НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ
(повна назва інституту/факультету)
КОНСТРУЮВАННЯ ЕЛЕКТРОННО-ОБЧИСЛЮВАЛЬНОЇ АПАРАТУРИ
(повна назва кафедри)

«На правах рукопису»
УДК 004.773

«До захисту допущено»
Завідувач кафедри
_____ О.М.Лисенко
(підпис) (ініціали, прізвище)
«17» грудня 2021р.

Магістерська дисертація

зі спеціальності (спеціалізації) 172 – Телекомунікації та радіотехніка
(код і назва спеціальності)

на тему: **Засоби та методики оцінки ефективності передачі відеопотоку на основі технології GigE Vision з використанням процесору загального призначення**

Виконав: студент II курсу, групи ДК-01МП
(шифр групи)
Кужильний Олег Вадимович _____
(прізвище, ім'я, по батькові) (підпис)

Науковий керівник к.т.н., доц. А.Ю. Варфоломєєв _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант Асистент кафедри КЕОА Т.А.Ходнєв _____
(назва розділу) (науковий ступінь, вчене звання, , прізвище, ініціали) (підпис)

Рецензент Директор ЦТ «КПІ-ТЕЛЕКОМ» А. Ільяшенко _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут/факультет Електроніки
(повна назва)

Кафедра Конструювання електронно-обчислювальної апаратури
(повна назва)

Рівень вищої освіти – другий (магістерський) за освітньо-професійною програмою

Спеціальність (спеціалізація) 172 – Телекомунікації та радіотехніка
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.М.Лисенко
(підпис) (ініціали, прізвище)
« 02 » вересня 2021 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Кужильному Олегу Вадимовичу
(прізвище, ім'я, по батькові)

1. Тема дисертації: Засоби та методики оцінки ефективності передачі відеопотоку на основі технології GigE Vision з використанням процесору загального призначення
   науковий керівник дисертації к.т.н, доц, А.Ю Варфоломєєв
   (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
   затверджені наказом по університету від 3 листопада 2021 року № 3666-С

2. Строк подання студентом дисертації 13 грудня 2021 року

3. Об'єкт дослідження: Методи підвищення швидкодії передачі відеопотоку.

4. Предмет дослідження (Вихідні дані – для магістерської дисертації за освітньо-професійною програмою): Засоби та методики для оцінки ефективності передачі відеопотоку каналом Ethernet.

5. Перелік завдань, які потрібно розробити 1. Провести огляд сучасних рішень у сфері передачі відеопотоку. 2. Реалізувати прототип GigE Vision камери з

використанням процесору загального призначення. 3. Провести дослідження ефективності передачі відеопотоку каналом Ethernet. 4. Розробити стартап-проєкт.

6. Перелік графічного (ілюстративного) матеріалу
Презентація у форматі Power Point

7. Орієнтовний перелік публікацій складає 2 публікації

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Другий	Т.А.Ходнєв	05.11.2021	17.11.2021

9. Дата видачі завдання 02.09.2021 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1.	Аналіз літературних джерел за темою дисертації	07.09.2021 – 22.09.2021	**виконано**
2.	Дослідження ринку на предмет попиту сучасних методів передачі відеопотоку, проведення вибору апаратної платформи	24.10.2021 – 29.10.2021	**виконано**
3.	Формування списку задач, які будуть вирішені в ході дослідження	02.11.2021 – 03.11.2021	**виконано**
4.	Розробка прототипу, проведення дослідження з оцінки ефективності передачі відеопотоку	05.11.2021 – 17.11.2021	**виконано**
5.	Розробка стартап-проєкту	17.11.2021 – 21.12.2021	**виконано**
6.	Оформлення дисертації	21.12.2021 – 29.12.2021	**виконано**

Студент _____ (підпис) Олег КУЖИЛЬНИЙ

Науковий Керівник _____ (підпис) Антон ВАРФОЛОМЄЄВ

РЕФЕРАТ

Магістерська дисертація складається з 83 сторінок, в якій міститься 15 рисунків, 23 таблиць, використано 23 джерела.

Актуальність. З попитом рішень, побудованих на базі машинного зору, зростає потреба у якісному транспорту відепотоку. Це значить, що постає питання пропускної здатності, швидкості каналу, затримок передавання, надійності передавання, довжини окремного кабельного сегменту тощо. Відповідно, підприємства, для яких подібні параметри є основними вимогами, ставлять питання виробу методу та технології передачі відепотоку. На нашу думку, технологія GigE Vision є доволі цікавою для будь-якого рішення де необхідна зворотня сумісність, висока пропускна здатність тощо. Але проблемою є те, що рішення, які використовують дану технологію мають доволі високі цінники через високу ціну на внутрішні компоненти. Дана робота є актуальною через відсутність актуальних досліджень, щодо використання GigE Vision технології на базі процесорів загального призначення.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційні дослідження проводилися у відповідності з науковими напрямками роботи кафедри конструювання електронно-обчислювальної апаратури КПІ ім. Ігоря Сікорського та пріоритетного напряму розвитку науки і техніки України "Інформаційні та комунікаційні технології ".

Метою дисертаційної роботи є визначення ефективності передачі відепотоку каналом Ethernet на основі технології GigE Vision з використанням саме процесорів загального призначення.

Об'єктом дослідження є методи підвищення швидкодії передачі відеопотоку.

Предметом дослідження є засоби та методики для оцінки ефективності передачі відеопотоку каналом Ethernet.

Методи дослідження. Порівняння існуючих рішень, аналіз та синтез програмно-апаратної реалізації, метод дедукції, експериментальні дослідження.

Наукова новизна отриманих результатів дослідження полягає в наступному:

- Запропоновано структурно-функціональну організацію джерела відеопотоку, що використовує технологією GigE Vision та реалізується на

ARM процесорі загального призначення, що в порівнянні з існуючими рішеннями на основі FPGA володіють меншою вартістю та здатні забезпечити прийнятну (до 10-15 мс) затримку передавання кадрів.

- Розроблено методику для оцінювання затримки передавання кадрів відеопослідовності, що основана на використанні протоколу точного часу PTP (стандарт IEEE-1588), що для розроблюваного джерела відеопотоку дозволяє оцінювати затримку передавання з точністю до 100 мкс.

Практичне значення отриманих результатів полягає у:

- Розробленні діючого прототипу джерела відеопотоку сумісного з технологією GigE Vision на основі поширеного одноплатного комп'ютера Raspberry Pi 4B, що використовує ARM процесор загального призначення.
- Формулюванні загальних рекомендацій щодо досягнення малих затримок передавання кадрів при реалізації GigE Vision сумісних камер, що побудовані на основі одноплатних комп'ютерів з процесорами загального призначення.

Публікації. За матеріалами дисертації опубліковано 2 друковані праці, одна з яких включена до наукометричної бази Web of Science:

1) Ходнєв Т.А., Голуб М.С., Кужильний О.В., Лисенко О.М., Варфоломєєв А.Ю. Акселерована реєстрація MIPI CSI відеопотоку в задачах передачі відео реального часу // Visnyk NTUU KPI Seriia – Radiotekhnika Radioaparatobuduvannia. – 2020. – №82. – С. 35–43. DOI: 10.20535/RADAP. 2020.82.35-43.
2) Кужильний О.В., Варфоломєєв А.Ю. Ходнєв Т.А. Засоби та методики оцінки ефективності передачі відеопотоку на основі технології GigE Vision з використанням процесору загального призначення // Мікросистеми електроніка та акустика (у друці).

Ключові слова: відеопотік; затримка передачі; синхронізація часу; Ethernet; GigE Vision; Aravis; Video4Linux

ABSTRACT

Master's thesis consists of 83 pages, 15 figures, 23 tables, used 23 sources.

Relevance. With the demand of solutions built based on machine vision, the need for high-quality transport of the video stream is growing. This means that there is a question of bandwidth, channel speed, transmission delays, transmission reliability, the length of a separate cable segment, etc. Accordingly, enterprises for which such parameters are the main requirements, ask questions about the product method and technology of video stream transmission. In our opinion, GigE Vision technology is quite interesting for any solution where backward compatibility is required, high bandwidth, etc. but the problem is that solutions that use this technology have quite high price due to the high price of internal components.

Connection of work with scientific programs, plans, topics. Dissertation research were carried out in accordance with the scientific directions of the department of design of electronic computing equipment of Igor Sikorsky Kyiv National University. Igor Sikorsky and priority direction of development of science and technology of Ukraine "Information and communication technologies".

The purpose of the dissertation is to determine the effectiveness of the transmission of the video stream by the Ethernet channel based on GigE Vision technology using general-purpose processors.

The object of the study are methods to increase the speed of video stream transmission.

The subject of the study is the means and techniques for assessing the effectiveness of video stream transmission over Ethernet.

Methods of research. Comparison of existing solutions, analysis and synthesis of software and hardware implementation, deduction method, experimental research.

The scientific contribution of the study is following:

- Structural and functional organization of the video stream source using GigE Vision technology and implemented on ARM general-purpose processor is proposed, which, compared to existing FPGA-based solutions, have a lower cost and are able to provide an acceptable (up to 10-15 ms) frame transfer delay.

- A methodology for assessing the delay in the transmission of video footage based on the use of the PTP exact time protocol (IEEE-1588 standard) has been developed, which allows the development of a video stream source to estimate the delay in transmission from accuracy up to 100 us.

The practical significance of the obtained results is:

- Development of an existing prototype video stream source compatible with GigE Vision technology based on a widely used single-board Raspberry Pi 4B computer that uses a general-purpose ARM processor.
- Formulate general recommendations for achieving small frame transfer delays when implementing GigE Vision compatible cameras based on single-board computers with general-purpose processors.

Publication. According to the dissertation materials 2 printed papers in the conference materials collection were published.

Keywords: video stream; delayed transmission; time synchronization; Ethernet; GigE Vision; Aravis; Video4Linux

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

DHCP	Протокол динамічної конфігурації вузла
LVDS	Низьковольтна диференціальна передача сигналів
FPGA	Програмована користувачем вентильна матриця
UDP	Протокол датаграм користувача
CSI	Послідовний інтерфейс передачі даних з камери
MTU	Максимальний розмір блоку передачі даних у мережі
API	Прикладний програмний інтерфейс
IP	Інтернет протокол
LAN	Локальна комп’ютерна мережа
PTP	Протокол точного часу
PHY	Фізичний рівень рівня мережевої моделі OSI
GMII	Інтерфейс незалежний від середовища передачі даних
ПДП	Прямий доступ до пам'яті
ПДД	Пам’ять з довільним доступом

ВСТУП

Актуальність З метою створення продуктів, які повинні відрізняються гнучкістю та невисокою вартістю, індустрія комп'ютерного зору роками розробляє високоефективні програмні рішення на основі протоколів передачі зображень на короткі та середні відстані, таких як Camera Link та LVDS. З цим пов'язано ажіотаж у світі машинного зору щодо розробки нових програм з використанням стандартних кабельних з'єднань Gigabit-Etherne [1, 2]. На підставі аналізу результатів минулих досліджень виникла ідея заміни FPGA технології процесором загального призначення при розгортанні даної технології. Сучасні системи на кристалі та вбудовані процесори загального призначення стали значно продуктивнішими, енергоефективнішими та мають широкий набір периферії, в тому числі для безпосереднього підключення відеосенсорів та Ethernet контролерів. Враховуючи це, а також нижчу вартість в порівнянні з FPGA, постає питання щодо їх використання як більш дешевих альтернативних рішень для вирішення задачі реалізації GigE Vision відеокамер.

Зв'язок роботи з науковими програмами, планами, темами. Дисертаційні дослідження проводилися у відповідності з науковими напрямками роботи кафедри конструювання електронно-обчислювальної апаратури КПІ ім. Ігоря Сікорського та пріорітетного напряму розвитку науки і техніки України "Інформаційні та комунікаційні технології ".

Метою дисертаційної роботи є визначення ефективності передачі відепотоку каналом Ethernet на основі технології GigE Vision з використанням саме процесорів загального призначення. При цьому передбачається вирішення задач оцінювання параметрів таких як рівень затримки між передавачем та приймачем, а також рівня навантаження на мережу та процесорні ядра.

Для досягнення поставленої мети в роботі вирішувалися наступні **задачі**:

- Досліджено ринок на предмет попиту сучасних технологій передачі

зображення на короткі та середні відстані, вивчено особливості їх передачі
- Поставлено цілі щодо майбутньої програмної реалізації, вибрано відповідну апаратну платформу
- Розроблено прототип камери, що відповідає вимогам, необхідним для проведення дослідження ефективності передачі
- Проведено дослідження з вимірювання ефективності передачі відеопотоку каналом Ethernet, що складається з вимірювання затримки каналу та завантаженості процесорних ядер.

Об'єктом дослідження є методи підвищення швидкодії передачі відеопотоку.

Предметом дослідження є засоби та методики для оцінки ефективності передачі відеопотоку каналом Ethernet.

Методи дослідження. Порівняння існуючих рішень, аналіз та синтез програмно-апаратної реалізації, метод дедукції, експериментальні дослідження.

Наукова новизна отриманих результатів дослідження полягає в наступному:

- Запропоновано структурно-функціональну організацію джерела відеопотоку, що використовує технологією GigE Vision та реалізується на ARM процесорі загального призначення, що в порівнянні з існуючими рішеннями на основі FPGA володіють меншою вартістю та здатні забезпечити прийнятну (до 10-15 мс) затримку передавання кадрів.
- Розроблено методику для оцінювання затримки передавання кадрів відеопослідовності, що основана на використанні протоколу точного часу PTP (стандарт IEEE-1588), що для розроблюваного джерела відеопотоку дозволяє оцінювати затримку передавання з точністю до 100 мкс.

Публікації. За матеріалами дисертації опубліковано 2 друковані праці, одна з яких включена до наукометричної бази Web of Science:

1) Ходнєв Т.А., Голуб М.С., Кужильний О.В., Лисенко О.М., Варфоломєєв А.Ю. Акселерована реєстрація MIPI CSI відеопотоку в задачах передачі відео реального часу // Visnyk NTUU KPI Seriia – Radiotekhnika Radioaparatobuduvannia. – 2020. – №82. – С. 35–43. DOI: 10.20535/RADAP. 2020.82.35-43.
2) Кужильний О.В., Варфоломєєв А.Ю. Ходнєв Т.А. Засоби та методики оцінки ефективності передачі відеопотоку на основі технології GigE Vision з використанням процесору загального призначення // Мікросистеми електроніка та акустика (у друці).

Копії публікацій наведено у додатках А та Б.

Структура дисертаційної роботи містить розділ вступу, 4 розділи, загальні висновки, перелік використаної літератури та два додатки – копії перших сторінок публікацій за дисертаційної роботою.

РОЗДІЛ 1 АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМОЮ

1.1 Потокова передавача відео та керування каналом Ethernet — технологія GigE Vision

GigE Vision — це комунікаційний інтерфейс для додатків комп'ютерного зору, що базується на передачі даних, використовуючи технологію Ethernet. Інтерфейс дозволяє передавати зображення з пристрою GigE Vision до мережевої карти за допомогою стандартного кабелю CAT-5e/6 або будь-якого іншого фізичного каналу передачі даних, що підтримує Ethernet. Крім того, технологія GigE Vision підтримує усі швидкості передачі доступні відповідному Ethernet стандарту. Так технологія GigE Vision спирається на технологію Ethernet, вона не покриває фізичний рівень, який вже реалізовано відповідними драйверами стеку TCP/IP.

Системи, що базується на технології GigE Vision, можуть бути підключені до широкого спектру різноманітних мережевих топології. Найпростішим з них є точкове з'єднання між персональним комп'ютером (ПК) і пристроєм потокового відео за допомогою одного кабелю. Або до більш складної системи, що складається з апаратних і програмних джерел відеопередачі та мережевого обладнання, що є частиною однієї IP мережі. При цьому, для GigE Vision продуктів дуже важливим є сумісність з іншими IP-пристроями в мережі. Відповідно, при розробці камер необхідно дотримуватися основних принципів, передбачених в публікаціях документу RFC (Request for Comments), що публікуються організацією IETF (Internet Engineering Task Force). Але при цьому кількість таких вимог може бути обмежена для того щоб кінцевий пристрій GigE Vision був економічно вигідним.

Незважаючи на те, що назва цієї специфікації конкретно відноситься до Gigabit Ethernet, GigE Vision може бути реалізований для будь-якого класу

швидкостей Ethernet. Будь-який фізичний носій, зазначений IEEE 802.3, може бути використаний для реалізації GigE Vision.

Підкреслимо можливі переваги при використанні технології GigE Vision:

- Для розгортання технології можуть бути використані будь-яке мережеве обладнання
- Відсутня потреба у спеціалізованому обладнанні
- Висока пропускна здатність при передачі даних гарантується за рахунок швидкості передачі близько 100 МБ/с. При цьому, можуть використовуватись більш складні рішення з встановленням декількох камер, або швидких промислових камери з більш високою роздільною здатністю.
- Будь-які інші цифрові інтерфейси, такі як USB, FireWire і CameraLink, явно обмежені у відношенні довжини кабельного сегменту, що складає від 5 до 10 м, на відміну від Ethernet, максимальна довжина якого складає 100 м.
- Стандарт програмного забезпечення GenICam спрощує простоту роботи, програмування та підтримки шляхом стандартизації
- Великий вибір апаратного забезпечення, який може бути використаний програмним забезпеченням, що полегшує вирішення прикладних завдань.
- Зручна підтримка рішень на базі даної технології, так як обладнання є у вільному доступі і може бути швидко замінене у разі потреби
- Передбачається, що GigE Vision є довготривалою технологією, оскільки дана технологія може бути розширена з використанням стандарту 10 Gigabit Ethernet.

1.2 Опис інтерфейсу GeniCam

GenICam - це універсальний інтерфейс управління камерою [3]. Тобто, такий механізм що дозволяє налаштувати будь-яку камеру, що підтримує даний інтерфейс, незалежно від моделі камери чи її функціоналу. Даний інтерфейс підтримують камери, що побудовані над такими інтерфейсами як GigE, CameraLink, FireWire і т.д.

Основна ідея полягає у тому, що сама камера інформує систему про те, які функції доступні, та забезпечує послідовний доступ до конфігурації без необхідності конкретного програмного забезпечення, що можна отримати тільки від виробника. Таким чином, GenICam є ключовим компонентом у роботі з технологією GigE Vision, але не обмежується лише цим напрямком. Тобто, два стандарти, GigE Vision та GenICam надають максимально можливу незалежність при виборі та розробці рішень для обробки зображень.

1.3 Принцип виявлення GigE Vision камери у мережі

Виявлення GigE Vision камери в мережі складається з послідовності дій, необхідних для конфігурації кінцевого пристрою, щоб встановити з'єднання через його мережевий інтерфейс і отримати робочу IP-адресу за допомогою стандартних IP-протоколів. Що, в свою чергу, може бути використано для реєстрації GigE Vision пристроїв в мережі. Після завершення виявлення пристрою програма готова надсилати керуючі повідомлення на пристрій. Виявлення пристрою складається з 3-х послідовних кроків:

1. Встановлення зв’язку
2. Запит IP адреси пристроєм та її отримання
3. Перерахування пристрою у мережі

Зазвичай, кінцевий пристрій може мати один або кілька мережних

інтерфейсів. За першим кроком, мережева підсистема повинна налаштувати тий інтерфейс, що під'єднано до мережі. Згідно стандарту Ethernet IEEE 802.3, швидкість на обох кінцях мережі повинна синхронізуватися. За другим кроком, що характеризується конфігурацією унікальної адреси в мережі, виконується на прикладному рівні та ініціюється пристроєм. Як правило, мова йде про DHCP протокол. При цьому, повинна існувати можливість встановити IP адресу статично, шляхом її зберігання в енергонезалежному носії, що є частиною пристрою. На третьому кроці, виконується перерахування пристрою, що ініціюється додатком для збору інформації про пристрій в мережі. Дана взаємодія реалізується за допомогою обміну GVCP повідомленнями між пристроєм та програмою. Також, відповідь включає в себе різні фрагменти інформації про пристрій, такі як назва виробника, модель пристрою і т.д. Приклад ідентифікації зображено на рисунку 2.8 – а.

1.4 GVCP – протокол керування GigE Vision

GVCP — це протокол прикладного рівня, що спирається на протокол транспортного рівня UDP (IPv4) [3]. Мета даного протоколу полягає в налаштовуванні конкретного пристрою. Як правило, за допомогою GigE Vision додатку ініціюється канал передачі відеопотоку. Пристрій, у відповідь, може повідомляти додаток про пов'язані з ним конкретні програмні події. Протокол GVCP гарантує наступну модель взаємодії. Взаємодія за протоколом відбувається тільки одним, головним додатком. Іншими словами, тільки головний додаток може керувати пристроєм. Цим самим, керуючі послідовності передаються від головного додатку до пристрою. При цьому, другорядні додатки можуть тільки відстежувати стан пристрою (режим моніторингу), якщо це було дозволено головним додатком. Але, додаток може запросити управління пристроєм, який вже знаходиться під контролем головної програми, за умови, що

це підтримується кінцевим пристроєм. Відповідно до протоколу GVCP, головний додаток є передавачем, а пристрій - приймачем.

За протоколом, окреме повідомлення складається з керуючого запиту та підтвердження попереднього. Тобто, при запиті, додаток повинен чекати на повідомлення про підтвердження перед надсиланням наступного керуючого кадру. Таким чином, виконується базове рукостискання (handshake), що забезпечує мінімальний контроль потоку. Повідомлення про підтвердження дає інформацію про те що команда була фактично отримана пристроєм та у відповідь, отримувач передає корисні дані.

Оскільки UDP є ненадійним транспортним протоколом, GVCP визначає механізми, що гарантують надійність передачі пакетів і забезпечують мінімальний контроль потоку.

Підсумуємо яку роль виконує керуючий керування GigE Vision (GVCP):

- Встановлення дозволу на обмін повідомленнями та виконання рукостискання між пристроєм GigE Vision та додатком
- Ініціювання каналу для передачі відеопотоку пристроєм
- Встановлення дозволу на надсилання асинхронних повідомлень про події з пристрою до певного додатку
- Гарантування унікальної схеми доступу при якій тільки головний додаток може контролювати пристрій
- Мінімізація IP стеку для GigE Vision пристрою

1.5 GVSP – протокол трансляції GigE Vision відеопотоку

GVSP — це протокол прикладного рівня, що спирається на протокол транспортного рівня UDP [3]. Цей протокол дозволяє GVSP приймач отримувати потік даних, що представляє собою зображення та службові дані. А також будь-які інші кадри від GVSP передавача. При цьому зауважимо, що GVSP пакети

завжди транслюються від GVSP передавача до приймача.

На даний момент, специфікації стандарту GigE Vision використовує протокол UDP IPv4 як протокол транспортного рівня. Відомо, що UDP є ненадійним, тому GVSP надає механізми, що гарантують надійність передачі пакетів (використовуючи протокол GVCP), цим самим забезпечується мінімальний контроль потоку.

Керування підключеннями здійснюється за допомогою протоколу GVCP. Для цього, відправляється пакет до приймача з певною періодичністю з метою отримати відповідь від пристрою. Якщо додаток, буде вимкнено, GVCP автоматично скине потокове з'єднання та GVSP передавача буде зупинена.

Для розмежування корисного навантаження (зображень), протокол використовує так звані розмежувальні пакети – початковий та кінцевий. Дані між ними, відповідно, корисні дані. Даний підхід є стандартним та підтримується з моменту створення GigE Vision.

1.6 Огляд технологій передачі відеопотоку на короткі на середні відстані

FireWire. Вважається найбільш розповсюдженим відеоінтерфейсом в індустрії машинного зору. Базовий протокол, що має назву IIDC, оперує поверх даного інтерфейсу. Даний протокол широко застосовується виробниками (вендорами) систем машинного зору. Він представляє собою карту регістрів за допомогою якого відбувається конфігурація камер. При цьому конфігурації можуть бути взаємозамінними, що дозволяє швидко інтегрувати камери у кінцеві системи. Даний інтерфейс є високоефективним при одночасній інтеграції масиву камер, що з'єднуються в однорангову мережу без використання концентраторів.

Camera Link. Набув популярності у період появи специфічних потреб у машинному навчанні та необхідності більшої пропускної здатності.

Особливістю камер технології Camera Link є передача кадрів блоками ПДП,

що значно знижує кількість інструкцій, які потенційно міг виконувати центральний процесор. У результаті дані камери підтримують швидкість передачі до 680МБ/с у противагу інтерфейсу FireWire з показником 80МБ/с. При цьому вартість системи побудованою за Camera Link технологією є значно вищою через необхідність встановлення додаткових блоків відеозахоплення. Тому дані камери застосовуються у системах реального часу де необхідна висока пропускна здатність.

USB Vision. Серед якісних переваг специфікації USB 3.0 є передача даних блоками ПДП з пропускною здатністю до 5 Гб/с, при потужності на один кабельний сегмент до 4.5 Вт. Кількість підключених до системи камер може бути збільшено за рахунок USB концентраторів. Камери підтримують програмний інтерфейс GeniCam. Даний програмний інтерфейс дозволяє абстрагуватись від карти регістрів, що обмежувала вендорів у впровадженні власних функцій для впровадження у логіку роботи камер.

1.7 Протокол точного часу

Протокол точного часу, як визначено в стандарті IEEE-1588, виконує точну синхронізацію комп'ютерів, передаючи повідомлення локальною мережею (LAN). Стверджується, що для апаратної реалізації протоколу, синхронізація можлива з точністю до 1 мікросекунди, тоді як для програмної реалізації PTP, точність зазвичай не перевищує 100 мікросекунд (PTPd, n.d.; Kovácsházy, 2010). Для досягнення кращих показників у локальній мережі потрібна архітектура, яка повністю відповідає вимогам IEEE-1588. Протокол визначає метод синхронізації повідомлень, якими обмінюється джерело та приймач точного часу. За подібною структурою виконується взаємодія між сервером та клієнтом, що використовується в протоколі мережевого часу (NTP).

Повідомлення PTP протоколу складаються з таких як початкове

синхроповідомлення та відповідь з інформацією про рівень затримки зі сторони джерела, а також повідомлення з рівнем затримки від приймача. При цьому, може бути використано декілька різних повідомлень синхронізації від джерел, з різними рівнями затримки, для того щоб приймач міг обрати найкраще.

Для синхронізації часу в локальній мережі може бути використано архітектуру, що складається принаймні з одного джерела точного часу та одного приймача. Зауважимо також, якщо виконувати з'єднання точка-точка, у мережі будуть наявні тільки два пристрої, тому загалом немає різниці хто саме з них буде джерелом точного часу, а хто буде його споживачем. При цьому, при налаштуванні декількох приймачів, можлива синхронізація від одного джерела.

За даним протоколом, джерело точного часу транслює повідомлення синхронізації, які використовуються приймачами для коректування часу. Час на стороні приймача розраховується за допомогою часових міток. Мітки часу в повідомленнях коригуються на величину часу, витраченого на проходження через мережеве обладнання. Ця схема підвищує точність видачі часу споживачам, компенсуючи затримки доставки повідомлень мережею. При цьому, синхроповідомлення від джерела, як правило, передаються кожні дві секунди. В іншу чергу, запит на синхронізації з інформацією про затримку рідше, близько одного запиту в хвилину.

Для синхронізації необхідно чотири повідомлення з мітками часу з яких необхідно вирахувати зміщення у часі [4]. Описана взаємодія зображена на рисунку 1.1.

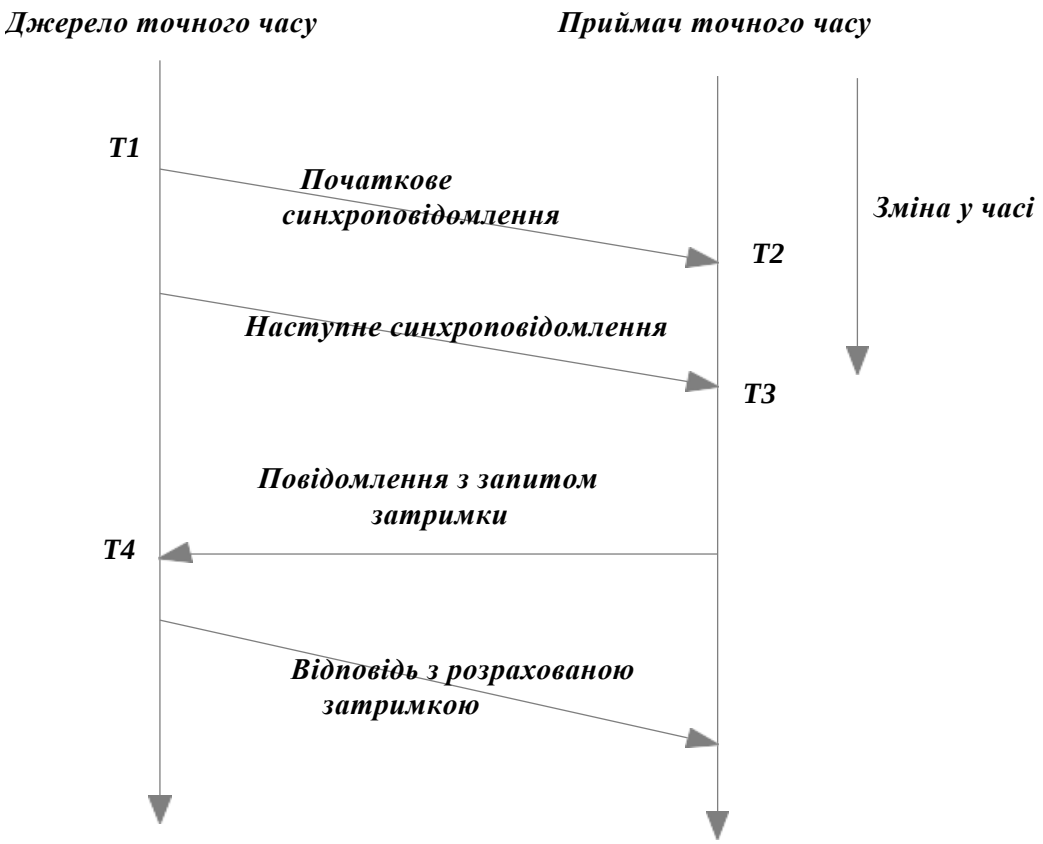


Рисунок 1.1 – Синхронізація часу за протоколом PTP з використанням часових міток T1 - T4.

