|  |
| --- |
| Роберт Шандор  **Реализација софтверске магистрале за дистрибуцију видео сигнала у возилу на „Adaptive AUTOSAR“ платформи**  МАСТЕР РАД  Нови Сад, 2019 |

|  |  |  |  |
| --- | --- | --- | --- |
| Редни број, **РБР**: | |  | |
| Идентификациони број, **ИБР**: | |  | |
| Тип документације, **ТД**: | | Монографска документација | |
| Тип записа, **ТЗ**: | | Текстуални штампани материјал | |
| Врста рада, **ВР**: | | Дипломски – мастер рад | |
| Аутор, **АУ**: | |  | |
| Ментор, **МН**: | |  | |
| Наслов рада, **НР**: | |  | |
| Језик публикације, **ЈП**: | | Српски / ћирилица | |
| Језик извода, **ЈИ**: | | Српски | |
| Земља публиковања, **ЗП**: | | Република Србија | |
| Уже географско подручје, **УГП**: | | Војводина | |
| Година, **ГО**: | |  | |
| Издавач, **ИЗ**: | | Ауторски репринт | |
| Место и адреса, **МА**: | | Нови Сад; трг Доситеја Обрадовића 6 | |
| Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога) | |  | |
| Научна област, **НО**: | | Електротехника и рачунарство | |
| Научна дисциплина, **НД**: | | Рачунарска техника | |
| Предметна одредница/Кqучне речи, **ПО**: | |  | |
| **УДК** | |  | |
| Чува се, **ЧУ**: | | У библиотеци Факултета техничких наука, Нови Сад | |
| Важна напомена, **ВН**: | |  | |
| Извод, **ИЗ**: | |  | |
| Датум прихватања теме, **ДП**: | |  | |
| Датум одбране, **ДО**: | |  | |
| Чланови комисије, **КО**: | Председник: |  |
|  | Члан: |  | Потпис ментора |
|  | Члан, ментор: | Проф. Др. Милан З. Бјелица |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual printed material | |
| Contents code, **CC**: | | Masters Thesis | |
| Author, **AU**: | |  | |
| Mentor, **MN**: | |  | |
| Title, **TI**: | |  | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian | |
| Country of publication, **CP**: | | Republic of Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | |  | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | Novi Sad, Dositeja Obradovica sq. 6 | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | |  | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Computer Engineering, Engineering of Computer Based Systems | |
| Subject/Key words, **S**/**KW**: | |  | |
| **UC** | |  | |
| Holding data, **HD**: | | The Library of Faculty of Technical Sciences, Novi Sad, Serbia | |
| Note, **N**: | |  | |
| Abstract, **AB**: | |  | |
| Accepted by the Scientific Board on, **ASB**: | |  | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: |  |
|  | Member: |  | Menthor's sign |
|  | Member, Mentor: |  |  |

**Захвалност**

**Садржај**

[1. Увод 1](#_Toc28443488)

[2. Теоријске основе 4](#_Toc28443489)

[2.1 *AUTOSAR* 4](#_Toc28443490)

[2.1.1 *AUTOSAR* *Classic* 5](#_Toc28443491)

[2.1.2 *AUTOSAR Adaptive* 6](#_Toc28443492)

[2.1.2.1 *Execution* *Management* 9](#_Toc28443493)

[2.1.2.2 *State* *Management* 10](#_Toc28443494)

[2.1.2.3 *Communication* *Management* 10](#_Toc28443495)

[2.2 Магистрале 12](#_Toc28443496)

[2.2.1 Хардверске магистрале 12](#_Toc28443497)

[2.2.2 Софтверске магистрале 12](#_Toc28443498)

[2.3 Хардверска платформа 13](#_Toc28443499)

[2.3.1 *ALPHA Automotive Development platform* 14](#_Toc28443500)

[3. Концепт решења 17](#_Toc28443501)

[3.1 Софтверска магистрала за дистрибуцију видео сигнала 17](#_Toc28443502)

[3.2 Слој за апстракцију камера наменске платформе 19](#_Toc28443503)

[3.3 Слој за апстракцију дистрибуције добављеног видео сигнала 20](#_Toc28443504)

[3.4 Слој за информисање о карактеристикама прибављеног видео сигнала 23](#_Toc28443505)

[3.5 Резиме функционалности концепта решења 24](#_Toc28443506)

[4. Програмско решење 26](#_Toc28443507)

[4.1 Реализација апстракције наменске платформе са камерама 27](#_Toc28443508)

[4.2 Реализација апстракције дистрибуирања видео сигнала кроз систем 30](#_Toc28443509)

[4.2.1 Реализација апстракције руковања фрејмовима - *FrameAccessEngine* 31](#_Toc28443510)

[4.2.2 Реализација апстракције добављања фрејмова – *FrameAccessClient* 35](#_Toc28443511)

[4.3 Реализација спреге за информисање о квалитативним карактеристикама сигнала у оквиру адаптивне платформе 36](#_Toc28443512)

[4.3.1 Опис конфигурације спреге за информисање у адаптивној платформи 37](#_Toc28443513)

[4.3.2 Апликативна програмска спрега за руковање слојем за дистрибуцију квалитативних информација видео сигнала 38](#_Toc28443514)

[5. Резултати 40](#_Toc28443515)

[5.1 Мерење брзине чувања једног фрејма у зони дељене меморије *Linux* оперативног система 40](#_Toc28443516)

[5.2 Мерење времена добављања фрејма камере 41](#_Toc28443517)

[5.2.1 Мерење времена потребно за чување једног фрејма у дељеној меморији *Linux* оперативног система 41](#_Toc28443518)

[5.2.2 Мерење времена потребног за достављање једног фрејма посредстом сервиса за информисање о доступности и квалитету сигнала 42](#_Toc28443519)

[6. Закључак 44](#_Toc28443520)

[7. Литература 45](#_Toc28443521)

**Списак слика**

Слика 2.1 Детаљни приказ чланова *AUTOSAR* конзорцијума [7] 5

Слика 2.2 Приказ комуникације у *SOA* адаптивне платформе [12] 8

Слика 2.3 Преглед доступних компоненти адаптивне платформе [12] 9

Слика 2.4 Дијаграм прелаза стања приликом покретања *ЕМ* 9

Слика 2.5 *ALPHA* развојна платформа [5] 14

Слика 2.6 Блок дијаграм *TDA2x* система на чипу [20] 15

Слика 2.7 Организација хардверских компоненти на *ALPHA* платформи [5] 15

Слика 3.1 Дијаграм концепта решења софтверске магистрале 18

Слика 3.2 Приказ зависности слоја за апстракцију хардвера и хардверске платформе 19

Слика 3.3 Дијаграм зависности функционалне компоненте нижих слојева 21

Слика 3.4 Дијаграм интеракције међусобно зависних слојева 22

Слика 3.5 Дијаграм прелаза стања при иницијализацији магистрале 23

Слика 3.6 Дијаграм зависности адаптивне апликације за обраду видео сигнала од функционаних компоненти софтверске магистрале 24

Слика 4.1 Структура пројекта софтверске магистрале 27

Слика 5.1 Време уписа једног фрејма у зону дељене меморије у us 41

Слика 5.2 Време копирања фрејма из дељене меморије наменске платформе у дељену меморије слоја за дистрибуцију фрејмова 42

Слика 5.3 Време потребно локалној апликацији адаптивне платформе да прибави фрејм, од момента његове доступности 43

Слика 5.4 Време потребно локалној апликацији адаптивне платформе да преузме фрејм из зоне дељене меморије 43

**Списак табела**

Табела 4.1 Опис поља потребних за конфигурисање камера и меморијских зона 28

Табела 4.2 Методе за конфигурацију и покретање камера намеске платформе 29

Табела 4.3 Методе за комуникацију са вишим слојевима софтверске магистрале 30

Табела 4.4 Методе *FrameAccessEngine* класе за њену иницијализацију 32

Табела 4.5 Методе *FrameAccessEngineSHMUtil* класе за њену иницијализацију 32

Табела 4.6 Опис метода за руковање дистрибуцијом фрејмова класе *FrameAccessEngine* 33

Табела 4.7 Опис метода за руковање дељеном меморијом класе *FrameAccessEngineSHMUtil* 34

Табела 4.8 Метода класе *FrameAccessClient* за њену иницијализацију 35

Табела 4.9 Методе за руковање инстанцом класе *FrameAccessClient* 36

Табела 4.10 Преглед метода адаптивне апликације за руковање иницијализацијом софтверске магистрале 39

Табела 4.11 Методе за руковање информацијама софтверске магистрале 39

**Скраћенице**

**ADAS – A**dvanced **D**river-**a**ssistance **s**ystems – Напредни системи за помоћ возачу у вожњи

**API** – **A**pplication **P**rograming **I**nterface – апликативна програмска спрега/интерфејс

**ARA** – **A**UTOSAR **R**untime for **A**daptive **A**pplications – *AUTOSAR* извршно окружење за апликације адаптивне платформе

**АUTOSAR - AUT**omotive **O**pen **S**ystem **A**rchitecture – Стандард отворене архитектуре у аутомобилској индустрији

**CM** – **C**ommunication **M**anagement

**CAN – C**ontroller **A**rea **N**etwork – Контролер мрежног региона

**DSP** – **D**igital **S**ignal **P**rocessor - наменски процесор за обраду дигиталног сигнала

**EM – E**xecution **M**anagement

**IP – I**nternet **p**rotocol – Интернет протокол

**IPC - I**nter **P**rocess **C**ommunication – Међупроцесна комуникација

**JTAG** - **J**oint **T**est **A**ction **G**roup

**LiDAR – Li**ght **D**etection **a**nd **R**anging – детекција објеката и њихове удаљености путем светлосног снопа

**OEM** – **O**riginal **E**quipment **M**anfuacturer – произвођач оригиналне опреме

**PCIe - P**eripheral **C**omponent **I**nterconnect **E**xpres**s**

**POSIX – P**ortable **O**perating **S**ystem **I**nterface – Интерфејс преносивог оперативног система

**RADAR – Ra**dio **D**etection **a**nd **R**anging – детекција објеката и њихове удаљености путем радио таласа

**RGB – R**ed **– G**reen **– B**lue – формат енкодовања видео сигнала, где сваки од канала представља вредност једне боје (црвене, зелене и плаве)

**RPC** - **R**emote **P**rocedure **C**all – Позивање функционалности удаљеног модула

**RTOS** *-* **R**eal-**t**ime **o**perating **s**ystem – оперативни систем за рад у реалном времену

**SDK – S**oftware **D**evelopment **K**it – Развојни пакет

**SOA** – **S**ervice **O**riented **A**rchitecture – сервисно оријентисана софтверска архитектура

**SM** – **S**tate **M**anagement

**SoC - S**ystem **o**na **C**hip – систем заснован на чипу

**UARТ - U**niversal **A**synchronous **R**eceiver **T**ransmitter – Универзални асинхрони пријемник-предајник

**USB – U**niversal **S**erial **B**us – универзална серијска веза

**YUV –** формат енкодирања видео сигнала где **Y** компонента представља осветљај, а **U** и **V** компоненте представљају боје.

