Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
освітній равень

Системне програмне забезпечення підтримки мультимедіа для мікроконтролерних систем на базі ATMega328
Назва теми

КвРКІ 20011. 20.01.02 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»-*
Назва

Виконав: студент IV курсу, група КІ2ін-20-1 _____ Оффул-Куайе Нана Ама
Підпис Ініціали, прізвище

Керівник _____ С.М. Лисенко
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ І.О. Засорнова
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«13» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"10" 01 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Нана Ама

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Системне програмне забезпечення підтримки мультимедіа для мікроконтролерних систем на базі ATMega328

Керівник проекту (роботи) Лисенко С.М., д.т.н., проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проєкту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Системи підтримки мультимедіа

Опис компонентів програмно-технічного засобу підтримки мультимедіа для мікроконтролерних систем на базі ATMega328

Реалізація системного програмного забезпечення підтримки мультимедіа для мікроконтролерних систем на базі ATMega328

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Схеми пристрою

Блок-схема алгоритму

Блок-схема алгоритму

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Засорнова І.О., к.т.н., доцент		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_10_»_____01_____2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2024	виконано
4	Робота над розділом 2 – вибір компонентів для проєктування системи адаптивного застосування моніторингових елементів розвідувального БПЛА	01.04.2024	виконано
5	Робота над розділом 3 – проєктування системи адаптивного застосування моніторингових елементів розвідувального БПЛА	29.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2024	виконано
7	Попередній захист ВКР	26.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент _____ Нана Ама
Підпис Ініціали, прізвище

Керівник роботи _____ С.М. Лисенко
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 20011. 20.01.02 ПЗ	Пояснювальна записка	67		
			Графічні матеріали			
2		КвРКІ 20011. 20.01.02 Е8	Схеми пристрою	1		
3		КвРКІ 20011. 20.01.02 Е8	Блок-схема алгоритму	1		
4		КвРКІ 20011. 20.01.02 Е8	Блок-схема алгоритму	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 20011. 20.01.02 ВП		
Розробив		Нана Ама			Відомість проекту	Літера	Аркуш 1 / Аркушів 1
Перевір.		Лисенко				У	
Н. контр.		Засорнова				ХНУ, КІ2ін-20-1	
Затв.		Говорущенко		13.06			

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Системне програмне забезпечення підтримки мультимедіа для мікроконтролерних систем на базі ATMega328».

Автор роботи: Оффул-Куайе Нана Ама.

Керівник роботи: Лисенко Сергій Сергійович.

Пояснювальна записка: 67 с., 32 рисунки, 3 таблиці, 4 додатки, 55 джерел.

Графічна частина: 3 рисунки.

СИСТЕМА МУЛЬТИМЕДІЙНОГО ЗАБЕЗПЕЧЕННЯ, СИСТЕМНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, МІКРОКОНТРОЛЕР, МІКРОКОНТРОЛЕРНА СИСТЕМА, ATMEGA328.

Метою дипломної роботи є розробка системного програмного забезпечення мультимедійної підтримки мікроконтролерних систем на базі ATMega328.

Об'єктом дослідження є програмне забезпечення системи підтримки мультимедіа.

Предметом дослідження є програмне забезпечення системи підтримки мультимедіа для мікроконтролерних систем на базі ATMega328.

Під час проведення даного дослідження використовувався метод систематичного огляду літератури для вивчення та аналізу предметної області даного дослідження з текстових джерел інформації.

_____ ___10.06.2024___

Підпис студента Дата

ЗМІСТ

ВСТУП

Актуальність системного програмного забезпечення для підтримки мультимедіа для мікроконтролерних систем на базі ATMega328 є цікавою темою, враховуючи останні досягнення в області вбудованих систем і зростаючий попит на інтеграцію мультимедійних функцій в різні додатки.

Системи на базі ATMega328 часто використовуються в побутовій електроніці, де достатньо базових мультимедійних можливостей, таких як відтворення аудіо та прості графічні дисплеї. Сюди відносяться такі додатки, як

- Цифрові годинники і годинники з графічним інтерфейсом.

- Пристрої домашньої автоматизації з дисплеєм і звуковими сповіщеннями.

- Іграшки та освітні набори, які використовують звук і візуальні ефекти для інтерактивного навчання.

Підтримка мультимедіа в пристроях IoT може покращити взаємодію з користувачем. Наприклад:

- Пристрої розумного будинку можуть використовувати звукові сповіщення або прості графічні дисплеї для передачі статусу або повідомлень.

- Технології, що носяться, можуть отримати вигоду від невеликих графічних дисплеїв для користувацьких інтерфейсів.

ATMega328 широко використовується в прототипуванні та освітніх проектах. Підтримка мультимедіа дозволяє студентам і любителям створювати більш цікаві і складні проекти, покращуючи навчальний процес за рахунок додавання візуальних і звукових елементів.

Незважаючи на ці обмеження, творчі прийоми, такі як оптимізація коду, використання зовнішньої пам'яті та ефективних бібліотек, можуть забезпечити базову мультимедійну функціональність.

Низьке енергоспоживання ATMega328 робить її придатною для пристроїв, що працюють від батарейок, але мультимедійними функціями потрібно ретельно керувати, щоб уникнути надмірного споживання енергії.

Хоча ATMega328 забезпечує базову підтримку мультимедіа, існує тенденція до інтеграції цих мікроконтролерів з більш потужними системами, такими як:

- Об'єднання з мікроконтролерами на базі ARM. Об'єднання з більш потужними мікроконтролерами може розвантажити важкі мультимедійні завдання, зберігаючи при цьому переваги дешевизни і простоти ATMega328.

- Використання співпроцесорів. Розвантаження обробки мультимедійних даних на спеціальні мікросхеми або співпроцесори може підвищити продуктивність, не перевантажуючи ATMega328.

Оскільки IoT продовжує зростати, потреба в недорогій, ефективній і базовій мультимедійній підтримці в невеликих пристроях залишається високою. Прості дисплеї та звукові сповіщення покращують користувацькі інтерфейси розумних пристроїв без значного збільшення вартості або складності.

Розробники повинні вміти оптимізувати код і керувати обмеженими ресурсами. Це може включати в себе:

- Використання зовнішнього сховища для великих мультимедійних файлів.

- Впровадження ефективних практик кодування для максимального використання доступної пам'яті та обчислювальної потужності.

Хоча мультимедіа у вбудованих системах є затребуваним, зазвичай це стосується простої графіки з низькою роздільною здатністю та базового звуку. Складні мультимедійні завдання, як правило, вимагають більш потужного обладнання.

ATMega328 залишається актуальним і практичним вибором для мікроконтролерних систем, де потрібна базова підтримка мультимедіа. Низька вартість, широка доступність і велика підтримка спільноти роблять його гарним вибором для освітніх цілей, побутової електроніки та простих пристроїв IoT. Однак розробники повинні пам'ятати про його обмеження і застосовувати творчі рішення для досягнення бажаної функціональності.

1 СИСТЕМИ ПІДТРИМКИ МУЛЬТИМЕДІА

1.1 Поняття про відеокодеки

Відеокодек (від англ. Video Codec) - це програмне або апаратне забезпечення, призначене для кодування і декодування відеоданих. Кодек визначає, як відеосигнал буде стискатися під час запису (кодування) і декодуватися під час відтворення або обробки. Такий підхід дозволяє зменшити обсяг відеофайлів і заощадити смугу пропускання при передачі відеоданих по мережі.

Розглянемо основні аспекти роботи відеокодеків.

Кодування. Процес стиснення відеоданих перед зберіганням або передачею. Кодек аналізує відеосигнал і використовує різні методи стиснення для зменшення розміру файлу.

Декодування (Decoding). Процес відновлення відеосигналу з закодованого формату під час відтворення або обробки відео. Декодер декодує стиснутий файл і відтворює оригінальний відеосигнал.

Стиснення з втратами та без втрат. Багато відеокодеків використовують стиснення з втратами, що означає втрату певної якості для досягнення високого ступеня стиснення. Існують також відеокодеки без втрат, які зберігають відеосигнал без втрати якості, але файл може бути більшим.

Швидкість передачі даних (бітрейт). Це кількість біт, що використовується для представлення одиниці відео часу. Визначає якість і розмір відеофайлу. Вищий бітрейт може покращити якість, але збільшить розмір файлу.

Формати відео. Відеокодеки можуть підтримувати різні відеоформати, такі як H.264, H.265 (HEVC), VP9, AV1 та інші. Кожен формат має свої переваги і недоліки, включаючи якість стиснення, підтримку різних пристроїв і пропускну здатність.

Підтримка апаратного прискорення. Деякі відеокодеки можуть використовувати апаратне прискорення, що дозволяє знизити навантаження на процесор і підвищити продуктивність під час відтворення відео.

Використання відеокодеків необхідне в багатьох сферах, таких як відеозапис, потокові сервіси, відеоконференції, мультимедійні пристрої та ігрові системи. Вибір конкретного відеокодека залежить від конкретних вимог та умов використання.

1.2 Актуальність використання відеокодеків

Розглянемо актуальність використання відеокодеків у сучасному світі з огляду на низку ключових факторів.

З ростом популярності потокових платформ, таких як Netflix, YouTube, Hulu та інших, відеокодеки відіграють важливу роль у забезпеченні ефективної передачі відеоданих мережею.

Зростання віддалених форм роботи і спілкування робить важливим використання ефективних відеокодеків для передачі відеосигналу в реальному часі.

Відеокодеки використовуються для стиснення відеопотоків у відеоіграх, а також для трансляції відеоігор через мережу.

Відеокодеки, вбудовані в медіаплеєри та телевізори, дозволяють відтворювати відео різних форматів без необхідності внутрішніх перетворень.

З розвитком цифрових технологій і зростанням обсягів відеоконтенту важливо використовувати ефективні відеокодеки для економії місця на диску та забезпечення швидкого доступу до відеоданих.

У зв'язку з популярністю відеозапису та перегляду відео на мобільних пристроях, важливо мати ефективні відеокодеки, які дозволяють забезпечити якісну передачу потокового відео з обмеженими ресурсами.

Відеокодеки використовуються у виробництві та монтажі відео для зберігання та обробки великих обсягів відеоматеріалу.

Відеокодеки відіграють ключову роль у передачі великих обсягів відеоданих для віртуальної і доповненої реальності, де важлива висока якість і плавність.

Актуальність відеокодеків зростає, оскільки вони впливають на якість відеоконтенту, ефективність передачі через мережі, розмір файлів і загальний користувацький досвід перегляду та використання кастомізованого відео.

1.3 Кодування та декодування відео

Кодування (Encoding) і декодування (Decoding) відеокодеків - це процеси, які визначають, як кодується відеоінформація під час створення файлу (кодування) і як вона відновлюється для відтворення або обробки відео (декодування).

Кодування (або стиснення) відео полягає в перетворенні великого обсягу відеоданих у менший обсяг.

Це робиться за допомогою різних спеціалізованих методів стиснення, таких як видалення надлишкових деталей, використання передбачення, квантування тощо.

Зменшення обсягу відеофайлу, що забезпечує ефективність передачі по мережі і оптимальне використання простору для зберігання інформації.

Для кодування використовуються кодеки H.264, H.265 (HEVC), VP9, AV1 та інші.

Декодування відео - це процес відновлення вихідного відеосигналу з кодованого формату під час відтворення або обробки.

Декодер розшифровує стиснутий файл, відтворюючи оригінальний відеосигнал.

Призначений для відтворення або обробки відео в зрозумілому і доступному форматі.

Декодери використовують ті ж самі або сумісні відеокодеки, які використовувалися для кодування.

Успішне кодування за допомогою відеокодека забезпечує високий ступінь стиснення, зберігаючи при цьому якість відео на прийнятному рівні. Декодування

відео полягає в розшифровці стиснених даних для відтворення оригінального відеосигналу для перегляду або обробки.

Сумісність між кодеками має важливе значення, забезпечуючи ефективну взаємодію між різними пристроями та програмами для кодування і декодування користувацького відео.

Багато сучасних відеокодеків, таких як H.264 і H.265, стали стандартами в багатьох додатках, що забезпечує високу сумісність і ефективність.

1.4 Актуальність використання звукових кодеків

Отже, розглянемо сфери застосування відеокодеків в Інтернеті речей (IoT), оскільки IoT охоплює широкий спектр застосувань, включаючи відеоспостереження, медичні пристрої, системи домашньої автентифікації, транспортні системи та інші.

Відеокодеки важливі для передачі і зберігання великих обсягів відеоданих, що генеруються системами відеоспостереження в системах безпеки IoT.

Використання відеокодеків дозволяє передавати відео з медичних пристроїв для віддалених консультацій, навчання і моніторингу стану і поведінки пацієнтів.

Відеокодеки можна використовувати для передачі потокового відео на різноманітні пристрої в домашньому середовищі, включаючи смарт-телевізори, відеопроектори та ігрові консолі.

Відеокодеки важливі для відеомоніторингу та взаємодії з транспортними системами, такими як камери спостереження на дорогах або в громадському транспорті.

Відеокодеки дозволяють передавати і відтворювати відеоконтент на інтерактивних пристроях та ігрових консолях в якості розважального елемента Інтернету речей.

Відеокодеки використовуються для передачі відеоданих з домашніх камер безпеки для віддаленого моніторингу та охорони.

З огляду на обмеженість ресурсів в пристроях IoT, важливо мати ефективні відеокодеки, які дозволяють стискати і передавати відео з мінімальним споживанням енергії.

Відеокодеки використовуються в системах відеомоніторингу та аналітики в розумних містах для підвищення безпеки та ефективності управління.

З огляду на різноманітність застосувань в Інтернеті речей, важливо мати оптимальні відеокодеки, які дозволяють передавати відео високої якості при обмежених ресурсах і умовах мережі.

Процес стиснення даних у відеокодеках включає в себе ряд прийомів і методів для зменшення обсягу відеофайлу без значної втрати якості зображення. Цей процес може бути з втратами або без втрат, залежно від того, чи втрачається якість під час стиснення.

Основні етапи стиснення даних у відеокодеках включають описані нижче аспекти.

Видалення надлишкових деталей і повторюваних патернів у просторовому вимірі. Застосовуються такі методи, як дедуплікація, квантування та фільтрація.

Використовується для зменшення часової надмірності між кадрами. Виділення міжкадрової залежності та використання прогнозування для передбачення майбутніх кадрів.

Застосування різних алгоритмів для кодування даних з втратою інформації. Одним з найпоширеніших методів є використання DCT (дискретного косинусного перетворення), яке застосовується, наприклад, у стандартах JPEG і H.264.

Процес квантування знижує точність представлення значення шляхом відкидання менш важливих бітів. Це ключовий крок у стисненні з втратами.

Використання різних типів кадрів, таких як ключові (I-кадри), очікувані (P-кадри) і зворотні (B-кадри), для подальшого зменшення обсягу даних.

Визначення та врахування руху об'єктів у відеосигналі для ефективного кодування зміни кадрів.

Застосування алгоритмів, які передбачають вміст кадрів, а також методів інтерполяції для зменшення кількості інформації, яку необхідно зберігати.

Ці методи сприяють створенню ефективних відеокодеків, які забезпечують баланс між обсягом даних і якістю відеозображення.

Стиснення з втратами є поширеним підходом у кодуванні відео, оскільки дозволяє значно зменшити розмір файлу і швидкість передачі даних, що важливо для потокового передавання і зберігання великих обсягів інформації.

1.5 Типи кодеків

Існує багато типів відеокодеків, кожен з яких має свої особливості, переваги та недоліки. Ось деякі з найпоширеніших відеокодеків.

H.264 (AVC) - це один з найпопулярніших стандартів кодування відео, який використовується для відео на YouTube, Vimeo, відеоконференцій та інших платформах. Забезпечує хорошу якість при низькому бітрейті.

Стандарт, що прийшов на зміну H.264 і забезпечує краще стиснення при збереженні високої якості відео. HEVC добре підходить для 4K і вищих роздільних здатностей.

Розроблений компанією Google, VP9 є відкритим стандартом з відмінним стисненням і підтримкою високої роздільної здатності. Використовується, зокрема, у відео на YouTube.

Ще один відкритий стандарт, розроблений групою Alliance for Open Media. AV1 пропонує високий рівень стиснення зі збереженням якості і призначений для використання в різних сценаріях, включаючи потокове мовлення та медіаплеєри.

Стандарт, який використовується для стиснення відео та аудіо для DVD, телевізійного мовлення та інших відеоформатів.

Використовується для стиснення відео для Інтернету та мобільних пристроїв. MPEG-4 має багато різновидів кодеків, таких як DivX і Xvid.

Відомий своїм високим ступенем стиснення та використанням у відеофайлах високої якості.

Xvid - популярний вільно розповсюджуваний кодек стиснення відео.

AppleProВикористовується переважно у професійному відеомонтажі та виробництві відеоконтенту на продуктах Apple.

Це лише кілька прикладів, існує багато інших відеокодеків, які використовуються в різних сферах, включаючи мовлення, стрімінг, відеоігри, відеопродакшн тощо.

Вибір конкретного кодека може залежати від потреб користувача, специфіки відеопроектів і платформи використання.

1.6 Відомі системні бібліотеки програмного забезпечення для кодування та декодування відеокодеків

Існує декілька відомих системних бібліотек програмного забезпечення, які надають функції кодування та декодування відеокодеків.

Деякі з найпопулярніших бібліотек у цій галузі включають системне програмне забезпечення, описане нижче.

FFmpeg - одна з найпотужніших і найпоширеніших бібліотек для обробки відео та аудіо. Вона надає інструменти для кодування, декодування, обрізки, конвертації та інших маніпуляцій з відео та аудіо даними.

OpenCV (Open Source Computer Vision Library) - це бібліотека комп'ютерного зору, але вона також надає функціональність для роботи з відео. OpenCV має можливості кодування та декодування відео за допомогою різних кодеків.

libavcodec - специфічний компонент FFmpeg, що відповідає за кодування та декодування аудіо та відео. Ця бібліотека містить велику кількість кодеків, зокрема H.264, H.265, VP9 та інші.

x264 та x265 - бібліотеки для кодування відео у стандартах H.264 та H.265 відповідно. Вони широко використовуються для створення відео з високим ступенем стиснення та якості.

VideoToolbox - фреймворк, який забезпечує апаратну підтримку кодування та декодування відео на пристроях Apple, таких як Mac та iPhone.

MediaCodec - це API для роботи з мультимедійними даними на платформі Android. Дозволяє використовувати апаратну підтримку кодування та декодування відео на пристроях Android.

VLC (VideoLAN Client) використовує бібліотеку libavcodec для реалізації кодеків та обробки мультимедійних даних.

Ці бібліотеки надають розробникам інструменти для роботи з різними відеокодеками та реалізації різних операцій над відеоданими. Вибір бібліотеки може залежати від конкретних потреб проекту, платформи та мови програмування.

1.7 Актуальність трансляції відео з використанням мікроконтролерів для систем Інтернету речей (IoT)

Використання потокового відео з використанням мікроконтролерів для систем Інтернету речей (IoT) може бути актуальним у багатьох сценаріях, де важливо віддалено контролювати або спостерігати за подіями, об'єктами або просторами.

Потокове відео можна використовувати для створення систем відеоспостереження для віддаленого моніторингу та безпеки об'єктів або територій, таких як будинки, офіси або об'єкти інфраструктури.

Передача відео може використовуватися для віддаленого спостереження за пацієнтами, проведення телемедицини або навіть проведення віртуальних консультацій.

У виробничих умовах потокове відео можна використовувати для віддаленого моніторингу робочих процесів, стану обладнання або навіть для навчання персоналу.

За допомогою передачі відео можна віддалено контролювати і спостерігати за домашніми процесами, наприклад, за безпекою або доглядом за тваринами.Потокове відео можна використовувати для моніторингу дорожнього руху, паркування або навіть для створення систем віртуального сприйняття для автономних транспортних засобів.

Потокове відео можна використовувати в побутових електронних пристроях, таких як домашні відеокамери для віддаленого моніторингу з мобільних пристроїв.

У сферах віртуальної та доповненої реальності потокове відео можна використовувати для створення ефекту занурення та взаємодії.

Потокове відео може бути важливим для систем громадської безпеки, таких як відеоспостереження на вулицях або в громадському транспорті.

Актуальність використання передачі відео за допомогою мікроконтролерів в системах IoT полягає в можливості створення ефективних та інтелектуальних рішень для віддаленого моніторингу та управління різними об'єктами або процесами в режимі реального часу.

Мікроконтролер ATMega328, що використовується в платформі Arduino Uno, має ресурси і обсяг пам'яті, які роблять його придатним для складних операцій з відеокодеками і відтворення або кодування відео в реальному часі.

Однак, ви можете використовувати деякі операції з відеокодеками та експериментувати зі стисненням і відтворенням відео, використовуючи цей мікроконтролер.

Розглянемо загальний підхід до роботи з відеокодеками на базі ATMega328.

Оскільки ATMega328 має обмежені ресурси, слід обирати відеокодеки, які підтримують низьку роздільну здатність і можуть бути реалізовані на цьому мікроконтролері. Наприклад, з роздільною здатністю QVGA (320x240) або ще нижчою.

Для взаємодії з відеокодеками, підтримуваними ATMega328, можна використовувати готові бібліотеки. Наприклад, деякі бібліотеки можуть працювати з JPEG кодеками для стиснення та розпакування зображень.

Вам слід звернути особливу увагу на оптимізацію вашого коду, щоб мінімізувати використання ресурсів мікроконтролера. Зменшення обсягу пам'яті та оптимізація швидкості виконання будуть важливими факторами.

Враховуючи обмежені можливості ATMega328, ви також можете розглянути можливість використання зовнішніх модулів або допоміжних мікроконтролерів для обробки відеоданих, залишаючи ATMega328 відповідальним за управління та інші функції. Для зменшення об'єму відеоданих необхідно обрати відеокодеки та налаштування, які дозволяють працювати з низьким бітрейтом. Важливо зазначити, що реалізація відеокодеків на таких мікроконтролерах, як ATMega328, є складним завданням через їх обмежені ресурси.

1.8 Висновок

У розділі розглянуто аспекти звукових кодеків, висвітливши їх значення, функціональність та застосування в сучасних цифрових системах.

Звукові кодеки мають вирішальне значення для перетворення аналогових аудіосигналів у цифровий формат і навпаки. Вони сприяють ефективній передачі та зберіганню аудіоданих, стискаючи їх без значної втрати якості. У розділі підкреслюється важливість розуміння звукових кодеків як будівельних блоків сучасних аудіотехнологій. У розділі детально розглядаються звукові кодеки, їхнє значення та застосування, зокрема у сфері Інтернету речей. Обговорюються технічні аспекти стиснення аудіоданих, типи доступних кодеків і практична реалізація аудіофункцій у сучасних цифрових системах з використанням мікроконтролерів, таких як ESP8266. Це всебічне дослідження не лише поглиблює розуміння звукових кодеків, але й підкреслює їхню важливу роль в еволюції цифрових аудіотехнологій та Інтернету речей.

2 ОПИС КОМПОНЕНТІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ПІДТРИМКИ МУЛЬТИМЕДІА ДЛЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ НА БАЗІ АТМЕГА328

2.1 ATmega 328

Мікроконтролери ATmega - це малопотужні 8-розрядні CMOS-пристрої, побудовані на основі передової RISC-архітектури AVR®.

Ці мікроконтролери виконують інструкції за один такт, що дозволяє їм досягати пропускної здатності процесора, близької до мільйона інструкцій в секунду (MIPS) на мегагерц. Така продуктивність дозволяє розробникам систем ефективно балансувати між енергоспоживанням і швидкістю обробки даних.

2.2 Конфігурації виводів

ATmega328 має виводи, як показано на рисунку 2.1. Детальна інформація про розташування виводів для ATmega328 наведена в таблиці 2.1.

Таблиця 2.1 - Розподіл виводів ATmega328

	1	2	3	4	5	6
A	PD2	PD1	PC6	PC4	PC2	PC1
B	PD3	PD4	PD0	PC5	PC3	PC0
C	GND	GND			ADC7	GND
D	VDD	VDD			AREF	ADC6
E	PB6	PD6	PB0	PB2	AVDD	PB5
F	PB7	PD5	PD7	PB1	PB3	PB4

Порт B - це 8-розрядний двонаправлений порт вводу/виводу з внутрішніми підтягуючими резисторами, які можна вибрати окремо для кожного біта. Вихідні буфери порту B мають симетричні характеристики накопичувача, що забезпечує сильні можливості приймача та джерела. При використанні в якості входів виводи

порту B будуть джерелом струму, якщо на них подається низький рівень ззовні і підтягуючі резистори увімкнені.

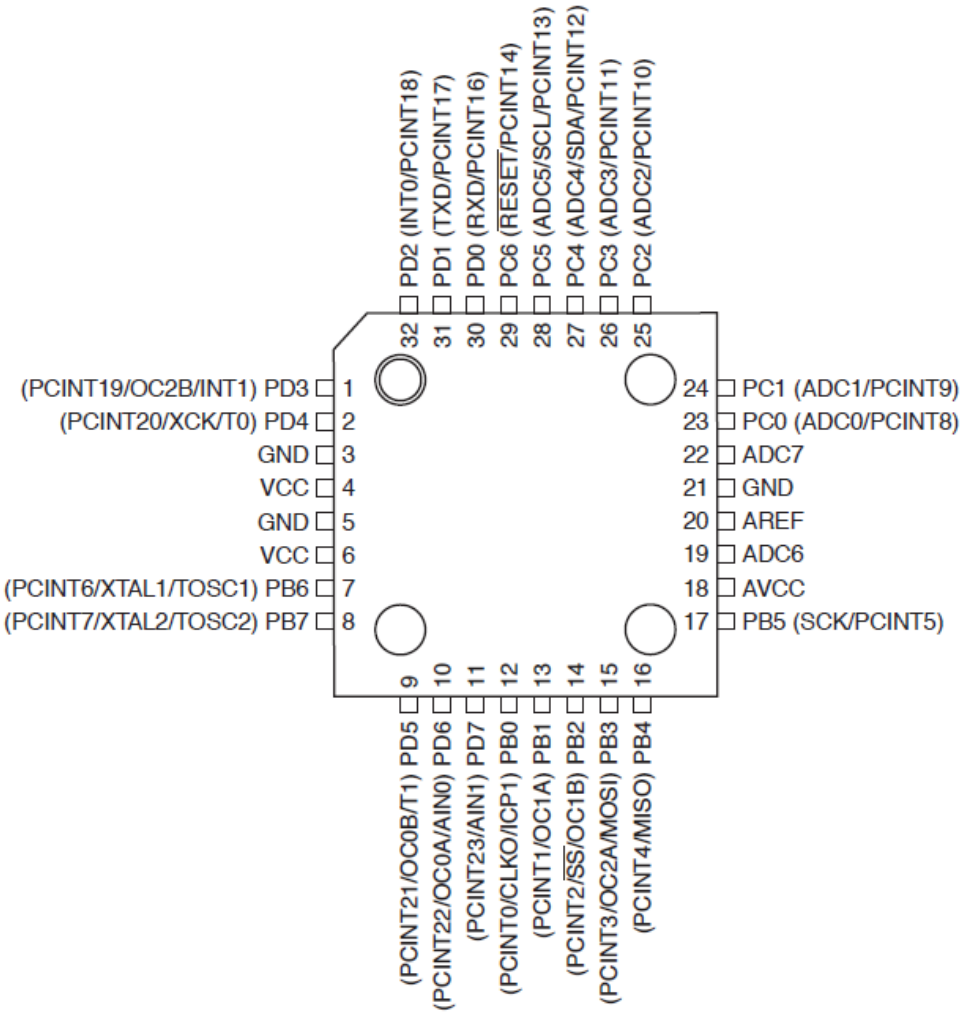


Рисунок 2.1 - Конфігурація виводів

Під час скидання виводи порту B переходять у високоімпедансний стан (тристадійний) незалежно від стану тактового генератора.

Залежно від налаштувань запобіжника вибору тактового генератора, PB6 може служити входом для підсилювача інвертуючого генератора і входом для внутрішньої схеми роботи тактового генератора. Аналогічно, PB7 можна налаштувати як вихід з підсилювача інвертуючого генератора на основі налаштувань запобіжника вибору тактового генератора.

Коли внутрішній калібрований RC-генератор обрано джерелом тактового сигналу мікросхеми, PB7 і PB6 слугують входами TOSC2 і TOSC1 для асинхронного таймера/лічильника2, за умови, що біт AS2 в ASSR встановлений.

Порт C - це 7-розрядний двонаправлений порт вводу/виводу з внутрішніми підтягуючими резисторами, які можна вмикати окремо для кожного біта. Вихідні буфери для виводів PC5 - PC0 мають симетричні характеристики накопичувача, що забезпечує сильні можливості для скидання та отримання даних. Якщо виводи порту C сконфігуровані як входи, то вони стають джерелом струму, якщо на них ззовні подати низький рівень і увімкнути підтягуючі резистори. У разі скидання, виводи порту C переходять у високоімпедансний стан (тристани), навіть якщо тактовий генератор неактивний.

Коли запрограмовано запобіжник RSTDISBL, PC6 функціонує як вивід вводу/виводу. Важливо зазначити, що електричні характеристики PC6 відрізняються від характеристик інших виводів порту C. Якщо запобіжник RSTDISBL не запрограмовано, PC6 працює як вхід скидання. Утримання цього виводу на низькому рівні довше, ніж мінімальна тривалість імпульсу, призведе до скидання, незалежно від того, чи працює годинник.

Порт D - це 8-бітний двонаправлений порт вводу/виводу з внутрішніми підтягуючими резисторами (підібраними для кожного біта). Вихідні буфери порту D мають симетричні характеристики накопичувача з високими можливостями як приймача, так і джерела. Як входи, виводи порту D, які ззовні затягнуті на низький рівень, будуть джерелом струму, якщо активовані підтягувальні резистори. Виводи порту D стають тристабілізованими, коли стає активним стан скидання, навіть якщо годинник не працює.

AVCC слугує виводом напруги живлення для АЦП, а також для виводів PC3:0 і ADC7:6. Рекомендується встановити зовнішнє з'єднання між AVCC і VCC, незалежно від того, чи використовується АЦП. У разі використання АЦП, з'єднання між AVCC і VCC слід встановити через фільтр нижніх частот. AREF функціонує як аналоговий опорний вивід для АЦП.

2.2 Структурна схема ATmega328

На рисунку 2.2 показано структурну схему мікроконтролера ATmega328.

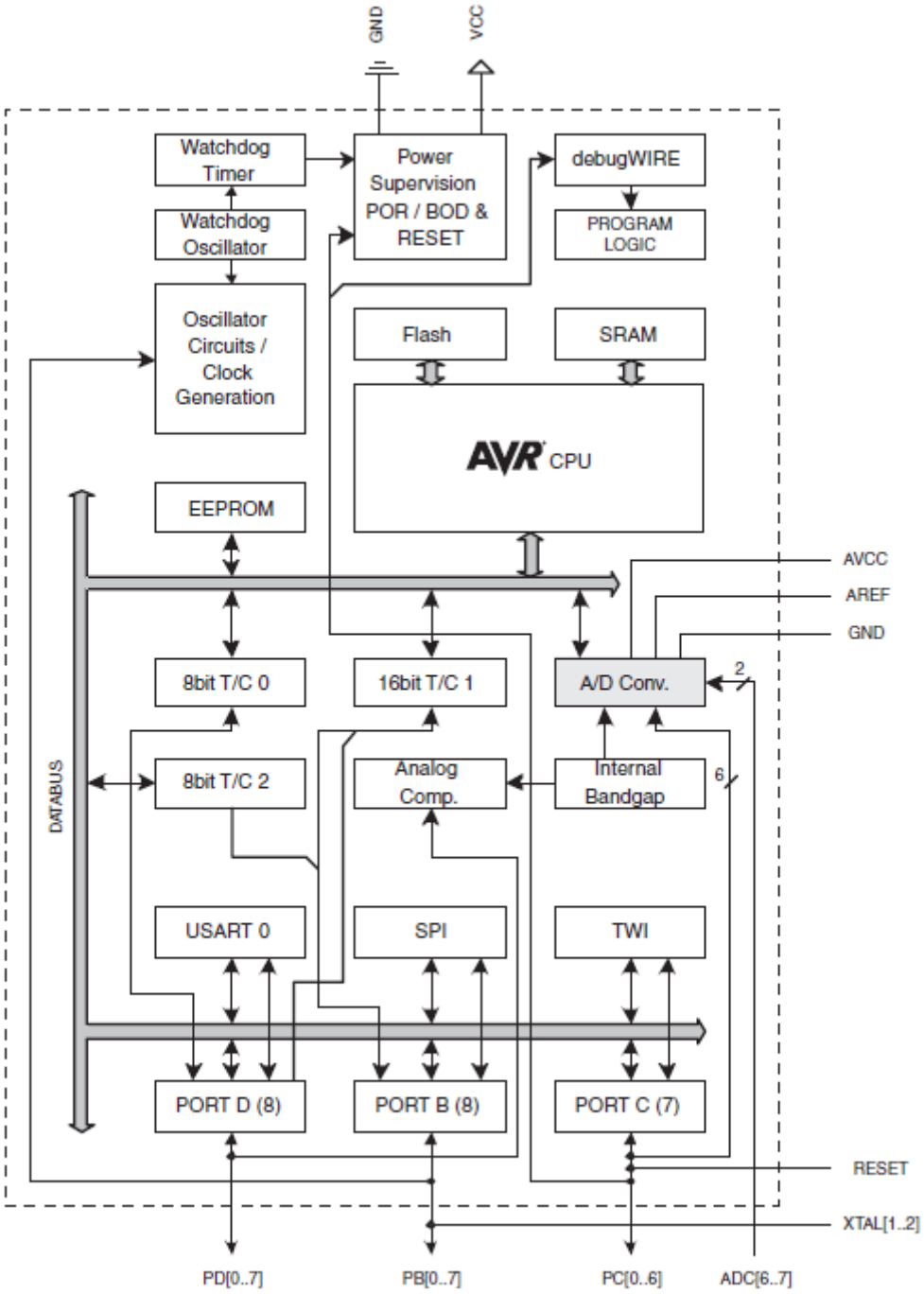


Рисунок 2.2 - Структурна схема ATmega328

Ядро AVR об'єднує різноманітний набір інструкцій з 32 робочими регістрами загального призначення. Кожен з цих 32 регістрів безпосередньо пов'язаний з

арифметико-логічним пристроєм (АЛУ), що дозволяє отримати доступ до двох незалежних регістрів в межах однієї інструкції, яка виконується за один такт.

Як наслідок, архітектура досягає вищої ефективності коду, забезпечуючи при цьому пропускну здатність до десяти разів більшу, ніж у традиційних мікроконтролерів CISC.

ATmega може похвалитися вражаючим набором функцій, включаючи вбудовану програмовану флеш-пам'ять об'ємом від 4 до 8 Кбайт з можливістю читання і запису, EEPROM від 256 до 1 Кбайт і SRAM від 512 до 2 Кбайт. Він має 23 лінії вводу/виводу загального призначення, а також 32 робочих регістри загального призначення. Крім того, він включає три універсальні таймери/лічильники з режимами порівняння, підтримку внутрішніх і зовнішніх переривань, а також такі функції, як послідовний програмований інтерфейс USART, байт-орієнтований 2-провідний послідовний інтерфейс і послідовний порт SPI. Крім того, він інтегрує 6-канальний 10-розрядний АЦП (з 8 каналами в корпусах TQFP і QFN/MLF), програмований сторожовий таймер з внутрішнім генератором і забезпечує п'ять режимів енергозбереження, що обираються програмно. У режимі простою процесор зупиняється, дозволяючи продовжувати роботу ОЗП, таймеру/лічильникам, USART, 2-провідному послідовному інтерфейсу, порту SPI та системі переривань. У режимі енергозбереження вміст регістрів зберігається, але генератор зупиняється, відключаючи всі інші функції мікросхеми до наступного переривання або апаратного скидання.

У режимі енергозбереження асинхронний таймер залишається активним, що дозволяє користувачеві підтримувати базу таймера, поки інші компоненти пристрою перебувають у сплячому режимі. Режим зменшення шуму АЦП зупиняє роботу ЦП і всіх модулів вводу/виводу, крім асинхронного таймера і АЦП, з метою зменшення шуму перемикання під час перетворень АЦП. У режимі очікування кварцовий/резонаторний генератор залишається робочим, в той час як решта пристрою перебуває в сплячому режимі, що сприяє швидкому запуску при мінімальному споживанні енергії.

Microchip надає бібліотеку QTouch, призначену для інтеграції ємнісних сенсорних кнопок, повзунків і коліщаток в мікроконтролери AVR®. Використовуючи запатентовану технологію збору сигналу перенесення заряду, вона забезпечує надійне зчитування і включає в себе повністю дешифровані звіти про сенсорні клавіші, а також технологію придушення сусідніх клавіш (Adjacent Key Suppression™ (AKS™)), що забезпечує чітке виявлення ключових подій. Зручний набір інструментів QTouch Suite дозволяє користувачам досліджувати, розробляти та усувати несправності власних сенсорних додатків.

Пристрій виготовлено з використанням передової технології енергонезалежної пам'яті високої щільності від Microchip. Вбудована флеш-пам'ять ISP Flash дозволяє перепрограмувати програмну пам'ять в системі через послідовний інтерфейс SPI, за допомогою звичайного програматора енергонезалежної пам'яті або за допомогою вбудованої програми завантаження, що виконується на ядрі AVR. Ця завантажувальна програма може використовувати будь-який інтерфейс для завантаження прикладної програми в прикладну флеш-пам'ять. Тим часом, програмне забезпечення в секції Boot Flash продовжує виконуватися, поки оновлення вносяться в секцію Application Flash, забезпечуючи справжню функціональність Read-While-Write (читання і запис). Завдяки інтеграції 8-розрядного RISC-процесора з вбудованою самопрограмованою флеш-пам'яттю на одному кристалі, ATmega328 є надійним мікроконтролером, що пропонує високоадаптивне і економічно ефективне рішення для різних вбудованих систем керування.

ATmega AVR супроводжується широким спектром засобів розробки програм і систем, включаючи компілятори C, макроасемблери, відладчики/симулятори програм, внутрішньосхемні емулятори та оціночні набори.

Результати кваліфікації надійності вказують на те, що очікувана частота збоїв у збереженні даних значно нижча за 1 PPM протягом 20 років при 85°C або 100 років при 25°C.

Для реалізації сенсорних інтерфейсів на більшості мікроконтролерів AVR бібліотека QTouch пропонує просте рішення. Вона включає в себе підтримку методів збору даних QTouch та QMatrix™.

Впровадження сенсорного вводу в будь-яку програму передбачає інтеграцію відповідної бібліотеки QTouch, розробленої для мікроконтролерів AVR. Цей процес передбачає використання простого набору API для позначення сенсорних каналів і датчиків, з подальшим викликом API сенсорного зондування для отримання даних про канал і визначення стану сенсорного датчика.

Для оптимізації продуктивності та паралелізму в AVR використовується гарвардська архітектура, що передбачає окрему пам'ять і шини для програм і даних. Інструкції, що зберігаються в пам'яті програм, виконуються за допомогою однорівневого конвеєрного механізму. Коли виконується одна інструкція, наступна інструкція попередньо витягується з пам'яті програм.

Цей принцип полегшує виконання інструкцій у кожному такті. Пам'ять програм складається з внутрішньосистемної перепрограмованої флеш-пам'яті.

Регістровий файл, який забезпечує швидкий доступ, складається з 32 x 8-бітних робочих регістрів загального призначення, доступних протягом одного такту. Ця особливість полегшує однотактні арифметико-логічні операції (ALU). Під час типової операції АЛП два операнди завантажуються з регістрового файлу, операція виконується, а результат зберігається назад у регістровому файлі - і все це протягом одного такту.

З 32 регістрів шість можуть функціонувати як три 16-розрядні непрямі адресні регістри-вказівники для адресації простору даних, що дозволяє ефективно обчислювати адреси. Крім того, один з цих покажчиків адреси може служити покажчиком адреси для таблиць пошуку в програмній пам'яті Flash. Цими додатковими функціональними регістрами є 16-розрядні X-, Y- та Z-регістри, які будуть детально розглянуті пізніше в цьому розділі.

