ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – СОФИЯ,
ФИЛИАЛ – ПЛОВДИВ

Факултет „Електроника и Автоматика”

Катедра ” Компютърни системи и технологии”

Маг. инж. Иванчо Тодоров Кънев

FPGA – базирани линейни пространствени филтри

А В Т О Р Е Ф Е Р А Т

на дисертация за придобиване на образователна и научна степен

"ДОКТОР"

Област: 5. Технически науки

Професионално направление: 5.3. Комуникационна и компютърна техника

Научна специалност: компютърни системи, комплекси и мрежи

Научен ръководител: доц. д-р Петя Павлова

ПЛОВДИВ 2017 г.

Дисертационният труд е обсъден и насочен за защита от Катедрения съвет на катедра „КСТ“ към Факултет ФЕА на ТУ-София на редовно заседание, проведено на 22.02.2017 г.

Публичната защита на дисертационния труд ще се състои на 05.07.2017г. от 14.00 часа в зала 4425 на Технически университет – София, Филиал-Пловдив на открито заседание на научното жури, определено със заповед № ОЖ-102 / 15.03.2017 г. на Ректора на ТУ-София в състав:

1. Доц. д-р Николай Каканаков – председател
2. Доц. д-р Петя Павлова – научен секретар
3. Проф. д-р Ангел Смрикаров
4. Доц. д-р Надежда Рускова
5. Доц. д-р Найден Василев

Рецензенти:

1. Проф. д-р Ангел Смрикаров
2. Доц. д-р Николай Каканаков

Материалите по защитата са на разположение на интересуващите се в канцеларията на Факултет ФЕА на ТУ-София, Филиал Пловдив, блок №4, кабинет № 4242.

Дисертантът е докторант на самостоятелна подготовка към катедра „Компютърни системи и технологии “ на факултет електроника и автоматика Изследванията по дисертационната разработка са направени от автора.

Автор: маг. инж. Иванчо Тодоров Кънев

Заглавие: **FPGA – базирани линейни пространствени филтри**

Тираж: 30 броя
Отпечатано в издателско звено на Технически университет – София,
Филиал Пловдив.

I. ОБЩА ХАРАКТЕРИСТИКА НА ДИСЕРТАЦИОННИЯ ТРУД

Актуалност на проблема

Концепцията за филтриране има своите корени в използването на трансформацията на Фурие за обработка на сигнали в честотната област. При цифровото филтриране на едно изображение се използват операции, които се прилагат директно върху пикселите на изображението. В този смисъл се използва терминът пространствено филтриране (spatial filtering) за да се разграничат тези процеси от филтрирането в честотната област.

Линейните пространствени филтри (LSF) се използват за размиване (blurring) и за редуциране на шума (noise reduction) в дадено изображение. Размиването се използва при предварителна обработка (*preprocessing*) на изображения, като например отстраняване на малки детайли от изображението или за преодоляване на малки пропуски в линии или криви. Редуциране на шума се използва за подобряване на качеството на изображенията.

С развитието на компютърните технологии се усъвършенстват както алгоритмите за LSF, така и сферите на тяхното приложение: медицина; астрономия; компютърно зрение; роботика, военно дело и др. Към настоящия момент са особенно актуални методите за хардуерното имплементиране на LSF, в FPGA и ASIC базирани платформи. Този подход, в сравнение с използването на конвенционални процесори с общо предназначение или DSP позволява значително да се ускори времето за изчисляване на алгоритмите.

Цел на дисертационния труд, основни задачи и методи за изследване

Цел на дисертационният труд е изследване на възможностите за редуциране на броя на операциите при изчисляване на LSF с FPGA-базирани платформи, като се използват конвейрни и паралелни алгоритми и базираните на тяхна основа хардуерни решения.

За да се постигне поставената цел трябва да се решат следните задачи:

1. Да се извърши сравнителен анализ на софтуерните и хардуерните методи за филтриране на цифрови изображения с използването на FPGA-базирани платформи.
2. Да се предложат алгоритми за генериране на тестови Гаусови ядра с целочислени коефициенти.
3. Да се създадат инструментални средства за проектиране на ROM-базирани крайни автомати.
4. Да се предложат алгоритми и конвейрни схеми на линейни пространствени филтри и се изследват алтернативни методи за нормализиране на филтрирания пиксел.
5. Да се предложат хибридни схеми на LSF базирани на конвейрни и паралелни методи с използване на частични суми.

Научна новост

Разработени са нови алгоритми и хардуерни решения за конвейрно и паралелно изчисляване на линейни пространствени филтри.

Практическа приложимост

Получените резултати са приложими в области като прагово детектиране при обработване на изображения или детектиране на контури.

Апробация

Резултатите от дисертационния труд са докладвани и обсъждани на:
Научни конференции с международно участие "Техсис 15, 16 "
Семинари на катедра "Компютърни системи и технологии" към ТУ-София, филиал Пловдив

Публикации

Основни постижения и резултати от дисертационния труд са публикувани в шест научни статии на английски език. Четири от тях са самостоятелни, а две са в съавторство с научния ръководител.

Четири от статиите са публикувани в списание Journal of the Technical University – Sofia Plovdiv branch, Bulgaria "Fundamental Sciences and Applications"

Една статия е публикувана в конференция в чужбина „Proceedings of the 16th International Conference on Computer Systems and Technologies, CompSysTech'15, Dublin, Ireland, ACM International Conference Proceeding"

Една статия е публикувана в международно периодично списание с импакт фактор „IOSR Journal of VLSI and Signal Processing"

Структура и обем на дисертационния труд

Дисертационният труд е в обем от 151 страници, като включва увод, 4 глави за решаване на формулираните основни задачи, списък на основните приноси, списък на публикациите по дисертацията, използвана литература и 25 страници приложения. Цитирани са общо 104 литературни източници, като всички са на латиница, а останалите са интернет адреси. Работата включва общо 66 фигури и 35 таблици. Номерата на фигурите и таблиците в автореферата съответстват на тези в дисертационния труд.

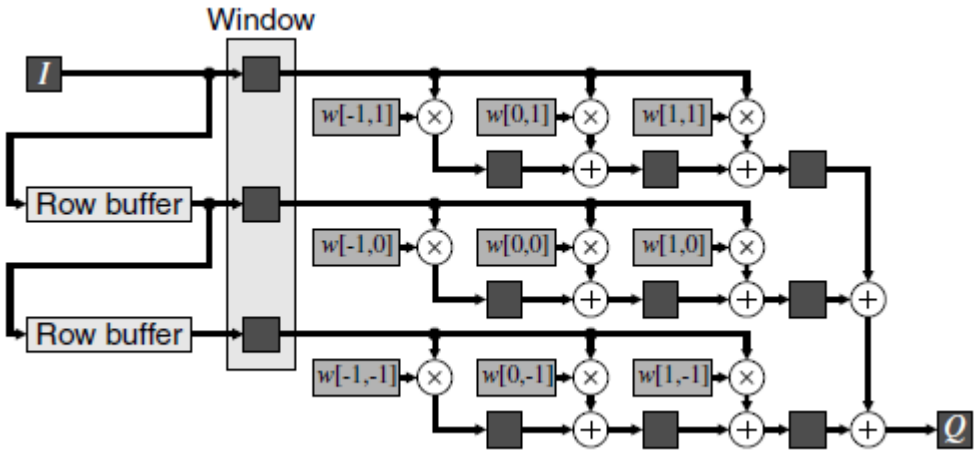
II. СЪДЪРЖАНИЕ НА ДИСЕРТАЦИОННИЯ ТРУД

ГЛАВА 1. ОБЗОР ПО ПРОБЛЕМА

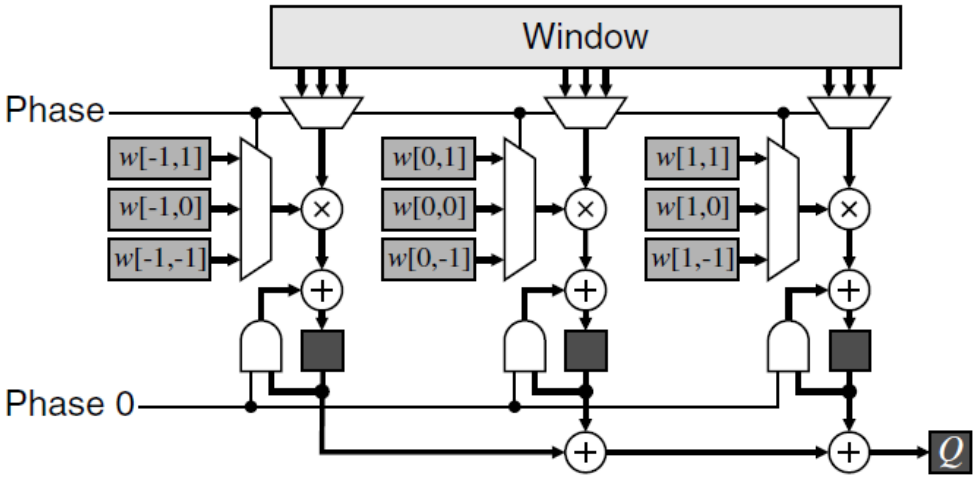
Филтрирането на изображения с оператора за съседство, най- общо може да бъде обяснено по следния начин: от изображение с размер $M \times N$, в определен прозорец (*window*) се заграждат пиксели с размер $m \times n:\ m,n \geq 3$ и m,n са нечетни числа ; на всеки прозорец се съпоставя маска на филтъра (*filter mask*) (или още ядро (*kernel*)), със същия размер; върху пикселите на изображението и елементите на ядрото които имат еднакви

координати се извършват определени операции, вследствие на което се получава ново качество на изображението.

Основен проблем при реализирането на линейните пространствени филтри е че при изчисляването на филтрираните пикселите се използват множество операции сумиране, умножение и деление. Броят на тези операции нараства с увеличаването на размера на изображението и достига до десетки и стотици милиони. Това ги прави трудно приложими при използването на микропроцесори с общо предназначение (GPP) или DSP процесори. Ограниченията които са присъщи на GPP и DSP процесори могат да бъдат преодолени чрез имплементирането на LSF в един чип (System on a Chip (SoC)) базиран на Field Programmable Gate Arrays (FPGAs). Този подход позволява използването на конвейрни и комбинирани схеми за изчисляване на LSF.



Фигура 1.2.3. Конвейрна реализация на FPGA-базиран линеен филтър.



Фигура 1.2.4. Хибридна схема на FPGA-базиран линеен филтър.

От особена важност е коефициентите на Гаусовите ядрата да се предстявят като цели положителни числа. Това би довело до реализирането на LSF с DSP блокове и процесори които използват вградените в FPGA компоненти. Очакваният ефект е намаляване на броя на използваните логически елементи на FPGA с общо предназначение.

В сравнение с конвейрното имплементиране на LSF комбинираните схеми позволяват да се редуцира броят на умножителите и операциите от S^2 до S пъти. Тези схеми могат да се усъвършенстват в следните направления: редуциране броя на регистрите в които се съхраняват коефициентите на ядрото; редуциране броя на операционните блокове и оптимизиране на структурата на акумулаторите.И двата метода имат следните недостатъци: големия брой на умножители, суматори и мултиплексори; възниква необходимост от множество регистри за съхраняване на коефициентите на Гаусовите ядра. Тези проблеми, могат да се решат, ако коефициентите на ядрата се комбинират с управляващите сигнали в един ROM-базиран краен автомат (FSM).

В сравнение с конвенционалните методи за проектиране на FSM, които изполват логически елементи (LEs), ROM базираните FSMs имат определени предимства:

1. За реализирането на FSM се използват само ресурсите на ROM. Това води до значителнио редуциране на LEs на FPGA. Ефекта от редуцирането на LEs при използването на ROM базирани FSMs, е особенно значим, когато автоматите имат множество вътрешни състояния, в които се генерират константи и управляващи сигнали.

2. Максималната тактова честота на FSMs имплементирани в ROM не-зависи от броя на състоянията на автомата.

3. При проектиране на FSMs, може точно да се определят ресурсите на използвания ROM.

Значителни затруднение при проектирането на ROM базирани FSM оказват три основни фактора:

1. Отсъствието на инструменти за дефиниране и описание на състоянията на автоматите.