# Увод

Решавајући бројне изазове аутономне вожње, аутомобилска индустрија данашњице се у многоме ослања на помоћ напредних алгоритама за помоћ возачу (енг. *ADAS -**Advanced Driver-assistance systems*). Иако се *ADAS* алгоритми разликују по степену безбедности који морају да испуне, свака информација од значаја мора правовремено бити достављена, како самом систему који управља возило, тако и самом путнику. Функционална безбедност [1] самих алгоритама се значајно разликује. Тако, одређени алгоритми могу бити информативног типа, попут оних који приказују окружење возила у току његовог мировања [2], док са друге стране постоје алгоритми за избегавање судара [3], чије отказивање може имати катастрофалне последице по путнике. Велики број оваквих алгоритама се развија и тестира у контролисаним лабораторијским условима. Ипак, крајњи циљ како произвођача возила, тако и твораца ових алгоритама, јесте њихово извршавање на платформама специјализованим за аутономну вожњу.

Како би извршавање сваког од ових алгоритама било оптимално, потребно је на циљним платформама извршити дистрибуцију послова, попут прикупљања, припреме и обраде информација од интереса. Овим се уводи јасна граница између различитих сензора, попут камера, ласерских (енг. *LiDAR – Light Detection and Ranging*) и радио даљинометра (енг. *RADAR – Radio Detection and Ranging*), уређаја који преводе сигнал из аналогног у дигитални и циљне платформе која врши обраду сигнала и екстрахује корисне информације из истог.

Овакав приступ омогућује оптималан рад и уску специјализацију сваке од компоненти у систему. Ипак, иако су перформансе сваке од компоненти значајно побољшане, архитектура система је хетерогена и као један од већих проблема истиче се дистрибуција података. Произвођачи хардверских компоненти тренутно немају униформни одговор на овај проблем, у својим решењима ослањају се на хардверске магистрале великих брзина преноса података, које најчешће прати програмска подршка затвореног кода.

Ипак, зарад олакшавања развоја и стандардизације софтвера за аутономну вожњу стварају се конзорцијуми који окупљају велике произвођаче аутомобила и аутомобилске индустрије. Један такав конзорцијум представља *AUTOSAR* [4]*,* који окупља произвођаче аутомобила као што су *BMW, Volkswagen, Toyota, Chrysler* и произвођаче опреме као што су *Bosch, Continental, Siemens VDO.* Циљ овог конзорцијума јесте развој стандарда којег сваки произвођач мора да се придржава, али има слободу у имплементацији свог решења. Један од стандарда који овај конзорцијум развија, *Adaptive AUTOSAR*, тиче се платформи које имају велику процесну моћ и намењене су управо за *ADAS* алгоритме.

Како је овај стандард још увек у настајању, те захтеви који морају бити испуњени нису строго дефинисани, овај рад представља једно могуће решење дистрибуције видео сигнала у оквиру *AUTOSAR Adaptive* платформе. Рад се ослања на употребу софтверске магистрале која је реализована апстракцијом етернет (енг. *ethernet*) магистрале и дељене меморије опертивног система, као и апстракције платформе која врши прикупљање видео сигнала, зарад вршења дистрибуције истог кроз остатак система. Као циљна платформа, употребљена је Алфа [5] развојна плоча (енг. *ALPHA Automotive Development platform)*, заснована на *TDA2x* систему на чипу (енг. *SoC, System on a Chip*), компаније *Texas Instruments*.

Овај рад је организован у следећих 5 целина:

1. **Теоријске основе –** преглед *AUTOSAR* стандарда и *AUTOSAR Adaptive* платформе. Опис магистрала, као и поређење софтверских и хардверских магистрала. Опис хардверске платформе на којој је тестирано идејно решење.
2. К**онцепт решења -** поглавље даје увид у реализоване модуле софтверске магистрале. Преглед и опис слојева за апстракцију хардверске платформе.
3. **Програмско решење –** реализација модула софтверске магистрале.
4. Т**естирање и валидација –** поглавље даје увид у перформансе реализоване софтверске магистрале као и њену функционалну употребу у два различита сценарија.
5. **Закључак –** ово поглавље даје преглед реализоване софтверске магистрале, резултата који су остварени, као и кораци за даље унапређење софтверске магистрале.

# Теоријске основе

Ово поглавље даје увид у теоријске основе неопоходне за разумевање реализованог решења. Посебна пажња посвећена је софтверској платформи *AUTOSAR Adaptive* на којој је заснована ова реализација софтверске магистрале за дистрибуцију видео сигнала. Како је *AUTOSAR Adaptive* платформа сервисно оријентисане архитектуре, део поглавља објашњава парадигме заступљене у оваквом развоју софтвера. Поред тога, направљен је осврт на хардверске и софтверске магистрале, као и на хардверску платформу на којој је реализовано и тестирано ово решење.

## *AUTOSAR*

*AUTOSAR* представља конзорцијум који окупља велике произвођаче возила, као и опреме за возила. Оформљена 2003. године, ова групација је имала за циљ да смањи како трошкове производње, тако и потребан труд и време за преношење постојећих решења на друге платформе [6]. У оквиру овог конзорцијума, прави се разлика између група чланова. Подела се врши на следећи начин:

1. Чланови оснивачи (енг. *Core Partner)* – чине девет првобитних чланова који су започели формирање овог конзорцијума.
2. Премијум чланови (енг. *Premium Members)* – чини их 46 чланова, где они могу бити произвођачи возила, произвођачи оригиналне опреме (енг. *Original Equipment Manufacturers, ОЕМ*), произвођачи различитих софтверских алата, полупроводничких компоненти, итд. Ови чланови су се касније прикључили конзорцијуму.
3. Партнери за равој и производњу (енг. *Development partners*) – ову групу чини 28 чланова који су задужени за развој компоненти које задовољавају прописане стандарде ове групације.
4. Сарадници (енг. *Associates and Attendees*) – последњу групацију у овој хијерархији чине компаније које сарађују са члановима свих осталих група и користе тренутно доступни стандард.

Слика 2.1 даје детаљнији увид у хијерархију ове групације.



Слика 2.1 Детаљни приказ чланова *AUTOSAR* конзорцијума [7]

Ова групација изродила је два стандарда. Први, старији, *AUTOSAR Classic* и други,  *AUTOSAR Adaptive*, који се још увек развија. Зарад стварања јасне слике и намене, како једног, тако и другог стандарда, следећа два потпоглавља позабавиће се сваким од њих, како би се поставила основа за решење које овај рад износи.

### *AUTOSAR* *Classic*

Овај стандард представља првобитну платформу коју је оформљени конзорцијум створио. Циљна хардверска платформа подразумева микроконтролерске јединице које имају малу процесорску моћ, ограничене меморијске ресурсе. На оваквим платформама извршава се оперативни систем за рад у реалном времену (енг. *Real-time operating system*, *RTOS*). Задатак ових система јесте вршење временски критичних радњи, обзиром да сваки оперативни систем за рад у реалном временом има јасно дефинисане оквире у којима се одређена операција може извршити. Управо су овакве хардверске и софтверске платформе идеалне за разноврсне функционалност које се тичу управљања возила и на њима се заснивају неки од система возила, попут кочионог.

Иако се овај стандард показао као поуздан у пракси, обзиром да га многобројни произвођачи аутомобила управо уграђују у своја возила, недостатак овог стандарда и софтверске платформе јесте његова непримењивост на хардверске платформе велике процесне моћи.

Потреба за великом процесном моћи јавља се као последица унапређења самих сензора, који су сада у могућности да прикупе значајно више информације из свог окружења и самим тиме омогуће алгоритмима на које ослања возач да буду знатно прецизнији. Ова појава за последицу има потребу за новим стандардом, који би подржао како софтверске, тако и хардверске платформе.

*AUTOSAR Adaptive* управо представља овакав стандард. Стварањем њега, не укида се потреба за постојањем *AUTOSAR Classic*. Циљ стандарда, а и самог конзорцијума јесте коегзистирање ових двају платформи.

Наредно поглавље детаљније описује *AUTOSAR Adaptive.*

### *AUTOSAR Adaptive*

Као што је већ напоменуто, ова платформа представља искорак аутомобилске индустрије у смеру стандардизације софтверске платформе која ће омогућити знатно већу процесорску моћ и комуникацију са спољним светом.

Задатак самог конзорцијума јесте стварање стандарда који ће омогућити комуникацију разноврсних електронских контролних јединица (енг. *ECU – Electronic Control Unit*) путем етернет магистрале [8]. Управо 100-мегабитни и гигабитни етернет представљају основе за пренос информација кроз систем и управо је циљ да оне преузму примат у комуникацији. Претходно коришћене магистрале у систему неће бити избачене из употребе, већ ће њихова намена бити ограничена. Управо употреба етернет магистрале омогућује флексибилнији приступ у протоку информација које стижу са разноврсних сензора. Поред тога, етернет магистрала овде омогућује и софтверске подршке на електронским компонентама, у току животног века једног возила, што претходна платформа није пружала.

Поред напретка у комуникацији који овај стандард доноси, велики напори су уложени у омогућавању употребе процесора и компоненти високих перформанси, како би потребе комплексних алгоритама биле задовољене. Управо овај стандард у својој имплементацији стога не представља засебни оперативни систем, већ искључиво његово проширење. Као што је случај код *AUTOSAR Classic* платформе, тако *AUTOSAR Adaptive* платформа подржава већи број хардверских платформи на којима ће се она налазити.

Како би овакво проширење могло бити примењено на оперативни систем, потребно је да систем подржава *POSIX* [9]стандард, специфично *POSIX51.* Како *Linux* оперативни систем задовољава овај стандард, он представља један од оперативних система који је погодан за ово проширење.

Ово проширење оперативног система са собом носи *AUTOSAR* извршно окружење за апликације адаптивне платформе (енг. *ARA – AUTOSAR Runtime for Adaptive Applications*). Оно пружа решења које се тичу основних функционалности апликација, као што су системски позиви, комуникација са локалним и удаљеним сервисима, као и сервисима које пружа сама платформа. Овај апликативни интерфејс (енг. *API* – *Application programing interface*) пружа могућност лаке интеграције већ постојећих апликација у саму адаптивну платформу. Поред тога, постојећи *ARA* *API* је стандардом дозвољено проширити уколико нека од потребних компоненти за функционисање апликација не улази у скуп функционалности *POSIX51*.

Адаптивна платформа је заснована на сервисно оријентисаној софтверској архитектури (енг. *SOA* – *Service Oriented Architecture*) [10]. Одлика овакве архитектуре огледа се могућности постојања већег броја независних софтверских компоненти, тј. сервиса, који посредством мреже или међупроцесне комуникације размењују информације [11]. Оваква софтверска архитектура омогућује велику модуларност и поновну употребу већ постојећег изворног кода из потребне измене. Због своје модуларности, ова архитектура се врло често користи у *Web* технологијама.

Механизам комуникације између два сервиса дат је на слици 2.2.



Слика 2.2 Приказ комуникације у *SOA* адаптивне платформе [12]

Као што се на слици може приметити, потребно је да постоји је постоји једна софтверска компонента која прати постојање сервиса који пружају различите услуге, тј. информације. Компоненте које пружају услуге (енг. *Provider*), морају да буду видљиве остатку система. То чине тиме што се представе (енг. *Register*) модулу који је задужен за праћене доступних сервиса. Сервиси који зависе од информација, такозвани претплатници (енг. *Subscriber*), прво морају претражити доступне сервисе. Уколико је тражени сервис доступан, тада је могуће добити вршити удаљене позиве њихових функционалности (енг. *RPC - Remote Procedure Call*).