АЛП здатен виконувати арифметичні та логічні операції між регістрами або між константою та регістром. Крім того, в АЛУ можна виконувати операції з одним

регістром. Після виконання арифметичної операції регістр стану оновлюється, щоб відобразити інформацію, яка стосується результату операції.

Потік програми полегшується за рахунок умовних і безумовних інструкцій переходу і виклику, що дозволяє пряму адресацію всього адресного простору. Більшість інструкцій AVR відформатовані в одне 16-бітне слово. Кожна адреса пам'яті програм містить 16- або 32-бітну інструкцію.

Простір програмної флеш-пам'яті розділений на дві секції: секцію завантажувальної програми і секцію прикладної програми. Обидві секції мають спеціальні біти блокування для захисту від запису та читання/запису. Зокрема, інструкція SPM, що відповідає за запис в секцію прикладної флеш-пам'яті, повинна знаходитися в секції завантажувальної програми.

Під час переривань і викликів підпрограм, адреса повернення лічильника програм (PC) зберігається в стеку. Стек ефективно розподілений у загальній пам'яті даних SRAM, отже, розмір стеку обмежується виключно загальним розміром SRAM та його використанням. Усі користувацькі програми повинні ініціалізувати вказівник стеку (SP) у процедурі Reset перед виконанням підпрограм або переривань. Покажчик стеку (SP) доступний для операцій читання та запису в межах простору вводу/виводу.

Доступ до ОЗП даних здійснюється через п'ять різних режимів адресації, що підтримуються в архітектурі AVR.

Всі комірки пам'яті в архітектурі AVR мають лінійні та уніфіковані карти пам'яті.

Адаптивний модуль переривань має свої регістри керування, розташовані в просторі вводу/виводу, доповнені додатковим бітом дозволу глобального переривання, розміщеним в регістрі стану. Кожне переривання має індивідуальний вектор переривання в таблиці векторів переривань, пріоритети яких визначаються відповідними позиціями вектора переривань. Менші адреси векторів переривань відповідають вищим рівням пріоритету.

Простір пам'яті вводу/виводу охоплює 64 адреси, призначені для периферійних функцій процесора, регістрів керування, SPI та інших функцій вводу/виводу. Можливий прямий доступ до пам'яті вводу/виводу, або ж доступ до неї можна отримати через комірки простору даних, які слідують за комірками регістрового файлу (0x20 - 0x5F). Крім того, ATmega має розширений простір вводу/виводу від 0x60 до 0xFF в межах SRAM, де застосовуються тільки інструкції ST/STS/STD і LD/LDS/LDD.

ALU, скорочення від Arithmetic Logic Unit, працює з високою ефективністю в архітектурі AVR, безпосередньо взаємодіючи з усіма 32 робочими регістрами загального призначення.

За один такт виконуються арифметичні операції між регістрами загального призначення або між регістром і безпосереднім значенням. Операції АЛУ поділяються на арифметичні, логічні та бітові функції. Певні реалізації архітектури можуть також мати надійний множник, здатний обробляти множення зі знаком/без знаку та дробовий формат. Для отримання детальної інформації зверніться до розділу «Набір інструкцій».

Регістр стану зберігає інформацію про результат останньої виконаної арифметичної команди. Ці дані можуть впливати на потік програми для виконання умовних операцій. Слід зазначити, що регістр стану оновлюється після виконання всіх операцій АЛУ, як зазначено в довіднику набору інструкцій. Це часто усуває необхідність використання спеціальних інструкцій порівняння, в результаті чого код стає швидшим і компактнішим.

Під час ініціювання підпрограми переривання та її подальшого повернення, регістр стану не зберігається та не відновлюється автоматично. Ця відповідальність покладається на програмне забезпечення. Регістр стану стабілізатора, позначений як SREG, показаний на рисунку 2.3.

Біт 7 - I: Глобальний дозвіл переривання. Для активації переривань необхідно встановити біт Global Interrupt Enable. Активація окремих переривань керується через окремі регістри керування.

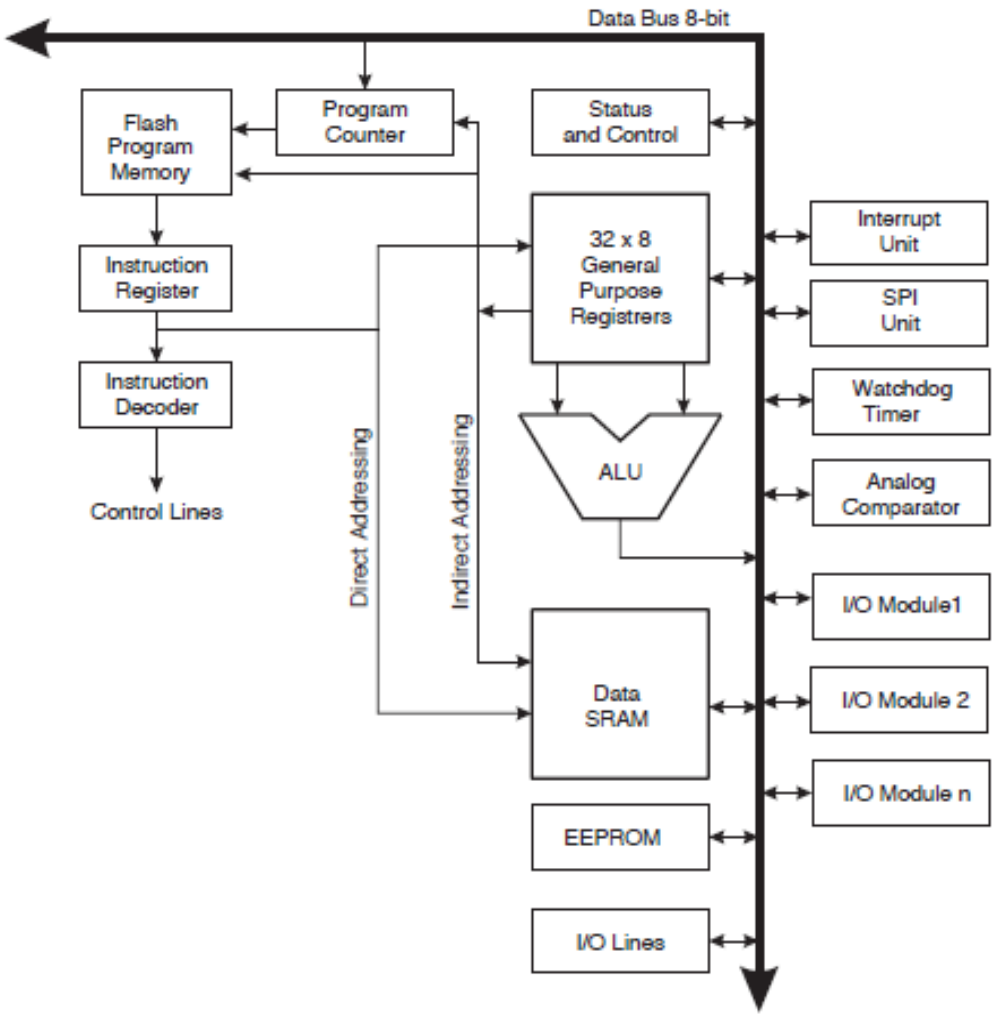


Рисунок 2.3 - Структурна схема архітектури AVR

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 2.4 - Регістр стану стабілізатора

Якщо регістр глобального дозволу переривань очищено, жодне з переривань не буде активоване, незалежно від налаштувань дозволу окремих переривань.

Після виникнення переривання I-біт автоматично очищується апаратно, а інструкція RETI встановлює його для дозволу наступних переривань. Програмний

контроль над I-біт також можливий за допомогою інструкцій SEI і CLI, як описано в довіднику по набору інструкцій.

Біт 6 - T: зберігання бітової копії. Інструкції копіювання бітів BLD (Bit LoaD) і BST (Bit STore) використовують T-біт як джерело або місце призначення для біта, що оперується. Інструкція BST переносить біт з регістра в регістровому файлі в T, а інструкція BLD копіює біт з T в регістр в регістровому файлі.

Біт 5 - H: прапорець напівперенесення. Прапорець половинного переносу H сигналізує про половинний перенос під час певних арифметичних операцій, що особливо корисно в двійково-десятковій арифметиці (BCD).

Біт 4 - S: Знаковий біт, S = N V. S-біт представляє виключне АБО між прапором заперечення N і прапором переповнення доповнення двох V.

Біт 3 - V: прапорець переповнення двійкового доповнення. Прапорець переповнення доповнення двох V полегшує арифметику доповнення двох.

Біт 2 - N: заперечний прапорець. Негативний прапорець N вказує на негативний результат арифметичної або логічної операції.

Біт 1 - Z: прапорець нуля. Нульовий прапорець Z вказує на нульовий результат арифметичної або логічної операції.

Біт 0 - C: прапорець перенесення. Прапорець переносу C сигналізує про перенос в арифметичній або логічній операції.

2.3 Регістровий файл загального призначення

Регістровий файл створено з урахуванням набору інструкцій розширеної RISC-програми AVR, що забезпечує необхідну продуктивність і універсальність завдяки наступним конфігураціям вводу/виводу:

- Один 8-бітовий вихідний операнд і один 8-бітовий вхід результату.
- Два 8-бітних вихідних операнда і один 8-бітовий вхід результату.
- Два 8-бітних вихідних операнди і один 16-бітовий вхід результату.
- Один 16-бітовий вихідний операнд і один 16-бітовий вхід результату.

Розташування 32 робочих регістрів загального призначення в ЦП зображено на рисунку 2.4.

Багато інструкцій, які взаємодіють з регістровим файлом, пропонують прямий доступ до всіх регістрів і, як правило, виконуються за один цикл.

Як показано на рисунку 2.4, кожен регістр асоціюється з адресою пам'яті даних, ефективно відображаючи їх у початкові 32 комірки простору даних користувача. Хоча фізично це не реалізовано у вигляді комірок SRAM, така організація пам'яті забезпечує значну гнучкість у доступі до регістрів.

Зокрема, регістри-покажчики X, Y і Z можуть бути налаштовані для індексації будь-якого регістра у файлі.

2.4 X-реєстр, Y-реєстр та Z-реєстр

Регістри R26 - R31 слугують для додаткових цілей, окрім їх загальної корисності. Зокрема, ці регістри функціонують як 16-розрядні покажчики адреси для непрямої адресації у просторі даних.

Ці регістри називаються регістрами непрямої адреси X, Y та Z, їх визначення детально описано на рисунку 2.5.

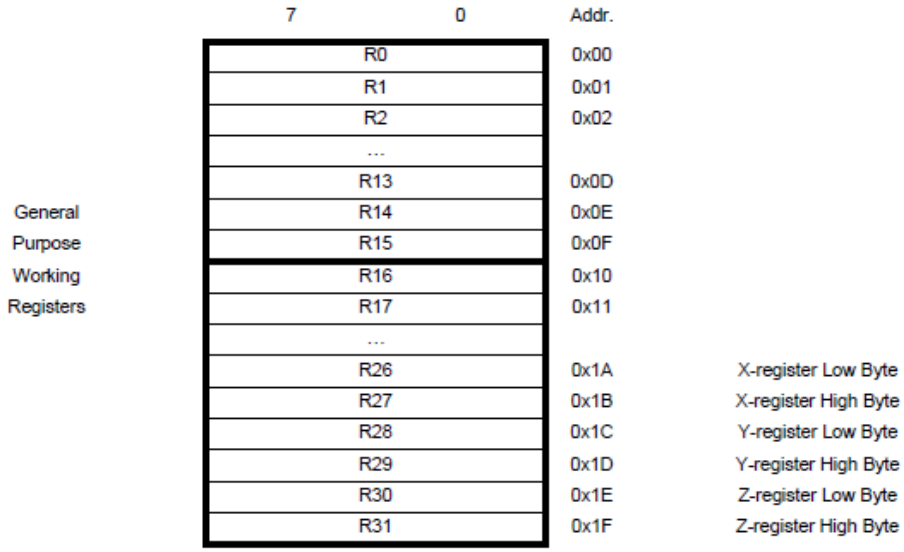


Рисунок 2.4 - Конфігурація 32 робочих регістрів загального призначення

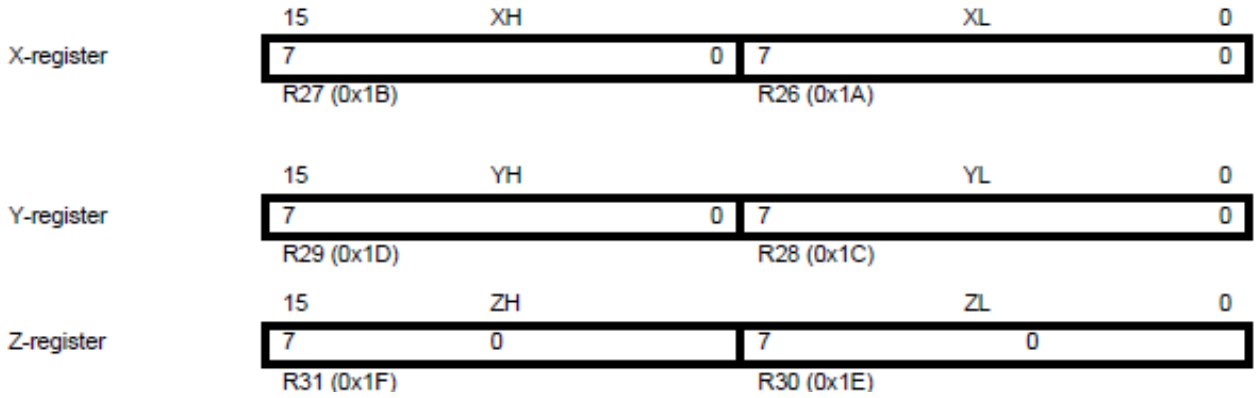


Рисунок 2.5 - X-, Y- та Z-регістри

У різних режимах адресації ці адресні регістри виконують різні функції, включаючи фіксоване зміщення, автоматичний інкремент та автоматичний декремент.

2.5 Покажчик стеку

Стек в основному слугує для зберігання тимчасових даних, локальних змінних та адрес повернення після переривань та викликів підпрограм. Важливо зазначити, що стек зростає від вищих до нижчих ділянок пам'яті. Регістр вказівника стека постійно вказує на верхню позицію стека, вказуючи на область стека в оперативній пам'яті даних, де знаходяться стек підпрограм і стек переривань. Виконання команди Stack PUSH призводить до зменшення вказівника стека.

Перед виконанням будь-яких викликів підпрограм або активацією переривань програма повинна визначити стек в оперативній пам'яті даних (SRAM). Початкове значення покажчика стеку відповідає останній адресі внутрішньої пам'яті SRAM, і воно повинно бути розташоване над початком SRAM.

Покажчик стеку AVR реалізовано у вигляді двох 8-бітних регістрів у просторі вводу/виводу. Кількість бітів, що фактично використовується, залежить від реалізації. Зауважте, що простір даних у деяких реалізаціях архітектури AVR

настільки малий, що потрібен лише SPL. У цьому випадку регістр SPH буде відсутній.

2.6 Час виконання інструкцій

Процесор AVR працює під керуванням тактового генератора clkCPU, який генерується безпосередньо з обраного джерела тактової частоти для мікросхеми. Внутрішній поділ тактової частоти не використовується. Як показано на рисунку 2.6, Гарвардська архітектура та ефективна концепція регістрового файлу дозволяють паралельно отримувати та виконувати інструкції. Це формує фундаментальну концепцію конвеєризації, спрямовану на досягнення до 1 мільйона інструкцій на секунду (MIPS) на мегагерц (МГц). Це дає унікальні результати щодо кількості функцій на вартість, кількості функцій на такт і кількості функцій на одиницю потужності.

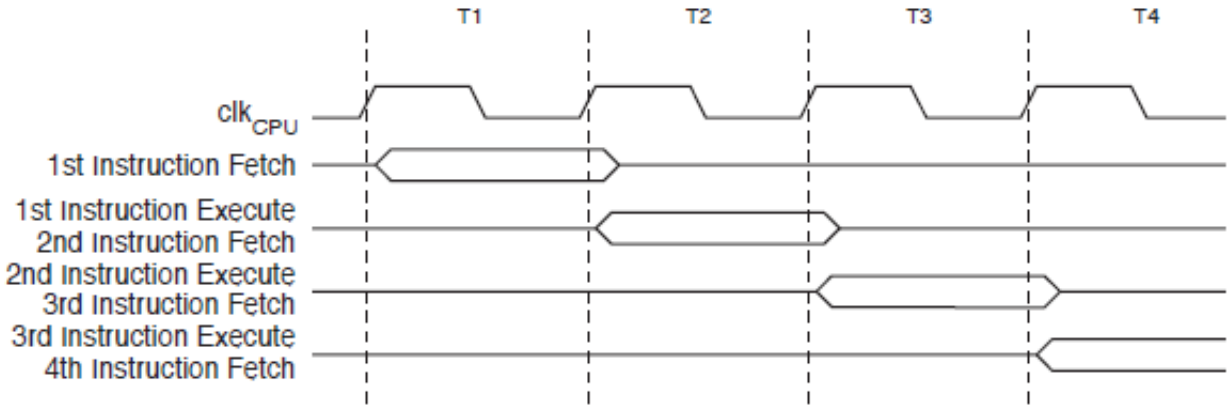


Рисунок 2.6 - Паралельна інструкція дибірки та виконання інструкцій

На рисунку 2.7 показано внутрішню концепцію синхронізації для файлу регістрів. Протягом одного такту арифметико-логічний пристрій (ALU) виконується з двома регістровими операндами, а результуюче значення зберігається назад у регістр призначення.

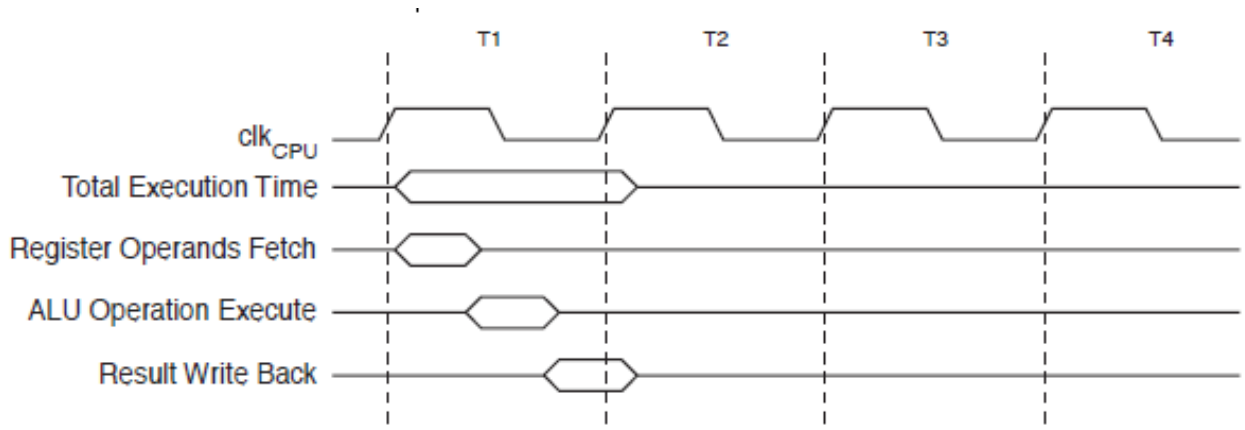


Рисунок 2.7 - Одноциклова робота ALU

2.7 Скидання та обробка переривань

Архітектура AVR пропонує різні джерела переривань, кожне з яких має власний програмний вектор у пам'яті програм. Цим перериванням, разом з окремим вектором скидання, присвоюються окремі біти дозволу. Щоб активувати переривання, ці біти дозволу повинні бути встановлені в логічну одиницю разом з бітом дозволу глобального переривання в регістрі стану.

Залежно від значення лічильника програм, переривання можуть автоматично відключатися, коли запрограмовані біти блокування завантаження BLB02 або BLB12.

За замовчуванням, найнижчі адреси в комірці пам'яті програм призначаються як вектори скидання та переривання. Рівні пріоритету різних переривань визначаються їхніми відповідними адресами, причому нижчі адреси означають вищий пріоритет. Найвищий пріоритет має RESET, за ним слідує INT0 - запит на зовнішнє переривання 0.

При виникненні переривання, глобальний біт дозволу переривання I очищується, вимикаючи всі переривання. Однак, користувацьке програмне забезпечення може увімкнути вкладені переривання, встановивши I-біт на логічну одиницю. У цьому випадку всі увімкнені переривання можуть переривати поточну

процедуру обробки переривань. Біт I встановлюється автоматично при виконанні інструкції повернення з переривання - RETI.

В AVR існує два типи переривань. Перший тип викликається подією, яка встановлює прапорець переривання. Коли виникає таке переривання, програмний лічильник спрямовується на відповідний вектор переривання для виконання процедури обробки переривання. Потім апаратне забезпечення скидає відповідний прапорець переривання. Крім того, прапори переривань можна очистити, записавши логічну одиницю у бітові позиції прапорів. Якщо умова переривання виникає, коли відповідний біт дозволу переривання очищено, прапорець переривання встановлюється і утримується доти, доки переривання не буде дозволено або прапорець не буде скинуто програмно. Аналогічно, якщо одна або більше умов переривання виникають при скинутому біті глобального дозволу переривання, відповідні прапори переривання будуть встановлені і запам'ятовуватимуться, доки не буде встановлено біт глобального дозволу переривання, після чого вони будуть виконані згідно з пріоритетом.

Другий тип переривань буде активовано доти, доки зберігається умова переривання. Ці переривання можуть не мати прапорців переривання. Якщо умова переривання припиняється до того, як переривання буде увімкнено, воно не спрацює.

Після виходу з переривання, AVR завжди повертається до основної програми і виконує ще одну інструкцію перед тим, як обслужити будь-яке переривання, що очікує на обробку. Важливо відзначити, що регістр стану не зберігається автоматично при вході в процедуру переривання і не відновлюється при виході з неї. Цим має керувати програма. Коли для вимкнення переривань використовується команда CLI, їх буде негайно вимкнено. Жодне переривання не буде виконано після команди CLI, навіть якщо воно виникає одночасно з командою CLI. Наступний приклад демонструє, як це можна використати для запобігання перериванням під час послідовності запису до EEPROM за таймером.

2.8 Час реакції на переривання

Час реакції на виконання переривання для всіх дозволених переривань AVR становить мінімум чотири такти. Протягом цих чотирьох тактів програмний лічильник виштовхується в стек. Після цього виконується програмний вектор адреси для фактичної процедури обробки переривання. Як правило, цей вектор передбачає перехід до підпрограми обробки переривання, що займає три такти.

У випадку, якщо переривання виникає під час виконання багатотактової інструкції, інструкція завершується до того, як переривання буде обслужено.

Коли MCU знаходиться в сплячому режимі і виникає переривання, час реакції на виконання переривання збільшується на чотири такту. Цей додатковий час є додатковим до часу запуску з обраного сплячого режиму. Повернення з процедури обробки переривання також займає чотири такту. Протягом цих чотирьох тактів програмний лічильник (що складається з двох байт) витягується зі стека, вказівник стека збільшується на два і встановлюється I-біт в SREG.

2.9 Пам'ять AVR

Архітектура AVR охоплює два основних простори пам'яті: пам'ять даних і пам'ять програм. Крім того, ATmega містить пам'ять EEPROM, призначену для зберігання даних. Всі три області пам'яті характеризуються лінійною та регулярною структурою.

ATmega оснащено вбудованою системною перепрограмованою флеш-пам'яттю об'ємом від 4 до 32 кілобайт для зберігання програм. Враховуючи, що всі інструкції AVR мають ширину 16 або 32 біт, флеш-пам'ять структурована у вигляді 2, 4, 8 або 16 кілобайтів по 16 біт. Для підвищення безпеки програмного забезпечення простір флеш-пам'яті розділений на дві секції: завантажувальну і прикладну. Таблиці констант можуть бути розподілені в межах усього адресного простору пам'яті програм, як описано в документації до інструкції LPM (Load Program Memory).

Рисунок 2.10 ілюструє організацію пам'яті SRAM у мікроконтролері ATmega.

Через свою складність і велику кількість периферійних пристроїв, ATmega перевищує 64 комірки, відведені в коді операцій для інструкцій IN і OUT. Отже, тільки інструкції ST/STS/STD і LD/LDS/LDD можуть бути використані для доступу до розширеного простору вводу/виводу в діапазоні від 0x60 до 0xFF в SRAM.

Нижня частина пам'яті даних містить 768/1280/1280/2303 комірки, що охоплюють регістровий файл, пам'ять вводу/виводу, розширену пам'ять вводу/виводу та внутрішню пам'ять даних SRAM. Зокрема, перші 32 комірки відповідають файлу регістрів, за ними йдуть 64 комірки для стандартної пам'яті вводу/виводу, 160 комірок для розширеної пам'яті вводу/виводу, а потім 512/1024/1024/2048 комірок для внутрішньої пам'яті даних SRAM.

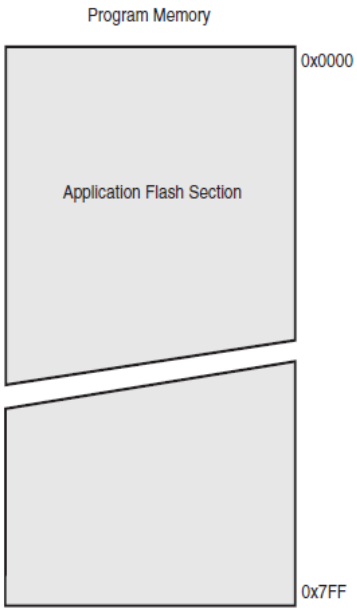


Рисунок 2.8 – Карта пам’яті ATmega 428

Розглянуто п'ять різних режимів адресації для пам'яті даних: Пряма, непряма зі зміщенням, непряма, непряма з попереднім декрементом та непряма з наступним інкрементом. У регістровому файлі регістри R26 - R31 слугують регістрами-покажчиками непрямої адресації.

Пряма адресація охоплює весь простір даних, тоді як режим непрямої адресації зі зміщенням поширюється на 63 адресні комірки від базової адреси, наданої Y- або Z- регістром.

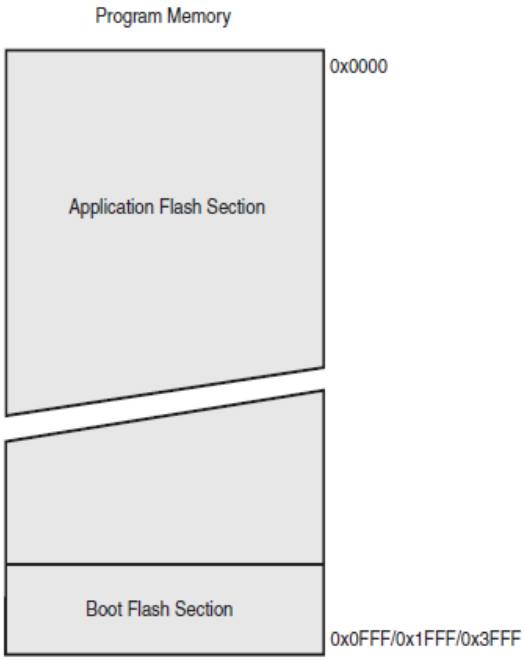


Рисунок 2.9 – Карта пам’яті ATmega328

При використанні режимів регістрової непрямої адресації з автоматичним попереднім декрементом і пост-інкрементом, адресні регістри X, Y і Z декрементуються або інкрементуються відповідно.

Всі 32 робочих регістри загального призначення, 64 регістри вводу/виводу, 160 розширених регістрів вводу/виводу і 512/1024/1024/2048 байт внутрішнього ОЗП даних в ATmega328 доступні за допомогою цих режимів адресації.

Доступ до внутрішньої пам'яті даних SRAM займає два цикли clkCPU, як показано на рисунку 2.11.

ATmega оснащено пам'яттю даних EEPROM ємністю 256/512/512/1 Кбайт, організованою у вигляді окремого простору даних, де окремі байти можуть бути

прочитані і записані. EEPROM може похвалитися витривалістю щонайменше 100 000 циклів запису/стирання.

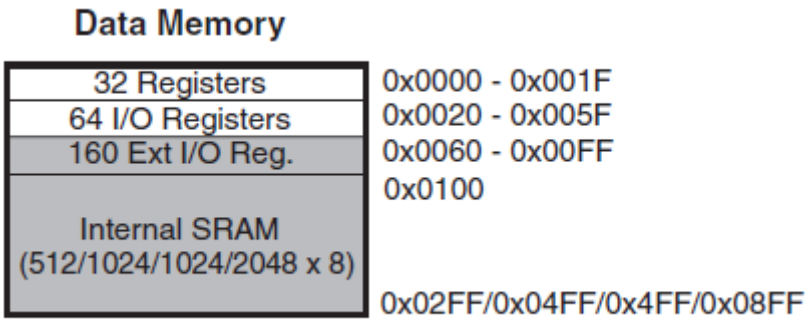


Рисунок 2.10 – Карта пам’яті даних

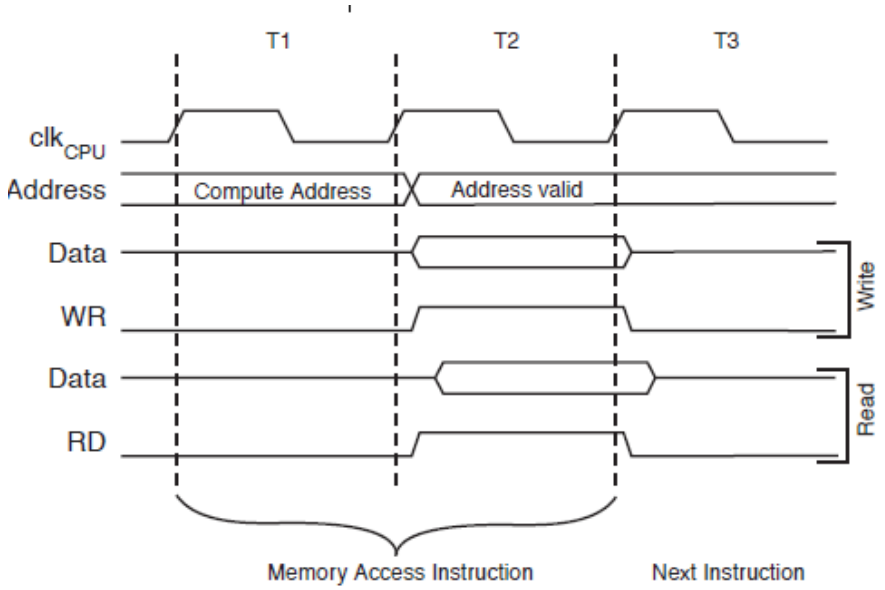


Рисунок 2.11 - Цикли доступу до внутрішньої пам'яті даних SRAM

Доступ між EEPROM і ЦП забезпечується за допомогою регістрів адреси EEPROM, регістра даних EEPROM і регістра керування EEPROM.

Доступ до регістрів доступу до EEPROM здійснюється з простору вводу/виводу.

Час доступу до EEPROM для запису наведено у Таблиці 8-2. Хоча функція автотаймінгу дозволяє користувацькому програмному забезпеченню визначати,

коли можна записувати наступний байт. Коли користувацький код містить інструкції для запису до EEPROM, слід дотримуватися певних запобіжних заходів. Зокрема, у блоках живлення з інтенсивною фільтрацією напруга VCC може повільно зростати або спадати під час увімкнення або вимкнення живлення, що призводить до роботи пристрою при напрузі, нижчій за вказаний мінімум для використовуваної тактової частоти.

Щоб запобігти ненавмисному запису EEPROM, необхідно дотримуватися спеціальної процедури запису. Зверніться до опису регістра керування EEPROM для отримання детальної інформації про цю процедуру.

Під час читання EEPROM процесор призупиняється на чотири такти перед виконанням наступної інструкції. І навпаки, під час запису до EEPROM процесор призупиняється на два такти перед виконанням наступної команди.

2.10 Запобігання пошкодженню EEPROM

При низьких значеннях напруги живлення дані EEPROM можуть бути пошкоджені через недостатню напругу живлення, необхідну для нормальної роботи процесора і EEPROM. Ці проблеми паралельні тим, що виникають у системах на рівні плати, які використовують EEPROM, і для їх вирішення слід застосовувати ті ж самі стратегії проектування. Пошкодження даних EEPROM може статися за двох умов, коли напруга недостатньо низька. По-перше, стандартна послідовність запису в EEPROM вимагає мінімальної напруги для точного функціонування. По-друге, сам процесор може помилково виконувати інструкції, якщо напруга живлення нижче необхідного порогу. Щоб запобігти пошкодженню даних EEPROM, рекомендується підтримувати активний (низький) рівень напруги AVR RESET під час періодів недостатньої напруги живлення. Цього можна досягти, активувавши внутрішній детектор вимкнення живлення (BOD). Якщо рівень виявлення внутрішнього BOD не збігається з необхідним рівнем виявлення, можна використати зовнішню схему захисту від скидання низької

напруги VCC. Якщо скидання відбувається під час операції запису, операція запису буде завершена доти, доки напруга живлення відповідає необхідним вимогам.

2.11 Пам'ять

Всі входи/виходи та периферійні пристрої в ATmega розміщені в межах простору вводу/виводу. Доступ до всіх місць вводу/виводу полегшується за допомогою інструкцій LD/LDS/LDD і ST/STS/STD, які забезпечують передачу даних між 32 робочими регістрами загального призначення і простором вводу/виводу.

Регістрами вводу/виводу в діапазоні адрес 0x00 - 0x1F можна безпосередньо маніпулювати на бітовому рівні за допомогою інструкцій SBI і CBI. Ці регістри дозволяють перевіряти значення окремих бітів за допомогою інструкцій SBIS і SBIC. Більш детальну інформацію можна знайти в розділі «Набір інструкцій».

При використанні специфічних для вводу/виводу команд IN та OUT слід використовувати адреси у діапазоні 0x00 - 0x3F. Однак, при зверненні до регістрів вводу/виводу як до простору даних за допомогою команд LD і ST, до цих адрес необхідно додати зміщення 0x20.

Через складність ATmega та велику кількість периферійних пристроїв, периферійних пристроїв може бути більше, ніж можна розмістити в 64 комірках, зарезервованих в операційному коді для інструкцій IN та OUT. Для розширеного простору вводу/виводу від 0x60 до 0xFF в SRAM можна використовувати лише інструкції ST/STS/STD та LD/LDS/LDD.

Для забезпечення сумісності з майбутніми пристроями зарезервовані біти повинні бути обнулені при зверненні. Зарезервовані адреси пам'яті вводу/виводу ніколи не повинні бути записані.

Деякі прапори стану скидаються шляхом запису в них логічної одиниці. На відміну від більшості інших AVR, команди CBI та SBI впливають лише на вказаний

біт, і тому їх можна використовувати для регістрів, що містять такі прапорці стану. Ці команди працюють виключно з регістрами в діапазоні від 0x00 до 0x1F.

Регістри керування вводом/виводом та периферійними пристроями детально описано у наступних розділах.

2.12 Регістри вводу/виводу загального призначення, системи синхронізації та їх розподіл

ATmega оснащено трьома регістрами вводу/виводу загального призначення, які слугують універсальними місцями для зберігання різних типів даних, включаючи глобальні змінні та прапорці стану. Ці регістри, розташовані в діапазоні адрес 0x00 - 0x1F, підтримують пряму маніпуляцію на бітовому рівні за допомогою таких інструкцій, як SBI, CBI, SBIS і SBIC.

Рисунок 2.12 ілюструє основні системи тактової частоти в архітектурі AVR та їх розподіл. Слід зазначити, що не всі тактові генератори повинні бути активними одночасно. Для мінімізації енергоспоживання, тактові генератори модулів, які не використовуються, можуть бути зупинені за допомогою різних режимів сну. Системи тактових частот детально описано нижче.

2.13 Тактовий генератор процесора - clkCPU

Тактовий сигнал процесора передається на компоненти системи, які необхідні для роботи ядра стабілізатора. До них належать файл регістрів загального призначення, регістр стану та пам'ять даних, що містить вказівник стека. Зупинка тактового генератора процесора перешкоджає виконанню ядром загальних операцій і обчислень. Тактовий генератор вводу/виводу, з іншого боку, використовується більшістю модулів вводу/виводу, таких як таймер/лічильники, SPI та USART. Крім того, тактовий генератор вводу/виводу використовується модулем зовнішнього переривання. Однак, важливо зазначити, що визначення умов запуску у модулі USI виконується асинхронно, коли clkI/O зупинено, а розпізнавання адреси TWI відбувається у всіх сплячих режимах. Зверніть увагу, що

при використанні переривання за рівнем для пробудження з режиму енергозбереження заданий рівень повинен підтримуватися протягом часу, достатнього для завершення процесу пробудження і спрацьовування переривання за рівнем. Якщо рівень зникає до завершення часу запуску, MCU все одно прокинеться, але переривання не буде згенеровано. Час запуску визначається запобіжниками SUT і CKSEL.