Для отримання точного часу, приймач виконує наступну послідовність операцій. По-перше, необхідно обчислити доступні шляхи затримки мережі: "джерело-приймач" та "приймач-джерело". Мітка часу T1 відповідає зміщенню у часі "джерело-приймач". Дана мітка є величиною точного часу, що вкладається у наступне синхроповідомлення, відправленого джерелом. Зауважимо, що дана мітка відправляється джерелом тільки після того як було відправлено перше синхроповідомлення. Другою міткою часу є T2. Це рівень точного часу повідомлення синхронізації що визначається на рівні приймача. При отримані приймачем обидва значення, T1 та T2, на стороні приймача розраховується рівень затримки. Рівень затримки джерело-приймач розраховується за наступною

формулою (див. формулу 1.1):

$$T_{\text{дп}} = T_1 - T_2 \quad (1.1)$$

Наступним кроком є розрахунок затримки приймач-джерело.

Звернімо увагу на часову мітку ТЗ. Дана мітка визначена на стороні приймача. У свою чергу, джерелу необхідно визначити значення даної мітки. Для цього приймач відправляє повідомлення з даної мітки до джерела. Отримавши дану інформацію, джерело розраховує затримку приймач-джерело за наступною формулою 1.2:

$$T_{\text{дп}} = T_1 - T_2 \quad (1.2)$$

У результаті, одностороння затримка може бути розрахована якщо обидві затримки джерело-приймач та приймач-джерело відомі для приймача. Дана затримка розраховується як (формула 1.3):

$$T_o = (T_{\text{дп}} + T_{\text{пд}})/2 \quad (1.3)$$

Як результат, дане значення є затримкою мережі, яку джерело повинно врахувати для передачі приймачу. У свою чергу, приймач використовує дану затримку для синхронізації з джерелом. Більш того, для встановлення точного часу виконуються розрахунки за певним алгоритмом, які повинні враховувати вплив зовнішніх факторів, такі як температура зовнішнього середовища.

1.8 Аналіз технічних рішень за результатами патентного пошуку

Патент KR102015956B1. Автоматична система розпізнавання номерних знаків і метод передачі інформації про транспортний засіб мережею Gigabit Ethernet.

Даний винахід розкриває систему розпізнавання номерів транспортних засобів у реальному часі та метод передачі інформації гігабітною мережею Ethernet. Пристрій для фотографування номера транспортного включає в себе блок захоплення зображення, блок виявлення номерних знаків для пошуку номерного знаку на зображенні та блок передачі відеопотоку за технологією GigE Vision.

Патент CN104647388A. Інтелектуальна система та метод управління промисловим роботом на основі машинного зору. Винахід розкриває метод інтелектуального управління промисловим роботом на основі машинного зору. Описується наступні методи керування роботом відбувається наступним чином:

1. Метод збору інформації про навколишній простір на основі відеопотоку отриманого з GigE Vision каналу
2. Метод конфігурації функціональних компонентів, що керують роботом
3. Метод налаштування на базі системи польової шини EtherCAT
4. Метод порівняння отриманої інформації про зображення за попередньо встановленим шаблоном

Патент CN104572574A. Реалізація GigE Vision контролеру у вигляді IP ядра.

Винахід розкриває IP-ядро у вигляді Ethernet контролеру на основі протоколу GigE Vision. IP-ядро складається з модуля управління, модуля інтерфейсу управління PHY (фізичного рівня), модуля управління передачею відео, модуля управління потоком та модуля керування прийомом та реалізованого через FPGA. Передача даних відповідає вимогам специфікаціям інтерфейсу Avalon, що дозволяє виконувати транзакції шиною Avalon-ST, та

інтерфейсу GMII.

IP-ядро контролера Ethernet на основі протоколу GigE - це спеціальне IP-ядро, розроблене відповідно до характеристик протоколу GigE Vision. Ідея полягає у захопленні та автоматичному зберіганні зображень у пам’яті. IP-ядро Ethernet виконує роль блоку зберігання зображень, що, у свою чергу, вирішує проблему енергоефективності. Тобто, знижуючи навантаження на процесор та підвищуючи ефективність передачі відеопотоку. Даний контролер працює за принципом традиційних контролерів Ethernet, але за рахунок паралельної обробки на FPGA, поліпшується швидкість прийому даних, і тим самим швидкість системи. В описі патенту було підкреслено, що при тестуванні виявилось, що збір зображень через IP-ядро контролера Ethernet на основі протоколу GigE зменшує більше половину споживання ресурсів FPGA в порівнянні з IP-ядром Ethernet компанії Altera.

1.9 Обґрунтування вибору методів дослідження

У ході дисераційної роботи розглянуто функціонування системи у вигляді детального опису кожного з її компонентів. При синтезі цих знань, розглядається композиція програмної та апратної взаємодії як єдиної системи. Тому, перший метод, який було використано має назву «Аналіз та синтез».

При аналізі існуючих теоретичних та практичних знань представлено власне рішення науково-прикладної задачі. Даний метод має назву «Дедукція».

Як заключний етап, метод експерименту, включає у себе дослідження швидкодії передавання кадру мережею на базі протопипу Gige vision камери.

Висновок до розділу 1

За результатом першого розділу детально досліджено принцип функціонування технології GigE Vision, а саме, принципи взаємодії основних компонентів – протоколів, якими керуються пристрої за стандартом GigE Vision. Досліджено швидкісні цифрові протоколи передачі відеопотоку на короткі та середні дистанції. Представлено основні порівняльні параметри, щодо якісного використання кожного з протоколів.

З метою проведення експериментів з використанням протоколу точного часу, було розглянуто метод синхронізації часу за рахунок часових міток для визначення затримок джерело-приймач, приймач-джерело.

Розглянуто ряд патентів, що пов'язані передачею відеопотоку каналом Ethernet за стандартом GigE Vision.

РОЗДІЛ 2 РЕАЛІЗАЦІЯ ПРОТОТИПУ GIGE VISION СУМІСНОЇ КАМЕРИ НА ОСНОВІ СИСТЕМИ НА КРИСТАЛІ З ПРОЦЕСОРНИМИ ЯДРАМИ ЗАГАЛЬНОГО ПРИЗНАЧЕННЯ

2.1 Дослідження апаратних платформ з метою побудови GigE Vision сумісного прототипу

Питання ефективності передачі відеопотоку каналом Ethernet з використанням технології GigE Vision було розглянуто у статтях [5-8] .Водночас, досліджувались особливості реалізації GigE Vision сумісної камери на базі технології FPGA. На підставі аналізу результатів минулих досліджень виникла ідея заміни FPGA технології процесором загального призначення при розгортанні даної технології. Водночас, у дослідженні [1] стверджується, що така заміна не є надто доцільною з огляду на низьку енергоефективність процесорів загального призначення, однак зважаючи на час, що пройшов з моменту публікації [1] (приблизно 15 років) ситуація суттєво змінилась. Пропонується дослідити як саме буде відрізнятися ефективність передачі відеопотоку з використанням рішень, реалізованих саме на процесорах загального призначення. При цьому передбачається вирішення задач оцінювання параметрів таких як рівень затримки між передавачем та приймачем, а також рівня навантаження на мережу та процесорні ядра.

Для оцінювання ефективності передачі відеопотоку системою на основі вбудованого процесора загального призначення було розроблено власний прототип GigE Vision сумісної камери. Перш за все, для цього необхідно було обрати апаратну платформу, що задовольняє критеріям собівартості, потужності, наявності необхідних периферійних пристроїв та розробити програмну підтримку первинного захоплення відеопотоку з його подальшого транслювання у Ethernet мережу.

Було розглянуто дві платформи, що незначно відрізняються за ціною, але мають велику різницю в потужності, зручності використання та популярності у розробників. Мається на увазі платформи Asus Tinker Board та Raspberry Pi 4B.

Кожна з платформ є специфічною – має різну підтримку для функціонування ядра Linux, драйверів тощо. Водночас, у всіх присутні необхідні апаратні засоби, а саме апаратний CSI інтерфейс та GMII контролер. Проблематикою розглянутих платформ є закритість документації на те як функціонує вбудований відеоспівпроцесор. Отримати доступ до якої можливе лише за умови підписання договору про нерозголошення. Тому даний блок розглядається як «чорний ящик», який захоплює відеокадри з сенсору та виконує їх первинну обробку. З програмної точки зору, реалізація доступу до даного блоку реалізується у форматі так званого «символьного пристрою» [9]. У системі Linux повинен ініціалізуватися файл /dev/video**X** відповідно (де **X** – номер пристрою відеозахоплення). Такий пристрій може оперувати як стандартний файл. Тобто його можна відкрити, прочитати (відобразити) блок даних до оперативної пам'яті та за потребою, та, наприкінці, закрити.

Для процесору лінійки RockChip RK3288 (Asus Tinker board), архітектура взаємодії відеоспівпроцесору відображена на рисунку 2.1.

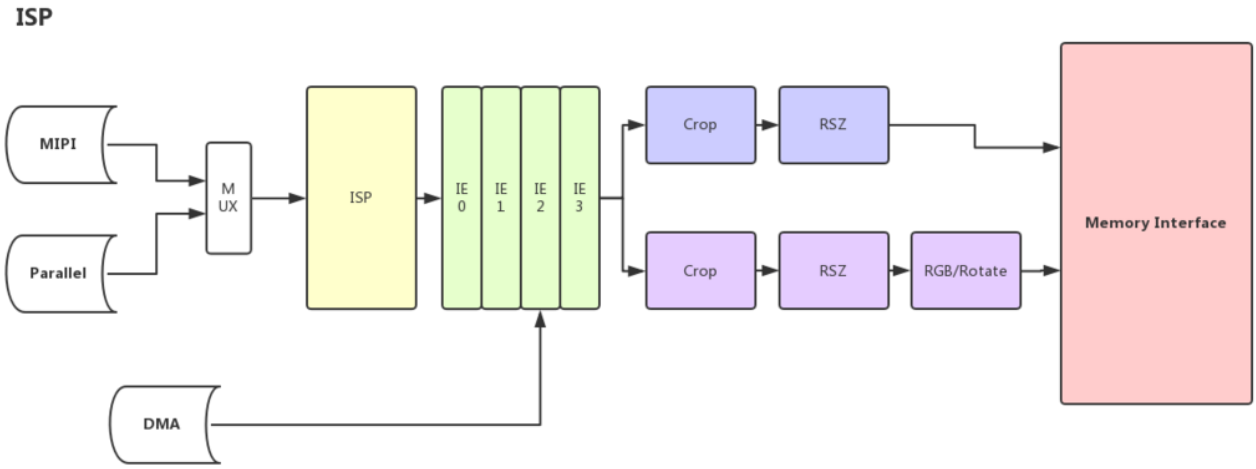


Рисунок 2.1 – Апаратний конвеєр обробки "сирих" зображень

Блок MIPI виступає у якості початкового драйверу, який взаємодіє з сенсором та транслює “сирі” зображення до блоку ISP – співпроцесору цифрових сигналів, що використовується для обробки зображень. Кінцеве захоплення кадрів зі співпроцесору відбувається з використанням вбудованої підсистеми ядра Linux – Video4Linux2 (V4L2) [10].

Зокрема, прототип GigE Vision камери вирішено створювати на базі апаратної платформи Raspberry Pi 4B та CSI сумісного відеосенсору OV5647. Платформа Raspberry Pi 4B була обрана за сукупністю властивостей, зокрема через її доступність, наявність широких обчислювальних можливостей завдяки обчислювальної потужному 4-х ядерному процесору BCM2711 на основі ядер Cortex-A72, що працюють на частоті 1,5 ГГц, наявність необхідної периферії: Ethernet контролера з підтримкою стандарту 1000Base-T Gigabit Ethernet, контролера відеозахоплення для отримання зображень з відеосенсору, а також, що важливо, базового пакету програмної підтримки, необхідного для швидкого прототипування. Так, наприклад, ввімкнення відповідного інтерфейсу камери, за допомогою утиліти raspi-config, необхідно обрати опцію зображену на рисунку 2.2 та перезавантажити систему.

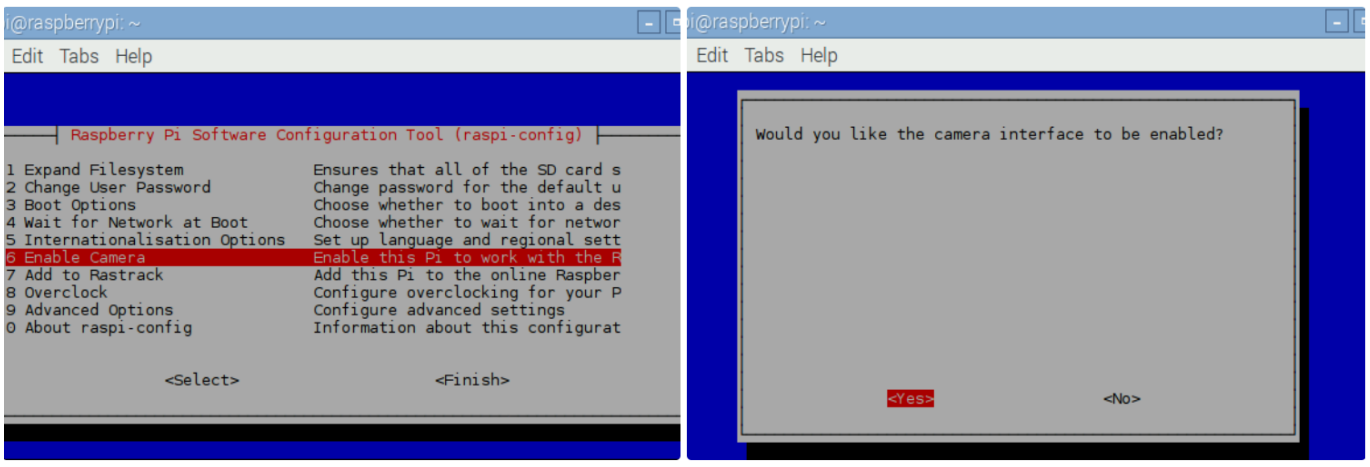


Рисунок 2.2 – Приклад конфігурації драйверу камери – утиліта raspi-config

Було прийнято рішення про використання дистрибутиву, що має назву Raspbian. Даний дистрибутив побудовано на базі дистрибутиву Debian. А тому, має вихід до встановлення необхідних пакетів для встановлення потрібних утиліт, бібліотек та модулів ядра.

З огляду на те, що реалізація GigE Vision сумісної камери можлива також і на інших обчислювальних платформах (на основі інших систем на кристалі) щоб за необхідності її можна було легко перенести на ці платформи, прототип створювався з якомога більш апаратно-незалежною структурною організацією, показаною на рисунку 2.3.

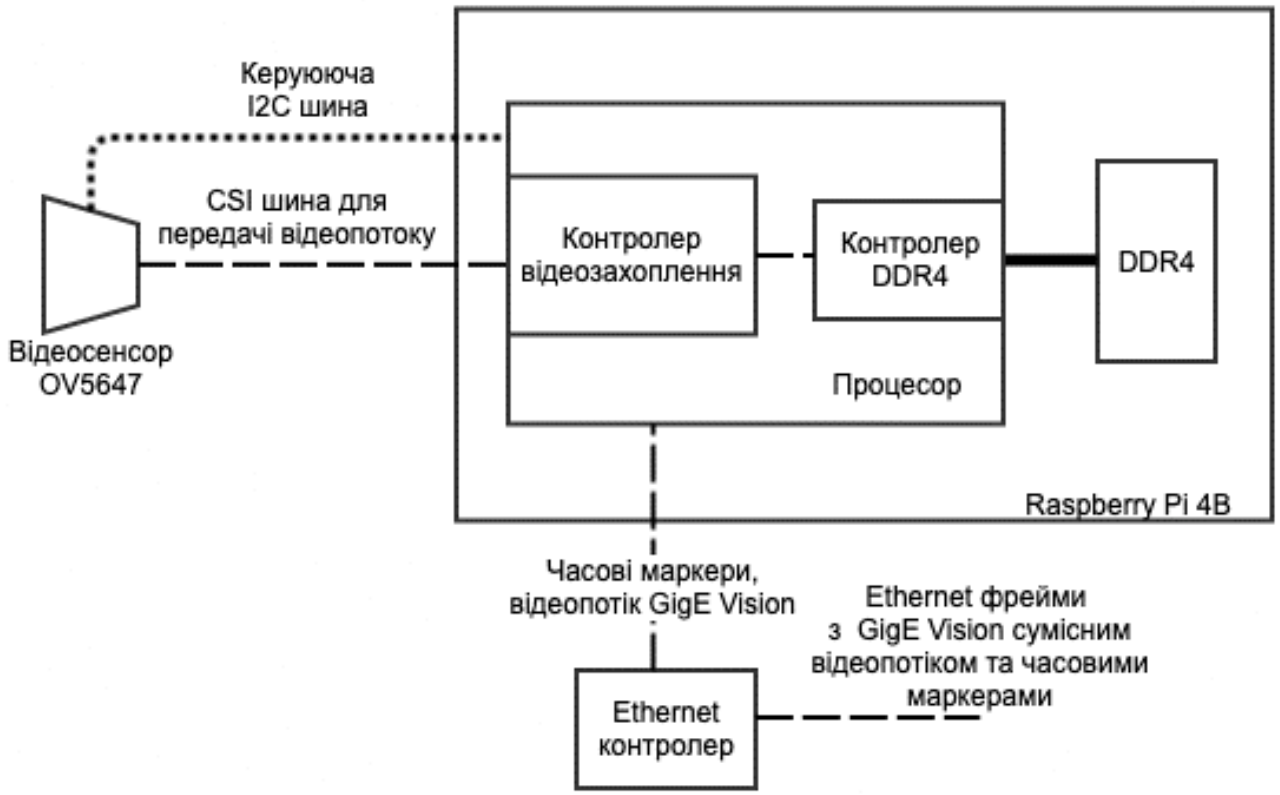


Рисунок 2.3 – Блок-схема розробленого прототипу системи захоплення відеопотоку

2.2 Огляд програмно-апаратних деталей механізму захоплення відеопотоку

Перш за все, необхідно було дослідити який саме відеосенсор може бути сумісним з вбудованим CSI контролером відлагоджувальної плати Raspberry Pi 4B. З документації спільноти Raspberry визначено, що для процесора BCM2711 доступними відеосенсорами, до яких офіційно реалізовано та підтримуються драйвери є лише три сенсори – OV5647, IMX219 та IMX477. Найбільш доступним серед них є сенсор OV5647, тому для подальших досліджень детальніше зупинимось саме на ньому.

Розглянемо детальніше архітектуру відеосенсору Omnivision OV5647. Для цього, розберемо архітектуру зображену на рисунку 2.4.

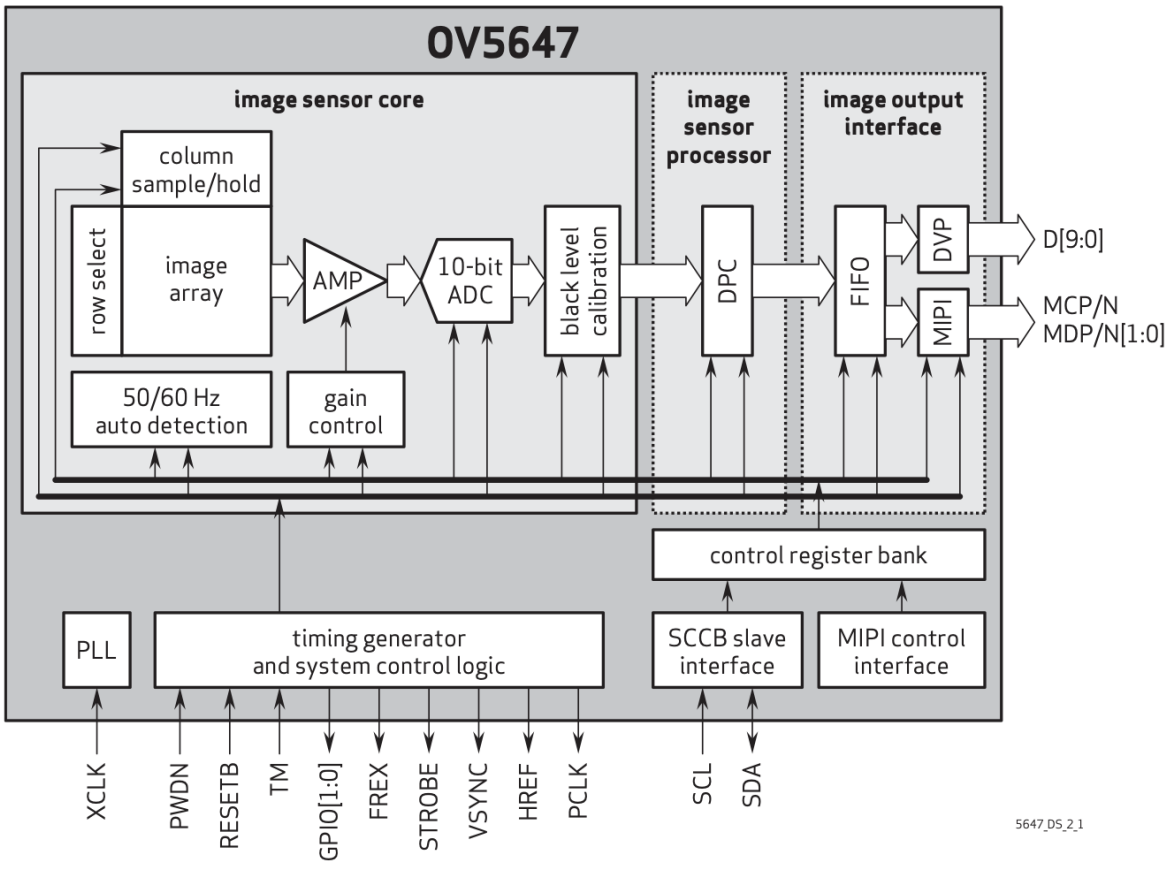


Рисунок 2.4 – Внутрішня будова модулю камери OV5647 у вигляді блок-схеми

Бачимо, що для керування модулем використовується стандартний I2C інтерфейс (Виводи SCL, SDA), за допомогою якого передаються керуючі послідовності, що зберігаються у регістрах керування. Для тактування PLL використовується опорна частота (вивід XCLK), рівна діапазону від 24 до 27 МГЦ. А також, опціональні виводи керування PWDN, RESETB, TM. З іншого боку, модуль має виводи, що дозволяють передавати дані від модулю до кінцевого пристрою, наприклад напряму до процесору. Шина D[9:0] є 10-ти бітною шиною даних, якою передається зображення, а виводи MCP/MDP є диференційним синхронізуючим сигналом. Такі виводи як VSYNC, STROBE, FREX, GPIO[1:0] інформують кінцевий пристрій про готовність кадру.

Розберемо з яких компонент складається модуль камери OV5647. Image array – Сенсор типу CMOS, представляє собою структуру з 2624 строк та 1956 стовпців (5,132,544 пікселів). Дана структура представляє собою масив кольорових фільтрів. Вони, у свою чергу вкривають фотодіоди та мають назву фільтр Байєра. Масив фільтру складається на 50% з зелених елементів та 25% з синіх та стільки ж червоних. Зауважимо, що зелений колір є переважним. Це пов’язано з тим, що око людини найкраще сприймає зелений спектр. Масив елементів фільтра зображено на рисунку 2.5.

Сигнал з кожного фотодіоду поступає на вхід аналогового підсилювача. Останній підсилює значення рівня напруги з коефіцієнтом, що встановлюється блоком Gain Control. Таким чином можливо регулювати яскравість зображення. Підсилені аналогові сигнали, один за одним, поступають на вхід АЦП. У результаті, на виході останнього, формується 10-ти бітне значення, що є представленням кольору. Оцифровані сигнали проходять калібрацію чорного кольору та поступають на вхід препроцесора, що виконує інтерполяцію пікселів фільтру. Кожне зображення синхронізується шляхом вкладання в чергу та транслюється до CSI інтерфейсу щоб передати на вихід (рисунок 2.6).

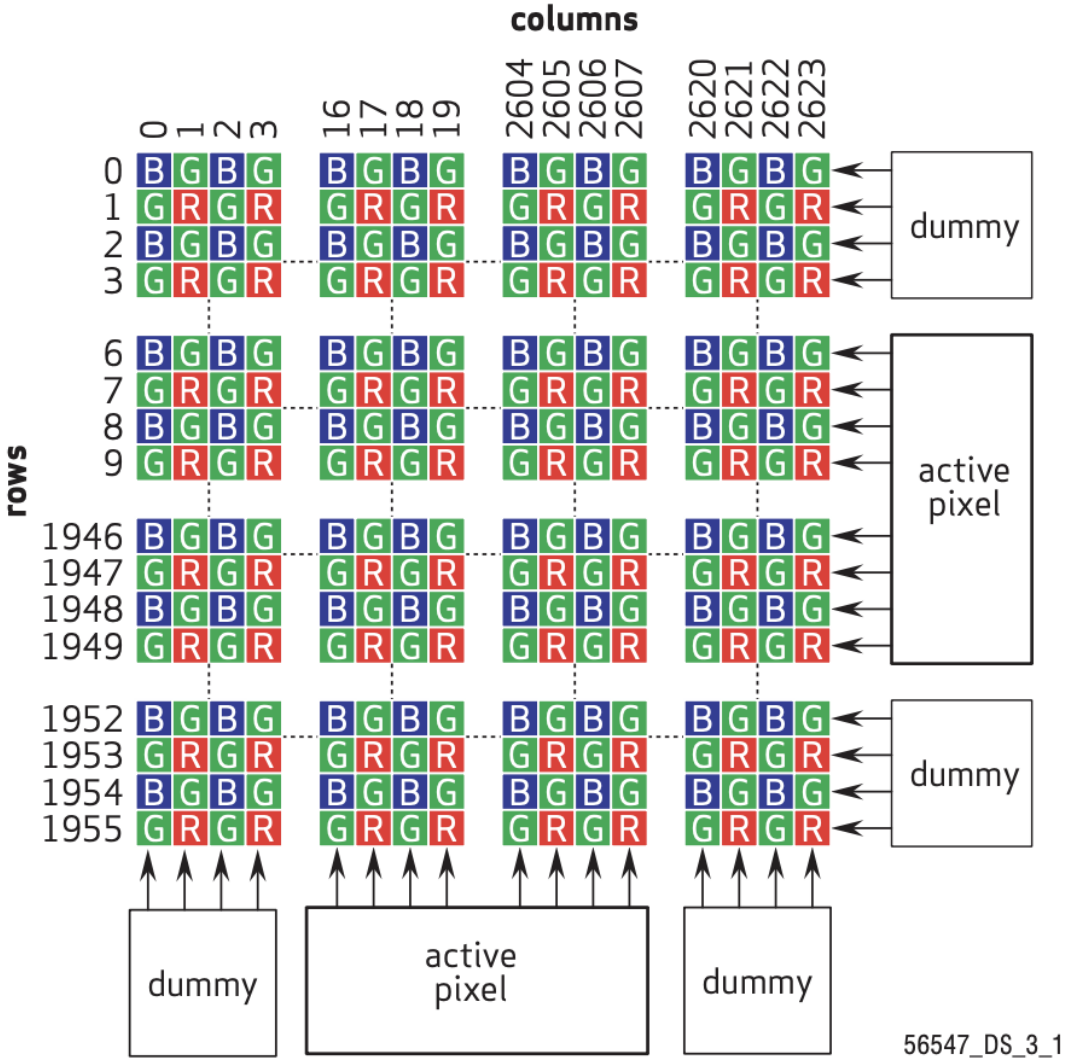


Рисунок 2.5 – Структурна організація пікселів сенсору розміром 2624x1956

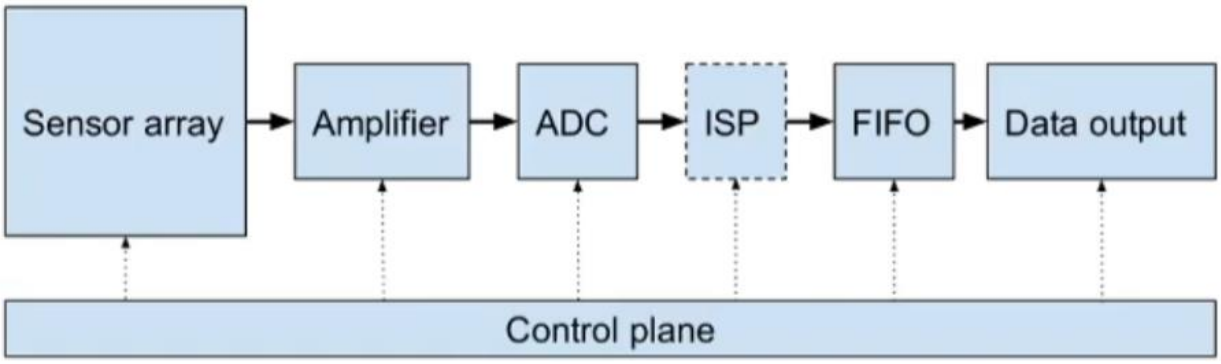


Рисунок 2.6 – Блок схема тракту оброки відеоданих

З апаратної точки зору обрана система на кристалі має вбудований блок відеозахоплення (BCM2711 VideoCore співроцесор), що виконує отримання кадрів від сенсора, первинну їх обробку та збереження до зовнішньої оперативної пам’яті. Завдяки цьому забезпечується програмна абстракція збереження у пам’яті зображень, які далі уніфікованим чином можуть бути передані користувацьким сервісам.