2. Затрудненото кодиране на файла за инициализиране на паметта (MIF) със състоянията на FSM.

3. Липсата на средства за визуализиране на графа на автоматите, или тяхното представяне с вълнова диаграма.

За да се постигне ефективно имплементиране на ROM-базираните FSM, е необходимо да се разработят инструментални средства (езикови процесори), които да генерират изходен файл във формат MIF и графичен интерфейс за визуализиране на състоянията на автомата.

ГЛАВА 2. МОДЕЛИ, МЕТОДИКИ И ИНСТРУМЕНТАЛНИ СРЕДСТВА ЗА ПРОЕКТИРАНЕ И ИЗСЛЕДВАНЕ НА ЛИНЕЙНИ ПРОСТРАНСТВЕНИ ФИЛТРИ

Пространственото филтриране е един от основните инструменти, използвани за обработване на изображения, които имат широк спектър на приложения. При филтриране на входно изображение $p(x,y)$ с размери $M \times N$ се създава нов пиксел $g(x,y)$ с координати, равни на координатите на центъра на обкръжението, и чиято стойност е резултат на операцията за филтриране. Операцията, с която се извършва филтрирането се определя от ефекта, който се очаква от филтрирането. За по-голямата част от линейните пространствени филтри (LSF), ефекта от филтрирането се определя от маската на тегловните коефициенти (или ядра) с размер $S \times S$, където S е нечетно число и $S \geq 3$.

В най-общ вид филтрирането на изображения с линейни пространствени филтри се определя от следната зависимост:

$$g(x,y) = \frac{\sum_{a=-k}^{k} \sum_{b=-k}^{k} w(a,b) \bullet p(x+a, y+b)}{\sum_{a=-k}^{k} \sum_{b=-k}^{k} w(a,b)}, \quad (2.1.1)$$

където:

$$k = \frac{S-1}{2}\ ,\ k = \{1,2,..,5\}; \quad (2.1.2)$$

$$x = 0,1,2,..,M-1; \quad (2.1.3)$$

$$y = 0,1,2,..,N-1; \quad (2.1.4)$$

$$w(a,b) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{a^2+b^2}{2\sigma^2}\right), \quad (2.1.5)$$

е нормализиран Гаусов 2D оператор.

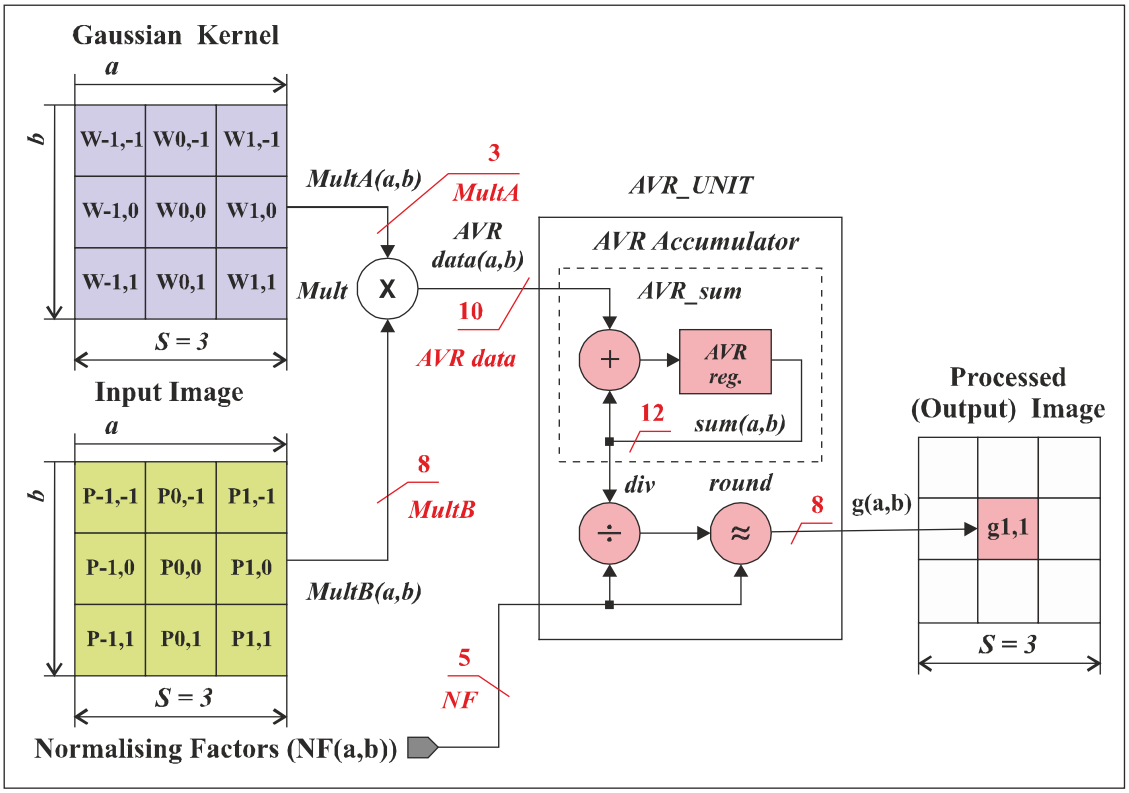
Хардуерно реализиране на филтрите средно претеглено, може да се извърши със следните три операции (Фиг.2.1.1):

1. Умножение на елементите с еднакви координати на матриците Gaussian Kernel и Input Image:

$$AVRdata(a,b) = w(a,b).p(a,b). \quad (2.1.7)$$

2. Сумиране на произведенията $AVRdata(a,b)$. С операция, $sum(a,b)$ се формира делимото на филтъра (2.1.1).

$$sum(a,b) = \sum_{a=-k}^{k} \sum_{b=-k}^{k} AVRdata(a,b). \quad (2.1.10)$$



Фигура 2.1.1. Концептуален модел на компоненти за хардуерно реализиране на линейни пространствени филтри

3. Нормализиране на филтрирания пиксел $g(1,1)$. Тази операция има за цел изравняване по разрядност на пикселите от входното изображение и пикселите на филтрираното изображение:

$$g(1,1) = div + round\,, \tag{2.1.11}$$

където:

$$div = sum(a,b)/NF(a,b)\,, \tag{2.1.12}$$

е операция, с която се нормализира изходната функция;

$NF(a,b)$ е фактор за нормализиране и

$$NF(a,b) = \sum_{a=-k}^{k}\sum_{b=-k}^{k} w(a,b); \tag{2.1.13}$$

round е операция, с която се извършва закръгление на резултата след операцията *div*.

Да означим с *tsum* времето за изпълнение на операцията $sum(a,b)$ (2.1.10), и приемем, че:

$$tsum = tclk.S^2\,, \tag{2.1.14}$$

а с *tdiv* означим времето, за което се изчислява операцията *div* (2.1.12).

Тогава отношението

$$Rm = tsum/tdiv\,, \tag{2.1.15}$$

може да се използва като критерий за оценяване на методите, по които ще се изчисляват филтрите. Доказано е, че ако $Rm > 1$, филтрите могат да се реализират конвейрно.

2.1. Сравнителен анализ при софтуерно и хардуерно реализиране на линейни пространствени филтри, имплементирани в FPGA

Възможни са два основни метода за реализиране на филтрите (2.1.1) с използване на FPGA платформи:

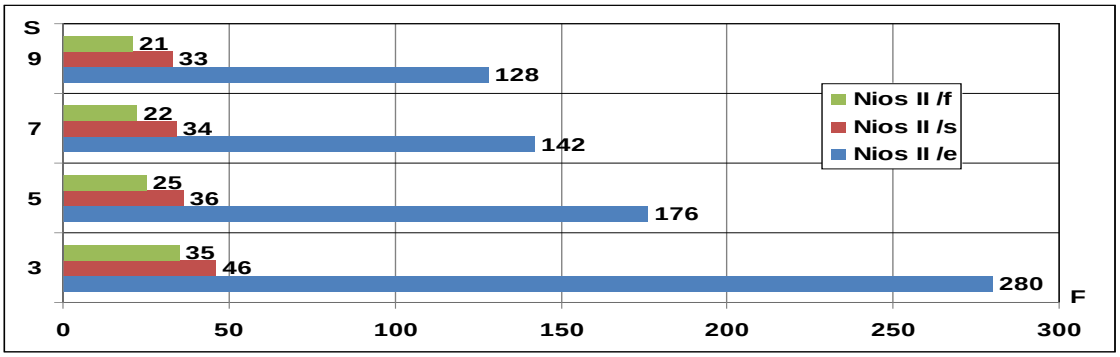
1. Софтуерен метод. При този метод филтрите се реализират с програми, които се изпълняват от микро контролери NIOS II/f/s/e имплементирани в FPGA.

2. Хардуерен метод. При този метод филтрите се реализират изцяло с компоненти синтезирани с логическите елементи на FPGA.

За да се сравнят двата метода, са разработени методика и инструментални средства. Дефиниран е фактора F :

$$F = Nsw / S^2, \qquad (2.1.23)$$

който отразява отношението на броя на циклите N_{SW} измерени при софтуерно реализиране на изследваните филтри и броя на циклите, необходими за хардуерно им реализиране.



Фигура 2.1.9. Относителни резултати (F) при софтуерно и хардуерно изчисляване на филтрите "средно-претеглено

От проведените изследвания могат да се направят следните изводи: хардуерно реализиране на линейни пространствени филтри с конвейрен метод превъзхождат софтуерните методи: от 21 пъти при процесора NIOS II/f и S = 9 до 280 пъти при процесора NIOS II/e и S = 3.

2.2. Генериране на тестови Гаусови ядра за проектиране и изследване на компоненти за LSF

За да се избегнат изчисленията на $w(a,b)$ (2.1.5) с компоненти, които оперират с числа, представени в плаваща запетая, кофициентите на Гаусови ядра трябва да се представят като цели положителни числа. Това ще ускори времето за изчисляване на филтрите и ще редуцира броя на използваните логическите елементи. При филтриране с филтъра „средно претеглено", всички пиксели на входно изображение участват в изчислението на филтрираното изображение. Това условие предполага, че всички кофициенти на Гаусови ядра са различни от нула.

Нека с $w_{\text{int}}(k,k)$ означим минималната стойност на функцията (2.2.1), преобразувана до цяло положително число. Да приемем, че за $w_{\text{int}}(k,k)$ е валидна следната зависимост:

$$w_{\text{int}}(k,k) = \lfloor w(k,k).V_{\text{int}} \rfloor = C_{\min}, \quad (2.2.4)$$

където: V_{int} , (2.2.5)

е коефициент, който преобразува функцията (2.2.4), до цели положителни числа;

$C_{\min}$, (2.2.6)

е константа, която определя конкретната стойност на $w_{\text{int}}(k,k)$.

2.2.2. Алгоритъм за генериране на тестови Гаусови ядра

Нека с $m, m+1$ са означени броя на двоичните битовете в интервала. в който ще се търси решение.

Начало на алгоритъма

Входни параметри: k (2.1.2); $C_{\min}$ (2.2.6) .

Стъпка #1. Определяне на началната стойност на m.

За дадено k , m може да се определи като се сравнят броя на двоичните комбинации в интервала

$$2^m, 2^{m+1} \quad (2.2.8)$$

и броя на елементите на Гаусовото ядро (2.1.5) :

Нека

$$2^m \geq (2k+1)^2, \quad (2.2.9)$$

тогава

$$m = \left\lceil \log_2 (2k+1)^2 \right\rceil. \quad (2.2.10)$$

Стъпка #2. Определяне на V_{int} *и* V_R

V_{int} може да се определи от горната граница на интервала, в който се търси решение:

$$V_{\text{int}} = 10^{\left\lceil \log(2^{m+1}+1) \right\rceil} \quad (2.2.11)$$

Нека с V_R означим нормираната към $C_{\min}$ реципрочна стойност на V_{int} и

$$V_R = C_{min} / V_{int}. \quad (2.2.12)$$

V_R се въвежда, за да се извършат изчисленията на Гаусовите ядра в интервала (2.2.8). Този подход позволява впоследствие да се извърши оценка на грешката от преобразуване на тегловните коефициентите на Гаусовите ядра от реални в цели числа.

