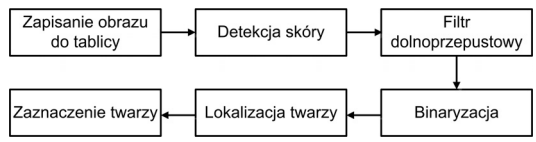
Detekcja twarzy, zwana również lokalizacją, polega na wyodrębnieniu z obrazu obiektów klasyfikowanych jako ludzka twarz i zaznaczeniu zlokalizowanego fragmentu prostokątną ramką oraz zapamiętaniu jej współrzędnych i rozmiaru.

W niniejszej pracy wybrano analizę obrazów kolorowych z 24–bitową głębią kolorów. W celu detekcji koloru skóry stosuje się różne metody reprezentacji barw. Najczęściej stosowane są modele HSV oraz RGB. Model HSV został odrzucony ze względu na konieczne obliczenia na liczbach niecałkowitych, które są trudne do implementacji w strukturach programowalnych. Drugą metodą reprezentacji barw jest model RGB. Najczęściej każdej ze składowych (w reprezentacji 24 bit głębi kolorów) przypisana jest wartość 0-255. W tym przypadku kolor skóry rozpoznaje się poprzez odjęcie od składowej R składowej G.

Wprawdzie sama idea stosowania filtrów konwolucyjnych jest znana i była stosowana m.in. w [8] i w [9], jednak głównym celem niniejszej pracy jest analiza efektywności działania filtru. Porównania dokonano w zależności od rozmiaru okna w implementacji programowej i sprzętowej wykorzystując układ FPGA.

2. Programowa symulacja FD

Jako środowisko symulacyjne użyto programu MATLAB. Wybrany algorytm operuje na obrazie kolorowym o rozdzielczości 640x480 pikseli z 24–bitową głębią kolorów. Obraz ten uzyskiwany jest z kamery internetowej podłączonej do komputera poprzez port USB. Algorytm detekcji twarzy zrealizowany w programie MATLAB przedstawiony został na rys. 1.



Rys. 1. Algorytm detekcji twarzy
Fig. 1. Face detection algorithm

Pierwszym krokiem jest odczytanie obrazu z kamery bądź pliku graficznego i zapisanie go do trójwymiarowej tablicy. Pierwszy wymiar stanowi wysokość obrazu, drugi szerokość, trzecim jest wybór składowej koloru (czerwony – R, zielony – G, niebieski - B). Następnie wyróżnia się obszary kolorów odpowiadających ludzkiej twarzy (testy przeprowadzano dla zdjęć osób o jasnej karnacji skóry). Literatura [6] oraz własne testy wykazały, że najlepsze efekty osiąga się stosując jako kryterium detekcji skóry:

$$Kolor\_skory = abs\left(R - G\right), \tag{1}$$

gdzie: *R*- składowa koloru czerwonego, *G*- składowa koloru zielonego.

Kolejnym krokiem jest zastosowanie filtru dolnoprzepustowego oraz binaryzacji. Obliczanie nowych wartości pikseli realizowane jest zgodnie z równaniem:

$$J_w(x,y) = \frac{1}{w}\sum_{i,j\in K} J(x-i, y-j), \tag{2}$$

gdzie: K – wymiar okna, J – jasność obrazu wejściowego filtru, J_w – jasność obrazu wyjściowego

Jako wagę filtru w powinno się zastosować liczbę 9. Jednak po przeprowadzeniu wielu prób ustalono, że ustawienie zera jako wartość środkowej oraz wybranie wagi filtru równiej 8 nie wnosi dużych zmian w obrazie wynikowym. Kolejnym krokiem w algorytmie jest przeprowadzenie binaryzacji obrazu według zależności:

$$L_{BIN}(x,y) = \begin{matrix} 0 \ dla \ L(x,y) \le p_B \\ 1 \ dla \ L(x,y) > p_B \end{matrix} \tag{3}$$

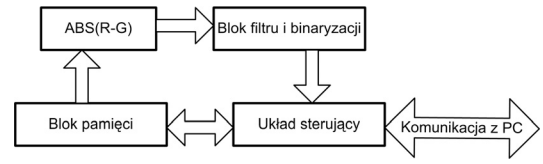
gdzie: $L(x,y)$ – stopnie szarości w obrazie monochromatycznym, $L_{BIN}(x,y)$ – obraz binarny, p_B – próg binaryzacji. Ustalono, że najlepszym progiem dla binaryzacji jest wartość 15.

