Politechnika Gdańska

Wydział Elektrotechniki i Automatyki

Gdańsk 2018

doi: 10.32016/1.61.10

IMPLEMENTACJA W FPGA ALGORYTMU DETEKCJI KRAWĘDZI OBRAZU W CZASIE RZECZYWISTYM

Paweł KOWALSKI[1], **Robert SMYK**[2]

1. Politechnika Gdańska, Wydział Elektrotechniki i Automatyki
   e-mail: pawel.kowalski2@pg.edu.pl
2. Politechnika Gdańska, Wydział Elektrotechniki i Automatyki
   e-mail: robert.smyk@pg.edu.pl

Streszczenie: W artykule przedstawiono projekt architektury oraz implementację układową toru przetwarzania wstępnego obrazu z modułem detekcji krawędzi. Układ został zaimplementowany w FPGA Intel Cyclone. Zrealizowany moduł wykorzystuje pięć wybranych algorytmów wykrywania krawędzi, w tym Robertsa, Sobela i Prewitt.

Słowa kluczowe: przetwarzanie obrazu, wykrywanie krawędzi, FPGA

1. ALGORYTMY WYKRYWANIA KRAWĘDZI

Krawędzie stanowią podstawową cechę obrazu. Definiują one granice między regionami w obrazie oraz mogą stanowić podstawowy krok w przypadku dalszej obróbki w procesie segmentacji lub detekcji. Obraz w skali odcieni szarości można zdefiniować jako dwuwymiarową funkcję $f(i,j)$, wiążącą wartość natężenia światła z położeniem piksela o współrzędnych i i j. Wykrywanie krawędzi (krawędziowanie) jest procedurą, która pozwala na wyodrębnienie i wzmocnienie obszarów obrazu charakteryzujących się wysokim progiem wartości intensywności sąsiednich pikseli, co w praktyce przekłada się na silne wzmocnienie tych wartości pikseli, które występują na granicach obszarów obrazu o dużym kontraście.

Podczas obróbki obrazu w czasie rzeczywistym, celem wykrywania i identyfikacji obiektów, krawędziowanie jest wykonywane w pierwszym etapie przetwarzania. Etapem poprzedzającym jest z reguły transformacja obrazu do skali odcieni szarości, zmniejszająca kilkukrotnie ilość danych do późniejszego przetworzenia. Celem zmniejszenia kosztu obliczeń stosuje się przetwarzanie w domenie przestrzennej (ang. spatial domain) [1], które sprowadza się do obliczenia splotu fragmentu obrazu z przyjętym jądrem. W literaturze dla terminu jądra często zamiennie funkcjonuje termin operatora przyjmującego formę macierzową. W dalszej części przedstawiono podstawowe właściwości operatora Robertsa [2], Sobela [3] i Prewitt [4]. Algorytmy wykorzystujące wymienione jądra posłużyły przy realizacji eksperymentu polegającego na implementacji modułu wykrywania krawędzi w strukturze FPGA, przedstawionego w niniejszym opracowaniu.

1.1. Operator krzyżowy Robertsa

Operator krzyżowy Robertsa wykorzystuje okno o rozmiarze 2×2 piksele. Bieżąca wartość piksela jest obliczana według zależności [2]

$$y_{i,j} = \sqrt{x_{i,j}} \tag{1a}$$

$$z_{i,j} = \sqrt{\left(y_{i,j} - y_{i+1,j+1}\right)^2 + \left(y_{i+1,j} - y_{i,j+1}\right)^2}, \tag{1b}$$

gdzie $x_{i,j}$ jest wartością piksela wejściowego o współrzędnych i i j, $y_{i,j}$ reprezentuje pierwiastek kwadratowy z wartości piksela wejściowego, a $z_{i,j}$ jest wartością piksela dla obrazu wynikowego. Podaną zależność można uprościć, zastępując wartością bezwzględną operację pierwiastkowania

