Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)

Кафедра Автоматизації проектування обчислювальної техніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)
(рівень вищої освіти)

Методи обробки зображень у вбудованих системах
(тема)

Виконав: студент 2 курсу, групи СКСм-20-1
Семененко І.Г.
(прізвище, ініціали)
Спеціальність 123 Комп'ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма_____
Спеціалізовані комп'ютерні системи
_____
(повна назва освітньої програми)
Керівник доц. Філіппенко І.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Чумаченко С.В.
(підпис) (прізвище, ініціали)

2021 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Автоматизації проектування обчислювальної техніки

Рівень вищої освіти другий (магістерський)

Спеціальність 123 Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Спеціалізовані комп'ютерні системи
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Семененку Івану Георгійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи обробки зображень у вбудованих системах

затверджена наказом університету від «04» 11 2021 р. № 1635Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 12 2021 р.

3. Вихідні дані до роботи
Вбудовані системи
Алгоритми обробки зображень
FPGA
ARM

4. Перелік питань, що потрібно опрацювати в роботі
Аналіз предметної області та постановка задачі
Розробка моделей для вбудованих систем
Аналіз отриманих даних

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 16 слайдів

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання	04.11.2021р. – 11.11.2021р.	
2	Аналіз предметної області	11.11.2021р. – 18.11.2021р.	
3	Аналіз проблемної галузі	18.11.2021р. – 25.11.2021р	
4	Розробка моделі на ARM	25.11.2021р. – 02.12.2021р.	
5	Розробка моделі на FPGA	02.12.2021р. – 09.12.2021р.	
6	Оформлення пояснювальної записки	09.12.2021р. – 17.12.2021р.	

Дата видачі завдання _____ _____20__ р.

Студент _____
(підпис)

Керівник роботи _____ _доц. Філіппенко І.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка містить 52 сторінки, 32 рисунка, 6 таблиць, список з 25 використаних джерел.

ОБРОБКА ЗОБРАЖЕНЬ, ARM, ПЛІС, FPGA, SOC, СИСТЕМИ РЕАЛЬНОГО ЧАСУ, ВБУДОВАНІ СИСТЕМИ, ЗГОРТКОВІ МЕРЕЖІ

Задачею дослідження є огляд існуючих методів обробки зображень у вбудованих систем, розробка моделі дослідження з використанням ARM та FPGA з метою поліпшення швидкості обробки цифрових даних.

У ході дослідження були розглянуті основні принципи цифрової обробки даних, виконано порівняння способів реалізації алгоритмів обробки зображень та протестована архітектура програми на FPGA для обробки зображень з використанням згорткових нейронних мереж.

Результатом дослідження є розроблена модель цифрової обробки зображень в режимі реального часу, а також модель, що використовує технологію нейронних мереж для швидкої обробки.

ABSTRACT

The explanatory note contains 52 pages, 32 figures, 6 tables, 24 references.

IMAGE PROCESSING, ARM, FPGA, SOC, REAL TIME, DPR, EMBEDDED SYSTEMS, CONVOLUTIONAL NETWORK

The main goal of the study is to review the existing methods of image processing in embedded systems and develop a model to improve the speed of digital image processing using ARM and FPGA.

The basic principles of digital data processing were examined. A comparison was done for various implementations of image processing algorithms. An architecture for using convolutional neural networks on FPGA was designed and tested during the project.

The result of the project is the developed model of digital image processing in real time mode, as well as the model that leverages neural network technology for quick data processing.

ЗМІСТ

2.CHEN, R., CHEN, G., AND CHEN, L.: SYSTEM DESIGN CONSIDERATION FOR DIGITAL WHEELCHAIR CONTROLLER. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, 2000. – 907 C.....56
3.CONVOLUTIONAL NEURAL NETWORK [ЕЛЕКТРОННИЙ РЕСУРС] – РЕЖИМ ДОСТУПУ ДО РЕСУРСУ: HTTPS://EN.WIKIPEDIA.ORG/WIKI/CONVOLUTIONAL_NEURAL_NETWORK.....56
4.DEAN J., CORRADO G., MONGA R., CHEN K., DEVIN M., MAO M., RANZATO M., SENIOR A., TUCKER P., KE YANG, QUOC V. LE, AND ANDREW Y. NG: LARGE SCALE DISTRIBUTED DEEP NETWORKS, BARTLETT, F.C.N. PEREIRA, C.J.C. BURGES, L. BOTTOU, AND K.Q. WEINBERGER, EDITORS, ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS, 2012. – 1240 C.....56
5.EMBEDDED SYSTEM [ЕЛЕКТРОННИЙ РЕСУРС] – РЕЖИМ ДОСТУПУ ДО РЕСУРСУ: HTTPS://EN.WIKIPEDIA.ORG/WIKI/EMBEDDED_SYSTEM 56
6.GABRICK, M., NICHOLSON, R., WINTERS. F., YOUNG, B., PATTON, J.: FPGA CONSIDERATIONS FOR AUTOMOTIVE APPLICATIONS. PROCEEDINGS OF SAE WORLD CONGRESS AND EXHIBITION, 2006.….56
7.FIELD PROGRAMMABLE GATE ARRAYS [ЕЛЕКТРОННИЙ РЕСУРС] – РЕЖИМ ДОСТУПУ ДО РЕСУРСУ: HTTP://EN.WIKIPEDIA.ORG/WIKI/FPGA.....56
8.HE, KAIMING: DELVING DEEP INTO RECTIFIERS: SURPASSING HUMAN-LEVEL PERFORMANCE ON IMAGENET CLASSIFICATION, 2015 IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV), 2015.....56
9.KAO, C.: BENEFITS OF PARTIAL RECONFIGURATION. XCELL JOURNAL, FOURTH QUARTER, XILINX, INC., 2005. – 68 C.....56

10.KRIZHEVSKY, ALEX: IMAGENET CLASSIFICATION WITH DEEP CONVOLUTIONAL NEURAL NETWORKS , COMMUNICATIONS OF THE ACM, 2017. – 90 C.....56
11.LECUN, Y.: GRADIENT-BASED LEARNING APPLIED TO DOCUMENT RECOGNITION – PROCEEDINGS OF THE IEEE, 1998. – 2324 C.....57
12.LEE, D., CHOI, A., KOO, J., LEE, J., KIM, B.: A WIDEBAND DS-CDMA MODEM FOR A MOBILE STATION.: IEEE TRANSACTIONS ON CONSUMER ELECTRONICS, 1999. – 1269 C.....57
13.LYSAGHT, P., BLODGET, B., MASON, J., YOUNG, J., BRIDGEFORD, B.: ENHANCED ARCHITECTURE, DESIGN METHODOLOGIES AND CAD TOOLS FOR DYNAMIC RECONFIGURATION FOR XILINX FPGAS. PROCEEDINGS OF INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, MADRID, SPAIN, 2006. – 367 C.....57
14.MATHEMATICAL MORPHOLOGY [ЕЛЕКТРОННИЙ РЕСУРС] – РЕЖИМ ДОСТУПУ ДО РЕСУРСУ: HTTPS://EN.WIKIPEDIA.ORG/WIKI/MATHEMATICAL_MORPHOLOGY. .57
15.MONMASSON, E., CRISTEA, M.: FPGA DESIGN METHODOLOGY FOR INDUSTRIAL CONTROL SYSTEMS – A REVIEW. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, 2007. – 1842 C.....57
16.OVASKA, S., AND VAINIO, O.: EVOLUTIONARY-PROGRAMMING-BASED OPTIMIZATION OF REDUCEDRANK ADAPTIVE FILTERS FOR REFERENCE GENERATION IN ACTIVE POWER FILTERS. IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, 2004. – 916 C.....57
17.PARUNAK, H. V. D. BRUECKNER, S. A., SAUTER, J.: DIGITAL PHEROMONES FOR COORDINATION OF UNMANNED VEHICLES. LECTURE NOTES IN COMPUTER SCIENCE, SPRINGER VERLAG, 2004. – 263 C.....57

ВСТУП

Цифрова обробка зображень – досить важливий напрямок у розвитку сучасної обчислювальної техніки. Вона знаходить своє застосування у безлічі різних галузях науки й техніки: медичні дослідження, військова техніка, розпізнання та класифікація даних, системи автоматичного керування транспортом, системи безпеки, системи сканування місцевості та стеження, машинний зір, обробка даних для виявлення характеристик об’єктів тощо.

Сучасні комп’ютерні системи часто будуються на основі мікроконтролерів, вбудованих як частина цілісного пристрою з електронним чи електро-механічним обладнанням. Оскільки, в такому випадку, вбудована система зазвичай контролює фізичні операції в яку вона вбудована, вона часто має обмеження в роботі в режимі реального часу.

Останнім часом вбудовані системи набирають популярності в більш складних комп’ютерних системах. Таке поширення зазвичай зумовлене невеликим розміром, низьким енергоспоживанням і відносно низькою ціною у порівнянні із більш комплексними рішеннями. Вбудовані системи також пропонують широкий спектр застосування: від загального призначення до тих, що спеціалізується на певному класі обчислень.

Варто також окремо сказати про FPGA та DSP. Основне призначення цифрового сигнального процесору (DSP) – оптимальна та швидка обробка сигналів в режимі реального часу, а програмовані вентильні матриці (FPGA) складаються з безлічі логічних блоків, конфігурація яких (схема) може бути змінена у будь-який час роботи пристрою.

У більшості випадках критичним для програмного забезпечення, що займається обробкою великої кількості даних є швидкість виконання. Навіть сучасні потужні процесори не завжди здатні своєчасно відреагувати на зміни у вхідних зображеннях, які надходять з досить високою швидкістю.

Зважаючи на те, що безумовним лідером зі швидкості обробки були б інтегральні схеми спеціального призначення, апаратні прискорювачі на базі FPGA все частіше вбудовуються у різного роду обчислювальні системи. У порівнянні з іншими рішеннями із жорстко спроектованою логікою, перевага пристроїв FPGA полягає у їх гнучкості, що виникає через програмовану природу. Також, більшість сучасних пристроїв FPGA включають зокрема готові блоки DSP, регістри пам’яті, блоки часткової реконфігурації, блоки введення-виведення інформації, що дозволяють користуватися майже будь-яким відомим протоколом даних. Все це полегшує процес розробки та тестування.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ЗАГАЛЬНІ ВІДОМОСТІ

1.1 Загальні відомості про цифрову обробку зображень

Обробка зображень – форма перетворення інформації, яка представлена у вигляді двомірного масиву даних. Обробка зображень може бути виконана як задля отримання іншого зображення на виході, так і для отримання зовсім іншої інформації (наприклад, розпізнавання тексту, визначення координат об’єктів тощо). Також, окрім статичних двомірних зображень досить часто потрібна обробка зображень, що змінюються в часі, наприклад, відеоряду. [12]

Більшість методів обробки одномірних сигналів можуть бути застосовані і для двомірного випадку. Таким чином, наприклад, алгоритми цифрової фільтрації, такі як медіанний фільтр, можна використовувати у цифровій обробці зображень. Деякі з зазначених одномірних методів значно ускладнюються з переходом до двомірного сигналу та вносять багато нових понять, таких як зв’язність та інваріантність даних.

В обробці зображень широко використовуються перетворення Фур’є, а також Вейвлет-перетворювання. Отже, обробку зображень загалом можна розділити на просторову (зміна яскравості, гамма-корекція тощо) та частотну (перетворення Фур’є та ін.).

Серед безлічі типових задач, яких можна досягти за допомогою цифрової обробки зображень виділяють наступні:

- комбінування зображень у різний спосіб;
- геометричні перетворювання – обертання, або масштабування;
- афінні перетворювання – більш складні геометричні трансформації, що включають зрушення по діагоналі та дзеркальне відображення;
- кольорова корекція – зміна яскравості та контрасту зображення, перехід в інший кольоровий простір;

- інтерполяція – знаходження проміжних значень кольору;
- анти-аліасинг – згладжування «ступінчатого» ефекту;
- фільтрація – поліпшення якості зображення. Підкреслення, або видалення окремих деталей;
- морфологічні перетворювання – поетапне накладання примітивів на вхідне зображення задля стиснення, або пошуку об’єктів.

1.2 Алгоритми цифрової обробки зображень

Медіанний фільтр. Медіанний фільтр – один із видів нелінійних цифрових фільтрів, що широко використовується в цифровій обробці сигналів та зображень для видалення (зменшення) шуму.

