AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I ELEKTRONIKI

KATEDRA ELEKTRONIKI

Projekt dyplomowy

inżynierski

Akceleracja symulacji wybranego sprzętowego kontekstowego filtru obrazu na platformie HES

Simulation acceleration of the example context image filter on HES platform

Imię i nazwisko	*Paweł Sokołowski*
Kierunek studiów	Elektronika i Telekomunikacja
Opiekun pracy	*dr inż. Jerzy Kasperek*

Kraków, rok 2010

Oświadczam, świadomy odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem osobiście i samodzielnie i że nie korzystałem ze źródeł innych niż wymienione w pracy.

.....

Podpis dyplomanta

Spis treści

Wstęp

Przetwarzanie i analiza obrazów jest jednym z najważniejszych dziedzin techniki w dzisiejszym świecie. Najistotniejszym zmysłem człowieka jest bowiem wzrok. Na jego podstawie podejmuje się większość decyzji. Dzięki systemom wizyjnym można usprawniać pracę w wielu dziedzinach. Przykładowo chirurg znajdujący się w jednym mieście może przeprowadzić operację w drugim mieście za pomocą sterowanego robota i obrazu pacjenta, który jest przesyłany i wyświetlany na jego stanowisku pracy. Bardzo ważnym aspektem w tej sytuacji jest uzyskanie wyraźnego obrazu bez zakłóceń. Można to uzyskać dzięki filtracji obrazu przesyłanego łączami komunikacyjnymi. Powinna wykonywać się ona na tyle szybko, aby obraz również nie był przerywany, czy klatkowany.

To właśnie jest zadaniem tego projektu, aby stworzyć syntezowalny kod filtracji kontekstowej, która będzie wykonywana z maksymalną prędkością i dokładnością. Moduł filtru będzie umieszczany na platformie sprzętowej z układem FPGA (ang. *Field Programmable Gate Array*), dzięki czemu operacje mogą być wykonywane równolegle. Właściwość ta przyspieszy wszystkie działania, które w normalnym systemie przeprowadzane byłyby szeregowo.
Wykonano także *tutorial*, który umożliwi studentom stworzenie prostego projektu, wykorzystującego opisane w pracy rozwiązania programowe i sprzętowe. Został on dołączony na końcu dokumentacji.

W ramach projektu stworzono dwa rodzaje filtru w celu przedstawienia możliwości układów FPGA oraz pokazania trudności i problemów związanych głównie z typami danych i wykonywanych operacji arytmetycznych. Pierwszym jest podstawowy filtr uśredniający, w przypadku którego wykonywane są proste operacje dodawania. Jednak pojawia się problem z normalizacją, gdyż stosując macierz konwolucji 3x3 powinno podzielić otrzymaną wartość przez 9. Natomiast proste i szybkie operacje wymagają obliczeń typu mnożenie/dzielenie na potęgach dwójki. Oczywiście można wykonać dzielenie lub mnożenie dowolnej wartości. Nawet układy FPGA mają dedykowane układy to tego celu, jednak, wykorzystując je, diametralnie zmniejsza się maksymalna szybkość taktowania.
Dużo lepszy efekt można uzyskać, wykonując tylko przesunięcia bitowe (dzielenie/mnożenie przez wielokrotność dwójki) i wykonanie pewnego przybliżenia, co zostało opisane w rozdziale 3.2.1 poświęconym tej filtracji.

Drugi z wykonanych filtrów to filtr wyostrzający (ang. *high boost*), który także posiada problem z normalizacją. Jest on tym większy, że operacje przeprowadzane są na liczbach ze znakiem. Należy wykonać odpowiednią interpretację próbki wynikowej, aby otrzymać na wyjściu format 8-bitowy bez znaku.

Na początku projektu opisana jest teoretycznie filtracja kontekstowa. Przedstawiony został opis matematyczny, problemy związane z tego typu filtracją oraz jak powinna wyglądać prawidłowa filtracja dolno- i górnoprzepustowa.
W kolejnym rozdziale omówiono w jaki sposób można przyspieszyć symulację. Opisano w nim działanie platformy HES (ang. *Hardware Embedded Simulation*) i środowiska DVM (ang. *Design Verification Manager*).
Następne dwa rozdziały przedstawiają wykonane moduły. Pierwszy z nich zawiera opis syntezowalnych układów filtrów kontekstowych. Natomiast w kolejnym omówiono sposób ich weryfikacji i potrzebne do tego środki.
W ostatnim rozdziale krótko zweryfikowano skuteczność i efektywność działania stworzonych modułów, które dowodzą słuszności stosowania szybkich układów i równoległych działań w przypadku filtracji kontekstowej obrazu. Porównano także czasy: symulacji programowej oraz kosymulacji sprzętowej.

1. Kontekstowa filtracja obrazu

1.1. Konwolucja dyskretna – filtracja splotowa

Obraz reprezentowany jest przez 2-wymiarową macierz wartości dyskretnych *L(m,n)*. W celu filtracji danego obrazu należy wykonać jej splot z macierzą filtrującą *w(i,j)*. Wzór tej operacji można zapisać w następujący sposób:

$$L'(m,n) = (w \times L)(m,n) = \sum_{i,j \in K} L(m-i, n-j) w(i,j) \tag{1}$$

Do obliczenia wartości funkcji *L'(m,n)* w danym punkcie na obrazie wynikowym wykorzystuje się współczynniki *w(i,j)*, wraz z odpowiednimi elementami obrazu *L(m-i,n-j)*, znajdującymi się w oknie rozlokowanym wokół punktu o współrzędnych *(m,n)*. Współczynniki *w(i,j)* wybiera się zwykle w taki sposób, by były liczbami całkowitymi. Jest to spowodowane tym, że dla wartości zmiennoprzecinkowych trzeba wykonać skomplikowane obliczenia. Wymagają one dużej mocy obliczeniowej oraz pochłaniają dużo więcej czasu. Jednak przy całkowitoliczbowych wartościach *w(i,j)* pojawia się problem normalizacji. Po wykonaniu wymaganych działań uzyskane wartości wynikowych pikseli wyjściowego obrazu nie będą zawierały się w przedziale $L'(m,n) \in [0, 2^B-1]$. Dlatego pełna operacja filtracji splotowej musi obejmować czynność normalizacji. Dla filtrów eliminujących proste zakłócenia, dla których wszystkie współczynniki spełniają warunek $w(i,j) \geq 0$ możliwe jest zastosowanie stosunkowo prostej techniki normalizacji, danej wzorem:

$$L'(m,n) = \frac{1}{\sum_{(i,j) \in K} w(i,j)} \sum_{(i,j) \in K} L(m-i, n-j) w(i,j) \tag{2}$$

W przypadku współczynników *w(i,j)* przyjmujących zarówno wartości dodatnie, jak i ujemne, operacja normalizacji musi odwoływać się do rzeczywistych, uzyskanych po przetworzeniu obrazu wartości *min L'(m,n)* oraz *max L'(m,n)* i opierać się na wzorze:

$$L''(m,n) = \frac{L'(m,n) - \min L'(m,n)}{\max L'(m,n) - \min L'(m,n)} 2^B \tag{3}$$

Otoczenie oraz liczba elementów wchodzących w jego skład mogą być dowolnie dobierane w zależności od potrzeb i zastosowań. Ogólnie im większy rozmiar otoczenia, tym bardziej radykalne działanie filtru. W zdecydowanej większości przypadków stosuje się jednak okna kwadratowe o wymiarach 3x3. Jest to spowodowane tym, że daną próbkę można umieścić w środku macierzy oraz wymaga dużo mniej operacji matematycznych niż w przypadku filtrów o większych rozmiarach okna. Dlatego też w projekcie stosowany jest właśnie taki wariant okna. Tablica współczynników *w(i,j)* takiego okna przyjmuje następującą postać:

$$\begin{bmatrix} w(1,1) & w(1,0) & w(1,-1) \\ w(0,1) & w(0,0) & w(0,-1) \\ w(-1,1) & w(-1,0) & w(-1,-1) \end{bmatrix}$$

Dla szybkiego i wygodnego opisywania kolejnych filtrów przyjęto natomiast uproszczoną notację, wykorzystującą jedynie kolejne numery współczynników:

$$\begin{bmatrix} w_1 & w_2 & w_3 \\ w_4 & w_5 & w_6 \\ w_7 & w_8 & w_9 \end{bmatrix}$$

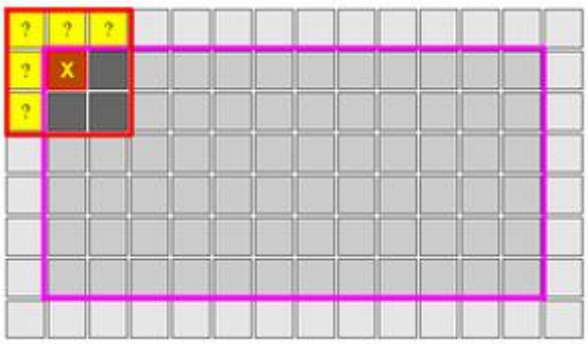
W tym przypadku funkcja realizująca proces filtracji (z użyciem konwolucji) może być zapisana w następujący sposób:

$$\begin{aligned} L'(m,n) = {} & w_1 L(m-1,n-1) + w_2 L(m-1,n) + w_3 L(m-1,n+1) + \\ & + w_4 L(m,n-1) + w_5 L(m,n) + w_6 L(m,n+1) + \\ & + w_7 L(m+1,n-1) + w_8 L(m+1,n) + w_9 L(m+1,n+1) \end{aligned} \tag{4}$$

1.2. Problemy związane z filtracją kontekstową

Jednym z problemów jest zagadnienie normalizacji związany z operacjami na współczynnikach w postaci liczb całkowitych, o czym była mowa w poprzednim rozdziale. Drugim problemem są warunki brzegowe filtracji. Do wyznaczenia jednego punktu obrazu wynikowego trzeba dokonać określonych obliczeń na wielu punktach z otoczenia danego punktu obrazu

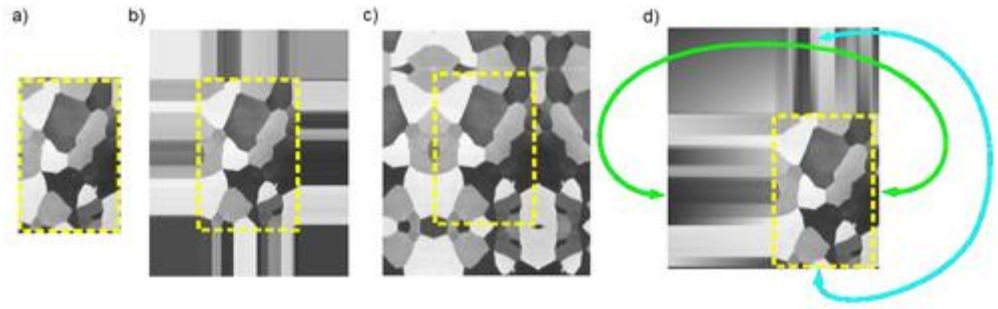
źródłowego. Z tego powodu nie można dokonać filtracji dla pikseli znajdujących się bezpośrednio na brzegu obrazu (rys. 1.), ponieważ do jej wykonania będzie brakować wartości argumentów oznaczonych symbolem „**?**”.



Rys. 1. Niemożność wykonania kontekstowych filtracji dla punktów położonych przy brzegu obrazu [2].

Najprostszym rozwiązaniem jest pominięcie tych elementów, dzięki czemu uzyskujemy tablicę sygnału wyjściowego mniejszą od tablicy wejściowej. Możliwa jest również ekstrapolacja elementów początkowych i końcowych tak, aby zastąpiły one elementy brakujące. Jest kilka sposobów wykonania tej operacji:

- powielenie pikseli brzegowych (rys. 2b),
- odbicie lustrzane – w przypadku okna 3x3 daje taki sam efekt jak w pierwszym punkcie (rys. 2c),
- interpolacja między pikselami brzegowymi (rys. 2d).



