Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту)

Кафедра обчислювальної техніки
(повна назва кафедри)

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітньо-кваліфікаційний рівень)

на тему: Система перевірки провідних ліній зв'язку на базі Arduino

Виконав: студент 2 курсу, групи КІ –18м
спеціальності:
123 «Комп'ютерна інженерія»
(шифр і назва напряму підготовки)

Піщенко Д. В.
(прізвище та ініціали)

Керівник: к.т.н., доц. Снігур А.В.
(прізвище та ініціали)

Рецензент: к.т.н, доц. Дудатьєв А.В.
(прізвище та ініціали)

м. Вінниця – 2020 рік

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет_Інформаційних технологій та комп'ютерної інженерії_____
Кафедра_Обчислювальної техніки_____
Освітньо-кваліфікаційний рівень_магістр_____
Спеціальність_123 «Комп'ютерні системи та мережі»_____
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри_____
д.т.н., професор Мартинюк Т. Б.
"____" _____20___ року

З А В Д А Н Н Я

на магістерську кваліфікаційну роботу

_____Піщенку Денису Валерійовичу_____
(прізвище, ім'я, по батькові)

1.Тема магістерської кваліфікаційної роботи: Система перевірки провідних ліній зв'язку на базі Arduino_____
керівник магістерської кваліфікаційної роботи: Снігур Анатолій Васильович, к. т. н., доцент кафедри ОТ._____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «06»березня 2020 року №76
2. Строк подання студентом роботи_____
3. Вихідні дані до роботи_технічні характеристики систем та пристроїв перевірки провідних ліній зв'язку, технічний опис платформ Arduino, технічний опис мікроконтролерів Atmel, середовище розробки ПЗ Arduino IDE.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_ Вступ, техніко-економічне обґрунтування доцільності розробки, суть технічної проблеми та підходи щодо вирішення, огляд існуючих систем перевірки провідних мереж, огляд і аналіз існуючих підходів побудови мікропроцесорних систем, огляд і аналіз існуючих мікропроцесорних платформ Arduino, опис середовища програмування Arduino IDE, опис основних функцій Arduino IDE, проектування мікропроцесорної системи перевірки провідних ліній зв'язку, принцип роботи центрального мікроконтролера, розробка принципової схеми, розробка алгоритму роботи, розробка та відлагодження програмного забезпечення, принцип роботи та рекомендації щодо використання
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Електрична принципова схема.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	к. т. н., доцент Снігур Анатолій Васильович		
4	к. е. н., доцент Бальзан Марина Володимирівна		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Пошук та огляд інформаційних джерел	11.02.19р.	виконано
2	Огляд і аналіз методів перевірки провідних ліній зв'язку	17.02.19р.	виконано
3	Дослідження способів побудови мікропроцесорних систем	03.03.19р.	виконано
4	Огляд основних функції та архітектури платформ	17.03.19р.	виконано
5	Розробка електричної принципової схеми	31.03.19р.	виконано
6	Розробка та відлагодження програмного забезпечення	14.04.19р.	виконано
7	Економічна частина	28.04.19р.	виконано
8	Оформлення пояснювальної записки і презентації	12.05.19р.	виконано
9	Попередній захист	25.05.19р.	виконано

Студент _____ Піщенко Д.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Снігур А.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Магістерську кваліфікаційну роботу присвячено системі перевірки провідних ліній зв'язку на базі Arduino. Здійснено огляд існуючих рішень, визначено їх переваги та недоліки. Обрано мікропроцесорну платформу Arduino Nano на базі мікроконтролера ATMega168, на основі якої побудовано систему. Детально описано вузли, що будуть застосовані для реалізації системи. Створено блок-схему алгоритму роботи та програмне забезпечення мовою програмування С++. Проведено економічний розрахунок доцільності розробки.

SUMMARY

The master's thesis is devoted to the system of testing of leading communication lines based on Arduino. An overview of existing solutions, their advantages and disadvantages. The Arduino Nano microprocessor platform based on the ATMega168 microcontroller was chosen, on the basis of which the system was built. The nodes that will be used to implement the system are described in detail. A block diagram of the algorithm and software in the C ++ programming language has been created. The economic calculation of expediency of development is carried out.

ЗМІСТ

ВСТУП

Arduino – це платформа для розробки пристроїв на базі мікроконтролера, на простій і зрозумілій мові програмування в інтегрованому середовищі Arduino. Важко навіть перерахувати все, на що здатна платформа Arduino, тому що можливості обмежені тільки нашою уявою. Основою системи є мікроконтролери, які стали початком розвитку мікропроцесорної техніки. Наявність в одному корпусі більшості системних пристроїв зробило мікроконтролер подібним звичайному комп'ютеру. У вітчизняній літературі вони навіть називалися однокристальними мікро ЕОМ. Відповідно і бажання використовувати мікроконтролери як звичайні комп'ютери з'явилося практично з їх появою. Але бажання це стримувалося багатьма факторами. Наприклад, щоб зібрати пристрій на мікроконтролері, необхідно знати основи схемотехніки, пристрій і роботу конкретного процесора, вміти програмувати на асемблері і виготовляти електронну техніку. Будуть потрібні також програматори, відладчики і інші допоміжні пристрої. В результаті без величезного обсягу знань і дорогого обладнання не обійтися. Така ситуація довго не дозволяла багатьом любителям використовувати мікроконтролери в своїх проектах. Зараз, з появою пристроїв, що дають можливість працювати з мікроконтролерами без наявності серйозної матеріальної бази і знання багатьох предметів, все змінилося. Прикладом такого пристрою може служити проект Arduino італійських розробників.

Arduino і його клони являють собою набори, до складу готового електронного блоку і програмного забезпечення. Електронний блок тут – це друкована плата з встановленим мікроконтролером і мінімумом елементів, необхідних для його роботи. Фактично електронний блок Arduino є аналогом материнської плати сучасного комп'ютера. На ньому є роз'єми для підключення зовнішніх пристроїв, а також роз'єм для зв'язку з комп'ютером, за яким і здійснюється програмування мікроконтролера. Особливості використовуваних мікроконтролерів ATmega фірми Atmel дозволяють виробляти програмування

без застосування спеціальних програматорів. Все, що потрібно для створення нового електронного пристрою це плата Arduino, кабель зв'язку та комп'ютер.

Другою частиною проекту Arduino є програмне забезпечення для створення керуючих програм. Воно об'єднало в собі найпростішу середовище у розробки і мову програмування, що представляє собою варіант мови C / C ++ для мікроконтролерів. У нього додані елементи, що дозволяють створювати програми без вивчення апаратної частини. Так що для роботи з Arduino практично досить знання тільки основ програмування на C / C ++. Створено для Arduino і безліч бібліотек, що містять код, який працює з різними пристроями.

Користувач сучасного комп'ютера не замислюється про функціонування окремих частин ПК. Він просто запускає потрібні програми і працює з ними.

Точно так само і Arduino дозволяє користувачеві зосередитися на розробці проектів, а не на вивченні пристрою і принципів функціонування окремих елементів. Немає потреби і в створенні закінчених плат і модулів. Розробник може використовувати готові плати розширення або просто безпосередньо підключити до Arduino необхідні елементи. Всі інші зусилля будуть спрямовані на розробку і налагодження керуючої програми на мові високого рівня. В результаті доступ до розробки мікропроцесорних пристроїв отримали не тільки професіонали, але і просто любителі щось зробити своїми руками.

Наявність готових модулів і бібліотек програм дозволяє непрофесіоналам в електроніці створювати готові працюючі пристрої для вирішення своїх завдань. А варіанти використання Arduino обмежені тільки можливостями мікроконтролера і наявного варіанту плати, ну і, звичайно, фантазією розробника.

Найперші Arduino базувалися він на мікроконтролері ATmega328 і складалися з дешевих і доступних компонентів. Але головне завдання полягало в тому, щоб гарантувати роботу пристрою за принципом plug-and-play, щоб користувач, діставши плату з коробки і підключивши до комп'ютера, міг негайно приступити до роботи. Перший прототип плати був зроблений в 2005

році, вона мала найпростіший дизайн і ще не називалася Arduino. Трохи пізніше Массімо Банц придумав назвати її так – по імені належного йому бару, розташованого в місті Іврея. Бренд "Arduino" без будь-якої реклами і залучення коштів маркетингу швидко придбав високу популярність в Інтернеті. З початку поширення продано понад 250 тис. Комплектів Arduino, і це не враховуючи безлічі клонів. У світі налічується понад двісті дистриб'юторів продукції Arduino — від великих фірм, таких як SparkFun Electronics, до дрібних компаній, що працюють на місцевий ринок. На сьогодні платформа Arduino представлена не однією платою, а цілим їх сімейством. На додаток до оригінального проекту, званому Arduino Uno, нові моделі, які мають на платі більш потужні засоби, звуться Arduino Mega, компактні моделі — Arduino Nano, плати в водонепроникному корпусі — LilyPad Arduino, а нова плата з 32-розрядним процесором Cortex-M3 ARM — Arduino Due.

Своїм успіхом проект Arduino зобов'язаний існуванню до нього мови Processing і платформі Wiring. Від цих проектів Arduino успадкував одну сильну рису — зручне для користувача середовище розробки. До появи Arduino програмування мікроконтролерів вимагало складного і рутинного попереднього навчання. А з Arduino навіть ті, хто не має досвіду роботи з електронними пристроями, тепер можуть долучитися до світу електроніки. Початківцям вже не доводиться витрачати багато часу на вивчення супутнього матеріалу — вони можуть швидко розробити прототип, який буде повноцінно робочим.

П'ятдесят років тому, щоб написати програмне забезпечення була потрібна команда людей в білих халатах, які знали все про електронні лампах. Тепер же, з появою Arduino, безліч людей отримали можливість створювати електронні пристрої самостійно.

Всі системи в тій чи іншій мірі мають зв’язок з мережею Інтернет, а зараз бурхливо розвивається напрямок «Інтернету речей». Зв'язок може здійснюватися через провідні та безпровідні мережі. І, як правило, безпровідні використовують в якості середовища передачі електромагнітні хвилі, то провідні, в якості ліній зв’язку – фізичне середовище провідників. Провідні

мережі можуть виходити з ладу під впливом різноманітних чинників, а тому гостро стоїть проблема їх ремонту і попередньої діагностики, що упередить поломки. Останнім часом все більша увага спеціалістів у напрямку діагностики та ремонту приділяється системам перевірки провідних ліній зв'язку.

Актуальність роботи у напрямку проектування систем перевірки провідних ліній зв'язку полягає у необхідності побудови надійних, повнофункціональних, адаптивних, надійних і дешевих систем діагностики з використанням існуючих знань та сучасних досягнень в напрямку електроніки та інформаційно-вимірювальних систем.

Об’єктом дослідження є процеси, що відбуваються у провідних лініях зв’язку.

Предметом дослідження є методи і засоби перевірки стану та характеристик провідних ліній зв’язку.

Мета полягає у реалізації системи перевірки провідних ліній зв’язку, використовуючи сучасні досягнення у напрямку мікропроцесорних систем, які дозволяють будувати повнофункціональні системи з можливістю адаптації функціональних можливостей під необхідні задачі.

Щоб досягнути мети слід вирішити такі задачі:

— здійснити огляд і аналіз методів перевірки провідних ліній зв’язку;

— здійснити огляд і аналіз мікропроцесорних систем Arduino;

— визначити переваги та недоліки існуючих, визначити напрямки дослідження;

— розробити структурну та принципову схеми;

— здійснити вибір електронних елементів;

— створити програмне забезпечення для роботи системи;

— здійснити техніко-економічне обґрунтування розробки та економічні розрахунки, у порівняні розробленої системи з існуючими аналогами з урахуванням переваг та недоліків.

Наукова новизна полягає у новому підході побудови системи перевірки провідних ліній зв'язку на базі Arduino, що дозволить сварити універсальний пристрій.

Практичне значення одержаних у результатів наукового дослідження полягає у розробці системи перевірки провідних ліній зв'язку на базі Arduino з меншою ціною, ніж у промислових аналогів.

Особистий внесок здобувача. Основні положення й результати магістерської роботи отримані автором самостійно та пов’язані з його професійною діяльністю.

1 ТЕХНІКО–ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ ДОЦІЛЬНОСТІ РОЗРОБКИ

1.1 Суть технічної проблеми та підходи щодо вирішення

Тестування ліній зв'язку (ЛЗ) має на увазі застосування відповідних методів і приладів. Два основні підходи – тестування на постійному і змінному струмі. У свою чергу, тестування на змінному струмі виконується двома способами – шляхом вимірювання падаючої хвилі або вимірювання відбитої хвилі (метод рефлектометрії).

Вимірювання на постійному струмі і вимірювання падаючої хвилі використовуються для визначення первинних і вторинних параметрів лінії. Обидва методи можуть бути реалізовані як шляхом безпосереднього вимірювання хвилі, так і з застосуванням методу порівняння, окремим випадком якого є бруківці метод. Основна перевага методу порівняння – його висока точність в широкому діапазоні вимірюваних значень.

Крім названої існують і інші класифікації методів тестування. Так, всю їх сукупність можна представити у вигляді великих груп, одна з яких вимагає обов'язкового закриття діючої системи зв'язку на час вимірювання, а інша може виконуватися в працюючій системі. Більш коротко, як спосіб із закриттям зв'язку і спосіб без закриття зв'язку.

Сучасна концепція тестування мереж зв'язку спирається на модель взаємодії відкритих систем OSI, відповідно до якої всі вимірювальні прилади для тестування мереж зв'язку поділяються на дві категорії:

— аналізатори фізичного рівня (перший рівень OSI);

— аналізатори більш високих рівнів (з другого по сьомий).

До аналізатора фізичного рівня відносяться універсальні вимірювальні пристрої, кабельні тестери, рефлектометри для металевих і оптичних кабелів, осцилографи, вимірювачі рівня сигналу і аналізатори спектра. Інша група аналізаторів другого-сьомого рівнів моделі OSI вимірює параметри циклів і пакетів, перевіряє цілісність даних, сеанси зв'язку, перетворення даних і додатки. Це можуть бути кишенькові тестери, аналізатори протоколів у вигляді

універсальних приладів зі спеціальними модулями для вирішення різних завдань або пакети програм для використання в комплексах тестування і для управління мережевих вузлів.

Тестування кабельних ліній зв'язку здійснюється лише за допомогою аналізаторів фізичного рівня. Надалі саме їх розглянемо більш докладно.

За кілька останніх десятиліть ринок аналізаторів фізичного рівня для тестування симетричних ліній зазнав революційні зміни. Причиною стала поява технологій xDSL та структурованих кабельних систем. Прилади цієї групи дозволяють оцінити такі параметри лінії зв'язку, як її довжина, опір, загасання, коефіцієнт відображення, перехідне загасання між кручеными парами мідних кабелів і ін. Вони застосовуються і для локації електричного стану кабельної лінії (визначення неоднорідностей, паралельних відводів, місць пошкодження лінії і т.д.).

У «аналогову епоху» прилади призначалися для вирішення проблем традиційних телефонних мереж з їх орієнтацією на діапазон звукових частот. Сучасні прилади для тестування симетричних ліній працюють в діапазоні частот до декількох сотень мегагерц. На додаток до групи низькочастотних приладів сформувалися дві нові. Одна з них орієнтована на тестування абонентських ліній з підтримкою xDSL, інша — на тестування СКС.

Ціна широкосмугових приладів значно вище, тому дешеві пристрої низькочастотного діапазону з ринку не зникли. Більш того, завдяки ряду еволюцій, область їх застосування істотно розширилася. Наприклад, реалізація нових методів тестування абонентських ліній підвищила якість діагностики, а автоматизація процесу вимірів полегшила роботу персоналу. В результаті низькочастотні прилади нового покоління забезпечують діагностику і локалізацію здебільшого дефектів кабельних ліній зв'язку та застосовуються до того ж для тестування абонентських ліній при розгортанні xDSL. Ще один приклад — група простих приладів з набором допоміжних функцій для початкового тестування СКС.

Подальший матеріал з діагностики кабельних ліній зв'язку буде присвячений детальному розгляду приладів і методів тестування симетричних ліній їх на основі. Деякі з пристроїв будуть лише згадані, параметри інших — детально описані.

Мультиметри служать для вимірювання параметрів лінії по постійному і змінному струму (напруга станційної батареї, опір шлейфа абонентської лінії та ін.), приклад якого показано на рисунку 1.1.

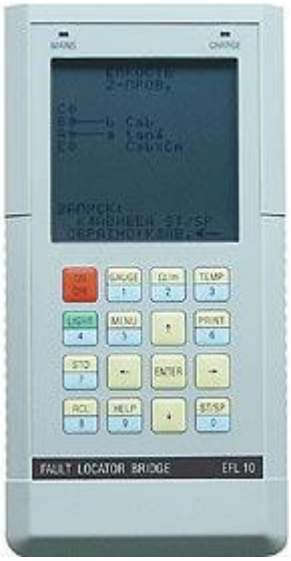


Рисунок 1.1 – Мультиметр для вимірювання параметрів лінії

Мости постійного і змінного струму доповнюють мультиметри, дозволяючи більш точно оцінювати первинні параметри лінії зв'язку.

Вимірювачі рівня сигналу становлять велику групу приладів, використовуваних при налаштуванні, експлуатації та усунення пошкоджень в системах передачі по металевих кабелях. З їх допомогою можна вимірювати загасання лінії, перехідне загасання, гармонійні перешкоди і шуми. Вимірювачі рівня працюють в селективному або широкосмуговим режимі. Селективні вимірювачі рівня дозволяють оцінювати рівні сигналу або шуму тільки в певній, досить вузької (100 Гц, 1 кГц, 3,1 кГц і т. Д.) Смузі частот. Завдяки цій властивості селективні вимірювачі здатні оцінювати дуже низькі рівні сигналів і перешкод.