Основна ідея алгоритму медіанної фільтрації полягає у побудові «вікна» (NxN матриця, де N підбирається в залежності від даних). Алгоритм рухає «вікно» вздовж вхідного сигналу, заповнюючи його поточним значенням k та його сусідніми величинами. Відліки в середині «вікна» сортуються, а поточне значення k перезаписується середнім значенням відсортованого списку. У разі парного числа відліків у «вікні», вихідне значення дорівнює середньому значенню двох відліків, що знаходяться у середині відсортованого списку.

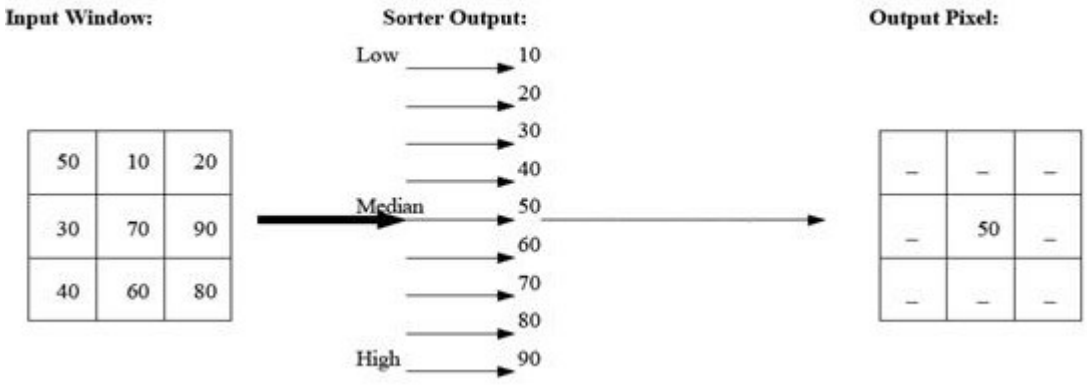


Рисунок 1.1 – Візуалізація роботи медіанного фільтру

Відтінки сірого. Перетворення у градації сірого кольору (grayscale) – досить простий алгоритм конвертації кольорового зображення у зображення, що містить тільки відтінки сірого (від чорного 0 до білого 255) кольору. В такому вигляді кожен піксель вихідного зображення представляє інформацію про його інтенсивність.

Як правило, компонент окремого кольору пікселя береться до уваги з певним коефіцієнтом інтенсивності (рисунок 1.2). Алгоритм часто використовується для зменшення об’єму даних, які будуть використовуватися в подальшій обробці, шляхом усереднення кольорових компонентів.

$$Y' = 0.299R + 0.587G + 0.114B$$

Рисунок 1.2 – Формула перетворення для PAL та NTSC

Бінаризація зображення. Бінаризація зображення – є ще одним прикладом досить простого перетворення. У якості вхідних даних зазвичай використовується зображення у відтінках сірого. Головним параметром такого перетворення виступає порогове значення – значення, яке буде критерієм перевірки інтенсивності кожного пікселя (рисунок 1.3).

Вихідне зображення містить масив двозначних величин. Якщо інтенсивність пікселя вхідного зображення вища зазначеного порогу, такий піксель вважається білим (1), якщо нижча – чорним (0).

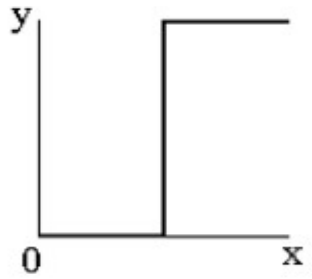


Рисунок 1.3 – Функція бінаризації

Морфологічні операції. Бінарне зображення представляється у вигляді упорядкованого набору (множини) чорно-білих точок. Кожна операція двійкової морфології є деяким перетворенням над цією множиною. У якості вхідних даних використовується бінарне зображення B та деякий структурний елемент S. Результатом операції також є бінарне зображення.

Структурний елемент S представляє деяке двійкове зображення (геометричну форму). Такий елемент може бути довільного розміру та довільної структури. Найчастіше використовуються симетричні елементи, як прямокутник фіксованого розміру, або коло деякого діаметру (рисунок 1.4). У кожному елементі виділяється особлива точка, яка називається початковою (origin). Вона може бути розташована в будь-якому місці елемента, хоча в симетричних структурах це зазвичай центральний піксель.

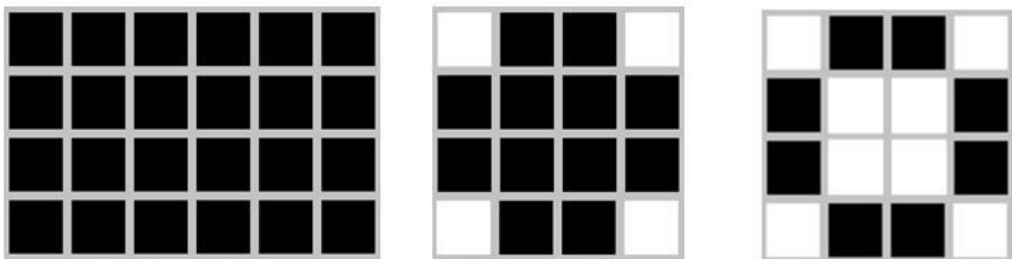


Рисунок 1.4 – Приклад структурних елементів

Спочатку результуюча поверхня заповнюється 0, утворюючи повністю біле зображення. Потім здійснюється зондування (probing) або сканування вхідного зображення структурним елементом піксель за пікселем. Для зондування кожного пікселя на зображення «накладається» структурний елемент так, щоб поєдналися зондована та початкові точки. Потім перевіряється деяка умова відповідність структурного елемента і точок зображення під ним. Якщо умова виконується, то на результуючому зображенні у відповідному місці ставиться 1.

За розглянутою схемою виконуються базові операції. Гарним прикладом таких операцій є операції розширення (dilation) та звуження (erosion). Також різні комбінації базових операцій, що виконуються послідовно утворюють похідні, такі як операцій розкриття та закриття.

При виконанні операції розширення структурний елемент S застосовується до всіх точок бінарного зображення. Щоразу, коли початок координат структурного елемента поєднується з одиничним бінарним пікселем, до всього структурного елемента застосовується перенесення і наступне логічне додавання відповідних точок бінарного зображення. Результати логічного додавання записуються у вихідне бінарне зображення.

При виконанні операції звуження структурний елемент проходить по всім точкам зображення. Якщо в певній позиції кожен точка структурного елемента збігається з одиничним пікселем бінарного зображення, виконується логічне додавання центрального пікселя структурного елемента з відповідним пікселем вихідного зображення.

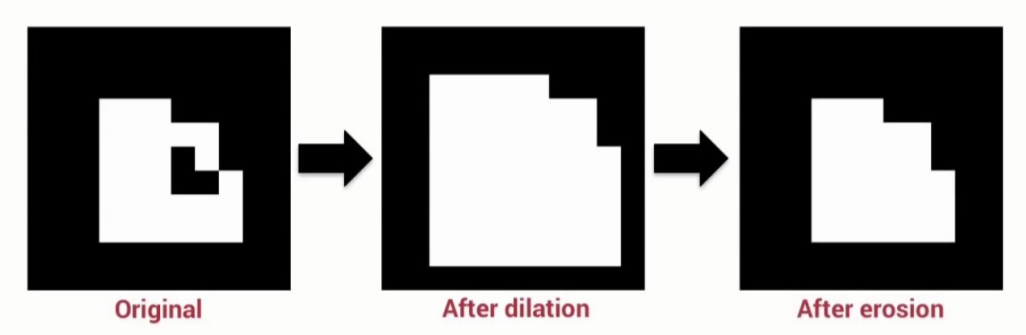


Рисунок 1.5 – Послідовне виконання розширення та звуження

Послідовне виконання операції розширення та звуження утворює похідну операцію – закриття (closing). Така операція дозволяє позбутися від невеликих отворів в середині та на межах об’єктів зображення. Якщо виконати тільки операцію розширення, об’єкти зображення позбудуться отворів, але при цьому відбудеться збільшення контурів об’єктів. Уникнути цього збільшення дозволяє операція звуження.

1.3 Використання штучного інтелекту

Згорткова нейронна мережа (convolutional neural network) — спеціальна архітектура штучних нейронних мереж, що відрізняється дуже високою здатністю до розпізнавання патернів на зображеннях. Робота згорткової нейронної мережі зазвичай інтерпретується як перехід від конкретних особливостей зображення до більш абстрактних деталей. При цьому мережа самоналаштовується і виробляє необхідну ієрархію абстрактних ознак, фільтруючи менш деталі та виділяючи більш важливі.

Двовимірна згортка – досить проста математична операція, що включає поетапне перемноження матриць. Основною частиною алгоритму виступає ядро (kernel) згортки, матриця невеликого розміру з деякими вагами. Ядро переміщується по матриці вхідних даних, здійснюючи поелементне множення матриць, що перетнулися. Сума всіх елементів результуючої матриці утворює вихідне значення певного етапу згортки.

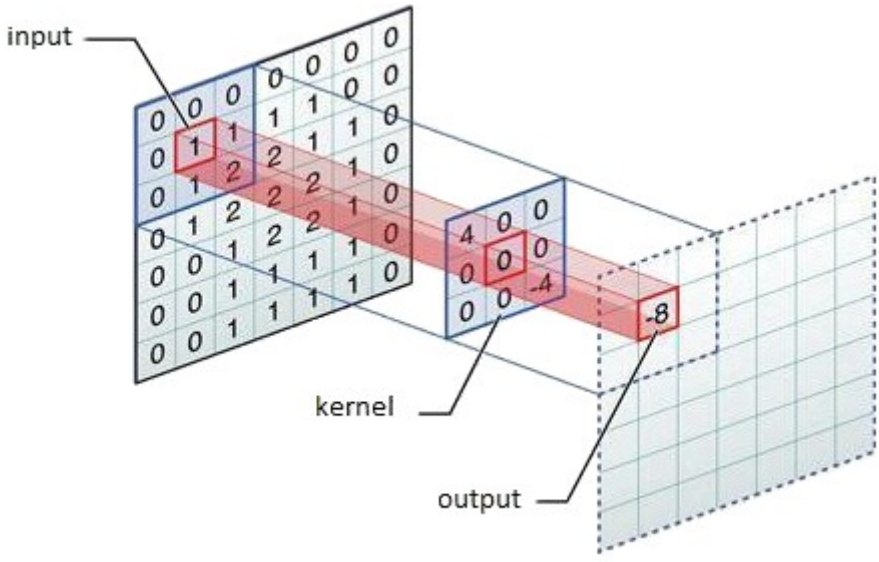


Рисунок 1.6 – Операція згортки

Наступним кроком є шар пулінгу (pooling), або субдескритизації – нелінійним ущільненням карти ознак, при цьому група пікселів 2x2 ущільнюється до одного пікселя, проходячи нелінійне перетворення. Найпоширенішою функцією такого перетворення виступає функція максимуму. Операція пулінгу дозволяє значно зменшити просторовий обсяг зображення та пришвидшити подальші обчислення.

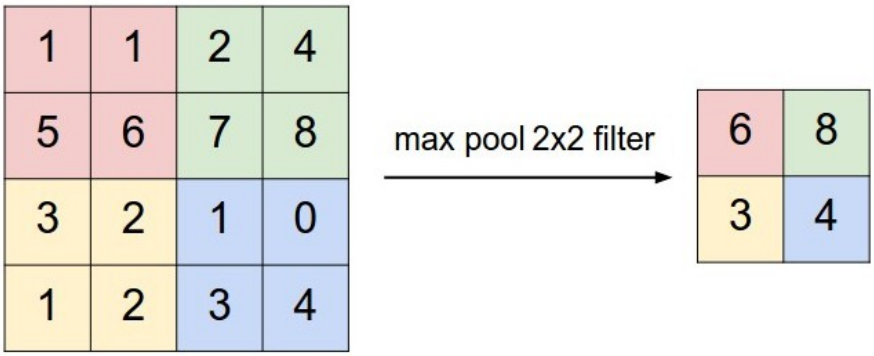


Рисунок 1.7 – Операція пулінгу

Після декількох проходжень згортки зображення та ущільнення за допомогою пулінга система перебудовується від конкретної сітки пікселів з високою роздільною здатністю до більш абстрактних карт ознак, як правило, на кожному наступному шарі збільшується кількість каналів і зменшується розмірність зображення в кожному каналі. Зрештою залишається набір каналів, що зберігають невелику кількість даних, виявлену з вихідного зображення.