$$z_{i,j} = \left|W_{i,j} * R_1\right| + \left|W_{i,j} * R_2\right|, \tag{2}$$

gdzie $z_{i,j}$ reprezentuje piksel obrazu wynikowego a $W_{i,j}$ reprezentuje okno pikseli obrazu wejściowego

$$W_{i,j} = \begin{bmatrix} x_{i,j} & x_{i+1,j} \\ x_{i,j+1} & x_{i+1,j+1} \end{bmatrix} \tag{3}$$

a R_1 i R_2 stanowią maski

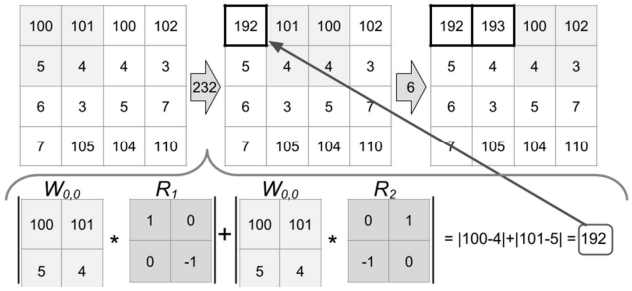
$$R_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, R_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{4}$$

Ostatecznie otrzymuje się

$$z_{i,j} = \left|x_{i,j} - x_{i+1,j+1}\right| + \left|x_{i+1,j} - x_{i,j+1}\right| \tag{5}$$

Na Rys. 1 przedstawiono dwa pełne kroki algorytmu Robertsa. Algorytm dokonuje zmian istniejącego obrazu bez tworzenia nowego, co ogranicza rozmiar używanej pamięci. W pierwszym kroku wykonuje się splot fragmentu obrazu reprezentowanego przez okno $W_{0,0}$ o rozmiarze 2×2 z maskami R_1 i R_2, a rezultat 192 zastępuje poprzednią wartość piksela z lewego górnego rogu ($x_{0,0}$). W drugim kroku nadpisywany jest piksel ($x_{0,1}$) wartością 193. Powstała macierz poddawana jest procesowi binaryzacji, w którym na

podstawie wybranego progu piksel kwalifikowany jest jako krawędź (1) lub brak krawędzi (0).



Rys. 1. Ilustracja kolejnych kroków algorytmu Robertsa

1.2. Operator Sobela

Operator Sobela [3] pracuje w oknie o rozmiarze $3x3$ piksele, wykorzystując dwie maski G_1 i G_2

$$G_1 = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, G_2 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad (6)$$

Procedura wykrywania krawędzi w tym przypadku jest zbliżona do algorytmu Robertsa, uwzględniając większe okno. Algorytm pozwalający na obliczenie wartości pojedynczego piksela jest następujący

$$z_{i,j} = \left| \begin{matrix} -x_{i,j} + x_{i+2,j} - 2\left(x_{i,j+1} - x_{i+2,j+1}\right) \\ -x_{i,j+2} + x_{i+2,j+2} \end{matrix} \right| + \left| \begin{matrix} x_{i,j} + x_{i+2,j} - 2\left(x_{i+1,j} - x_{i+1,j+2}\right) \\ -x_{i,j+2} - x_{i+2,j+2} \end{matrix} \right| \quad (7)$$

gdzie $x_{i,j}$ jest wartością piksela obrazu wejściowego. W celu zachowania skali szarości tożsamej z obrazem wejściowym, wyliczoną wartość $z_{i,j}$ należy przed zapisem do pamięci podzielić przez 4. Operację tę można efektywnie wykonać stosując przesunięcie w prawo o dwa bity.