Розглянемо драйвер, що входить в склад ядра Linux для роботи з модулем OV5647.

Виявилось, що для керування модулем існує спеціалізований модуль Linux ядра, що має назву ov5647.c. Зауважимо, що задача даного драйверу є налаштування на керування модулем.

Розглянемо чим саме керує даний драйвер.

Вмиканням та вимиканням модулю камери:

```
static int ov5647_power_on(struct device *dev)
static int ov5647_power_off(struct device *dev)
```

Ввімкнення та вимкнення відеопотоку:

```
static int ov5647_s_stream(struct v4l2_subdev *sd, int enable)
```

де `enable` – аргумент з відповідним значенням стану.

Налаштування балансу білого:

```
static int ov5647_s_auto_white_balance(struct v4l2_subdev *sd,
                                       int val)
```

Налаштування підсилювача напруги:

```
static int ov5647_s_analogue_gain(struct v4l2_subdev *sd, u32 val)
```

Налаштування експозиції:

```
static int ov5647_s_exposure(struct v4l2_subdev *sd, u32 val)
```

Варто зазначити, що описані вище процедури є низькорівневими. У свою чергу, більш високорівнева, підсистема відео у Linux, Video4Linux, керує модулем використовуючи саме ці процедури. Приклад того як вона може бути

використана:

```
static int ov5647_s_ctrl(struct v4l2_ctrl *ctrl)
{
…
     switch (ctrl->id) {
     case V4L2_CID_AUTO_WHITE_BALANCE:
          ret = ov5647_s_auto_white_balance(sd, ctrl->val);
          break;
     case V4L2_CID_AUTOGAIN:
          ret = ov5647_s_autogain(sd, ctrl->val);
          break;
     case V4L2_CID_EXPOSURE_AUTO:
          ret = ov5647_s_exposure_auto(sd, ctrl->val);
          break;
     case V4L2_CID_ANALOGUE_GAIN:
          ret =  ov5647_s_analogue_gain(sd, ctrl->val);
          break;
     case V4L2_CID_EXPOSURE:
          ret = ov5647_s_exposure(sd, ctrl->val);
          break;
}
```

Де `*ctrl` вказівник на структуру, що заповнюється за допомогою підсистеми драйверів V4L автоматично або з конфігурації користувача.

За концепцією ядра Linux, при відключенні або підключенні пристрою до системи, драйвер автоматично вивантажується або завантажується. При завантаженні визиваються функції реєстрації. Для того щоб система мала представлення які процедури необхідно визвати, потрібно описати відповідну структуру даних:

```
static struct i2c_driver ov5647_driver = {
     .driver = {
          .of_match_table = of_match_ptr(ov5647_of_match),
          .name     = "ov5647",
          .pm  = &ov5647_pm_ops,
     },
     .probe_new     = ov5647_probe,
     .remove   = ov5647_remove,
     .id_table = ov5647_id,
};
```

Де `ov5647_id` відповідає моделі модуля, що сконфігурована наступним чином:

```
static const struct of_device_id ov5647_of_match[] = {
     { .compatible = "ovti,ov5647" },
     { /* sentinel */ },
};
```

При підключенні модуля, визивається процедура ov5647_probe, що конфігурує початкові регістри модуля, передає керування відеопідсистемі та реєструється як окремий її компонент:

```
v4l2_i2c_subdev_init(sd, client, &ov5647_subdev_ops);
v4l2_async_register_subdev(sd);
```

З цього моменту, підсистема V4L2 має запис про новий пристрій в системі та буде виступати мостом між драйверами апаратури (модулем камери) та користувачем. У свою чергу, користувач може взаємодіяти з даною підсистемою системними викликами, не вивчаючи особливості кожного модулю ядра, що відповідає за захоплення відеопотоку.

Коли ми визначили, що даний модуль ядра готовий для взаємодії з реальним пристроєм, необхідно повідомити Ядро Linux про наявність такого пристрою на платі, додавши його опис до основного дерева пристроїв. Фрагмент, що описує пристрій на платі Raspberry Pi взгляда наступним чином:

```
fragment@0 {
          target = <&i2c_csi_dsi>;
          __overlay__ {
               #address-cells = <1>;
               #size-cells = <0>;
               status = "okay";
               ov5647: ov5647@36 {
                    compatible = "ovti,ov5647";
                    reg = <0x36>;
                    status = "okay";
                    pwdn-gpios = <&gpio 41 1>, <&gpio 32 1>;
                    clocks = <&ov5647_clk>;
                    rotation = <0>;
                    orientation = <2>;
                    port {
                         ov5647_0: endpoint {
                              remote-endpoint = <&csi1_ep>;
                              clock-lanes = <0>;
                              data-lanes = <1 2>;
                              clock-noncontinuous;
```

```
                        link-frequencies =
                             /bits/ 64 <297000000>;
                    };
                };
            };
        };
```

Зауважимо, для того щоб ядро знайшло який саме модуль ядра необхідно завантажити, поле `compatible` повинно співпадати з таким ж самим полем структури `of_device_id`. Дані, що вказані у полях фрагменту, описують яке джерело тактової частоти, CSI інтерфейс, адреса модуля керування, поключеного до на шини I2C, тощо. За допомогою процедури `ov5647_parse_dt()`, дані фрагменту буде зчитано та збережено при включенні драйверу ov5647.c у систему.

Загалом, на рисунку 2.7. графічно показано трьошарову програмно-апаратну взаємодію програмних та апаратних блоків в ОС під керуванням ядра Linux, що було описано вище [11].

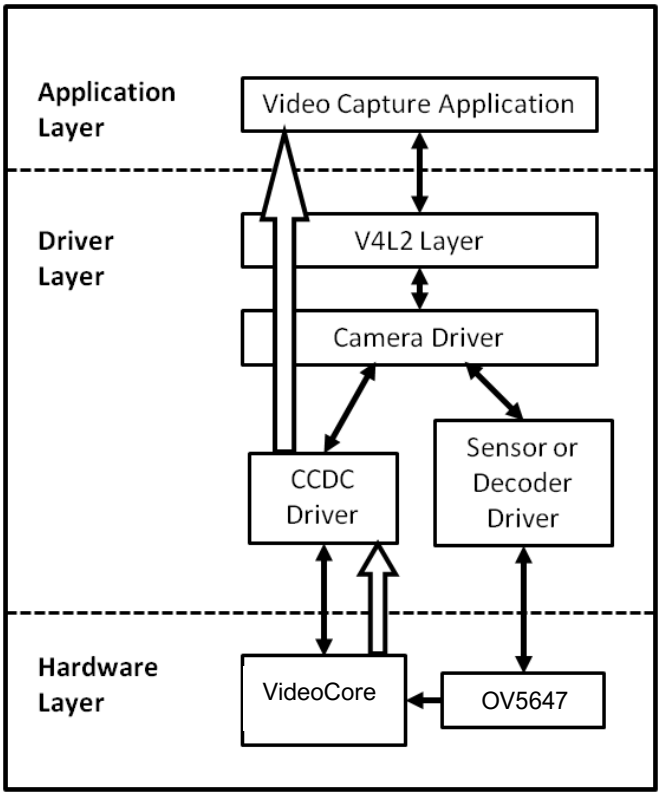


Рисунок 2.7 – Трьошарова модель апаратного та програмного рівнів взаємодії

2.3 Розробка драйверу захоплення відеопотоку

Як вже відомо, що з огляду на те, що дана підсистема Video4Linux працює у просторі ядра, її мета надати користувацький інтерфейс керування камерою.

Для отримання кадрів використано відповідну V4L2 (Video4Linux2) підсистему драйверів ядра Linux. З огляду на те, що дана підсистема працює у просторі ядра, V4L2 надає користувацький інтерфейс керування у вигляді виклику системної функції **`ioctl`**(). Даний виклик виконується до відеопристрою, що зазвичай ініціалізується у системі як файл з директорії **`/dev`** з іменем **`videoX`**, де **`X`** – номер зареєстрованого в системі пристрою відеозахоплення. Процедура отримання кадру із пристрою джерела відеопотоку детально описана у відповідній документації Linux [5, 6].

Використовуючи підсистему Video4Linux, було розроблено власну процедуру відеозахоплення [12], що працює у користувацькому просторі та передає захоплені кадри відеопотоку до підсистеми, що реалізує мережеву складову створюваного прототипу камери. Оскільки, через помилки передачі або дію зовнішніх факторів, потік даних може передаватись мережею з неоднорідним темпом. При цьому, зображення перед відправленням зберігається до кільцевого буферу, кожен елемент якого містить окремий кадр [13]. Розмір буферу може задаватися користувачем як параметр на етапі конфігурації. Організація первинного захоплення кадрів з підсистеми Video4Linux є наступною: підсистема зберігає кадри з простору пам’яті ядра у елементи буферу користувацької пам’яті, інформує про готовність кадру, передає вказівник на початок кадру та його розмір. У результаті, вказівник на буфер та його розмір може бути передано до користувацьких сервісів (в тому числі мережевої підсистеми), які, у свою чергу, матимуть змогу скопіювати вказану кількість байтів, з яких складається зображення.

Підсистема V4L виступає як клас драйверів, що уніфікують механізм для

простору користувача з метою взаємодії з відеосенсорами різних виробників за допомогою відповідних апаратно-залежних драйверів. Таким чином, один V4L драйвер може бути використаний для взаємодії з багатьма камерами, причому на різних вбудованих апаратних платформах.

Розглянемо програмну частину драйвера захвату відеопотоку:

Створення об'єкту камери:

```
c = v4l2_create_camera(w, h, pix_fmt);
 if (c == NULL)
      goto camera_not_ready;
```

Дана функція включає у себе виділення пам’яті до «кучі», розміром структури v4l2_camera:

```
struct v4l2_camera *c;
c = calloc(1, sizeof(*c));
```

Та встановлення параметрів: ширина, висота та формат кадру, який буде захоплено за камери:

```
c->params.width = width ? width : V4L2_WIDTH_DEFAULT;
c->params.height = height ? height : V4L2_HEIGHT_DEFAULT;
c->params.pixel_format = pix_fmt ? pix_fmt :
                            V4L2_PIXEL_FORMAT_DEFAULT;
```

Відкриття файлу камери, наприклад /dev/video0:

```
c->vfd = v4l2_open_device(video_dev);
 if (c->vfd < 0)
      goto null_camera;
```

Що базується на системному виклику **open()**:

```
    vfd = open(dev_path ? dev_path : def_dev_path, O_RDWR |
O_NONBLOCK, 0);
```

Встановлення налаштувань камери:

```
 rc = v4l2_getset_capability(c);
 if (rc != 0)
      goto broken_camera;
```

Для виконання даної процедури необхідно запросити структуру

конфігурацією, яку має камера та, навпаки, може встановити:

```
struct v4l2_capability cap;
rc = xioctl(c->vfd, VIDIOC_QUERYCAP, &cap);
```

Після чого виконується завантаження нової конфігурації:

```
c->params.field = V4L2_FIELD_DEFAULT;
c->params.type  = V4L2_BUF_TYPE_VIDEO_CAPTURE;
```

Виділення буферів в оперативній пам'яті кількістю **buf_num**. Для досліду, кількість mplane повинна бути рівною одиниці:

```
rc = v4l2_allocate_fb(c, buf_num, mplane_num);
if (rc != 0)
     goto broken_camera;
```

Виділення пам'яті для кожного буферу:

```
for ( ; i < c->fb_num && rc == 0; i++) {
     c->fb[i].f.head = calloc(c->mplane_num,
                              sizeof(*(c->fb[i].f.head)));
     if (c->fb[i].f.head == NULL) {
          rc = -ENOMEM;
          break;
     }

     c->fb[i].f.length = calloc(c->mplane_num,
                                sizeof(*(c->fb[i].f.length)));
     if (c->fb[i].f.length == NULL) {
          rc = -ENOMEM;
          break;
     }
}
```

Налаштування формату кадру:

```
rc = v4l2_getset_format(c);
if (rc != 0)
     goto null_buf;
```

Необхідно запросити структуру для внесення даних про формат:

```
struct v4l2_format  fmt;
rc = xioctl(c->vfd, VIDIOC_G_FMT, &fmt);
```

Встановлення формату (ширина, виоста, формат кадру):

```
fmt.fmt.pix.width  = c->params.width;
fmt.fmt.pix.height = c->params.height;
fmt.fmt.pix.field  = c->params.field;
fmt.fmt.pix.pixelformat = c->params.pixel_format;
```

Відображення кожного v4l2 буферу до буферу користувача:

```
rc = v4l2_mmap_camera(c);
if (rc != 0)
     goto null_buf;
```

Початок процедури. Процедура повторюється **fb_num** разів:

```
for (size_t i = 0; i < c->fb_num && rc == 0; i++) {
     struct v4l2_buffer mmap_fb;
     memset(&mmap_fb, 0, sizeof(mmap_fb));

     mmap_fb.index  = i;
     mmap_fb.memory = V4L2_MEMORY_MMAP;
     mmap_fb.type   = c->params.type;
```

Запит v4l2 буферу:

```
rc = xioctl(c->vfd, VIDIOC_QUERYBUF, &mmap_fb);

if (rc != 0) {
     LOG("VIDIOC_QUERYBUF failed: %d, %s\n", errno,
strerror(errno));

if (errno == EINVAL)
     LOG("The buffer type is not supported, or the index is
out of bounds");
     }
```

Відображення v4l2 буферу:

```
if (rc == 0) rc = v4l2_mmap_fb(c, &mmap_fb, i);
```

Механізм відображення виконується за рахунок системного визову **mmap()**.

```
case V4L2_BUF_TYPE_VIDEO_CAPTURE:
               c->fb[buf_index].f.length[j] = mmap_fb->length;
               c->fb[buf_index].f.head[j] = mmap(NULL,
                              mmap_fb->length,
                              PROT_READ | PROT_WRITE,
                              MAP_SHARED,
                              c->vfd,
                              mmap_fb->m.offset);
```

Запит на додавання буферів до черги:

```
rc = v4l2_enqueue_all_buf(c);
if (rc != 0)
     goto mmap_failed;
```

Початок процедури:

```
for (size_t i = 0; i < c->fb_num && rc == 0; i++) {
     struct v4l2_buffer v4l2_buf;
```

Запит на отримання вказівника на v4l2 буфери:

```
memset(&v4l2_buf, 0, sizeof(v4l2_buf));

v4l2_buf.memory     = V4L2_MEMORY_MMAP;      v4l2_buf.field =
          c->params.field;
v4l2_buf.type  = c->params.type;
v4l2_buf.index = i;
```

Додавання буферів у чергу:

```
rc = xioctl(c->vfd, VIDIOC_QBUF, &v4l2_buf);
if (rc != 0)
     LOG("VIDIOC_QBUF failed");
     LOG("Camera buffer[%zu] flag: 0x%x", i, v4l2_buf.flags);
}
```

Вмикання відеопотоку:

```
rc = v4l2_stream_on(c);
if (rc != 0)
     goto mmap_failed;
```

Запит на вмикання відеопотоку з камери:

```
xioctl(c->vfd, VIDIOC_STREAMON, &c->params.type);
```

Ініціювання запуску POSIX потоку для опитування відеофайлу на наявність даних:

```
rc = v4l2_start_thread(c);
if (rc != 0) goto stream_off;
```

Основа даної процедури:

```
rc = pthread_create((pthread_t *)&v4l2_thread,
              NULL,
              v4l2_poll_frame_thread,
              (void*)c);
```

У свою чергу, зворотня функція **v4l2_poll_frame_thread** буде запущена в окремому потоці.

Процедура опитування файлового дескриптору:

```
rc = poll(&fds, 1, 2);
```

При наявності даний та встановленні прапору готовності даних, викликається процедура запиту на зчитування номеру кадру з відеокадром та запит нового.

```
if (fds.revents & POLLIN &&
    c->v4l2_is_frame_ready == false)
      v4l2_dequeue_enqueue_buf(c);
```

У свою чергу, процедура вивантаження номеру коректорону буферу з даними у вигляді зображення:

```
rc = xioctl(c->vfd, VIDIOC_DQBUF, &v4l2_buff);
```

Копіювання номеру буферу де зберігається зображення:

```
c->fb->index = v4l2_buff.index;
```

Копіювання розміру зображення зберігаємого у буфері:

```
c->fb->bytes_used = v4l2_buff.bytesused;
```

Запит нового номера буферу:

```
rc = xioctl(c->vfd, VIDIOC_QBUF, &v4l2_buff);
```

Таким чином, у контексті програми, користувач знає адресу початку буферу, його номер та розмір. Відповідно, має надається доступ до зображення. Приклад запису кадру до файлу:

```
write(record_fd,
      c->fb[c->fb->index].f.head[0],
      c->fb->bytes_used);
```

Де record_fd – файловий дескриптор відкритого у файловій системі об’єкту у вигляді файлу.

Зауважимо, що кожний модуль камери має власний набір характеристик (форматів), яким один відрізняється від іншого. Тому, при розробці драйверу захвату відеопотоку дані параметри необхідно взяти до уваги. Для вирішення даної задачі, скористаємось утилітою v4l2-ctl:

```
pi@raspberrypi:~ $ v4l2-ctl -d /dev/video0 --list-formats-ext
```

```
ioctl: VIDIOC_ENUM_FMT
  Type: Video Capture
  [0]: 'YU12' (Planar YUV 4:2:0)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [1]: 'YUYV' (YUYV 4:2:2)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [2]: 'RGB3' (24-bit RGB 8-8-8)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [3]: 'JPEG' (JFIF JPEG, compressed)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [4]: 'H264' (H.264, compressed)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [5]: 'MJPG' (Motion-JPEG, compressed)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [6]: 'YVYU' (YVYU 4:2:2)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [7]: 'VYUY' (VYUY 4:2:2)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [8]: 'UYVY' (UYVY 4:2:2)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [9]: 'NV12' (Y/CbCr 4:2:0)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [10]: 'BGR3' (24-bit BGR 8-8-8)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [11]: 'YV12' (Planar YVU 4:2:0)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [12]: 'NV21' (Y/CrCb 4:2:0)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
  [13]: 'RX24' (32-bit XBGR 8-8-8-8)
    Size: Stepwise 32x32 - 2592x1944 with step 2/2
```

Проаналізувавши набір доступних форматів, оберемо один з них, вказавши відповідний параметри у заголовному файлі програми.

```
#define V4L2_PIXEL_FORMAT_DEFAULT  V4L2_PIX_FMT_YUYV
```

Подібним чином, вкажемо розширення з яким буде отримано відеопотік:

```
#define V4L2_WIDTH_DEFAULT         640
#define V4L2_HEIGHT_DEFAULT        480
```

Та на кількість буферів для збереження кадрів:

```
#define V4L2_REQUESTED_BUFFERS_NUM 10
```

Повний код для захоплення відеопотоку наводиться у додатку В.

2.4 Інтеграція GigE Vision сумісної бібліотеки Aravis

Бібліотека Aravis є однією, якщо не єдиною відкритою та найбільш повноцінною бібліотекою для роботи з GigE Vision сумісними пристроями [14]. Aravis основана на низькорівневих бібліотеках glib/gobject та розповсюджується за вільною ліцензією LGPL v2+. Виходячи з цього, було прийнято рішення дослідити дану бібліотеку на предмет її використання як програмного компоненту, що працює у користувацькому просторі Linux та реалізує мережеву складову прототипу камери, що розробляється.

Ідея використання бібліотеки Aravis полягає у тому, що в її складі реалізовано емулятор GigE Vision камери. Підмінивши в даному емуляторі синтетичне зображення на реальне дозволить створити власну GigE Vision сумісну камеру. Така підміна може бути легко здійснена шляхом передачі вказівника на кадри, які зберігаються у відеобуфері, отримувані підсистемою відеозахоплення, що була описана у попередньому підрозділі [15].

Зокрема, для створення віртуальної GigE Vision камери, що здатна транслювати реальний відеопотік необхідно у модулі arvfakecamera.c бібліотеки Aravis внести незначні зміни, що включають:

1) ініціалізацію процесу захоплення кадрів – запуск підсистеми відеозахоплення, що заповнюватиме циклічний відеобуфер;

```
c = v4l2_start_video_capturing(NULL, 0 ,0 ,0, 0, 0);
```

2) вказати функцію, зворотного виклику, що має викликатись при запиті на передавання чергового кадру, для чого присвоїти вказівник на функцію у поле:

```
fake_camera->priv->fill_pattern_callback = v4l2_arv_camera_capture;
fake_camera->priv->fill_pattern_data = c;
```

3) реалізувати механізм відображення відеобуферу, тобто надати вказівник на початок кадру, що має бути переданий, шляхом вибору коректного

кадру із циклічного буферу та запису його адреси у поле:

```
buffer->priv->data = c->fb[c->fb->index].f.head[0];
```

При цьому, бібліотека Aravis бере на себе задачу конвертування зображень у Ethernet фрейми та їх передачу через мережевий сокет з використанням стеку TCP/IP, що є підсистемою ядра Linux.

Тестування передачі відеопотоку відбувалося за допомогою програми Aravis Viewer на стороні приймача. Зауважимо, що тільки декілька параметрів камери є важливими та є статичними відносно приймача. Приклад того як прототип GigE Vision камери ідентифікується у мережі зображено на рисунку 2.8 – а.

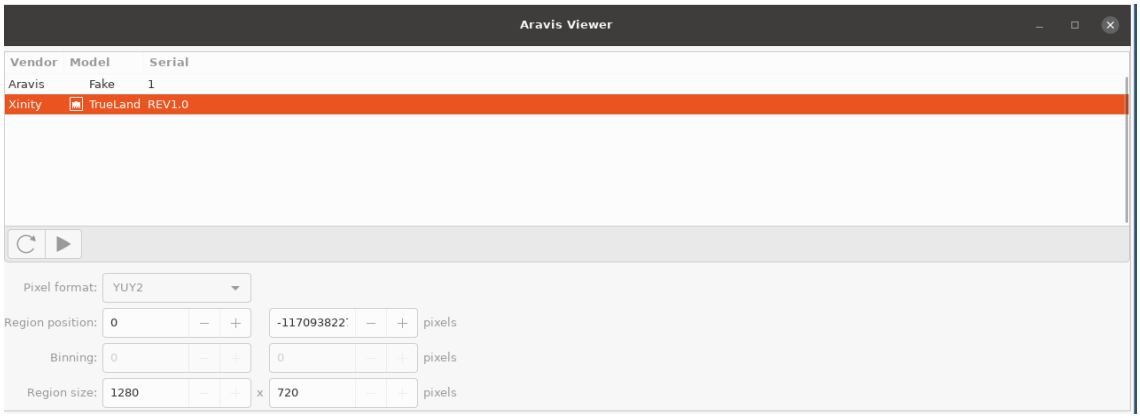
- Vendor - виробник, Model - модель, Serial – версія
- Pixel format – формат відео.
- Region size – розширення зображення.

Скомпілюємо систему захоплення, бібліотеку Aravis завдяки програмному засобу – системі компіляції ninja та запустимо програму **arv-fake-gv-camera-0.8**:

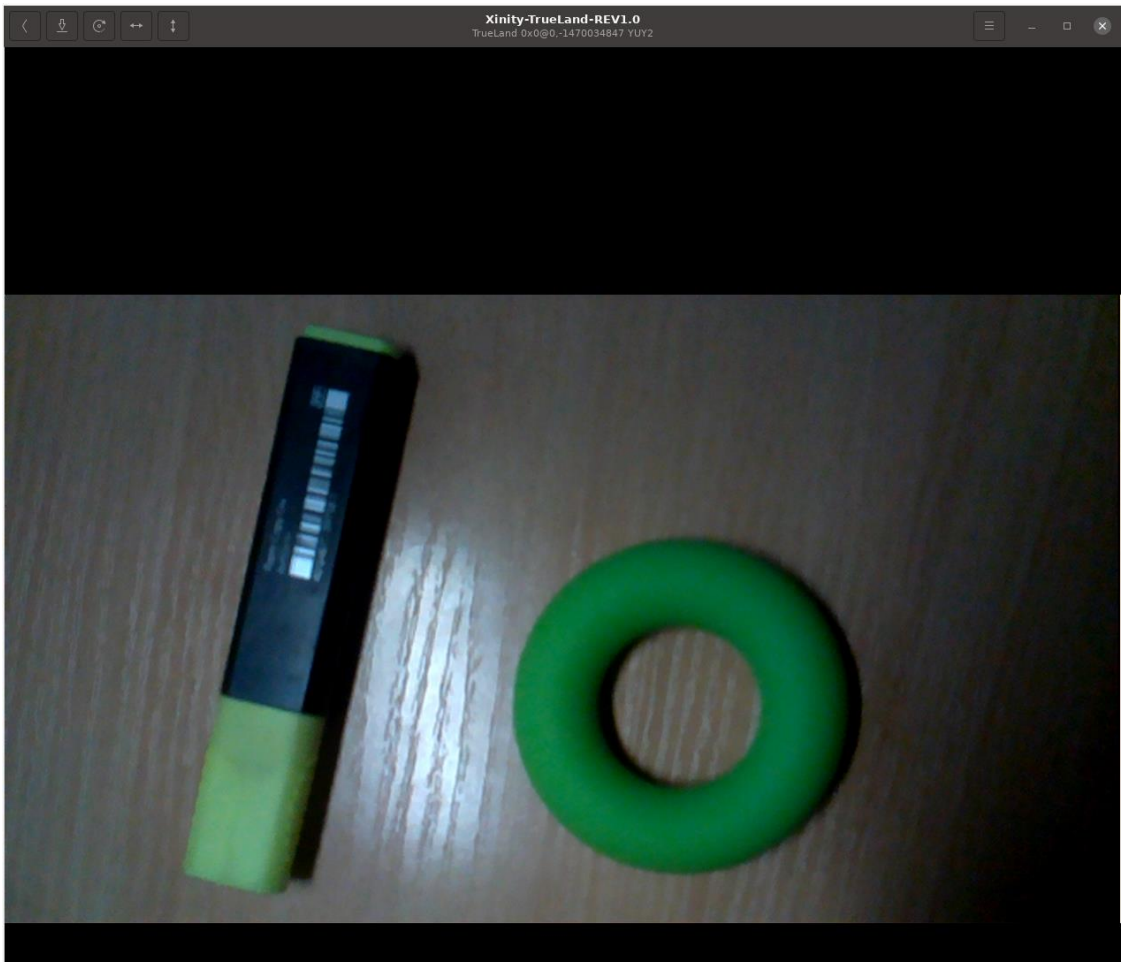
```
pi@raspberrypi:~/aravis-0.8.6/build $ ./src/arv-fake-gv-camera-
0.8 -i eth1
v4l2_capture: Camera has been created
v4l2_capture: Camera was opened: /dev/video0
v4l2_capture: V4L2_CAP_VIDEO_CAPTURE mode
v4l2_capture: Buffers number: 6
v4l2_capture: Mplane number: 1
v4l2_capture: Camera buffers have been allocated
v4l2_capture: Camera width: 640
v4l2_capture: Camera height: 480
v4l2_capture: Camera field was set 0
v4l2_capture: Camera type was set 1 type
v4l2_capture: Camera mmap mplane[0] length: 614400
v4l2_capture: Camera frame buffer [0] address: 0xb45eb000
v4l2_capture: Camera mmap mplane[0] length: 614400
v4l2_capture: Camera frame buffer [1] address: 0xb4555000
v4l2_capture: Camera mmap mplane[0] length: 614400
v4l2_capture: Camera frame buffer [2] address: 0xb44bf000
v4l2_capture: Camera has been mapped
```

На рисунку 2.8 - б зображено окремий кадр, переданий каналом Ethernet з

прототипу GigE Vision камери.



а)



б)

Рисунок 2.8 – а) ідентифікація GigE Vision камери у мережі; б) кадр формату YUY2, переданий каналом Ethernet

З дослідницькою метою, на заміну кадрів, отриманих з відеосенсору, було розроблено другу програмну частину, що представляє собою передачу статичного зображення (рисунки 2.9-а та 2.9-б).

Структурна організація програмної частини створюваного прототипу камери показана на рисунку 2.10. В наведеній структурній організації підсистема захоплення відеопотоку реалізується блоком «Драйвер відеозахоплення», а мережева підсистема модифікованим модулем **arvfakecam.c**.

З метою дослідження затримок каналу передачі, до кожного кадру додається певна часова мітка. Необхідність передачі часових міток розкривається першому розділі, а саме мета дослідження у розділі три. Для збереження часу, було використано структуру timeval.