Ці дані об'єднуються і передаються на звичайну повнозв’язну нейронну мережу, яка також може складатися з кількох шарів. При цьому повнозв'язні шари вже втрачають просторову структуру пікселів і мають порівняно невелику розмірність (стосовно кількості пікселів вихідного зображення).

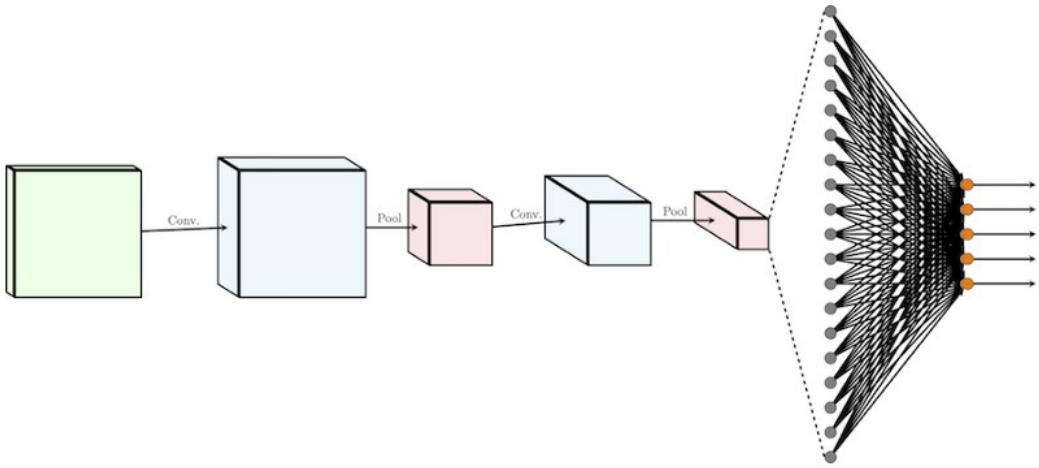


Рисунок 1.8 – Загальна структура згорткової мережі

Найбільш простим та популярним способом навчання згорткової мережі є метод навчання з учителем (на позначених даних). Таке навчання побудоване на основі методу зворотного поширення помилки (backpropogation) та його модифікаціям. Але є також ряд технік навчання згорткової мережі без вчителя. Наприклад, фільтри операції згортки можна навчити окремо та автономно, подаючи на них вирізані випадковим чином шматочки вихідних зображень навчальної вибірки та застосовуючи для них будь-який відомий алгоритм навчання без вчителя.

1.4 Вбудовані системи та їх застосування

Вбудована система (Embedded System) – це комп'ютерна система, що складається з належним чином підібраних апаратних і програмних компонентів та розроблена для конкретного програмного застосування. Вбудована система відповідає за виконання своїх функцій та впливає на спосіб спілкування з користувачем.

Розвиток комп'ютерних систем та зростаючі вимоги до якості контролю призвели до появи двох напрямів у спеціалізованих рішеннях контролю. Це

програмовані логічні контролери та мікроконтролери. На основі цих тенденцій створюються спеціалізовані системи, які виконують усі функції управління об'єктами чи процесами та тісно пов'язані з ними.

Кожна вбудована система заснована на мікропроцесорі (або мікроконтролері), запрограмованому на виконання обмеженої кількості завдань або навіть лише одного завдання. Залежно від призначення, він може містити програмне забезпечення, призначене лише для цього пристрою (прошивка) або операційна система зі спеціалізованим програмним забезпеченням.

Такі рішення можна знайти у вимірювальному обладнанні, у тому числі осцилографах, аналізаторах спектру, в автомобілях (наприклад, бортові комп'ютери), комп'ютерному устаткуванні (жорсткі диски, оптичні приводи, маршрутизатори), у рішеннях для телекомунікацій, так званих інтелектуальних будівлях, у пристроях , що використовуються в медичній діагностиці, системах управління польотом, а також, природно, у верстатах з ЧПУ, роботах та промислових машинах та ряді систем управління в автоматизації.

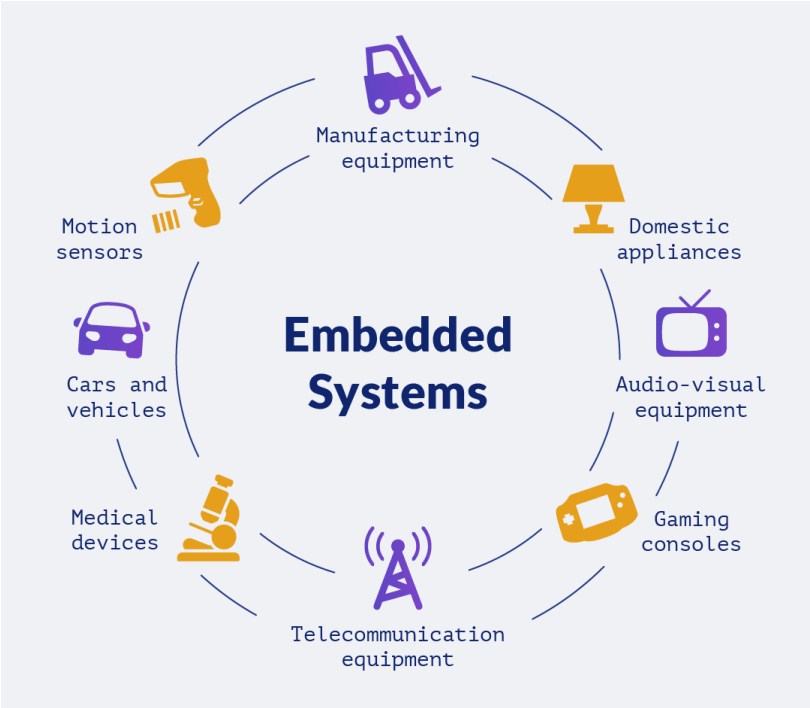


Рисунок 1.9 – Сфери застосування вбудованих систем

Складність цього типу рішень дуже різноманітна – від простих споживчих систем на основі малопотужних мікроконтролерів до багатопроцесорних розподілених систем, що використовуються у робототехніці.

Головною особливістю, яка відрізняє вбудовані системи від інших комп'ютерних систем, є, крім спеціалізованого характеру, якість програмного забезпечення та апаратних компонентів, що використовуються.

Вбудовані системи керуються мікроконтролерами (MCU) або процесорами цифрових сигналів (DSP), спеціалізованими інтегральними схемами (ASIC), логічними матрицями (FPGA). Ці системи обробки інтегровані з компонентами, призначеними для роботи з електричними та/або механічними інтерфейсами.

Інструкції з програмування вбудованих систем, так звані мікропрограмами, зберігаються в постійному пристрої або мікросхемах флеш-пам'яті, що працюють з обмеженими апаратними ресурсами комп'ютера. Вбудовані системи з'єднуються із зовнішнім світом через периферійні пристрої, пов'язуючи пристрої введення та виведення.

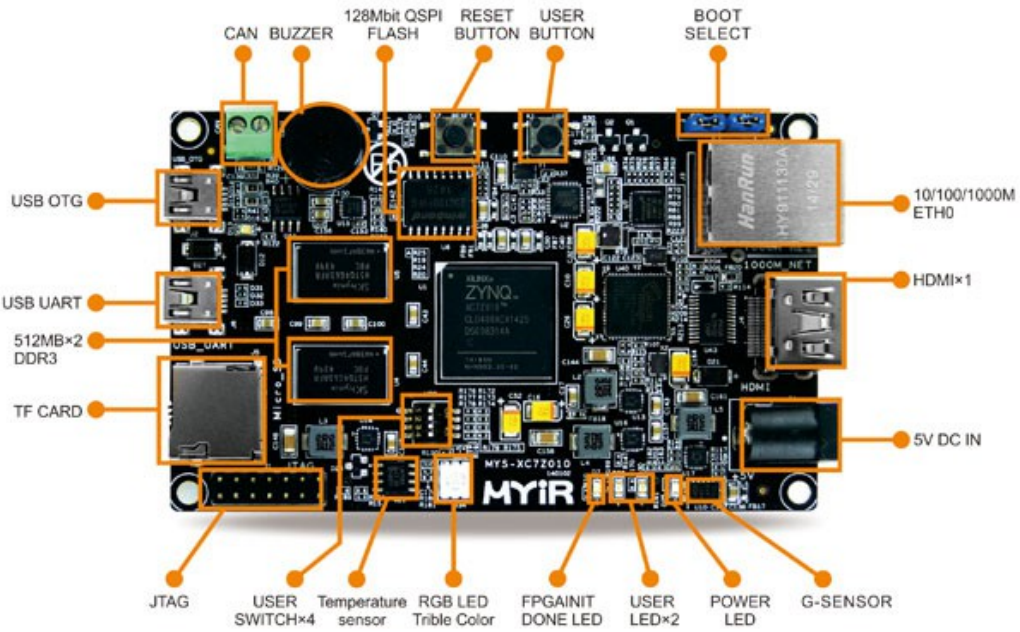


Рисунок 1.10 – Відлагоджувальна плата з MCU та FPGA

2 РОЗРОБКА МОДЕЛІ НА ОСНОВІ ARM CPU

2.1 Особливості архітектури ARM

ARM – це сімейство архітектур комп’ютера зі зменшеним набором команд (RISC) для комп’ютерних процесорів, налаштованих для різних середовищ.

До характерні особливостей RISC-процесорів можна віднести фіксовану довжину машинних інструкцій, простий формат команди, спеціалізовані команди для операцій із пам'яттю (читання або запису), велика кількість регістрів загального призначення та відсутність мікропрограм усередині самого процесора. Те, що в CISC-процесорі виконується мікропрограмами, в RISC-процесорі виконується як звичайний машинний код, що не відрізняється принципово від коду ядра ОС та додатків.

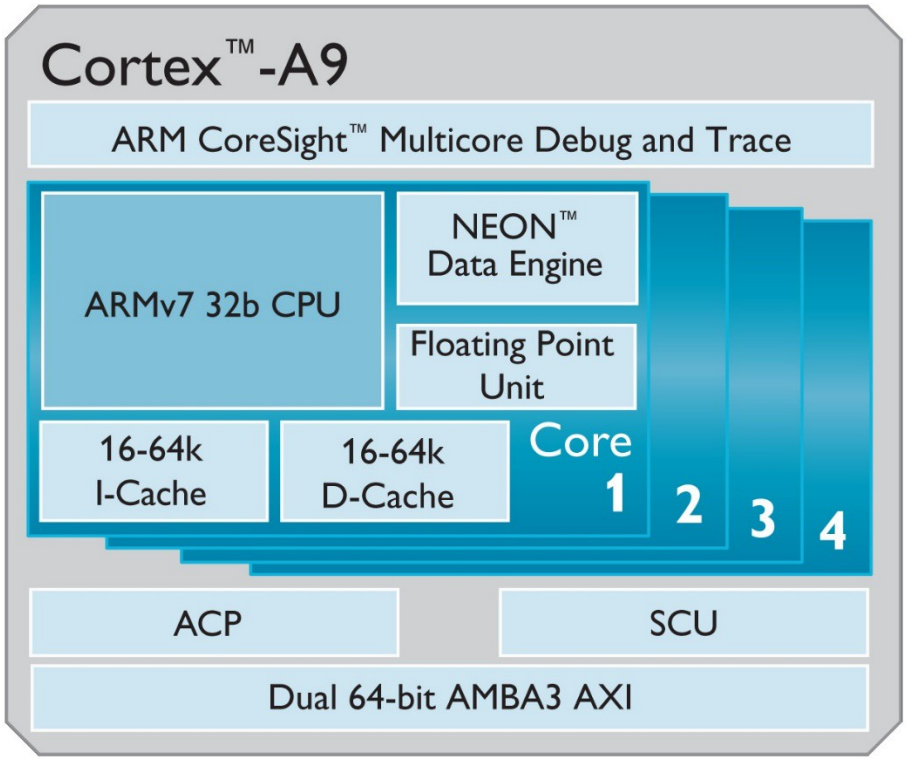


Рисунок 2.1 – Архітектура процесора ARM Cortex-A9

Технологія VFP (Vector Floating Point) – розширення сопроцесора в архітектурі ARM. Вона проводить низьковитратні обчислення над векторами чисел з плаваючою точкою одинарної та подвійної точності. Зі зростанням популярності архітектури APM та складності завдань, які на неї покладалися виникла потреба у більш потужній технології обробки векторних даних.