1.3. Operator Prewitt

Operator Prewitt [4] wykorzystuje okno o rozmiarze 3×3 piksele. Różnica w odniesieniu do operatora Sobela występuje jedynie w prostszej postaci masek P_1 i P_2

$$P_1 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, P_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}. \quad (8)$$

1.4. Operator wykorzystujący 2 piksele

Operatory wykorzysujące 2 piksele, opisane w [5], wymagają okna o rozmiarze 1x2 oraz 1x3

$$M_{1x2} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \; M_{1x3} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \quad (9)$$

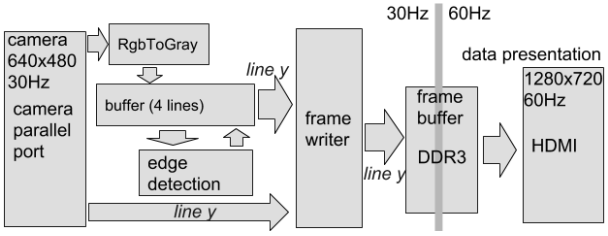
W dalszej części artykułu algorytmy te nazywane są odpowiednio „1x2" oraz „1x3".

2. OPIS EKSPERYMENTU

2.1. Platforma testowa

Przedstawione algorytmy zostały zaimplementowane z wykorzystaniem języka opisu sprzętu VHDL. System wykrywający krawędzie został zaimplementowany w układzie FPGA Intel Cyclone V, 5CSEBA6U2317 umieszczonym w zestawie rozwojowym DE10-Nano [6]. Do zestawu przez interfejs równoległy została podłączona kamera cyfrowa OV7670 [7] oraz monitor HDMI. Dane pobierane są z kamery z prędkością 30 klatek/s, co jest maksymalną wartością dla tej kamery.

Część odpowiedzialna za wyświetlanie obrazu z wykorzystaniem złącza HDMI została zbudowana przy użyciu narzędzia Qsys wykorzystując rdzenie Frame Buffer II IP Core oraz Clocked Video IP Core [8] wchodzące w skład środowiska Quartus Prime 16. Pozostałe części systemu zostały zaprojektowane i opisane w języku VHDL oraz Verilog.



Rys. 2. Schemat blokowy systemu wykrywania krawędzi

Rys. 2 przedstawia schemat blokowy układowej procedury wykrywania krawędzi. Na schemacie można wyróżnić:

- kamerę, która przesyła dane za pomocą portu równoległego z częstotliwością 30 klatek/s,
- bufor wykorzystywany do obliczeń związanych z wykrywaniem krawędzi, bufor ten może pomieścić do 4 linii pikseli, z czego jedna jest zarezerwowana dla danych wyjściowych (wykryte krawędzie)
- moduł wykrywania krawędzi *edge detection*,
- moduł konwersji piksela kolorowego do skali odcieni szarości *RgbToGray*,
- moduł zapisu ramki do pamięci DDR3 *frame writer*,
- bufor ramki *frame buffer* przechowujący jedną ramkę obrazu o rozdzielczości 1280x720, jego zawartość jest na bieżąco modyfikowana przez moduł zapisu ramki,
- moduł obsługi HDMI odpowiedzialny za odczyt danych z bufora ramki (DDR3) oraz przesyłanie ich do monitora za pomocą HDMI.

Prezentacja danych została zrealizowana z wykorzystaniem obrazu o rozdzielczości 1280x720 pikseli, co daje możliwość wyświetlenia dwóch klatek obrazu o rozdzielczości 640x480 pikseli obok siebie (obraz wejściowy - aktualna klatka rejestrowana przez kamerę oraz obraz po krawędziowaniu). Pojedyncza linia pikseli zapisywana przez moduł zapisu ramki podzielona jest na dwie części: obraz z kamery, obraz z krawędziami. Obraz z kamery zapisywany jest na bieżąco z częstotliwością wysyłania pikseli przez kamerę. Kamera po przesłaniu całej linii wysyła sygnał synchronizacji poziomej HREF. W czasie trwania HREF przesyłana jest z bufora druga połowa linii (wykryte krawędzie), wymaga to zwiększenia prędkości przesyłania danych tak, aby przesłać policzoną linijkę y zanim kamera rozpocznie przesyłanie kolejnej linii ($y+1$).

