Универзитет у Београду
Електротехнички факултет

Мастер рад

Издвајање неуронских сигнала из података добијених снимањем калцијумске флуоресценције

ментор:
доц. др Јелена Поповић–Божовић

студент:
Владимир Петровић
бр. индекса: 3029/2014

Београд, септембар 2015.

Сажетак

У овом раду је приказан алгоритам за аутоматско издвајање неуронских сигнала из података добијених снимањем калцијумске флуоресценције погодан за имплементацију на *FPGA* чипу. Алгоритам се базира на методу максимално стабилних ектремалних региона (*Maximally Stable Extremal Regions - MSER*), а хардверска имплементација алгоритма би могла да ради у реалном времену. Алгоритам је имплементиран у програмском језику *C* и програмском пакету *MATLAB*, али са посебним нагласком на каснију хардверску имплементацију.

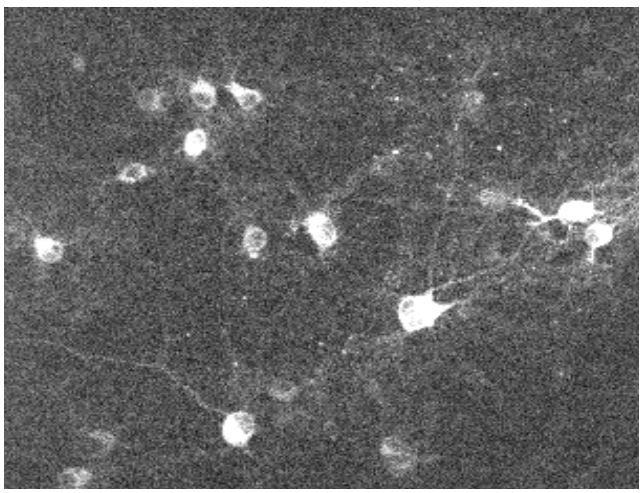
Кључне речи: Неурони, калцијумска флуоресценција, издвајање сигнала у реалном времену, максимално стабилни екстремални региони (*MSER*), *Union–Find* алгоритам

Садржај

Поглавље 1

Увод

Снимање калцијумске флуоресценције (*Calcium fluorescence imaging*) је врло популарна техника за симултано посматрање активности на стотине неуронских ћелија и то тако да се свака ћелија може посматрати независно. Велика предност ове технике у односу на електрофизиолошке технике је што се мождана активност може посматрати са много већом резолуцијом јер електрофизиолошке методе користе електроде за детектовање активности. Подаци добијени овом техником су видео снимци на којима различити интензитети пиксела у времену представљају сигнале које је потребно детектовати, а чијом се анализом може установити активност одређених делова мозга. Пример једне слике добијене снимањем калцијумске флуоресценције је дат на слици 1.1.



Слика 1.1: Део слике из видеа добијеног снимањем калцијумске флуоресценције неурона [1]

Проблеми са којима се ова техника суочава су низак однос сигнал-шум (SNR), неједнаки интензитети позадине, непрецизни интензитети са позиција неурона који се добијају услед јаког расејања светла, као и то што се неурони на сликама често преклапају [2].

До сада су развијене различите технике за екстракцију неуронских сигнала из снимака добијених снимањем флуоресценције. Већина се заснива на детекцији позиција неурона, а затим очитавања интензитета пиксела са одређених позиција. Неке једноставније технике се заснивају на ручном одређивању позиција [3], неке технике раде полу-аутоматски [4], а развијене су технике које аутоматски одређују позиције неурона и екстрахују сигнале из снимака [5], [6], [7], [8]. Међутим, све оне раде постпроцесирање. Такође, аутоматске технике су врло захтевне са становишта потребних ресурса (количине меморије и броја потребних операција) јер

су базиране на захтевним алгоритмима који не могу радити у реалном времену, као што су машинско учење [5], [8], сложене декомпозиције матрица [6], [7] или оптимизациони алгоритми [8].

За сада се калцијумска флуоресценција користи у истраживањима мождане активности мишева, али су кретање, а самим тим и многе остале активности животиње, ограничене јер се снимање ради на микроскопима или сензорима које је неопходно повезати са централним рачунаром. За истраживања у неуро наукама, ово је значајно ограничење јер су истраживања моторних активности лимитирана. Бежично слање података са сензора камере би омогућило флексибилније кретање, али оно захтева велики проток информација (за видео од 30 фрејма у секунди и резолуције 0,5 мегапиксела, потребан је најмање проток од 15 MB/s). Због тога би, у случају бежичног слања, морао да постоји неки вид компресије података.

Циљ овог рада је да се осмисли и истестира алгоритам за детекцију позиција неурона и екстракцију неуронских сигнала који захтева мало меморије и не превелики број рачунских операција, тако да може да ради у реалном времену. Животиња би тада могла да носи уређај који рачунару за приказ података бежично шаље само податке о позицијама неурона и интензитетима пиксела. Овај уређај би морао да буде мали по димензијама и да не троши много енергије јер батерија коју животиња треба да понесе не сме да буде тешка. Ограничена максимална потрошња енергије узрокује да је за ову примену једино могуће имплементирати цео алгоритам на *ASIC* чипу. Због тога је у овом раду, приликом развоја алгоритма, посебан акценат стављен на имплементацију самог алгоритма на *FPGA* чипу што би касније могло да доведе до реализације коначне идеје. Дат је предлог хардверског система који би могао да реализује алгоритам, а сам алгоритам је истестиран у софтверу.

Као додатак раду, описан је систем за снимање калцијумске флуоресценције и смештање снимка на *SD* картицу. Овај систем омогућава слободније кретање животиње јер се подаци о снимку чувају на *SD* картицу коју животиња носи са собом, а не на централном рачунару. Међутим, овакав систем и даље захтева постпроцесирање, па је главни део овог рада горепоменути алгоритам за детекцију неуронских сигнала.

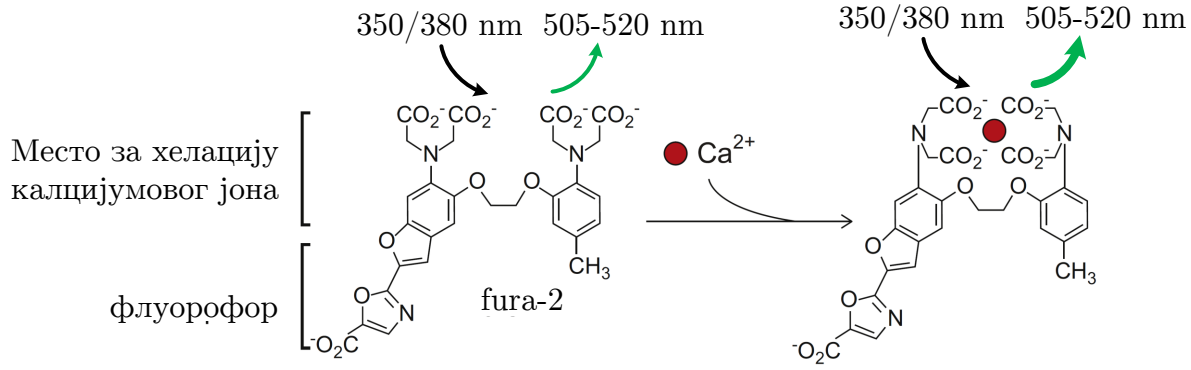
У даљем тексту овог увода објашњен је настанак калцијумске флуоресценције и поступак снимања, а затим је дат преглед до сада развијених алгоритама за аутоматску идентификацију неуронских сигнала. На крају увода је дат кратак опис идеје развијеног алгоритма за рад у реалном времену који је главни део овог рада.

1.1 Снимање калцијумске флуоресценције

Калцијумови јони имају важну улогу у физиологији и биохемији организма и ћелије. Њихова улога је кључна за путеве преноса сигнала. У биологији, пренос сигнала је поступак којим се конвертују механички или хемијски ћелијски стимулуси у одређени ћелијски одговор. Калцијум у том поступку преноси сигнале од рецептора на ћелијској

површини до циљних молекула унутар ћелије, у цитоплазми или у језгру. У неактивном стању, концентрација Ca^{2+} јона у цитоплазми ћелије је уобичајено мала (10–100 $nmol/dm^3$), а већина јона се чува у неким органелама (ендоплазматични ретикулум, митохондрије). Разне активности ћелије могу довести до преласка калцијумових јона из органела у цитоплазму и даље у међућелијски простор. Нпр. код неурона, приликом електричне активности концентрација јона калцијума у цитоплазми расте 10 до 100 пута у односу на неактивно стање [9], [10].

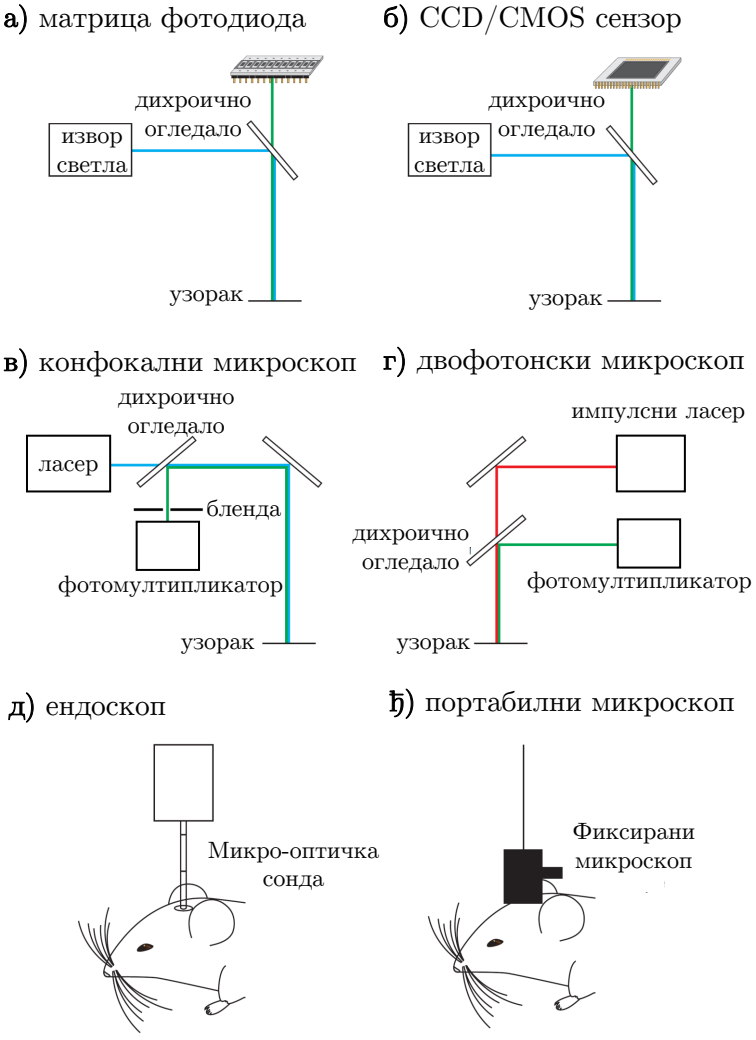
Значај Ca^{2+} јона у истраживањима је јако велики јер се различите активности ћелија могу видети као промена концентрације јона која се даље може детектовати снимањем калцијумске флуоресценције. Сама флуоресценција настаје услед дејства светлости на одређене молекуле (индикаторе) које је неопходно убризгати у ћелије. Када упадни (ексцитациони) фотон слободном електрону из индикатора преда довољно енергије да електрон пређе на виши енергетски ниво, приликом враћања електрона са вишег на нижи енергетски ниво, емитује се фотон који је често различите таласне дужине од ексцитационог фотона. Флуоресценција се поспешује везивањем јона калцијума за молекуле индикатора (поступак хелације), па се на тај начин могу приметити повећане концентрације калцијумових јона. Постоји неколико различитих метода за снимање калцијумске флуоресценције, а број различитих индикатора који се користе за изазивање флуоресценције је велики. На слици 1.2 је приказан настанак флуоресценције коришћењем *fura*-2 синтетичког индикатора под дејством ултраљубичастог зрачења. Ако је концентрација калцијумових јона велика, онда ће и интензитет флуоресценције бити велики. Њеним снимањем могу се детектовати нервни сигнали јер се већи интензитет флуоресценције на снимку види као значајно повећање вредности одређених пиксела [10].



Слика 1.2: Стварање флуоресценције под дејством спољашње светлости и везивање Ca^{2+} јона за индикатор [10]

Поред различитих индикатора који се користе за изазивање калцијумске флуоресценције, постоји неколико различитих начина за снимање флуоресценције. Најчешће се уређај за снимање састоји од фото-осетљивог сензора постављеног на микроскоп чији се светлосни путеви комбинују са зрацима светлосног извора који служи за екцитацију електрона индикатора. На слици 1.3 су приказани неки уређаји за снимање калцијумске

флуоресценције који се често користе. На слици 1.3.а је приказан уређај за снимање калцијумске флуоресценције који за сензор има матрицу фотодиода, где једна диода представља један пиксел. Наравно, ове сензоре су касније заменили *CCD* (*Charched Coupled Detector*) (слика 1.3.б) и *CMOS* сензори. Најчешће се за истовремено снимање и осветљење површине која се снима користе дихроична огледала која пропуштају неке таласне дужине, а друге одбијају. Светлосни извор емитује ексцитациону светлост једне таласне дужине која се одбија од огледала, а емисиона светлост је друге таласне дужине која пролази кроз дихроично огледало [10].



Слика 1.3: Различити уређаји за снимање калцијумске флуоресценције [10]

За снимање флуоресценције на дубљим локацијама у ткиву мозга или кичмене мождине се чешће користе конфокална и двофотонска микроскопија (слика 1.3. в и 1.3.г). У оба случаја светлосни извор је ласер. Фотосензори код конфокалне микроскопије могу бити фотомултипликатори или у новије време лавинске фотодиоде, али и *CCD* и *CMOS* сензори. Главна предност конфокалне микроскопије у односу на обичну микроскопију је што се у предњој жижној равни оптичког система постави бленда која пропушта само оне зраке који крећу из задње жижне равни објектива, тј. само светлосне зраке оних неурона који су у фокусу, чиме се побољшава контраст и оптичка резолуција. Међутим, изражено је расејање фотона на слојевима ткива које је испред задње жижне равни. Овај проблем је значајно смањен открићем двофотонске микроскопије.

Код двофотонске микроскопије се за светлосни извор користи светлост блиска инфрацрвеном делу спектра, али су за емисију једног флуоресцентног фотона потребна два ексцитациона фотона. Да би се десила флуоресценција и емитовао фотон веће енергије, потребно је да пар екцитационих фотона побуди један исти електрон у временском периоду

од око једне фемтосекунде, па је због тога вероватноћа за успешну ексцитацију мала. Стога се за светлосни извор користе импулсни ласери који могу да емитују већи број фотона, али у кратким временским интервалима. Вероватноћа успешне ексцитације опада квадратно са интензитетом светла, па је због тога највећи број успешних ексцитација у самом фокусу јер је ту интензитет светла највећи. Како је највећи број фотона насталих флуоресценцијом настао у жижној равни објектива није потребно постављати бленду у предњој жижној равни. Предност двофотонске микроскопије се огледа и у томе што су уобичајене таласне дужине ексцитационе светлости блиске инфрацрвеном делу спектра, па је пролазак зрака кроз ткиво лакши него код видљиве светлости коришћене у конфокалној микроскопији. Разлог за то је што су веће таласне дужине мање подложне расејању и мање апсорбоване од стране природних хромофора који се налазе у мозгу. Двофотонска микроскопија даје најјасније снимке калцијумске флуоресценције и данас се најчешће користи у анализи неуронске активности [10].

Све претходне методе су захтевале да глава животиње буде фиксирана. Неки нови приступи омогућавају кретање животиње тако што се на главу фиксирају оптички елементи и сензор, а онда се оптичким влакном подаци шаљу хардверу (и софтверу) за снимање видеа (слика 1.3.д и 1.3.ђ) [10].

1.2 Алгоритми за аутоматску детекцију неуронских сигнала

Као што је већ речено, све до сада развијене технике које аутоматски одређују позиције неурона и екстрахују сигнале из снимака раде постпроцесирање и базиране су на рачунски и меморијски захтевним алгоритмима који не могу радити у реалном времену.

Први алгоритам [6] који је омогућио аутоматску детекцију неуронских сигнала се заснива на анализи независних компонената (*ICA – Independent Component Analysis*). Алгоритам користи информације о томе да су сигнали који се генеришу по природи ретки сигнали (*sparse signals*), тј. од значаја су само скокови који се дешавају у ретким тренуцима времена (*temporal sparseness*), као и да су неурони ретки на слици ако су димензије неурона мале у поређењу са димензијама слике (*spatial sparseness*). Такође, важна претпоставка је да су позиције различитих неурона и сигнали које они генеришу статистички независни један од другог. Улазни подаци су сви пиксели сваког фрејма у снимку. Број димензија ових података је превелики и неопходно је најпре урадити редукцију димензија и она се ради анализом главних компонената (*PCA – Principal Component Analysis*). Метод анализе главних компонената даје линеарну трансформацију података чији су резултат базисни вектори, главне компоненте (*principal components*), који су распоређени по рангу одређеним варијансом података дуж сваког од вектора. Одбацивањем компоненти вишег реда добија се значајно мања димензионалност података и смањење нивоа шума. Преостале компоненте, тј. пројекције података на преостале базисне векторе представљају улаз за анализу независних компонената. Метод анализе

независних компонената претпоставља да су подаци који потичу од различитих неурона статистички независни. Метод налази мешајућу матрицу (*mixing matrix*) чија инверзна матрица множи вектор главних компонената и даје независне компоненте. Независне компоненте су више просторно локализоване од главних и стога се може сматрати да свака одговара независној ћелији. За метод анализе независних компонената је важно да компоненте нису гаусовске. Због тога се мешајућа матрица налази максимизацијом описне функције (*objective function*) која се дефинише тако да математички опише несиметричност компонената и то као линеарна комбинација критеријума несиметричности у просторном и у временском домену. Овај метод је инспирисао многе касније методе који користе оптимизационе алгоритме за детекцију позиција и сигнала неурона, само што су критеријуми за оптимизацију другачији.

Други приступ који се јавља у литератури [8], [11] је посебан вид машинског учења чији је циљ пронаћи декомпозицију $\mathbf{X} \approx \mathbf{D}\mathbf{U}^{\mathrm{T}}$, где је $\mathbf{X}$ матрица која у једној врсти има све пикселе једног фрејма, а има онолико врста колико је фрејмова у снимку, дакле садржи све податке из видео снимка. Матрица $\mathbf{D}$ се назива речником (одакле је и назив за ово машинско учење *dictionary learning*) у чијим се колонама налазе базисне функције које представљају просторни опис сваке од ћелија. Матрица $\mathbf{U}$ је испуњена коефицијентима који представљају временске податке о појави сваке од базисних функција у снимку у току времена. Декомпозиција се тражи неким од оптимизационих алгоритама, при чему је једно ограничење засновано на томе да су сигнали у времену по природи ретки сигнали, па је важно да матрица $\mathbf{U}$ буде таква да је мали број коефицијената различит од нуле. Налажењем матрице речника, одређене су позиције неурона.

У литератури постоји још неколико приступа као што је просторно-временска деконволуција [12] или неки други видови машинског учења [5]. Сви они су, попут описана прва два, рачунски врло сложени и захтевају много меморије јер сви раде са целим снимком, што је понекад и 1 GB података.

У овом раду је представљен другачији приступ. Свака од слика из видеа се процесира независно што не захтева учитавање целог снимка у меморију, већ се обрађује један по један фрејм или чак делови једног фрејма. Идеја је да се пиксели који су од интереса на слици детектују коришћењем неког од алгоритама за рачунарску визију. Због природе проблема погодан је један од метода за детекцију региона на слици (*blob detection*). Неки стандардни методи за детекцију региона су базирани на *LoG* оператору (*Laplacian of Gaussian*), разлици гаусијана (*Difference of Gaussians*) и детерминанти хесијана (*Determinant of Hessian*). За овај рад је најподеснији метод максимално стабилних екстремалних региона (*Maximally Stable Extremal Regions - MSER*) [13] који је базиран на *union-find* алгоритму [14], [15]. Овај метод је, уз мале модификације, искоришћен за детекцију региона на слици који су кандидати за регионе који одговарају неуронима. Крајњи циљ је хардверска имплементација развијеног алгоритма и то за рад у реалном времену. *MSER* алгоритам је раније имплементиран у хардверу и то тако да ради у реалном времену (25

fps), али за слике максималних димензија 350×350 пиксела [16]. Поред брзине која опада са повећањем димензија слике, количина меморије потребна за реализацију алгоритма расте. Због тога је предложена реализација алгоритма за детекцију неурона на слици која користи *MSER* алгоритам, али на мањим блоковима. Свака слика је подељена на блокове, а сваки блок је обрађен коришћењем *MSER* алгоритма. Излазни подаци су затим додатно процесирани како би се добили региони који припадају неуронима. Када су познате позиције неурона, екстракција сигнала на основу интензитета пиксела са слике не представља проблем.

У даљем току рада детаљно је описан *MSER* алгоритам, а затим и детаљи софтверске имплементације алгоритма која се уз мале модификације може искористити за хардверску имплементацију. Затим је описан предлог система за детекцију неурона који се састоји од блокова за процесирање делова слике и од блокова за процесирање детектованих региона на слици. На крају су приказани резултати које даје развијени алгоритам и дато је кратко поређење са алгоритмом који ради на принципу анализе независних компонената.