Rys. 2. Ekstrapolacja elementów brzegowych: a) obraz źródłowy, b) powielenie pikseli brzegowych, c) odbicie lustrzane, d) interpolacja między pikselami brzegowymi [2].

1.3. Filtry dolnoprzepustowe

Typowym zastosowaniem filtrów jest usuwanie zakłóceń z obrazu. Przy tego typu operacji często korzysta się z prostego filtru uśredniającego, którego macierz konwolucji ma postać:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Obrazem poddawanym filtracji jest tradycyjnie „Lena”, która została przedstawiona na rysunku 3.

Rys. 3. Obraz źródłowy „Lena” w formacie RBG i w skali szarości [3].

Na rysunku 4 przedstawiono przykład efektu filtracji dolnoprzepustowej uśredniającej na zakłócenie typu „sól i pieprz” o gęstości 0.008. Niestety przekształcenie to nie powoduje usunięcia szumu, lecz powoduje jedynie jego rozmycie. Skutkiem ubocznym tej filtracji jest także wygładzenie wszystkich krawędzi obiektów. Następuje pogorszenie rozpoznawalności ich kształtów. Obraz staje się pozbawiony szczegółów i niewyraźny.

Rys. 4. Przykład filtracji uśredniającej, od lewej: obraz z zakłóceniem typu „sól i pieprz”, efekt filtracji.

Filtracja uśredniająca daje lepsze wyniki w przypadku szumu białego w postaci funkcji Gaussa. W przykładzie przedstawionym na rysunku 5 funkcja Gaussa miała średnią równą 0 i wariancję wynoszącą 0.003.

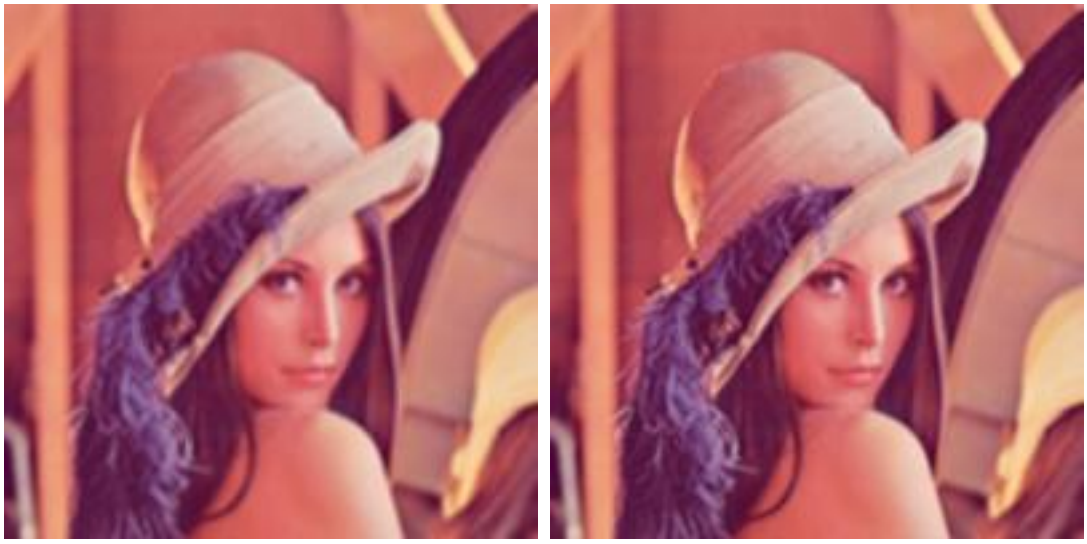
Rys. 5. Efekt filtracji uśredniającej na szum biały w postaci funkcji Gaussa. Od lewej: obraz zakłócony, obraz po filtracji.

Utratę ostrości obrazu podczas filtracji splotowej można zmniejszyć przez zastosowanie filtrów uśredniających wykonujące konwolucję w sposób ważony. Pierwotna wartość piksela *L(m,n)* w większym stopniu wpływa na wartość piksela po przetworzeniu *L'(m,n)*. Aby uzyskać taki efekt wykorzystuje się macierze konwolucji zawierające większe wzmocnienie dla punktu *L(m,n)* w stosunku do jego sąsiadów. Przykładowe maski wyglądają następująco:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Druga z przedstawionych masek reprezentuje funkcję Gaussa. Jeśliby kolejne elementy tej maski przedstawiono za pomocą słupków o wysokości odpowiadającej przypisanej wadze to w efekcie otrzymano by bryłę podobną do krzywej rozkładu normalnego - krzywej Gaussa. Zatem znaczenie wartości punktu rośnie wraz ze zmniejszaniem się odległości do obliczanego punktu, w sposób opisany przez funkcję Gaussa.

Zastosowanie takich masek powoduje zmniejszenie efektu rozmycia obrazu, co można zaobserwować na rysunku 6.



Rys. 6. Przykład obrazu poddanego filtracji z uwydatnionym punktem centralnym w macierzy konwolucji. Od lewej: obraz po filtracji uśredniającej podstawowej, obraz po filtracji maską Gaussa.

Można również zastosować konwolucję z maską pozbawioną całkowicie centralnego elementu:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

W ten sposób wpływ piksela środkowego *L(m,n)* jest zerowy i wartość punktu wyjściowego *L'(m,n)* zostaje wyznaczony na podstawie tylko jego otoczenia. Efekt tej filtracji można zaobserwować na rysunku 7. W wyniku tej operacji otrzymano jeszcze większe rozmycie konturów obrazu.

Rys. 7. Obraz poddany filtracji z pominiętym punktem centralnym w macierzy konwolucji. Od lewej: plik źródłowy, obraz po filtracji.

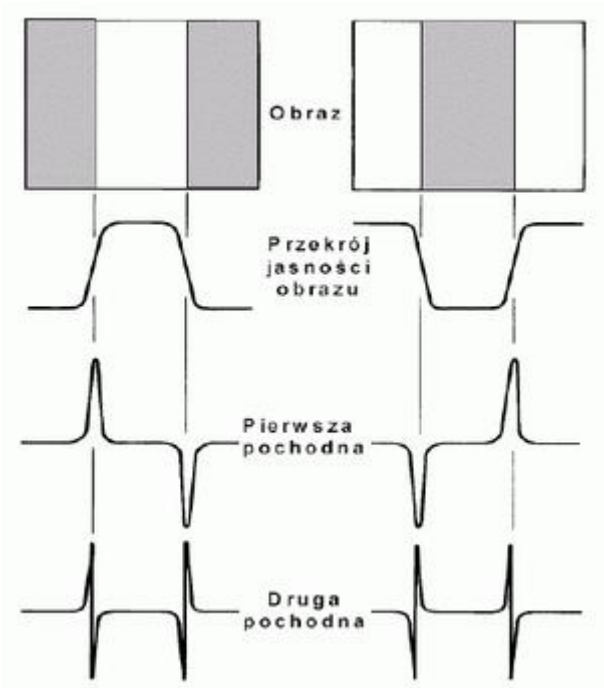
1.4. Filtry górnoprzepustowe

Filtracja górnoprzepustowa ma na celu wyostrzenie obrazu, podkreślenie jego krawędzi, konturów, czy też wydobycie elementów faktury. Wzmacnia szybkie zmiany jasności pomiędzy pikselami (wzmacnia wyższe częstotliwości obrazu). Negatywnym skutkiem tego zabiegu jest uwydatnienie również szumów i zakłóceń oraz niepotrzebnych różnic na gładkich powierzchniach obrazu.

1.4.1. Filtry gradientowe

Lokalne zmiany jasności obrazu niosą informację o granicach obszarów (obiektów). Detekcja krawędzi pozwala na identyfikację położenia obiektów w obrazie. Większość metod wykrywania krawędzi polega na wyznaczaniu lokalnych pochodnych (tzw. operatorów gradiento-

wych). Pierwsza pochodna obrazu wykorzystywana jest do detekcji krawędzi oraz jej kierunku. Druga pochodna natomiast, pozwala określić czy intensywność obrazu za krawędzią jest większa, czy mniejsza od intensywności obrazu przed krawędzią [1] (patrz rys. 8).



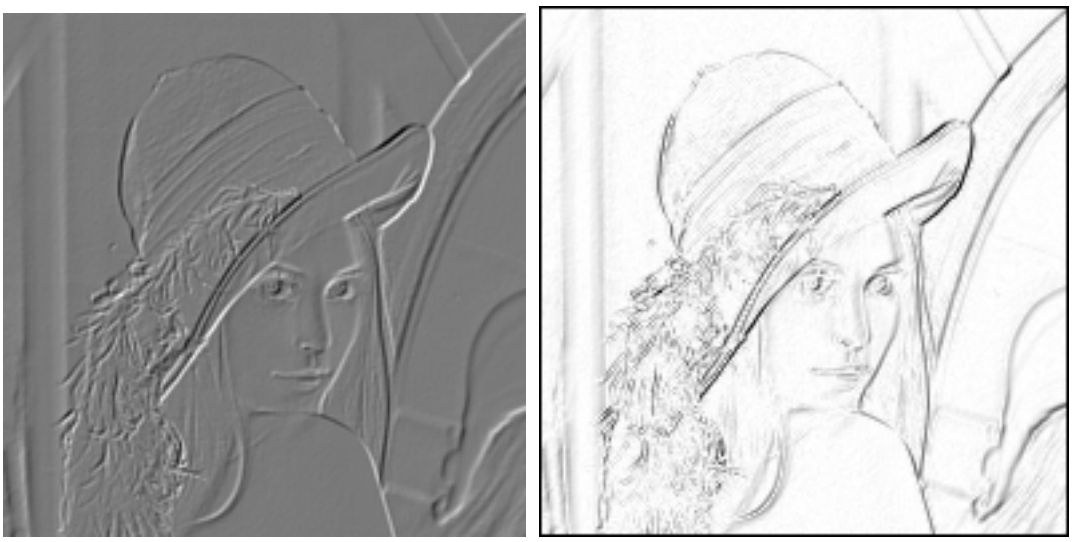
Rys. 8. Detekcja granic obszarów operatorami gradientowymi [2].

Najprostszym przykładem tego typu filtru jest gradient Robertsa. Jedna z wielu macierzy współczynników *w(i,j)* dla tej filtracji wygląda następująco:

$$\begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

Pomimo tego że gradient Robertsa można opisać za pomocą macierzy o wymiarach 2x2, zastosowano macierz o wymiarach 3x3, ponieważ dzięki takiej budowie jednoznacznie można określić, dla którego elementu przeprowadzana jest filtracja. Przykład różniczkowania tym oknem można zobaczyć na rysunku 9.

W celu wykonania poprawnej filtracji, należy obraz w formacie RBG przekonwertować na obraz w odcieniach szarości. W związku z tym obecnym obrazem źródłowym jest prawy obraz „Lena” z rysunku 3.



Rys. 9. Filtracja obrazu gradientem Robertsa. Od lewej: obraz po wykonaniu filtracji, obraz po wzięciu modułu wartości pikseli po wykonaniu filtracji.