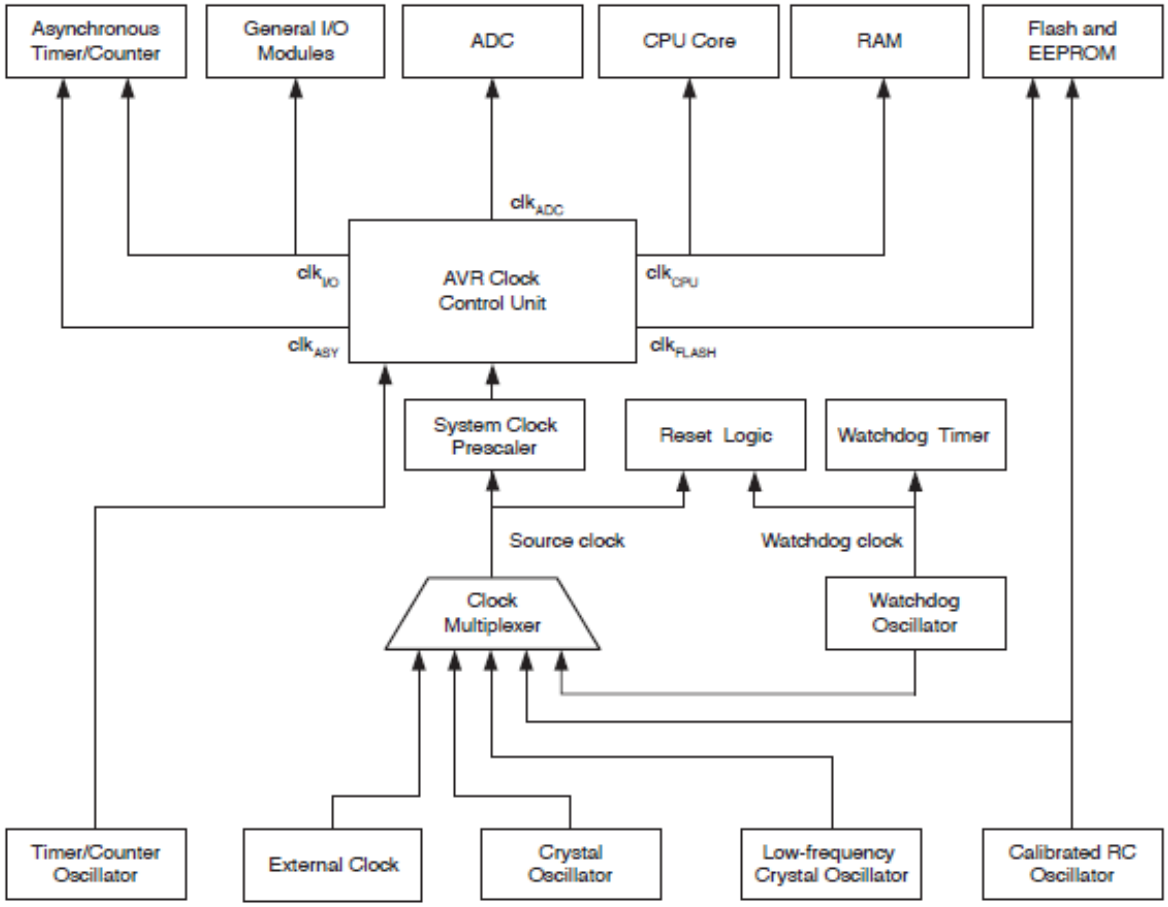


Рисунок 2.13 - Clock Distribution

2.14 Годинник флеш-пам'яті - clkFLASH

Тактовий генератор флеш-пам'яті керує роботою інтерфейсу флеш-пам'яті і зазвичай працює паралельно з тактовим генератором центрального процесора. Тактовий генератор асинхронного таймера полегшує пряму синхронізацію асинхронного таймера/лічильника від зовнішнього тактового генератора або

зовнішнього тактового кристала 32 кГц. Виділена область тактової частоти дозволяє використовувати цей таймер/лічильник як лічильник реального часу, навіть коли пристрій перебуває в сплячому режимі. АЦП оснащений виділеною тактовою областю, що дозволяє зупиняти тактові генератори процесора і вводу/виводу для зменшення шуму, що генерується цифровими схемами, тим самим підвищуючи точність результатів перетворення АЦП. Пристрій пропонує різні варіанти джерел тактової частоти, які можна вибрати за допомогою бітів Flash Fuse. Тактовий сигнал з обраного джерела надходить на вхід генератора тактових імпульсів AVR і направляється на відповідні модулі. З коробки пристрій конфігурується з внутрішнім RC-генератором на 8,0 МГц і запрограмованим запобіжником CKDIV8, в результаті чого системний тактовий генератор працює на частоті 1,0 МГц. Час запуску встановлюється на максимум з увімкненим періодом тайм-ауту (CKSEL = «0010», SUT = «10», CKDIV8 = «0»). Це налаштування за замовчуванням дозволяє користувачам змінювати налаштування джерела тактового генератора за допомогою будь-якого доступного інтерфейсу програмування. Кожне джерело тактового сигналу вимагає достатнього VCC для запуску коливань і мінімальної кількості циклів для стабілізації. Щоб забезпечити достатній рівень VCC, пристрій ініціює внутрішнє скидання з затримкою на тайм-аут (tTOUT) після того, як всі інші джерела скидання скинули пристрій. Затримка (tTOUT) вимірюється від сторожового генератора, а кількість його циклів визначається бітами запобіжників SUTx і CKSELx. Основна мета затримки - утримувати стабілізатор у стані скидання, поки він не отримає мінімально необхідну напругу VCC. Хоча затримка не контролює фактичну напругу, дуже важливо вибрати затримку довшу, ніж час наростання VCC. Якщо це неможливо, рекомендується використовувати внутрішню або зовнішню схему виявлення коричневого кольору. Схема BOD забезпечує достатній рівень VCC перед скиданням і дозволяє вимкнути затримку тайм-ауту. Не рекомендується вимикати затримку тайм-ауту без використання схеми виявлення коричневого вимкнення. Генератор повинен коливатися протягом мінімальної кількості циклів, перш ніж

годинник буде вважатися стабільним. Внутрішній лічильник пульсацій контролює вихідну частоту генератора, підтримуючи внутрішнє скидання активним протягом заданої кількості тактів. Після завершення скидання скидається, і пристрій починає виконання. Рекомендований час запуску генератора залежить від типу тактового генератора: від 6 циклів для зовнішніх тактових генераторів до 32К циклів для низькочастотного кристала. Послідовність запуску для годинника включає в себе як затримку тайм-ауту, так і час запуску під час ініціалізації пристрою зі скидання. Однак під час запуску з режиму енергозбереження або вимкнення живлення передбачається, що VCC знаходиться на достатньому рівні, і враховується лише час запуску.

2.16 Висновок

Розділ присвячено аналізу функціональних можливостей мікроконтролера ATmega, який вирізняється своїми надійними системами тактової частоти та широкими можливостями, призначеними для підвищення функціональності та продуктивності різноманітних застосувань. Кожен аспект архітектури годинника, від Flash-годинника, що керує Flash-інтерфейсом, до асинхронного таймера, що забезпечує відлік часу в реальному часі навіть у сплячому режимі, демонструє ретельну увагу до деталей та оптимізацію для різних сценаріїв роботи. Універсальність мікроконтролера поширюється і на варіанти джерел синхронізації, надаючи користувачам гнучкість у налаштуванні параметрів годинника відповідно до конкретних вимог додатків. Крім того, включення вбудованих засобів захисту, таких як затримка тайм-ауту і послідовність запуску, підкреслює його надійність, забезпечуючи стабільну роботу завдяки контролю рівнів напруги і стабільності генератора під час процесів ініціалізації. Мікроконтролери ATmega є переконливим вибором, пропонуючи не тільки високопродуктивні можливості, але й адаптивність до широкого спектру вбудованих додатків.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ ПІДТРИМКИ МУЛЬТИМЕДІА ДЛЯ МІКРОКОНТРОЛЕРНИХ СИСТЕМ НА БАЗІ ATMEGA328

3.1 Опис програмного забезпечення системи

Розглянемо схему з поясненнями до наведеного коду програмного забезпечення системи підтримки мультимедіа для мікроконтролерних систем на базі atmega328.

Функція VGAX::begin(bool enable Tone) встановлює необхідні таймери, виводи та переривання для генерації сигналу VGA. Вона вимикає переривання TIMER0, налаштовує звуковий вивід (якщо він увімкнений), налаштовує TIMER1 для імпульсів вертикальної синхронізації, налаштовує TIMER2 для імпульсів горизонтальної синхронізації та налаштовує виводи кольору. Нарешті, вона вмикає глобальні переривання.

Функція VGAX::end() вимикає TIMER0, TIMER1 і TIMER2, очищаючи їхні регістри керування. Функція VGAX::clear(byte color) упаковує 2-бітне значення кольору у байт, а потім заповнює весь фреймбуфер (vgaxfb) упакованим значенням кольору за допомогою memset. Функція VGAX::copy(byte src) копіює вміст вихідного байтового масиву у фреймбуфер vgaxfb за допомогою циклу while. Функція VGAX::fillrect(байт x, байт y, байт width, байт height, байт color) малює прямокутник на дисплеї шляхом перебору висоти та ширини прямокутника, перевірки, чи поточні координати пікселя не виходять за межі дисплея, а потім виклику функції putpixel () для встановлення пікселя з вказаним кольором. Функція VGAX::tone(unsigned int frequency) обчислює значення змінних afreq і afreq0 на основі заданої частоти для генерації звукової модуляції. Функція VGAX::no Tone() вимикає звукову модуляцію, встановлюючи змінну afreq0 у 0. Функція VGAX::delay(int msec) створює затримку на вказану кількість мілісекунд шляхом виконання циклу з серією інструкцій NOP. Переривання VSYNC (TIMER1_OVF_vect) відповідає за скидання змінної aline, встановлення змінної

vskip, інкремент змінної vtimer та скидання змінної rlinecnt для підготовки до наступного кадру.

Нижче наведено детальні пояснення до наданого коду бібліотеки VGAX:

1. Запуск бібліотеки VGAX є точкою входу для бібліотеки VGAX, з якої починаються процеси ініціалізації та налаштування.

2. Initialize VGAX представляє собою ініціалізацію бібліотеки VGAX. Вона встановлює необхідні конфігурації, зокрема конфігурацію TIMER0, TIMER1 і TIMER2, ініціалізацію масиву фреймбуферів vgaxfb та ініціалізацію вертикального таймера vtimer.

3. VGAX містить різноманітні функції, що надаються бібліотекою VGAX, зокрема:

- putpixel(x, y, color): встановлює колір пікселя за вказаними координатами у буфері кадрів.

- getpixel(x, y): отримує колір пікселя за вказаними координатами з фреймбуфера.

- putpixel4(bx, y, fourpixels): встановлює кольори чотирьох пікселів за вказаними координатами байта та рядка у фреймбуфері.

- getpixel4(bx, y): отримує значення кольорів чотирьох пікселів за вказаними байтовими та рядковими координатами з фреймбуфера.

- clear(color): зафарбовує весь фреймбуфер вказаним кольором.

- copy(src): копіює вміст масиву src до фреймбуфера.

- bitblit(src, swidth, height, dx, dy, color): викреслює (копіює) вихідне зображення у фреймбуфер з підтримкою прозорості.

- blit(src, swidth, height, dx, dy): викреслює (копіює) вихідне зображення у фреймбуфер без підтримки прозорості.

- blit[n]aligned(src, height, dbx, dy): викреслює (копіює) вихідне зображення до буфера кадру з вирівняними за байтами координатами призначення.

- fillrect(x, y, width, height, color): малює зафарбований прямокутник у фреймбуфері заданим кольором.

- print(fnt, glyphscount, fntheight, hspace, vspace, str, dx, dy, color): друкує рядок тексту у фреймбуфері з використанням наданого визначення шрифту.

4. Timing містить функції, пов'язані з синхронізацією, що надаються бібліотекою VGAX, зокрема:

- delay(msec): призупиняє виконання програми на вказану кількість мілісекунд.

- millis(): повертає кількість мілісекунд, що минули з моменту запуску бібліотеки vgax.

- micros(): повертає кількість мікросекунд, що минули з моменту запуску бібліотеки vgax.

- tone(frequency): генерує тон із заданою частотою.

- notone(): зупиняє генерацію тону.

Він ілюструє взаємозв'язки та взаємодію між різними компонентами бібліотеки VGAX, надаючи вичерпне уявлення про її функціональність та структуру.

Наведений код визначає утилітний клас VGAXUtils, який містить різні методи для малювання графіки на VGA-дисплеї з використанням бібліотеки VGAX. Клас включає наступні ключові компоненти:

1. метод draw_line реалізує лінійний алгоритм для малювання лінії між двома точками на екрані. Він обробляє як горизонтальні, так і вертикальні лінії, а також діагональні лінії. Безпечна версія методу draw_line_safe також виконує обрізання екрану, щоб гарантувати, що лінія буде намальована в межах екрану.

2. метод draw_circlen малює на екрані коло, з необов'язковим кольором заливки. Він використовує модифіковану версію алгоритму Midpoint circle для обчислення точок кола та їх малювання. Версія draw_circle_safe виконує обрізання екрану, щоб гарантувати, що коло буде намальоване в межах екрану.

3. метод draw_rect малює прямокутник на екрані, з необов'язковим кольором заливки. Він використовує метод draw_line для малювання чотирьох сторін

прямокутника. Версія draw_rect_safe виконує обрізання екрану, щоб гарантувати, що прямокутник буде намальовано у межах екрану.

4. Методи draw_row та draw_row_safe малюють на екрані горизонтальну лінію з пікселів. Версія draw_row_safe виконує відсікання екрану, щоб гарантувати, що лінія буде намальована в межах екрану.

5. методи draw_column та draw_column_safe малюють на екрані вертикальну лінію з пікселів. Версія draw_column_safe виконує обрізання екрану, щоб лінія не виходила за межі екрану.

6. putpixel_safe - допоміжний метод, який безпечно малює один піксель на екрані, виконуючи обрізання екрану, щоб переконатися, що піксель знаходиться в межах екрану.

Блок-схема надає детальне представлення логіки та функціональності кожного з цих методів, включаючи механізми обрізання екрану та обробки помилок. Підграфи для кожного методу чітко ілюструють покрокове виконання коду, що полегшує користувачеві розуміння основної логіки та реалізації.

Загалом, код надає набір утилітарних функцій, які можна використовувати для створення різноманітної графіки та фігур на VGA-дисплеї за допомогою бібліотеки VGAX. Безпечні версії методів гарантують, що операції малювання виконуються в межах екрану, запобігаючи будь-яким помилкам, що виходять за межі екрану, або неочікуваній поведінці.

Нижче наведено детальне пояснення наданого коду та блок-схему Mermaid, що відображає його логіку:Пояснення коду:

1. Функція bitblit відповідає за виведення растрового зображення на екран. Вона отримує наступні параметри:

- src: вказівник на вихідні растрові дані.
- swidth: ширина вихідного растрового зображення.
- height: висота вихідного растрового зображення.
- dx: x-координата місця призначення на екрані.
- dy: y-координата місця призначення на дисплеї.

- color: колір, який буде використано для відображення пікселів.

2. Функція ініціалізує вказівник на дані вихідного растрового зображення, а потім входить у цикл, який перебирає рядки вихідного растрового зображення.

3. Для кожного рядка функція зчитує байт з вихідного растрового зображення і обробляє його у дві частини:

- First Nibble (Higher 4 Bits): функція перевіряє, чи координата X поточного пікселя знаходиться в межах ширини дисплея, і чи встановлено відповідний біт у вихідному байті. Якщо обидві умови істинні, вона викликає функцію putpixel для відображення пікселя на екрані.

- Second Nibble (Lower 4 Bits): функція виконує аналогічний процес для наступних 4 пікселів у поточному рядку.

4. Після обробки поточного рядка функція переходить до наступного рядка, збільшуючи координату Y та вказівник вихідного растрового зображення.

5. Цикл продовжується до тих пір, поки не буде оброблено і виведено на екран все вихідне растрове зображення.

Блок-схема надає детальне уявлення про логіку та хід роботи коду. Вона включає наступні ключові компоненти:

1. Основна функція bitblit, яка є точкою входу в код.

2. Підграфи для обробки першого та другого відрізків вихідного байта, які відповідають за рендеринг окремих пікселів.

3. Детальні пояснення для кожного кроку в коді, включаючи мету, функціональність і будь-які відповідні умови або перевірки.

4. Потік керування між головною функцією та підграфами, а також потік всередині підграфів.

Ця комплексна блок-схема повинна допомогти користувачеві зрозуміти структуру коду, логіку та основну функціональність функції bitblit. Ось детальна блок-схема Mermaid з поясненнями для наданого коду на C++:Наданий код складається з двох функцій. Функція VGAX::blit(byte src, byte swidth, byte height, char dx, char dy) відповідає за розбиття (копіювання) вихідного зображення у

фреймбуфер VGAX. Спочатку вона ініціалізує необхідні змінні, такі як висота джерела, вказівник рядка джерела та розмір рядка джерела. Потім перевіряється, чи операція blit повністю вписується у межі екрана, чи частково обрізається. Якщо витиснення повністю в межах екрана, виконується операція витиснення без обрізання, яка повторюється для кожного рядка джерела і встановлює відповідні пікселі у фреймбуфері. Якщо блимання частково обрізано, виконується операція обрізаного блимання, перевіряючи координати пікселів призначення і встановлюючи відповідні пікселі у буфері кадру.

Функція VGAX::blitwmask(byte src, byte mask, byte swidth, byte height, char dx, char dy) відповідає за накладення вихідного зображення на фреймбуфер VGAX за допомогою маски. Спочатку вона ініціалізує необхідні змінні, такі як висота джерела, вказівник рядка джерела, вказівник рядка маски та розмір рядка. Потім виконується операція витирання за маскою, ітерація по кожному рядку джерела і використання маски для виконання операції витирання І+АБО для встановлення пікселів фреймбуфера.

Блок-схема детально відображає логіку та хід виконання цих двох функцій з поясненнями для кожного кроку, точки прийняття рішення та ключового компонента. Підграфи чітко розмежовують функціональність кожної функції, а анотації всередині підграфів пояснюють призначення та поведінку коду. Наданий код містить дві функції, VGAX::blit4aligned та VGAX::blit4, які відповідають за розбиття (копіювання) 4-бітових зображень з вихідного буфера до буфера кадру VGAX. VGAX::blit4aligned використовується для розмиття 4-бітового вирівняного зображення, тобто дані зображення вже вирівняно до меж байта. Вона ініціалізує вказівники джерела і призначення, а потім входить у цикл, який повторюється через рядки зображення. На кожній ітерації він зчитує байт з буфера джерела за допомогою функції pgm_read_byte() і записує його безпосередньо у відповідне місце у буфері кадру VGAX. Вказівник призначення збільшується на VGAX_BWIDTH для переходу до наступного рядка у буфері кадру. VGAX::blit4 використовується для вирівнювання 4-бітового зображення, обробляючи різні

сценарії вирівнювання та відсікання. Спочатку вона ініціалізує вказівники джерела і призначення на основі наданих dx (координата призначення x) і dy (координата призначення y). Потім перевіряється вертикальне обрізання, щоб переконатися, що зображення не виходить за межі буфера кадру VGAX. Якщо зображення частково або повністю виходить за вертикальні межі, функція відповідно коригує вказівник джерела і висоту. Далі перевіряється горизонтальне обрізання, щоб переконатися, що зображення не виходить за межі буфера кадру VGAX. Якщо зображення частково або повністю виходить за горизонтальні межі, функція відповідним чином коригує вказівник призначення. Залежно від ступеня горизонтального вирівнювання (0, 1, 2 або 3 пікселі) функція викликає різні процедури вирівнювання для обробки частково або повністю вирівняних пікселів. Процедури Blit UnalignedLeft і BlitUnalignedRight обробляють випадки, коли зображення частково виходить за межі лівого або правого боку буфера кадру VGAX відповідно. Вони виконують спеціальні операції з вирівнювання для обробки часткових пікселів. Процедура BlitAligned обробляє випадок, коли зображення повністю знаходиться у горизонтальних межах буфера кадру VGAX. Вона виконує відповідну операцію вирівнювання залежно від ступеня вирівнювання. Блок-схема надає детальне візуальне представлення логіки і потоку цих двох функцій з поясненнями для кожного ключового кроку і точки прийняття рішення. Підграфи для VGAX::blit4aligned і VGAX::blit4 чітко ілюструють функціональність кожної функції, включаючи обробку вирівнювання, відсікання і різні операції blit.

Наданий код містить три функції:

1. Функція VGAX::blit8aligned(байт src, байт height, байт dbx, байт dy) відповідає за вирівнювання (копіювання) 8-бітового вирівняного зображення у відеобуфер. Вона отримує вказівник на вихідне зображення, висоту вихідного зображення, x-координату місця призначення та y-координату місця призначення. Функція ітераційно перебирає висоту вихідного зображення і копіює дані пікселів у відповідне місце у фреймбуфері.

2. Функція VGAX::blit8(byte src, byte height, char dx, char dy) відповідає за блітування (копіювання) 8-бітового зображення у відеобуфер, обробляючи різні сценарії обрізання та вирівнювання. Вона отримує вказівник на вихідне зображення, висоту вихідного зображення, x-координату місця призначення та y-координату місця призначення. Спочатку функція перевіряє вертикальне обрізання, коригуючи вказівник джерела і висоту, якщо зображення частково або повністю виходить за межі фреймбуфера. Потім вона перевіряє горизонтальне обрізання, обробляючи випадки, коли зображення частково або повністю виходить за межі фреймбуфера. Нарешті, функція перебирає висоту вихідного зображення і копіює дані пікселів до фреймбуфера, обробляючи різні сценарії вирівнювання.

3. Функція VGAX::blit8wmask(byte src, byte mask, byte height, char dx, char dy) відповідає за блітування (копіювання) 8-бітового зображення у відеокадр, використовуючи маску для вибіркового оновлення пікселів. Вона отримує вказівник на вихідне зображення, вказівник на маску, висоту вихідного зображення, x-координату місця призначення та y-координату місця призначення. Спочатку функція перевіряє вертикальне обрізання, коригуючи джерело, маску і висоту, якщо зображення частково або повністю виходить за межі фреймбуфера. Потім вона перевіряє горизонтальне обрізання, обробляючи випадки, коли зображення частково або повністю виходить за межі фреймбуфера. Нарешті, функція ітераційно перебирає висоту вихідного зображення і копіює дані пікселів у фреймбуфер, використовуючи маску для вибіркового оновлення пікселів і обробляючи різні сценарії вирівнювання. Наведена вище блок-схема детально відображає логіку і структуру цих трьох функцій. Кожна функція інкапсульована в підграф, а потік і пояснення для ключових компонентів включені в діаграму. На діаграмі показано різні кроки, пов'язані з операціями blit, такі як ініціалізація вказівників, обробка вертикального та горизонтального обрізання, ітерація по вихідному зображенню, копіювання пікселів та оновлення вказівника призначення. Пояснення, надані для кожного кроку, допомагають зрозуміти призначення та функціональність коду. Загалом, блок-схема надає повне візуальне представлення

логіки та структури коду, що полегшує користувачеві розуміння основної функціональності бібліотеки VGAX. На блок-схемі представлено бібліотеку ExampleLibrary, яка надає набір класів та утиліт для роботи з VGA-дисплеями. Основними компонентами бібліотеки є

- Клас VGAX: надає методи для керування VGA-дисплеєм, такі як малювання пікселів, вимкнення зображення та очищення екрану.

- Клас vgaxutils: Надає утиліти, зокрема друк на VGA-дисплей, керування затримками та генерування тонів.

- Глобальні змінні: оголошує та ініціалізує глобальні змінні, що використовуються у бібліотеці, такі як екземпляри VGAX та vtimer, а також різні константи.

3. Клас VGAX містить методи для ініціалізації та вимкнення VGA-дисплея, а також широкий спектр функцій для маніпуляцій з дисплеєм, серед яких:

- putpixel() та getpixel(): встановлюють та отримують колір окремих пікселів.

- putpixel4() та getpixel4(): встановлюють та отримують колір 4 пікселів за раз.

- clear(): очищає увесь VGA-дисплей вказаним кольором фону.

- copy(): Копіює вміст одного VGA-буфера до іншого.

- bitblit(), blit(), blit4(), blit8(), blitwmask(), blit8wmask(), blit4aligned() і blit8aligned(): виконують різноманітні операції з бітовим копіюванням даних зображення з буфера-джерела до буфера-приймача.

- fillrect(): зафарбовує прямокутну область VGA-дисплея заданим кольором.

4. Клас VGAXUtils надає службові функції, зокрема:

- printPROGMEM() та printSRAM(): виводять вміст буферів PROGMEM та SRAM на VGA-дисплей.

- delay(), millis() та micros(): керують функціями, пов'язаними з таймінгом.

- tone() і noTone(): генерують і вимикають звукові сигнали на вказаному виводі.

5. Розділ Global Variables ініціалізує екземпляри VGAX та VTimer, а також різні константи, пов'язані з розмірами та розміром VGA-дисплея.

6. Загальний потік ExampleLibrary починається з ініціалізації бібліотеки, яка встановлює необхідні компоненти та глобальні змінні. Після цього бібліотека може бути використана для взаємодії з VGA-дисплеєм та використання наданих утиліт.

Зверніть увагу, що ця блок-схема не містить жодних циклів або самопосилань, відповідно до наданих вказівок.

Підграф VGAXLibrary:

- Початок: точка входу до бібліотеки VGAX.

- initialize: встановлення необхідних виводів та конфігурацій для виводу vga.

- setresolution: визначити відповідну роздільну здатність на основі моделі плати Arduino (UNO або MEGA).

- UNOResolution - це налаштування VGA виходу на 120x60px 2bpp на Arduino UNO.

- MEGAResolution - це налаштування VGA виходу на 120x240px 2bpp на Arduino MEGA.

- SetupVGASignal генерує необхідні сигнали синхронізації VGA (горизонтальна та вертикальна синхронізація) для обраної роздільної здатності.

- SetupColorBuffer виділяє та ініціалізує буфер пам'яті для зберігання даних про пікселі.

- End означає, що бібліотека VGAX успішно ініціалізована і готова до використання.

2. Підграф VGAXUsage:

- UseVGAX: взаємодіяти з бібліотекою VGAX для виведення вмісту на VGA-вихід.

- UpdateColorBuffer змінює дані пікселів у колірному буфері для зміни зображення, що виводиться на екран.

- RefreshVGAOutput запускає генерацію VGA-сигналу для оновлення зображення новими піксельними даними.

3. Підграф VGAXLibrary надає необхідну функціональність для взаємодії підграфа VGAXUsage з бібліотекою VGAX.

На блок-схемі показано ініціалізацію та використання бібліотеки VGAX. Бібліотека налаштовує генерацію VGA-сигналу та буфер кольорів на основі моделі плати Arduino (UNO або MEGA). Використання бібліотеки передбачає оновлення колірного буферу та оновлення VGA-виходу для відображення бажаного контенту.

Наданий код є частиною бібліотеки VGAX, яка відповідає за відображення тексту на дисплеї. Код включає дві основні функції: printPROGMEM і printSRAM, які обробляють друк рядків, що зберігаються у PROGMEM (пам'ять програм) і SRAM (статична пам'ять з довільним доступом), відповідно.

Функція printPROGMEM приймає наступні параметри:

- fnt: вказівник на дані шрифту, що зберігаються у PROGMEM.
- glyphscount: кількість гліфів (символів) у шрифті.
- fntheight: висота кожного гліфа.
- hspace: горизонтальний інтервал між гліфами.
- vspace: вертикальний інтервал між рядками.
- str: вказівник на рядок для друку, що зберігається у PROGMEM.
- dx: початкова x-координата для друку.
- dy0: початкова y-координата для друку.
- color: колір, який буде використано для друку.

Функція printSRAM має ті самі параметри, але параметр str вказує на рядок, що зберігається в SRAM, а не на PROGMEM.

Ключові компоненти та функціональність коду наступні:

1. Цикл друку: Обидві функції printPROGMEM і printSRAM мають основний цикл, який перебирає кожен символ у вхідному рядку і виконує необхідні дії для виведення символу на друк.

2. Обробка символів: У циклі друку код перевіряє тип поточного символу і виконує відповідну дію:

- Символ нового рядка ('\n'): Скидає координату x і збільшує координату y для переходу на наступний рядок.
- Пробіл (' '): Збільшує координату x для переходу на наступну позицію.

- Дійсний гліф: отримує дані гліфа з масиву шрифтів, викликає функцію bitblit для відображення гліфа на екрані та оновлює координату x.

- Неправильний гліф: збільшує координату x для переходу до наступної позиції.

3. Функція Bitblit: Функція bitblit відповідає за відображення гліфа на дисплеї. Вона повторює кожен рядок гліфа, перевіряє дані пікселів і викликає функцію putpixel, щоб встановити відповідні пікселі на дисплеї.

4. PROGMEM проти SRAM: Основна відмінність між printPROGMEM і printSRAM полягає у способі доступу до вхідного рядка. printPROGMEM зчитує рядок з PROGMEM за допомогою pgm_read_byte, тоді як printSRAM зчитує рядок безпосередньо з SRAM за допомогою розіменування вказівника (pstr++).

Наведена вище блок-схема Mermaid відображає структуру та логіку коду з детальними поясненнями для кожного підграфа та компонента. На діаграмі виділено основні функції, їх взаємодію та покроковий хід процесу рендерингу тексту. Код виглядає добре оптимізованим та ефективним, з використанням макросів та вбудованих функцій для мінімізації накладних витрат на виклик функцій та використання стеку. Однак є кілька потенційних областей для покращення:

1. Код обробки помилок не має явних механізмів обробки помилок. Було б корисно додати перевірки на невірні вхідні параметри або дані шрифту, щоб забезпечити коректну обробку помилок і надати користувачеві змістовний зворотний зв'язок.

2. Функцію bitblit потенційно можна було б додатково оптимізувати, використовуючи більш ефективні методи малювання пікселів, такі як використання графічних функцій з апаратним прискоренням або оптимізація шаблонів доступу до пам'яті.

3. Хоча код добре структурований і прокоментований, широке використання макросів та вбудованих функцій може ускладнити читання і розуміння коду. Розгляд альтернативних підходів, таких як розбиття функціональності на менші,

більш модульні функції, міг би покращити загальну читабельність та ремонтопридатність кодової бази. Загалом, наданий код демонструє надійну та ефективну реалізацію рендерингу тексту, з чітким фокусом на продуктивність та оптимізацію ресурсів. Детальна блок-схема Mermaid має допомогти користувачам зрозуміти функціональність та структуру коду, що дозволить їм ефективно використовувати та потенційно розширювати бібліотеку VGAX для своїх конкретних потреб. Блок-схеми алгоритмів представлено на рисунках 3.1-3.4.

3.2 Реалізація

3.2.1 Реалізація фреймбуфера

Бібліотека реалізує фреймбуфер з роздільною здатністю 120x60 пікселів, де кожен піксель представлений 2 бітами (що дозволяє отримати 4 кольори).

На Arduino MEGA роздільну здатність можна збільшити до 120x240 пікселів.

Кадровий буфер зберігається в SRAM і займає не менше 1800 байт.

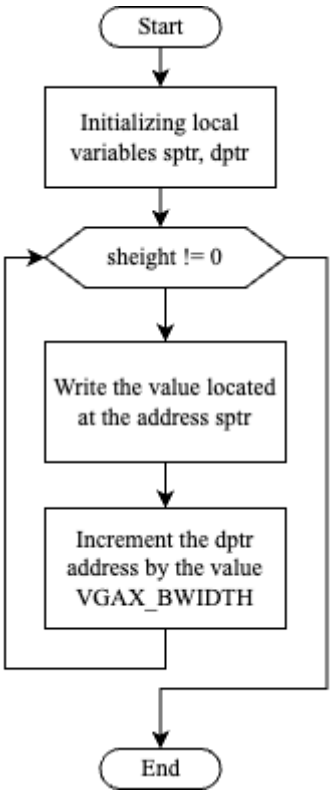


Рисунок 3.1 - Блок-схема алгоритму алгоритму

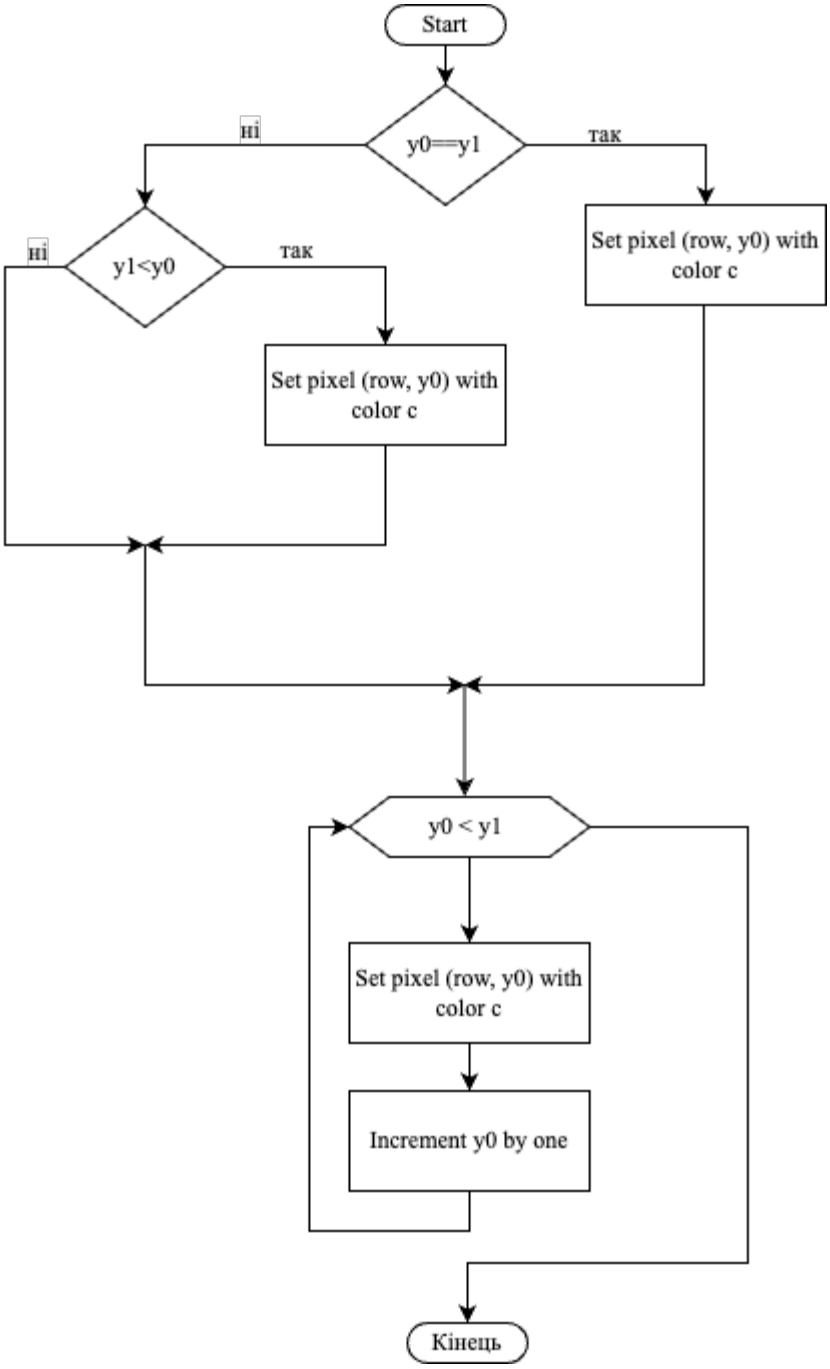


Рисунок 3.2 - В Блок-схема алгоритму алгоритму

Це означає, що на ATMega328 ваші програми не можуть використовувати більше 200 байт SRAM. На ATMega2560 у вас є більше SRAM, але якщо ви розширите фреймбуфер до 120x240 пікселів, у вас залишиться 800 байт вільної SRAM. Фреймбуфер VGAX використовує 2 біти на піксель, зберігаючи 4 пікселі в байті. У кожному байті крайній лівий піксель займає старші 2 біти, а крайній правий - молодші 2 біти.

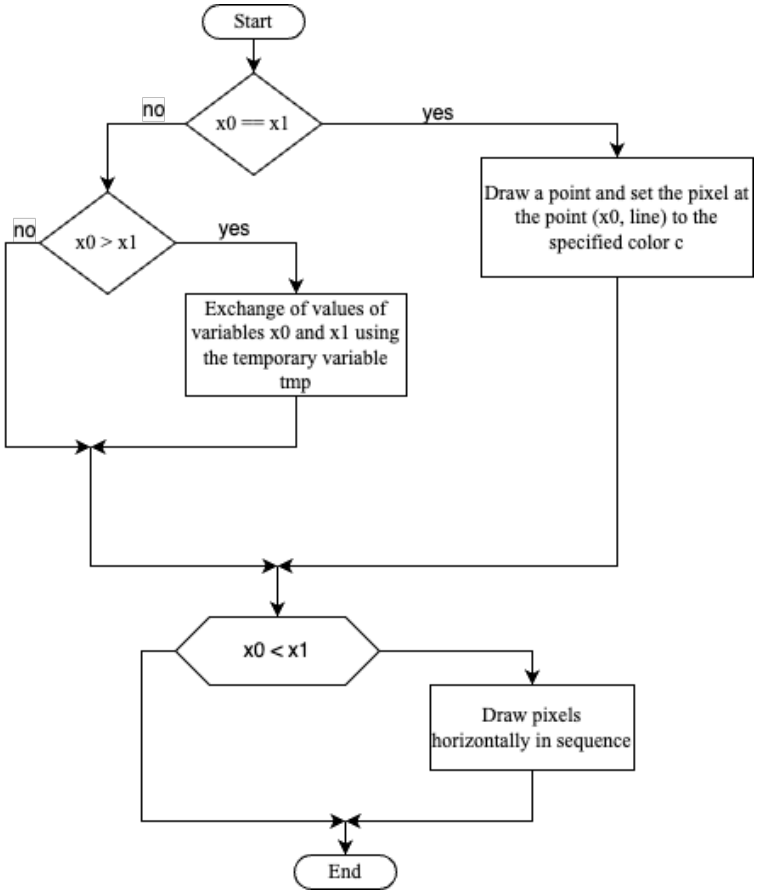


Рисунок 3.3 - Блок-схема алгоритму алгоритму

Формування відео представлено на рисунку 3.5.

На Arduino MEGA (ATMega2560) фреймбуфер можна розширити до 120x90 пікселів з квадратними пікселями або до 120x240 пікселів з прямокутними пікселями. Щоб увімкнути ці альтернативні роздільні здатності, потрібно розкоментувати відповідні константи у заголовному файлі VGAX.h.

3.2.2 Реалізація звуку

Аудіо-бібліотека забезпечує асинхронну генерацію тонів. Аудіосигнал генерується приблизно за 15 тактів під час горизонтальної розгортки VGA, що призводить до низької якості звуку, придатного для відтворення простих мелодій.

Ви можете використовувати функції tone(frequency) і noTone() для керування тривалістю нот.

Зумер слід під'єднати до виводу A0. Для реалізації потрібні два резистори по 470 Ом, два резистори по 68 Ом і гніздовий роз'єм DSUB15.

Зверніться до схеми, показаної на Рисунок 3.6 і Рисунок 3.7, щоб дізнатися про деталі підключення.

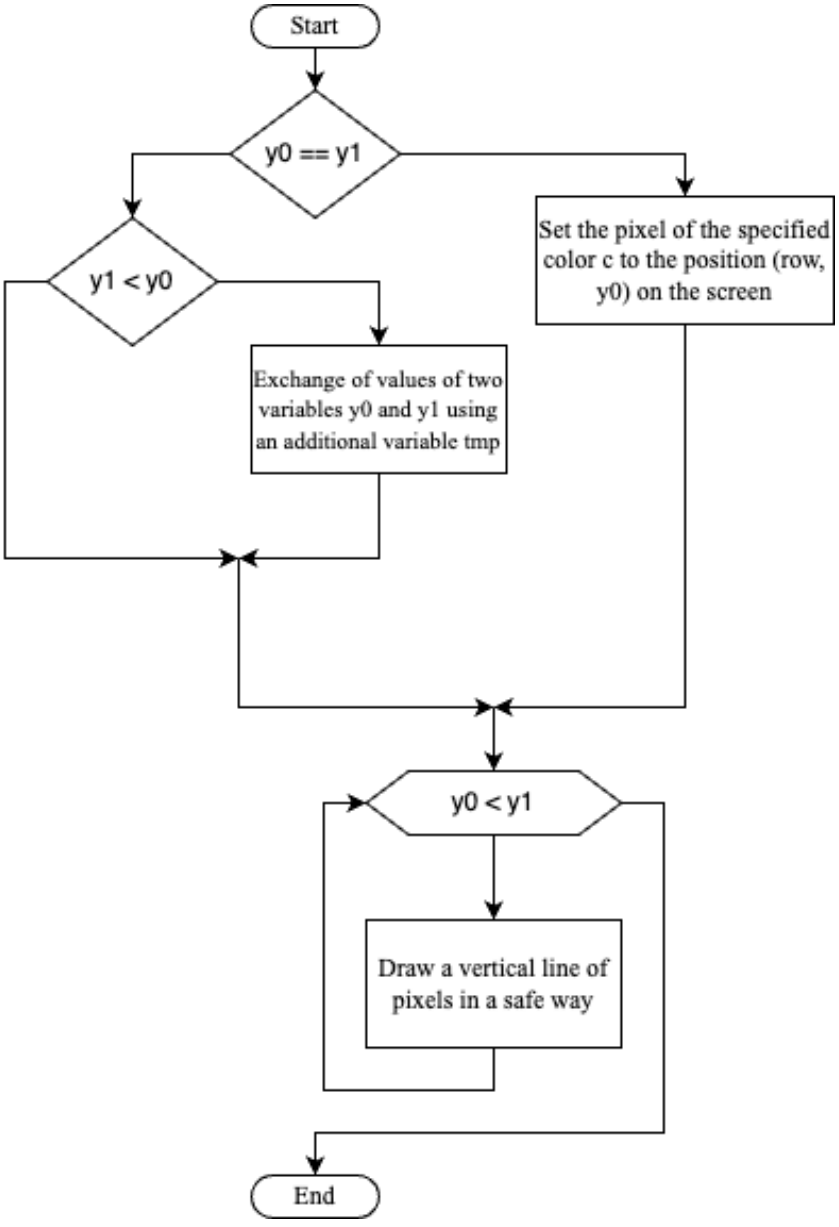


Рисунок 3.4 - Блок-схема алгоритму алгоритму

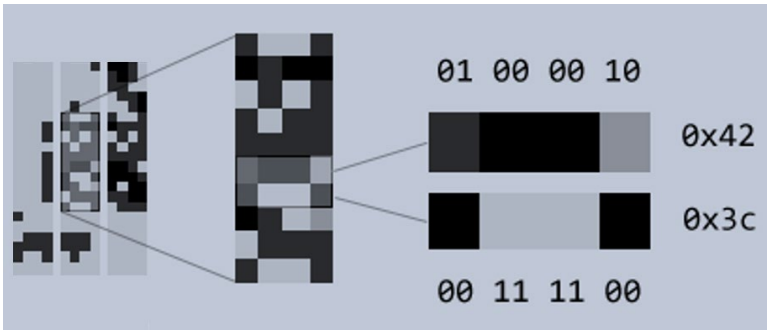


Рисунок 3.5 – Video formation

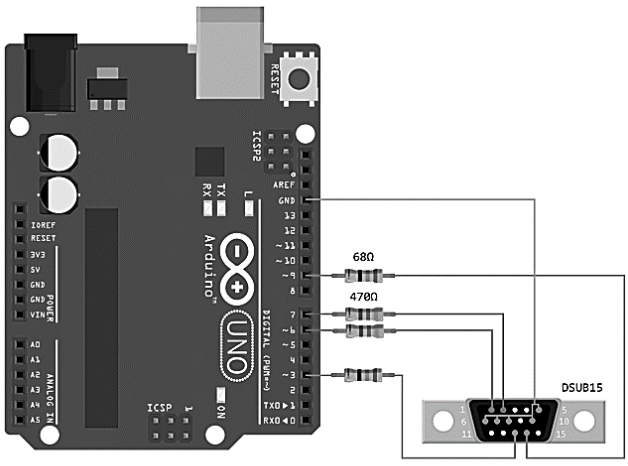


Рисунок 3.6 – System connetions

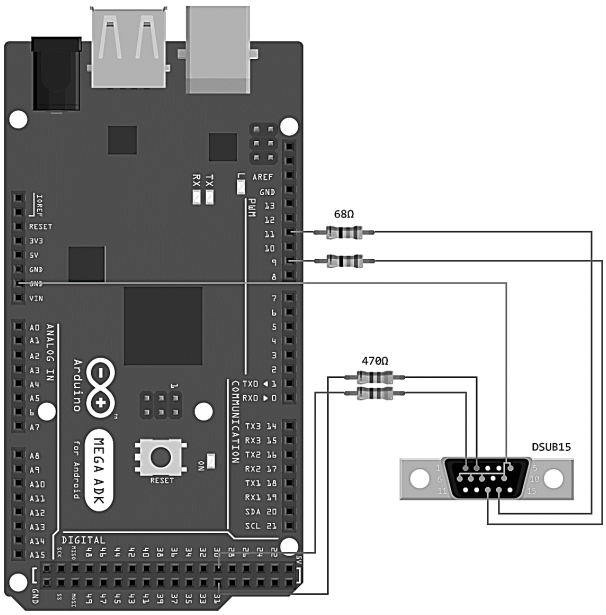


Рисунок 3.7 - System connetions

3.2.2 Реалізація кольорів

Чотири кольори, що генеруються бібліотекою, не є попередньо встановленими. На Arduino UNO ви можете підключити контакти 6 і 7 до будь-яких двох контактів RGB на роз'ємі VGA DSUB15, що дозволить вам вибрати бажану комбінацію кольорів.