Стъпка #3. Определяне на σ_L *за долната граница на интервала.*

Нека с $w_L(0,0)$ означим максималната стойност на Гаусовата функция за долната граница на интервала и приемем, че:

$$w_L(0,0) = 2^{m+1}.V_R, \quad (2.2.13)$$

тогава може да се определи σ_L за долната граница на интервала:

$$\sigma_L = \frac{1}{\sqrt{2\pi.2^{m+1}.V_R}} \tag{2.2.14}$$

Стъпка #4. Определяне на σ_H за горната граница на интервала.

Нека с $w_H(0,0)$ означим максималната стойност на Гаусовата функция за горната граница на интервала и приемем, че

$$w_H(0,0) = 2^m.V_R \tag{2.2.15}$$

Тогава може да се определи σ_H за долната граница на интервала:

$$\sigma_H = \frac{1}{\sqrt{2\pi.2^m.V_R}} \tag{2.2.16}$$

Стъпка #5. Определяне на $w(a,b)$ и $w_{\text{int}}(a,b)$.

В интервала:

$$\sigma_L \le \sigma \le \sigma_H \tag{2.2.17}$$

се търси такава стойност на σ, за която

$$w(k,k) = V_R. \tag{2.2.18}$$

Ако равенството не е изпълнено, алгоритъмът се повтаря от *стъпка #2* за $m = m+1$, в противен случай с получената стойност на σ се изчисляват $w(a,b)$ (2.1.5) и целочислените коефициенти на Гаусовото ядро:

$$w_{int}(a,b) = \lfloor w(a,b).C_{int} \rfloor. \tag{2.2.19}$$

Край на алгоритъма.

След като са определени целочислените коефициенти на Гаусовото ядро, се изчислятват разрядноста на сигналите: $MultA(a,b)$; $MultB(a,b)$; $AVRdata(a,b)$; NF; $sum(a,b)$ и факторът за нормализиране на изходната функция $NF_{\text{int}}(a,b)$.

1. Нека с $MultA$ означим разрядността на сигналите $MultA(a,b)$, а с w_{max} означим стойността на централния елемент на матрицата на тегловните коефициенти

$$w_{max} = \lfloor w(0,0).C_{int} \rfloor, \tag{2.2.20}$$

$$\text{тогава: } MultA = \lceil \log_2(w_{\max} + 1) \rceil. \tag{2.2.21}$$

2. Нека с $MultB$ означим разрядността на сигналите $MultB(a,b)$, а с p_{max} означим максималната стойност на пикселите във входното изображение.

$$\text{Тогава: } MultB = \lceil \log_2(p_{\max} + 1) \rceil. \tag{2.2.22}$$

За това изследване е прието, че пикселите във входното изображение са ограничени в интервала:

$$0 \le p(a,b) \le p_{max},\ p_{max} = 255 \tag{2.2.23}$$

3. Нека с $AVRdata$ означим разрядността на сигналите $AVRdata(a,b)$.

Тогава:

$$AVRdata = \lceil \log_2((p_{\max} + 1).(w_{\max} + 1)) \rceil. \tag{2.2.24}$$

4. Нека с NF означим разрядността на сигналите $NF_{int}(a,b)$, където:

$$NF_{int}(a,b) = \sum_{a=-k}^{k} \sum_{b=-k}^{k} w_{int}(a,b), \qquad (2.2.25)$$

е стойността на фактора за нормализиране на изходната функция , изчислен с използване на целочислените коефициенти на Гаусовото ядро.

Тогава:

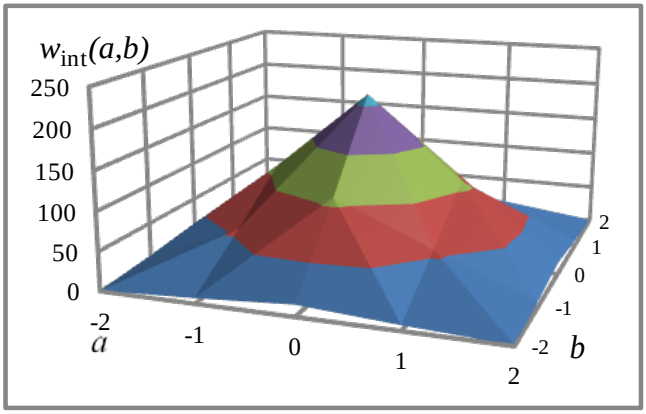
$$NF = \lceil \log_2 (NF_{\text{int}}(a,b)) \rceil \qquad (2.2.26)$$

5. Нека с *sum* означим разрядността на сигналите $sum(a,b)$(10).

Тогава:

$$sum = \lceil \log_2 (NF_{\text{int}}(a,b).(p_{\max}+1)) \rceil. \qquad (2.2.27)$$

Базирайки се на алгоритъма за генериране на тестови Гаусови ядра и правилата за изчисляване на разрядността на сигналите, е разработен генератор на тестови Гаусови ядра (Gaussian Kernel Generator) GKG.



Фигура 2.2.9. Графично представяне на $w_{int}(a,b)$

Таблица 2.2.7

a/b	-2	-1	0	1	2
-2	1	8	15	8	1
-1	8	56	109	56	8
0	15	109	213	109	15
1	8	56	109	56	8
2	1	8	15	8	1

Дискретни стойности на Гаусови ядра представени с цели числа: $k = 2$

В таблица 2.2.13, са показани обобщените резултати при изчисляване с GKG на разрядността на сигналите и фактора за нормализиране на филтрирания пиксел.

Таблица 2.2.13

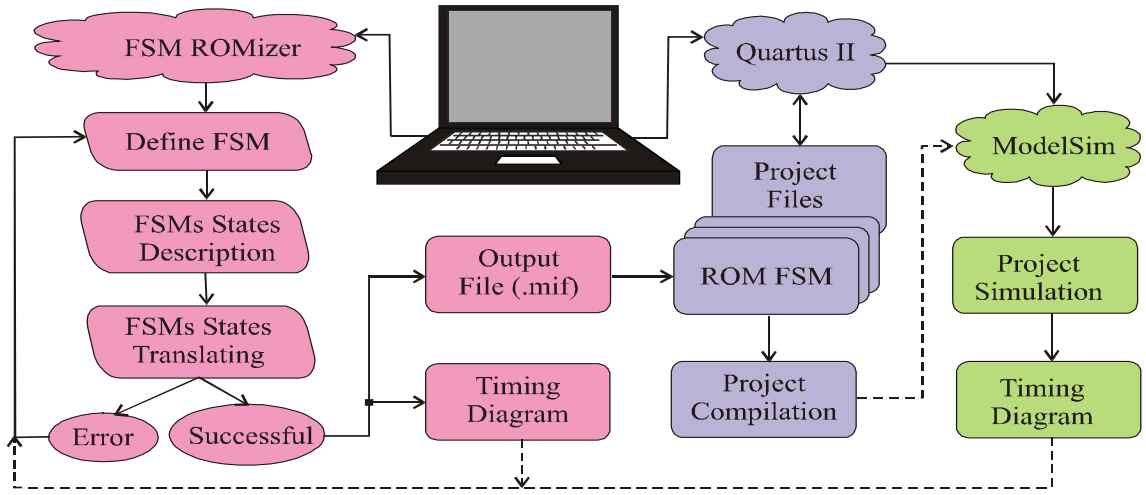
k	1*	1	2	3
MultA	1	3	8	11
MultB	8	8	8	8
AVRdata	8	10	16	19
Sum	12	12	18	22
NF	9	16	993	9974
$NF_{\text{int}}(a,b)$	4	5	10	14
Забележка: Box mask*				

Таблица 2.2.13. Обобщени резултати при изчисляване с GKG на разрядността на сигналите и фактора за нормализиране на изходната функция.

С получените в Таблица 2.2.13 резултати се определя разрядността на сигналите, които свързват отделните субкомпоненти (Фиг. 2.2.1).

2.3. Инструментални средства за проектиране на ROM базирани крайни автомати, за генериране коефициентите на Гаусовите ядра и сигнали за управление на компоненти за LSF

С езика Perl са създадени инструментални средства, наречени „FSM ROMizer", с които се улеснява и ускорява проектирането на ROM базирани крайни автомати. С тези автомати се генерират коефициентите на тестовите Гаусови ядра и сигнали за синхронизиране на процесите на компонента AVR Unit. Предложен е метод за дефиниране на автоматите. Описани са правилата с които се дефинира съответствието между множеството от променливи, които формират входната азбука на автоматите и променливите, които формират изходните функции. Създадени са правила за компактно описание на състоянията на дефинираните крайни автомати. Дефиниран е синтксис на „FSM ROMizer" с използването на разширена Бакус-Наурова форма (Extended BNF). На тази база, е създаден парсер, който след транслиране на състоянията на автоматите генерира: списък с откритите грешки; изходен файл за инициализиране на паметта (.MIF) в която ще се имплементират крайните автомати; вълнова диаграма със състоянията на крайните автомати.



Фигура 2.3.1. Проектиране на ROM базирани FSM, с „FSM ROMizer"

Един или повече крайни автомати, имплементирани в един ROM, се дефинират в *Section of definitions,* ограничена между директивите *DEFINE FSM* и *END DEFINE*. С *DEFINE FSM* се слага начало на дефиниционната част. Веднага след тази директива следва двуеточие и името на дефинирания ROM FSM. С това име, *FSM-ROMizer* ще създаде файла за инициализиране на паметта (.mif) със състоянията на автоматите.

DEFINE FSM: *MyFsmName*;

Section of definitions

END DEFINE;

В дефиниционната секция се описва съответствието между множеството от сигналите, които формират входната азбука и сигналите, които формират изходните функции.

Състоянията на вече дефинирания краен автомат се описват в „описателна секция”, ограничена между директивите *CONTENT BEGIN* и *END*:

CONTENT BEGIN

descriptive of section

END;.

В общ вид, синтаксиса на описателната секция може да се представи по следния начин:

address of ROM : Description of states;,

където:

с *address of ROM* се задава абсолютната стойност на адреса, в който ще се опишат състоянията на автоматите. В този смисъл *address of ROM,* може да се разглежда още и като номер на състоянията на автоматите;

c description of states се описват стойностите на променливите:

<description of states>:= variable[msb : 0] = b"value",.., variable= b'value';

За да създат предпоставки за компактност при описанието на състоянията на FSM са дефинирани следните правила: след Reset всички входни и изходни променливи се установяват в нула; ако за дадено състояние на автомата, на една променлива е присвоена определена стойност, тя ще запази тази стойност до промяната и в друго състояние на автомата; предварително дефинираните входни или изходни променливи (state, next_state) могат да бъдат последвани от унарната операция “post increment” (++). Автоматичното инкрементиране ще продължи до следващото състояние на автомата, в което променливите приемат нова стойност.

След като са дефинирани един или по-вече крайни автомати, и са описани техните състояния, може да се пристъпи към транслиране. Транслирането се извършва на три етапа:

1. Parsing. На този етап се извършват типичните за един транслатор анализи . Ако се установи несъответствие между правилата за дефиниране и описание на автомата и синтаксиса на FSM ROMizer, процеса на транслиране се преустановява и се генера списък с грешките.

2. Генериране на изходен файл. При успешен анализ се генерира изходен файл с името на автомата и разширение .mif . Синтаксисът с който се описва изходния файл, има следния вид:

ROM address : ROM data; -- comment,

където:

“ *ROM address”* е абсолютен адрес от ROM (или още номер на състоянията), който се образува от конкатенацията (“&”) на всички входни променливи;“ *ROM data”* е изходна функция, която се образува от конкатенацията на всички изходни променливи;

3. Генериране на диаграма на състоянията. За да се визуализират състоянията на автоматите, се генерира вълнова диаграма, с която графичен вид се показват състоянията на входните и изходни променливи.

ГЛАВА 3. КОНВЕЙРНО ИЗЧИСЛЯВАНЕ НА LSF

В Глава 2 е доказано, че линейните пространствени филтри могат да се реализират конвейрно, ако отношението Rm>1 (2.1.15).