Kolejnym zadaniem jest zaznaczenie zlokalizowanego obszaru na obrazie. Polega to na wyznaczeniu współrzędnych początku oraz wielkości prostokąta opisanego na zlokalizowanym obszarze. Złożoność i rozwiązanie tego problemu zależy od tego, czy na obrazie spodziewamy się tylko jednego obszaru, który może zostać zidentyfikowany jako twarz, czy wielu takich obszarów. Algorytm poszukuje białego piksela z określonym krokiem (w tym przypadku 40 pikseli). Jeżeli trafi na biały piksel, przeszukuje jego okolice w każdym z czterech kierunków, co 10 pikseli. Jeśli natrafi na kolejne piksele, zmienia wartość współrzędnych prostokąta. W momencie, gdy znajdzie czarny piksel algorytm przeszukiwania w danym kierunku zostaje zakończony. Algorytm wyszukiwania wielu obiektów zaklasyfikowanych jako twarz działa na podobnej zasadzie. Różnica pojawia się w momencie aktualizowania współrzędnych. Jeśli nie trafimy na obszar biały to dane zapisywane są w tablicy. Przejście do kolejnej wartości punktu startowego wiąże się ze sprawdzeniem warunku, czy jest on większy od współrzędnych aktualnie zapisanych w tablicy.

Ostatnim krokiem wyszukiwania twarzy w obrazie jest stwierdzenie, czy znaleziony obszar o kolorze odpowiadającym skórze jest twarzą, czy nie. Kolor skóry ma również ręka, ramię, itp. Te fragmenty powinny być odrzucone. Sprawdzanie to odbywa się na kilka sposobów. Pierwszym z nich jest procentowe określenie zawartości w obrazie binarnym białych pikseli. Nie może ono być większe niż 95% oraz nie może być mniejsze niż 50%. Zostało to ustalone na podstawie eksperymentów. Całe powierzchnie (np. pięść) odpowiadające kolorowi ludzkiej twarzy mają bliskie 100% wypełnienie białych pikseli. Natomiast otwarte dłonie lub ramiona mają wypełnienie niższe niż 50%. Kolejnym warunkiem koniecznym do spełnienia jest sprawdzenie, jaki jest współczynnik wysokości do szerokości obszaru. Ostatnim kryterium jest próba znalezienia oczu w obszarze twarzy. Widoczne są one jako dwa czarne obszary (najczęściej ułożone w jednej linii) w środku białego obszaru (potencjalnej twarzy). Jeśli wybrany obszar przejdzie przez te filtry, zostaje zaklasyfikowany jako obszar twarzy i poddawany jest dalszemu przetwarzaniu.

3. Implementacja FD w układzie FPGA

System zaimplementowany w sprzęcie składa się z bloku sterującego, bloku wyróżnienia koloru skóry (abs(R-G)), bloku filtru dolnoprzepustowego raz bloku pamięci (rys. 2).



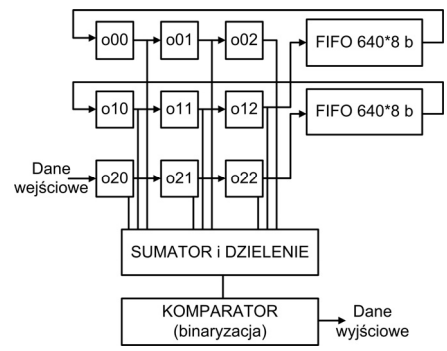
Rys. 2. Schemat blokowy sytemu detekcji twarzy
Fig. 2. Block diagram of the face detection system

Przyjęto, że obraz będzie znajdował się w pamięci, a w dalszych testach w zewnętrznej pamięci SRAM lub DRAM. Rolę układu sterującego pełni sprzętowy procesor PowerPC. Jego dodatkową rolą jest przepisanie danych odczytanych z komputera do pamięci układu FPGA. Blok filtru dolnoprzepustowego zrealizowany jest jako niezależny moduł obliczający funkcję konwolucji o wymiarze 3x3. Funkcji konwolucji o wymiarach 3x3, pozwala obliczyć wartość punktu obrazu na podstawie średniej arytmetycznej punktów z jego otoczenia. Do realizacji takiego zadania niezbędne są dwie linie opóźniające o opóźnieniu równym długości linii obrazu. Zastosowanie linii opóźniających pozwala na jednoczesny dostęp do otoczenia danego punktu obrazu. Wprowadzają one opóźnienie równe dwukrotnej długości linii opóźniających, którego nie można wyeliminować. Tablica 1 pokazuje zależność między wymiarem analizowanego otoczenia, liczbą linii opóźniających oraz liczbą układów arytmetycznych.

Tab. 1. Zależność pomiędzy wymiarem analizowanego otoczenia i liczbą wymaganych zasobów
Tab. 1. Analysis area size vs. required resources

Wielkość otoczenia	Liczba linii opóźniających	Liczba układów mnożących	Liczba układów sumujących
3x3	2	9	3
5x5	4	25	5
7x7	6	49	7

Przykładowa realizacja bloku konwolucji dla punktu z otoczeniem 3x3 przedstawiona została na rys. 3.



