

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему **КАСОВА СИСТЕМА НА FPGA**

Виконав: студент 2 курсу, групи КНТ-522м
спеціальності _____

123 Комп'ютерна інженерія

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні системи та мережі

НАЙДЬОНОВ М. Ю.

(ПРИЗВИЩЕ та ініціали)

Керівник ГРУШКО С.С.

(ПРИЗВИЩЕ та ініціали)

Рецензент САМОЙЛИК С.С.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Комп'ютерних наук і технологій
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти магістерський
 Спеціальність 123 Комп'ютерна інженерія
 (код і найменування)
 Освітня програма (спеціалізація) «Комп'ютерні системи та мережі»
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Зав. кафедри Кудерметов Р.К.

“24” жовтня 2023 року

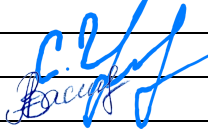
З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

НАЙДЬОНОВА Микити Юрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Касова система на FPGA
 керівник проєкту (роботи) к. т. н., доцент, Грушко Світлана Сергіївна
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
 затверджені наказом вищого навчального закладу від “24” жовтня 2023 року № 400
2. Строк подання студентом проєкту (роботи) 10 грудня 2023 року
3. Вихідні дані до проєкту (роботи) наявні бібліотеки комп'ютерного зору, наявні апаратні платформи, наявні вебкамери
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Огляд літератури;
2) Архітектура касової системи на FPGA;
3) Розробка системи
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	ГРУШКО С. С., доцент		
нормоконтроль	ПОЛЬСЬКА О.В., ст. викл.		

7. Дата видачі завдання 01.10.2023 р.

КАЛЕНДАРНИЙ ПЛАН

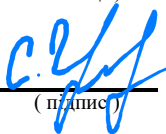
№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз технологій касових систем	10.10.2023 р.	
2	Розробка архітектури системи	15.10.2023 р.	
3	Налагодження програмного забезпечення	20.10.2023 р.	
4	Реалізація архітектури системи	05.10.2023 р.	
5	Відлагодження апаратного забезпечення	20.11.2023 р.	
6	Оформлення отриманих результатів у ПЗ	25.11.2023 р.	
7	Оформлення графічного матеріалу	01.12.2023 р.	
8	Оформлення допоміжного матеріалу	10.12.2023 р.	

Студент

Микита НАЙДЬОНОВ

(ім'я, ПРІЗВИЩЕ)

Керівник проєкту (роботи)

Світлана ГРУШКО

(ім'я, ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 69 сторінок, 14 рисунків, 1 таблиця, 26 джерел

FPGA, ВИЯВЛЕННЯ, КАСОВА СИСТЕМА, СКАНУВАННЯ

Предмет дослідження - розробка та вивчення касової системи на базі програмованих логічних пристроїв (FPGA).

Об'єкт дослідження - касові системи та їх ефективність, надійність та функціональність, особливо з використанням FPGA.

Мета дослідження - створення високоефективної, надійної та гнучкої касової системи, яка використовує FPGA для обробки фіскальних операцій, яка може бути адаптована для подальших покращень у цій області.

В першому розділі здійснено класифікацію та огляд існуючих касових систем, включаючи електронні касові апарати, програмні касові системи, мобільні касові системи, касові системи самообслуговування та хмарні касові системи. Розглянуто основні принципи функціонування цих систем та наведено приклади.

В другому розділі описано розроблювану систему, її апаратне та програмне забезпечення, з акцентом на використанні FPGA. Представлено різні види FPGA, такі як Intel Stratix, Arria, Microchip PolarFire та інші, і їх застосування у касових системах.

Третій розділ охоплює практичні аспекти розробки системи, включаючи налаштування середовища, компіляцію, тестування та аналіз результатів.

ABSTRACT

EN: 69 pages, 14 figures, 1 table, 26 sources

FPGA, DETECTION, CASH REGISTER SYSTEM, SCANNING

Research Subject - Development and study of a cash register system based on Field-Programmable Gate Arrays (FPGA).

Research Object - Cash register systems and their efficiency, reliability, and functionality, especially with the use of FPGA.

Research Goal - To create a highly efficient, reliable, and flexible cash register system that uses FPGA for processing fiscal operations, adaptable for further improvements in this area.

In the first chapter, classification and review of existing cash register systems were conducted, including electronic cash registers, software-based systems, mobile cash systems, self-service systems, and cloud-based cash systems. The basic principles of operation of these systems were considered and examples were provided.

In the second chapter, the system under development, its hardware and software, with an emphasis on the use of FPGA, is described. Various types of FPGA, such as Intel Stratix, Arria, Microchip PolarFire, and others, and their application in cash register systems are presented.

The third chapter covers practical aspects of system development, including environment setup, compilation, testing, and analysis of results.

ЗМІСТ

Зміст	6
Вступ	8
1 Огляд літератури	10
1.1 Класифікація касових систем	10
1.1.1 Види касових систем	10
1.1.2 Види касових систем самообслуговування	11
1.2 Основні принципи функціонування касових систем	12
1.3 Приклади касових систем	13
1.4 Огляд існуючих касових систем самообслуговування	16
1.5 Висновки до розділу 1	20
2 Архітектура касової системи на FPGA	21
2.1 Опис розроблюваної системи	21
2.2 Апаратне забезпечення	23
2.2.1 Intel Stratix 10 FPGA	23
2.2.2 FPGA Intel Arria 10	24
2.2.3 Microchip PolarFire SoC	26
2.2.4 Lattice Semiconductor CrossLink-NX	27
2.2.5 AMD Xilinx Zynq UltraScale	29
2.2.6 Webcam Logitech® HD Pro	31
2.3 Програмне забезпечення	33
2.3.1 PetaLinux	33
2.3.2 AMD Vitis Unified Software Platform	35
2.3.3 Vivado Design Suite	39
2.3.4 Xilinx Vitis AI	41
2.3.5 OpenCV (Open Source Computer Vision Library)	46
2.3.6 Docker	46
2.4 Висновки до розділу 2	47

3 Розробка системи	48
3.1 Налаштування середовища для плати ZCU104.....	48
3.2 Компіляція Vitis-AI	51
3.3 Крос-компіляція коду виведення DPU.....	53
3.4 Висновки до розділу 3	54
Висновки	56
Перелік джерел посилання	57
Додаток А.....	60

ВСТУП

Касові системи є невід’ємною частиною сучасної торгівлі та обслуговування клієнтів у різноманітних комерційних установах. Здійснюючи реєстрацію продажів, облік оплати і надання фіскальних документів, вони відіграють ключову роль у забезпеченні порядку та точності фінансових операцій. У контексті сучасних технологічних досягнень, використання програмованих логічних пристроїв (FPGA) стає актуальним способом покращення та оптимізації касових систем.

Ця дипломна робота присвячена розробці та дослідженню касової системи на базі FPGA. Метою цього дослідження є створення високоефективної та надійної касової системи, яка використовує FPGA для обробки фіскальних операцій та має потенціал для подальших покращень у цьому напрямку.

Сучасна технологічно насичена область торгівлі і обслуговування клієнтів вимагає відповідних інструментів для забезпечення ефективності та точності фінансових операцій. Касові системи стали невід’ємною частиною цього процесу, але існуючі рішення не завжди відповідають сучасним вимогам.

Проблема полягає в тому, що багато касових систем використовують застаріле обладнання та програмне забезпечення, що може призводити до низької продуктивності, помилок та збоїв в роботі, а також відкриває можливості для фінансових маніпуляцій. Деякі існуючі рішення також можуть бути вразливими перед можливими атаками, що загрожують конфіденційності фінансової інформації та інтегритету операцій.

Крім того, розширення та модернізація функціональності касових систем можуть бути обмеженими в існуючих рішеннях, і це може ускладнювати впровадження нових технологій та фіскальних стандартів.

Таким чином, в основу цієї дипломної роботи покладено завдання створення ефективної та безпечної касової системи на FPGA, яка має потенціал для

покращення продуктивності, надійності та розширення функціональності касових операцій, а також забезпечення захисту фінансових даних від несанкціонованого доступу.

Метою цього дослідження є розробка та вивчення касової системи на базі програмованих логічних пристроїв з метою покращення продуктивності, надійності та функціональності касових операцій. Досягнення цієї мети передбачає вирішення наступних завдань:

- проектування касової системи – розробка архітектури та функціонального опису касової системи, включаючи визначення потрібного обладнання та програмного забезпечення;

- розробка апаратної частини - реалізація апаратної частини касової системи на FPGA, включаючи проектування логічних схем, інтерфейсів та обчислювальних блоків;

- розробка програмного забезпечення - розробка програмного забезпечення для взаємодії з FPGA, включаючи операції реєстрації продажів, обліку оплати та генерації фіскальних документів.

Тестування та валідація - проведення тестів для перевірки працездатності та надійності касової системи, а також порівняння її продуктивності з існуючими рішеннями.

Аналіз та висновки - аналіз результатів дослідження з метою підсумків та висновків щодо досягнутого рівня ефективності та відповідності меті дослідження.

Рекомендації для подальших досліджень - надання рекомендацій для подальших досліджень та розвитку касових систем на базі FPGA.

Це дослідження ставить перед собою завдання покращити функціональність та ефективність касових систем, а також сприяти розвитку та використанню програмованих логічних пристроїв у сфері фінансових операцій, що має великий потенціал для покращення та інновацій в цій галузі.

1 ОГЛЯД ЛІТЕРАТУРИ

1.1 Класифікація касових систем

1.1.1 Види касових систем

Касові системи представляють собою різноманітні рішення та обладнання, які використовуються для реєстрації продажів, обліку оплати та надання фіскальних документів у різних галузях бізнесу. Вони відіграють ключову роль у забезпеченні точності та порядку фінансових операцій. Розглядаючи різні сфери застосування, види касових систем включають:

Електронні касові апарати (ЕКА) [1]. ЕКА є стандартними касовими пристроями, що використовуються в багатьох магазинах і ресторанах. Вони призначені для реєстрації продажів, обчислення податків та надання фіскальних чеків. ЕКА використовуються для роздрібної торгівлі та обслуговування клієнтів на місцях.

Програмні касові системи (ПКС) [2]. ПКС - це програмні рішення, які можуть бути встановлені на звичайних комп'ютерах або планшетах. Вони надають широкі можливості для керування запасами, аналізу продажів та звітності. ПКС використовуються в різних сферах бізнесу, включаючи ресторани, готелі, магазини та інші сфери.

Мобільні касові системи [3]. Мобільні касові системи використовують сучасні мобільні пристрої, такі як смартфони та планшети, для обробки оплати та видачі фіскальних чеків. Це зручне рішення для підприємств, що надають послуги на виїзних заходах, як ресторани, фестивалі та інші події.

Касові системи самообслуговування (Self-Service Kiosks) [4]. Касові кіоски дозволяють клієнтам самостійно вибирати продукти або послуги, реєструвати їх та здійснювати оплату без прямого взаємодії з обслуговуючим персоналом. Вони використовуються в ресторанах, магазинах та інших місцях обслуговування.

Хмарні касові системи (Cloud-Based POS) [5, 6]. Хмарні касові системи дозволяють зберігати дані та взаємодіяти з ними через інтернет, що полегшує керування ресторанами та магазинами в різних місцях і сприяє забезпеченню оновлень та безпеки даних.

Ці різні види касових систем відповідають різним потребам бізнесу та галузям, а їхні можливості постійно розширюються завдяки технологічному прогресу та змінам у вимогах клієнтів.

1.1.2 Види касових систем самообслуговування

Самообслуговування стає все більш популярним і важливим аспектом в сфері обслуговування клієнтів, особливо в контексті роздрібної торгівлі та громадських місць, де клієнти шукають швидше та зручне обслуговування. Існує кілька видів касових систем самообслуговування, які відповідають різним потребам і вимогам.

Самообслуговування на основі кіосків (Kiosk Self-Service). Кіоски самообслуговування дозволяють клієнтам вибирати товари або послуги, реєструвати їх та здійснювати оплату без прямого взаємодії з обслуговуючим персоналом. Це особливо популярно в ресторанах, готелях, аеропортах та інших місцях, де швидкість обслуговування має велике значення.

Мобільні додатки для самообслуговування (Mobile Self-Service Apps). Завдяки розвитку смартфонів, клієнти можуть використовувати мобільні додатки для вибору продуктів або послуг, замовлення та оплати. Це зручне рішення для роздрібних магазинів, ресторанів та послуг, які пропонують доставку або заборну оплату.

Самообслуговування в банківському секторі (Self-Service Banking). У банківському галузі самообслуговування дозволяє клієнтам здійснювати фінансові операції через банкомати, онлайн-банкінг або мобільні додатки без необхідності відвідування фізичних відділень.

Самообслуговування в громадських місцях (Public Self-Service). Цей вид самообслуговування включає в себе автомати з продажу квитків на громадський

транспорт, кінотеатри, музеї та інші розважальні заклади, де клієнти можуть самостійно купувати квитки та використовувати інші послуги.

Самообслуговування в сфері ресторанного обслуговування (Restaurant Self-Service). Ресторани та кафе надають можливість клієнтам замовляти їжу через електронні меню на планшетах, самостійно оплачувати рахунок та вибирати столиці для обслуговування.

Ці види касових систем самообслуговування допомагають покращити обслуговування клієнтів, зменшити черги та збільшити ефективність в різних галузях бізнесу.

1.2 Основні принципи функціонування касових систем

Робота касових систем базується на кількох ключових принципах, які визначають їхню функціональність та ефективність. Поговоримо про ці принципи більш докладно.

Перший із них - це реєстрація фінансових операцій. Усі фінансові операції, які відбуваються в організації, мають бути точно відображені в системі. Це включає продаж товарів чи послуг, отримання платежів, виплату зарплати та інші операції. Реєстрація може виконуватися автоматично чи вручну, в залежності від характеру операцій.

Другий принцип - зберігання фінансової інформації. Касові системи відповідають за зберігання всієї інформації, що стосується фінансових операцій. Це може включати в себе дані про суми операцій, дати їх проведення, інформацію про клієнтів та інші важливі деталі. Ці дані є цінним резервуаром інформації для подальшого обліку та аналізу фінансів.

Третій принцип - обробка інформації. Касові системи проводять обробку зібраної інформації. Це може включати в себе розрахунок сум операцій, визначення податків, формування фінансових звітів та інші обчислення, які є необхідними для точного фінансового обліку.

Четвертий принцип - забезпечення безпеки і конфіденційності. Касові системи повинні гарантувати безпеку фінансових даних. Це включає захист від несанкціонованого доступу, контроль доступу до функцій системи, а також резервне копіювання даних для запобігання їх втраті.

П'ятий принцип - відображення інформації для клієнтів. Деякі касові системи також відображають інформацію для клієнтів, таку як чеки, квитанції та інші документи, які підтверджують проведені операції. Це сприяє створенню довіри між організацією та її клієнтами, надаючи їм доступну та зрозумілу інформацію щодо їх фінансових транзакцій.

Ці принципи є важливими для правильного функціонування касових систем і визначають їхню роль у забезпеченні точності та ефективності фінансового обліку в різних організаціях та підприємствах.

1.3 Приклади касових систем

Касові системи є необхідним інструментом в сучасному світі роздрібної торгівлі та гостинності. Вони забезпечують не тільки зручність розрахунків, але й дають можливість для аналізу продажів, управління запасами та обліку клієнтів.