Адаптивна платформа се ослања на овај принцип рада. Доступност сервиса адаптивне платформе је такође могуће мењати у току времена рада саме платформе (енг. *Runtime*) као и у току дизајнирања сервиса. Особине сервиса покрива датока која описује сваки од доступних сервиса (енг. *Service Instance Manifest*). Ова датотека се испоручује са самим извршним кодом, како би се јасно нагласили сви протоколи које апликација користи у својој комуникацији. Поред овог манифеста, који је од већег значаја за овај рад, у адаптивној платформи се користе још два. *Execution manifest* чији је задатак да опише када се који сервис и апликација извршава на адаптивној платформи и *Machine Manifest*, чији је задатак да пружи детаљнији опис саме хардверске платформе и стања у којима ова платформа може да се налази.

Слика 2.3 даје детаљнији увид у сервисе адаптивне платформе који су доступни.



Слика 2.3 Преглед доступних компоненти адаптивне платформе [12]

Детаљнији преглед функционалности адаптивне архитектуре дат је у следећим поглављима, кроз опис функционалности сервиса и модула који су од значаја за овај рад.

#### *Execution* *Management*

Ова компонента представља модул задужен покретање, како платформе, тако сервиса и апликација које се налазе на платформи. Приликом покретања оперативног система, ова компонента је прва прозвана. Њен задатак је да, из манифеста који описују платформу и остале апликације, распореди када је потребно да се која апликација покрене. *EM* нема за задатак да врши распоређивање апликација у току њиховог извршавања, како је то дужност оперативног система, чије је адаптивна платформа проширење. Дијаграм прелаза стања дат је на слици 2.4.



Слика 2.4 Дијаграм прелаза стања приликом покретања *ЕМ*

Поред свог задужења покретања адаптивне платформе и њених апликација у оквиру хладног (енг. *Cold boot*)или поновног покретања (енг. *Reboot/Restart*) оперативног система, *ЕМ* је компонента која је такође задуже за руковање животним циклусом апликација адаптивне платформе. Ова компонента, на основу информација које су доступне из датотека *Machine Manifest* и *Execution Manifest*, доноси закључак која апликација, односно сервис, треба да буде покренута у зависности од стања у којем се како адаптивна, тако и сама хардверска платформа налази. *EM* не врши прелазе платформе из једног стања у друго, већ се на то ослања на другу компоненту, *State Management* (*SM*).

Како је задужен за праћење стања адаптивних апликација и платформе, *ЕМ* је такође задужен и за враћање апликација у њихово претходно стање (енг. *Recovery*) када платформа прелази из једног стања у друго.

#### *State* *Management*

Ова компонента представља део адаптивне платформе који је задужен за праћење и прелазак између дефинисаних стања. *SM* је компонента која свој финални облик добија од стране систем интегратора и зависи од саме хардверске платформе за коју је адаптивна платформа намењена. Као што је већ речено, ова компонента у многоме зависи од саме платформе и намене исте, те стандард не поставља детаљне захтеве за реализацију ове компоненте, већ само обавезује ову компоненту да обавести остатак система преко којих интерфејса се са њом комуницира.

За комуникацију са остатком система, *SM* се ослања на функционалности компоненте *Communication Management* (*CM*). Ова компонента задуже је комуникацију апликација и сервиса адаптивне платформе.

#### *Communication* *Management*

Као што је већ споменуто, ова компонента је задужена за комуникацију осталих компоненти у систему. Задатак *CM* јесте апстракција начина комуникација једне апликације, тј. сервиса са апликацијама и сервисима који се налазе на истој адаптивној платформи или чак на некој другој, удаљеној адаптивној платформи.

Начин комуникације апликација, тј. сервиса адаптивне платформе могуће је одредити у тренуцима дизајнирања апликације, њеног покретања или у току њеног самог рада, променом конфигурације апликације, односно сервиса.

Ова компонента рукује са три типа података, методама (енг. *Methods*), догађајима (енг. *Events*) и пољима (енг. *Fields*). Методе представљају функцију, процедуру или рутину која се прозива овим путем. Догађаји представљају излазне вредности позива метода и на њих реагују одговарајуће компоненте, уколико је потребно. На крају, поље мора представља статус адаптивних апликација и као такве морају увек имати дефинисане вредности, како би се оне могле изчитати или променити.

Ова компонента се ослања на SOME/IP [13], који представља скалабилни протокол за комуникацију у аутомотив индустрији. Сам протокол се користи за слање контролних информација. Значајан је за слање информациј са камера, *AUTOSAR* сертификованих уређаја и многобројних других. У зависности од његове имплементације, могуће је користити овај протокол чак и у сврхе информација и забаве (енг. *Infotainment*).

## Магистрале

Магистрала (енг. *Communication bus*) представљају физичку компоненту чија је намена повезивање екстерних или интерних уређаја процесором. Под магистралом, у рачунарству, подразумевају се подједнако софтверске компоненте, тачније комуникациони протоколи, и хардверске компоненте, као што су оптичка влакна, каблови и многобројни други медијуми. Како би било могуће руковање самом магистралом, потребно је да постоји руковалаоц (енг. *Driver*) који пружа одређену дозу апстракције саме магистрале, како би њена употреба била лакша, за крајњег корисника.

Наредна два поглавља даће мало детаљнији опис ова два типа магистрала.

### Хардверске магистрале

Хардверске магистрале представљају директну везу између процесора и компоненте која може бити интерна или екстерна у односу на читав систем. Овај тип магистрала, поред потребе за постојањем физичког медијума преко којег се врши комуникација, такође захтева и постојање руковалаоца. Управо реализација руковалаоца олакшава употребу физичког медијума, јер он врши апстракцију комуникационог протокола оваквих магистрала.

Како би се омогућио оптималан рад и функционисање магистрале, потребно је да управо оне буду стандардизоване. Најбољи пример тога јесу магистрале попут универзалне серијске магистрале (енг. *USB – Universal Serial Bus*) [13], *PCI (Peripheral Component Interconnect)* и *PCI-Е (Peripheral Component Interconnect - Express).* Одлика ових магистрала у систему јесте могућност преношења података великим брзинама. Оне се налазе у персоналним рачунарима и њихова намена је разнолика.

Поред магистрала које су карактеристичне за искључиво за персоналне рачунарске системе, на наменским платформама постоје стандардизоване магистрале, које су карактеристичне различите гране индустрије. Тако магистрала попут *CAN (*енг. *Controller Area Network)* [14], *LIN* (енг. *Local Interconnected Network*) [15] представљају неке од стандарда који су карактеристични за аутомобилску индустрију и користе се како за контролне информација са микроконтролера, тако и за информације са различитих сензора.

### Софтверске магистрале

Софтверске магистрале представљају начин на који се остварује међупроцесна комуникација (енг. *IPC -* *Inter process communication*) [16]. Када посматрамо овај тип магистрала, оне не морају поседовати посебан физички медијум, како би се вршила размена информација. Са друге стране, као што је то случај и са хардверским магистралама, потребно је да постоји установљени протокол у комуникацији између пријемне и предајне стране, како би информација која тече системом била валидна.

Како софтверска магистрала не мора поседовати физички медијум за комуникацију, потребно је постојање оперативног система. Тако на пример *Linux*, оперативни систем отвореног кода (енг. *Open source*), нуди механизме комуникације као што је *D-bus* [17] система за размену порука, као и проточна обрада (енг. *Pipes*) [18] подржана у језгру (енг. *Kernel*) самог оперативног система. Поред ових видова комуникације, употреба дељене меморије (енг. *Shared memory*) која је такође подржана у самом оперативном систему или реализација неких других комуникационих протокола који нису део *Linux* оперативног система такође представљају својеврсну реализацију софтверске магистрале.

## Хардверска платформа

Како би наменска хардверска платформа задовољила потребе адаптивне платформе, на којој се заснива ово решење софтверске магистрале, она мора да задовољи неколико критеријума. Прво, као што је већ напоменуто, адаптивна платформа представља проширење оперативног система заснованог на најмање *PSE51* стандарду. Поред тога, платформа такође мора да поседује велику процесорску моћ и специфично за овај рад, мора поседовати камере за добављање видео сигнала који ће бити дистрибуиран кроз систем.

Једна оваква платформа је *ALPHA Automotive Development platform* дизајнирана и развијена на институ за Рачунарску Технику и рачунарске комуникације (РТ-РК).

Следеће поглавље даје детаљнији опис платформе хардверских компоненти и софтверске подршке за дату платформу.

На слици 2.5 налази се *ALPHA* развојна платформа.



Слика 2.5 *ALPHA* развојна платформа [5]

### *ALPHA Automotive Development platform*

Развојна платформа *ALPHA* заснована је на систему на чипу (енг. *SoC*) *TDA2x* [19] компаније *Texas Instruments.* Сама платформа поседује три оваква чипа, где сваки од њих поседује више процесорских компоненти, као што су:

* 2 x *ARM Cortex A15 @* 1176MHz;
* 4 x  *ARM Cortex M4 @* 220MHz;
* 2 x DSP C66x @ 750MHz;
* IVA HD *Coprocessor*;
* GPU SGX544 @ 560MHz.

Поред велике процесорске моћи, платформа подржава повезивање до десет камера за аутомобилску индустрију. Иако на платформи постоје три *SoC* компоненте, хардверски дизајн платформе омогућује директан приступ камерама само двема *SoC* компонентама. На платформи такође постоје три *HDMI* излаза, који су доступни сваком од чипова. На платформи се још налази и *JTAG*, *UART* и *DCAN* конектори. Поред ових магистрала, како би се саобраћај на платформи одвијао са што мањим кашњењем, доступни су гигабитни етернет и *PCIe*.На плочи се налазе и три *MicroSD* слота на којима се налази оперативни систем који се извршава на свакој од три доступне *SoC* компоненте. На слици 2.6 дат је блок дијаграм свих хардверских компоненти поменутог *TDA2x* чипа, док слика 2.7 представља организацију хардверских компоненти на описаној платформи.



Слика 2.6 Блок дијаграм *TDA2x* система на чипу [20]



Слика 2.7 Организација хардверских компоненти на *ALPHA* платформи [5]

Два од три чипа покрећу оперативни систем за рад у реалном времену, *SysBios,* док трећи, преостали покреће оперативни систем *Linux.* Чипови на којима је поменути *SysBios* задужени су за извршавање алгоритама за надгледање стања возача, детекцију слободног простора, детекцију возила у околини и многобројне друге, док је чип на којем се извршава *Linux* задужен за покретање адаптивне платформе и апликација.

*TDA2x* компаније *Texas Instruments* долази са софтверском подршком у виду *Vision SDK* [21]. Он представља начин да се развијани алгоритми брзо интегришу у систем. На овај начин је извршена апстракција хардверске платформе и омогућена је једноставнија проточна обрада података добављених са камера које ова платформа обезбеђује. У његовој позадини, налази се *Links&Chains* радни оквир који омогућује једноставнију реализацију комуникаицје између компоненти у проточној обради. Кроз управо овај алат врши се конфигурација корака које апликације предузима и генерише се изворни код апликације. Њега је потребно још додатно конфигурисати, свим параметрима који су од важности за апликацију, као на пример, број камера који се користи у некој апликацији.

# Концепт решења

Ово поглавље даје опис решења постављеног проблема. Дат је приказ решења софтверске магистрале за дистрибуцију видео сигнала, као и логички приказ сваке од функционалних компоненти реализоване софтверске магистрале. На крају, ово поглавље ће сумирати концепт функционалности овог решења.