Поглавље 2

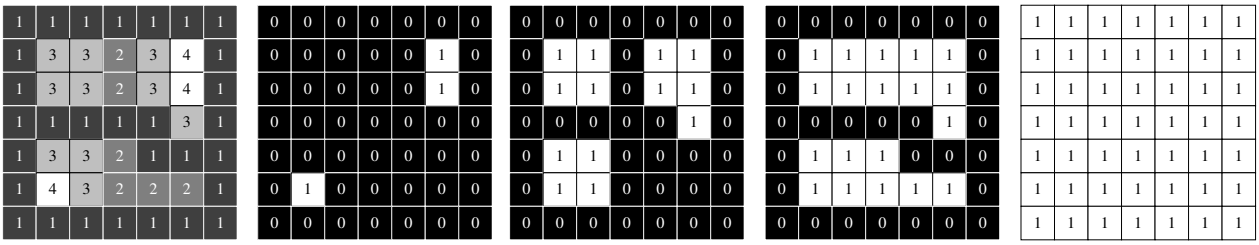
Метод максимално стабилних екстремалних региона – *MSER*

У овом поглављу је представљен један тип објеката на слици користан за различите апликације у рачунарској визији – максимално стабилни екстремални региони (*Maximally Stable Extremal Regions*, у даљем тексту *MSER* региони), као и алгоритам за њихово налажење. *MSER* региони су први пут представљени 2002. године [13] где су коришћени за препознавање објеката на слици (*image matching*).

2.1 Дефиниција *MSER* региона

Најпре ће бити описан појам екстремалних региона на простом примеру слике која има само 4 нивоа (слика 2.1.а). Најчешће, црно-беле слике имају 256 нивоа (0–255) и такве слике су коришћене у овом раду, али ради једноставности објашњења, први пример има само 4 нивоа (1, 2, 3 и 4). Екстремални региони могу бити светли или тамни. Светли екстремални региони су повезане области на слици чији пиксели нису мањи од неке унапред задате вредности – прага (t – *threshold*). Ови региони за различите вредности прага t су обележени јединицама на сликама 2.1.б–2.1.д. Тамни екстремални региони се дефинишу обрнуто, као повезане области чији су пиксели мањи од прага (нуле на сликама 2.1.б–2.1.д). У овом раду су коришћени једино светли екстремални региони и у даљем тексту ће се подразумевати да су сви екстремални региони од интереса светли. Величина и број екстремалних региона зависи од вредности прага t.

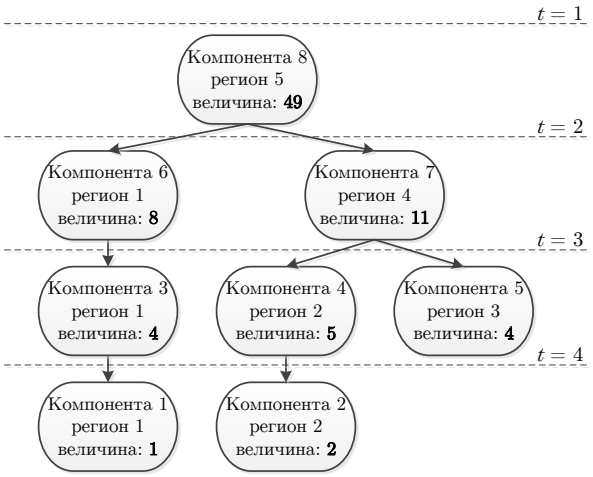
На слици 2.1 су приказани различити екстремални региони за различите вредности прага. Ако поставимо праг на $t = 5$ неће постојати ни један регион на слици јер сви пиксели имају вредност мању од 5. За вредност прага $t = 4$ јављају се два региона. За $t = 3$ појављује се још један регион, а величина претходна два је порасла. За $t = 2$ долази до спајања два од три региона, док се за $t = 1$ добија да је цела слика јединствен регион.



а) Полазна слика б) $t = 4$ в) $t = 3$ г) $t = 2$ д) $t = 1$

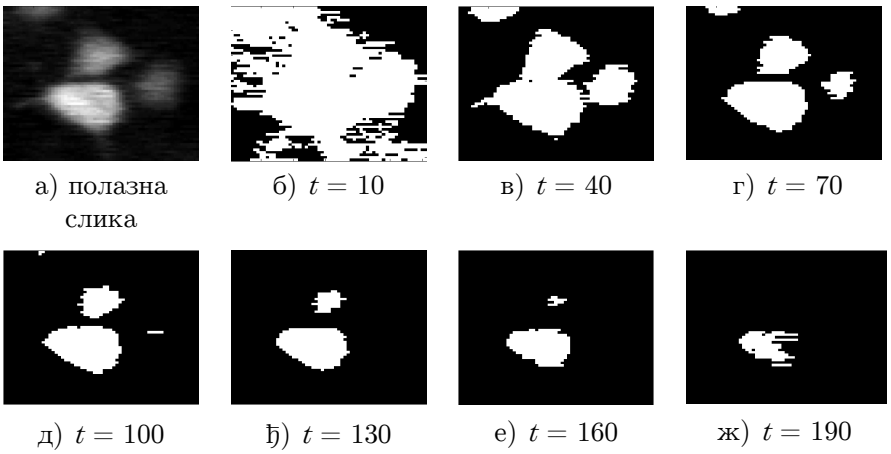
Слика 2.1: Оригинална слика са 4 нивоа (а) и повезани региони (беле површине на сликама) за различите вредности прага t (б–д) [17]

Оваквом постепеном анализом, ради лакшег праћења, може се креирати стабло чије су компоненте региони са величинама при свакој од вредности прага t. Стабло које одговара анализираној слици 2.1 је приказано на слици 2.2. Корен овог стабла је компонента која одговара региону који се добија при најмањој вредности прага t, зато што се сви мањи региони на крају стапају у њега. Листови стабла су региони који нису настали спајањем два мања региона.



Слика 2.2: Стабло које прати промене величина и спајање региона при примени различите вредности прага t за пример са слике 2.1

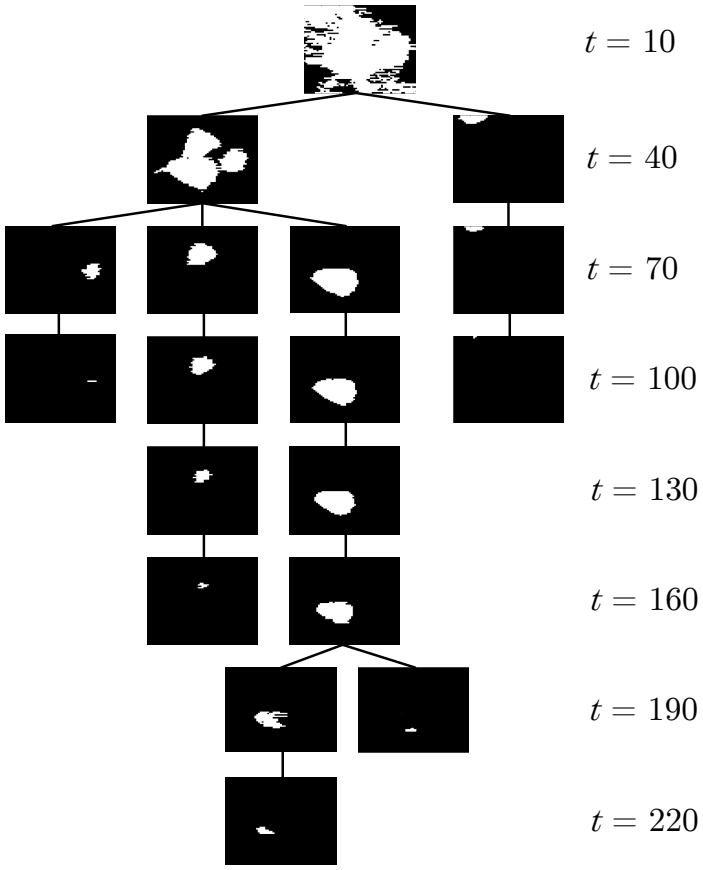
На слици 2.3.а је приказан део фрејма из једног од снимака који је коришћен за тестирање алгоритма. На слици су приказана 4 неурона, а на сликама 2.3.б–2.3.ж су приказани региони при различитим вредностима прага ($10 < t < 190$) за овај реалан случај.



а) полазна слика б) $t = 10$ в) $t = 40$ г) $t = 70$

д) $t = 100$ ђ) $t = 130$ е) $t = 160$ ж) $t = 190$

Слика 2.3: Део фрејма са 4 неурона (а) и региони за различите вредности прага t (б–ж)



Слика 2.4: Стабло које одговара слици 2.3

Као и у једноставнијем случају са 4 нивоа, може се добити стабло које одговара слици 2.3.а. На слици 2.4 је приказано то стабло. Међутим, оно приказује само промене у регионима за укупно 8 прагова. Потпуно стабло би се добило када би се анализирали региони и њихове величине за све могуће вредности прага (између 0 и 255).

Осим тога што су погодна за визуелно представљање региона за различите вредности прага, стабла су врло погодне структуре за програмирање. Због тога, најчешће све софтверске имплементације *MSER* алгоритма најпре формирају стабло региона, а затим оперишу са подацима из стабла. Само формирање стабла је главни део сваке имплементације *MSER* алгоритма и оно ће касније бити детаљно објашњено.

Сада, под претпоставком да постоје сви подаци о регионима за различите вредности прага, може се дефинисати појам максимално стабилног екстремалног региона.

MSER региони су, укратко речено, региони чија се величина најмање мења за различите вредности прага. За сваки регион (скуп пиксела) R_i, добијен при вредности прага $t = i$, најпре се дефинише варијациони фактор или фактор стабилности $q(i)$ као:

$$q(i) = \frac{|R_{i-\Delta}| - |R_{i+\Delta}|}{|R_i|},$$

где $|\cdot|$ представља кардиналност, тј. број пиксела одређеног региона. При томе је $R_{i+\Delta} \subset R_i \subset R_{i-\Delta}$, тј. $R_{i-\Delta}$ је регион R само добијен при прагу $i - \Delta$. Слично је $R_{i+\Delta}$ регион R добијен при прагу $i+\Delta$. Регион R_{i*} је максимално стабилан, ако $q(i)$ има локални минимум у $i*$. Параметар Δ је параметар који одређује опсег провере стабилности. Што је мање Δ, то ће бити више детектованих $MSER$ региона [13], [18].

Често се поред параметра Δ, као ограничења задају још неки параметри као што су максимална и минимална величина $MSER$ региона (*maximal size*, *minimal size*) и максимална варијација региона (*maximal variation* – q_{max}). Ова ограничења су уведена јер нису сви *MSER* региони једнако значајни. Рецимо, у неким применама нису важни региони који имају свега неколико пиксела или заузимају већи део површине слике. Такође,

могуће је да ће се за један регион при различитим вредностима прага појавити доста локалних минимума фактора стабилности, тј. појавиће се доста *MSER* региона који се налазе на само једној грани стабла, па се може међу њима одабрати најстабилнији регион или неколико најстабилнијих. Нпр. у овом раду је током тестирања параметар q_{max} мењан између 0,2 и 0,3 и одбацивани су сви *MSER* региони R_{i*} за које је $q(i*) > q_{max}$.

Као што је речено, формирање стабла компонената које представљају регионе за различите вредности прага t, најкомплекснији је део алгоритма. У оригиналном раду [13] и у многим другим имплементацијама и применама [18], [19], [20], за формирање стабла користи се *union-find* алгоритам [14], [15]. Овај алгоритам је коришћен и у овом раду и детаљно је објашњен у одељку који следи.

2.2 Формирање стабла за тражење *MSER* региона

У овом одељку је објашњен *union-find* алгоритам уз помоћ математичког формализма са освртом на конкретну примену у налажењу региона на слици. Ради лакшег објашњења, на неким местима је дат квази-код могуће имплементације. Детаљи имплементације *MSER* алгоритма коришћене у овом раду објашњени су у следећем поглављу. Важно је напоменути да је имплементација *MSER* алгоритма коришћена у овом раду специфична по много детаља и не мора одговарати опису из овог поглавља. Циљ је да се најпре објасни суштина алгоритма, а касније детаљи имплементације.

2.2.1 Дисјунктни скупови и *union-find* алгоритам

Нека је дата колекција Q дисјунктних подскупова скупа I. Скуп I је у конкретном случају скуп свих пиксела улазне слике. Сваки скуп $X \subset I$ из Q је представљен јединственим елементом $x \in X$ који се назива канонички елемент (*canonical element*). У овом раду елемент x је структура података која се односи на један пиксел слике. Ако су x и y различити елементи скупа I, могу се дефинисати три операције које омогућавају рад са колекцијом Q, њеним елементима који су подскупови скупа I и елементима тих подскупова:

- **MakeSet**(x): додаје скуп $\{x\}$ у колекцију Q, под условом да елемент x већ не припада неком скупу из Q. При томе је елемент x аутоматски постављен за канонички елемент додатог новог скупа.
- **Find**(x): као резултат даје канонички елемент скупа из Q коме припада x.
- **Union**(x, y): ако су X и Y два скупа из Q чији су канонички елементи редом x и y ($x \neq y$), оба скупа се уклањају из Q , а њихова унија $Z = X \cup Y$ се додаје у колекцију Q, при чему се бира нови канонички елемент за Z.

Алгоритам који може да изврши било коју секвенцу горенаведених операција са квазилинеарном сложеношћу представљен је први пут 1975. године [14] и назива се *union-find* алгоритам. Тачније, најлошија могућа сложеност износи $\Theta(m\alpha(m,n))$, где је m број операција, а n број елемената. Функција α је инверзна Акерманова функција [21] која врло споро расте са порастом аргумената и за све практичне примене важи да је увек $\alpha(m,n) < 4$ [17], [13].

Имплементација алгоритма је дата у табелама 2.1–2.3. Сваки скуп колекције Q је представљен као стабло код кога је канонички елемент корен стабла. Сваком елементу x се додељују две вредности – родитељ ($\texttt{Parent}(x)$) и ранг ($\texttt{Rank}(x)$). Ранг елемента представља дубину подстабла чији је корен тај елемент.

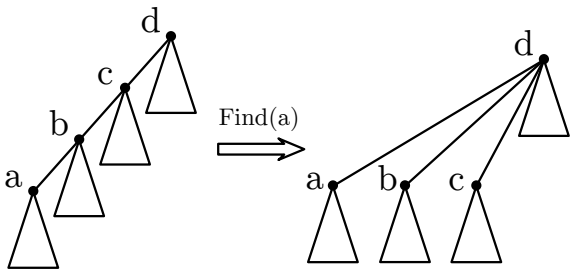
Како су канонички елементи корени стабла, $\texttt{Find}(x)$ треба да пронађе корен стабла у коме се налази x. Да би се дошло до корена стабла, најпре је потребно испитати да ли је текући елемент корен стабла. Ако није, треба испитати да ли је родитељ датог елемента корен стабла и тако редом све док се не појави елемент који је сам себи родитељ, тј. који је корен стабла. Очигледно, операција је рекурзивна.

Procedure `MakeSet`(element x)
$\texttt{Parent}(x) = x$; $\texttt{Rank}(x) = 0$;

Табела 2.1: Псеудо код имплементације операције `MakeSet` [17]

Function element `Find`(element x)
if $\big(\texttt{Parent}(x) \neq x\big)$ **then** $\texttt{Parent}(x) = \texttt{Find}\big(\texttt{Parent}(x)\big)$;
return $\texttt{Parent}(x)$;

Табела 2.2: Псеудо код имплементације операције `Find` [17]



Слика 2.5: Пример компресије путање при позивању операције $\texttt{Find}(a)$

Прва од две кључне идеје која значајно смањује сложеност алгоритма је тзв. техника компресије путање (*path-compression*) која смањује сложеност `Find` операције. Техника компресије путање се састоји у томе да се приликом пењања кроз стабло свим елементима који се налазе на путањи до корена, корен стабла постави за родитеља. Овим се мења структура стабла и то тако да је потребан мањи број испитивања приликом следећег позивања `Find` операције за велики број елемената, а често се и смањује дубина стабла. На слици 2.5 приказан је пример компресије путање при позивању операције `Find` за један елемент који

је далеко од корена стабла. Троуглови представљају подстабла чији су корени елементи a, b, c и d.

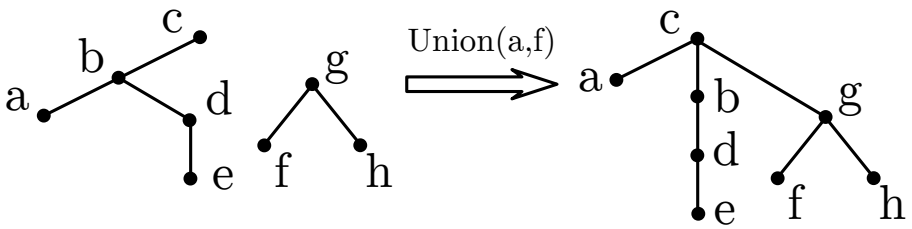
Друга важна техника која смањује сложеност алгоритма је везана за `Union` операцију. Наиме, приликом спајања два различита стабла, увек се за корен бира корен оног стабла чији је ранг већи (*union by rank*). У табели 2.3 је приказана имплементација ове операције. Дубина стабла једино расте за 1 ако је ранг првог стабла једнак рангу другог јер једно од та два стабла мора постати подстабло другог [17].

Function element `Union`(element x, element y)

```
if (Rank(Find(x)) > Rank(Find(y)))
    then
        Parent(Find(y)) = Find(x);
        return Find(x);
    else
        if (Rank(Find(x)) == Rank(Find(y)))
            then Rank(Find(y)) = Rank(Find(y)) + 1;
        Parent(Find(x)) = Find(y);
        return Find(y);
```

Табела 2.3: Псеудо код имплементације операције *Union* [17]

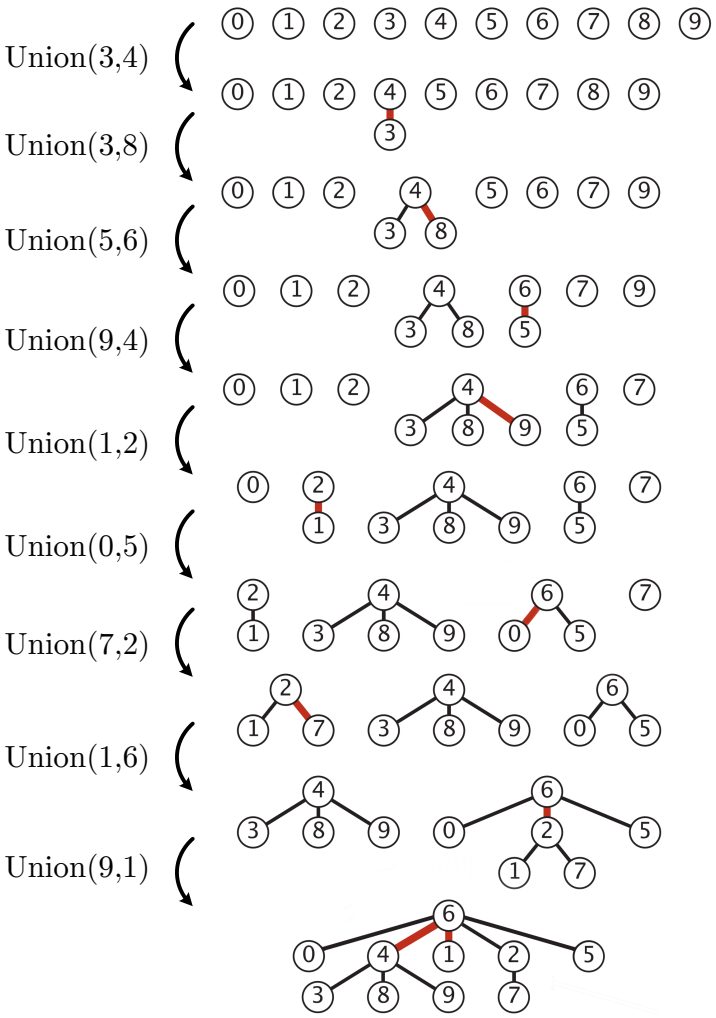
На слици 2.6 је дат пример спајања два стабла. Стабло чији је корен елемент c има дубину 3, па њему треба додати друго стабло чија је дубина 1. Приликом позива операције `Union`(a,f), позивају се и `Find` операције за оба елемента. У већем стаблу долази до компресије путање – елементу a је промењен родитељ, али дубина стабла је остала иста.



Слика 2.6: Спајање два стабла позивом операције `Union` за два елемента из тих стабала

На слици 2.7 је приказан пример једне секвенце `Union` операција која формира стабло од 10 почетних елемената. Црвеном бојом су означене све нове везе које су настале после извршавања претходне `Union` операције. Пример је уз мале модификације преузет из [15]. Интересантан је позив последње `Union` операције где поред спајања два подстабла долази и до компресије путање (елемент 1).

Важно је напоменути да постоје различите варијанте имплементације `Union` операције [15]. Често се уместо поређења по рангу користи поређење по величини стабала која се спајају, тј. по броју елемената у стаблима. Тај приступ је коришћен у овом раду.



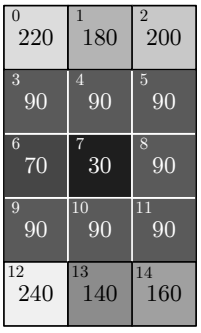
Слика 2.7: Пример извршавања једне секвенце `Union` наредби [15]

У овом раду, елементи стабала су пиксели на слици. Раздвојени региони на слици представљају одвојена стабла добијена `Union` операцијама примењеним на различите пикселе. Ако се претпостави да су елементи у секвенци са слике 2.7 неки пиксели онда је на почетку сваки пиксел засебан регион на слици. У претпоследњем кораку, на слици се издвајају два региона од којих се први састоји од пиксела 4, 3, 8 и 9, а други од пиксела 6, 0, 2, 5, 1 и 7. На крају ови региони се спајају у један. *Union-find* алгоритам омогућава ефикасно спајање више мањих региона у један већи што је врло важно јер се тако постепено може формирати стабло чије су компоненте региони на слици за различите вредности прага t, што је и крајњи циљ.