Ринок пропонує широкий спектр касових систем, які варіюються від простих недорогих моделей до високотехнологічних комплексних рішень. Наприклад, прості касові апарати, такі як Casio SE-S10 [<http://arch.casio-intl.com/asia-mea/en/sa/entry/SE-S10/>], широко використовуються в малому бізнесі завдяки своїй доступності та простоті в обслуговуванні. Вони забезпечують

основні потреби торгового місця: реєстрацію продажів, друк чеків та управління готівкою.



Рисунок 1.1 - Casio SE-S10 Cash Register

Для середнього бізнесу популярність набувають POS-системи (Point of Sale) з сенсорними екранами, як от Square, Shopify, TouchBistro, IT Retail, Швидкість світла які інтегруються з іншими пристроями та програмним забезпеченням. Такі системи дозволяють не лише приймати платежі різними способами, але й управляти інвентаризацією, вести облік робочого часу співробітників та навіть здійснювати маркетинговий аналіз [7].

В топі ринку знаходяться інтегровані системи для великих рітейлерів та мереж ресторанів, такі як ті, що пропонує IBM. Ці системи включають в себе розширені можливості, такі як автоматичне управління замовленнями постачальників, інтегровані CRM-системи (Customer Relationship Management) для покращення взаємодії з клієнтами та аналітичні інструменти для оптимізації бізнес-процесів.

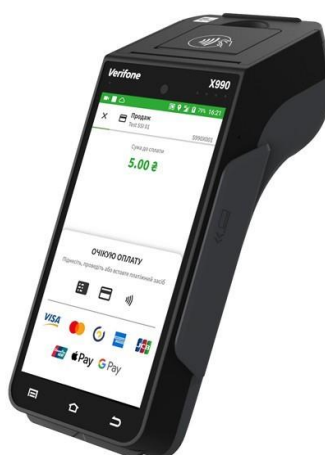


Рисунок 1.2 - POS-термінал з ПРРО Verifone X990 Android

Не оминули увагою касові системи й інновації, такі як впровадження хмарних технологій, які дозволяють бізнесам віддалено керувати продажами та даними через інтернет. Хмарні касові системи, такі як Lightspeed [8] або Shopify [9], надають користувачам гнучкість та доступність, дозволяючи власникам бізнесу переглядати продажі та запаси в реальному часі з будь-якого пристрою з інтернет-з'єднанням.



Рисунок 1.3 – Торгова точка і комерційна платформа Lightspeed

У відповідь на сучасні тенденції, також були розроблені мобільні касові системи, що дозволяють вести торгівлю навіть поза межами стаціонарного магазину. Такі системи, як iZettle [10] або SumUp [11], дають можливість приймати

платежі через кредитні картки використовуючи лише смартфон або планшет із спеціальним додатком і картковим рідером. Це стало революційним рішенням для вуличних торговців, ярмарків та фестивалів, дозволяючи легко розширювати торгівлю за межі традиційних торгових точок.

Важливою тенденцією на ринку є інтеграція касових систем з іншими бізнес-процесами. Так, наприклад, ERP-системи (Enterprise Resource Planning) [12] інтегруються з касовими апаратами, щоб у реальному часі відстежувати запаси та забезпечувати управління ланцюгом поставок. Використання комплексних систем, як Oracle або SAP, допомагає великим підприємствам забезпечувати безперебійність та ефективність бізнес-операцій.

Окрім того, сучасні касові системи надають широкі можливості для забезпечення безпеки платежів. Впровадження технологій шифрування та використання безконтактних платіжних методів, таких як NFC (Near Field Communication), дозволяє збільшити безпеку транзакцій та зменшити ризик шахрайства.

Таким чином, касові системи сьогодні представляють широкий спектр технологій, кожна з яких має свої особливості та переваги. Від простих до складних, вони адаптуються до потреб різного роду бізнесу, пропонуючи рішення, що можуть підвищити ефективність, безпеку та обслуговування клієнтів у будь-якій торгівельній точці.

1.4 Огляд існуючих касових систем самообслуговування

Касові системи самообслуговування з'явилися як відповідь на зростаючу потребу в ефективності та скороченні часу очікування в роздрібних мережах та громадському харчуванні. Ця технологія відкриває нові можливості для взаємодії з покупцями, дозволяючи їм контролювати процес покупки від початку до кінця.

Спочатку інтегровані у великих супермаркетах та домашніх магазинах, таких як ІКЕА, ці системи надавали покупцям можливість самостійно сканувати товари та проводити платіжні операції, тим самим скорочуючи час очікування та звільняючи персонал для виконання інших задач. Великі рітейлери, такі як Walmart і Tesco, розробили власні версії цих систем, що включають сенсорні екрани, інтеграцію з мобільними додатками для сканування та системи лояльності, що підтримують індивідуальний досвід покупок.

Ресторани швидкого обслуговування, як-от McDonald's і KFC, також приєдналися до тренду, впроваджуючи кіоски самообслуговування, де відвідувачі можуть замовляти та оплачувати їжу без необхідності стояти в черзі до каси. Це не тільки підвищило ефективність процесу замовлення, але й дало можливість клієнтам налаштувати свої замовлення з більшою точністю.

В аеропортах по всьому світу, системи самообслуговування використовуються для спрощення процесу реєстрації на рейси. Пасажири можуть використовувати термінали для швидкої реєстрації та друку посадочних талонів, що зменшує навантаження на персонал і покращує загальний досвід подорожі.

Останні інновації включають в себе використання мобільних додатків та безконтактних технологій, які дозволяють виконувати покупки без необхідності фізичного контакту з касовою системою. Наприклад, Amazon Go представив концепцію "Just Walk Out" торгівлі, де клієнти можуть взяти необхідні товари та просто піти з магазину, виплати здійснюються автоматично через їхні Amazon акаунти. Ці технології підвищують рівень комфорту та ефективності для покупців, забезпечуючи при цьому додаткову інформацію для підприємств про покупкові звички та уподобання клієнтів.

З розвитком штучного інтелекту та машинного навчання, касові системи самообслуговування почали пропонувати ще більше персоналізований досвід. Вони можуть запам'ятовувати покупки клієнтів та пропонувати релевантні товари або знижки на основі попередньої поведінки покупців, тим самим стимулюючи додаткові продажі.

Також, сфера готельного бізнесу не залишилась осторонь від впровадження цих технологій. Готелі, як Hilton та Marriott, використовують системи самообслуговування для швидкої реєстрації заїзду та виїзду, що знову ж таки дозволяє зосередити зусилля персоналу на забезпеченні більш якісного обслуговування та взаємодії з гостями.

З огляду на широке розповсюдження мобільних платежів та електронних гаманців, багато касових систем самообслуговування інтегрують такі платформи, як Apple Pay, Google Wallet і Samsung Pay, що ще більше полегшує процес транзакцій.

З урахуванням глобальної пандемії COVID-19, контактність у роздрібній торгівлі та послугах стала питанням здоров'я та безпеки. Такі системи дозволяють мінімізувати безпосередній контакт між людьми, що є ключовим аспектом в підтримці соціальної дистанції та запобіганні поширенню вірусу.

Касові системи самообслуговування продовжують еволюціонувати, відіграючи ключову роль у цифровій трансформації роздрібною торгівлі та послуг. Вони забезпечують зручність та ефективність для клієнтів, одночасно пропонуючи бізнесам додаткові дані для аналізу та вдосконалення сервісу. У майбутньому можна очікувати ще більшої інтеграції цих систем з іншими цифровими технологіями, що відкриє нові можливості для інновацій у сфері обслуговування клієнтів.

В якості прикладів можна розглянути касу самообслуговування GEOS SCO-01. Вона може працювати під операційними системами Linux, Windows 7/8/10 і має характеристики, що вказані далі [13].

Каса самообслуговування GEOS SCO-01 - комплекс торгового обладнання:

- сканер Zebra MP7000 (з ваговою коміркою);
- світлофор;
- Pos-монітор GEOS Pro SM1502;
- GEOS BOX 2;
- MG-P777TL.



Рисунок 1.4 – Каса самообслуговування GEOS SCO-01

Каса самообслуговування GEOS SCO-01 - комплекс торгового обладнання:

- сканер Zebra MP7000 (з ваговою коміркою);
- світлофор;
- Pos-монітор GEOS Pro SM1502;
- GEOS BOX 2;
- MG-P777TL.

Основні характеристики:

- операційна система: Linux, Windows 7/8/10;
- ПЗ: Маркет+;
- виробник: GEOS;
- габарити: 1221x623x519 мм.

Конфігурація коробки-корпуса каси самообслуговування GEOS SCO-01:

- тримач принтера;
- тримач банківського термінала;
- тримач дисплею;

- LED / RGB-індикатор;
- контролер ваг;
- комірка для біотичного сканер;
- висувна комутаційна скриня;
- висувний лоток для сміття;
- поличка для кошика;
- MSR-зчитувач: опція;
- антикражні елементи системи;
- мережевий фільтр;
- блок живлення.

1.5 Висновки до розділу 1

Касові системи є неодмінною частиною фінансової інфраструктури бізнесу, забезпечуючи точність і легальність фінансових операцій. Вони класифікуються за типами на електронні касові апарати, програмні касові системи, мобільні касові системи, касові системи самообслуговування, та хмарні касові системи. Кожен тип має свої особливості та переваги, відповідно до конкретних потреб галузі та формату обслуговування клієнтів. Системи самообслуговування стають все більш популярними завдяки їх здатності оптимізувати процеси обслуговування і зменшувати час очікування. Загалом, різноманіття та гнучкість сучасних касових систем дозволяють підприємствам вибирати оптимальні рішення для забезпечення ефективної роботи, високого рівня обслуговування клієнтів та дотримання законодавчих вимог.

2 АРХІТЕКТУРА КАСОВОЇ СИСТЕМИ НА FPGA

2.1 Опис розроблюваної системи

У сучасному світі торгівлі, касові системи самообслуговування відіграють роль надійних помічників, здатних швидко та ефективно обслуговувати клієнтів. Вони поєднують у собі декілька передових технологій.

Перш за все, це системи сканування та виявлення об'єктів. Завдяки сучасним сканерам штрих-кодів та камерам, ці системи ідентифікують товари з високою точністю, легко розпізнаючи їх серед тисяч інших.

Сенсорні екрани додають інтерактивності, дозволяючи покупцям легко управляти процесом покупки. Це робить весь досвід користування приємнішим та зручнішим.

Крім того, сучасні касові системи інтегровані з різними способами оплати. Від безготівкових транзакцій до мобільних платежів - вони забезпечують гнучкість та швидкість у фінансових операціях.

А ще важливим компонентом є програмне забезпечення для управління продажами. Воно дозволяє вести інвентаризацію, контролювати продажі та формувати звіти, спрощуючи адміністративні процеси.

Але особливу увагу варто звернути на використання FPGA в таких системах. Ці мікросхеми вражають своєю продуктивністю, здатністю обробляти величезні обсяги даних майже миттєво. Їх гнучкість полягає у можливості перепрограмування для виконання різноманітних функцій, що дозволяє адаптувати системи під змінювані потреби бізнесу та технологічний прогрес.

Не варто забувати й про енергоефективність FPGA, яка робить їх більш вигідними для тривалого використання у порівнянні з традиційними мікропроцесорами. І, звісно, високий рівень безпеки та надійності FPGA гарантує захист фінансових транзакцій та персональних даних клієнтів.

Таким чином, касові системи самообслуговування на FPGA стають взірцем інновацій та ефективності в роздрібній торгівлі.

В роботі розробляється прототип каси самообслуговування, реалізований на FPGA і призначений для ідентифікації товару і розрахунку його вартості. Передбачається, що система зможе працювати за двома основними сценаріями.

Перший сценарій - у ролі хмарного сервісу:

- відеозапис товарів - все починається з відео, яке записує товари. Це відео (у форматі, наприклад, mp4) надсилається до системи;
- обробка відео VCU та DPU - з використанням декодера VCU, кадри відео після декодування подаються в DPU (Data Processing Unit) для розрахунку вартості товарів;
- відображення результатів - результати, такі як деталі товарів, їх ціна та відео з детекцією зображень, відображаються на моніторі.

Другий сценарій - у ролі пристрою edge:

- захоплення інформації з камери - інформація про товари зчитується з камери, підключеної до FPGA;
 - обробка зображень DPU - зображення, захоплені камерою, надсилаються в DPU для розрахунку результату товарів;
 - відображення та запис результатів - деталі відображаються на моніторі.
- Відрізняючись від першого сценарію, результати на моніторі записуються кадр за кадром. За допомогою кодера VCU передбачається зберігати відео, складене з цих кадрів. Це має дозволяти переглядати весь процес у разі виникнення спорів.

2.2 Апаратне забезпечення

2.2.1 Intel Stratix 10 FPGA

Для вирішення поставленої задачі потрібна потужна платформа, яка знайшла широке застосування у таких сферах, як вбудовані системи, штучний інтелект, машинне навчання та відео обробка. Розглянемо продукти провідних виробників у галузі FPGA та SoC, таких як Intel (раніше Altera) SoC FPGA, Microsemi (частина Microchip) SoC FPGA, Lattice Semiconductor, AMD Xilinx [15].

Intel Stratix 10 FPGA та SoC FPGA відкривають нову еру в області високопродуктивних FPGA, пропонуючи інноваційні переваги у продуктивності, енергоефективності, щільності та інтеграції систем. Завдяки революційній архітектурі Intel Hyperflex™ FPGA та використанню технології Embedded Multi-Die Interconnect Bridge (EMIB), Advanced Interface Bus (AIB), та зростаючого портфеля чиплетів, ці пристрої забезпечують до двократного збільшення продуктивності порівняно з попереднім поколінням.

Серія Intel Stratix 10 включає декілька моделей, кожна з яких має унікальні характеристики. Наприклад, Intel Stratix 10 SX SoC FPGA має 64-бітну чотириядерну ARM Cortex-A53 систему на чипі, в той час як Intel Stratix 10 TX FPGA надає передові можливості передавача, поєднуючи Н- та Е- передавачі. Intel Stratix 10 MX FPGA є незамінним багатофункціональним прискорювачем для обчислень високої продуктивності (HPC), а Intel Stratix 10 DX FPGA підтримує Intel Ultra Path Interconnect для прямого когерентного з'єднання з майбутніми обраними процесорами Intel Xeon Scalable.

INTEL® STRATIX® 10 GX 10M FPGA – HIGHEST CAPACITY	
Features	Capability
Logic Elements	10.2M
Data Interface Bus via EMIB	Up to 25,920 connections
Memory	308Mbits
DSP - 18x19 Multipliers	6,912
User I/O pins	2,304
LVDS	1,152 @ 1.4Gbps
Transceivers	48 @ 17.4Gbps
Package	70 mm x 74 mm
Debug IP	Readback / Writeback
Availability	Now Shipping Production Silicon

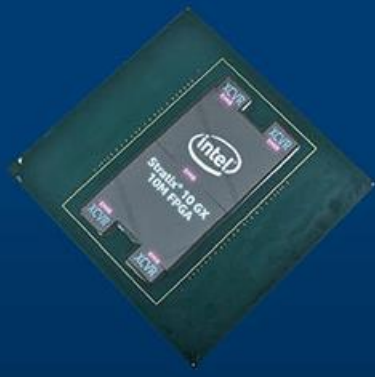


Рисунок 2.1 – Intel Stratix 10

Intel Stratix 10 пропонує до 2X продуктивності ядра завдяки архітектурі Intel Hyperflex™ FPGA, досягаючи до 8.6 TFLOPS одинарної точності обчислень DSP та до двадцяти 100 GbE інтерфейсів. Ці пристрої також пропонують до 7X пропускної здатності передавача порівняно з попередніми поколіннями FPGA, підтримуючи до 144 передавачів в одному пристрої, з швидкістю передачі даних до 57.8 Gbps PAM-4 та 28.9 Gbps NRZ.