## Софтверска магистрала за дистрибуцију видео сигнала

Основна идеја софтверске магистрале за дистрибуцију видео сигнала јесте олакшавање и апстракција достављања овог садржај крајњем кориснику, тј. апликацији која врши манипулацију истог.

Наиме, вршењем апстракције добављања сигнала знатно се смањује вероватноћа недозвољеног приступа одређеним сегментима меморије. На тај начин је елиминсана могућност нарушавања интегритета и квалитета добављених података, јер апликација која врши обраду поменутог сигнала није у могућности да приступи оригиналним подацима, већ користи копију истих. Поред тога, постиже се и апстракција читаве платформе и омогућује се фокус инжењера који развија софтвер базиран на алгоритму за асистенцију возачу. С тога, инжењер, па самим тиме и алгоритам, искључиво мора бити упознат са карактеристикама сигнала који добија.

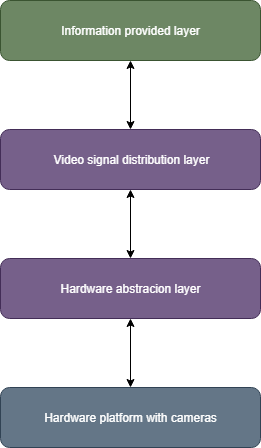
Из приложених проблема, софтверска магистрала најпре мора да пружи једноставан и безбедан приступ потребним информацијама, тј. видео сигналу, како би крајњег корисника исте лишила потребе познавања детаља хардверске платформе на којој се извршава алгоритам. Након тога, потребно је извршити апстракцију достављања видео сигнала кориснику, тј. апликацији која обрађује исти. Овиме се постиже додатни ниво апстракције којим се ограничава писање и читање одређених зона меморије који су намењени искључиво за сам сигнал. На крају, потребно створити пружити информације о сигналу који се доставља, као и његовој доступности. Те информације представљају његов формат (нпр. *YUV422*, *YUV444*, *RGB*, итд.), затим меморијске захтеве за прихватања сигнала, као и у случају да је већи број доступних камера са једне хардверске платформе, информација са које камере, тј. из које перспективе долази сам видео садржај.

Из свих ових захтева уочава се да је потребно решење разложити на три слоја, где постоје следећи слојеви:

1. Слој за апстракцију наменске платформе;
2. Слој за апстракцију дистрибуције добављеног видео сигнала;
3. Слој за информисање о карактеристикама прибављеног видео сигнала.

Овако створени слојеви чине једну реализацију софтверске магистрале на адаптивној платформи, обзиром да своју доступност и функционалности пружа ослањајући се на сервисе адаптивне платформе.

Слика 3.1 представља дијаграм софтверске архитектуре овог решења.



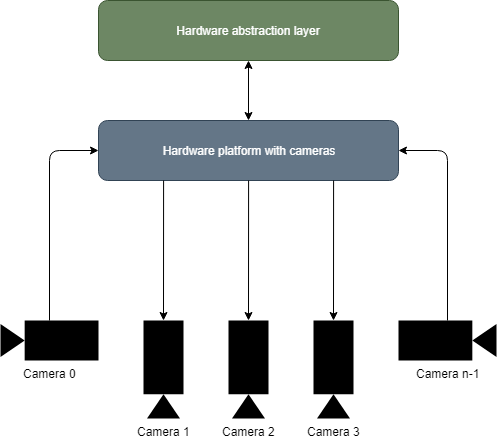
Слика 3.1 Дијаграм концепта решења софтверске магистрале

Следећа поглавља даће детаљнији увид у сваку од наведених компоненти, као и њихове међусобне зависности.

## Слој за апстракцију камера наменске платформе

Као што је већ напоменуто, први корак у реализацији софтверске магистрале представља сама апстракција хардверске платформе на којој се извршава софтверска магистрала. Тиме, инжењер који управо развија неки од алгоритама не мора детаљно да познаје саму хардверску платформу на којој се извршава алгоритам, обзиром да су детаљи попут конфигурације платформе и камера које се налазе на њој сакривени у овом слоју. Такође, инжењер који развија тај алгоритам, није оптерећен ни детаљима складиштења добављених слика са камере, већ то овај слој такође омогућује.

С тога, овај слој софтверске магистрале мора бити уско повезана са хардверском компонентом на којој се извршава. Слика 3.2 представља дијаграм организације хардверске платформе и слоја који је задужен за апстракцију функционалности коју пружа.



Слика 3.2 Приказ зависности слоја за апстракцију хардвера и хардверске платформе

Како је улога овог слоја апстракција, у овом случају *ALPHA* развојне платформе, његов задатак је да од крајњег корисника сакрије читав поступак конфигурације коју је потребно извршити путем *Vision SDK* развојног пакета. Као што је већ описано, на платформу долази унапред генерисани код, управо кроз овај алат. Поред тога, извршена је додатна конфигурација параметара попут броја камера којима се може приступити, као и сам формат добијеног сигнала. Овај слој се ослања на већ постојећу конфигурацију *ALPHA* развојне платформе која обезбеђује шест камера чипу на којем се извршава *Linux.*

Како се читава апликација ослања на руковање меморијом оперативног система *Linux,* овај слој такође мора да обезбеди одговарајућ начин за чување доспелог сигнала са камера. Поред тога, овај слој мора да обезбеди спрегу вишим слојевим како би они приступили меморији у којој је сачуван сигнал. На тај начин, слој за апстракцију дистрибуције видео сингала може једноставно да рукује даље доступним сигналом.

## Слој за апстракцију дистрибуције добављеног видео сигнала

Овај слој софтверске магистрале је, за разлику од претходно описаног, у потпуности независан од хардверске платформе на којој се извршава. Међутим, овај слој се ослања на функционалности *Linux* оперативног система, за који је развијан.

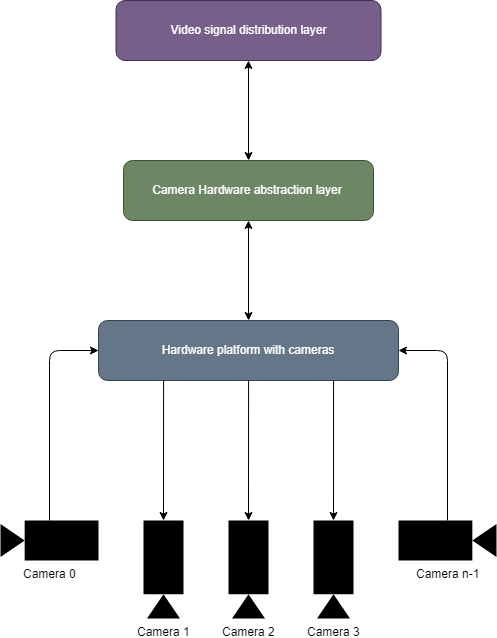
Овај слој има двојаку улогу. Прво, пропагира информације о сигналу са камера до последњег слоја софтверске магистрале. Тиме се омогућује да слој који се налази изнад овог и врши информисање осталих апликација о особинама и доступности сигнала буде реализован као својеврстан сервис у оквиру адаптивне платформе.

Поред тога, друга функционалност коју овај слој испуњава јесте дистрибуција видео сигнала, како до удаљених платформи, тако и у оквиру локалне. Овај слој се ослања искључиво на функционалности које му пружа *Linux* оперативни систем, у виду руковања дељеном меморијом и руковање етернет интерфејсом. На овај начин је могуће остварити међупроцесну комуникацију и пружити апликацијама које врше обраду видео сигнала потребне податке, у виду начина приступа сачуваног сигнала, његовох меморијских захтева, резолуције и формата боја.

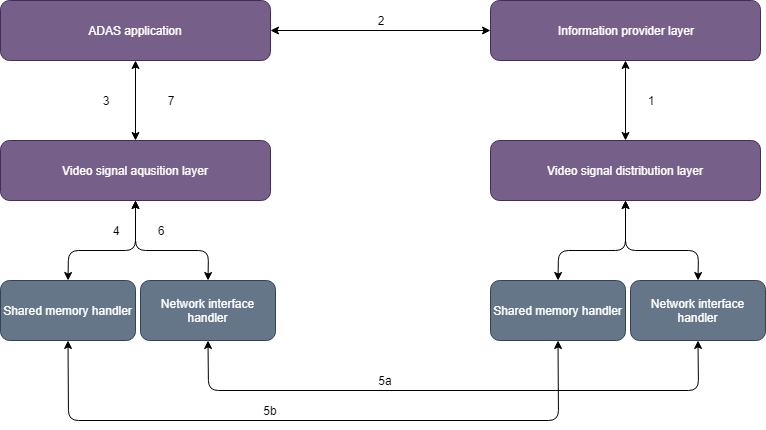
За разлику од претходног слоја, иако логички овај слој представља једну целину, он се функционално састоји из две компоненте. Једна се налази на страни саме хардверске платформе и пружа приступ видео сигналу, као и његовим информацијама кроз највиши слој ове софтверске магистрале, који се представља у виду сервиса адаптивне платформе. Путем дела интегрисаног у адаптивну платформу, остале апликације у систему се могу претплати управо на тај сервис. Пружене информације претплаћеним апликацијама откривају начин приступа потребним подацима. Уколико је апликација на локалној платформи, добија информације о зонама дељене меморије којима мора приступити, односно *IP* адресу и *Port* са којег може добавити видео сигнал.

Друга функционална компонента налази се на страни апликације која врши обраду сигнала. Ова компонента се може налазити како на удаљеној платформи, на којој се извршава апликација за обраду сигнала, тако и на локалној. Независно од начина приступа сигналу, путем дељене меморије или етернет магистрале, апликација која се ослања на ову компоненту мора бити део адаптивне платформе, како би могла да се претплати на слој софтверске магистрале који представља сервис адаптивне платформе.

Слика 3.3 представља дијаграм организације компоненте задужене за руковање меморијом и мрежним интерфејсом на страни хардверске платформе, док слика 3.4 представља дијаграм комуникације виших слојева реализације софтверске магистрале.



Слика 3.3 Дијаграм зависности функционалне компоненте нижих слојева



Слика 3.4 Дијаграм интеракције међусобно зависних слојева

Као што је на дијаграму 3.4 приказано, како би била омогућена размена између слојева за дистрибуцију, потребно је изврпити следеће кораке:

1. Након успешног покретања хардверске и адаптивне платформе, потребно је пружити познате информације о сигналу слоју који врши информисање претлаћених апликација;
2. Омогућити приступ информацијама свакој апликацији која се претплати на емитовани садржај овог сервиса;
3. Пружити информације о начину приступа потребном садржају функционалној јединици која се налази на страни апликације за обраду сигнала (*IP* адреса и *Port*, тј. зона дељене меморије);
4. Конфигурисати потребну функционалну јединицу за приступ (приступ путем етернет модула или приступ дељеној меморији);
5. Приступ потребној информацији путем:
   1. Употребом етернет модула;
   2. Употребом дељене меморије;
6. Прихватити потребну информацију на корисничкој страни;
7. Проследити информацију апликацији за обраду сигнала.