Рисунок 1.2 – Зовнішній вигляд вимірювача рівня сигналу

Широкосмугові вимірювачі рівня застосовуються, як правило, для вимірювання широкосмугових перешкод (наприклад, теплових шумів регенераторів і підсилювачів). В принципі вони придатні і для вимірювання рівнів моночастотних сигналів, якщо ті значно перевищують рівень широкосмугового перешкоди. Важлива перевага селективних вимірників у порівнянні з широкосмуговими полягає також у тому, що вони дозволяють виробляти тестування діючої системи зв'язку.

Тестери коефіцієнтів бітових помилок BER — основний інструмент для оцінки лінії цифрового зв'язку як при її початковому налаштуванні, так і в процесі експлуатації. В останньому випадку роботу системи зв'язку потрібно призупинити. Принцип дії приладу заснований на використанні псевдовипадкових послідовностей. Алгоритми функціонування тестерів BER спираються на рекомендації ITU-T — G.821, G.826, V.53 і M.2100. Тестери помилок дозволяють оцінювати бітові і блокові помилки, а також помилки в секундних інтервалах, включаючи частку таких інтервалів без помилок EFS, з помилками ES і з численними помилками SES.

Результати тестування помилок зазвичай представляють у вигляді числових значень або гістограми. Деякі аналізатори протоколів високого рівня мають вбудовані функції тестування помилок. На відміну від вимірників рівня, тестери помилок вимагають обов'язкового закриття системи зв'язку.

Рисунок 1.3 – Тестер коефіцієнта бітових помилок BER

Рефлектометри в часовій області,TDR, дозволяють оцінити характерні точки лінії зв'язку, включаючи неоднорідності, пошкодження і т. д.

Через складну природи пошкоджень кручених пар окреме тестування в тимчасовій або частотній області не дозволяє вичерпно ідентифікувати причину пошкодження і його місце розташування.

До переваг рефлектометра відноситься той факт, що вимірювання можуть проводитися тільки з одного кінця. Однак таке підключення не завжди дозволяє точно визначити причину відображень (особливо в разі множинних дефектів). Наприклад, рефлектометр не може відрізнити відображення внаслідок присутності пупиновських котушки від відображення через обрив кручений пари. Маючи низький вихідний опір, близьке до 100 Ом для вузьких випробувальних імпульсів, рефлектометр не в змозі надійно виявляти відображення від місць пошкодження з опором близько 1000 Ом і більше.

Крім того, тестування в частотної області має істотно більший набір функцій, включаючи вимір первинних параметрів — опору, витоку і ємності, а також параметрів передачі, впливу, шумів, асиметрії та ін.

Тому розробники вимірювальних приладів все частіше замислюються про необхідність об'єднання в одному пристрої функцій тестування в тимчасовій і частотній області. Сьогодні подібні комплексні прилади вже існують і дозволяють домогтися більш високої точності діагностики при одночасному скороченні витрат часу.

Рисунок 1.4 – Зовнішній вигляд рефлектометра TDR

Осцилографи і аналізатори спектру зазвичай використовуються при ідентифікації складних пошкоджень, коли потрібне точне визначення форми сигналу або його частотного складу. Наприклад, великий коефіцієнт помилок BER може бути викликаний безліччю причин: дефектної вихідний щаблем передавача, занадто великими значеннями потужності шуму або тремтіння через включення електричного двигуна або перехідними впливами з боку систем передачі, що працюють по тому ж кабелю. Осцилограф надає єдину можливість для вичерпної деталізації параметрів сигналу, включаючи його форму, частоту, час наростання і спаду.

Логічні аналізатори використовуються для запису сигналів синхронізації. Вони схожі на осцилографи з додатковими функціями тестування цифрових сигналів, контролюють одночасно кілька синхросигналов і забезпечені можливістю автоматичного запуску при певному стані контрольованих сигналів.

1.2 Огляд існуючих систем перевірки провідних мереж

Найчастіше для перевірки стану кабельних ліній або кабелів використовують кабельні тестери. Вони являють собою електронні пристрої,

що складаються, як правило, з двох частин. Дані прилади бувають різними, і деякі з них дозволяють визначати характеристики кабельних ліній або кабелів. Сьогодні на ринку зустрічаються кабельні тестери трьох класів:

— для базової перевірки кабелів;

— для кваліфікації кабельних систем;

— для сертифікації кабельних систем.

За типом тестованого кабелю прилади поділяються на:

— тестери для оптичних кабелів;

— тестери для коаксіальних кабелів, телефонних кабелів, кручених пар.

Останні відрізняються універсальністю, з їх допомогою можна тестувати абсолютно різні типи електричних кабелів, широко застосовуваних сьогодні.

Найважливішими параметрами, які можна виміряти за допомогою кабельного тестера є:

— довжина кабеля;

— схема розведення провідників в кабелі;

— величина загасання;

— рівень перехресних наведень на ближньому кінці кабельної лінії — NEXT;

— величина опору за постійним струмом по мідному шлейфу;

— рівень зворотних втрат — Return loss.

У найпростішому вигляді кабельний тестер зі світлодіодною індикацією здатний показати мінімальне відповідність характеристик кабелю заданим вимогам. Даний тип тестера дозволяє більш ефективно виконувати монтаж кабелю або простої проводки , і відразу виявляти несправності, якщо такі матимуть місце.

Безумовно, функціонал простих тестерів не дозволить виміряти відстань до місця пошкодження, і не виявить розщеплені пари. Однак перевірити, чи правильно з'єднані дроти, і виявити типові механічні пошкодження (замикання

або обрив) найпростіший тестер зможе. Про перевірку оптичних кабелів тут говорити, звичайно, не доводиться.

Як приклад простого кабельного тестера можна привести тестер кабелю RJ-45 + BNC (HT-C003) (TL-5248) від REXANT, зовнішній вигляд якого показано на рисунку 1.1.

Рисунок 1.5 – Тестер кабеля RJ-45 + BNC (HT-C003) (TL-5248)

Даний прилад підійде для тестування кабелів на основі кручених пар, а також коаксіальних кабелів. Він включає в себе два блоки, один з яких — передавач, другий — приймач. Передавач і приймач підключаються до кінців кабелю за допомогою роз'ємів BNC або RJ-45.

Прилад перевіряє, чи правильно виконано обтиск, чи є обрив, чи немає короткого замикання в лінії, чи цілий екран, якщо мова про перевірку екранованої кручений пари. На обох блоках є індикація, що показує результат тесту. Матеріал корпусу — ударостійкий пластик.

Більш складні тестери мають розширений функціонал — в них є генератори тональних сигналів, що дозволяє виявляти розщеплені пари.

Сучасні тестери оснащені дисплеями здатні знаходити всі види помилок в схемах розводки. Тут є можливість виявити і розщеплені пари, і дізнатися

довжину кабельної лінії, а також виміряти відстань до короткого замикання або до обриву, і навіть визначити якого типу розетка встановлена на іншій стороні лінії (мережева або телефонна).

Прилади для кваліфікації кабелів (кваліфікуючі тестери) спочатку почала випускати фірма Fluke Networks. Тестери цього класу можуть визначати швидкісні можливості кабелю і кабельних систем, чи зможуть системи працювати на більш високих швидкостях. Кожен прилад кваліфікуючої класу має функції вимірювання параметрів Return loss і NEXT, а також загасання в кабелях.

Як бачимо, прилади цього класу можуть не тільки «продзвонити» кабель, але і значно більше. Для фахівців IT-галузі дані прилади стануть без перебільшення незамінними помічниками, без необхідності купувати дорогий кабельний аналізатор.

Сьогодні на ринку країни широко представлені кілька виробників кабельних тестерів кваліфікуючої класу, це: Ideal Industries, JDSU і Fluke Networks.

Як простий приклад кваліфікованого тестера можна привести NCT-3, — портативний цифровий LAN-Тестер для RG-45, RG-58, RJ-12/11 від Gembird.

Даний прилад легко виявить проблеми в мережевих кабелях категорій 5e та 6e, а також в коаксіальних або телефонних лініях. Коротке замикання, розімкнення, кросовер з'єднання, замикання провідників — все це може виявити тестер.

Прилад вимірює довжину проводу і встановить відстань до обриву. Швидка діагностика проблем локальних мереж — аж ніяк не проблема для майстра, що має в своєму арсеналі даний або подібний прилад. Точність вимірювань досягає 97% завдяки калібруванню.

Перший же прилад сертифікованого класу побачив світ в 1993 році, і був випущений американською компанією Microtest, яку пізніше (в 2001 році) купила Fluke Networks.

Рисунок 1.6 – Тестер кабеля NCT-3 Gembird

Головне завдання цих тестерів — перевірка, наскільки та чи інша кабельна система відповідає міжнародним стандартам, бо етап сертифікації невід'ємний при проектуванні і монтажі будь-якої структурованої кабельної системи. Категорії і класи кабелів визначаються галузевими стандартами організацій TIA і ISO.

Сертифікуючий кабельний тестер повністю дозволяє перевірити кабель, і виводить на екран частотні взаємозалежності різних параметрів, важливих для TIA і ISO. Інформація виводиться у вигляді графіків на екран тестера, і фахівець з вигляду цих графіків розуміє, що і де потрібно в лінії поліпшити. Якщо користувач фахівцем не є, то тестер все одно вкаже, який результат перевірки — Pass або Fail.

Pass — кабель повністю справний, стан відмінний, всі тести пройдені успішно;

Fail — мають місце несправності.

Прилад для сертифікації зазвичай має також можливість роздрукувати дані вимірювань в стандартизованої формі, щоб в подальшому по роздруківці можна було б відповідно до законодавства затвердити введення в експлуатацію

обмеренной лінії зв'язку. Таке тестування універсально з точки зору обивателя, бо немає прив'язки до мережевої технології.

Припустимо, кабель 6 категорії здатний передавати дані зі швидкістю від 10 Мбіт / с до 10 Гбіт / с, а кабель категорії 5e — від 10 Мбіт / с до 1 Гбіт / с. Якщо результат тесту виходить «Fail», то потрібно діагностика. Кабельний тестер, що володіє функцією діагностики, покаже і дозволить зрозуміти (по тестах NEXT і Return Loss), чи проблема в роз'ємах або безпосередньо в кабелі.

Прикладом приладу з дуже широкими функціональними можливостями може служити кабельний тестер Microscanner2 (FLN-MS2-100) від Fluke.

На великому РК-дисплеї відображаються схеми з'єднань, а ще ідентифікатор кабелю, його довжина, а також відстань до місця наявності несправності.

Рисунок 1.7 – Кабельний тестер Microscanner2 (FLN-MS2-100)

Тестер може перевіряти всі основні типи провідників, включаючи RJ45, RJ11 і коаксіальні без додаткових адаптерів. Є генератор тону IntelliTone, що дозволяє виявити місце розташування кабелів або пар провідників за допомогою подачі аналогових і цифрових тонів. Функція визначення сервісів

VDV дозволяє розпізнавати сучасні сервіси комунікацій, включаючи POTS, 10/100/1000 Ethernet і PoE.

Тестери класу MicroScanner2 кардинально спрощують процес перевірки кабелів для передачі голосу, даних і відео, а високоякісні кабельні системи стають тепер ефективними як ніколи.

Безліч завдань діагностики вирішується швидко: чи є в телефонних мережах напруга; яка його полярність; чи присутній комутатор Ethernet на протилежному кінці; чи доступно PoE. MicroScanner2 одночасно розглядає всі ці фактори і пропонує фахівцеві якісні візуальні засоби для перевірки найбільш поширених сьогодні сервісів передачі відео, даних і голосу.

Проте вагомим недоліком, який є ключовим при виборі системи перевірки ліній зв’язку, є вартість.

2 ОГЛЯД І АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ПОБУДОВИ МІКРОПРОЦЕСОРНИХ СИСТЕМ

2.1 Огляд і аналіз існуючих мікропроцесорних платформ Arduino

Відповідно до завдання систему перевірки провідних ліній зв’язку необхідно реалізувати на мікропроцесорній платформі Arduino.

Перша і головна перевага Arduino, що зумовила їх широке поширення, полягає в тому, що один компактний електронний пристрій може замінити десятки і сотні електромеханічних реле. Друга перевага в тому, що функції логічних контролерів реалізуються не апаратно, а програмно, що дозволяє постійно адаптувати їх до роботи в нових умовах з мінімальними зусиллями і витратами.

Arduino відрізняються від традиційних неперепрограмованих пристроїв управління наступними перевагами: вони більш гнучкі, надійніше, мають менші габарити, можуть бути об'єднані в мережі з іншими пристроями і перенастроюватися по Інтернету, швидше виявляють помилки, витрачають менше електроенергії, вимагають менше витрат на зміну своїх функцій і структури і менш затратні на великих відрізках часу.

Застосування Arduino забезпечує високу надійність, просте тиражування та обслуговування систем управління, прискорює монтаж і налагодження обладнання, забезпечує можливість швидкого поновлення алгоритмів управління (в тому числі і на працюючому обладнанні).

У теперішній час існує велика кількість моделей-оригіналів та їх клонів. Найбільш вдалими моделями для вирішення задач побудови системи перевірки провідних ліній зв’язку варто розглянути Arduino Mega 2560 та Arduino Nano. Arduino Mega 2560 завдяки потужнішому мікроконтролеру, Arduino Nano — менші габаритні розміри та вартість.

2.1.1 Arduino Mega 2560

Arduino Mega 2560 — це нове сімейство мікроконтролерів Arduino для вирішення найрізноманітніших завдань автоматизації малого рівня. Це пристрій на основі мікроконтролера ATmega2560.

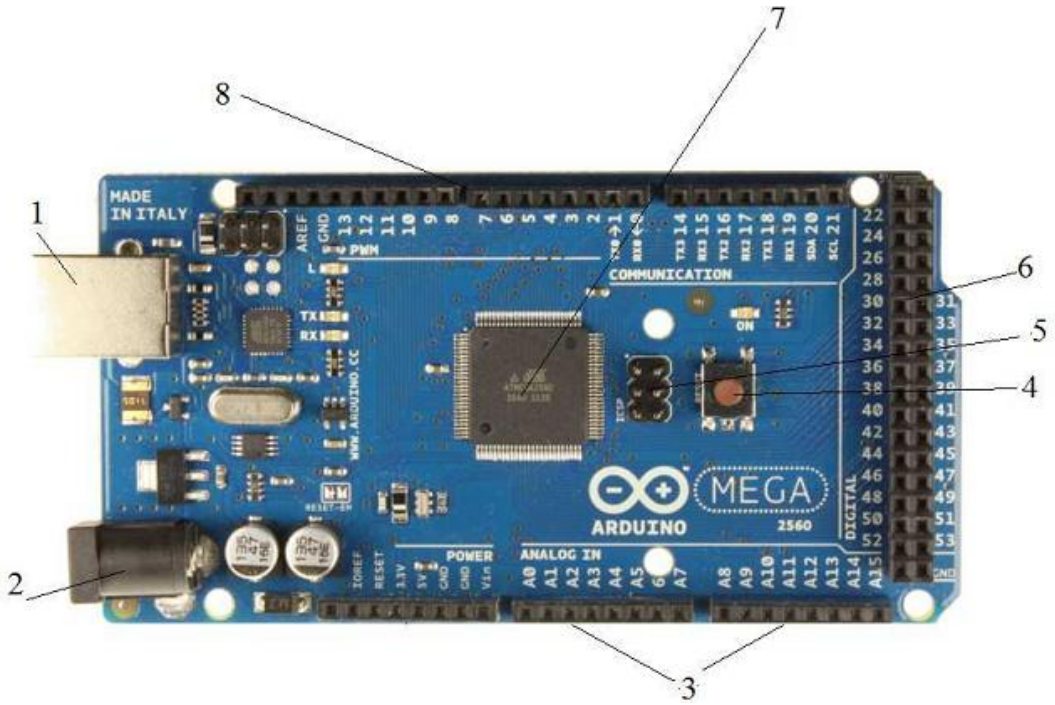


Рисунок 2.1 – Зовнішній вигляд Arduino Mega 2560

На рисунку позначено наступні складові Arduino Mega 2560:

— 1. USB роз'єм;

— 2. зовнішнє живлення;

— 3. 16 аналогових входів;

— 4. кнопка перезавантаження; 5-ISCP;

— 5. 54 цифрових входів/виходів;

— 6 .ATmega2560;

— 7. входи ШІМ;

— 8. Порти вводу/виводу.

Плата має 54 цифрових входів/виходів, 15 з яких використовуються як ШІМ, 16 аналогових входів, 4 апаратних послідовних входу для реалізації послідовних інтерфейсів UART, кварцовий резонатор з частотою 16МГц, USB

порт, роз'єм живлення, роз'єм ISCP для програмування в пристроїв по послідовному протоколу і кнопка скидання мікроконтролера. Для початку роботи з пристроєм досить просто подати живлення від AC / DC-адаптера або батарейки, або підключити його до комп'ютера за допомогою USB-кабелю. Arduino Mega сумісний з більшістю плат розширення, розроблених для Arduino Duemilanove і Diecimila.

Mega2560 має компактний розмір: довжиною 10,2 см і шириною 5,4 см з урахуванням USB роз'єму і роз'єму живлення, які виступають з корпусу плати. Плата може працювати від джерела живлення в діапазоні від 6 до 20 вольт. При меншій напрузі плата працює нестабільно. При більш високій напрузі плата починає нагріватися і може вийти з ладу.