Розширення вдосконаленого SIMD, також зване технологією NEON – це комбінований 64- та 128-бітний набір команд SIMD спрямований на покращення роботи користувачів мультимедіа за рахунок прискорення кодування / декодування аудіо та відео, інтерфейсу користувача, 2D / 3D-графіки або ігор. NEON також може прискорювати алгоритми та функції обробки сигналів для прискорення таких програм, як обробка аудіо та відео, розпізнавання мови та особи, комп'ютерний зір та глибоке навчання.

Після увімкнення системи на базі ARM-процесора з ROM-пам'яті завантажується початковий завантажувач та адреса його точки входу. Початковий завантажувач проводить попередню ініціалізацію системи, виконуючи тим самим роль, яку виконує BIOS на системах x86, після чого може завантажити або системний завантажувач, або безпосередньо ОС.

Архітектура ARM підтримується безліччю операційних систем. Найбільш широко використовувані: Linux (зокрема Android), iOS, Windows Phone.

2.2 Реалізація алгоритмів обробки зображень

Розробка проводилася з використанням мови C++ на базі відкритої бібліотеки для обробки зображень OpenCV (версія 4.2.0). Задля доступу до всіх можливостей бібліотеки OpenCV, вона була попередньо зібрана спеціально для процесора архітектури ARMv7 з урахуваннями усіх його особливостей (включаючи NEON).

Фільтрація вхідного кольорового зображення за допомогою медіанного фільтру наведена у лістингу 2.1. У якості розміру тимчасового вікна та вікна пошуку було вибрано матрицю 3x3 пікселя.

Лістинг 2.1 – Фільтрація зображення

```
int ImgProc::MedianDenoise(const cv::Mat& src, cv::Mat& dst)
{
    return CpuTimer::MeasureTime([&src, &dst]() {
        cv::medianBlur(src, dst, 3);});
}
```

Рисунок 2.2 – Результат роботи медіанного фільтру

Подальше перетворення відфільтрованого кольорового зображення у зображення з градаціями сірого та двійкове зображення наведене у лістингах 2.2-2.3. Для бінаризації було використано алгоритм порогування (threshold) з параметром інтенсивності 50.

Лістинг 2.2 – Реалізація переходу у відтінки сірого

```
int ImgProc::Grayscale(const cv::Mat& src, cv::Mat& dst)
{
    return CpuTimer::MeasureTime([&src, &dst]() {
        cv::cvtColor(src, dst, cv::COLOR_BGR2GRAY);
    });
}
```

Лістинг 2.3 – Реалізація бінаризації зображення

```
int ImgProc::Binarization(const cv::Mat& src, cv::Mat& dst)
{
    return CpuTimer::MeasureTime([&src, &dst]() {
        cv::threshold(src, dst, 50, 255, cv::THRESH_BINARY);
    });
}
```

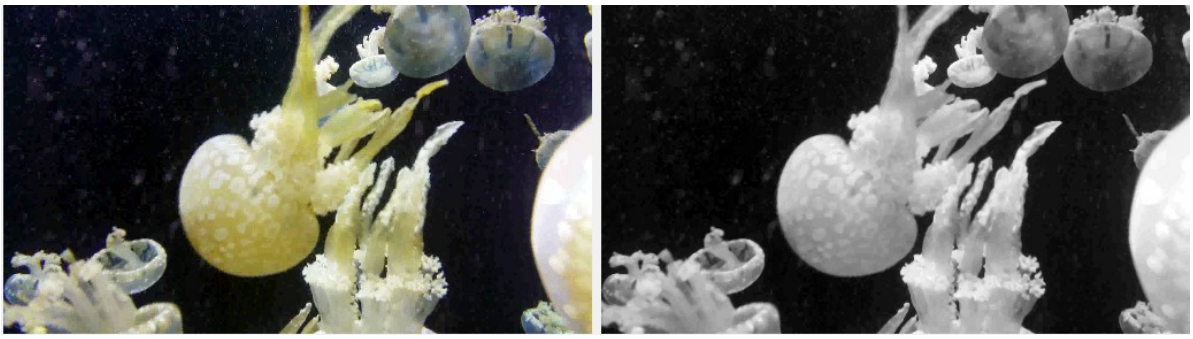


Рисунок 2.3 – Зображення з відтінками сірого

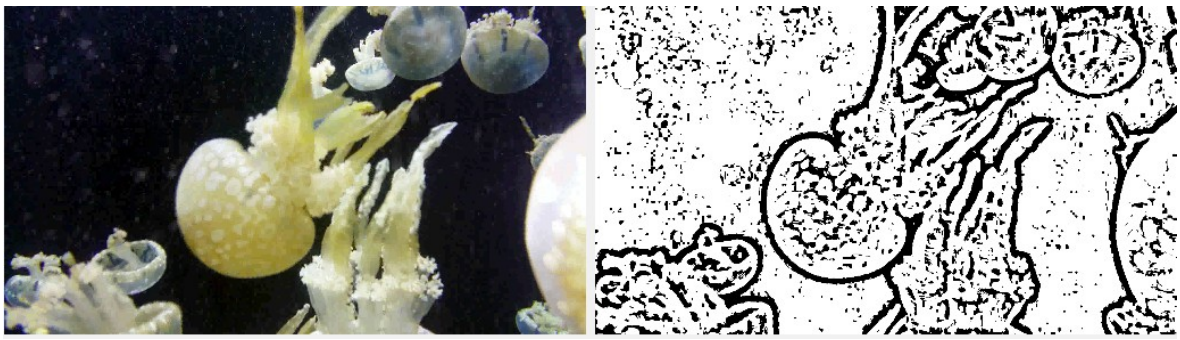


Рисунок 2.4 – Двійкове зображення

Для того, щоб позбутися від більшості дрібних об’єктів та досить чітко виділити контури великих об’єктів зображення було використано морфологічні перетворення відкриття (opening) та закриття (closing). Розмір структурного елементу цих операцій змінювався в залежності від розміру вхідного зображення та в середньому становив 10x10 пікселів. Приклад використання морфологічних операцій наведено у лістингу 2.4.

Лістинг 2.4 – Реалізація морфологічних операцій

```
int ImgProc::MorphOpCl(const cv::Mat& src, cv::Mat& dst)
{
    cv::Mat tempMat;
    return CpuTimer::MeasureTime([&]() {
        cv::morphologyEx(src, tempMat, cv::MORPH_OPEN,
                         m_kernel);
        cv::morphologyEx(tempMat, dst, cv::MORPH_CLOSE,
                         m_kernel);
    });
}
```

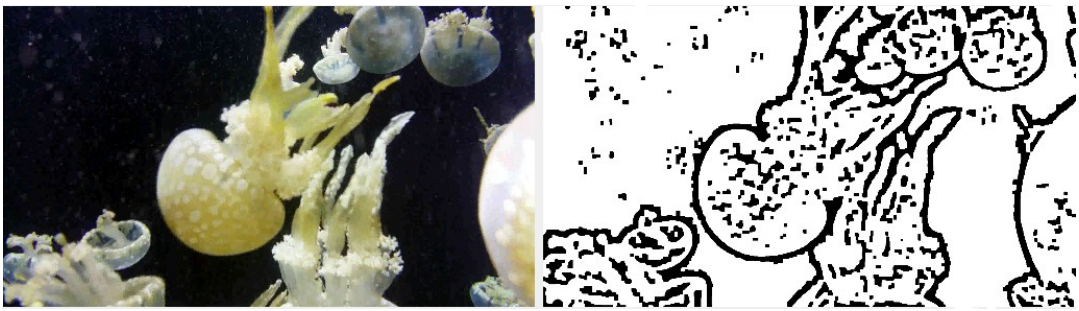


Рисунок 2.5 – Результат морфологічних перетворювань

2.3 Реалізація згорткової нейронної мережі

Перед початком виконання операції згортки вхідне зображення необхідно розділити на три канали. Також можна використовувати зображення в градаціях сірого, або чорно-біле зображення у якості вхідних даних.

На першому етапі обробки було використано вісім ядер згортки з довільно згенерованими значенням. Результати виконання алгоритму представлені на рисунку 2.6.

Лістинг 2.5 – Реалізація згортки вхідної матриці

```
void convolution(const cv::Mat& src,const cv::Mat& kernel,
                 cv::Mat& dst){
```

```
        cv::filter2D(src, dst, -1, kernel, cv::Point(-1,-1), 0,
4);}
```

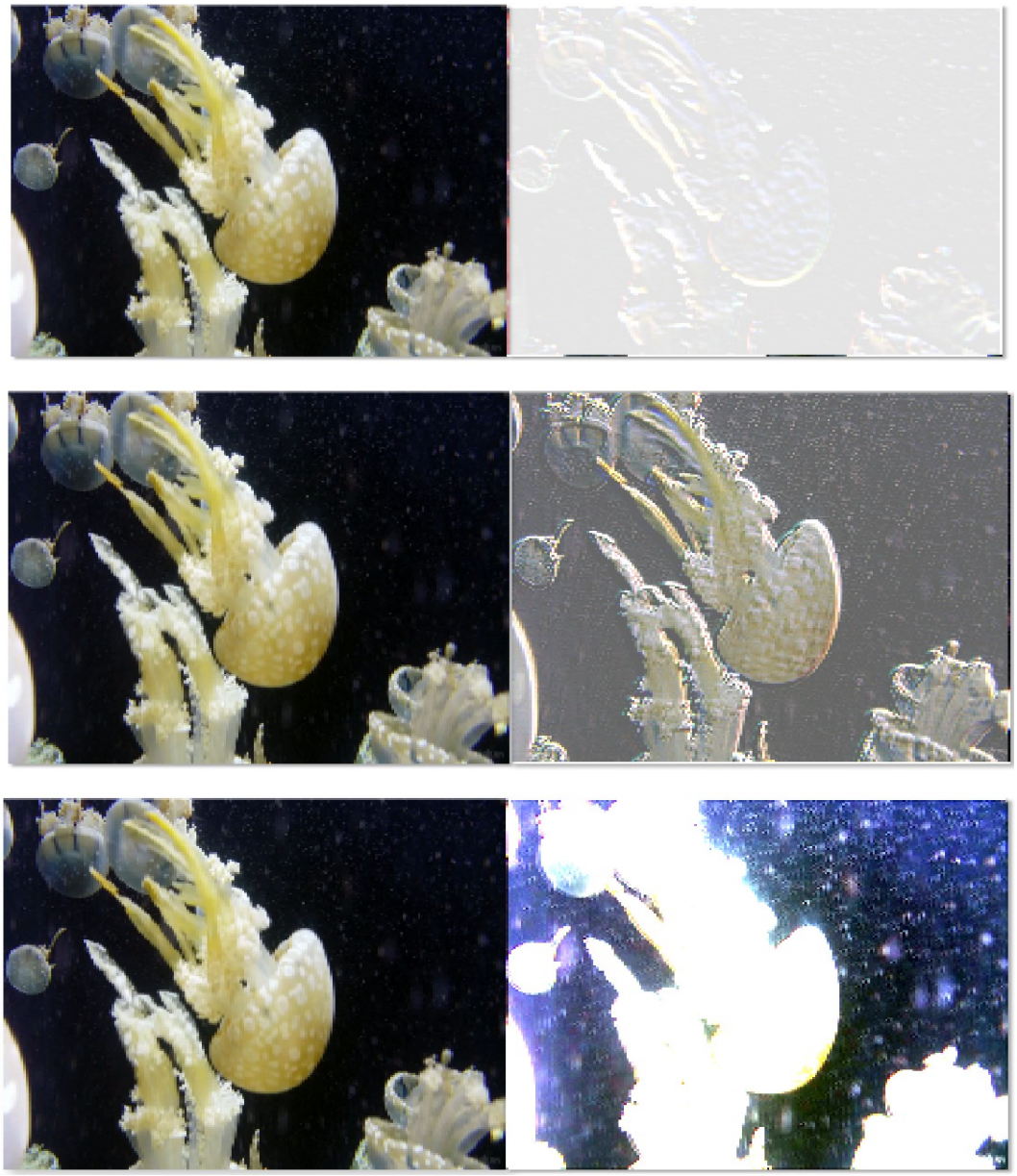


Рисунок 2.6 – Результати згортки кольорового зображення

Наступним шагом обробки виступає субдискретизація отриманих каналів зображення для зменшення розмірності даних. Алгоритм виконується в декілька проходів зменшуючи вхідне зображення у два рази. Ущільнення відбувається з використання функції максимуму в локальній матриці розміром 2x2.