Структура описує системний час, що складається зі змінних (полей структури) секунди (tv_sec) та мікросекунди (tv_usec). При переповненні останнього, інкрементується перший.

```
struct timeval {
      time_t      tv_sec;     /* секунди */
      suseconds_t tv_usec;    /* мікросекунди */
 };
```

Задекларуємо структуру з міткою часу.

```
struct timeval lapse;
```

Визначимо системний час у формі структури timeval.

```
gettimeofday(&lapse, NULL);
```

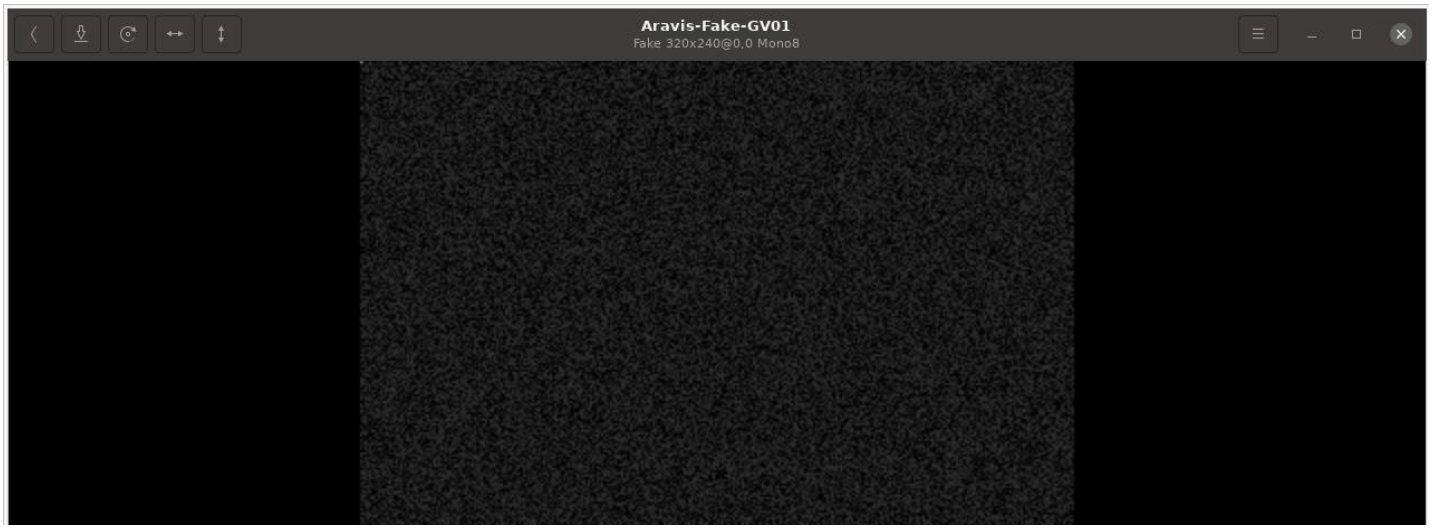
Побітово скопіюємо структуру з часом на початок кадру, що буде передано.

```
memcpy(buffer->priv->data,
       &lapse.tv_usec,
       sizeof(lapse.tv_usec));
```

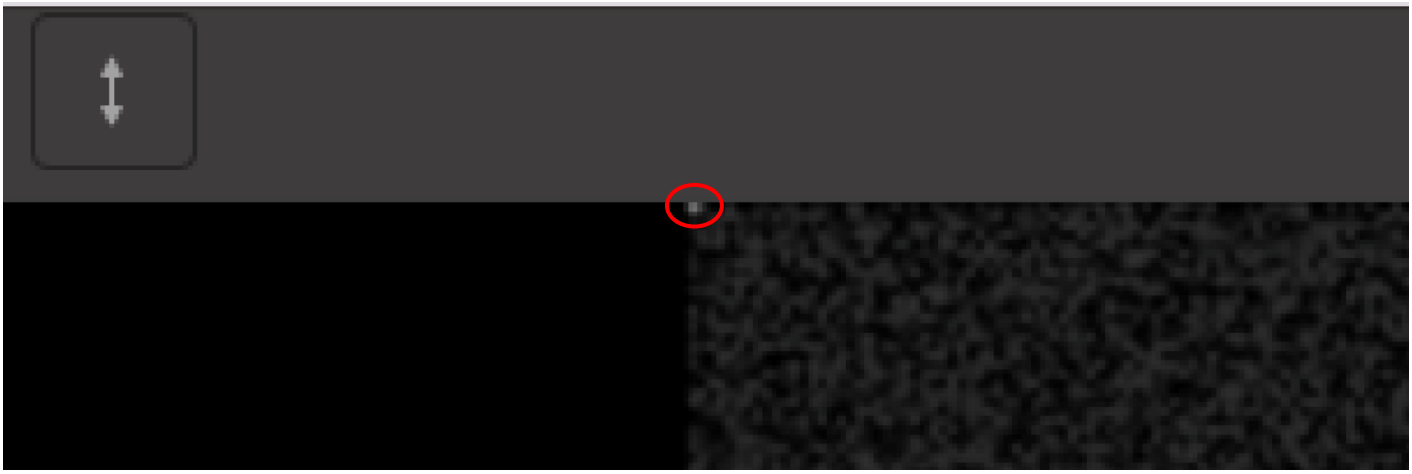
На рисунку 2.9-а зображено кадр з часовою міткою, отриманий на приймаючій стороні.

На рисунку 2.9-б видно ряд пікселів на початку кадру, що відрізняються від

інших рівнем яскравості. Саме вони несуть інформацію про час коли кадр було відправлено мережею.



а)



б)

Рисунок 2.9 – Статичне зображення передане каналом Ethernet: а) повне зображення; б) зображення збільшене з метою ідентифікувати дані з вкладеними часовими мітками

Також, рисунок 2.10 ілюструє повну програмну архітектуру підсистеми

щодо взаємодії бібліотек з компонентами операційної системи Linux та кінцевого клієнта що отримує відеопотік мережею.

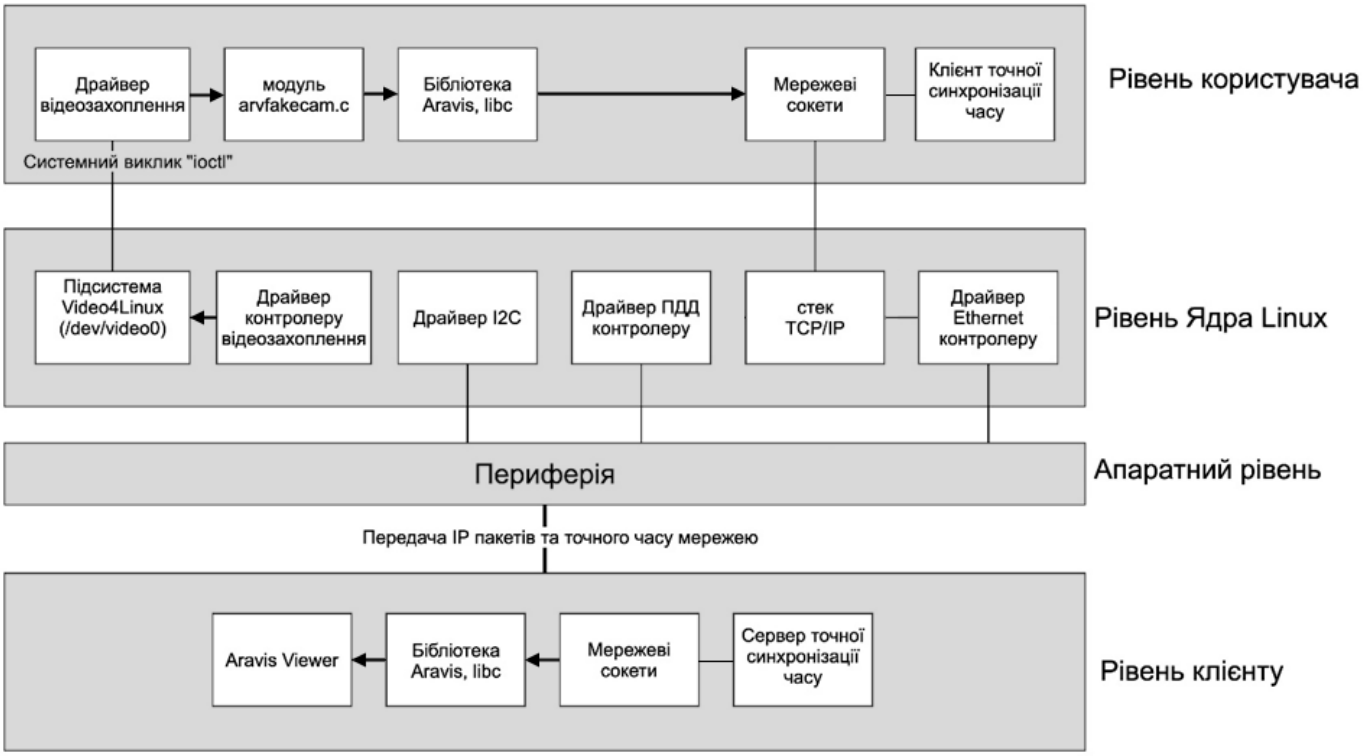


Рисунок 2.10 – Структурно-функціональна організація програмної частини прототипу GigE Vision камери

Висновок до розділу 2

У ході виконання другого розділу, розглянуто програмну архітектуру, драйвери та компоненти, що використовується в операційних системах побудованих на базі ядра Linux. Запропоновано структурну організацію програмних блоків, з яких має складатися прототип GigE камери. За структурною організацією, задачу розробки прототипу було розділено та виконано за двома блокам: розробка драйверу відеозахоплення та ітеграції бібліотеки Aravis для передачі відеопотоку каналом Ethernet. З дослідницької мети було розглянуто метод додавання часових міток для вимірювання затримок передавання кадру. За результатом розроблено прототип GigE Vision камери. Результат роботи

прототипу продемонстровано у вигляді ілюстрації з кадром кадру, отриманого на приймаючій стороні.

РОЗДІЛ 3 ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ПЕРЕДАВАННЯ ДАНИХ НА БАЗІ ПРОТОТИПУ GIGE VISION КАМЕРИ

Дослідження ефективності формування та передавання GigE Vision відеопотоку створеним прототипом включає два аспекти. Перший полягає у оцінюванні затримки передавання кадрів відеопотоку, а другий – у визначенні навантаження на обчислювальну систему, зокрема, процесор, що лежить в основі створюваного прототипу.

3.1 Методика оцінювання затримок передавання кадрів у дослідженні

Оцінювання затримок передавання кадрів було здійснено за часовими мітками формування кадру на стороні передавача та часу його отримання приймачем достатньо простим способом:

$$\tau = t_r - t_t, \tag{3.1}$$

де τ – затримка передавання; t_r – час отримання кадру приймачем; t_t – час початку процедури передавання кадру на стороні передавача.

Не дивлячись на простоту реалізації такої оцінки, з нею пов’язана порівняно суттєва проблема – для того, щоб отримати коректну затримку із різниці часових міток, час на приймачі та передавачі має йти синхронно.

Доречно зазначити, що дана проблема має потенційне рішення при застосуванні протоколу PTP (Precision Time Protocol), визначеного стандартом IEEE-1588. Суть цього протоколу полягає у тому, що вузли, на яких необхідно синхронізувати час, постійно обмінюються сповіщеннями, в яких передається час ведучого вузла (сервера часу), а також здійснюється корегування на величину часу, витраченого на проходження через мережеве обладнання [16]. Ця схема

підвищує точність видачі часу споживачам, компенсуючи затримки доставки повідомлень мережею. Стверджується, що для апаратної реалізації протоколу, синхронізація можлива з точністю до 1 мікросекунди, тоді як для програмної реалізації PTP, точність зазвичай не перевищує 100 мікросекунд [16]. Оскільки час формування зображення на відеосенсорі з огляду на реальні величини експозиції перевищують 100 мкс в рази, якщо не на порядки, точності, що забезпечується навіть програмною реалізацією PTP протоколу для оцінювання затримок можна вважати достатньою.

Зауважимо також, що оскільки дослідження проводиться для з'єднання точка-точка, у мережі наявні тільки два пристрої, тому загалом не становить суттєвої різниці, який саме з них буде джерелом точного часу, а який буде його відповідним споживачем. Водночас, в рамках даної роботи сервер точного часу працює на стороні приймача (клієнта).

З практичної точки зору функціонування PTP протоколу в операційній системі GNU/Linux забезпечується програмою **ptpd**. Її виклик з відповідними аргументами може здійснюватись, наприклад, наступним чином:

- для серверу точного часу:

```
ptpd -C --p2p -i eth0 –M
```

- для клієнту, що має синхронізувати свій час:

```
ptpd -C --p2p -i eth0 -s
```

де `eth0` – системне ім'я використовуваного мережевого інтерфейсу Ethernet. Синхронізація часу врахована у структурі створюваного прототипу камери відповідними блоками (див. рис. 2.10).

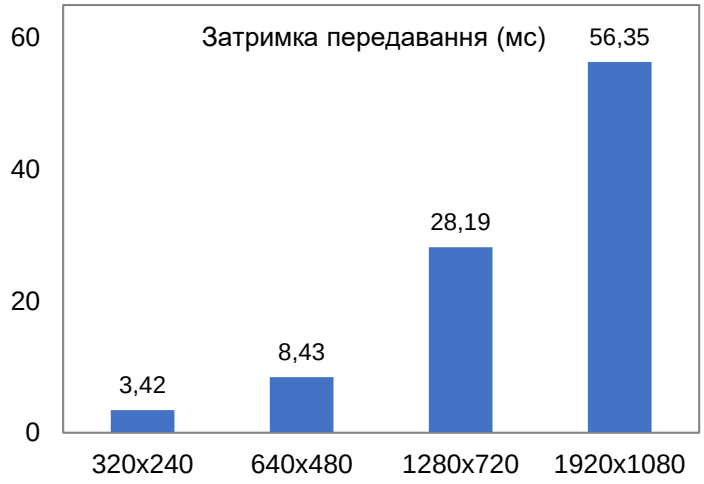
3.2 Результати досліджень

Оцінювання часу затримки передавання кадрів здійснено з використанням методики описаної в попередньому підрозділі для різних роздільних здатностей

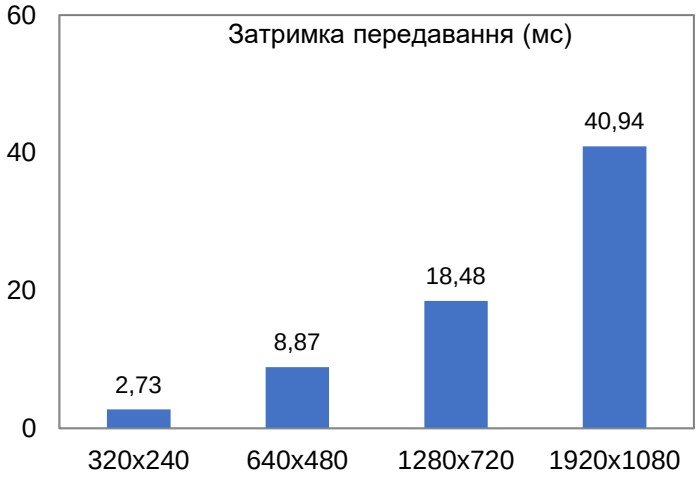
зображень та показано на рис. 3.1 - а та рис 3.1 - б. На рис. 4 показано відсоток завантаження процесорних ядер передавача (створюваного прототипу камери) при передаванні кадрів з відповідною роздільною здатністю.

З метою виключення впливу на час передавання процедури первинного захоплення кадрів, було проведено додатковий експеримент, в якому здійснювалось надсилання одного незмінного кадру, причому підсистема відеозахоплення на основі V4L залишалася відключеною. Результати експерименту показано на рис. 3.1 - в та рис. 3.1 - г, де наведено затримку передавання та відсоток завантаження ядер процесора відповідно.

Порівнюючи діаграми рис. 3.1 - а та 3.1 - б можна бачити, що час отримання кадрів з підсистемою V4L2 є досить високим та становить близько 30% від загального часу формування та передавання кадрів. Більше того, згідно графіків завантаження процесорних ядер (рис. 3.1 - б та 3.1 -г) близько 50% відсотків припадає на первинне захоплення кадрів. Причину цього можна пояснити тим, що в експерименті для зручності подальшого відображення кадрів на стороні передавача здійснювалось перетворення кадрів у формат YUV. У випадку, якщо це перетворення не виконувати, як це зазвичай відбувається в реальних камерах і передавати від сенсора вихідні (raw) дані минаючи процес перетворення, зазначені затримки можна помітно знизити. З іншого боку з рис. 3.1 - б та 3.1 - г видно, що постійно завантажується тільки одне ядро процесора, тому навіть за потреби виконання перетворень на тестовій платформі є резерв обчислювальних можливостей, які можуть бути використані шляхом розпаралелювання обчислень. Наприклад, за допомогою POSIX – сумнісної бібліотеки, що дає можливість роботи з потоками у середовищі адрес користувача що має назву POSIX thread.

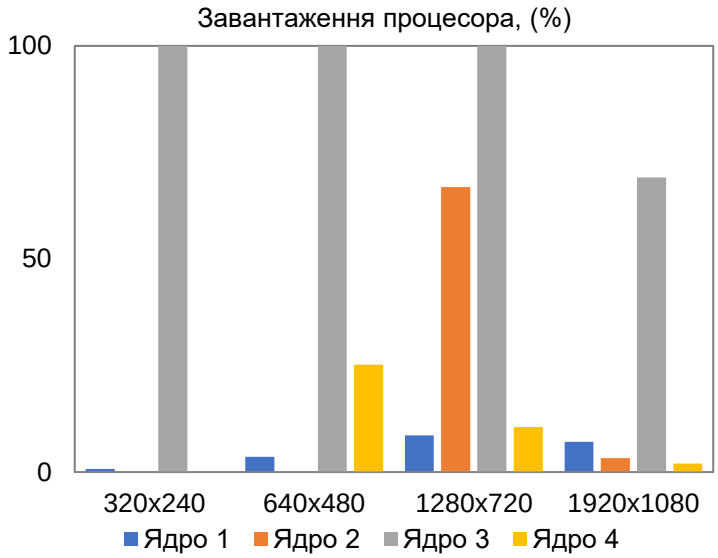


а)

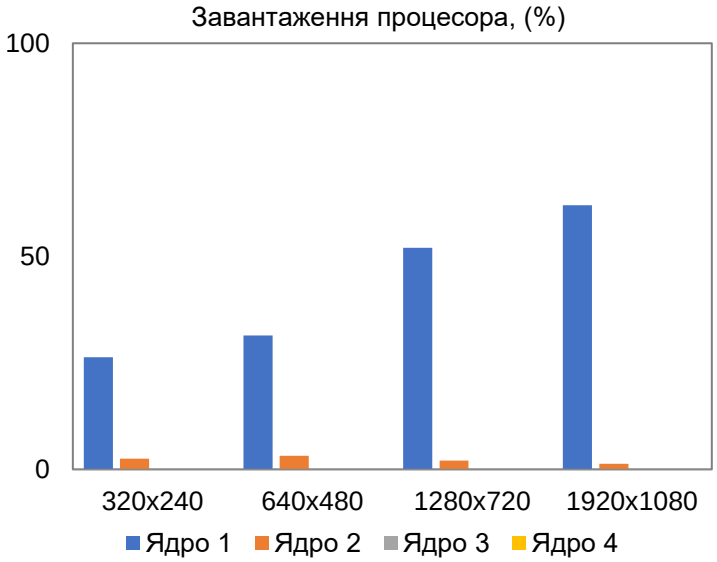


б)

Рисунок 3.1 – Усереднена затримка часу передавання кадрів різної роздільної здатності зі значенням MTU рівному 2000: а) з урахуванням процесу захоплення кадрів; б) для статичного згенерованого зображення



а)



б)

Рисунок 3.2 – Середній відсоток завантаження процесорних ядер передавача при передаванні кадрів різної роздільної здатності зі значенням MTU рівному 2000: а) з урахуванням процесу захоплення кадрів; б) для статичного згенерованого зображення

Водночас, з діаграм рис. 3.1 - а та 3.1 - в видно, що найбільший внесок у затримку вносить саме передавання кадрів. Як вдалося встановити причиною цього є занадто низьке значення параметру MTU, що може бути реалізоване на тестовій платформі – реальне максимальне підтримуване штатним драйвером MAC Raspberry Pi 4B значення MTU виявилось на рівні лише 2000 байт [17]. Дослідження впливу розміру MTU на час передавання кадрів показало, що, наприклад, при збільшенні MTU з 1400 до 2000 байт час передавання Full HD кадру (роздільною здатністю 1920×1080 пікселів) зменшується приблизно з 40 мс до 32 мс.

Щоб спрогнозувати затримки передавання при більших значеннях MTU, також було проведено експеримент із залученням персонального комп'ютера, де підтримувались так звані «jumbo-кадри» з максимальним значення MTU на рівні 9000 байт. При цьому, час передавання Full HD кадру при MTU на рівні 8192 байт становив близько 4 мс. Не дивлячись на те, що тест проводився з обміном через мережевий loopback-інтерфейс, а отже отримана оцінка є дещо оптимістичною,

Висновок про те що при збільшенні MTU підвищення швидкодії якісно підвищується, призвів до тимчасової апаратної модифікації у вигляді контролеру, що виступає мостом між апаратним USB платформи Raspberry Pi та мережею до якої дане рішення може бути під'єднано. Необхідно визначити чи підтримує даний контролер стандарт Gagbit Ethernet. Для цього необхідно його дізнатися конфігурацію. Конфігурація контролеру, що знаходиться в його в енергонезалежній пам'яті, може бути зчитана за допомогою програмного рішення – утиліти **ethtool**.

Для того щоб виконати дану операцію, визначемо назву віртуального мережевого інтерфейсу, пов'язаного за даним контролером наступним чином:

```
pi@raspberry: $ ip all
3: enx98ded01cc80c: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500
qdisc fq_codel state DOWN group default qlen 1000
```

```
    link/ether 98:de:d0:1c:c8:0c brd ff:ff:ff:ff:ff:ff
```

де **enx98ded01cc80c** – назва віртуального Ethernet контролеру. Для визначення конфігурації виконаємо наступний виклик вищезгаданої утиліти:

```
pi@raspberry: $ ethtool enx98ded01cc80c
Settings for enx98ded01cc80c:
  Supported ports: [ TP MII ]
  Supported link modes:   10baseT/Half 10baseT/Full
                          100baseT/Half 100baseT/Full
                          1000baseT/Half 1000baseT/Full
  Supported pause frame use: No
  Supports auto-negotiation: Yes
  Supported FEC modes: Not reported
  Advertised link modes:  10baseT/Half 10baseT/Full
                          100baseT/Half 100baseT/Full
                          1000baseT/Full
  Advertised pause frame use: Symmetric Receive-only
  Advertised auto-negotiation: Yes
  Advertised FEC modes: Not reported
  Speed: 10Mb/s
  Duplex: Half
  Port: MII
  PHYAD: 32
  Transceiver: internal
  Auto-negotiation: on
  Current message level: 0x00007fff (32767)
             drv probe link timer ifdown ifup rx_err tx_err
tx_queued intr tx_done rx_status pktdata hw wol
  Link detected: no
```

Жирним шрифтом відмічено, що даний контролер підтримує стандарти 1000baseT/Full.

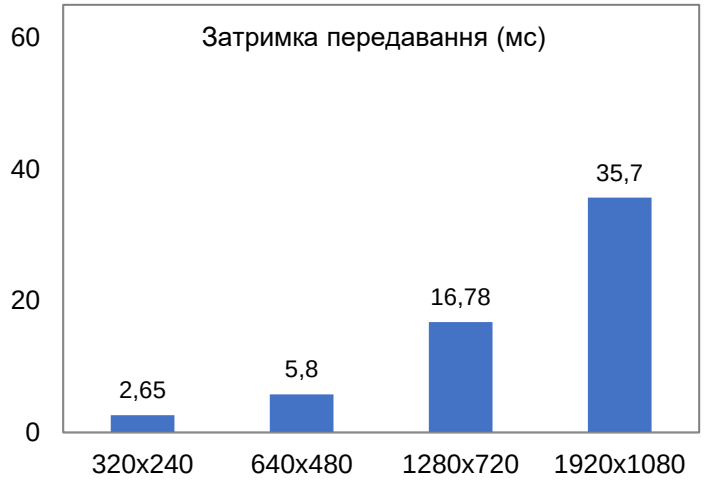
Далі, програмно спробуємо підвищити значення MTU для данного інтерфейсу:

```
pi@raspberry: $ ifconfig enx98ded01cc80c mtu 8000
```

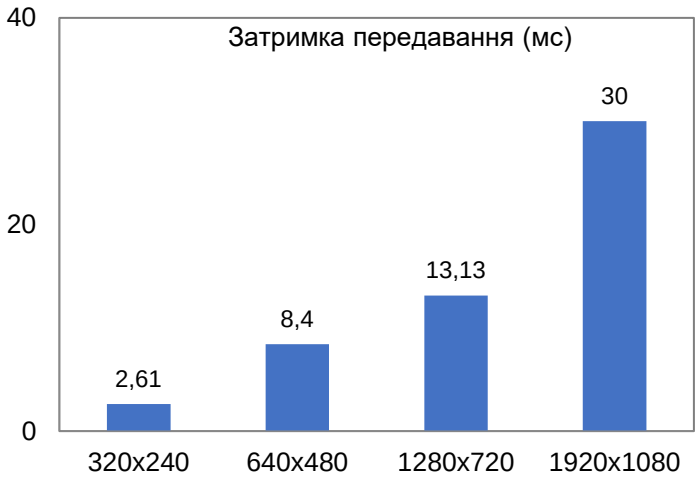
Перевіримо виконану результат виконаної операції:

```
pi@raspberry: $ ip all
3: enx98ded01cc80c: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 8000
qdisc fq_codel state DOWN group default qlen 1000
    link/ether 98:de:d0:1c:c8:0c brd ff:ff:ff:ff:ff:ff
```

За результатом, отриманим шляхом проведення описаних вище операцій, було проведено аналогічний дослід з передавання кадрів зі значенням MTU рівному 8000. Результати досліду зображено на рисунках 3.3 та 3.4.

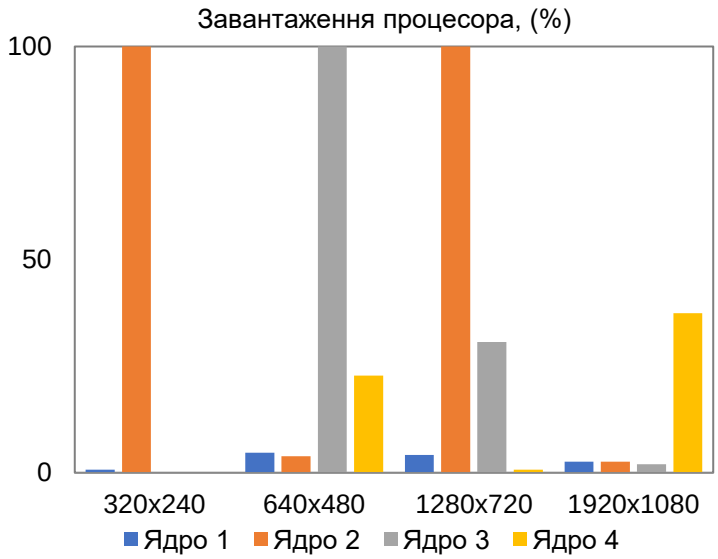


а)

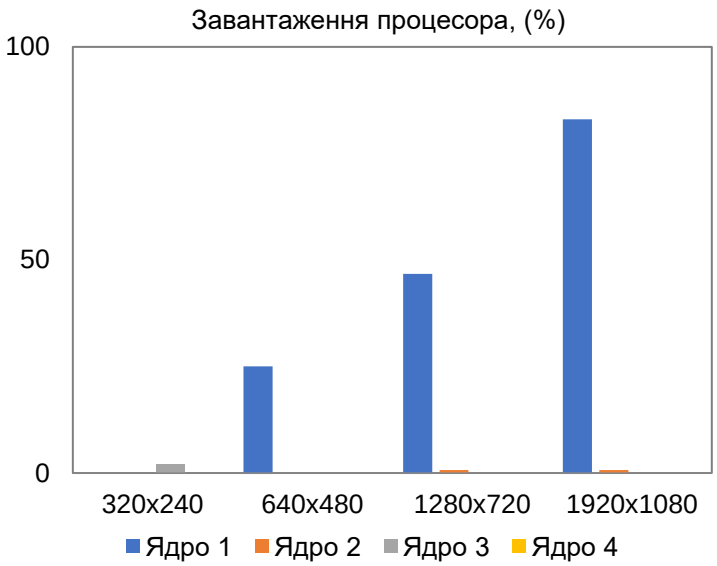


б)

Рисунок 3.3 – Усереднена затримка часу передавання кадрів різної роздільної здатності зі значенням MTU рівному 8000: а) з урахуванням процесу захоплення кадрів; б) для статичного згенерованого зображення



а)



б)

Рисунок 3.4 – Середній відсоток завантаження процесорних ядер передавача при передаванні кадрів різної роздільної здатності зі значенням MTU рівному 8000: а) з урахуванням процесу захоплення кадрів; б) для статичного згенерованого зображення

При порівнянні результатів досліду у вигляді графічних зображень, показаних на рисунках 3.3 та 3.4 видно, що при незначному зниженні

навантаженості процесорних ядер, значним є зменшення часу передавання кадру мережею. Відповідно, даний результат може бути покращено за умовою використання процесорів загального призначення, що мають апаратну можливість підвищення MTU до необхідного розміру.

Отримані результати все ж можуть вважатися досить перспективними.