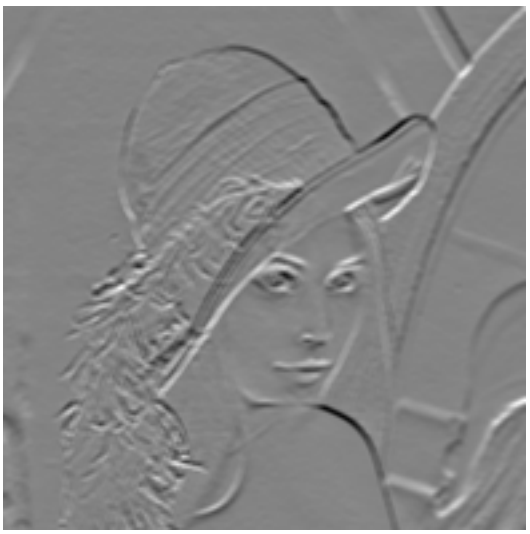
Szarość na pierwszym obrazie na rysunku 9 jest spowodowane tym, że po wykonaniu konwolucji gradientem Robertsa w obrazie wynikowym występują zarówno piksele o wartościach dodatnich, jak i ujemnych. Aby obraz został poprawnie wyświetlony należy wykonać skalowania i środek skali przypada na wartość 128, co odpowiada tej szarości.
Jest to cecha wszystkich konwolucji filtrów górnoprzepustowych, ponieważ mają one w swoich oknach przekształceń zarówno wartości dodatnie współczynników, jak i ujemne. Zjawisko to natomiast nie występowało dla filtrów dolnoprzepustowych.

Porównując obraz źródłowy (patrz rys. 3) i otrzymany po filtracji można zauważyć, że gradient Robertsa ma kierunkowy charakter. Jest to spowodowane tym, iż różniczkowanie funkcji dwuwymiarowej musi zawsze odbywać się wzdłuż pewnego kierunku.

Na podstawie gradientu Robertsa można stworzyć tzw. maski Prewitta, które pozwalają na różniczkowanie obrazu w różnych kierunkach. Przykładowo maska dokonująca konwolucji w kierunku poziomym wygląda następująco:

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

Efekt działania tej maski przedstawiony jest na rysunku 10.



Rys. 10. Obraz filtrowany górnoprzepustowo z użyciem poziomej maski Prewitta.

Przez prostą operację transpozycji maski Prewitta można uzyskać filtr wykrywający elementy o orientacji pionowej. Maska wygląda wówczas następująco:

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Przykład działania tej maski można zaobserwować na rysunku 11.

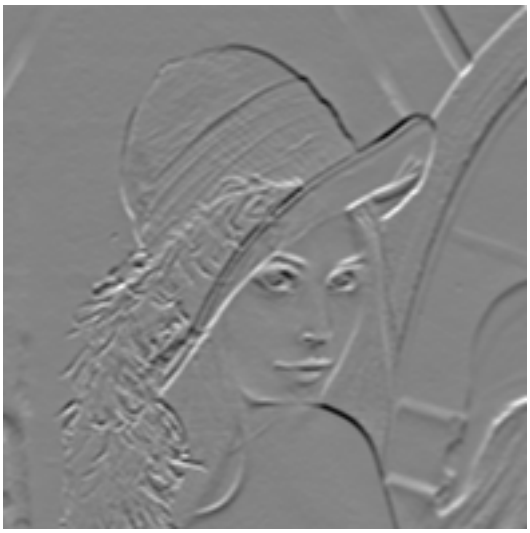
Rys. 11. Obraz filtrowany górnoprzepustowo z użyciem pionowej maski Prewitta.

Patrząc na otrzymane obrazy przez filtrację maskami Prewitta, dokładnie widać kierunkowość filtracji. Można także dokonać wzmocnienia wpływu najbliższego otoczenia piksela, dla któ-

rego wyznaczana jest wartość piksela na obrazie wynikowym. Taką filtrację można wykonać za pomocą tzw. masek Sobela. Pozioma maska Sobela ma postać:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Skutek działania tej maski pokazuje rysunek 12.



Rys. 12. Obraz filtrowany górnoprzepustowo z użyciem poziomej maski Sobela.

Maskę Sobela podobnie jak Prewitta można transponować uzyskując maskę pionową, która wygląda następująco:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Przykładowy obraz przefiltrowany tą maską przedstawiono na rysunku 13.

Rys. 13. Obraz filtrowany górnoprzepustowo z użyciem pionowej maski Sobela.

W celu uzyskania innych kierunkowości filtracji można obrócić maskę Sobela o 45º. Taką samą operację można również wykonać dla wcześniej prezentowanych masek. Poniżej zaprezentowane zostały dwie spośród dziewięciu możliwych masek Sobela:

$$\begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix} \qquad \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$$

1.4.2. Filtry wykrywające krawędzie

Filtracje wykonane za pomocą macierzy konwolucji opisanych w poprzednim podrozdziale (patrz rozdział 1.4.1) mają wspólną cechę, jaką jest uwydatnienie pewnych właściwości filtrowanego obrazu. Były to krawędzie i kontury elementów znajdujących się na obrazie źródłowym. Jednak wadą tych filtracji jest ich kierunkowość. Można to zauważyć, analizując przykłady z poprzedniego rozdziału. Akcentowały one elementy o danej orientacji, natomiast pozostałe były tłumione.

Czasami jednak potrzebne jest wykrycie wszelkich krawędzi występujących na badanym obrazie, bez względu na ich kierunkowość. Wówczas należy użyć innych masek. Jedną z metod jest zastosowanie filtracji nieliniowej (patrz [1]). Dobre efekty dają również tak zwane *lapla-*

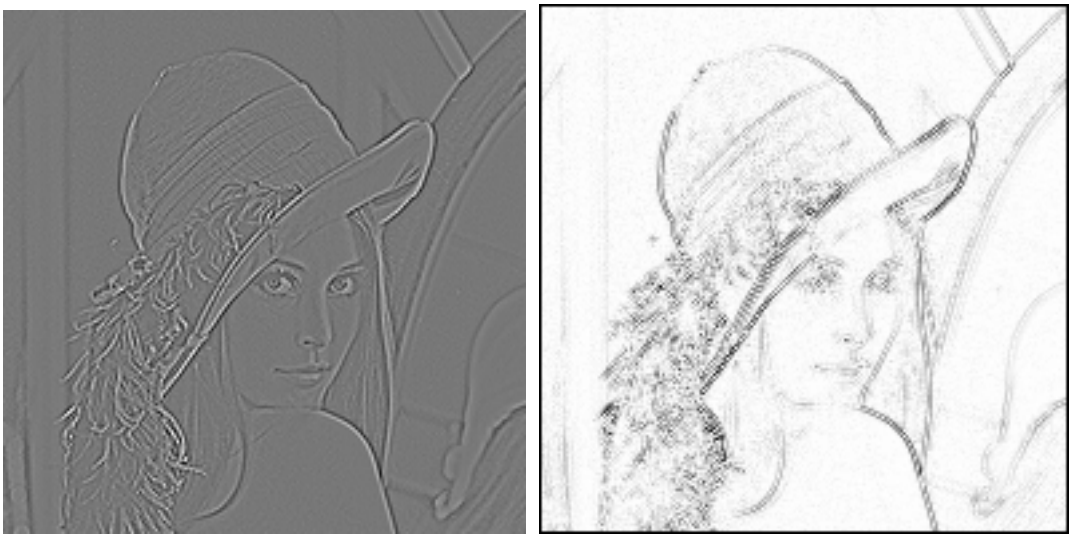
sjany. Z matematycznego punktu widzenia laplasjan jest kombinacją drugich pochodnych cząstkowych wejściowej funkcji *L(m,n)*:

$$L'(m,n) = \frac{\partial^2 L(m,n)}{\partial m^2} + \frac{\partial^2 L(m,n)}{\partial n^2} \tag{5}$$

Natomiast w przetwarzaniu obrazów praktycznie maska konwolucji nazywana laplasjanem wygląda następująco:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Efekt filtracji tym oknem obrazu sztucznego i rzeczywistego przedstawiony jest na rysunku 14.



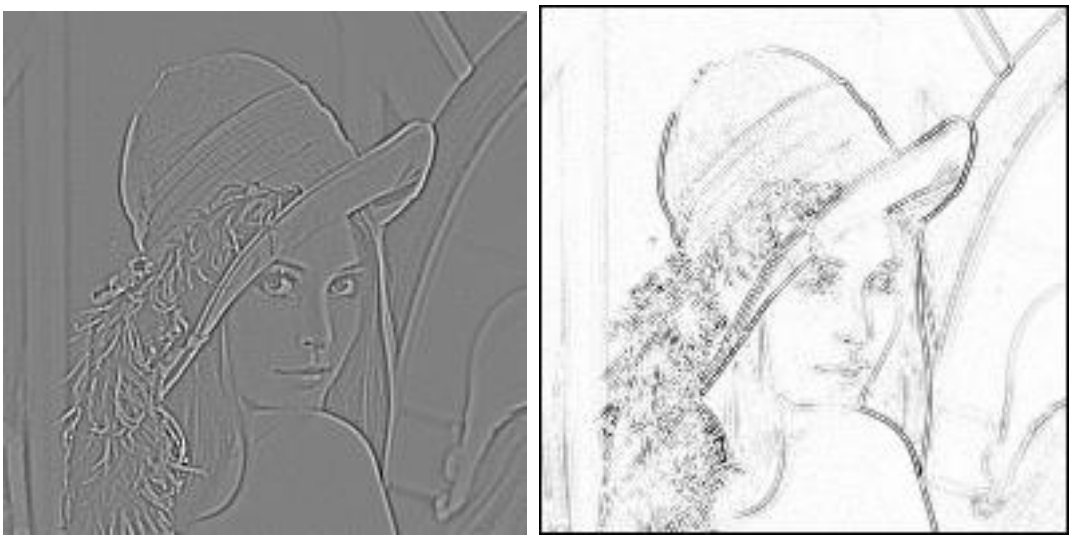
Rys. 14. Wynik splotu obrazu z maską laplasjanu. Od lewej: obraz po wykonaniu filtracji, obraz po wzięciu modułu wartości pikseli po wykonaniu filtracji.

Na przedstawionym przykładzie (patrz rys.14) można zauważyć, że filtracja za pomocą maski laplasjanu uwydatnia wszelkie linie i krawędzie na obrazie.

Innym przykładem maski laplasjanu, tym razem bardziej rozbudowanej, może być następujące okno:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Skutki działania konwolucji za pomocą tej maski widać na rysunku 15.



Rys. 15. Efekt splotu obrazu z rozbudowaną maską laplasjanu. Od lewej: obraz po wykonaniu filtracji, obraz po wzięciu modułu wartości pikseli po wykonaniu filtracji.

1.4.3. Filtry wyostrzające – *high boost*

Przedstawione w poprzednich rozdziałach filtry górnoprzepustowe wykrywały pewne własności obrazu, jakimi są kontury i krawędzie. Jednak tracona była informacja niesiona przez pozostałe fragmenty obrazu źródłowego. W celu zachowania tej informacji należy przeprowadzić filtrację wyostrzającą (ang. *high boost*). Filtracja ta podkreśla wysokie częstotliwości obrazu, reprezentujące jego szczegóły, bez likwidacji niskich częstotliwości, reprezentujące pozostałe piksele. Tę cechę przedstawia następujący wzór:

$$L''(m,n) = c \cdot L(m,n) + L'(m,n) = [(c \cdot W_{allpass} + W_{highpass}) \times L](m,n) = (W_{highboost} \times L)(m,n) \quad (6)$$

gdzie: c – stała;

$W_{allpass}$ – macierz wszechprzepustowa;

$W_{highpass}$ – macierz filtracji górnoprzepustowej;

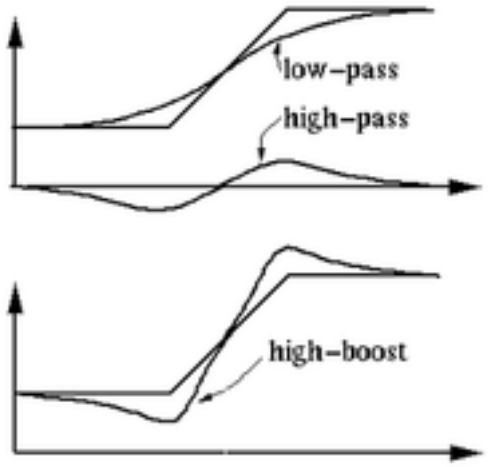
$W_{highboost}$ – macierz filtracji wyostrzającej;

Przykładem maski wyostrzającej może być połączenie maski wszechprzepustowej przemnożonej przez stałą c z laplasjanami przedstawionymi w poprzednim podrozdziale (patrz rozdział 1.4.2):

$$W_{highboost} = c \cdot W_{allpass} + W_{highpass} = c \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4+c & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad (7)$$

$$W_{highboost} = c \cdot W_{allpass} + W_{highpass} = c \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8+c & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (8)$$

Z powyższych wzorów wynika, że dla $c = 0$ będzie taki sam efekt, jak przy filtracji górnoprzepustowej. Natomiast dla $c > 0$ część oryginalnego obrazu zostanie zachowana.
Dokładniej można to zaobserwować na wykresach przedstawionych na rysunku 16.