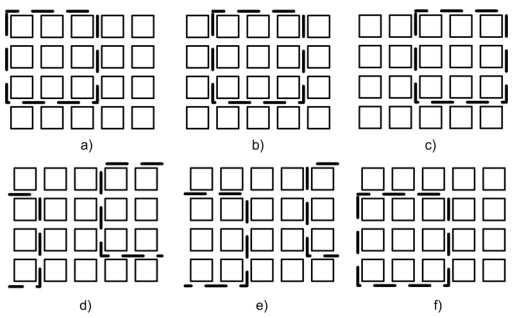
Rys. 3. Schemat bloku konwolucji 3x3
Fig. 3. Block diagram of 3x3 convolution

Dane sumowane są z wagą 1 w sumatorze a następnie poddawane normalizacji, która polega na podzieleniu uzyskanej wartości przez 8. W opisanym przypadku zastosowano bloki opóźniające o długości 640 bajtów wraz z multiplekserami, które umożliwiają regulowania długości bloku opóźniającego spośród następujących wartości: 640, 320.

Dodatkową funkcjonalność można osiągnąć stosując zmienne wartości wag (elementy od o00 do o22). W celu zwiększenia funkcjonalności bloku konwolucji dodany został do niego moduł odpowiedzialny za usuwanie z wyjścia niepotrzebnych wyników. Po wykonaniu operacji konwolucji otrzymujemy obraz, którego rozmiar został zmniejszony o jeden piksel z każdej strony (w przypadku okna 3x3).

Na rys. 4 poprawnie obliczana jest funkcja konwolucji dla kroku a), b), c), f). Wartości obliczone dla kroków d) i e) są błędne i należy je odrzucić.

Blok odpowiedzialny za odrzucanie niepoprawnych wyników składa się z licznika liczącego do 640 (długość linii obrazu), który produkuje sygnał *enable* w momencie, gdy obliczane dane są poprawne. Drugi licznik (długość linii obrazu pomnożona przez 2) wykorzystywany jest do sygnalizowania (*enable_wy*) pojawienia się pierwszego poprawnego wyniku na wyjściu bloku. Sygnały *enable* i *enable_wy* pochodzące od obydwu liczników podłączone są razem z wejściowym sygnałem zegarowym do bramki AND. Na jej wyjściu uzyskiwany jest sygnał zegarowy. na którego każdym zboczu narastającym wystawiana jest kolejna poprawnie obliczona wartość funkcji konwolucji.



Rys. 4. Kolejne kroki przeglądania tablicy obrazu za pomocą okna 3x3
Fig. 4. Successive steps by 3x3 windowing

4. Wyniki badań eksperymentalnych

Porównano czas obliczenia funkcji konwolucji oraz binaryzacji obrazów za pomocą algorytmu zaimplementowanego programowo i sprzętowo. Jako platforma programowa posłużył komputer PC z procesorem Intel Core 2 QUAD 2,5 GHz z 3 GB RAM, WIN XP SP3 oraz środowiskiem MATLAB 7.8.0 (R2009a). W badaniach sprzętowych użyto płytę uruchomieniową Virtex-4 FX12 Evaluation Board z układem FPGA Virtex – 4 XC4VFX12 z blokami pamięci o czasie dostępu 10ns oraz zegarem 100MHz. Porównanie czasu niezbędnego do obliczenia funkcji konwolucji oraz binaryzacji obrazu w zależności od wielkości okna dla obrazu o rozdzielczości 640x480 zestawiono w tablicy 2.

Tab. 2. Porównania czasów obliczeń dla różnych wielkości okna
Tab. 2. Comparison of calculation time for different window size

Wielkość okna	Program	Sprzęt
3x3	54 ms	31 ms
5x5	112 ms	31 ms

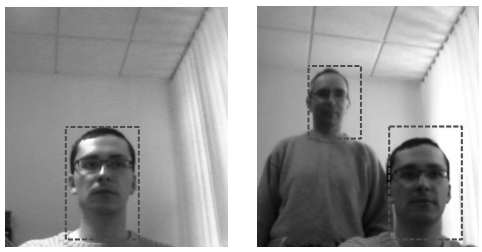
Z powyższej tablicy widać, że programowy algorytm przy zwiększeniu okna wydłuża czas obliczeń dwukrotnie. Natomiast przy sprzętowej implementacji czas trwania operacji przy zwiększeniu okna pozostaje bez zmian. Różnica pojawia się w momencie porównania zasobów. Przy implementacji programowej wymagana jest tylko większa liczba odwołań do pamięci oraz dopisanie kilku działań. W implementacji sprzętowej wymagane jest wykorzystanie dwukrotnie większej liczby linii opóźniających. W testowanym układzie blok filtru dolnoprzepustowego wraz z binaryzacją dla okna 3x3 zajmuje ok. 26% zasobów. Znaczącą część stanowią linie opóźniające. Zwiększenie liczby linii opóźniających z 2 do 4 powoduje prawie dwukrotne zwiększenie wykorzystanych zasobów. Przedstawione zostało to w tablicy 3.

Tab. 3. Zasoby wykorzystywane w układzie FPGA
Tab. 3. Utilization of FPGA resources

	Dostępne zasoby	3x3		5x5	
		Użyte	Użyte	Użyte	Użyte
Liczba komórek	5472	1429	26%	2791	51%
Liczba przerzutników	10944	2028	18%	3988	36%
Liczba 4 wejściowych LUT	10944	1242	11%	2406	21%

Problem tak dużego zużycia zasobów można rozwiązać na dwa sposoby. Pierwszym jest wykorzystanie gotowych linii opóźniających zrealizowanych w postaci gotowych bloków kolejek FIFO. Drugim sposobem jest wykorzystanie drugiej struktury FPGA, której zdaniem będzie tylko realizacja funkcji kowolucji oraz binaryzacja.

Na rys. 5 przedstawiony został efekt działania algorytmu FD dla dwóch przykładowych obrazów (z jedną i dwoma twarzami). Przesunięcie ramki na prawym obrazie wynika z wybranego kroku przeglądania tablicy obrazu.



Rys. 5. Przykładowe obrazy po zastosowaniu algorytmu FD
Fig. 5. Example pictures after FD

5. Wnioski

Zaproponowany algorytm lokalizacji twarzy działa szybciej w wersji sprzętowej niż w symulacji PC. Ponadto dodanie dowolnej operacji w algorytmie na komputerze powoduje dalsze wydłużenie czasu obliczeń. Widoczne jest to w przypadku zwiększenia rozmiaru okna funkcji konwolucji. Czas przetwarzania jednego obrazu na komputerze wydłuża się prawie dwukrotnie. Podobna zmiana w strukturze programowalnej nie wpływa na czas przetwarzania, ponieważ wiele operacji może być wykonywanych równolegle. Zaletą implementacji algorytmu na komputerze PC jest natomiast łatwość wprowadzania zmian oraz możliwość wykorzystania gotowych funkcji. W układzie programowalnym dowolna zmiana niesie ze sobą konieczność wielu symulacji oraz syntezy kodu całej struktury. Trudność pojawia się również przy skomplikowanych algorytmach bazujących na wielu pętlach oraz wielu instrukcjach warunkowych. Rozwiązaniem może być wykorzystanie sprzętowego procesora PowerPC.

6. Literatura

[1] Zhang C., Zhang Z.: A Survey of Recent Advances in Face Detection, Technical Report, MSR-TR-2010-66, Microsoft Corp. June 2010.
[2] Viola P. and Jones M.: Rapid object detection using a boosted cascade of simple features. In Proc. of CVPR, 2001.
[3] Viola P., Jones M.: Robust Real-Time Face Detection, Int. Journal of Computer Vision, Vol. 57, No 2, 2004, pp. 137-154.
[4] Cho J., et.al.: FPGA-based Face Detection System Using Haar Classifiers, Proc. ACM/SIGDA 2009, pp. 103-112.
[5] Hefenbrock D., et.al.: Accelerating Viola-Jones Face Detection to FPGA=Level using GPUs, IEEE 2010 18th FCCM Int. Symposium, pp. 11-18.
[6] Kuchariew G., Ponikowski T., Chen L.: A Few Approaches To Face Detection In Face Recognition Systems - Proceedings of the Eighth International Conference: Advanced Computer Systems (ACS'2001), Mielno, October 17-19, 2001.
[7] Nikolaidis A., Pitas I.: Robust Watermarking of Facial Images Based on Salient Geometric Pattern Matching - IEEE Transactions on Multimedia, vol. 2, no. 3, September 2000.
[8] Barros A.M., Akil M. 1992: Study and implementation of a real time 3×3 programmable convolver with reconfigurable technology, Euro ASIC '92 Proceedings, pp.392-395, (0-8186-2845-6/92), IEEE.
[9] Yadav D.K., Gupta A.K., Mishra A.K. 2008: A fast and area efficient 2-D convolver for real time image processing TENCON 2008 - 2008 IEEE Region 10 Conference.

otrzymano / received: 13.05.2011
przyjęto do druku / accepted: 04.07.2011 *artykuł recenzowany*