Лістинг 2.6 – Реалізація алгоритму max pooling

```
void maxpool(const cv::Mat& src, cv::Mat& dst)
{
    int newHeight = (src.size().height - 2) / 2 + 1;
    int newWidth = (src.size().width - 2) / 2 + 1;

    cv::Mat downscaled = cv::Mat::zeros(newHeight,
                                newWidth, cv::CV_32FC);

    for (int i = 0; i < newHeight; i+=2) {
        for (int j = 0; j < newWidth; j += 2) {
              double f_val1 = std::max(src[i][j], src[i][j +
1]);
            double f_val2 = std::max(src[i + 1][j],
                                src[i + 1][j + 1]);
            downscaled[i/2][ /2] = std::max(f_val1, f_val2);
        }
    }

    downscaled.copyTo(dst);
}
```

Для подальшої обробки було створено повнозв’язну нейронну мережу з дванадцятьма (12) прихованими шарами та дев’ятьма (9) вихідними. У якості функції активації було обрано алгоритм швидкого підрахунку функції sigmoid, яку наведено у лістингу 2.7.

Лістинг 2.7 – Функція активації

```
double fastSigmoid(double x)
{
    return x / (1 + std::abs(x));
}
```

Лістинг 2.8 – Порядок обробки даних

```
std::vector<double> run_once(const cv::Mat& image,
                             std::vector<unsigned char> dump){
    const int convNumb = 8;
    const int convDepth = 1;
```

```
        const auto dnnHlayers = { 12, 9 };

        // load all weights from dump and initialize
        NetworkBuilder builder(convNumb, dnnHlayers, dump);

        cv::Mat channels[3];
        cv::split(image, channels);

        // load all convolutional kernels
        cv::Mat convKernels[3 * convNumb];
        builder.loadConvKernels(convKernels);

        cv::Mat outImgData[3 * convNumb];
        for (int convIdx = 0; convIdx < convNumb; ++convIdx) {
            outImgData[convIdx * 3 + 0] = channels[0];
            outImgData[convIdx * 3 + 1] = channels[1];
            outImgData[convIdx * 3 + 2] = channels[2];
        }

        for (int depthIdx = 0; depthIdx < convDepth; ++depthIdx){
                for (int convIdx = 0; convIdx < 3*convNumb; ++convIdx){
                    cv::Mat tempImg;
                    convolution(outImgData[convIdx], tempImg);
                    maxpool(tempImg, outImgData[convIdx]);
        }

        auto dnn = builder.buildNetwork();
        dnn.feedForward(outImgData);
        return dnn.getAwaiter().getResult();
    }
```

Результат виконання обробки одного зображення з використання згорткової нейронної мережі наведено на рисунку 2.7. Представленні значення вихідного шару нейронної мережі.

```
out: 0.89 0.12 0.23 0.18 0.06 0.4 0.31 0.09 0.28
```

Рисунок 2.7 – Результат виконання обробки

2.4 Тестування на апаратній платформі

Для дослідження було використано плату Z-turn Board на базі SoC Xilinx Zynq 7020, що включає Dual-Core ARM Cortex-A9. Перелік доступних можливостей CPU наведений у лістингу 2.9.

Лістинг 2.9 – Перелік характеристик ARM Cortex-A9

```
root#: cat /proc/cpuinfo

processor       : 0
model name      : ARMv7 Processor rev 0 (v7l)
Features: swp half thumb fastmult vfp edsp neon vfpv3 tls
vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0

processor       : 1
model name      : ARMv7 Processor rev 0 (v7l)
Features: swp half thumb fastmult vfp edsp neon vfpv3 tls
vfpd32
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x3
CPU part        : 0xc09
CPU revision    : 0
```

Вимірювання часу, що потребує для виконання той чи інший алгоритм відбувалося з використання системного таймеру високої точності, який включено до стандартної бібліотеки. Приклад функції для вимірювання часу наведено у лістингу 2.10.

Лістинг 2.10 – Вимірювання часу виконання алгоритму

```
int CpuTimer::MeasureTime(std::function<void()> func)
{
    using namespace std::chrono;
    using tmstamp = time_point<high_resolution_clock>;
    tmstamp before = high_resolution_clock::now();
    func(); // execute algorithm
    timestamp after = high_resolution_clock::now();
    return duration_cast<milliseconds>(after - before);
```

```
}
```

Таким чином, отримані результати виконання описаних алгоритмів для кольорових зображень різного розміру наведено у таблиці 2.3.1.

Таблиця 2.1 – Результати роботи алгоритмів обробки зображень

Розмір вхідного зображення	Час виконання алгоритму (мс)			
	Median filter	Grayscale	Binarization	Morph transf.
1920x1080	867	132	29	1020
1280x720	386	50	12	456
640x480	195	21	5	156
320x240	100	12	1	42

3 РОЗРОБКА МОДЕЛІ З ВИКОРИСТАННЯМ FPGA

3.1 Особливості структури систем FPGA

FPGA – програмована логічна інтегральні схема, напівпровідниковий пристрій, конфігурацію якого можна змінювати шляхом зміни логіки роботи внутрішньої принципової схеми. Для проектування складних функціональних систем існують мови опису апаратури (HDL), що дозволяють за допомогою програмного коду керувати потоком даних.

Конфігурація FPGA може бути модифікована практично у будь-який момент роботи пристрою. Інтегральна схема складається із безлічі конфігурованих логічних блоків, подібних перемикачам з великою кількістю входів та одним виходом (логічних вентилів). В цифрових схемах такі перемикачі реалізують базові двійкові операції: AND, NAND, OR, NOR, XOR.

В більшості сучасних мікропроцесорів функції логічних блоків фіксовані та не можуть змінюватися, що відрізняє їх від програмованих IC, які за допомогою спеціальних сигналів, що надсилаються до схеми можуть модифікувати топологію з’єднань в процесі роботи.

FPGA містять в собі три головних програмованих елементи:

програмовані логічні блоки (CLB);

блоки введення-виведення (I/O Pads);

внутрішні зв'язки (routing channels).

CLB є основним функціональним елементом для побудови логіки користувача, I/O Pads забезпечують зв'язок кристалу з зовнішніми пристроями, а внутрішні сигнальні лінії використовуються для сполучення елементів схеми та обміну даними між ними.

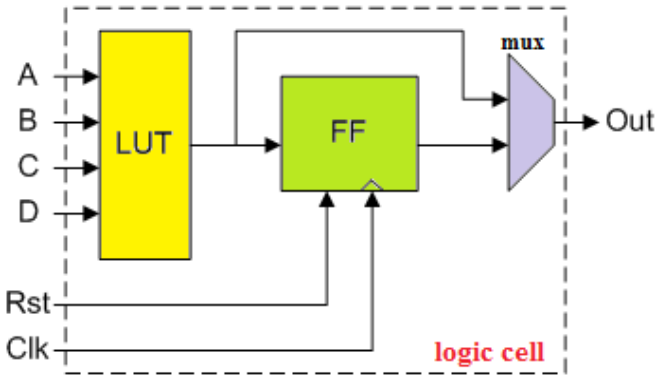
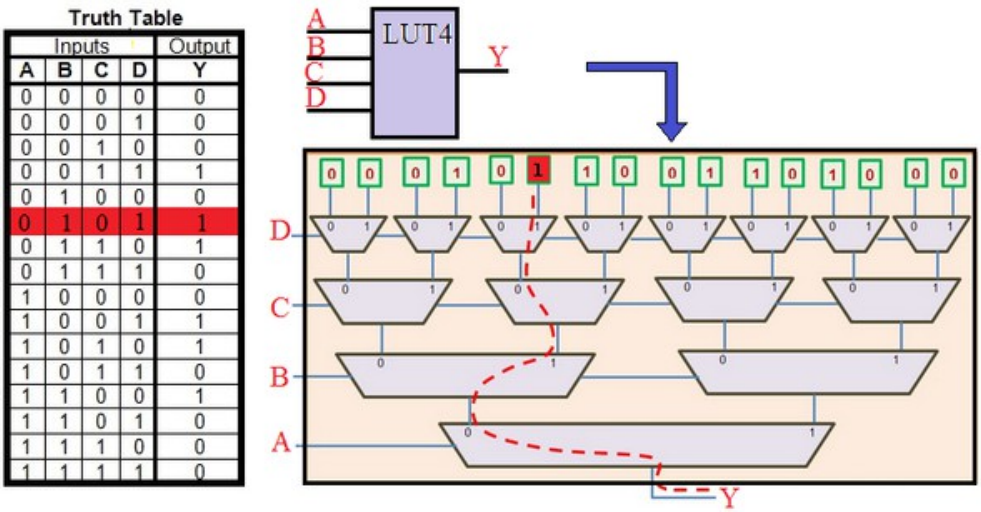


Рисунок 3.1 – Спрощена структура CLB

Логічний блок FPGA (рисунок 3.1) складається з LUT (look-up table) на 4 входи і D тригера. LUT може працювати як пам'ять 16×1 (16×1 RAM) або як 16-бітний зсувний регістр (16-bit SR), а також мультиплексор (MUX 2-to-4) і регістр (елемент пам'яті).

Елементи логічного блоку можуть бути сконфігуровані у вигляді Dтригерів, чутливих до фронту сигналу (flip-flop), або у вигляді тригерів, чутливих до рівня сигналу (latch). На рисунку 3.2 наведено приклад LUT з чотирма входами, що реалізує деяку функцію відповідно до таблиці істинності.



Truth Table

Inputs				Output
A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Рисунок 3.2 – LUT на 4 входи та реалізуюча таблиця істинності

Входи розташовані на окремих сторонах логічного блоку (рисунок 3.3), вихідний контакт може трасуватися у двох каналах: або праворуч від блоку, або знизу. Вихідні контакти кожного логічного блоку можуть з'єднуватися з трасованими сегментами в суміжних каналах. Аналогічно, контактний майданчик блоку введення-виведення (I/O Pad) може з'єднуватися з трасованим елементом у будь-якому суміжному каналі. [7]

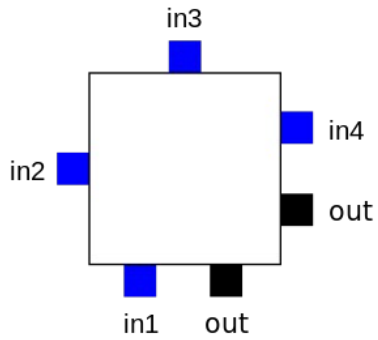


Рисунок 3.3 – Розташування контактів логічного блоку

Перемикаючі блоки створюються у місці перетину вертикальних і горизонтальних каналів. За такої архітектури для кожного провідника, що входить до блоку, існують три програмованих перемикачі, що дозволяють підключатися до трьох інших в суміжних сегментах каналу.

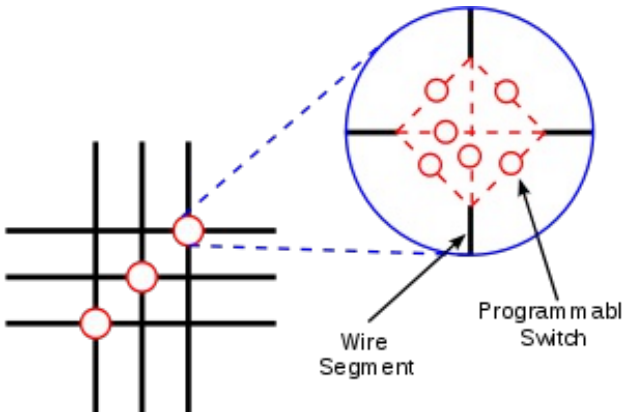


Рисунок 3.4 – Топологія перемикальних блоків

Сучасні пристрої FPGA розширюють вказані вище можливості та забезпечують вбудовану функціональність високого рівня, пропонуючи окремі блоки цифрової обробки сигналів, мультиплексори, вбудовані процесори, швидку логіка введення-виведення, вбудовану пам'ять тощо. Таким чином можна значно скоротити площу кристала та досягти навіть кращої швидкодії, ніж якщо їх створювати на базі примітивів.

3.2 Реалізація алгоритмів обробки зображень

Вхідними даними медіанного фільтру слугує двомірна матриця, відліки якої сортуються в порядку зростання, або спадання. Значення, що знаходиться в середині упорядкованого списку надходить на вихід фільтра та зберігаються у відповідному місці вихідного зображення (рисунок 3.5).