Виводи живлення Arduino Mega 2560 перераховані нижче:

VIN. Напруга, що надходить в Arduino безпосередньо від зовнішнього джерела живлення (при відсутності подачі напруги на USB порт або іншого джерела). Через цей вивід можна як подавати зовнішнє живлення, так і споживати струм, коли пристрій живиться від зовнішнього адаптера.

5V. На цей вивід надходить напруга 5В від стабілізатора напруги на платі, поза незалежності від того, як живиться пристрій: від адаптера (7-12В), від USB (5В) або через вивід VIN (7-12В). Живити пристрій через виводи 5V або 3V3 не рекомендується, оскільки в цьому випадку не використовується стабілізатор напруги, що може привести до виходу плати з ладу.

3V3. 3.3В, що надходять від стабілізатора напруги на платі.

Максимальний струм, споживаний від цього виводу, становить 50 мА.

GND. Вивід землі.

IOREF. Цей вивід надає платам розширення інформацію про робочій напрузі мікроконтролера Arduino. Залежно від напруги, зчитаного з виведення IOREF, плата розширення може переключитися на відповідне джерело живлення або задіяти перетворювачі рівнів, що дозволить їй працювати як з 5В, так і з 3.3В-пристроями.

Arduino Mega 2560 має 225 Кб флеш-пам'ять, яка використовується для зберігання програмного коду, 8Кб з яких використовуються для завантаження і 4 КБ незалежній пам'яті служить для роботи з бібліотекою EEPROM.

Кожен з 54 цифрових пінів на Arduino Mega може працювати в режимі входу або виходу, використовуючи функції pinMode, digitalWrite і digitalRead. Виходи працюють на 5 В. Кожен пін може віддати або прийняти максимум 40 мА і має внутрішній підтягаючий резистор 20-50 кОм (відключений за замовчуванням). Плюс до цього, деякі виводи мають спеціальні функції:

Послідовний інтерфейс Serial: 0 (RX) і 1 (TX); Serial 1: 19 (RX) і 18 (TX); Serial 2: 17 (RX) і 16 (TX); Serial 3: 15 (RX) і 14 (TX). Дані виводи використовуються для отримання (RX) і передачі (TX) даних по послідовному інтерфейсу. 0 і 1 підключені до виводів мікросхеми ATmega16U2, яка виконує роль перетворювача USB-to-UART ATmega16U2.

ШІМ виводи 2-13 і 44-46 за допомогою функції analogWrite можуть виводити 8-бітові аналогові значення в вигляді ШІМ сигналу.

Інтерфейс SPI: виводи 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Виводи здійснюють передачу зв'язку по інтерфейсу SPI, використовуючи бібліотеку SPI. Піни SPI можуть бути виведені на ISCP, сумісний з Arduino Uno, Duemilanove і Diecimila.

LED 13. Це вбудований в плату світлодіод, який включений на 13 вивыд. При подачі сигналу HIGH, світлодіод загоряється, а під час надходження LOW—вимикається. Служить сигналом про успішну завантаження програми в плату.

Виводи 20 (SDA) і 21 (SCL) дозволяють здійснювати передачу зв'язку по інтерфейсу TWI.

У Mega2560 є 16 аналогових входів, кожен з яких можна представити в аналогову напругу у вигляді 10-бітного числа.

Вивід AREF опорного напруги для аналогових входів

Вивід RESET подає низький рівень сигналу для перезавантаження мікроконтролера.

Arduino Mega 2560 надає ряд можливостей для здійснення зв'язку з комп'ютером, ще одним Arduino або іншими мікроконтролерами. У ATmega2560 є чотири апаратних приймально-передавача UART для реалізації послідовних інтерфейсів (с логічним рівнем TTL 5В). Мікроконтролер ATmega16U2 забезпечує зв'язок одного з приймачів з USB-портом комп'ютера, і при підключенні до ПК дозволяє Arduino визначатися як віртуальний COM-порт. У пакет програмного забезпечення Arduino входить спеціальна програма SerialMonitor, що дозволяє зчитувати і відправляти на Arduino прості текстові дані. При передачі даних через мікросхему ATmega8U2 / ATmega16U2 під час USB-з'єднання з комп'ютером, на платі будуть мигати світлодіоди RX і TX. (При послідовній передачі даних за допомогою виводів 0 і 1, без використання USB-перетворювача, дані світлодіоди задіюються).

Бібліотека SoftwareSerial дозволяє працювати з підключенням по послідовної зв'язку на будь-яких цифрових виводах Mega2560.

У мікроконтролері ATmega2560 також реалізована апаратна підтримка послідовних інтерфейсів TWI і SPI. У програмне забезпечення Arduino входить бібліотека Wire, що дозволяє спростити роботу з шиною TWI. Для роботи з інтерфейсом SPI використовується бібліотека SPI.

Для роботи з Arduino Mega, необхідно використовувати програмне забезпечення Arduino IDE.

Мікроконтролер ATmega2560 на платі Arduino Mega поставляється з прошитим загрузчиком, який дозволяє завантажувати новий код в мікроконтролер без використання зовнішнього апаратного програматора. Завантажувач використовує оригінальний протокол ST. Також є можливість не використовувати завантажувач і програмувати мікроконтролер через виводи блоку ISCP, використовуючи Arduino ISP або аналогічний.

Вихідний код прошивки ATmega16U2 доступний для скачування в репозиторії Arduino. ATmega16U2 / 8U2 завантажується, використовуючи завантажувач DFU, для активації якого необхідно:

На платах версії R1: замкнути перемичку на звороті плати (біля зображення карти Італії), після перезавантажуємо 8U2.

На платах версій R2 і вище для спрощення переходу в режим DFU присутній резистор, що підтягує до землі лінію HWB мікроконтролера 8U2 / 16U2. Після переходу в DFU-режим для завантаження нової прошивки можна використовувати програмне забезпечення Atmel's FLIP (для Windows).

2.1.2 Arduino Nano

Платформа Nano, побудована на мікроконтролері ATmega328 (Arduino Nano 3.0), має невеликі розміри і може використовуватися в лабораторних роботах і місцях обмеженого простору. Вона має схожу з Arduino Duemilanove функціональність, проте відрізняється складанням. Відмінність полягає у відсутності силового роз'єму постійного струму і роботі через кабель Mini-B USB. Nano розроблена і продається компанією Gravitech.

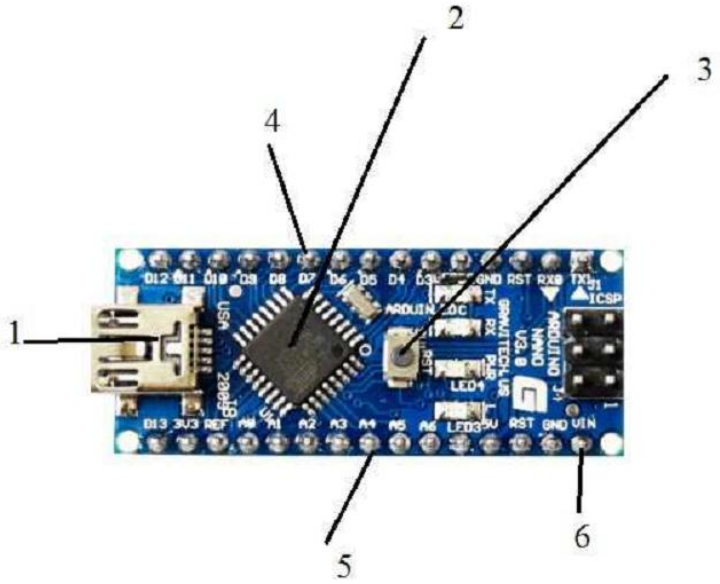


Рисунок 2.2 – Зовнішній вигляд Arduino Nano

На рисунку позначено наступні складові Arduino Mega 2560:

— 1. мікро USB роз'єм;

— 2. ATmega328;

— 3. кнопка перезавантаження;

— 4 .14 цифрових входу / виходу, 6 з яких можуть використовуватися як вихід ШІМ;

— 5. 8 аналогових входів;

— 6. Зовнішнє живлення VIN;

Arduino Nano може отримувати живлення через підключення Mini-B USB, або від нерегульованого 6-20 В (вивід 30), або регульованого 5 В (вивід 27), зовнішнього джерела живлення. Автоматично вибирається джерело з найвищим напругою.

Мікросхема FTDI FT232RL отримує живлення, тільки якщо сама платформа заживлена від USB. Таким чином при роботі від зовнішнього джерела (не USB), буде відсутня напруга 3.3 В, що генерується мікросхемою FTDI, при цьому світлодіоди RX і TX блимають тільки при наявності сигналу високого рівня на виводах 0 і 1. Мікроконтролер ATmega328, в свою чергу, має 32 кБ флеш-пам'яті. ATmega328 має 2 кБ ОЗП і 1 Кб EEPROM.

Кожен з 14 цифрових виводів Nano, використовуючи функції pinMode (), digitalWrite (), і digitalRead (), може налаштовуватися як вхід або вихід. Виводи працюють при напрузі 5 В. Кожен вивід має навантажувальний резистор (стандартно відключений) 20-50 кОм і може пропускати до 40 мА. Деякі виводи мають особливі функції:

Послідовна шина: 0 (RX) і 1 (TX). Виводи використовуються для отримання (RX) і передачі (TX) даних TTL. Дані виводи підключені до відповідних виводів мікросхеми послідовної шини FTDI USB-to-TTL.

Зовнішнє переривання: 2 і 3. Дані виводи можуть бути налаштовані на виклик переривання або на молодшому значенні, або на передньому чи задньому фронті, або при зміні значення. Детальна інформація знаходиться в описі функції attachInterrupt ().

ШІМ: 3, 5, 6, 9, 10, і 11. Будь-який з виводів забезпечує ШІМ з роздільною здатністю 8 біт за допомогою функції analogWrite ().

SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). За допомогою даних виводів здійснюється зв'язок SPI, яка, хоча і підтримується апаратною частиною, не включена в мову Arduino.

LED: 13. Вбудований світлодіод, підключений до цифрового виводу 13.

Якщо значення на виведення має високий потенціал, то світлодіод горить.

На платформі Nano встановлені 8 аналогових входів, кожен дозволом 10 біт (тобто може приймати тисячу двадцять чотири різних значення). Стандартно виводи мають діапазон вимірювання до 5 В щодо землі, проте є можливість змінити верхню межу за допомогою функції analogReference (). Деякі виводи мають додаткові функції:

I2C: A4 (SDA) і A5 (SCL). За допомогою виводів здійснюється зв'язок I2C (TWI). Для створення використовується бібліотека Wire (інформація на сайті Wiring).

Додаткова пара виводів платформи:

AREF. Опорна напруга для аналогових входів. Використовується з функцією analogReference ().

Reset. Низький рівень сигналу на виводі перезавантажує мікроконтролер. Зазвичай застосовується для підключення кнопки перезавантаження на платі розширення, що закриває доступ до кнопки на самій платі Arduino.

На платформі Arduino Nano встановлено кілька пристроїв для здійснення зв'язку з комп'ютером, іншими пристроями Arduino або микроконтролерами. ATmega328 підтримують послідовний інтерфейс UART TTL (5 В),, що здійснюється виводами 0 (RX) і 1 (TX). Встановлена на платі мікросхема FTDI FT232RL направляє даний інтерфейс через USB, а драйвери FTDI (включені в програму Arduino) надають віртуальний COM порт програмі на комп'ютері. Моніторинг послідовної шини (Serial Monitor) програми Arduino дозволяє посилати і отримувати текстові дані при підключенні до платформи. Світлодіоди RX і TX на платформі будуть мигати при передачі даних через мікросхему FTDI або USB підключення (але не при використанні послідовної передачі через виводи 0 і 1).

Бібліотекою SoftwareSerial можливо створити послідовну передачу даних через будь-який з цифрових вивід Nano.

ATmega328 підтримують інтерфейси I2C (TWI) і SPI. В Arduino включена бібліотека Wire для зручності використання шини I2C. Більш детальна інформація знаходиться в документації.

Nano розроблена таким чином, щоб перед записом нового коду перезавантаження здійснювалася самою програмою, а не натисканням кнопки на платформі. Одна з ліній FT232RL, керуючих потоком даних (DTR), підключена до висводу перезавантаження мікроконтролерів ATmega168 або ATmega328 через конденсатор 100 нФ. Активація даної лінії, тобто подача сигналу низького рівня, перезавантажує мікроконтролер. Програма Arduino, використовуючи цю функцію, завантажує код одним натисканням кнопки Upload в самому середовищі програмування. Подача сигналу низького рівня по лінії DTR скоординована з початком запису коду, що скорочує таймаут завантажувача.

Функція має ще одне застосування. Перезавантаження Nano відбувається кожного разу при підключенні до програми Arduino на комп'ютері з ОС Mac X або Linux (через USB). Наступні півсекунди після перезавантаження працює завантажувач. Під час програмування відбувається затримка декількох перших байтів коду, щоб уникнути отримання платформою некоректних даних (всіх, окрім коду нової програми). Якщо проводиться разове налагодження скетчу, записаного в платформу, або введення будь-яких інших даних при першому запуску, необхідно переконатися, що програма на комп'ютері очікує протягом секунди перед передачею даних.

2.2 Опис середовища програмування Arduino IDE

Для створення розробок на базі Arduino, необхідно програмне забезпечення для написання і завантаження програм в мікроконтролер. Дані функції виконуються за допомогою Arduino IDE. Середовище розробки Arduino складається з вбудованого текстового редактора програмного коду, панелі

інструментів, вікна виведення тексту. Для завантаження програм необхідно підключити середу розробки до апаратної частини мікроконтролера.

Будь-яка програма, написана в середовищі програмування Arduino IDE, називається скетч. Скетч пишеться в текстовому редакторі, що має інструменти, що дозволяють виробляти над ним будь-які дії, такі як вставка / видалення, заміна / пошук тексту. Під час збереження або компіляції скетчу в області повідомлень з'являються пояснення, або повідомлення про помилку, в разі неправильного написання програмного коду.

Мова програмування Arduino стандартний C ++, які мають особливості, які полегшують написання програм:

— файли програм перед компіляцією обробляються препроцесором Arduino;

— програміст зобов'язаний написати дві обов'язкові функції: setup () викликається при старті і loop () повторюється в нескінченному циклі;

— в текст програми не обов'язково записувати заголовки при використанні стандартних бібліотек;

— відсутність попередніх налаштувань компілятора.

Завантаження скетчу відбувається через вбудований завантажувач (Bootloader), який є програмою, яка дозволяє завантажувати програмний код без використання додаткових апаратних засобів і може працювати через інтерфейси RS-232, USB і Ethernet.

Завантажувач активний протягом декількох секунд при завантаженні будь-якого скетчу в мікроконтролер. Про роботу завантажувача сигналізує світлодіод, вбудований в плату мікроконтролера.

Для створення рішення задачі в Arduino IDE необхідно виконати наступні етапи настройки:

— створення самого проекту;

— конфігурація обладнання;

— підключення пристроїв до мережі;

— програмування мікроконтролера;

— налаштування візуалізації;

— завантаження даних конфігурації;

— діагностика роботи різних функцій обладнання.

Інтерфейс програми забезпечує просту навігацію по задачам і даними проекту (рисунок 2.3).

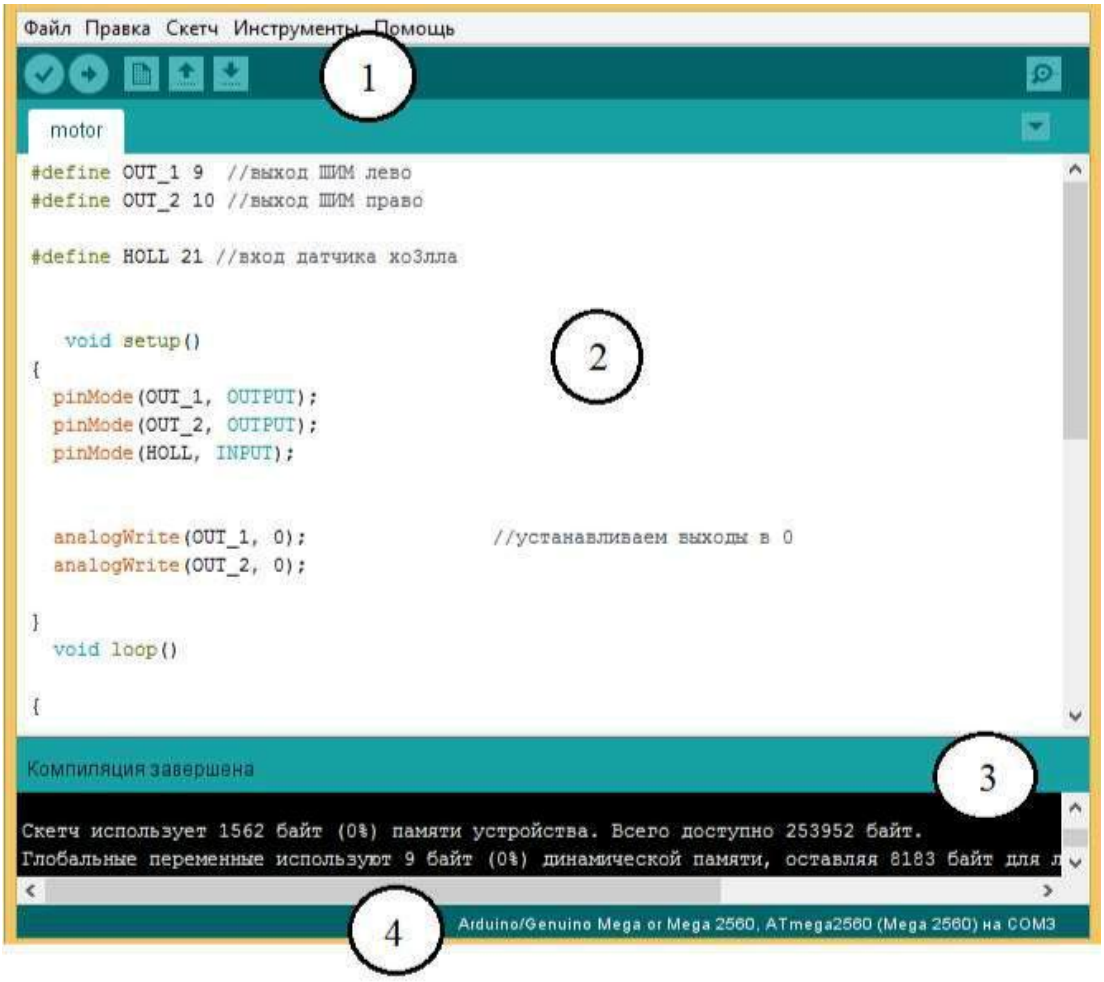


Рисунок 2.3 – Структура інтерфейсу створення скетчу

1) Кнопки панелей інструментів, що дозволяють перевірити, завантажити, створити програму, відкрити або зберегти скетч, відкрити моніторинг порту;

2) Текстовий редактор, призначений для написання скетчу;

3) Вікно виведення тексту (консоль);

4) Область повідомлень з інформацією про обраний поточному мікроконтролері і порту виводу.