Rys. 16. Wykresy przedstawiające różnice pomiędzy filtracjami: dolnoprzepustowej, górnoprzepustowej i wyostrzającej (ang. *high-boost*) [4].

Przykładowa maska tej filtracji dla $c = 1$ i wykorzystując rozbudowaną maskę laplasjanu wygląda następująco:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Efekt działania tego okna przedstawiony został na rysunku 17.

Rys. 17. Rzeczywisty obraz przefiltrowany maską wyostrzającą (ang. *high-boost*). Od lewej: obraz źródłowy, obraz po filtracji.

Przez skalowanie maski filtracji górnoprzepustowej można wpływać na intensywność wyostrzania obrazu:

$$W_{highboost} = c \cdot W_{allpass} + \frac{1}{k} \cdot W_{highpass} = c \cdot \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \frac{1}{k} \cdot \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (9)$$

gdzie: k – stała;

Dla wartości $k > 1$ następuje zmniejszenie wpływu filtracji górnoprzepustowej na obraz wynikowy. Efekt przetwarzania dla $k = 4$ pokazany jest na rysunku 18.

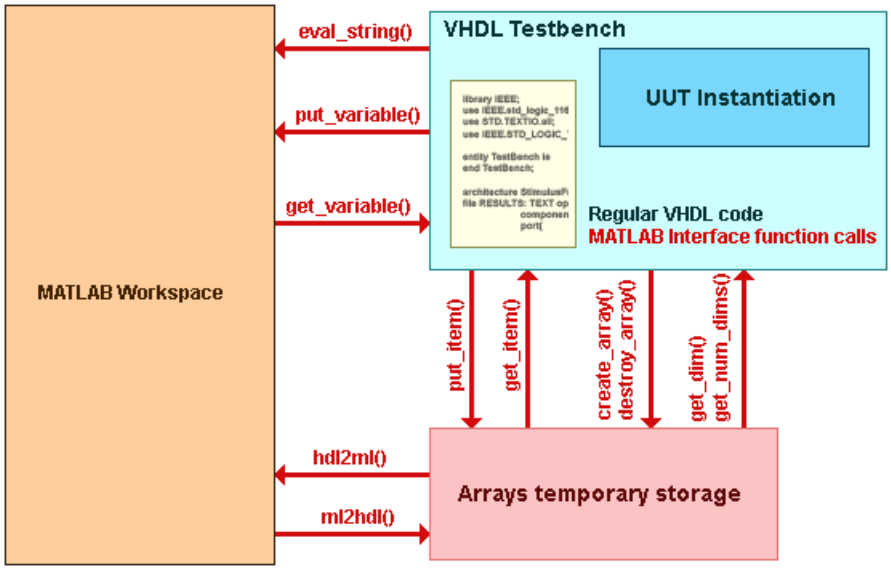
Rys. 18. Rzeczywisty obraz przefiltrowany maską wyostrzającą (ang. *high-boost*) z przeskalowaną maską filtracji górnoprzepustowej. Od lewej: obraz źródłowy, obraz po filtracji.

2. Akceleracja symulacji

Przyspieszenie symulacji możliwe jest dzięki umieszczeniu elementu testowanego w sprzęcie. W tym celu wykorzystywane są specjalnie to tego przystosowane platformy HES (ang. *Hardware Embedded Simulation*) oraz środowisko DVM (ang. *Design Verification Manager*), za pomocą którego obsługiwana jest ta platforma. Projekt zawierający moduł właściwy oraz moduły testujące wykonane w programie Active-HDL, zostają przeniesione do programu DVM. Tutaj moduł właściwy poddawany jest syntezie oraz implementacji i umieszczany jest w dedykowanym układzie. Program DVM komunikuję się z nim, przekazując sygnały wejściowe oraz odbierając sygnały wyjściowe z modułu.
W celu sprawniejszego dostarczania sygnałów stymulujących oraz przedstawienia wyników otrzymanych na wyjściu modułu, stosuje się kosymulację pomiędzy programem Active-HDL a programem MATLAB.
Poszczególne elementy akceleracji symulacji zostały przedstawione w kolejnych podrozdziałach.

2.1. Kosymulacja AHDL – MATLAB

Proces weryfikacji jest bardzo czasochłonnym etapem podczas projektowania układów. Jest on jednak konieczny, ponieważ pozwala wykryć większość niedociągnięć i błędów występujących w projekcie. Języki HDL (ang. *Hardware Description Language*) bardzo dobrze spełniają swoją rolę w opisywaniu układów na poziomie przesyłu danych. Natomiast na wyższym poziomie abstrakcji nie jest to najbardziej optymalny sposób. W celu dostarczania sygnałów wejściowych i prezentowania wyników pracy układu znacznie lepsze są inne typy języków. Rozwiązaniem tego problemu jest możliwa kosymulacja programu Active-HDL z najbardziej popularnym środowiskiem do obliczeń matematycznych, jakim jest program MATLAB. Upraszcza to weryfikację projektowanego sprzętu, zapewniając profesjonalne narzędzia do wizualizacji i analizy, które umożliwiają w łatwy sposób dostarczanie złożonych bodźców, wykonanie analizy wynikowych danych oraz ich wizualizacji w najdogodniejszej formie. Schemat interfejsu pokazany jest na rysunku 25.



Rys. 25. Schemat interfejsu kosymulacji Active-HDL <-> MATLAB [5].

Interfejs kosymulacji pozwala wykonywać komendy programu MATLAB, uruchamiać M-pliki oraz przesyłać dane z jednego środowiska do drugiego. Wystarczy do projektu napisanego w języku VHDL (ang. *VHSIC hardware description language*) dodać bibliotekę, zawierającą funkcje, stworzone do tego zadania (patrz [5]):

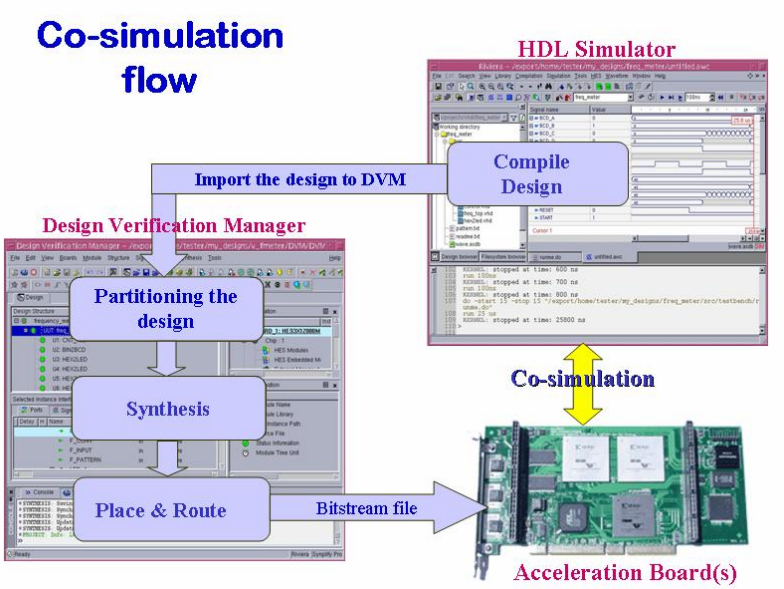
```
library aldec;
use aldec.matlab.all;
```

Kiedy z poziomu języka VHDL zostanie wywołana funkcja związana z programem MATLAB, wówczas następuje otworzenie lokalnego okna komend i bezpośrednie połączenie z tym programem. Transfer danych, jak wspominano wcześniej, jest dwukierunkowy. Poza tym z tego poziomu jest dostępna pełna kontrola nad przestrzenią roboczą w programie MATLAB oraz nad wszystkimi jego komendami i funkcjami. Dzięki temu można tworzyć w prosty sposób rozbudowane i profesjonalne układy weryfikujące.

2.2. Platforma HES i środowisko DVM

Symulacja i weryfikacja układu jest najbardziej czasochłonnym, ale koniecznym etapem w tworzeniu projektu. Jest to duży problem zwłaszcza dla rozbudowanych i skomplikowanych systemów. Ten proces można przyspieszyć, stosując kosymulację sprzętową. Wystarczy

zainstalować do komputera specjalną platformę HES, podłączaną za pomocą interfejsu PCI. Dzięki środowiskowi DVM możliwa jest praca we wcześniejszym symulatorze, bez konieczności uczenia się nowych programów symulacyjnych. Schemat kosymulacji przedstawiony jest na rysunku 26.



Rys. 26. Schemat kosymulacji symulatora z platformą HES za pomocą środowiska DVM [5].

Dzięki zastosowaniu takiego rozwiązania, czas symulacji może skrócić się 10-cio a nawet 100-tu krotnie.

Przy akceleracji sprzętowej, projekt jest umieszczany w sprzęcie. Jednak proces weryfikacji jest sterowany z poziomu symulatora HDL. Połączony jest on z platformą sprzętową przez interfejsy kosymulacji, takie jak PLI (ang. *Program Language Interface*) lub VHPI (ang. *VHDL Programming Language Interface*). Interfejsy te zapewniają synchronizację i komunikację pomiędzy symulatorem i sprzętem. Wszystkie sygnały między symulatorem HDL i projektem w sprzęcie są przekazywane na podstawie zdarzenie po zdarzeniu. W symulatorze HDL wykonywany jest program testowy, który stymuluje moduł w sprzęcie.

Metoda ta łączy w sobie zalety weryfikacji symulacji HDL (wizualizacja sygnałów) i emulacji (szybkość). Pomimo, że projekty HDL są symulowane w sprzęcie, projektanci mogą używać symulatora HDL jako główne narzędzie krokowania, ponieważ wszystkie sygnały wyjściowe projektu są przesyłane do symulatora i pokazane na graficznych przebiegach. Dlatego możliwości krokowania w przypadku akceleracji są podobne, jak w przypadku symulacji pro-

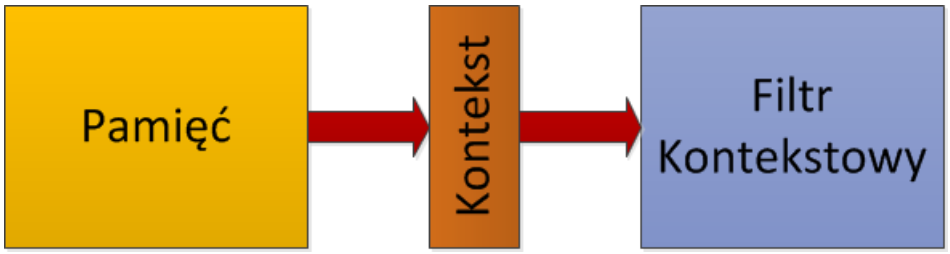
gramowej. Największą korzyścią z przyspieszenia sprzętowego jest 10-100 razy szybsza symulacja, w porównaniu do symulacji modelu po implementacji. Pozwala to symulować większą ilość sygnałów testujących oraz projekty znacznie większe o coraz dłuższym czasie symulacji.