2.2.2 Креирање стабла уз помоћ *union-find* алгоритма

Поступак формирања стабла за тражење *MSER* региона је најлакше приказати на примеру. На слици 2.8 је приказан сегмент слике од 15 пиксела. Пример је уз мале модификације нивоа преузет из [17]. На њему су показане све ситуације које се могу јавити приликом креирања било ког стабла, па се његовим коришћењем не губи на општости.

Обрада креће од најсветлијих тачака на слици. Најпре је потребно направити редослед обраде пиксела. Редослед може бити низ позиција пиксела који је сортиран према вредности пиксела на тим позицијама у опадајућем поретку.

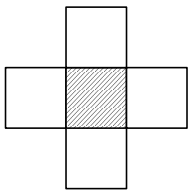


Слика 2.8: Сегмент слике на који се примењује *union-find* алгоритам. Бројеви у горњем левом углу су позиције пиксела, а бројеви у средини вредности пиксела.

Ако се траже тамни екстремални региони, позиције треба сортирати у растућем поретку.

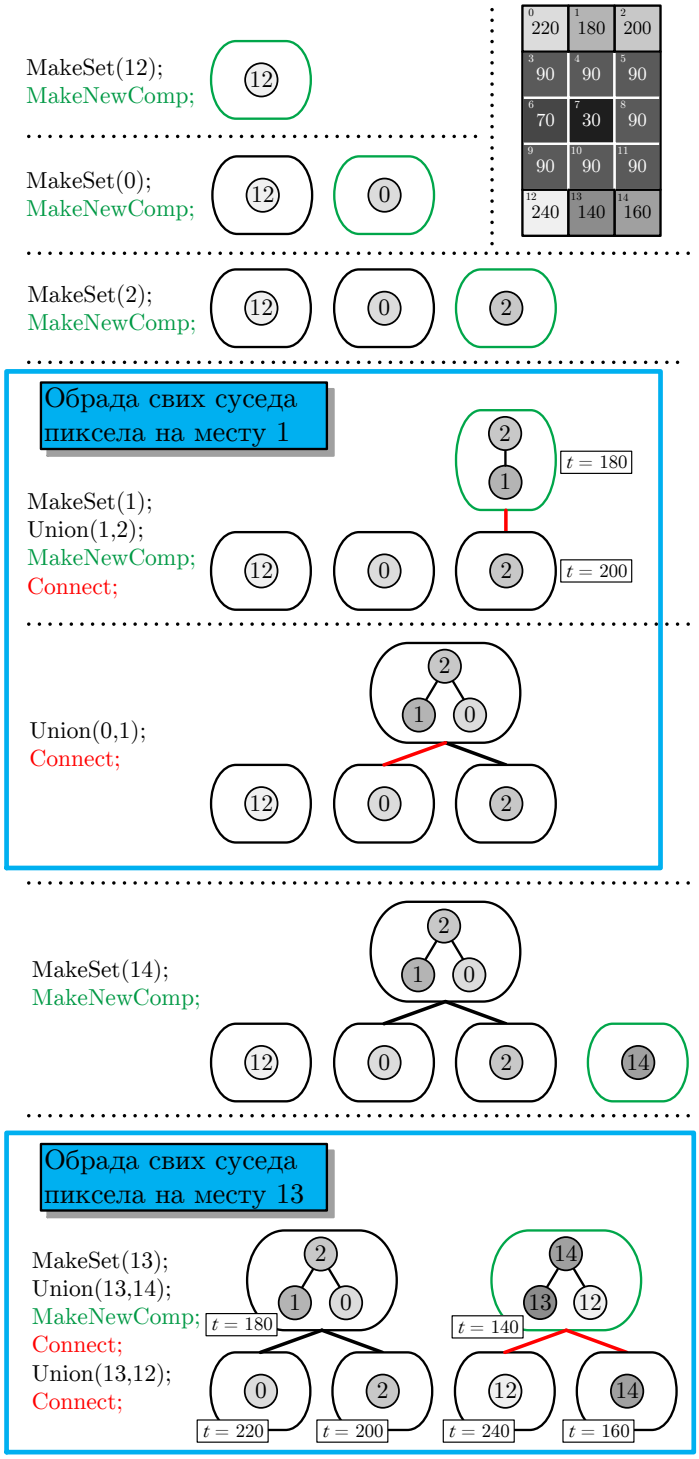
У примеру са слике 2.8 најсветлији пиксел је пиксел 12 чија је вредност 240. У току рада, води се евиденција који су све пиксели обрађени. Ако пиксел није већ обрађен, потребно је додати га у компоненту стабла региона који му је суседан. Ако нема суседних региона, потребно је направити нову компоненту која га садржи и ту компоненту сместити у стабло региона. Постепено креирање компоненти и повезивање региона, за првих неколико корака обраде сегмента са слике 2.8, приказано је на слици 2.9.

Пиксели се обрађују према сортираном редоследу. Сваком пикселу који се обрађује, истражује се локално суседство. Најчешће се локално суседство састоји од 4 или 8 околних пиксела (4-*neighbourhood*, 8-*neighbourhood*). Локално суседство од 4 пиксела које је коришћено у раду је приказано на слици 2.10.



Слика 2.10: Локално суседство од 4 пиксела.

Ако је неки од суседа већ обрађен, а то значи да је додељен неком региону, тј. некој компоненти из стабла свих региона, за тренутни и суседни пиксел који је обрађен се позива `Union` операција. Тиме је



Слика 2.9: Стварање стабла приликом обраде једног по једног пиксела из примера са слике 2.8. Наредбе `MakeSet` и `Union` су наредбе *union-find* алгоритма. Наредба `MakeNewComp` прави нову компоненту у стаблу региона, а наредба `Connect` спаја ту компоненту са одговарајућим компонентама у стаблу региона

тренутни пиксел додат региону коме припада суседни пиксел. На слици 2.9, у прва три корака нема обрађених суседних пиксела и само се праве нове компоненте у стаблу региона, најпре за пиксел на позицији 12 чија је вредност 240, затим за пиксел на позицији 0 чија је вредност 220 и за пиксел на позицији 2 чија је вредност 200.

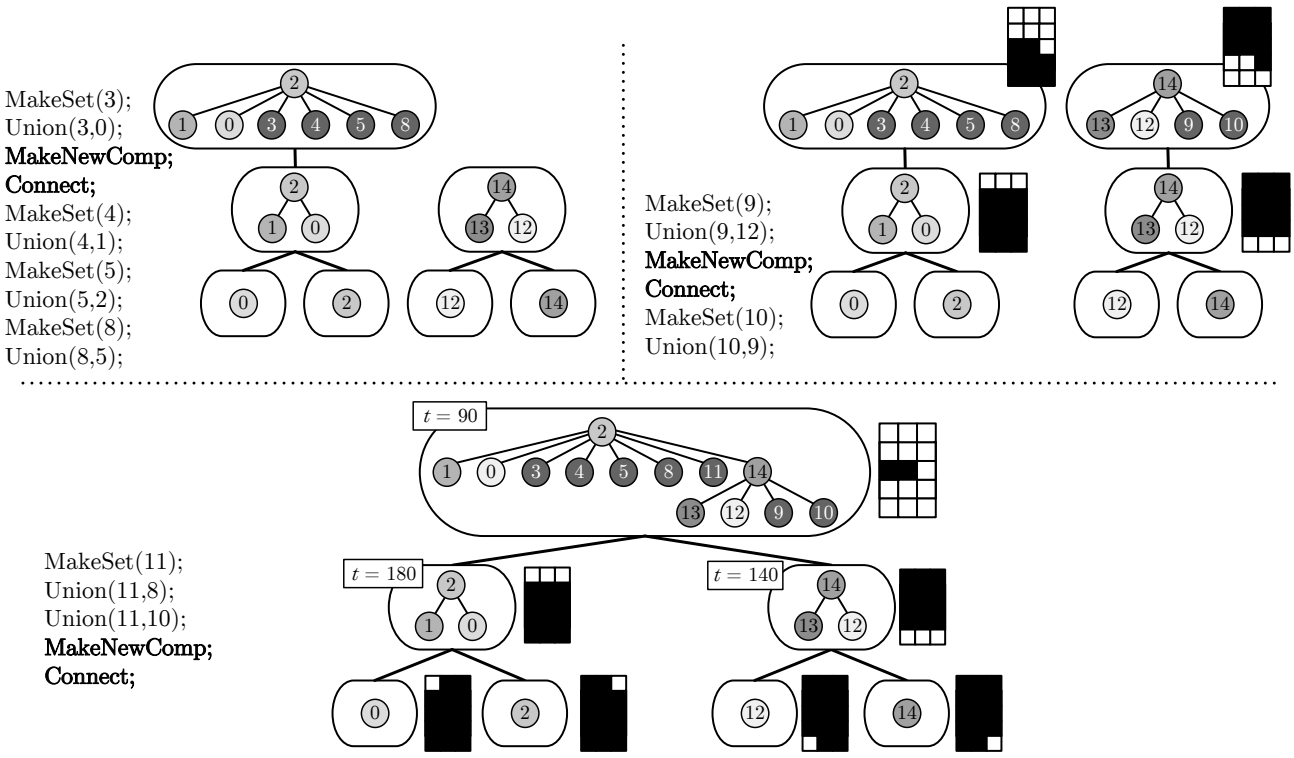
У формирању стабла региона, при додавању тренутног пиксела региону коме припада сусед, треба водити рачуна о вредности тренутног пиксела. Ако је она мања од вредности суседног пиксела, потребно је направити нову компоненту стабла која постаје родитељ компоненти у којој се налази суседни пиксел. Пример је обрада пиксела на позицији 1 што је четврти корак на слици 2.9. Тестира се најпре да ли је већ обрађен пиксел на позицији 2 и пошто јесте позива се `Union` операција за та два пиксела. Међутим, пошто је вредност пиксела на позицији 1 мања од вредности пиксела на позицији 2, потребно је креирати нову компоненту у стаблу. Заправо, вредност пиксела који се тренутно обрађује је уједно и вредност тренутног прага t. Због тога, као и у одељку 2.1, мања вредност прага значи да треба креирати нову компоненту, а не само додати тренутни пиксел у компоненту суседа. Нова компонента у стаблу одговара прагу $t = 180$.

Након тестирања суседа на позицији 2, потребно је тестирати и остале суседе. Пошто нема горњег суседа, прелази се на тестирање левог суседа, а то је пиксел на позицији 0 (слика 2.9, пети корак). Пошто је пиксел на позицији 0 већ обрађен, позива се `Union` операција за пикселе 1 и 0, чиме се већ направљеној компоненти која одговара прагу $t = 180$ додаје пиксел 0. Овим је показано како један пиксел који је сусед двама регионима доводи до спајања региона.

Преостали сусед је пиксел на позицији 4, али он није раније обрађен, па се с њим не ради ништа. На крају се тренутни пиксел прогласи обрађеним и прелази се на обраду следећег пиксела, што је у примеру са слике 2.8 пиксел на позицији 14. Наредни кораци у примеру са слике 2.9 раде исто само са пикселима 12, 14 и 13.

Сада су на реду пиксели чија је вредност 90. Први такав пиксел је пиксел на позицији 3. Слично као и раније, пиксел са позиције 4 се додаје региону коме припада пиксел са позиције 0, пошто је он једини обрађени сусед, и креира се нова компонента стабла која одговара прагу $t = 90$. Даље се додају следећи пиксели чија је вредност 90 – пиксели са позиција 4, 5 и 8. На слици 2.11 у горњем левом углу приказано је стање након обраде пиксела са позиције 8. Следећи су пиксели са позиције 9 и 10 и они се на исти начин додају региону коме припада пиксел са позиције 12, односно 13. На слици 2.11 у горњем десном углу приказано је стање након обраде пиксела са позиције 11.

Преостали пиксел чија је вредност 90 је пиксел са позиције 11. Овај пиксел доводи до спајања до тада два раздвојена региона на исти начин као и код једноставнијих компоненти са слике 2.9. Резултујуће стабло је приказано на слици 2.11. Следећи пиксел који се процесира је пиксел на позицији 6 и на крају пиксел на позицији 7. Због промене прага ($t = 70$, односно $t = 30$), креираће се још две компоненте стабла. Прва, поред осталих, садржи и пиксел са позиције 6, а друга садржи све пикселе стабла [17], [16].



Слика 2.11: Креирање комплекснијих делова стабла. Уз неке компоненте, дата је и одговарајућа вредност прага t, као и визуелна представа региона.

На исти начин се долази до стабла региона знатно већих слика. Када је познато стабло, лако је наћи који региони су максимално стабилни.

Постоје и други методи који за креирање стабла не користе *union-find* алгоритам. Нпр. у [22] и [23] се не ради сортирање пиксела на почетку. Креће се из случајно одабраног пиксела и локално се испитује околина тог пиксела. Најлакше је схватити овај приступ, ако се слика замисли као површина која на себи има долине и брегове. Најпре се тражи најближи минимум на слици (или максимум, зависно од тога који екстремални региони се траже), тј. прва долина на слици. Та долина постаје регион чија се величина прати у току обраде околних пиксела чији су нивои све већи и већи. Интензитет околних пиксела који се испитују расте све до првог брега иза кога се налази нова долина и тако испочетка док се не прође кроз све пикселе и не заврши на највишем брегу.

Циљ овог поглавља је био да се опишу максимално стабилни екстремални региони и у алгоритамском смислу опише начин њихове детекције на слици. У следећем поглављу су објашњени детаљи имплементације *MSER* алгоритма коришћене у овом раду.

Поглавље 3

Имплементација *MSER* алгоритма

У овом поглављу су објашњени детаљи имплементације *MSER* алгоритма коришћене у раду. Имплементација је заснована на раније познатој хардверској имплементацији у *FPGA* чипу која користи благо модификован *union-find* алгоритам [16]. Након што се цела слика учита у улазну меморију, најпре је потребно урадити препроцесирање слике што је заправо сортирање пиксела у опадајућем поретку. Затим се коришћењем *union-find* алгоритма прати стварање, раст и спајање региона са променом прага t. Током те обраде, упоредо се ради детекција да ли је неки од постојећих региона максимално стабилан и уколико јесте, на излаз се шаљу податак о величини региона и позиције свих пиксела који му припадају. Алгоритам као резултат може да дâ све *MSER* регионе, али је подешен тако да многе регионе који нису значајни одбаци. Нпр, за конкретну примену, од интереса су само *MSER* региони који нису настали спајањем два већ детектована *MSER* региона. О овоме ће касније бити више речи.

3.1 Сортирање пиксела

Резултат сортирања представља вектор позиција пиксела сортиран по интензитетима пиксела на тим позицијама. За смештање тог вектора потребно је $N \log_2 N$ битова, где је N укупан број пиксела на слици. На пример ако је слика 64×64 пиксела, она има укупно $4096 = 2^{12}$ пиксела, па је за смештање позиције потребно 12 битова по позицији.

У овом раду је коришћено сортирање базирано на бинарном претраживању (*binary sort*). Овај алгоритам је погодан када број различитих елемената који се сортирају није велики јер користи хистограм као један од улазних података. С обзиром на то да пиксели на слици могу имати вредности само од 0 до 255, бинарно сортирање је добар избор.

Најпре се израчуна хистограм улазне слике. Ради уштеде времена, хистограм се може израчунавати и током учитавања слике, пошто свака слика од камере стиже пиксел по пиксел – линија по линија. Када је познат хистограм формира се кумулативни хистограм. Када је сортирање у растућем поретку, вредност кумулативног хистограма на једној позицији је једнака суми вредности са свих нижих позиција у оригиналном хистограму.

На пример, нека је дат хистограм од укупно 13 елемената који могу имати 4 вредности – 0, 1, 2 и 3: $hist = \{1, 4, 2, 6\}$. Кумулативни хистограм је тада $CumHistUp = \{0, 1, 1+4, 1+4+2\} = \{0, 1, 5, 7\}$. Пиксели се тада сортирају на следећи начин: позиција на коју се смешта пиксел се очитава из кумулативног хистограма и то са места које одговара вредности пиксела. Затим се вредност из кумулативног хистограма увећа за 1, како би следећи пиксел који има исту вредност био смештен на за 1 већу позицију. Када се сортирање ради у опадајућем поретку, кумулативни хистограм се израчунава уназад, почев од последње позиције. Тада је вредност кумулативног хистограма на једној позицији једнака суми вредности са свих виших позиција у оригиналном хистограму. За наведени пример, кумулативни хистограм је $CumHistDown = \{4+2+6, 2+6, 6, 0\} = \{12, 8, 6, 0\}$. Сортирање се даље ради исто као и у растућем поретку. На пример, ако се распоређује елемент чија је вредност 2, он ће бити смештен на позицију $CumHistDown[2] = 6$, а вредност $CumHistDown[2]$ се увећа за један. Ако се распоређује елемент чија је вредност 1, биће смештен на позицију $CumHistDown[1] = 8$. Након распоређивања та два елемента, кумулативни хистограм је $CumHistDown = \{12, 9, 7, 0\}$, тако да се сваком следећем елементу зна позиција.

Поред меморије за резултат, потребно је још и обезебдити меморију за смештање оригиналног и кумулативног хистограма што је још $2 \cdot 256 \cdot \log_2 N$ битова. Алгоритам сортирања базиран на бинарном претраживању је погодан пошто подаци никад не мењају места и идекси у сортираном низу се очитавају директно из кумулативног хистограма [16].

3.2 Имплементација *union-find* алгоритма

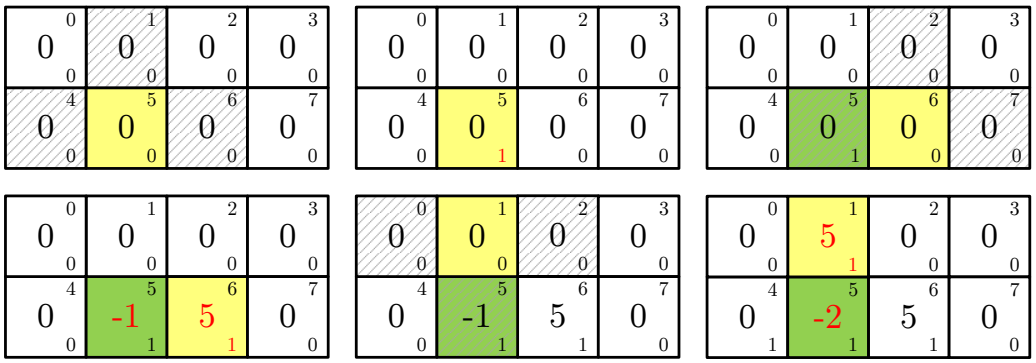
Имплементација *union-find* алгоритма користи једну меморију која се састоји од онолико меморијских локација колико је пиксела у улазној слици. Свака локација у тој меморији, која се даље у раду назива мапа региона (*Region Map - RM*), одговара статусу пиксела са исте локације у оригиналној меморији слике и садржи три податка. Први податак је број, U, који се интерпретира на следећи начин:

- ако је $U = 0$: пиксел са те локације није повезан ни са једним другим пикселом и представља засебан регион за себе, или пиксел са те локације још увек није обрађен, тј. није смештен у мапу региона.
- ако је $U > 0$: пиксел са те локације је део истог региона као и пиксел са позиције U.
- ако је $U < 0$: пиксел са те локације је референтни пиксел (референтна тачка) једног региона, што одговара каноничком елементу из описа *union-find* алгоритма из поглавља 2. Број пиксела који припадају том региону је једнак $1 - U$. Регион се карактерише референтним пикселом и величином (бројем пиксела).

На основу овога се може закључити да су сви региони у мапи региона који имају више од једног пиксела представљени једним елементом чије је $U < 0$ и осталим елементима

чије је $U > 0$.[I] Како би се у току рада знало који пиксели су већ обрађени, други податак у свакој локацији мапе региона је само један бит чија вредност 0 значи да тај пиксел није смештен у мапу региона, а вредност 1 да јесте. Трећи податак ће бити описан у следећем одељку.

На слици 3.1 је приказано стварање новог региона од три пиксела. Као што је објашњено у поглављу 2, обрада креће од најсветлијег пиксела. За пример са слике најсветлији пиксел је пиксел на позицији 5. Испитује се да ли је неки од суседних региона већ процесиран и пошто није, пиксел на позицији 5 се само прогласи обрађеним. Затим се процесира пиксел на позицији 6. За њега постоји суседни пиксел који је обрађен и он се додаје региону коме припада суседни (5.) пиксел. Тачније, 5. пиксел се прогласи референтним пикселом, а вредност броја U шестог пиксела се постави на $U_6 = 5$. Слично је и са пикселом на позицији 1.