Arduino IDE також містить безліч передбачених бібліотек, що додають функціональність скетчам при роботі з апаратною частиною або обробці даних. Для використання бібліотеки необхідно вибрати меню «Скетч> Бібліотеки». Одна або кілька директив «#include» будуть розміщені на початку коду скетчу з подальшою компіляцією бібліотек і разом зі скетчем. Завантаження бібліотек вимагає додаткового місця в пам'яті Arduino. Невикористані бібліотеки можна видалити з скетчу прибравши директиву «#include». Також присутня можливість додавання бібліотек зі сторонніх ресурсів і створення власних.

Вибір апаратної платформи впливає на швидкість передачі даних, швидкість ЦП, параметри при компіляції і завантаженні скетчів.

Можливість моніторингу порту дозволяє відображати їх посилають дані в платформу Arduino (плата USB). Для відправки даних необхідно ввести текст і вибрати меню Інструменти> Моніторинг порту. Потім вибирається швидкість передачі даних, відповідна значенню Serial. Набір функцій Serial служить для зв'язку пристрою Arduino з комп'ютером або іншими пристроями, що підтримують послідовний інтерфейс обміну даними. Для обміну даними Serial використовують цифрові порти введення / виводу 0 (RX) і 1 (TX), а також USB порт. Важливо враховувати, що якщо ви використовуєте функції Serial, то не можна одночасно з цим використовувати порти 0 і 1 для інших цілей.

Плата Arduino Mega має три додаткових послідовних порту: Serial1 на портах 19 (RX) і 18 (TX), Serial2 на портах на портах 17 (RX) і 16 (TX), Serial3 на портах на портах 15 (RX) і 14 (TX). Щоб використовувати ці порти для зв'язку з комп'ютером знадобиться додаткові адаптери USB-to-serial, тому що вони не підключені до вбудованого адаптера плати Mega. Для зв'язку із зовнішнім пристроєм через послідовний інтерфейс необхідно з'єднати TX порт пристрою з RX портом зовнішнього пристрою і RX порт вашого пристрою з портом TX зовнішнього і з'єднайте «землю» на пристроях.

2.4 Опис основних функцій Arduino IDE

Мова програмування пристроїв Arduino заснований на C ++. Мова Arduino ділиться на три розділи: оператори, дані і функції.

Призначена для користувача програма може складатися з декількох функцій, кожна з яких необхідна для вирішення завдання управління.

Директива «#define» зображена на рисунку 2.4 за допомогою якої задаються імена констант перед компіляцією програми. Певні цією директивою константи не займають програмної пам'яті, компілятор замінює всі звернення до них їх значеннями на етапі компіляції, відповідно вони служать виключно для зручності програміста і поліпшення читаності тексту програми.

Недолік цієї функції полягає в тому, що під час запису імені константи, заданої цією функцією «#define», записати в імені іншої змінної, то значення буде змінено.

```
#define knopka 52
#define in1 9
#define in2 10
```

Рисунок 2.4 – Запис констант «#define» в програмі скетчу

Першим записується ім'я входу, зручне для програміста, другим записується (присвоюється) номер входу мікроконтролера.

Логічний тип даних, який показано на рисунку 2.5, записується «bool». Приймає значення «true» і «false», займає в пам'ять один байт. «False» визначається в логічному вираженні як 0. «true» в логічному вираз приймає значення 1, а також значення -1, -2, -200.

```
bool gtv1;
bool gtv2;
```

Рисунок 2.5 – Запис даних «bool» в програмі скетчу

Тип даних «int» (рисунок 2.6) використовується для зберігання чисел в діапазоні від -32768 до 32767, займає 2 байта пам'яті. Arduino компілятор сам піклється про розміщення в пам'яті і уявлення негативних чисел, тому арифметичні дії над цілими числами не вимагають додаткових дій.

```
int count1P = 0;
```

Рисунок 2.6 – Запис даних «int» в програмі скетчу

При арифметичних операціях змінна типу «int», коли досягає свого максимального значення, «перескакує» на мінімальне значення і навпаки.

Тип даних записуються як «unsigned long» використовується для зберігання позитивних цілих чисел в діапазоні від 0 до 4294967295 і займає 32 біта (4 байта) в пам'яті.

```
unsigned long bounseInputD2P = 0UL;
bool bounseInputD4S = 0;
bool bounseInputD4O = 0;
```

Рисунок 2.7 – Запис даних «unsigned long» в програмі скетчу

Кваліфікатор «volatile» записується перед типом змінної і є дороговказом для компілятора. Він вказує компілятору не завантажувати змінну з запам'ятовує регістра — тимчасової комірки пам'яті, в якій зберігаються змінні програми і проводяться операції з ними. При певних умовах значення змінних, що зберігаються в регістрах, можуть виявитися неточними. При оголошенні змінної як «volatile», ми задаємо микроконтроллеру завдання, що значення змінної може бути змінено в будь-який момент будь-чим за межами програми, а значить мікроконтролер повинен перезавантажити значення змінної кожен раз при її використанні.

Функція «setup ()», що викликається при старті скетчу. Ініціалізує змінні, визначає режим роботи виводів контролера, запускає використовувані

бібліотеки. Запускається один раз після кожної подачі живлення на мікроконтролер.

Функція «loop ()» повторюється в нескінченному циклі. Ініціалізує і встановлює початкові значення змінних, виробляє обчислення в програмі і реагує на них.

Функція «pinMode» використовується для ініціалізації режиму роботи вииводів плати мікроконтролера. Запис Pin ініціалізує вхід мікроконтролера, який буде використовуватися в роботі. Mode встановлює режим роботи входи або вихід.

```
pinMode(52, INPUT);
pinMode(in1, OUTPUT);/
pinMode(in2, OUTPUT);/
pinMode(holla, INPUT);
```

Рисунок 2.8 – Функція «pinMode ()» в програмі скетчу

При записи функції pinMode (52, INPUT) перша змінна записується як номер входу мікроконтролера або призначена назва директивою «#define», потім режим роботи.

Ця функція необхідна, так як всі виводи платформи Arduino можуть працювати як входи і виходи. Всі виводи знаходяться в високоімпедансному стані, тобто порт виводу дає мале навантаження на схему, в яку підключена Arduino. Для перекладу стану виведення з одного стан в інше необхідно мале значення струму. Якщо порт мікроконтролера нікуди не підключений, то значення на ньому братимуть випадкові величини через наявність електричних перешкод і ємнісних зв'язків між портами.

Режим роботи «INPUT» встановлює обраний порт як вхід. Режим роботи «OUTPUT» встановлює порт як вхід і буде знаходиться в низкоімпедансному стані, пропускаючи через себе максимально допустимий струм 40мА для плати.

Функція «digitalRead ()» записується за таким же принципом, як і функція «pinMode ()», вона зчитує значення з заданого функцією «pinMode ()» входа- «HIGH» або «LOW» цифрового порту.

```
bool sensor = digitalRead(holla);
```

Рисунок 2.9 – Функція «digitalRead ()»

Запис читається так: за допомогою функції digitalRead () зчитується значення зі входу, що має ім'я holla. Для спрощення програмного коду можна присвоїти ім'я sensor.

Значення входу / виходу цифрового порту «HIGH» може позначати дещо різне в залежності від режиму порту як «INPUT» або «OUTPUT». Коли порт вхід / виходу встановлений в режим «INPUT» за допомогою функції «pinMode ()», і зчитується функцією digitalRead () », мікроконтролер віддасть значення «HIGH »напругою 3В або вище на зазначеному порту.

Також порт може бути встановлений як «INPUT» функцією «pinMode ()», і потім встановлений в «HIGH» значення функцією «digitalWrite ()». Це підключить до порту внутрішній підтягаючий резистор 20кОм, що дозволить отримувати постійне значення «HIGH» при читання цього порту, якщо тільки значення відсутнє в режимі очікування надається «LOW» зовнішньої ланцюгом, підключеним до цього порту.

Коли порт вхід / вихід налаштований як «OUTPUT» функцією «pinMode()», і встановлено значення «HIGH» функцією «digitalWrite ()», на порту буде постійна напруга 5В.

Значення «LOW» також різний для режиму «INPUT» і «OUTPUT». Коли порт налаштований як «INPUT», і зчитується функцією «digitalRead ()», мікроконтролер поверне «LOW» якщо напруга на даному порту менше або дорівнює 2В.

Якщо ж порт встановлений в «OUTPUT» і «LOW», то напруга на виході порту буде 0 вольт.

Оператор «if ... else» здійснює контроль і перевірку над процесом виконання коду програми.

```
if (pinFiveInput < 500)
{
  // действие A
}
else
{
  // действие B
}
```

Рисунок 2.10 – Приклад використання оператора «if ... else»

У дужках записується умова, яку необхідно перевіряти і виконувати відповідно дію А. В разі іншого умови оператор «else» виконує перевірку, відмінну від умови «if», що дозволяє перевіряти кілька взаємовиключних перевірок. Кожна перевірка дозволяє переходити до наступного за нею оператору не раніше, ніж отримає логічний результат ІСТИНА. Коли перевірка з результатом ІСТИНА знайдена, запускається вкладена в неї блок операторів, і потім програма ігнорує всі наступні рядки в конструкції «if ... else». Якщо жодна з перевірок не отримала результат ІСТИНА, за замовчуванням виконується блок операторів в «else», якщо останній присутній, і встановлюється дія за замовчуванням.

Функція «analogWrite ()» виводить аналогову величину ШІМ. Наприклад analogWrite (9, 255), де 9 це номер порту виведення Arduino (замість цифри може бути записано присвоєне такого виводу ім'я, написане програмістом), а число 255 це ширина імпульсу ШІМ. Діапазон скважності від 0 до 255.

ШІМ на виході являє собою прямокутну хвилю із заданою шириною імпульсу, аж до виклику нового імпульсу. Частота ШІМ сигналу для Arduino становить приблизно 500Гц. Для виклику цієї функції не використовується задає функція «pinMode ()».

Оператор «switch» діє як альтернатива оператору «if», дозволяючи створювати альтернативний код, що виконується при різних умовах, як на рисунку 2.11.

Оператор «switch» порівнює значення змінної var зі значенням, визначеним оператором «case». Коли умова «case» задане значення змінної виконується, виконується програмний код.

```
switch (var) {
    case 1:
      //выполняется, когда var равно 1
      break;
    case 2:
      //выполняется когда  var равно 2
      break;
    default:

  }
```

Рисунок 2.11 – Приклад використання оператора «switch»

Оператор «break» виводить виконання програмного коду з оператора «case». Без «break» оператор «switch» буде продовжувати виконання програми, ігноруючи значення змінної оператора «case».

Функція «millis ()» використовується для роботи з часом. Вона повертає кількість мілісекунд з моменту початку виконання роботи програми. Це кількість скидається на нуль, в наслідок переповнення значення, приблизно через 50 днів.

3 ПРОЕКТУВАННЯ МІКРОПРОЦЕСОРНОЇ СИСТЕМИ ПЕРЕВІРКИ ПРОВІДНИХ ЛІНІЙ ЗВ’ЯЗКУ

3.1 Принцип роботи центрального мікроконтролера

Мікроконтролери серії Arduino мають основу AVR-сімейство восьмибітних мікроконтролерів фірми Atmel. Головна функція ядра AVR — гарантувати правильне програмне виконання. Контролер повинен виконувати операції, такі як звернення до пам'яті, виконання обчислень, управління зовнішніми пристроями, і управління переривань.

Для того, щоб максимізувати виконання всіх функцій, AVR користується Гарвардської архітектурою (рисунок 3.1) — має поділ програми і даних в різних адресних просторах.

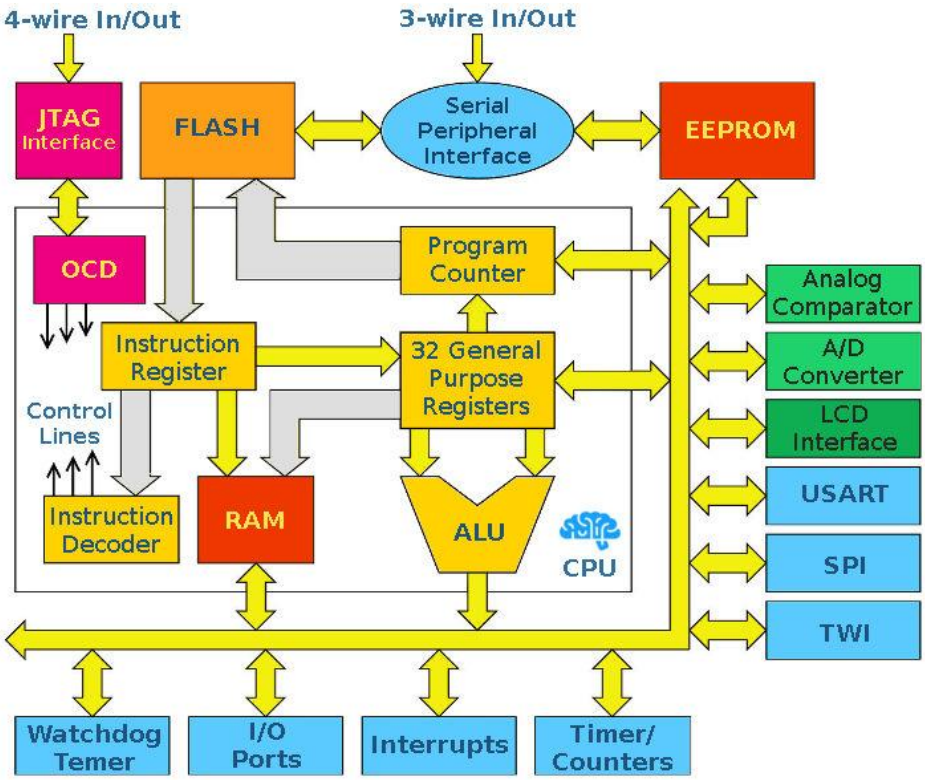


Рисунок 3.1 – Функціональна схема архітектури AVR

Інструкції в програмній пам'яті виконуються з єдиною горизонтальною конвеєрної обробкою. Поки одна інструкція виконується, така інструкція

попередньо витягується з програмної пам'яті. Це концепція надає можливість інструкцій виконуватися в кожному циклі. Програмна пам'ять — це вбудована флеш-пам'ять, що може бути перепрограмована.

Файл регістра швидкого доступу містить 32 8-бітових регістра, що створюють записи з одним часом доступу до циклу. Це дозволяє виконувати однотактной арифметичне логічний пристрій (АЛП). У типовій операції АЛП з файлу регістра виводяться два операнда (аргументи операції, дані, що обробляються командою), виконується операція, і результат зберігається в файлі регістра — за один цикл. Шість з 32 регістрів можуть використовуватися в якості трьох 16-бітних покажчиків для адресації даних — з метою ефективного обчислення. програмна пам'ять — це вбудована флеш-пам'ять. Ці додані функціональні регістри представляють собою 16-бітові X-, Y- і Z-регістри.

АЛП підтримує арифметику і логічні дії між регістрами або між константою і регістром. Дії однієї операції можуть також виконуватися в АЛП. Після арифметичної операції, статус регістра оновлюється, щоб відобразити інформацію про результат дії.

Хід виконання програми забезпечується умовним і безумовним переходами і командами звернення, здатних безпосередньо звернутися до цілого адресного простору. Більшість інструкцій AVR мають єдиний формат 16-бітного слова. Кожен адреса програмної пам'яті містить 16- або 32-бітну інструкцію.

Програмний простір флеш-пам'яті розділений на дві секції, секції програми завантаження та секції прикладної програми. Обидві секції мають виділені біти блокування для захисту запису і читання / запису. Інструкція SPM, яка записується в секцію прикладної програми, повинна знаходитися в розділі програми завантаження. SPM — абревіатура, що означає програмне забезпечення для управління проектами.

Протягом переривань і викликів підпрограми, лічильник команд зберігається в стеку. Стек ефективно розподіляється в загальній статичної

пам'яті (SRAM), отже, розмір стека обмежений тільки загальним розміром статичної пам'яті і використанням її використанням. SRAM - статична пам’ять з довільним доступом, напівпровідникова оперативна пам'ять.