Нека с *G(x,y)* означим крос-корелационната функция на линейните пространствени филтри (2.1.1)

$$G(x,y)=\sum_{a=-k}^{k}\sum_{b=-k}^{k} w(a,b)\bullet p(x+a,y+b). \qquad (3.1)$$

Тогава, нормализираната стойност на филтрирания пиксел *g(x,y)* може да се изчисли с два метода:

Метод #1 - посредством деление на *G(x,y)* с нормализиращия фактор *NF(a,b),* (2.1.13):

$$g(x,y)=\frac{G(x,y)}{NF(a,b)}. \qquad (3.2)$$

Метод #2 - посредством умножение на *G(x,y)* с реципрочната стойност на нормализиращия фактор *NF(a,b):*

$$g(x,y)=G(x,y).(NF(a,b))^{-1}. \qquad (3.3)$$

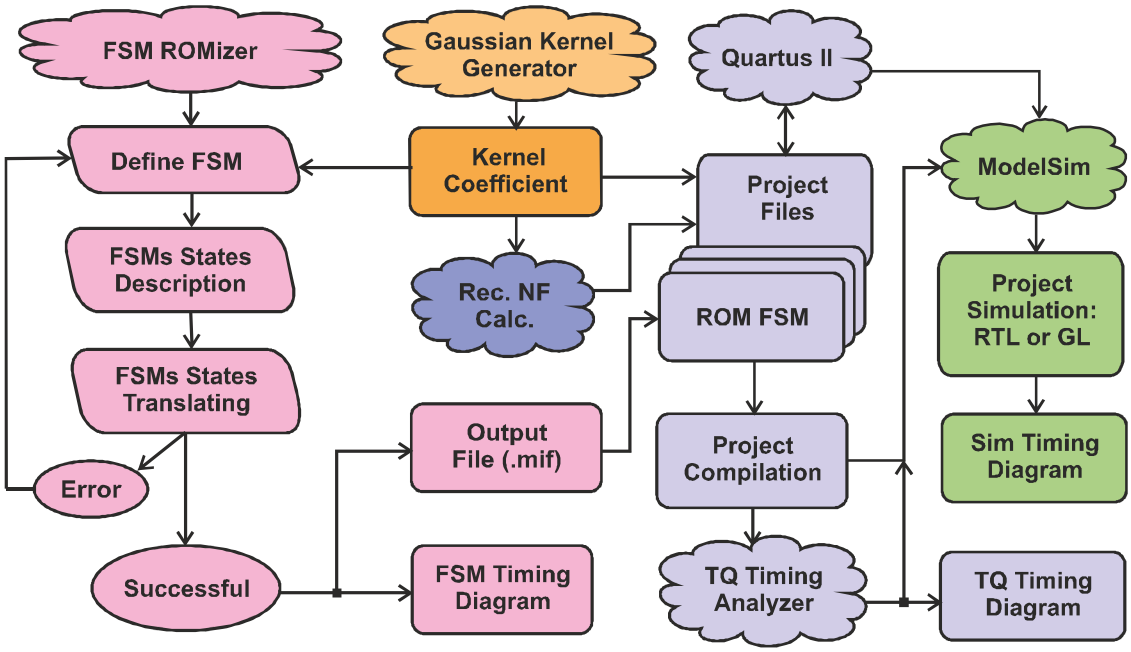
Основните различия на двата метода се състоят в компонентите и операциите, с които те ще се реализират при имплементирането им в FPGA. При Метод #1 нормализираната стойност на филтрирания пиксел се изчислява с делител, синтезиран с логическите елементи на FPGA. При Метод #2 изчисленията се извършват с вградените в FPGA умножители.

Проведените изследванията имат за цел сравняване на двата метода по следните критерии: еквивалентност на получените резултати; заемани ресурси при имплементиране на филтрите в FPGA; времето за което се изчислява нормализираната стойност на филтрирания пиксел g(x,y) и експериментално установяване на Rm.

За постигане на поставената цел е разработена методика за проектиране и изследване на LSF (Фиг. 3.2). Методиката се състои от 6 етапа:

1. С Gaussian Kernel Generator (GKG), за размери на прозорците *S*x*S* =3x3, 5x5, 7x7, .. се генерират коефициентите на ядрата. Изчисляват се размерите на променливите (Глава 2.2).
2. Изчисляват се реципрочните стойности на фактора за нормализиране NF (Rec. NF Calc.).
3. С VHDL в развойната (CAD) среда Quartos II се създава проект на DSP блокове за конвейрно изчисляване на LSF. Определят се ресурсите, които са необходими за имплементиране на проекта в FPGA.
4. С FSM ROMizer се дефинира ROM-базиран краен автомат за управление и синхронизиране на процесите в DSP процесора. След определяне на състоянията на автомата, се извършва транслиране, при което се генерира вълнова диаграма на състоянията и файл за инициализиране на ROM (,mif) (Глава 2.3).
5. След компилиране на проекта с Time Quest Timing Analyzer се определя *Rm.*

6. Проектът се симулира, за да се установи функционалността на модела.



Фиг.3.2. Методика за проектиране и изследване на LSF.

За изчисляване на реципрочните стойности на фактора за нормализиране на филтрирания пиксел $g(x,y)$ (3.3) е предложен алгоритъм и програма (калкулатор) (Rec. NF Calc.). При зададени стойности на NF, калкулаторът последователно изчислява $(NF(a,b))^{-1}$ (Таблица 3.5.1).

Таблица 3.5.1
Реципрочни стойности на нормализиращия фактор: r = 0.5

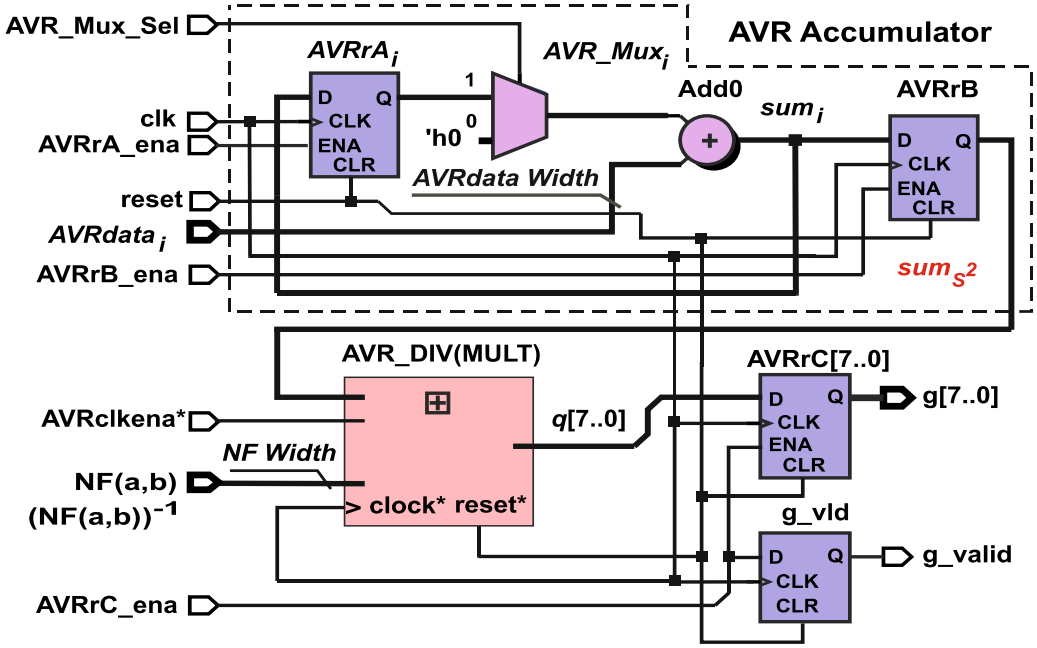
NF	Rec'D	Rec'B	RelNF [%]
9	0.111111	111000111	99.9756
10	0.1	110011001	99.9756
16	0.0625	100000000	100
993	0.00100705	100000111	99.8137
9974	0.000100261	1101001	99.8755

При достигане на еднакви резултати за нормализиране на филтрирания пиксел с двата метода, калкулаторът генерира окончателната стойност на $(NF(a,b))^{-1}$.

Дефиниран е архитектурен модел на компонент за изчисляване на LSF с двата метода. В този модел са определени: кои са основните субкомпоненти, от които се изгражда компонента и какво е тяхното функционално предназначение; кои са сигналите, които свързват отделните субкомпоненти; с кои портове се дефинира компонента.

Създаден е функционален модел на компонент за изчисляване на LSF с двата метода. В този модел са определени: процесите за установяване на регистрите; условията за нормализиране на филтрирания пиксел и входно-изходните портове.

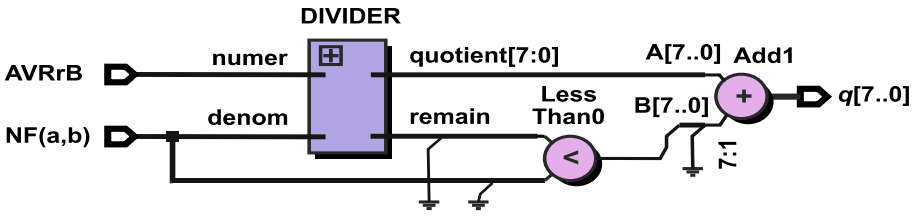
Дефинираните архитектурен и функционален модел е подходяща база за създаване на RTL дизайн (фиг. 3.3).



Фиг.3.3. RTL дизайн на компоненти за изчисляване на LSF с методите #1 и #2.

Нормализирането на филтрирания пиксел се реализира със субкомпонента AVR_DIV(MULT). Разгледани са два варианта на RTL дизайн:

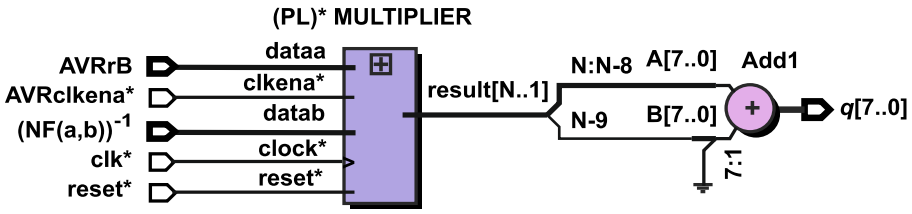
1. Метод #1 (Фиг. 3.4) - посредством деление на $G(x,y)$ с нормализиращия фактор $NF(a,b)$.



Фиг.3.4. RTL дизайн на субкомпонент AVR_DIV

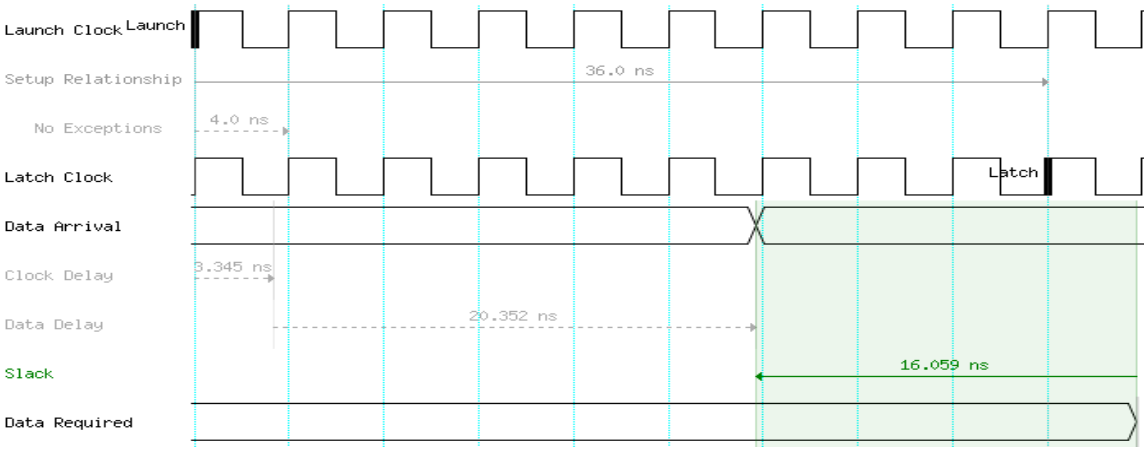
При този вариант, нормализирането се извършва с компонент „DIVIDER", синтезиран с логическите елементи на FPGA. Закръгляването на резултата от делението се реализира с компаратора „Less Than0" и суматора „Add1".