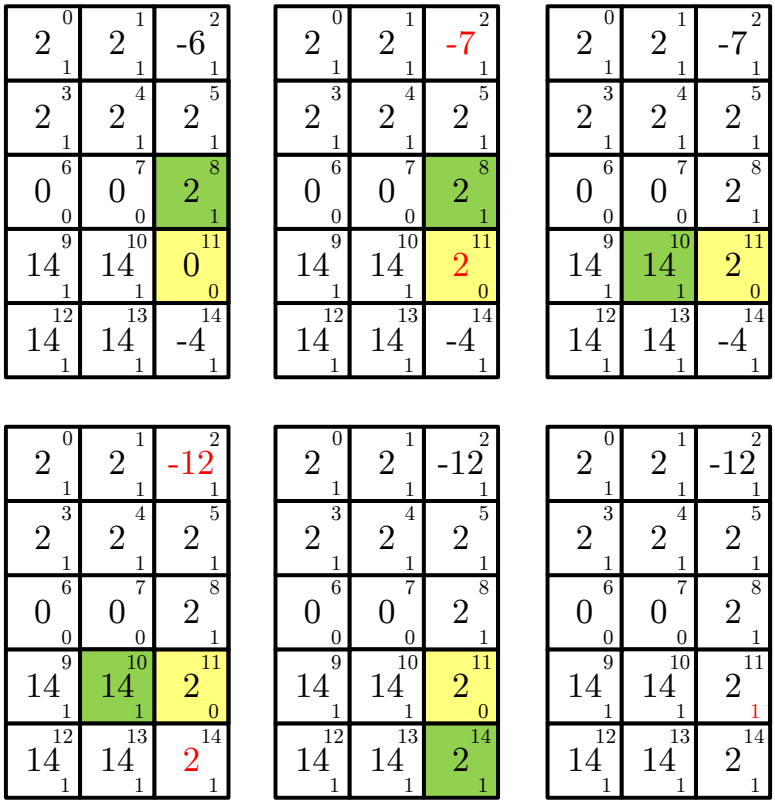
Слика 3.1: Стварање региона на слици. Сваки квадрат представља једну локацију у мапи региона. У горњем десном углу је позиција пиксела, а број у доњем десном углу показује да ли је пиксел већ процесиран (1) или није (0). У средини је број U. Секвенца је приказана слева на десно и одозго на доле. Жутом бојом су означени пиксели који се тренутно процесирају. Редослед процесирања пиксела је 5–6–1. Шрафирани су сви суседи тренутног пиксела, а зеленом бојом су обележени суседи који су већ додати у мапу региона. Црвеном бојом су обојене промене у мапи региона у односу на претходни корак.

У току обраде, често је потребно имати информацију ком региону припада неки пиксел, тј. која је референтна тачка региона коме припада тај пиксел. Ово се може утврдити узастопним читањима меморије са локација U све док се не појави локација за коју је $U < 0$. На пример, ако је потребно одредити ком прегиону припада пиксел са локације p,

[I] Ако позиције пиксела почињу од броја 0, као што је случај у свим примерима у овом раду, пиксел на позицији 0 не може бити референтни пиксел јер би у том случају у статусу пиксела који припадају том региону било $U = 0$, па би се очитавањем статуса тих пиксела стекла погрешна представа да они не припадају ни једном региону. Ако заиста постоји регион у горњем левом углу и нулти пиксел би требало да буде референтни, референтни пиксел ће бити неки други пиксел из околине. Овим се не губе региони већ само један пискел. Ограничење је уведено због смањења меморијских ресурса. На пример, адреса 4096 локација (0–4095) се може приказати са 12 битова, али ако би енумерација кренула од броја 1, било би потребно 13 битова.

прво се прочита податак U са те локације: $U_0 = RM(p, U)$.[II] Ако је $U_0 > 0$, чита се податак са локације U_0: $U_1 = RM(U_0, U)$ и тако даље све док се не прочита податак $U_k < 0$. Тада је U_{k-1} позиција референтне тачке региона, а величина региона је $1 - U_k$.

Пиксели припадају истом региону ако су референтне тачке оба пиксела, одређене на описани начин, једнаке. Ако је потребно спојити два региона, увек се мањи регион додаје већем региону и то тако што се на позицији референтне тачке мањег региона за вредност броја $U = U_{small}$ постави позиција референтне тачке већег региона ($U_{small} = U_{big}$). Пошто је потребно ажурирати величину региона, пре промене вредности U_{small}, потребно је одузети величину мањег региона од вредности броја U референтне тачке већег региона: $U_{big} = U_{big} - (1 - U_{small})$ [16].



Слика 3.2: Процесирање једног пиксела које доводи до спајања њему суседних региона за пример са слике 2.8. Секвенца је приказана слева на десно и одозго на доле. Тренутни пиксел је пиксел на позицији 11. Зеленом бојом су обележени суседи који се испитују током процесирања тренутног пиксела. Црвеном бојом су приказане промене вредности у мапи региона у односу на претходни корак.

На слици 3.2 је приказана секвенца спајања два региона за пример са слике 2.8. Најпре се пиксел на позицији 11 додаје региону коме припада горњи сусед, а то је регион чија је референтна тачка на позицији 2. Пошто леви сусед припада региону чија је референтна тачка на позицији 14, долази до спајања два поменута региона. Регион са референтном тачком на позицији 2 у том тренутку има $1 - (-7) = 8$ пиксела, а регион са референтном

[II] Нотација $RM(p, U)$ значи да се чита податак U са позиције p.

тачком на позицији 14 има $1 - (-4) = 5$ пиксела, па се због тога други додаје првом региону. Последњи сусед је доњи сусед који сада припада истом региону као и пиксел који се процесира, па се даље пиксел са позиције 11 само прогласи процесираним.

Оваква имплементација *union-find* алгоритма где се спајање ради на основу величине региона, за разлику од основне верзије где се спајање радило према рангу то јест према дубини стабла, доноси одређене предности. Главна предност је што референтне тачке већих региона приликом спајања са мањим остају фиксне, чиме се омогућава лакше праћење величине региона што је од кључног значаја за детекцију региона који су максимално стабилни. Наравно, референтне тачке могу нестати ако се региони спајају са већим регионима. Ако се спајање ради према дубини стабла, иако ретко, може се десити да једно стабло има већу дубину од другог, али има значајно мањи број пиксела од другог. Сматра се да су региони који су већи, они који су репрезентативнији. Пример су два региона при прагу $t = 190$ са слике 2.4. Број операција јесте порастао, али не много јер се често критеријуми дубина стабла и величина региона поклапају.

У *C* имплементацији алгоритма, један елемент мапе региона представља структуру која садржи описане податке, а мапа региона је статички алоциран низ елемената. Како је цела имплементација заснована на *FPGA* имплементацији из [16], *C* код је организован тако да се логичке целине могу искористити приликом даље имплементације овог рада на *FPGA* чипу.

3.3 Детекција *MSER* региона

У другом поглављу је описано креирање стабла региона за све вредности прага t из кога се одређује који региони су максимално стабилни. С обзиром на то да је за смештање целог стабла потребно пуно меморије, важно је избећи креирање целог стабла већ чувати само онолико информација колико је довољно да се детектује који региони су максимално стабилни. Пошто се за израчунавање фактора стабилности региона користе само величине региона при различитим вредностима прага t, за проверу да ли је регион максимално стабилан при прагу $t = i$ довољно је само чувати податке о величини региона за вредности прага $i - \Delta \leq t \leq i + \Delta$. Овде се испољава важност тога да су референтне тачке региона на фиксним позицијама јер је у том случају довољно чувати вредност референтног пиксела региона, поред података о величини региона, да би се установило који региони су максимално стабилни. Очигледно, да би се уштедела меморија, потребно је вршити проверу да ли су региони максимално стабилни у току стварања стабла. Сви подаци који у току обраде више нису потребни се бришу, тј. њихове меморијске локације се могу искористити за нове податке.

За детекцију *MSER* региона користи се меморија величина (*sizes memory*) у којој се чувају наведени подаци о величинама региона. Сваком региону R се додељује простор за смештање локације референтног пиксела, један кружни бафер у који се смештају величине

региона ($|R|$), као и простор за бројач ($count$) који одређује тренутну локацију у баферу на коју треба уписати величину региона. Кружни бафер има 16 локација, па је у њега могуће сместити највише 16 вредности величине региона, чиме је параметар Δ ограничен на $\Delta < 8$. Свака нова величина се смешта на позицију на коју показује бројач, а потом се вредност бројача увећа за 1. Пошто бафер има 16 позиција, бројач је 4-битни. Кружни бафер са бројем локација једнаким степену броја два је погодан јер није потребно никакво додатно израчунавање за позиције величина при праговима $i+\Delta$ и $i-\Delta$ осим сабирања и одузимања ($count-\Delta$ и $count+\Delta$).[III] У хардверу овај кружни бафер се може направити као померачки регистар, па онда не би било потребе за израчунавањем позиција за величине региона при праговима $i+\Delta$ и $i-\Delta$. У софтверу је ефикасније користити поступак који користи бројач јер је померање података на 16 локација захтевније.

Када год се креира нови регион, на прву слободну позицију у меморији величина се упише локација референтне тачке новог региона и његова величина. Увек при промени вредности прага, величина свих региона, било да су они променили величину или нису, упише се на позицију у кружном баферу која је једнака вредности бројача. Ако неки регион више не постоји, тј. спојен је са неким другим регионом, онда се део меморије величина у коме су смештени подаци везани за сада непостојећи регион ослобађа за нове детектоване регионе. Како би се рационалније користили меморијски ресурси уводе се параметри максималне величине региона и минималне величине региона. У меморију величина се не уписују региони чија је величина мања од минималне величине региона (у овом раду је то најчешће било 20 пиксела). Такође, из меморије се бришу подаци о величинама региона који је већи од параметра максималне величине и такви региони се више не разматрају. У овом раду је детекција *MSER* региона имплементирана у програмском језику *C*, где су улазни подаци слика и параметри (максимална и минимална величина, као и максимална варијација величине региона). Код је компајлиран да се може користити у програмском пакету *MATLAB*, па се из главне *MATLAB* скрипте позива *MSER* детекција имплементирана у *C*-у.

Како би се одредило да ли је регион максимално стабилан, поред горенаведених података о величинама и референтној тачки потребно је чувати и тренутну вредност фактора стабилности $q(j) = (|R_{j-\Delta}| - |R_{j+\Delta}|)/|R_j|$, као и репрезентацију првог извода $dq(j)/dj$ која је једнобитни флег који показује да ли $q(j)$ тренутно расте или опада, тј. чува се резултат операције $sign(q(j) - q(j+1))$.[IV] За тестирање да ли $q(j)$ има локални минимум, најпре се израчуна $q(j-1)$. Ако је $q(j-1)$ веће од $q(j)$ и ако је $dq(j)/dj$ негативан, $q(j)$ је локални минимум. На крају, ради даљег израчунавања, тренутна вредност фактора стабилности и знак $dq(j)/dj$ се ажурирају вредностима $q(j-1)$ и $sign(q(j-1) - q(j))$ респективно.

[III] Прекорачење које може да се деси је овде корисно. На пример: $count + \Delta = 12 + 7 = 1100_2 + 0111_2 = 0011_2 = 3$ што и јесте позиција на којој је смештена величина при прагу $t = i + \Delta$.

[IV] Приметити да се $q(j)$ посматра уназад. Што је j мање, то су региони већи, па су формално, због тога термини „расте“ и „опада“ обрнути.

Укратко, имплементација *MSER* детекције се може описати у два процеса:

1. Сви пискели интензитета i се смештају у мапу региона коришћењем *union-find* алгоритма.[V] Величина свих региона се уписује на одговарајућа места у кружним баферима у меморији величина.

2. Сада се за неке регионе може проверити да ли су максимално стабилни при прагу $j = i+\Delta$. Ако је $j+\Delta \leq 255$, читају се подаци из меморије величина. Ако је величина $|R_{j-\Delta}| \neq 0$, израчунава се $q(j)$ и ради провера да ли је R_j максимално стабилан.

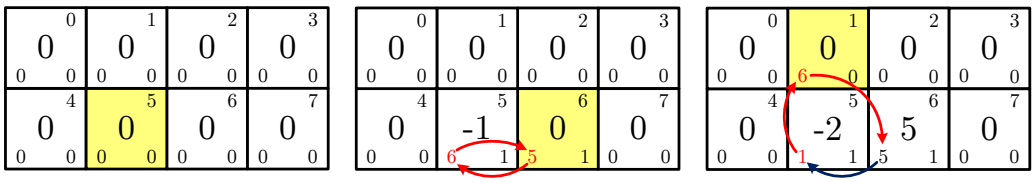
Са свом досадашњом анализом, сви подаци о *MSER* регионима који могу да се добију на ефикасан начин су позиција референтне тачке *MSER* региона и његова величина. Да би се добиле позиције свих пиксела који припадају одређеном региону, мора се проћи кроз све локације у мапи региона и за сваки пиксел утврдити ком региону припада читањем вредности броја U. Такође, у тренутку детекције *MSER* региона при прагу $t = j$, у мапи региона су већ додати сви пиксели који припадају том региону и при прагу $i = j - \Delta$, па је немогуће утврдити који су пиксели додати у међувремену, а који су припадали региону при прагу $t = j$. Неприхватљиво је поред величина региона чувати и све мапе региона при различитим праговима јер се меморијски захтеви повећавају више од 10 пута.

Оба ова проблема, решена су додавањем још једног податка на сваку локацију мапе региона поред броја U и бита који је индикатор да ли је пиксел процесиран или не. Циљ је да се приликом додавања нових пиксела у регион истовремено креира уланчана листа свих пиксела у региону. Због тога је трећи податак на свакој од локација у мапи региона показивач на следећи пиксел који припада региону, тј. његова вредност одговара локацији следећег пиксела. Тиме се креира кружни ланац који нема почетак ни крај, али очитавање најшће почиње од референтне тачке.

На слици 3.3 је приказан исти пример са слике 3.1 само што се упоредо са додавањем пиксела у регион чија је референтна тачка на позицији 5 креира и уланчана листа пиксела. Приликом додавања новог пиксела у регион, показивач на следећи пиксел добија вредност једнаку показивачу из референтног пиксела, а показивач из референтног пиксела се постави да показује на нови пиксел. Приликом спајања два региона, листе се спајају тако што показивачи референтних пиксела замене места. Приликом очитавања пиксела из ланца, после референтног пиксела увек долазе најпре пиксели коју су најкасније додати у регион што има значаја у даљем разматрању.

Увођењем уланчане листе пиксела значајно је смањено време потребно за дохватање свих пиксела једног региона, али су меморијски захтеви повећани. Међутим, уланчани пиксели омогућавају да се из листе пиксела који припадају региону детектованом при прагу $t = t_0$ очитају само пиксели који припадају региону детектованом при прагу $t_1 = t_0 + \Delta$ и то ако се само знају величине региона при оба прага, што спречава чување мапа

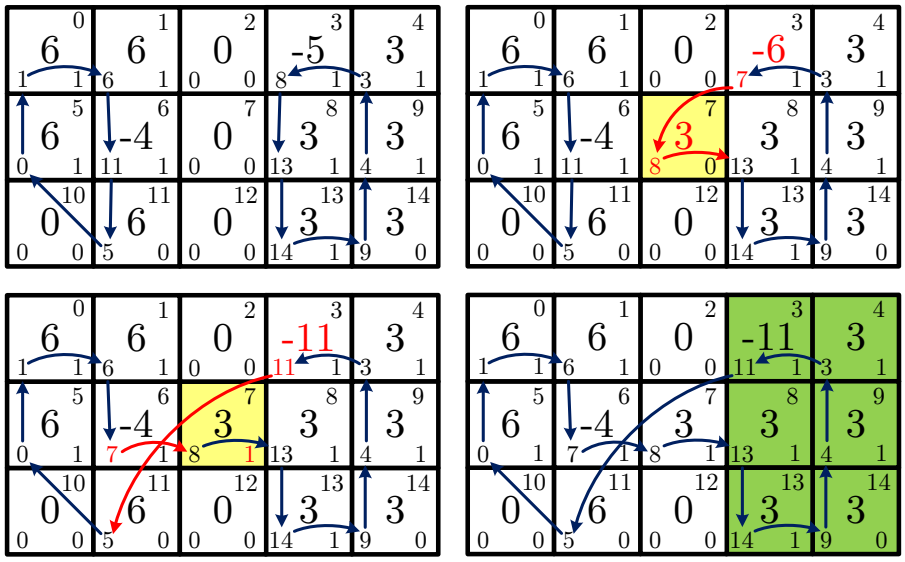
[V] Подразумева се да су пиксели интензитета већег од i већ смештени у мапу региона.



Слика 3.3: Стварање региона и уланчане листе пиксела који припадају том региону

региона за више прагова. Ефективно, овим је количина меморије смањена. Због особине да су пиксели који се најкасније додају региону најближи референтној тачки у уланчаној листи, могуће је у току читања уланчане листе одбацити све оне пикселе који су додати при праговима који су између $t_0 + \Delta$ и t_0. Број пиксела који се одбацују једнак је разлици величина региона при прагу t_0 и прагу $t_0 + \Delta$.

На слици 3.4 је приказан пример спајања два региона и њихових уланчаних листи, као и могућност накнадног очитавања пиксела који су припадали једном региону пре спајања.



Слика 3.4: Спајање два региона и њихових уланчаних листи. Секвенца је приказана слева на десно и одозго на доле. Најпре се процесира пиксел на позицији 7 чиме долази до спајања два региона. Ако треба прочитати само пикселе региона са референтном тачком 3 који су припадали региону пре процесирања пиксела 7, пропушта се $12 - 6 = 6$ пиксела. Зеленом бојом су обојени пиксели који се читају. Пример преузет из [16].

Када је детектован *MSER* регион, његова величина, локације референтног и свих осталих пиксела чувају се у посебном меморијском блоку за детектоване *MSER* регионе. Ако се детектује *MSER* регион са истом референтном тачком као и неки већ детектован регион при неком већем прагу, подаци о том региону се ажурирају, али уз два ограничења која не постоје оригинално у увођењу појма *MSER* региона у [13]. Пошто постоји више детекција *MSER* региона који се налазе на истом месту и једино им се разликују величине, потребно је одабрати најрепрезентативнији од њих. Због тога се приликом чувања података о *MSER* регионима, чува и вредност варијационог фактора q при детекцији. Подаци о

већ детектованом региону се ажурирају једино ако је варијациони фактор новог детектованог региона мањи од оног који је сачуван. Тиме је критеријум за одабир најрепрезентативнијег региона на једној позицији његова стабилност, тј. више није довољно да варијациони фактор има локални минимум, већ се бира регион чији је варијациони фактор глобални минимум.

Друго веома важно ограничење за овај рад је одбацивање свих *MSER* региона који су настали спајањем два већ детектована *MSER* региона или региона који су настали од њих. Укратко речено, важно је детектовати *MSER* регионе при што већим праговима јер је тада већа вероватноћа да ће они представљати одвојене неуроне. За пример се могу узети региони и стабло са слика 2.3 и 2.4. Иако би можда регион који постоји при прагу $t = 40$ био детектован као максимално стабилан, он није важан јер је очигледно настао на месту где се налазе три неурона од којих ће сигурно постојати три раније детектована *MSER* региона.

3.4 Анализа сложености имплементације

С обзиром на то да се *MSER* алгоритам користи у циљу да се слика обрађује у реалном времену на *ASIC* чипу који би требало да користи што мање меморије како би површина чипа била што мања, важно је анализирати имплементацију алгоритма по питању брзине извршавања и количине меморије коју користи.

Брзина извршавања свакако зависи од тога каква је улазна слика, тј. од броја и величине детектованих региона, па није лако дати процену времена извршавања. У [16] се наводи да препроцесирање, тј. сортирање слике на *FPGA* чипу траје $T_{sort} = 3NT_{CLK}$, где је N укупан број пиксела, а T_{CLK} период једног тактног циклуса. Ово време је независно од тога каква је улазна слика пошто код бинарног сортирања нема мењања места пикселима, док у другим алгоритмима за сортирање број мењања места зависи од тога каква је улазна слика.

Време извршавања остатка детекције *MSER* региона се у [16] процењује на $T_{MSER} = 7NT_{CLK}$, што је утврђено након покретања алгоритма на неколико различитих улазних слика. Оваква брзина извршавања омогућава број слика по секунди од 25 fps за слику од 350×350 пиксела, при учестаности такта од 42 MHz [16].

За разлику од брзине извршавања, максималне меморијске захтеве је могуће проценити независно од тога каква је улазна слика. Пошто су сви пиксели 8-битни, за смештање слике потребно је $M_{image} = 8N$ битова, где је N укупан број пиксела.

Локација пиксела се може сместити на $\log_2 N$ битова, а пошто је резултат сортирања вектор у коме сваки елемент представља адресу једног пиксела, то је за резултат сортирања потребно $N \log_2 N$ битова што је укупно са меморијом за чување хистограма $M_{sort} = (N + 512) \log_2 N$.

У мапи региона се на свакој позицији налазе три податка, један је локација наредног

пиксела у ланцу дужине $\log_2 N$ битова, други је једнобитни податак који говори о томе да ли је пиксел процесиран, а трећи је број U који је најчешће вредност локације неког пиксела, али је за референтне пикселе то негативан број па је за његово смештање потребно $\log_2 N+1$ битова. Укупно, за мапу региона је потребно $M_{region\_map} = (2+2\log_2 N)N$ битова.

Количина меморије потребна за смештање резултујућих *MSER* региона зависи од величине и броја региона. У најгорем случају је потребно $M_{detected\_regions} = N\log_2 N$ битова и то ако детектовани региони прекривају више од пола слике.

Свака локација у меморији величина захтева највише $16\log_2 N$ битова за кружни бафер, 4 бита за бројач, 1 бит за знак извода и простор за фактор стабилности. У досадашњем разматрању фактор стабилности је био реалан број и то често број мањи од 1. Ако би се радило са реалним бројевима, то би значајно укомпликовало хардвер, па се фактор стабилности може скалирати бројем који зависи од жељене тачности. На пример, ако се жели тачност од три децимална места, током израчунавања фактора стабилности, вредност $|R_{j-\Delta}| - |R_{j+\Delta}|$ треба помножити са 1000, а затим резултат поделити са $|R_j|$. Разумно је усвојити да у том случају фактор стабилности неће прелазити највећу 16-битну целобројну вредност. Велике вредности фактора стабилности су једино могуће при наглом расту малих региона. Ако се зада минимална величина региона од 20 пиксела, да би се прекорачила максимална вредност скалираног фактора стабилности, потребно је да у опсегу $j-\Delta$ до $j+\Delta$ регион порасте са 20 на 3296 пиксела, што је за природне слике практично немогуће. Укупно, једна локација у меморији величина захтева највише $16\log_2 N + 4 + 1 + 16 = 21 + 16\log_2 N$ битова. Број региона који се јављају у току обраде директно утиче на величину ове меморије, али је у сваком случају величина ове меморије за бар један ред величине мања од укупне величине меморије за остале функционалности.

Не рачунајући меморију величина, за детекцију *MSER* региона је приближно потребно укупно $M_{MSER} = M_{image}+M_{sort}+M_{region\_map}+M_{detected\_regions} = (10+4\log_2 N)N+512\log_2 N$ битова.

На пример, ако је слика величине 1024×1024 пиксела, потребна количина меморије је $M_{MSER} = (10+4\cdot 20)\cdot 1024\cdot 1024+512\cdot 20 = 94382080\,\text{bit} = 11{,}798\,\text{MB}$, што је неприхватљиво за *ASIC* чип који треба да троши мало енергије и да буде релативно мале површине. Такође, време извршавања је тада $T_{MSER} = 10NT_{CLK}$, што је 250 ms при такту учестаности 42 MHz, што је далеко од рада у реалном времену.

Слике које се добијају снимањем калцијумске флуоресценције могу да буду величине 1 мегапиксел и више, па коришћење оваквог *MSER* детектора директно на тако великим сликама не може дати захтеване перформансе. Због тога се прибегава методама као што су дељење слике на блокове и паралелизам чиме се постиже смањење потребне меморије и повећава брзина извршавања за више од једног реда величине. Међутим, тада се могу јавити други проблеми о којима ће више бити речи у следећем поглављу.

Поглавље 4

Алгоритам за издвајање неуронских сигнала

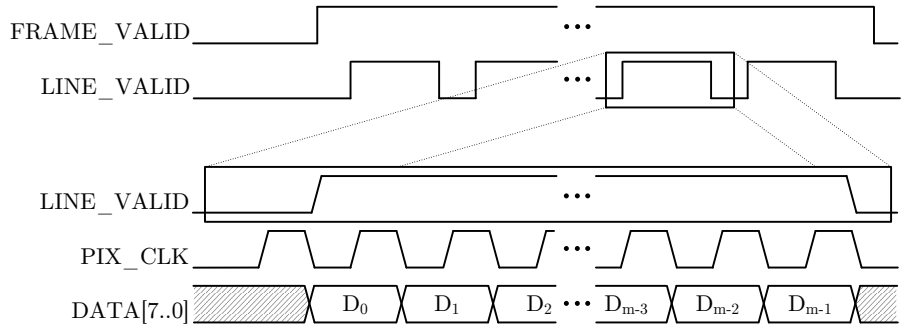
У овом поглављу је описан алгоритам за детекцију позиција неурона намењен хардверској имплементацији. Када су познате позиције неурона, онда се сигнали лако издвајају очитавањем интензитета пиксела из околине позиција неурона.

Алгоритам за детекцију позиција користи већ детаљно описан алгоритам за детекцију *MSER* региона. Детектовани $MSER$ региони, уз одбацивање сувишних услед раније наведених ограничења, представљају неуроне. Међутим, у претходном поглављу је показано да рад у реалном времену са великим сликама директном применом *MSER* алгоритма није могућ, а и меморијски захтеви су превелики.

4.1 Дељење слике на блокове

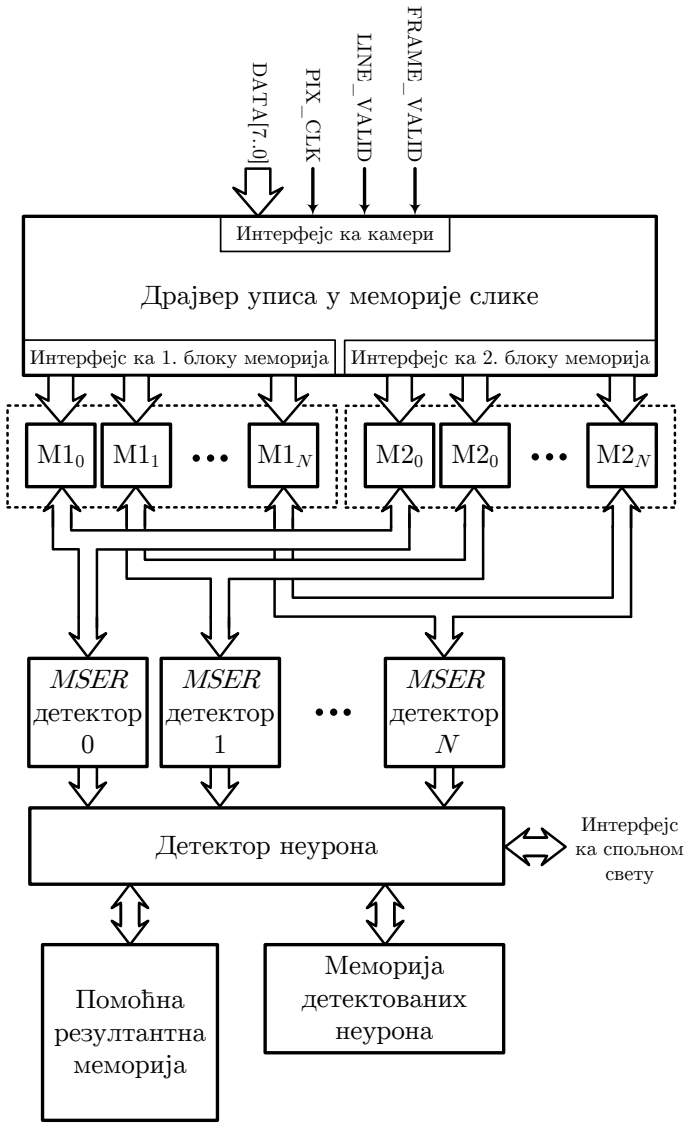
Како би се најпре смањили меморијски захтеви, основна идеја алгоритма за издвајање неуронских сигнала је дељење слике на блокове који се независно обрађују, а онда се резултати обраде додатно обрађују како би се добили резултати који одговарају целој слици. Слика од једног мегапиксела у меморији заузима 1 MB података, што је већ значајна количина *on-chip* меморије, па је размотрена могућност да се делови слике обрађују у току њеног очитавања. Већина камера има интерфејс за податке којим се вредности пиксела шаљу једна по једна и то најчешће линију по линију. На слици 4.1 су приказани најчешћи сигнали оваквих интерфејса и њихови временски облици.

Са оваквим очитавањем података, могуће је прво учитати у меморију одређени број првих линија, а затим, док траје учитавање другог сета линија, обрадити само један део слике. Након обраде првог дела слике, почиње обрада другог дела и учитавање трећег дела. Поред уштеде меморије за смештање улазне слике, количина меморије коју је довољно користити за обраду делова слике је значајно мања него количина меморије за обраду целе слике одједанпут.



Слика 4.1: Сигнали и протокол очитавања података са сензора дигиталне камере

4.2 Паралелизам и предлог хардверског система



Слика 4.2: Предлог хардверске структуре која реализује алгоритам за детекцију неурона

Да би се остварило убрзање у обради слике, једино је могуће паралелизовати процес обраде и то даљим дељењем учитаних делова на блокове и паралелним процесирањем сваког од тих блокова. На слици 4.2 је приказана блок шема предложеног хардверског система који би требало да реализује алгоритам за детекцију неурона.

У овом раду је одабрано да су блокови на које се дели слика величине 64×64 пиксела.[I] Сваки од тих блокова се приликом учитавања слике смешта у засебну меморију из које се даље читају подаци за обраду тог блока (блокови обележени са $M1_0 - M1_N$ и $M2_0 - M2_N$). Драјвер за упис у меморије слике прима податке са камере и смешта прве 64 линије у први блок меморија (M1). Након учитавања прве 64 линије, креће њихова обрада, а драјвер за упис смешта друге 64 линије у други блок меморија (M2). Након учитавања друге 64 линије, креће њихова обрада, а следеће 64 линије се учитавају у први блок меморија итд.

[I] 64 је степен броја 2, па се дељење са 64 може реализовати операцијом померања удесно што је додатна погодност.

MSER детектор је блок у коме је реализован *MSER* алгоритам описан у трећем поглављу. У систему треба да постоји онолико блокова за детекцију *MSER* региона на колико меморијских блокова је подељена трака од 64 линије. Сваки *MSER* детектор податке о детектованим регионима (референтне тачке, величину и све локације пиксела) прослеђује блоку названом „детектор неурона“ који их даље процесира.

Детектовани региони у *MSER* блоковима представљају неуроне. Наравно, неће сви неурони бити видљиви на сваком фрејму у току времена. Неки ће бити видљиви одмах на почетку, неки ће постати видљиви тек кад се концентрација калцијумових јона у њиховој цитоплазми и околини повећа. Због тога се детекција неурона ради док год се ради снимање неуронске активности, а када год се појави нови неурон, даље се снимају интензитети пиксела и са његове позиције.

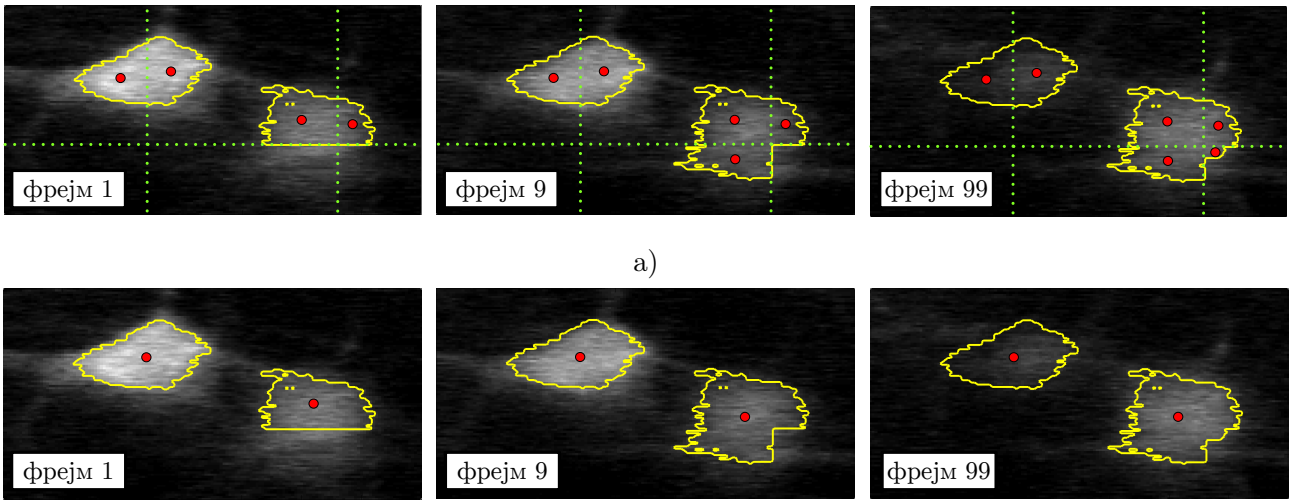
Такође, један неурон ће свакако бити детектован на више фрејмова у току времена, па је због тога потребно водити евиденцију о томе који су неурони већ детектовани. Референтна тачка детектованог *MSER* региона који одговара једном неурону на једном фрејму, највероватније неће бити иста као референтна тачка детектованог *MSER* региона који одговара истом том неурону на другом фрејму. Због тога није довољно проверавати да ли су референтне тачке нових и старих региона једнаке, па је потребно проверити да ли се већина пиксела региона детектованог у првом фрејму и региона детектованог у другом фрејму разликује. Ако се већина пиксела поклапа, сматра се да је регион већ детектован у претходним фрејмовима и нови регион се одбацује. За ову проверу се користи помоћна резултантна меморија величине N бита где је N број пиксела целе улазне слике. Сваки бит одговара једној локацији у целој улазној слици и поставља се на 1 ако је пиксел на тој локацији део детектованог региона.

Сваки пут кад неки од *MSER* детектора детектује нови *MSER* регион, детектор неурона најпре провери да ли се на позицијама које одговарају новом детектованом региону налазе уписане јединице у помоћној резултантној меморији. Ако број очитаних јединица пређе 10% од укупног броја пиксела новог региона, тај нови регион се одбацује јер се сматра да је неурон коме одговара нови регион већ детектован.[II] У супротном, сматра се да је детектован нови неурон и у помоћну меморију резултата се на позиције свих пиксела новог региона уписују јединице. Такође, важно је, због уштеде меморијског простора, неуроне представити на неки други репрезентативан начин који не захтева чување свих пиксела који им одговарају. Референтна тачка не мора бити најсветлија тачка региона и може се налазити близу ивице региона, па се израчунава средња вредност хоризонталних и вертикалних координата свих пиксела који припадају региону. Пиксел који има координате

[II] Овај проценат може бити различит, али је због померања слике у току времена задржан на 10%. О томе ће више бити речи у резултатима и дискусији у разматрању проблема који настају због померања слике. Дакле, ако се два различита неурона преклапају највише 10%, они ће бити детектовани као два неурона, у супротном ће бити детектован само један. За снимке на којима је тестиран алгоритам ово је био довољан проценат за велики број исправних детекција.

које одговарају израчунатим средњим вредностима се сматра репрезентативним и његова позиција се чува у меморији детектованих неурона. Поред позиције средишњег пиксела, чува се и величина региона који одговара неурону.

Дељењем слике на блокове може да се деси да се неки од неурона пресеку, тј. да се један део детектује у једном блоку, а други део у другом, њему суседном, блоку. До сада описаном детекцијом, а без додатног процесирања, сваки неурон који се састоји из два дела из два различита блока биће детектован као два неурона (слика 4.3.а). Овај проблем је решен спајањем ивичних региона.



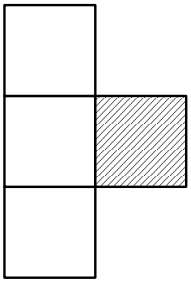
Слика 4.3: Детекција неурона без спајања ивичних региона (а) и са спајањем ивичних региона (б). Црвене тачке су средишње дачке неурона. Жутом бојом су оивичене све јединице у помоћној резултантној меморији, а зелене линије су границе између блокова.

За сваки нови регион, најпре се провери да ли је ивични регион, тј. да ли се неки његови пиксели налазе у првој или последњој врсти, односно првој или последњој колони блока. Ако регион јесте ивични и не преклапа се више од 10% са јединицама из помоћне резултантне меморије, онда је потребно проверити да ли у суседним блоковима има региона који се додирују са њим. Ово је могуће урадити, читањем помоћне резултантне меморије на локацијама које су суседне ивичном региону у целој слици. Ако су на тим локацијама уписане јединице, онда је и у суседним блоковима раније детектован ивични регион. Потребно је да се нови ивични регион додирује са раније детектованим ивичним регионом у бар неколико пиксела. Минималан број додирних пиксела зависи од величине региона. Ако постоји довољан број додирних пиксела, нови регион треба спојити са раније детектованим суседним регионом.

С обзиром на то да се у резултантној меморији не чувају подаци о томе ком региону припадају уписане јединице, већ оне само значе да пиксели са тих позиција припадају неком, било ком региону, потребно је пронаћи суседни регион коме припадају ивичне јединице. Тај регион припада барем једном од три суседна блока ако се посматра суседство

од 8 блокова. На пример, ако се нови ивични регион налази уз леву ивицу блока, онда је суседни регион у једном од левих суседа тог блока (слика 4.4).

Суседни ивични регион са којим треба спојити нови ивични регион се тражи у суседним блоковима, при чему се рачуна Еуклидово растојање између средишњих тачака свих региона у суседним блоковима и новог ивичног региона. Онај регион за кога је Еуклидово растојање најмање, сматра се ивичним суседом и спаја се са новим ивичним регионом. Спајање се ради ажурирањем средишње тачке и величине региона у меморији детектованих региона. Нове координате средишње тачке се рачунају као



Слика 4.4: Леви суседни блокови у суседству од 8 суседа.

$$n_{1\_new} = \frac{n_1 size_1 + n_2 size_2}{size_1 + size_2}, m_{1\_new} = \frac{m_1 size_1 + m_2 size_2}{size_1 + size_2},$$

где су n_1, m_1 и $size_1$ координате и величина суседног ивичног региона, а n_2, m_2 и $size_2$ координате и величина новог ивичног региона.[III] Резултат оваквог спајања региона је приказан на слици 4.3.б.

4.3 Анализа сложености алгоритма

Уштеде у количини меморије и брзини извршавања услед дељења слике на блокове је најбоље приказати на примеру који је већ коришћен, а то је слика димензија 1024×1024 пиксела. С обзиром на то да су за смештање улазних блокова од 64×64 пиксела у току извршавања потребне две меморије, једна за блок који се тренутно обрађује, а друга за блок који се тренутно учитава, то је за смештање сваког пара улазних блокова из две траке потребно $M_{block} = 2 \cdot 8N_{block}$ битова, где је $N_{block} = 64 \cdot 64 = 4096$.

У осталим деловима *MSER* алгоритма нема модификација, па је за детекцију *MSER* региона приближно потребно укупно: $M_{MSER\_block} = (10+4\log_2 N_{block})N_{block}+512\log_2 N_{block} +8N_{block} = 276480$ битова по блоку. Укупно има $1024/64 = 16$ паралелних блокова, па је за *MSER* детекцију укупно потребно: $M_{MSER} = 16M_{MSER\_block} = 4423680\,\text{bit} = 0{,}553\,\text{MB}$.

За реализацију алгоритма је још потребно N битова за помоћну резултантну меморију. Остали меморијски захтеви су занемарљиви у односу на већ постојеће, тако да је укупно за реализацију алгортима потребно: 0,68 MB што је 17 пута мање него у случају када се обрађују целе слике.

Време детекције *MSER* региона на целој слици је: $N_{strips} \cdot 10N_{block}T_{CLK}$, где је N_{strips} број трака на које се дели слика $N_{strips} = 1024/64 = 16$, па је за анализирани пример $T_{MSER} = 16 \cdot 10 \cdot 4096T_{CLK} = 655360T_{CLK} = 15{,}6\,\text{ms}$, што може да да брзину смењивања слика од 64 fps. Замишљено је да детектор неурона обрађује детектоване *MSER* регионе

[III] Овакво израчунавање је према аналогији са израчунавањем центра масе два тела различитих маса.

из једне траке упоредо са *MSER* детекцијом из друге траке, тако да осим иницијалног кашњења за обраду прве траке, нема додатних кашњења.

Функционалност другог дела алгоритма који ради коначну детекцију неурона и спајање ивичних региона тестирана је у програмском пакету *MATLAB*. Дељењем слике на блокове од 64×64 пиксела и подешавањем параметара за *MSER* детекцију, показује се да предложени алгоритам даје одличне резултате за слике код којих нема померених фрејмова. Иако резултати које алгоритам даје за снимке код којих се фрејмови померају нису тако добри, алгоритам показује робусност и висок проценат исправних детекција и у таквим случајевима. У наредном поглављу су приказани резултати за више различитих тестова и дата њихова детаљна дискусија, а у закључку су дати предлози за унапређење алгоритма и даљи рад.

Поглавље 5

Резултати и дискусија

У овом поглављу приказани су резултати детекције неурона применом описаног алгоритма. Поред приказа детектованих неурона, дати су и прикази сигнала које ти неурони емитују. За тестирање алгоритма коришћена су три снимка добијена двофотонском микроскопијом. Два снимка која су коришћена у току развоја алгоритма добијени су са Универзитета Стенфорд. Снимци су резолуције 256×512 пиксела и имају 720 фрејмова. Један снимак приказује увећаних око 10 неурона (у даљем тексту снимак 1), док други приказује активност између 70 и 90 неурона (у даљем тексту снимак 2). На сликама 5.1.а и 5.1.б су приказане слике које представљају усреднене све фрејмове у току времена и на њима се може видети највећи број неурона који се јављају.

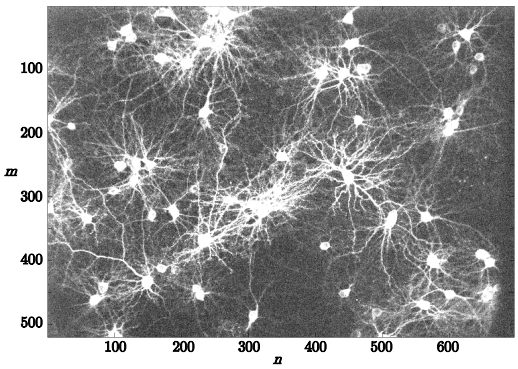
За поређење са алгоритмом базираним на анализи независних компонената [6] коришћен је снимак из [1] (у даљем тексту снимак 3).[I] Снимак је резолуције 520×696 пиксела, има 1200 фрејмова и знатно је бољег квалитета од прва два. Слика која представља усреднене фрејмове овог снимка у току времена приказана је на слици 5.1.в.

За сваки од снимака, детекција је рађена за различите вредности параметра Δ, као и параметара минималне и максималне величине детектованих региона. За прва два снимка, није било потребно радити никакво додатно препроцесирање. Међутим, приликом рада са трећим снимком из [1], примећено је да је током снимања камера често улазила у засићење. Наиме, приликом великог повећања концентрације калцијумових јона, флуоресценција је толико велика, да се на снимку јако засветле многи дендрити и аксони (слика 5.2). Због тога је добијен велики број погрешних детекција. Овај проблем је решен ублажавањем ивица које настају услед сатурације просторним усредњавањем Гаусовим нискофреквентним филтром. Маска са којом се ради филтрирање је димензија 3×3, а стандардна девијација је $\sigma = 1$:

$$w = \begin{bmatrix} 0.0751 & 0.1238 & 0.0751 \\ 0.1238 & 0.2042 & 0.1238 \\ 0.0751 & 0.1238 & 0.0751 \end{bmatrix}$$

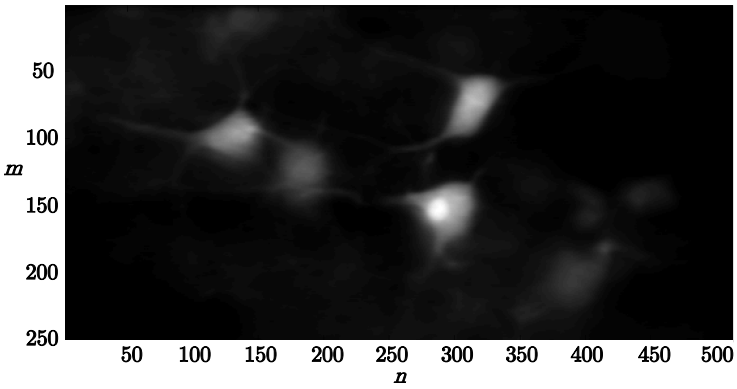
[I] Доступно на: www.seas.upenn.edu/ molneuro/fluorosnnap.html

Такође, неки неурони су већ довољно светли у неактивном стању, да се њихова активност уопште не може детектовати јер је на њиховим позицијама камера ушла у засићење и у неактивном стању. Свакако, овај проблем се могао избећи бољим подешавањем параметара камере која је снимала флуоресценцију. Због тога се овде не разматрају нови ресурси потребни за филтрирање. Оно је рађено само да би се снимак који је коришћен прилагодио за тестирање алгоритма развијеног у овом раду.

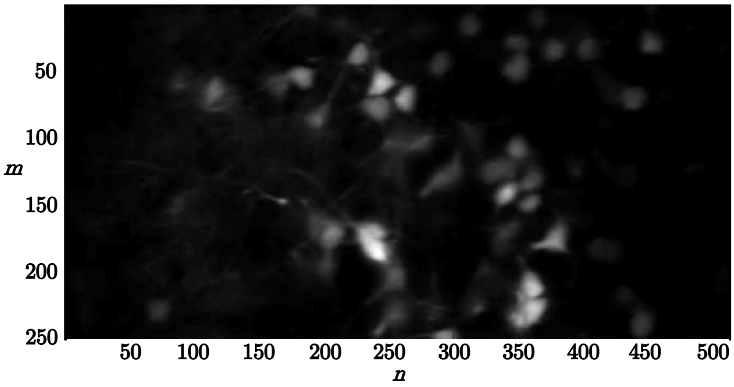


Слика 5.2: Слика која је снимљена када је камера великим делом била у засићењу

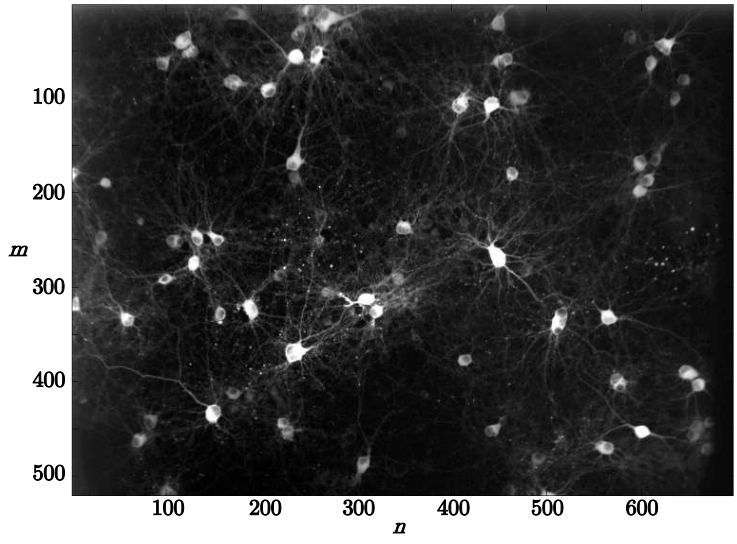
Резултати детекције неурона на снимку 1, приказани су на сликама 5.3.а и 5.3.б. Сигнали приказани на слици 5.3.б су добијени очитавањем вредности средишњег и 8 околних пиксела за сваки неурон и сваки фрејм на снимку. Најпре је израчуната средња вредност средишњег пиксела и његове околине за сваки фрејм (ознака I у даљем тексту). Ради јаснијег приказа сигнала, на слици 5.3.б су за сваки неурон приказане вредности $(I - I_0)/I_0$ скалиране између 0 и 0,8, где је I_0 средња вредност средишњег пиксела и његове околине за цео снимак.



а) Снимак 1



б) Снимак 2

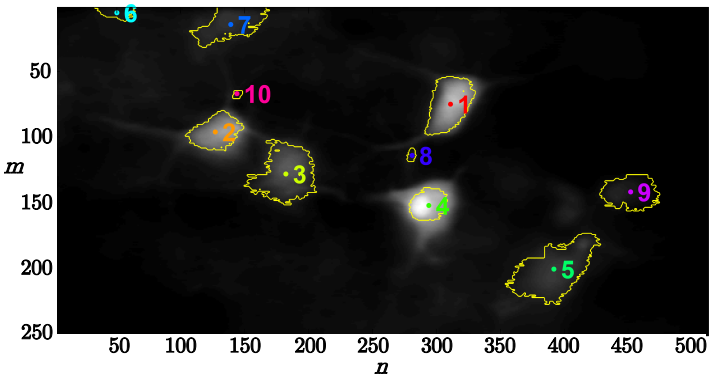


в) Снимак 3

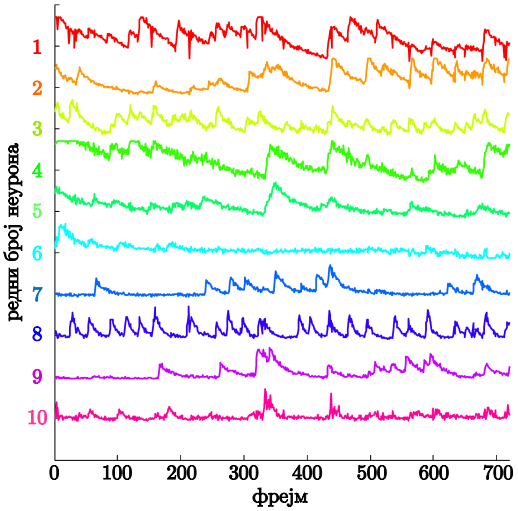
Слика 5.1: Слике које представљају усредњене фрејмове у току времена за коришћене снимке

За првих 9 детекција је јасно да оне одговарају различитим неуронима, док за детекцију

број 10 то није могуће утврдити на основу доступних података.



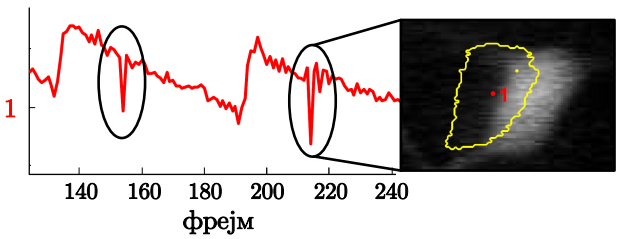
а) детектовани неурони



б) екстраховани сигнали

Слика 5.3: Детектовани неурони са снимка 1 (а) и екстраховани сигнали које они емитују (б). Жутом бојом су оивичене све јединице у помоћној резултантној меморији, а тачке унутар оивичених региона су средишње тачке неурона. Параметри сваког *MSER* детектора су: $\Delta = 6$, $min\_size = 40$, $max\_size = 2000$.

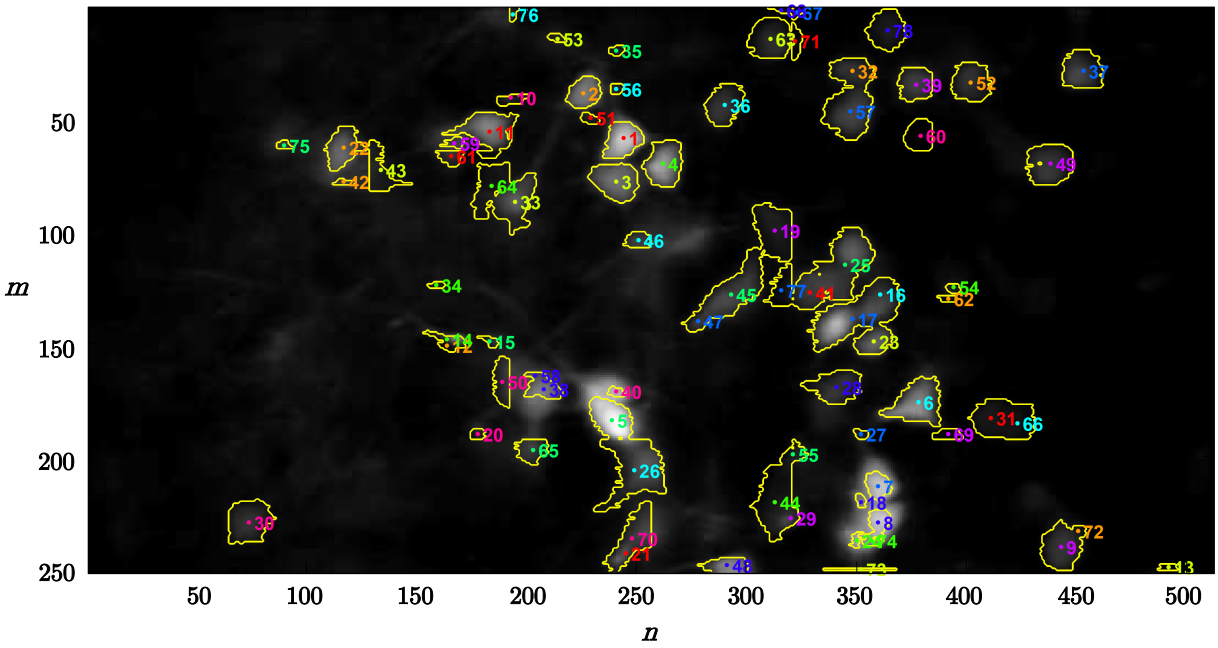
Може се приметити да постоје пропади вредности сигнала 1 на неким местима. Овај проблем се јавља због тога што су одређени фрејмови померени у односу на остале. На слици 5.4 је приказан пропад вредности сигнала приликом помераја фрејма 214 из снимка 1. Фрејм је толико померен да се неурон сада налази поред раније детектоване средишње тачке неурона. Често се у литератури наводи да је у току препроцесирања снимка потребно урадити поравнање свих фрејмова у снимку [6], [1]. Ово би било решење наведеног проблема, али је поравнање свих фрејмова најчешће рачунски захтеван процес и због дељења слике на делове, у овом раду неприменљив.



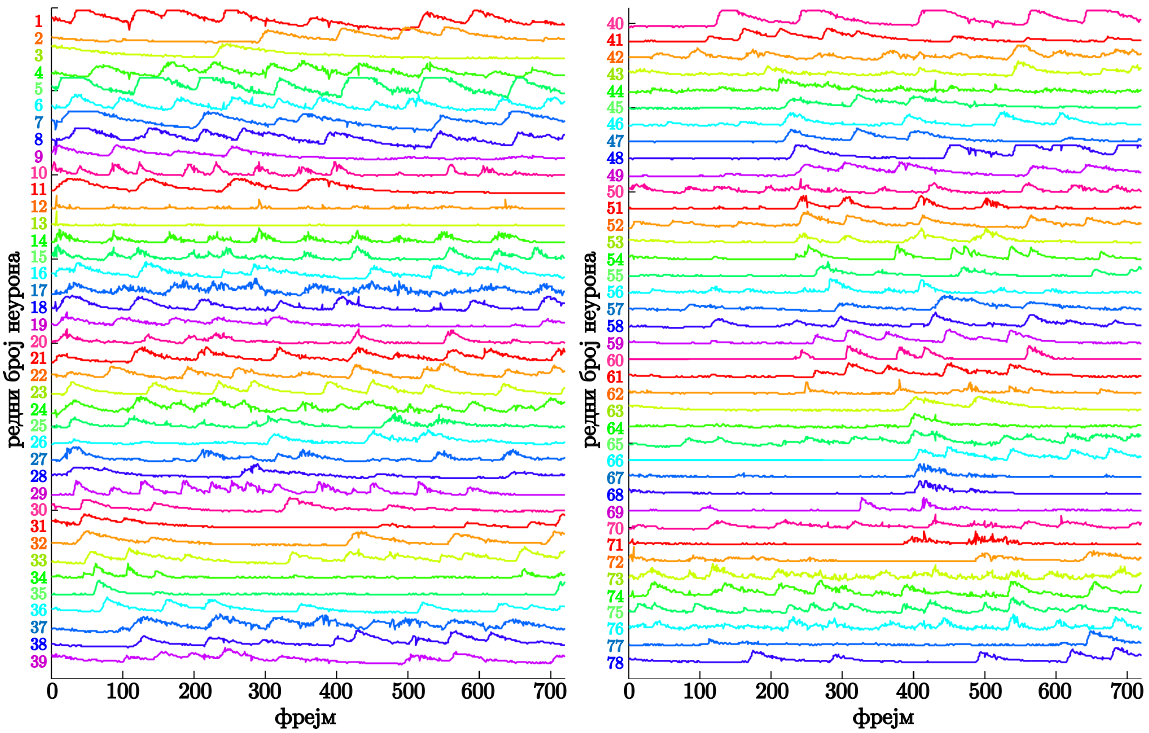
Слика 5.4: Пропад вредности сигнала услед помераја одређених фрејмова

На слици 5.5.а су приказане детекције неурона на снимку 2, а на слици 5.5.б сигнали који одговарају тим детекцијама. Може се приметити да је велики број неурона исправно детектован, али и да постоји више детекција које одговарају једном неурону. Један од узрока дуплих детекција је и померај фрејмова у снимку. Наиме, може се десити да је услед помераја неурон делом прешао из једног блока од 64×64 пиксела у други блок где

раније није био детектован, а сада ће бити детектован нови регион који јесте ивични, али је раније цео неурон био у првом блоку и не постоји довољно суседних пиксела између два региона да би они били спојени. Пример су детекције 63 и 71 на слици 5.5.а.



а) детектовани неурони

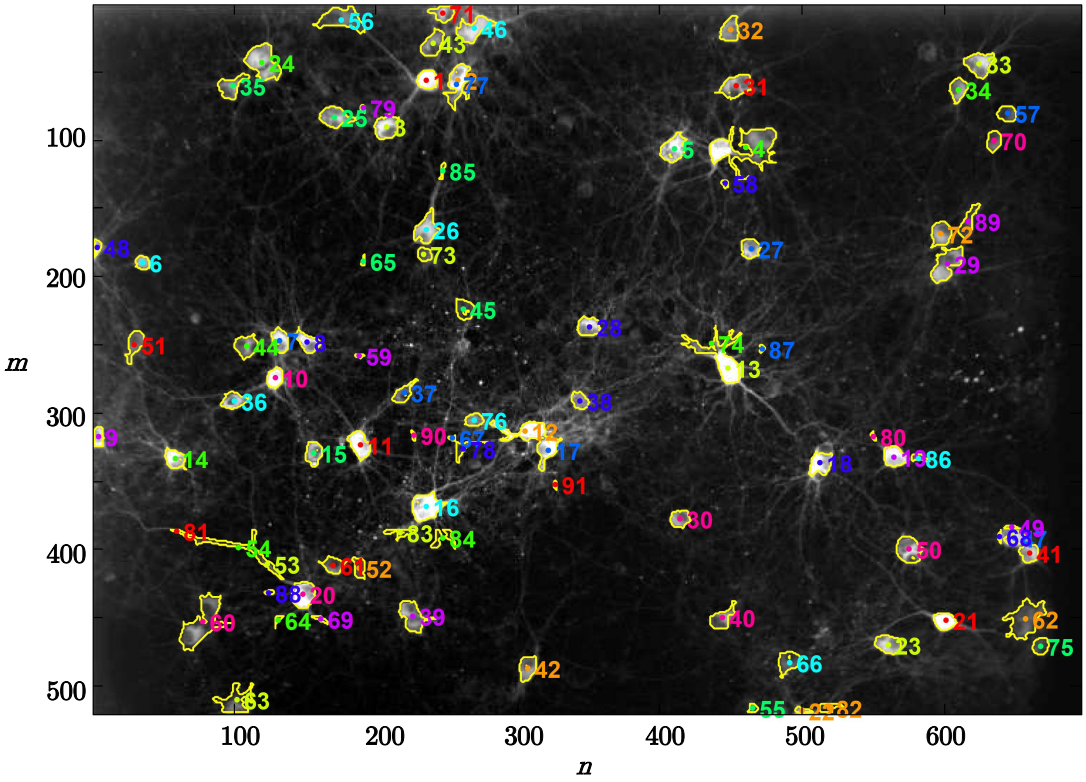


б) екстраховани сигнали

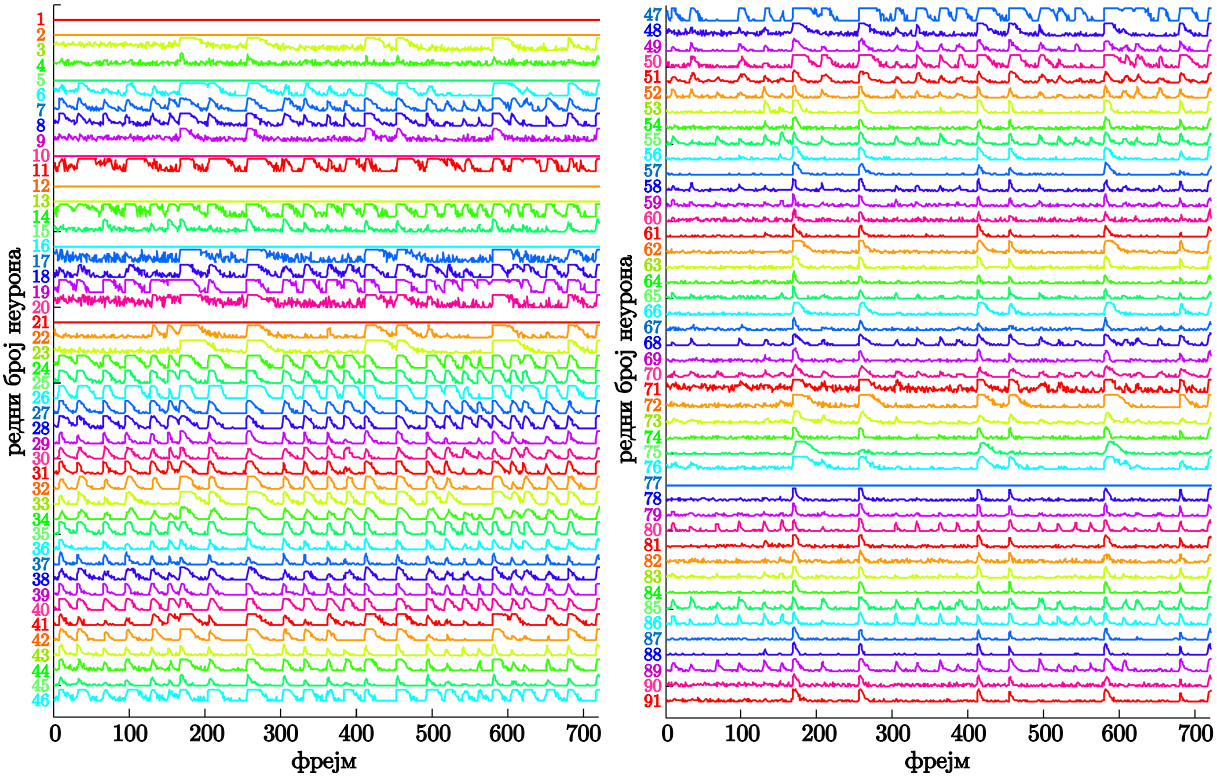
Слика 5.5: Детектовани неурони и екстраховани сигнали са снимка 2. Параметри: $\Delta = 7$, $min\_size = 20$, $max\_size = 1000$.

На слици 5.6.а су приказане детекције неурона на снимку 3, а на слици 5.6.б сигнали који одговарају тим детекцијама. Сигнали су, ради прегледности, приказани само за 720

од 1200 фрејмова.



а) детектовани неурони



б) екстраховани сигнали

Слика 5.6: Детектовани неурони и екстраховани сигнали са снимка 3. Параметри: $\Delta = 7$, $min\_size = 20$, $max\_size = 1000$.

На слици 5.6.б се могу приметити сигнали који су константни (на пример сигнали који потичу од детекција 1, 2, 10). Иако они одговарају неуронима који су исправно детектовани, на њиховим позицијама интензитет флуоре- сценције био је довољно велики да је камера ушла у засићење. Такође, на слици 5.6.а се може видети да постоје дупле детекције (нпр. 49 и 68), али у мањем броју него код снимка 2. Поред тога, због нискофреквентног филтрирања, неки блиски неурони су детектовани као један регион (нпр. детекције 29 и 60). За овакав квалитет снимања калцијумске флуоресценције, сви ови проблеми би били значајно смањени само исправним подешавањем камере, тј. спречавањем да она улази у засићење. Тада не би било потребе за филтрирањем, а и детекције дендрита и аксона (нпр. региони 53, 54, 81 на слици 5.6.а) би биле смањене.

У [1] се за детекцију неурона на слици у основи користи алгоритам базиран на анализи независних компонената описан у уводу. Без посебног улажења у детаље, важно је напоменути да алгоритам развијан у овом раду даје једнако добре резултате као и алгоритам из [1] и [6] за видео секвенцу коришћену у [1] (снимак 3).

Поглавље 6

Закључак

У овом раду је приказан нови приступ за издвајање неуронских сигнала из података добијених снимањем калцијумске флуоресенције. За разлику од свих ранијих приступа који раде постпроцесирање у софтверу, у овом раду је развијен алгоритам намењен за имплементацију у хардверу који са великом тачношћу детектује позиције неурона на слици у реалном времену. Даље се са тих позиција лако издвајају сигнали снимањем интензитета одговарајућих пиксела.

Нови алгоритам за детекцију неурона у основи користи метод максимално стабилних екстремалних региона (*Maximally Stable Extremal Regions*) који је раније имплементиран у хардверу, али за мале димензије слике. Због тога се у овом раду, ради уштеде меморије и повећања брзине детекције, слике деле на делове и користи концепт паралелизма.

Алгоритам је ради тестирања имплементиран делом у програмском језику *C*, а делом у програмском пакету *MATLAB*. Резултати су упоредиви са значајно компликованијим алгоритмима који раде постпроцесирање. Међутим, тема је само започета овим мастер радом и постоји доста простора за даље усавршавање алгоритма.

У даљем раду, први проблеми које је потребно решити су они који настају услед помераја одређених фрејмова у току снимања. Једна од идеја којом би се могло доћи до решења је мењање позиције већ детектованих неурона тако да она прати нове детекције тих истих неурона. За даљи рад се предлаже и примена овог алгоритма на снимке добијене другим техникама, а не само двофотонском микроскопијом. Наравно, крајњи циљ је имплементација целог алгоритма на *FPGA* чипу.

Невезано за издвајање неуронских сигнала, отворено је питање паралелизације *MSER* алгоритма на *FPGA* чипу. Постоје радови и патенти који се баве проблемима паралелизације овог алгоритма на графичким процесорима [24]. Дељењем слике на делове детекција, *MSER* региона на целој слици се компликује, али је идеја вредна разматрања. Поред тога, важна је примена *MSER* алгоритма у рачунарској визији и даљи рад се може позабавити применом хардверске имплементације *MSER* алгоритма за детекцију и препознавање облика у рачунарској визији.

Као што је већ речено, на слици се могу добити неки региони који одговарају дендрити-

ма и аксонима, а не само нуклеусу неурона. Уколико се тим регионима припишу дескриптори који карактеришу њихов облик, оријентацију и димензије, алгоритам би се даље могао развијати у правцу детекције и препознавања различитих делова неурона.

Захвалност

Захваљујем се проф. др Дејану Марковићу, професору на Универзитету Калифорније у Лос Анђелесу (*University of California, Los Angeles*) и Дејану Розгићу, докторанту на Универзитету Калифорније у Лос Анђелесу, на идеји за овај мастер рад, обимним материјалима и несебичној помоћи. Такође, захваљујем се свом ментору доц. др Јелени Поповић-Божовић, као и Јовани Перовић, проф. др Лазару Сарановцу и асистенту Драгомиру Ел Мезенију са Електротехничког факултета у Београду на саветима током израде и/или писања рада.

Аутор

Литература

[1] T. P. Patel, K. Man, B. L. Firestein, and D. F. Meaney, *Automated quantification of neuronal networks and single-cell calcium dynamics using calcium imaging*, Journal of Neuroscience Methods, vol. **243**, pp. 26–38, **2015**.

[2] S. Reichinnek, A. von Kameke, A. M. Hagenston, E. Freitag, F. C. Roth, H. Bading, M. T. Hasan, A. Draguhn, and M. Both, *Reliable optical detection of coherent neuronal activity in fast oscillating networks in vitro*, NeuroImage, vol. **60**, no. **1**, pp. 139–152, **2012**.

[3] X. Li, G. Ouyang, A. Usami, Y. Ikegaya, and A. Sik, *Scale-free topology of the CA3 hippocampal network: A novel method to analyze functional neuronal assemblies*, Biophysical Journal, vol. **98**, no. **9**, pp. 1733 – 1741, **2010**.

[4] I. Ozden, H. M. Lee, M. R. Sullivan, and S. S. Wang, *Identification and clustering of event patterns from in vivo multiphoton optical recordings of neuronal ensembles*, Journal of Neurophysiology, vol. **100**, no. **1**, pp. 495 – 503, **2008**.

[5] I. Valmianski, A. Y. Shih, J. D. Driscoll, D. W. Matthews, Y. Freund, and D. Kleinfeld, *Automatic identification of fluorescently labeled brain cells for rapid functional imaging*, Journal of Neurophysiology, vol. **104** no. **3**, pp. 1803–1811, **2010**.

[6] E. A. Mukamel, A. Nimmerjahn, and M. J. Schnitzer, *Automated analysis of cellular signals from large-scale calcium imaging data*, Neuron, vol. **63**, no. **6**, pp. 747–760, **2009**.

[7] R. Maruyama, M. Kazuma, M. Hiroyoshi, and A. Toru, *Detection of cells from calcium imaging data using non-negative matrix factorization*, Proceedings of the 21st Annual Conference of the Japanese Neural Network Society, **2011**.

[8] F. Diego, S. Reichinnek, M. Both, and F. A. Hamprecht, *Automated identification of neuronal activity from calcium imaging by sparse dictionary learning*, IEEE 10th International Symposium on Biomedical Imaging, **2013**.

[9] M. J. Berridge, P. Lipp, and M. D. Bootman, *The versatility and universality of calcium signalling*. Nature Reviews Molecular Cell Biology, vol. **1**, no. **1**, pp. 11–21. **2000**.

[10] C. Grienberger and A. Konnerth, *Imaging calcium in neurons*, Neuron, vol. **73**, no. **5**, pp. 862–885, **2012**.

[11] F. D. Andilla and F. A. Hamprecht, *Learning Multi-level Sparse Representations*, Advances in Neural Information Processing Systems **26**, **2013**.

[12] F. D. Andilla and F. A. Hamprecht, *Sparse Space-Time Deconvolution for Calcium Image Analysis*, Advances in Neural Information Processing Systems **26**, **2013**.

[13] J. Matas, O. Chum, M. Urban, and T. Pajdla, *Robust Wide Baseline Stereo from Maximally Stable Extremal Regions*, Proceedings of the British Machine Vision Conference (BMVC), London, UK, pp. 384–393, **2002**.

[14] R. E. Tarjan, *Efficiency of a Good But Not Linear Set Union Algorithm*, Journal of the ACM, vol. **22**, no. **2**, pp. 215-225, **1975**.

[15] R. Sedgewick and K. D. Wayne, *Algorithms*, 4th ed. Addison-Wesley, **2011**.

[16] F. Kristensen and W.J. MacLean, *Real-time extraction of maximally stable extremal regions on an FPGA*, IEEE International Symposium on Circuits and Systems, **2007**.

[17] L. Najman, M. Couprie, *Quasi-linear algorithm for the component tree*, Proceedings of SPIE – Vision Geometry XII, vol. **5300**, pp. 98–107, **2004**.

[18] M. Donoser, H. Bischof, *Efficient Maximally Stable Extremal Region (MSER) Tracking*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. **1**, pp. 553 - 560, **2006**.

[19] L. Gómez, D. Karatzas, *MSER-Based Real-Time Text Detection and Tracking*, 22nd International Conference on Pattern Recognition (ICPR), **2014**.

[20] A. Vedaldi, B. Fulkerson, K. Lenc, D. Perrone, M. Perdoch, M. Sulc, H. Sarbortova, *VLFeat open source library*, **2007**–**2013**. dostupno na: www.vlfeat.org

[21] mathworld.wolfram.com/AckermannFunction.html

[22] D. Nistér, H. Stewénius, *Linear Time Maximally Stable Extremal Regions*, Proceedings on 10th European Conference on Computer Vision – ECCV, vol. **5303**, part II, pp. 183-196, **2008**.

[23] *OpenCV – Open Source Computer Vision Library*, dostupno na: opencv.org

[24] N. Grujic, and I. Mihajlovic, *Determining Maximally Stable External Regions Using a Parallel Processor*, MotionDSP, Inc., assignee. Patent US20140286579 A1. 25 Sept. **2014**.

Додатак А

Систем за снимање калцијумске флуоресценције на *SD* картицу

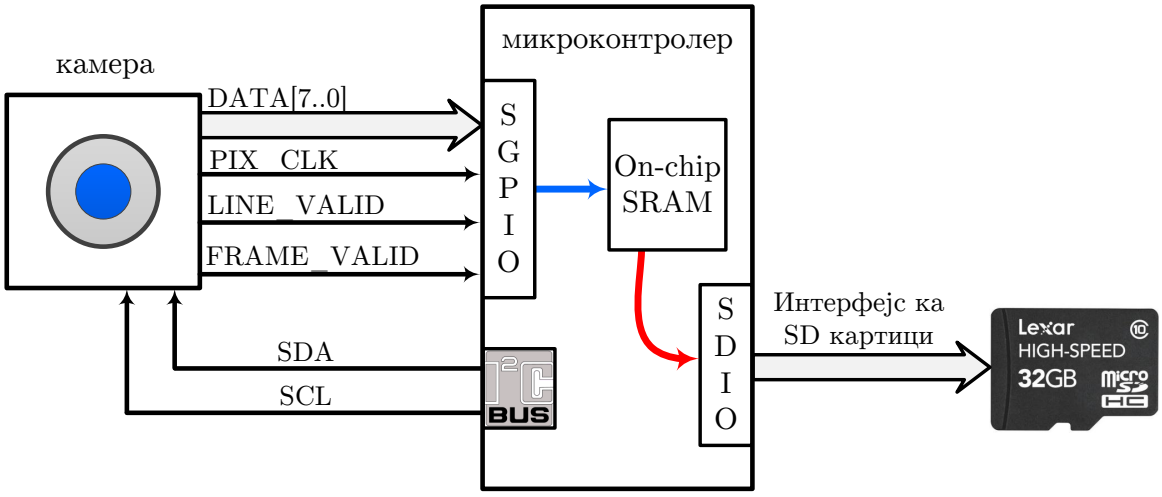
Као што је речено у уводу овог мастер рада, за потребе снимања калцијумске флуоресценције направљен је систем који податке са сензора камере чува на микро *SD* картицу. Циљ овог пројекта је да омогући снимање флуоресценције коришћењем уређаја фиксираног на главу миша, али тако да кретање миша, осим тежином уређаја за снимање, није ограничено. Предвиђено је да миш, поред малог фиксираног микроскопа и сензора на глави, на леђима носи плочицу са *SD* картицом, микроконтролером, пратећим компонентама и батеријом. Због тога, систем не сме да буде тежак, тј. максимална дозвољена тежина је између 5 и 10 грама.

Уређај који садржи мали микрокоп и сензор је развијен на Одељењу за неурологију Универзитета Калифорније у Лос Анђелесу[I], а плочица која омогућава снимање података са сензора на *SD* картицу је направљена током праксе на Универзитету Калифорније у Лос Анђелесу у групи професора Дејана Марковића [II].

За ову апликацију коришћен је *NXP*-ов микроконтролер из $LPC43xx$ фамилије ($LPC4337JET100$). Микроконтролери из ове серије су базирани на *ARM Cortex-M*4 процесору који може користити независни *ARM Cortex-M*0 као копроцесор. *ARM Cortex-M*4 може да ради на учестаностима до 204 MHz, а у овом пројекту је подешено да ради на 192 MHz. $LPC4337$ има укупно 136 kB *SRAM* меморије подељене у неколико секција различите величине. Микроконтролер има *SDIO* периферију која је коришћена за упис на *SD* картицу и врло корисну периферију серијски *GPIO* (*SGPIO*) која се може користити за имплементацију различитих, за микроконтролере нестандардних паралелних и серијских комуникационих интерфејса. Подешавања сензора су рађена уписом у контролне регистре преко I^2C серијског интерфејса. Основни блок дијаграм система је дат на слици А.1.

[I] golshanilab.neurology.ucla.edu

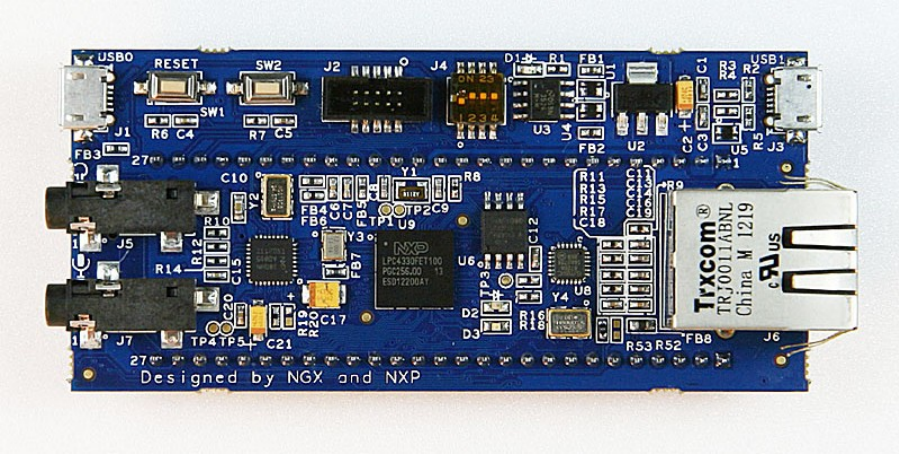
[II] icslwebs.ee.ucla.edu/dejan/researchwiki



Слика А.1: Основна блок шема система за снимање калцијумске флуоресценције

Фотоосетљиви сензор који је коришћен је *CMOS* сензор $MT9V034$ [III], који може да дâ слике максималне резолуције 752×480 пиксела при максималној брзини смењивања слика од 60 fps. Интерфејс преко кога се очитавају вредности пиксела је приказан у одељку 4.1. Једина разлика је што сензор даје 10-битну вредност за сваки од пиксела. У овом раду је коришћено само горњих 8 бита због брзине смештања података на *SD* картицу која је тада дупло већа, што је последица начина адресирања микроконтролера.

Цео *firmware* за микроконтролер, најпре је развијен и тестиран на $LPC4330 - Xplorer$ развојној плочи[IV](слика А.2).



Слика А.2: $LPC4330 - Xplorer$ развојна плоча

У наставку овог додатка описана је *SGPIO* периферија и очитавање података са камере коришћењем ове периферије. Затим је дат кратак опис уписа података на *SD* картицу. На крају, описан је цео систем и дате шеме које одговарају лејауту плочице.

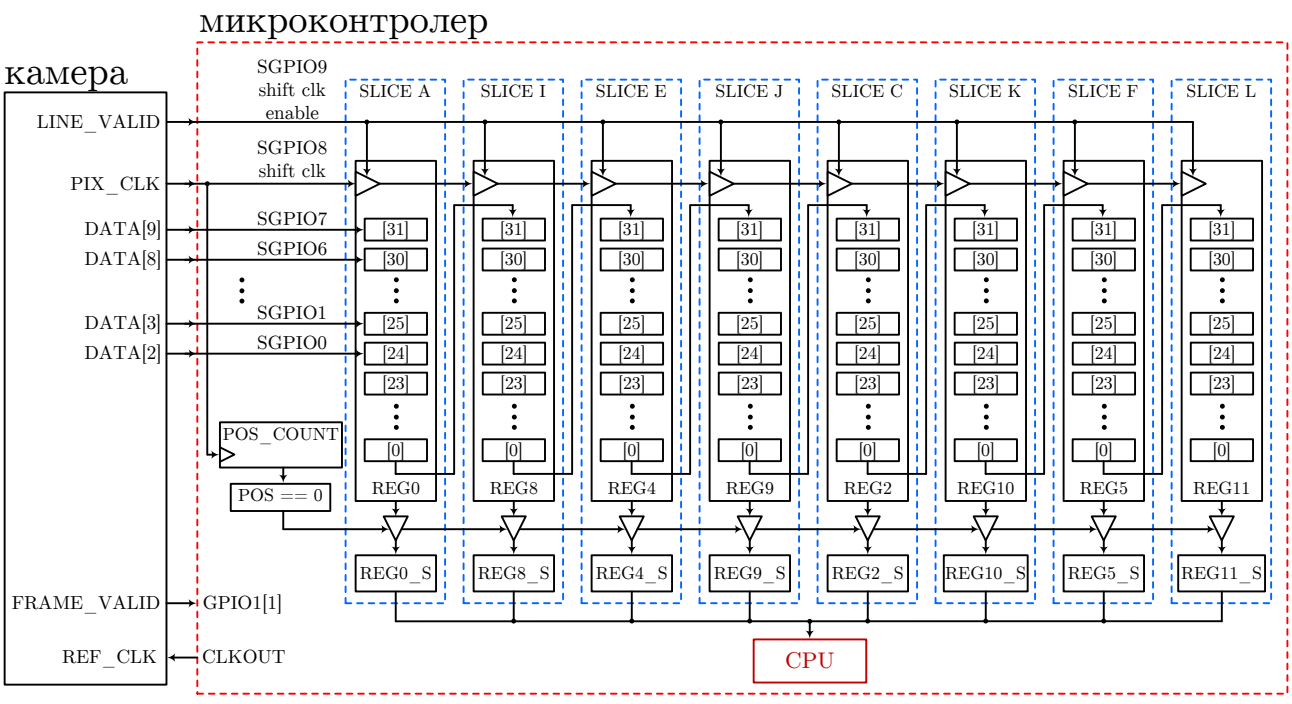
[III] $MT9V034 : 1/3 - Inch\,Wide - VGA\,Digital\,Image\,Sensor\,Features$, MT9V034_DS, Rev. A 10/08 EN, Aptina Imaging Corporation, **2008**.

[IV] www.nxp.com/board/OM13027.html

A.1 *SGPIO* периферија и емулација интерфејса камере

Серијска *GPIO* периферија омогућава стандардне *GPIO* функционалности са проширењима која омогућавају убрзање рада са подацима који треба да се пошаљу или приме серијским трансфером. Конкретно, интерфејс камере је 8-битни синхрони серијски интерфејс. То значи да подаци о вредностима пиксела стижу серијским путем преко 8-битне паралелне магистрале. *SGPIO* периферија је погодна за пријем ових података јер садржи такозване слајсеве (*slices*). Сваки слајс има 32 једнобитна регистра (*REG*) повезана у ланац, тј. у један 32-битни померачки регистар. На сваки тактни сигнал, подаци из регистра се померају удесно за 1, 2, 4 или 8 позиција у зависности од подешавања периферије. Слајсеви се могу увезати у ланац тако да представљају сви заједно један велики померачки регистар, па се у њега могу учитавати вредности пиксела и тек кад се регистар напуни, очитати све вредности, што омогућава процесору дуже временске интервале за обраду пристиглих или неких других података.

На слици А.3 је приказана блок шема ове периферије у конфигурацији која одговара примени у овом пројекту, а то је емулација интерфејса према камери. Микроконтролер је извор сигнала такта за сензор камере. На основу тог сигнала, сензор камере генерише такт PIX_CLK на чије узлазне ивице треба очитавати линије за податке. Управо овај такт се доводи као такт за померачке регистре у слајсевима. Укупно је у ланац повезано 8 слајсева од 32 бита, а при свакој узлазној ивици PIX_CLK сигнала такта се у горњих 8 бита регистра $REG0$ из првог слајса у ланцу упише вредност пиксела. Претходно се све већ уписане вредности у ланац померачких регистара помере за 8 бита удесно. Тако се за 32 тактна циклуса, може испунити цео бафер вредностима 32 узастопна пиксела.



Слика А.3: *SGPIO* периферија у конфигурацији за емулацију интерфејса према камери

С обзиром на то да је такт PIX_CLK активан и када је пауза између линија, као $ENABLE$ сигнал за све померачке регистре доводи се сигнал $LINE\_VALID$. Овај сигнал је на активном нивоу само онда када су вредности на $DATA$ линијама заиста вредности пиксела слике.

Сваки слајс има бројач који броји уназад и његова тренутна вредност представља број слободних локација у баферу. На слици је овај бројач обележен са POS_COUNT и када он падне на нулу генерише се прекид процесору који значи да су подаци спремни за читање. Како би се омогућило несметано читање, у сваком слајсу постоји по један 32-битни баферски регистар (*shadow register* – $REGx\_S$). У ове регистре се преписују вредности из померачких регистара када вредност бројача POS_COUNT падне на нулу. Процесор у прекидној рутини очитава вредности из баферских регистара. Приликом прве следеће узлазне ивице такта PIX_CLK, вредност бројача POS_COUNT се ресетује на 31 и даље се све понавља испочетка.

Сигнал $FRAME\_VALID$ се доводи на један од $GPIO$ портова, а он генерише прекид при свакој промени $FRAME\_VALID$ сигнала. Овај прекид служи за исправну синхронизацију почетка новог фрејма. Почетак новог фрејма се детектује увек када се десио прекид $GPIO$ порта на узлазну ивицу сигнала $FRAME\_VALID$[V,VI].

Пре почетка очитавања података, у одређене регистре $CMOS$ сензора су уписани подаци за подешавање, како би се сензор подесио да даје слике резолуције 480×376 пиксела брзином од 15 fps. Ово су максималне перформансе које су могле да се добију због ограничене брзине уписа на SD картицу. Упис података у регистре сензора се ради преко I^2C магистрале. Најпре је подешен параметар хоризонталног затамњења (*Horizontal Blanking*) на велику вредност, чиме је направљена пауза између две линије, што је смањило просечну брзину трансфера података више од два пута. Затим је подешен мод читања (*Read Mode*) тако да се величина слике смањи на пола. Уписом одговарајућег податка у регистар за мод читања, постиже се да сензор шаље сваку другу колону на свој излаз, чим се не мења брзина смењивања слика, али је учестаност такта PIX_CLK дупло мања и брзина преноса пиксела је смањена. Иницијално, сензор ради на такту од 26 MHz, што даје проток корисних података од преко 20 MB/s. Међутим, овај проток је било неопходно смањити јер брзина уписа на SD картицу није довољно велика. Да би се добио горепоменути проток од 15 fps за слике резолуције 480×376 пиксела, потребно је још и смањити радну учестаност сензора на 16 MHz, што се ради након осталих горепоменутих подешавања сензора преко I^2C магистрале.