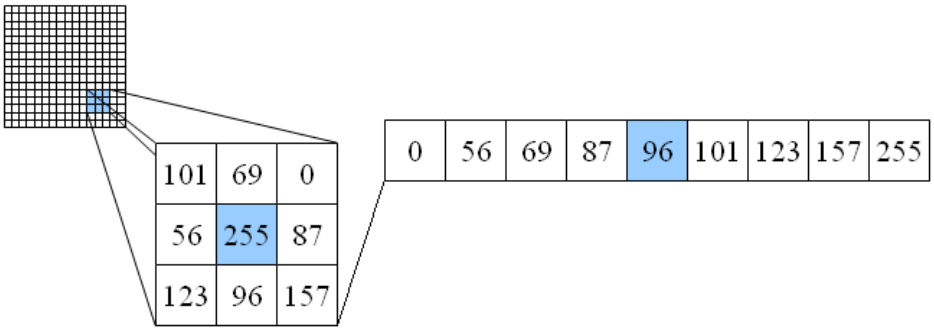


Рисунок 3.5 – Принцип роботи медіанного фільтру

Для порівняння кожного елементу матриці було розроблено схему, яка приймає на вхід два вектори, що представляють числові значення відповідно до кольору цифрового зображення та з’єднує їх із виходами в залежності від величини. Реалізація комбінаційної схеми порівняння двомірних матриць наведено у лістингу 3.1.

Лістинг 3.1 – Реалізація схеми порівняння

```
module comp_node #(
    parameter low_enabled = 1,
    parameter high_enabled = 1
)(
    input [7:0] data_a,
    input [7:0] data_b,

    output reg [7:0] data_hi,
    output reg [7:0] data_lo
);

    reg sel0;

    always @(*) begin
        if(data_a < data_b)
            sel0 = 1'b0;
        else
            sel0 = 1'b1;
    end

    always @(*) begin
        case (sel0)
            1'b0 :
            begin
                if(low_enabled)
                    data_lo = data_a;
                if(high_enabled)
                    data_hi = data_b;
            end
            1'b1 :
            begin
                if(low_enabled)
                    data_lo = data_b;
                if(high_enabled)
                    data_hi = data_a;
            end
            default :
            begin
                data_lo = 7'b0;
                data_hi = 7'b0;
            end
        endcase
    end

endmodule
```

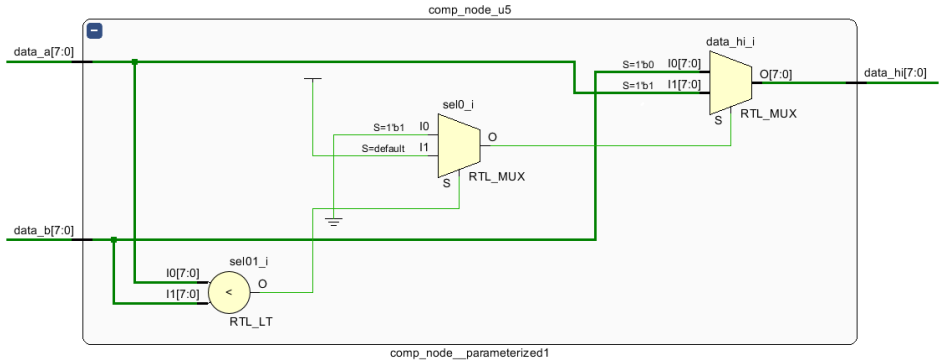


Рисунок 3.6 – RTL схема спроектованого елементу

Для визначення медіанного значення двомірної матриці розміром 3x3 потрібно, відповідно, дев’ять плюс один таких елементів (рисунок 3.7).

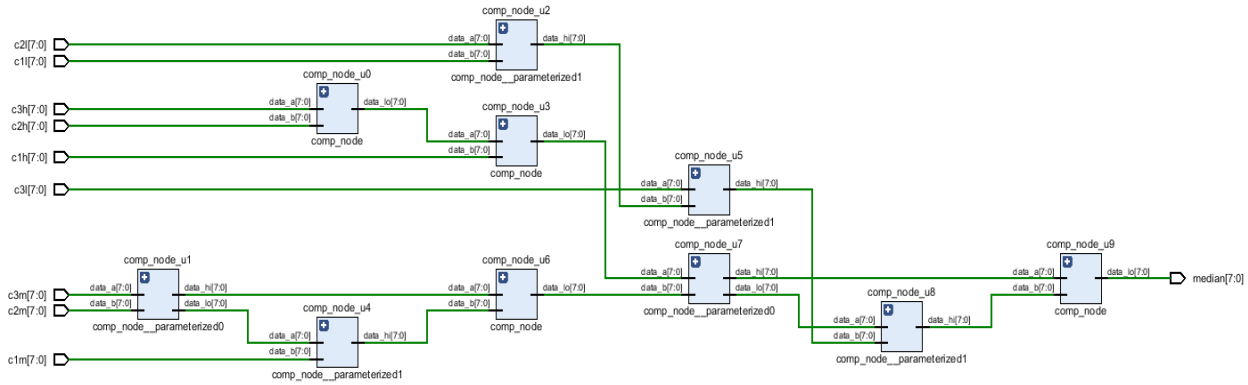


Рисунок 3.7 – RTL схема пристрою для визначення медіани

Для видалення шуму з кольорового зображення медіана знаходиться з урахуванням кожного з кольорових каналів (RGB). Таким чином, весь алгоритм не потребує додаткових циклів роботи, та виконується за допомогою комбінаційної схеми. Кінцевий результат пристрою зображено на рисунку 3.8. Результат симуляції в програмі Active-HDL (рисунок 3.9) показує, що робота фільтра з одним пікселем зображення займає один такт сигналу синхронізації.

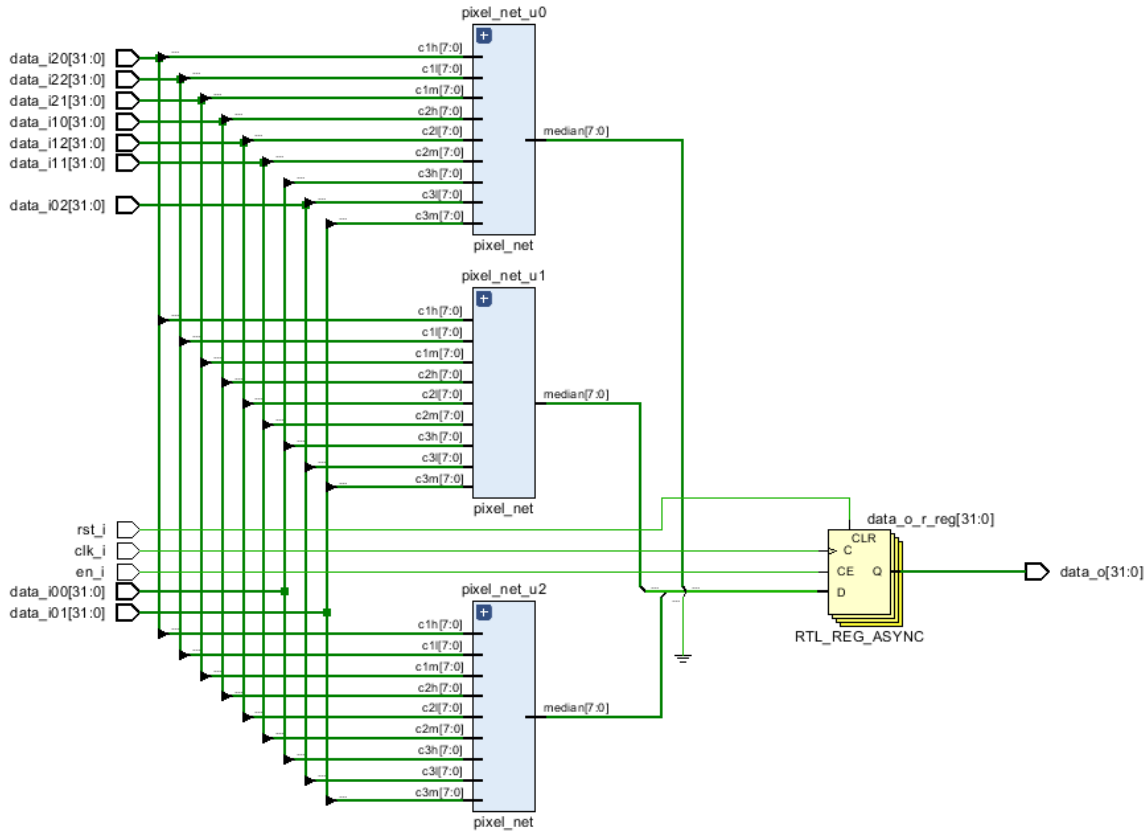


Рисунок 3.8 – RTL схема медіанного фільтру з вікном 3x3

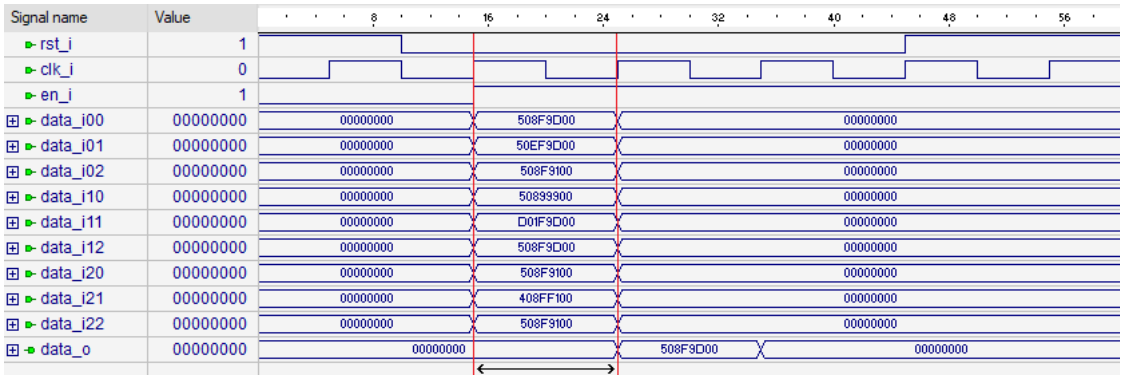


Рисунок 3.9 – Результат симуляції в програмі Active-HDL

Для переведення кольорового зображення у зображення з відтінками сірого (grayscale) достатньо знайти середнє арифметичне значень всіх каналів для кожного пікселя.

Лістинг 3.2 – Реалізація блока конвертації зображення

```
module grayscale_unit(
    input         clk_i,
    input         rst_i,
    input         en_i,
    input  [31:0]data_i,
    output [31:0]data_o
);
    wire [7:0]red_t;
    wire [7:0]green_t;
    wire [7:0]blue_t;
    reg [31:0]data_o_r;
    reg [7:0]avg_r;

    assign red_t = data_i[31:24];
    assign green_t = data_i[23:16];
    assign blue_t = data_i[15:8];

    always @(posedge clk_i or posedge rst_i) begin
        if (rst_i) begin
            data_o_r <= 32'd0;
        end
        else if (en_i) begin
            avg_r <= (red_t + green_t + blue_t) / 'd3;
            data_o_r <= {avg_r, avg_r, avg_r, 8'd0};
        end
    end

    assign data_o = data_o_r;

endmodule
```

Через операцію ділення над двома векторами загальна кількість періодів сигналу синхронізації, необхідних для отримання результату дорівнює двом (рисунок 3.10).

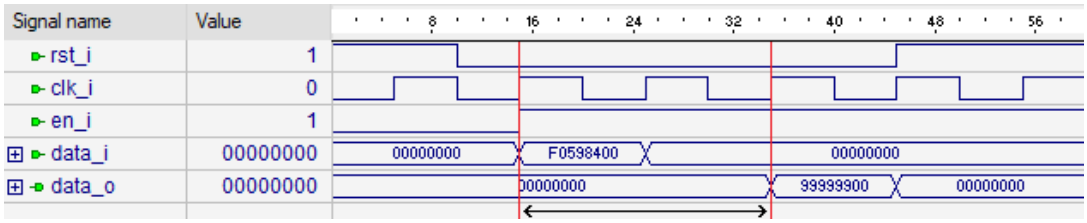


Рисунок 3.10 – Результат симуляції в програмі Active-HDL

Для отримання бінарного зображення із зображення з відтінками сірого було використано алгоритм порогування. Його основна ідея полягає в наявності одного чи декількох порогових значень. Якщо числове значення кольору на вході перевищує порогове, на виході утворюється білий колір, якщо ні – чорний.