Всі призначені для користувача програми повинні ініціалізувати вказівник стека (ВС) в підпрограмі «Скидання» (до виконання підпрограм або переривань). ВС доступний для читання / запису в просторі вводу / виводу. За допомогою SRAM можна легко отримати доступ через п'ять різних режимів адресації, які підтримуються в архітектурі AVR. Простір пам'яті в архітектурі AVR є лінійними і регулярні карти пам'яті.

Модуль гнучкого переривання має свої регістри управління в просторі вводу / виводу з додатковим бітом дозволу глобального переривання в регістрі стану. Всі переривання мають окремий вектор переривання в таблиці векторів переривань. Переривання мають пріоритет відповідно до їх позицією вектора переривання. Чим нижче адресу вектора переривань, тим вище пріоритет. Обсяг пам'яті введення / виводу містить 64 адреси для периферійних функцій AVR, таких як регістри управління, SPI і інші функції введення-виведення. Доступ до пам'яті введення / виводу можна отримати безпосередньо або в якості місця розташування простору даних, наступного за файлом регістра. Крім того, цей пристрій має розширене простір введення / виводу в SRAM, де можуть використовуватися тільки інструкції ST / STS / STD і LD / LDS / LDD.

Високопродуктивний АЛП AVR працює в прямому сполученні з усіма 32 робочими регістрами загального призначення. Протягом одного циклу, виконуються арифметичні операції між регістрами загального призначення або між регістрами і безпосереднього виконання. Операції АЛП діляться на три головних категорії: арифметичні, логічні, і бітові. Деякі реалізації архітектури також забезпечують потужний множник, що підтримує як знакове / без знакове множення, так і дробовий формат.

Регістр стану містить інформацію про результат самої останньої арифметичної команди, що виконувалася. Ця інформація може бути використана для зміни ходу виконання програми для того, щоб виконувати

умовні дії. Регістр стану оновлюється в кінці операції АЛП. Це в багатьох випадках усуне необхідність в використання спеціальних команд порівняння, що призведе до більш швидкого і компактному коду. Регістр стану не зберігається автоматично при введенні підпрограми переривання і відновлюється при поверненні з переривання. Це повинно бути оброблено програмним забезпеченням.

Регістровий файл оптимізований під AVR. Для досягнення необхідної швидкодії і гнучкості, використовуються методи, які підтримуються реєстрових файлом:

— один 8-бітний вихідний операнд і один 8-бітний вхідний результат;

— два 8-бітних вихідних операнда і один 8-бітний вхідний результат;

— два 8-бітних вихідних операнда і один 16-бітний вхідний результат;

— один 16-розрядний вихідний операнд і один 16-бітний вхідний результат.

Більшість з інструкцій, які працюють з реєстрових файлом, мають прямий доступ до всіх записів, і більшість з них є єдиними інструкціями циклу. Кожному запису також присвоєно адресу пам'яті даних, що відображає їх безпосередньо в перші 32 місця просторів перед даних. І хоча фізично вони не розташовуються в SRAM, ця організація пам'яті забезпечує більшу гнучкість в доступі до регістрів, і X-, Y-, і Z-покажчики регістра можуть використовуватися, щоб індексувати будь-який регістр в файлі. X-, Y-, і Z-регістри, що представляють собою 16-бітові адресні покажчики для непрямої адресації простору даних.

Стек головним чином використовується для зберігання тимчасових даних, для зберігання місцевих змінних і для зберігання адрес повернення після переривань і викликів підпрограми. Реалізація стек здійснюється від більш високих до більш низьких місць пам'яті. Покажчик стека вказує на область SRAM даних, в якій розташовані підпрограми і стеки переривань. Команда Stack PUSH зменшить покажчик стека. Стек в SRAM даних повинен бути визначений програмою перед виконанням будь-яких викликів підпрограм або

дозволених переривань. Початкове значення покажчика стека має дорівнювати останньою адресою внутрішньої SRAM і повинен бути встановлений в положення, вказане вище початку SRAM.

Процесор AVR управляється тактовою частотою процесора clkCPU, безпосередньо генерується з обраного джерела синхронізації для чіпа. Немає внутрішнього поділу таймера. На наведеному нижче малюнку 3.2 показані вибірки та інструкції паралельних команд, дозволені архітектурою Гарварда і концепцією реєстрового файлу з швидким доступом. Це базова концепція конвеєрної обробки для отримання до 1 MIPS на МГц (MIPS — одиниця обчислювальної швидкості, еквівалентна мільйону команд в секунду).

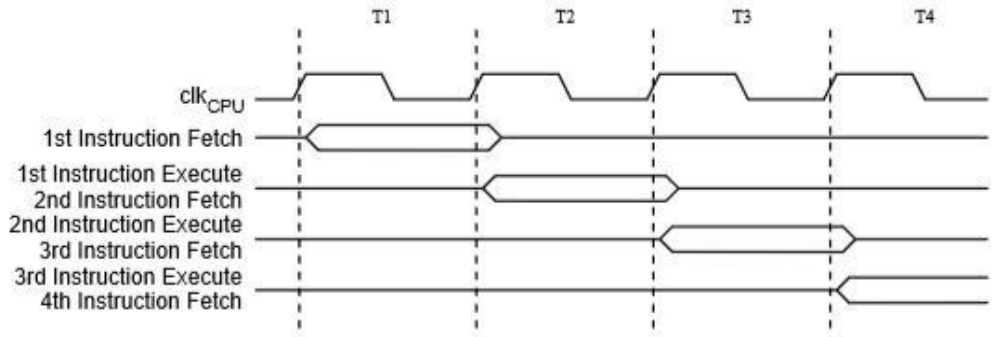


Рисунок 3.2 – Інструкції паралельних команд

На наступному малюнку 3.3 показана концепція внутрішнього таймера для реєстрового файлу. За один такт виконується операція АЛП з використанням двох реєстрових операндів, результат зберігається в реєстрі призначення.

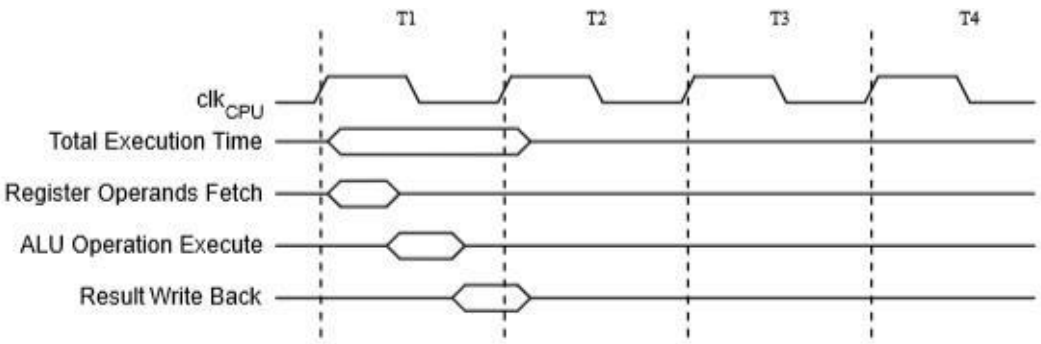


Рисунок 3.3 – Один цикл виконання команди АЛП

Скидання і обробка переривань

AVR надає кілька різних джерел переривань. Ці переривання і вектора переривань мають окремий програмний вектор в програмній пам'яті. Всі переривання призначаються індивідуальними бітами включення, які повинні бути записані логічно разом з бітом дозволу глобального переривання в регістрі стану, щоб включити переривання. Залежно від значення лічильника програм переривання можуть бути автоматично відключені, якщо запрограмовані біти блокування завантаження BLB02 або BLB12. Ця функція покращує безпеку програмного забезпечення.

Найнижчі адреси в програмній пам'яті за замовчуванням визначаються як скидання і вектора переривань. Вони визначають рівні пріоритету: чим нижче адресу, тим вище рівень пріоритету.

Скидання має найвищий пріоритет, а наступний — INT0-запит зовнішнього переривання 0. Вектора переривань можуть бути перенесені на початок завантажувального сектора, встановивши біт IVSEL в регістр управління / контролю MCU. Вектор скидання також можна перенести в початок завантажувального сектора, за допомогою біта BOOTRST.

Коли відбувається переривання, 1-біт глобального переривання очищається, і все переривання відключені. Користувача програмне забезпечення може записувати логічний код в 1-біт, щоб включити вкладені переривання. Всі дозволені переривання можуть зупиняти поточну процедуру переривання. 1-біт автоматично встановлюється, коли виконується команда вихід з переривання — RETI.

Існують в основному два типи переривань: перший тип запускається подією, яке встановлює прапор переривання. Для цих переривань програмний лічильник векторизованних до фактичного вектору переривань для виконання процедури обробки переривань, а апаратне забезпечення очищає відповідний прапор переривання. Прапори переривань також можна очистити, записавши логічний біт в позицію біт-прапора, що підлягає очищенню. Якщо умова переривання відбувається, коли біт дозволу переривання очищається, прапор

переривання буде встановлений і збережений до тих пір, поки переривання не буде включено, або прапор не буде очищено програмним забезпеченням. Аналогічно, якщо виконується одна або кілька умов переривання, коли біт глобального дозволу переривань очищається, відповідний прапор переривання буде встановлений і збережений до тих пір, поки біт глобального дозволу переривання не буде встановлено, і потім буде виконуватися по порядку пріоритету.

Другий тип переривань буде спрацьовувати до тих пір, поки присутня умова переривання. Цим перериванням не обов'язково мати прапори переривання. Якщо умова переривання зникає до того, як переривання включено, переривання не буде запущено. Коли AVR виходить з переривання, він завжди повертається до основної програми і виконує ще одну команду до того, як буде обслуговуватися якесь очікує переривання.

Регістр стану не зберігається автоматично при введенні підпрограми переривання і не відновлюється при поверненні з процедури переривання. Це повинно бути оброблено програмним забезпеченням. При використанні команда CLI для відключення переривань переривання будуть негайно відключені. Команда CLI — відноситься до класу інструкцій введення / виведення, вона скидає прапори переривання. Після команди CLI всі наступні переривання НЕ будуть виконуватися, навіть якщо воно відбувається одночасно з інструкцією CLI.

Відповідь на виконання переривання для всіх дозволених переривань AVR дорівнює чотирьом тактам. Після чотирьох тактових циклів виконується програмний адреса вектора для дійсної процедури обробки переривань. Під час цього чотирьох тактового циклу лічильник програм поміщається в стек. Зазвичай вектор являє собою перехід до процедури переривання, і цей перехід займає три тактових циклу. Якщо переривання відбувається під час виконання команди з декількома циклами, ця команда завершується до того, як переривання буде обслуговуватися. Якщо переривання відбувається, коли регістр контролю MCU знаходиться в сплячому режимі, час відгуку на

виконання переривання збільшується на чотири тактових циклу. Це збільшення відбувається на додаток до часу запуску з обраного сплячого режиму. Повернення з процедури обробки переривань займає чотири тактових циклу.

Пам'ять AVR

Архітектура AVR має два основні сектори пам'яті: пам'ять даних Data Memory і програмну пам'ять Program Memory. Всі сектора пам'яті є лінійними і регулярними.

ATmega328 містить 32-кілобайтні вбудовану перезавантажувати флеш-пам'ять для зберігання програм. Оскільки всі інструкції AVR мають дозвіл 16 або 32 біт, флеш-пам'ять має розмірність 16Kx16. Для безпеки програмного забезпечення простір флеш-пам'яті розділене на дві секції, секції програми завантаження та секції прикладної програми. Флеш-пам'ять має витримку не менше 10 000 циклів запису / стирання. Лічильник програм ATmega328 має дозвіл 14 біт, таким

чином звертаючись до осередків пам'яті 16K. Робота секції програми завантаження і програма блокування завантаження пов'язані одними битами для захисту програмного забезпечення.

Системний годинник

На рисунку 3.4 показані основні системи годин в пристрої і їх розподіл. Всі годинники не повинні бути активні в один момент часу. Щоб зменшити споживання енергії, годинник для що не використовуються модулів можуть бути зупинені з використанням різних режимів сну. Все тактові виходи блоку управління годин AVR працюють з однаковою частотою.

Центральний процесор спрямований в частині системи, пов'язані з роботою ядра AVR. Прикладами таких модулів є файл регістра загального призначення, регістр стану і пам'ять даних, що містить покажчик стека. Припинення тактових імпульсів центрального процесора блокує ядро від виконання загальних операцій і обчислень.

Годинники введення / виведення clkI / O використовуються більшістю модулів введення / виведення, таких як таймер / лічильники, SPI і USART.

Годинники введення-виведення також використовуються модулем зовнішнього переривання, але виявлення стану запуску в модулі USI виконується асинхронно, коли clkI / O зупиняється. Якщо для пробудження від відключення живлення використовується переривання з рівнем, необхідний рівень повинен бути достатньо тривалим, щоб регістр контролю MCU завершив пробудження, щоб викликати переривання рівня. Якщо рівень зникне до закінчення часу запуску, MCU все одно включиться, але переривання не буде згенеровано.

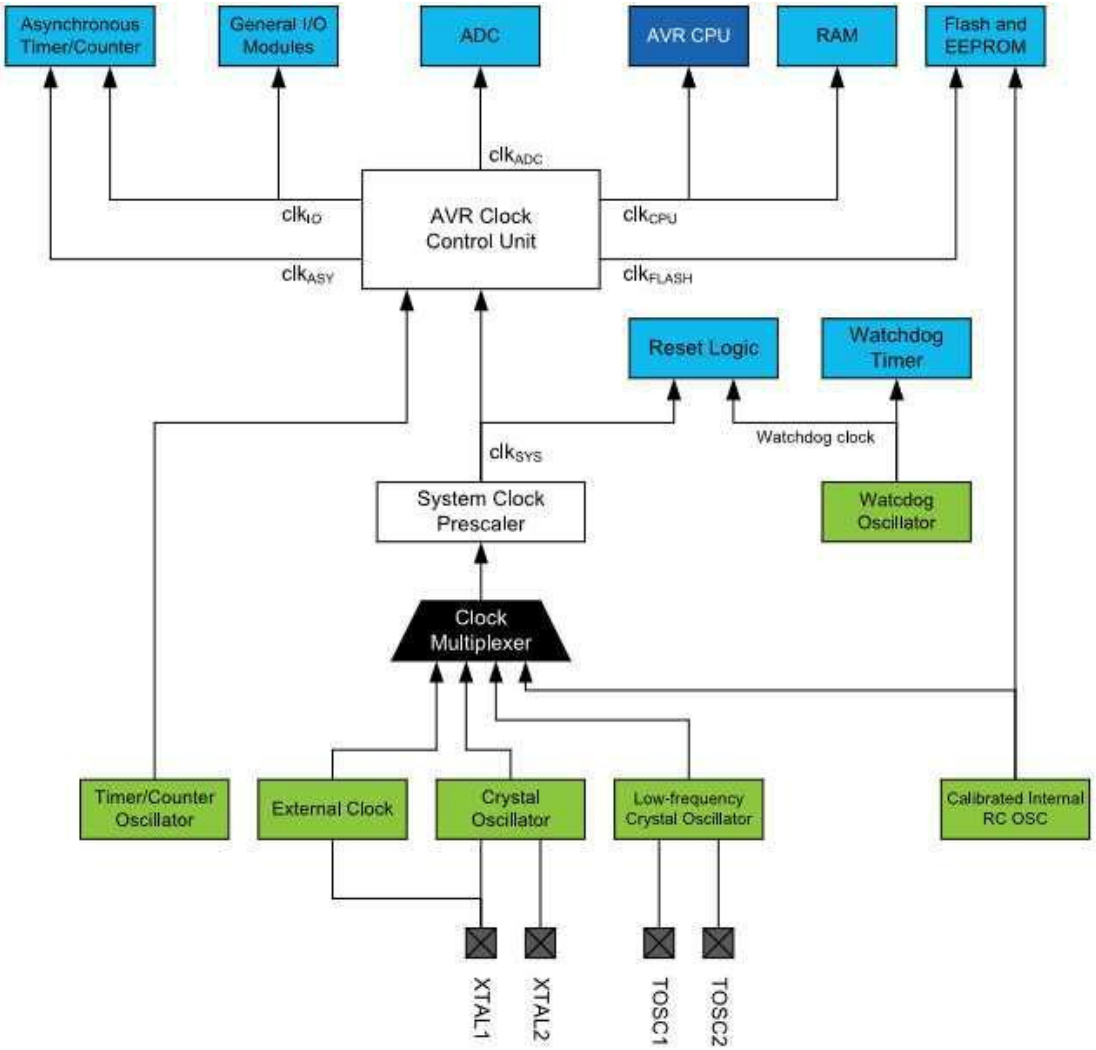


Рисунок 3.4 – Розподіл годин

Асинхронний таймер — clkASY дозволяють синхронізувати асинхронні таймери / лічильники безпосередньо з зовнішнього тактового генератора або зовнішнього тактового кристала з частотою 32 кГц. Спеціальний тактовий

домен дозволяє використовувати цей таймер / лічильник в якості лічильника в реальному часі, навіть коли пристрій знаходиться в сплячому режимі.

Годинники АЦП забезпечені виділеної тактовою областю. Це дозволяє заблокувати годинник центрального процесора, щоб зменшити шум, створюваний цифровий схемою. Це дає більш точні результати перетворення АЦП.

Пристрій має внутрішній RC-генератор з частотою 8,0 МГц і з запрограмованим запобіжником CKDIV8, що призводить до системних годинах з частотою 1,0 МГц. Час запуску встановлюється на максимум, і період тайм-ауту включений: CKSEL = 0010, SUT = 10, CKDIV8 = 0. Цей параметр за замовчуванням гарантує, що всі користувачі зможуть налаштувати бажаний джерело синхронізації за допомогою будь-якого доступного інтерфейсу програмування.