Інтеграція до 16 ГБ пам'яті HBM2 в пакеті, найбільшого монолітного FPGA пристрою з 2.8 мільйонами логічних елементів та гетерогенними 3D SiP рішеннями, включаючи передавачі та інші передові компоненти, такі як HBM2, розширює можливості системної інтеграції. Intel® Stratix® 10 NX FPGA вбудовує новий тип AI-оптимізованого блоку, званого AI Tensor Block, який налаштований для загальних матричних операцій, використовуваних у AI обчисленнях.

2.2.2 FPGA Intel Arria 10

Серія FPGA Intel Arria 10 поєднує в собі інноваційну архітектуру та технологію 20 нм від TSMC, що забезпечує підвищену продуктивність і знижене споживання енергії.

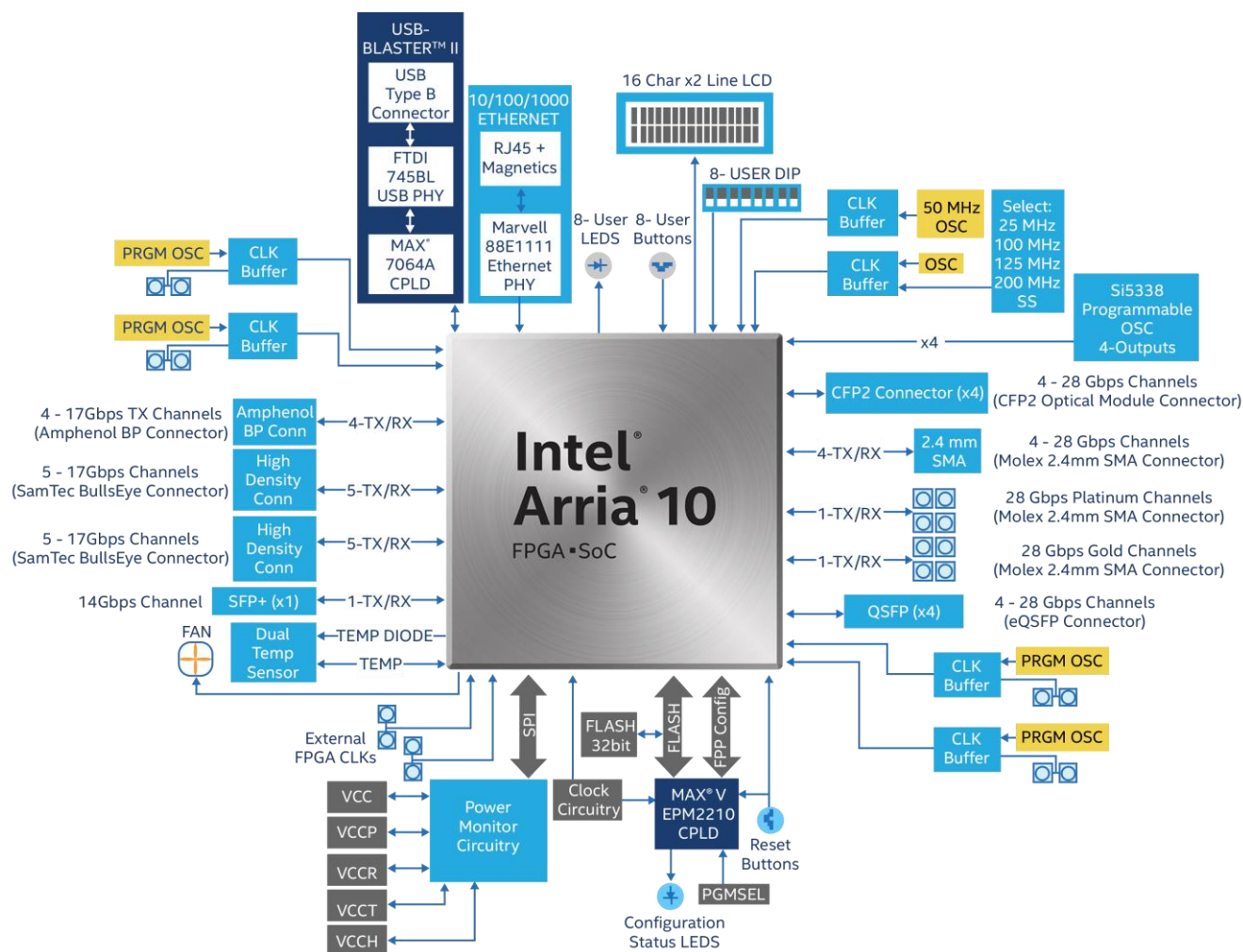


Рисунок 2.2 – Intel Arria 10

Серія включає кілька моделей, які пропонують різні можливості [16]:

- здатний підтримувати до 78 двосторонніх передавачів зі швидкістю передачі даних до 25.78 Gbps;
- підтримує до 96 двосторонніх передавачів зі швидкістю до 17.4 Gbps;
- має двоядерну систему на чипі (SoC) з процесорами ARM Cortex-A9 і до 48 двосторонніх передавачів.

Ці FPGA ідеально підходять для широкого спектру застосувань, включаючи зв'язок, центри обробки даних, військові застосування, мовлення, автомобільну промисловість та інші середнього рівня FPGA додатки.

Intel Arria 10 SoC FPGA забезпечує високий рівень інтеграції системи, з підвищенням продуктивності процесора на 65% до 1.5 GHz на ядро та в 4 рази більшою пропускною здатністю передавача порівняно з попереднім поколінням.

Ця серія також орієнтована на продуктивність, забезпечуючи широкий екосистемний підхід для розробки програмного забезпечення на базі ARM, повну сумісність програмного забезпечення з попередніми поколіннями Intel SoC FPGA та підтримку популярних операційних систем.

2.2.3 Microchip PolarFire SoC

FPGA PolarFire SoC від Microchip є новітнім рішенням у галузі систем на кристалі (SoC), що поєднує в собі високопродуктивну FPGA архітектуру PolarFire з енергоефективними та багатофункціональними процесорами ARM. Ось декілька ключових особливостей та переваг PolarFire SoC FPGA [17].

Енергоефективність - PolarFire SoC FPGA забезпечує високу продуктивність при значно нижчому споживанні енергії порівняно з іншими рішеннями на ринку, що робить їх ідеальними для використання у вбудованих системах, де енергоефективність є критичною.

Продуктивність - висока обчислювальна потужність FPGA дозволяє виконувати складні завдання обробки даних і забезпечує гнучкість для різних застосувань, включаючи обробку зображень, машинне навчання та інші.

Інтеграція з ARM-процесорами - інтеграція з процесорами ARM Cortex-M у SoC забезпечує багатозадачність і здатність обробляти великі потоки даних, а також виконувати складні контрольно-управлінські функції.

Безпека та надійність - PolarFire SoC FPGA пропонують передові рішення для забезпечення безпеки даних та системи, що є важливим для застосувань у таких сферах, як оборонна промисловість та критична інфраструктура.

Ця серія є відмінним вибором для розробників, які шукають високопродуктивні, надійні та енергоефективні рішення для своїх вбудованих систем і застосувань.

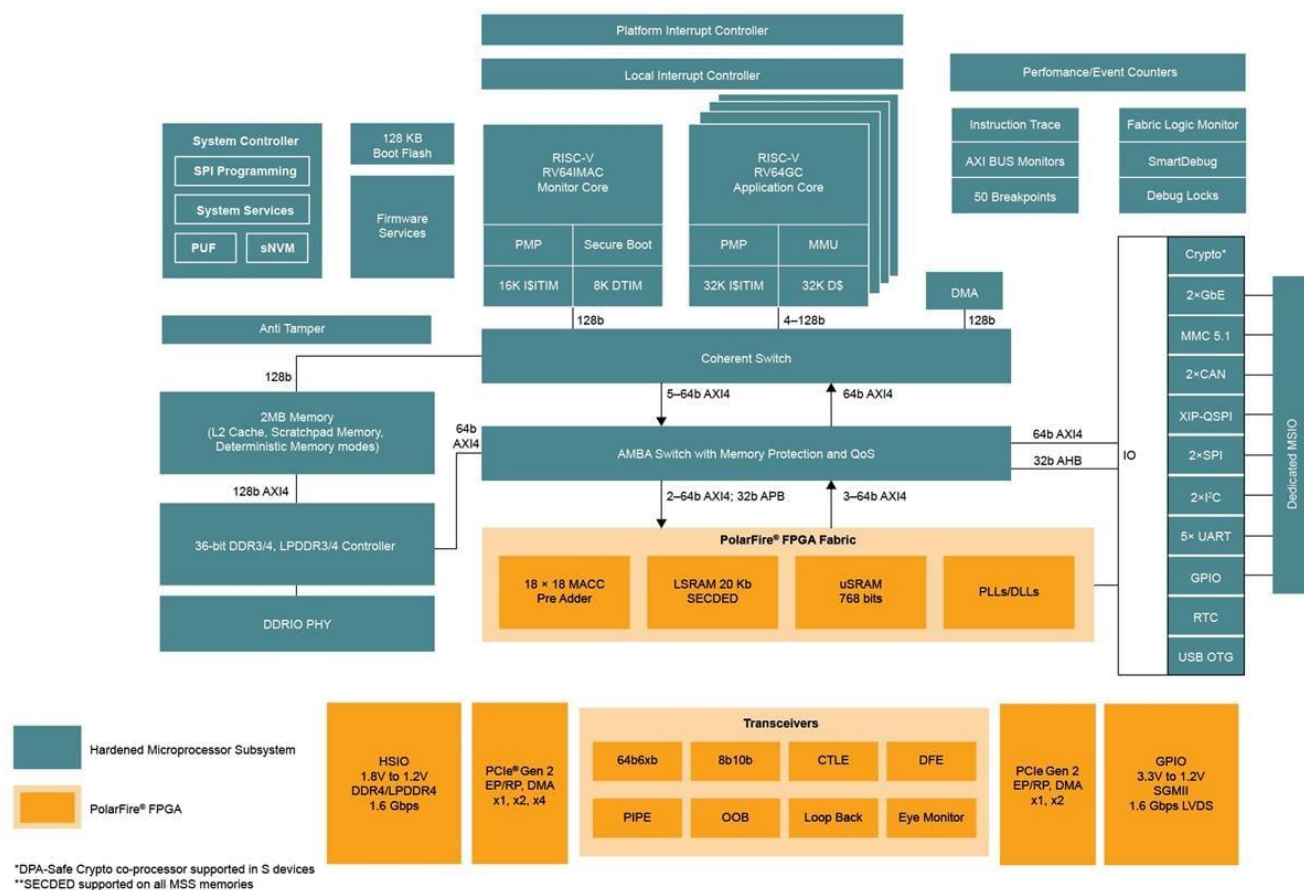


Рисунок 2.3 – FPGA PolarFire SoC

2.2.4 Lattice Semiconductor CrossLink-NX

Lattice Semiconductor CrossLink-NX - це сімейство FPGA від компанії Lattice Semiconductor, відомої своїми інноваційними рішеннями в області програмованих логічних інтегральних схем. Ці FPGA пристрої відрізняються високою продуктивністю, енергоефективністю та невеликими розмірами, що робить їх ідеальними для застосувань, де важливі компактність та ефективність енергоспоживання [18].

CrossLink-NX характеризуються:

- високою швидкістю та енергоефективністю;
- маленьким форм-фактором;
- гнучкістю;

- підтримкою інтерфейсів.

Використовуючи технологію 28-нм FD-SOI (Fully Depleted Silicon On Insulator), CrossLink-NX забезпечує високу продуктивність при низькому енергоспоживанні. Це робить їх ідеальними для мобільних та портативних пристроїв.

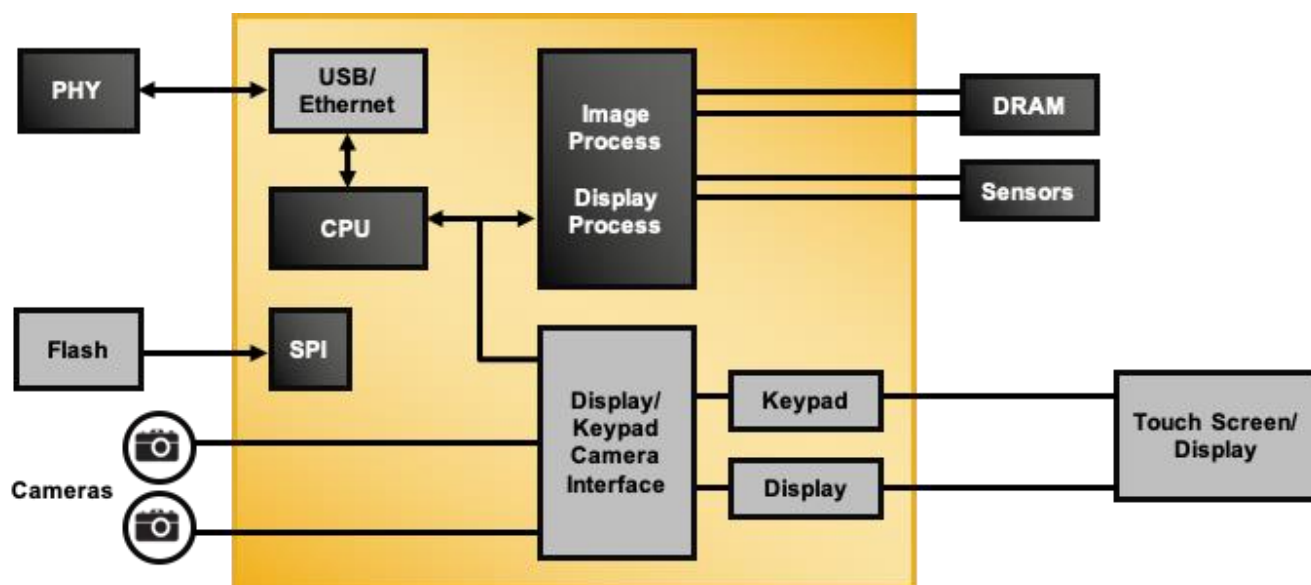


Рисунок 2.4 – Lattice Semiconductor CrossLink-NX

Завдяки своїм компактним розмірам, ці FPGA можуть бути використані у пристроях з обмеженим простором, таких як носимі пристрої або невеликі IoT пристрої. CrossLink-NX пропонує значну гнучкість у програмуванні, дозволяючи розробникам налаштовувати логіку для конкретних потреб їхніх додатків.

Ці FPGA підтримують різноманітні інтерфейси для відео, зображень та інших даних, що робить їх відмінним вибором для додатків, пов'язаних із обробкою та передачею візуальної інформації.

Галузі застосування CrossLink-NX досить різноманітні:

- використання в смартфонах та інших портативних пристроях для управління відео-інтерфейсами та забезпечення енергоефективності;
- інтеграція в промислове обладнання для управління процесами, моніторингу та збору даних;
- використання в автомобільних системах для управління сенсорами, камерами та іншими системами безпеки;
- впровадження в різні IoT пристрої для обробки даних, управління з'єднаннями та забезпечення енергоефективності.

2.2.5 AMD Xilinx Zynq UltraScale

Сімейство AMD Xilinx Zynq UltraScale представляє собою інноваційний набір мультипроцесорних систем на кристалі (MPSoC), які втілюють в собі унікальне поєднання традиційних процесорних ядер ARM і програмованих логічних пристроїв FPGA. Ця комбінація забезпечує високу продуктивність та адаптивність, що робить ці системи ідеальними для широкого спектру застосувань.