Без додаткових компонентів можна досягти різних можливих комбінацій, як показано на рисунках 3.8-3.13.

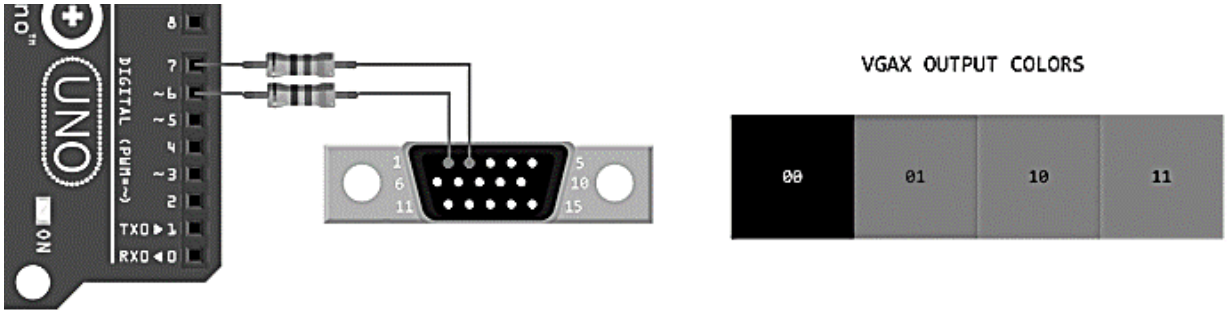


Рисунок 3.8 - Colors implementation

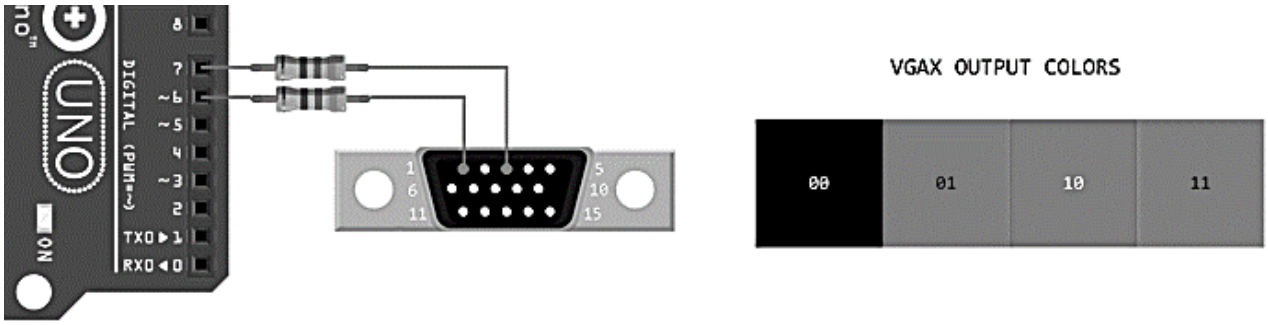


Рисунок 3.9 - Colors implementation

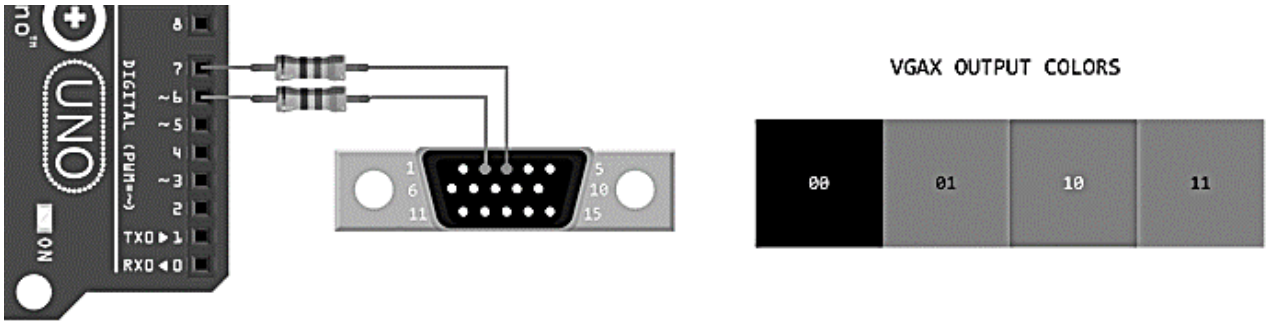


Рисунок 3.10 - Colors implementation

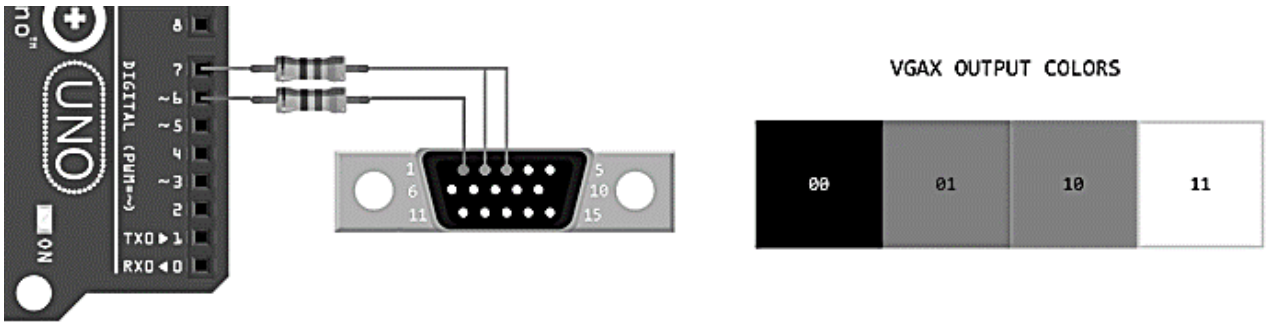


Рисунок 3.11 - Colors implementation

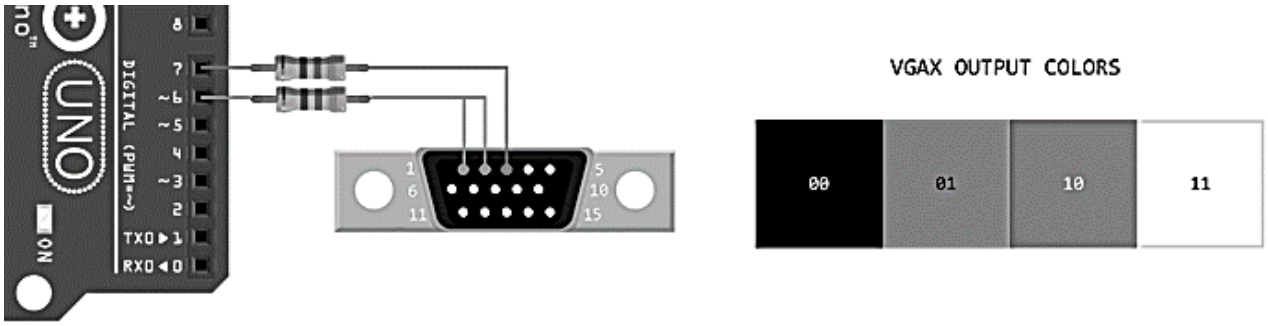


Рисунок 3.12 - Colors implementation

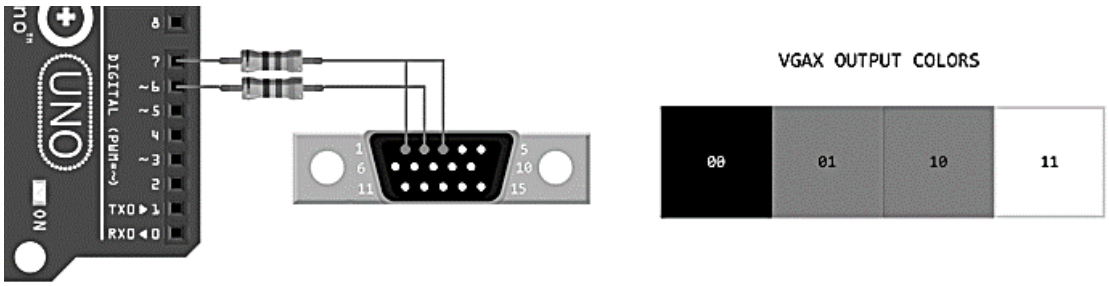


Рисунок 3.13 - Colors implementation

Виводи 6 і 7 на Arduino MEGA слід замінити на виводи 30 і 31, але для MEGA застосовується та сама логіка вибору кольору.

3.2.3 Генерація відео

Генерація відео здійснюється за допомогою PORTD, тому жоден з виводів PORTD не може бути використаний для інших цілей. Сигнал вертикальної синхронізації генерується на виводі 9. Хоча у версії Gammon використовується вивід 10, я вважаю за краще залишити виводи 10, 11, 12 і 13 доступними для

звичайного використання SPI. На Arduino MEGA PORTD замінено на PORTA, з вертикальною синхронізацією на виводі 11 і горизонтальною синхронізацією на виводі 9.

Бібліотека VGAX генерує відеосигнал, використовуючи переривання, що дозволяє виконувати інші завдання в межах функції main(). Код генерує пікселі рядків у функції main. Однак я вважаю за краще обробляти генерацію ліній у перериваннях, щоб звільнити MCU для інших завдань, таких як запуск ігор або відтворення звуків.

Зауважте, що додавання інших переривань порушить генерацію VGA-сигналу. Бібліотека використовує всі три таймери мікроконтролера ATMega328. На ATMega2560 доступні додаткові невикористані таймери. TIMER1 і TIMER2 налаштовані на генерацію імпульсів HSYNC і VSYNC, з кодом налаштування, створеним Ніком Геммоном (Nick Gammon), модифікованим для використання виводу 9 замість виводу 10. На ATMega2560 виводи HSYNC і VSYNC відрізняються.

TIMER0 використовується для усунення джиттера переривань. Я адаптував асемблерний трюк, спочатку написаний Charles CNLOHR. За замовчуванням TIMER0 використовується Arduino для реалізації певних функцій. Замість використання цих функцій, ви повинні використовувати альтернативні версії, надані цією бібліотекою.

3.2.4 Реалізація пікселів

Це простий приклад функції putpixel4. Функція putpixel4 додає 4 пікселі за один раз (рисунок 3.14). У прикладі показано функцію копіювання. Функція копіювання копіює всі пікселі з масиву байт. Масив має зберігатися у PROGMEM, а розмір масиву має дорівнювати розміру фреймбуфера VGAX:

Зверніть увагу, що розмір у байтах обчислюється з використанням BWIDTH, а не WIDTH, оскільки пікселі упаковуються у байти (4 пікселі в одному байті).

Рисунок 3.14 - Результат

3.2.5 Реалізація шрифту

Створення шрифту з одного зображення за допомогою 2bitfont складається з декількох кроків.

Налаштування середовища включає встановлення Python:

bash

pip install pillow

pip install 2bitfont

Потім нам потрібно підготувати зображення і переконатися, що це чорно-біле растрове зображення, де кожен символ є чітким і правильно вирівняним у сітці.

Щоб підготувати зображення, ми повинні перетворити його у растрове зображення з роздільною здатністю 1 біт на піксель. Це можна зробити за допомогою графічного редактора або програмно за допомогою PIL.

python

з PIL імпортуємо зображення

Відкриваємо зображення

image = Image.open(«шлях_до_вашого_зображення.png»)

Перетворити зображення у чорно-біле

bw_image = image.convert(«1»)

Зберегти перетворене зображення

bw_image.save(«bw_image.bmp»)

Після цього потрібно обрізати окремі символи. Це можна автоматизувати, якщо символи знаходяться в сітці:

```
python
def crop_characters(path до_зображення, char_width, char_height, output_dir):
    image = Image.open(шлях до зображення)
    img_width, img_height = image.size
    num_chars_x = img_width // char_width
    num_chars_y = img_height // char_height
    for y in range(num_chars_y)
        for x in range(num_chars_x):
            left = x char_width
            upper = y char_height
            right = left + char_width
            lower = upper + char_height
            char_image = image.crop((left, upper, right, lower))
char_image.save(f«{output_dir}/char_{ynum_chars_x + x}.bmp»)
crop_characters(«bw_image.bmp», char_width=8, char_height=8, output_dir=«chars»)
```

Крок 3 - виділення шрифту за допомогою 2bitfont^.

1. Ініціалізуйте 2bitfont, створіть новий об'єкт шрифту і додайте кожен символ до шрифту.

```
python
import os
з bitfont import Font, Glyph
font = Font()
char_dir = «chars»
char_files = sorted(os.listdir(char_dir))
for i, char_file in enumerate(char_files):
    char_image = Image.open(os.path.join(char_dir, char_file))
    pixels = list(char_image.getdata())
    width, height = char_image.size
    bitmap = [pixels[i width:(i + 1) width] for i in range(height)].
```

```
        glyph = Glyph(bitmap)
        font.add_glyph(chr(32 + i), glyph) # Починаємо з ASCII 32 (пробіл)
    # Зберегти шрифт
    за допомогою open(«custom_font.bf», «wb») як f:
        font.save(f)
```

Щоб використовувати шрифт, ми повинні завантажити f у додатку:

```
    python
    з bitfont import Font
    with open(«custom_font.bf», «rb») as f:
        custom_font = Font.load(f)
```

Результати роботи програми наведено на рисунках 3.15-3.19.

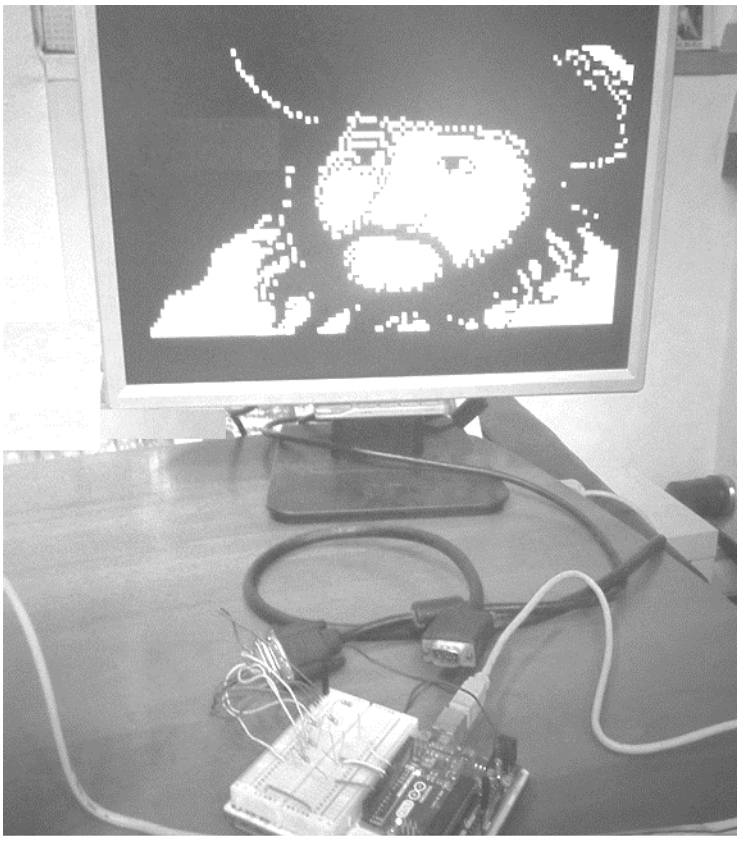


Рисунок 3.15 – Екран із зображенням

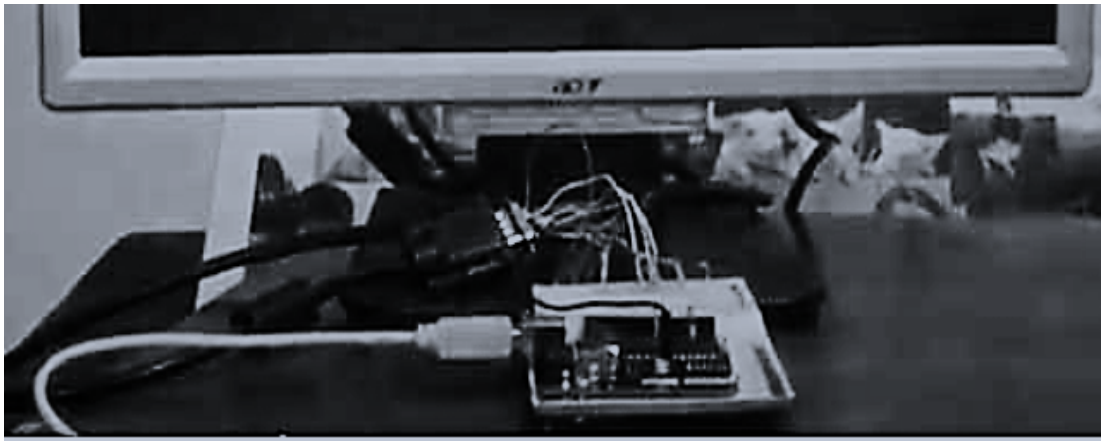


Рисунок 3.16 - Пристрій

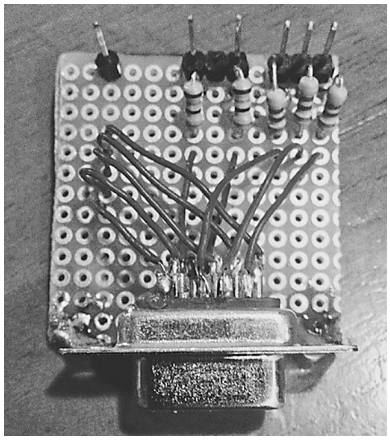


Рисунок 3.17 - Конектор

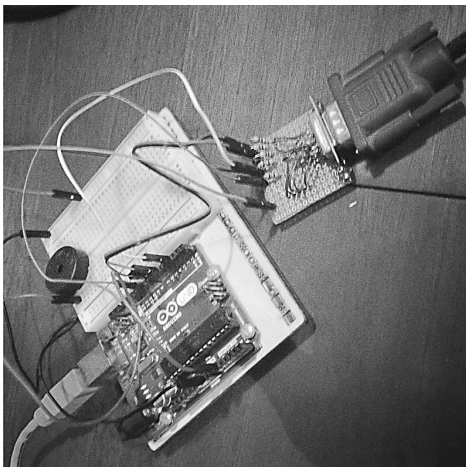


Рисунок 3.18 – З’єднання

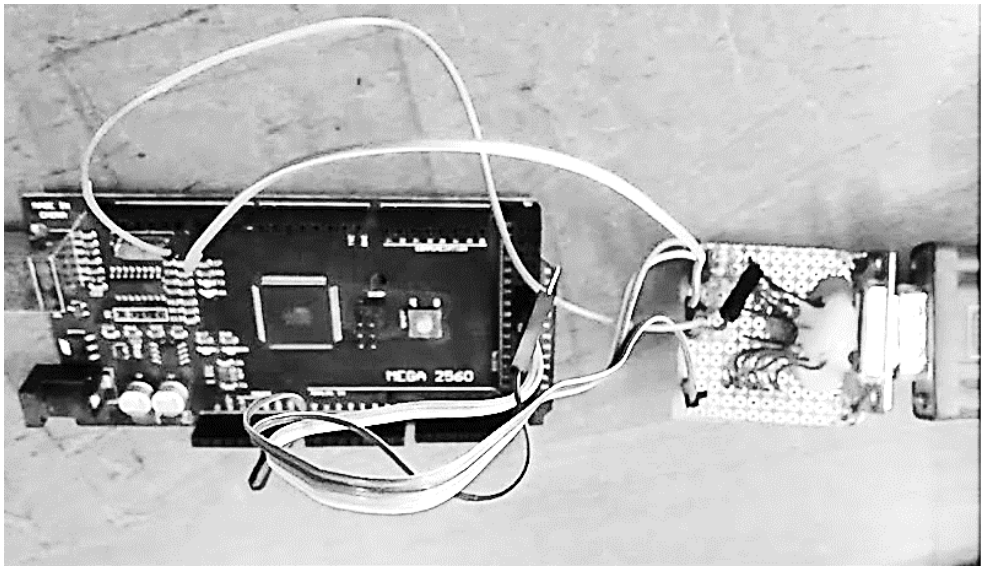


Рисунок 3.19 – Пристрій

3.6. Висновки

У розділі детально описано реалізацію спеціального системного програмного забезпечення, яке слугує бібліотекою VGA для Arduino UNO та Arduino MEGA. Ця бібліотека полегшує створення кольорового відео у форматі VGA, вимагаючи лише мінімального налаштування з 4 резисторів та одного роз'єму DSUB15. Вона ефективно працює на ATMega328, пропонуючи роздільну здатність 120x60px з 4 кольорами на піксель, що зберігаються у фреймбуфері в SRAM. Крім того, адаптація бібліотеки VGAX для ESP8266 розширює підтримку роздільної здатності до 512x480 пікселів з кадровим буфером 1 біт/с. У розділі також представлено аудіофункціональність, що дозволяє асинхронну генерацію тонів у горизонтальному задньому порті VGA, хоча і з обмеженою якістю. Нарешті, викладено міркування щодо вибору кольору та використання виводів для оптимальної продуктивності на платформах Arduino UNO та Arduino MEGA. Завдяки цим реалізаціям користувачі можуть заглибитися в різноманітні проекти, пов'язані з генерацією VGA-відео та простим аудіовиходом, відкриваючи шляхи для творчих пошуків та експериментів у сфері програмування Arduino.

ВИСНОВКИ

Перший розділ розглядає аспекти звукових кодеків, висвітлюючи їхнє значення, функціональність і застосування в сучасних цифрових системах. Дослідження починається з фундаментальної концепції звукових кодеків, що забезпечує розуміння їхньої ролі в кодуванні та декодуванні аудіосигналів. Це закладає основу для розуміння важливості звукових кодеків для різних застосувань, особливо для стиснення даних.

Другий розділ присвячено аналізу функціональних можливостей мікроконтролера ATmega, який вирізняється своїми надійними системами тактової частоти та широкими можливостями, призначеними для підвищення функціональності та продуктивності в різноманітних додатках.

У третьому розділі детально описано реалізацію спеціального системного програмного забезпечення, яке слугує бібліотекою VGA для Arduino UNO та Arduino MEGA. Ця бібліотека полегшує створення кольорового відео у форматі VGA, вимагаючи лише мінімального налаштування з 4 резисторів та одного роз'єму DSUB15. Вона ефективно працює на ATMega328, пропонуючи роздільну здатність 120x60px з 4 кольорами на піксель, що зберігаються у фреймбуфері в SRAM. Крім того, адаптація бібліотеки VGAX для ESP8266 розширює підтримку роздільної здатності до 512x480 пікселів з кадровим буфером 1 біт/с. У розділі також представлено аудіофункціональність, що дозволяє асинхронну генерацію тонів у горизонтальному задньому порті VGA, хоча і з обмеженою якістю. Викладено міркування щодо вибору кольору та використання виводів для оптимальної продуктивності на платформах Arduino UNO та Arduino MEGA. Завдяки цим реалізаціям користувачі можуть заглибитися в різноманітні проекти, пов'язані з генерацією VGA-відео та простим аудіовиходом, відкриваючи шляхи для творчих пошуків та експериментів у сфері програмування Arduino.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Ritikapahuja, Narendra Kumar. Android phone controlled Bluetooth robot using 8051 microcontroller. IJSER. 2014. Vol. 2. Issue 7. Pp. 14-17.
2. Sharma A., Verma R., Gupta S., Bhatia S. K. Android phone controlled robot using Bluetooth. IJEEE. 2014. Vol. 7. Pp. 443-448.
3. Selvam M. Smart phone based robotic control for surveillance application. IJRET. 2014. Vol. 3. Issue 3. Pp. 229-232.
4. Van Delden S., Whigham A. A Bluetooth-based architecture for Android communication with an articulated robot. IEEE. 2012. Pp. 104-108.
5. Lu X., Liu W., Wang H., Sun Q. Robot control design based on smartphone. IEEE. 2013. Pp. 2820-2823.
6. Pahuja R., Kumar N. Android phone controlled Bluetooth robot using 8051 microcontroller. IJSER. 2014. Vol. 2. Issue 7. Pp. 14-17.
7. Sharma A., Verma R., Gupta S., Bhatia S. K. Android phone controlled robot using Bluetooth. IJEEE. 2014. Vol. 7. Pp. 443-448.
8. Selvam M. Smart phone based robotic control for surveillance application. IJRET. 2014. Vol. 3. Issue 3. Pp. 229-232.
9. Van Delden S., Whigham A. A Bluetooth-based architecture for Android communication with an articulated robot. IEEE. 2012. Pp. 104-108.
10. Lu X., Liu W., Wang H., Sun Q. Robot control design based on smartphone. IEEE. 2013. Pp. 2820-2823.
11. Dickey N., Banks D., Sukittanon S. Home automation using cloud network and mobile devices. IEEE Southeastcon. 2012. DOI: 10.1109/secon.2012.6197003.
12. Yashiro T., Kobayashi S., Koshizuka N., Sakamura K. An Internet of Things (IoT) architecture for embedded appliances. IEEE Region 10 Humanitarian Technology Conference. 2013. DOI: 10.1109/r10-htc.2013.6669062.
13. Amudha A. Home automation using IoT. 2017.

14. Anusha S., Madhavi M., Hemalatha R. Home automation using ATmega328 microcontroller and Android application. International Research Journal of Engineering and Technology (IRJET). 2015. Vol. 2. Issue 6.

15. Rani P. J., Bakthakumar J., Kumaar B. P., Kumaar U. P., Kumar S. Voice controlled home automation system using natural language processing (NLP) and Internet of Things (IoT). 2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM). 2017. DOI: 10.1109/iconstem.2017.8261311.

16. Bello O., Zeadally S. Intelligent device-to-device communication in the Internet of Things. IEEE Systems Journal. 2016. 10(3). Pp. 1172-1182. DOI: 10.1109/jsyst.2014.2298837.

17. Piyare R., Tazil M. Bluetooth based home automation system using cell phone. IEEE 15th International Symposium on Consumer Electronics (ISCE). 2011. Pp. 192-195.

18. Naresh D., Chakradhar B., Krishnaveni S. Bluetooth based home automation and security system using ARM9. International Journal of Engineering Trends and Technology (IJETT). 2013. Vol. 4. Pp. 4052.

19. Wong E. M. A phone-based remote controller for home and office automation. IEEE Transactions on Consumer Electronics. 1994. Vol. 40. Pp. 28-34.

20. Koyuncu B. PC remote control of appliances by using telephone lines. IEEE Transactions on Consumer Electronics. 1995. Vol. 41. Pp. 201-209.

21. Coskun I., Ardam H. A remote controller for home and office by telephone. IEEE Transactions on Consumer Electronics. 1998. Vol. 44. Pp. 1291-1297.

22. Al-Thobaiti B. M., Abosolaiman I. I., Alzahrani M. H., Almalki S. H., Soliman M. S. Design and implementation of a reliable wireless real-time home automation system based on Arduino Uno single-board microcontroller. International Journal of Control and Automation Systems. 2014. Vol. 3. Pp. 11-15.

23. Shepherd R. Bluetooth wireless technology in the home. Electronics Communication Engineering Journal. 2001. Vol. 13. Pp. 195-203.

24. Sriskanthan N., Tan F., Karande A. Bluetooth based home automation system. Microprocessors and Microsystems. 2002. Vol. 26. Pp. 281-289.

25. Alkar A. Z., Buhur U. An Internet based wireless home automation system for multifunctional devices. IEEE Transactions on Consumer Electronics. 2005. Vol. 51. Pp. 1169-1174.

26. Zhang A. R., Zhang J. L. The building of home automation electricity distribution system based on PLC. Advanced Materials Research. 2012. Vol. 442. Pp. 407-411.

27. Madan V., Reddy S. R. GSM-Bluetooth based remote monitoring and control system with automatic light controller. International Journal of Computer Applications. 2012. Vol. 46. Pp. 20-28.

28. Debono C. J., Abela K. Implementation of a home automation system through a central FPGA controller. IEEE 16th Mediterranean Electronics Conference. 2012. Pp. 641-644.

29. El-Shafee A., Hamed K. A. Design and implementation of a WIFI based home automation system. World Academy of Science Engineering and Technology. 2012. Vol. 68. Pp. 2177-2180.

30. Tutorials for Arduino. URL: https://www.arduino.cc/en/Tutorial/HomePage.

31. Abidin Z. Rancang bangun sistem monitoring dan controlling pintu air dam berbasis Arduino menggunakan implementasi Internet of Things. JATI (Jurnal Mahasiswa Teknik Informatika). 2018. Vol. 2. Issue 2. Pp. 282-289.

32. Abbas Z. W. Design and implementation of a smart digital door lock based on BBS PRNG. Journal of Kerbala University. 2018. Vol. 14. Issue 2. Pp. 170-178.

33. Adriano D. B., Budi W. A. C. IoT-based integrated home security and monitoring system. Journal of Physics: Conference Series. 2018. Vol. 1140. No. 1. P. 012006. IOP Publishing.

34. Alqarni A. S. Honeybee foraging, nectar secretion, and honey potential of wild jujube trees, Ziziphus nummularia. Neotropical Entomology. 2015. Vol. 44. Issue 3. Pp. 232-241.

35. Badamasi Y. A. The working principle of an Arduino. 2014 11th International Conference on Electronics, Computer and Computation (ICECCO). 2014. Pp. 1-4. IEEE.

36. Durani H., Sheth M., Vaghasia M., Kotech S. Smart automated home application using IoT with Blynk app. 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT). 2018. Pp. 393-397. IEEE.

37. Hutabarat D. P., Budijono S., Saleh R. Development of home security system using ESP8266 and android smartphone as the monitoring tool. IOP Conference Series: Earth and Environmental Science. 2018. Vol. 195. No. 1. P. 012065. IOP Publishing.

38. Idris K., Yusof N. N. M., Hipni H. I., Rahman T. A. F. T. A. Ziziphus mauritiana in Bathing Deceased: Reflection from Hadith Sahih Al-Bukhari 1253, Book 23, Hadith 344. 2020.

39. Ismail N., Rajendran S., Tak W. C., Xin T. K., Anuar N. S. S., Zakaria F. A., Rahim H. A. Smart irrigation system based on Internet of Things (IoT). Journal of Physics: Conference Series. 2019. Vol. 1339. No. 1. P. 012012. IOP Publishing.

40. Jaedun A. Metodologi penelitian eksperimen. Fakultas Teknik UNY. 2011. Vol. 12.

41. Jariyayothin P., Jeravong-aram K., Ratanachaijaroen N., Tantidham T., Intakot P. IoT Backyard: Smart watering control system. 2018 Seventh ICT International Student Project Conference (ICT-ISPC). 2018. Pp. 1-6. IEEE.

42. Kahrizi D., Molsaghi M., Faramarzi A., Yari K., Kazemi E., Farhadzadeh A. M., Yousofvand N. Medicinal plants in holy Quran. American Journal of Science Research. 2012. Vol. 42. Pp. 62-71.

43. Kamelia L., Ramdhani M. A., Faroqi A., Rifadiapriyana V. Implementation of automation system for humidity monitoring and irrigation system. IOP Conference Series: Materials Science and Engineering. 2018. Vol. 288. No. 1. P. 012092. IOP Publishing.

44. Karim A., Nouman M., Munir S., Sattar S. Pharmacology and phytochemistry of Pakistani herbs and herbal drugs used for treatment of diabetes. International Journal of Pharmacology. 2011. Vol. 7. Issue 4. Pp. 419-439.

45. Malichah T. Buah-buahan dalam Al-Qur'an (kajian tematik). Doctoral dissertation, UIN Walisongo. 2016.

46. Mardika A. G., Kartadie R. Mengatur kelembaban tanah menggunakan sensor kelembaban tanah YL-69 berbasis Arduino pada media tanam pohon gaharu. JoEICT (Journal of Education And ICT). 2019. Vol. 3. Issue 2.

47. Muchlas M. T., Bailey C., Freeman M. Simulator Breadboard: Perangkat Pembelajaran Teknik Digital. UAD PRESS. 2021.

48. Pribadi E. R. Pasokan dan Permintaan Tanaman Obat Indonesia Serta Arah Penelitian dan Pengembangannya. Perspektif. 2015. Vol. 8. Issue 1. Pp. 52-64.

49. Putra W. K. Dampak Perkembangan Pariwisata Terhadap Kondisi Ekonomi Sosial di Desa Cihideung. Doctoral dissertation, Universitas Pendidikan Indonesia. 2013.

50. Qamariah N. Ethnobotanical Study of Quran Plants. Pharmacognosy Journal. 2019. Vol. 11. Issue 5.

51. Rahayu A. A. D., Riendriasari S. D. Pengaruh Beberapa Jenis Zat Pengatur Tumbuh terhadap Pertumbuhan Stek Batang Bidara Laut (Strychnos ligustrina Bl). Jurnal Perbenihan Tanaman Hutan. 2016. Vol. 4. Issue 1. Pp. 25-31.

52. Raju K. L., Chandrani V., Begum S. S., Devi M. P. Home automation and security system with Node MCU using Internet of Things. 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN). 2019. Pp. 1-5. IEEE.

53. Sudirman A., Yusoff Y., Mun'im A., Yanti D., Ariyanto R. Medicinal plants database and three-dimensional structure of the chemical compounds from medicinal plants in Indonesia. arXiv preprint arXiv:1111.7183. 2011.

54. Yusof M. Y., Salleh R. A. A. R., Yahaya F., Hassan P., Abd Munir Mohamed Noh M. Z., Abidin H. Z. Funeral management in the Malay world: local knowledge and

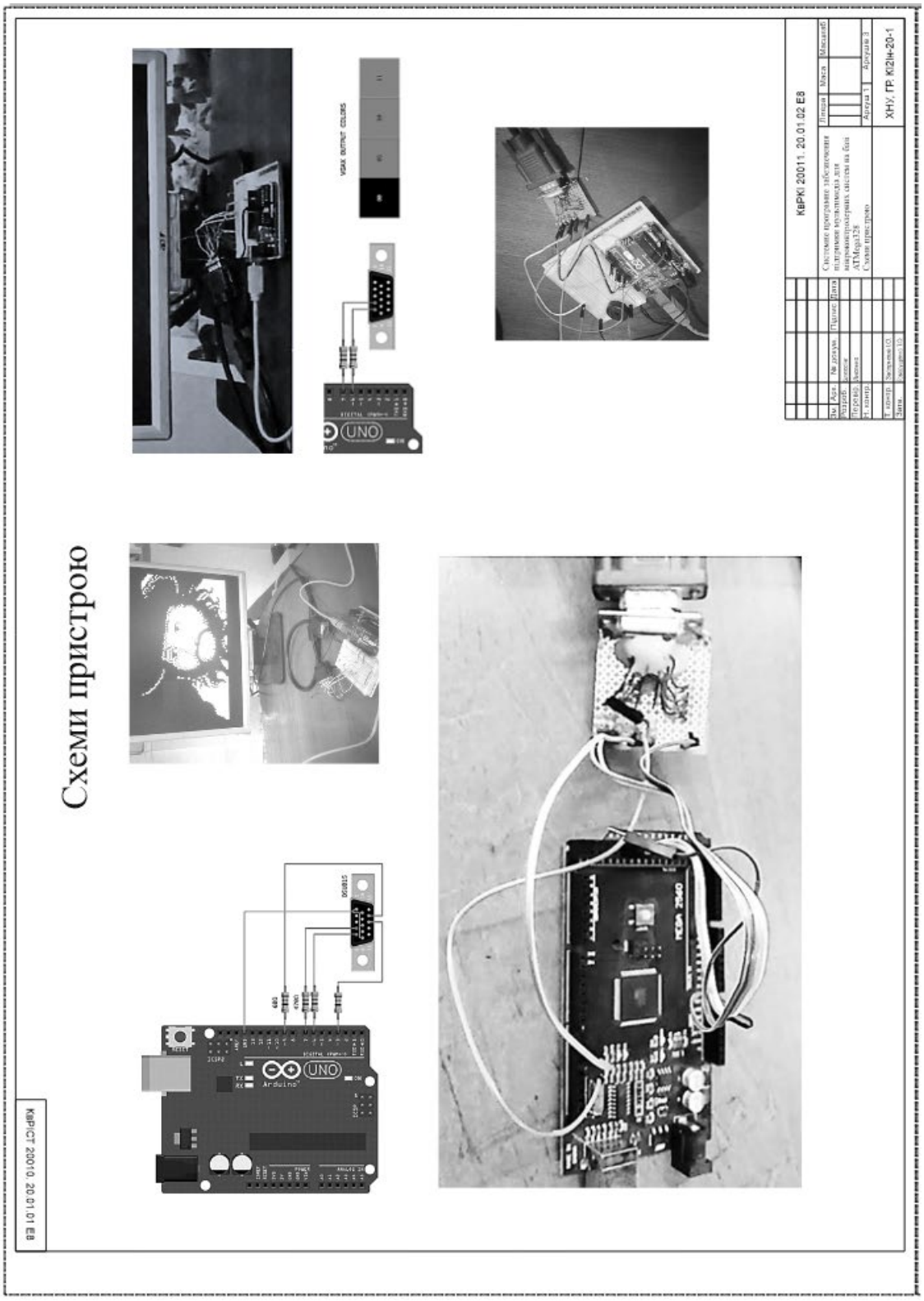
practices. Journal of Applied Environmental Biological Sciences. 2017. Vol. 7. Issue 1S. Pp. 72-77.

55. Each reference is formatted with a clear structure, separating the author(s), title, journal or conference name, volume, issue, pages, and year, following the pattern you provided.