Лістинг 3.3 – Реалізація блока бінаризації зображення

```
module gray2bin_unit(
    input        clk_i,
    input        rst_i,
    input        en_i,
    input  [31:0]data_i,
    output [31:0]data_o
);

    localparam [7:0] Threshold_High = 8'd160;
    localparam [7:0] Threshold_Low  = 8'd60;

    wire [7:0] gray_input;
    assign gray_input = data_i[31:24];

    reg [31:0]data_o_r;

    always @(posedge clk_i or posedge rst_i) begin
        if (rst_i) begin
            data_o_r <= 32'd0;
        end
        else if (en_i) begin
            if (gray_input > Threshold_Low &&
                gray_input < Threshold_High)
                data_o_r <= {8'd255, 8'd255, 8'd255, 8'd0};
            else
                data_o_r <= {8'd0, 8'd0, 8'd0, 8'd0};
        end
    end

    assign data_o = data_o_r;

endmodule
```

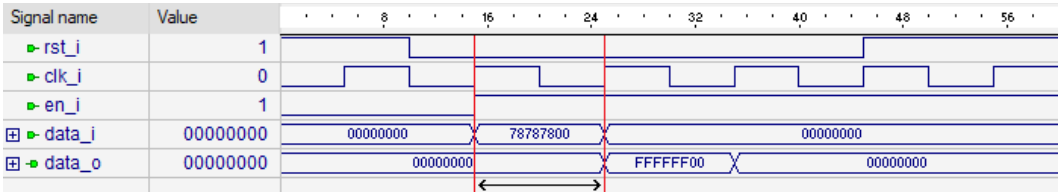


Рисунок 3.11 – Результат симуляції в програмі Active-HDL

3.3 Реалізація згорткової нейронної мережі

Операція згортки була реалізована за допомогою комбінаційної схеми перемноження двох двомірних матриць. Вхідне зображення представляється у вигляді N двомірних матриць, вихідними даними є позиція та перемножена величина. Повна реалізація модуля перемноження наведена у лістингу 3.4.

Лістинг 3.4 – Реалізація перемноження двох матриць

```
module convolution #(
  parameter BIT_LEN = `BIT_LEN,
  parameter CONV_LEN = `CONV_LEN,
  parameter CONV_LPOS = `CONV_LPOS,
  parameter M_LEN = `M_LEN,
  localparam BIT_ARRAY = BIT_LEN * M_LEN
  )(
    output[CONV_LPOS - 1:0]  o_data,
    input[BIT_LEN - 1:0]   i_dato0,
    input[BIT_LEN - 1:0]   i_dato1,
    input[BIT_LEN - 1:0]   i_dato2,
    input i_selecK_I,
    input i_reset,
    input i_valid,
    input i_CLK

    );

reg signed[BIT_ARRAY - 1:0]  kernel[0:M_LEN - 1];
reg signed[BIT_ARRAY - 1:0]  imagen[0:M_LEN - 1];
reg signed[CONV_LPOS - 1:0]  conv_reg;
reg signed[CONV_LEN - 3:0]   parcial0, parcial1;
wire signed[CONV_LEN - 1:0]   resultado;
reg signed[(2*BIT_LEN) - 1:0] array_prod[0:(M_LEN*M_LEN)-1];
```

```
    integer ptr0, ptr1, i;
    integer shift;

    assign{ o_data[CONV_LPOS - 1],o_data[CONV_LPOS - 2:0] } =
    { ~conv_reg[CONV_LPOS - 1], conv_reg[CONV_LPOS - 2:0] };

    always @(posedge i_CLK) begin
    if (i_reset) begin
    for (shift = 0; shift < M_LEN; shift = shift + 1) begin
      imagen[shift] <= {BIT_ARRAY{ 1'b0}};
      kernel[shift] <= {BIT_ARRAY{1'b0}};
    end

    conv_reg <= {CONV_LPOS{1'b0}};
    end
    else if (i_valid)begin
      case (i_selecK_I)
        1'b1: begin
      for (shift = 0; shift < M_LEN - 1; shift = shift + 1)
        imagen[shift] <= imagen[shift + 1];

      imagen[M_LEN - 1] <= {i_dato2,i_dato1,i_dato0};
       conv_reg <= resultado[CONV_LEN - 2:CONV_LEN - CONV_LPOS -
1];
    end
    1'b0: begin

    for (shift = 0; shift < M_LEN - 1; shift = shift + 1)
      kernel[shift] <= kernel[shift + 1];

    kernel[M_LEN - 1] <= {i_dato2,i_dato1,i_dato0};

    conv_reg <= conv_reg;
    end
    endcase
    end
    else begin

    for (shift = 0; shift < M_LEN; shift = shift + 1) begin:
elseequ
      imagen[shift] <= imagen[shift];
      kernel[shift] <= kernel[shift];
    end

    conv_reg <= conv_reg;
    end
    end

    always@(posedge i_CLK)
      for (i = 0; i < (M_LEN * M_LEN); i = i + 1) begin
        array_prod[i] =
          $signed(kernel[i/3][((i%3)+1)*BIT_LEN - 1 - :BIT_LEN]) *
```

```
      $signed(imagen[i/3][((i%3)+1)*BIT_LEN - 1 - :BIT_LEN]);
  end
always @(*) begin
  parcial0 = 0;
  for (ptr0 = 0; ptr0 < 4; ptr0 = ptr0 + 1)begin
    parcial0 = parcial0 + array_prod[ptr0];
  end
end

always @(*) begin
  parcial1 = 0;
  for (ptr1 = 0; ptr1 < 4; ptr1 = ptr1 + 1)begin
    parcial1 = parcial1 + array_prod[4 + ptr1];
  end
end

assign  resultado = parcial0 + parcial1 + array_prod[8];
endmodule
```

Для операції субдескритизації зображення (maxpooling) комбінаційна схема знаходить максимальне значення в межах матриць розміром 2x2. Реалізація схеми наведена у лістингу 3.5.

Лістинг 3.5 – Реалізація операції субдескритизації

```
module maxpooling(
  input clk,
  input [21:0] input1,
  input [21:0] input2,
  input [21:0] input3,
  input [21:0] input4,
  input enable,
    output reg signed [21:0] output1,
  output reg maxPoolingDone
  );

   reg [7:0] inputArray [0:7];

  reg [21:0] initialMax = 22'b1000000000000000000000;
  reg [21:0] tempOutput;

  always @ (posedge clk) begin
    if(enable) begin
      if($signed(initialMax) < $signed(input1)) begin
        if($signed(input2) < $signed(input1)) begin
          if($signed(input3) < $signed(input1)) begin
            if($signed(input4) < $signed(input1)) begin
```

```
                output1 <= input1;
                maxPoolingDone <= 1;
              end
              else begin
                output1 <= input4;
                maxPoolingDone <= 1;
              end
            end
            else begin
              if($signed(input3) < $signed(input4)) begin
                output1 <= input4;
                maxPoolingDone <= 1;
              end
              else begin
                output1 <= input3;
                maxPoolingDone <= 1;
              end
            end
          end
          else begin
            if($signed(input3) < $signed(input2)) begin
              if($signed(input4) < $signed(input2)) begin
                output1 <= input2;
                maxPoolingDone <= 1;
              end
              else begin
                output1 <= input4;
                maxPoolingDone <= 1;
              end
            end
            else begin
              if($signed(input3) < $signed(input4)) begin
                output1 <= input4;
                maxPoolingDone <= 1;
              end
              else begin
                output1 <= input3;
                maxPoolingDone <= 1;
              end
            end
          end
        end
        else begin
          output1 <= initialMax;
          maxPoolingDone <= 1;
        end
      end
      else begin
        output1 <= 0;
        maxPoolingDone <= 0;
      end
    end
```

```
endmodule
```

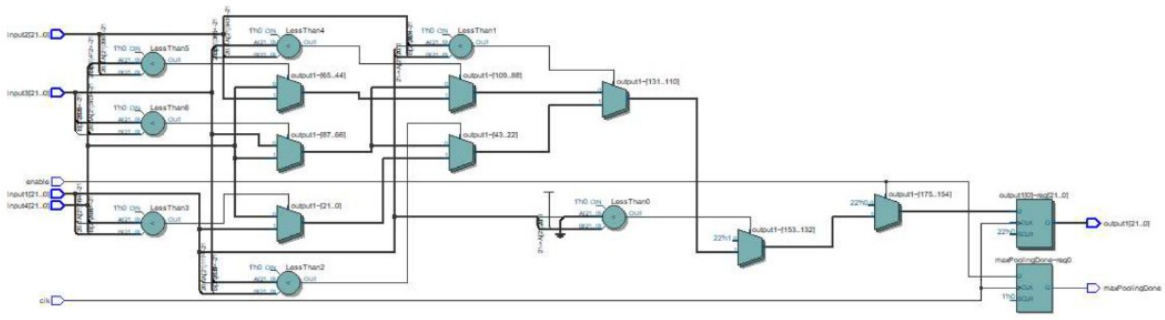


Рисунок 3.12 – RTL схема пристрою субдескритизації

У якості функції активації було використано заздалегідь прораховані значення функції sigmoid на відрізку від -2 до 2. Значення були занесені в пам'ять пристрою, задля пришвидшення роботи нейронної мережі.

Повнозв’язна нейронна мережа має вигляд схеми з безліччю суматорів на різних рівнях обчислення. Приклад реалізації одного з суматорів, що було використано для побудови мережі наведено у лістингу 3.6.

Лістинг 3.6 – Реалізація суматора

```
module adderSt2t2(
    input[4:0] input1,
    input[4:0] input2,
    input[5:0] input3,
    output reg[6:0] output1,
    input clk,
    input enable,
    output reg done
);

    always @ (posedge clk) begin
        if (enable) begin
            output1 <= { {2{input1[4]}}, input1}
         + { {2{input2[4]}}, input2} + {1'b0, input3};
            done <= 1'b1;
        end
        else begin
            output1 <= 0;
            done <= 1'b0;
        end
```

```
        end
endmodule
```

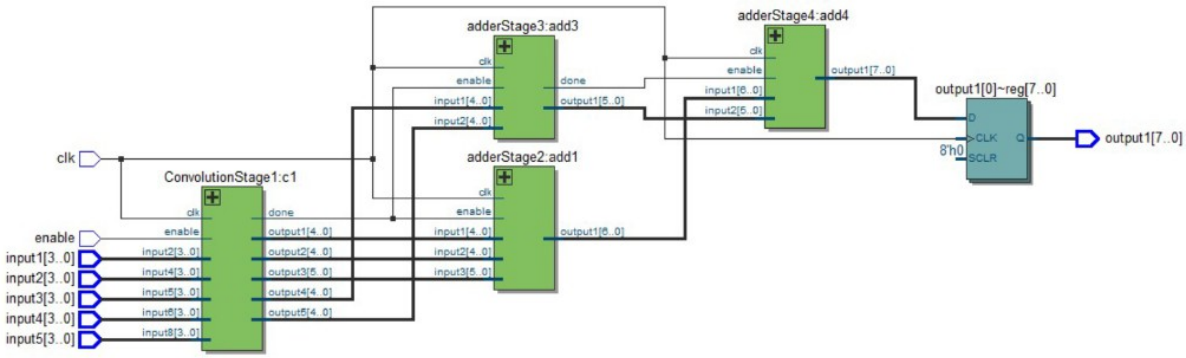


Рисунок 3.13 – RTL схема одного рівня суматорів

3.4 Додаткові підсистеми для роботи пристрою

Плата Z-turn Board на базі Xilinx Zynq7020 має SiI9022A відео трансмітер, що керується за участі PL. Трансмітер приймає дані у форматі RGB888 (24bit), але дизайн плати обмежує використання ліній даних та налаштований на роботу з форматом RGB565 (16bit).