Основною особливістю сімейства Zynq UltraScale є їх гетерогенна обчислювальна архітектура. Вона включає потужні ядра ARM Cortex-A53 / Cortex-R5, які ефективно виконують загальні обчислювальні задачі та критично важливі системні функції. Водночас, інтегрована FPGA-логіка надає користувачам можливість кастомізувати апаратні рішення під специфічні завдання, що забезпечує надзвичайну гнучкість [19].

Важливою перевагою Zynq UltraScale є високий рівень інтеграції, що об'єднує CPU, FPGA, DSP та інші периферійні блоки в одному чіпі. Це сприяє не тільки компактності та ефективності рішень, але й підвищує загальну продуктивність системи. Крім того, сімейство підтримує широкий спектр інтерфейсів, включаючи Gigabit Ethernet, USB, PCIe, SATA, що робить ці системи дуже гнучкими у взаємодії з різноманітними зовнішніми системами.

Zynq UltraScale MPSoC також відомі своєю високою продуктивністю та енергоефективністю. Це особливо важливо для мобільних та вбудованих систем, де енергоспоживання є ключовим фактором. Водночас, гнучкість сімейства

дозволяє масштабувати рішення від невеликих до великих та вимогливих додатків, забезпечуючи широкий спектр можливостей для розробників.

Завдяки цим властивостям, Zynq UltraScale MPSoC знаходять широке застосування у промисловій автоматизації, автомобільній промисловості, телекомунікаціях, охороні здоров'я, а також у обробці зображень та відео. Вони дозволяють розробникам ефективно вирішувати складні технічні завдання, адаптуючи апаратні рішення під специфічні потреби їхніх проектів.

Кожен з цих аналогів має свої унікальні особливості та специфікації, що робить їх більш або менш підходящими для конкретних застосувань.

Для даної роботи було обрано AMD Xilinx Zynq UltraScale+ MPSoC ZCU104. AMD Xilinx Zynq UltraScale+ MPSoC ZCU104 представляє собою високопродуктивну розробницьку платформу, яка базується на інноваційній архітектурі Zynq UltraScale+ MPSoC. Ця плата є унікальною комбінацією процесорних ядер ARM і програмованої логіки FPGA, що забезпечує великі можливості для створення гнучких і потужних систем.

Основою ZCU104 є мультипроцесорна система на кристалі Zynq UltraScale+ MPSoC, яка включає ядра ARM Cortex-A53 та ARM Cortex-R5, об'єднуючи в собі потужні обчислювальні можливості з ефективним керуванням ресурсами. Це дозволяє розробникам використовувати ZCU104 для широкого спектру завдань, від загальних обчислень до високоспеціалізованих апаратних застосувань.

ZCU104 оснащена інтегрованою FPGA-логікою, що надає додаткову гнучкість та можливості кастомізації. Це дозволяє користувачам не просто програмувати систему під конкретні потреби, але й оптимізувати апаратну частину для специфічних аплікацій, таких як обробка зображень, машинне навчання або передача даних.

Завдяки своїм високопродуктивним характеристикам, ZCU104 знайшла застосування в різноманітних галузях, включаючи промислову автоматизацію, автомобільну промисловість, телекомунікації та інші. Її можливості роблять її

ідеальним рішенням для розробки вбудованих систем, де потрібна висока обчислювальна потужність, а також гнучкість та адаптивність до змінних вимог.

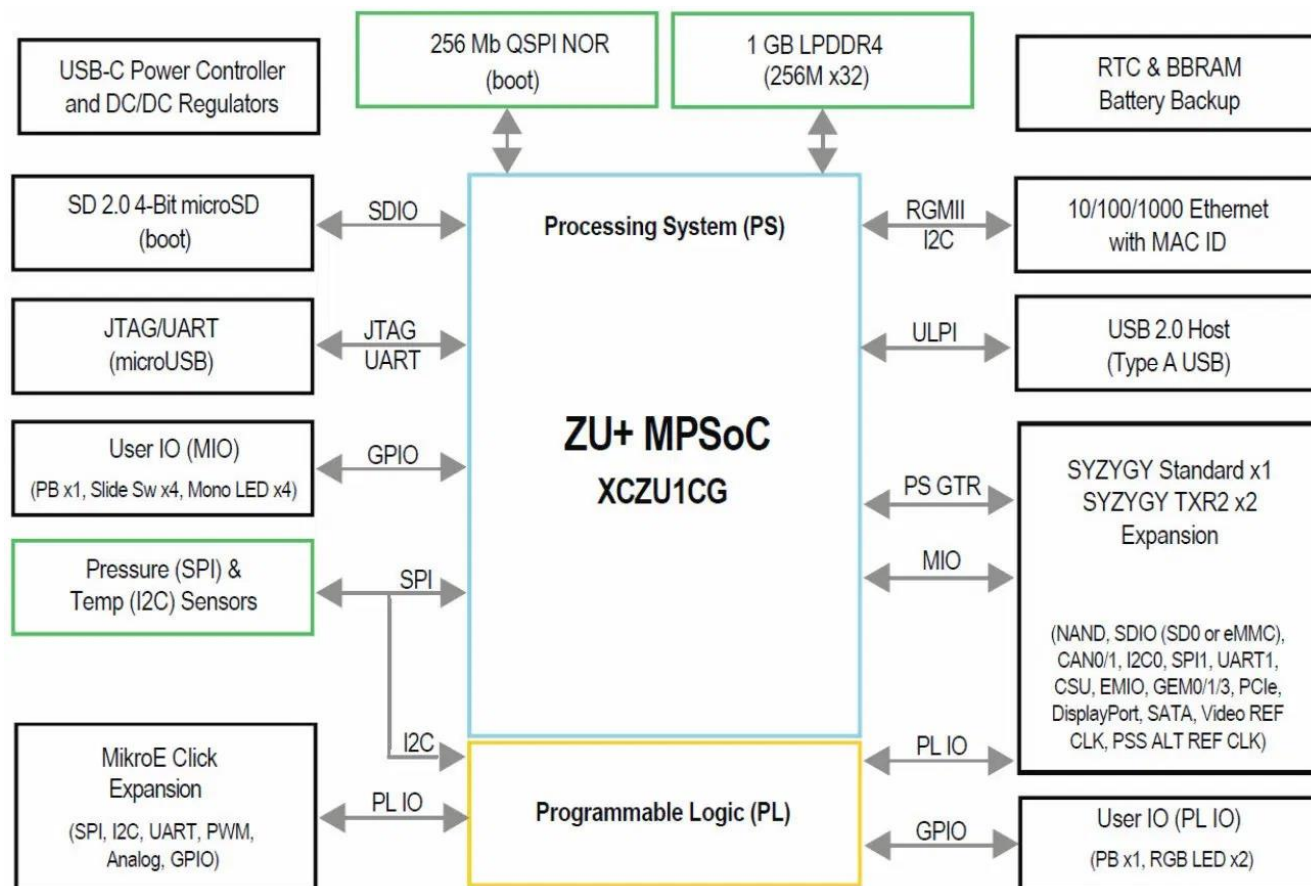


Рисунок 2.5 – AMD Xilinx Zynq UltraScale+ MPSoC ZCU104

2.2.6 Webcam Logitech® HD Pro

Webcam Logitech® HD Pro є високоякісною веб-камерою, яка відома своїм чітким зображенням і надійністю. Вона оснащена технологією Full HD, яка забезпечує відео роздільною здатністю 1080p, що робить її ідеальним вибором для відеоконференцій, стрімінгу та запису відео. Завдяки вбудованому мікрофону з функцією шумозаглушення, Logitech HD Pro також забезпечує високоякісний звук, що робить спілкування більш чітким та приємним [20].



Рисунок - Webcam Logitech® HD Pro

Камера має широкий кут огляду та автофокус, що дозволяє легко вмістити кілька людей у кадрі, а також забезпечує чітке зображення незалежно від відстані. Також, вона сумісна з більшістю платформ відеоконференцій та оперативних систем, що робить її універсальним рішенням для різних користувачів.

Основні характеристики камери наведені в табл. 2.1.

Таблиця 2.1 – Основны характеристики камери Logitech® HD Pro

Характеристика	Опис
Роздільна здатність відео	Full HD 1080p
Мікрофон	Вбудований, з функцією шумозаглушення
Кут огляду	Широкий
Автофокус	Так
Корекція освітлення	Автоматична корекція за умов слабого освітлення
Сумісність з ОС	Сумісна з більшістю платформ та операційних систем
Додаткові функції	Різні фільтри та ефекти, налаштування кадрування
Дизайн	Стильний, сучасний
Кріплення	Гнучке, підходить для моніторів, ноутбуків та штативів

Додатково, веб-камера Logitech HD Pro має стильний дизайн та легко кріпиться до різних поверхонь, включаючи монітори, ноутбуки та штативи, забезпечуючи гнучкість в розміщенні. Все це робить її відмінним вибором для професіоналів, стрімерів та будь-яких користувачів, які шукають якісну веб-камеру для повсякденного використання.

2.3 Програмне забезпечення

2.3.1 PetaLinux

PetaLinux представляє собою спеціалізовану версію Linux, розроблену та оптимізовану компанією Xilinx, яка тепер є частиною AMD після їхнього злиття у 2020 році. Ця система призначена для використання на програмованих логічних інтегральних схемах (FPGA) та системах на кристалі (SoC) виробництва Xilinx. PetaLinux відіграє ключову роль у спрощенні процесу розробки та реалізації програмного забезпечення для вбудованих систем, забезпечуючи при цьому гнучкість та ефективність у використанні ресурсів FPGA та SoC.

Історія PetaLinux тісно пов'язана з розвитком та еволюцією Xilinx, компанії, яка відома своїми новаторськими розробками у сфері програмованих логічних інтегральних схем. З моменту свого заснування, Xilinx була піонером у розробці FPGA та SoC, пропонуючи рішення, які постійно еволюціонують для задоволення зростаючих потреб ринку вбудованих систем. Злиття з AMD, одним із лідерів у галузі виробництва мікропроцесорів, відкрило нові горизонти для PetaLinux, забезпечуючи ще більше ресурсів та можливостей для розвитку та інтеграції з широким спектром технологій [21]

В основі PetaLinux лежить ядро Linux, яке адаптовано для оптимальної роботи з FPGA та SoC виробництва Xilinx. Це ядро надає стабільну та гнучку основу для вбудованих систем, дозволяючи розробникам налаштовувати його

відповідно до специфічних вимог їхніх проєктів. Ядро Linux у PetaLinux підтримує широкий спектр апаратного забезпечення, що дозволяє легко інтегрувати різні периферійні пристрої та інтерфейси.

PetaLinux включає гнучку файлову систему, що може бути налаштована для задоволення потреб вбудованих додатків. Вона підтримує різні формати файлових систем, включаючи ті, що оптимізовані для носіїв з обмеженою пам'яттю, таких як Flash-пам'ять. Крім того, PetaLinux забезпечує ефективне керування пам'яттю, оптимізуючи використання ресурсів FPGA та SoC для досягнення високої продуктивності системи.

Однією з ключових особливостей PetaLinux є його здатність інтегруватися з FPGA та SoC від Xilinx. Ця інтеграція включає підтримку специфічних для Xilinx функцій та інтерфейсів, таких як програмовані логічні блоки та конфігуруємі I/O порти. Розробники можуть використовувати PetaLinux для створення систем, що ефективно використовують можливості FPGA та SoC для обробки даних, управління сигналами та виконання складних обчислювальних задач.

Встановлення PetaLinux здійснюється на комп'ютер, який використовується для розробки та налаштування систем, базованих на FPGA та SoC від Xilinx.

Вимоги до системи:

Перш ніж встановлювати PetaLinux, переконайтеся, що ваш комп'ютер відповідає наступним вимогам:

- операційна система. Найчастіше PetaLinux встановлюють на Linux, наприклад, на версії Ubuntu або CentOS. Важливо мати сумісну версію Linux;
- апаратне забезпечення. Рекомендуються достатній обсяг оперативної пам'яті (наприклад, 8 ГБ або більше) та достатньо місця на жорсткому диску;
- залежності. Перед встановленням PetaLinux необхідно встановити певні залежності, такі як GCC, Make, Git та інші інструменти розробки.

Після встановлення PetaLinux можна розробляти, налаштовувати та розгортати вбудовані системи, що використовують FPGA або SoC від Xilinx, забезпечуючи високу гнучкість та ефективність у роботі.

2.3.2 AMD Vitis Unified Software Platform

AMD Vitis Unified Software Platform - це комплексний інструментарій, який розроблений для спрощення та прискорення розробки додатків на FPGA, SoC та інших апаратних платформах, що входять до портфеля AMD після їх злиття з Xilinx. Vitis забезпечує інтегроване середовище, яке дозволяє розробникам використовувати як високорівневі мови програмування, такі як C/C++ та Python, так і низькорівневі апаратні описові мови, такі як VHDL або Verilog [22].

AMD Vitis Unified Software Platform включає в себе різноманітні інструменти та компоненти, які разом створюють інтегроване середовище для розробки, оптимізації та розгортання додатків на FPGA, SoC та інших апаратних платформах Xilinx. Ключові інструментарії, які входять до складу Vitis:

- Vitis Core Development Kit. Підтримка мов C/C++ та Python для розробки додатків;
- Xilinx Runtime (XRT) - рантайм для взаємодії з апаратним забезпеченням Xilinx;
- набір API та драйверів для спрощення роботи з апаратним забезпеченням;
- Vitis Accelerated Libraries. Набір оптимізованих бібліотек для прискорення різних обчислювальних завдань, таких як обробка даних, машинне навчання, фінансові обчислення та інше;
- Vitis AI. Інструменти для розробки та оптимізації моделей машинного навчання для виконання на FPGA та SoC; сумісність з популярними фреймворками машинного навчання, такими як TensorFlow та PyTorch;
- Vitis Model Composer. Інструмент для візуального проектування, моделювання та аналізу систем на базі блок-схем, що спрощує розробку складних систем.
- Vitis HLS (High-Level Synthesis). Перетворює високорівневий код на низькорівневий апаратний опис, що дозволяє використовувати FPGA для апаратного прискорення без необхідності ручного опису апаратури.

– інтегроване середовище розробки (IDE), яке інтегрує усі інструменти та функції, необхідні для розробки, тестування та оптимізації додатків.