2.2. Sprzętowy moduł wykrywania krawędzi

Na wejście modułu wykrywania krawędzi podawany jest strumień obrazu w skali odcieni szarości. Kamera została skonfigurowana tak, aby wysyłała obraz kolorowy (RGB), umożliwia to wyświetlenie kolorowego podglądu, ale wymusza przed wykrywaniem krawędzi konwersję każdego piksela do skali szarości. Jest to realizowane za pomocą modułu *RgbToGray* z wykorzystaniem zależności

$$gray = R \cdot 0.3 + G \cdot 0.6 + B \cdot 0.1 \quad (9)$$

gdzie *R* oznacza wartość koloru czerwonego, *G* zielonego, a *B* niebieskiego.

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

entity RgbToGray is
   generic (
            In_width:integer:=4
          );
   port (   clk: in std_logic;
            red: in std_logic_vector(In_width downto 0);
            green: in std_logic_vector(In_width downto 0);
            blue : in std_logic_vector(In_width downto 0);
            gray : out std_logic_vector(8 downto 0)
          );
end RgbToGray;

architecture Bechavioral of RgbToGray is
   signal latched_gray: std_logic_vector(In_width+3 downto 0);
begin
   p1: process(clk)
   begin -- 3*red + 6*green + blue
         gray <= ("000" & red & '0') + ("0000" & red)
                 + ( "00" & green & "00") + ("000" & green & '0')
                 + ("0000" & blue);
   end process p1;
end Bechavioral;
```

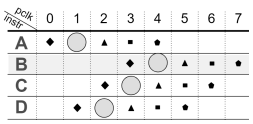
Rys. 3. Moduł konwersji piksela kolorowego do skali odcieni szarości

Moduł *RgbToGray* (Rys. 3) odpowiada za przeniesienie piksela kolorowego do skali odcieni szarości. Obliczenia zostały zrealizowane przy użyciu przesunięć bitowych oraz sumowania.

```
if (rising_edge(pclk)) then
A  cam_buffor(cam_y2, to_integer(unsigned(camX))) <= camGrayData(8 downto 1);

B  if (camX>2) then
      cam_buffor(buffHeight, to_integer(unsigned(camX-3))) <= edge;
   end if;

   if SV0 > SV1 then
C     edge <= SV0-SV1;
   else
      edge <= SV1-SV0;
   end if;

D  SV0 <= cam_buffor(cam_y1, to_integer(unsigned(camX-1)));
   SV1 <= cam_buffor(cam_y2, to_integer(unsigned(camX-1)));
end if;
```

Rys. 4. Kod wykrywania krawędzi metodą 1x2

Rysunek 4 przedstawia kod VHDL odpowiedzialny za wykrywanie krawędzi z wykorzystaniem dwóch pikseli (algorytm 1x2). Zawarte operacje wykonywane są potokowo. Przepływ danych odbywa się zgodnie z taktami zegara *pclk* w kolejności A, D, C, B, gdzie A oznacza zapis aktualnego piksela do bufora, D odczyt z bufora pikseli niezbędnych do wykonania pojedynczego kroku wykrywania krawędzi. Dalej - C to wykonanie obliczeń z wykorzystaniem odczytanych pikseli, a B to zapis wyniku do bufora. Dane wejściowe algorytmu to: *camGrayData* - wartość piksela w skali szarości, *camX* - indeks kolumny w której znajduje się aktualnie przesyłany piksel przez kamerę, *cam_y2* – adres w buforze aktualnie przesyłanego przez kamerę wiersza pikseli, *cam_y1* - adres wiersza o numerze mniejszym o 1.



Rys. 5. Kolejność przetwarzania danych

Na Rys. 5 przedstawiono przepływ danych w trakcie wykrywania krawędzi (Rys. 4). Indeks kolumny odpowiada kolejnym cyklom zegara *pclk*, a nazwa wiersza - wykonywanym blokom instrukcji. Symbolem w kształcie koła zaznaczono przykładowy przepływ danych. Dla wybranego piksela w takcie pierwszym wykonywany jest kod A (piksel jest zapisywany do bufora). Piksel ten z bufora do zmiennej *SV0* wczytywany jest takcie drugim (kod D). W takcie trzecim piksel bierze udział w obliczeniach związanych z wykryciem krawędzi (C). W takcie czwartym odliczona krawędź zapisywana jest do bufora (kod B). W każdym takcie, równolegle wykonywane są kody A, B, C, D, każdy dla innych danych. Przykładowo, w takcie czwartym zostaną wykonane obliczenia (C) z wykorzystaniem piksela wczytanego w takcie 2 (kod A).