Ponieważ cały projekt znajduje się w sprzęcie, największe ograniczenie prędkości symulacji powoduje symulator HDL. Jednak niektóre elementy układu testującego można zapisać jako syntezowalnych kodu HDL. Kiedy ten kod zostanie także umieszczony w sprzęcie można osiągnąć znaczne przyspieszenie testów.

Oprócz szybkiej symulacji, akceleracja sprzętowa pozwala na tworzenie projektu etapami. Proces projektowania jest szczególnie ważny podczas pracy z już istniejącymi projektami, których funkcjonalność musi zostać rozszerzona oraz które zbudowane są z za pomocą *IP core'ów* z różnych projektów lub od różnych dostawców. W tej metodzie rozwoju projektu, moduły, które zostały już sprawdzone w symulator HDL lub w poprzednich projektach, można załadować do sprzętu i podłączyć bezpośrednio do modułów, które są ciągle w fazie rozwoju. Zapewnia to dużą elastyczność i pozwala projektantom naprawę niespodziewanych problemów z projektem HDL lub jego syntezą już w najwcześniejszych etapach pracy [5].

3. Syntezowalne filtry kontekstowe

Opis teoretyczny przedstawiony w rozdziale 1 pokazuje, jak powinna wyglądać filtracja kontekstowa. Jednak chcąc wykonać te przekształcenia sprzętowo w układzie FPGA (ang. *Field Programmable Gate Array*) należy zoptymalizować działania w taki sposób, aby działały one jak najefektywniej. Syntezowalny kod powinien przeprowadzać filtrację z jak największą precyzją, ale nie kosztem diametralnie dużego i rozbudowanego układu, ponieważ operacje te będą wykonywać się nieefektywnie i z małą prędkością. Spowodowane jest to między innymi tym, że im bardziej skomplikowany układ i im więcej powierzchni zajmuje, tym więcej czasu potrzeba, aby sygnał wejściowy przeszedł przez stworzony układ i ustalił się sygnał wyjściowy. Czas ten wymusza zastosowanie wolniejszego zegara taktującego, czego następstwem jest wolniejsze działanie całego układu.



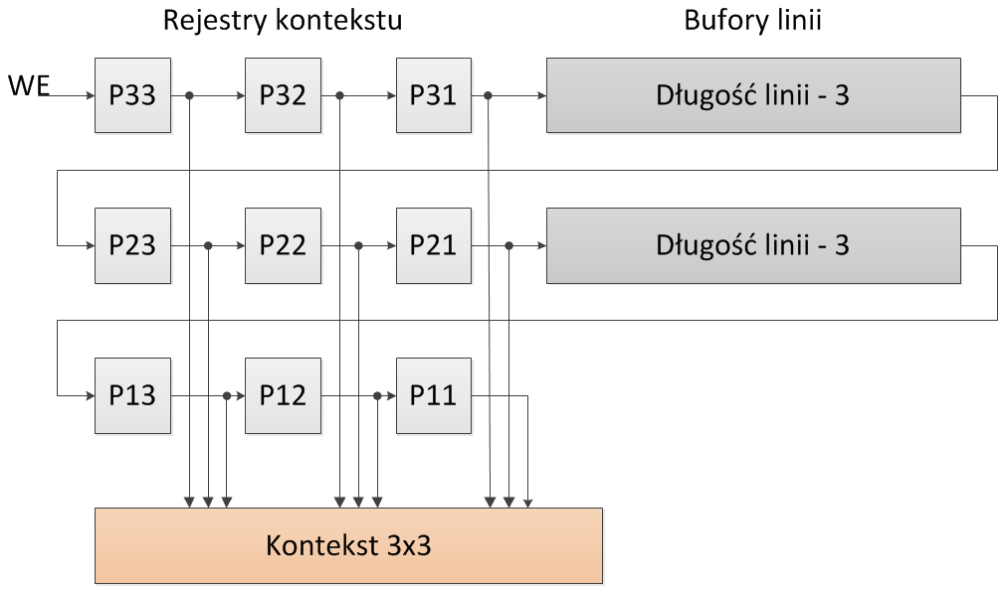
Rys. 27. Schemat blokowy syntezowalnego filtru kontekstowego.

W celu znalezienia pewnego optimum należy zastanowić się, jakie bloki funkcjonalne są potrzebne do wykonania filtracji. Podstawowym elementem w przetwarzaniu kontekstowym jest pamięć. Muszą zostać zapamiętane piksele, na podstawie których zwracany jest kontekst. Drugim elementem jest moduł filtrujący, który wykonuje właściwą filtrację. Schemat tak podzielonego układu przedstawiony jest na rysunku 27. W kolejnych podrozdziałach zostały opisane poszczególne moduły.

3.1. Moduł pamięci

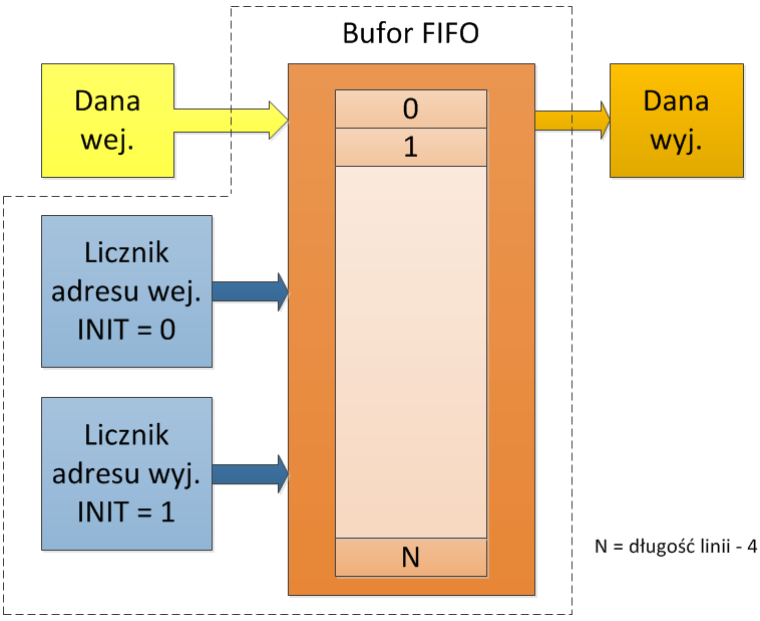
W celu otrzymania kontekstu w postaci okna 3x3 trzeba zapamiętać przynajmniej dwie linie i 3 piksele obrazu źródłowego. Dzięki odpowiedniemu uformowaniu bufora, piksele czytane są szeregowo, a pozostałe przesuwane o jedną pozycję (patrz rys. 28). Za każdym razem

otrzymywany jest prawidłowy kontekst. Jedynie kontekst pikseli, znajdujących się na brzegu, będzie niepoprawny. Szerzej ten problem opisany jest w rozdziale 1.2.



Rys. 28. Schemat modułu pamięci filtru kontekstowego, okno 3x3.

Bufory linii są to bufory typu FIFO syntezowane w postaci Block RAM. Dzięki temu można zaoszczędzić dużą liczbę przerzutników. W celu stworzenia prawidłowo funkcjonującego bufora FIFO należy użyć pamięci Block RAM dwuportowej. Dla każdej linii adresowej trzeba jeszcze stworzyć odpowiedni licznik zwiększający wartość adresu o 1. Licznik odpowiedzialny za adresowanie próbek wyjściowych musi wskazywać adres komórki pamięci o jedną pozycję większą (patrz rys. 29). Adresacja jest wykonywana cyklicznie, tzn. po zapełnieniu bufora, kolejna wartość wejściowa wpisywana jest do pierwszej komórki pamięci.



Rys. 29. Schemat bufora FIFO.

Do stworzenia modułu pamięci wykorzystywane są dwa bufory FIFO, które pracują niezależnie od siebie. Jak to zostało pokazane na rysunku 29, nie wykorzystywane są żadne sygnały informujące o przepełnieniu, czy pustym buforze. Nie są one konieczne, ponieważ działają one cyklicznie i nie jest ważne, w którym aktualnie miejscu znajdują się wskaźniki adresów. Wynika to z tego, że w momencie wpisywania ostatniej próbki odczytywana jest pierwsza zapisana w danym cyklu (patrz rys. 29).

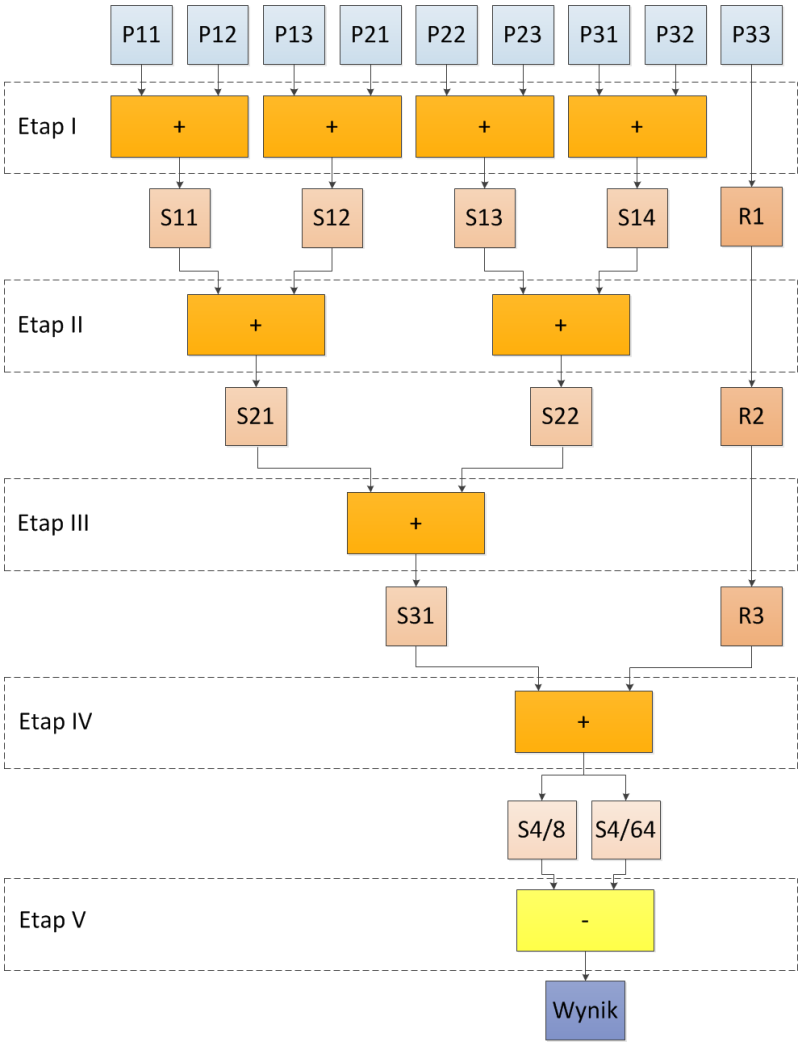
3.2. Moduł filtracji kontekstowej

Moduł wykonujący filtrację odczytuje kontekst zwracany przez moduł pamięci opisany w poprzednim rozdziale. Na podstawie tych próbek wylicza wartość wyjściową. Operacje wykonywane są w sposób potokowy, dlatego też każdy etap musi zostać zapamiętany w rejestrach, co powoduje zwiększenie zużycia przerzutników. W celu przyspieszenia operacji wykorzystywane są tylko operacje dodawania, odejmowania i przesuwania. Przykładowe filtracje i ich struktury zostały opisane w kolejnych podrozdziałach.