Додаток А

(обов’язковий)

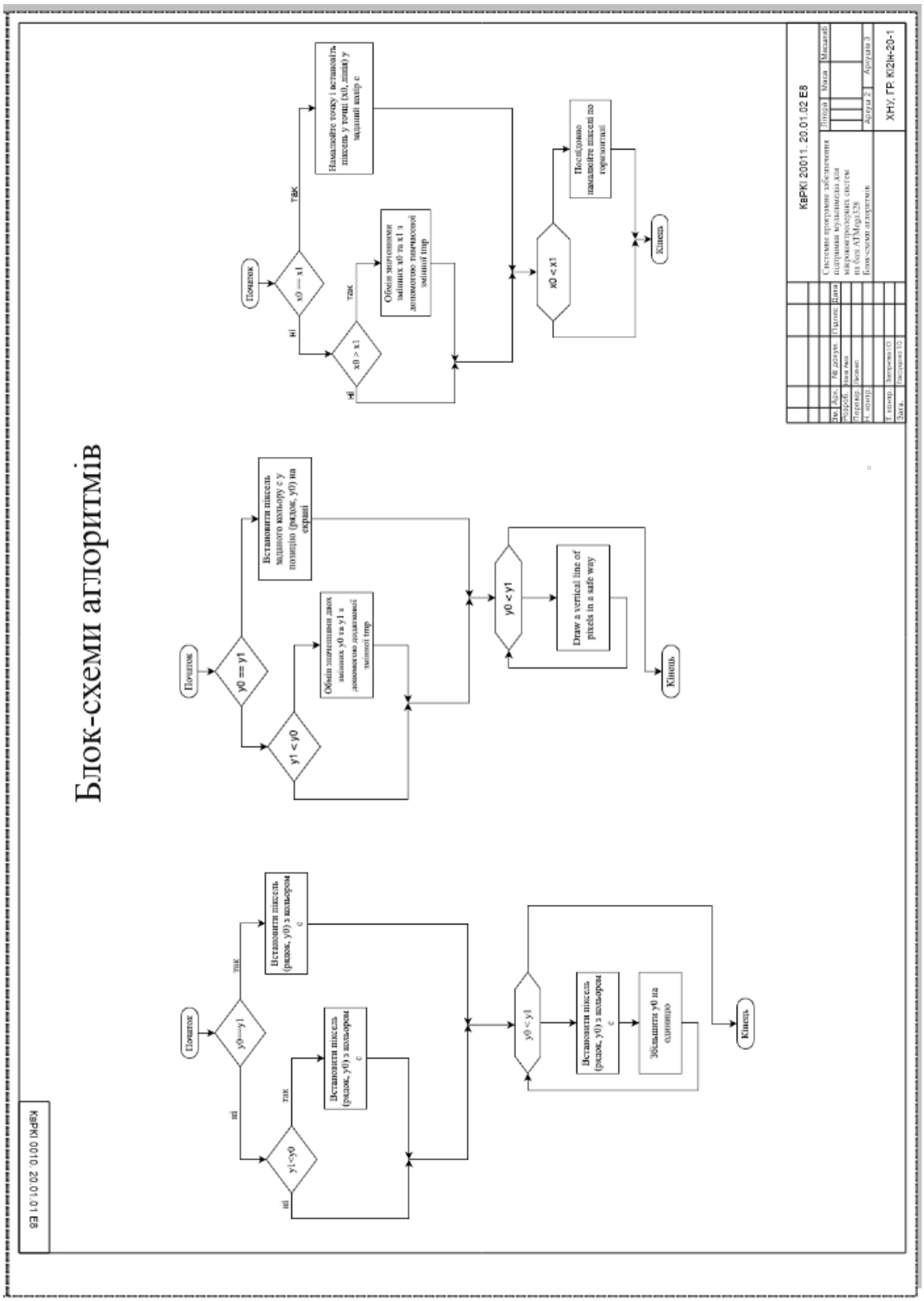
Копія креслення «Схеми пристрою»



Додаток Б

(обов’язковий)

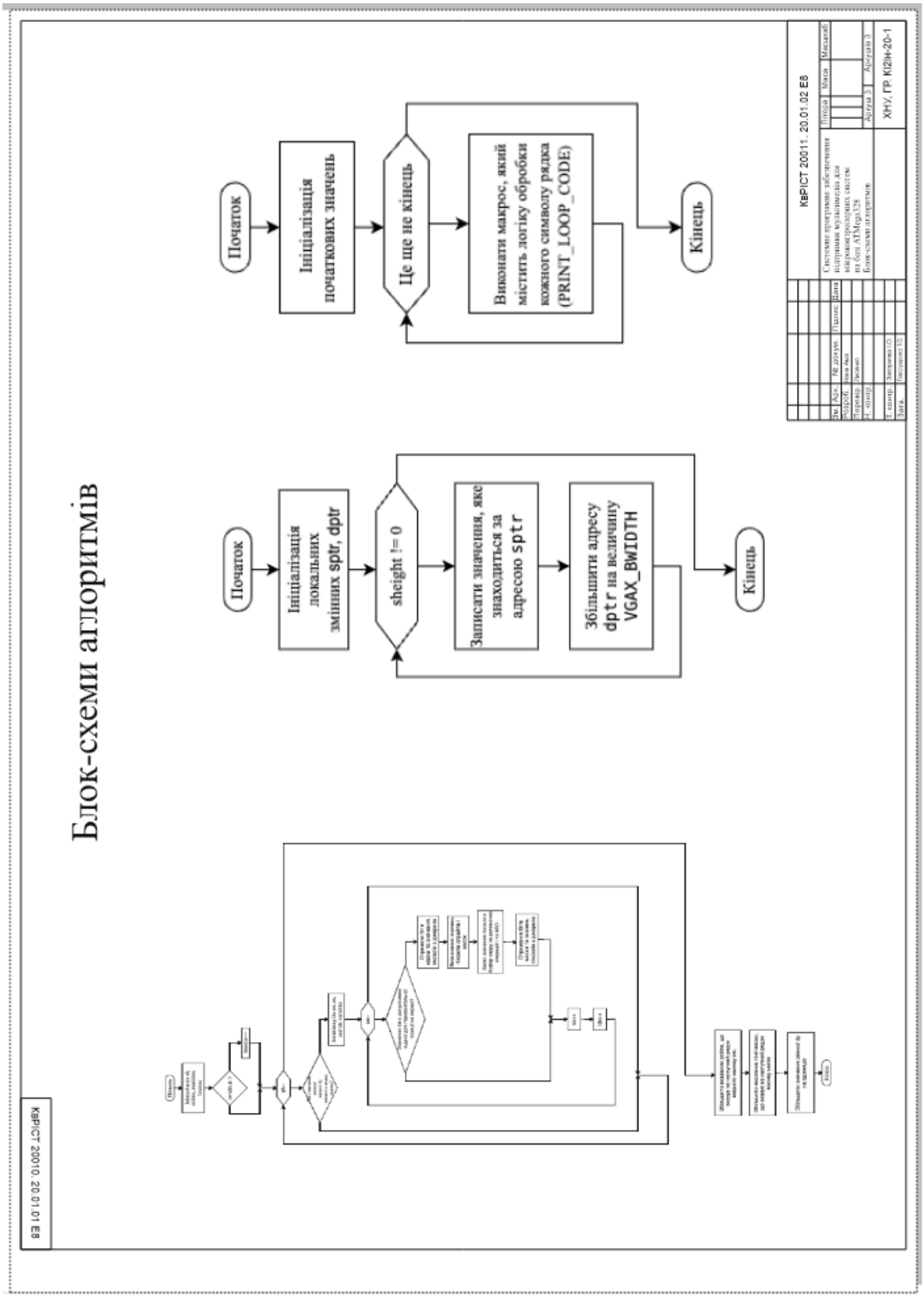
Копія креслення «Блок-схеми алгоритмів»



Додаток В

(обов’язковий)

Копія креслення «Блок-схеми алгоритмів»



Khmelnytsky National University
Faculty of Information Technology
Department of Computer Engineering and Information Systems

QUALIFICATION WORK

bachelor

Educational level

Multimedia support system software for microcontroller systems based on ATMega328

topic

QWCE 20011. 20.01.02 NE

code

Branch of knowledge *12 "Information technologies"*

Code, name

Specialty *123 "Computer Engineering"*

Cipher, name

Educational program *"Computer engineering and programming"*

Name

Performed by: IV year student, group КІ2ін-20-1 _____ Nana Ama Offul- Quaye

Signature Initials, surname

Head _____ S.M. Lysenko

Signature, date Initials, surname

Normocontroller _____ I.O. Zasornova

Signature, date Initials, surname

I admit to protection:
Chief computer department
engineering and information
systems _____ T.O. Govorushchenko

Signature Initials, surname

"13" June 2024

Khmelnytskyi 2024

Khmelnytskyi National University

Faculty OF INFORMATION TECHNOLOGIES

Department OF COMPUTER ENGINEERING AND INFORMATION SYSTEMS

BACHELOR'S level of education

Field of knowledge 12 INFORMATION TECHNOLOGIES

Specialty 123 COMPUTER ENGINEERING

Educational program "COMPUTER ENGINEERING AND PROGRAMMING"

I APPROVE

Chief departments

T.O. Govorushchenko

" 10 " 01 2024

TASK
ON BACHELOR'S QUALIFICATION WORK

Nana Ama Offul- Quaye

Surname, first name, patronymic of the student

1. Topic of the project (work) Multimedia support system software for microcontroller systems based on ATMega328

Project manager (works) S.M. Lysenko, doctor of technical sciences, prof.

Surname, first name, patronymic, academic degree, academic

Approved by the order of the rector of the university dated 15 02.2024 No. 8

2. The deadline for student submission of a project (work) to the department is June 1, 2024.

3. Initial data for the project (work) Assignment for qualification work

4. Content of the explanatory note (list of issues to be developed)

Multimedia support systems

Descripion of the components for the multimedia support system software for microcontroller systems based on atmega328

Implementation of the multimedia support system software for microcontroller systems based on atmega328

5. List of graphic material (indicating mandatory drawings)

Device schemes

The flowchart of the algorithm

The flowchart of the algorithm

6. Consultants of sections of the diploma project (work)

Section	Surname, initials and position of the consultant	Signature, date	
		issued the task	accepted the task
Standard control	Zasornova I.O., Ph.D, associate professor		
Anti-plagiarism	Nicheporuk A.O., Ph.D, associate professor		

7. Issue date of the assignment "_10_" _____01_____2024

CALENDAR PLAN

No. z/p	Name of stages (sections) diploma project (work)	Deadline stages of the project (work)	Note
1	Choosing a research direction and agreeing the topic of the qualification work with the supervisor	10.01.2024	done
2	Acquaintance with the subject area; formulation of the goal and objectives of the research; definition of the object and subject of research	01.02.2024	done
3	Work on chapter 1 - research of the subject area and formulation of the problem	01.03.2024	done
4	Work on section 2 - selection of components for designing a system of adaptive application of monitoring elements of a reconnaissance UAV	04.01.2024	done
5	Work on chapter 3 – designing a system of adaptive application of monitoring elements of a reconnaissance UAV	04.29.2024	done
6	Issuance of an explanatory note according to the requirements	05.25.2024	done
7	Preliminary protection of the WKR	05.26.2024	done
8	Protection of the WKR at the meeting of the EC	June 2024	

Student _____ Nana Ama
Signature Initials, surname

Head of work _____ S.M. Lysenko
Signature Initials, surname

№ of Row	format	Designation	Name	Am. of sheets	№ instance	Note
			Text documents			
1		QWCE. 20011. 20.01.02 EN	Explanatory note	60		
			Graphic materials			
2		QWCE. 20011. 20.01.02 E8	Device schemes	1		
3		QWCE. 20011. 20.01.02 E8	The flowchart of the	1		
			algorithm			
4		QWCE. 20011. 20.01.02 E8	The flowchart of the	1		
			algorithm			

M.	Let	№ of doc	Sign.	Date	QWCE.20011. 20.01.02 EN		
Author		Nana Ama			Project information	Letter	Sheet	Sheets
Supervisor		Lysenko				E	1	1
						KhNU, КІіи-20-1		
Reg. contr.		Zasornova						
Approve		Hovorushchenko		13.06				

ANNOTATIN

The subject of the qualification work: " Multimedia support system software for microcontroller systems based on ATMega328".

The author of the work: Nana Ama.

Head of work: Lysenko Sergii.

Explanatory note: 60 pp., 32 figures, 3 tables, 4 appendices, 55 sources.

Graphic part: 3 drawings.

MULTIMEDIA SUPPORT SYSTEM, SYSTEM SOFTWARE, MICROCONTROLLER, MICROCONTROLLER SYSTEM, ATMEGA328.

The aim of the thesis is to develop multimedia support system software for microcontroller systems based on ATMega328.

The object of the research is the c multimedia support system software.

The subject of the study is a multimedia support system software for microcontroller systems based on ATMega328.

During the conduct of this study, the method of systematic literature review was used to study and analyze the subject area of this study from text sources of information.

Student signature

10.06.2024

date

CONTENT

INTRODUCTION

The relevance of system software to support multimedia for ATMega328-based microcontroller systems is an interesting topic, given the recent advances in embedded systems and the growing demand for integration of multimedia functions in various applications.

ATMega328-based systems are often used in consumer electronics applications where basic multimedia capabilities such as audio playback and simple graphical displays are sufficient. These include applications such as

- Digital watches and clocks with a graphical interface.

- Home automation devices with a display and sound alerts.

- Toys and educational kits that use sound and visuals for interactive learning.

Multimedia support in IoT devices can improve the user experience. For example:

- Smart home devices can use audio alerts or simple graphical displays to communicate status or messages.

- Wearable technologies can benefit from small graphical displays for user interfaces.

The ATMega328 is widely used in prototyping and educational projects. The multimedia support allows students and hobbyists to create more interesting and complex projects, enhancing the learning process by adding visual and audio elements.

Despite these limitations, creative techniques such as code optimization, use of external memory, and efficient libraries can provide basic multimedia functionality.

The low power consumption of the ATMega328 makes it suitable for battery-powered devices, but multimedia functions must be carefully managed to avoid excessive power consumption.

While the ATMega328 provides basic multimedia support, there is a trend toward integrating these microcontrollers with more powerful systems such as: teaming up with ARM-based microcontrollers. Combining with more powerful microcontrollers can offload heavy multimedia tasks while maintaining the advantages of ATMega328's low

cost and simplicity, and use of coprocessors. Offloading multimedia processing to dedicated chips or coprocessors can improve performance without overloading the ATMega328.

As the IoT continues to grow, the need for low-cost, efficient and basic multimedia support in small devices remains high. Simple displays and audio alerts enhance the user interfaces of smart devices without significantly increasing cost or complexity.

Developers must be able to optimize code and manage limited resources. This may include:

- Using external storage for large media files.

- Implementing efficient coding practices to maximize the use of available memory and processing power.

While multimedia is in demand in embedded systems, it is usually limited to simple, low-resolution graphics and basic audio. Complex multimedia tasks usually require more powerful hardware.

The ATMega328 remains a relevant and practical choice for microcontroller systems that require basic multimedia support. Its low cost, wide availability, and strong community support make it a good choice for educational applications, consumer electronics, and simple IoT devices. However, developers should be aware of its limitations and apply creative solutions to achieve the desired functionality.

1 MULTIMEDIA SUPPORT SYSTEMS

1.1 The concept of video codecs

A video codec (from the English Video Codec) is software or hardware designed for encoding and decoding video data. A codec determines how a video signal will be compressed during recording (encoding) and decoded during playback or processing. This approach allows you to reduce the volume of video files and save bandwidth when transmitting video data over the network.

Let's consider the main aspects of video codecs.

Encoding. The process of compressing video data before it is stored or transmitted. The codec analyzes the video signal and uses various compression techniques to reduce the file size.

Decoding. The process of restoring a video signal from an encoded format during video playback or processing. The decoder decodes the compressed file and reproduces the original video signal.

Lossy and lossless compression. Many video codecs use lossy compression, which means losing some quality to achieve a high degree of compression. There are also lossless video codecs that store the video signal without losing quality, but the file can be larger.

Bit rate (Bitrate). This is the number of bits used to represent a unit of video time. Determines the quality and size of the video file. A higher bit rate may improve quality, but will increase the file size.

Video formats. Video codecs can support different video formats such as H.264, H.265 (HEVC), VP9, AV1 and others. Each format has its advantages and disadvantages, including compression quality, support for different devices, and bandwidth.

Support for hardware acceleration. Some video codecs can use hardware acceleration, which allows you to reduce the load on the processor and improve performance during video playback.

The use of video codecs is necessary in many areas, such as video recording, streaming services, video conferencing, multimedia devices and gaming systems. The choice of a specific video codec depends on specific requirements and conditions of use.

1.2 Relevance of the use of video codecs

Let's consider the relevance of the use of video codecs in the modern world in view of a number of key factors.

With the popularity of streaming platforms such as Netflix, YouTube, Hulu and others, video codecs play an important role in ensuring the efficient transmission of video data over the network.

The growth of remote forms of work and communications makes it important to use efficient video codecs for real-time video signal transmission.

Video codecs are used to compress video streams in video games, as well as to stream video games over the network.

Video codecs built into media players and TVs allow you to play videos of various formats without the need for internal conversions.

With the development of digital technologies and the growth of video content, it is important to use efficient video codecs to save storage space and ensure fast access to video data.

Due to the popularity of video recording and video viewing on mobile devices, it is important to have efficient video codecs that allow you to provide quality video streaming with limited resources.

Video codecs are used in video production and editing to store and process large volumes of video material.

Video codecs play a key role in the transmission of large volumes of video data for virtual and augmented realities, where high quality and smoothness are important.

The relevance of video codecs is increasing because they affect the quality of video content, the efficiency of transmission over networks, the size of files, and the overall user experience of viewing and using custom-processed video.

1.3 Codingand video decoding

Encoding (Encoding) and Decoding (Decoding) of video codecs are processes that determine how video information is encoded during file creation (encoding) and how it is recovered for video playback or processing (decoding).

Video coding (or compression) consists in transforming a large amount of video data into a smaller volume.

This is done by using various specialized compression techniques, such as removing redundant details, using prediction, quantization, etc.

Reducing the volume of the video file, ensuring the efficiency of transmission over the network and ensuring optimal use of information storage space.

Codecs used for encoding include H.264, H.265 (HEVC), VP9, AV1, and others.

Video decoding is the process of restoring the original video signal from the encoded format during playback or processing.

The decoder decodes the compressed file, reproducing the original video signal.

Designed to play or process video in an understandable and accessible format.

Decoders use the same or compatible video codecs that were used for encoding.

Successful coding with a video codec provides high compression, while maintaining video quality at an acceptable level. Video decoding involves decoding the compressed data to reproduce the original video signal for viewing or processing.

Compatibility between codecs is important, allowing for efficient interoperability between different devices and applications for encoding and decoding custom video.

Many modern video codecs, such as H.264 and H.265, have become standards in many applications, allowing for high compatibility and efficiency.

1.4 Relevanceuse of sound codecs

Thus, let's consider the application areas of video codecs in the Internet of Things (IoT), as IoT covers a wide range of applications, including surveillance, medical devices, home authentication systems, transportation systems, and others.

Video codecs are important for transmitting and storing large amounts of video data generated by video surveillance in IoT security systems.

The use of video codecs allows you to transmit video from medical devices for remote consultations, training and monitoring of the condition and behavior of patients.

Video codecs can be used to stream video to a variety of devices in the home environment, including smart TVs, video projectors, and game consoles.

Video codecs are important for video monitoring and interaction with transport systems, such as surveillance cameras on roads or in public transport.

Video codecs allow the transmission and playback of video content on interactive devices and gaming consoles as an IoT entertainment element.

Video codecs are used to transmit video data from home security cameras for remote monitoring and security.

Given the limited resources in IoT devices, it is important to have efficient video codecs that allow video compression and transmission with minimal energy consumption.

Video codecs are used for video monitoring and analytics systems in smart cities to improve security and management efficiency.

Given the variety of applications in the Internet of Things, it is important to have optimal video codecs that allow for high quality video transmission with limited resources and network conditions.

1.5 The process of data compression in audio codecs

The process of data compression in video codecs includes a number of techniques and methods for reducing the volume of a video file without significant loss of image

quality. This process can be lossy or lossless, depending on whether quality is lost during compression.

The main stages of data compression in video codecs include the aspects described below.

Removal of redundant details and repetitive patterns in the spatial dimension. Techniques such as deduplication, quantization, and filtering are applied.

Used to reduce temporal redundancy between frames. Highlighting interframe dependencies and using forecasting to predict future frames.

Application of various algorithms for encoding data with loss of information. One of the most common methods is the use of DCT (discrete cosine transform), which is used, for example, in the JPEG and H.264 standards.

The quantization process reduces the precision of the value representation by discarding the less important bits. This is a key step in lossy compression.

The use of different types of frames, such as key (I-frames), expected (P-frames) and reverse (B-frames), to further reduce the amount of data.

Determining and taking into account the movement of objects in the video signal for effective encoding of changes in frames.

Application of algorithms that predict the content of frames, as well as interpolation methods to reduce the amount of information that needs to be stored.

These methods contribute to the creation of efficient video codecs that provide a balance between the amount of data and the quality of the video image.

Lossy compression is a common approach in video encoding, as it allows for significant reductions in file size and data transfer, which is important for streaming and storing large.

1.6 Types of codecs

There are many types of video codecs, each of which has its own characteristics, advantages and disadvantages. Here are some of the most common video codecs.

H.264 (AVC) is one of the most popular video encoding standards used for videos on YouTube, Vimeo, video conferencing and other platforms. It provides good quality at a low bit rate.

The successor standard to H.264 and provides better compression while maintaining high video quality. HEVC is well suited for 4K and higher resolutions.

Developed by Google, VP9 is an open standard with excellent compression and high resolution support. Used in YouTube videos in particular.

Another open standard developed by the Alliance for Open Media group. AV1 offers a high level of compression while maintaining quality and is designed for use in a variety of scenarios, including streaming and media players.

A standard used to compress video and audio for DVD, broadcast television, and other video formats.

Used to compress video for web and mobile devices. MPEG-4 has many varieties of codecs, such as DivX and Xvid.

Known for its high compression and use in high quality video files.

Xvidis a popular freely distributed video compression codec.

AppleProResused mainly in professional video editing and production of video content on Apple products.

These are just a few examples, there are many other video codecs that are used in various fields, including broadcasting, streaming, video games, video production, and more.

The choice of a specific codec may depend on the needs of the user, the specifics of video projects and the platform of use.

1.7 Known software system libraries for encoding and decoding of video codecs

There are several well-known software system libraries that provide video codec encoding and decoding functionality.

Some of the most popular libraries in this area include the system software described below.

FFmpeg is one of the most powerful and widely used libraries for video and audio processing. It provides tools for encoding, decoding, trimming, converting and other manipulations with video and audio data.

OpenCV (Open Source Computer Vision Library) is a computer vision library, but it also provides functionality for working with video. OpenCV has capabilities to encode and decode video using various codecs.

libavcodec is a specific FFmpeg component responsible for audio and video encoding and decoding. This library has a large number of codecs including H.264, H.265, VP9 and others.

x264 and x265 are libraries for encoding video in the H.264 and H.265 standards (respectively). They are widely used to create videos with high compression and quality.

VideoToolbox is a framework that provides hardware support for video encoding and decoding on Apple devices such as Mac and iPhone.

MediaCodec is an API for handling multimedia data on the Android platform. It allows you to use hardware support for video encoding and decoding on Android devices.

VLC (VideoLAN Client) uses the libavcodec library to implement codecs and multimedia processing.

These libraries provide developers with tools to work with different video codecs and implement different operations on video data. The choice of library may depend on the specific needs of the project, platform and programming language.

1.8 The relevance of video broadcasting using microcontrollers for Internet of Things (IoT) systems

The use of video streaming using microcontrollers for Internet of Things (IoT) systems can be relevant in many scenarios where it is important to remotely monitor or observe events, objects or spaces.

Video streaming can be used to create video surveillance systems for remote monitoring and security of objects or areas, such as homes, offices or infrastructure facilities.

Video transmission can be used for remote monitoring of patients, conducting telemedicine or even conducting virtual consultations.

In production environments, video streaming can be used for remote monitoring of work processes, equipment status or even for staff training.

With the help of video transmission, you can remotely monitor and observe home processes, for example, security or animal care.

Video streaming can be used to monitor traffic, parking, or even to create virtual perception systems for autonomous vehicles.

Video streaming can be used in consumer electronic devices such as home video cameras for remote monitoring from mobile devices.

In the fields of virtual reality and augmented reality, video streaming can be used to create immersive experiences and interactions.

Video streaming can be important for public safety systems, such as video surveillance on streets or public transport.

The relevance of using video transmission with microcontrollers in IoT systems lies in the possibility of creating effective and intelligent solutions for remote monitoring and management of various objects or processes in real time.

1.9 Control of video codecs based on the ATMega328 microcontroller

The ATMega328 microcontroller used in the Arduino Uno platform has the resources and memory capacity that makes it suitable for complex video codec operations and real-time video playback or encoding.

However, you can use some video codec operations and experiment with video compression and playback using this microcontroller.

Let's consider the general approach to work with ATMega328-based video codecs.

Since the ATMega328 has limited resources, you should choose video codecs that support low resolution and can be implemented on this microcontroller. For example, using QVGA (320x240) or even lower resolution.

You can use ready-made libraries to interact with video codecs supported by ATMega328. For example, some libraries can work with JPEG codecs to compress and decompress images.

You should pay particular attention to optimizing your code to minimize the use of microcontroller resources.

Reducing the amount of memory and optimizing the execution speed will be important factors.

Given the limited capabilities of the ATMega328, you may also consider using external modules or auxiliary microcontrollers to handle the video data, leaving the ATMega328 responsible for control and other functions.

It is necessary to choose video codecs and settings that allow working with a low bitrate to reduce the amount of video data.

It is important to note that implementing video codecs on microcontrollers such as the ATMega328 is a challenge due to their limited resources.

1.10 Conclusion

The chapter delves into the intricate and essential aspects of sound codecs, highlighting their significance, functionality, and application in modern digital systems. The exploration begins with the fundamental concept of sound codecs, providing a foundational understanding of their role in encoding and decoding audio signals. This lays the groundwork for appreciating the relevance of sound codecs in various applications, especially in data compression.

Sound codecs are crucial for converting analog audio signals into digital format and vice versa. They facilitate the efficient transmission and storage of audio data by

compressing it without significant loss of quality. The chapter underscores the importance of understanding sound codecs as the building blocks for modern audio technology.

The chapter provides a thorough examination of sound codecs, their significance, and their applications, particularly in the realm of IoT. The discussions encompass the technical aspects of audio data compression, the types of codecs available, and the practical implementation of audio functionalities in modern digital systems using microcontrollers like the ESP8266.

This comprehensive exploration not only enhances the understanding of sound codecs but also underscores their vital role in the evolution of digital audio technology and the Internet of Things.

2 DESCRIPION OF THE COMPONENTS FOR THE MULTIMEDIA SUPPORT SYSTEM SOFTWARE FOR MICROCONTROLLER SYSTEMS BASED ON ATMEGA328

2.1 ATmega 328

The ATmega series of microcontrollers are energy-efficient, CMOS-based 8-bit devices constructed on the advanced AVR® RISC architecture.

These microcontrollers are capable of executing instructions in just one clock cycle, which enables them to deliver nearly one MIPS per megahertz.

This high efficiency allows system designers to optimize both power consumption and processing speed.

2.2 Pin configurations

The ATmega328 includes pins as illustrated in Figure 2.1. The pinout details for the ATmega328 are provided in Table 2.1.

Table 2.1 – Pinout of ATmega328

	1	2	3	4	5	6
A	PD2	PD1	PC6	PC4	PC2	PC1
B	PD3	PD4	PD0	PC5	PC3	PC0
C	GND	GND			ADC7	GND
D	VDD	VDD			AREF	ADC6
E	PB6	PD6	PB0	PB2	AVDD	PB5
F	PB7	PD5	PD7	PB1	PB3	PB4

VCC is the supply voltage for digital circuits.

GND represents the ground connection.

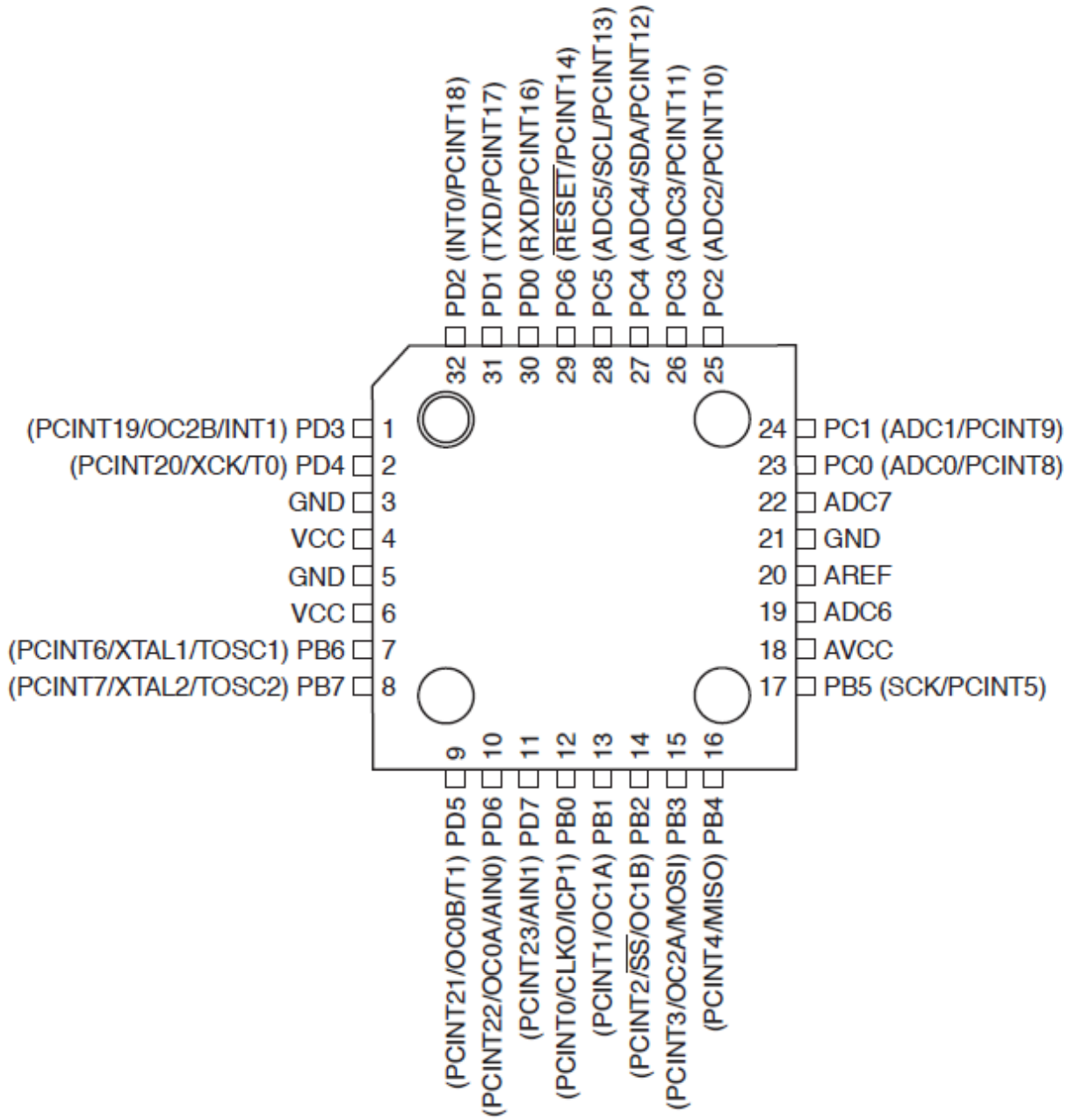


Figure 2.1 - Pin configurations

Port B is an 8-bit bidirectional I/O that includes resistors, which can be individually configured for a bit.

The output buffers for Port B are designed with symmetrical drive characteristics, providing both strong sinking and sourcing capabilities.

When functioning as inputs, if the pull-up resistors are enabled and the pins are externally pulled low, they will source current.

During a reset condition, the Port B pins enter a high-impedance state (tristate), regardless of the clock's activity.

Port C is a 7-bit bidirectional I/O port with resistors, which can be enabled individually for a pin.

The output buffers for pins PC5 through PC0 are designed with symmetrical drive characteristics, providing strong sinking and sourcing capabilities.

As inputs, if the pull-up resistors are enabled and the pins are externally pulled low, they will source current. In the event of a reset, the Port C pins transition to a high-impedance state (tristate), even if the clock is inactive.

Port D is an 8-bit bidirectional I/O port, also featuring internal pull-up resistors that can be enabled on a per-bit basis.

The output buffers for Port D are characterized by symmetrical drive capabilities, providing robust sinking and sourcing capacities.

When used as inputs, Port D pins source current if the pull-up resistors are activated and the pins are externally pulled low.

During a reset condition, the Port D pins enter a high-impedance state (tristate), even if the clock is not running.

AVCC serves as the supply voltage pin for the A/D Converter, as well as for pins PC3:0 and ADC7:6.

It is advised to connect AVCC externally to VCC, whether or not the ADC is used. If the ADC is employed, AVCC should be connected to VCC through a low-pass filter for optimal performance.

The Figure 2.2 illustrates the Block Diagram of the ATmega328.

The AVR core combines an extensive instructions with thirty two general-purpose working registers.

Each of these 32 registers is directly connected to ALU, allowing for simultaneous access to registers within a single instruction cycle.

This design achieves higher code efficiency.

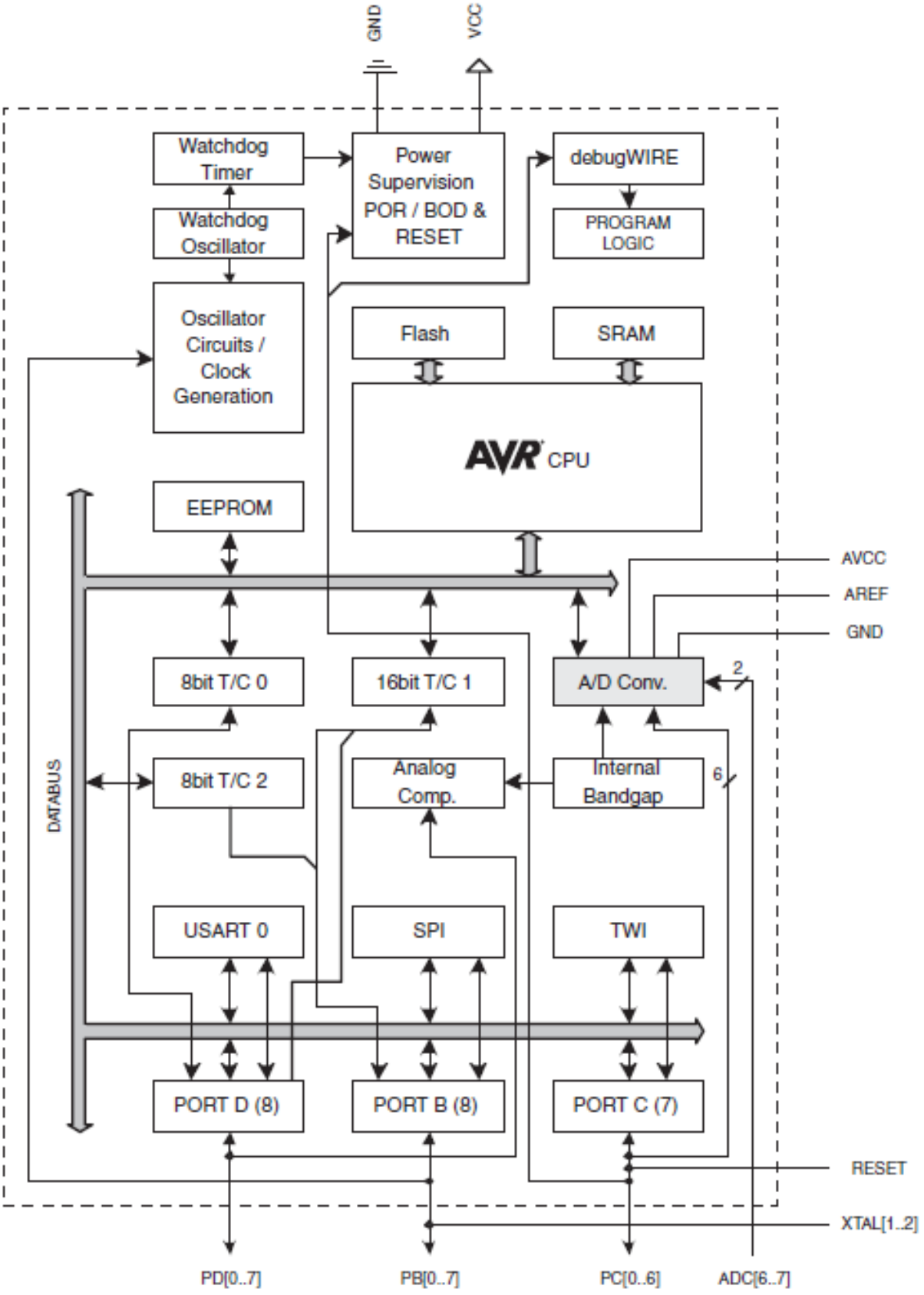


Figure 2.2 - Block Diagram of the ATmega328

The ATmega microcontrollers feature a comprehensive range of capabilities.

These include In-System Programmable Flash memory ranging from 4K to 8K bytes with Read-While-Write functionality, EEPROM capacities from 256 bytes to 1K bytes, and SRAM sizes from 512 bytes to 2K bytes. The devices provide 23 general-purpose I/O lines and 32 working registers.

In Power-down mode, register contents are retained, but the Oscillator is halted, disabling all other functions until an interrupt or hardware reset occurs.

Power-save mode allows the asynchronous timer to remain active, enabling a maintained timer base while other device components are in sleep mode.

The QTouch Suite toolchain is designed to facilitate the exploration, development, and debugging of touch applications.

The device is fabricated utilizing Microchip's advanced high-density non-volatile memory technology.

Its On-chip ISP Flash enables reprogramming of program memory In-System via an SPI serial interface, through a conventional non-volatile memory programmer.

Meanwhile, software in the Boot Flash section continues to execute while updates are made to the Application Flash section, ensuring genuine Read-While-Write functionality.

To facilitate touch-sensitive interface implementation on most AVR microcontrollers, the QTouch Library provides a simple solution.

It supports both QTouch and QMatrix™ acquisition techniques, allowing seamless integration of touch sensing functionality.

To incorporate touch capabilities into an application, the Library for the AVR microcontroller is used.

This involves a straightforward API set to define touch channels and sensors, and subsequently calling the touch sensing APIs to retrieve channel data and determine the status of the touch sensors.

The AVR architecture employs a architecture, which separates program and data memories and buses, optimizing performance and parallelism. Instructions stored are

executed with a single-level pipelining system. As one instruction executes, the next one is pre-fetched from program memory, ensuring continuous instruction flow.

The Register File, a key feature for rapid access, includes 32 general-purpose 8-bit working registers.

This setup supports single-cycle operations for the Arithmetic Logic Unit (ALU). During typical ALU operations, 2 operands are fetched from File, the operation is executed.

Among registers, six can function as three 16-bit registers for addressing.

The ALU also supports single register operations, and updates the Status Register to reflect the result of each operation.

Program control is managed through instructions, which allow access to address.

I/O Memory is able to access locations beyond the Register File (0x20 - 0x5F).

The ATmega also includes Extended I/O space from 0x60 to 0xFF within the SRAM, which can be accessed using the ST/STS/STD and LD/LDS/LDD instructions.

The ALU is a highly efficient component of the AVR, directly interacting with all 32 general-purpose working registers.

The ALU's operations cover implementations include a powerful multiplier for handling signed/unsigned multiplication and fractional formats.

The Status Register stores information and is able to affect program flow by enabling conditional execution. It is updated after every ALU operation.

Consequently, there is often no need to use specific compare instructions, leading to faster and more efficient code. SREG is depicted in Figure 2.3.

Bit 7 – I: Global Interrupt Enable. To activate interrupts, I-bit has to be set. Individual interrupt activation is then managed through separate control registers.

Bit 6 – T: Bit Copy Storage.

Bit 5 – H: Half Carry Flag.

Bit 4 – S: Sign Bit (S = N V).