3.3 Загальні рекомендації щодо реалізації GigE Vision сумісних камер з мінімальними затримками передавання відеопотоку на процесорах загального призначення

На основі проведених досліджень можна сформулювати загальні рекомендації, що будуть доцільними при реалізації GigE Vision сумісних камер на обчислювальних платформах загального призначення із забезпеченням мінімальних затримок передавання відеопотоку. Вони зокрема полягають у наступному:

1. Основним джерелом затримок є саме передавання відеопотоку через мережу.
2. Для зниження часу передавання відеопотоку мережею необхідно забезпечити максимально можливе значення MTU, бажано на рівні 8000-9000 байт. Водночас, для більшості сучасних систем-на-кристалі загального призначення таке значення не є доступним. Це підтвердив проведений під час дослідження аналіз існуючих рішень, в рамках якого було розглянуто не тільки можливості Raspberry Pi 4, на якому реалізовувався прототип камери, але й Asus Tinkerboard та інші одноплатні комп’ютери побудовані на найбільш розповсюджених доступних на сьогодні систем-на-кристалі з процесорними ядрами ARM.
3. Якщо високе значення MTU безпосередньо не доступне для деякої системи-на-кристалі, цю проблему все ще можна вирішити

використовуючи додаткові мости USB-Ethernet. Проте, таке рішення, очевидно, дещо ускладнює та здорощує процес розробки.

4. Завдяки наявності багатоядерності, а також у деяких системах-на-кристалі окремих відведених для обробки зображень модулів та співпроцесорів, процедури первинної обробки кадру (захоплення відеопотоку, перетворення кольору тощо) можна за необхідності виконувати у незалежних обчислювальних потоках. З рахунок цього додатково зменшити загальний час передавання кадрів.

Висновок до розділу 3

На базі розробленого прототипу, принцип роботи якого описано у другому розділі, проведено дослідження ефективності передачі відеопотоку шляхом вимірювання затримок передавання кожного кадру каналом Ethernet. Дослідження проводилось з використанням різних роздільних здатностей відеосенсору та статичного зображення. З метою підвищення швидкодії передачі відеопотоку, було збільшено значення MTU. Виявлено, що для процесору на базі платформи Raspberry Pi 4B дане значення не може перевищувати 2000 через внутрішні апаратні обмеження.

При дослідженні часу передавання кадру віртуальним мережевим інтерфейсом на персональному комп’ютері, визначено, що підвищення значення MTU мають суттєвий вплив на ефективність передачі відеопотоку. За цим висновком, було проведено аналогічний експеримент зі значенням MTU рівному 8000. Дані що було отримано у ході дослідження, представлено у вигляді діаграм.

РОЗДІЛ 4 РОЗРОБКА СТАРТАП ПРОЄКТУ НА ОСНОВІ МАГІСТЕРСЬКОЇ ДИСЕРТАЦІЇ

4.1 Опис ідеї проєкту

Так як програмна реалізація GigE Vision камери є апаратно-абстрагованим продуктом, компоненти захоплення та передачі відеопотоку можуть бути портованими на будь-яку платформу. Відповідно, даний продукт може бути використаний великими компаніями для побудови власних реальних продуктів з передачі відеопотоку.

Передбачається, що доступ до даного продукту може передаватися за підписанням договору про нерозголошення та придбання відповідної ліцензії. Продукт може поставлятися у вигляді API, внутрішні компоненти якого будуть закритими, у вигляді скомпільованим бібліотек.

Таблиця 4.1 – Опис ідеї стартап-проєкту

Зміст ідеї	Напрямок застосування	Вигоди для користувача
Створення програмної реалізації GigE камери	Розповсюдження програмного продукту у вигляді бінарних бібліотек під закритою ліцензією	Користувач що придбає дану ліцензію буде мати змогу використати готовий програмний продукт для запуску GigE Vision сумісного продукту. Наприклад камери для задач машинного зору.

Споживачами даного продукту може бути використано як стартапами, так и великими компаніями, що можуть придбати ліцензію на даний продукт та, за потребою, його покращувати. За результатом аналізу ринку, конкурентів не було знайдено.

Висновки: в таблиці 4.1 представлено основні напрямки використання запропонованого програмного продукту у вигляді програмної реалізації GigE Vision камери.

Конкурент 1: Allied Vision

Конкурент 2: Pleora Technologies

Конкурент 3: Omron Automation

4.2 Технологічний аудит ідеї проєкту

Таблиця 4.3 – Технологічна здійсненність ідеї проєкту

№ п/п	Ідея проєкту	Технології її реалізації	Наявність технологій	Доступність технологій
1	Захоплення відеопотоку з V4l2 сумісних камер та передача відео каналом Ethernet	Aravis Library, V4L2, Linux	Наявна	Доступна
2	Запуск продукту на одній з апаратних платформ	Raspberry Pi 4B, камера OV5647, дистрибутив Debian	Наявна	Доступна
За основу необхідно взяти перший пункт, так як його використання дозволить продукту отримати значний попит.				

4.3 Аналіз ринкових можливостей запуску стартап-проєкту

Таблиця 4.4 – Попередня характеристика потенційного ринку стартап-проєкту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних гравців, од	3
2	Загальний обсяг продаж, грн/ум.од	5000
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу	Немає
5	Специфічні вимоги до стандартизації та сертифікації	GigE Vision x. Licensing Application
6	Середня норма рентабельності в галузі (або по ринку), %	75%

Висновки: вихід на ринок є максимально рентабельним у зв'язку з відсутністю конкуренції у контексті програмних рішень, що спеціалізуються на процесорах загального призначення

Таблиця 4.5 – Характеристика потенційних клієнтів стартап-проєкту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові Сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Наявність готового рішення з захоплення та передачі відеопотоку	Промислові підприємства, стартапи, тех. гіганти, правоохоронні органи	Вимоги до різноманітності функціоналу	Вимоги до надійності програмних рішень

Висновки: формування ринку визначається потребою простоти та надійності програмного рішення. Основними споживачами продукту є усі підприємства та компанії, які мають специфічне апаратне забезпечення та необхідність передачі відео у мережу. Тому головними вимогами до товару є надійність та функціональність API.

Таблиця 4.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1	Економічний	Поява нових конкурентів	Зміцнення рекламної компанії
2	Якісний	Неналежна якість кінцевого продукту	Підвищення технічного рівня розробників
3	Технологічний розвиток	Поява нових технологій передачі відеопотоку	Розробка нової бази з освоєння даних технологій
4	Політичний	Політична ситуація країни виробника	Зміна країни виробника

Висновки: основними факторами загрози є конкуренція та економічно-політичний стан країни виробника. При цьому, технологічний розвиток методів передачі відео може здвинути знизити попит на ринці мережевих рішень. Також економічна та політична ситуація країни-виробника може зіграти значну роль у втраті прибутку.

Таблиця 4.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1	Зростання попиту	Різке збільшення Зацікавленості до продукту	Підвищення виробництва
2	Новітні технології	Можливість створення систем з підвищеною якістю мереж у повсякденному	Стандартизація продукту з метою покращення інших технологій
3	Індивідуальне замовлення	Можливість додавати індивідуальні потреби для клієнтів	Домовленність про реалізацію додаткового функціоналу
4	Кооперація з лідерами ринку	Конкуренти запропонували об’єднання компаній	Оцінка можливих переваг та ризиків об’єднання

Висновки: Ефективність комп’ютерного зору відома у всьому світі, тому засоби для реалізації даного напряму матимуть попит на десятиріччя. Тому, необхідно врахувати потенційний попит у разі успіху на даному ринці.

Таблиця 4.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Тип конкуренції: олігополія	На ринку присутня невелика кількість фірм, які займаються виробництвом такого типу систем, але у кожної є свої відмінності у характеристиках продукту.	Підвищувати якість продукту за рахунок використання передових розробок та залучення кваліфікованого персоналу
2. За рівнем конкурентної боротьби - міжнародний	Місцезнаходження фірм не обмежується територіально; лабораторії та офіси розміщено у різних країнах в залежності від поставок	Створити веб-сайт компанії, укладати договори поставок на міжнародний ринок
3. За галузевою ознакою міжгалузева	Використання у різних галузях	Проведення потужної рекламної кампанії
4.Конкуренція за видами товарів: товарно-видова	Конкуренція між товарами одного виду	Покращувати якість товару
5. За характером конкурентних переваг - цінова	Основним є ціна товару	Продаж товару по нижчій ціні

Висновки: ринок є конкурентним, міжнародним та міжгалузевим. Конкуренція за видами товарів – товарно-видова

Таблиця 4.9 – Аналіз конкуренції в галузі за М. Портером

	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складові аналізу	“Imperx”, “Pleora”, “Basler”	Logitec, Partizan	Товар продається безпосередньо розробниками	Державний та приватний сектори	IP-камери
Висновки	Конкурен-ція є низькою.	Вихід на ринок є відносно простим. Наявні потенційні конкуренти	Постальники не диктують умови роботи на ринку	Товар має бути якісним	Менша надій-ність

Для того, щоб бути конкурентоспроможним на ринку для розробки товару потрібно залучати висококваліфікованих спеціалістів.

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проєктів значущим)
мм1	Собівартість	Низька собівартість – більша доступність кінцевого пристрою
2	Якість	Краща якість даних в порівнянні з конкурентами
3	Надійність	Вища надійність процесу передачі відео
4	Адаптивність	Продукт може бути використано для будь-якої апаратної платформи

Таблиця 4.11 – Порівняльний аналіз сильних та слабких сторін проєкту

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з “ GVision”						
			–3	–2	–1	0	+1	+2	+3
1	Собівартість	19		+					
2	Якість	15					+		
3	Надійність	18			+				

Висновки: аналізуючи таблицю 4.11 можна зробити висновок, що запропонований пристрій має більший рейтинг відносно головного конкурента.

Дана таблиця демонструє основні особливості продукту, які відрізняють його від основного конкурента

Таблиця 4.12 – SWOT- аналіз стартап-проєкту

Сильні сторони: Низька ціна товару Гарна якість даних Висока надійність	*Слабкі сторони:* Високе енергоспоживання
Можливості: Вихід на міжнародний ринок Збільшення попиту	*Загрози:* Конкуренція Економічна нестабільність Політична нестабільність

Таблиця 4.13 – Альтернативи ринкового впровадження стартап-проєкту

№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Максимізація власного виграшу (індивідуалізм)	85 %	2 роки
2	Максимізація спільного виграшу (кооперація)	60%	3 роки
3	Суперництво	30%	3 роки

Висновки: було обрано кооперацію, як альтернативну ринкову поведінку, так як за відносно не високий термін існує велика ймовірність отримання ресурсів.

4.4 Розроблення ринкової стратегії проєкту

Таблиця 4.14 Вибір цільових груп потенційних споживачів

№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Тех.компанії	+	Високий	Середня	Низька
2	Стартапи	+	Високий	Висока	Середня

Таблиця 4.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проєкту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1	Індивідуалізм	Стратегія недиференційова-ного маркетингу	Адаптація до вимог ринку	Стратегія спеціалізації

Висновки: через існування на ринку більш сильних та розкручених гравців було обрано стратегію розвитку спеціалізація.

Таблиця 4.16 – Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проєкт «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристик и товару конкурента, і які?	Стратегія конкурентної поведінки*
1	Так	Шукати нових та забирати існуючих	Не буде	Стратегія виклику лідера

Висновки: оскільки на ринку вже є проєкти-конкуренти, компанія може обрати стратегію виклику лідера, так як проєкт має переваги.

Таблиця 4.17 – Визначення стратегії позиціонування

№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспромож ні позиції власного стартап-проєкту	Вибір асоціацій, які мають сформувати комплексну позицію власного проєкту (три ключових)
1	Якість даних	Стратегія спеціалізації	Задоволення індивідуальних вимог замовника	Швидкість інтеграції, підхід до кожного споживача

Висновки: підвищення якості зображення підвищує рівень довіри споживача до продукту.

4.5 Розроблення маркетингової програми стартап-проєкту

Таблиця 4.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити
1	Бібліотека для захоплення та передачі відеопотоку	Швидке прототипування, надійність у використанні	Ціна, надійність, зручність

Висновки: визначившись з основними перевагами концепції товару, буде необхідне створення рекламної кампанії з метою залучення потенційних клієнтів.

Таблиця 4.19 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>1.Товар за задумом</td><td colspan="3">Програмний продукт, що має функціонал з захвату зображення та його передачі у мережу</td></tr>
<tr><td rowspan="4">2.Товар у реальному виконанні</td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>1. Надійність
2. Ефективність
3. Зручність</td><td>М
М
М</td><td>Тх
Тх
Тх</td></tr>
<tr><td colspan="3">Якість: відповідність стандартом Ethernet</td></tr>
<tr><td colspan="3">Пакування: файли з документацією, набір бібліотек</td></tr>
<tr><td rowspan="2">3.Товар із підкріпленням</td><td colspan="3">Марка: "Xinity"</td></tr>
<tr><td colspan="3">Сертифікат відповідності</td></tr>
<tr><td colspan="4">Сервісне обслуговування</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності шляхом патентування.</td></tr>
</table>

Висновки: шляхом патентування товару створюється захист від його копіювання. Також закладені характеристики на другому рівні товару роблять його досить унікальним та затребуваним.

Таблиця 4.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1	$800-2000	$1000-2200	$100М/рік	$3000 – 5000

Висновки: обрано середню категорію цін для позиціювання як продукту належної якості

Таблиця 4.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Оформлення доступу до ліцензії шляхом придбання на сайті	Розробка, тестування, реліз	Канал нульового рівня	Виробник безпосередньо збуває продукцію покупцям

Висновки: основним каналом збуту є продаж товару. На старті компанії очікуються відносно невеликі об’єми продаж, тому на даному етапі можливо обійтись без посередників, і продавати товар напряму клієнтам. Саме тому було обрано нульовий рівень глибини каналу збуту та пряму систему збуту.

Таблиця 4.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1	Пошук нових рішень, що будуть актуальними певний період часу	Міжнародні виставки	Ознайомити можливості використання продукту	Демонстрація прототипу з викорстанням продукту	Звернення засноване на відмінностях між рекламованим товаром і тим, що пропонують конкуренти.

Висновки: маркетингова кампанія відбувається за рахунок технологічних конференцій, соціальних мереж (наприклад ділова соціальна мережа LinkedIn), реклама на сайтах партнерів.

Висновок до розділу 4

За розділом сплановано план дій щодо початку реалізації стартап проєкту. Для цього виконано розкриття зміст ідеї проєкту та можливостей щодо реального застосування продукту для отримання моделі того який ефект може бути отримано. За висновками, розглянуті ризики не повинні значно вплинути на реалізацію продукту через передбачені методи їх теоретичного подолання.

Аналіз аспектів ринку показав, що реалізація проєкту в реальних умовах є можливою, але варто враховувати, що на ринку уже існують компанії з досить високою репутацією. Даний фактор може зіграти важливу роль у просуванні проєкту. Для успіху, також необхідно провести рекламну кампанію, в якій донести до клієнта переваги використання даного продукту.

ВИСНОВКИ

За результатом дисертаційної роботи було вирішено актуальну та важливу науково-прикладну задачу з оцінки ефективності передавання відеопотоку каналом Ethernet на базі технології GigE Vision з використанням процесору загального призначення. Під час виконання дисертаційної роботи отримано наступні наукові та практичні результати:

1. Здійснено аналіз технологій передачі відеопотоку на короткі та середні дистанції. Детально розглянуто програмні компоненти стандарту GigE Vision. За патентним пошуком було здійснено огляд трьох патентів, що використовують технологію GigE Vision як основу для передачі відеопотоку між апаратними блоками, з'єднаними у мережу. Розглянуто принцип синхронізації часу за протоколом PTP.

2. Запропоновано структурно-функціональну організацію прототипу камери, що підтримує стандарт передачі відеопотоку GigE Vision каналом Ethernet. На базі даної структурно-функціональної організації та одноплатного компютера RaspberyPi розроблено діючий прототип камери. Для цього використано програмну підсистему Linux для первинного захоплення відео Video4Linux2 та бібліотеку, що реалізує підтримку технології GigE Vision, що має назву Aravis. Варто зауважити, що дане рішення може бути перенесено на будь-яку програмно-апаратну платформу з операційною системою Linux та потрібними блоками периферійних пристроїв.

3. На основі протоколу PTP розроблено методику оцінювання затримки часу передавання GigE Vision кадрів, що передбачає додавання у кадри окремих часових міток. Користуючись розробленою методикою проведено дослідження щодо визначенням затримки передачі відеопотоку каналом Ethernet за умови, що час на обох сторонах синхронізовано за допомогою протоколу PTP.

4. За результатами дослідження встановлено, що програмна реалізація GigEVision сумісної камери на одноплатних комп’ютерах є доцільною та в цілому може вважатися перспективною. Без додаткових оптимізацій така реалізація дозволяє досягти більш-менш прийнятних результатів затримки для відносно невеликих кадрів (з роздільною здатністю до 640×480 пікселів), де досягається затримка до 10 мс. Встановлено, що найбільш вагомим джерелом затримок є низьке підтримуване значення MTU, тому для кадрів більшого розміру можна рекомендувати вибір обчислювальної платформи, на якій би підтримувалось достатньо велике (рекомендовано до 9000 байт) значення MTU. Крім того, помітний внесок в сумарну затримку передавання може давати процедура перетворення формату зображень.

З огляду на це, в рамках подальшої оптимізації, ця процедура може бути перенесена на бік приймача, при цьому передавач буде транслювати вихідні зображення (у форматі raw) або, якщо це дозволяє архітектура платформи, що лежить в основі передавача, перетворення формату може бути розпаралелене і виконуватись на вільних процесорних ядрах або у окремо відведених блоках DSP обробки.

За темою дисертаційної роботи було розроблено стартап-проєкт з метою комерціалізації ідеї пов’язаної з дешевизною рішення на базі «дешевого» рішення на базі процесорів загального призначення у порівнянні з рішеннями на базі FPGA.

ПЕРЕЛІК ПОСИЛАНЬ

1. GigE Vision: standard route to video over IP [Електронний ресурс] // G. Chamberlain. – 2005. – Режим доступу до ресурсу: https://iebmedia.com/technology/gige-vision-standard-route-to-video-over-ip.
2. Guide to Understanding Machine Vision Standards [Електронний ресурс] // AIA, EMVA, JIIA, VDMA and CMVU. – 2018. – Режим доступу до ресурсу: https://www.emva.org/wp-content/uploads/FSF-VS-Brochure-2018-A4-full.pdf.
3. Video Streaming and Device Control Over Ethernet Standard, version 2.0 [Електронний ресурс] // ASSOCIATION FOR ADVANCING AUTOMATION. – 2013. – Режим доступу до ресурсу: https://www.visiononline.org/userAssets/aiaUploads/File/GigE_Vision_Specification_2-0-03.pdf.
4. 4. Precision Time Protocol (PTP/IEEE-1588). // White paper. – 2018. – №130711.
5. A high resolution smart camera with GigE Vision extension for surveillance applications / E.Norouznezhad, A. Bigdeli, A. Postula, B. C. Lovell. // Second ACM/IEEE International Conference on Distributed Smart Cameras. – 2008. – №10. – С. 1–8.
6. A resource-efficient multi-camera GigE vision IP core for embedded vision processing platforms / [O. W. Ibraheem, A. Irwansyah, J. Hagemeyer та ін.]. // International Conference on ReConFigurable Computing and FPGAs (ReConFig). – 2015. – №10. – С. 1–6.
7. Marchenko V. Programno-aparatna realizatsiya videocamery, sumisnoyi zi standartom GigE Vision / V. Marchenko, T. Khodniev, A. Varfolomieiev. // Microsystems, Electronics and Acoustics. – 2018. – №10. – С. 32–37.

8. Accelerated MIPI CSI video stream acquision in tasks of real-time video streaming / [T. A. Khodniev, M. S. Holub, O. V. Kuzhylnyi та ін.]. // Visnyk NTUU KPI Seriia - Radiotekhnika Radioaparatobuduvannia. – 2020. – №82. – С. 35–43.
9. Kernel Documentation [Електронний ресурс] // Block. – 2021. – Режим доступу до ресурсу: https://www.kernel.org/doc/html/latest/block/index.html.
10. Video for Linux API capture example [Електронний ресурс] // The Linux Kernel. – 2021. – Режим доступу до ресурсу: https://www.kernel.org/doc/html/v4.12/media/uapi/v4l/capture.c.html.
11. The Video Processing Front End [Електронний ресурс] // Texas Instruments. – 2021. – Режим доступу до ресурсу: https://software-dl.ti.com/processor-sdk-linux/esd/docs/06_03_00_106/AM437X/linux/Foundational_Components/Kernel/Kernel_Drivers/Camera/VPFE.html.
12. “v4l2-capture”, Jayawardenepura [Електронний ресурс] // github. – 2021. – Режим доступу до ресурсу: https://github.com/Jayawardenepura/v4l2-capture.
13. Corbet J. On the management of the Video4Linux subsystem tree [Електронний ресурс] / Jonathan Corbet // lwn.net. – 2009. – Режим доступу до ресурсу: https://lwn.net/Articles/320472/.
14. Pacaud E. Aravis – A vision library for genicam based cameras [Електронний ресурс] / Pacaud // Gnome. – 2021. – Режим доступу до ресурсу: https://wiki.gnome.org/Projects/Aravis.
15. “Aravis” fork, Jayawardenepura [Електронний ресурс] // github. – 2021. – Режим доступу до ресурсу: https://github.com/Jayawardenepura/aravis.
16. PTPd [Електронний ресурс] // wikipedia. – 2021. – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/PTPd.
17. Kernel build with Jumbo Frames Support [Електронний ресурс] //

github. – 2021. – Режим доступу до ресурсу: https://github.com/johnduffymsc/picluster/wiki/KernelBuildWIthJumboFramesSupport.

18. Media subsystem kernel internal API [Електронний ресурс] // The Linux Kernel. – 2016. – Режим доступу до ресурсу: https://www.kernel.org/doc/html/v4.12/media/media_kapi.html.
19. Understanding and Applying Precision Time Protocol / S. T.Watt, S. Achanta, H. Abubakari, E. Sagen. // Saudi Arabia Smart Grid (SASG). – 2015.
20. Kovácsházy T. Hardware assisted PTPd home page [Електронний ресурс] / Kovácsházy. – 2010. – Режим доступу до ресурсу: http://home.mit.bme.hu/~khazy/ptpd/.
21. GigE Vision for Real-Time [Електронний ресурс] // DALSA Corporation. – 2010. – Режим доступу до ресурсу: https://nstx.pppl.gov/nstxhome/DragNDrop/Operations/Diagnostics_&_Support_Sys/D1CCD/GigE_Vision_for_Realtime_MV_11052010.pdf.
22. DALSA GENIE HM1400 Camera [Електронний ресурс] – Режим доступу до ресурсу: https://www.dalsa-camera-distributor.com/products/genie-hm1400-sv.
23. RockChip ISP [Електронний ресурс] // Rockchip Wiki. – 2020. – Режим доступу до ресурсу: http://opensource.rock-chips.com/wiki_Rockchip-isp1 .

ДОДАТОК А. Копія першої публікації за темою дисертаційної роботи

УДК 004.773

DOI: 10.20535/2523-4455.mea.n

Засоби та методики оцінки ефективності передачі відеопотоку на основі технології GigE Vision з використанням процесору загального призначення

Кужильний[f] О. В., ORCID 0000-0003-2681-5569
Варфоломєєв[s] А. Ю., к.т.н., ORCID 0000-0002-6990-7140
Ходнєв[s] Т. А., ORCID 0000-0001-9168-0504

Кафедра конструювання електронно-обчислювальної апаратури, Факультет електроніки
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Київ, Україна

***Анотація*—У роботі досліджено ефективність реалізації GigE Vision сумісного джерела відеопотоку на обчислювальній платформі, основаній на ARM процесорі загального призначення. Зокрема, для реалізації джерела створено прототип GigE Vision сумісної камери з використанням порівняно розповсюдженого одноплатного комп'ютера Raspberry Pi 4. З використанням програмного інтерфейсу Video4Linux2 розроблено програмну реалізацію процедури захоплення зображень із відеосенсора, підключеного до одноплатного комп'ютера та за допомогою бібліотеки Aravis створено процедуру конвертування і передавання мережею захоплених кадрів у сумісному з технологією GigE Vision форматі. Запропоновано метод вимірювання затримок передачі кадрів каналом Ethernet та проведено відповідні вимірювання. Встановлено, що програмна реалізація GigE Vision сумісної відеокамери на сучасних одноплатних комп'ютерах може вважати перспективною, в особливості, за подальшого вдосконалення шляхом оптимізації відповідних програмних та/або апаратних складових.**

Ключові слова — відеопотік; затримка передачі; синхронізація часу; Ethernet; GigE Vision

I. Вступ

З метою створення продуктів, які повинні відрізняються гнучкістю та невисокою вартістю, індустрія комп'ютерного зору роками розробляє високоефективні програмні рішення на основі засобів передачі зображень на короткі та середні відстані, таких як Camera Link та LVDS. З цим пов'язаний певний інтерес в галузі машинного зору щодо розробки нових програм з використанням загальнорозповсюджених комунікаційних з'єднань Gigabit-Ethernet [1, 2].

GigE Vision – це технологія передачі відеопотоку на основі Ethernet, зокрема, для задач машинного зору. Дана технологія є відносно економічно-вигідною у використанні, працює на транспортному рівні моделі OSI та дозволяє передавати дані на великі відстані за допомогою стандартизованого кабелю CAT-5e/6 або будь-якого іншого середовища передачі, що підтримується стандартом Ethernet. Як наслідок, GigE Vision підтримує цілий спектр відповідних швидкостей передачі Ethernet каналу, забезпечуючи достатньо високу пропускну здатність, необхідну для потокової передачі зображень з відеосенсорів у реальному часі.

За стандартом GigE Vision версії 2.0, транспортним шаром є протокол прикладного рівня, що має назву GVSP (GigE Vision Streaming Protocol). Для доставки відеоданих GVSP використовує стандартний протокол нижнього, транспортного рівня UDP. Це дозволяє передавати нестиснутий або стиснутий (JPEG, H.264) відеопотік, інформацію про зображення, або інші дані з мінімальними затримками та накладними витратами засобами мережі. Варто зауважити, що взаємодія між передавачем та приймачем будується за асиметричним принципом – основний потік даних надсилається передавачем, тоді як приймач надсилає лише керуючу інформацію. Оскільки протокол UDP не передбачає підтвердження та не гарантує доставки даних, цілісність передачі пакетів, GVSP має контролювати приймач, який у разі їх пошкодження чи відсутності робить запит у передавача про повторне надсилання втрачених даних. Цей запит здійснюється через інший протокол – GVCP (GigE Vision Control Protocol). Через GVCP також здійснюється загальне керування GigE Vision камерою, пошук камер у мережі та встановлюється початкове з'єднання.

Питання ефективності передачі відеопотоку каналом Ethernet з використанням технології GigE

DOI: 10.20535/2523-4455.mea.n

Vision було розглянуто у статтях [3, 4, 5, 6]. Водночас, в усіх роботах досліджувались особливості реалізації GigE Vision сумісних камер на базі технології FPGA. На підставі аналізу результатів минулих досліджень виникла ідея заміни FPGA процесором загального призначення при розгортанні GigE Vision технології. Водночас, у дослідженні [1] стверджується, що така заміна не є надто доцільною з огляду на низьку енергоефективність процесорів загального призначення, однак зважаючи на час, що пройшов з моменту публікації [1] (приблизно 15 років) ситуація суттєво змінилась. Сучасні системи на кристалі та вбудовані процесори загального призначення стали значно продуктивнішими, енергоефективнішими та мають широкий набір периферії, в тому числі для безпосереднього підключення відеосенсорів та Ethernet контролерів. Враховуючи це, а також нижчу вартість в порівнянні з FPGA, постає питання щодо їх використання як більш дешевих альтернативних рішень для реалізації GigE Vision відеокамер. Цікаво відмітити, що на момент написання даної статті публікацій, окрім роботи [1] на тему програмної реалізації GigE Vision камер з використанням одноплатних комп'ютерів знайдено не було. Причиною цього є імовірно той факт, що на момент появи стандарту GigE Vision на початку 2000 років, використання одноплатних комп'ютерів із процесорами загального призначення не було доцільним і з того моменту реалізацію камер на їх основі всерйоз ніхто не розглядав. З огляду на зазначене *метою* даної статті є дослідження ефективності передачі відеопотоку каналом Ethernet на основі технології GigE Vision з використанням саме процесорів загального призначення. При цьому передбачається вирішення задач оцінювання параметрів таких як рівень затримки між передавачем та приймачем, а також рівня навантаження на мережу та процесорні ядра.

II. Реалізація прототипу GigE Vision сумісної камери на основі системи на кристалі з процесорними ядрами загального призначення

Для оцінювання ефективності передачі відеопотоку системою на основі вбудованого процесора загального призначення було розроблено власний прототип GigE Vision сумісної камери. Для цього обрано апаратну платформу, що задовольняє критеріям порівняно низької собівартості, відносно низької споживаної потужності, наявності необхідних периферійних пристроїв та розроблено програмну підтримку первинного захоплення відеопотоку і його подальшого транслювання у Ethernet мережу.

Зокрема, прототип GigE Vision камери вирішено створювати на базі апаратної платформи Raspberry Pi 4B та CSI сумісного відеосенсору OV5647. Платформа Raspberry Pi 4B була обрана за сукупністю властивостей, зокрема через її доступність, наявність широких обчислювальних можливостей завдяки обчислювально потужному 4-х ядерному процесору BCM2711 на основі ядер Cortex-A72, що працюють на частоті 1,5 ГГц, наявність необхідної периферії: Ethernet контролера

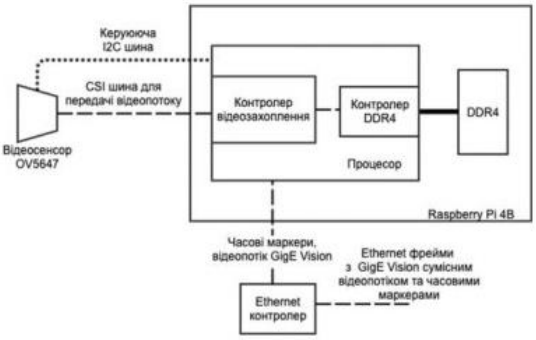


Рис. 1 Структурна організація апаратної частини створюваного прототипу GigE Vision сумісної камери.