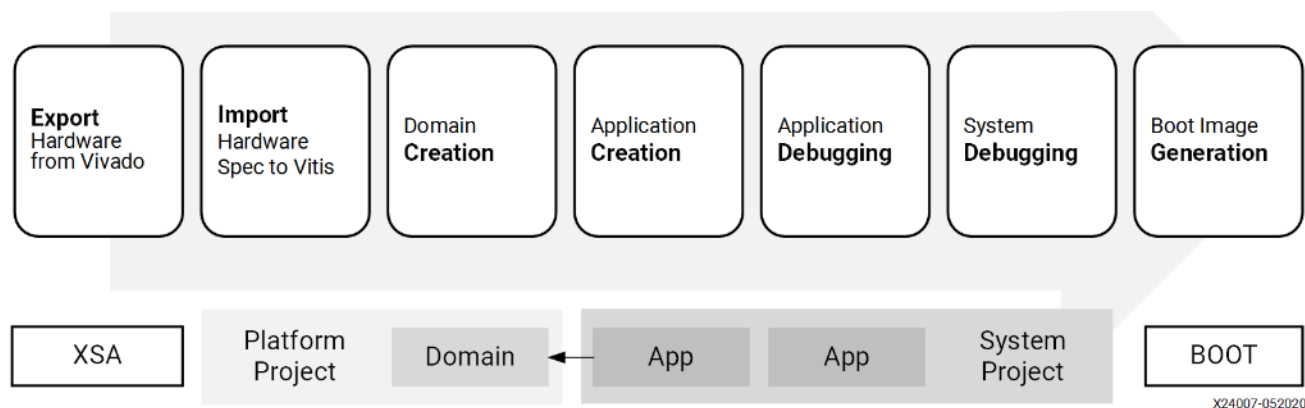


Рисунок 2.1 - Процес розробки вбудованого програмного забезпечення

У єдиному робочому просторі Vitis є два типи проектів, рис

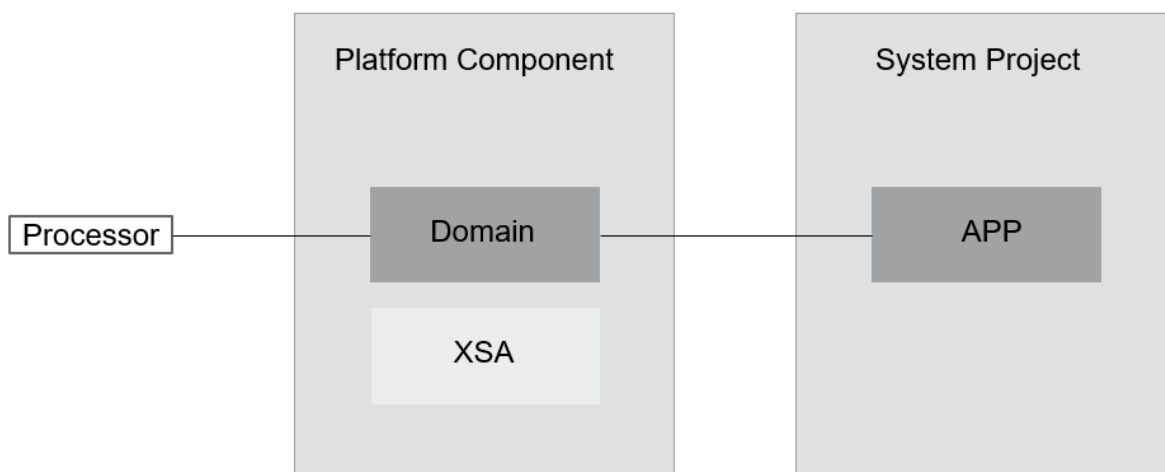


Рисунок 2.2 - Типи проектів програмної платформи Vitis

Workspace (Робочий простір). Коли ви відкриваєте уніфіковану програмну платформу Vitis, ви створюєте робочий простір. Робочий простір - це місцезнаходження директорії, яке використовується уніфікованою програмною платформою Vitis для зберігання даних проєкту та метаданих. Початкове

місцезнаходження робочого простору має бути надано під час запуску програмної платформи Vitis.

XSA. XSA експортуються з пакету проектування Vivado. Вони містять специфікації апаратного забезпечення, такі як властивості конфігурації процесора, інформація про підключення периферійних пристроїв, карта адрес та код ініціалізації пристрою. Вам необхідно надати XSA при створенні проєкту платформи.

Platform (Платформа). Цільова платформа або платформа - це комбінація апаратних компонентів (XSA) та програмних компонентів (домени/BSP, компоненти завантаження, такі як FSBL, і так далі). Платформи у репозиторії не можуть бути редаговані. Платформи у робочому просторі є редагованими і називаються компонентами платформи.

Platform Component (Компонент платформи). Компонент платформи - це проєкт у єдиному IDE Vitis для визначення платформи.

Домен. Домен - це пакет підтримки плати (BSP) або операційна система (ОС) з набором програмних драйверів, на основі яких будується ваш додаток. Створений програмний образ містить лише ті частини бібліотеки AMD, які ви використовуєте у вашому вбудованому дизайні. Ви можете створити кілька додатків для роботи на домені. Домен пов'язаний з одним процесором або кластером ізоморфних процесорів (наприклад, A53_0 або A53) у платформі.

Системний проєкт. Системний проєкт об'єднує додатки, які працюють одночасно на пристрої. Дві окремі програми для одного процесора не можуть бути в одному системному проєкті. Дві програми Linux можуть бути разом у системному проєкті. Робочий простір може містити кілька системних проєктів.

Додаток (Програмний проєкт). Програмний проєкт містить один або кілька вихідних файлів разом з необхідними файлами заголовків, що дозволяють компіляцію та створення двійкового вихідного файлу (ELF). Системний проєкт може містити кілька проєктів додатків. Кожен програмний проєкт повинен мати відповідний домен.

Платформа Vitis має різні конфігурації для підтримки різних випадків використання, які описані нижче:

- вбудовані платформи. Вбудовані платформи є фіксованими платформами. Ця платформа підтримує лише розробку вбудованого програмного забезпечення для процесорів Arm® та процесорів MicroBlaze™. Апаратний дизайн не може бути змінений у Vitis.

- вбудована акселерація. Окрім розробки вбудованого програмного забезпечення, також підтримується акселерація додатків на цьому типі платформи. Платформа забезпечує годинники, інтерфейси шини та контролери переривань для використання ядром акселерації.

- акселерація для центрів обробки даних. На цій платформі можна розробляти ядра акселерації та хост-додатки для x86. Ядро керується за допомогою шини PCIe.

Нижче наведено приклад типового робочого середовища розробки програмного забезпечення Vitis для AMD Zynq™ UltraScale+™ MPSoC.

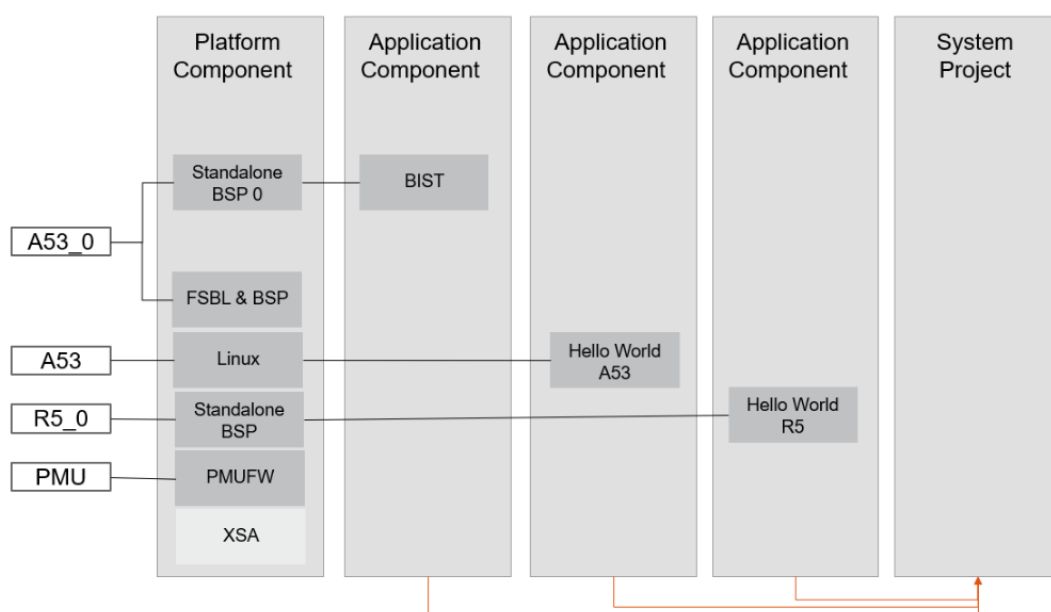


Рисунок 2.3 - Приклад робочого середовища розробки програмного забезпечення Vitis для Zynq UltraScale+ MPSoC

Домени Linux можна створювати для кластерів Arm® Cortex®-A53 SMP. Додатки Linux можна компілювати та зв'язувати з бібліотеками, які надає sysroot домену Linux.

Ядро Arm Cortex-A53 core 0 та ядро Arm Cortex®-R5F core 0 можуть одночасно запускати додаток "hello world". Ці два додатки можуть бути об'єднані в один системний проєкт.

Вбудований додаток для самотестування на ядрі Arm Cortex-A53 core 0 може працювати у своєму власному системному проєкті та мати власні налаштування BSP.

Цей системний проєкт слугує для управління кількома компонентами додатків. Додавання кількох компонентів додатків в один системний проєкт означає, що ці додатки працюватимуть одночасно. Системний проєкт не потрібен, якщо в один час працює лише один додаток.

Компоненти завантаження, такі як FSBL та прошивка PMU, можуть бути автоматично створені у проєктах платформи. Компоненти завантаження мають власні налаштування BSP.

2.3.3 Vivado Design Suite

Vivado Design Suite HLx Editions, за словами Xilinx, дозволяють застосувати підхід надвисокої продуктивності для проектування всіх програмованих систем на кристалі (SoCs), FPGA та створення багаторазових платформ. Видання HLx включають HL System Edition, HL Design Edition та HL WebPACK Edition. Усі видання HLx містять Vivado High-Level Synthesis (HLS), включно з бібліотеками C/C++, Vivado IP Integrator (IPI), підсистеми LogicCORE IP, а також повний набір інструментів реалізації Vivado, щоб основні користувачі могли легко прийняти найбільш продуктивні та передові потоки проектування на основі C та IP. У поєднанні з UltraFast High-Level Productivity Design Methodology Guide, Xilinx стверджує, що користувачі можуть досягнути збільшення продуктивності у 10-15 разів порівняно з традиційними методами. Видання HLx доступне як безкоштовне оновлення до Vivado Design Suite [23].

Продуктивність, як стверджує Xilinx, досягається завдяки всьому або підмножині наступного: 1) проектування на базі C та оптимізоване повторне використання, 2) повторне використання підсистем IP, 3) автоматизація інтеграції, та 4) прискорене закриття дизайну.

На відміну від традиційного RTL-базованого проектування, де більшість зусиль дизайнера витрачається на завершення процесу дизайну, проектування на основі C та IP дозволяє значно краще повторно використовувати дизайн для прискорення створення, швидке дослідження дизайну для кращих мікроархітектур, замінює схильні до помилок ручні конверсії C в RTL, усуває час та помилки під час інтеграції C та IP на основі RTL, та скорочує час верифікації. Використовуючи високі рівні абстракції, клієнти виявили, що вони можуть швидко отримати загалом кращі або еквівалентні результати якості (продуктивність, енергоефективність, використання ресурсів).

Для активації цих потоків високої продуктивності видання HLx включають Vivado HLS, Vivado IPI, підсистеми LogicCORE IP та повний набір інструментів для реалізації Vivado. До того ж, Xilinx та його екосистема Alliance постійно розширюють специфічні для ринку бібліотеки C, такі як OpenCV для відео та обробки зображень, а також Machine Learning для систем допомоги водіям автомобілів (ADAS) та додатків для дата-центрів. Нові підсистеми IP LogiCORE від Xilinx є високо конфігурованими, орієнтованими на ринок будівельними блоками, які інтегрують до 80 окремих IP-ядер, драйверів програм, прикладів дизайну та тестових стендів. Нові підсистеми IP доступні для Ethernet, PCIe®, обробки відео, обробки зображень датчиків і розробки OTN. Ці підсистеми IP базуються на промислових стандартах, таких як протокол міжпідключення AMBA® AXI 4, шифрування IEEE P1735 та IP-XACT, що забезпечують сумісність з IP Xilinx та членів Alliance, та прискорюють інтеграцію.

Комбінація IP на базі C і попередньо запакованих підсистем IP швидко поєднуються, використовуючи Vivado IP Integrator для автоматизації інтеграції. Автоматизація інтеграції Vivado IPI забезпечує пристрій та платформу з

інтерактивним середовищем, яке підтримує розумне автоматичне підключення ключових інтерфейсів IP, генерацію підсистем IP в один клік, реальні DRC та розповсюдження змін інтерфейсу, поєднане з потужною здатністю до налагодження. Інтелект, орієнтований на платформу, може попередньо налаштувати систему обробки Zynq® SoCs та MPSoCs із відповідними периферійними пристроями, драйверами та картою пам'яті для підтримки цільової плати. Тепер команди дизайнерів можуть швидко ідентифікувати, використовувати повторно та інтегрувати як програмне, так і апаратне IP, націлюючись на системи обробки ARM® та високопродуктивну FPGA логіку.

2.3.4 Xilinx Vitis AI

Vitis AI — це уніфікований стек рішень для інференції штучного інтелекту. Він включає специфічне для домену середовище розробки, а також прискорені бібліотеки Vitis. Ці бібліотеки охоплюють різні області, включаючи бібліотеку OpenCV для обробки зображень і відео, бібліотеку BLAS для наукових обчислень, а також фінансову бібліотеку.

Vitis AI також співпрацює з партнерами у сфері відео, геноміки, аналітики даних та інших областях. Це дозволяє використовувати розширений набір інструментів для різноманітних застосунків у сфері штучного інтелекту, підтримуючи широкий спектр вимог до обчислень та обробки даних [24].

Vitis AI від Xilinx — це комплексний стек рішень для інференції штучного інтелекту, який включає в себе декілька ключових компонентів:

Xilinx IR (Intermediate Representation). Представляє проміжне представлення моделей для подальшої оптимізації та компіляції. IR є ключовою частиною у процесі оптимізації та компіляції нейронних мереж для їх ефективного використання на апаратному забезпеченні. Воно діє як міст між високорівневими моделями машинного навчання (наприклад, створеними в TensorFlow або PyTorch) та низькорівневим кодом, необхідним для виконання на специфічному апаратному забезпеченні.

Використання IR дозволяє оптимізувати моделі для кращої продуктивності та ефективності, враховуючи особливості цільових платформ, таких як FPGA або ASIC. Це включає оптимізацію розмірів моделей, швидкості обчислень та енергоефективності.

Xilinx Compiler: Компілятор, який перетворює проміжне представлення моделей у формат, оптимізований для виконання на обладнанні Xilinx.

Vitis AI Compiler є частиною платформи Vitis AI від Xilinx, яка спрямована на прискорення роботи з штучним інтелектом на FPGA (Field-Programmable Gate Arrays) та інших апаратних платформах. Цей компілятор використовується для перетворення нейронних мереж, створених за допомогою популярних фреймворків машинного навчання, таких як TensorFlow або PyTorch, у оптимізований для апаратного забезпечення формат.

Основні характеристики та функції Vitis AI Compiler включають:

- підтримку різних фреймворків машинного навчання: Компілятор може обробляти моделі, створені в різних фреймворках, таких як TensorFlow, PyTorch, Caffe тощо;
- оптимізація моделей - компілятор оптимізує моделі для конкретного апаратного забезпечення, забезпечуючи підвищену продуктивність та ефективність;
- перетворення моделей у спеціалізований формат - після обробки моделі перетворюються у формат, сумісний з FPGA або іншим цільовим обчислювальним пристроєм;
- сумісність з FPGA. Оскільки FPGA мають унікальні характеристики та можливості перепрограмування, Vitis AI Compiler спеціалізується на оптимізації моделей саме під ці пристрої;
- підвищення ефективності та швидкості виконання. Використання компілятора дозволяє значно підвищити швидкість виконання нейронних мереж на апаратному забезпеченні, порівняно з традиційними CPU або GPU.

Vitis AI Compiler є ключовим інструментом для інженерів та розробників, які хочуть максимально використати переваги FPGA для розгортання моделей машинного навчання, особливо в сферах, де потрібна висока швидкість обробки даних та енергоефективність.