3.2.1. Filtr uśredniający

Moduł ten pobiera kontekst z modułu pamięci, na podstawie którego wykonuje kolejne operacje dodawania zgodnie z opisaną maską przetwarzania (patrz rozdział 1.3). Są one wykony-

wane potokowo. W każdym etapie przeprowadzane są operacje w sposób równoległy na maksymalnej liczbie próbek. Schemat działania tego modułu pokazany jest na rysunku 30.



Rys. 30. Schemat działania i zastosowanych elementów w procesie filtracji uśredniającej.

Cały proces filtracji jednej próbki na podstawie pobranego kontekstu przeprowadzony jest w pięciu etapach. Pomiędzy dwoma etapami znajdują się odpowiednie rejestry pamiętające potrzebne dane do następnego etapu. Z powodu nieparzystej liczby próbek, jeden piksel musi być przepisywany i zapamiętywany w nowym rejestrze przez trzy etapy. W czwartym następuje wstępna normalizacja przez podzielenie próbki przez 8. Ta operacja wykonana jest przez odpowiednie logiczne przesunięcie próbki w prawo. Poprawna normalizacja wymaga dzielenia przez 9. Dlatego dodatkowo wykonano analogiczne do poprzedniego dzielenie przez 64 i dodano piąty etap, w którym następuje przybliżenie do oczekiwanej wartości wynikowej. To także nie jest idealne wyliczenie, ponieważ powinno zostać odjęte 1/72. Można by dodać ko-

lejny etap przybliżający do pożądanej wartości, jednak jest to nieefektywne i niepotrzebne, ponieważ otrzymana dokładność jest wystarczająca (patrz rozdział 5.1).

Na rysunku 31 pokazana jest lista wynikowa wygenerowanych elementów po syntezie modułu. W sumie wykorzystanych jest 9 różnego rodzaju układów dodających i odejmujących oraz 113 przerzutników.

```
=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 9
 10-bit adder                                         : 2
 11-bit adder                                         : 1
 12-bit adder                                         : 1
 8-bit subtractor                                     : 1
 9-bit adder                                          : 4
# Registers                                            : 13
 10-bit register                                      : 2
 11-bit register                                      : 1
 6-bit register                                       : 1
 8-bit register                                       : 5
 9-bit register                                       : 4

=========================================================================
```

Rys. 31. Lista wygenerowanych elementów układu po syntezie filtru uśredniającego.

Podsumowanie parametrów czasowych oraz maksymalna możliwa częstotliwość w układzie VIRTEX2 pokazana jest na rysunku 32. Częstotliwość taktowania nie może być większa od podanej, w przeciwnym wypadku moduł będzie działał niepoprawnie, gdyż nie będą spełnione warunki czasowe ustalenia i utrzymania.

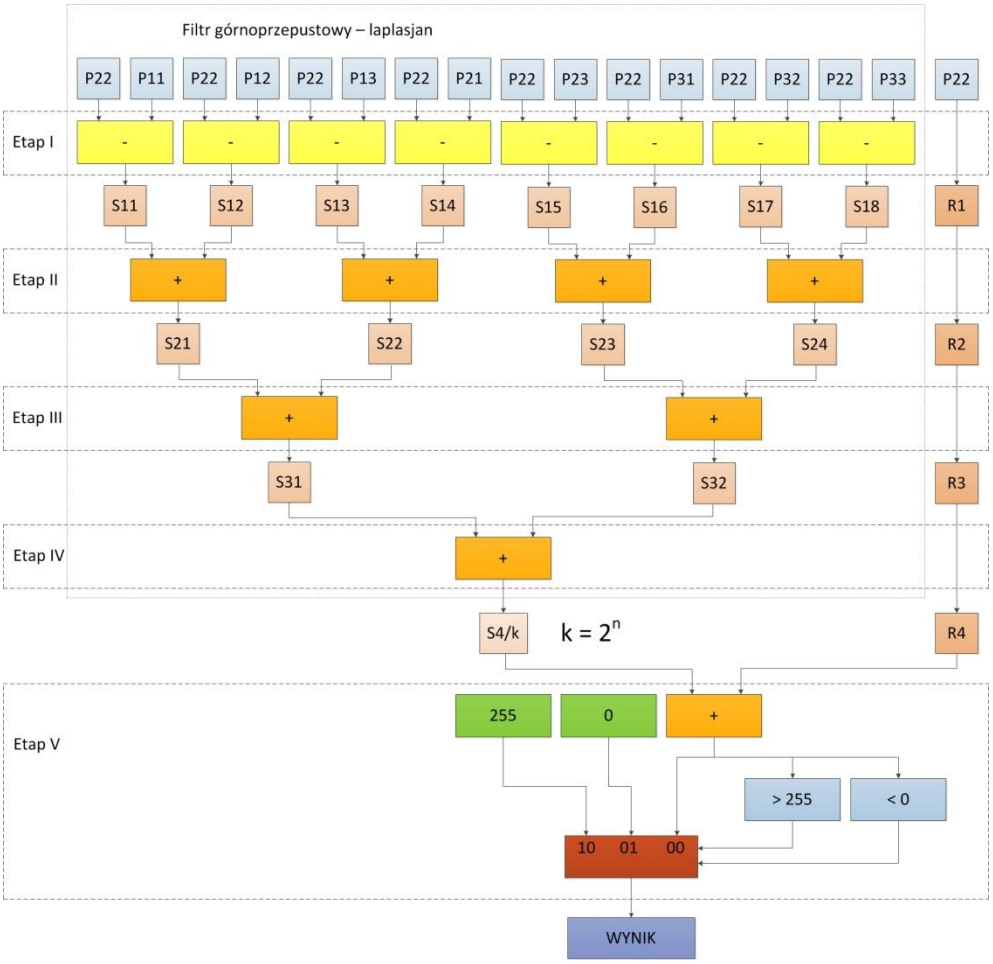
```
Timing Summary:
---------------
Speed Grade: -6

   Minimum period: 3.160ns (Maximum Frequency: 316.506MHz)
   Minimum input arrival time before clock: 3.238ns
   Maximum output required time after clock: 4.575ns
   Maximum combinational path delay: No path found
```

Rys. 32. Podsumowanie parametrów czasowych dla układu VIRTEX2 po syntezie filtru uśredniającego.

3.2.2. Filtr wyostrzający – *high boost*

Moduł ten, podobnie jak opisany w poprzednim podrozdziale, wykonuje operacje potokowo. Jest on jednak bardziej skomplikowany, ponieważ macierz tej filtracji zawiera zarówno dodatnie, jak i ujemne współczynniki (patrz rozdział 1.4.3. - przykład). W związku z tym należy przeprowadzać operacje na wartościach ze znakiem. Rejestry muszą być o jeden bit większe niż w przypadku filtracji uśredniającej, gdzie operowano na wartościach bez znaku. Schemat syntezowalnej filtracji wyostrzającej przedstawiono na rysunku 33.



Rys. 33. Schemat działania i zastosowanych elementów w procesie filtracji wyostrzającej (ang. *high boost).*

Ta filtracja także wykonuje się w pięciu etapach. Jednak potrzebuje dwa razy więcej zasobów sprzętowych. Lista wykorzystanych elementów po syntezie pokazana jest na rysunku 34. W sumie wykorzystanych jest 16 różnego rodzaju układów dodających i odejmujących, 186 przerzutników oraz dwa komparatory 13-bitowe.

```
=========================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 16
 10-bit adder                                          : 4
 11-bit adder                                          : 2
 12-bit adder                                          : 1
 13-bit adder                                          : 1
 9-bit subtractor                                      : 8
# Registers                                            : 20
 10-bit register                                       : 4
 11-bit register                                       : 2
 12-bit register                                       : 1
 8-bit register                                        : 5
 9-bit register                                        : 8
# Comparators                                          : 2
 13-bit comparator greater                             : 1
 13-bit comparator less                                : 1

=========================================================
```

Rys. 34. Lista wygenerowanych elementów układu po syntezie filtru wyostrzającego.

Na rysunku 33 wyodrębniony jest fragment odpowiedzialny za filtrację górnoprzepustową. Za pomocą stałej *k* skalowany jest wpływ tej wartości na piksel wynikowy. Wykorzystuje się go w celu zminimalizowania przeskalowań. W tej filtracji pojawia się bowiem problem związany z prawidłowym unormowaniem wyników. Po operacji dodawania wartości piksela i wartości otrzymanej po filtracji górnoprzepustowej, wartość wynikowa obcinana jest do wartości w formacie 8-bitowej bez znaku. Wykonane jest to za pomocą dwóch komparatorów i multipleksera, który jest sterowany ich sygnałami wyjściowymi. Oznacza to, że jeśli wartość piksela wynikowego jest mniejsza od zera, to wówczas na wyjściu jest wartość 0, natomiast jeśli jest ona większa od wartości maksymalnej, czyli 255, wtedy na wyjściu wpisywana jest ta wartość maksymalna. Pozostałe wartości pikseli przepisywane są bez zmian. Dzięki tak zastosowanemu formatowaniu próbki wyjściowej, zachowany jest naturalny widok obrazu źródłowego.

```
Timing Summary:
---------------
Speed Grade: -6

   Minimum period: 4.797ns (Maximum Frequency: 208.464MHz)
   Minimum input arrival time before clock: 3.509ns
   Maximum output required time after clock: 4.575ns
   Maximum combinational path delay: No path found

Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================
Timing constraint: Default period analysis for Clock 'CLK'
  Clock period: 4.797ns (frequency: 208.464MHz)
  Total number of paths / destination ports: 2404 / 90
---------------------------------------------------------
Delay:               4.797ns (Levels of Logic = 12)
  Source:            sum_reg_31_0 (FF)
  Destination:       final_sample_1 (FF)
  Source Clock:      CLK rising
  Destination Clock: CLK rising
```

Rys. 35. Podsumowanie parametrów czasowych dla układu VIRTEX2 po syntezie filtru wyostrzającego.

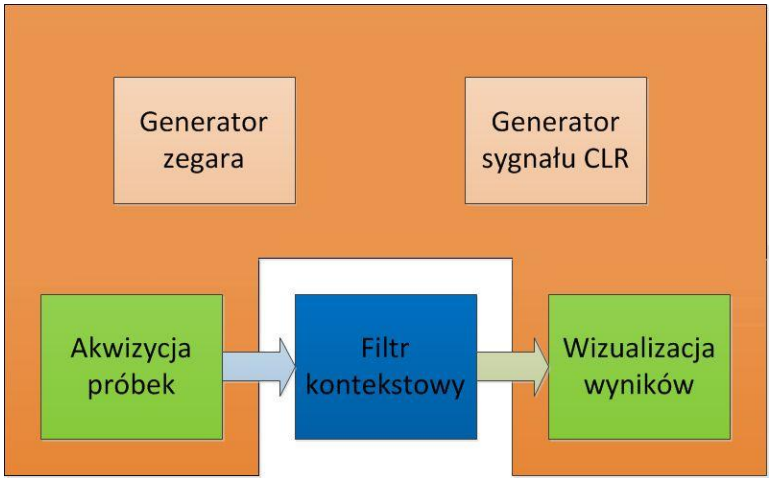
Z konstrukcji widocznej w etapie V (patrz rys. 33) można wnioskować, że układ będzie działał niepoprawnie, ponieważ sygnał wyjściowy z komparatora ustali się minimalnie później niż sygnał wyjściowy po wykonaniu modułu. Jednak sygnał wynikowy przepisywany jest na wyjście wraz ze zboczem narastającym zegara. Dlatego też dobierając odpowiedni okres sygnału taktującego wszystko będzie działało poprawnie, ponieważ w czasie pomiędzy kolejnymi zboczami wyzwalającymi wszystkie sygnały będą ustalone.