2. Метод #2 (Фиг.3.5) - посредством умножение на $G(x,y)$ с реципрочната стойност на нормализиращият фактор $(NF(a,b))^{-1}$. При този вариант, нормализирането се извършва с компонент „(PL)* MULTIPLIER", синтезиран с вградените в FPGA умножители. Предвидена е възможност да се използват умножители със или без Pipe Line. Закръгляването на резултата от умножението q[7..0] се реализира със суматора „Add1".

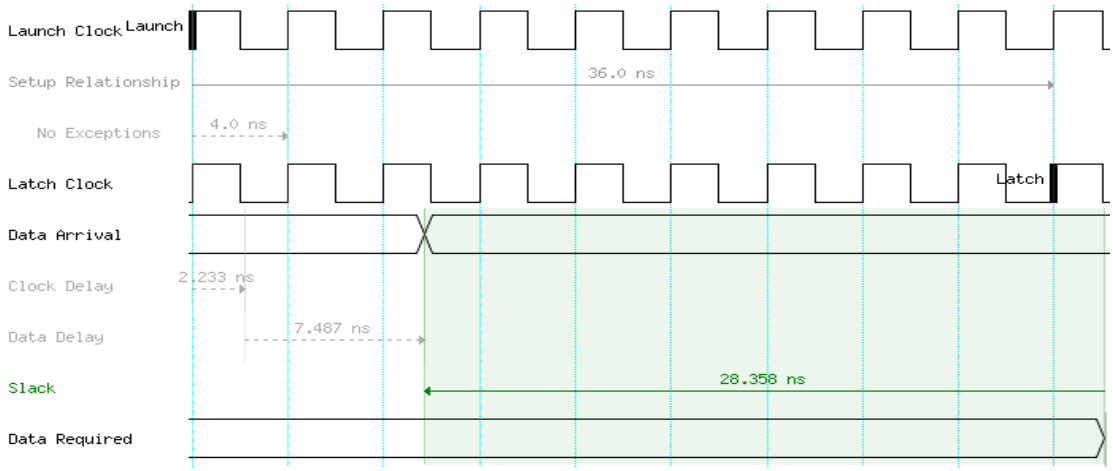


Фиг. 3.5. RTL дизайн на компонент AVR_MULT.

С Quartus II TimeQuest Timing Analyzer е проведен статичен времеви анализ между регистрите AVRrB и AVRrC. Синтезът на компонентите за умножение и деление е извършен без използване на pipe line.



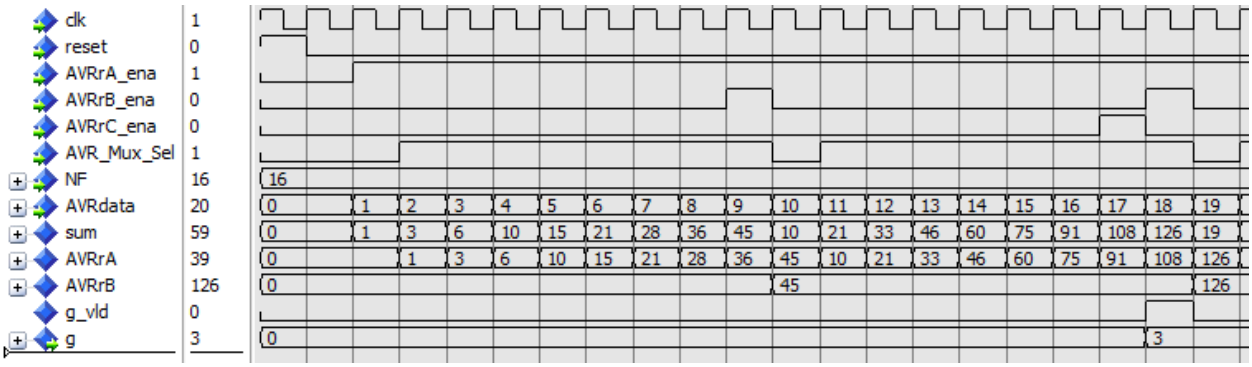
Фиг 3.8. Нормализиране на филтрирания пиксел с Метод #1 - времеви анализ: *S*=3; FPGA - 5CEBA4F17C6; *Rm* =1.519



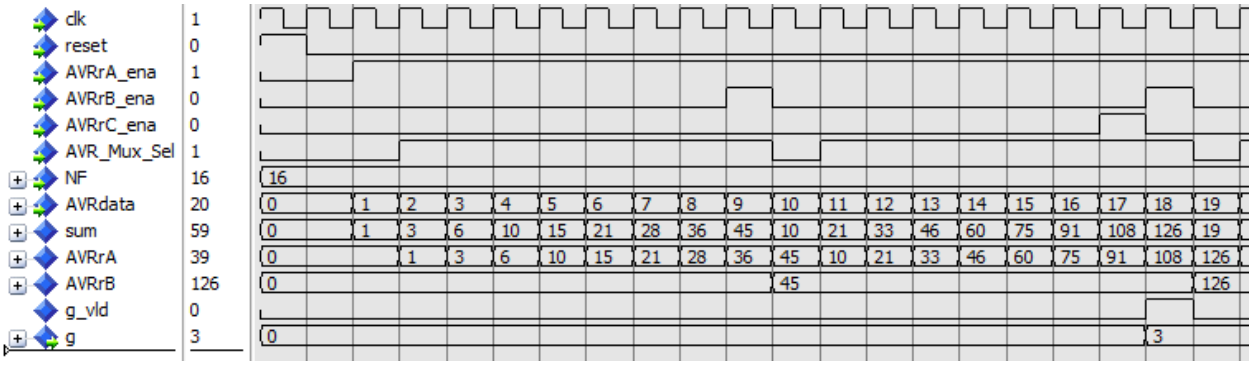
Фиг 3.9. Нормализиране на филтрирания пиксел с Метод #2 - времеви анализ: S=3; FPGA - 5CEBA4F17C6; *Rm* = 3.69

Констатирано е, че и при двата метода, изчисляването на LSF може да се реализира конвейрно.

За симулиране на субкомпонента е създадена тестова среда (Test Bench). Средата се използва за наблюдаване поведението на дизайна на RTL и Gate Level нива. Резултатите от симулирането с ModelSim са показани на Фиг.3.14 и 3.15.



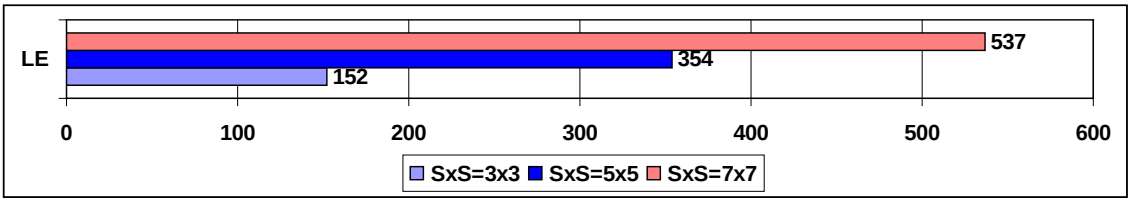
Фиг.3.14 Симулиране. – Метод #1



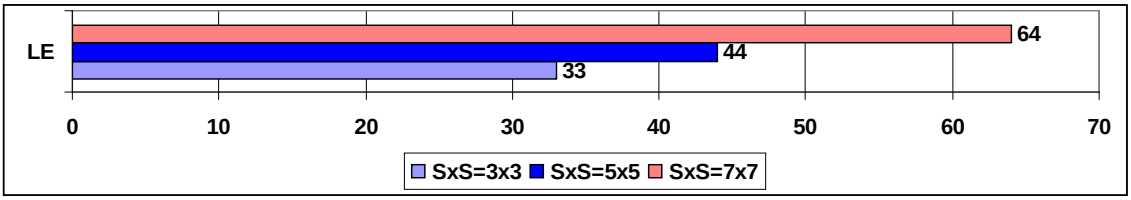
Фиг.3.15 Симулиране. – Метод #2

След извършеното симулиране е констатирано пълно съответствие на резултатите получени с двата метода.

С Quartus II на Altera са проведени изследвания за определяне на заеманите ресурси при имплементиране на компонентите в FPGA с двата предложени метода.



Фиг.3.12. Заемани ресурси във FPGA – метод 1



Фиг.3.13. Заемани ресурси във FPGA – Метод #2

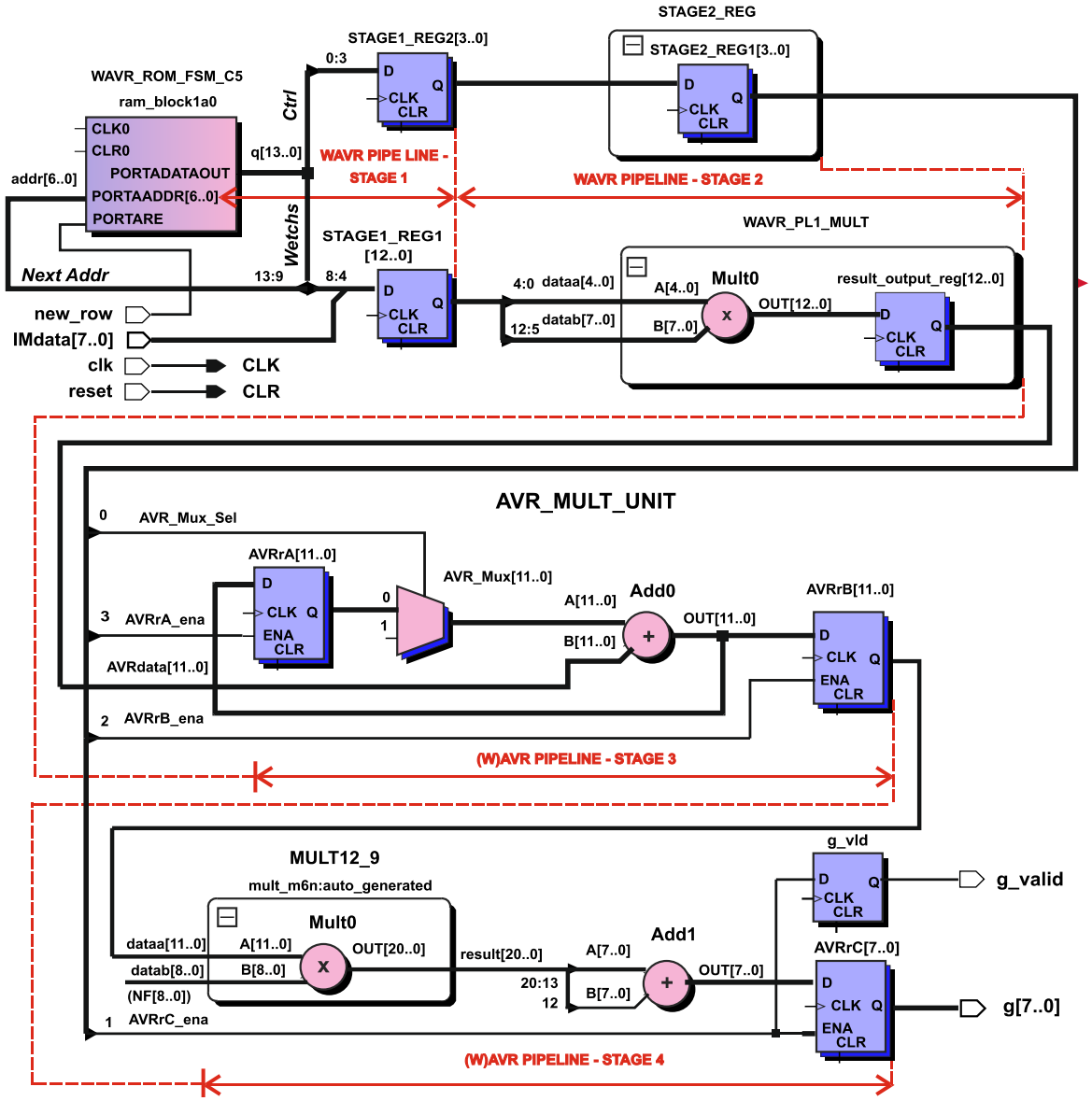
За различни стойности на прозореца $S \times S$ са определени броя на логическите елементи (LE) в зависимост от AVRdata Width и NF Width. Констатирано е, че реализирането на компонентите с метод #2, сравнен с Метод #1, изисква 4.6 пъти по-малко логически елементи (LE) при SxS = 3x3, до 8.3 пъти при SxS = 7x7. Проектиран е четириетапен DSP операционен блок за конвейрно изчисляване на LSF (Фиг3.17).