Будь-яке джерело синхронізації потребує достатньої напрузі живлення, щоб почати генерування коливань і генерування мінімальної кількості циклів, перш ніж його можна вважати стабільним. Для забезпечення достатнього напрузі живлення пристрій видає внутрішній скидання з затримкою тайм-ауту (tTOUT) після перезавантаження пристрою усіма іншими джерелами скидання. Затримка (tTOUT) синхронізується з генератором сторожового таймера, а кількість циклів затримки встановлюється битами SUTx і CKSELx. Частота генератора сторожового таймера залежить від напруги.

Основна мета затримки — зберегти пристрій в режимі скидання, поки воно не буде забезпечено мінімальним напругою живлення. Затримка не контролюватиме фактичне напруга, тому потрібно вибрати затримку довше, ніж час наростання напругою живлення. Якщо це неможливо, використовується внутрішня або зовнішня схема обмеження доступу електроенергії — Brown-Out Detection. Ланцюг BOD забезпечить достатню напруга живлення до скидання, і можна відключити затримку тайм-ауту. Відключення затримки тайм-ауту без використання схеми обмеження доступу електроенергії не рекомендується.

Генератор повинен видавати коливання протягом мінімальної кількості циклів, щоб вважати його стабільним для роботи. Внутрішній лічильник пульсацій контролює вихід генератора і зберігає внутрішній скидання активним для заданої кількості тактових циклів. Потім скидання відпускається, і пристрій починає виконання. Рекомендований час запуску генератора залежить від типу тактового сигналу і варіюється від 6 циклів для зовнішніх тактових імпульсів до 32K циклів для низькочастотного кристала.

Послідовність запуску для годин включає в себе як тимчасову затримку, так і час запуску, коли пристрій запускається з моменту скидання. При запуску з режиму енергозбереження або відключення живлення передбачається, що напруга живлення знаходиться на достатньому рівні і включається тільки час запуску.

Виводи XTAL1 і XTAL2 є вхідними та вихідними сигналами інвертую чого підсилювача, який може бути налаштований для використання в якості генератора, як показано на малюнку нижче. Можна використовувати або кварцовий кристал, або керамічний резонатор.

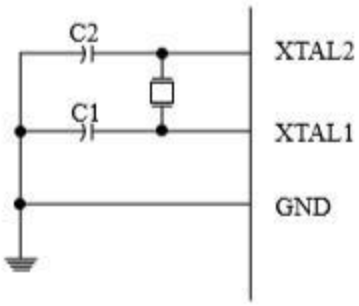


Рисунок 3.5 – Схема кристального генератора

C1 і C2 завжди повинні бути рівні для обох кристалів і резонаторів. Оптимальне значення конденсаторів залежить від використовуваного кристала або резонатора, кількості паразитної ємності і електромагнітного шуму навколишнього середовища. Для керамічних резонаторів слід використовувати значення конденсатора, зазначені виробником.

Таймер / лічильник генератора використовує той же кварцовий генератор для низькочастотного генератора і генератора таймера / лічильника. На цьому пристрої контакти генератора таймера / лічильника (TOSC1 і TOSC2) спільно використовуються з XTAL1 і XTAL2. При використанні генератора таймера / лічильника частота системних годин повинна бути в чотири рази більше частоти генератора. У зв'язку з цим і спільним використанням контактів генератор таймера / лічильника може використовуватися тільки тоді, коли в якості джерела системних годин обраний калібрований внутрішній RC-генератор. Застосування зовнішнього джерела синхронізації до TOSC1 може бути виконано, якщо біт входу «Включити зовнішні годинник» в регістрі асинхронного стану (ASSR.EXCLK) записується в «1».

Пристрій може виводити системний годинник на вивід CLKO. Щоб включити вихід, необхідно запрограмувати біт CKOUT. Цей режим підходить, коли годинник мікросхеми використовуються для управління іншими ланцюгами в системі. Годинник також будуть виводитися під час скидання, і нормальна робота виведення буде скасована при програмуванні. Будь-яке джерело синхронізації, включаючи внутрішній RC-генератор, можна вибрати, коли годинник виводяться на CLKO.

Пристрій має попередній дільник системних годин, а системний годинник можна розділити, налаштувавши регістр Prescale Clock (CLKPR). Ця функція може використовуватися для зменшення тактової частоти системи і енергоспоживання, коли вимога до потужності обробки низька. Це можна використовувати з усіма годинами джерела, і це вплине на тактову частоту процесора і всіх синхронних периферійних пристроїв. При перемиканні між настройками попереднього дільника система Prescaler System Clock забезпечує відсутність збоїв в системі годин. Це також гарантує, що ніяка проміжна частота не буде більше, ніж частота, відповідна попередньої налаштуванні, ні тактова частота, відповідна новому налаштуванню. Лічильник пульсацій, який реалізує переддільник, працює на частоті неподільних годин, що може бути швидше, ніж частота процесора. Отже, неможливо визначити стан преддльника

— навіть якби воно було читаним, точне передбачення часу, який потрібен для перемикання з одного такту на інший, неможливо точно передбачити. У цей момент створюються два активних інтервалу часу. Час T1 — це попередній період часу, а T2 — період, відповідний новому налаштуванню попереднього подільника. Щоб уникнути випадкових змін тактової частоти, необхідно виконати спеціальну процедуру запису, щоб змінити біт CLKPS: Час T1 — це попередній період часу, а T2 — період, відповідний новому налаштуванню попереднього подільника. Щоб уникнути випадкових змін тактової частоти, необхідно виконати спеціальну процедуру запису, щоб змінити біт CLKPS:

1. Записати біт Prescaler Change Enable (CLKPCE) в значення «1» і всі інші біт CLKPR в нуль.
2. Протягом чотирьох циклів написати потрібне значення CLKPS [3: 0] під час запису нуля в CLKPCE.

Переривання повинні бути відключені при зміні налаштувань преддільника, щоб переконатися, що процедура запису не переривається.

Система контролю і скидання (SCRT)

Під час скидання все регістри введення / виводу встановлюються на їх початкові значення, і програма запускає виконання з вектора скидання. Інструкція, розміщена в векторі скидання, повинна бути інструкцією абсолютного переходу (JMP) для процедури обробки скидання. Якщо програма ніколи не включає джерело переривання, вектори переривань не використовуються, і в цих місцях може бути розміщений звичайний програмний код. Це також має місце, якщо вектор скидання знаходиться в секції додатки, в той час як вектори переривань знаходяться в розділі завантаження або навпаки. Порти введення-виведення AVR негайно повертаються в початковий стан, коли джерело скидання активується. Для цього не потрібно запуск будь-якого джерела синхронізації. Після того, як всі джерела скидання неактивні, викликається лічильник затримки, розтягуючи внутрішній скидання. Це дозволяє досягти стабільного рівня потужності до

початку нормальної роботи. Період тайм-ауту лічильника затримки визначається користувачем через біти SUT і CKSEL.

Сторожовий таймер

Якщо сторожовий таймер не використовується в програмі, модуль повинен бути відключений. Якщо сторожовий таймер включений, він буде включений у всіх режимах сну і, отже, завжди буде споживати електроенергію. У режимах більш глибокого сну це значно вплине на загальне споживання струму.

Пристрій має розширений сторожовий таймер (Wachdog Prescaler). Wachdog Prescaler — це цикли підрахунку таймера окремого вбудованого генератора 128 кГц. Він дає переривання або системне скидання, коли лічильник досягає заданого значення тайм-ауту. У нормальному режимі роботи потрібно, щоб система використовувала інструкцію скидання сторожового таймера (Watchdog Timer Reset) для перезапуску лічильника до досягнення значення тайм-ауту. Якщо система не перезапускає лічильник, видається переривання або скидання системи.

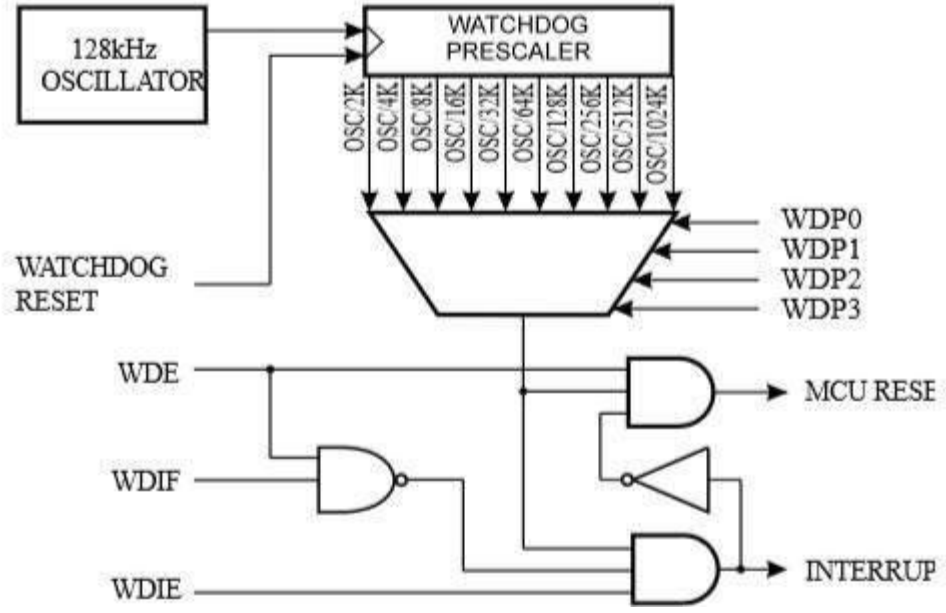


Рисунок 3.6 – Функціональна схема роботи сторожового таймера

В режимі переривання таймер дає переривання після закінчення часу. Це переривання можна використовувати для пробудження пристрою з режимів очікування, а також в якості загального системного таймера. Одним із прикладів є обмеження максимального часу, дозволеного для певних операцій, що дає переривання, коли операція працює довше, ніж очікувалося. У режимі скидання системи сторожовий таймер дає скидання, коли закінчується час таймера. Зазвичай це використовується для запобігання зависання системи в разі пропуску виконання коду. Третій режим, режим переривання і скидання системи об'єднує два інших режиму, спочатку передаючи переривання, а потім перемикається в режим скидання системи. Цей режим, наприклад, дозволить безпечне вимикання, зберігаючи критичні параметри перед скиданням системи.

Запобіжник таймера завжди включений (WDTON), і якщо він запрограмований, змусить сторожовий таймер переключитися в режим скидання системи. З запрограмованим запобіжником біт режиму скидання (WDE) і біт переривання (WDIE) блокуються на 1 і 0 відповідно. Щоб забезпечити безпеку програми, зміни в налаштування сторожового таймера повинні виконуватися послідовностями за часом.

Порти вводу / виводу

Всі порти AVR мають справжню функцію Read-Modify-Write при використанні в якості загальних цифрових портів введення-виведення. Це означає, що напрямок одного порту може бути змінено без ненавмисного зміни напрямку будь-якого іншого порту з інструкціями ВГО і СВІ. Те ж саме відбувається при зміні значення с LOW на HIGH (якщо воно налаштоване як вихід) або включення / вимикання підтягують резисторів (якщо налаштоване як вхід). Драйвер введення досить міцний, щоб безпосередньо керувати світлодіодними дисплеями. Всі виводи портів мають індивідуально обрані підтягують резистори з опором напруги харчування. Всі виводи мають захисні діоди як для напруги живлення, так і для заземлення, як показано на наступному рисунку.

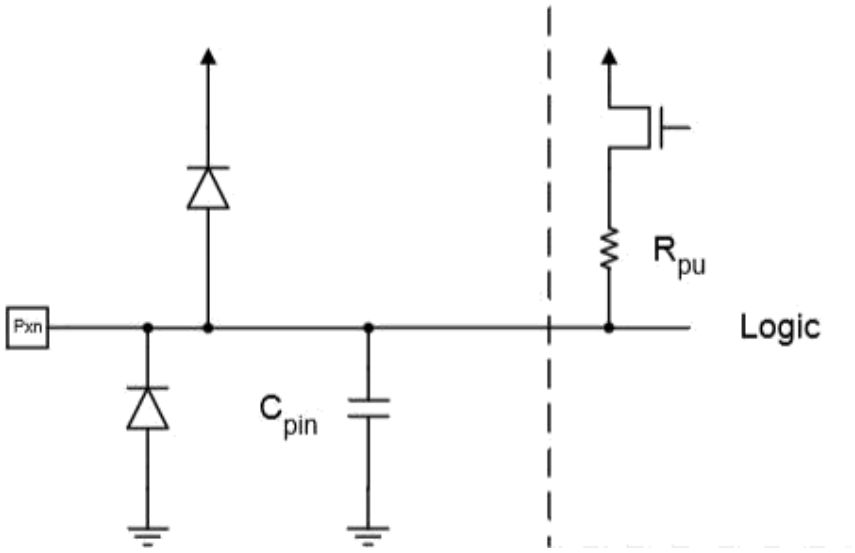


Рисунок 3.7 – Схема введення / виведення

Кожен порт мікроконтролера є двонаправлені порти введення-виведення з додатковими внутрішніми підтягуються резисторами. На попередньому рисунку показано функціональне опис одного порту, названий Pxn.

Кожен порт складається з трьох бітів регістрів: DDxn, PORTxn і PIN. До бітів DDxn звертаються за адресою DDRx, бітам PORTxn на адресу введення-виведення PORTx і бітам PINxn на адресу введення-виведення PINx. Біт DDxn в регістрі DDRx вибирає напрямок цього виводу. Якщо DDxn записується в «1», Pxn конфигурируется як вихідний вивід. Якщо DDxn записується в «0», Pxn конфигурируется як вхідний вивід. Якщо PORTxn записується в «1», коли контакт налаштований як вхідний контакт, активується навантажувальний резистор. Щоб відключити підтягаючий резистор, PORTxn повинен бути записаний в «0», або контакт повинен бути налаштований як вихідний вивід. Порти тричі вказуються, коли режим скидання стає активним, навіть якщо таймери не працюють. Якщо PORTxn записується в «1», коли порт налаштований як вихідний вивід, присвоюється значення високого рівня.

Якщо деякі контакти не використовуються, рекомендується переконатися, що ці контакти мають певний рівень. Незважаючи на те, що більшість цифрових входів відключені в режимах глибокого сну, слід уникати плаваючих входів, щоб зменшити споживання струму у всіх інших режимах, де

цифрові входи активовані (скидання, активний режим і режим очікування). Найпростіший спосіб забезпечити певний рівень невикористаного виведення — це включити внутрішнє підтягування. В цьому випадку підтягування буде відключено під час скидання. Якщо важлива низька споживана потужність під час скидання, рекомендується використовувати зовнішнє підтягування. Не рекомендується підключати невикористовувані контакти безпосередньо до джерела живлення або заземлення, так як це може призвести до надмірних струмів, якщо контакт випадково налаштований як вихід.

EXINT — зовнішні переривання

Зовнішні переривання запускаються за допомогою контактів INT або будь-якого з контактів PCINT. Якщо включено, переривання будуть спрацьовувати, навіть якщо контакти INT або PCINT налаштовані як виходи. Ця функція забезпечує спосіб програмного переривання. Запит переривання буде спрацьовувати, якщо який-небудь активоване контакт переключиться. Запит переривання зміни буде спрацьовувати, якщо активований будь-який контакт.

Це означає, що ці переривання можуть використовуватися для пробудження також з режимів сну, крім режиму очікування.

Зовнішні переривання можуть бути викликані спадаючим або наростаючим фронтом або низьким рівнем. Це налаштоване так, як вказано у специфікації для регістра управління зовнішніми перериваннями A (EICRA). Коли зовнішні переривання включені і налаштовані як спрацьовування рівня, переривання будуть спрацьовувати до тих пір, поки контакт буде утримуватися на низькому рівні. Переривання низького рівня в INT визначається асинхронно. Це означає, що це переривання можна використовувати для пробудження частини також з режимів сну, крім режиму очікування. Таймер введення / виведення зупиняються у всіх режимах сну, крім режиму очікування. Якщо для пробудження від відключення живлення використовується переривання з рівнем, необхідний рівень повинен бути достатньо тривалим, щоб регістр контролю MCU завершив пробудження, щоб викликати переривання рівня.

Якщо рівень зникне до закінчення часу запуску, MCU все одно прокинеться, але переривання не буде згенеровано.

TC0 — 8-бітний таймер / лічильник з ШІМ

ШІМ на Arduino цілком і повністю реалізований завдяки вбудованим таймерам yArduino Nano. Їх два 8-бітних (Timer0 і Timer2). Кожен таймер управляється кількома регістрами. Основні регістри управління — TCCRnA і TCCRnB. Нижче представлені біти, що належать цим регістрів, які нам знадобляться:

CSn [2: 0] (Clock Select) — біти регістра TCCRnB відповідають за вибір джерела синхронізації, преддільника і режиму роботи таймера.

WGMn [2: 0] (Waveform Generation Mode) — вибір режиму генерації хвилі. Біти розташовані в регістрах TCCRnA і TCCRnB, COMnA [1: 0] і COMnB [1: 0] — режим збігу виведення A / B — включений / виключений / інвертований (біти розташовані в регістрі TCCRnA).

Timer / Counter0 (TC0) — це 8-бітний модуль таймера / лічильника загального призначення з двома незалежними модулями порівняння вихідних даних і підтримкою ШІМ. Він дозволяє точно визначати терміни виконання програми (управління подіями) і генерувати ШІМ. Нижче приведена спрощена блок-схема 8-бітний таймер / лічильника. TC0 активується шляхом запису біта PRTIM0 на «0». TC0 активується, коли біт PRTIM0 в регістрі зниження потужності (PRR.PRTIM0) записується в «1».