Program do wykonywania syntezy informuje jaka jest dopuszczalna maksymalna częstotliwość taktowania, aby układ działał w prawidłowo. Na rysunku 35 widać, że zegar ustalany jest na podstawie przejścia sygnału od *S4/k* (*sum_reg_31_0*) do wartości wynikowej (*final_sample_1).*

Porównując otrzymaną wartość częstotliwości maksymalnej z częstotliwością dla filtracji uśredniającej opisanej w poprzednim podrozdziale, można stwierdzić, że filtracja wyostrzająca w danym układzie wykonuje się wykonuję się o ponad 100MHz wolniej (patrz rys. 35 i rys. 32).

4. Weryfikacja

W celu sprawdzenia poprawności wykonywanych operacji modułów filtrujących, konieczne jest stworzenie układu weryfikującego. Do tego napisany został tzw. *testbench*, który wykonuje wszystkie związane z weryfikacją operacje. Zawiera on dwa moduły. Jeden z nich pobiera próbki ze źródła danych, natomiast drugi przedstawia w najdogodniejszy sposób otrzymane wartości po przefiltrowaniu. Generuje także inne potrzebne sygnały wejściowe. Jego schemat pokazany jest na rysunku 36. Poszczególne moduły składowe opisane są w kolejnych podrozdziałach.



Rys. 36. Schemat modułu testującego.

Moduł generatora zegara służy do tworzenia sygnału taktującego, potrzebnego do przeprowadzenia symulacji. Bez niego nie wykonałby się żaden test. Natomiast generator sygnału CLR, potrzebny jest dla modułów należących do układu testującego, aby ustawiły się na stan początkowy oraz dla układu pamięci modułu filtracji kontekstowej, w celu ustawienia prawidłowej wartości początkowych adresów w buforach FIFO.

4.1. Akwizycja próbek

Chcąc przeprowadzić jakiekolwiek testy, należy dostarczyć sygnały wejściowe do układu testowanego. Na podstawie tych próbek oraz rezultatów wyjściowych można dopiero zweryfikować poprawność działania tego układu.

W celu zrealizowania tego zadania dla filtru kontekstowego najlepszym i jedynym właściwym jest podanie na jego wejście rzeczywiste wartości pikseli danego obrazu testowego, który może być w postaci obrazu w odcieniach szarości, w formacie RGB, fragmentu filmu, czy też obrazu z kamery.
Dzięki kosymulacji opisanej w rozdziale 3.1 można wykonać to w bardzo prosty sposób. Po dodaniu potrzebnej biblioteki mamy dostępny pełny asortyment dostarczany przez możliwości programu MATLAB.

Podczas operacji na obrazach bardzo ważny jest typ reprezentacji danej. Odczytując tablicę wartości obrazu w programie MATLAB zwracana jest zmienna kilkuwymiarowa w formacie 8-bitowym bez znaku (uint8). W celu zachowania poprawności obliczeń w programie Active-HDL należy pobrać dane w takiej samej formie. Można to ustawić za pomocą następującej funkcji:

ml_setup(desktop : BOOLEAN; point : INTEGER; class : mxClassID; cast : mxCastID); (12)

gdzie:

desktop – ustawia czy przy odwołaniu się do programu MATLAB ma zostać wyświetlone okno programu (*true*), czy okno komend (*false*); domyślnie *false*;
point – definiuje pozycję przecinka dziesiętnego; domyślnie 0;
class – definiuje domyślną klasę obiektów tworzonych w środowisku MATLAB [5]; domyślnie mxDOUBLE;
cast – definiuje interpretację przesyłanych wektorów z programu Active-HDL, czy są ze znakiem (mxSIGNED), czy bez znaku (mxUNSIGNED); domyślnie ustawiona jest wartość mxDEFAULT, gdzie zależy to od typu argumentu, jednak w przypadku języka VHDL zawsze równoważne jest to z mxSIGNED, co należy mieć na uwadze;

W projekcie użyta jest następująca forma:

ml_setup(false, 0, mxUINT8, mxUNSIGNED); (13)

W celu odczytania zmiennej z przestrzeni roboczej programu MATLAB, należy wywołać komendę, która to wykona. Możliwe jest to dzięki funkcji:

```
eval_string("ml_cmd");                                    (14)
```

gdzie: *ml_cmd* – komenda lub wyrażenie, które powinno być wpisane w oknie komend;

W przypadku akwizycji obrazu za pomocą funkcji (14) wywoływany jest cały M-plik, który wykonuje operacje konieczne do prawidłowego odczytania pliku źródłowego oraz wyświetlenie jej rezultatu. Wykorzystywane są dwie formy pobierania pliku źródłowego:

1. Odczytanie tablicy pikseli z pliku.

```
img_src = imread(filename, fmt);                          (15)
```

   gdzie:

   filename – nazwa pliku w formacie *string*;
   fmt – rozszerzenie pliku w formacie *string*;

   W większości przypadków zwracany format to uint8.

2. Wykonanie inicjalizacji kamery i pobieranie z niej obrazu za pomocą sterowników systemowych i odpowiedniego adaptera dostarczonego przez program MATLAB [6].

```
vid = videoinput(adaptorname, deviceID, format)           (16)
```

   gdzie:

   vid – struktura zwracana przez funkcję; obiekt komunikacji z urządzeniem;
   adaptorname – nazwa adaptera dostępnego w programie MATLAB;
   deviceID – numer urządzenia obsługiwanego przez podany adapter;
   format – format zwracany przez urządzenie (patrz [6]);

   Następnie w celu pobrania ramki obrazu wywoływana jest komenda:

```
frame = getsnapshot(vid);                                 (17)
```

Do akwizycji próbek z kamery można użyć wbudowanego narzędzia *Image Acquisition Tool*, za pomocą którego w łatwy sposób graficznie można pobrać obraz i zapisać go do pamięci roboczej lub na dysk (patrz [6]).

Kiedy zmienna znajduje się już w przestrzeni roboczej programu MATLAB, wówczas w kodzie VHDL wykonywana jest funkcja, która zwraca identyfikator tej zmiennej. Za jego pomocą można się do niej odwoływać:

```
array_id := ml2hdl(var_name);
```
(18)

gdzie: *var_name* – nazwa zmiennej w formacie *string*;

Następnie pobierane są kolejne elementy tablicy przy pomocy funkcji:

```
get_item(hdl_var, mxCastID, array_id, dim_loc);
```
(19)

gdzie:

hdl_var – nazwa zmiennej w programie Active-HDL;

mxCastID - definiuje interpretację wektorów w programie Active-HDL (porównaj wzór (12));

array_id – identyfikator tablicy;

dim_loc – współrzędne elementu tablicy w programie MATLAB w formacie *TDims* [5];

Wykorzystywany jest ten wariant funkcji, ponieważ argumentem *hdl_var* jest wektor typu *std_logic_vector*. W celu poprawnego zdefiniowania typu zmiennej zapisanej w programie Active-HDL należy podać odpowiedni argument *mxCastID* (tu: *mxUNSIGNED*). Jest to konieczne pomimo zastosowania wcześniej funkcji (12).

Wymiary odczytywanej zmiennej muszą być wcześniej znane, ponieważ wymagane jest odpowiednie ustawienie stałych w module filtrującym. Jest to konieczne do prawidłowego działania filtru kontekstowego. Dlatego też niepotrzebne jest odczytywanie ich za pomocą funkcji z biblioteki.

Odczytana wartość piksela wraz ze zboczem narastającym zegara podawana jest na wyjście układu akwizycji. Dzięki sygnałowi *enable* można manipulować trybem pobieranych i wystawianych na wyjściu próbek.

4.2. Wizualizacja wyników

Dzięki kosymulacji z programem MATLAB (patrz rozdział 2.1) również przedstawienie wyników jest proste i szybkie. Korzystając z odpowiednich funkcji można wyświetlić obraz po filtracji w najbardziej dogodnej formie. W celu przesłania zmiennej (tablicy) jest wykonana funkcja, która tworzy zmienna w programie MATLAB i zwraca do niej identyfikator:

```
array_id := create_array (name, ndims, dim_constr);
```
(20)

gdzie:

name – nazwa zmiennej tworzonej w programie MATLAB w formacie *string*;

ndims – liczba wymiarów zmiennej;

dim_constr – wymiary zmiennej w formacie *TDims*;

Następnie wpisywane są kolejne wartości do tablicy za pomocą funkcji:

```
put_item(hdl_var, mxCastID, array_id, dim_loc);
```
(21)

Zastosowanie tego wariantu funkcji jest analogiczny jak opisany w przypadku funkcji (19).

Po wpisaniu wszystkich elementów tablicy należy przesłać tablicę do środowiska MATLAB za pomocą funkcji:

```
hdl2ml(array_id);
```
(22)

Niepotrzebną zmienna usuwana jest z pamięci funkcją:

```
destroy_array (array_id);
```
(23)

Teraz w celu wyświetlenia otrzymanego wyniku wykonywana jest funkcja programu MATLAB:

imshow(name) (24)

Wywołana jest za pomocą funkcji (14).

5. Weryfikacja filtracji sprzętowej

5.1. Porównanie otrzymanych wyników

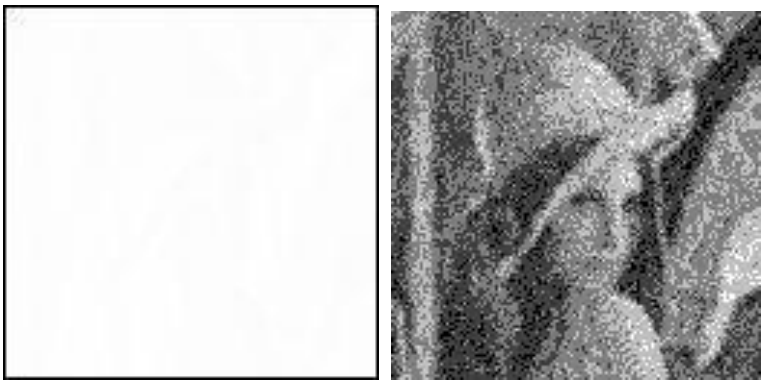
W przypadku filtracji uśredniającej dokładność wykonanego przekształcenia można obliczyć w następujący sposób: po zsumowaniu wszystkich dziewięciu pikseli z kontekstu należy podzielić przez 9, w celu normalizacji; natomiast w wykonanym module jest wykonane dzielenie przez 8 i dodatkowo odejmowana jest 1/64 część wartości sumy, na podstawie czego można wyliczyć procentową dokładność:

$$\frac{\frac{1}{8}-\frac{1}{64}}{\frac{1}{9}}\cdot 100\% = \frac{\frac{7}{64}}{\frac{1}{9}}\cdot 100\% = \frac{63}{64}\cdot 100\% = 98{,}4\% \qquad (25)$$

Z powyższego równania wynika, że błąd przetwarzania jest na poziomie 1,6%. Przeprowadzone rachunki potwierdzają wyliczenia, ponieważ różnica pomiędzy obrazem przefiltrowanym za pomocą programu MATLAB na zmiennych typu *double* a obrazem wynikowym filtracji wykonanej w symulacji filtru syntezowalnego wynosiła maksymalnie 4, co odpowiada 1,6% z maksymalnej wartości piksela (255). Porównanie otrzymanych obrazów tej filtracji za pomocą programu MATLAB i po przejściu przez filtr syntezowalny przedstawione jest na rysunku 37.

Rys. 37. Porównanie filtracji uśredniającej. Od lewej: plik źródłowy, po filtracji w programie MATLAB, po filtracji sprzętowej.

Obraz po filtracji sprzętowej jest mniejszy, ponieważ odrzuca źle wyliczone wartości brzegowe. Różnicę obrazów po filtracji przedstawiono na rysunku 38.