Stage #1. В този етап на конвейра синхронно с постъпването на входните пиксели IMdata, крайния автомат за управление на DSP процесора (WAVR_ROM_FSM_C5) генерира три групи сигнали:

1. Control – управление и синхронизиране на DSP блока AVR_MULT_UNIT.
2. Weights – тегловни коефициенти на гаусовите ядра.
3. Next Addr – следващо състояние на автомата.

При следващия клок (clk), текущите стойности на Control и Weights се съхраняват в регистрите STAGE1_REG0 и STAGE1_REG1.

Stage #2. В този етап на конвейра, с умножителя WAVR_PL1_MULT се реализира конволюцията AVRdata на входните пиксели IMdata и тегловни коефициенти на Гаусовите ядра. С регистъра STAGE2_REG сигналите Control закъсняват с един такт на clk, за да се синхронизират с AVRdata.



Фиг. 3.17.Четириетапен DSP операционен блок за конвейрно изчисляване на LSF – етапи на конвейрите

Stage #3. В този етап на конвейра се изчислява крос-корелационната функция *G(x,y)*, с междинната променлива sum_i :

$$sum_i = \sum_{i=1}^{S^2} AVRdata_i \ .$$

При $i=S^2$ окончателната стойност на sum_{S^2} се съхранява в регистъра AVRrB без промени до изчисляване на новата и стойност в следващия прозорец.

Stage #4. Нормализиране на филтрирания пиксел g посредством умножение с умножителя MULT12_9 на sum_{S^2} и реципрочната стойност на нормализиращия фактор $(NF(a,b))^{-1}$- Метод #2.

Извършен е статичен времеви анализ между регистрите на отделните етапи на конвейра. Сравнени са реализации при които са използване две фамилии FPGA.

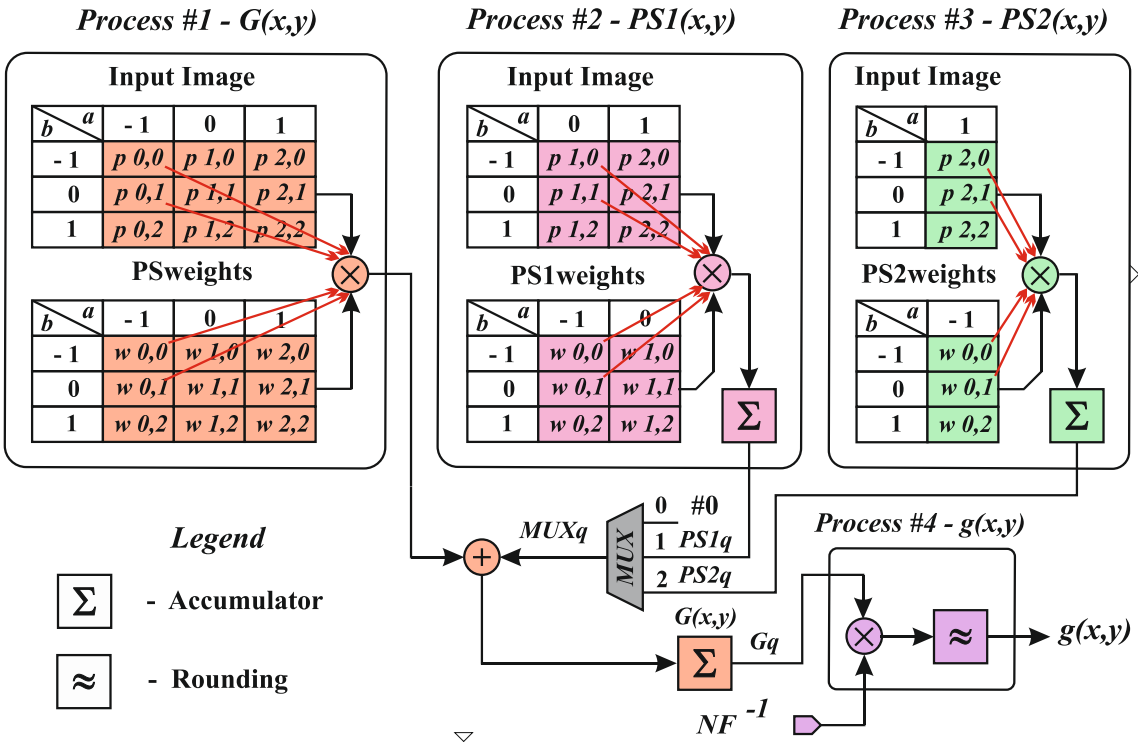
Таблица 3.7
DSP операционен блок - статичен времеви анализ

	Cyclone IVE EP4CE15E22C6			Cyclone VE Base 5CEBA4F17C6		
Stage	Clock Delay	Data Delay	Slack	Clock Delay	Data Delay	Slack
1	2.57	3.81	0.498	4.297	2.188	1.94
2	2.171	2.62	2.62	3.645	1.846	2.961
3	2.236	3.514	3.514	3.349	1.887	2.929
4	2.236	9.08	35.747	3.149	8.214	37.194
	Data Arrival			Data Arrival		
	Data Required			Data Required		
	Conditions: tclk = 5ns; S=3					

Установено е, че и при двете фамилии FPGA, критични са времевите параметри, които възникват на Stage #1. По отношение на критичните стойности на параметъра Slack, фамилията ”Cyclone VE Base” превъзхожда 3.9 пъти фамилията ”Cyclone IVE”, което я прави предпочитана за конвейрно изчисляване на LSF.

ГЛАВА 4. ПАРАЛЕЛНО ИЗЧИСЛЯВАНЕ НА ЛИНЕЙНИ ПРОСТРАНСТВЕНИ ФИЛТРИ С ИЗПОЛЗВАНЕ НА ЧАСТИЧНИ СУМИ

Алгоритъмът за паралелно изчисляване на LSF с използването на частични суми се базира на факта, че при филтрирането в няколко последователни прозореца се използват пиксели, които имат общи локални координати. Това обстоятелство може да се използва за създаване на алгоритми, при които паралелно с изчисляване на текущата стойност на $G(x,y)$ и $g(x,y)$ се изчисляват частичните суми $PS1(x,y)$, $PS2(x,y)$, съставени от пиксели и тегловни коефициенти, които се използват в следващите итерации.



Фиг. 4.3. Концептуален модел на метод за паралелно изчисляване на LSF с използването на частични суми. : Step #1; x,y =1;

При размер на прозореца $S=3$, в три последователни стъпки се реализират четири паралелни процеса, с които се изчисляват $G(x,y)$, $g(x,y)$, $PS1(x,y)$, и $PS2(x,y)$. Окончателните стойности на получените резултати се съхраняват в изходните регистрите Gq, $PS1q$ и $PS2q$ на акумулаторите. Нормализираната стойност на филтрирания пиксел $g(x,y)$ се закръглява с фактора r, $r \in \{0,1\}$, за да се осигури по-голяма точност на изчисленията. Предложен е алгоритъм за паралелно изчисляване на LSF с използването на частични суми.

Начало на алгоритъма

y=0

Step #1. Тази стъпка се изпълнява за всички $x = 1$. Реализират се три паралелни процеса в които се изчисляват $G(x,y)$, окончателната стойност на $PS1(x,y)$, началната стойност на $PS2(x,y)$ и се установяват текущите стойности на регистрите Gq, $PS1q$, $PS2q$. За изпълняване на изчисленията са необходими S^2 операции.

$x = 1;\ y = y + 1$.

Process #1:

$$G(x,y) = MUXq + \sum_{a=-1}^{1} \sum_{b=-1}^{1} w(a+1,b+1)\,.\,p(x+a,y+b); \tag{4.1}$$

$$MUXq = \#0\ ; \tag{4.2}$$

if $a = 1$ ***and*** $b = 0$ ***then*** $Gq = G(x,y) + w(1,1)\,p(x+1,y+1)$. (4.3)

Process #2:

$$PS1(x,y) = \sum_{a=-1}^{0} \sum_{b=-1}^{1} w(a+1,b+1)\,.\,p(x+a+1,\ y+b); \tag{4.4}$$

if $a = 0$ ***and*** $b = 0$ ***then*** $Gq = PS1(x,y) + w(1,2)\,p(x+1,y+1)$. (4.5)

Process #3:

$$PS2(x,y) = \sum_{b=-1}^{1} w(0,b+1)\,.\,p(x+1,\ y+b). \tag{4.6}$$

Step #2. Тази стъпка се изпълнява за $\forall x, x \neq 1$. Реализират се четири паралелни процеса, с които се изчисляват текущите стойности на $g(x,y)$ и $G(x,y)$, началната стойност на *PS1(x,y)* и окончателната стойност на *PS2(x,y).* За изпълняване на изчисленията са необходими *S* операции.

Process #4:

$$g(x,y) = Gq.NF^{-1} + r\ . \tag{4.7}$$

Преместване на прозореца с една позиция надясно

$x = x + 1$.

Process #1:

$$G(x,y) = MUXq + \sum_{b=-1}^{1} w(2,b+1)\,.\,p(x+1,\ y+b); \tag{4.8}$$

if $b=0$ ***then*** $MUXq = PS1q$ ***else*** $MUXq = \#0$; (4.9)

if $b=0$ ***then*** $Gq = G(x,y) + w(2,2).p(x+1,y+1)$. (4.10)

Process #2:

$$PS1(x,y) = \sum_{b=-1}^{1} w(2,b+1).p(x+1,\ y+b). \tag{4.11}$$

Process #3:

$$PS2(x,y) = PS2(x-1,y) + \sum_{b=-1}^{1} w(2,b+1).\ p(x+1,\ y+b); \tag{4.12}$$

if $b=0$ ***then*** $PS2q = PS2(x,y) + w(0,2).p(x+1,y+1)$. (4.13)

Step #3. Тази стъпка се изпълнява за $\forall x, x \neq 1$. Реализират се четири паралелни процеса с които се изчисляват текущите стойности на $g(x,y)$ и $G(x,y)$, окончателната стойност *PS1(x,y)* и началната стойност на *PS2(x,y).* За изпълняване на изчисленията са необходими *S* операции.

Process #4:.

$$g(x,y) = Gq.NF^{-1} + r\ . \tag{4.14}$$

Преместване на прозореца с една позиция надясно

$$x = x + 1.$$

Process #1:

$$G(x,y) = MUXq + \sum_{b=-1}^{1} w(2,b+1)\,.\,p(x+1,\ y+b)\,; \tag{4.15}$$

$$\textbf{\textit{if}}\ \ b=0\ \ \textbf{\textit{then}}\ \ MUXq = PS2q\ \ \ \textbf{\textit{else}}\ MUXq = \#0\,; \tag{4.16}$$

$$\textbf{\textit{if}}\ \ b=0\ \ \textbf{\textit{then}}\ \ Gq = G(x,y) + w(2,2).p(x+1,y+1)\,. \tag{4.17}$$

Process #2:

$$PS1(x,y) = PS1(x-1,y) + \sum_{b=-1}^{1} w(2,b+1).\ p(x+1,\ y+b); \tag{4.18}$$

$$\textbf{\textit{if}}\ \ b=0\ \textbf{\textit{then}}\ \ PS1q = PS1x,y) + w(0,2).p(x+1,y+1)\,. \tag{4.19}$$

***Process #3*:**

$$PS2(x,y) = \sum_{b=-1}^{1} w(2,b+1).p(x+1,\ y+b). \tag{4.20}$$

Операции за управление на циклите:

if $x \le M-2$ ***then*** *Step #2*

else if $y \le N-1$ ***then*** *Step #1*

else **край на алгоритъма.**

Ефекта от прилагането на алгоритъма с използване на частични суми може да се оцени като се сравни с алгоритъма без използване на частични суми. Нека, при дадено k с Wn означим броя на прозорците, необходими за филтрира на входно изображение с размер $M \times N$, а с S броя на циклите в един прозорец.