## Слој за информисање о карактеристикама прибављеног видео сигнала

Као што је већ напоменуто у претходним поглављима, последњи слој софтверске магистрале задужен је за пружање квалитативних особина видео сигнала и начина приступа истом, апликацији која врши обраду сигнала. Главна одлика овог слоја јесте његова интеграција у адаптивну платформу. Као таква, ова целина се ослања на комуникациони сервисе адаптивне платформе, *Communication Management* како би комуницирала са осталим апликацијама адаптивне платформе које врше обраду сигнала.

Задатак овог слоја јесте да изврши иницијализацију слоја који зависи од саме хардверске платформе. За ову иницијализацију се користи унапред дефинисана конфигурација, дефинисана помоћу *Vision SDK* алата. Уколико је сама иницијализација овог слоја успешно извршена, тада се може прећи на виши слој, тј. слој задужен за дистрибуцију сигнала.

Како се овај слој ослања на мрежни интерфејс, тј. етернет, и дељену меморију, уколико су они доступни и њихова иницијализација буде успешна, тада овај слој може да прогласи своју доступност осталим апликацијама адаптивне платформе, кроз утврђене механизме.

Слика 3.5 представља дијаграм прелаза стања приликом иницијализације софтверске магистрале.



Слика 3.5 Дијаграм прелаза стања при иницијализацији магистрале

Уколико иницијализација читаве софтверске магистрале буде успешна, овај слој софтверске магистрале може да пријаве доступност својих сервиса адаптивној платформи и на тај начин комуницира са свим сервисима који се на њега претплате. Поред квалитативних информација сигнала, овај слој такође пружа информације начину приступа сигналу, у зависности од тога, да ли је апликација којој је потребан сигнал на удаљеној или локалној платформи.

Слика 3.6 представља дијаграм комуникације апликације адаптивне платформе за обраду видео сигнала и слоја софтверске магистрале, интегрисаног у адаптивну платформу.



Слика 3.6 Дијаграм зависности адаптивне апликације за обраду видео сигнала од функционаних компоненти софтверске магистрале

## Резиме функционалности концепта решења

Као што је описано у претходним поглављима, како би једна адаптивна апликација која врши обраду видео сигнала могла да се ослони на функционалности софтверске магистрале, потребно је да:

1. Слој софтверске магистрале интегрисан у адаптивну платформу који задужен за пружање информација о видео сигналу покрене функционалне слојеве за апстракцију хардверске платформе и дистрибуцију видео сигнала на које се ослања;
2. У зависности од резултата покретања ових слојева, највиши слој софтверске магистрале ће:
   1. Пријавити своју доступност осталим апликацијима адаптивне платформе сервисима адаптивне платформе који су за то задужени;
   2. Ослободити заузете ресурсе у случају неуспешне иницијализације било којег од слојева софтверске магистрале.

Са друге стране, апликације адаптивне платформе које се ослањају на функционалности софтверске магистрале постају свесне њене доступности тек након њеног успешног покретања и пријављивања модулу *Communication Management*. Тада су те апликације у могућности да се пријаве на садржај који највиши слој софтверске магистрале пружа. Тиме, поред добијања квалитативних информација сигнала, ове апликације су такође у могућности да конфигуришу свој модул за приступ видео сигналу. Затим, по приступу овој софтверској магистрали, сама апликација добија потребни садржај који даље користи.

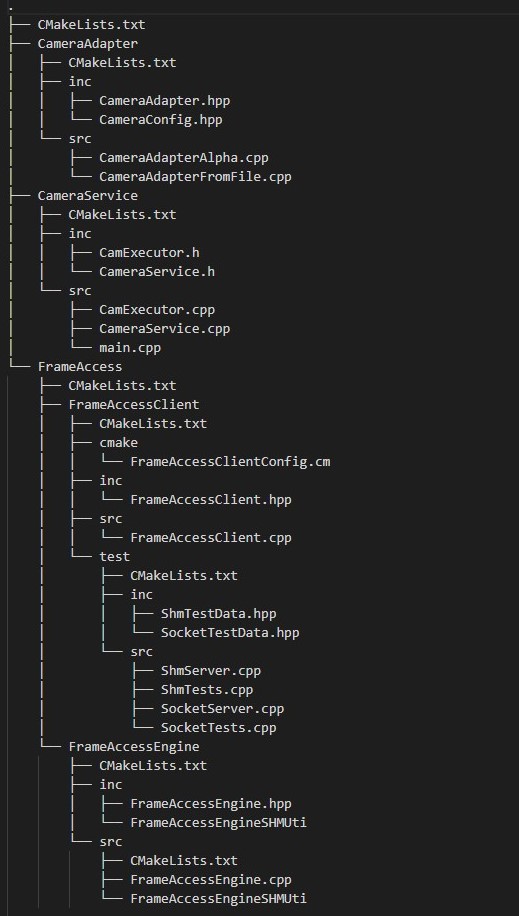
# Програмско решење

У овом поглављу приказана је реализација модула софтверске магистрале на развојној аутомотив платформи *ALPHA*. Приказана је међусобна зависност сваког од модула ове магистрале, као и њена интеграција у адаптивну платформу. Верзија адаптивне платформе коришћена у овом решењу је *R18-03*, тј. ревизија стандарда издата марта 2018. године, и она представља проширење *Linux* оперативног система за ову наменску платформу. Приложено програмско решење прати предложене праксе *AUTOSAR* конзорцијума за развој *C*++ програма [22], које је засновано на *MISRA* [23] стандарду.

Имплементација ове софтверске магистрале реализована ја у виду слојева. Разлог оваквог приступа имплементацији овог решења јесте могућност независног развијања сваке од компоненти и могућност промене комплетне реализације компоненте у зависности од платформе за коју је намењена сама софтверска магистрала.

Поред тога, зарад једноставности коришћења алата за генерисање и превођење програмског кода, коришћен је *CMake* [24] систем. Он представља модернији приступ од до сада коришћеног *Makefile* [25] система и значајно је једноставнији за одржавање.

Сама софтверска магистрала реализована је кроз три модула, где сваки од њих представља додатни ниво апстракције претходног. Коначна структура пројекта ове софтверске магистрале дата је на слици 4.1.



Слика 4.1 Структура пројекта софтверске магистрале

## Реализација апстракције наменске платформе са камерама

Као што је већ описано концептом решења, како би се омогућило лакше руковање видео сигналом крајњем кориснику, потребно је извршити апстракцију наменске платформе која управо добавља сам сигнал. Како за *ALPHA* аутомотив намеску платформу већ постоји решење, у виду библиотеке за *Linux* оперативни систем, које управо омогућује руковање камерама, те је потребно извршити његову апстракцију. Библиотека је написана у програмском језику *C*, те како би се испратиле препоручене праксе за кодирање препоручене за адаптивну платформу при развоју софтверске магистрале, потребно је направити софтверски омотач (енг. *Wrapper*) у програмском језику *C*++. Задатак ове омотачке компоненте је, поред апстракције, пружање униформног начина руковања како овој, тако и било којој другој платформи на коју се пренесе ово решење, обзиром да оно једино зависи од оперативног система *Linux* и адаптивне платформе која је његово проширење.

У реализацији овог софтверског модула, потребно је раздвојити неколико логичких целина. Тако оне чине:

1. Конфигурацију камера и потребних меморијских ресурса на наменској платформи;
2. Униформну спрегу ка вишим слојевима софтверске магистрале.

Како би се направио први корак у апстракцији наменске платформе, реализован је софтверски модул *CameraAdapter* у виду класе *C++* програмског језика. Ова класа зависи од *CameraConfig.hpp* у којем су описани подаци потребни за руковање камерама, у виду кључева меморијских зона дељене меморије, броја пиксела видео сигнала (у даљем тексту фрејма, енг. *Frame* - оквир) по висини и ширини, као број бајтова у једном пикселу. Поред тога, дефинисан је и број камера који се може користити као и величина једног фрејма који камера може да обезбеди.

Табела 4.1 даје увид у конфигурациона поља која су у овом заглављу дефинисана.

|  |  |  |
| --- | --- | --- |
| Поље | Тип | Опи |
| FRAME\_WIDTH | constexpr unsigned int | Број пиксела једног фрејма по ширини. |
| FRAME\_HEIGHT | constexpr unsigned int | Број пиксела једног фрејма по висини. |
| BPP | constexpr float | Број бајтова по једном пикселу фрејма. |
| FRAME\_SIZE | constexpr unsigned int | Величина једног фрејма са камере. |
| MAX\_CAMERAS | constexpr unsigned int | Укупан број доступних камера. |
| CAM\_ID\_# | constexpr char | Кључ за приступање зони дељене меморије. Свака од доступних камера добија своју зону која је намењена за складиштење њених фрејмова. |

Табела 4.1 Опис поља потребних за конфигурисање камера и меморијских зона

Како би се у потпуности онемогућила промена поља која дефинишу карактеристике камера наменске платформе, ова поља су означена *constexpr* кључном речи програмског језика *C++* како би се програмском преводиоцу (у даљем тексту компајлер, енг. *Compiler* – програмски преводилац) експлицитно нагласило да је реч о променљивој која је константна чак и у време превођења програма.

Како би се одговарајућа конфигурација проследила наменској платформи, потребно је позивима метода дефинисаних у *CameraAdapter.hpp*. Класа описана овим заглављем је реализована у виду *Singleton* [26] дизајн шаблона. Овај шаблон гарантује постојање максимално једне инстанце класе, како би се онемогућило вишеструко креирање објеката ове класе и потенцијално постојање вишеструког броја конфигурација.

Методе овог слоја софтверске магистрале које се користе за конфигурацију и покретање дате су у табели 4.2.

|  |  |
| --- | --- |
| Метода | Опис |
| CameraAdapter | Конструктор класе задужен за креирање једине инстанце ове класе. |
| ~CameraAdapter | Деструкор задужен за ослобађање заузетих при инстанцирању објекта класе. |
| GetInstance | Метода задужена за добављање показивача на једину инстанцу ове класе, помоћу којег се рукује камерама наменске платформе. |
| CamInit | Метода задужена за иницијализацију камера наменске платформе. Помоћу ње се приступа библиотеци *ALPHA* аутомотиве платформе како би се започео рад камера. |
| CamDeinit | Метода задужена за прекид рада камера наменске платформе. Помоћу ње се приступа библиотеци *ALPHA* аутомотиве платформе како би се зауставио рад камера и тиме ослободили сви заузети ресурси. |
| CamStart | Метода задужена за руковање изабраном камером. Помоћу ње се конфигуришу меморијска зона у којој се чува добављени фрејм, као и функција која врши чување истог. |
| CamStop | Метода задужена за руковање изабраном камером. Помоћу ње се ослобађају заузети ресурси. |
| GetNumberOfCams | Метода задужена за добављање броја доступних камера вишим слојевима апстракције софтверске магистрале. |

Табела 4.2 Методе за конфигурацију и покретање камера намеске платформе

Приликом покретања ове реализације софтверске магистрале на *ALPHA* аутомотив развојној платформи, потребно је ослонити се на инстанцу објекта класе *CameraAdapter* како би се прво извршила конфигурација потребних ресурса наменске платформе, а након тога и иницијализација модула саме платформе који рукује камерама.

Позивом методе *GetInstance* добавља се инстанца поменуте класе, која је потом у могућности да позивима метода *CamInit* изврши конфигурацију и иницијализацију камера платформе. Након тога, уколико је иницијализација камера успешно извршена, позивом методе *CamStart* врши се покретање рада једне од камера платформе која је одабрана.