Rys. 38. Różnica obrazów po filtracji w programie MATLAB i po filtracji sprzętowej. Od lewej: obraz w skali 0-255 (dodano obwiednie w celu wyznaczenia granic obrazu), obraz przeskalowany.

Patrząc na drugi obraz na rysunku 38 można rozróżnić tylko 4 odcienie szarości, które odpowiadają przedziałowi błędu 0-4.

Natomiast filtracja wyostrzająca (ang. *high boost*) przeprowadzana jest identycznie jak za pomocą programu MATLAB. Różnice jedynie mogą wystąpić, gdy stała k będzie inna niż potęga dwójki. Dla $k = 1$ wykonano filtrację w programie MATLAB i za pomocą filtru syntezowalnego. Otrzymany wynik przedstawiono na rysunku 39.

Rys. 39. Wynik filtracji wyostrzającej (ang. *high boost*). Od lewej: obraz źródłowy, po filtracji w programie MATLAB, po filtracji sprzętowej.

5.2. Porównanie czasu przetwarzania

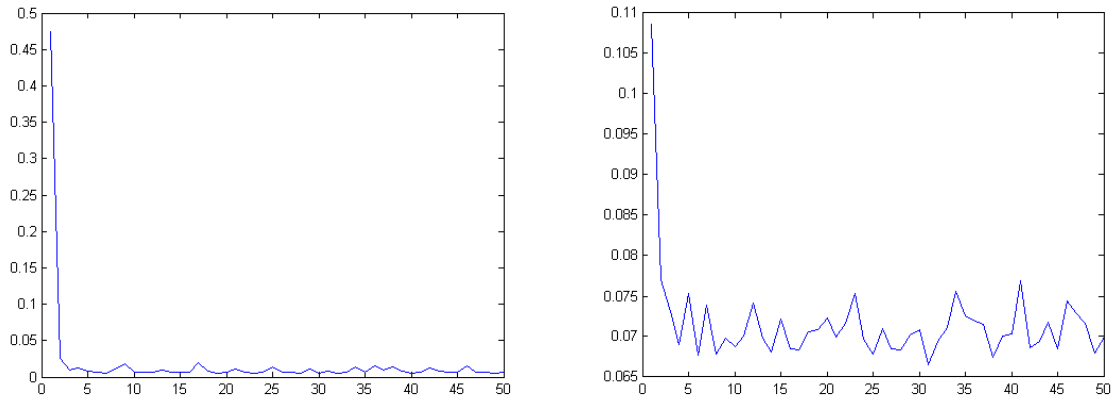
5.2.1. Porównanie czasu realizacji filtracji programowej i sprzętowej

W związku z wykonywaniem operacji w sposób równoległy, filtracja sprzętowa przeprowadzana jest kilka razy szybciej. Potwierdzają to wartości przedstawione w tabeli 1.
Podczas testów, wykonanych za pomocą symulacji, stosowany był zegar 100MHz. Komputer przeprowadzający wszystkie operacje posiadał procesor Pentium 4 2.40GHz i 1GB RAM.

Tab. 1. Porównanie czasu obliczeń programowych i sprzętowych.

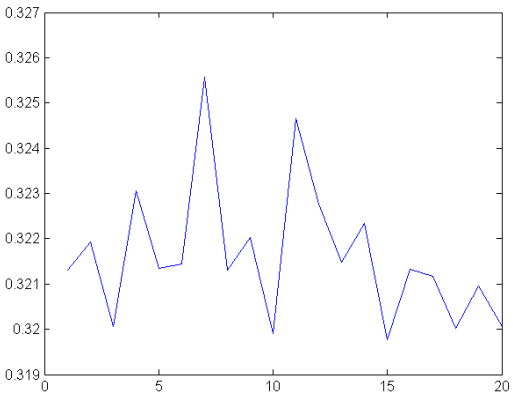
Obraz źródłowy	**Czas obliczeń programowych [ms]**	**Czas obliczeń sprzętowych [ms]**
obraz „Lena” 512x512 RGB	44,9	7,9
plik AVI 240x180 RGB, 100 ramek	321,1	129,6

W celu wyznaczenia przybliżonego czasu filtracji obrazu „Lena”, uwzględniającego zmieniające się stany komputera, wykonującego przetwarzanie, wykonano dwa rodzaje obliczeń. Pierwszy z nich w pętli *for* filtrował 50 razy ten sam plik źródłowy. Natomiast drugi stworzył tablicę zawierającą 50 elementów. Każdy z nich był identyczny i reprezentował macierz obrazu „Lena”. W obu przypadkach czas odmierzano za pomocą funkcji programu MATLAB *tic* i *toc*, zapisujące w każdej iteracji wartość czasu przetwarzania do zmiennej. Wykresy zmierzonych czasów przedstawiono na rysunku 40. Na podstawie otrzymanych wartości wyliczono średnią wartość czasu filtracji, którą uznano za bliską rzeczywistej.



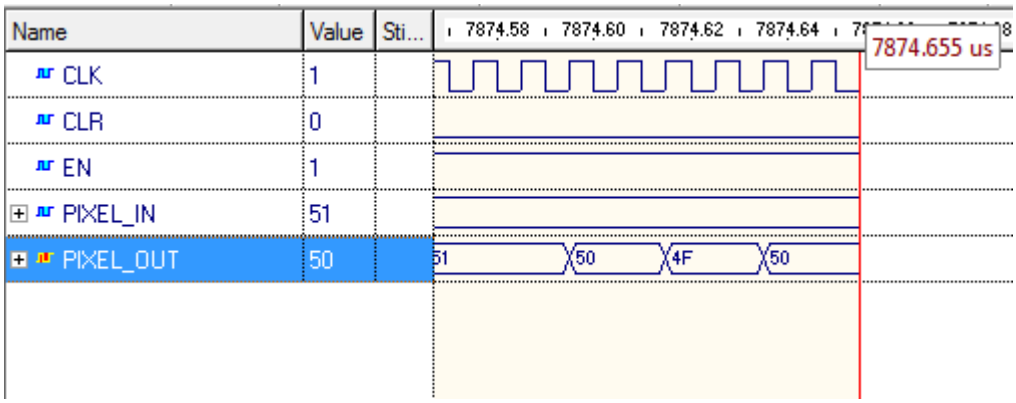
Rys. 40. Przebieg kolejnych wartości czasów przetwarzania obrazu „Lena” 512x512 RGB. Od lewej: dla filtracji wykonanej na jednym obrazie, czasy wykonane na tablicy tych samych obrazów.

Z przebiegów pokazanych na rysunku 40 wynika, że pierwsza filtracja wykonuje się znacznie wolniej. Jest to spowodowane tym, że kolejne procesy przetwarzania wykorzystują już zapamiętane próbki w pamięci podręcznej (ang. *cache*). Jednak filtracja w rzeczywistych warunkach nie będzie wykonywana na tej samej macierzy pikseli, lecz na ciągle zmieniających się obrazach. W tym celu wykonano również porównanie czasów przetwarzania pliku AVI (patrz tab. 1). W tym przypadku wykonano kolejno 20 filtracji tego samego pliku AVI i obliczono średnią z otrzymanych czasów. Koleje czasy są tym razem zbliżone do siebie co można zauważyć na rysunku 41.



Rys. 42. Przebieg kolejnych wartości czasów przetwarzania pliku AVI.

Czasy filtracji sprzętowej otrzymano na podstawie czasu symulacji, odczytanej zgodnie z rysunkiem 42.



Rys. 42. Odczytanie czasu filtracji na podstawie przebiegu graficznego, wygenerowanego podczas symulacji behawioralnej w programie Active-HDL.

Otrzymane wyniki dowodzą, że opłaca się implementować kontekstową filtracje obrazu w sprzęcie. Jest ona wystarczająca dokładna i o wiele szybsza niż filtracja wykonywana za pomocą zwykłego komputera, który musi wykonywać dodatkowo inne operacje. Należy zauważyć, że w symulacji został zastosowany stosunkowo wolny sygnał taktujący wynoszący 100MHz.

5.2.2. Porównanie czasu symulacji

Drugim zagadnieniem rozważanym i badanym w tym rozdziale jest porównanie prędkości symulacji modelu behawioralnego, po syntezie, po implementacji oraz kosymulacji sprzętowej. Otrzymane wartości przedstawiono w tabeli 2.

Tab. 2. Porównanie czasu symulacji.

Obraz źródłowy	**Czas symulacji modelu behawioralnego [s]**	**Czas symulacji modelu po syntezie [s]**	**Czas symulacji modelu po implementacji [s]**	**Czas kosymulacji w środowisku HES [s]**
obraz „Lena” 512x512 RGB	50,61	173,94	2185,78	36,52
plik AVI 240x180 RGB 100 ramek	737,69	2227,16	25534,35	549,03

Otrzymane wartości, przedstawione w tabeli 2, potwierdzają słuszność stosowania kosymulacji sprzętowej. Dzięki niej symulacja układu przeprowadzana już po syntezie i implementacji jest kilkanaście razy szybsza. Wykonuję się również szybciej od symulacji modelu behawioralnego. Dodatkowym atutem tej metody jest to, że układ weryfikowany jest już po implementacji. Badane jest zatem zachowanie modułu po umieszczeniu go w rzeczywistym układzie.

Podsumowanie

W ramach projektu zaprojektowane zostały filtry, które wykonują się w sposób szybki i z bardzo dobrą dokładnością. Można by zastanowić się nad stworzeniem większej liczby filtrów, jednak te dwa, które zostały wykonane w projekcie w wystarczającym stopniu obejmują zagadnienia związane z trudnością napisania syntezowalnego kodu.

W czasie projektu zapoznano się z możliwościami układów FPGA oraz wykorzystano je do stworzenia funkcjonalnych modułów. Dzięki temu, że na uczelni dostępne były platformy HES i potrzebny do kosymulacji z nimi program DVM, możliwe było uruchomienie projektu w sprzęcie i przeprowadzenie profesjonalnej symulacji.

W ten sposób wykonano w pełni zadania zawarte w temacie projektu.

Bibliografia

[1] Tadeusiewicz R., Korohoda P., *Komputerowa analiza i przetwarzanie obrazów*, Kraków, Fundacja Postępu Telekomunikacji, 1997.

[2] Forczmański P., *Filtracja Splotowa*, Instytut Grafiki Komputerowej i Systemów Multimedialnych, Wydział Informatyki Politechniki Szczecińskiej, 2006, http://www.informa.tanihosting.com/sem5/PO/W/wyklad_3.pdf (05-12-2010)

[3] Obrazy źródłowe: http://www.eecs.qmul.ac.uk/~phao/CIP/Images/ (04-01-2011)

[4] High-boost filtering, http://fourier.eng.hmc.edu/e161/lectures/gradient/node2.html (04-01-2011)

[5] http://www.aldec.com/ (04-01-2011)

[6] http://www.mathworks.com/ (04-01-2011)

[7] Filtrowanie obrazów, http://www.algorytm.org/przetwarzanie-obrazow/filtrowanie-obrazow.html (04-01-2011)

[8] http://www.xilinx.com/ (04-01-2011)

[9] Zwoliński M., *Projektowanie układów cyfrowych z wykorzystaniem języka VHDL*, Warszawa, Wydawnictwo Komunikacji i Łączności, 2007.

Wykaz załączników

1. Tutorial – *Kontekstowa filtracja obrazu*.
2. Płyta CD zawierająca następujące elementy:
    - dokumentację projektu w formacie PDF (2010_Filtr_obrazu_Projekt_dyplomowy_inżynierski.pdf),
    - tutorial w formacie PDF (Kontekstowa_filtracja_obrazu[tutorial].pdf),
    - pliki wykorzystywane w tutorialu w formie archiwum RAR (filtr_obrazu.rar),
    - folder zawierający wykonany projekt w programie Active-HDL.
3. Zawartość płyty CD.