Тогава:

$$Wn = (M-k)\times(N-k); \tag{4.21}$$

$$O_C = S^2.Wn + 1; \tag{4.22}$$

$$O_P = S^2(N-k) + S.W_n, \tag{4.23}$$

където:

O_C е броя на операциите при алгоритми без използване на частични суми;

O_P е броя на операциите при алгоритми с използване на частични суми.

Нека съставим отношението

$$F_R = O_C / O_P, \tag{4.25}$$

тогава F_R може да се използва като критерий за оценяване на максималната стойност, с която се редуцира броя на операциите при алгоритмите с използване на частични суми спрямо алгоритмите без използване на частични суми.

Реалния ефект, с който се редуцира броя на операциите при алгоритмите с използване на частични суми може да се определи от отношението:

$$T_R = T_C / T_P, \quad (4.26)$$

където:

$$T_C = tclk1.O_C; \quad (4.27)$$

$$T_P = tclk2.O_P; \quad (4.28)$$

$$tclk1, \quad (4.29)$$

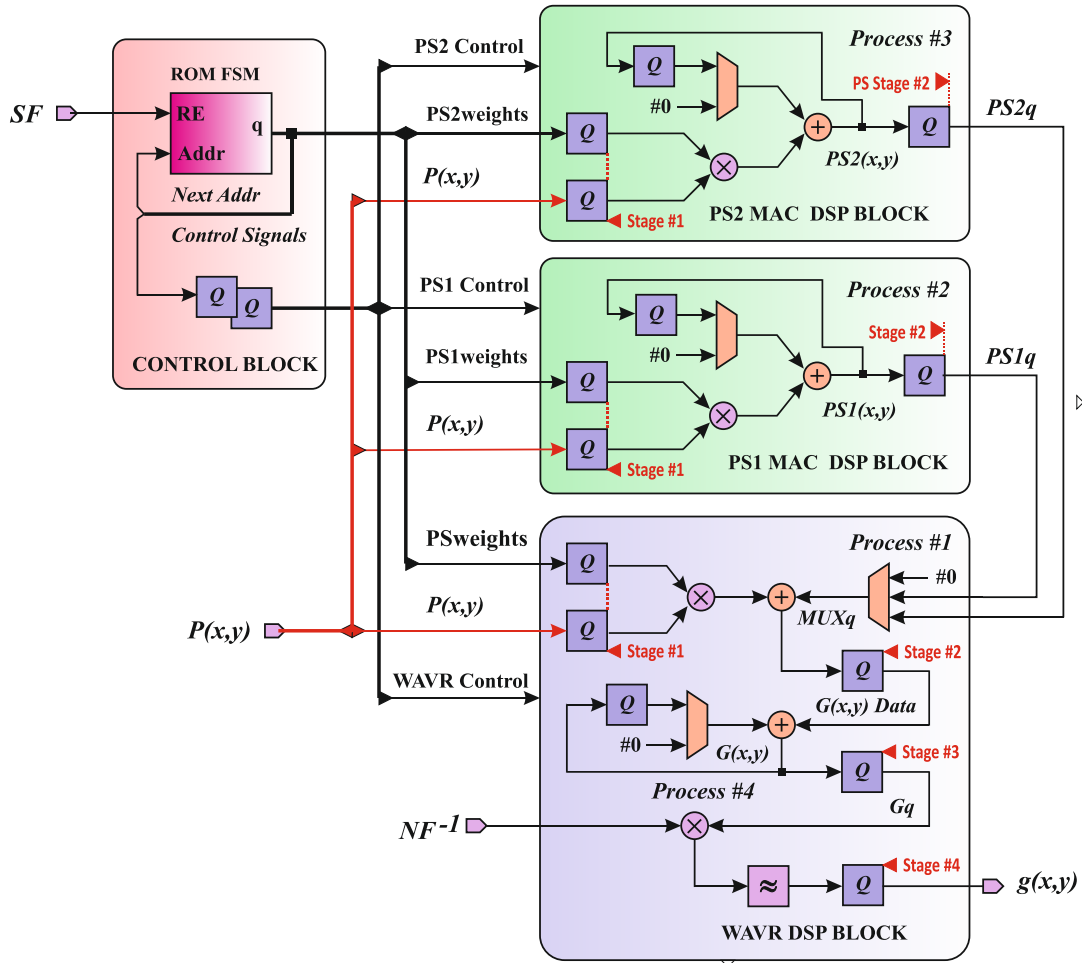
е периода на *clk* при алгоритми без използване на частични суми;

$$tclk2, \quad (4.30)$$

е периода на *clk* при алгоритми с използване на частични суми.

За разлика от F_R, отношението T_R отчита влянието на периода на clk при който се достига положителен Slack и се изчислява след времеви анализ на хардуерната реализация на филтрите.

За реализиране на Алгоритъма е разработен FPGA-базиран DSP операционен блок (Фиг.4.5). Използвана е комбинирана схема, при която паралелните процеси се съчетават с използването на конвейри.



Фиг. 4.5. Структурна схема на FPGA-базиран DSP операционен блок: *k=1*.

DSP операционният блок се състои от четири подблока:

1. WAVR DSP BLOCK. В този блок се изчислява нормализираната средно-претеглена стойност на филтрирания пиксел $g(x,y)$ (4.7), (4.14). За реализиране на операциите в този блок се използва четиристепенен конвейр

2. PS1(2) MAC DSP BLOCK. При *k=1*, два идентични multiply accumulate (MAC) DSP блока се използват за изчисляване на частичните суми *PS1(x,y)* (4.4), (4.11), (4.18) и *PS2(x,y)* (4.6), (4.12), (4.20). Изходните регистри *PS1q* и *PS2q* се установяват с текущите стойности на *PS1(x,y)* и *PS2(x,y)* в съответствие с условия (4.5), (4.13), и (4.19). За реализиране на операциите в тези блокове се използва двустепенен конвейр.

3. CONTROL BLOCK. За управление и синхронизиране на процесите в DSP се използва краен автомат синтезиран с вградените в FPGA ROM памети (ROM FSM). Този FSM генерира три групи сигнали:

3.1. Weight coefficients

Таблица 4.1.

Тегловни коефициенти: *k=1*; *NF =16*

Weights/Step	Step #1									Step #2			Step #3		
PSweights	1	2	1	2	4	2	1	2	1	1	2	1	1	2	1
PS1weights	NA			1	2	1	2	4	2	1	2	1	2	4	2
PS2weights	NA						1	2	1	2	4	2	1	2	1

3.2 Control signals. Сигналите WAVR Control, PS1 Control и PS2 Control се използват за управление на регистрите и мултиплексорите в отделните DSP блокове.

3.3. Next Address. С тези сигнали се определя следващо състояние на ROM FSM.

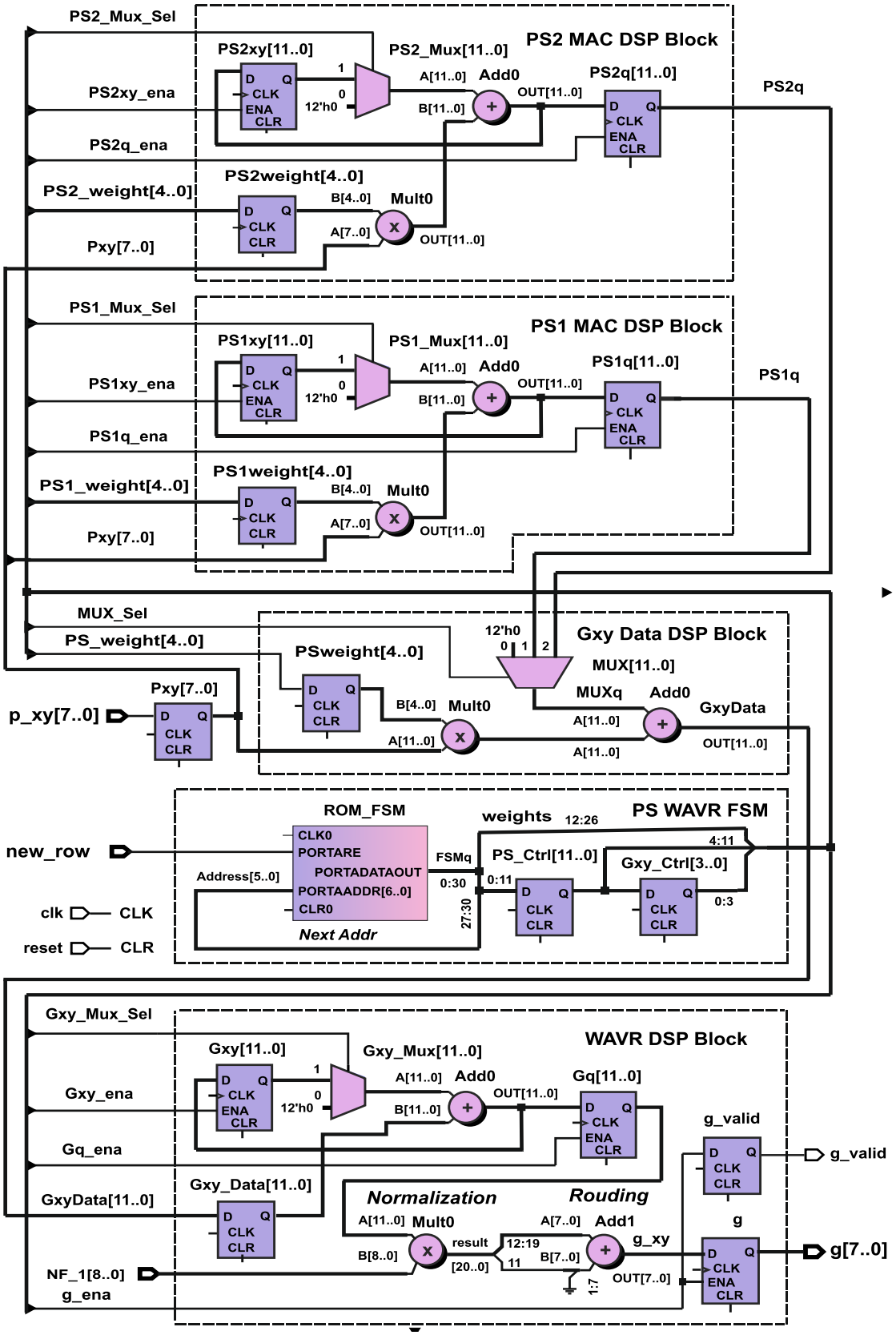
Интерфейсът на DSP операционен блок се реализира със следните входно-изходни портове:

- Start filtering (SF) – входен порт. Когато този сигнал се установи във високо ниво се стартират операциите за филтриране с LSF.
- Input pixsel (*p*(*x*,*y*)) – входен порт. Пикселите на входното изображение постъпват последователно на този порт по колоните на прозореца, в който се извършва филтрирането.
- Реципрочна стойност на нормализиращия фактор NF^{-1} – входен порт. Константата NF^{-1} (3.2) се установява на този порт преди активирането на *SF* и се задържа без промяна до филтрирането на всички пиксели в изображението.
- *g(x,y)* – изходен порт. На този порт се получава текущата стойност на филтрирания пиксел .

FPGA-базирания DSP операционен блок за паралелно изчисляване на LSF с използването на частични суми (Фиг.4.5), може да се разширява за други стойности на *k*, с инстанциране в проекта на нова двойка PS MAC DSP Block, за всяка нова стойност на *k* .

За имплементиране в FPGA на DSP операционния блок е разработен проект в който хардуерния дизайн е кодиран на VHDL. Коефициентите на Гаусовите ядра са изчислени калкулатора GKG, а реципрочната им стойност с калкулатора Rec. NF Calc. Файла за инициализиране на крайния автомат

„ROM_FSM” е получен с транслатора FSM ROMmizer. Резултатите след компилиране на проекта са показани във фигура 4.10.



Фиг. 4.10. RTL дизайн на DSP операционен блок за паралелно изчисляване на линейни пространствени филтри с използването на частични суми.