Регістр таймера / лічильника (TCNTn) і регістр вихідних даних TC0x (OCRnx) є 8-бітними. Сигнали запиту переривання (скорочено Int.Req. На малюнку) видимі в регістрі прапорів переривання таймера 0 (TIFRn). Всі переривання індивідуально маркуються за допомогою регістра маски переривання таймера (TIMSKn). TIFR0 і TIMSK0 не показані на малюнку. Таймер / лічильник може бути синхронізований всередині, через попередній дільник (Prescaler) або зовнішнім джерелом синхронізації на виводі Tn. Логічний блок «Вибір годин» (Clock Select) визначає, яке джерело синхронізації буде використовуватися таймером / лічильником для збільшення

(або зменшення) його значення. Таймер / лічильник неактивний, якщо Ви не оберете джерело синхронізації. Вихід з логіки вибору годин називається годинним таймером (clkT0). Регістри порівняння з подвійним буфером (OCRnA і OCRnB) завжди порівнюються із значенням таймера / лічильника. Результат порівняння може бути використаний генератором сигналів (Waveform Generation) для генерації ШІМ або вихід змінної частоти на виводах порівняння виведення (OCnA і OCnB).

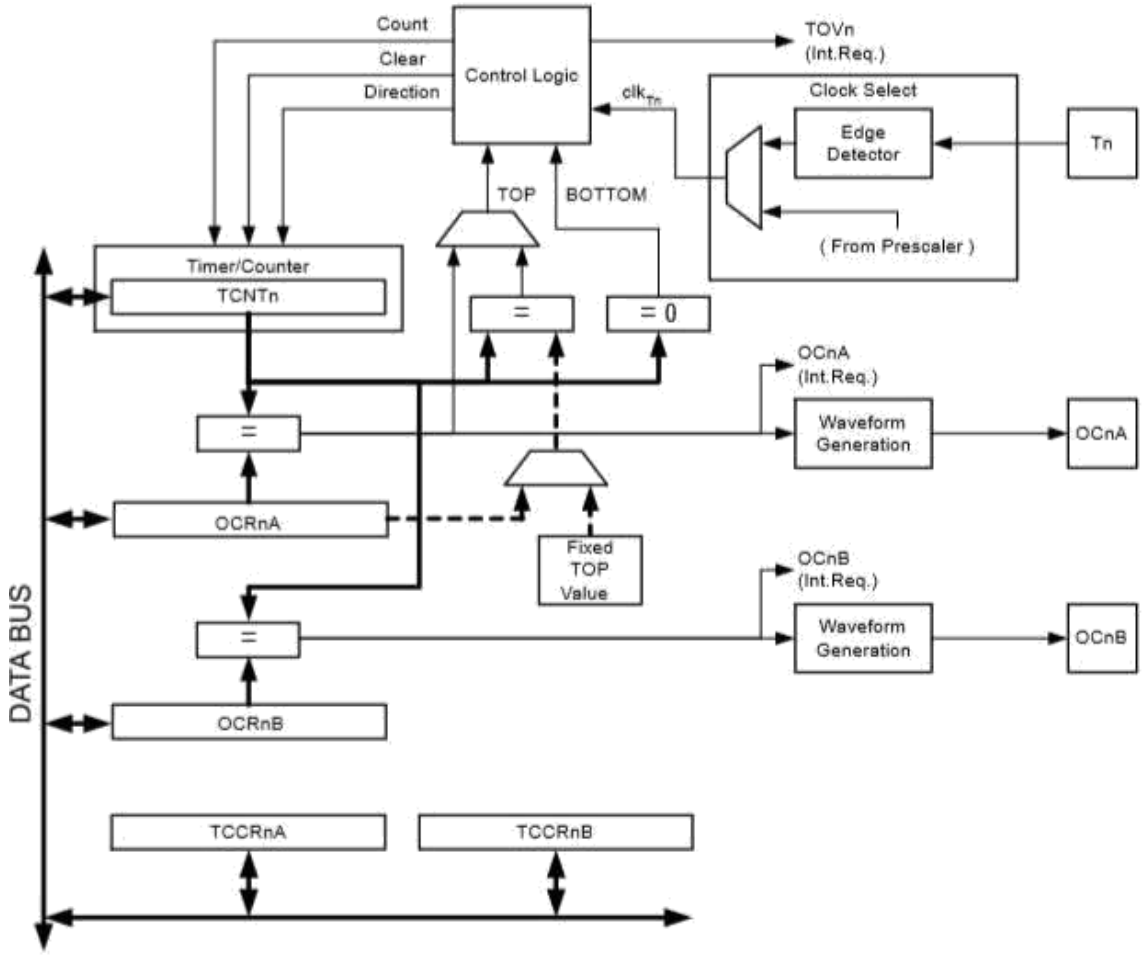


Рисунок 3.8 – Блок схема 8-бітного таймера / лічильника

Основна частина 8-бітного таймера / лічильника — це програмований двонаправлений лічильник. Нижче наведена блок-схема лічильника і його компонентів. Лічильний блок таймера — це програмований двонаправлений лічильник. Нижче наведена блок-схема лічильника і його оточення. 8-бітний компаратор безперервно порівнює TCNTn з регістрами порівняння вихідних

даних (OCRnA і OCRnB). Всякий раз, коли TCNTn дорівнює OCRnA або OCRnB, компаратор сигналізує про збіг. Збіг встановить прапорець порівняння (OCFnA або OCFnB) в наступному циклі таймера. Якщо відповідне переривання включено, прапор порівняння вихідних даних генерує переривання порівняння вихідних даних. Прапор порівняння автоматично очищається при виконанні переривання. В якості альтернативи, прапор може бути очищений програмним забезпеченням, написавши «1» в його місцезнаходження біт введення-виведення. Генератор сигналів використовує сигнал відповідності для генерації ШІМ на вфводе відповідно до режиму роботи, встановленим бітами WGM02, WGM01 і WGM00, і бітами компаратора. Максимальні (top) і мінімальні (bottom) сигнали використовуються генератором сигналів для обробки особливих випадків екстремальних значень в деяких режимах роботи.

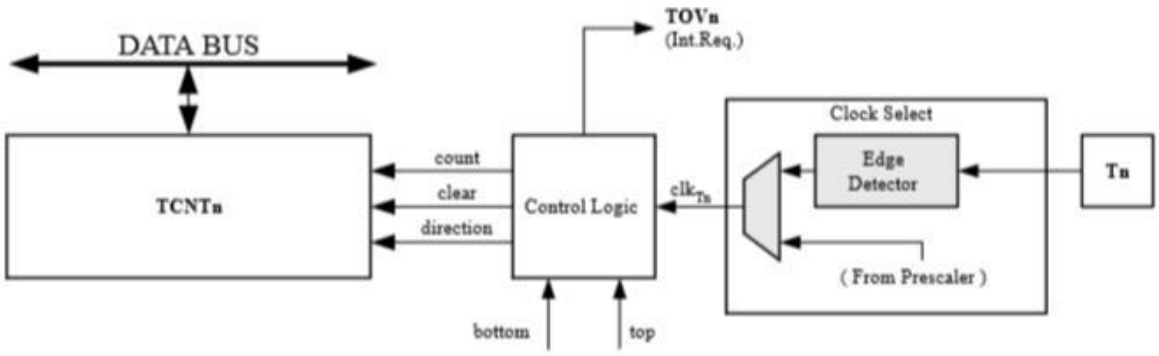


Рисунок 3.9 – Блок схема лічильника

Регістри OCR0x мають подвійний буфер при використанні будь-якого з режимів ШІМ. Коли включена подвійна буферизація, процесор має доступ до регістру буфера OCR0x, що відповідає за синхронізацію. Синхронізація запобігає появі несиметричних імпульсів ШІМ непарної довжини, тим самим роблячи вихід без перешкод. Подвійна буферизація відключена для режиму нормальної роботи, а центральний процесор буде безпосередньо звертатися до регістру OCR0x.

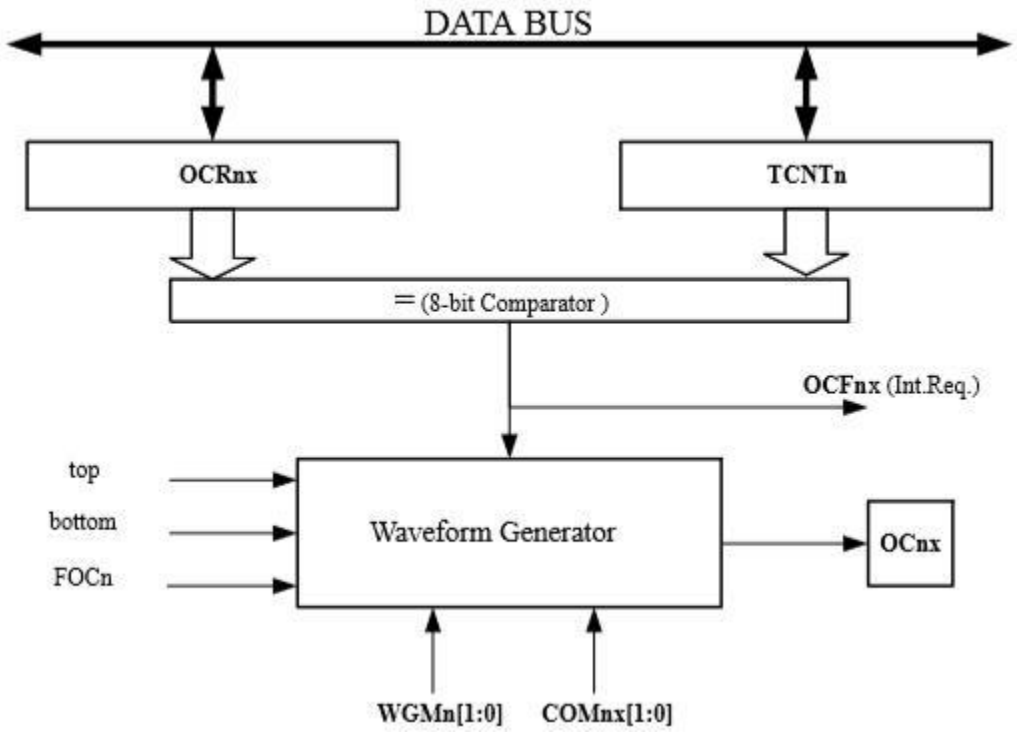


Рисунок 3.10 – Блок схема 8-бітного компаратора.

Всі операції центрального процесора в регістрі TCNTn блокуватимуть будь-яке порівняння, яке зустрічається в наступному циклі таймера, навіть коли таймер зупинений. Ця функція дозволяє форматувати OCR1x з тим же значенням, що і TCNTn, без спрацьовування переривання при включенні таймера / лічильника.

Оскільки запис TCNT1 в будь-якому режимі роботи буде блокувати всі порівняння збігів для одного тактового таймера, існують ризики, пов'язані зі зміною TCNT1 при використанні модуля порівняння вихідних даних незалежно від того, чи працює таймер / лічильник чи ні. Якщо значення, записане в TCNT1, дорівнює значенню OCR1x, порівняння буде пропущено, що призведе до неправильної генерації сигналу.

Біти в регістрі контролю А таймера / лічильника (TCCR0A.COM0x)мають дві функції:

— генератор сигналів використовує біти COMnx для визначення стану регістра вихідних даних (OCnx) при наступному зіставленні порівняння;

— біти COMnx управляють вихідним джерелом виведення OCnx.

На наведеному нижче рисунку 3.11 показана спрощена схема логіки, порушеної COMnx. Відображаються лише частини загальних регістрів управління портами введення-виведення, на які впливають біти COMtx, а саме PORT і DDR.

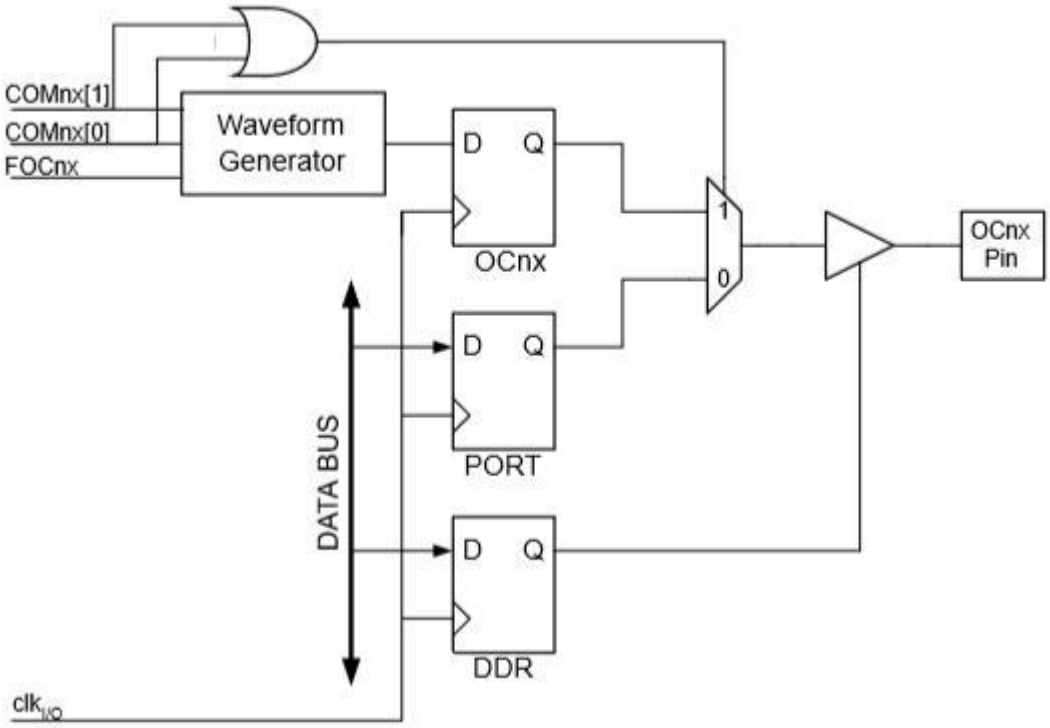


Рисунок 3.11 – Схема модуля порту введення / виведення

Загальна функція порту введення / виведення визначається вихідним значенням біта для OCnx pin з генератора форми сигналу, якщо встановлений біт COMnx [1: 0]. Однак напрямок порту OCnx (вхід або вихід) як і раніше контролюється регістром напрямки даних (DDR) для виведення порту. У регістрі напрямки даних біт для OC1nx pin (DDR.OC0x) повинен бути встановлений як вихідний сигнал до того. Функція перевизначення портів не залежить від режиму генерації форми хвилі. Конструкція логіки виведення дозволяє форматувати стан регістра OC0x до того, як вихід буде включений.

Режим роботи генератора ШІМ визначає поведінку контактів таймера / лічильника і виведення. Він визначається комбінацією бітів режиму генерації форми сигналу і режиму порівняння вихідного сигналу в регістрах управління

таймером / лічильником A і B. Біти COMnx [1: 0] визначають, чи повинен ШІМ вивід бути інвертований чи ні.

Найпростішим режимом роботи є режим нормальний (Normal).

В цьому режимі напрямок підрахунку завжди збільшується, а лічильник не очищається. Лічильник просто перехоплюється, коли він передає своє максимальне 8-бітове значення, а потім перезавантажується. В режимі нормального режиму прапор переповнення таймера / лічильника (TOV1) буде встановлено в той же такт, в якому TCNTn стає нульовим. В цьому випадку прапор TOV1 поводиться як дев'ятий біт, за винятком того, що він встановлений, а не очищений. Однак в поєднанні з перериванням переповнення таймера, який автоматично очищає прапор TOV1, дозвіл таймера може бути збільшено за допомогою програмного забезпечення. У нормальному режимі нове значення лічильника може бути записано в будь-який час.

Аналогово-цифровий перетворювач

Мікроконтролер оснащений 10-бітовим АЦП з послідовним наближенням. АЦП підключений до 8-канальному аналоговому мультиплексору. Входи з одностороннім напругою відносяться до 0 В (GND). АЦП містить схему, яка гарантує, що вхідна напруга на АЦП утримується на постійному рівні під час перетворення. АЦП має окремий аналоговий вивід напруги живлення, AVCC.

Канал аналогового введення вибирається шляхом запису в біти MUX в регістрі вибору ADC мультиплексора ADMUX.MUX [3: 0]. Мультиплексор по команді АЦП з декількох аналогових входів використовує тільки один. Будь-який з вхідних АЦП портів, а також GND і опорного напруги може бути обраний в якості одного складу входів в АЦП. АЦП активується шляхом запису «1» в біт АЦП «Активувати» в регістрі управління АЦП і регістра стану (ADCSRA.ADEN). Джерело опорного напруги і вхідного каналу вибір не вступить в силу до тих пір, поки ADEN встановлений. АЦП генерує 10-бітний результат, який представлений в регістрах даних ADC, ADCH і ADCL.

Для здійснення перетворення АЦП використовує еталонне значення напруги, з яким порівнюється вхідний сигнал з аналогового входу мікроконтролера. Для цього призначений джерело опорного напруги, в якості якого використовується напруга живлення.

Роздільна здатність 10 біт дозволяє отримувати значення сигналу від 0 до 1023 або $\mathbf{2^{10}}$.