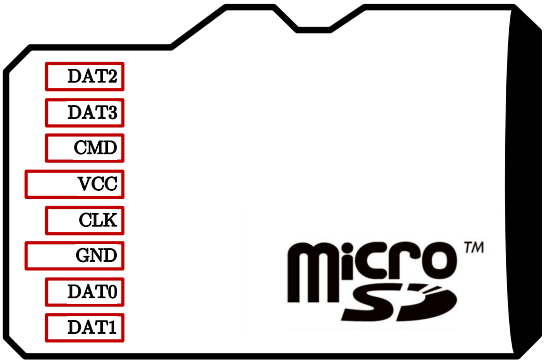
Овим је омогућен пријем података са камере у неку локалну меморију микроконтролера, а затим трансфер тих података на SD картицу коришћењем $SDIO$ периферије.

V *LPC43xx ARM Cortex-M4/M0 multi-core microcontroller*, UM10503, Rev. 1.8, NXP Semiconductors, **2014**.

VI *LPC4300 camera interface design using SGPIO*, AN11196, Rev. 1.1, NXP Semiconductors, **2012**.

A.2 Упис података на *SD* картицу

За упис података на микро *SD* картицу користи се *SDIO* периферија микроконтролера. Ова периферија контролише упис и читање података преко *SD* интерфејса. Сигнали, као и положај пинова који одговарају тим сигналима и напајању на микро *SD* картици приказани су на слици А.4.



Слика А.4: Распоред пинова на микро *SD* картици. Поглед је одозго, а пинови се налазе са доње стране.

Интерфејс је синхрони и има 4 линије за податке ($DAT0$–3), а контрола уписа и читања на картицу се врши преко линије за команде (CMD). Упис и читање се могу радити у једнобитном моду, када се користи само линија ($DAT0$) или у 4-битном моду, када се користе све линије за податке, а подаци се преносе у нибловима.

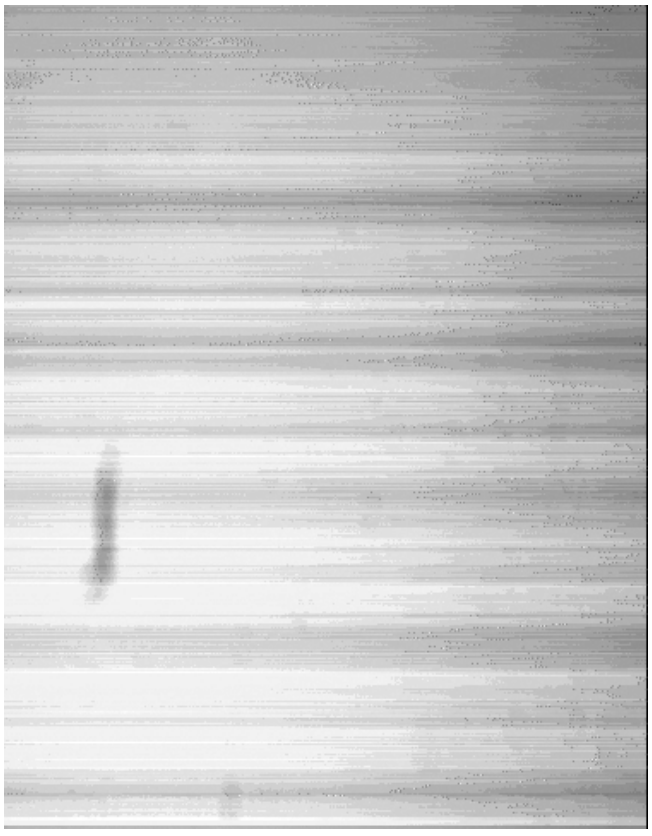
SDIO периферија микроконтролера има интерни DMA контролер којим се омогућава пренос података из меморије у периферију без асистенције процесора, а који је пожељно користити када год се ради блоковски пренос података. У овом пројекту је коришћен блоковски пренос, а за упис и читање је коришћен 4-битни мод.

Сваки фрејм је величине 480×376 пиксела, што је 176,25 kB података. Укупан капацитет $SRAM$ меморије микроконтролера је 136 kB, што значи да је упис података на *SD* картицу потребно започети када се учита одређени број линија, пошто нема довољно меморије за учитавање целог фрејма. Врло важан недостатак овог микроконтролера је неконзистентност адреса $SRAM$ меморије. Наиме, меморија је подељена у неколико блокова од којих су два локална, којима $ARMCortex - M4$ приступа преко своје локалне магистрале, а осталима се приступа преко AHB магистрале на коју су прикључене и неке периферије. Један локални блок је величине 32 kB и у њему је главна секција за податке у току рада. Други локални блок је величине 40 kB, а блокови на AHB магистрали су величине 32 kB или мањи. Пошто DMA контролер приступа конзистентним адресама, није могуће да се започне један трансфер за више од 40 kB података. Такође, да не би дошло до конфликта приликом читања података са камере и њиховог смештања у $SRAM$ меморију са DMA преносом из $SRAM$ меморије у *SDIO* периферију, и даље на *SD* картицу, увек се одређени број линија учита у један блок, а исти број наредних линија у други блок. Тако се у току учитавања другог сета линија са камере, први сет линија уписује на *SD* картицу. Због тога се наизменично уписују по 83 линије у један, па други блок $SRAM$ меморије од којих је први локални величине 40 kB, а други блок са AHB магистрале, величине 32 kB. У главном програму постоји бројач који броји учитане линије

и када год се учитају нове 83 линије, започне се упис података на *SD* картицу преко *SDIO* периферије.

Коришћена је *Lexar High-Performance MicroSDHC* 300x 32GB *UHS-I/U*1 микро *SD* картица која подржава *Ultra High Speed* стандард који омогућава велике брзине уписа и читања података, али *SDIO* периферија микроконтролера не подржава овај стандард, па је због тога брзина преноса података морала бити смањена.

На *SD* картици није формиран фајл систем, па се очитавање података на рачунару не може урадити на стандардан начин. За то је коришћен програм *HxD - Freeware Hex Editor and Disk Editor*[VII] којим се подаци са картице чувају у бинарни фајл на рачунару. За формирање слика из бинарног фајла, написана је $MATLAB$ скрипта. Систем није тестиран на животињи, али се на слици А.5 може видети један фрејм сачуван на картицу. Сензору је додато врло мало оптике, па нема јасног фокуса, али се може видети зрнце прашине на стаклу.



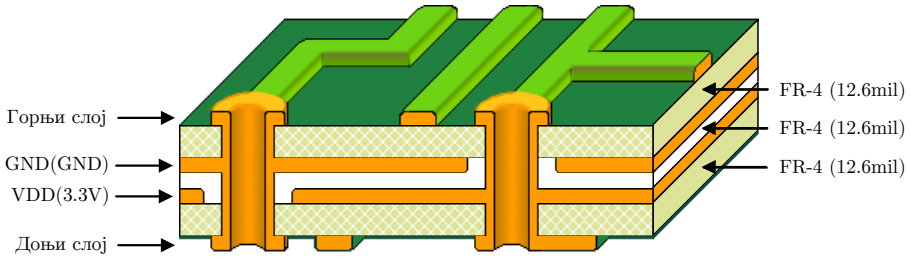
Слика А.5: Један фрејм сачуван на *SD* картицу

Ради лакше контроле система, предвиђена је комуникација са корисником коришћењем $UART$ периферије. На терминалу се на почетку исписују статусне поруке. На пример, корисник се обавештава ако *SD* картица није убачена на своје место. На дизајнираној плочици је предвиђено да се након иницијализације серијска веза преко $UART$-а прекине, како би миш могао слободно да се креће.

[VII] mh-nexus.de/en/hxd/

A.3 Дизајн плочице

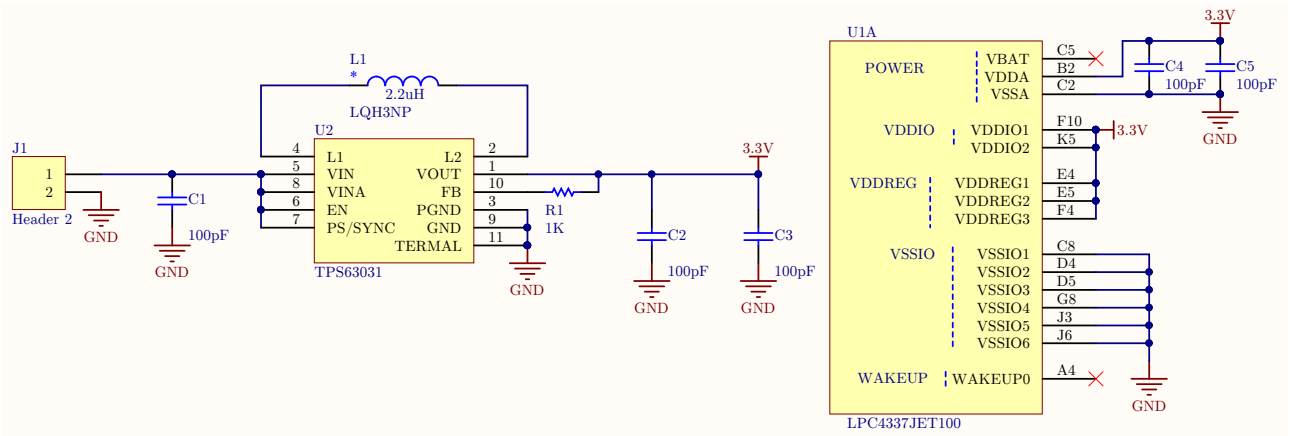
Након реализације *firmware*-а и теста на развојној плочи, направљена је плочица димензија $2,5 \times 2,5$ cm коју је потребно повезати са сензором оптичког система фиксираног на главу миша. Плочица је урађена у 4 слоја метала од којих је један узет за масу, а други за напајање (слика А.6).



Слика А.6: Коришћени слојеви у дизајну плочице

Као што је већ речено, коришћени микроконтролер је $LPC4337JET100$. Кућиште чипа је типа $TFBGA100$ и димензија $9 \times 9 \times 0,7$ mm. У даљем тексту и на сликама које следе, микроконтролер је компонента $U1$.

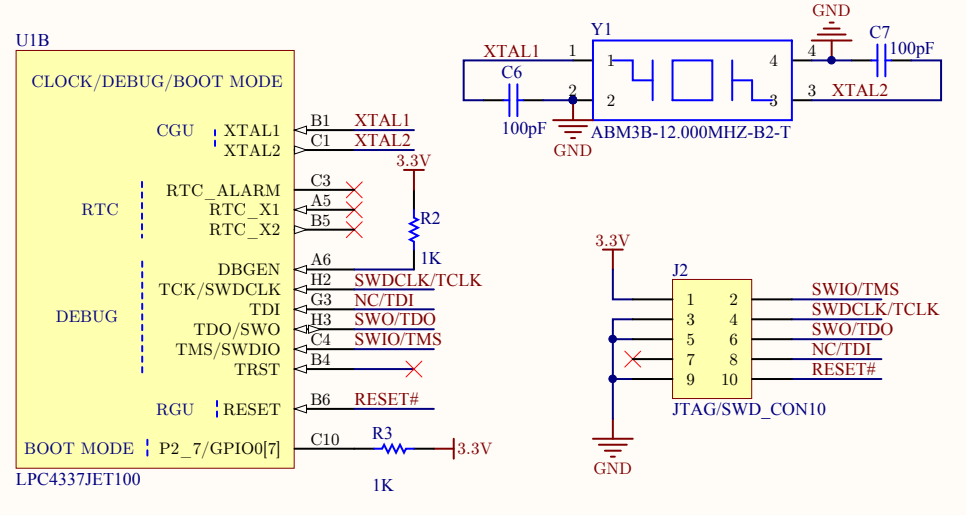
Након одабира микроконтролера, што је најважнији део, могу се одабрати остале компоненте, а најпре компоненте за напајање. Плочица се напаја из батерије, а пошто упис на SD картицу може да буде енергетски захтеван, најбољи однос капацитета и димензија дају литијум-полимерске батерије. Конкретно, коришћена је *Lectron Pro* 3.7V 100mAh 20C *Lipo* батерија. За стабилизацију напона, коришћен је $TPS63031$ *buck-boost* конвертор[VIII] који стабилизује излазни напон на 3,3 V. Излаз конвертора се води на метални слој који одговара напајању. Електрична шема повезивања *buck-boost* конвертора и целог напајања са микроконтролером приказана је на слици А.7. Сви кондензатори на шеми су керамички кондензатори.



Слика А.7: Напајање плочице

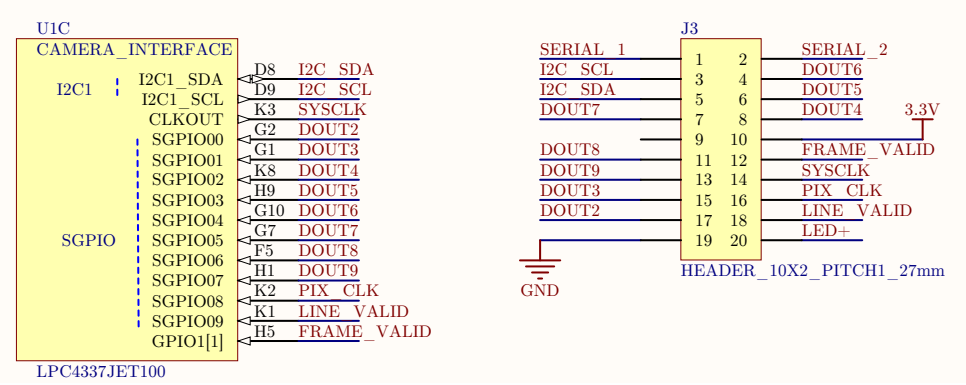
[VIII] www.ti.com/product/TPS63031/technicaldocuments

Микроконтролер има интегрисан интерни осцилатор, али је због поузданости и тачности коришћен екстерни осцилатор. За потребе генерисања такта коришћен је *Abracon LLC ABM3B-12.000MHZ-B2-T* кристал[IX] који кад се повеже са два кондензатора као на слици А.8 генерише сигнал такта учестаности 12 MHz. За програмирање микроконтролера потребно је обезбедити *JTAG* конектор. Како би плочица била мање тешка и како би конектор заузимао мање места, уместо стандардног *JTAG SWD* конектора коришћен је *Harwin M50* конектор[X]. Електрична шема повезивања кола за генерисање такта са микроконтролером, као и конектора за програмирање приказана је на слици А.8.



Слика А.8: Шема повезивања осцилатора и *JTAG* конектора са микроконтролером

На слици А.9 је приказана шема повезивања микроконтролера са конектором за сензор камере. Конектор је исти као и конектор за програмирање микроконтролера, само са већим бројем пинова. Приметити да на шеми нема *pull-up* отпорника за I^2C магистралу. То је због тога што су отпорници већ били постављени на плочици где је залемљен сензор.

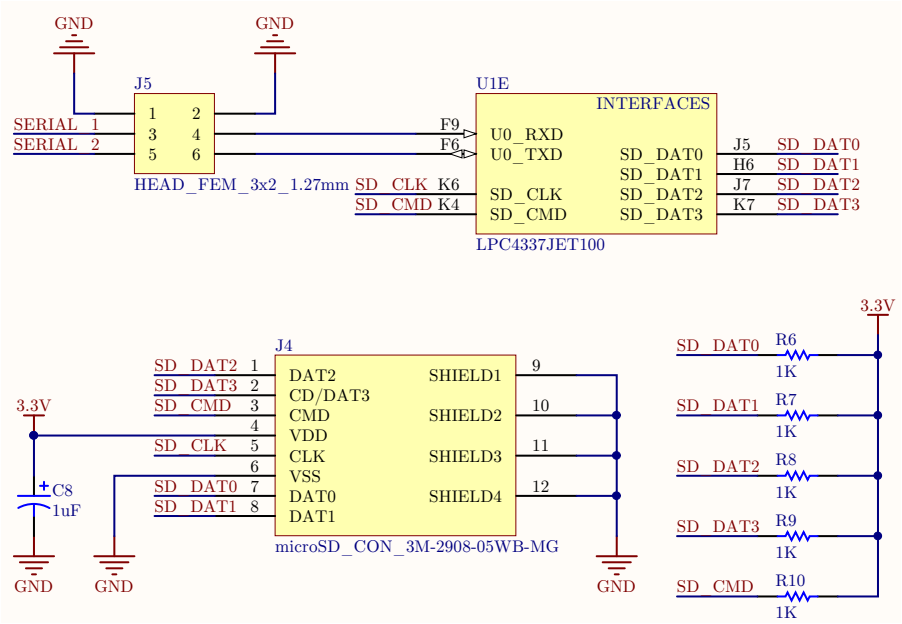


Слика А.9: Шема повезивања микроконтролера са конектором за сензор камере

На слици А.10 је приказана шема повезивања конектора за *UART* и конектора за *SD* картицу са микроконтролером. Све линије за податке и линија за команду интерфејса

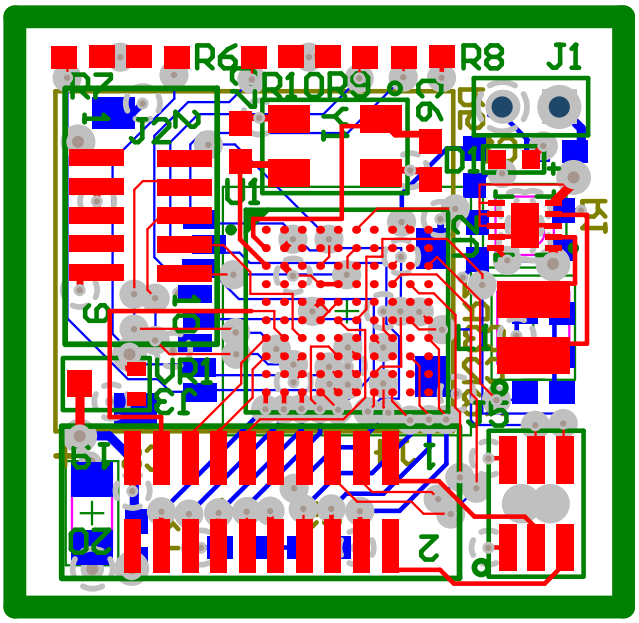
[IX] www.abracon.com/Resonators/abm3b.pdf
[X] cdn.harwin.com/pdfs/M50-360.pdf

за упис и читање података са *SD* картице за исправно функционисање захтевају *pull-up* отпорнике. Такође, поред конектора за *SD* картицу уз контакт за напајање постављен је тантални кондензатор који обезбеђује да не долази до пропада напона напајања ако приликом читања или уписа *SD* картица нагло повуче велику струју.

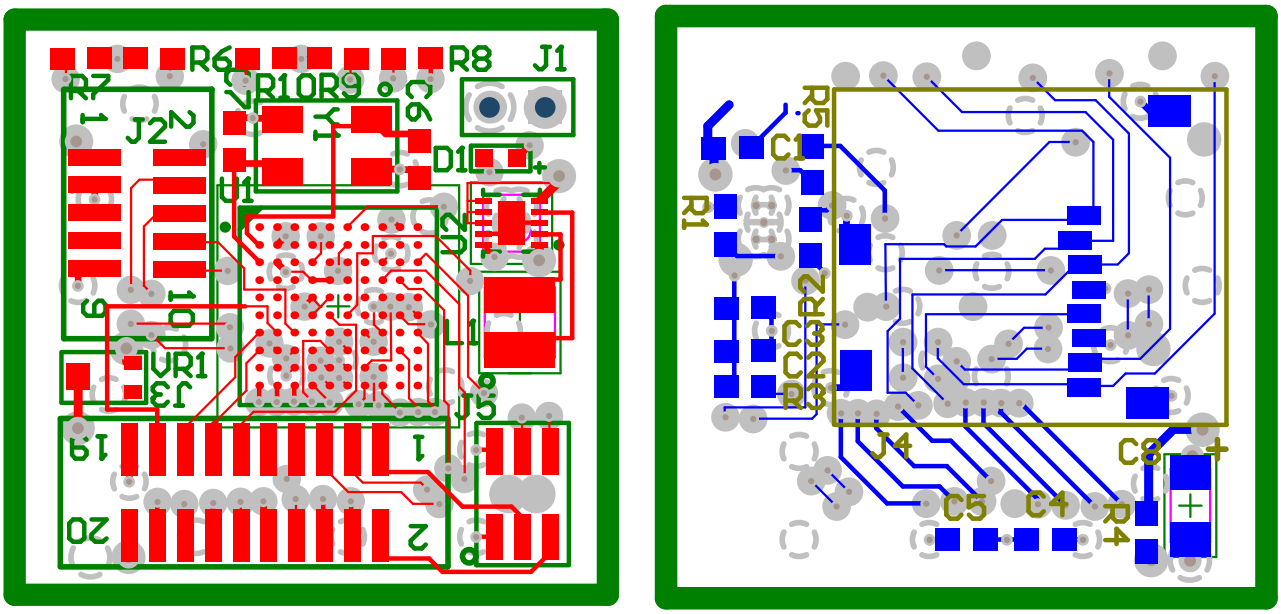


Слика А.10: Шема повезивања конектора за *UART* и конектора за *SD* картицу са микроконтролером

На слици А.11 приказан је лејаут дизајна плочице израђен у алату *Altium Designer* 6, а на слици А.12 су приказани одвојено горњи и доњи слојеви лејаута.



Слика А.11: *PCB* дизајн лејаута



а) *PCB* дизајн - поглед одозго

б) *PCB* дизајн - поглед одоздо

Слика А.12: *PCB* дизајн лејаута са активним горњим слојевима (а) и са активним доњим слојевима (б)