Поред иницијализације и конфигурације камера, потребно је омогућити и комуникациону спрегу између камера и виших слојева. Табела 4.3. даје списак метода чији је задатак да поједноставе добављање фрејмова до виших слојева софтверске магистрале.

|  |  |
| --- | --- |
| Метода | Опис |
| CamFrameReady | Метода класе задужена за информисање виших слојева софтверске магистрале о спремности. |
| registerCallback | Метода задужена регистровање фунцкије вишег слоја, којом се виши слој обавештава о доступности фрејма. |
| GetBufferSize | Метода класе задужена за информисање виших слојева о величини фрејма. |

Табела 4.3 Методе за комуникацију са вишим слојевима софтверске магистрале

Након успешне иницијализације, која је покренута из виших слојева ове магистрале, потребно је завршити међусобно увезивање компоненте која врши апстракцију дистрибуције фрејмова. Како би се то учинило, потребно је прво прозвати методу *GetBufferSize* како би се обезбедило довољно меморије за руковање фрејмовима у вишим слојевима. Након тога, позивом методе *registerCallback* врши се увезивање крајње увезивање методе слоја за апстракцију дистрибуције, са овим слојем софтверске магистрале.

Приликом заустављања рада ове компоненте, потребно је прво прозвати методу *CamStop* којом се прекида рад одабране камере. Након успешног ослобађања камере, *CamDeinit* метода ослобађа све заузете ресурсе платформе. На крају, ослобађање преосталих заузетих ресурса ове компоненте је ослобођено имплицитним позивом десктруктора ове класе.

## Реализација апстракције дистрибуирања видео сигнала кроз систем

Зарад постизања једноставне дистрибуције фрејмова, апликацијама које врше обраду видео сигнала, ова софтверска компонента врши апстракцију чувања и преузимања истих. Реализацијом ове компоненте, омогућено је дистрибуирање фрејмова како до апликација адаптивне платформе на истој хардверској платформи, тако и до апликација адаптивне платформе на удаљеној хардверској платформи.

Како би ова компонента била реализована, поред претходно описане интеграције са компонентом која врши апстракцију аутомотив *ALPHA* развојне платформе, потребно је ослонити се на функционалности *Linux* оперативног система. Конкретно, за реализацију овог решења, од значаја је руковање дељеном меморијом [27] и остваривање комуникације путем *Linux* *socket* [28] апликативне програмске спреге.

Поред ових функционалности потребно је, такође, пружити програмску спрегу за руковање овом компонентом, највишем слоју софтверске магистрале, који представља сервис адаптивне платформе.

Овај слој за дистрибуцију реализован је у виду две компоненте. Прва се налази на страни која извршава софтверску магистралу, у овом случају *ALPHA* развојна платформа, док се друга може налазити или на удаљеној платформи или такође на платформи која извршава софтверску магистралу. С тога, овде се посматрају две компоненте:

1. Компонента реализована кроз класу *FrameAccessEngine* – задужена за апстракцију руковања и дистрибуције фрејмова на страни *ALPHA* развојне платформе;
2. Компонента реализована кроз класу *FrameAccessClient* – задужена за добављање фрејмова на удаљеној платформи или у оквиру локалне платформе.

### Реализација апстракције руковања фрејмовима - *FrameAccessEngine*

*FrameAccessEngine* представља класу којом је извршена апстракција дистрибуције. Поред ослањања функционалности које *socket* апликативна програмска спрега пружа, ова компонента се ослања и на *wrapper* класу *FrameAccessEngineSHMUtil*, којом је сакривено руковање дељеном меморијом.

Табеле 4.4 и 4.5 представљају методе класа *FrameAccessEngine* и *FrameAccessEngineSHMUtil* на које се ослања софтверска магистрала како би извршила иницијалнизацију ове компоненте и како би се омогућило руковање платформом на вишем нову апстракције.

|  |  |
| --- | --- |
| Метода | Опис |
| FrameAccessEngine | Конструктор класе, задужен за постављање поља класе подразумеване вредности. |
| ~FrameAccessEngine | Деструктор класе, задужен за ослобађање заузетих ресурса. |
| FrameAccessEngineCommunicationInit | Метода класе задужена за иницијализацију комуникације ослањајући се на *Linux socket*. |
| FrameAccessEngineDetermineBoardIp | Метода класе задужена за аутоматско проналажење *IP* адресе платформе. На основу ње се врши даље конфигурисање модула за комуникацију платформе на којој је се извршава софтверска магистрала. |
| FrameAccessEngineInit | Метода класе задужена за покретање модула на које се ослања ова класа. Покретање модула попут *CameraAdapter*, модула за комуникацију, као и модула руковање дељеном меморијом. Позивом ове методе врши се инцијализација контекста једне камере, тј. ресурса потребних за њено функционисање. |
| FrameAccessEngineDeinit | Метода задужена за ослобађање свих ресурса који се користе за руковање *CameraAdapter* модулом, модулом за комуникацију и модулом за дељену меморију. |
| FrameAccessEngineInitCameraAdapter | Метода класе која експлицитно покреће *CameraAdapter.* Помоћу ње се ова компонента поставља у дефинисано стање. |
| FrameAccessEngineCommunicationDeinit | Метода класе која ослобађа све заузете ресурсе који се тичу дељене меморије, *Linux Socket* функционалности, као и *CameraAdapter* компоненте. |

Табела 4.4 Методе *FrameAccessEngine* класе за њену иницијализацију

|  |  |
| --- | --- |
| Метода | Опис |
| FrameAccessEngineSHMUtil | Конструктор класе, задужен за постављање поља класе подразумеване вредности. |
| ~FrameAccessEngineSHMUtil | Деструктор класе, задужен за ослобађање заузетих ресурса. |
| FrameAccessEngineSHMemInit | Метода класе задужена за иницијализацију руковања дељеном меморијом *Linux* оперативног система.. |
| FrameAccessEngineStoreFrame | Метода класе задужена за руковање фрејмовима који стижу из нижих слојева софтверске магистрале. |
| FrameAccessEngineSHMemDeinit | Метода класе задужена за ослобађање свих заузетих ресурса за руковање дељеном меморијом *Linux* оперативног система. |
| FrameAccessEngineSetSHMZoneSize | Метода задужена за постављање одговарајуће величине меморијске регије дељене меморије *Linux* оперативног система. |

Табела 4.5 Методе *FrameAccessEngineSHMUtil* класе за њену иницијализацију

Приликом инстанцирања класе *FrameAccessEngine*, позивом њеног конструктора, поља инстанце се постављају на подразумеване вредности. Како би се омогућила потпуна функционалност ове класе, позивом *FrameAccessEngineInit* инцијализује се контекст потребан за руковање специфичном камером зарад добијања фрејмова са плаформе. Имплицитно, ова метода прозива друге методе којима се омогућаују остале функционалности. Тако, конструктор класе *FrameAccessEngineSHMUtil* којом се рукује дељеном меморијом, потом позив методе *FrameAccessEngineSHMemInit* која заузима потребне ресурсе дељене меморије, затим *FrameAccessEngineStartCamera* којом се започиње рад изабране камера платформе и *FrameAccessEngineCommunicationInit* којом се омогућује комуникација путем мреже, се прозивају иницијализацијом ове класе. Како је потребно да достављање фрејмова буде што сигурније, *FrameAccessEngine* се у својој имплементацији ослања на *TCP/IP* протокол стек, којим се омогућује непрекидно слање фрејмова. *FrameAccessEngineCommunicationInit* користећи функционалности овог протокол стека иницијализује сервер, којем се приступа, како би се добавили фрејмови и користили на удаљеној платформи.

Уколико је иницијализација ове класе успешна, методе *FrameAccessEngine* описане у табели 4.6 и методе класе *FrameAccessEngineSHMUtil* описане у табели 4.7 омогућују даље дистрибуирање фрејмова до осталих делова система.

|  |  |
| --- | --- |
| Метода | Опис |
| FrameAccessEngineListener | Метода класе задужена за ослушкивање нових фрејмова које платформа пружа. |
| FrameAccessEngineNewConnectionListener | Метода класе задужена за руковање новим клијентима, који путем мрежног интерфејса добављају фрејмове од овог модула. |
| FrameAccessEngineSendFrame | Метода класе задужена за слање фрејмова клијентима који путем мрежног интерфејса добављају фрејмове. |
| FrameAccessEngineRegisterServiceCallback | Метода класе задужена за обавештавање вишег слоја софтверске магистрале о доступности фрејма. |

Табела 4.6 Опис метода за руковање дистрибуцијом фрејмова класе *FrameAccessEngine*

|  |  |
| --- | --- |
| Метода | Опис |
| FrameAccessEngineStoreFrame | Метода класе задужена за чување фрејмова у дељеној меморији |
| FrameAccessEngineGetSemaphorePath | Метода класе задужена за руковање сигнализацијом дозволе уписа у дељену меморију. |
| FrameAccessEngineGetSharedMemoryKey | Метода класе задужена за достављање кључа за приступ дељеној меморији. |
| FrameAccessEngineGetCameraId | Метода задужена за добављање идентификатора камере чији је фрејм сачуван у меморији. |
| FrameAccessEngineGetBuffer | Метода класе задужена за добављање фрејма и достављање *FrameAccessEngine* компоненти како би она путем мрежног интерфејса могла да га даље дистрибуира. |

Табела 4.7 Опис метода за руковање дељеном меморијом класе *FrameAccessEngineSHMUtil*

Ослањајући се на методу *FrameAccessEngineListener* класе *FrameAccessEngine*, слој за дистрибуцију је у могућности да константно добавља нове фрејмове. Након тога, користећи методу *FrameAccessEngineStoreFrame* класе *FrameAccessEngineSHMUtil*, омогућује се чување сваког новог фрејма у дељеној меморији. Међутим, како би се онемогућило преписивање фрејмова и нарушавање интегритета података, потребно је увести механизам заштите, омогућен кроз апликативну програмску спрегу на *Linux* оперативном систему, у виду сигнализације (енг. *semaphore*) [29]. Позивом методе *FrameAccessEngineGetSemaphorePath,* добија се екслузивно право писања, тј. читања одабране зоне дељене меморије.

Како је могуће вршити потражњу фрејмова са удаљене платформе, руковање мрежним интерфејсом и праћење долазећих захтева за фрејмове врши се методама *FrameAccessEngineNewConnectionListener*, односно *FrameAccessEngineSendFrame* класе *FrameAccessEngine.* Претходно поменуте методе, добављајући фрејмове ослањајући се на методу *FrameAccessEngineGetBuffer* класе *FrameAccessEngineSHMUtil*, у зависности од специфициране камере од самог потражиоца, омогућују приступ потребним фрејмовима.

Како би фрејм био успешно достављен потражиоцу, потребно је да се он ослони на компоненту *FrameAccessClient*, коју описује следеће поглавље.

### Реализација апстракције добављања фрејмова – *FrameAccessClient*

Како би фрејм био успешно достављен апликацији која врши његову обраду, потребно је да се та апликација ослони на функционалности које јој пружа ова компонента.

Као и код претходно описане компоненте, тек након инстанцирања ове класе, могуће је ослонити се на њене функционалности зарад добијања фрејмова. Ова класа пружа могућност добављања фрејма како удаљеној, тако и апликацијама које се одвијају на истој платформи на којој се извршава и софтверска магистрала.