```
if (rising_edge(pclk)) then
   cam_buffor(cam_y2, to_integer(unsigned(camX))) <= camGrayData(8 downto 1);

   if (camX>2) then
      cam_buffor(buffHeight, to_integer(unsigned(camX-3))) <= edge;
   end if;

   if SV0 > SV1 then
      edge <= SV0-SV1;
   else
      edge <= SV1-SV0;
   end if;

   SV0 <= cam_buffor(cam_y0, to_integer(unsigned(camX-1)));
   SV1 <= cam_buffor(cam_y2, to_integer(unsigned(camX-1)));
end if;
```

Rys. 6. Kod odpowiedzialny za wykrywanie krawędzi metodą 1x3

Rys. 6 przedstawia sposób wykrywania krawędzi metodą 1x3. Kod ten różni się od sposobu 1x2 wysokością buforu, został on zwiększony o jedną linię.

```
if (rising_edge(cam_pclk)) then
   cam_buffor(cam_y2, to_integer(unsigned(camX))) <= camGrayData(8 downto 1);
   if (camX>2) then
      cam_buffor(buffHeight, to_integer(unsigned(camX-3))) <= edge;
   end if;

   if SV0 > SV1 then
      if SH0 > SH1 then
         edge <= ('0' & ( SV0-SV1 )) + ('0' & ( SH0-SH1 ));
      else
         edge <= ('0' & ( SV0-SV1 )) + ('0' & ( SH1-SH0 ));
      end if;
   else
      if SH0 > SH1 then
         edge <= ('0' & ( SV1-SV0 )) + ('0' & ( SH0-SH1 ));
      else
         edge <= ('0' & ( SV1-SV0 )) + ('0' & ( SH1-SH0 ));
      end if;
   end if;

   SV0 <= ('0' & cam_buffor(cam_y1, to_integer(unsigned(camX-3))));
   SV1 <= ('0' & cam_buffor(cam_y2, to_integer(unsigned(camX-2))));

   SH0 <= ('0' & cam_buffor(cam_y1, to_integer(unsigned(camX-2))));
   SH1 <= ('0' & cam_buffor(cam_y2, to_integer(unsigned(camX-3))));
end if;
```

Rys. 7. Wykrywanie krawędzi algorytmem Robertsa

Rys. 7 przedstawia implementację algorytmu Robertsa. Algorytm ten wykorzystuje 4 piksele w każdym kroku. W przypadku algorytmów Prewitt i Sobela w porównaniu do implementacji z Rys. 7 zmodyfikowana została część D.

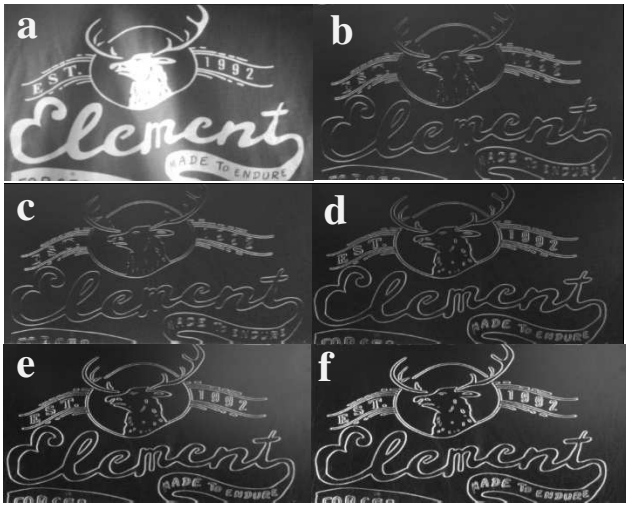
Rys. 8 przedstawia fragmenty implementacji algorytmów Prewitt oraz Sobela, w których występuje różnica względem algorytmu Robertsa (Rys. 7).