Проведените са изследвания които, имат за цел да се докаже ефекта от прилагането на алгоритмите и схемните решения по следните критерии:

1. Заемани ресурси.

Таблица 4.5.
Използвани ресурси (Altera Cyclone VE 5CEBA4F17C6 Device)

Resource utilization	Available	Without partial sum		With partial sum	
		Used	Utilization	Used	Utilization
Logic utilization (in ALMs)	18480	14	0.0756 %	36	0.1948 %
Total registers	-	41	-	86	-
Total block memory bits	3153920	1792	0.0568 %	1984	0.0629 %
Total (Mult) DSP Blocks	66	2	3 %	4	6 %

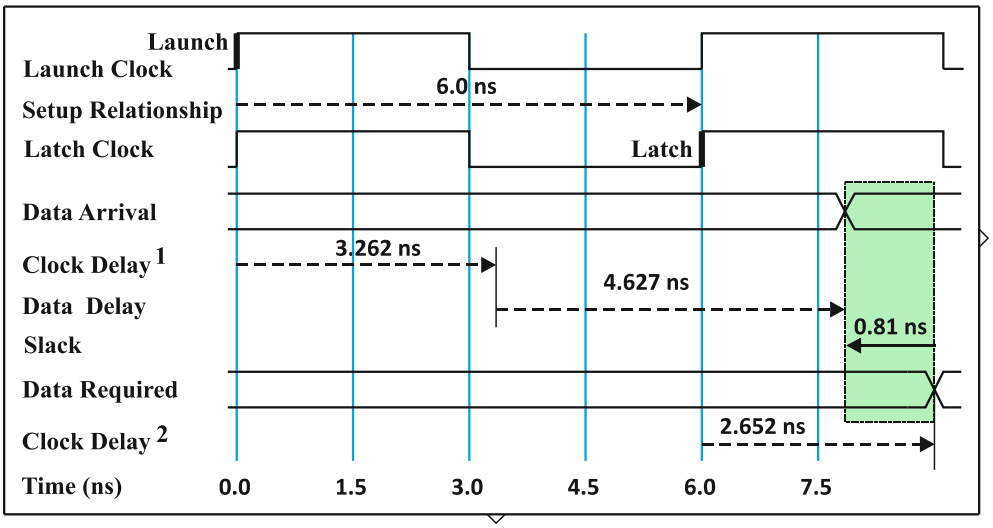
Сравнени са заеманите ресурси със и без използване на частични суми. Относително малкия дял на използваните adaptive logic modules (ALMs) се дължи на използването на вградените в FPGA DSP Multiplier Blocks и съчетаването в един ROM-базиран FSM на сигналите за управление на операционния блок и тегловните коефициенти.

2. Времеви анализ. Извършен е статичен тайминг анализ между свързаните регистрите на DSP операционния блок. Минималните стойности на достигнатия в отделните процеси Slack, са показани в таблица 4.6.

Таблица 4.6. Обобщени минимални стойности на Slack.

Process #	Registers Path	Clock Delay[1]	Clock Delay[2]	Data Delay	Slack
1	Gxy_Data to Gq	3.264	2.962	3.807	1.888
2	PS1weight to PS1q	3.262	2.652	4.627	0.81
3	PS2weight to PS2q	3.252	2.664	4.529	0.93
4	Gq to g	3.263	3.211	6.602	10.85

Установено е, че параметъра Slack достига критични стойности между регистрите PS1(2) weight to PS1(2)q (фиг. 4.11).



Фигура 4.11. Времеви анализ: PS1weight to PS1q.

Времето на Slack може да се подобри, с увеличаване на броя на етапите на конвейрите с които се реализират PS1(2) MAC DSP Block.

3. Сравнени са конвейрни и паралелни алгоритми с използване на частични суми. В таблица 4.7 е показана зависимостта на *OC* (4.26), *OP* (4.27), *TC* (4.30), *TR* (4.31), *FR* (4.28) от размера на изображението M × N.

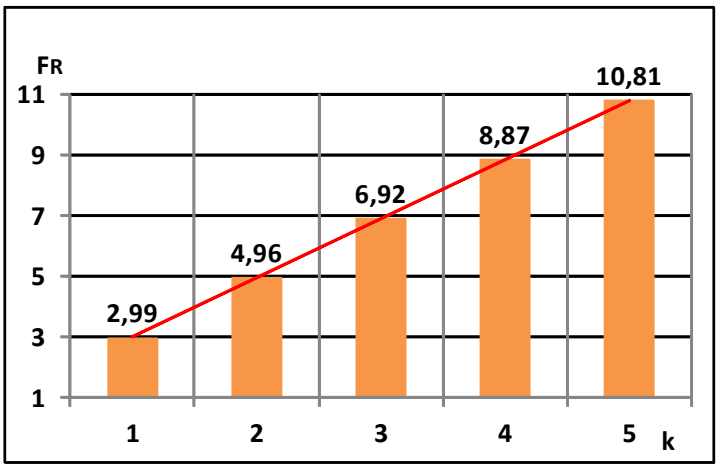
Таблица 4.7.

Зависимост на *OC*, *OP* , *TC* , *TR* , *FR* от размера на изображението M × N

M × N	640×480	1024×768	1600×1200
OC	2754730	7061770	16103529
OP	922554	2360826	5378634
TC [ms]	13,7736	35,3088	80,5176
TP [ms]	5,5353	14,1650	32,2718
FR	2,9860	2,9912	2.9940
TR	2,4883	2,4927	2,4950

Изследванията са извършен при k=1, tclk1 = 5ns (4.29) и tclk2 = 6ns (4.30). Независимо че за избраната фамилия FPGA, положителен Slack се достига при tclk2 > tclk1, фактора *TR* достига до 83% от *FR* .

В резултат от проведените изследвания е установено, че с нарастването на k се увеличава стойността на *FR*, като $FR \approx 2k+1$. Тенденцията на нарастване на *FR* е показана на фиг. 4.13.



Фиг. 4.13. Зависимост на *FR* от размера на прозореца *k:* M × N = 640×480

При увеличаване на размера на прозореца, се реализират 2(k+1) паралелни процеси, което води до увеличаване на *FR* .

НАУЧНО-ПРИЛОЖНИ ПРИНОСИ

1. Разработен е алгоритъм за генериране на тестови Гаусови ядра с целочислени коефициенти. Създаден е генератор с който се изчисляват коефициентите на ядрата и разрядността на операндите.
2. Създадени са правила за дефиниране на ROM-базирани крайни автомати (FSM) за генериране на Гаусови ядра и управление на DSP блокове. Предложен е синтаксис за описание на състоянията на автоматите. Разработен е транслатор за генериране на диаграма на състоянията на автоматите и изходен файл за инициализиране на ROM.
3. Предложени са два метода за изчисляване на нормализираната стойност на филтрирания пиксел. Разработена е методика за проектиране и изследване на DSP блокове за конвейрно изчисляване на LSF. Разработен е четириетапен DSP блок за конвейрно изчисляване на LSF.
4. Разработен е алгоритъм и DSP операционен блок за паралелно изчисляване на LSF с използване на частични суми. Доказано, че броя на операциите при алгоритмите с използване на частични суми се редуцират спрямо броя на операциите при алгоритмите без използване на частични суми до $2k+1$ пъти.

ПРИЛОЖНИ ПРИНОСИ

1. Разработена е методика за сравнителен анализ на софтуерните и хардуерните методи за филтриране на цифрови изображения с използването на FPGA-базирани платформи. Доказано е, че в зависимост от вида на процесора, хардуерните методи превъзхождат по бързодействие софтуерните от 21 пъти до 280 пъти.
2. Предложен е алгоритъм и програма за определяне на реципрочните стойностите на нормализиращия фактор NF^{-1}.

СПИСЪК НА ПУБЛИКАЦИИТЕ ПО ДИСЕРТАЦИОННИЯ ТРУД

1. Kanev I., *Software Tools for the Design of ROM Based FSM Implemented in FPGA*, Journal of the Technical University – Sofia Plovdiv branch, Bulgaria, "Fundamental Sciences and Applications" Vol. 20, pp 27-32 ISSN 1310-8271, 2014.

2. Kanev I., *Comparative Analysis of Software and Hardware FPGA-based Implementation of Weighted Average Linear Spatial Filters*, Journal of the Technical University – Sofia Plovdiv branch, Bulgaria "Fundamental Sciences and Applications" Vol. 21, pp 257-262 ISSN1310-8271, 2015.

3. Kanev I., *An Approach to FPGA- based Design of Weighted Average Linear Spatial Filters*, Journal of the Technical University – Sofia Plovdiv branch, Bulgaria "Fundamental Sciences and Applications" Vol. 21, pp 263-268, ISSN1310-8271, 2015.

4. Kanev I., *Comparison of Two Methods for Computing FPGA-based Weighted Average Linear Spatial Filters*, Proceedings of the 16th International Conference on Computer Systems and Technologies, CompSysTech'15, June 25-26, Dublin, Ireland, ACM International Conference Proceeding Series, Vol. 1008, ACM Inc., N.Y. USA, pages 276-283, ISBN 978-1-4503-3357-3, doi>10.1145/2812428.2812429, 2015.

5. Ivan Kanev, Petya Pavlova, Dimitre Kromichev, Jordan Genoff, *Generating 2D Gaussian Kernels for the Purpose of Studying FPGA-Based Linear Spatial Filters* , Fifth International Scientific Conference "Engineering, Techologies, and Systems" TECHSYS 2016, 26 - 28 May, Plovdiv, pp 111-118, ISSN 1310-8271,2016

6. Ivan Kanev, Petya Pavlova, *Optimized linear spatial filters implemented in FPGA*, IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), Volume 7, Issue 1, Ver. I (Jan. - Feb. 2017), PP 01-07, e-ISSN: 2319 – 4200, p-ISSN No. : 2319 – 4197, DOI: 10.9790/4200-0701010107, 2017. (Impact Factor 2.82)

SUMMARY

The dissertation is aimed at studying the possibilities of implementating Linear SpatialFiltering (LSF) on Field Programmable Gate Arrays (FPGAs). LSF is used to filter digital images for the purpose of blurring, noise reduction, detail enhancement, etc. A major problem in the implementation of LSF is that calculating the filtered pixels requires the utilization of a huge amount of addition, multiplication and division operations. The total of these operations increases proportionally to the image size's being enlarged, and can be of the order of tens or hundreds of millions.

The goal of this dissertation is to study the possibilities to reduce the number of operations in calculating the LSF with FPGA-based platforms through using pipelined and parallel algorithms, and the hardware solutions based on them.

To accomplish this goal, the following scientific and research tasks are fulfilled: a comparative analysis of hardware and software methods for filtering digital images using FPGA-based platforms is performed; proposed are algorithms for generaing test Gaussian kernels with integer coefficients; software tools for the designing of a ROM-based FSM are presented; LSF schemes based on pipelining are proposed, and alternative methods for the normalization of the filtered pixel are studied. Proposed are hybrid schemes for LSF methods utilizing pipelined and parallel computations.

The conducting of the studies results in:

- methodology for comparing software and hardware methods for filtering digital images using FPGA-based platforms. It is shown that, depending on the type of processor, hardware methods are from 21 to 280 times faster than the software methods;
- an algorithm for generating test Gaussian kernels with integer coefficients; a generator which calculates the kernels' coefficients and operands' width: determining rules for defining a ROM-based finite state machines (FSM) for generating Gaussian kernels and managing the DSP blocks;
- proposing syntax for describing the states of the machines; a translator for generating a state diagram of machines and output file to initialize the ROM; roposing two methods for calculating the normalized value of the filtered pixel;
- methodology for designing and testing the DSP blocks used in the pipelined computation of the LSF; an algorithm for determining the reciprocal value of the normalization factor;
- a four stage DSP block for the pipelined calculation of the LSF; an algorithm and DSP operating unit for the parallel calculation of LSF using partial sums;
- proving that the reduction in the number of operations in the algorithms using the partial sums compared to the number of operations in the algorithms which do not utilize partial sums can reach $2k + 1$ times.

The dissertation encompasses four chapters. It has a conclusion, contributions , future work, references and applications.

Key parts of the dissertation are published in six papers.