Xilinx Embedded Software: Вбудоване програмне забезпечення, яке дозволяє інтегрувати моделі штучного інтелекту з апаратним забезпеченням Xilinx.

Xilinx Model Zoo: Набір готових моделей інференції, які можна використовувати з обладнанням Xilinx. Цей "зоопарк моделей" включає як публічні, так і користувацькі моделі, які оптимізовані для роботи на Xilinx пристроях. Xilinx Model Zoo є частиною ширшої екосистеми Vitis AI, розробленої Xilinx. Ця збірка попередньо навчених моделей глибокого навчання служить як ресурс для розробників, які шукають швидкого та ефективного способу впровадження складних нейронних мереж в апаратне забезпечення, таке як FPGA та АСАР. Ці моделі включають різноманітні застосування, від класифікації зображень до обробки мови, і були оптимізовані для ефективного виконання на апаратному рівні. Вони сумісні з різними фреймворками машинного навчання, що робить їх гнучкими у використанні.

Те, що моделі вже попередньо навчені, істотно знижує поріг входження для розробників, що бажають використовувати нейронні мережі, особливо в умовах, коли вони працюють з обмеженими ресурсами або потребують швидкого впровадження. Крім того, детальна документація та підтримка від Xilinx сприяють більш глибокому розумінню та ефективному використанню цих моделей.

AI Parser та AI Quantizer. Інструменти для аналізу та квантування моделей, відповідно. Вони дозволяють оптимізувати моделі для кращої продуктивності на апаратному рівні. AI Parser та AI Quantizer є ключовими компонентами платформи Vitis AI від Xilinx, кожен з яких виконує унікальну роль у процесі адаптації та оптимізації нейронних мереж для апаратного забезпечення, такого як FPGA.

AI Parser служить для аналізу та перетворення моделей глибокого навчання, створених у стандартних фреймворках, як-от TensorFlow або PyTorch. Цей

інструмент аналізує структуру моделі, її шари та параметри, готуючи її до подальшої оптимізації та компіляції. Це важливо для забезпечення того, що модель може бути ефективно використана на специфічному апаратному забезпеченні.

AI Quantizer, з іншого боку, займається процесом квантизації моделі. Квантизація важлива для оптимізації продуктивності моделі на апаратному рівні, оскільки вона зменшує обсяг вимаганих обчислень та пам'яті без значної втрати точності. Це досягається шляхом зменшення точності чисел, які використовуються в моделі, з плаваючої коми до фіксованої.

Обидва інструменти разом створюють ефективний механізм для підготовки складних нейронних мереж до роботи на FPGA, дозволяючи розробникам використовувати силу машинного навчання з підвищеною продуктивністю та ефективністю. Вони відіграють вирішальну роль у забезпеченні гнучкості та швидкості в розгортанні моделей машинного навчання на різноманітному апаратному забезпеченні.

AI Compiler. Компілятор для перетворення оптимізованих моделей у виконуваний код, сумісний з обладнанням Xilinx. AI Compiler від Xilinx є важливим інструментом у процесі перетворення та оптимізації моделей машинного навчання для використання на специфічному апаратному забезпеченні, такому як FPGA. Цей компілятор працює після етапів аналізу та квантизації моделі, що виконуються іншими інструментами платформи Vitis AI, і зосереджений на перетворенні оптимізованої моделі у формат, який ефективно працює на апаратному рівні.

Цей процес включає перетворення високорівневих абстракцій та структур моделі машинного навчання на низькорівневі інструкції, які можуть бути безпосередньо виконані на FPGA. AI Compiler забезпечує, що це перетворення відбувається з максимальною ефективністю, забезпечуючи оптимальне використання ресурсів апаратного забезпечення та знижуючи час виконання.

Цей компілятор також враховує унікальні особливості апаратного забезпечення, такі як обмеження пам'яті та обчислювальні можливості, щоб забезпечити, що моделі машинного навчання працюють не тільки точно, але й

ефективно. Використання AI Compiler є ключовим кроком у розгортанні складних нейронних мереж, дозволяючи розробникам використовувати потужність FPGA для машинного навчання, забезпечуючи при цьому високу продуктивність і енергоефективність.

AI Library та AI Runtime. Бібліотеки та середовище виконання для підтримки інференції штучного інтелекту на обладнанні Xilinx, забезпечуючи швидке та ефективне виконання моделей у реальному часі.

AI Library та AI Runtime від Xilinx є ключовими компонентами у екосистемі Vitis AI, спрямованими на спрощення та ефективізацію розгортання моделей машинного навчання на апаратному забезпеченні, такому як FPGA. AI Library, як частина цієї платформи, надає бібліотеку оптимізованих функцій та алгоритмів, які можуть бути використані для швидкого розгортання та виконання різноманітних завдань машинного навчання, таких як класифікація зображень, виявлення об'єктів, аналіз мови тощо. Ці функції та алгоритми розроблені для того, щоб максимально використовувати переваги апаратного забезпечення, як-от FPGA, забезпечуючи при цьому високу продуктивність та ефективність.

AI Runtime, з іншого боку, забезпечує середовище для виконання цих оптимізованих моделей на апаратному забезпеченні. Воно відіграє роль містка між моделями, скомпільованими та оптимізованими за допомогою інших інструментів Vitis AI, та апаратним забезпеченням, на якому ці моделі будуть виконуватися. AI Runtime забезпечує управління ресурсами, розподіл роботи та інші критичні аспекти виконання, що дозволяє моделям працювати ефективно та безперебійно.

Разом AI Library та AI Runtime формують ефективну та масштабовану платформу для розгортання та виконання моделей машинного навчання, використовуючи спеціалізоване апаратне забезпечення. Це значно спрощує процес розгортання для розробників, дозволяючи їм концентруватися на розвитку та оптимізації своїх моделей, замість вирішення апаратних та інфраструктурних завдань.

2.3.5 OpenCV (Open Source Computer Vision Library)

OpenCV (Open Source Computer Vision Library) - це бібліотека програмного забезпечення, відкритого коду, яка містить численні алгоритми для обробки зображень та комп'ютерного зору. Ця бібліотека широко використовується у різних областях, включаючи розпізнавання облич, автоматичне виявлення об'єктів, обробку відео, робототехніку та навіть у розробці інтерактивних художніх проєктів [25].

OpenCV була спочатку розроблена Intel у 1999 році та випущена як відкритий проєкт у 2000 році. З того часу вона стала однією з найпопулярніших бібліотек у світі для комп'ютерного зору. Однією з основних причин її популярності є те, що вона написана на C/C++, пропонує інтерфейси для Python, Java та інших мов, і підтримує різні операційні системи, включаючи Windows, Linux та macOS.

Бібліотека OpenCV включає в себе тисячі оптимізованих алгоритмів, які можна використовувати для вирішення різних задач обробки зображень та комп'ютерного зору. Це дозволяє розробникам швидко реалізовувати складні функціональності, пов'язані з зором, у своїх додатках без необхідності глибокого занурення у деталі алгоритмів та обчислювальні теорії.

Завдяки своїй відкритості, гнучкості та широкому спектру можливостей, OpenCV є важливим інструментом у сучасному світі цифрових технологій, зокрема в областях, де потрібно швидке та ефективне використання комп'ютерного зору.

2.3.6 Docker

Docker - це популярна платформа для розробки, доставки та запуску додатків за допомогою технології контейнеризації. Контейнеризація дозволяє упаковувати додаток із усіма його залежностями та налаштуваннями в стандартизований блок, відомий як контейнер. Це сприяє легшому розгортанню та більшій надійності додатків, оскільки контейнери працюють однаково незалежно від середовища [26].

Основні аспекти Docker:

- ізоляція ресурсів. Кожен Docker контейнер ізольований і має свої ресурси, включаючи файлову систему, CPU, пам'ять та процеси вводу-виводу, що забезпечує консистентність у роботі додатків;
- легкість та швидкість. Контейнери Docker використовують спільну операційну систему хоста, що робить їх значно легшими та швидшими порівняно з традиційними віртуальними машинами, які потребують окремої ОС;
- портативність. Оскільки контейнери включають у себе все необхідне для роботи додатка, вони можуть бути легко переміщені між різними середовищами, такими як розробка, тестування та продакшн;
- масштабованість та керування. Docker спрощує процес масштабування додатків, дозволяючи легко створювати, запускати та керувати численними контейнерами;
- екосистема та спільнота. Docker має велику спільноту та екосистему, яка включає безліч готових до використання образів контейнерів, доступних через Docker Hub.

Docker знайшов широке застосування у розробці програмного забезпечення, оскільки він дозволяє розробникам та системним адміністраторам з легкістю створювати, розгортати та управляти додатками в контейнеризованих середовищах, сприяючи більшій ефективності та гнучкості в процесах розробки та розгортання.

2.4 Висновки до розділу 2

Цей розділ охоплює глибоке дослідження та аналіз різних аспектів програмного забезпечення в контексті вбудованих систем, зокрема зосереджуючись на технологіях та платформах, розроблених Xilinx (тепер частиною AMD). Описується PetaLinux, спеціалізована версія Linux для FPGA та

SoC, яка спрощує процес розробки для вбудованих систем, надаючи гнучкість та ефективність у використанні ресурсів.

Далі йдеться про AMD Vitis Unified Software Platform, яка є комплексним інструментом для розробки додатків на FPGA, SoC та інших апаратних платформах. Ця платформа об'єднує в собі різні інструменти та компоненти, що дозволяє розробникам використовувати як високо-, так і низькорівневі мови програмування, забезпечуючи інтегроване середовище для оптимізації та розгортання додатків.

В розділі також розглядається Vitis AI - комплексний стек рішень для інференції штучного інтелекту, що включає інструменти для розробки та оптимізації моделей машинного навчання для виконання на FPGA та SoC. Висвітлюються також AI Parser та AI Quantizer, інструменти для аналізу та квантизації моделей, які допомагають оптимізувати моделі для кращої продуктивності на апаратному рівні.

Окрім цього, розділ описує OpenCV, бібліотеку програмного забезпечення відкритого коду для обробки зображень та комп'ютерного зору, та Docker, платформу для розробки, доставки та запуску додатків з використанням технології контейнеризації. Кожен з цих елементів грає важливу роль у розвитку та оптимізації вбудованих систем, забезпечуючи гнучкість, ефективність та швидкість розробки та розгортання додатків.

3 РОЗРОБКА СИСТЕМИ

3.1 Налаштування середовища для плати ZCU104

Для проєкту було обрано плату ZCU104. На рис. 3.1 показано підключення до плати периферійних пристроїв.

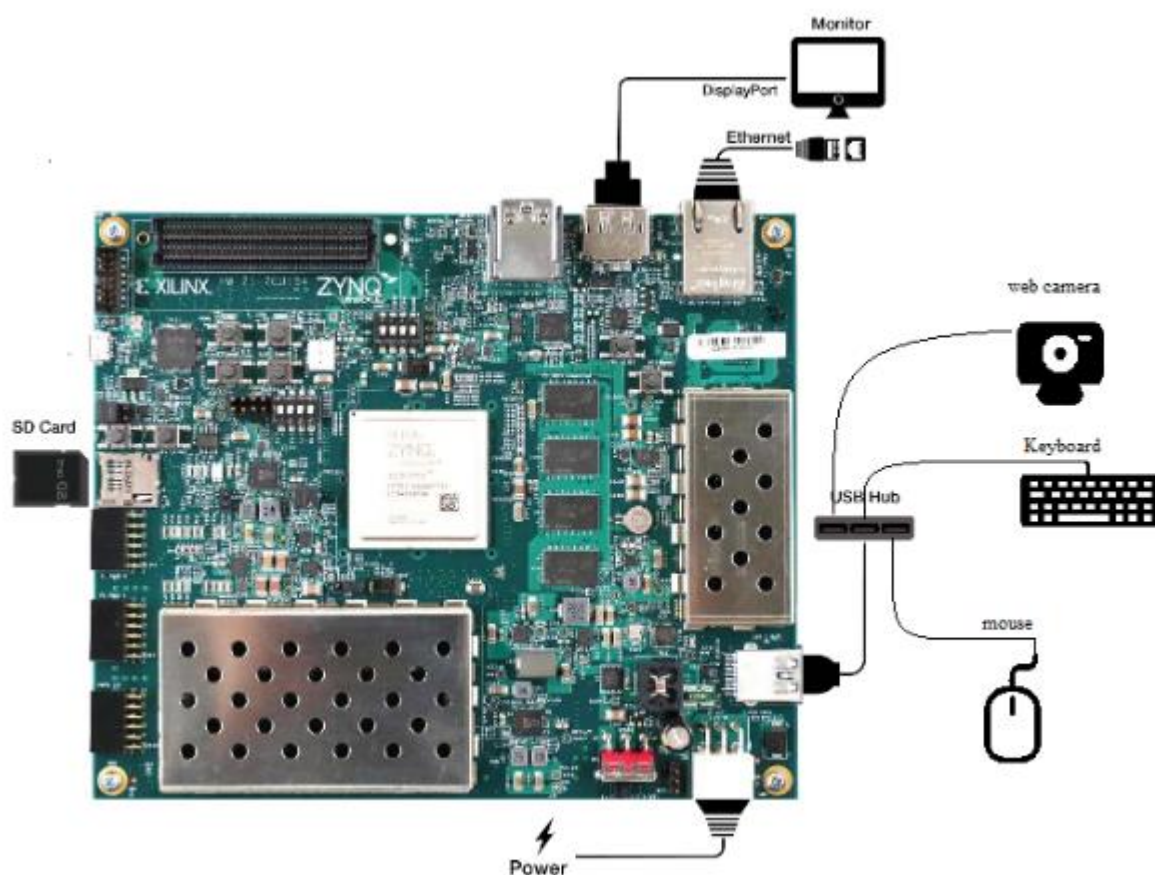


Рисунок 3.1 – Підключення периферійних пристроїв до плати ZCU104

Проект передбачає використання великої кореневої файлової системи. У Vitis 2020.1 може виникати проблема з формуванням образів SD-карток із розділами ext4 об'ємом понад 2 ГБ, тому необхідно застосувати патч, який модифікує процес упаковки, округляючи початковий розмір кореневої файлової системи до першого повного кратного - 512 МБ, що перевищує розмір розділу ext4. Для його встановлення необхідно клонувати повний репозиторій (включно з підмодулями) (ліст. 3.1):

Лістинг 3.1 – Застосування патчу

```
$ cp ./vitis_patch/mkfsImage.sh ${XILINX_VITIS}/scripts/vitis/util
```

Далі підключаємо необхідні джерела (ліст. 3.2)

Лістинг 3.2 – Підключення джерел

```
$ source ${XILINX_VITIS}/settings64.sh
$ source ${XILINX_PetaLinux}/settings.sh
$ source ${XILINX_XRT}/setup.sh
```

Створюємо апаратну платформу (ліст. 3.3)

Лістинг 3.3 – Створення платформи

```
$ cd platform/dev/zcu104_vcu
$ make
```

Створюємо проєкт Vitis (додаємо DPU IP) (ліст. 3.4). DPU (Data Processing Unit) IP відноситься до інтегрального модуля обробки даних, який використовується в області програмованої логіки. "IP" у цьому контексті означає "Intellectual Property" (інтелектуальну власність), що вказує на те, що DPU є конструктивним рішенням або архітектурою, яка розроблена та захищена як інтелектуальна власність компанії або індивіда.

DPU IP часто використовують для спеціалізованої обробки даних, особливо в областях, де потрібна висока продуктивність або ефективність обробки, таких як штучний інтелект, машинне навчання, обробка зображень чи відео. Вони можуть бути інтегровані в більш широкі системи, такі як системи на кристалі (SoC) або інші програмовані пристрої для забезпечення підвищеної продуктивності для специфічних задач обробки даних.