```
-- PREWITT
SV0 <= ('0' & cam_buffor(cam_y0, to_integer(unsigned(camX-3))))
      +('0' & cam_buffor(cam_y1, to_integer(unsigned(camX-3))))
      +('0' & cam_buffor(cam_y2, to_integer(unsigned(camX-3))));
SV1 <= ('0' & cam_buffor(cam_y0, to_integer(unsigned(camX-1))))
      +('0' & cam_buffor(cam_y1, to_integer(unsigned(camX-1))))
      +('0' & cam_buffor(cam_y2, to_integer(unsigned(camX-1))));

SH0 <= ('0' & cam_buffor(cam_y0, to_integer(unsigned(camX-3))))
      +('0' & cam_buffor(cam_y0, to_integer(unsigned(camX-2))))
      +('0' & cam_buffor(cam_y0, to_integer(unsigned(camX-1))));
SH1 <= ('0' & cam_buffor(cam_y2, to_integer(unsigned(camX-3))))
      +('0' & cam_buffor(cam_y2, to_integer(unsigned(camX-2))))
      +('0' & cam_buffor(cam_y2, to_integer(unsigned(camX-1))));
```

```
-- SOBEL
SV0 <= ("00" & cam_buffor(cam_y0, to_integer(unsigned(camX-3))))
      +('0' & cam_buffor(cam_y1, to_integer(unsigned(camX-3))) & '0')
      +("00" & cam_buffor(cam_y2, to_integer(unsigned(camX-3))));

SV1 <= ("00" & cam_buffor(cam_y0, to_integer(unsigned(camX-1))))
      +('0' & cam_buffor(cam_y1, to_integer(unsigned(camX-1))) & '0')
      +("00" & cam_buffor(cam_y2, to_integer(unsigned(camX-1))));

SH0 <= ("00" & cam_buffor(cam_y0, to_integer(unsigned(camX-3))))
      +('0' & cam_buffor(cam_y0, to_integer(unsigned(camX-2))) & '0')
      +("00" & cam_buffor(cam_y0, to_integer(unsigned(camX-1))));