Bit 3 – V: Two’s Complement Overflow Flag. The V-bit (Two’s Complement Overflow Flag) is used to detect overflow in two’s complement arithmetic operations.

Bit 2 – N: Negative Flag. The N-bit (Negative Flag) is set to indicate a negative result in an arithmetic or logical operation.

Bit 1 – Z: Zero Flag.

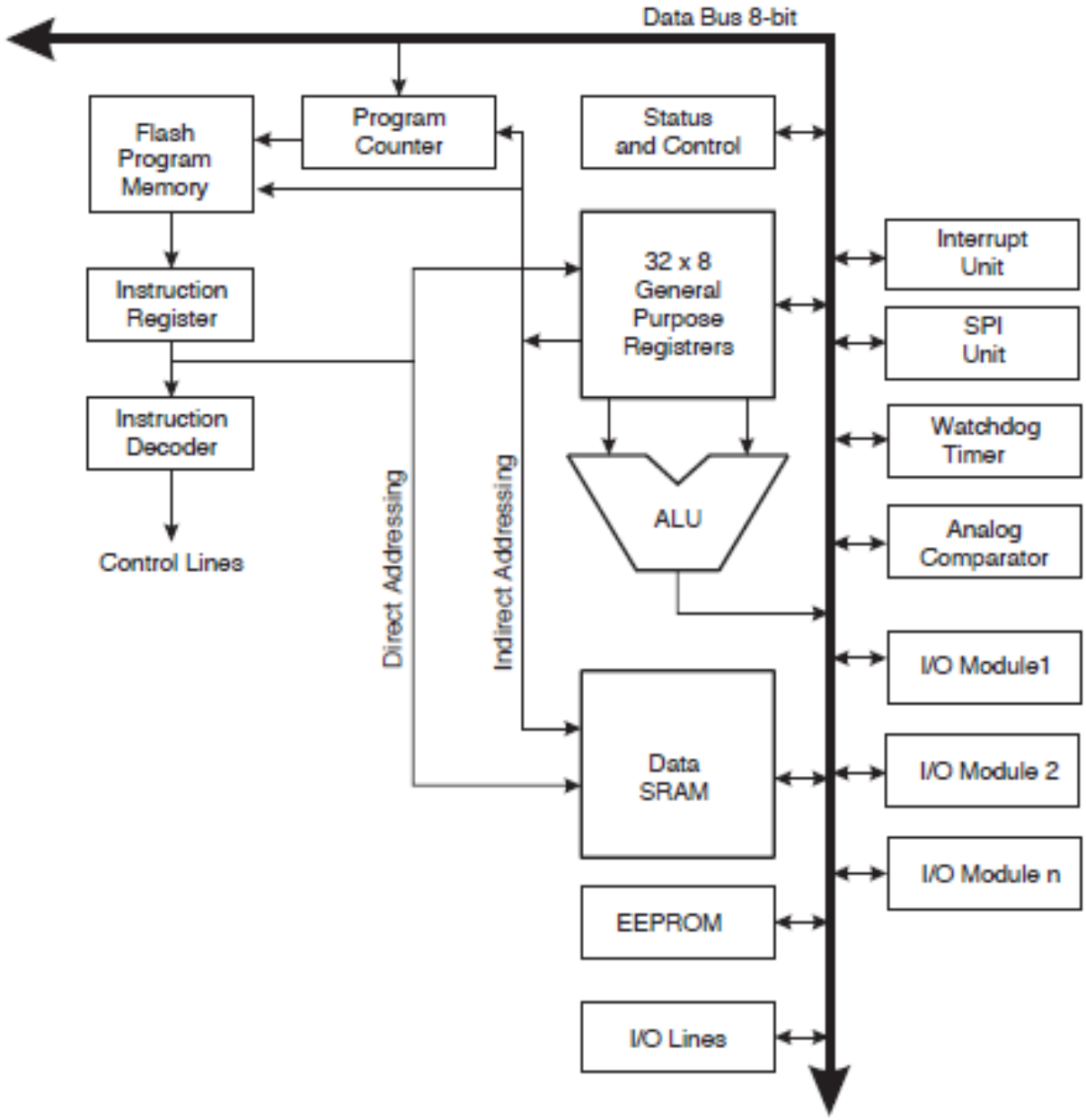


Figure 2.3 – Block Diagram of the AVR Architecture

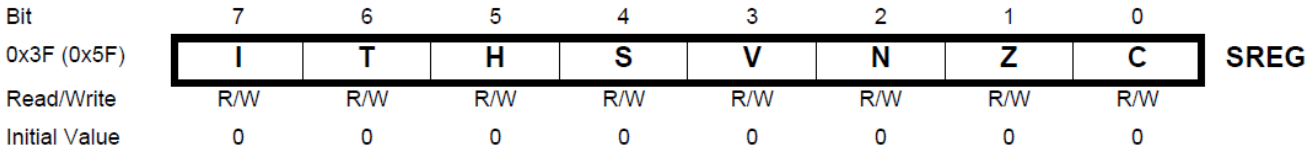


Figure 2.4 – AVR Status Register

Bit 0 – C: Carry Flag. The C-bit (Carry Flag) signals a carry or borrow in arithmetic or logical operations.

2.3 GPRF

GPRF is designed to support the AVR RISC instructions.

The arrangement of the 32 registers is shown in Figure 2.4.

2.5 X-, Y-, and Z-registers

R31 serves additional purposes beyond their general utility, which are outlined in detail in Figure 2.5.

2.6 Stack Pointer

The Stack's primary function is to hold temporary data, such as local variables and return addresses during subroutine calls and interrupts.

The initial value is set to address of SRAM, and is outlined in Table 2.1.

The AVR Stack Pointer is designed as two separate 8-bit registers located within the I/O space.

The actual number of bits utilized depends on the specific implementation.

It’s important to mention that, in certain versions of the AVR architecture where the data space is limited, only the SPL register is required, and the SPH register may not be included.

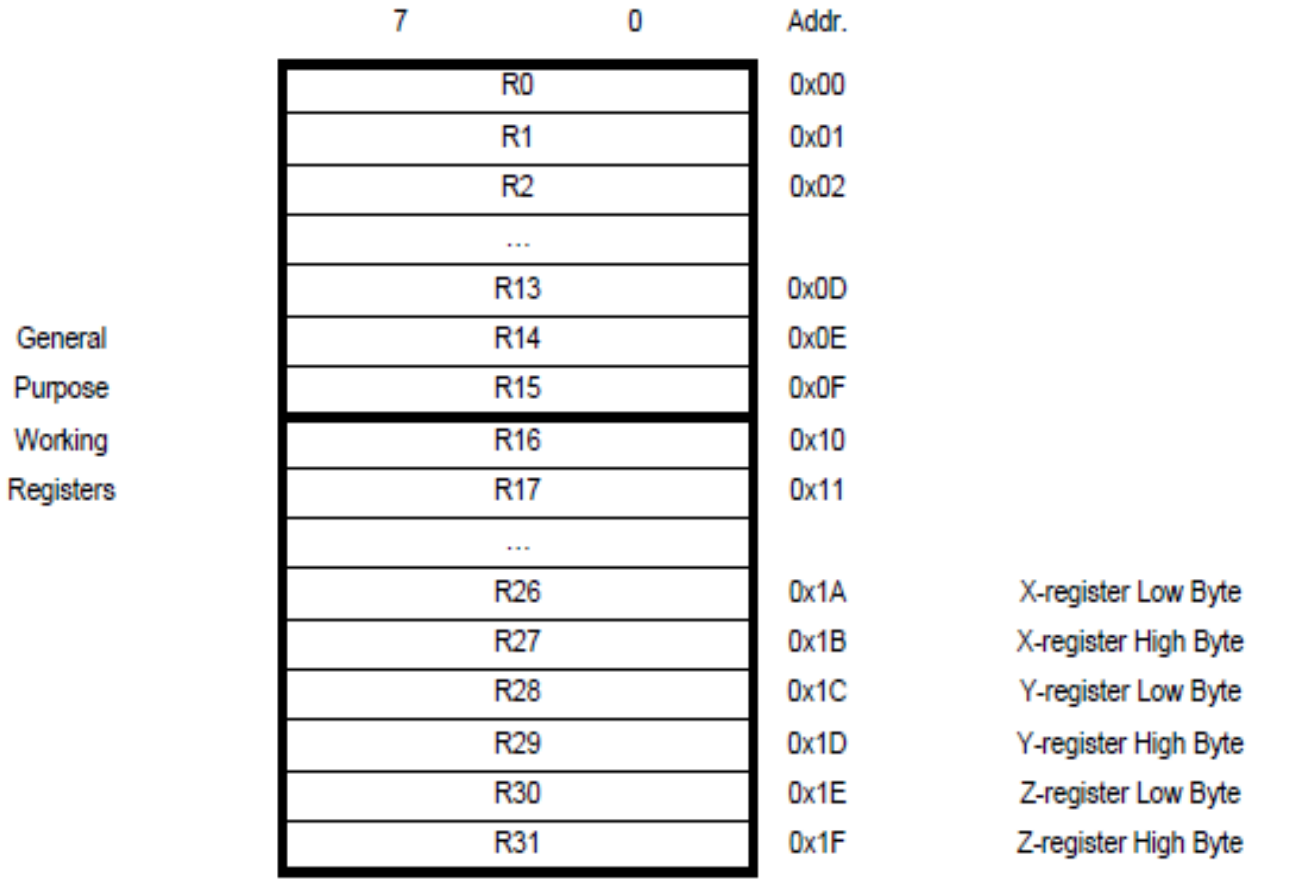


Figure 2.4 - The configuration of the 32 general-purpose working registers within the CPU

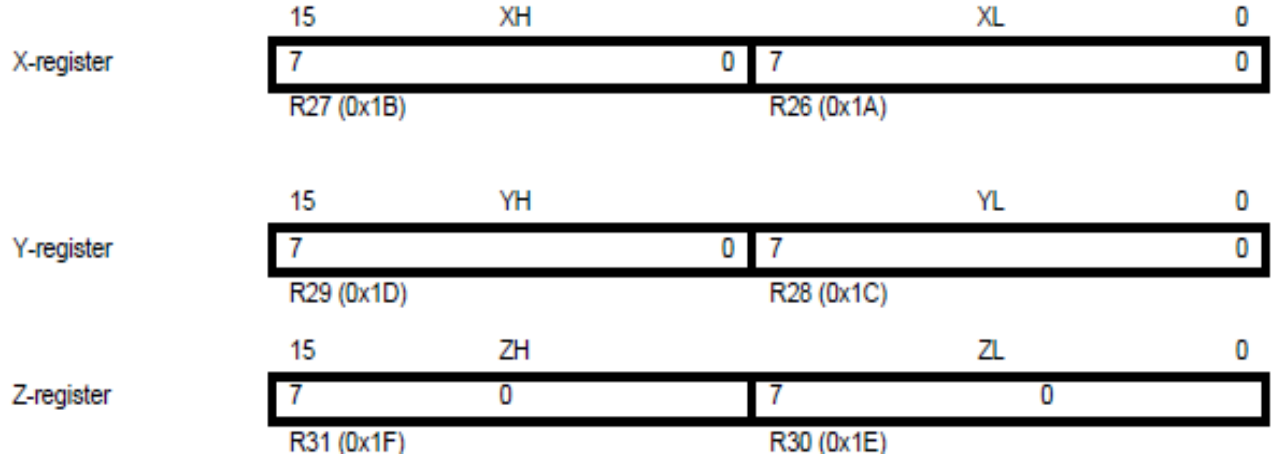


Figure 2.5 - Registers

Table 2.1 – Stack pointer description

Instruction	Stack pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

2.7 Instruction Execution Timing

There's no internal clock division utilized. As depicted in Figure 2.6, the architecture enables the parallel instruction fetches and executions.

This forms the fundamental pipelining concept aimed at achieving up to 1 Million Instructions Per Second (MIPS) per Megahertz (MHz).

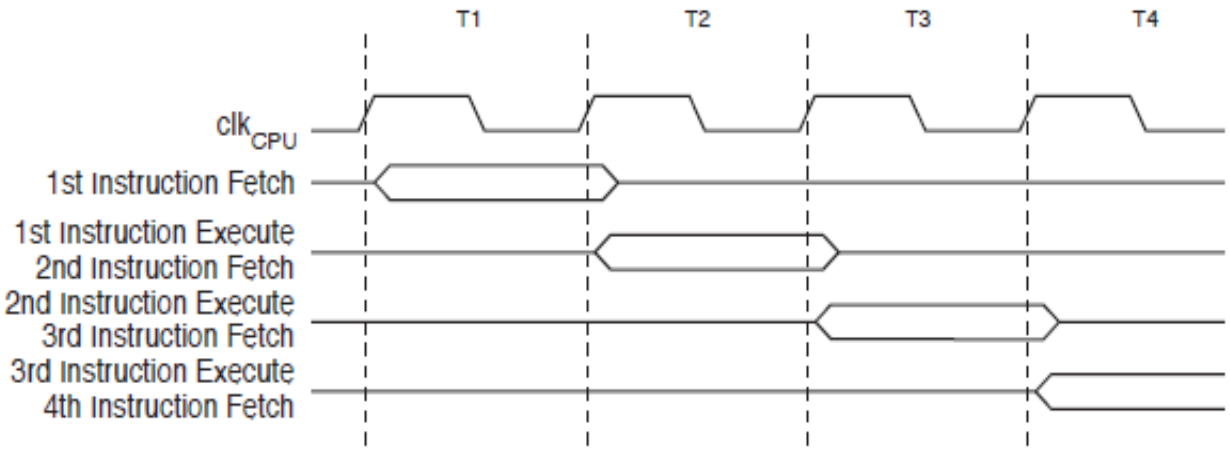


Figure 2.6 - Instruction and executions

Figure 2.7 illustrates the internal timing concept

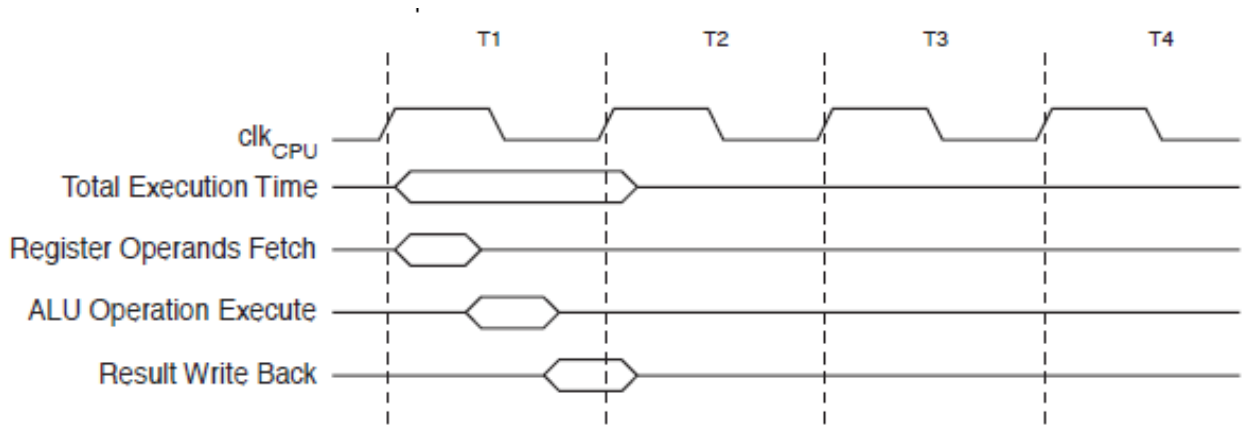


Figure 2.7 - Single Cycle ALU Operation

2.8 Reset

AVR includes various interrupt sources, each linked to a specific program vector within the memory.

These interrupts, along with the Reset Vector, have individual enable bits.

By default, the lowest addresses in the memory are allocated to the Reset.

The highest priority is given to RESET, followed by INT0, which is 0.

The user software can allow nested interrupts by setting the I-bit back to logic one, permitting all enabled interrupts to preempt the current interrupt routine.

There are essentially two categories of interrupts in AVR. The 1st class is implemented by Interrupt Flag.

The second category of interrupts is continuously. These interrupts may not have specific Interrupt Flags.

When exiting an interrupt, the AVR resumes execution of the main program and completes one additional instruction before addressing any pending interrupts.

When the CLI instruction is executed to disable interrupts, they are immediately turned off. This can be particularly useful for preventing interruptions during operations such as the timed EEPROM write sequence.

2.9 Interrupt Response Time

The minimum response time for all active AVR interrupts is four clock cycles. During this time, the Program Counter is pushed onto the Stack, and the interrupt vector address is loaded for the interrupt handling routine. Typically, this routine involves a three-clock cycle jump to the interrupt service routine (ISR).

If an interrupt occurs while a multi-cycle instruction is being executed, the instruction completes before the interrupt is processed. This ensures the integrity of the instruction being executed.

When the AVR microcontroller is in sleep mode and an interrupt is triggered, the interrupt execution response time is extended by four additional clock cycles. This delay is in addition to the wake-up time required from the specific sleep mode chosen.

Exiting from an interrupt handling routine also requires four clock cycles. This involves retrieving the Program Counter from the Stack, incrementing the Stack Pointer, and setting the Interrupt Enable bit (I-bit) in the Status Register (SREG).

These details underscore the precise timing considerations and operations involved in managing interrupts effectively on AVR microcontrollers, ensuring proper handling of events without compromising the execution of critical instructions or system sleep functionality.

2.10 AVR Memories

The AVR architecture encompasses two primary memory spaces: Data Memory and Program Memory. Additionally, the ATmega microcontroller features an EEPROM Memory dedicated to data storage.

Figure 2.10 depicts the structure of SRAM Memory within the ATmega microcontroller.

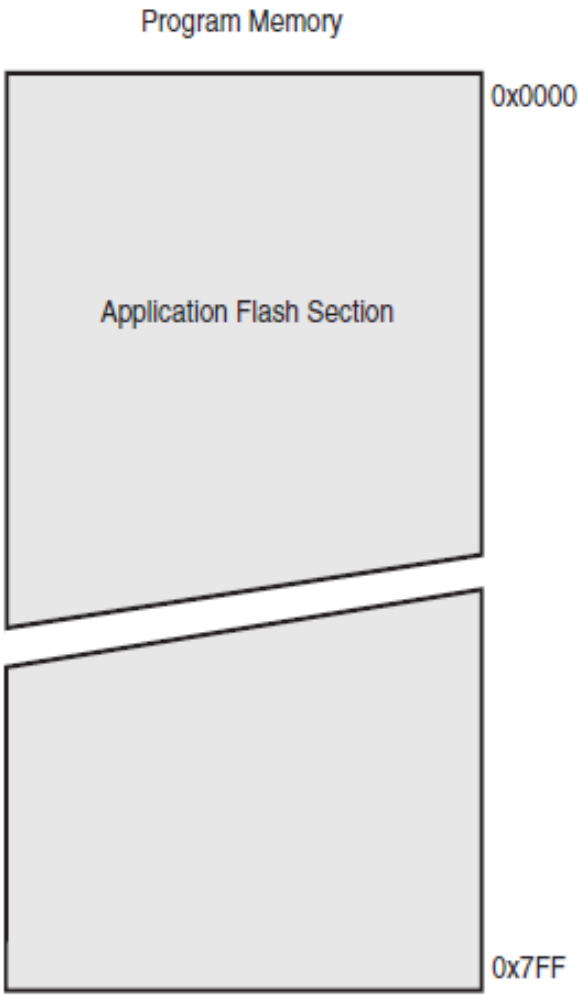


Figure 2.8 - Program Memory Map ATmega 48A/48PA

The ATmega microcontroller's Opcode for the IN and OUT instructions is limited to 64 locations. Due to the microcontroller's complexity and extensive peripheral units, only the ST/STS/STD and LD/LDS/LDD instructions are suitable for accessing the Extended I/O space, which spans from 0x60 to 0xFF in SRAM.

The data memory is divided into several segments:

- Register File has initial 32 locations are reserved for the Register File.
- Standard I/O Memory consists of 64 locations.
- Extended I/O Memory occupies 160 locations.
- Internal Data SRAM: The largest segment, containing 512/1024/2048 locations depending on the specific ATmega variant mentioned (768/1280/1280/2303 locations likely refer to different ATmega models or configurations).

This segmentation underscores the diverse memory requirements and the specific addressing schemes necessary to effectively utilize the ATmega microcontroller's capabilities, accommodating its extensive peripheral functionalities and ensuring efficient data access across various memory spaces.

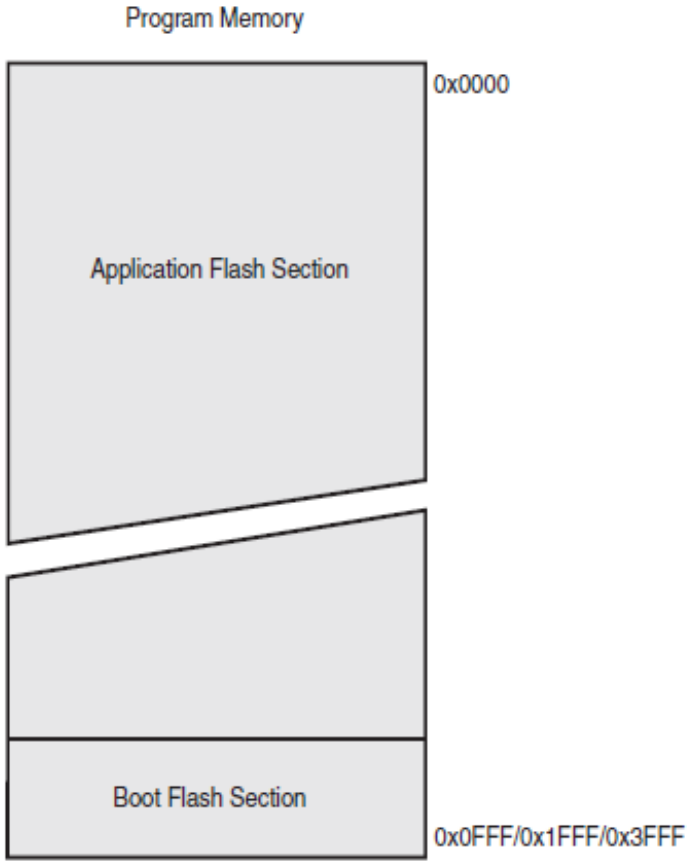


Figure 2.9 - Program Memory Map ATmega328

Addressing Modes:

- Direct Addressing allows direct access to any location within the data space.

- Indirect Addressing uses the content of a register (X, Y, or Z) as a pointer to access data indirectly.

- Indirect with Displacement extends the indirect addressing mode by allowing an offset (displacement) from the base address provided by the Y- or Z-register, spanning 63 locations.

- Indirect with Pre-decrement adjusts the address register (X, Y, or Z) before accessing the data, useful for stack operations or accessing arrays in reverse order.

- Indirect with Post-increment adjusts the address register (X, Y, or Z) after accessing the data, facilitating sequential access to arrays or buffers.

Pointer Registers R26 to R31 within the Register File serve as indirect addressing pointer registers (X, Y, and Z registers), crucial for manipulating memory addresses in indirect addressing modes.

These addressing modes enable flexible and efficient data manipulation within the ATmega microcontroller's data memory, accommodating diverse programming needs from direct access to sequential and indexed access patterns.

The versatility of the ATmega microcontroller's addressing capabilities, essential for managing data efficiently in embedded systems and other applications requiring precise memory access.All 32 general-purpose working registers, 64 I/O Registers, 160 Extended I/O Registers, and the 512/1024/1024/2048 bytes of internal data SRAM within the ATmega microcontroller are accessible through these addressing modes.

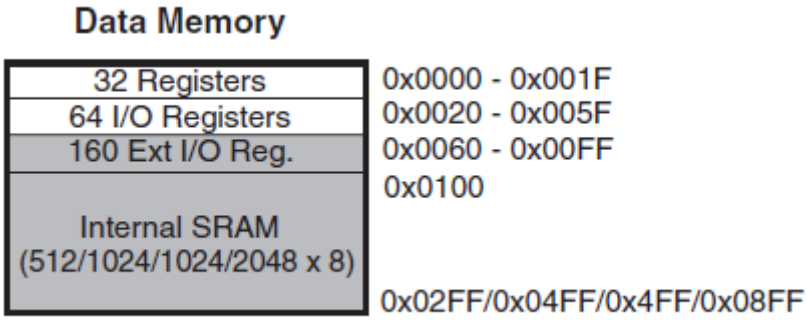


Figure 2.10 - Data Memory Map

Internal data of SRAM is detailed in Figure 2.11.

Accessing the internal data SRAM requires two clock cycles of the CPU.

The ATmega microcontroller includes 256/512/512/1Kbytes (depending on the specific model) of EEPROM memory, organized separately from the SRAM. EEPROM allows individual bytes to be read and written.

EEPROM provides a durability of at least 100,000 write/erase cycles.

Communication between the CPU and EEPROM utilizes specialized registers: EEPROM Address Registers, EEPROM Data Register, and EEPROM Control Register, all located in the I/O space.

A self-timing function enables user software to determine the appropriate timing for EEPROM write operations.

Due to potential voltage fluctuations during power-up or power-down phases, precautions are necessary to avoid inadvertent EEPROM writes.

Specific instructions and procedures must be followed to ensure reliable EEPROM operations, especially concerning the EEPROM Control Register.

During EEPROM reads, the CPU pauses for four clock cycles before executing the next instruction.

During EEPROM writes, the CPU pauses for two clock cycles before proceeding with the next instruction.

ATmega microcontroller's robust EEPROM capabilities, its integration into the overall memory architecture, and the precautions required for reliable EEPROM operations in varying voltage conditions.

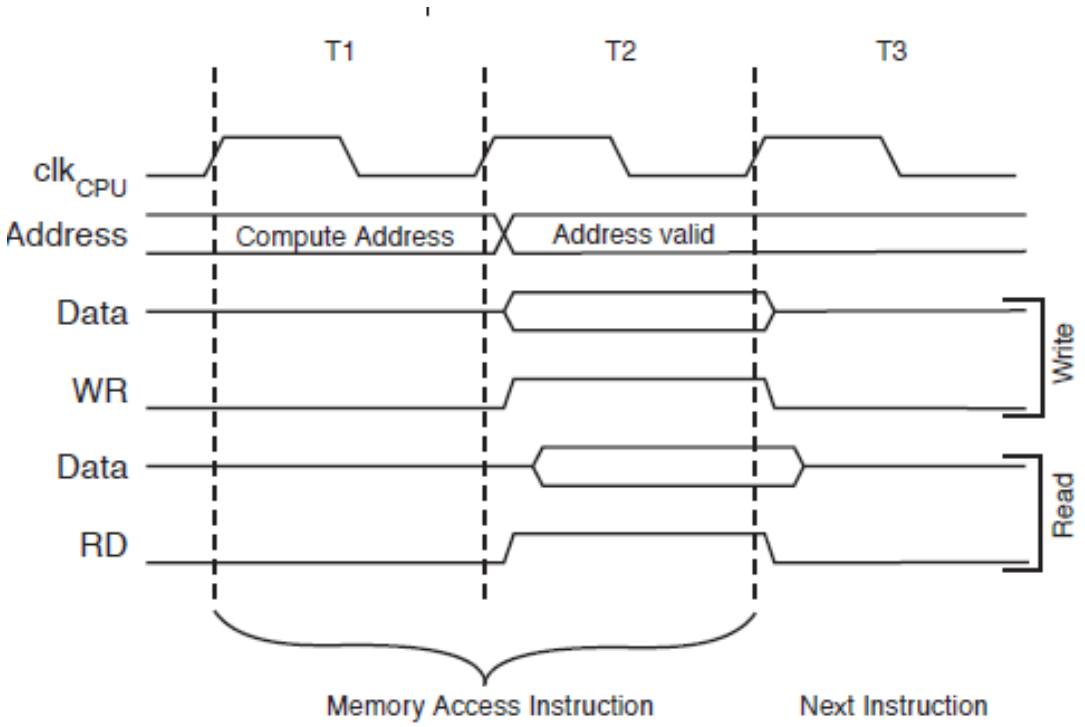


Figure 2.11 - On-chip Data SRAM Access Cycles

2.11 Preventing EEPROM Corruption

When VCC drops to low levels, there's a risk of EEPROM data corruption due to insufficient supply voltage for both the CPU and the EEPROM to function properly. These concerns mirror those encountered in board-level systems utilizing EEPROM, and similar design strategies should be employed. The text emphasizes strategies to prevent EEPROM data corruption in the ATmega microcontroller:

Proper functioning of EEPROM write sequences necessitates a minimum voltage threshold. If this threshold is not met, EEPROM data can become corrupted.

Insufficient supply voltage can cause the CPU to execute instructions erroneously, leading to potential EEPROM corruption.

To mitigate EEPROM data corruption risks, it is recommended to keep the AVR RESET active (low) during periods of inadequate power supply voltage.

This can be achieved by enabling the internal Brown-out Detector (BOD), which helps maintain stable operation within specified voltage ranges.

If the internal BOD's detection level does not align with operational requirements, an external low VCC reset protection circuit can be employed.

If a reset event occurs during an EEPROM write operation, the write operation will proceed and complete successfully only if the power supply voltage remains above the minimum required level.

2.12 Memory

The intricacies of I/O management in the ATmega microcontroller are:

1. I/O Space and Access:

- All I/Os and peripherals are located within the I/O space of the ATmega microcontroller.

- Instructions like LD/LDS/LDD and ST/STS/STD facilitate seamless data transfer between the 32 general-purpose working registers and the I/O space.

2. Direct Bit Manipulation:

- I/O Registers within the address range of 0x00 to 0x1F can be manipulated at the bit level using commands such as SBI (Set Bit in I/O Register) and CBI (Clear Bit in I/O Register).

- Specific bit values can be checked using SBIS (Skip if Bit in I/O Register is Set) and SBIC (Skip if Bit in I/O Register is Cleared) instructions.

3. Addressing and Instruction Set:

- IN and OUT instructions address I/O Registers within the range of 0x00 to 0x3F.

- When using LD and ST instructions to access I/O Registers as part of the data space, an offset of 0x20 must be added to these addresses.

4. Extended I/O Space:

- The Extended I/O space, spanning from 0x60 to 0xFF in SRAM, necessitates the use of ST/STS/STD and LD/LDS/LDD instructions due to the abundance of peripheral units exceeding the capacity of the IN and OUT instruction Opcode.

5. Operational Considerations:

- Reserved bits should be set to zero when accessed, and writing to reserved I/O memory addresses should be avoided to ensure compatibility with future devices.

- Certain Status Flags can be cleared by writing a logical one to them.

6. Instruction Set and Peripheral Control:

- Instructions like CBI and SBI operate exclusively on registers within the range of 0x00 to 0x1F, affecting only the specified bit.

- Detailed descriptions of control registers for I/O and peripherals are available in subsequent sections, providing further insights into their configuration and usage.

2.13 General Purpose I/O Registers, Clock Systems and their Distribution

The ATmega microcontroller features three General Purpose I/O Registers located in the address range of 0x00 to 0x1F.

These registers serve as flexible storage spaces for various data types, including global variables and Status Flags.

They support direct manipulation of individual bits using instructions such as SBI (Set Bit in I/O Register), CBI (Clear Bit in I/O Register), SBIS (Skip if Bit in I/O Register is Set), and SBIC (Skip if Bit in I/O Register is Cleared).

Figure 2.12 illustrates the primary clock systems within the AVR architecture and their distribution.

The architecture allows for selective activation of clocks, enabling modules that are not in use to be powered down to conserve energy.

Different sleep modes facilitate halting clocks to inactive modules, thereby reducing overall power consumption.

2.14 CPU Clock – clkCPU

The CPU clock drives critical components like the General Purpose Register File, Status Register, and data memory holding the Stack Pointer.

Halting the CPU clock suspends general operations and calculations of the AVR core, effectively pausing its execution.

The I/O clock is crucial for various I/O modules such as Timer/Counters, SPI, USART, and External Interrupts.

Specific functionalities like start condition detection in the USI module and TWI address recognition operate asynchronously or across different sleep modes, despite halting the clkI/O.

Level-triggered interrupts are employed to wake the microcontroller from Power-down mode.

It's essential to maintain the interrupt trigger level for a duration sufficient to complete the wake-up process and generate the interrupt.

The start-up time, determined by SUT and CKSEL Fuses, dictates how quickly the microcontroller can respond to external events after waking from sleep.

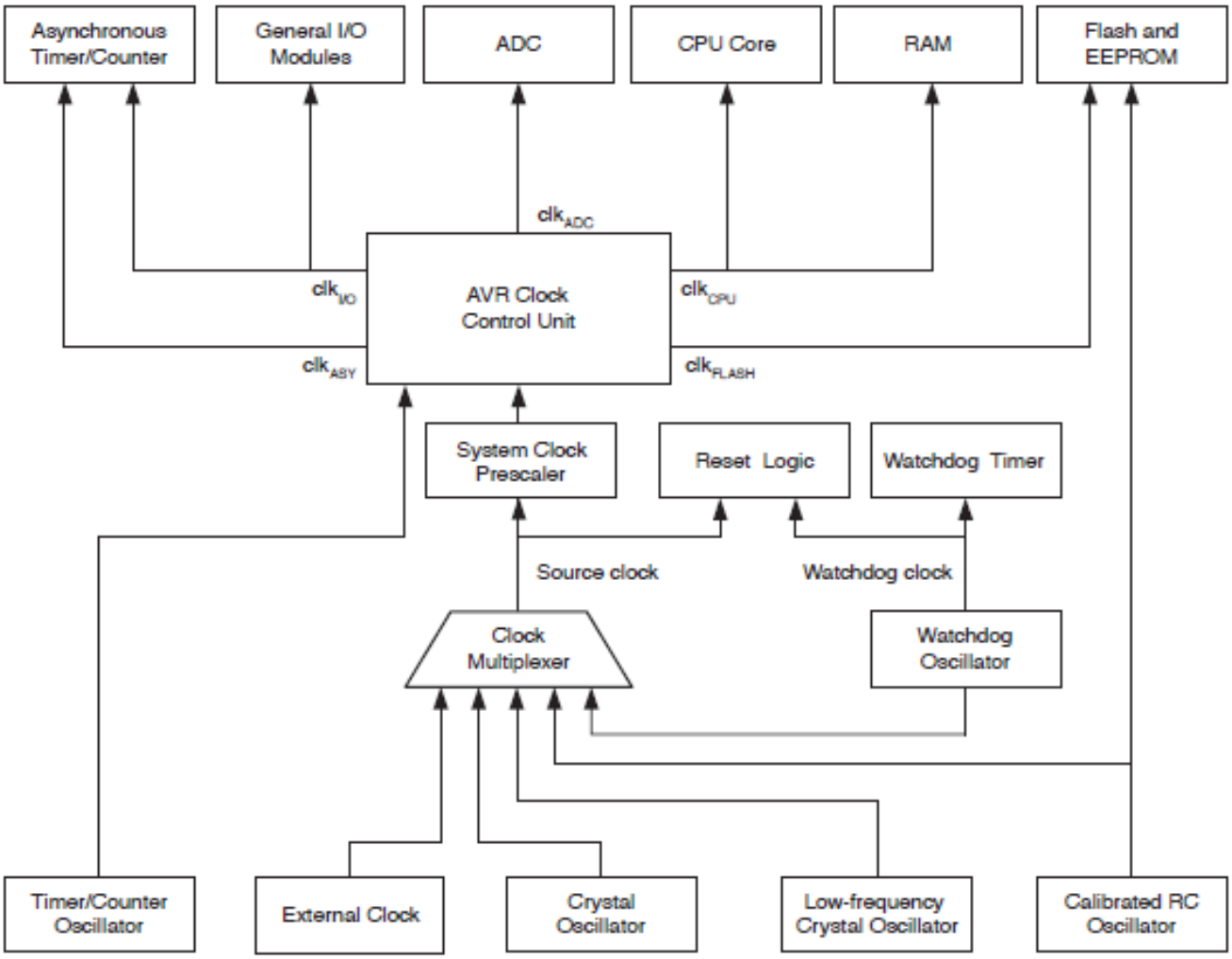


Figure 2.13 - Clock Distribution

2.15 Flash Clock – clkFLASH

The Flash clock synchronizes with the CPU clock and governs the operation of the Flash interface, crucial for program execution.

Asynchronous Timer Clock domain independently drives the Asynchronous Timer/Counter, supporting real-time counting even in sleep modes using an external clock or 32kHz crystal.

The ADC operates in its dedicated clock domain, allowing halting of CPU and I/O clocks to reduce digital noise, ensuring accurate conversion results.

Various clock sources are available and selectable via Flash Fuse bits, determining the clock fed into the AVR clock generator and distributed to modules.

Default configuration includes an 8.0MHz internal RC oscillator with CKDIV8 fuse, resulting in a 1.0MHz system clock. Start-up settings enable customization via programming interfaces.

Start-up involves ensuring adequate VCC for clock oscillation and a defined number of stable cycles for each clock source.

A timeout delay (tTOUT) from the Watchdog Oscillator, controlled by SUTx and CKSELx fuses, ensures VCC is sufficient before releasing the reset condition.

BOD circuitry further safeguards against inadequate VCC by delaying reset release until voltage stabilizes, enhancing system reliability.

2.16 Conclusion

The chapter is devoted to the analysis of fuctional possibilities of the ATmega microcontroller, that stands out for its robust clock systems and comprehensive features, designed to enhance functionality and performance across diverse applications.

Each aspect of its clock architecture, from the Flash clock governing the Flash interface to the Asynchronous Timer clock facilitating real-time counting even during sleep mode, demonstrates meticulous attention to detail and optimization for various operational scenarios.

The microcontroller's versatility extends to its clock source options, providing users with flexibility to fine-tune clock settings to meet specific application demands. Furthermore, the inclusion of built-in safeguards, such as the timeout delay and startup sequence, underscores its reliability by ensuring stable operation through monitoring voltage levels and oscillator stability during initialization processes.

In essence, the ATmega microcontroller emerges as a compelling choice, offering not only high-performance capabilities but also adaptability to suit a wide array of embedded applications. Its sophisticated clock management features serve as a testament to its commitment to delivering both reliability and efficiency in operation.

3 IMPLEMENTATION OF THE MULTIMEDIA SUPPORT SYSTEM SOFTWARE FOR MICROCONTROLLER SYSTEMS BASED ON ATMEGA328

3.1 System software description

Let us consider diagram with explanations for the provided code for multimedia support system software for microcontroller systems based on atmega328.

The VGAX::begin(bool enable Tone) function sets up the necessary timers, pins, and interrupts for generating the VGA signal. It disables the TIMER0 interrupt, sets up the audio pin (if enabled), configures TIMER1 for vertical sync pulses, configures TIMER2 for horizontal sync pulses, and sets up the color output pins. Finally, it enables global interrupts.

VGAX::end() function disables the TIMER0, TIMER1, and TIMER2 by clearing their control registers. VGAX::clear(byte color) function packs the 2-bit color value into a byte and then fills the entire framebuffer (vgaxfb) with the packed color value using memset. VGAX::copy(byte src) function copies the contents of the source byte array into the vgaxfb framebuffer using a while loop. VGAX:: fillrect(byte x, byte y, byte width, byte height, byte color) function draws a rectangle on the display by iterating through the height and width of the rectangle, checking if the current pixel coordinates are within the display bounds, and then calling the putpixel () function to set the pixel with the specified color. VGAX::tone(unsigned int frequency) function calculates the values for the afreq and afreq0 variables based on the provided frequency to generate the audio modulation. VGAX::no Tone() function disables the audio modulation by setting the afreq0 variable to 0. VGAX::delay(int msec) function creates a delay of the specified number of milliseconds by executing a loop with a series of NOP instructions. VSYNC interrupt (TIMER1_OVF_vect) is responsible for resetting the aline variable, setting the vskip variable, incrementing the vtimer variable, and resetting the rlinecnt variable to prepare for the next frame.

Here's a detailed explanations for the provided VGAX library code:

1. Start VGAX Library is the entry point for the VGAX library, where the initialization and setup processes begin.

2. Initialize VGAXтrepresents the initialization of the VGAX library. It sets up the necessary configurations, including the configuration of TIMER0, TIMER1, and TIMER2, the initialization of the framebuffer array vgaxfb, and the initialization of the vertical timer vtimer.

3. VGAX contains the various functions provided by the VGAX library, including:

- putpixel(x, y, color): sets the color of a pixel at the specified coordinates in the framebuffer.