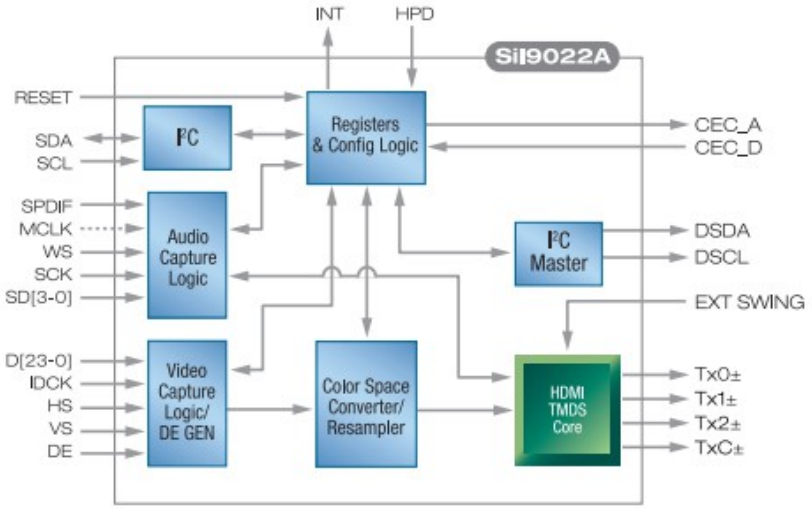


Рисунок 3.14 – Діаграма внутрішньої структури трансмітера

Для взаємодії з трансмітером було використано 2 домени сигналів синхронізації (рисунок 3.15). Генерування даних та адресу відбувається окремо один від одного, а протокол I2C використовується для роботи с VDMA.

Output Clock	Port Name	Output Freq (MHz)	
		Requested	Actual
☑ clk_out1	clk_out1	100.000	100.000
☑ clk_out2	clk_out2	148.500	148.438

Рисунок 3.15 – Налаштування сигналів синхронізації

Лістинг 3.7 – Визначення адресу даних

```
assign x_pos_r = (hsync_r&&vsync_r)?
          (x_cnt-H_SyncPulse-H_BackPorch+12'd1):12'd0;
assign y_pos_r = (hsync_r&&vsync_r)?
          (y_cnt-V_SyncPulse-V_BackPorch+12'd1):12'd0;

assign addr_data = x_pos_r+y_pos_r*H_ActivePix;
always@(posedge i_clk or negedge i_rst_n)begin
      if (i_rst_n == 1'b0) begin
               VDEN_r     <= 1'b0;
               HSYNC_r    <= 1'b0;
               VSYNC_r    <= 1'b0;
               addr_out_r <= 1'b0;
               end
     else begin
               VDEN_r     <= {hsync_r && vsync_r};
               HSYNC_r    <= hsync;
               VSYNC_r    <= vsync;
               addr_out_r <= addr_data;
     end
  end
```

Лістинг 3.8 – Передача даних до трансмітера

```
input  wire i_clk;
input  wire i_rst_n;
input  wire [31:0] data_in;
input  wire [18:0] addr_i;
output wire [15:0] data_out;
```

```
reg [15:0] data_out_r;
assign data_out = data_out_r;

always @(posedge i_clk or negedge i_rst_n)begin
      if (i_rst_n == 1'b0)
          data_out_r <= 16'd0;
      else
        data_out_r <= ((data_in[31:24] & 'b11111000) << 8) |
                      ((data_in[23:16] & 'b11111100) << 3) |
                      (data_in[15:8] >> 3);
   end
```

3.5 Тестування на апаратній платформі

Ядро ARM в системі Zynq7020 знаходиться під управлінням OS Linux. Доступ до Block RAM пам’яті кристалу відбувається за допомогою файлу пристрою (special device file), таким чином можна здійснювати запис та зчитування даних.

Лістинг 3.9 – Ініціалізація роботи з пам’яттю кристала

```
constexpr const char* device_name = "/dev/mem";
constexpr std::size_t bram_size = 0x4EC000;
constexpr u_int64_t bram_base = 0x40000000;

int fd = open(device_name, O_RDWR | O_SYNC);
if (fd != -1) {
    u_int8_t* bram8_ptr = static_cast<u_int8_t*>(mmap(nullptr,
                  bram_size, // BRAM buffer size
                  PROT_READ | PROT_WRITE, // access flags
                  MAP_SHARED, // shared for other processes
                  fd, bram_base // descriptor and base address
    ));
}
```

Для зручності при роботі з даними цифрових зображень в PS була використана бібліотека OpenCV. Код ініціалізації відео файлу та покадрового зчитування даних наведено у лістингу 3.10.

Лістинг 3.10 – Ініціалізації відео файлу з використанням OpenCV

```
std::unique_ptr<cv::VideoCapture> captr(
    new cv::VideoCapture);
captr->open(file_name);

if (captr && captr->isOpened()) {
    while (true) {
        cv::Mat frame;
        if (!m_ captr->read(frame)) {
            // Assume the file ended and restart the playback
            captr->set(cv::CAP_PROP_POS_FRAMES, 0);
            captr->read(frame);
    }
}
```

Кінцевий результат програмного забезпечення, що виконується в PS, по черзі отримує кожне зображення з відеофайлу (або камери) та передає PL для подальшої обробки і виведення на екран. Отримані результати виконання описаних алгоритмів для кольорових зображень наведено у таблиці 3.1.

Таблиця 3.1 – Результати роботи алгоритмів обробки зображень

Розмір вхідного зображення	Час виконання алгоритму (мс)		
	Median filter	Grayscale	Binarization
1920x1080	12	4	2
1280x720	4	2	1
640x480	2	1	1
320x240	2	1	1

4 АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

Під час проведення тестування були використані алгоритми цифрової обробки зображення наведені у розділах 2.2 та 3.2. Головним показником для оцінки якості роботи розробленого пристрою є швидкість обробки даних у порівнянні з іншими, у тому числі й програмними, рішеннями.

Для оцінки швидкості виконання модулів обробки зображень було використано сигнали start_i та done_o. Таким чином, підрахувавши кількість імпульсів сигналу синхронізації між цими двома подіями, було отримано час, який знадобився на повну обробку одного зображення. Результати порівняння представлені у таблиці 4.1-4.4.

Таблиця 4.1 – Порівняльна таблиця часу виконання (1920x1080)

1920x1080	Час виконання (мс)			
	CPU	GPU	ARM	FPGA
Image filter	723	86	867	12
Grayscale	110	3	132	4
Binarization	23	1	29	2

Таблиця 4.2 – Порівняльна таблиця часу виконання (1280x720)

1280x720	Час виконання (мс)			
	CPU	GPU	ARM	FPGA
Image filter	322	45	386	4
Grayscale	42	2	50	2
Binarization	10	1	12	1

Таблиця 4.3 – Порівняльна таблиця часу виконання (640x480)

640x480	Час виконання (мс)			
	CPU	GPU	ARM	FPGA
Image filter	163	22	195	2
Grayscale	18	1	21	1
Binarization	4	1	5	1

Таблиця 4.3 – Порівняльна таблиця часу виконання (320x240)

320x240	Час виконання (мс)			
	CPU	GPU	ARM	FPGA
Image filter	84	14	100	2
Grayscale	10	1	12	1
Binarization	1	1	1	1

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було досліджено різні методи цифрової обробки зображень у вбудованих системах.

Під час дослідження було розроблено алгоритми для швидкої обробки зображень в системах ARM, а також в системі на кристалі з використанням FPGA. Налагоджено інтерфейси взаємодії мікроконтролера з програмованою логікою. Спроектована та протестована архітектура згорткової нейронної мережі з багатьма оптимізаціями для пришвидшення обробки даних.

Система дозволяє завантажувати зображення до оперативної пам’яті FPGA використовуючи ресурси ядра ARM для подальшого опрацювання даних програмованою логікою та виведення через відеоінтерфейс HDMI.

Виконано порівняльний аналіз отриманих результатів тестування швидкодії алгоритмів на різних вхідних даних та системах. Результати швидкості обробки також порівнянні з програмною реалізацією, що включає дані про швидкодію алгоритмів в системі з процесором загального призначення та оптимізовані варіанти алгоритмів з використанням технологій Nvidia CUDA

З отриманих результатів можна побачити значну перевагу апаратного прискорення обробки зображень за допомогою програмованої логіки перед програмними аналогами.

Результати кваліфікаційної роботи можуть бути використані під час розробки вбудованих рішень на базі систем на кристалі, що потребують прискорення обробки великої кількості даних.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Bruecknerand, S., Parunak, H.: Swarming Distributed Pattern Detection and Classification. Lecture Notes in Computer Science, Springer Verlag, 2005. – 245 с.

2. Chen, R., Chen, G., and Chen, L.: System Design Consideration for Digital Wheelchair Controller. IEEE Transactions on Industrial Electronics, 2000. – 907 с.

3. Convolutional neural network [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Convolutional_neural_network

4. Dean J., Corrado G., Monga R., Chen K., Devin M., Mao M., Ranzato M., Senior A., Tucker P., Ke Yang, Quoc V. Le, and Andrew Y. Ng: Large scale distributed deep networks, Bartlett, F.c.n. Pereira, C.j.c. Burges, L. Bottou, and K.q. Weinberger, editors, Advances in Neural Information Processing Systems, 2012. – 1240 с.

5. Embedded system [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Embedded_system

6. Gabrick, M., Nicholson, R., Winters. F., Young, B., Patton, J.: FPGA Considerations for Automotive Applications. Proceedings of SAE World Congress and Exhibition, 2006.

7. Field Programmable Gate Arrays [Електронний ресурс] – Режим доступу до ресурсу: http://en.wikipedia.org/wiki/fpga

8. He, Kaiming: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, 2015 IEEE International Conference on Computer Vision (ICCV), 2015.

9. Kao, C.: Benefits of Partial Reconfiguration. Xcell Journal, Fourth Quarter, Xilinx, Inc., 2005. – 68 с.

10. Krizhevsky, Alex: ImageNet Classification with Deep Convolutional Neural Networks , Communications of the ACM, 2017. – 90 с.

11. Lecun, Y.: Gradient-Based Learning Applied to Document Recognition – Proceedings of the IEEE, 1998. – 2324 c.

12. Lee, D., Choi, A., Koo, J., Lee, J., Kim, B.: A Wideband DS-CDMA Modem for a Mobile Station.: IEEE Transactions on Consumer Electronics, 1999. – 1269 c.

13. Lysaght, P., Blodget, B., Mason, J., Young, J., Bridgeford, B.: Enhanced Architecture, Design Methodologies and CAD tools for Dynamic Reconfiguration for Xilinx FPGAs. Proceedings of International Conference on Field Programmable Logic and Applications, Madrid, Spain, 2006. – 367 c.

14. Mathematical morphology [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Mathematical_morphology

15. Monmasson, E., Cristea, M.: FPGA Design Methodology for Industrial Control Systems – A Review. IEEE Transactions on Industrial Electronics, 2007. – 1842 c.

16. Ovaska, S., and Vainio, O.: Evolutionary-programming-based Optimization of Reducedrank Adaptive Filters for Reference Generation in Active Power Filters. IEEE Transactions on Industrial Electronics, 2004. – 916 c.

17. Parunak, H. V. D. Brueckner, S. A., Sauter, J.: Digital Pheromones for Coordination of Unmanned Vehicles. Lecture Notes in Computer Science, Springer Verlag, 2004. – 263 c.

18. Pirsch, P., Demassieux, N., Gehrke, W.: VLSI Architectures for Video Compression - A Survey. Proceedings of IEEE, 1995. – 246 c.

19. Simonyan K., Zisserman A.: Very Deep Convolutional Networks for Large-Scale Image Recognition, Conference paper at ICLR 2015, 2015.

20. Sridharan, K., and Priya, T.: The Design of a Hardware Accelerator for Real-time Complete Visibility Graph Construction and Efficient FPGA Implementation. IEEE Transactions on Industrial Electronics, 2005. – 1187 c.

21. Sutskever I, Martens J., George E. Dahl, and Geoffrey E. Hinton: On the importance of initialization and momentum in deep learning, Proceedings of the 30th International Conference on Machine Learning, 2013. – 1147 c.

22. Valckenaers, P., T. Holvoet: An essential abstraction formanaging complexity in MASbased manufacturing. Lecture Notes in Computer Science, Springer Verlag, 2005. – 245 c.

23. Wang, J., Katz, R., Sun, J., Cronquist, B., McCollum, J., Speers, T., Plants, W.: SRAM based Reprogrammable FPGA for Space Applications. IEEE Transactions on Nuclear Science, 1999. – 1735 c.

24. Weyns., K., Holvoet, T.: Exploiting a Virtual Environment in a Real-world Application. Lecture Notes in Computer Science, Springer Verlag, 2006. – 16 c.

25. Wooldridge, M., Jenning, N.: Intelligent Agents – Theories, architectures, and Languages. Lectures Notes in Artificial Intelligence, 1995. – 62 c.