SH1 <= ("00" & cam_buffor(cam_y2, to_integer(unsigned(camX-3))))
      +('0' & cam_buffor(cam_y2, to_integer(unsigned(camX-2))) & '0')
      +("00" & cam_buffor(cam_y2, to_integer(unsigned(camX-1))));
```

Rys. 8. Wykrywanie krawędzi algorytmem Prewitt oraz Sobela

Efekt wykrywania krawędzi w obrazie z kamery został przedstawiony na Rys. 9. Różnica w ilości zajmowanych elementów logicznych w układzie FPGA została zaprezentowana w Tabeli 1.



Rys. 9. Efekt wykrywania krawędzi. a) obraz wejściowy, b) algorytm 1x2, c) algorytm 1x3, d) algorytm Robertsa, e) algorytm Prewitt, f) algorytm Sobela

Tabela 1. Liczba wykorzystanych elementów logicznych FPGA dla zaimplementowanych algorytmów

algorytm	**Elementy logiczne**
bez wykrywania krawędzi	6955
Roberts	21208
Sobel	27983
Prewitt	27958
1x2	16501
1x3	16373

Porównanie zajętości elementów logicznych przedstawione w Tab. 1 dotyczy całej aplikacji przetwarzającej obraz. Struktura przesyłająca obraz z kamery na monitor zajmuje ok. 7 tys. elementów logicznych. Struktury wykrywania krawędzi wykorzystujące dwie wartości (1x2, 1x3) zajmują ok. 9.5 tyś (16501 – 6955). Struktura wykorzystująca algorytm Robertsa (4 piksele) zajmuje 50% więcej miejsca, a struktury wykorzystujące 9 pikseli (Sobel, Prewitt) ponad 2 razy więcej miejsca.

Tabela 2. Opóźnienia zaimplementowanych algorytmów podczas wykonania pojedynczego kroku procedury wykrywania krawędzi

Algorytm	**t [ns]**	**max f_{pix} [MHz]**	**max [fps]**	
			640x480	**FullHD**
1x2	6.9	145	472	69
1x3	6.4	156	507	75
Roberts	6.7	149	485	71
Prewitt	6.4	156	507	75
Sobel	7.5	133	432	64

W Tab. 2 przedstawiono zestawienie algorytmów, w którym uwzględniono: t – czas wykonania pojedynczego kroku zaimplementowanego krawędziowania w układzie FPGA Intel Cyclone V, 5CSEBA6U2317 oszacowany przy użyciu TimeQuest Timing Analyzer, $max\ f_{pix}$ – maksymalna częstotliwość dla jakiej krawędzie będą wykrywane poprawnie (wyliczone na podstawie t) oraz *max [fps]* - maksymalna częstotliwość obrazu o rozdzielczości 640x480 i 1920x1080 (FullHD) wyrażona w klatkach/s.

3. PODSUMOWANIE

W artykule przedstawiono sposób implementacji wybranych algorytmów wykrywania krawędzi w FPGA. Funkcjonowanie algorytmów zostało sprawdzone eksperymentalnie. Porównana została liczba elementów logicznych, jaką zajmuje każda implementacja. Wykonano również oszacowanie opóźnień.

4. BIBLIOGRAFIA

1. Juneja M, Sandhu PS. Performance Evaluation of Edge Detection Techniques for Images in Spatial Domain. International Journal of Computer Theory and Engineering. 2009;614–21.
2. Roberts L. G.: Machine perception of three-dimensional solids, PhD thesis, MIT, Lincoln Laboratory, May 22, 1963, pp. 82.
3. Sobel I., Feldman G., An isotropic 3x3 image gradient operator, presented at Stanford Artificial Intelligence Project (SAIL), 1968.
4. Prewitt J.M.S.,Object Enhancement and Extraction, Picture processing and Psychopictorics, Academic Press,1970.
5. Burns J. B., Hanson A. R., Riseman E. M., Extracting Straight Lines, IEEE Transactions on Pattern Analysis & Machine Intelligence, Volume 8, Number 4, 1986, pp. 425-455.
6. DE10-nano Cyclone V Soc with Dual-core ARM Cortex-A9 User Manual. Terasic Technologies; 2018.
7. OV7670/OV7171 CMOS VGA(640X480) CameraChip with OmniPixel Technology Advances Information Preliminary Datasheet. OmniVision; 2005.
8. Video and Image Processing Suite User Guide. Intel; 2018.

FPGA IMPLEMENTATION OF EDGE DETECTION ALGORITHMS ON REAL-TIME IMAGE

The paper presents FPGA implementation details of the hardware image processing block with the edge detection module. In the implemented video processing module we use five selected edge detection algorithms, including Roberts, Sobel and Prewitt. The structure was synthesized and packed using hardware design platform built around the Intel Cyclone V FPGA. The number of logic elements used in each implementation was compared. We also estimated the execution time and maximum possible frame rate in VGA (640x480) and FullHD (1920x1080) video stream.

Keywords: image processing, edge detection, FPGA.