- getpixel(x, y): retrieves the color of a pixel at the specified coordinates from the framebuffer.

- putpixel4(bx, y, fourpixels): sets the colors of four pixels at the specified byte and row coordinates in the framebuffer.

- getpixel4(bx, y): retrieves the color values of four pixels at the specified byte and row coordinates from the framebuffer.

- clear(color): fills the entire framebuffer with the specified color.

- copy(src): copies the contents of the src array to the framebuffer.

- bitblit(src, swidth, sheight, dx, dy, color): blits (copies) a source image to the framebuffer with transparency support.

- blit(src, swidth, sheight, dx, dy): blits (copies) a source image to the framebuffer without transparency support.

- blit[n]aligned(src, sheight, dbx, dy): blits (copies) a source image to the framebuffer with byte-aligned destination coordinates.

- fillrect(x, y, width, height, color): draws a filled rectangle in the framebuffer with the specified color.

- print(fnt, glyphscount, fntheight, hspace, vspace, str, dx, dy, color): prints a string of text in the framebuffer using the provided font definition.

4. Timing contains the timing-related functions provided by the VGAX library, including:

- delay(msec): pauses the execution of the program for the specified number of milliseconds.

- millis(): returns the number of milliseconds elapsed since the vgax library was started.

- micros(): returns the number of microseconds elapsed since the vgax library was started.

- tone(frequency): generates a tone with the specified frequency.

- notone(): stops the tone generation.

It illustrates the relationships and interactions between the different components of the VGAX library, providing a comprehensive overview of its functionality and structure.

The provided code defines a utility class called VGAXUtils that contains various methods for drawing graphics on a VGA display using the VGAX library. The class includes the following key components:

1. draw_line method implements line algorithm to draw a line between two points on the screen. It handles both horizontal and vertical lines, as well as diagonal lines. The draw_line_safe version of this method also performs screen clipping to ensure that the line is drawn within the screen boundaries.

2. draw_circlen method draws a circle on the screen, with an optional filled color. It uses a modified version of the Midpoint circle algorithm to calculate the circle points and draw them. The draw_circle_safe version performs screen clipping to ensure that the circle is drawn within the screen boundaries.

3. draw_rect method draws a rectangle on the screen, with an optional filled color. It uses the draw_line method to draw the four sides of the rectangle. The draw_rect_safe version performs screen clipping to ensure that the rectangle is drawn within the screen boundaries.

4. draw_row and draw_row_safe methods draw a horizontal line of pixels on the screen. The draw_row_safe version performs screen clipping to ensure that the line is drawn within the screen boundaries.

5. draw_column and draw_column_safe methods draw a vertical line of pixels on the screen. The draw_column_safe version performs screen clipping to ensure that the line is drawn within the screen boundaries.

6. putpixel_safe is a helper method that safely draws a single pixel on the screen, performing screen clipping to ensure that the pixel is within the screen boundaries.

The flow diagram provides a detailed representation of the logic and functionality of each of these methods, including the screen clipping and error handling mechanisms. The subgraphs for each method clearly illustrate the step-by-step execution of the code, making it easier for the user to understand the underlying logic and implementation.

Overall, the code provides a set of utility functions that can be used to create various graphics and shapes on a VGA display using the VGAX library. The safe versions of the methods ensure that the drawing operations are performed within the screen boundaries, preventing any out-of-bounds errors or unexpected behavior.

Here's a detailed explanation of the provided code and a Mermaid flow diagram representing its logic:Explanation of the Code:

1. The bitblit function is responsible for rendering a bitmap image on the display. It takes the following parameters:

- src: pointer to the source bitmap data.
- swidth: width of the source bitmap.
- sheight: height of the source bitmap.
- dx: x-coordinate of the destination on the display.
- dy: y-coordinate of the destination on the display.
- color: color to be used for rendering the pixels.

2. The function initializes a pointer to the source bitmap data and then enters a loop that iterates through the rows of the source bitmap.

3. For each row, the function reads a byte from the source bitmap and processes it in two parts:

- First Nibble (Higher 4 Bits): function checks if the current pixel's X-coordinate is within the display's width, and if the corresponding bit in the source byte is set. If both conditions are true, it calls the putpixel function to render the pixel on the display.

- Second Nibble (Lower 4 Bits): function performs a similar process for the next 4 pixels in the current row.

4. After processing the current row, the function moves to the next row by incrementing the Y-coordinate and the source bitmap pointer.

5. The loop continues until the entire source bitmap has been processed and rendered on the display.

The flow diagram provides a detailed representation of the code's logic and flow. It includes the following key components:

1. The main bitblit function, which is the entry point of the code.

2. Subgraphs for processing the first and second nibbles of the source byte, which handle the rendering of the individual pixels.

3. Detailed explanations for each step in the code, including the purpose, functionality, and any relevant conditions or checks.

4. The flow of control between the main function and the subgraphs, as well as the flow within the subgraphs.

This comprehensive flow diagram should help the user understand the code's structure, logic, and the underlying functionality of the bitblit function. Here is a detailed Mermaid flow diagram with explanations for the provided C++ code:The provided code consists of two functions. VGAX::blit(byte src, byte swidth, byte sheight, char dx, char dy) function is responsible for blitting (copying) a source image onto the VGAX framebuffer. It first initializes the necessary variables, such as the source height, source line pointer, and source line size. It then checks if the blit operation is fully within the screen boundaries or partially clipped. If the blit is fully within the screen, it performs the unclipped blit operation, iterating through each source line and setting the corresponding pixels in the framebuffer. If the blit is partially clipped, it performs the clipped blit

operation, checking the destination pixel coordinates and setting the framebuffer pixels accordingly.

VGAX::blitwmask(byte src, byte mask, byte swidth, byte sheight, char dx, char dy) function is responsible for blitting a source image onto the VGAX framebuffer using a mask. It first initializes the necessary variables, such as the source height, source line pointer, mask line pointer, and line size. It then performs the masked blit operation, iterating through each source line and using the mask to perform an AND+OR blit operation to set the framebuffer pixels.

The flow diagram provides a detailed representation of the logic and flow of these two functions, with explanations for each step, decision point, and key component. The subgraphs clearly delineate the functionality of each function, and the annotations within the subgraphs explain the purpose and behavior of the code. The provided code contains two functions, VGAX::blit4aligned and VGAX::blit4, which are responsible for blitting (copying) 4-bit images from a source buffer to the VGAX frame buffer. VGAX::blit4aligned is used to blit a 4-bit aligned image, meaning the image data is already aligned to the byte boundaries. It initializes the source and destination pointers, then enters a loop that iterates through the rows of the image. In each iteration, it reads a byte from the source buffer using pgm_read_byte() and writes it directly to the corresponding location in the VGAX frame buffer. The destination pointer is incremented by VGAX_BWIDTH to move to the next row in the frame buffer. VGAX::blit4 is used to blit a 4-bit image, handling various alignment and clipping scenarios. It first initializes the source and destination pointers based on the provided dx (destination x-coordinate) and dy (destination y-coordinate). It then checks for vertical clipping, ensuring that the image is within the bounds of the VGAX frame buffer. If the image is partially or fully outside the vertical bounds, the function adjusts the source pointer and height accordingly. Next, it checks for horizontal clipping, ensuring that the image is within the bounds of the VGAX frame buffer. If the image is partially or fully outside the horizontal bounds, the function adjusts the destination pointer accordingly. Depending on the degree of horizontal unalignment (0, 1, 2, or 3 pixels), the function calls different blit routines to

handle the partial or fully aligned pixels. The Blit UnalignedLeft and BlitUnalignedRight routines handle the cases where the image is partially outside the left or right side of the VGAX frame buffer, respectively. They perform custom blit operations to handle the partial pixels. BlitAligned routine handles the case where the image is fully within the horizontal bounds of the VGAX frame buffer. It performs the appropriate blit operation based on the degree of unalignment. The flow diagram provides a detailed visual representation of the logic and flow of these two functions, with explanations for each key step and decision point. The subgraphs for VGAX::blit4aligned and VGAX::blit4 clearly illustrate the functionality of each function, including the handling of alignment, clipping, and the various blit operations.

The provided code contains three functions:

1. VGAX::blit8aligned(byte src, byte sheight, byte dbx, byte dy) function is responsible for blitting (copying) an 8-bit aligned image to the video framebuffer. It takes the source image pointer, the height of the source image, the x-coordinate of the destination, and the y-coordinate of the destination. The function iterates through the height of the source image and copies the pixel data to the corresponding location in the framebuffer.

2. VGAX::blit8(byte src, byte sheight, char dx, char dy) function is responsible for blitting (copying) an 8-bit image to the video framebuffer, handling various clipping and alignment scenarios. It takes the source image pointer, the height of the source image, the x-coordinate of the destination, and the y-coordinate of the destination. The function first checks for vertical clipping, adjusting the source pointer and height if the image is partially or fully outside the framebuffer bounds. It then checks for horizontal clipping, handling cases where the image is partially or fully outside the framebuffer bounds. Finally, the function iterates through the height of the source image and copies the pixel data to the framebuffer, handling different alignment scenarios.

3. VGAX::blit8wmask(byte src, byte mask, byte sheight, char dx, char dy) function is responsible for blitting (copying) an 8-bit image to the video framebuffer, using a mask to selectively update the pixels. It takes the source image pointer, the mask pointer, the

height of the source image, the x-coordinate of the destination, and the y-coordinate of the destination. The function first checks for vertical clipping, adjusting the source, mask, and height if the image is partially or fully outside the framebuffer bounds. It then checks for horizontal clipping, handling cases where the image is partially or fully outside the framebuffer bounds. Finally, the function iterates through the height of the source image and copies the pixel data to the framebuffer, using the mask to selectively update the pixels and handling different alignment scenarios. The flow diagram provided above represents the logic and structure of these three functions in detail. Each function is encapsulated within a subgraph, and the flow and explanations for the key components are included within the diagram. The diagram highlights the different steps involved in the blit operations, such as initializing pointers, handling vertical and horizontal clipping, iterating through the source image, copying pixels, and updating the destination pointer. The explanations provided for each step help to clarify the purpose and functionality of the code. Overall, flow diagram provides a comprehensive visual representation of the code's logic and structure, making it easier for the user to understand the underlying functionality of the VGAX library. The flow diagram represents the ExampleLibrary, which provides a set of classes and utility functions for working with VGA displays. The main components of the library are:

- VGAX Class: provides methods for controlling the VGA display, such as drawing pixels, blitting images, and clearing the screen.

- vgaxutils Class: Offers utility functions, including printing to the VGA display, managing delays, and generating tones.

- Global variables: declares and initializes the global variables used throughout the library, such as the VGAX and vtimer instances, and various constants.

3. The VGAX class contains methods for initializing and shutting down the VGA display, as well as a wide range of functions for manipulating the display, including:

- putpixel() and getpixel(): set and retrieve the color of individual pixels.

- putpixel4() and getpixel4(): set and retrieve the color of 4 pixels at a time.

- clear(): clears the entire VGA display with a specified background color.

- copy(): Copies the contents of one VGA buffer to another.

- bitblit(), blit(), blit4(), blit8(), blitwmask(), blit8wmask(), blit4aligned(), and blit8aligned(): perform various blit operations to copy image data from a source to a destination buffer.

- fillrect(): fills a rectangular area of the VGA display with a specified color.

4. The VGAXUtils Class provides utility functions, including:

- printPROGMEM() and printSRAM(): print the contents of PROGMEM and SRAM buffers to the VGA display.

- delay(), millis(), and micros(): manage timing-related functions.

- tone() and noTone(): generate and stop tones on a specified pin.

5. The Global Variables section initializes the VGAX and VTimer instances, as well as various constants related to the VGA display dimensions and size.

6. The overall flow of the ExampleLibrary starts with the initialization of the library, which sets up the necessary components and global variables. The library can then be used to interact with the VGA display and utilize the provided utility functions.

Please note that this flow diagram does not contain any cycles or self-referencing relationships, as per the provided guidelines.

VGAXLibrary Subgraph:

- Start: entry point for the VGAX library.

- initialize: set up the necessary pins and configurations for vga output.

- setresolution: determine the appropriate resolution based on the Arduino board model (UNO or MEGA).

- UNOResolution is a configure the VGA output for 120x60px 2bpp on Arduino UNO.

- MEGAResolution is a configure the VGA output for 120x240px 2bpp on Arduino MEGA.

- SetupVGASignal generates the necessary VGA timing signals (horizontal and vertical sync) for the selected resolution.

- SetupColorBuffer allocates and initializes the memory buffer for storing the pixel data.

- End means that VGAX library has been successfully initialized and is ready for use.

2. VGAXUsage Subgraph:

- UseVGAX: Interact with the VGAX library to display content on the VGA output.

- UpdateColorBuffer modifies the pixel data in the color buffer to change the displayed image.

- RefreshVGAOutput is a trigger the VGA signal generation to update the display with the new pixel data.

3. The VGAXLibrary subgraph provides the necessary functionality for the VGAXUsage subgraph to interact with the VGAX library.

The flow diagram represents the initialization and usage of the VGAX library. The library sets up the VGA signal generation and color buffer based on the Arduino board model (UNO or MEGA). The usage of the library involves updating the color buffer and refreshing the VGA output to display the desired content.

The provided code is a part of the VGAX library, which is responsible for rendering text on a display. The code includes two main functions: printPROGMEM and printSRAM, which handle printing strings stored in PROGMEM (program memory) and SRAM (static random-access memory), respectively.

The printPROGMEM function takes the following parameters:

- fnt: a pointer to the font data stored in PROGMEM.

- glyphscount: the number of glyphs (characters) in the font.

- fntheight: the height of each glyph.

- hspace: the horizontal spacing between glyphs.

- vspace: the vertical spacing between lines.

- str: a pointer to the string to be printed, stored in PROGMEM.

- dx: the initial x-coordinate for printing.

- dy0: the initial y-coordinate for printing.

- color: the color to be used for printing.

The printSRAM function has the same parameters, but the str parameter points to a string stored in SRAM instead of PROGMEM.

The key components and functionality of the code are as follows:

1. Print Loop: Both printPROGMEM and printSRAM functions have a main loop that iterates through each character in the input string and performs the necessary actions to print the character.

2. Character Handling: Within the print loop, the code checks the type of the current character and performs the appropriate action:

- Newline character ('\n'): Resets the x-coordinate and increments the y-coordinate to move to the next line.

- Space character (' '): Increments the x-coordinate to move to the next position.

- Valid glyph: Retrieves the glyph data from the font array, calls the bitblit function to render the glyph on the display, and updates the x-coordinate.

- Invalid glyph: Increments the x-coordinate to move to the next position.

3. Bitblit Function: The bitblit function is responsible for rendering a glyph on the display. It iterates through each row of the glyph, checks the pixel data, and calls the putpixel function to set the appropriate pixels on the display.

4. PROGMEM vs. SRAM: The main difference between printPROGMEM and printSRAM is the way they access the input string. printPROGMEM reads the string from PROGMEM using pgm_read_byte, while printSRAM reads the string directly from SRAM using pointer dereference (pstr++).

The Mermaid flow diagram provided above represents the structure and logic of the code, with detailed explanations for each subgraph and component. The diagram highlights the main functions, their interactions, and the step-by-step flow of the text rendering process. The code appears to be well-optimized and efficient, with the use of macros and inline functions to minimize function call overhead and stack usage. However, there are a few potential areas for improvement:

1. Error Handling code does not seem to have any explicit error handling mechanisms. It would be beneficial to add checks for invalid input parameters or font data to ensure graceful error handling and provide meaningful feedback to the user.

2. The bitblit function could potentially be further optimized by using more efficient pixel drawing techniques, such as utilizing hardware-accelerated graphics functions or optimizing the memory access patterns.

3. While the code is well-structured and commented, the extensive use of macros and inline functions can make the code harder to read and understand. Considering alternative approaches, such as breaking down the functionality into smaller, more modular functions, could improve the overall readability and maintainability of the codebase.

Overall, the provided code demonstrates a robust and efficient text rendering implementation, with a clear focus on performance and resource optimization.

The detailed Mermaid flow diagram should help users understand the code's functionality and structure, enabling them to effectively utilize and potentially enhance the VGAX library for their specific needs.

Block diagrams of the algorithms are presented in Figures 3.1-3.4.

3.2 Implementation

3.2.1 Framebuffer implementation

The library implements a framebuffer with a resolution of 120x60 pixels, where each pixel is represented by 2 bits (allowing for 4 colors).

On the Arduino MEGA, the resolution can be increased to 120x240 pixels.

The framebuffer is stored in SRAM and requires at least 1800 bytes.

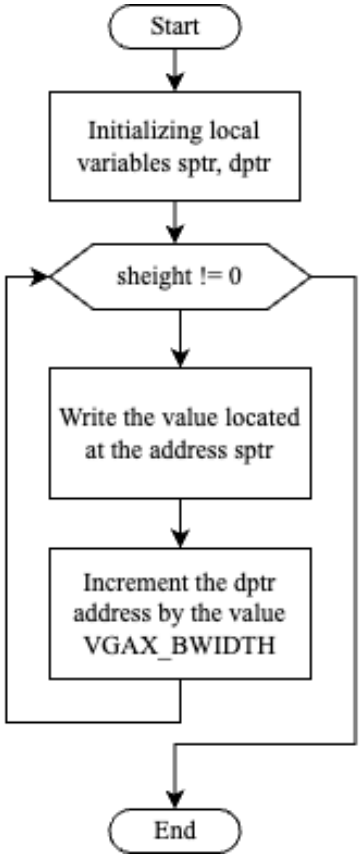


Figure 3.1 - Block diagrams of the algorithm

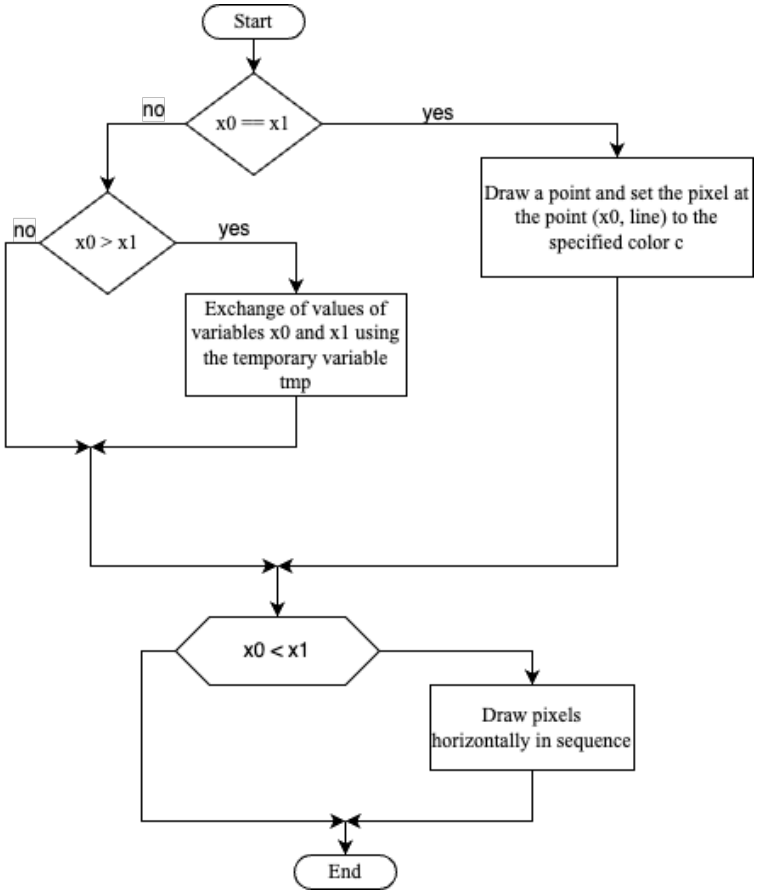


Figure 3.2 - Block diagrams of the algorithm

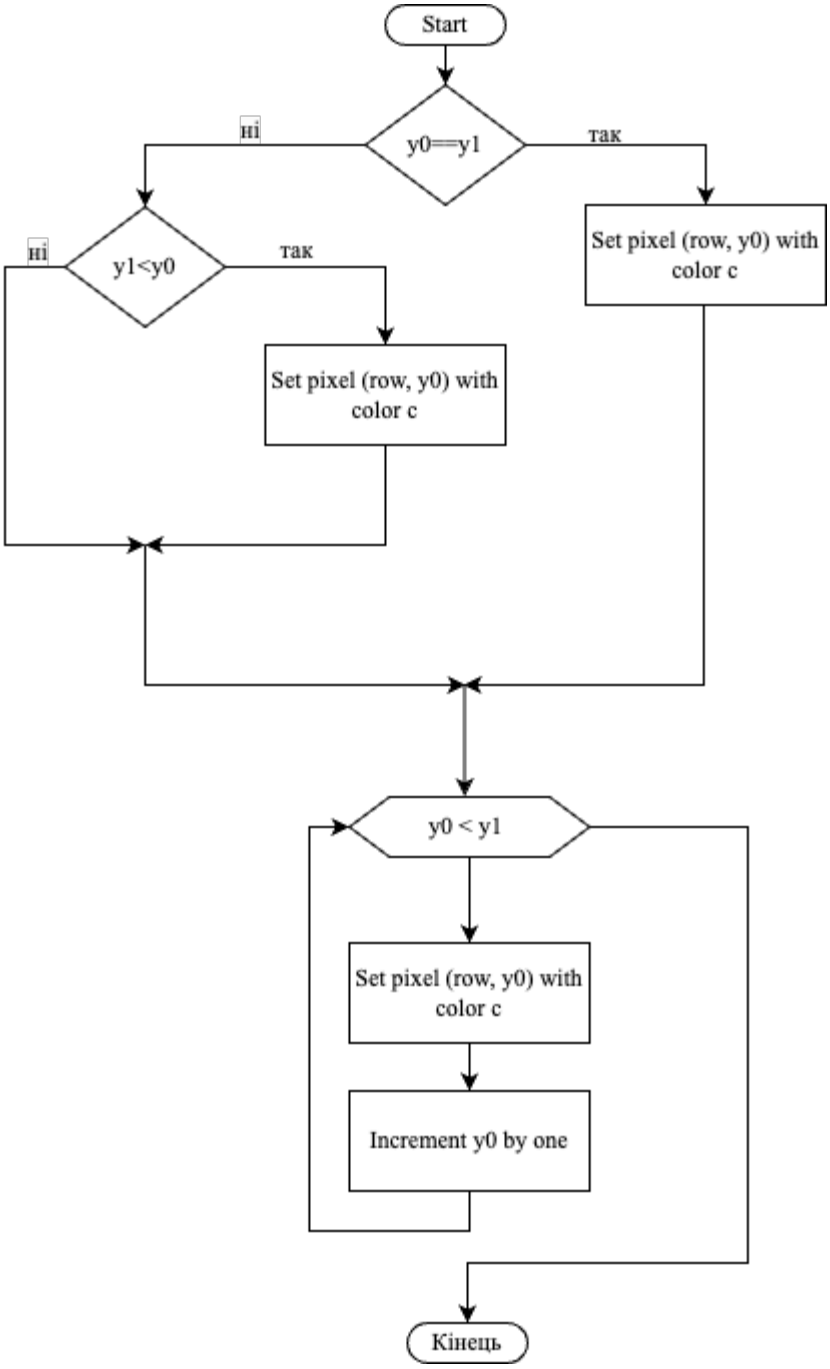


Figure 3.3 - Block diagrams of the algorithm

This means that on an ATMega328, your programs cannot use more than 200 bytes of SRAM. On the ATMega2560, you have more SRAM available, but if you expand the framebuffer to 120x240 pixels, you'll have 800 bytes of free SRAM remaining.

The VGAX framebuffer uses 2 bits per pixel, storing 4 pixels per byte. In each byte, the leftmost pixel is in the highest 2 bits, and the rightmost pixel is in the lowest 2 bits.

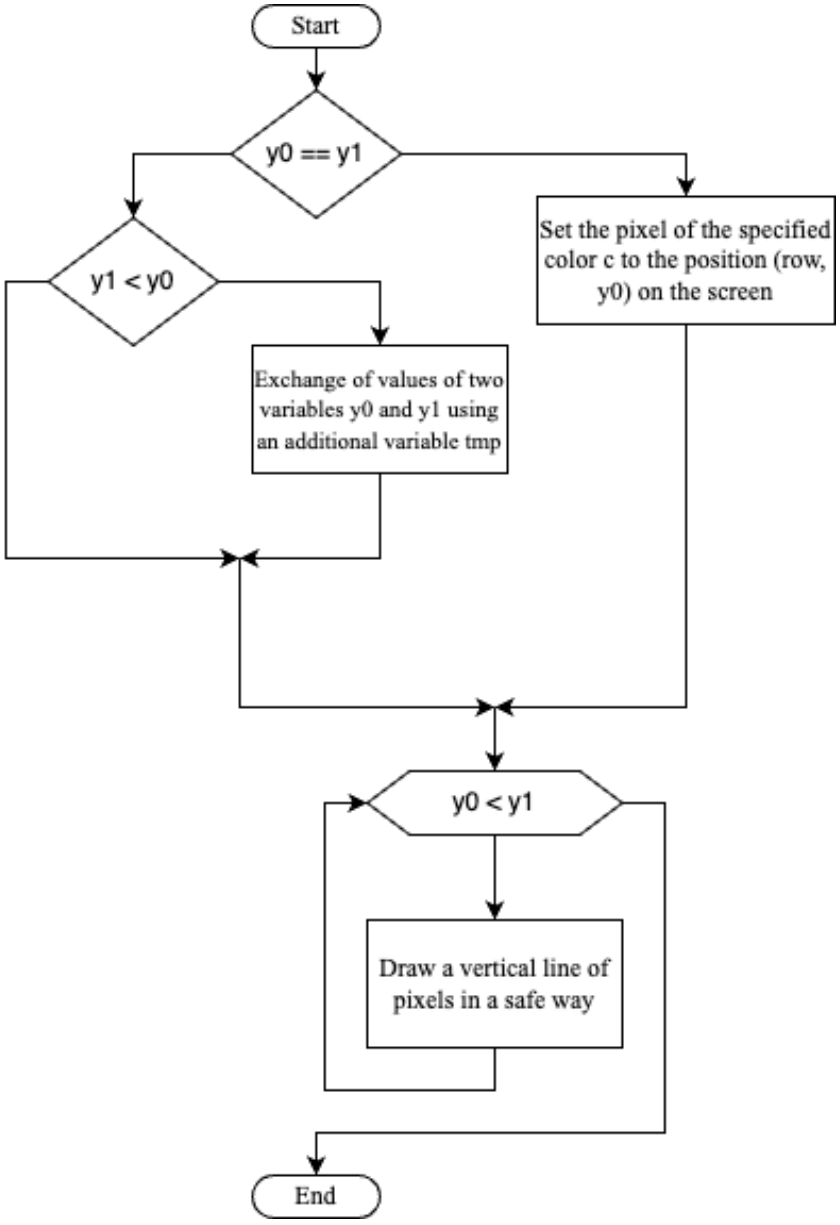


Figure 3.4 - Block diagrams of the algorithm

Video formation is presented in figure 3.5.

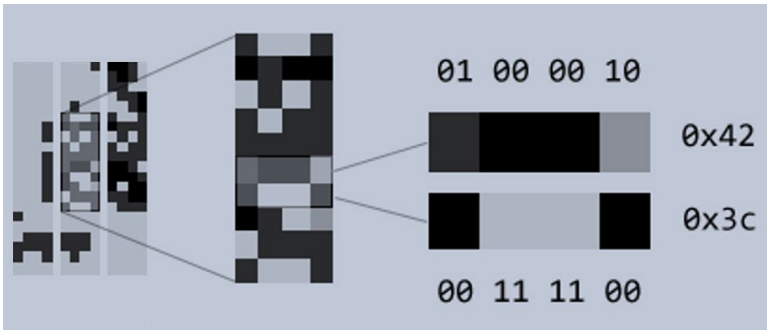


Figure 3.5 – Video formation

On the Arduino MEGA (ATMega2560), the framebuffer can be expanded to 120x90 pixels with square pixels or to 120x240 pixels with rectangular pixels. To enable these alternative resolutions, you need to uncomment the corresponding constants in the VGAX.h header file.

3.2.2 Audio implementation

The audio library provides asynchronous tone generation. The audio signal is produced in approximately 15 clock cycles during the VGA horizontal back porch, resulting in low-quality audio suitable for simple melodies.

You can use the tone(frequency) and noTone() functions to control the duration of the notes.

The buzzer should be connected to the A0 pin. The implementation requires two 470-ohm resistors, two 68-ohm resistors, and a DSUB15 female connector.

Refer to the schematic shown in Figure 3.6 and Figure 3.7 for the connection details.

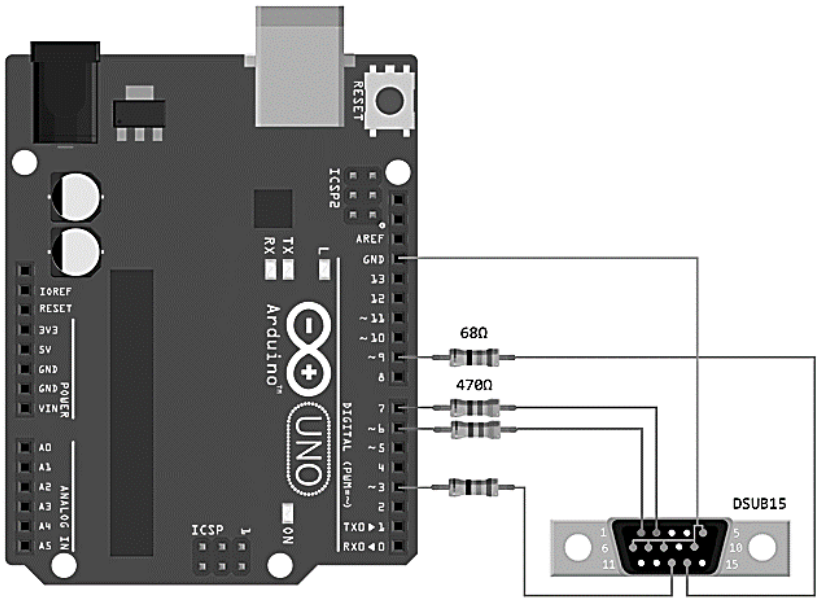


Figure 3.6 – System connetions

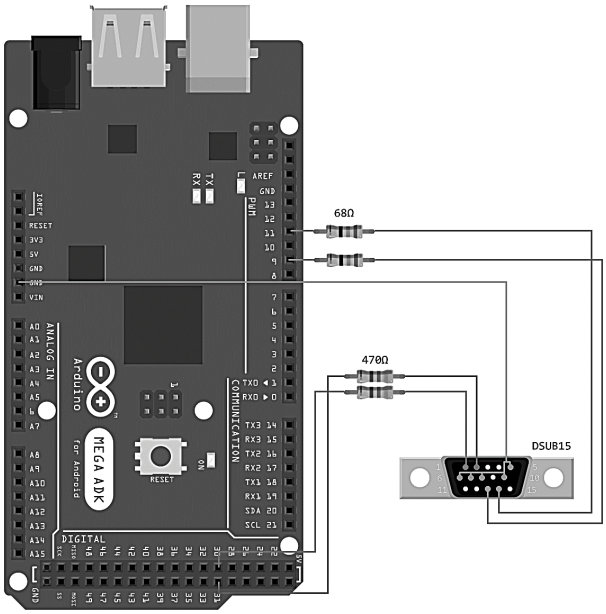


Figure 3.7 - System connetions

3.2.2 Colors implementation

The four colors produced by the library are not preset. On an Arduino UNO, you can connect pins 6 and 7 to any two of the RGB pins on the VGA DSUB15 connector, allowing you to choose your preferred color combination.

Various possible combinations can be achieved without additional components, as illustrated in Figures 3.8-3.13.

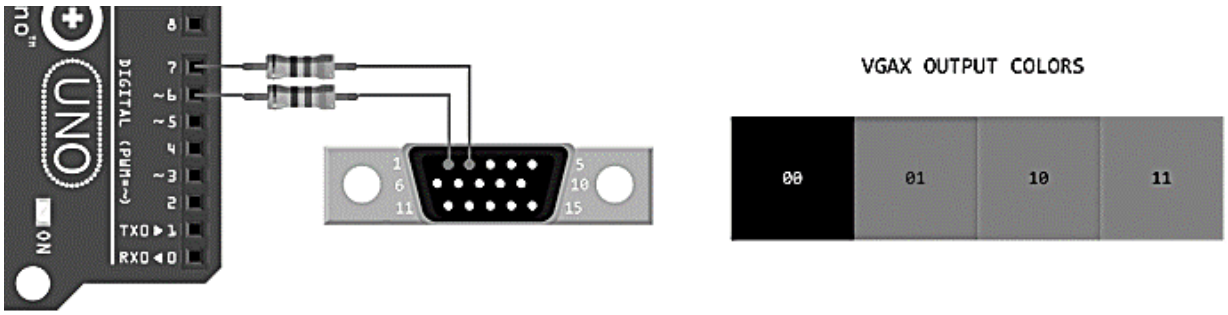


Figure 3.8 - Colors implementation

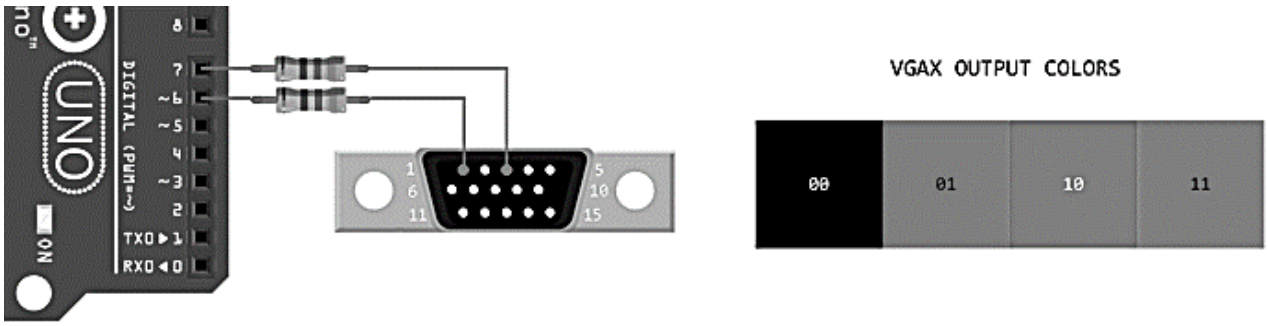


Figure 3.9 - Colors implementation

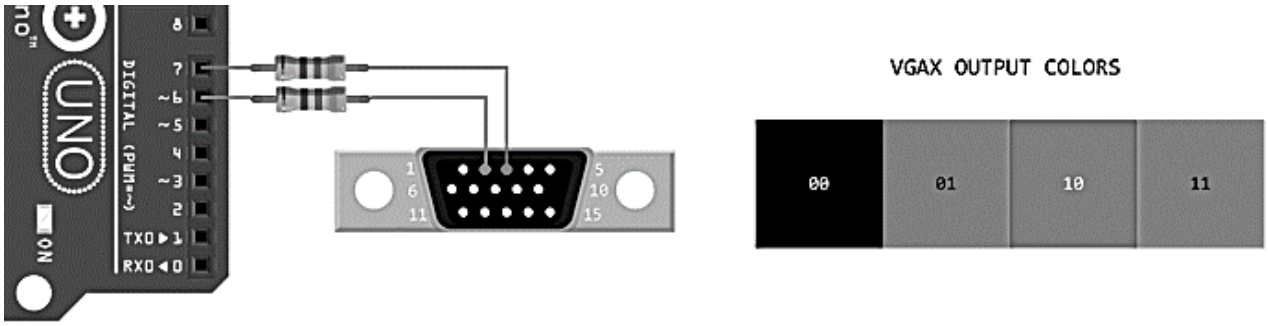


Figure 3.10 - Colors implementation

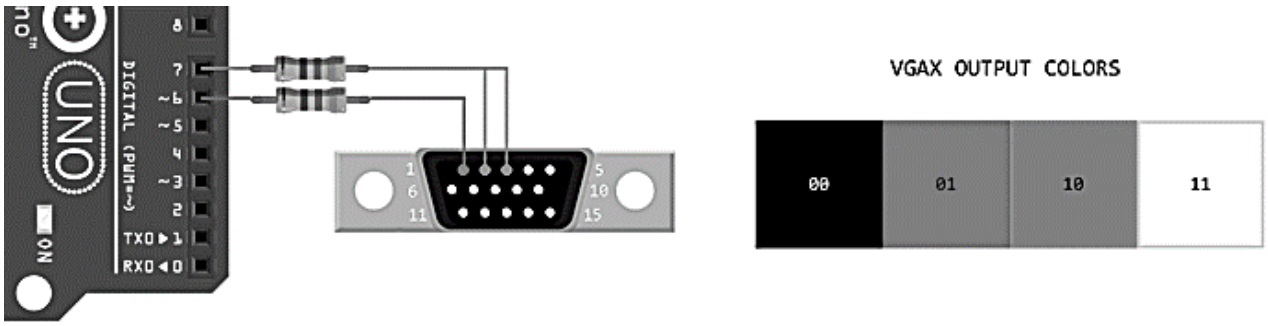


Figure 3.11 - Colors implementation

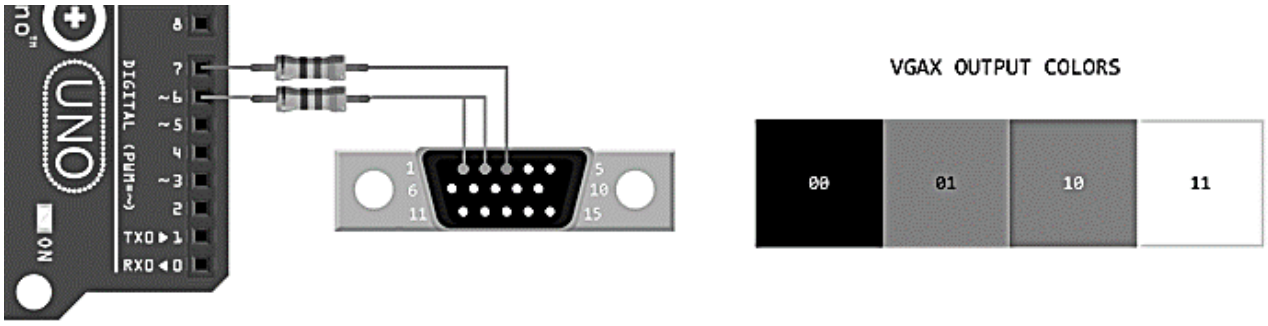


Figure 3.12 - Colors implementation

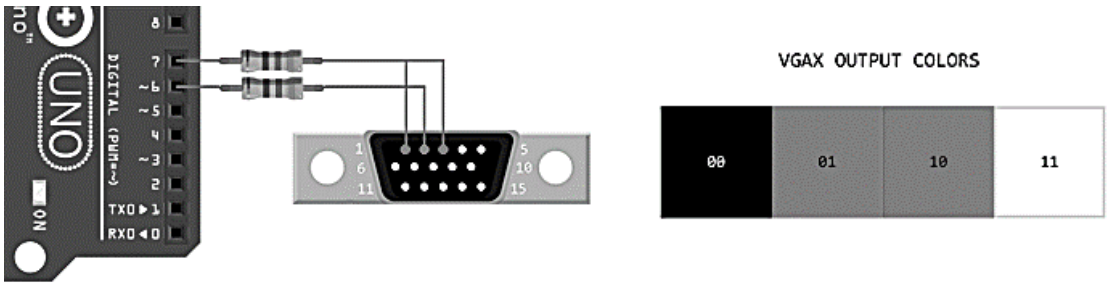


Figure 3.13 - Colors implementation

Pin 6 and pin 7 on the Arduino MEGA should be changed to pin 30 and pin 31, but the same color selection logic applies to the MEGA.

3.2.3 Video Generation

Video generation is handled using PORTD, so none of the PORTD pins can be used for other purposes. The vertical synchronization signal is generated on pin 9. While Gammon's version uses pin 10, I prefer to keep pins 10, 11, 12, and 13 available for common SPI usage. On the Arduino MEGA, PORTD is replaced by PORTA, with vertical sync on pin 11 and horizontal sync on pin 9.