Лістинг 3.4 – Використання DPU IP

```
$ cd ../../../../hw_src/Vitis_Libraries
$ patch -p1 < ../vision_lib_area_resize_ii_fix.patch
```

```
$ cp ../../../../../../dpu_conf_zcu104.vh .
$ cp ../../../../../../zcu10x_config .
$ make
```

Після виконання вищезазначених кроків отримуємо образ SD-карти `sd_card.img` у `sd_card_zcu104/`.

3.2 Компіляція Vitis-AI

Створюємо середовище Vitis-AI, дотримуючись кроків Vitis-AI, щоб створити Docker контейнер: `./docker_run.sh xilinx/vitis-ai-gpu:latest`

Docker контейнер - це стандартизована упаковка програмного забезпечення, яка включає в себе все необхідне для його запуску: код, системні бібліотеки, середовище виконання, інструменти та інші файли. Контейнери відрізняються від традиційних віртуальних машин тим, що вони не вимагають окремої операційної системи для кожного контейнера. Замість цього, вони ділять ядро операційної системи хоста та ізолюють програмне забезпечення в контейнері від інших процесів на цьому хості.

Це робить Docker контейнери дуже швидкими в порівнянні з віртуальними машинами. Вони забезпечують узгодженість середовища від розробки до виробництва, полегшують масштабування та розгортання програм, а також сприяють вищій ефективності використання ресурсів системи.

Завантажуємо моделі tensorflow із Xilinx AI Model Zoo. Xilinx AI Model Zoo (зоопарк моделей штучного інтелекту Xilinx) - це колекція предтренованих моделей глибокого навчання, оптимізованих для використання на пристроях Xilinx, зокрема на FPGA (полегшено програмованих вентильних матрицях) та SoC

(системах на кристалі). Ці моделі покривають широкий спектр застосувань, від обробки зображень та відео до аналізу мови та звуку.

Ключові особливості Xilinx AI Model Zoo:

- моделі вже навчені на великих датасетах і готові до використання, що економить час та ресурси;
- моделі оптимізовані для максимальної продуктивності на обладнанні Xilinx, зокрема, вони адаптовані до особливостей обчислювальної потужності та архітектури FPGA та SoC;
- моделі включають різні типи нейронних мереж, призначені для вирішення завдань класифікації, детекції об'єктів, сегментації зображень, аналізу тексту та інших задач;
- завдяки наявності предтренованих моделей, розробники можуть швидко інтегрувати рішення штучного інтелекту у свої продукти або системи.

Моделі можна отримати, дотримуючись інструкцій на Xilinx/Vitis-AI/AI-Model-Zoo.

В цьому проекті використано `xilinx_model_sample/tf/ssdmobilenetv2_coco_300_300_3.75G` як модель виявлення об'єктів.

`Xilinx_model_sample/tf/ssdmobilenetv2_coco_300_300_3.75G` відноситься до моделі машинного навчання, яка оптимізована для використання на платформах або обладнанні від Xilinx:

- `tf_ssdmobilenetv2_coco` - означає, що модель базується на архітектурі SSD (Single Shot Multibox Detector) з мережею MobileNetV2 як основою. SSD - це популярний алгоритм для виявлення об'єктів, який здатний виявляти кілька об'єктів в одному кадрі. MobileNetV2 - це ефективна архітектура нейронної мережі для мобільних та вбудованих візуальних застосунків. Суфікс "coco" означає, що модель була навчена на датасеті COCO (Common Objects in Context), який є одним із стандартних наборів даних для завдань комп'ютерного зору;
- `300_300` - ці числа вказують на розмір вхідного зображення для моделі, тобто 300x300 пікселів;

– 3.75G - це означає загальну кількість операцій, які модель виконує при обробці одного зображення, вимірювану в гігаопераціях (GOPS). Цей показник дає уявлення про складність та вимоги до обчислювальних ресурсів для моделі.

Така модель може бути використана в застосунках, що вимагають виявлення об'єктів у реальному часі, таких як системи відеоспостереження, автономні транспортні засоби, аналіз зображень з дронів тощо.

Копіюємо `xilinx_model_sample/tf_ssdmobilenetv2_coco_300_300_3.75G/quantized/deploy_model.pb` на `host/ssd_mobilenet/`.

Компілюємо модель тензорного потоку. Після виконання `./ssdmobilenet_compile_b4096.sh` отримуємо `dpu_ssd_mobilenet_v2_coco_tf.elf` (Файл з розширенням `.elf` (Executable and Linkable Format) - це стандартний формат бінарного файлу в операційних системах на базі Unix та Linux. Файли `.elf` використовуються для зберігання скомпільованих виконавчих програм, об'єктних кодів, динамічних бібліотек та інших форм бінарних даних).

3.3 Крос-компіляція коду виведення DPU

Налаштовуємо хост, виконуючи кроки в Xilinx/Vitis-AI-Library.

Далі виконуємо крос-компіляцію. Для цього запускаємо `./build_final.sh` і отримуємо скомпільований файл `tfssctest.so` зі спільним об'єктом ELF 64-bit LSB, формат ARM aarch64.

Код, який виконує реалізацію системи розпізнавання об'єктів з використанням камери поданий в додатку А (ліст. А.1). Він включає завантаження зовнішньої бібліотеки для обробки зображень, встановлення з'єднання з камерою, обробку кадрів з розпізнаванням об'єктів, накладання рамок на розпізнані об'єкти та відображення результатів на екрані. Завершується код підрахунком кадрів в секунду (FPS) та середнім часом виклику функції DPU.

В лістингу A.2 (додаток A) функція `detect` призначена для виклику зовнішньої бібліотеки C для обробки зображень. Вона передає дані кадру у форматі, сумісному з C, і отримує результати, такі як координати та класифікації виявлених об'єктів. Масив `data` використовується для зберігання цих результатів, а `size` повертає кількість виявлених об'єктів або іншу важливу інформацію, пов'язану з результатами обробки.

Клас `ThreadedCapture` в лістингу A.3 (додаток A) організовує читання відео або потоку з камери у фоновому потоці. Це дозволяє зменшити затримки у основному потоці програми, забезпечуючи більш плавне читання кадрів. Потокове читання кадрів виконується в методі `update`, а метод `read` повертає останній прочитаний кадр.

Клас `WindowSystem` відповідає за створення та управління графічним інтерфейсом користувача. Він включає методи для відображення зображень, тексту та інформації про виявлені об'єкти, а також для показу зображення з камери. Клас також дозволяє динамічно оновлювати вміст інтерфейсу, включаючи список розпізнаних об'єктів і їхніх цін.

Код (ліст. A.4, додаток A) виконує обробку відео з камери, детектування об'єктів і відображення інформації про них у графічному інтерфейсі користувача. Кожен об'єкт, який виявлено, відображається з відповідною назвою та кількістю. Інтерфейс оновлюється кожну секунду для відображення актуальної інформації. Код також враховує загальний час обробки та кількість кадрів для розрахунку середнього часу обробки та кадрів за секунду (FPS).

3.4 Висновки до розділу 3

У цьому розділі дипломної роботи описується процес налаштування середовища для плати ZCU104, що включає в себе декілька ключових етапів.

Спочатку йдеться про вирішення проблеми з формуванням образів SD-карток великого об'єму в Vitis 2020.1, яке вимагає застосування спеціального патчу. Далі, описується процес підключення необхідних джерел та створення апаратної платформи, включаючи додавання DPU IP, який є важливою частиною для спеціалізованої обробки даних.

Процедура підготовки SD-карти для Vitis AI Library v1.2 включає розміщення певних файлів та каталогів у відповідних директоріях. Розмір кореневої файлової системи змінюється для забезпечення оптимальної роботи. Далі йдеться про встановлення залежностей та бібліотеки Vitis AI v1.2, яке потребує доступу до Інтернету для завантаження необхідних компонентів за допомогою wget.

Процес компіляції Vitis-AI включає створення Docker контейнера та завантаження моделей tensorflow із Xilinx AI Model Zoo. Використовується модель `xilinx_model_sample/tf_ssdmobilenetv2_coco_300_300_3.75G` для виявлення об'єктів. Компіляція та крос-компіляція моделі та коду виведення DPU відбувається з використанням відповідних скриптів та інструментів.

ВИСНОВКИ

Касові системи є неодмінною частиною фінансової інфраструктури бізнесу, забезпечуючи точність і легальність фінансових операцій. Кожен тип має свої особливості та переваги, відповідно до конкретних потреб галузі та формату обслуговування клієнтів. Системи самообслуговування стають все більш популярними завдяки їх здатності оптимізувати процеси обслуговування і зменшувати час очікування.

В цій роботі проведена спроба створення касової системи самообслуговування, реалізація якої базується на FPGA. За рахунок підключеної камери і застосуванню алгоритму розпізнавання зображень система має визначати товар та його ціну. Відео товару може бути отримано з камери, хмари або іншого джерела (наприклад, SD карти).

В роботі розглянуто апаратне забезпечення, за допомогою якого можливо було б реалізувати проєкт. На основі аналізу апаратних платформ було обрано AMD Xilinx Zynq UltraScale+MPSoC ZCU104, яка представляє собою високопродуктивну розробницьку платформу. Ця плата є унікальною комбінацією процесорних ядер ARM і програмованої логіки FPGA, що забезпечує великі можливості для створення гнучких і потужних систем.

Як модель виявлення об'єктів в магістерській роботі використано Xilinx AI Model Zoo. Це моделі вже навчені на великих датасетах і готові до використання.

В процесі роботи системи ми маємо завантаження зовнішньої бібліотеки для обробки зображень, встановлення з'єднання з камерою, обробку кадрів з розпізнаванням об'єктів, накладання рамок на розпізнані об'єкти та відображення інформації про них у графічному інтерфейсі користувача. Кожен об'єкт, який виявлено, відображається з відповідною назвою та кількістю.

Цей проєкт ще не є повноцінним заміном касової системи, але легко може бути розширений і адаптований під специфічні товари і завдання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Одюшина Л.В., Сосинстка І.В. Виконання етапів підготовки залу до роботи та процесу обслуговування відвідувачів. Електронний посібник. URL: <https://sites.google.com/view/odiushuna> (дата звернення 04/11/2023).
2. Бетлій О. Застосування реєстраторів розрахункових операцій: міжнародний досвід – уроки для України URL: https://pdf.usaid.gov/pdf_docs/PA00N5CQ.pdf (дата звернення 04/11/2023).
3. Нове рішення для бізнесу: пристроїв менше, а користі більше — для чого потрібна Смарт каса і як її вибрати URL: <https://www.telegraf.in.ua/advertisement/10100144-pristrojiv-menshe-a-koristi-bilshe-dlja-chogo-biznesu-smart-kasa-i-jak-jiji-vibrati.html> (дата звернення 04/11/2023).
4. Каси самообслуговування (Self-Checkout) URL: <https://www.ics-market.com/kasi-samoobslugovuvannya-self-checkout/> (дата звернення 04/11/2023).
5. Рубцова М. Ю. Хмарні технології як інструмент поглиблення віртуалізації фінансового сектор // «Ефективна економіка», №5, 2020. DOI: 10.32702/2307-2105-2020.5.110
6. Топ 10 Українських хмарних сервісів для бізнесу URL: <https://www.livebusiness.com.ua/ua/tools/saasua/> (дата звернення 04/11/2023).
7. Праїз Г. Найкраща POS-система для малого бізнесу: Найкращі POS-системи для малого бізнесу в 2023 році URL: <https://businessyield.com/uk/business-services/best-pos-system-for-small-business/> (дата звернення 04/11/2023).
8. POS & Payments platform URL: <https://www.lightspeedhq.com/> (дата звернення 04/11/2023).
9. Shopify URL: <https://www.shopify.com/> (дата звернення 04/11/2023).
10. Zettle URL: <https://www.zettle.com/gb> (дата звернення 04/11/2023).
11. Sumup URL: <https://www.sumup.com/> (дата звернення 04/11/2023).

12. Що таке ERP і як отримати максимум від впровадження системи?
URL: <https://bc.smart-it.com/news-and-articles/what-is-erp/> (дата звернення 04/11/2023).

13. Сіріус. Центр автоматизації URL: <https://sirius.if.ua/> (дата звернення 04/11/2023).

14. Найдьонов М. Ю., Грушко С. С., Куликовська Н. А., Тіменко А. В. Інтеграція великих даних у технологію інтернет речей // Збірник матеріалів XIV Міжнародної науково-практичної конференції молодих вчених «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ: ЕКОНОМІКА, ТЕХНІКА, ОСВІТА '2023», 26–27 жовтня 2023 року, НУБіП України, Київ. – 250 с. (електронне видання) с. 157 – 158.

15. Intel. Intel® Stratix® 10 FPGA and SoC FPGA [Електронний ресурс]. – Режим доступу: <https://www.intel.com/content/www/us/en/products/details/fpga/stratix/10.html> (Дата звернення: 05.12.2023).

16. Intel. Intel® Arria® 10 FPGA and SoC FPGA [Електронний ресурс]. – Режим доступу: <https://www.intel.com/content/www/us/en/products/details/fpga/arria/10.html> (дата звернення: 05.12.2023).

17. Microchip Technology Inc. PolarFire® SoC FPGAs [Електронний ресурс]. – Режим доступу: <https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/polarfire-soc-fpgas> (дата звернення: 05.12.2023).

18. Lattice Semiconductor. CrossLink-NX Embedded Vision and Processing FPGA [Електронний ресурс]. – Режим доступу: <https://www.latticesemi.com/Products/FPGAandCPLD/CrossLink-NX> (дата звернення: 05.12.2023).

19. Xilinx. Zynq™ UltraScale+™ MPSoC Heterogeneous Multiprocessing Platform for Broad Range of Embedded Applications [Електронний ресурс]. – Режим доступу: <https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html> (дата звернення: 05.12.2023).

20. Logitech. WEB-камера Logitech HD Pro Webcam C920 [Электронный ресурс]. – Режим доступа: <https://ek.ua/ua/LOGITECH-HD-PRO-WEBCAM-C920.htm> (дата звернення: 05.12.2023).

21. Xilinx. PetaLinux Tools [Электронный ресурс]. – Режим доступа: <https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html> (дата звернення: 05.12.2023).

22. Xilinx. The Vitis Software Platform Development Environment [Электронный ресурс]. – Режим доступа: <https://www.xilinx.com/products/design-tools/vitis.html> (дата звернення: 05.12.2023).

23. Xilinx. Overview: Enabling faster design iterations and quickly meeting your FMAX targets [Электронный ресурс]. – Режим доступа: <https://www.xilinx.com/products/design-tools/vivado.html> (дата звернення: 05.12.2023).

24. Xilinx. Adaptable and Real-Time AI Inference Acceleration [Электронный ресурс]. – Режим доступа: <https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html> (дата звернення: 05.12.2023).

25. OpenCV is the world's biggest computer vision library [Электронный ресурс]. – Режим доступа: <https://opencv.org/> (дата звернення: 05.12.2023).

26. Docker. Make better, secure software from the start [Электронный ресурс]. – Режим доступа: <https://www.docker.com/> (дата звернення: 05.12.2023).

ДОДАТОК А

Лістинг А.1 – main.py

```

import cv2
import ctypes
import numpy as np
import SSD
from threadedcapture import ThreadedCapture
import time
import windowsystem as window_system

# Завантаження бібліотеки C за допомогою ctypes
ll = ctypes.cdll.LoadLibrary
lib = ll("./tfssdtest.so") # використання бібліотеки tfssdtest
#lib = ll("./ssdhardnet.so") # альтернативно можна використати
ssdhardnet
lib.process.restype = ctypes.c_int

# Ініціалізація камери та системи вікон
cam = ThreadedCapture(0)
win_sys = window_system.WindowSystem((1900,1000))

# Ініціалізація змінних для обрахунку часу та кількості кадрів
frame_count = 0
time_start = time.time()
total_call_func_time = 0

# Класи об'єктів для розпізнавання
CLASS_NUM = 91
class_table =
dict([(44,"bottle"),(47,"cup"),(48,"fork"),(50,"spoon"),(51,"bowl"),
(52,"banana"),(53,"apple"),(55,"orange"),(58,"hot dog"),
(60,"donut"),(87,"scissors")])

# Читання кадру з камери та налаштування масштабу
frame = cam.read()
print(frame.shape[1],frame.shape[0])
x_scale = frame.shape[1]/300
y_scale = frame.shape[0]/300

# Головний цикл обробки відео
while True:
    objects_cnt = np.zeros(CLASS_NUM,dtype='int')
    frame_count += 1
    frame = cam.read()

    # Виклик функції DPU для розпізнавання об'єктів
    frame_resized = cv2.resize(frame,(300,300))

```

```

call_func_start = time.time()
size, data = SSD.detect(lib, frame_resized)
call_func_end = time.time()
total_call_func_time += (call_func_end - call_func_start)

# Накладання рамок на розпізнані об'єкти
for i in range(int(int(size)/6)):
    cv2.rectangle(frame,
(int(data[i*6+1]*x_scale),int(data[i*6+2]*y_scale)),
(int(data[i*6+3]*x_scale),int(data[i*6+4]*y_scale)), (255,0,0),5)
    cv2.putText(frame, class_table[int(data[i*6])],
(int(data[i*6+1]*x_scale+5),int(data[i*6+4]*y_scale-5)),
cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)
    objects_cnt[int(data[i*6])] += 1

# Відображення результатів на екрані
win_sys.clear()
for i in range(CLASS_NUM):
    if objects_cnt[i] != 0:
        win_sys.Add_result(i,objects_cnt[i])

win_sys.show(frame)
if cv2.waitKey(1) & 0xFF == ord('q'):
    break

# Підсумкові розрахунки та завершення роботи
time_end = time.time()
print("fps                : ", frame_count / (time_end -
time_start))
print("dpu average time : ", total_call_func_time /
frame_count, "s")
cv2.destroyAllWindows()

```

Лістинг A.2 – CCD.py

```

import cv2
import ctypes
import numpy as np

def detect(lib, frame):
    # Конвертування даних кадру до формату, сумісного з C
    frame_data = np.asarray(frame, dtype=np.uint8)
    frame_data = frame_data.ctypes.data_as(ctypes.c_char_p)

    # Створення масиву для зберігання результатів детектування
    data = (ctypes.c_float * 120)()

    # Виклик функції обробки з зовнішньої бібліотеки
    # Функція повертає розмір вихідних даних

```

```

        size      =      lib.process(frame.shape[0],      frame.shape[1],
frame_data, data)

        # Повернення розміру та даних детектування
        return size, data

```

Лістинг A.3 – hreadedCapture.py

```

import cv2
from threading import Thread
import time

class ThreadedCapture():
    def __init__(self, src=0):
        # Ініціалізація камери або відеофайлу через GStreamer
        self.cap      =      cv2.VideoCapture('filesrc
location=object_vid.mp4 ! qtdemux ! queue ! h264parse ! omxh264dec !
queue ! videoconvert ! queue ! video/x-raw, format=BGR ! appsink',
cv2.CAP_GSTREAMER)

        # Встановлення параметрів частоти кадрів
        self.fps = 60
        self.delay = 1 / self.fps  # Затримка між кадрами

        # Створення та запуск потоку для зчитування кадрів
        self.thread = Thread(target=self.update, args=())
        self.thread.daemon = True  # Позначення потоку як
демонічного

        self.thread.start()
        time.sleep(0.3)  # Невелика затримка для запуску потоку

    def update(self):
        # Оновлення кадрів у окремому потоці
        while True:
            _, self.img = self.cap.read()  # Читання кадру
            #time.sleep(self.delay)        # Затримка (можна
розкоментувати при необхідності)

    def read(self):
        # Повернення останнього прочитаного кадру
        return self.img

```

Лістинг A.4 – window_system.py

```

import numpy as np
import cv2
from threadedcapture import ThreadedCapture
import time

```

```

class_table =
dict([(44,"coke"), (47,"cup"), (48,"fork"), (50,"spoon"), (51,"bowl"), (5
2,"banana"), (53,"apple"), (55,"orange"), (58,"hot
dog"), (60,"donut"), (87,"scissors")])
price_table = dict([(44, 1.89), (47, 2.00), (48,
1.20), (50, 1.30), (51, 1.40), (52, 0.59), (53, 1.15), (55,
0.99), (58, 1.25), (60, 2.49), (87, 5.50)])

class WindowSystem () :

    def __init__ (self, GUI_SIZE) :

        # parameter settings
        self.GUI_SIZE = GUI_SIZE
        self.gui = np.empty( (self.GUI_SIZE[1],
self.GUI_SIZE[0], 3), dtype="uint8" )
        self.gui[:, :, :] = 255

        # logo
        self.PutImg(0, 0, 'logo.jpg')

        # camera img (size & position)
        self.camimg_x = 160
        self.camimg_y = 10
        self.camimg_h = 600
        self.camimg_w = 1060

        # usage
        self.usage_x = 765
        self.usage_y = 10
        self.PutImg(self.usage_x, self.usage_y, 'usage.jpg')

        # payment
        self.payment_x = 10
        self.payment_y = 1090
        self.PutImg(self.payment_x, self.payment_y,
'payment.jpg')

        # label
        self.label_x = 114
        self.label_y = 1090
        self.PutImg(self.label_x, self.label_y, 'label.png')

        # space for item list
        self.itemdisp_x = 165
        self.itemdisp_y = 1090
        self.itemdisp_h = 730
        self.itemdisp_w = 800

```

```

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 0] = 230

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 1] = 255

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 2] = 245

    # ntd
    self.ntd_x = 900
    self.ntd_y = 1090
    self.PutImg(self.ntd_x, self.ntd_y, 'ntd.jpg')

    # paynow
    self.paynow_x = 900
    self.paynow_y = 1493
    self.PutImg(self.paynow_x, self.paynow_y, 'paynow.jpg')
    #-----
-----

    self.item_x = 210
    self.text_1 = 1110
    self.text_2 = 1400
    self.text_3 = 1600
    self.text_4 = 1760

    self.item_cnt = 0
    self.price = 0

    def PutImg(self, x, y, img_name):
        img = cv2.imread('./img/'+img_name)
        self.gui[x:x+img.shape[0], y:y+img.shape[1]] = img

    def PutCamImg (self, img):
        img = cv2.resize(img, (self.camimg_w, self.camimg_h),
cv2.INTER_NEAREST)
        self.gui[self.camimg_x:self.camimg_x+img.shape[0],
self.camimg_y:self.camimg_y+img.shape[1], :] = img

    def Add_result(self,object_class,object_num):
        text1 = class_table[object_class]
        text2 = str(format(price_table[object_class], '.2f'))
        text3 = str(object_num)
        text4 =
str(format(price_table[object_class]*object_num, '.2f'))
        cv2.putText(self.gui, text1,(self.text_1, self.item_x +
60*self.item_cnt) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)
        cv2.putText(self.gui, text2,(self.text_2, self.item_x +
60*self.item_cnt) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)

```

```

        cv2.putText(self.gui, text3, (self.text_3, self.item_x +
60*self.item_cnt) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)
        cv2.putText(self.gui, text4, (self.text_4, self.item_x +
60*self.item_cnt) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)

        self.item_cnt += 1
        self.price += price_table[object_class]*object_num

    def clear (self):

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 0] = 230

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 1] = 255

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 2] = 245
        self.PutImg(self.ntd_x, self.ntd_y, 'ntd.jpg')
        self.item_cnt = 0
        self.price = 0

    def show (self, img) :
        # show price
        cv2.putText(self.gui, str(format(self.price, '.2f')),
(self.ntd_y + 200, self.ntd_x + 70), cv2.FONT_HERSHEY_COMPLEX, 2,
(200, 0, 0), 6)

        # paste image
        img = cv2.resize(img, (self.camimg_w, self.camimg_h),
cv2.INTER_NEAREST)
        self.gui[self.camimg_x:self.camimg_x+img.shape[0],
self.camimg_y:self.camimg_y+img.shape[1], :] = img
        gui = cv2.resize(self.gui, (1200, 660))
        cv2.imshow('Check Out So Easy', gui)

```

Лістинг A.4 – windowssystem.py

```

import numpy as np
import cv2
from threadedcapture import ThreadedCapture
import time

class table
dict([(44, "bottle"), (47, "cup"), (48, "fork"), (50, "spoon"), (51, "bowl"),
(52, "banana"), (53, "apple"), (55, "orange"), (58, "hot
dog"), (60, "donut"), (87, "scissors")])
=

```

```

        price_table = dict([(44,      30), (47,      49), (48,      25), (50,
20), (51,      39), (52,      18), (53,      40), (55,      25), (58,
39), (60,      40), (87,      140)])

class WindowSystem () :

    def __init__ (self, GUI_SIZE) :

        # parameter settings
        self.GUI_SIZE = GUI_SIZE
        self.gui = np.empty(      (self.GUI_SIZE[1],
self.GUI_SIZE[0], 3), dtype="uint8" )
        self.gui[:, :, :] = 255

        # logo
        self.PutImg(0, 0, 'logo.jpg')

        # camera img (size & position)
        self.camimg_x = 160
        self.camimg_y = 10
        self.camimg_h = 600
        self.camimg_w = 1060

        # usage
        self.usage_x = 765
        self.usage_y = 10
        self.PutImg(self.usage_x, self.usage_y, 'usage.jpg')

        # payment
        self.payment_x = 10
        self.payment_y = 1090
        self.PutImg(self.payment_x,      self.payment_y,
'payment.jpg')

        # label
        self.label_x = 114
        self.label_y = 1090
        self.PutImg(self.label_x, self.label_y, 'label.png')

        # space for item list
        self.itemdisp_x = 165
        self.itemdisp_y = 1090
        self.itemdisp_h = 730
        self.itemdisp_w = 800

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 0] = 230

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 1] = 255

```

```

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 2] = 245

    # ntd
    self.ntd_x = 900
    self.ntd_y = 1090
    self.PutImg(self.ntd_x, self.ntd_y, 'ntd.jpg')

    # paynow
    self.paynow_x = 900
    self.paynow_y = 1493
    self.PutImg(self.paynow_x, self.paynow_y, 'paynow.jpg')
    #-----
-----

    self.item_x = 210
    self.text_1 = 1110
    self.text_2 = 1400
    self.text_3 = 1600
    self.text_4 = 1760

    self.item_cnt = 0
    self.price = 0

    def PutImg(self, x, y, img_name):
        img = cv2.imread('./img/'+img_name)
        self.gui[x:x+img.shape[0], y:y+img.shape[1]] = img

    def PutCamImg (self, img):
        img = cv2.resize(img, (self.camimg_w, self.camimg_h),
cv2.INTER_NEAREST)
        self.gui[self.camimg_x:self.camimg_x+img.shape[0],
self.camimg_y:self.camimg_y+img.shape[1], :] = img

    def Add_result(self,object_class,object_num):
        text1 = class_table[object_class]
        text2 = str(price_table[object_class])
        text3 = str(object_num)
        text4 = str(price_table[object_class]*object_num)
        cv2.putText(self.gui, text1,(self.text_1, self.item_x +
60*self.item_cnt) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)
        cv2.putText(self.gui, text2,(self.text_2, self.item_x +
60*self.item_cnt) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)
        cv2.putText(self.gui, text3,(self.text_3, self.item_x +
60*self.item_cnt) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)
        cv2.putText(self.gui, text4,(self.text_4, self.item_x +
60*self.item_cnt) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)

        self.item_cnt += 1
        self.price += price_table[object_class]*object_num

```

```

def clear (self):

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 0] = 230

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 1] = 255

self.gui[self.itemdisp_x:self.itemdisp_x+self.itemdisp_h,
self.itemdisp_y:self.itemdisp_y+self.itemdisp_w, 2] = 245
    self.PutImg(self.ntd_x, self.ntd_y, 'ntd.jpg')
    self.item_cnt = 0
    self.price = 0

def show (self, img) :

    # show price
    cv2.putText(self.gui, str(self.price), (self.ntd_y +
250, self.ntd_x + 70), cv2.FONT_HERSHEY_COMPLEX, 2, (200, 0, 0), 6)

    # paste image
    img = cv2.resize(img, (self.caming_w, self.caming_h),
cv2.INTER_NEAREST)
    self.gui[self.caming_x:self.caming_x+img.shape[0],
self.caming_y:self.caming_y+img.shape[1], :] = img

    cv2.imshow('Check Out So Easy', self.gui)

```

Лістинг А.5 – DEMO.py

```

class_table
dict([(44,"coke"),(47,"cup"),(48,"fork"),(50,"spoon"),(51,"bowl"),(52,"banana"),(53,"apple"),(55,"orange"),(58,"hot
dog"),(60,"donut"),(87,"scissors")])
total_cnt = np.zeros(CLASS_NUM,dtype='float')
objects_cnt = np.zeros(CLASS_NUM,dtype='int')
current_frame_count = 0
while True:
    frame_count+=1;
    frame = cam.read()
    if frame is None:
        break

```

```

frame_resized = cv2.resize(frame, (300,300))
#frame = cv2.imread('pedestrian.jpg')
#print(type(frame))
#frame = cv2.reshape()
call_func_start =time.time()
size , data = SSD.detect(lib,frame_resized)
call_func_end = time.time()
total_call_func_time += (call_func_end - call_func_start)
for i in range(int(int(size)/6)):

#print(data[i+0],data[i+1],data[i+2],data[i+3],data[i+4],data[i+5])
    cv2.rectangle(frame,
(int(data[i*6+1]*x_scale),int(data[i*6+2]*y_scale)),
(int(data[i*6+3]*x_scale),int(data[i*6+4]*y_scale)), (255,0,0),5)
    cv2.putText(frame,
class_table[int(data[i*6])],(int(data[i*6+1]*x_scale+5),int(data[i*6
+4]*y_scale-5)) , cv2.FONT_HERSHEY_COMPLEX, 1, (200, 0, 0), 2)
    total_cnt[int(data[i*6])]+=1
#print(time.time()-update_time)
if time.time()-update_time > 1:
    update_time = time.time()
    win_sys.clear()
    #win_sys.PutCamImg(frame)
    ff = (frame_count-current_frame_count)
    current_frame_count = frame_count
    for i in range(CLASS_NUM):
        objects_cnt[i] = int((total_cnt[i])/ff+0.8)
    total_cnt = np.zeros(CLASS_NUM,dtype='float')
    for i in range(CLASS_NUM):
        if objects_cnt[i]!=0:
            win_sys.Add_result(i,objects_cnt[i])
    #cv2.imshow('result',frame)
    #win_sys.PutCamImg(frame)
    win_sys.show(frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    time_end = time.time()
    print("fps          : ",frame_count/(time_end-time_start))
    print("dpu          average          time          :
",total_call_func_time/frame_count,"s")
    cv2.destroyAllWindows()
    #for i in range(int(size)):
    # print(data[i])

```