з підтримкою стандарту 1000Base-T Gigabit Ethernet, контролера відеозахоплення для отримання зображень з відеосенсору, а також, що важливо, базового пакету програмної підтримки, необхідного для швидкого прототипування.

З огляду на те, що реалізація GigE Vision сумісної камери можлива також і на інших обчислювальних платформах (на основі інших систем на кристалі) щоб за необхідності її можна було легко перенести на ці платформи, прототип створювався з якомога більш апаратно-незалежною структурною організацією, показаною на рис. 1.

III. Реалізація програмної частини дослідження

Програмна частини GigE Vision сумісного прототипу камери складається з двох структурних компонентів: драйверів захоплення відеопотоку та компонента, який конвертує отримані зображення в кадри Ethernet для транслювання у мережу.

A. Підсистема захоплення відеопотоку

З точки зору апаратної складової, обрана система на кристалі має вбудований блок відеозахоплення (BCM2711 VideoCore співроцесор), що виконує отримання кадрів від сенсора та подальше збереження до зовнішньої оперативної пам'яті. Завдяки цьому забезпечується програмна абстракція функціоналу збереження у пам'яті зображень, які далі уніфікованим чином можуть бути передані користувацьким сервісам.

Для отримання кадрів використано відповідну V4L2 (Video4Linux2) підсистему драйверів ядра Linux. З огляду на те, що дана підсистема працює у просторі ядра, V4L2 надає користувацький інтерфейс керування у вигляді виклику системної функції `ioctl()`. Даний виклик виконується до відеопристрою, що зазвичай ініціалізується у системі як файл з директорії `/dev` з іменем `videoX`, де `X` – номер зареєстрованого в системі пристрою відеозахоплення. Процедура отримання кадру із пристрою джерела відеопотоку детально описана у відповідній документації Linux [3, 4].

Використовуючи підсистему Video4Linux, було розроблено власну процедуру відеозахоплення, що

DOI: 10.20535/2523-4455.mea.n

працює у користувацькому просторі та передає захоплені кадри відеопотоку до підсистеми, що реалізує мережеву складову створюваного прототипу камери. Оскільки, через помилки передачі або дію зовнішніх факторів, потік даних може передаватись мережею з неоднорідним темпом. При цьому, зображення перед відправленням зберігається до кільцевого буферу, кожен елемент якого містить окремий кадр. Розмір буферу може задаватися користувачем як параметр на етапі конфігурації. Організація первинного захоплення кадрів з підсистеми Video4Linux є наступною: підсистема зберігає кадри з простору пам'яті ядра у елементи буферу користувацької пам'яті, інформує про готовність кадру, передає вказівник на початок кадру та його розмір. У результаті, вказівник на буфер та його розмір може бути передано до користувацьких сервісів (в тому числі мережевої підсистеми), які, у свою чергу, матимуть змогу скопіювати вказану кількість байтів, з яких складається зображення.

Підсистема V4L виступає як клас драйверів, що уніфікують механізм для простору користувача з метою взаємодії з відеосенсорами різних виробників за допомогою відповідних апаратно-залежних драйверів. Таким чином, один V4L драйвер може бути використаний для взаємодії з багатьма камерами, причому на різних вбудованих апаратних платформах.

B. Мережева підсистема, що реалізує програмний компонент GigE Vision

Бібліотека Aravis є однією, якщо не єдиною відкритою та найбільш повноцінною бібліотекою для роботи з GigE Vision сумісними пристроями [7]. Aravis основана на низькорівневих бібліотеках GLib/GObject та розповсюджується за вільною ліцензією LGPL v2+. Виходячи з цього, було прийнято рішення дослідити дану бібліотеку на предмет її використання як програмного компоненту, що працює у користувацькому просторі Linux та реалізує мережеву складову прототипу камери, що розробляється.

Ідея використання бібліотеки Aravis полягає у тому, що в її складі реалізовано емулятор GigE Vision камери. Підмінивши в даному емуляторі синтетичне зображення на реальне дозволить створити власну GigE Vision сумісну камеру. Така підміна може бути тривіально здійснена шляхом передачі вказівника на кадри, які зберігаються у відеобуфері, отримувані підсистемою відеозахоплення, що була описана у попередньому підрозділі.

Зокрема, створення віртуальної GigE Vision камери, що здатна транслювати реальний відеопотік потребує внесення незначних змін у модулі `arvfakecamera.c` бібліотеки Aravis, що включають:

1) ініціалізацію процесу захоплення кадрів – запуск підсистеми відеозахоплення, що заповнюватиме циклічний відеобуфер;
2) вказати функцію, зворотного виклику, що має викликатись при запиті на передавання чергового кадру, для чого присвоїти вказівник на функцію у поле:
   `fake_camera->priv->fill_callback`;
3) реалізувати механізм відображення відеобуферу, тобто надати вказівник на початок кадру, що має бути переданий, шляхом вибору коректного кадру із циклічного буферу та запису його адреси у поле:
   `arv_buffer->priv->data`.

При цьому, бібліотека Aravis бере на себе задачу конвертування зображень у Ethernet фрейми та їх передачу через мережевий сокет з використанням стеку TCP/IP, що реалізується підсистемою ядра Linux.

Структурна організація програмної частини створюваного прототипу камери показана на рис. 2. В наведеній структурній організації підсистема

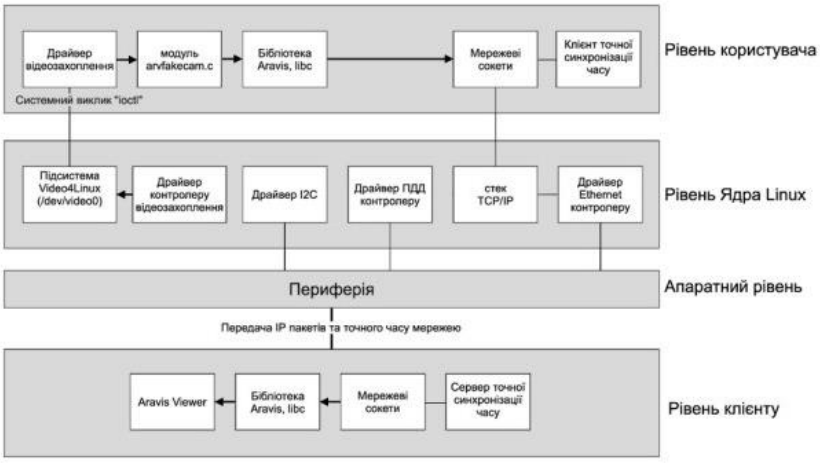


Рис. 2 Структурна організація програмної частини створюваного прототипу GigE Vision сумісної камери. Хоча обмін даними між більшістю компонентів системи є двостороннім, стрілки вказують на напрямок основного потоку інформації

DOI: 10.20535/2523-4455.mea.n

захоплення відеопотоку реалізується блоком «Драйвер відеозахоплення», а мережева підсистема – модифікованим модулем arvfakecam.c. Рис. 2 також ілюструє як наведені підсистеми взаємодіють з бібліотеками та компонентами операційної системи Linux.

IV. Дослідження ефективності передавання даних створеним прототипом камери

Дослідження ефективності формування та передавання GigE Vision відеопотоку створеним прототипом включає два аспекти. Перший полягає у оцінюванні затримки передавання кадрів відеопотоку, а другий – у визначенні навантаження на обчислювальну систему, зокрема, процесор, що лежить в основі створюваного прототипу.

A. Методика оцінювання затримок передавання кадрів у дослідженні

Оцінювання затримок передавання кадрів можна здійснити за часовими мітками формування кадру на стороні передавача та часу його отримання приймачем достатньо простим способом:

$$\tau = t_r - t_t,$$

де τ – затримка передавання; t_r – час отримання кадру приймачем; t_t – час початку процедури передавання кадру на стороні передавача.

Не дивлячись на простоту реалізації такої оцінки, з нею пов'язана порівняно суттєва проблема – для того, щоб отримати коректну затримку із різниці часових міток, час на приймачі та передавачі має йти синхронно.

Доречно зазначити, що данапроблема має потенційне рішення при застосуванні протоколу PTP (Precision Time Protocol), визначеного стандартом IEEE-1588. Суть цього протоколу полягає у тому, що вузли, на яких необхідно синхронізувати час, постійно обмінюються сповіщеннями, в яких передається час ведучого вузла (сервера часу), а також здійснюється корегування на величину часу, витраченого на проходження через мережеве обладнання [2]. Ця схема підвищує точність видачі часу споживачам, компенсуючи затримки доставки повідомлень мережею. Стверджується, що для апаратної реалізації протоколу, синхронізація можлива з точністю до 1 мікросекунди, тоді як для програмної реалізації PTP, точність зазвичай не перевищує 100 мікросекунд [3, 4]. Оскільки час формування зображення на відеосенсорі з огляду на реальні величини експозиції перевищують 100 мкс в рази, якщо не на порядки, точності, що забезпечується навіть програмною реалізацією PTP протоколу для оцінювання затримок можна вважати достатньою.

Зауважимо також, що оскільки дослідження проводиться для з'єднання точка-точка, у мережі наявні тільки два пристрої, тому загалом не становить суттєвої різниці, який саме з них буде джерелом точного часу, а який буде його відповідним споживачем. Водночас, в рамках даної роботи сервер точного часу працює на стороні приймача (клієнта).

З практичної точки зору функціонування PTP протоколу в операційній системі GNU/Linux забезпечується програмою ptpd. Її виклик з відповідними аргументами може здійснюватись, наприклад, наступним чином:

- для серверу точного часу:

```
ptpd -C --p2p -i eth0 -M
```

- для клієнту, що має синхронізувати свій час:

```
ptpd -C --p2p -i eth0 -s
```

де *eth0* – системне ім'я використовуваного мережевого інтерфейсу Ethernet. Синхронізація часу врахована у структурі створюваного прототипу камери відповідними блоками (див. рис. 2).

B. Результати досліджень

Оцінювання часу затримки передавання кадрів здійснено з використанням методики описаної в попередньому підрозділі для різних роздільних здатностей зображень та показано на рис. 3. На рис. 4 показано відсоток завантаження процесорних ядер передавача (створюваного прототипу камери) при передаванні кадрів з відповідною роздільною здатністю.

З метою виключення впливу на час передавання процедури первинного захоплення кадрів, було проведено додатковий експеримент, в якому здійснювалось надсилання одного незмінного кадру, причому підсистема відеозахоплення на основі V4L

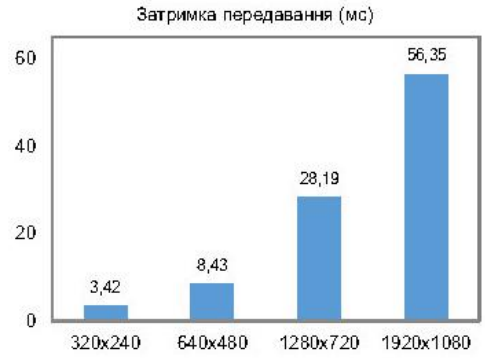


Рис. 3 Усереднена затримка часу передавання кадрів різної роздільної здатності

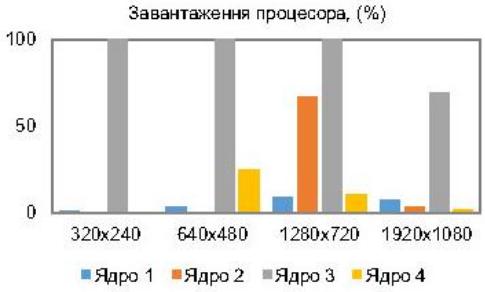


Рис. 4 Середній відсоток завантаження процесорних ядер передавача при передаванні кадрів різної роздільної здатності

DOI: 10.20535/2523-4455.mea.n

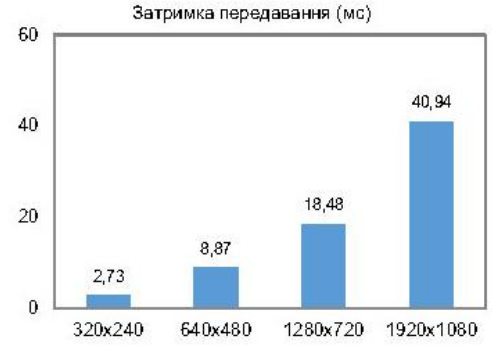


Рис. 5 Усереднена затримка часу передавання кадрів різної роздільної здатності (без виконання процедури захоплення кадрів)

Рис. 6 Середній відсоток завантаження процесорних ядер передавача при передаванні кадрів різної роздільної здатності (без виконання процедури захоплення кадрів)

залишалася відключеною. Результати експерименту показано на рис. 5 та рис. 6, де наведено затримку передавання та відсоток завантаження ядер процесора відповідно.

Порівнюючи діаграми рис. 3 та 5 можна бачити, що час отримання кадрів з підсистемою V4L2 є досить високим та становить близько 30% від загального часу формування та передавання кадрів. Більше того, згідно графіків завантаження процесорних ядер (рис. 4 та 6) близько 50% відсотків припадає на первинне захоплення кадрів. Причину цього можна пояснити тим, що в експерименті для зручності подальшого відображення кадрів на стороні передавача здійснювалось перетворення кадрів у формат YUV. У випадку, якщо це перетворення не виконувати, як це зазвичай відбувається в реальних камерах і передавати від сенсора вихідні (raw) дані минаючи процес перетворення, зазначені затримки можна помітно знизити. З іншого боку з рис. 4 та 6 видно, що постійно завантажується тільки одне ядро процесора, тому навіть за потреби виконання перетворень на тестовій платформі є резерв обчислювальних можливостей, які можуть бути використані шляхом розпаралелювання обчислень.

Водночас, з діаграм рис. 3 та 5 видно, що найбільший внесок у затримку вносить саме передавання кадрів. Як вдалося встановити причиною цього є занадто низьке значення MTU (Maximum Transmission Unit), що може бути реалізоване на тестовій платформі – реальне максимальне підтримуване штатним драйвером MAC Raspberry Pi 4 значення MTU виявилось на рівні лише 2000 байт. Дослідження впливу розміру MTU на час передавання кадрів показало, що, наприклад, при збільшенні MTU з 1400 до 2000 байт час передавання Full HD кадру (роздільною здатністю 1920×1080 пікселів) зменшується приблизно з 40 мс до 32 мс. Щоб спрогнозувати затримки передавання при більших значеннях MTU також було проведено експеримент із залученням персонального комп'ютера, де підтримувались так звані «jumbo-кадри» з максимальним значення MTU на рівні 9000 байт. При цьому, час передавання Full HD кадру при MTU на рівні 8192 байт становив близько 4 мс (тест проводився з обміном через мережевий loopback-інтерфейс, а отже отримана оцінка є дещо оптимістичною). Порівняти зазначені затримки із відповідними затримками для промислових зразків камер складно, оскільки такий параметр у специфікаціях зазвичай не вказують. Водночас, у звіті [13] побіжно згадується, що для серійного зразка камери DALSA Genie HM1400 [14] при налаштуваннях витримки у 100 мкс та передаванні кадрів роздільною здатністю 1400×1024 пікселів на частоті 64 кадрів/с, час отримання кадру (camera readout time) становить близько 15,6 мс. Перераховуючи затримку для створеної програмної реалізації камери пропорційно до роздільної здатності промислового зразка, отримаємо значення затримки на рівні 29 мс, що перевищує час отримання кадру для DALSA Genie HM1400 майже в 2 рази. Не зважаючи на це, на думку авторів створення на одноплатному комп'ютері програмної реалізації камери все ж може вважатись перспективним, оскільки:

- існує можливість подальшої оптимізації по досягненню меншого часу передавання (за рахунок збільшення MTU);
- за умови, якщо не вдасться досягти часу затримки на рівні промислових зразків, то отримана реалізація буде принаймні дешевшою, так як не використовуватиме дороговартісну елементну базу на основі FPGA та все ж зможе знайти застосування у менш чутливих до затримок сферах використання;
- враховуючи, що затримка у 10 мс може вважатись прийнятною для більшості систем комп'ютерного зору [13], при невеликих роздільних здатностях (близько 640×480 пікселів) навіть не оптимізована програмна реалізація на одноплатному комп'ютері цілком здатна забезпечити необхідні вимоги (див. рис. 3 та 5).

Висновки

Виходячи з результатів дослідження, можна зробити висновок, що програмна реалізація GigE Vision сумісної камери на одноплатних комп'ютерах

DOI: 10.20535/2523-4455.mea.n

є доцільною та може вважатися перспективною. Без додаткових оптимізацій така реалізація дозволяє досягти більш-менш прийнятних результатів затримки для відносно невеликих кадрів (з роздільною здатністю до 640×480 пікселів), де досягається затримка до 10 мс. Встановлено, що найбільш вагомим джерелом затримок є низьке підтримуване значення MTU, тому для кадрів більшого розміру можна рекомендувати вибір обчислювальної платформи, на якій би підтримувалось достатньо велике (рекомендовано до 9000 байт) значення MTU. Крім того, помітний внесок в сумарну затримку передавання може давати процедура перетворення формату зображень. З огляду на це, в рамках подальшої оптимізації, ця процедура може бути перенесена на бік приймача, при цьому передавач буде транслювати вихідні зображення (у форматі raw) або, якщо це дозволяє архітектура платформи, що лежить в основі передавача, перетворення формату може бути розпаралелене і виконуватись на вільних процесорних ядрах.

ПЕРЕЛІК ПОСИЛАНЬ

[1] G. Chamberlain, "GigE Vision: standard route to video over IP," 2005. [Online]. Available: https://iebmedia.com/technology/gige-vision-standard-route-to-video-over-ip. [Accessed 25 Oct. 2021].

[2] AIA, EMVA, JIA, VDMA and CMVU, "Guide to Understanding Machine Vision Standards," 2018. [Online]. Available: https://www.emva.org/wp-content/uploads/FSF-VS-Brochure-2018-A4-full.pdf. [Accessed 27 Oct. 2021].

[3] E. Norouznezhad, A. Bigdeli, A. Postula and B. C. Lovell, "A high resolution smart camera with GigE Vision extension for surveillance applications," *2008 Second ACM/IEEE International Conference on Distributed Smart Cameras*, pp. 1-8, 2008. **DOI:** 10.1109/ICDSC.2008.4635711.

[4] O. W. Ibraheem, A. Irwansyah, J. Hagemeyer, M. Porrmann and U. Rueckert, "A resource-efficient multi-camera GigE vision IP core for embedded vision processing platforms," *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1-6, 2015. **DOI:** 10.1109/ReConFig.2015.7393282.

[5] V. Marchenko, T. Khodniev and A. Varfolomieiev, "Programno-aparatna realizatsiya videocamery, sumisnoyi zi standartom GigE Vision," *Microsystems, Electronics and Acoustics*, vol. 23, no. 5, pp. 32-37, 2018. **DOI:** 10.20535/2523-4455.2018.23.5.147686.

[6] T. A. Khodniev, M. S. Holub, O. V. Kuzhylnyi, O. M. Lysenko and A. Y. Varfolomieiev, "Accelerated MIPI CSI video stream acquision in tasks of real-time video streaming," *Visnyk NTUU KPI Seriia - Radiotekhnika Radioaparatobuduvannia*, no. 82, pp. 35-43, 2020. **DOI:** 10.20535/RADAP.2020.82.35-43.

[7] "Media subsystem kernel internal API," [Online]. Available: https://www.kernel.org/doc/html/v4.12/media/media_kapi.html. [Accessed 27 Oct. 2021].

[8] "Video for Linux API capture example," [Online]. Available: https://www.kernel.org/doc/html/v4.12/media/uapi/v4l/capture.c.html. [Accessed 27 Oct. 2021].

[9] E. Pacaud, "Aravis – A vision library for genicam based cameras," [Online]. Available: https://wiki.gnome.org/Projects/Aravis. [Accessed 27 Oct. 2021].

[10] S. T. Watt, S. Achanta, H. Abubakari and E. Sagen, "Understanding and Applying Precision Time Protocol," in *Saudi Arabia Smart Grid (SASG)*, Jeddah, 2015. **DOI:** 10.1109/SASG.2015.7449285.

[11] "PTPd," [Online]. Available: https://en.wikipedia.org/wiki/PTPd. [Accessed 27 Oct. 2021].

[12] T. Kovácsházy, "Hardware assisted PTPd home page," 2010. [Online]. Available: http://home.mit.bme.hu/~khazy/ptpd/. [Accessed 27 Oct. 2021].

[13] DALSA Corporation, "GigE Vision for Real-Time," 2010. [Online]. Available: https://nstx.pppl.gov/nstxhome/DragNDrop/Operations/Diagnostics_&_Support_Sys/D1CCD/GigE_Vision_for_Realtime_MV_11052010.pdf. [Accessed 01 Dec. 2021].

[14] "DALSA GENIE HM1400 Camera," [Online]. Available: https://www.dalsa-camera-distributor.com/products/genie-hm1400-sv. [Accessed 01 Dec. 2021].

UDC 004.773

Means and methods of efficiency estimation of video stream transmission based on GigE Vision technology using application processor

O.V. Kuzhylnyi[f], ORCID 0000-0003-2681-5569
A.Y. Varfolomieiev[s], PhD, ORCID 0000-0002-6990-7140
T.A. Khodniev[s], ORCID 0000-0001-9168-0504
Department of Design of Electronic Computing Equipment, Faculty of Electronics
National technical university of Ukraine
"Igor Sikorsky Kyiv polytechnic institute"
Kyiv, Ukraine

DOI: 10.20535/2523-4455.mea.n

Abstract—**The paper investigates the possibility of efficient implementation of a GigE Vision compatible video stream source on a computing platform based on a system-on-a-chip with general-purpose ARM processor cores. In particular, to implement the aforementioned video source, a proprietary prototype of a GigE Vision compatible camera was developed based on the Raspberry Pi 4 single-board computer. This computing platform was chosen due to its widespread use and wide community support. The software part of the camera is implemented using the Video4Linux and Aravis libraries. The first library is used for the primary image capturing from a video sensor connected to a single board computer. The second library is intended for forming and transmission of video stream frames compatible with GigE Vision technology over the network. To estimate the delays in the transmission of a video stream over an Ethernet channel, a methodology based on the Precise Time Protocol (PTP) has been proposed and applied. During the experiments, it was found that the software implementation of a GigE Vision compatible camera on single-board computers with general-purpose processor cores is quite promising. Without additional optimization, such an implementation can be successfully used to transmit small frames (with a resolution of up to 640 × 480 pixels), giving a delay less than 10 ms. At the same time, some additional optimizations may be required to transmit larger frames. Namely, a MTU (maximum transmission unit) size value plays the crucial role in latency formation. Thus, to implement a faster camera, it is necessary to select a platform that supports the largest possible MTU (unfortunately, it turned out that it is not possible with Raspberry Pi 4, as it supports relatively small MTU size of up to 2000 bytes). In addition, the image format conversion procedure can noticeably affect the delay. Therefore, it is highly desirable to avoid any frame processing on the transmitter side and, if it is possible, to broadcast raw images. If the conversion of the frame format is necessary, the platform should be chosen so that there are free computing cores on it, which will permit to distribute all necessary frame conversions between these cores using parallelization techniques.**

Keywords — video stream;transmission delay; time synchronization; Ethernet; GigE Vision

ДОДАТОК Б. Копія другої публікації за темою дисертаційної роботи

Visnyk NTUU KPI Seriia – Radiotekhnika Radioaparatobuduvannia, 2020, Iss. 82, pp. 35–43

УДК 004.7:621.39

Акселерована реєстрація MIPI CSI відеопотоку в задачах передачі відео реального часу

Ходнєв Т. А., Голуб М. С., Кужильний О. В., Лисенко О. М., Варфоломєєв А. Ю.

Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

E-mail: *t.khodniev@kpi.ua*

В роботі розглянуто питання зменшення затримок передачі відеопотоку в реальному часі з камер, що передбачають підключення через інтерфейс MIPI CSI. Наведено основні складові затримки реєстрації/передачі відеопотоку, проаналізовано міру їхнього внеску в сумарну затримку, дано оцінку можливості потенційного впливу на них при розробці систем реєстрації/передачі відеопотоку реального часу. Окреслено проблематику, пов'язану з застосуванням буферизації в таких системах, головним чином, вплив наявності в системі покадрової буферизації на величину сумарної затримки. Охарактеризовано обмеження реалізацій модулів MIPI, що призводять до збільшення затримок реєстрації відеопотоку з MIPI CSI камер в певних ARM-мікропроцесорах. Запропоновано структурно-функціональну організацію систем реєстрації MIPI CSI відеопотоку з застосуванням потокових цифрових шин, фрагментації кадрів відеопотоку та DMA транзакцій, що не потребує використання покадрової буферизації та, відповідно, дозволяє зменшити сумарну затримку реєстрації відеопотоку. Запропоновану структурно-функціональну організацію може бути реалізовано на базі SoC-FPGA рішень, в тому числі, з використанням існуючих IP-ядер. Наведено прагматичні особливості та відповідний оціночний вираз для визначення обмежень величини затримки при застосуванні запропонованих рішень. Для експериментальної перевірки, створено прототип системи реєстрації/передачі відеопотоку на основі SoC-FPGA Xilinx сімейства Zynq-7000 відповідно до запропонованої структурно-функціональної організації, розглянуто його специфіку та особливості реалізації. Дано оцінку отриманій швидкодії прототипу та розглянуто можливі напрямки подальшого зменшення сумарної затримки реєстрації/передачі відеопотоку. Результати роботи можуть бути використані для зменшення величини затримок реєстрації відеопотоку з MIPI CSI камер в системах відеопередачі реального часу на основі SoC-FPGA.

Ключові слова: SoC, FPGA, MIPI CSI, GigE Vision, відео, потокова передача

DOI: 10.20535/RADAP.2020.82.35-43

Вступ

По мірі того, як задачі, що потребують використання відеокамер стають складнішими, все більшою стає потреба у датчиках зображення з високою роздільною здатністю. Актуальні вимоги ринку до датчиків зображення призводять до обмеженої застосовуваності типових паралельних інтерфейсів, оскільки вони відносно складно масштабуються за рахунок необхідності у збільшенні кількості сигнальних ліній. Це стало однією з передумов розробки альянсом MIPI (Mobile Industry Processor Interface) стандарту CSI (Camera Serial Interface) для забезпечення стандартизованого, надійного та високошвидкісного інтерфейсу сенсорів зображень з низьким енергоспоживанням, який би підтримував широкий спектр сучасних рішень з використанням камер [1].

В попередніх роботах [2–5] було показано, що мінімізація транспортних затримок є однією з ключових задач передачі відео в реальному часі. Успішне вирішення даної задачі потребує накладення ряду обмежень як на апаратну, так і на програмну складову взаємодіючих систем. В роботі [3] було проаналізовано ефективність використання тракту зв'язку широковживаними прикладними технологіями передачі відео реального часу — на базі RTSP та на базі GigE Vision стеків та порівняно ці конкуруючі технології між собою за критерієм ефективності використання ними тракту зв'язку топології точка-точка з Ethernet-каналом. Результати роботи [3] свідчать про те, що технологія відеопередачі на базі GigE Vision створює менше навантаження на канал передачі, призводить до меншої кількості помилок в процесі передачі (а відповідно і практично елімінує затримки, пов'язані з необхідністю повторної передачі кадрів відео чи їх складових), є більш ефективною та потенційно більш придатною для вирішення задач передачі відео в реальному часі. В роботах [3,4] було представлено та апробовано модифіковану версію програмної бібліотеки ARAVIS, що

дозволяє реалізувати сумісну зі стандартами GigE Vision трансляцію відеопотоку з камери.

Дана ж робота, в свою чергу, присвячена вирішенню задачі мінімізації затримки реєстрації та подальшої передачі відеопотоку з MIPI CSI камер за рахунок застосування акселерованих IP-модулів (Intellectual Property) реєстрації відео на базі SoC-FPGA (System on a Chip – Field Programmable Gate Array) та є логічним продовженням циклу попередніх робіт за прикладною тематикою організації високошвідкісних трактів відеопередачі реального часу.

1 Проблематика

В системах, що отримують потік відео з камери, підключеної через інтерфейс MIPI CSI, здійснюють його попередню обробку та подальшу передачу кінцевому пристрою-приймачу, сумарна затримка передачі відеопотоку може бути оцінена, виходячи з (1):

$$T_{D\Sigma} = T_{Dcam} + T_{Dif} + T_{Dpp} + T_{Dtx}, \qquad (1)$$

де $T_{D\Sigma}$ – сумарна затримка передачі відеопотоку, с; T_{Dcam} – затримка реєстрації та обробки зображення камерою, с; T_{Dif} – затримка інтерфейсу (MIPI CSI), с; T_{Dpp} – затримка попередньої обробки відеопотоку, с; T_{Dtx} – затримка передачі відеопотоку кінцевому вузлу, с.

Розглянемо внесок окремих складових (1) в сумарну затримку передачі відеопотоку та можливість потенційного впливу на них при розробці відповідних апаратно-програмних рішень:

– Затримка реєстрації та обробки зображення камерою (T_{Dcam}) пов'язана з часом, необхідним для фіксації зображення матрицею камери, а також обробки зображення самою камерою перед подальшою передачею з використанням інтерфейсу MIPI CSI, є специфічною для кожного конкретного модуля камери та, як правило, наводиться у відповідній документації. Тим не менш, можна виділити ряд особливостей даної затримки, характерних для більшості MIPI CSI камер.

  По-перше, це тип затвору модуля камери (rolling/global shutter). Зображення рухомих об'єктів, отримані з застосуванням глобального затвору є більш чіткими, однак, використання плаваючого затвору дозволяє досягти потенційно менших затримок реєстрації відео [6]. Варто зазначити, що спотворення, викликані застосуванням плаваючого затвору при зйомці рухомих об'єктів можуть бути частково компенсовані шляхом утилізації відповідних алгоритмів корекції на етапах попередньої обробки відеопотоку системою відеопередачі чи обробки результуючого відеопотоку на кінцевому вузлі-приймачі [7].

  По-друге, це особливості апаратної організації модуля камери та її постініціалізаційна конфігурація. Варто зазначити, що не всі модулі камер здатні передавати реєстрований потік відео безпосередньо та можуть вносити додаткові затримки в міру особливостей структури та організації їх апаратної складової. Загалом, деякі камери дозволяють розпочинати відправку кадру відео лише після його повного захоплення та буферизації. Деякі камери надають додаткові можливості обробки реєстрованих кадрів перед подальшою передачею відеопотоку через інтерфейс MIPI CSI. Зокрема, вони можуть здійснювати просторове фільтрування для подавлення шумів, поворот кадрів, стиснення у JPEG тощо [8]. Вимкнення певного функціоналу обробки, що не підтримує операцій над потоком, потребує буферизації, за наявності такої можливості, може призвести до суттєвого зменшення загальної затримки реєстрації відеопотоку.

– Затримка інтерфейсу (T_{Dif}) включає в себе час, необхідний для формування MIPI пакетів камерою, їх відправки трансівером камери, проходження сигнальних ліній C-PHY (D-PHY або M-PHY), отримання їх приймачем системи реєстрації відео, а також час, необхідний для представлення системою реєстрації результуючого відеопотоку у вигляді, придатному для його подальшої обробки [9, 10]. Характерною особливістю даної затримки є необхідність фрагментації вихідного відеопотоку та його інкапсуляції в пакети MIPI у відповідності до специфікації інтерфейсу [1]. При цьому, істотний вплив на величину даної затримки (а відповідно, і сумарної затримки передачі відеопотоку) має специфіка організації роботи системи реєстрації відео з одержаними через CSI-інтерфейс пакетами. У випадках, коли система (або окрема її складова) виконує покадрову буферизацію вхідного відеопотоку, розпочати подальшу обробку стає можливим лише після повного збереження системою останнього отриманого кадру в буфері, відповідно сумарна затримка становитиме принаймні (2):

$$T_{D\Sigma}\Big|_{FB} > T_{Dif} > \frac{1}{N_{FPS}}, \qquad (2)$$

де N_{FPS} – кількість кадрів, що надходять за секунду; $|_{FB}$ – нотація виконання умови застосування покадрової буферизації.

Так, наприклад, якщо частота кадрів камери становить 25 кадрів/с, при застосуванні покадрової буферизації затримка передачі відео-

потоку системою становитиме більше 40 мс. Варто зазначити, що подібне значення затримки вважається неприйнятним при вирішенні значної кількості задач, пов'язаних з відеопередачею реального часу (таких, як системи машинного зору, системи відеотелеметрії літальних апаратів, системи автоматизованого контролю дорожньої обстановки тощо). Питання зменшення даної затримки детально розглядається в роботі надалі.

– Затримка попередньої обробки відеопотоку (T_{Dpp}) включає в себе суму затримок кожної з виконуваних операцій обробки відеопотоку (таких, як гамма-корекція, подавлення шумів тощо). Виходячи з тої ж прагматики, що була зазначена вище, необхідною умовою для зменшення даної затримки є забезпечення використання лише тих операцій попередньої обробки, які підтримують обробку відео в потоковому режимі, не потребуючи при цьому повної буферизації кожного відеокадру.

Окремо розглянемо операції компресії. При передачі відеопотоку без стиснення, збільшення роздільної здатності та частоти кадрів відео призводить до збільшення навантаження на тракт передачі. У випадках, коли обмежена смуга пропускання тракту системи відеопередачі реального часу не дозволяє передавати відео з бажаними характеристиками в нестисненому вигляді, для подолання таких обмежень доцільно використовувати алгоритми відеокомпресії. При цьому варто враховувати, що не всі алгоритми підтримують стиснення відеокадрів частками та потребують, як мінімум, наявності одного цілого кадру відео у буфері. Інші алгоритми дозволяють порційну потокову обробку відеопотоку (наприклад, H.264 в intra-frame режимі) [11]. Для їх застосування, як правило, достатньою є наявність буферу в певну кількість рядків (або стовпців) відеокадру.

Варто також враховувати, що використання операцій компресії відеопотоку, в загальному випадку, призводить не лише до збільшення затримок на стороні передавача, а й на стороні кінцевого вузла, оскільки такий відеопотік потребуватиме подальшого декодування.

Іншим аспектом використання компресії відеопотоку є здатність алгоритму коректно відпрацьовувати завади у разі їх виникнення. Такі завади можуть бути викликані пошкодженням пікселів камери, викривленнями через надходження в лінії передачі сильних електромагнітних сигналів ззовні тощо.

Резюмуючи, можна стверджувати, що в міру ряду характерних особливостей, застосування алгоритмів відеокомпресії значно ускладнює задачу створення систем потокової відеопередачі реального часу та може бути як необхідним, так і неприйнятним в конкретній системі, виходячи з вирішуваних нею задач.

– Затримка передачі відеопотоку кінцевому вузлу (T_{Dlt}) включає в себе затримку на представлення результату попередньої обробки в форматі, придатному для подальшої передачі, а також затримки, пов'язані з роботою використовуваного системою відеопередачі стеку мережевих протоколів та відповідних технологій. В раніш представлених роботах розглядалися ключові аспекти деяких широкозастосовуваних стрімінгових протоколів, що задовольняють вимогам передачі відео в реальному часі – загалом, на основі GigE Vision та RTSP [2–4].

Таким чином, виходячи з вищенаведеного огляду затримок систем відеопередачі, які використовують отримання відеопотоку з MIPI CSI камер, можна зробити висновок, що затримки, викликані покадровою буферизацією мають істотний вплив на сумарну затримку передачі та, відповідно, призводять до обмежень можливості передачі відеопотоку в реальному часі. І якщо покадрова буферизація з боку модуля камери може бути усунена шляхом конфігурації існуючої камери або вибором іншого модуля камери, то усунення покадрової буферизації з боку реєструючої відеопотік системи є більш проблематичним, оскільки пов'язане з архітектурою апаратного модуля MIPI самої системи.

В ході підготовчих етапів дослідження, було проаналізовано ряд доступних та комерційно придатних ARM-мікропроцесорів на предмет потенційної доцільності створення системи реєстрації та передачі відеопотоку реального часу з MIPI CSI камер на їх базі. Встановлено, що більшість таких мікропроцесорів (наприклад, сімейства i.MX6 компанії NXP чи сімейства AM57x компанії Texas Instruments), в міру особливостей апаратної реалізації їхніх модулів MIPI CSI або взагалі не підтримують захоплення відеопотоку MIPI-камери без його буферизації в процесі, або не мають належної підтримки такого функціоналу на рівні низькорівневих програмних компонентів системи (відповідні модулі ядра Linux тощо) [12, 13].

З іншого боку, з появою систем на кристалі з вбудованим апаратним ARM-ядром та FPGA-матрицею для втілення довільної програмованої користувацької логіки (таких, як Zynq компанії Xilinx або Cyclone V компанії Intel), стає можливою гнучка реалізація апаратних компонентів систем відеореєстрації відповідно до вирішуваних задач, в тому числі, створення такого технічного рішення захоплення, яке б не потребувало покадрової буферизації і, відповідно, мало б зменшену в порівнянні з буферизуючими системами захоплення затримку реєстрації відеопотоку. Структурно-функціональна

організація та принципи побудови таких систем розглядаються в роботі надалі.

2 Запропоноване рішення

2.1 Структурно-функціональна організація системи

Для вирішення вищезазначених проблем, пов'язаних із затримкою інтерфейсу через застосування покадрової буферизації, запропоновано структурно-функціональну організацію системи реєстрації відеопотоку без покадрового буферу на базі FPGA-SoC. На рис. 1 наведено запропоноване рішення, нижче представлено відповідний опис структурно-функціональної організації системи та її окремих компонентів.

В основі запропонованого рішення лежить застосування MIPI-модуля реєстрації відеопотоку, що підтримує подальшу передачу отримуваних з камери даних через високопродуктивну цифрову потокову шину (наприклад, AXI4-Stream [14]). На відміну від розглянутих у попередньому розділі існуючих та розповсюджених реалізацій MIPI-модулів, даний модуль не містить в своєму складі вбудованого кадрового буфера та не виконує будь-яких затратних (в сенсі внесеної затримки) операцій над вхідним відеопотоком, натомість, делегуючи всі подальші задачі обробки іншим компонентам системи реєстрації відеопотоку. Ключовою особливістю потокових цифрових шин є можливість забезпечення безпосередньої передачі байт (або групи байт), без необхідності у таких операціях, як перетворення формату даних, їх попередє збирання у пакети шини, обов'язкова адресація чи підтримка арбітрації великої кількості паралельних запитів доступу до шини тощо [14, 15]. В контексті запропонованого рішення це дозволяє, по-перше, елімінувати додаткові затримки за рахунок спрощення набору підтримуваних операцій шини, а по-друге, забезпечити можливість коректного відпрацювання більшості з вказаних в попередньому розділі завад передачі відеопотоку камери. За рахунок наявності значної кількості вже готових програмних компонентів, необхідних для створення систем реєстрації/передачі відеопотоку (операційні системи реального часу, стеки мережевих протоколів, бібліотеки обробки відеопотоку та ін.), з врахуванням наявності інтегрованих апаратних процесорних ядер в SoC-FPGA рішеннях таких виробників, як Intel та Xilinx, що мають більшу швидкодію, ніж аналогічні софт-ядра на базі FPGA [16], автори вважають програмну реалізацію відповідних алгоритмів відеообробки та передачі більш простою, виправданою та менш затратною у порівнянні з реалізацією на базі лише цифрової логіки. Представлена на рис. 1 структурно-функціональна організація враховує особливості функціонування вбудованих мікропроцесорних ядер SoC-FPGA з метою зменшення можливих затримок. Розглянемо дані особливості. Головним чином, мікропроцесорні ядра ARM оптимізовані для обробки даних, що знаходяться в оперативній пам'яті [17]. Разом із використанням вбудованих операційних систем, це дозволяє досягти водночас як достатньо швидкої реакції ядра на зовнішні події (наприклад, переривання щодо надходження нової порції вхідних даних), так і високопродуктивного виконання пріорітетних обчислювальних процедур, так і виконання ряду фонових, менш пріорітетних задач, а також дозволяє зменшити енергоспоживання мікропроцесорного ядра під час очікування. І хоча виробники SoC-FPGA мікросхем надають можливість з'єднання FPGA-логіки з ARM-ядром через спеціалізовані інтерфейси вводу-виводу з низькою затримкою (наприклад, EMIO в Xilinx Zynq-7000 [18]), такий підхід в значній мірі ускладнює задачу синхронізації доступу до даних, вочевидь потребує застосування операційних систем строгого реального часу з заздалегідь визначеним максимальним часом реакції на зовнішні події [19] та накладає обмеження на застосування високопродуктивних наборів інструкцій (типу SIMD) для обробки відеопотоку [17]. Виходячи з наведених обмежень, прийнято рішення про використання принципу порційного доступу до даних відеопотоку мікропроцесорним ядром із застосуванням DMA-контролера (Direct Memory Access), фрагментатора відеопотоку та механізму переривань.

Відповідно до запропонованого рішення (рис. 1), після отримання від MIPI-камери відеопотоку, через цифрову потокову шину дані надходять до блоку фрагментатора потоку. Фрагментатор збирає складові відеокадру у фрагменти та направляє їх контролеру DMA. Такими складовими можуть бути один чи декілька рядків/стовпців кадру відео або група пікселів одного рядку (стовпця), в залежності від вимог до системи реєстрації відеопотоку та відповідних обмежень максимально допустимої затримки. DMA-контролер, в свою чергу, забезпечує направлення фрагменту до оперативної пам'яті системи та подачу запиту контролеру переривань системи після успішного запису фрагменту у пам'ять. Контролер переривань встановлює переривання, що сигналізує мікропроцесорному ядру про доступність отриманого фрагменту відеопотоку для подальшої обробки.

Таким чином, при реалізації запропонованої структурно-функціональної організації системи, теоретичне значення мінімальної величини затримки інтерфейсу (T_{Dif}), за грубої оцінки, обмежується знизу відповідно до (3):

$$T_{D\Sigma}\Big|_{SF} > T_{Dif} > \frac{1}{N_{SF\Sigma} \cdot N_{FPS}}, \qquad (3)$$

де $N_{SF\Sigma}$ – кількість фрагментів, на які поділяється кадр відео; N_{FPS} – кількість кадрів, що надходять

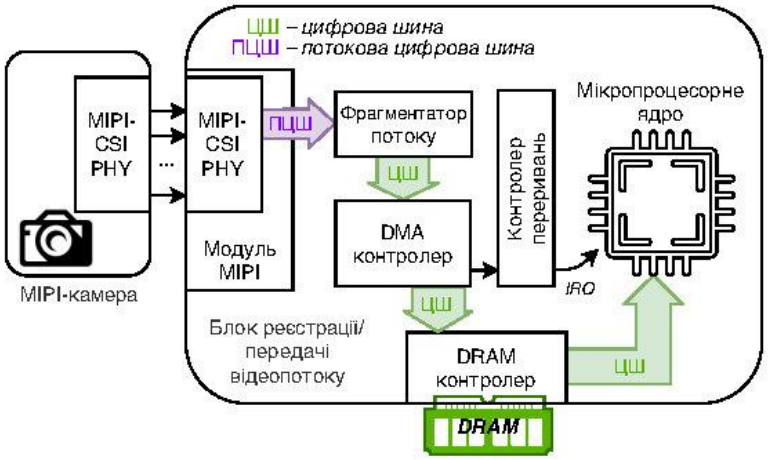


Рис. 1. Структурно-функціональна організація системи

за секунду; $|_{SF}$ – нотація виконання умови застосування запропонованої структурно-функціональної організації (з фрагментацією кадрів відеопотоку).

Розглядаючи наявні засоби для реалізації запропонованого рішення, варто зазначити про потенційну можливість застосування існуючих IP-ядер (за відповідності вимогам до створюваних систем реєстрації відеопотоку). Оскільки IP-ядра є готовими та протестованими компонентами, їх використання дозволяє в істотній мірі спростити задачу створення таких систем.

2.2 Експериментальна перевірка

На основі запропонованої структурно-функціональної організації, було створено експериментальний прототип системи реєстрації/передачі відеопотоку та представлено на конкурсі Digilent Design Contest 2019 [20]. Створений прототип реалізує захоплення відеопотоку з MIPI CSI камери (модуль Digilent Pcam 5C на базі сенсора OV5640 [8, 21]), підключеної до плати Zybo Z7-10 (на базі SoC-FPGA Xilinx XC7Z010 сімейства Zynq-7000 [18, 22]) та подальшу його передачу кінцевому вузлу з застосуванням сумісної зі стандарами реалізації стеку протоколів GigE Vision на основі модифікованої програмної бібліотеки ARAVIS [3, 23].

Для створення компонентів програмованої логіки системи використано IP-ядра Xilnix, синтез проводився в середовищі Vivado Design Suite версії 2018.2 [24]. Загалом, для отримання потоку з камери, використано IP-ядро MIPI CSI-2 Receiver Subsystem [25]; в якості цифрової потокової шини (відповідно до рис. 1) застосовано AXI4-Stream [14]; для реалізації фрагментатора потоку використано набір IP-ядер AXI4-Stream Infrastructure [15].

Для побудови програмної складової системи, в якості операційної системи використано спеціалізований embedded Linux дистрибутив на базі Petalinux версії 2017.4, для збірки якого застосовано інструментарій Yocto [26, 27]. До базових мета-шарів (в термінології Yocto), додано шари підтримки Linux-ядром використовуваних системою IP-ядер з автоматично-генерованою конфігурацією BSP (Board Support Package), а також ряд користувацьких шарів, що забезпечують підтримку роботи бібліотеки ARAVIS, загалом, meta-aravis [28]. Для передачі відеопотоку використано представлену в попередніх роботах модифікацію ArvFakeCamera компоненту ARAVIS [3, 4], до якого в подальшому було заплановано внесення змін з метою отримання відеопотоку в режимі DMABUF програмного інтерфейсу V4L2 (Video 4 Linux 2) підсистеми ядра Linux [29]. Даний режим дозволяє отримувати відеопотік в просторі користувача (userspace) напряму з DMA-буферів, що виділяються драйвером відеопристрою, тим самим мінімізуючи затримку доступу до даних.

В ході роботи над прототипом, було виявлено ряд помилок в реалізації програмних компонентів системного рівня підтримки IP-ядер (некоректне рапортування підсистемі V4L2 формату кадру відеопотоку, помилки синхронізації DMA транзакцій, відносно неефективна реалізація роботи з багатопланарними DMA-буферами в драйвері відеопристрою), що в значній мірі обмежило можливість більш оптимізованої реалізації проекту. Загалом, при надходженні потоку з використанням DMA, для його захоплення було використано режим MMAP підсистеми V4L2 замість дещо більш ефективного режиму DMABUF [29]. Задача виправлення виявлених помилок виходить за рамки даної роботи, оскільки, з одного боку, підтримка таких програмних компонентів покладена на компанію-розробника, а з іншого боку, пов'язана з дотриманням встановленого порядку прийняття змін до відповідних підсистем Linux-ядра.

Тим не менш, для створеного прототипу системи реєстрації/передачі відеопотоку, в ході оцінки результатів, при передачі відеопотоку з роздільною здатністю 1920x1080 пікселів в форматі YUV 4:2:2,

25 кадрів/с, досягнуто скорочення затримки отримання відеопотоку в 11.7 мс, в порівнянні прототипу створеної системи з реалізацією, представленою у референс-дизайнах компанії Digilent [22]. Проміжні результати проекту створеної системи розміщено в GitHub-репозиторії [30].

Розглядаючи можливості подальшого скорочення затримок в системах на базі SoC-FPGA Xilinx, варто зазначити про можливість застосування IP-ядер AXI VDMA (AXI Video Direct Memory Access) та VFB R/W (Video Frame Buffer Read/Write) [31, 32]. Ключовою особливістю IP-ядра VFB R/W є забезпечення порівняно швидкого захоплення кадру потоку з AXI4-Stream та подальшої буферизації. У разі, якщо замість цілого кадру, передавати на вхід даного ядра фрагмент кадру, існує потенційна можливість подальшого зменшення затримок за рахунок усунення інших проміжних IP-ядер з шляху передачі даних, кожне з яких вносить свою затримку. Проте, такий підхід вочевидь потребуватиме створення власних специфічних модулів програмованої логіки.

3 Обговорення результатів

Виходячи з концепції запропонованої структурно-функціональної організації системи та отриманих результатів її експериментальної реалізації, можна стверджувати, що задача мінімізації затримок захоплення відеопотоку в створюваних системах реєстрації/передачі відео реального часу потребує комплексного підходу як до архітектури таких систем, так і до вибору апаратної складової та відповідних програмних компонентів користувацького та системного рівнів. При цьому, варто зазначити про можливість створення таких систем у вигляді програмно-апаратних комплексів (наприклад, на основі SoC-FPGA з використанням процесорних ядер) або у вигляді суто апаратних рішень (наприклад, на основі FPGA без використання процесорних ядер). Застосування суто апаратного підходу може дозволити досягти менших величин затримок, проте, є на порядок більш складним у реалізації та подальшій довгостроковій підтримці створюваних систем.

При застосуванні програмно-апаратного підходу до створення систем реєстрації/передачі відеопотоку, важливим фактором є необхідність забезпечення програмною складовою системи обмежень реального часу відповідно до вирішуваної задачі. Для системного рівня програмної складової, виправданим є використання операційних систем реального часу (RTOS — Real-Time Operating System). Варто зазначити, що ядро Linux є ядром операційної системи широкого призначення та не вважається RTOS-ядром, хоча і частково підтримує певний функціонал реального часу. При цьому, екосистема вбудованого Linux має значну кількість вже готових й протестованих програмних компонентів та інструментів, доступних розробнику системи для перевикористання, що дозволяє в істотній мірі скоротити час, необхідний для вирішення задач розробки. Отже, з одного боку, при створенні систем реєстрації/передачі відеопотоку, реалізація програмної складової системи на основі Linux є досить доцільною, а з іншого боку, призводить до обмежень реального часу. Для подолання таких обмежень, до Linux-ядра може бути застосовано набір патчів реального часу (наприклад, RT-Preempt), що в свою чергу може потребувати внесення змін до драйверів пристроїв для їх коректної роботи. Іншою альтернативою є використання спеціалізованих Linux-сумісних операційних систем реального часу, таких як QNX. QNX запускає ядро Linux як привілейовану задачу власного планувальника реального часу. В порівнянні з вищезазначеними патчами Linux, це дозволяє досягти більш строгих обмежень реального часу. Тим не менш, це потребує реалізації відповідних драйверів, що повинні виконуватися в реальному часі, у вигляді драйверів QNX (з можливістю делегації Linux певних не критичних операцій). Окрім цього, на відміну від Linux, QNX розповсюджується на комерційній основі за пропріетарною ліцензією. Останньою з зазначених альтернатив, є використання окремих RTOS (наприклад, FreeRTOS), що дозволяють досягти строгих обмежень реального часу. Але недоліком такого підходу є більші витрати часу на розробку через відсутність значної кількості вже готових програмних компонентів.

Представлена в роботі структурно-функціональна організація систем реєстрації/передачі відеопотоку призначена для реалізації відповідно до програмно-апаратного підходу, проте, може бути адаптована для суто апаратного підходу у разі необхідності.

Висновки

В роботі розглянуто питання зменшення затримок в системах реєстрації/передачі відеопотоку з MIPI-CSI камер, наведено аналіз окремих складових сумарної затримки реєстрації/передачі відеопотоку, їхніх характерних особливостей, відповідних обмежень та можливостей потенційного впливу на кожну з цих складових з метою зменшення сумарної затримки при створенні таких систем. Головним чином, окреслено проблематику, пов'язану із використанням в системах реєстрації/передачі відеопотоку покадрової буферизації, що призводить до обмеження мінімально досяжної затримки захоплення.

Запропоновано структурно-функціональну організацію системи реєстрації/передачі відеопотоку без покадрової буферизації, що дозволяє досягти зменшеної затримки реєстрації відеопотоку з MIPI CSI камер. Запропоноване рішення може бути реалізовано на базі SoC-FPGA, в тому числі з використа-

нням вже готових IP-ядер. В основі запропонованої структурно-функціональної організації лежить застосування цифрової потокової шини, фрагментатора відеопотоку та модуля DMA. Такий підхід дозволяє реалізувати операції попередньої обробки вхідного відеопотоку програмно, на наявних в SoC-FPGA мікропроцесорних ядрах. Для експериментальної перевірки запропонованого рішення, на його основі було створено прототип системи реєстрації/передачі відеопотоку на базі SoC-FPGA Xilinx XC7Z010 сімейства Zynq-7000. Не зважаючи на виявлені в процесі роботи над прототипом обмеження підтримуваних виробником відповідних IP-модулів Xilinx програмних компонентів системного рівня, в ході експерименту досягнуто зменшення затримки отримання відеопотоку в 11.7 мс, що доводить потенційну придатність запропонованих рішень.

Результати роботи можуть бути використані для зменшення сумарної затримки захоплення відеопотоку з MIPI CSI камер в системах реєстрації/передачі відеопотоку, що потребують дотримання обмежень реального часу та допускають реалізацію з використанням SoC-FPGA.

Подяка

Роботу було виконано в рамках держбюджетної теми «Багатоканальний тепловізійно-телевізійний комплекс пошуку-виявлення із завадостійким швидкісним інтерфейсом передачі даних» (д/р №0118U003751) за підтримки Міністерства освіти і науки України.

Перелік посилань

1. MIPI Camera Serial Interface 3 (MIPI CSI-3)/*MIPI Alliance*

2. Ходнев Т. А. Оценка эффективности использования тракта связи протоколами RTSP-видеовещания в задачах передачи видеопоследовательностей реального времени / Т. А. Ходнев, А. Ю. Варфоломеев // *XII Міжнародно науково-технічна конференція "Проблеми телекомунікацій" ПТ-2018: Збірник матеріалів конференції*. – Київ : КПІ ім. Ігоря Сікорського, 2018. – 500 с. – с. 332-335.

3. Khodniev T. A., Varfolomieiev A. Y., Lysenko O. M. , Antonyuk O. I. (2018) Comparison of RTSP and GigE Vision video streaming technologies in terms of communication path utilization efficiency: an experimental approach. *2018 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo)*. Одеса, с. 1-4. – DOI:10.1109/UkrMiCo43733.2018.9047531.

4. Марченко В. І. Програмно-апаратна реалізація відеокамери, сумісної зі стандартом GigE Vision / В. І. Марченко, Т. А. Ходнев, А. Ю. Варфоломеєв // *Мікросистеми, Електроніка та Акустика*. – 2018. – Т. 23, № 5. – с. 32-37. DOI : 10.20535/2523-4455.2018.23.5.147686.

5. Ходнев Т. А. Поуровнево-декомпозиционная модель оценки интегральной эффективности использования тракта связи с учетом помех / Т. А. Ходнев, А. И. Антонюк, А. Ю. Варфоломеев, А. Н. Лысенко // *Мікросистеми, Електроніка та Акустика*. – 2018. – Т. 23, № 6. – с. 29-33. DOI : 10.20535/2523-4455.2018.23.6.154720.

6. *Rolling Shutter vs. Global Shutter* / Teledyne QImaging

7. Liang C. K. Analysis and compensation of rolling shutter effect / C. K. Liang, L. W. Chang, H. H. Chen // *IEEE Transactions on Image Processing*. – 2008. – Т. 17, № 8. – с. 1323-1330. – DOI : 10.1109/TIP.2008.925384

8. OV5640: color CMOS QSXGA (5 megapixel) image sensor with OmniBSI technology / *OmniVision Technologies Inc*.

9. Low-latency design considerations for video-enabled drones (SPRY301) / *Texas Instruments Inc*.

10. Ahmad J. FPGA based Deterministic Latency Image Acquisition and Processing System for Automated Driving Systems / J. Ahmad, A. Warren // *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. – 2018. DOI : 10.1109/iscas.2018.8351472.

11. Understanding and Reducing Latency in Video Compression Systems / *CAST Inc*.

12. MIPI-CSI2 Peripheral on i.MX6 MPUs (AN5305) / *NXP Semiconductors Inc*.

13. Camera Abstraction Layer - Processor SDK Linux Documentation / *Texas Instruments Inc*.

14. AMBA 4 AXI4-Stream Protocol / *ARM Ltd*.

15. AXI4-Stream Infrastructure IP Suite v3.0 (PG085) / *Xilinx Inc*.

16. Jayakrishnan V. Embedded Processors on FPGA: Soft vs Hard / V. Jayakrishnan, C. Parikh // *Proceedings of the 2019 ASEE North Central Section Conference*. – 2019

17. ARM A-Profile Architecture Specifications / *ARM Ltd*.

18. Zynq-7000 SoC Data Sheet: Overview (DS190) / *Xilinx Inc*.

19. Siewert S. Real-Time Embedded Components And Systems Using Linux And RTOS / Siewert S. and Pratt J. – 2-re вид. – Dulles, VA : Mercury Learning and Information, 2016. – 500 с. – ISBN: 1942270046

20. Digilent Design Contest 2019 EU Region Finalists / *Digilent Inc*.

21. Pcam 5C: 5 MP Color Camera Sensor / *Digilent Inc*.

22. Zybo Z7: Zynq-7000 ARM/FPGA SoC Development Board / *Digilent Inc*.

23. ARAVIS: A vision library for genicam based cameras / *Github*

24. Vivado Design Suite / *Xilinx Inc*.

25. MIPI CSI-2 Receiver Subsystem v4.1 IP (PG232) / *Xilinx Inc*.

26. PetaLinux Tools / *Xilinx Inc*.

27. Yocto Project / *Linux Foundation*

28. Meta-aravis: Yocto layer for the Aravis application / *Github*

29. V4L2 API Input/Output: Streaming I/O (DMA buffer importing) / *Linux Kernel Organization Inc*.

30. Т. А. Ходнев, М. С. Голуб, О. В. Кужильний. *Gigevision-xilinx: GigE Vision compatibe video streaming from MIPI-CSI camera with Zybo Z7-10 board*

31. AXI Video Direct Memory Access v6.3 (PG020) / Xilinx Inc.

32. Video Frame Buffer Read and Video Frame Buffer Write v2.1 (PG278) /Xilinx Inc.

References

[1] MIPI Alliance, *MIPI Camera Serial Interface 3 (MIPI CSI-3)*, standard, viewed 18 Jun 2020.

[2] Khodniev T. A. and Varfolomieiev A. Y. (2018) Evaluating the efficiency of communication path utilization by RTSP broadcasting protocols in tasks of real-time video sequences transmission. *Conference materials of XIIth International Scientific Conference "Modern challenges in telecommunications"*, pp. 332-335, in Russian.

[3] Khodniev T. A., Varfolomieiev A. Y., Lysenko O. M. and Antonyuk O. I. (2018) Comparison of RTSP and GigE Vision video streaming technologies in terms of communication path utilization efficiency: an experimental approach. *2018 International Conference on Information and Telecommunication Technologies and Radio Electronics (UkrMiCo)*, pp.1-4. DOI: 10.1109/ukrmico43733.2018.9047531.