Методе за иницијализацију и постављање ове компоненте у дефинисано стање дате су у табели 4.8.

|  |  |
| --- | --- |
| Метода | Опис |
| FrameAccessClient | Конструктор класе, задужен за постављање инстанце класе у подразумевано стање. |
| ~FrameAccessClient | Деструктор класе, задужен за ослобађање заузетих ресурса класе. |

Табела 4.8 Метода класе *FrameAccessClient* за њену иницијализацију

Приликом иницијализације ове класе, конструктор исту поставља у подразумевано стање. Обзиром да ова класа описује модул помоћу којег се приступа потребним фрејмовима са дефинисане камере, она преостале информације добија од апликације која врши обраду фрејмова и комуницира са највишим слојем софтверске магистрале.

Табела 4.9 описује методе које је потребно прозвати како би се омогућио приступ потребних фрејмовима.

|  |  |
| --- | --- |
| Метода | Опис |
| AttachShm | Метода класе задужена за заузимање дељене меморије у коју се смештају преузети фрејмови. |
| ShmTransfer | Метода класе задужена за преузимање фрејмова, уколико се апликација за обраду фрејмова налази на истој платформи, као и софтверска магистрала. |
| SetFrameSize | Метода класе задужена за обавештавање ове класе о величини фрејма. |
| SocketTransfer | Метода класе задужена за преузимање фрејмова, уколико се апликација за обраду фрејмова налази на удаљеној платформи, у односу на платформу на којој се извршава софтверска магистрала. |
| IsValidIpAddress | Метода класе задужена за одређивање типа приступа фрејму. На основу података које ова компонента добије од вишег слоја, разрешава метод приступа, тј. да ли је потребно приступити удаљеној платформи путем мрежног интерфејса или приступити зони дељене меморије. |

Табела 4.9 Методе за руковање инстанцом класе *FrameAccessClient*

Како би се омогућио приступ фрејмовима, потребно је прво доставити податке који се тичу начина приступа, као и меморијских захтева за преузимање једног фрејма. Тако, позивом метода *SetFrameSize* и *IsValidIpAddress*, прво се наводе меморијски захтеви, а након тога и сам начин преузимања. Уколико се преузима фрејм са удаљене платформе, потребно је ослонити се на мрежни интерфејс, и позвати методу *SocketTransfer*. Користећи управо ову методу, ова страна адаптивне платформе ослања се на функционалности *TCP/IP* стека. Тиме се формира клијентска компонента, која коришћењем функционалности овог протокол стека, добавља фрејмове који су потребни апликацији. Поред оваквог начина добављања фрејмова, њих је могуће добити и читањем дељене меморије, ослањајући се на методу *ShmTransfer*, како би се приступило овом ресурсу у оквиру исте платформе. Независно од начина добављања фрејма, потребне информације доставља виши слој, тј. апликација за обраду фрејмова која комуницира са највишим слојем софтверске магистрале.

## Реализација спреге за информисање о квалитативним карактеристикама сигнала у оквиру адаптивне платформе

Како би се изршила интеграција софтверске магистрале у адаптивну платформу, потребно је имплементирати последњи слој исте. У њеној реализацији, потребно је ослонити се на сервисно оријентисану архитектуру адаптивне платформе и софтверске компоненте које она пружа.

Да би се омогућила видљивост ове компоненте, како на локалној адаптивној платформи, тако и на удаљеној адаптивној платформи, потребно је ослонити се на функционалности *Communication Management* (*COM*) софтверске компоненте. Поред тога, како потребно је ослонити се и на *Execution Management* (*EM*) који је задужен за покретање извршавања ове софтверске магистрале. Као што је већ објашњено, *EM* добија искључиво добија манифест са стањима у којима су потребне апликације адаптивне платформе покренуте. Како је ова апликација неопходна у сваком од стања адаптивне платформе, приликом њеног покретања она је такође покренута и активна је до самог гашења платформе.

Како би се остварила комуникација са другим апликацијама адаптивне платформе, употребом *COM* софтверске компоненте, потребно је изршити конфигурисање овог слоја софтверске магистрале. Његово конфигурисање врши се прво путем описивања кроз одговарајућу датотеку (*arxml*) у којој се наводе порт и интерфејс који су видљиви апликацијама адаптивне платформе. Поред тога, ова датотека такође описује и типове података које овај сервис прослеђује претплаћеним апликацијама адаптивне платформе. Ова датотке се доставља уз изворни код апликације, обзиром да се њоме генеришу сви потребни типови података којима апликација располаже.

Такође, исти овај сервис је потребно описати у манифест датотеци. Тиме се описују портови и интерфејси система на које се ослања овај сервис, како би комуницирао са осталим сервисима и апликацијама адаптивне платформе.

### Опис конфигурације спреге за информисање у адаптивној платформи

Како би се омогућила комуникација овог сервиса са апликацијама које се претплаћују на информације које он пружа, управо у претходно поменутој датотеци, *arxml*, морају се дефинисати све потребне информације којима се описује овај сервис. Како овај сервис рукује догађајима (енг. *Events*), потребно је описати два типа догађаја:

1. *Remote events* – удаљени догађаји – представљају догађаје који се достављају информације компонентама које су на удаљеној платформи. Поред информација о фрејму, ови догађаји пружају податке попут *IP* адресе и порта, како би се конфигурисала клијентска *TCP/IP* компонента, којом се приступа удаљеној платформи.
2. *Local events* – локални догађаји – представљају догађаје који се достављају информације компонентама које су на локалној платформи. Поред информација о фрејму, ови догађаји пружају податке попут кључа којим се приступа ресурсу дељене меморије.

Дефинисањем ових догађаја, омогућује се генерисање посебних структура у програмском језику *C++* којима се достављају следеће информације:

1. Ширина фрејма;
2. Висина фрејма
3. Величина целог фрејма.

Ове три информације представљају клучне информације које описују фрејмове и као такве се не разликују између ова два различита догађаја који су описани. Међутим, информације које се разликују, тичу се начина приступа фрејму. Тако, структура која описује удаљени догађај, поседује информације о *IP* адреси и порту удаљене платформе, док структуре које описују локални догађај садрже податке везане за приступ локалној дељеној меморији у виду кључа за приступ, као и податка о сигнализацији дозволе приступа. Типови података кориштени за ова поља су стандардни типови *C++* програмског језика, али, уколико је потребно променити их и користити кориснички дефинисане типове, то је такође подржано.

Поред дефинисања догађаја, у манифест датотеци која описује систем, потребно је дефинисати портове на које се ослања ова апликација за комуникацију са удаљеним адаптивним платформама и локалном адаптивном платформом. Дефинисање ових протова је врло битно, обзиром да се апликација ослања *COM* компоненту адаптивне платформе.

Након дефинисања ових потребних информација, овај сервис постаје доступан на адаптивној платформи и могуће је претплатити се на информације које он објављује.

Следеће поглавље описује руковање овако конфигурисаним сервисом адаптивне платформе.

### Апликативна програмска спрега за руковање слојем за дистрибуцију квалитативних информација видео сигнала

Како би се омогућила употреба овог адаптивног сервиса, потребно је поред његове претходно описане интеграције прозвати све његове функције које омогућују његову инцијализацију. *C++* класа *CameraService* реализује функционалност овог сервиса.

Табела 4.10 даје увид у све методе класе које је потребно прозвати, како би се иницијализовао последњи слој софтверске магистрале и уједно апликација адаптивне платформе. Део кода је генерисан и он представља начин комуникације на који се врши комуникација путем *COM* модула и као такав, потребно је прозвати генерисане методе при иницијализацији и заустављању софтверске магистрале.

|  |  |
| --- | --- |
| Метода | Опис |
| CameraService | Констурктор класе задужен за постављање поља класе на њихове подразумеване вредности. |
| ~CameraService | Деструктор класе задужен за ослобађање свих заузетих ресурса, као StopOfferService методе за прекидање пружања функционалности овог сервиса. |
| Init | Метода класе која врши иницијализацију потребног контекста сервиса адаптивне платформе, али и нижих слојева, све до слоја за руковање камерама. |
| DeInit | Метода класе задужена за ослобађање контекста заузетог прилико инстанцирања класе. |
| OfferService | Метода класе којом се омогућује видљивост овог сервиса у оквиру адаптивне платформе на једној платформи или видљивост удаљеним адаптивним платформама. |
| StopOfferService | Метода класе којом сео онемогућује видљивост сервиса на адаптивној платформи. |

Табела 4.10 Преглед метода адаптивне апликације за руковање иницијализацијом софтверске магистрале

Приликом инстанцирања објекта ове класе, врши се постављање свих његових поља, попут поља за руковање догађајима који се шаљу кроз адаптивну платформу на подразумеване вредности.

Након тога, позивом методе *Init* ове класе, покрећу се нижи слојеви софтверске магистрале, задужени за руковање камерама и руковање дистрибуцијом видео сигнала. Поред тога, у овој методи се прозива и метода *OfferService* којом се омогућује видљивост овог сервиса у оквиру адаптивне платформе.

Поред тога, како би се прикупљале потребне информације са саме платформе на којој се извршава софтверска магистрала, ова класа се ослања методе описане у табели 4.11.

|  |  |
| --- | --- |
| Метода | Опис |
| CameraServiceNotify | Метода класе задужена за добављање информација од нижих слојева софтверске магистрале. Када фрејм буде доступан, највиши слој софтверске магистрале управо овом методом буде обавештен. |
| SendLocalData | Метода којом се достављају информације у оквиру исте адаптивне платформе, добављене од нижих слојева. |
| SendRemoteData | Метода којом се достављају информације удаљеним адаптивним платформама, добављене од нижих слојева. |

Табела 4.11 Методе за руковање информацијама софтверске магистрале

Ослањајући се на методу *CameraServiceNotify*, сервисни слој адаптивне платформе у могућности је да добави све информације којима се описује фрејм. Ова метода се када фрејмови постану доступни. Тада, овај сервис адаптивне платформе објављује локалним и удаљеним апликацијама адаптивне платформе све познате информације, које се састоје од начина приступа фрејму, као и од његових карактеристика, попут величине самог фрејма, ширине, висине и простора боја, који је у овој конкретној имплементацији YUV420. Позивајући методе *SendLocalData*, односно *SendRemoteData* информације о самим фрејмовима буду достављене до свих апликација и сервиса адаптивне платформе који су се претплатили на исте.

# Резултати

Ово поглавље описује коришћено тестно окружење, као и тестне случајеве који показују перформансе софтверске магистрале. За почетак, дат је преглед потребног времена пропагације фрејма кроз систем. Како је магистрала заснована на сервисно оријентисаној архитектури, приказано је време за које је један фрејм доступан у неком од слојева.

Након тога, како софтверска магистрала пружа могућност дистрибуције у два случаја, то јест локалном и удаљеном, дат је приказ брзине добављања фрејмова у два случаја, када се магистрала користи за достављање фрејмова до апликација која се извршава на истој платформи на којој се извршава и софтверска магистрала, као и достављање фрејмова до апликације која се извршава на удаљеној платформи. Поред тога, ово поглавље бави се перформансама софтверске магистрале у зависности од броја активних камера. Тестни случајеви такође покривају и време које је потребно једној *ADAS* апликацији да преузме фрејм и потом обрађује. На крају, ово поглавље даје један пример употребе камера сервиса за потребе информативне *ADAS* апликације.

Тестно окружење састојало се из две *ALPHA* развојне плоче и једног рутера пропусне моћи 1Gbit/s. На једној *ALPHA* развојној платформи, могуће је приступити шест камера и овај тест обухвата управо резултате свих 6 камера.

## Мерење брзине чувања једног фрејма у зони дељене меморије *Linux* оперативног система

Посматрајући приступ брзину приступа добављеним фрејмовима са камера, потребно је прво направити осврт на брзину добављања фрејмова једне камере из дељене меморије *Linux* оперативног система. Тако, када се приступа зони дељене меморије у којој се налазе фрејмови са једне камере, на узорку од 200 фрејмова, установљено је да је у просеку потребно 622.6 us за читање једног фрејма. Сваки од копираних фрејмова је резолуције 1280х720, односно величине 1,7 **MB**.

На слици 5.1 налази се график, који приказује време читања за сваки од 200 фрејмова.

Слика 5.1 Време уписа једног фрејма у зону дељене меморије у us

Како су по карактеристикама, све камере ове платформе у потпуности исте, тако су и сами фрејмови добављани са сваке понаособ исти. Тиме, време потребно за упис једног фрејма у зону дељене меморије је у потпуности исто за свих 6 камера, које су доступне на овој платформи.

## Мерење времена добављања фрејма камере

Као што је већ описано, софтверска магистрала заснована је на сервисно оријентисаној архитектури. Тако, потребно је уочити времена која су потребна податку да буде доступан у сваком од слојева софтверске магистрале.

### Мерење времена потребно за чување једног фрејма у дељеној меморији *Linux* оперативног система

Како се сама магистрала ослања на дељену меморију *Linux*  оперативног система, за складиштење фрејмова, потребно је прво установити време чувања једног фрејма. Овај фрејм потиче од слоја за апстракцију наменске платформе и чува се у зони дељене меморије којом рукује слој за њихову дистрибуцију.

Како би се установило средње време потребно за добављање једног фрејма до слоја за дистрибуцију фрејмова, коришћен је узорак од 200 фрејмова.

Слика 5.2 приказује време потребно за сваки од 200 фрејмова.

Слика 5.2 Време копирања фрејма из дељене меморије наменске платформе у дељену меморије слоја за дистрибуцију фрејмова

Може се приметити да у већини случајева, време копирања једног фрејма је приближно 5ms, тачније, просечно време копирања једног фрејма износи 4.8ms.

Након мерења овог времена, потребно је такође измерити и време за које један фрејм, посредством сервиса за информисање о доступности и квалитету фрејма, буде доступан другој адаптивној апликацији.

### Мерење времена потребног за достављање једног фрејма посредстом сервиса за информисање о доступности и квалитету сигнала

Сервис за информисање о доступности и квалитету сигнала, то јест фрејма, омогућује два начина приступа фрејмовима наменске платформе. Тако, фрејмовима могу приступати све локалне апликације адаптивне платформе које се претплате на овај сервис, као и удаљене апликације адаптивне платформе које се претплате на овај сервис.

Као и у претходним мерењима, и у овом случају мерење је вршено над 200 фрејмова.

Први, од два посматрана случаја мерења времена, посматра случај у којем локална апликација адаптивне платформе преузима доступни фрејм. У овом посматраном случају, просечно време преузимања једног фрејма износи 12.3ms. Добијено време представља временски период од прве појаве фрејма у зони дељене меморије, до момента њене доступности претплаћеној апликацији адаптивне платформе.

Слика 5.2 представља брзину којом локална апликација адаптивне платформе преузима фрејм, након обавештења о доступности.

Слика 5.3 Време потребно локалној апликацији адаптивне платформе да прибави фрејм, од момента његове доступности

Ефективно, локалној апликацији адаптивне платформе приликом преузимања једног фрејма је потребно знатно мање времена. Наиме, такође на узорку од 200 фрејмова, може се видети да сам приступ дељеној меморији зарад читања у просеку износи 3.2ms.

Слика 5.4 приказује времена читања за 200 фрејмова.

Слика 5.4 Време потребно локалној апликацији адаптивне платформе да преузме фрејм из зоне дељене меморије

Други случај представља случај у коме апликација удаљене адаптивне платформе добавља фрејм једне од камера.

# Закључак

# Литература

|  |  |
| --- | --- |
| [1] | R. Debouk, "Overview of the 2nd Edition of ISO 26262: Functional Safety – Road Vehicles," in *36th International System Safety Conference*, Phoenix, AZ , 2018. |
| [2] | B. Zhang, V. Appia, I. Pekkucuksen, A. U. Batur, P. Shastry, S. Liu, S. Sivasankaran, K. Chitnis and Y. Liu, "A surround view camera solution for embedded systems," *IEEE Conference on Computer Vision and Pattern Recognition Workshops,* 2014. |
| [3] | K. Lee and H. Peng, "Evaluation of automotive forward collision warning andcollision avoidance algorithms," *Vehicle System Dynamics,* vol. 43, no. 10, pp. 735-751, 2005. |
| [4] | S. Fürst and J. Mössinger, "AUTOSAR – A Worldwide Standard is on the Road.," [Online]. Available: https://www.win.tue.nl/~mvdbrand/courses/sse/0809/papers/AUTOSAR.pdf. [Accessed 12 Септембар 2019]. |
| [5] | "Развојна платформа Алфа," [Online]. Available: http://www.rt-rk.com/services/automotive. [Accessed 12 Септембар 2019]. |
| [6] | H. Fennel, S. Bunzel, H. Heinecke, J. Bielefeld, S. Fürst, K.-P. Schnelle, W. Grote, N. Maldener, T. Weber, F. Wohlgemuth, J. Ruh, L. Lundh, T. Sandén, P. Heitkämper, R. Rimkus, J. Leflour, A. Gilberg, U. Virnich, S. Voget, K. Nishikawa, K. Kajio, K. Lange, T. Scharnhorst and B. Kunkel, "Achievements and exploitation of the AUTOSAR development partnership," 2006. [Online]. Available: https://tinyurl.com/atsr-prtnrshp. [Accessed 16 Септембар 2019]. |
| [7] | H. Hellgren, "AUTOSAR in a Nutshell," 14 Април 2018. [Online]. Available: https://hackernoon.com/adaptive-autosar-in-a-nutshell-1cc609c1c5f5. [Accessed 16 Септембар 2019]. |
| [8] | M. Tische, "The Computing Center in the Vehicle - AUTOSAR Adaptive," Септембар 2018. [Online]. Available: https://assets.vector.com/cms/content/know-how/\_technical-articles/AUTOSAR/AUTOSAR\_Adaptive\_ElektronikAutomotive\_201809\_PressArticle\_EN.pdf. [Accessed 21 Септембар 2019]. |
| [9] | Austin Common Standards Revision Group, "POSIX® 1003.1 Frequently Asked Questions," 8 Јун 2017. [Online]. Available: http://www.opengroup.org/austin/papers/posix\_faq.html. [Accessed 21 Септембар 2019]. |
| [10] | AUTOSAR, "Explanation of Adaptive Platform Design," 27 Октобар 2017. [Online]. Available: https://www.autosar.org/fileadmin/user\_upload/standards/adaptive/17-10/AUTOSAR\_EXP\_PlatformDesign.pdf. [Accessed 21 Септембар 2019]. |
| [11] | IBM Knowledge Center, "Service Oriented Architecture," IBM, [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSMQ79\_9.5.1/com.ibm.egl.pg.doc/topics/pegl\_serv\_overview.html. [Accessed 21 Септембар 2019]. |
| [12] | AUTOSAR, "AUTOSAR," Март 2019. [Online]. Available: https://www.autosar.org/standards/adaptive-platform/adaptive-platform-1903/. [Accessed 22 Септембар 2019]. |
| [13] | D. L. Völker, "Scalable service-Oriented MiddlewarE over IP (SOME/IP)," 2019. [Online]. Available: http://some-ip.com/. [Accessed 16 октобар 2019]. |
| [14] | Compaq,Hewlett-Packard,Intel,Lucent,Microsoft,NEC,Philips, "Universal Serial BusSpecification," 27 Април 2000. [Online]. Available: http://sdphca.ucsd.edu/lab\_equip\_manuals/usb\_20.pdf. [Accessed 29 Септембар 2019]. |
| [15] | Texas Instruments, "Introductionto the Controller Area Network(CAN," Мај 2016. [Online]. Available: http://www.ti.com/lit/an/sloa101b/sloa101b.pdf. [Accessed 29 Септембар 2019]. |
| [16] | LIN Consortium, "LIN - Local Interconnect Network," 13 Јун 2015. [Online]. Available: http://www.interfacebus.com/Design\_Connector\_LIN\_Bus.html. [Accessed 29 Септембар 2019]. |
| [17] | L. Lamport, "Interporcess communication," 11 Јун 1985. [Online]. Available: https://apps.dtic.mil/dtic/tr/fulltext/u2/a156337.pdf. [Accessed 28 Септембар 2019]. |
| [18] | "D-bus," freedesktop.org, 7 Јул 2018. [Online]. Available: https://www.freedesktop.org/wiki/Software/dbus/. [Accessed 28 Септембар 2019]. |
| [19] | D. A. Rusling, The Linux Kernel, 2001. |
| [20] | Texas Instruments Incorporated, "TDA2x ADAS System-on-Chip," 2013. [Online]. Available: http://www.ti.com/lit/ml/sprt681/sprt681.pdf. [Accessed 29 Септембар 2019]. |
| [21] | Texas Instruments, "TDA2x," Октобар 2013. [Online]. Available: http://processors.wiki.ti.com/index.php/TDA2x. [Accessed 6 Октобар 2019]. |
| [22] | K. Chitnis, R. Staszewski and G. Agarwal, "TI Vision SDK, Optimized Vision Libraries for ADAS Systems," 2014. [Online]. Available: http://www.ti.com/lit/wp/spry260/spry260.pdf. [Accessed 6 Октобар 2019]. |
| [23] | AUTOSAR , "Guidelines for the use of theC++14 language in critical andsafety-related systems," Март 2017. [Online]. Available: https://www.autosar.org/fileadmin/user\_upload/standards/adaptive/17-03/AUTOSAR\_RS\_CPP14Guidelines.pdf. [Accessed 12 Октобар 2019]. |
| [24] | Perforce, "Perforce," [Online]. Available: https://www.perforce.com/resources/qac/misra-c-cpp. [Accessed 12 Октобар 2019]. |
| [25] | Сmake, "CMake," [Online]. Available: https://cmake.org/. [Accessed 12 Октобар 2019]. |
| [26] | GNU, "GNU make," [Online]. Available: https://www.gnu.org/software/make/manual/make.html. [Accessed 12 Октобар 2019]. |
| [27] | "ООDesign," [Online]. Available: https://www.oodesign.com/singleton-pattern.html. [Accessed 12 октобар 2019]. |
| [28] | M. Kerrisk, "Linux Programmer's Manual," 11 Октобар 2019. [Online]. Available: http://man7.org/linux/man-pages/man7/shm\_overview.7.html. [Accessed 13 Октобар 2019]. |
| [29] | M. Kerrisk, "Linux Programmer's Manual," 11 Октобар 2019. [Online]. Available: http://man7.org/linux/man-pages/man2/socket.2.html. [Accessed 13 Октобар 2019]. |
| [30] | M. Kerrisk, "Linux Programmer's Manual," 5 Март 2017. [Online]. Available: http://man7.org/linux/man-pages/man7/sem\_overview.7.html. [Accessed 14 Октобар 2019]. |