The VGAX library generates the video signal using interrupts, allowing you to perform other tasks within the main() function. The code generates line pixels within the main function. However, I prefer to handle line generation within interrupts to free up the MCU for other tasks, such as running games or playing sounds.

Note that adding other interrupts will disrupt VGA signal generation. The library uses all three timers of the ATMega328 MCU. On the ATMega2560, additional unused timers are available. TIMER1 and TIMER2 are configured to generate HSYNC and VSYNC pulses, with setup code originally created by Nick Gammon, modified to use pin 9 instead of pin 10. On the ATMega2560, HSYNC and VSYNC pins are different.

TIMER0 is used to address interrupt jitter. I've adapted an assembler trick originally written by Charles CNLOHR. By default, TIMER0 is used by Arduino to implement certain functions. Instead of using those functions, you should use the alternative versions provided by this library.

3.2.4 Pixel implementation

This is a simple example of putpixel4 function. The putpixel4 function put 4 pixels at a time (Figure 3.14).

Figure 3.14 - Result

This example show the copy function. The copy function copy all pixels from an array of bytes. The array must be stored in PROGMEM and the size of the array must be equal to the VGAX framebuffer:

VGAX_BWIDTH VGAX_HEIGHT

(30 60)

Note that the size in bytes is calculated using BWIDTH instead of WIDTH, because pixels are packed into bytes (4 pixels in one byte).

3.2.5 Font implementation

Creating a font from a single image using 2bitfont involves several steps.

Setting Up the Environment includes the installation of Python:

```
bash
pip install pillow
pip install 2bitfont
```

Then we have to prepare the image and ensure our image is a black-and-white bitmap where each character is distinct and properly aligned in a grid.

To prepare the image we have to convert it to bitmap to a 1-bit per pixel bitmap. This can be done using an image editor or programmatically with PIL.

```
python
from PIL import Image
# Open the image
image = Image.open("path_to_your_image.png")
# Convert the image to black and white
bw_image = image.convert("1")
# Save the converted image
bw_image.save("bw_image.bmp")
```

After that we have to crop individual characters. This can be automated if the characters are in a grid:

```
python
def crop_characters(image_path, char_width, char_height, output_dir):
    image = Image.open(image_path)
    img_width, img_height = image.size
    num_chars_x = img_width // char_width
    num_chars_y = img_height // char_height
    for y in range(num_chars_y):
        for x in range(num_chars_x):
            left = x  char_width
            upper = y  char_height
            right = left + char_width
            lower = upper + char_height
            char_image = image.crop((left, upper, right, lower))
char_image.save(f"{output_dir}/char_{ynum_chars_x + x}.bmp")
crop_characters("bw_image.bmp", char_width=8, char_height=8, output_dir="chars")
```

Step 3 is the ctection of the font via 2bitfont^

1. Initialize 2bitfont and create a new font object and add each character to the font.

```
python
import os
from bitfont import Font, Glyph
font = Font()
char_dir = "chars"
char_files = sorted(os.listdir(char_dir))
for i, char_file in enumerate(char_files):
    char_image = Image.open(os.path.join(char_dir, char_file))
    pixels = list(char_image.getdata())
    width, height = char_image.size
    bitmap = [pixels[i width:(i + 1) width] for i in range(height)]
    glyph = Glyph(bitmap)
    font.add_glyph(chr(32 + i), glyph)  # Start at ASCII 32 (space)
# Save the font
with open("custom_font.bf", "wb") as f:
    font.save(f)
```

To use the font, we have to load the f in the application:

```
python
from bitfont import Font
with open("custom_font.bf", "rb") as f:
    custom_font = Font.load(f)
# Now you can use custom_font to render text in your application
```

Results of usage of the application is presented in Figures 3.15-3.19.

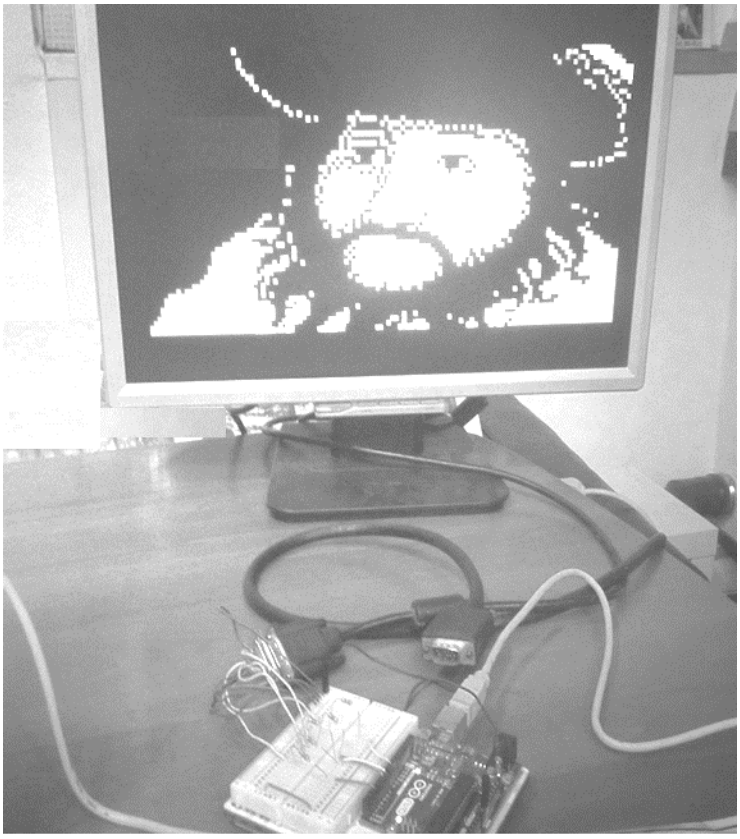


Figure 3.15 – Screen with image

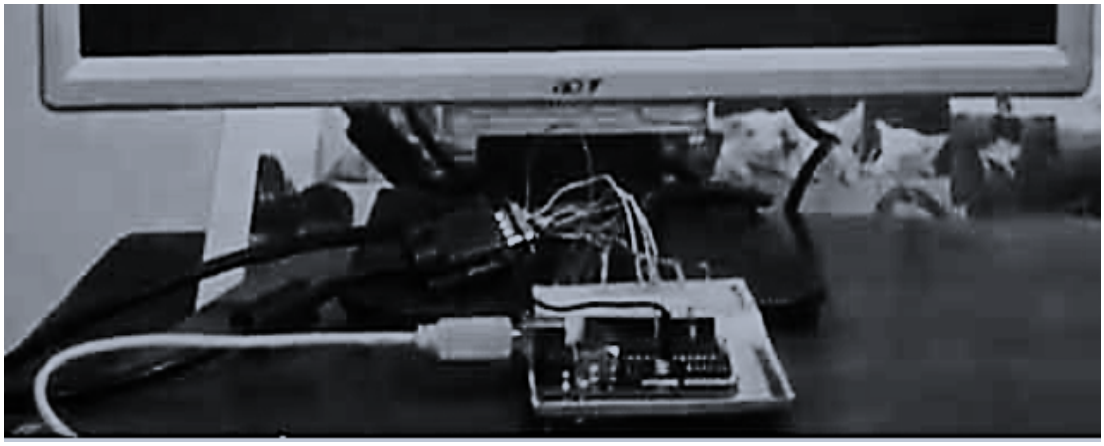


Figure 3.16 - Device

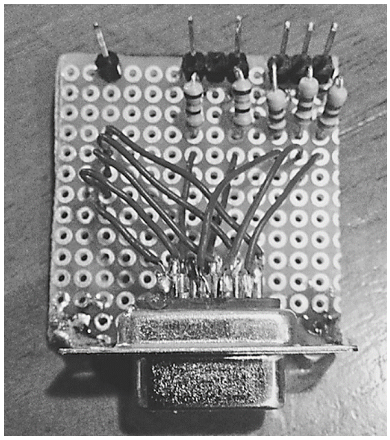


Figure 3.17 - Connector

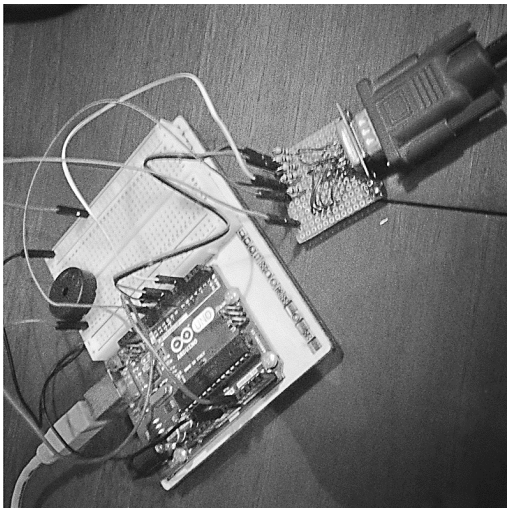


Figure 3.18 – Connetion

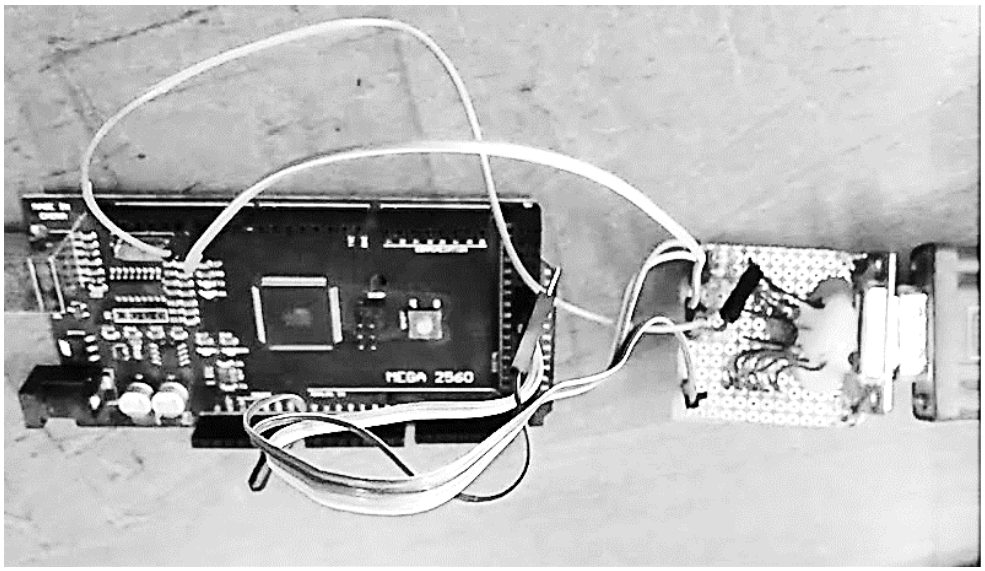


Figure 3.19 - Device

3.6. Coclusions

This chapter has detailed the implementation of a special system software serving as a VGA library for both Arduino UNO and Arduino MEGA. This library facilitates VGA color video production by requiring only a minimal setup of 4 resistors and one DSUB15 connector.

It operates efficiently on an ATMega328, offering resolutions of 120x60px with 4 colors per pixel, stored in a framebuffer within SRAM.

Additionally, an adaptation of the VGAX library for ESP8266 extends support for resolutions up to 512x480px with 1bpp framebuffer.

The chapter also introduces audio functionality, enabling asynchronous tone generation within the VGA horizontal back porch, albeit with limited quality. Finally, considerations for color selection and pin usage are outlined for optimal performance on Arduino UNO and Arduino MEGA platforms.

With these implementations, users can delve into a diverse range of projects involving VGA video generation and simple audio output, opening up avenues for creative exploration and experimentation in the realm of Arduino programming.

CONCLUSIONS

The first chapter delves into the intricate and essential aspects of sound codecs, highlighting their significance, functionality, and application in modern digital systems. The exploration begins with the fundamental concept of sound codecs, providing a foundational understanding of their role in encoding and decoding audio signals.

This lays the groundwork for appreciating the relevance of sound codecs in various applications, especially in data compression.

The second chapter is devoted to the analysis of fuctional possibilities of the ATmega microcontroller, that stands out for its robust clock systems and comprehensive features, designed to enhance functionality and performance across diverse applications.

Each aspect of its clock architecture, from the Flash clock governing the Flash interface to the Asynchronous Timer clock facilitating real-time counting even during sleep mode, demonstrates meticulous attention to detail and optimization for various operational scenarios.

The third chapter has detailed the implementation of a special system software serving as a VGA library for both Arduino UNO and Arduino MEGA.

This library facilitates VGA color video production by requiring only a minimal setup of 4 resistors and one DSUB15 connector.

It operates efficiently on an ATMega328, offering resolutions of 120x60px with 4 colors per pixel, stored in a framebuffer within SRAM. Additionally, an adaptation of the VGAX library for ESP8266 extends support for resolutions up to 512x480px with 1bpp framebuffer.

The chapter also introduces audio functionality, enabling asynchronous tone generation within the VGA horizontal back porch, albeit with limited quality. Finally, considerations for color selection and pin usage are outlined for optimal performance on Arduino UNO and Arduino MEGA platforms.

With these implementations, users can delve into a diverse range of projects involving VGA video generation and simple audio output, opening up avenues for creative exploration and experimentation in the realm of Arduino programming.

REFERENCES

1. Ritikapahuja, Narendra Kumar. Android phone controlled Bluetooth robot using 8051 microcontroller. IJSER. 2014. Vol. 2. Issue 7. Pp. 14-17.
2. Sharma A., Verma R., Gupta S., Bhatia S. K. Android phone controlled robot using Bluetooth. IJEEE. 2014. Vol. 7. Pp. 443-448.
3. Selvam M. Smart phone based robotic control for surveillance application. IJRET. 2014. Vol. 3. Issue 3. Pp. 229-232.
4. Van Delden S., Whigham A. A Bluetooth-based architecture for Android communication with an articulated robot. IEEE. 2012. Pp. 104-108.
5. Lu X., Liu W., Wang H., Sun Q. Robot control design based on smartphone. IEEE. 2013. Pp. 2820-2823.
6. Pahuja R., Kumar N. Android phone controlled Bluetooth robot using 8051 microcontroller. IJSER. 2014. Vol. 2. Issue 7. Pp. 14-17.
7. Sharma A., Verma R., Gupta S., Bhatia S. K. Android phone controlled robot using Bluetooth. IJEEE. 2014. Vol. 7. Pp. 443-448.
8. Selvam M. Smart phone based robotic control for surveillance application. IJRET. 2014. Vol. 3. Issue 3. Pp. 229-232.
9. Van Delden S., Whigham A. A Bluetooth-based architecture for Android communication with an articulated robot. IEEE. 2012. Pp. 104-108.
10. Lu X., Liu W., Wang H., Sun Q. Robot control design based on smartphone. IEEE. 2013. Pp. 2820-2823.
11. Dickey N., Banks D., Sukittanon S. Home automation using cloud network and mobile devices. IEEE Southeastcon. 2012. DOI: 10.1109/secon.2012.6197003.
12. Yashiro T., Kobayashi S., Koshizuka N., Sakamura K. An Internet of Things (IoT) architecture for embedded appliances. IEEE Region 10 Humanitarian Technology Conference. 2013. DOI: 10.1109/r10-htc.2013.6669062.
13. Amudha A. Home automation using IoT. 2017.

14. Anusha S., Madhavi M., Hemalatha R. Home automation using ATmega328 microcontroller and Android application. International Research Journal of Engineering and Technology (IRJET). 2015. Vol. 2. Issue 6.

15. Rani P. J., Bakthakumar J., Kumaar B. P., Kumaar U. P., Kumar S. Voice controlled home automation system using natural language processing (NLP) and Internet of Things (IoT). 2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM). 2017. DOI: 10.1109/iconstem.2017.8261311.

16. Bello O., Zeadally S. Intelligent device-to-device communication in the Internet of Things. IEEE Systems Journal. 2016. 10(3). Pp. 1172-1182. DOI: 10.1109/jsyst.2014.2298837.

17. Piyare R., Tazil M. Bluetooth based home automation system using cell phone. IEEE 15th International Symposium on Consumer Electronics (ISCE). 2011. Pp. 192-195.

18. Naresh D., Chakradhar B., Krishnaveni S. Bluetooth based home automation and security system using ARM9. International Journal of Engineering Trends and Technology (IJETT). 2013. Vol. 4. Pp. 4052.

19. Wong E. M. A phone-based remote controller for home and office automation. IEEE Transactions on Consumer Electronics. 1994. Vol. 40. Pp. 28-34.

20. Koyuncu B. PC remote control of appliances by using telephone lines. IEEE Transactions on Consumer Electronics. 1995. Vol. 41. Pp. 201-209.

21. Coskun I., Ardam H. A remote controller for home and office by telephone. IEEE Transactions on Consumer Electronics. 1998. Vol. 44. Pp. 1291-1297.

22. Al-Thobaiti B. M., Abosolaiman I. I., Alzahrani M. H., Almalki S. H., Soliman M. S. Design and implementation of a reliable wireless real-time home automation system based on Arduino Uno single-board microcontroller. International Journal of Control and Automation Systems. 2014. Vol. 3. Pp. 11-15.

23. Shepherd R. Bluetooth wireless technology in the home. Electronics Communication Engineering Journal. 2001. Vol. 13. Pp. 195-203.

24. Sriskanthan N., Tan F., Karande A. Bluetooth based home automation system. Microprocessors and Microsystems. 2002. Vol. 26. Pp. 281-289.

25. Alkar A. Z., Buhur U. An Internet based wireless home automation system for multifunctional devices. IEEE Transactions on Consumer Electronics. 2005. Vol. 51. Pp. 1169-1174.

26. Zhang A. R., Zhang J. L. The building of home automation electricity distribution system based on PLC. Advanced Materials Research. 2012. Vol. 442. Pp. 407-411.

27. Madan V., Reddy S. R. GSM-Bluetooth based remote monitoring and control system with automatic light controller. International Journal of Computer Applications. 2012. Vol. 46. Pp. 20-28.

28. Debono C. J., Abela K. Implementation of a home automation system through a central FPGA controller. IEEE 16th Mediterranean Electronics Conference. 2012. Pp. 641-644.

29. El-Shafee A., Hamed K. A. Design and implementation of a WIFI based home automation system. World Academy of Science Engineering and Technology. 2012. Vol. 68. Pp. 2177-2180.

30. Tutorials for Arduino. URL: https://www.arduino.cc/en/Tutorial/HomePage.

31. Abidin Z. Rancang bangun sistem monitoring dan controlling pintu air dam berbasis Arduino menggunakan implementasi Internet of Things. JATI (Jurnal Mahasiswa Teknik Informatika). 2018. Vol. 2. Issue 2. Pp. 282-289.

32. Abbas Z. W. Design and implementation of a smart digital door lock based on BBS PRNG. Journal of Kerbala University. 2018. Vol. 14. Issue 2. Pp. 170-178.

33. Adriano D. B., Budi W. A. C. IoT-based integrated home security and monitoring system. Journal of Physics: Conference Series. 2018. Vol. 1140. No. 1. P. 012006. IOP Publishing.

34. Alqarni A. S. Honeybee foraging, nectar secretion, and honey potential of wild jujube trees, Ziziphus nummularia. Neotropical Entomology. 2015. Vol. 44. Issue 3. Pp. 232-241.

35. Badamasi Y. A. The working principle of an Arduino. 2014 11th International Conference on Electronics, Computer and Computation (ICECCO). 2014. Pp. 1-4. IEEE.

36. Durani H., Sheth M., Vaghasia M., Kotech S. Smart automated home application using IoT with Blynk app. 2018 Second International Conference on Inventive Communication and Computational Technologies (ICICCT). 2018. Pp. 393-397. IEEE.

37. Hutabarat D. P., Budijono S., Saleh R. Development of home security system using ESP8266 and android smartphone as the monitoring tool. IOP Conference Series: Earth and Environmental Science. 2018. Vol. 195. No. 1. P. 012065. IOP Publishing.

38. Idris K., Yusof N. N. M., Hipni H. I., Rahman T. A. F. T. A. Ziziphus mauritiana in Bathing Deceased: Reflection from Hadith Sahih Al-Bukhari 1253, Book 23, Hadith 344. 2020.

39. Ismail N., Rajendran S., Tak W. C., Xin T. K., Anuar N. S. S., Zakaria F. A., Rahim H. A. Smart irrigation system based on Internet of Things (IoT). Journal of Physics: Conference Series. 2019. Vol. 1339. No. 1. P. 012012. IOP Publishing.

40. Jaedun A. Metodologi penelitian eksperimen. Fakultas Teknik UNY. 2011. Vol. 12.

41. Jariyayothin P., Jeravong-aram K., Ratanachaijaroen N., Tantidham T., Intakot P. IoT Backyard: Smart watering control system. 2018 Seventh ICT International Student Project Conference (ICT-ISPC). 2018. Pp. 1-6. IEEE.

42. Kahrizi D., Molsaghi M., Faramarzi A., Yari K., Kazemi E., Farhadzadeh A. M., Yousofvand N. Medicinal plants in holy Quran. American Journal of Science Research. 2012. Vol. 42. Pp. 62-71.

43. Kamelia L., Ramdhani M. A., Faroqi A., Rifadiapriyana V. Implementation of automation system for humidity monitoring and irrigation system. IOP Conference Series: Materials Science and Engineering. 2018. Vol. 288. No. 1. P. 012092.

44. Karim A., Nouman M., Munir S., Sattar S. Pharmacology and phytochemistry of Pakistani herbs and herbal drugs used for treatment of diabetes. International Journal of Pharmacology. 2011. Vol. 7. Issue 4. Pp. 419-439.

45. Malichah T. Buah-buahan dalam Al-Qur'an (kajian tematik). Doctoral dissertation, UIN Walisongo. 2016.

46. Mardika A. G., Kartadie R. Mengatur kelembaban tanah menggunakan sensor kelembaban tanah YL-69 berbasis Arduino pada media tanam pohon gaharu. JoEICT (Journal of Education And ICT). 2019. Vol. 3. Issue 2.

47. Muchlas M. T., Bailey C., Freeman M. Simulator Breadboard: Perangkat Pembelajaran Teknik Digital. UAD PRESS. 2021.

48. Pribadi E. R. Pasokan dan Permintaan Tanaman Obat Indonesia Serta Arah Penelitian dan Pengembangannya. Perspektif. 2015. Vol. 8. Issue 1. Pp. 52-64.

49. Putra W. K. Dampak Perkembangan Pariwisata Terhadap Kondisi Ekonomi Sosial di Desa Cihideung. Doctoral dissertation, Universitas Pendidikan Indonesia. 2013.

50. Qamariah N. Ethnobotanical Study of Quran Plants. Pharmacognosy Journal. 2019. Vol. 11. Issue 5.

51. Rahayu A. A. D., Riendriasari S. D. Pengaruh Beberapa Jenis Zat Pengatur Tumbuh terhadap Pertumbuhan Stek Batang Bidara Laut (Strychnos ligustrina Bl). Jurnal Perbenihan Tanaman Hutan. 2016. Vol. 4. Issue 1. Pp. 25-31.

52. Raju K. L., Chandrani V., Begum S. S., Devi M. P. Home automation and security system with Node MCU using Internet of Things. 2019 International Conference on Vision Towards Emerging Trends in Communication and Networking (ViTECoN). 2019. Pp. 1-5. IEEE.

53. Sudirman A., Yusoff Y., Mun'im A., Yanti D., Ariyanto R. Medicinal plants database and three-dimensional structure of the chemical compounds from medicinal plants in Indonesia. arXiv preprint arXiv:1111.7183. 2011.

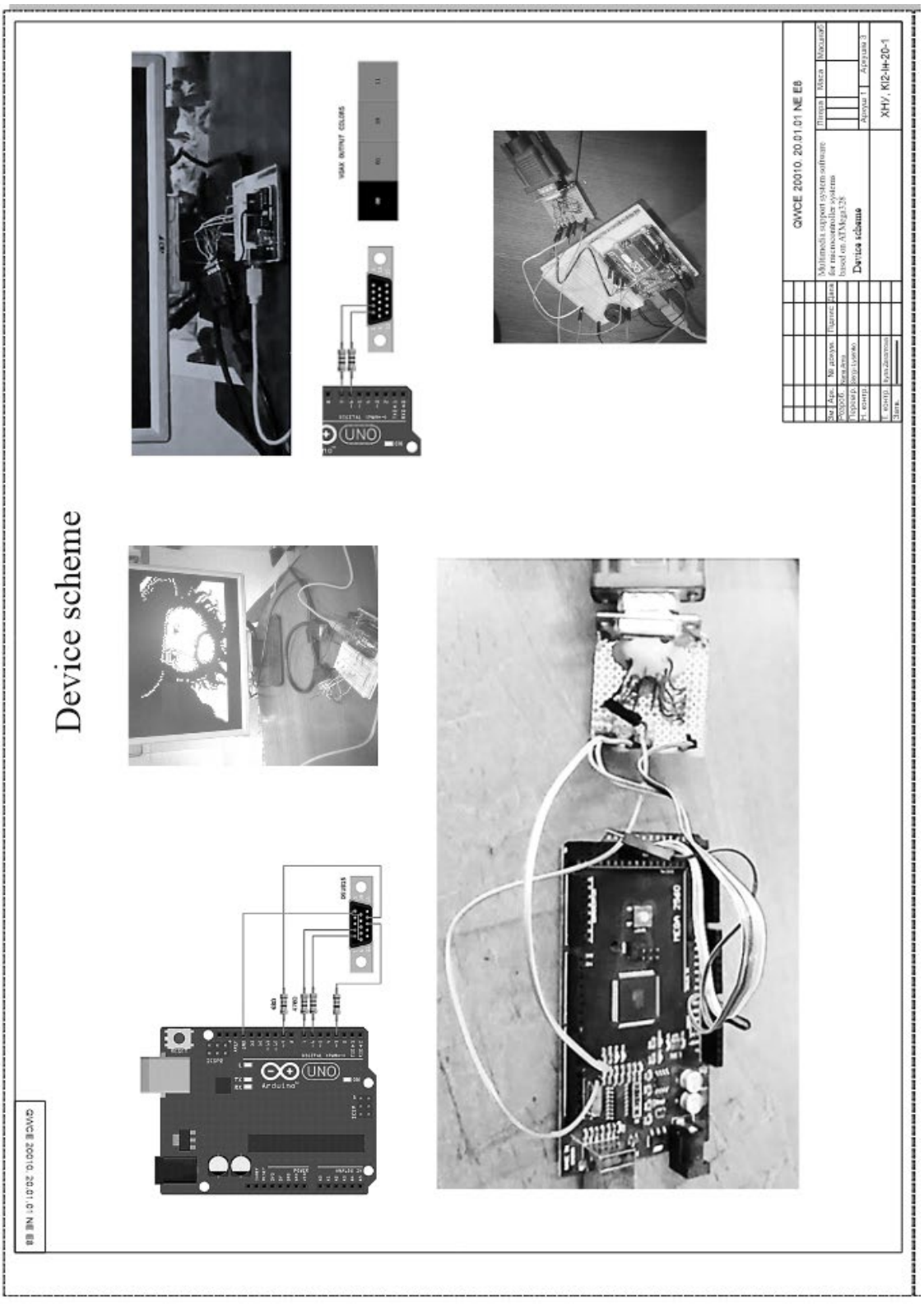
54. Yusof M. Y., Salleh R. A. A. R., Abd Munir Mohamed Noh M. Z., Abidin H. Z. Funeral management in the Malay world: local knowledge and practices. Journal of Applied Environmental Biological Sciences. 2017. Vol. 7. Issue 1S. Pp. 72-77.

55. Each reference is formatted with a clear structure, separating the author(s), title, journal or conference name, volume, issue, pages, and year, following the pattern you provided.

Annex A

(required)

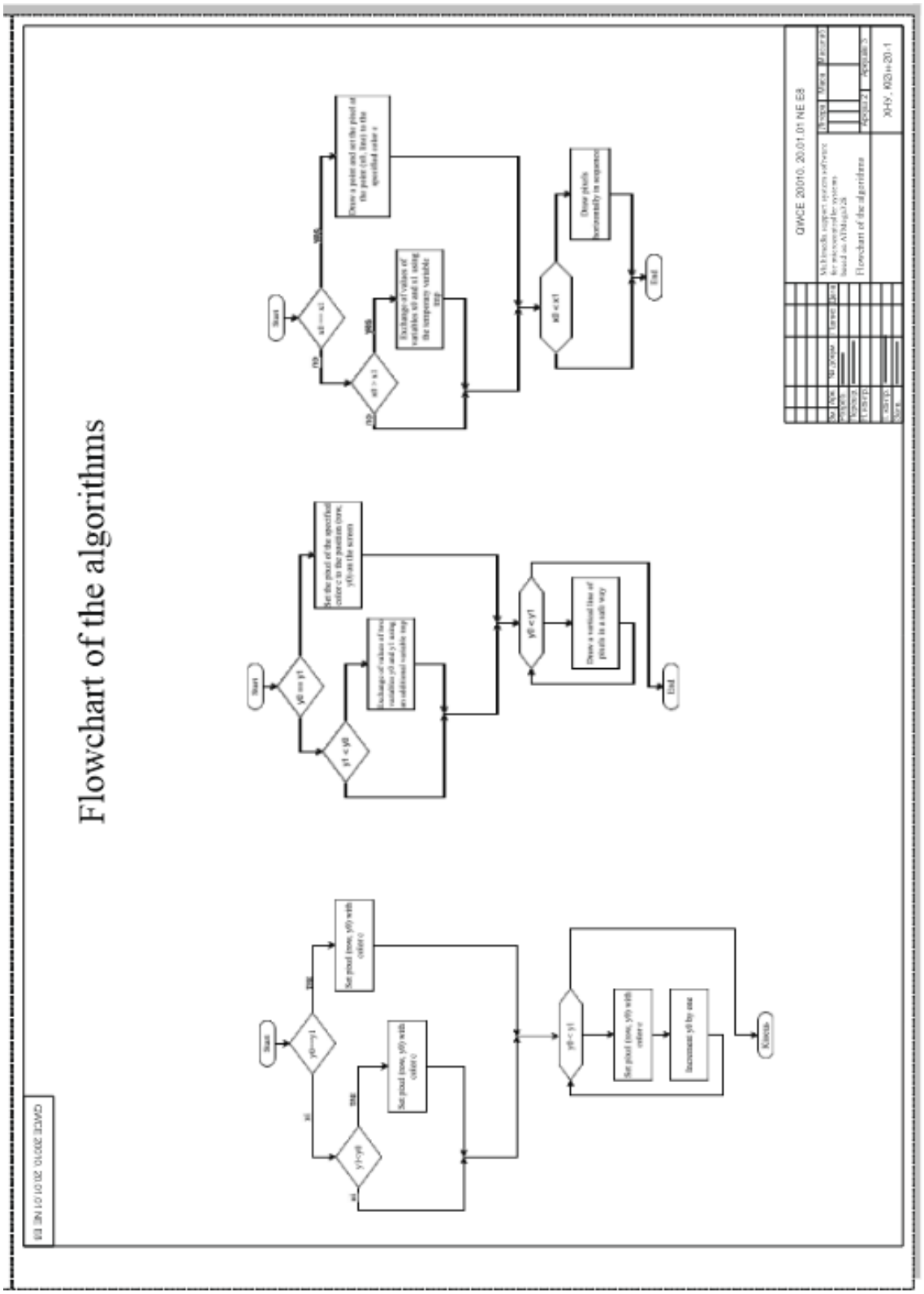
Copy of the drawing “Device scheme”



Annex B

(required)

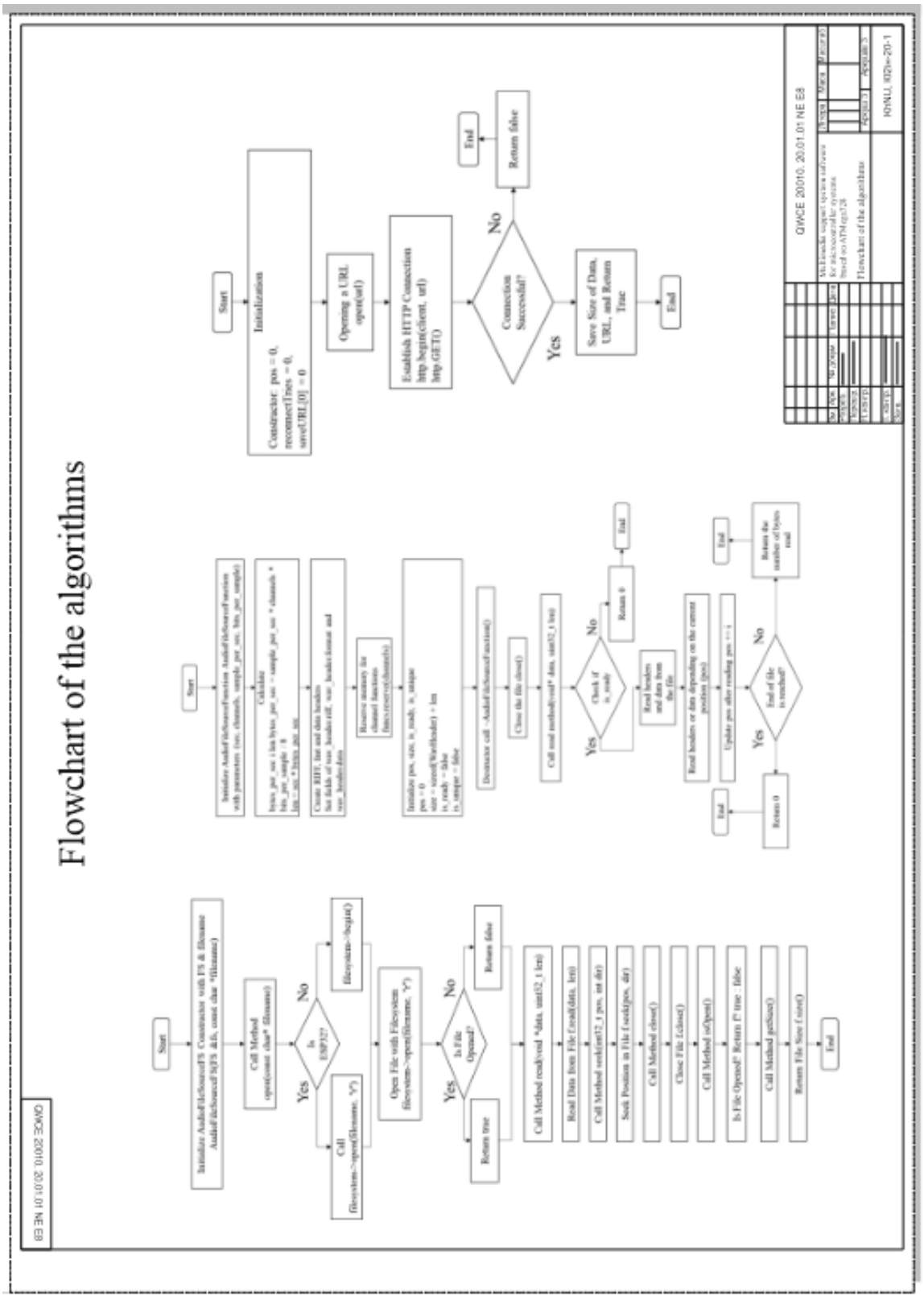
Copy of the drawing “Flowchart of the algorithms”



Annex C

(required)

Copy of the drawing «Flowchart of the algorithms»



UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016355016

Дата перевірки:
13.06.2024 07:16:26 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.06.2024 07:27:02 EEST

ID користувача:
100005591

Назва документа: Nana_Multimedia support system software for microcontroller systems based on ATMega328

Кількість сторінок: 72 **Кількість слів:** 13841 **Кількість символів:** 95311 **Розмір файлу:** 2.78 MB **ID файлу:** 1016159129

4.81%
Схожість

Найбільша схожість: 1.3% з Інтернет-джерелом (https://usermanual.wiki/Document/ATmega48A48PA88A88PA168A168P...

3.94% Джерела з Інтернету	141	Сторінка 74
0.93% Джерела з Бібліотеки	4	Сторінка 75

1.93% Цитат

Цитати	12	Сторінка 76
Посилання	1	Сторінка 76

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016355016

Дата перевірки:
13.06.2024 07:16:26 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.06.2024 07:27:02 EEST

ID користувача:
100005591

Назва документа: Nana_Multimedia support system software for microcontroller systems based on ATMega328

Кількість сторінок: 72 **Кількість слів:** 13841 **Кількість символів:** 95311 **Розмір файлу:** 2.78 MB **ID файлу:** 1016159129

4.81%
Схожість

Найбільша схожість: 1.3% з Інтернет-джерелом (https://usermanual.wiki/Document/ATmega48A48PA88A88PA168A168P...

3.94% Джерела з Інтернету	141	Сторінка 74
0.93% Джерела з Бібліотеки	4	Сторінка 75

1.93% Цитат

Цитати	12	Сторінка 76
Посилання	1	Сторінка 76

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	1

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 7.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 31%**

	Документ		Сумарний збіг по Базі Даних	
ID: 130075 Назва: БКР Multimedia support system software for microcontroller systems based on ATMega328 Додано в БД: 2024-06-13 Автора: Nana Ama Offul- Quaye Керівники: S.M. Lysenko Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	76980	631	5974 (8%)	77 (12%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
130074	Назва: БКР Системне програмне забезпечення підтримки мультимедіа для мікроконтролерних систем на базі ATMega328 Додано в БД: 2024-06-13 Автора: Нана Ама Керівники: С.М. Лисенко Консультанти: Опоненти:	5536 (7.0%)	81 (13.0%)

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 0.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 14%**

	Документ		Сумарний збіг по Базі Даних	
ID: 130074 Назва: БКР Системне програмне забезпечення підтримки мультимедіа для мікроконтролерних систем на базі ATMega328 Додано в БД: 2024-06-13 Автора: Нана Ама Керівники: С.М. Лисенко Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	94415	783	1145 (1%)	17 (2%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Оффул-Куайе Нана Ама

Тема: Системне програмне забезпечення підтримки мультимедіа для мікроконтролерних систем на базі ATMega328

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___67___

1. Короткий зміст роботи та прийнятих рішень: Метою дипломної роботи є розробка системного програмного забезпечення мультимедійної підтримки мікроконтролерних систем на базі ATMega328.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: Перший розділ розглядає аспекти звукових кодеків, висвітлюючи їхнє значення, функціональність і застосування в сучасних цифрових системах. Дослідження починається з фундаментальної концепції звукових кодеків, що забезпечує розуміння їхньої ролі в кодуванні та декодуванні аудіосигналів. Це закладає основу для розуміння важливості звукових кодеків для різних застосувань, особливо для стиснення даних.

Другий розділ присвячено аналізу функціональних можливостей мікроконтролера ATmega, який вирізняється своїми надійними системами тактової частоти та широкими можливостями, призначеними для підвищення функціональності та продуктивності в різноманітних додатках. Кожен аспект архітектури годинника, від Flash-годинника, що керує Flash-інтерфейсом, до асинхронного таймера, що забезпечує відлік часу в реальному часі навіть у сплячому режимі, демонструє ретельну увагу до деталей та оптимізацію для різних робочих сценаріїв.

Третій розділ розглядає деталі реалізації програмного забезпечення для обробки аудіофайлів у контексті вбудованих систем або мультимедійних проектів. За допомогою методичної документації та кодування коду пояснюється кожен компонент, задіяний в управлінні аудіофайлами. Основна увага приділяється різним заголовним файлам та їхнім відповідним реалізаціям, що відображають основні будівельні блоки, необхідні для надійної та універсальної системи обробки аудіофайлів.

4. Позитивні сторони роботи: висока практична цінність роботи.

5. Негативні сторони роботи: недостатня увага аналізу аналогам систем.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: добре, С

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Корецька Л.О., к.т.н., доцент, доцент кафедри Автоматизації, комп'ютерно-інтегрованої технологій та робототехніки

_____

_____

_____

"10" 06 2024 р. _____(підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Нана Ама

ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2ін-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

22 квітня 2024 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Системне програмне забезпечення підтримки мультимедіа для мікроконтролерних систем на базі ATMega328

Автор: Оффул-Куайе Нана Ама

Спеціальність: 123 – Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Лисенко Сергій Миколайович, д.т.н, професор

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення;

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 5.44% і адресується до 109 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ С.М. Лисенко

Гарант ОП _____ С.М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко