УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs
ИНТЕГРИСАНИ СИСТЕМ МЕНАЏМЕНТА СЕРТИФИКОВАН ОД:

ДОКУМЕНТАЦИЈА ЗА ПРЕДМЕТ: ПРОЈЕКТОВАЊЕ ЕЛЕКТРОНСКИХ УРЕЂАЈА НА СИСТЕМСКОМ НИВОУ

ТЕМА ПРОЈЕКТА:

Детекција слободних паркинг места применом функција за обраду слике

ПРОЈЕКАТ ИЗРАДИЛЕ:

Група 3: Ивана Гордић ЕЕ 23/2020
Ана Мокан ЕЕ 37/2020
Нина Антић ЕЕ 47/2020

Садржај

1. Спецификација

1.1 Уводни део

Тема пројекта је проналажење и маркирање слободних паркинг места на видеу надзорне камере паркинга. Такође, омогућено је праћење броја заузетог у односу на укупан број паркинг места. Рад система заснован је на примени различитих функција за обраду слике где су у средишту *Gaussian Blur* и *Adaptive Threshold* као методе обраде слике модификацијом вредности на нивоу пиксела.

1.2 Главне функције

1.2.1 *Gaussian Blur*

Ефекат замућења, *Image blurring* постиже се конволуцијом оригиналне слике са матрицом (кернелом) нископропусног филтра. Користи се за одстрањивање садржаја високе фреквенције, попут шума и ивица. *Gaussian blur* је један специфичан тип филтрирања слике где се за конволуцију користи *Gaussian kernel* који на основу подешавања одређених параметара (величине кернела, стандардне девијације - сигма) добија коефицијенте израчунате Гаусовом функцијом.

Коефицијенти кернела прате Гаусову расподелу, што значи да ће средиште квадратне матрице имати највећу тежину а крајеви мању. Подешавање вредности сигме одређују колико ће се постепено одвијати тај прелаз, а тиме и колико ће "јак" утицај имати суседни пиксели на онај који је тренутно у фокусу, док величина матрице одређује површину суседства које узимамо у обзир при обрађивању тренутног пиксела.

1.2.2 *Adaptive Threshold*

Функција која зависно од вредности прага компарације додељује пикселу одређену вредност - црно или бело. Код *Adaptive Thresholding*-а вредност прага (*threshold*) није унапред дефинисана већ се израчунава у односу на неки мањи регион слике коју обрађујемо и пикселе суседне оном у фокусу. За наше потребе коришћен је Гаусов тип израчунавања прага функције.

1.3 Опис алгоритма

Програм за одређивање броја слободних паркинг места се састоји из неколико делова.

Део програма написан је помоћу *python* програмског језика. Овај део узима фрејмове улазног видеа, фрејм, односно фотографију конвертује у црно-белу фотографију. Ова црно-бела фотографију представља улаз самог *C++* алгоритма.

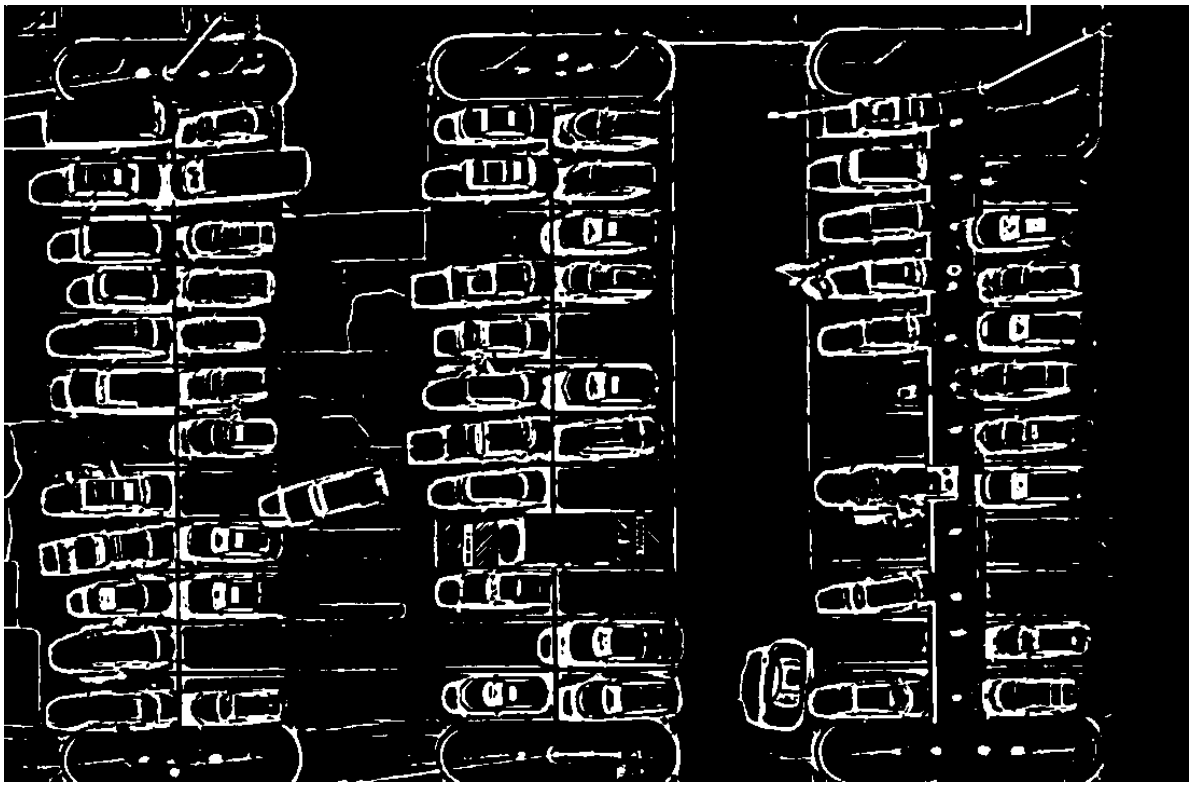
Део кода написан *python* програмским језиком такође прима и број слободних паркинг места (излаз *C++* алгоритма), те врши њихово обележавање на фотографији и приказује поменуту фотографију Овај део кода није био укључен у профајлирање перформанси.

Сам *C++* алгоритам за проналазак слободних паркинг места се састоји из следећих сегмената:

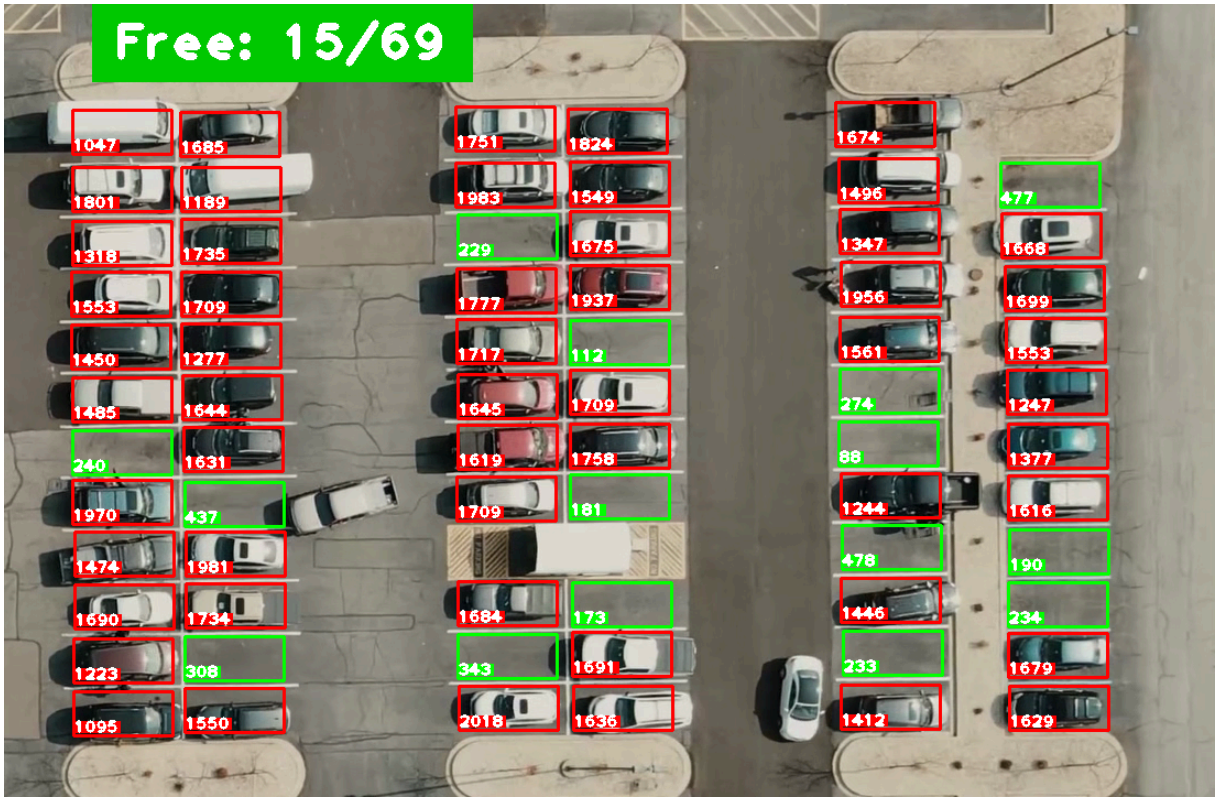
1. Стварање Гаусовог кернела за замућење фотографије уз помоћ Гаусове функције, врши се два пута: први пут са кернелом величине 3 пута 3, а затим са величином кернела од 15 пута 15, до 25 пута 25, пропорционално величини фотографије (функција ***gaussianBlur***).
2. Примена Гаусовог кернела на фотографију, односно замућење фотографије, (функција ***applyGaussian***), позиви ове функције врше се унутар функције ***gaussianBlur,***
3. Одређивање вредности прага у односу на који пиксели фотографије постају или црни или бели, врши се у односу на вредности пиксела у околини одређеној величином кернела при другом замућивању фотографије (функција ***adaptiveThreshold***); други позив ***gaussianBlur*** функције одиграва се унутар ***adaptiveThreshold*** функције,
4. Проширивање области фотографије на којима се налазе бели пиксели (функција ***dilateImage***),
5. Проналазак слободних паркинг места, (функција ***checkParkingSpace***), унутар ове функције се врши отварање текстуалног фајла са координатама паркинг места, издвајање делова фотографије означених тим координатама позивом функције ***cropImage***. Затим се врши пребројавање белих пиксела на издвојеним деловима фотографије функцијом ***countNonZero*** и поређење броја белих пиксела са прослеђеном граничном вредношћу како би се одредило да ли је паркинг место слободно. Ова гранична вредност представља потребан број белих пиксела у оквиру једног паркинг места, да би се оно сматрало заузетим. Она се утврђује експериментално и зависи од величине слике.

Функција ***checkParkingSpace*** враћа низ нула и јединица, где јединице означавају заузета паркинг места, а нуле слободна паркинг места. Овај низ представља излаз алгоритма.

Слика 1. Изглед фотографије након замућења Гаусовим кернелом 25x25



Слика 2. Изглед фотографије након примене прага у оквиру функције adaptiveThreshold

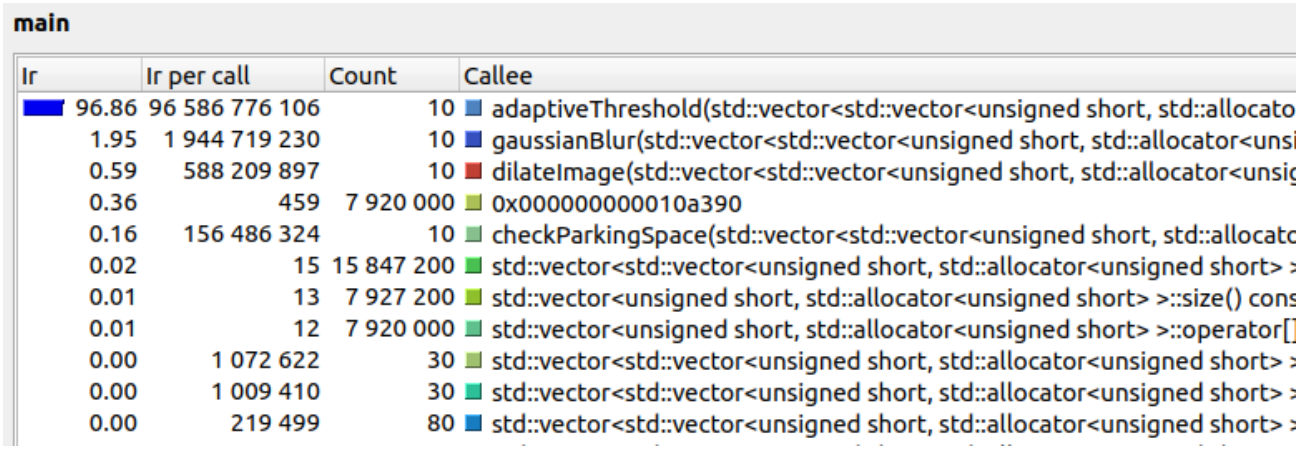


Слика 3. Приказ слободних паркинг места у python скрипти

2. Резултати профајлирања

При профајлирању у обзир је узет *C++* алгоритам, без дела програма писаног у *python* програмском језику. Алгоритам се састоји из етапа описаних у претходном одељку.

Исход профајлирања *C++* алгоритма, при чему је било прослеђено десет фотографија димензије 1100x720, је приказан испод:

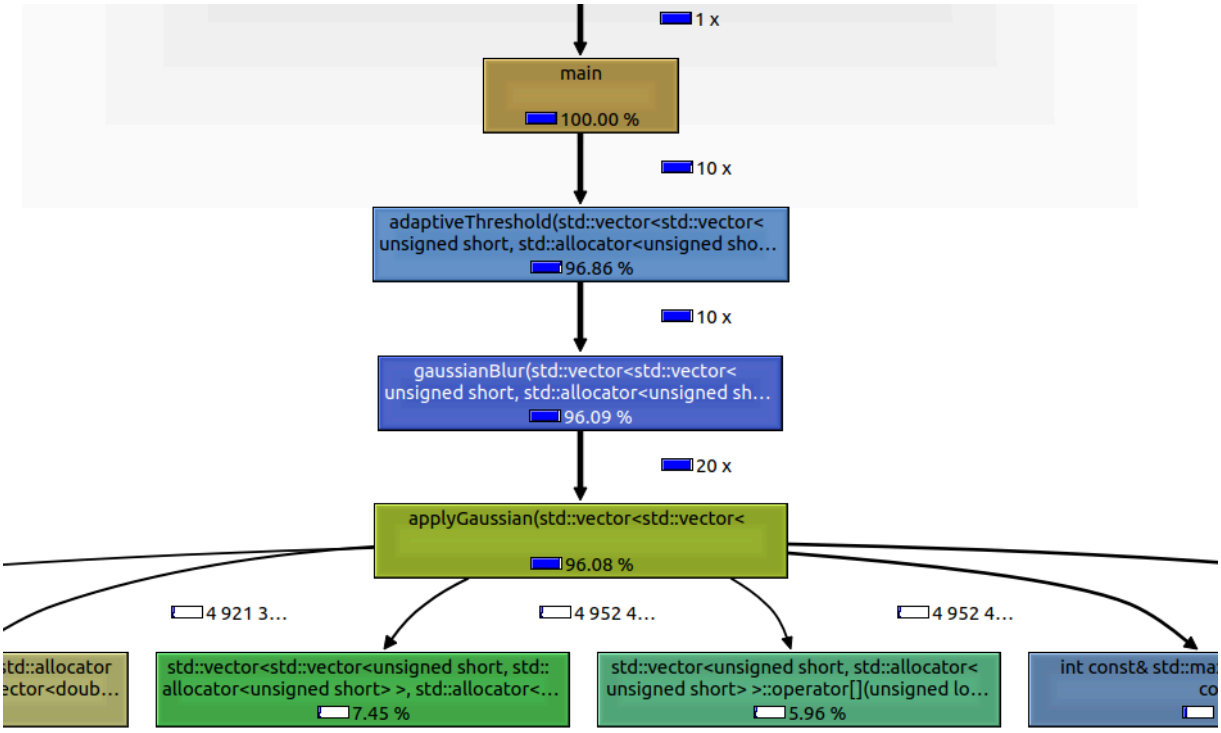


Слика 4. Резултати профајлирања, приказане су функције које се позивају унтар main функције

На *слици 4* приказани су резултати профајлирања. У питању су само функције које се позивају директно из *main*-а. Приликом извршавања програма, од свих позваних унутар *main* функције, најзначајнији део времена одлази на извршавање функције *adaptiveThreshold*. Међутим, разлог томе су функције које се позивају унутар ње: *gaussianBlur* и *applyGaussian.*

Функција *applyGaussian* се позива два пута и временски је захтевнија када ради над већим кернелом, што је случај када се позива унутар *gaussianBlur*-а у оквиру *adaptiveThreshold* функције што се види на графикону са *слике 5*.

Дакле, заправо је функција *applyGaussian* временски најзахтевнија узима чак 98.04% времена приликом извршавања алгоритма и због тога је одлучено да се управо она акцелира. Ово је приказано на *слици 6*, где се виде све фукције позване током рада алгоритма, а не само оне позване дирекно из *main* функције.



Слика 5. Приказ другог позива applyGaussian функције

main

Incl.	Self	Distance	Calling	Callee
98.04	0.00	1-2 (2)	20	gaussianBlur(std::vector<std::vector<unsigne
98.03	41.62	2-3 (3)	20	applyGaussian(std::vector<std::vector<unsigr
96.86	0.11	1	10	adaptiveThreshold(std::vector<std::vector<u
14.10	14.10	3-4 (4)	10 042 560 000	int const& std::max<int>(int const&, int const
14.10	14.10	3-4 (4)	10 042 560 000	int const& std::min<int>(int const&, int const
7.91	7.91	1-4 (4)	5 263 565 750	std::vector<std::vector<unsigned short, std::a
7.55	7.55	3-4 (4)	5 021 299 020	std::vector<std::vector<double, std::allocator
6.25	6.25	2-4 (4)	5 210 964 880	std::vector<unsigned short, std::allocator<un
6.04	6.04	3-4 (4)	5 021 299 020	std::vector<double, std::allocator<double> >:

Слика 6. Резултати профајлирања, приказане су најзахтевније функције које се позивају током рада читавог алгоритма

3. Битска анализа

Вредности две променљиве које фигуришу у *C++* коду су реални бројеви, те је било потребно дефинисати типове са фиксном тачком ради касније имплементације у хардверу, а у циљу избегавања типова са покретном тачком.

Покретањем алгоритма за различите вредности целобројних и разломљених делова вектора, уочено је да и мале промене вредности мењају резултат и да квалитет резултата стрмо опада. Пошто је алгоритам такве природе да и мале разлике у вредности пиксела могу дати добар број слободних паркинг места, у обзир су узете и саме вредности пиксела, а не само крајњи испис броја слободних паркинг места како би утврдили да је слика добро обрађена. Након овог тестирања дошло се до вредности које дају задовољавајуће резултате:

променљива	**укупан број бита**	**број бита за репрезентацију целобројног дела**
sumPixel	16	8
kernel	16	0

Табела 1. Формати променљивих у fixed point-у

Дефинисан је тип *typedef sc_dt::sc_ufix_fast num_t*. Овај тип користи се за обе реалне променљиве, а одабран је неозначени тип, јер поменуте променљиве никада немају негативну вредност.

Променљива *sumPixel* је типа *num_t*: *num_t sumPixel(16, 8, Q, O);*

Променљива *kernel* је матрица чији су елементи типа *num_t*, односно:

```
typedef std::deque<num_t> array_t;
typedef std::deque<array_t> matrix_t;
matrix_t kernel(kernel_size, array_t(kernel_size, num_t(16, 0, Q, O)));
```

Димензије слике које се могу обрадити алгоритмом су такве да фотографија треба да има хоризонталну димензију већу од вертикалне (*landscape* формат). Минимална дозвољена висина слике јесте 540 пиксела, а максимална 720 пиксела, док је минимална ширина слике 825, а максимална 1100 пиксела. Горње границе ових димензија одређене су у циљу уштеде ресурса на плочи. Доње границе су утврђене експериментално, односно, када се фотографија сувише смањи тешко је одредити гранични параметар који се прослеђује *checkParkingSpace* функцији, јер се смањује разлика у количини белих пиксела на слободним и заузетим паркинг местима.

Одређене су и минимална и максимална величина кернела: од 3x3 до 25x25. За први позив функције gaussianBlur идеалан је кернел 3x3, док је за њен други позив у оквиру функције *adaptiveThreshold* идеална величина кернела пропорционална укупном броју пиксела на фотографији (отприлике укупан број пиксела фотографије подељен са 31). Такође, величина кернела мора бити непаран број.

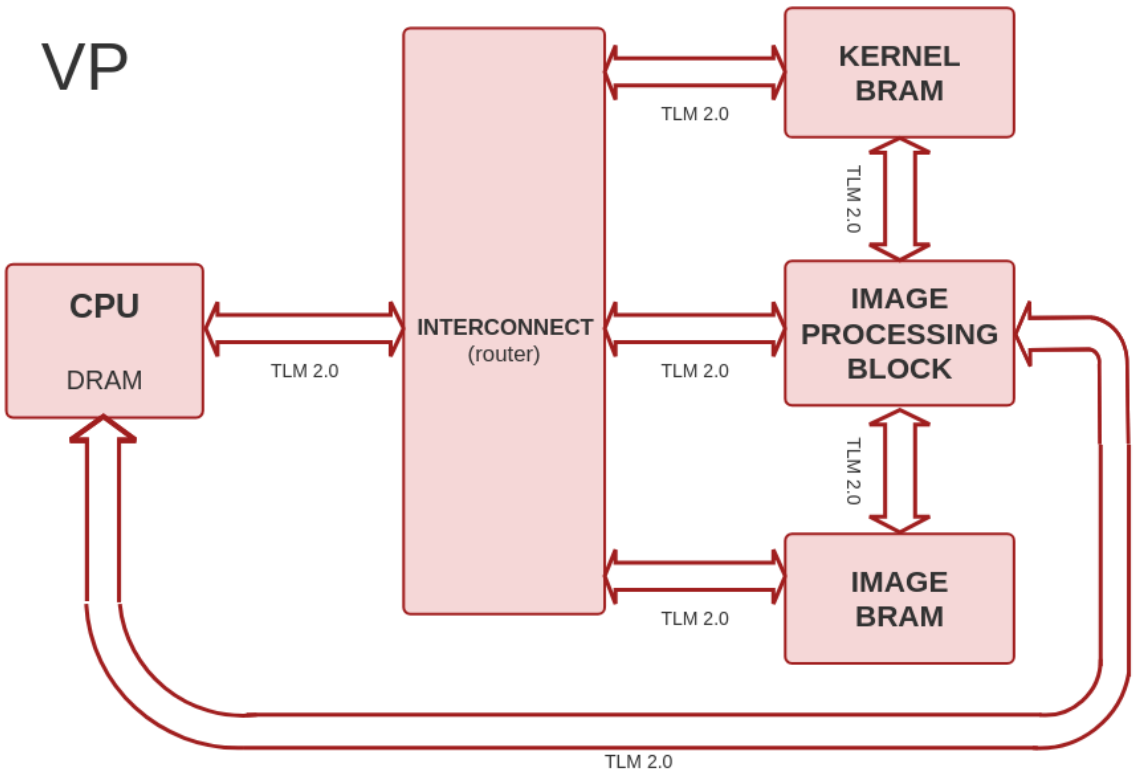
4. Виртуална платформа

На *слици 7* је приказана блок схема виртуалне платформе. Виртуална платформа је софтверски модел нашег алгоритма, који служи као замена хардверском прототипу у раним фазама развоја система. Виртуална платформа извршава функционалност нашег алгоритма, њен улаз јесте црно-бела фотографија, а излаз је низ јединица и нула који представља слободна паркинг места и на основу ког се исписује порука о броју слободних паркинг места.

Виртуална платформа се састоји из следећих компоненти:

- **SW** - модул који представља софтвер,
- **Router** (*interconnect*) - модул којим се врши меморијско мапирање,
- **IPB** (*Image Processing Block*) - модул који врши *applyGaussian* функцију, односно акцелератор критичног дела кода,
- **Kernel BRAM** - део меморије за складиштење нормализованих вредности Гаусовог кернела,
- **Img BRAM** - део меморије за складиштење дела слике.

Као посредник до *DRAM* меморије на *RTL* нивоу планиран је *DMA* блок.



Слика 7. Блок схема виртуалне платформе

4. 1 Функционисање виртуалне платформе

Софтвер врши учитавање фотографије, затим рачуна вредности Гаусовог кернела и смешта их у модул *Kernel BRAM*, такође и иницијализује модул *Img BRAM* почетним исечком фотографије. Затим се конфигурише регистар *IPB* модула који садржи вредност величине кернела. Након тога софтвер *start* битом покреће *IPB*, односно извршавање хардвера. *IPB* помоћу *ready* бита обавештава софтвер да ли је спреман за рад.

Када хардвер започне извршавање, обрађује се пиксел по пиксел фотографије, а за сваки пиксел се при рачунању користе и околни пиксели, где је околина дефинисана величином кернела. Из *Img BRAM*-а се узимају одговарајући пиксели из околине пиксела који се обрађује, а из *Kernel BRAM*-а одговарајућа вредност Гаусовог кернела. Променљиве *colIndex* и *rowIndex* представљају положај потребног пиксела на фотографији, али се оне морају преточити у адресу на којој се налази тај пиксел у оквиру *Img BRAM* модула. Две вредности прочитане из *BRAM*-ова се за сваки пиксел околине множе, а збир свих производа представља вредност замућеног тј. обрађеног пиксела. По завршетку обраде, резултат се уписује у софтвер.

Након обраде једне колоне слике, врши се додавање новог дела слике у модул *Img BRAM*.

Када хардвер заврши са радом, софтвер ће читањем *ready* бита установити да може да настави обраду фотографије. Софтвер ће затим извршити остале функције које нису убрзаване, попут *adaptiveThreshold,* и произвести резултат.

4. 2 *Image processing block* модул

Назив регистра	Ширина регистра	Опис
start	1 бит	вредост 1 - команда за почетак рада *IPB* модула вредност 0 - *IPB* модул је започео обраду или чека команду за почетак
ready	1 бит	вредност 1 - *IPB* модул је спреман да започне обраду вредност 0 - обрада је у току и *IPB* модул не може да започне нову обраду
kernel_size	5 бита	димензија странице матрице кернела

Табела 2. Регистарска мапа

На *слици 5* је приказан код из виртуалне платформе којим се моделује наш акцелатор. Овај модул имплементира конволуциони филтар на слику и који ефикасно обрађује пикселе ивице слике реплицирањем истих. Повезан је и на портове *Img* и *Kernel BRAM*-а одакле узима податке, а резултате уписује у *DRAM.* Такође, овај модул захтева допуну *Img BRAM*-а, односно чита *DRAM*.

```
for (sc_int<12> j = 0; j < width_hw; j++) {
	for (sc_int<12> i = 0; i < height_hw; i++){
		sumPixel = 0.0;

		for (sc_int<5> l = -halfSize; l <= halfSize; l++) {
			for (sc_int<5> k = -halfSize; k <= halfSize; k++) {
				if(i + k > 0){
					if(i + k < height_hw - 1){
						rowIndex = i + k;
					}
					else{
						rowIndex = height_hw - 1;
					}
				}
				else{
					rowIndex = 0;
				}
				if(j + l > 0){
					if(j + l < width_hw - 1){
						colIndex = j + l;
					}
					else{
						colIndex = width_hw - 1;
					}
				}
				else{
					colIndex = 0;
				}


			matrix_kernel = read_kernel_bram((l + halfSize)*kernel_size + k + halfSize);
			cache_size_mod =  (colIndex - req_cnt + req_cnt_mod) * height_hw + rowIndex;


		if(cache_size_mod >= cache_size){
			image = read_img_bram(cache_size_mod -cache_size);
			}
			else{
				image = read_img_bram(cache_size_mod);
			}


				sumPixel += matrix_kernel * image;

			}
		}

		if(i==719 && j < (width_hw - halfSize - 1) && j >= halfSize){
			dram_pos = kernel_size + req_cnt;
			bram_addr = req_cnt_mod*height_hw;
			for(sc_uint<11> n = 0; n < height_hw; n++){
				pixel = read_dram(dram_pos * height_hw + n);
				write_img_bram(bram_addr, pixel);
				bram_addr++;
			}
			req_cnt++;
			req_cnt_mod++;
			if(req_cnt_mod == kernel_size){ req_cnt_mod = 0; }
		}


		write_dram((j*height_hw + i), conv2int(sumPixel,16,8));

	}
}
```

Слика 8. Код за IPB модул

5. Перформансе система

Унутар *IPB (Image Processing Block)* модула често се јављају операције сабирања, множења и поређења. Претпоставка је да ће нека од комбинационих путања која укључује и до четири овакве операције бити критична комбинациона путања, која ће на каснијим нивоима одређивати фреквенцију. На основу претпостављене критичне путање процењена је фреквенција од 100 MHz, односно периода такта од 10 ns (моделована променљивом *DELAY)*.

При процени броја обрађених фотографија по секунди, претпоставка је била да ће се функционалност алгоритма у хардверу извршавати без проточне обраде, међутим, на овакав начин, предвиђене перформансе система доводе до успорења у односу на почетни алгоритам. Ова процена нас је навела на размишљање о могућности увођења проточне обраде на каснијим нивоима.

Када се претпостави проточна обрада са седам фаза, као и када се уочи на којим местима је могуће искористити *burst* могућност *AXI Full* интерфејса, добија се да је наш систем у могућности да обради једну фотографију за 5,181 секунди, односно да има пропусну моћ (*frame rate*) од 0,193 фотографија по секунди, за фотографију димензија 720x1100.

Додатно убрзање је могуће постићи развијањем неке од петљи, рецимо развијањем петље са итератором *i,* фактором k = 2. У том случају би једна фотографија била обрађена за 2,591 секунди.

Претходно је за изворни *C++* алгоритам, употребом *chrono* библиотеке за праћење времена, добијено време за обраду једне слике од 6,5 до 8 секунди.

6. Литература

[1] Вук Врањковић, *Пројектовање електронских уређаја на системском нивоу - рачунарске вежбе,* ФТН, Нови Сад 2020

[2] Документација *opencv python* библиотеке: https://docs.opencv.org (29.8.2024.)

[3] Документација *SystemC* библиотеке: https://systemc.org (1.9.2024.)

[4] Изворни *python* алгоритам: https://www.youtube.com/watch?v=caKnQlCMIYI&t=2001s (12.3.2024.)

[5] D. Gajski, S. Abdi, A. Gerstlauer, G. Schirner, *Embedded System Design Modeling, Synthesis and Verification,* Springer 2009

[6] G. Martin, B. Bailey, A. Piziali, *ESL Design and Verification - A Prescription for Electronic System Level Methodology*, Systems on Silicon 2007

[7] B. Bailey, G. Martin, *ESL Design and Verification - A Prescription for Electronic System Level Methodology*, Springer 2010

[8] Варијанта *C++* кода за Гаусово замућивање фотографије:

https://github.com/Zeljko9999/GaussianBlur (20.4.2024.)