[4] Marchenko V. I., Khodniev T. A. and Varfolomieiev A. Y. (2018) Software and Hardware Implementation of Video Camera, Compatible with GigE Vision Standard. *Microsystems, Electronics and Acoustics*, Vol. 23, Iss. 5, pp. 32-37. DOI: 10.20535/2523-4455.2018.23.5.147686.

[5] Khodniev T. A., Antoniuk O. I., Varfolomieiev A. Y. and Lysenko O. M. (2018) By-Layer Decomposition Model for Evaluating the Integral Communication Path Utilization Efficiency with Account for Errors. *Microsystems, Electronics and Acoustics*, Vol. 23, Iss. 6, pp. 29-33. DOI: 10.20535/2523-4455.2018.23.6.154720.

[6] Teledyne QImaging, *Rolling Shutter vs. Global Shutter*, technical note, viewed 18 Jun 2020.

[7] Liang C., Chang L. and Chen H. (2008) Analysis and Compensation of Rolling Shutter Effect. *IEEE Transactions on Image Processing*, Vol. 17, Iss. 8, pp. 1323-1330. DOI: 10.1109/tip.2008.925384.

[8] OV5640: color CMOS QSXGA (5 megapixel) image sensor with OmniBSI technology. *OmniVision Technologies Inc.*, datasheet, archived, viewed 18 Jun 2020.

[9] Low-latency design considerations for video-enabled drones (SPRY301). *Texas Instruments Inc.*, application note, viewed 18 Jun 2020.

[10] Ahmad J. and Warren A. (2018) FPGA based Deterministic Latency Image Acquisition and Processing System for Automated Driving Systems. *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*. DOI: 10.1109/iscas.2018.8351472

[11] Understanding and Reducing Latency in Video Compression Systems. *CAST Inc.*, viewed 18 Jun 2020.

[12] MIPI-CSI2 Peripheral on i.MX6 MPUs (AN5305). *NXP Semiconductors Inc.*, application note, viewed 18 Jun 2020.

[13] Camera Abstraction Layer - Processor SDK Linux Documentation. *Texas Instruments Inc.*, developer's guide, viewed 18 Jun 2020.

[14] AMBA 4 AXI4-Stream Protocol. *ARM Ltd.*, specification, viewed 18 Jun 2020.

[15] AXI4-Stream Infrastructure IP Suite v3.0 (PG085). *Xilinx Inc.*, product guide, viewed 18 Jun 2020.

[16] Jayakrishnan V. and Parikh C. (2019) Embedded Processors on FPGA: Soft vs Hard. *Proceedings of the 2019 ASEE North Central Section Conference*.

[17] ARM A-Profile Architecture Specifications. *ARM Ltd.*, reference manual, viewed 18 Jun 2020.

[18] Zynq-7000 SoC Data Sheet: Overview (DS190). *Xilinx Inc.*, product specification, viewed 18 Jun 2020.

[19] Siewert S. and Pratt J. (2016) *Real-Time Embedded Components And Systems Using Linux And RTOS*, 2nd ed., Dulles, Virginia: Mercury Learning and Information, ISBN: 1942270046

[20] Digilent Design Contest 2019 EU Region Finalists. *Digilent Inc.*

[21] Pcam 5C: 5 MP Color Camera Sensor. *Digilent Inc.*, product page, viewed 18 Jun 2020.

[22] Zybo Z7: Zynq-7000 ARM/FPGA SoC Development Board. *Digilent Inc.*, product page, viewed 18 Jun 2020.

[23] ARAVIS: A vision library for genicam based cameras. *GitHub, software library repository*, viewed 18 Jun 2020.

[24] Vivado Design Suite. *Xilinx Inc.*, product page, viewed 18 Jun 2020.

[25] MIPI CSI-2 Receiver Subsystem v4.1 IP (PG232). *Xilinx Inc.*, product guide, viewed 18 Jun 2020.

[26] PetaLinux Tools. *Xilinx Inc.*, product page, viewed 18 Jun 2020.

[27] *Yocto Project*, Linux Foundation project home, viewed 18 Jun 2020.

[28] Meta-aravis: Yocto layer for the Aravis application. *GitHub, software repository*, viewed 18 Jun 2020.

[29] V4L2 API Input/Output: Streaming I/O (DMA buffer importing). *Linux Kernel Organization Inc.*, Linux kernel v. 4.9 documentation, viewed 18 Jun 2020.

[30] Khodniev T. A., Holub M. S. and Kuzhylnyi O. V. (2020) Gigevision-xilinx: GigE Vision compatibe video streaming from MIPI-CSI camera with Zybo Z7-10 board. *GitHub, project software repository*, viewed 18 Jun 2020.

[31] AXI Video Direct Memory Access v6.3 (PG020). *Xilinx Inc.*, product guide, viewed 18 Jun 2020.

[32] Video Frame Buffer Read and Video Frame Buffer Write v2.1 (PG278). *Xilinx Inc.*, product guide, viewed 18 Jun 2020.

Акселерированная регистрация MIPI CSI видеопотока в задачах передачи видео реального времени

Ходнев Т. А., Голуб М. С., Кужильный О. В., Лысенко А. Н., Варфоломеев А. Ю.

В работе рассмотрены вопросы уменьшения задержек передачи видеопотока в реальном времени с камер, предусматривающих подключение через интерфейс MIPI CSI. Приведены основные составляющие задержки регистрации/передачи видеопотока, проанализирована степень их вклада в суммарную задержку, дана оценка возможности потенциального воздействия на них при разработке систем регистрации/передачи видеопотока реального времени. Обозначена проблематика, связанная с применением покадровой буферизации в таких системах, главным образом, воздействие наличия покадровой буферизации в системе на величину суммарной задержки. Охарактеризованы ограничения реализаций модулей MIPI, приводящие к увеличению задержек регистрации видеопотока с MIPI CSI камер в некоторых ARM-микропроцессорах.

Предложена структурно-функциональная организация систем регистрации MIPI CSI видеопотока с использованием потоковых цифровых шин, фрагментации кадров видеопотока и DMA транзакций, которая не требует использования покадровой буферизации и, соответственно, позволяет уменьшить суммарную задержку регистрации видеопотока. Предложенная структурно-функциональная организация может быть реализована на основе SoC-FPGA решений, в том числе, с использованием существующих IP-ядер. Приведены прагматические особенности и соответствующее оценочное выражение для определения ограничений величины задержки при использовании предложенных решений.

Для экспериментальной проверки, создан прототип системы регистрации/передачи видеопотока на основе SoC-FPGA Xilinx семейства Zynq-7000, в соответствии с предложенной структурно-функциональной организацией, рассмотрена его специфика и соответствующие особенности реализации. Дана оценка полученному быстродействию прототипа и рассмотрены возможные направления дальнейшего уменьшения суммарной задержки регистрации/передачи видеопотока.

Результаты работы могут быть использованы для уменьшения задержек регистрации видеопотока с MIPI CSI камер в системах видеопередачи реального времени на основе SoC-FPGA.

Ключевые слова: SoC; FPGA; MIPI CSI; GigE Vision; видео; потоковая передача

Accelerated MIPI CSI video stream acquision in tasks of real-time video streaming

Khodniev T. A., Holub M. S., Kuzhylnyi O. V., Lysenko O. M., Varfolomieiev A. Y.

In present study the challenges of reducing transmission latencies of a real-time video stream acquired from cameras connected via the MIPI CSI interface were addressed. In the study, the main components of the video stream acquisition/transmission latency are given, the degree of their contribution to the total latency was analyzed, the assessment on the potential ability to influence them when developing a real-time video stream acquisition/transmission systems was given. The issues connected with using the frame buffering in such systems are designated, primarily the impact on the total latency value when having a framebuffer in such a system. The limitations of the existing MIPI module implementations of some ARM microprocessors resulting in latency increase for MIPI CSI camera video stream acquisition were characterized.

The structural and functional organization based on the use of digital streaming buses, fragmentation of video frames and DMA transactions for MIPI CSI video stream acquisition systems was proposed, which does not require the use of framebuffers and, as a result, provides the possibility of reducing the overall video stream acquisition latency. The proposed structural and functional organization could be implemented based on SoC-FPGA solutions, including the use of the existing IP-cores. Pragmatic peculiar features were described and the corresponding expression for estimating the limiting value of the latency for the proposed structural and functional organization was given.

For experimental verification, a prototype of the video stream acquisition/transmission system, based on the Zynq-7000 SoC-FPGA family of Xilinx following the proposed structural and functional organization was created. Its specifics and corresponding features of its implementation were discussed in the paper. The performance of the obtained prototype was estimated, and the possible directions towards further reduction of the overall latency of video stream acquisition/transmission were considered.

The results of the study may prove useful to reduce the latencies of the video streams acquired from MIPI CSI cameras in real-time video stream transmission systems based on SoC-FPGA.

Key words: SoC; FPGA; MIPI CSI; GigE Vision; video; streaming

ДОДАТОК В. Лістинг драйверу захоплення відеопотоку

v4l2-capture.c

```
#include "v4l2-capture.h"

#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>
#include <string.h>

#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <poll.h>
#include <signal.h>
#include <pthread.h>

#include <linux/videodev2.h>

static const char *def_dev_path = "/dev/video0";

bool v4l2_is_polling;

#define LOG(...)       do { \
                            printf("v4l2_capture: "); \
                            printf(__VA_ARGS__); printf("\n"); \
                          } while (0)

static int xioctl(int fh, int request, void *arg)
{
        int rc;

        do {
                rc = ioctl(fh, request, arg);
        } while (-1 == rc && EINTR == errno);

        return rc;
}

static void v4l2_poll_exit(int sig)
{
        if (sig == SIGINT)
                v4l2_is_polling = false;
}

static struct v4l2_camera *v4l2_create_camera(size_t width,
                                              size_t height,
                                              size_t pix_fmt)
{
        struct v4l2_camera *c;

        c = calloc(1, sizeof(*c));

        if (c != NULL) {
```

```
                c->params.width = width ? width : V4L2_WIDTH_DEFAULT;
                c->params.height = height ? height : V4L2_HEIGHT_DEFAULT;
                c->params.pixel_format = pix_fmt ? pix_fmt :
V4L2_PIXEL_FORMAT_DEFAULT;

                LOG("Camera has been created");
        } else
                LOG("Not able to create camera");

        return c;
}

static int v4l2_open_device(const char *dev_path)
{
        int vfd;

        vfd = open(dev_path ? dev_path : def_dev_path, O_RDWR | O_NONBLOCK, 0);

        if (vfd < 0)
                LOG("Cannot open '%s': %d, %s", dev_path, errno,
strerror(errno));
        else
                LOG("Camera was opened: %s", dev_path ? dev_path : def_dev_path);

        return vfd;
}

static int v4l2_getset_capability(struct v4l2_camera *c)
{
        struct v4l2_capability cap;
        int                    rc;

        memset(&cap, 0, sizeof(cap));

        rc = xioctl(c->vfd, VIDIOC_QUERYCAP, &cap);
        if (rc != 0) {
                LOG("Camera capability wasn't fetched");
                if (errno == EINVAL)
                        LOG("Opened device not a V4L2 device");
        }

        if (rc == 0) {
                if (cap.capabilities & V4L2_CAP_VIDEO_CAPTURE) {
                        c->params.field = V4L2_FIELD_DEFAULT;
                        c->params.type  = V4L2_BUF_TYPE_VIDEO_CAPTURE;
                        LOG("V4L2_CAP_VIDEO_CAPTURE mode");
                } else if (cap.capabilities & V4L2_CAP_VIDEO_CAPTURE_MPLANE) {
                        c->params.field = V4L2_FIELD_NONE;
                        c->params.type  = V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE;
                        LOG("V4L2_CAP_VIDEO_CAPTURE_MPLANE mode");
                } else {
                        LOG("Camera doesn't support specified capabilities");
                        rc = -1;
                }
        }

        return rc;
}
```

```
static int v4l2_getset_format(struct v4l2_camera *c)
{
        struct v4l2_format     fmt;
        int                    rc;

        memset(&fmt, 0, sizeof(struct v4l2_format));

        fmt.type = c->params.type;
        rc = xioctl(c->vfd, VIDIOC_G_FMT, &fmt);
        if (rc != 0)
                LOG("VIDIOC_G_FMT failed");

        if (rc == 0) {
                switch(c->params.type) {
                        case V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE:
                                fmt.fmt.pix_mp.width  = c->params.width;
                                fmt.fmt.pix_mp.height = c->params.height;
                                fmt.fmt.pix_mp.field  = c->params.field;
                                fmt.fmt.pix_mp.num_planes  = c->mplane_num;
                                fmt.fmt.pix_mp.pixelformat = c-
>params.pixel_format;
                        break;
                        case V4L2_BUF_TYPE_VIDEO_CAPTURE:
                                fmt.fmt.pix.width  = c->params.width;
                                fmt.fmt.pix.height = c->params.height;
                                fmt.fmt.pix.field  = c->params.field;
                                fmt.fmt.pix.pixelformat = c->params.pixel_format;
                        break;
                        default:
                                LOG("Camera format is not supported");
                                rc = -1;
                        break;
                        }
        }

        if (rc == 0) {
                rc = xioctl(c->vfd, VIDIOC_S_FMT, &fmt);
                if (rc == 0) {
                        LOG("Camera width: %zu", c->params.width);
                        LOG("Camera height: %zu", c->params.height);
                        LOG("Camera field was set %zu", c->params.field);
                        LOG("Camera type was set %u type", c->params.type);
                } else
                        LOG("VIDIOC_S_FMT failed");
        }

        return rc;
}

static void v4l2_mplane_destroy(struct v4l2_camera *c, size_t n)
{
        for (size_t i = 0; i < n; i++) {
                free(c->fb[i].f.head);
                free(c->fb[i].f.length);
        }
}

static int v4l2_allocate_fb(struct v4l2_camera *c, size_t fb_num, size_t
mplane_num)
```

```
{
        size_t                  i;
        int                     rc;

        rc = 0;
        i  = 0;
        c->fb_num = fb_num ? fb_num : V4L2_REQUESTED_BUFFERS_NUM;
        c->mplane_num = mplane_num ? mplane_num : V4L2_REQUESTED_PLANES_NUM;
        c->fb = calloc(c->fb_num, sizeof(*c->fb));

        LOG("Buffers number: %zu", c->fb_num);
        LOG("Mplane number: %zu", c->mplane_num);

        if (c->fb == NULL) {
                rc = -ENOMEM;
                goto enomem;
        }

        for ( ; i < c->fb_num && rc == 0; i++) {
                c->fb[i].f.head = calloc(c->mplane_num, sizeof(*(c-
>fb[i].f.head)));
                if (c->fb[i].f.head == NULL) {
                        rc = -ENOMEM;
                        break;
                }

                c->fb[i].f.length = calloc(c->mplane_num, sizeof(*(c-
>fb[i].f.length)));
                if (c->fb[i].f.length == NULL) {
                        rc = -ENOMEM;
                        break;
                }
        }

        if (rc != 0) {
                v4l2_mplane_destroy(c, i);
                free(c->fb);
        }

enomem:

        if (rc == 0)
                LOG("Camera buffers have been allocated");
        else
                LOG("Not sufficient memory for %zu buffers", c->fb_num);

        return rc;
}

static int v4l2_mmap_fb(struct v4l2_camera *c, struct v4l2_buffer *mmap_fb,
size_t buf_index)
{
        int rc;

        rc = 0;
        for (size_t j = 0; j < c->mplane_num && rc == 0; j++) {
                switch(c->params.type) {
                        case V4L2_BUF_TYPE_VIDEO_CAPTURE:
                                c->fb[buf_index].f.length[j] = mmap_fb->length;
```

```
                                c->fb[buf_index].f.head[j] = mmap(NULL,
                                                                  mmap_fb->length,
                                                                  PROT_READ | PROT_WRITE,
                                                                  MAP_SHARED,
                                                                  c->vfd,
                                                                  mmap_fb->m.offset);
                        break;
                        case V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE:
                                c->fb[buf_index].f.length[j] = mmap_fb-
>m.planes[j].length;
                                c->fb[buf_index].f.head[j] = mmap(NULL,
                                                                  c->fb->f.length[j],
                                                                  PROT_READ | PROT_WRITE,
                                                                  MAP_SHARED,
                                                                  c->vfd,
                                                                  mmap_fb-
>m.planes[j].m.mem_offset);
                        break;
                        default:
                                rc = -1;
                        break;
                }

                LOG("Camera mmap mplane[%zu] length: %zu", j, c-
>fb[buf_index].f.length[j]);
                LOG("Camera frame buffer [%zu] address: %p", buf_index, c-
>fb[buf_index].f.head[j]);

                if (rc == 0 && c->fb->f.head[j] == MAP_FAILED) {
                        LOG("Camera mmap failed");
                        rc = -1;
                }
        }

        return rc;
}

static int v4l2_mmap_camera(struct v4l2_camera *c)
{
        struct v4l2_requestbuffers req;
        int                        rc;

        memset(&req, 0, sizeof(req));

        req.count       = c->fb_num;
        req.type        = c->params.type;
        req.memory      = V4L2_MEMORY_MMAP;

        rc = xioctl(c->vfd, VIDIOC_REQBUFS, &req);
        if (rc != 0)
                LOG("VIDIOC_REQBUFS failed: %d, %s", errno, strerror(errno));

        for (size_t i = 0; i < c->fb_num && rc == 0; i++) {
                struct v4l2_buffer mmap_fb;
                memset(&mmap_fb, 0, sizeof(mmap_fb));

                mmap_fb.index  = i;
                mmap_fb.memory = V4L2_MEMORY_MMAP;
                mmap_fb.type   = c->params.type;
```

```
                if (c->params.type == V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE) {
                        struct v4l2_plane mmap_mplane;
                        memset(&mmap_mplane, 0, sizeof(mmap_mplane));

                        mmap_fb.length   = c->mplane_num;
                        mmap_fb.m.planes = &mmap_mplane;
                }

                rc = xioctl(c->vfd, VIDIOC_QUERYBUF, &mmap_fb);

                if (rc != 0) {
                        LOG("VIDIOC_QUERYBUF failed: %d, %s\n", errno,
strerror(errno));

                        if (errno == EINVAL)
                                LOG("The buffer type is not supported, or the
index is out of bounds");
                }

                if (rc == 0)
                        rc = v4l2_mmap_fb(c, &mmap_fb, i);
        }

        if (rc == 0)
                LOG("Camera has been mapped");

        return rc;
}

static int v4l2_enqueue_all_buf(struct v4l2_camera *c)
{
        int               rc;

        rc = 0;
        for (size_t i = 0; i < c->fb_num && rc == 0; i++) {
                struct v4l2_buffer v4l2_buf;
                memset(&v4l2_buf, 0, sizeof(v4l2_buf));

                v4l2_buf.memory= V4L2_MEMORY_MMAP;
                v4l2_buf.field = c->params.field;
                v4l2_buf.type  = c->params.type;
                v4l2_buf.index = i;

                if (c->params.type == V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE) {
                        struct v4l2_plane  mplane;
                        memset(&mplane, 0, sizeof(mplane));

                        v4l2_buf.m.planes = &mplane;
                        v4l2_buf.length = c->mplane_num;
                }

                rc = xioctl(c->vfd, VIDIOC_QBUF, &v4l2_buf);
                if (rc != 0)
                        LOG("VIDIOC_QBUF failed");
                LOG("Camera buffer[%zu] flag: 0x%x", i, v4l2_buf.flags);
        }

        if (rc == 0)
```

```
                LOG("The initial camera buffer was enqueued");

        return rc;
}

static int v4l2_dequeue_enqueue_buf(struct v4l2_camera *c)
{
        int                rc;
        struct v4l2_buffer v4l2_buff;

        memset(&v4l2_buff, 0, sizeof(v4l2_buff));

        v4l2_buff.memory = V4L2_MEMORY_MMAP;
        v4l2_buff.field  = c->params.field;
        v4l2_buff.type   = c->params.type;

        /* TODO: Not sure that it will correct for mplanes dequeue */
        if (c->params.type == V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE) {
                struct v4l2_plane  mplanes[c->mplane_num];
                memset(mplanes, 0, c->mplane_num * sizeof(mplanes));

                v4l2_buff.m.planes = mplanes;
                v4l2_buff.length = c->mplane_num;
        }

        rc = xioctl(c->vfd, VIDIOC_DQBUF, &v4l2_buff);
        if (rc != 0) {
                if (errno == EAGAIN)
                        LOG("VIDIOC_DQBUF - frame not ready: %d, %s", errno,
strerror(errno));

                LOG("Camera flags: 0x%x", v4l2_buff.flags);
                LOG("VIDIOC_DQBUF failed: %d, %s", errno, strerror(errno));
        }

        if (rc == 0) {
                if (c->params.type == V4L2_BUF_TYPE_VIDEO_CAPTURE) {
                        c->fb->bytes_used = v4l2_buff.bytesused;
                        c->fb->index = v4l2_buff.index;
                        LOG("Bytes num:%zu", c->fb->bytes_used);
                        LOG("Index of frame buffer:%zu", c->fb->index);
                        LOG("Address of frame buffer:%p", c->fb[c->fb-
>index].f.head);
                }

                pthread_mutex_lock(&c->c_lock);
                c->v4l2_is_frame_ready = true;
                pthread_mutex_unlock(&c->c_lock);

                rc = xioctl(c->vfd, VIDIOC_QBUF, &v4l2_buff);
                if (rc != 0)
                        LOG("VIDIOC_QBUF failed");
        }

        if (rc !=0 )
                LOG("Enqueue dequeue frame buffer failed");

        return rc;
}
```

```
static int v4l2_stream_on(struct v4l2_camera *c)
{
        int rc;

        rc = xioctl(c->vfd, VIDIOC_STREAMON, &c->params.type);
        if (rc != 0)
                LOG("VIDIOC_STREAMON failed: %d, %s", errno, strerror(errno));
        else
                LOG("Camera stream has been started");

        return rc;
}

static int v4l2_stream_off(struct v4l2_camera *c)
{
        int rc;

        rc = xioctl(c->vfd, VIDIOC_STREAMOFF, &c->params.type);
        if (rc != 0)
                LOG("VIDIOC_STREAMOFF failed: %d, %s", errno, strerror(errno));
        else
                LOG("Camera stream has been stopped");

        return rc;
}

static void v4l2_close_camera(struct v4l2_camera *c)
{
        int rc;

        rc = close(c->vfd);

        if (rc != 0)
                LOG("Camera device closing failure: %d, %s", errno,
strerror(errno));
        else
                LOG("Camera was closed");
}

static int v4l2_munmap_camera(struct v4l2_camera *c)
{
        int rc;

        rc = 0;

        for (size_t i = 0; i < c->fb_num; i++) {
                for (size_t j = 0; j < c->mplane_num; j++) {
                        rc = munmap(c->fb[i].f.head[j], c->fb[i].f.length[j]);
                        if (rc != 0)
                                LOG("V4L2 munmap[%zu] failed: %d, %s", i, errno,
strerror(errno));
                        else
                                LOG("Camera munmap %p buffer [%zu] with length
%zu",
                                        c->fb[i].f.head[j], i, c-
>fb[i].f.length[j]);
                }
        }
```

```
        return rc;
}

static void v4l2_release_device(struct v4l2_camera *c)
{
        v4l2_stream_off(c);
        pthread_mutex_destroy(&c->c_lock);
        v4l2_munmap_camera(c);
        v4l2_mplane_destroy(c, c->fb_num);
        free(c->fb);
        v4l2_close_camera(c);
        free(c);

        LOG("Camera was released");
}

static void *v4l2_poll_frame_thread(void *camera)
{
        struct v4l2_camera *c;
        struct pollfd     fds;
        int               rc;

        c = (struct v4l2_camera *)camera;

        memset(&fds, 0, sizeof(fds));

        LOG("Control signal were initialized");

        fds.fd = c->vfd;
        fds.events = POLLIN;

        do {
                rc = poll(&fds, 1, 2);

                if (rc < 0) {
                        if (errno != EINTR)
                                break;
                        continue;
                }

                if (fds.revents & POLLIN && c->v4l2_is_frame_ready == false)
                        v4l2_dequeue_enqueue_buf(c);


        } while (v4l2_is_polling == true);

        v4l2_release_device(c);

        return NULL;
}

static int v4l2_start_thread(struct v4l2_camera *c)
{
        pthread_t      v4l2_thread;
        int            rc;

        rc = pthread_mutex_init(&c->c_lock, NULL);
        if (rc == 0) {
```

```
                rc = pthread_create((pthread_t *)&v4l2_thread,
                                    NULL,
                                    v4l2_poll_frame_thread,
                                    (void*)c);
                if (rc == 0)
                        pthread_detach(v4l2_thread);
        }

        if (rc < 0)
                LOG("Not able to start video thread");


        return rc;
}

struct v4l2_camera *v4l2_start_video_capturing(const char *video_dev,
                                               size_t w, size_t h, size_t pix_fmt,
                                               size_t buf_num, size_t mplane_num)
{
        struct v4l2_camera      *c;
        int                     rc;

        v4l2_is_polling = true;

        c = v4l2_create_camera(w, h, pix_fmt);
        if (c == NULL)
                goto camera_not_ready;

        c->vfd = v4l2_open_device(video_dev);
        if (c->vfd < 0)
                goto null_camera;

        rc = v4l2_getset_capability(c);
        if (rc != 0)
                goto broken_camera;

        rc = v4l2_allocate_fb(c, buf_num, mplane_num);
        if (rc != 0)
                goto broken_camera;

        rc = v4l2_getset_format(c);
        if (rc != 0)
                goto null_buf;

        rc = v4l2_mmap_camera(c);
        if (rc != 0)
                goto null_buf;

        rc = v4l2_enqueue_all_buf(c);
        if (rc != 0)
                goto mmap_failed;

        rc = v4l2_stream_on(c);
        if (rc != 0)
                goto mmap_failed;

        rc = v4l2_start_thread(c);
        if (rc != 0)
                goto stream_off;
```

```
        signal(SIGINT, v4l2_poll_exit);

        return c;

stream_off:
        v4l2_stream_off(c);
        pthread_mutex_destroy(&c->c_lock);

mmap_failed:
        v4l2_munmap_camera(c);
        v4l2_mplane_destroy(c, c->fb_num);

null_buf:
        free(c->fb);

broken_camera:
        v4l2_close_camera(c);

null_camera:
        free(c);

camera_not_ready:
        LOG("Camera start failed");

        return NULL;
}

static int v4l2_capture_test(void)
{
        struct v4l2_camera *c;
        int                rc;
        int                record_fd;

        record_fd = open("demo.raw", O_CREAT | O_TRUNC | O_RDWR | O_NONBLOCK,
0644);
        rc = record_fd < 0 ? -1 : 0;

        if (rc != 0)
                LOG("Not able to create output video file");

        if (rc == 0)
                c = v4l2_start_video_capturing(NULL, 0 ,0 ,0, 0, 0);

        if (c != NULL && c->fb != NULL && rc == 0) {
                do {
                        if (c->v4l2_is_frame_ready) {
                                switch(c->params.type) {
                                case V4L2_BUF_TYPE_VIDEO_CAPTURE:
                                        write(record_fd, c->fb[c->fb-
>index].f.head[0], c->fb->bytes_used);
                                break;
                                case V4L2_BUF_TYPE_VIDEO_CAPTURE_MPLANE:
                                        write(record_fd, c->fb->f.head[0], c-
>params.width * c->params.height * 2);
                                break;
                                default:
                                break;
                                }
```

```
                                c->v4l2_is_frame_ready = false;
                        }
                }
                while (v4l2_is_polling);
        }
        pthread_exit(0);
        close(record_fd);
        return rc;
}

int main(void)
{
        return v4l2_capture_test();
}
```

v4l2-capture.h

```
#pragma once

#include <stdbool.h>
#include <stddef.h>
#include <pthread.h>
#include <linux/videodev2.h>

#define V4L2_REQUESTED_BUFFERS_NUM    10
#define V4L2_REQUESTED_PLANES_NUM     1

#define V4L2_WIDTH_DEFAULT            640
#define V4L2_HEIGHT_DEFAULT           480
#define V4L2_PIXEL_FORMAT_DEFAULT     V4L2_PIX_FMT_YUYV
#define V4L2_FIELD_DEFAULT            V4L2_FIELD_ANY

struct v4l2_frame {
        void    **head;          // Beginning of valid frame buffer
        size_t  *length;         // Length of whole frame buffer
};

struct v4l2_frame_buffer {
        struct  v4l2_frame f;
        size_t  index;          // Index of a valid buffer
        size_t  bytes_used;     // Number of used bytes to write frame, may be
less then page_size
};

struct v4l2_camera_params {
        size_t width;
        size_t height;
        size_t pixel_format;
        size_t field;
        enum   v4l2_buf_type type;
};
```

```
struct v4l2_camera {
        /* Descriptor for /dev/videoN node */
        int vfd;

        /* Frame buffers number */
        size_t fb_num;

        /* Number of planes */
        size_t mplane_num;

        /* Pointer to frame buffers */
        struct v4l2_frame_buffer *fb;

        struct v4l2_camera_params params;

        /* Indicated that buffer is ready to be fetched */
        bool v4l2_is_frame_ready;

        /* Lock for v4l2_is_frame_ready */
        pthread_mutex_t c_lock;
};

extern bool v4l2_is_polling;

struct v4l2_camera *v4l2_start_video_capturing(const char *video_dev,
                                               size_t w, size_t h, size_t pix_fmt,
                                               size_t buf_num, size_t mplane_num);
```