POZNAN UNIVERSITY OF TECHNOLOGY ACADEMIC JOURNALS
No 106 Electrical Engineering 2021

DOI 10.21008/j.1897-0737.2021.106.0004

Paweł KOWALSKI*, Robert SMYK*

SPRZĘTOWA IMPLEMENTACJA TRANSFORMACJI HOUGHA W CZASIE RZECZYWISTYM

W artykule przedstawiono implementację sprzętową w FPGA algorytmu do wykrywania kształtów aproksymowanych zbiorem linii prostych podczas przetwarzania obrazu cyfrowego w czasie rzeczywistym. W opracowanej strukturze sprzętowej podniesiono efektywność przetwarzania poprzez zastosowanie przetwarzania przepływowego, lookup table, wykorzystanie wyłącznie arytmetyki liczb całkowitych oraz rozproszenie pamięci głosowania. Eksperymentalnie wykorzystano przedstawioną strukturę w torze przetwarzania obrazu w czasie rzeczywistym złożonym z kamery OV7670, płyty deweloperskiej Terasic DE10-nano oraz monitora podłączonego za pomocą HDMI. Pełny tor przetwarzania został zaimplementowany w pojedynczym układzie FPGA Intel Cyclone V. Maksymalna prędkość przetwarzania obrazu z wykorzystaniem opracowanej implementacji została określona na 275 MHz.

SŁOWA KLUCZOWE: przetwarzanie obrazu, FPGA, transformacja Hougha, wykrywanie prostych.

1. WPROWADZENIE

Dedykowany tor przetwarzania obrazu jest obecnie typowym składnikiem urządzeń rejestrujących obraz wyposażonych w kamerę. Choć takie systemy można konstruować na wiele różnych sposobów, istnieje pewien typowy schemat w przetwarzaniu obrazu, który składa się z kilku etapów. W przedstawionym toku rozumowania przyjęto, że wejściem jest moduł kamery, gdzie początkowo obraz jest przetwarzany na poziomie pikseli, co jest naturalnym efektem wynikającym ze sposobu odczytu danych pochodzących z modułu kamery. W dalszych etapach po złożeniu z obrazu pełnej klatki dokonuje się przykładowo ekstrakcji obiektów czy analizy sceny. Finalnie obraz może być zapisywany wraz ze zbiorem jego cech lub też strumieniowany, np. do wyświetlenia. W związku z tym projekt dedykowanego toru oraz zaimplementowane w nim algorytmy realizuje się biorąc pod uwagę wejście, sposób przetwarzania i wyjście. W niniejszej pracy skupiono się na implementacji toru przetwarzania dedykowanego do wykrywania prostych, który ma zastosowanie w przetwarzaniu

* Państwowa Wyższa Szkoła Zawodowa w Elblągu.

obrazów w pojazdach i statkach powietrznych, widzeniu komputerowym, przetwarzaniu obrazów medycznych i sztucznej inteligencji [1]. Jednym ze znanych algorytmów wykrywania prostych jest zastosowanie transformacji Hougha. Metodę tę pierwotnie opisano w 1962 roku [2]. Charakteryzuje się ona wysoką odpornością na zakłócenia oraz brak ciągłości linii [3]. Jedną z wad metody jest jej wysoki koszt obliczeniowy wynikający głównie z konieczności wielokrotnego dostępu do pamięci. Możliwość użycia jej do przetwarzania obrazów w czasie rzeczywistym w formie programowych implementacji z wykorzystaniem procesorów (CPU) jest tu bardzo ograniczona. Okazuje się, że możliwe są efektywne implementacje w postaci sprzętowej z wykorzystaniem FPGA (ang. Field Programmable Gate Array) z zastosowaniem przetwarzania potokowego i równoległego. Dodatkowo można poprawić efektywności poprzez wprowadzenie modyfikacji takich jak zastosowanie lookup table, algorytmu CORDIC oraz projektowanie dedykowanej pamięci o jednoczesnym dostępie do wielu komórek [1]. W niniejszej pracy została przedstawiona efektywna sprzętowa implementacja w FPGA transformacji Hougha umożliwiająca przetwarzanie obrazu w czasie rzeczywistym.

2. PRZEGLĄD METOD OBLICZANIA TRANSFORMACJI HOUGHA

Pojedyncza prosta w przestrzeni dwuwymiarowej definiowana jest przez wektor ψ ortogonalny do prostej i łączący ją z początkiem układu odniesienia. Wektor definiowany jest przez dwa współczynniki ρ oraz α, gdzie ρ to długość wektora, a α to kąt zawarty pomiędzy osią x, a wektorem ψ. Współczynnik ρ wyznaczany jest z wykorzystaniem formuły (1).

$$\rho = \sin\alpha \, y + \cos\alpha \, x \,, \tag{1}$$

gdzie x i y to współrzędne piksela odpowiednio w osi x i y. Współczynniki α i ρ wykorzystywane są do tworzenia przestrzeni Hougha złożonej z $n \times d$ komórek, gdzie n definiuje liczność zestawu współczynników α, a d maksymalną długość ρ. Transformacji poddawany jest obraz binarny. Dla każdego ustawionego piksela tego obrazu inkrementowane są komórki (α_i, ρ_i) tablicy reprezentującej przestrzeń Hougha. Zestaw kątów $\alpha_1 ... \alpha_n$ wybrany jest arbitralnie. Natomiast współczynniki $\rho_1 ... \rho_n$ wyznaczane są z wykorzystaniem formuły (1). Procedura ta jest nazywana głosowaniem, ponieważ w wyniku serii inkrementacji otrzymywana jest tablica, w której każdy element reprezentuje pojedynczą prostą, a jego wartość odpowiada liczbie pikseli leżących na tej prostej.

W literaturze można znaleźć modyfikacje transformacji Hougha [1] zwiększające klasę wykrywanych obiektów takie jak Circular Hough Transform oraz Generalized Hough Transform [1, 4]. Prezentowane są również modyfikacje oraz techniki implementacji poprawiające efektywność metody poprzez zmniej-

szenie kosztu obliczeniowego oraz poprawę szybkości działania. Przegląd metod implementacji transformacji Hougha przedstawiono w [1, 4].

Na koszt obliczeniowy w znacznym stopniu wpływa procedura wyznaczania głosów z wykorzystaniem funkcji trygonometrycznych. Jednym ze sposobów zmniejszenia tego kosztu jest tablicowanie wartości funkcji trygonometrycznych wykorzystywanych w obliczeniach. Podczas przetwarzania obrazu nie będą one już obliczane, a jedynie odczytywane z pamięci w trybie lookup. Wadą tego podejścia jest zajmowanie przez lookup table dodatkowych zasobów FPGA.

Innym sposobem na podwyższenie efektywności jest wykorzystanie algorytmu CORDIC (ang. COordinate Rotation DIgital Computer) do szybkiego wyznaczania wartości funkcji trygonometrycznych. Metoda ta została wykorzystana w implementacji transformacji Hougha opisanej w [5]. Podstawową zaletą CORDICa jest możliwość uniknięcia wykonywania mnożenia poprzez zastąpienie go przesunięciami binarnymi. Przegląd architektur do jego realizacji przedstawiono w [6]. W typowym podejściu jeden stopień sprzętowy przy implementacji algorytmu CORDIC wymaga cztery do ośmiu sumatorów i tej samej liczby rejestrów przesuwnych. Warto zaznaczyć, że w formie bezpośredniej CORDIC wymaga minimalnie ośmiu stopni obliczeniowych. W końcowym etapie obliczeń konieczne jest dzielenie przez stałą związaną z zwiększeniem długości wektora, wynikającą ze stosowania pseudorotacji wektora zamiast rotacji [7]. Istotnym zagadnieniem jest też rodzaj stosowanej arytmetyki, typowo jest to typ całkowity. Dla układów FPGA dostępne są tzw. rdzenie CORDIC (ang. CORDIC Cores), umożliwiające implementację algorytmu wraz z wyborem realizowanej funkcji, rodzaju arytmetyki, zakresu liczbowego i typu architektury.

Transformacja Hougha wymaga wyznaczenia współczynnika ρ (1) dla wielu kątów α. Należy zauważyć, że już na etapie ekstrakcji krawędzi, na podstawie gradientu, można w przybliżeniu określić je nachylenie. Pozwala to na ograniczenie liczby sprawdzanych kątów. Metoda ta została nazwana gradientową, ponieważ na podstawie gradientu określany jest zestaw kątów. Została ona zastosowana w [8].

Innym sposobem zmniejszenia kosztu obliczeniowego jest przetworzenie jedynie części pikseli. Jest to realizowane poprzez losowe wybranie pikseli do transformacji. Przykładem takich metod są Probabilistic Hougha Transform (PHT), Randomized Hough Transform (RHT), Monte Carlo Hough Transform (MCHT). Charakteryzują się one mniejszym kosztem obliczeniowym od metod przetwarzających wszystkie piksele, ale są również mniej odporne na zakłócenia i szumy występujące w obrazie [1].

Istotnym problemem podczas implementacji transformacji Hougha jest realizacja przestrzeni Hougha używanej podczas głosowania. Z uwagi na wymaganą dużą ilość miejsca do przechowywania wyników głosowania można znaleźć implementacje wykorzystujące pamięć zewnętrzną [8, 9]. Podejście to charakteryzuje się oszczędnością zasobów układu FPGA. Pozwala to na zastosowanie

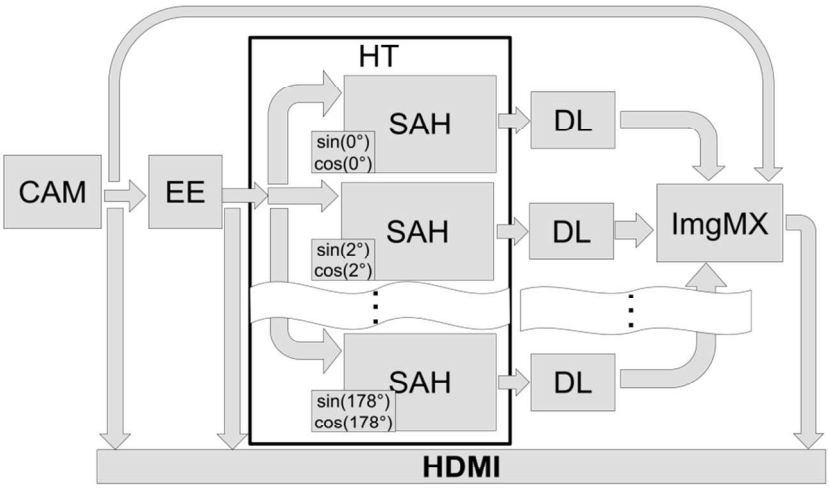
tańszych układów FPGA zawierających mniejszą liczbę komórek logicznych. Należy jednak wziąć pod uwagę przepustowość zewnętrznych pamięci, która ogranicza maksymalną rozdzielczości i prędkość przetwarzania obrazu. Ze względu na rozwój układów cyfrowych podejście to jest coraz rzadziej stosowane. Obecne układy zawierają znacznie więcej elementów logicznych. Pozwala to na tworzenie implantacji zajmujących więcej zasobów, ale charakteryzujących się szybszym działaniem. Przykładem jest utworzenie wewnątrz układu wielu niezależnych bloków pamięci, do których dostęp wykonywany jest równolegle. Podejście to zostało między innymi zastosowane w [8, 10]. W [8] procedura głosowania została podzielona na moduły. Każdy z nich został wyposażony w podwójnie buforowaną pamięć. Obliczenia wykonywane są na liczbach stałoprzecinkowych, a kąt α jest próbkowany co 11.25°. Tak opracowana architektura [8], zaimplementowana w FPGA Virtex-4 pozwala na przetwarzanie obrazu o rozdzielczości 800×600 z prędkością 30 fps. W [10] zaprezentowano implementację uruchomioną na płycie DE4-320-C2 zawierającej układ FPGA Altera Stratix-IV EP4SGX230KF40C2. Opracowane rozwiązanie zostało zastosowane do wykrywania pasów drogowych. Umożliwia ono przetworzenie klatki obrazu XGA w czasie 5.4 ms. Implementacja złożona jest z zestawu modułów obliczeniowych. Każdy z nich zawiera jednowymiarową tablicę będącą fragmentem przestrzeni Hougha zapewniając równoległy dostęp do wielu komórek pamięci podczas głosowania. Każdy z równoległych modułów zawiera dodatkowo dwa mnożniki i dwie tablice lookup zaimplementowane jako RAM. Krawędzie wykrywane są przy użyciu operatora Sobela, a początek układu współrzędnych jest w geometrycznym środku obrazu. Aby zapobiec wystąpieniu ujemnego ρ wykorzystano kąty α z zakresu od 0° do 360°. W [11] również zaprezentowano system do wykrywania pasów drogowych z wykorzystaniem krawędziowania operatorem Sobela oraz transformacji Hougha. Implementacja sprzętowa została opisana w języku C i zsyntezowana z wykorzystaniem Xilinx Vivado High Level Synthesis (HLS). Dla obrazu o rozdzielczości 720×1280 uzyskano prędkość 59.13 fps.

W [12] zaprezentowano metodę, w której standardowa procedura wyznaczania głosów została zmodyfikowana. W zmodyfikowanej wersji parametry ρ i α definiujące prostą są wyznaczane poprzez przesunięcie prostej do niej równoległej. Opracowana implementacja pozwala na przetwarzanie obrazów z częstotliwością do 226,76 MHz.

3. OPRACOWANA IMPLEMENTACJA TORU PRZETWARZANIA

W pracy przedstawiono własną sprzętową implementację transformacji Hougha zrealizowaną w postaci modułu *HoughTransform (HT)* (rys. 1). W celach eksperymentalnych został on zaimplementowany w systemie przetwarzania obra-

zu z kamery w czasie rzeczywistym. System składa się z kamery OV7670 o rozdzielczości 640×480, płyty deweloperskiej Terasic DE10-nano wyposażonej w układ FPGA Intel Cyclone V 5CSEBA6U23I7 oraz monitora podłączonego za pomocą HDMI. Pełen schemat toru przetwarzania został przedstawiono na rys. 1.



Rys. 1. Schemat toru przetwarzania z wykorzystaniem modułu *HoughTransform (HT)*

Opracowany system przetwarzania obrazu składa się z bloków pobierania obrazu z kamery *CAM*, ekstrakcji krawędzi *EE* z wykorzystaniem operatora Previtt [13], transformacji Hougha *HT*, generowania linii na podstawie wyników transformacji *DrawLine* (*DL)*, nakładania obrazu binarnego na obraz kolorowy *ImgMX* oraz wyświetlania obrazów na ekranie *HDMI*. Wszystkie moduły oprócz *HDMI* są taktowane sygnałem zegarowym generowanym w *CAM*, gdzie zbocze narastające sygnalizuje wczytanie nowego piksela.

Głównym elementem układu jest blok transformacji Hougha *HT*. Przyjmuje on jako wejście obraz binarny po krawędziowaniu, przesyłany z wykorzystaniem sygnałów *posX*, *posY* i *edgeBin*. Piksele oznaczone współrzędnymi $posX \in \langle 1;640 \rangle$ i $posY \in \langle 1;480 \rangle$ są pikselami zarejestrowanego obrazu. Strumień wideo z kamery zawiera również sygnały synchronizacji pionowej (piksele oznaczone $posY = 0$) i poziomej (piksele oznaczone $posX = 0$). Blok *HT* złożony jest z zestawu bloków *SingleAngleHough* (*SAH*). Do każdego z nich przekazywane są wartości wartość *sin* wyznaczone według formuły $\lfloor sin\alpha \cdot 128 \rfloor$ oraz *cos* wyznaczone według formuły $\lfloor cos\alpha \cdot 128 \rfloor$. Sygnały te są stałe, wyznaczone dla każdego bloku *SAH* przed syntezą układu. Jednocześnie do każdego bloku *SAH* za pośrednictwem sygnałów *posX, posY* i *edgeBin* przekazywany jest binarny obraz krawędzi pozyskany z *EE*.

Pojedynczy moduł *SAH* realizuje transformację Hougha dla kąta α oraz $\alpha+180°$. Procedura ta polega na wypełnieniu jednego wiersza tablicy reprezentu-

jącej przestrzeń Hougha. Dla każdego piksela krawędziowego (*edgeBin* = 1) obrazu wejściowego wykonywana jest inkrementacja odpowiedniej komórki pamięci, gdzie adres komórki wyznacza część całkowita ρ (1). Kod źródłowy modułu *SAH* w języku Verilog został przedstawiony na listingu 1.

Listing 1. Kod źródłowy modułu *SingleAngleHough* (*SAH*)

```
module SingleAngleHough( input clk,
    input [9:0] posX, input [8:0] posY, input edgeBin,
    output reg [37:0] votes, output reg cls,
    input signed [8:0] sin, input signed [8:0] cos );

  wire clearMem;
  wire enableAdd;
  wire [9:0] rdAddress;
  wire [9:0] q;
  wire signed [9:0] centerX;
  wire signed [9:0] centerY;
  reg signed [16:0] centerP;
  reg [9:0] data;
  reg [29:0] shiftReadAddress;
  reg [10:0] clearAddr;
  reg [9:0] wraddress;
  reg [3:0] shiftEdge;
  reg wren;

  assign enableAdd = (posX>1 && posX<640 && !clearMem)? 1'b1 : 1'b0;
  assign clearMem = ((posY>479) || (posY<2))? 1'b1 : 1'b0;
  assign rdAddress = clearMem? clearAddr[9:0] : centerP[16:7];
  assign centerX = posX-320;
  assign centerY = posY-240;

  always @(posedge clk) begin
     centerP <= sin*centerY + cos*centerX;
     shiftReadAddress <= {shiftReadAddress[19:0], rdAddress};
     shiftEdge <= {shiftEdge[2:0], edgeBin & enableAdd};

     if(clearMem && shiftEdge==0) begin
        data<=0;
        if (clearAddr<1024 ) begin
           clearAddr <= clearAddr+1;
           wraddress <= rdAddress;
           cls <= 1'b1;
           votes<={sin, cos, shiftReadAddress[9:0], q};
        end
        else cls <= 1'b0;
     end
     else begin
        data<=q+shiftEdge[2];
        wren <= shiftEdge[2];
        wraddress<=shiftReadAddress[19:10];
        clearAddr<=11'd0;
     end
  end
  RAM10io1024 RAM10io1024_HoughWorkSpace(
     .clock(clk) ,  // input
     .data(data) ,  // input [9:0]
     .rdaddress(rdAddress) , // input [9:0]
     .wraddress(wraddress) , // input [9:0]
     .wren(wren | cls) ,  // input
     .q(q)    /* output [9:0] */ );

  endmodule
```

W module *SAH* można wyróżnić trzy podstawowe sekcje: sekcję asynchroniczną (linijki 20-24), sekcję synchroniczną (linijki 26-27) oraz pamięć RAM (linijki 49-55). W sekcji asynchronicznej wyznaczane są flagi *clearMem, enableAdd* oraz sygnały *rdAddress*, *centerX* i *centerY*. Flaga *clearMem* określa tryb pracy: głosowanie lub przesyłanie wyników z jednoczesnym czyszczeniem pamięci. Flaga *enableAdd* odpowiada za odblokowanie możliwości głosowania. Zastosowana ekstrakcja krawędzi powoduje zmniejszenie obrazu o 1 piksel z każdej strony, więc w głosowaniu uwzględnione są jedynie piksele z zakresu $posX \in \langle 2;639 \rangle$ i $posY \in \langle 2;479 \rangle$. Sygnał *rdAddress* zawiera adres do odczytu z pamięci. Źródło tego sygnału jest zmieniane w zależności od trybu pracy określonego przez flagę *clearMem*. Sygnały *centerX* i *centerY* reprezentują pozycję piksela znormalizowaną względem geometrycznego środka obrazu ($PosX = 320$, $PosY = 240$). Normalizacja zwiększa efektywność wykorzystania pamięci. Przed normalizacją ρ przyjmuje wartości z zakresu (0;800), a α z zakresu (-90°; 180°), a po normalizacji ρ z zakresu (0;400), a α z zakresu (0°;360°). W pierwszym przypadku wielkość tablicy głosowania wynosi 800×270. Natomiast w drugim przypadku wymagana jest mniejsza tablica o rozmiarze 400×360.

Sekcja synchroniczna wyzwalana jest zboczem narastającym sygnału *clk* sygnalizującym nowy piksel. W każdym cyklu wyznaczana jest wartość *centerP, shiftReadAddress* oraz *shiftEdge.* Rejestr *centerP* reprezentuje współczynnik ρ (1). Ze względu na zwiększone wartości *sin* i *cos*, jego wartość jest również 128 razy większa od rzeczywistego wyniku. Więc 10 najstarszych bitów można interpretować jako część całkowitą liczby, a 8 najmłodszych bitów jako część rzeczywistą. W takiej reprezentacji, część całkowita (10 bitów) wyznacza adres komórki pamięci do inkrementacji. Rejestry przesuwne *shiftReadAddress* oraz *shiftReadAddress* przechowują odpowiednio adresy *rdAddress* oraz krawędzie *edgeBin* z ostatnich czterech cykli zegarowych.

W sekcji synchronicznej są również umieszczone bloki głosowania oraz przesyłania wyników z czyszczeniem pamięci. Blok przesyłania wyników z czyszczeniem pamięci jest wykonywany po zakończeniu głosowania sygnalizowanego pustym rejestrem *shiftEdge* oraz ustawioną flagą *clearMem*. W bloku tym kolejne odczytywane z pamięci głosy zapisywane są w rejestrze *votes* wyprowadzonym jako port wyjściowy. Jednocześnie odczytane komórki są kasowane poprzez przypisanie 0. Przygotowuje to pamięć do kolejnego głosowania. Podczas procedury przesyłania głosów i kasowania pamięci ustawiana jest również flaga *cls*.

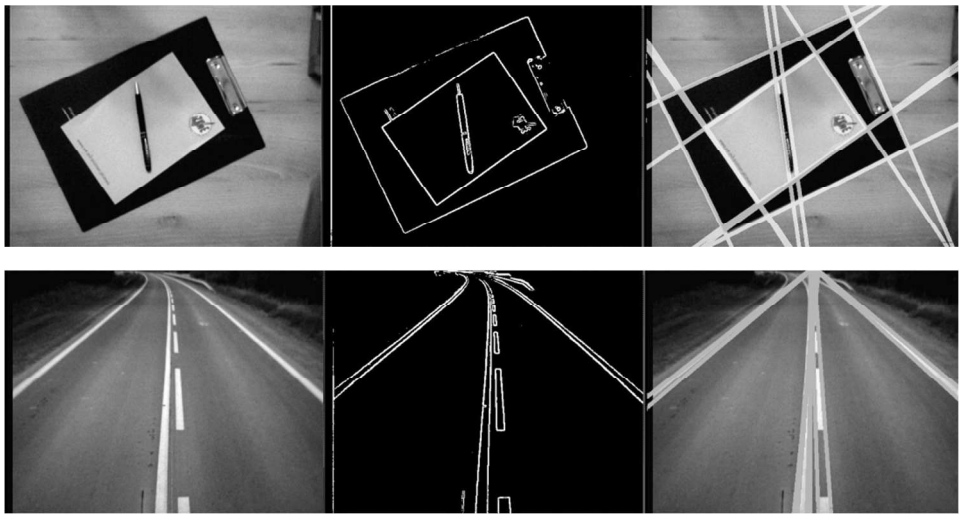
W *SAH* zawarta jest lokalna dwuportowa pamięć RAM *RAM10io1024* o pojemności 10240 bitów zorganizowana w postaci 1024 słów 10 bitowych. Służy ona do realizacji głosowania na proste definiowane przez współczynniki ρ i α. Współczynnik α jest określany w bloku *HT*, więc w obrębie jednego bloku *SAH* jest on stały. Natomiast współczynnik ρ jest wyznaczany z wykorzystaniem (1). Dla przetwarzanego obrazu o rozdzielczości 640×480 przyjmuje on wartości

od –400 do 400. Więc zastosowanie pamięci o rozmiarze 1024 powoduje, że 224 komórki nie są wykorzystywane. Należy zaznaczyć, że w zastosowanym układzie FPGA pamięci budowane są z bloków M10K o pojemności 10 Kb, więc jeden blok zostanie zużyty zarówno w przypadku utworzenia pamięci o pojemności 800, jak i 1024 komórek. Wybrano pojemności 1024 komórek, ponieważ pozwala ona na wyznaczenie adresu bez operacji sumy czy różnicy, a jedynie poprzez wybranie 10 bitów rejestru *centerP*.

Implementacja bloku *SAH* przedstawionego w listingu 1 zajmuje 36 konfigurowalnych bloków logicznych ALM (ang. Adaptive Logic Module), 87 rejestrów oraz pojedynczy blok pamięci M10K. Maksymalna prędkość bloku *SAH* została określona przez Quartus Timing Analysis na 275 Mhz. Umożliwia to przetwarzanie obrazu o rozdzielczości 640×480 z częstotliwością do 895 fps.

4. BADANIA EKSPERYMENTALNE

Opracowane rozwiązanie w postaci toru przetwarzania z rys. 1 zostało zastosowane do przetwarzania obrazu w czasie rzeczywistym. W torze zastosowano blok *HT* złożony z 90 modułów *SAH* pozwalający na wykrywanie prostych z próbkowaniem 2. Przykład wyników przetwarzania obrazu z wykorzystaniem opracowanego systemu pokazano na rys. 2.



Rys. 2. Przykład przetwarzania obrazu z wykorzystaniem przedstawionej metody

Rysunek 2 przedstawia dwa przykłady przetwarzania obrazu z wykorzystaniem przedstawionej metody. Każdy z nich składa się z trzech obrazów: obraz surowy, obraz z wykrytymi krawędziami po progowaniu oraz obraz surowy z nałożonymi wykrytymi prostymi. Wartości progów zostały dobrane ręcznie.

Przeprowadzono analizę opracowanej struktury. Przebadano szybkość oraz ilość zajmowanych zasobów przez moduł *HT* w wersji z próbkowaniem $\Delta\alpha$ równym 1°, 2°, 3°, 5°, 9° oraz 15°. Zmiana próbkowania kąta α możliwa jest poprzez zmianę liczby wykorzystanych bloków *SAH*.

Opracowana implementacja została porównana z implementacjami opisanymi w [1, 2, 3, 7, 8]. Wyniki porównania zostały przedstawione w tabeli 1. W porównaniu uwzględniono rozdzielczość obrazu użytego do testów metody, maksymalną prędkość przetwarzania tego obrazu określoną w klatkach na sekundę *fps*, maksymalną prędkość przetwarzania kolejnych pikseli f_{max}, liczbę użytych programowalnych elementów logicznych CLB (ang. configurable logic blocks), liczbę bloków DSP, liczbę rejestrów *reg*, ilość użytej pamięci RAM, oraz próbkowanie kąta wykrywanych prostych $\Delta\alpha$. Należy zaznaczyć, że przedstawione implementacje zostały zrealizowane z wykorzystaniem innych układów i syntezowane w różnych środowiskach FPGA, co może mieć wpływ na uzyskiwane prędkości przetwarzania oraz ilość zużytych zasobów sprzętowych.

Tabela 1. Porównanie sprzętowych implementacji transformacji Hougha

Metoda	Rozdzielczość	fps	f_{max} [MHz]	CLB	DSP	reg	RAM [b]	$\Delta\alpha$
HT1	640×480	833	258,00	6301,000	0	18900	1843200	1,00°
HT2	640×480	878	270,00	3151,000	0	9450	921600	2,00°
HT3	640×480	878	270,00	2101,000	0	6300	614400	3,00°
HT5	640×480	885	272,00	1261,000	0	3780	368640	5,00°
HT9	640×480	895	275,00	701,000	0	2100	204800	9,00°
HT15	640×480	895	275,00	421,000	0	1260	122880	15,00°
[8]	600×800	30	14,40	N/A	N/A	N/A	N/A	11,25°
[9]	256×256	413	27,10	324,000	N/A	N/A	N/A	1,00°
[10]	1024×768	30	147,00	850,000	N/A	645	1513712	2,00°
[11]	1280×720	60	54,37	77,658	167	66186	8476	9,00°
[12]	640×480	736	226,76	5717,000	N/A	6010	936	1,00°

W pracy przetestowane zostały moduły *HT* w wersjach z próbkowaniem $\Delta\alpha$ równym 1°, 2°, 3°, 5°, 9° oraz 15°. W tabeli 1 zostały one oznaczona jako HT$\Delta\alpha$. Zwiększenie dokładności wykrywania kąta nachylenia prostej wymaga większej ilości zasobów, ale należy zauważyć, że maksymalna częstotliwość przetwarzania wyznaczona przez Quartus Time Analysis zmniejsza się nieznacznie i dla przebadanych wariantów mieści się w zakresie od 258 do 275 MHz.

5. PODSUMOWANIE

W artykule przedstawiono sprzętową implementację Transformacji Hougha do wykrywania prostych w obrazie w czasie rzeczywistym. Zaprojektowany system składa się z kamery OV7670, płyty deweloperskiej Terasic DE10-nano

wyposażonej w układ FPGA Intel Cyclone V 5CSEBA6U23I7 oraz monitora podłączonego za pomocą HDMI. Podczas projektowania zastosowano techniki zwiększające efektywność wykorzystania struktury sprzętowej FPGA. Uzyskano w pełni przepływowe przetwarzanie gdzie obraz próbkowany jest na bieżąco piksel po pikselu. W zaproponowanym rozwiązaniu nie występuje konieczność buforowania, tj. zapisywania całej klatki obrazu w celu przetwarzania. Zastosowano tzw. lookup table do przechowywania wykorzystywanych wartości funkcji trygonometrycznych. Obliczenia wykonywane są wyłącznie z wykorzystaniem arytmetyki liczb całkowitych. Zakres dynamiczny operandów został dostosowany do właściwości przetwarzanego obrazu, operacje takie jak sumowanie i mnożenie są wykonywane na liczbach nieprzekraczających 11 bitów. Przestrzeń głosowania została rozproszona, a procedura głosowania na pełen zestaw prostych przechodzących przez pojedynczy piksel realizowana jest równolegle bez kolizji adresowej. Pamięć została zorganizowana z uwzględnieniem wielkości używanych bloków. Maksymalna częstotliwość przetwarzania obrazu wyznaczona z wykorzystaniem narzędzia Time Analysis w środowisku Quartus II wynosi 275 Mhz. Pozwala to na przetwarzanie w czasie rzeczywistym obrazów o rozdzielczości 640×480 z prędkością do 895 fps.

LITERATURA

[1] Mukhopadhyay P., Chaudhuri B. B., A survey of Hough Transform. Pattern Recognition, Volume 48, Number 3, 2015.

[2] Hough, P. V., U.S. Patent No. 3,069,654. Washington, DC: U.S. Patent and Trademark Office, 1962.

[3] Kowalski, P., Smyk, R., Wykrywanie prostych w obrazie cyfrowym z wykorzystaniem transformacji Hougha, Zeszyty Naukowe Wydziału Elektrotechniki i Automatyki Politechniki Gdańskiej, 2018.

[4] Illingworth, J., Kittler, J., A survey of the Hough transform. Computer vision, graphics, and image processing, Volume 44, Number 1, 1988.

[5] Lu X., Song L., Shen S., He K., Yu S., Ling N., Parallel Hough Transform-based straight line detection and its FPGA implementation in embedded vision, Sensors, Volume 13, Number 7, ISSN 1424-8220, 2013.

[6] Meher P. K., Vallis J., Tso-Bing Juang, Sridharan K., Maharanta K., 50 Years of CORDIC: Algorithms, Architectures, and Applications. IEEE Trans. Circuits Syst. Regul. Pap., Volume 56, Number 9, pp. 1893–1907, 2009.

[7] Czyżak, M., Smyk R., FPGA computation of magnitude of complex numbers using modified CORDIC algorithm. Zeszyty Naukowe Wydziału Elektrotechniki I Automatyki Politechniki Gdańskiej, 47, 35-38., 2015.

[8] Elhossini A., Moussa M., Memory efficient FPGA implementation of Hough transform for line and circle detection, 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE) pp. 1-5, 2012.

[9] Tagzout S., Achour K., Djekoune O., Hough transform algorithm for FPGA implementation, Signal Processing, Volume 81, Number 6, ISSN 0165-1684, 2001.

[10] Guan J., An F., Zhang X., Chen L., Mattausch, H. J., Real-time straight-line detection for XGA-size videos by Hough transform with parallelized voting procedures, Sensors, Volume 17, Number 2, 2017.
[11] Malmir S., Shalchian M., Design and FPGA implementation of dual-stage lane detection, based on Hough transform and localized stripe features, Microprocessors and Microsystems, Volume 43, 2019.
[12] Dong Z., Hu T., Fuchikami R., Ikenaga T., Encoding-free Incrementing Hough Transform for High Frame Rate and Ultra-low Delay Straight-line Detection, 17th International Conference on Machine Vision and Applications (MVA), IEEE, 2021.
[13] Prewitt J. M. S.,Object Enhancement and Extraction, Picture processing and Psychopictorics, Academic Press New York, Volume 10, Number 1,1970.

HARDWARE IMPLEMENTATION OF HOUGH TRANSFORM FOR REAL-TIME LINE DETECTION

The article presents the hardware implementation in FPGA of the algorithm for detecting shapes approximated by a set of straight lines. In the developed hardware structure, the efficiency of processing was increased through the use of pipeline processing, lookup table, using only integer arithmetic and distributed memory. The presented structure was used experimentally in the real-time image processing circuit consisting of the OV7670 camera, Terasic DE10-nano development board and a monitor connected via HDMI. The full processing path has been implemented in a single Intel Cyclone V FPGA chip. The maximum speed of image processing with the use of the developed implementation is 275 MHz.

(Received: 20.09.2021, revised: 25.10.2021)