Процес оцифровки відбувається підбором близького значення напруги по відношенню до вхідного і складається з 10 етапів. Спочатку АЦП формує напруга рівне половині опорного і порівнює його з вхідною напругою. Якщо напруга АЦП менше вхідного то відповідному біту присвоюється значення «1». Потім напруга АЦП збільшується на 1/4 опорного напруги і якщо воно виявиться більше вхідного, то біту присвоюється значення «0», а напруга АЦП зменшується на 1/4 опорного. Далі корекція становить 1/8 опорного, потім 1/16 і т.д.

Для управління АЦП існує 5 основних восьмибітних регістрів:

— Регістри ADCSRA (ADC Control and Status Register A), ADMUX, ADCSRB, ADCL, ADCH.

— Призначення бітів регістра ADCSRA (ADC Control and Status Register A, регістр управління і стану):

— ADEN (ADC Enable, включення АЦП) — прапор, який дозволяє використання АЦП, при скиданні прапора під час перетворення процес зупиняється.

— ADATE (ADC Auto Trigger Enable) — вибір режиму роботи АЦП. «0» — разове перетворення (Single Conversion Mode); «1» — включення автоматичного перетворення по спрацьовуванню тригера (заданого сигналу). Джерело автоматичного запуску задається бітами ADTS регістра ADCSRB.

— ADSC (ADC Start Conversion, запуск перетворення) встановлений в «1» запускає процес перетворення. У режимі Single Conversion (ADATE = 0) для запуску кожного нового перетворення цей прапор повинен бути

встановлений в одиницю. Після завершення перетворення, прапор ADSC скидається в «0».

В режимі Free Running Mode (ADATE = 1, ADTS [2: 0] = 000) цей прапор повинен бути встановлений в одиницю один раз — для запуску першого перетворення, такі відбуваються автоматично. Якщо установка прапора ADSC відбувається під час або після дозволу АЦП (ADEN), то перед запитуваною перетворенням відбувається додаткове (extended) перетворення, під час якого відбувається ініціалізація АЦП. Скидання прапора під час перетворення не робить ніякого впливу на процес.

Free Running Mode — режим вільного запуску, що запускає нове перетворення і воно буде розпочато відразу ж після завершення перетворення, в той час як значення ADCRSA.ADSC залишається високим.

ADIF (ADC Interrupt Flag) — прапор переривання від компаратора.

ADIE (ADC Interrupt Enable) — дозвіл переривання від компаратора.

ADPS [2: 0] (ADC Prescaler Select) — комбінацією бітів задається частота перетворення АЦП по відношенню до частоти мікроконтролера. Обрана частота впливає на точність — чим вище частота, тим нижче точність. АЦП тактується через обраний дільник від частоти ядра мікроконтролера.

Частота роботи МК Atmega 328P 16МГц. При налаштуваннях за замовчуванням використовується переддільник 128 (ADPS [2: 0] = [111]), а це означає, що АЦП працює на частоті 16МГц / 128 = 125КГц. За замовчуванням для послідовної схеми апроксимації потрібно вхідні тактова частота від 50 кГц до 200 кГц, щоб отримати максимальний дозвіл. Це дає низьку швидкість перетворення, але високу точність. Якщо потрібна менша дозвіл, ніж 10 біт, вхідні тактова частота для АЦП може бути вище 200 кГц, щоб отримати більш високу частоту дискретизації. Модуль ADC містить попередній дільник, який генерує прийнятну тактову частоту АЦП з будь-якою частотою процесора вище 100 кГц. Попередня калібрування вибирається битами ADC Prescaler Select в регістрі управління АЦП і регістром стану А (ADCSRA.ADPS). Переддільник починає відлік з моменту включення АЦП, записавши біт ADCSRA.ADEN

АЦП ADAD на «1». Переддільник продовжує працювати до тих пір, поки ADEN = 1, і безперервно скидається, коли ADEN = 0. При ініціюванні одноразового конвертації шляхом запису «1» в біт конверсії запуску ADC (ADCSRA.ADSC) перетворення починається з наступного зростаючого фронту циклу синхронізації АЦП.

Регістр ADMUX (ADC Multiplexer Selection Register)

Призначення бітів регістра ADMUX:

REFS [1: 0] (Reference Selection Bit) — біти визначають джерело опорного напруги.

ADLAR (ADC Left Adjust Result) — біт відповідає за порядок запису бітів результату в регістри ADCL і ADCH.

MUX [3: 0] (Multiplexer Selection Input) — біти вибору аналогового входу.

3.2 Розробка принципової схеми

Основою побудови будь-якої системи є розробка функціональної схеми, яка дозволяє зрозуміти всі тонкощі роботи та призначення вузлів системи, а також як вони між собою взаємодіють. Систему перевірки провідних ліній зв’язку можна представити, як показано на рисунку 3.12.

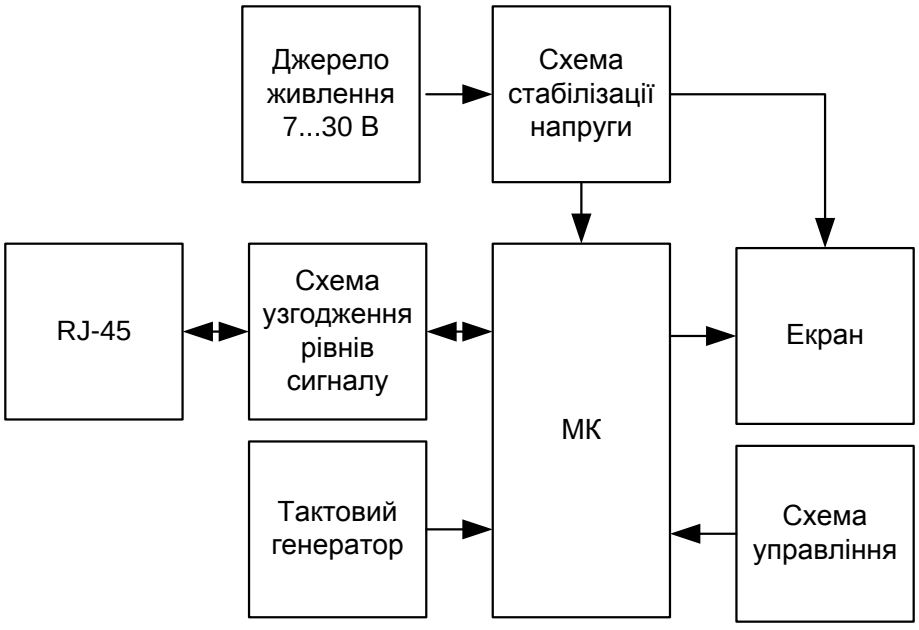


Рисунок 3.12 – Функціональна схема пристрою

Батарея живлення забезпечує використання широкого діапазону напруг, що дозволяє використовувати найбільш поширені джерела живлення напругами 7,2, 10,5, 12, 19 та 24 В. Вибір здійснюється залежно від умов використання.

RJ-45 — вузол системи, що дозволяє підключати найбільш вживані провідні лінії зв’язку типу вита пара.

Блок узгодження рівнів сигналу допомагає коректно підключати провідні лінії зв’язку з сигналами, які будуть сумісні з сигналами потрів мікроконтролера.

Тактовий генератор потрібен для створення опорної частоти для роботи таймерів/лічильників та тактового сигналу системи вцілому.

Екран дозволяє відображати результати вимірювання у форматі, що є зрозумілим та необхідним для користувача.

Схема управління призначена для вибору режимів роботи і початкових налаштування та використання керування процесом роботи системи.

Функціональна схема є основою для побудови електричної принципової схеми.

Схема електрична принципова будується на електронних елементах, які є доступними на сучасному ринку електроніки. Для схеми стабілізації напруги обрано LP2950, що є мікропотужним регулятором напруги з дуже малою напругою падіння (40 мВ на малих навантаженнях і 380мВ при 100 мА), і дуже низьким струмом спокою (близько 75мкА). Вони можуть охопити широкий діапазон вхідної напруги аж до 30 В.

Струм споживання LP2950 зростає не дуже сильно при значному падінні напруги, тим самим продовжуючи термін служби батареї. Ця особливість, робить стабілізатора LP2950 ідеальним рішенням для використання в пристроях з акумуляторним живленням.

Зовнішній вигляд показано на рисунку 3.13.

Рисунок 3.13 – Зовнішній вигляд стабілізатора напруги LP2950

Ключова особливість LP2950

— Висока точність 5В зі стабільним вихідним током100 мА.
— Надзвичайно низький струм спокою.
— Низьке падіння напруги.
— Дуже низький температурний коефіцієнт.
— Використання в якості регулятора або опорного напруги.
— Для стабільності необхідний тільки 1 мкФ.
— Обмеження по струму і температурі.

LP2950, що випускається в 3-вивідному корпусі типу TO-92 і TO-252, забезпечує фіксовану вихідну напругу 5В, 3,3 і 3В в залежності від версії.

У випадку відсутності можна використати аналог LP2951 у 8- вивідному корпусі, зовнішній вигляд якого показано на рисунку 3.14.

Рисунок 3.14 – Зовнішній вигляд стабілізатора напруги LP2951

Типові варіанти включення LP2951, показано на рисунках 3.15, 3.16.

Поєднуючи вихід 1 (OUT) з виводом 7 зворотного зв'язку по напрузі (FEEDBACK) і з виводом 6 внутрішнього дільника (VTAP) можна на виводі 2 (SENSE) отримати фіксований напруга 5В, 3,3 або 3В (в залежності від версії).

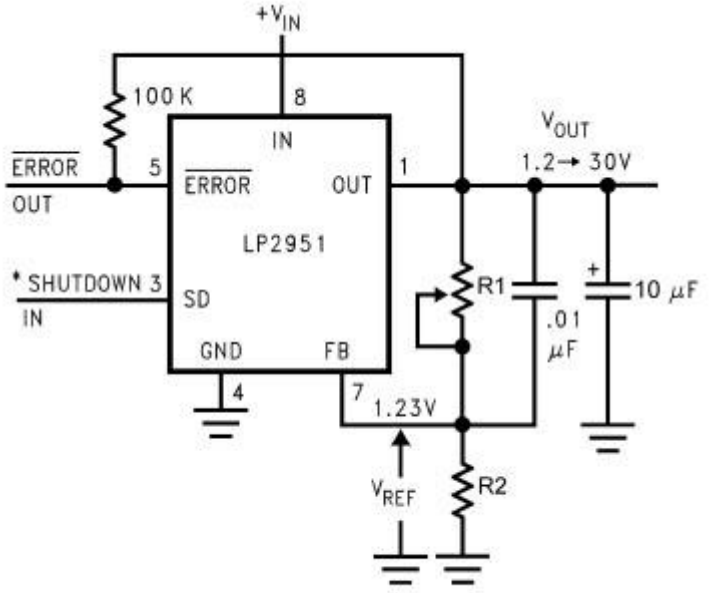


Рисунок 3.15 – Включення стабілізатора напруги LP2951 на фіксовану напругу

В якості альтернативи, залишаючи виводи 2 (SENSE) і 6 (VTAP) вільними, а виводи 1 (OUT) і 7 (FEEDBACK) з'єднавши з зовнішнім дільником напруги можна отримати будь-яке значення вихідної напруги від 1,235 до 30 вольт.

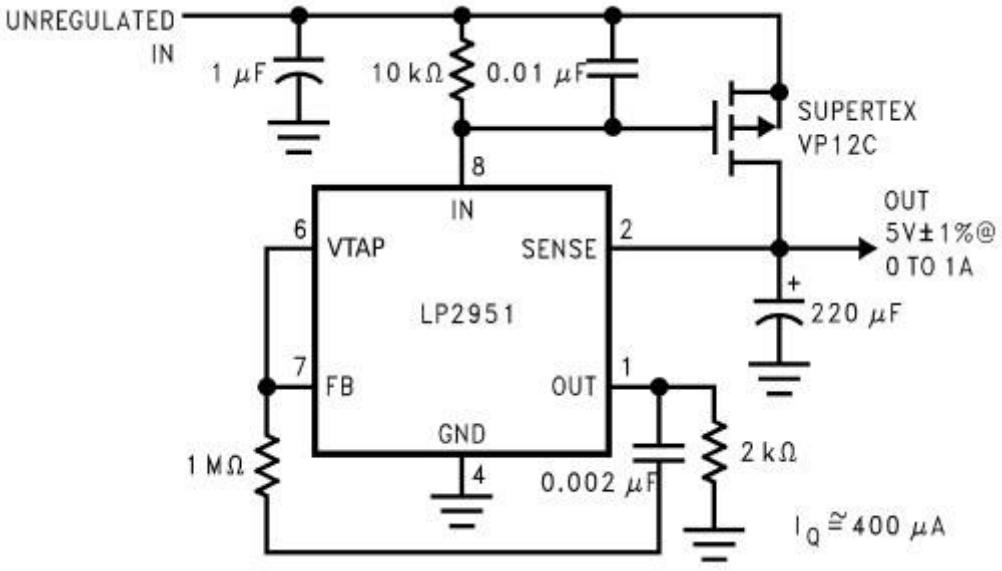


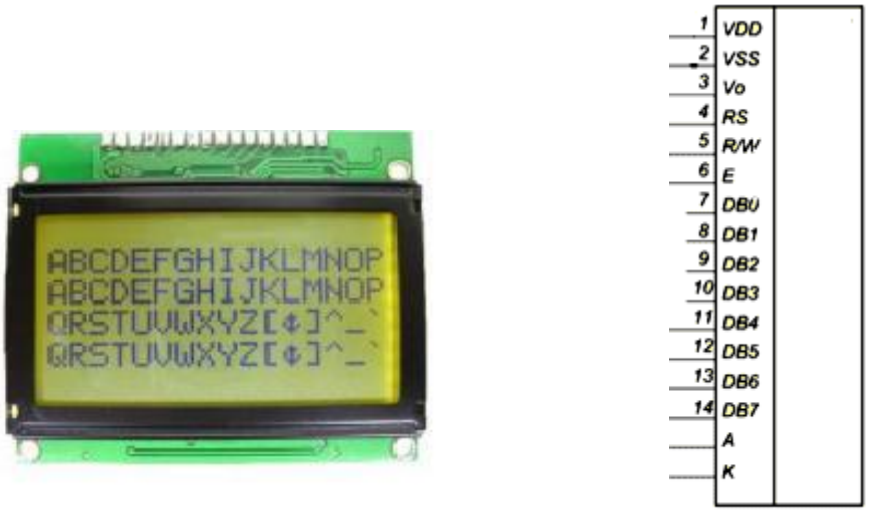
Рисунок 3.16 – Включення стабілізатора напруги LP2951 на перемінну напругу

Восьми контактний стабілізатор LP2951 наділений додатковими функціональними можливостями, що робить її особливо підходящим для додатків на батарейках. Наприклад, функція логічного виключення, дозволяє регулятору переходити в режим очікування з метою економії електроенергії.

Крім того, є вбудований супервизор скидання, який визначає помилку при зниженні вихідної напруги на 6% від номінального значення з яких-небудь причин, наприклад, у зв'язку з падінням вхідної напруги, обмеження струму або відключення при перегріві.

Екран обрано модель WH-1604A-YYH-CT#, який побудований на контролері HD-44780 та може бути замінений аналогічним інших фірм.

Зовнішній вигляд та позначення виводів показано на рисунку 3.17.



а) б)

Рисунок 3.17 – Екран WH-1604A-YYH-CT#: а) зовнішній вигляд; б) призначення виводів

Порт A мікроконтролера — це входи АЦП, на порту B у нас ISP і пара службових функцій, порт C використовуємо для формування тестових сигналів, ну а порт D — для спілкування з користувачем за допомогою HD44780-сумісного дисплея.

Вхідні ланцюги схеми узгодження рівнів сигналу захистимо за допомогою супресорів VD1.1 — VD1.8, 1,5КЕ6,8СА. Від потрапляння в 220В вони, звичайно, не врятують, а от 60В з якою-небудь телефонної лінії погасити цілком зможуть.

Ланцюжок VD2 — R4 служить для виявлення розряду батареї. На стабілітроні падає 5,1В, Таким чином, коли напруга батареї впаде нижче 6В, на PB2 з'явиться лог. 0.

Варто сказати про номінал опору R5, що задає яскравість підсвічування. Чим більше номінал, тим довше буде жити батарейка — вся інша схема споживає менше 5 мА, основний споживач саме підсвічування дисплея.

Схема електрична принципова показано в додатку Б.

3.3 Розробка алгоритму роботи

Алгоритм роботи пояснює логіку роботи системи та є основою для розробки програмного забезпечення. Блок схема показана на рисунку 3.18.

Робота тестера ділиться на кілька етапів, які повторюються циклічно.

Етап 1. Початкові перевірки

— перевіримо, чи не підключено до лінії якесь активне обладнання. Всі керуючі лінії (порт С, нагадаю) переводимо в Hi-Z стан, вимірюємо напругу на всіх лініях. Вони повинні бути близько нульових. В іншому випадку розуміємо, що з іншого боку проводу підключено що завгодно, але не наша відповідна частина, і далі продовжувати сенсу не має. Зате має сенс повідомити користувачеві, що «на лінії є напруга!».

— перевіримо рівень сигналу на PB2. Якщо там 0, то батарея розряджена. Повідомимо про неполадку користувачеві.

Етап 2. Перевірка цілісності ліній і наявності коротких замикань

Для кожної з 8 ліній проробляємо наступне. Подаємо на неї +5 з порту С, зберігаючи всі інші лінії порту в високоімпедансних стані, і вимірюємо напруга на інших лініях. Якщо на всіх лініях близьконульових значення —

досліджувана лінія обірвана. Якщо ж на якийсь із ліній теж з'явилося +5 - це КЗ. У нормі ми побачимо якісь проміжні значення.

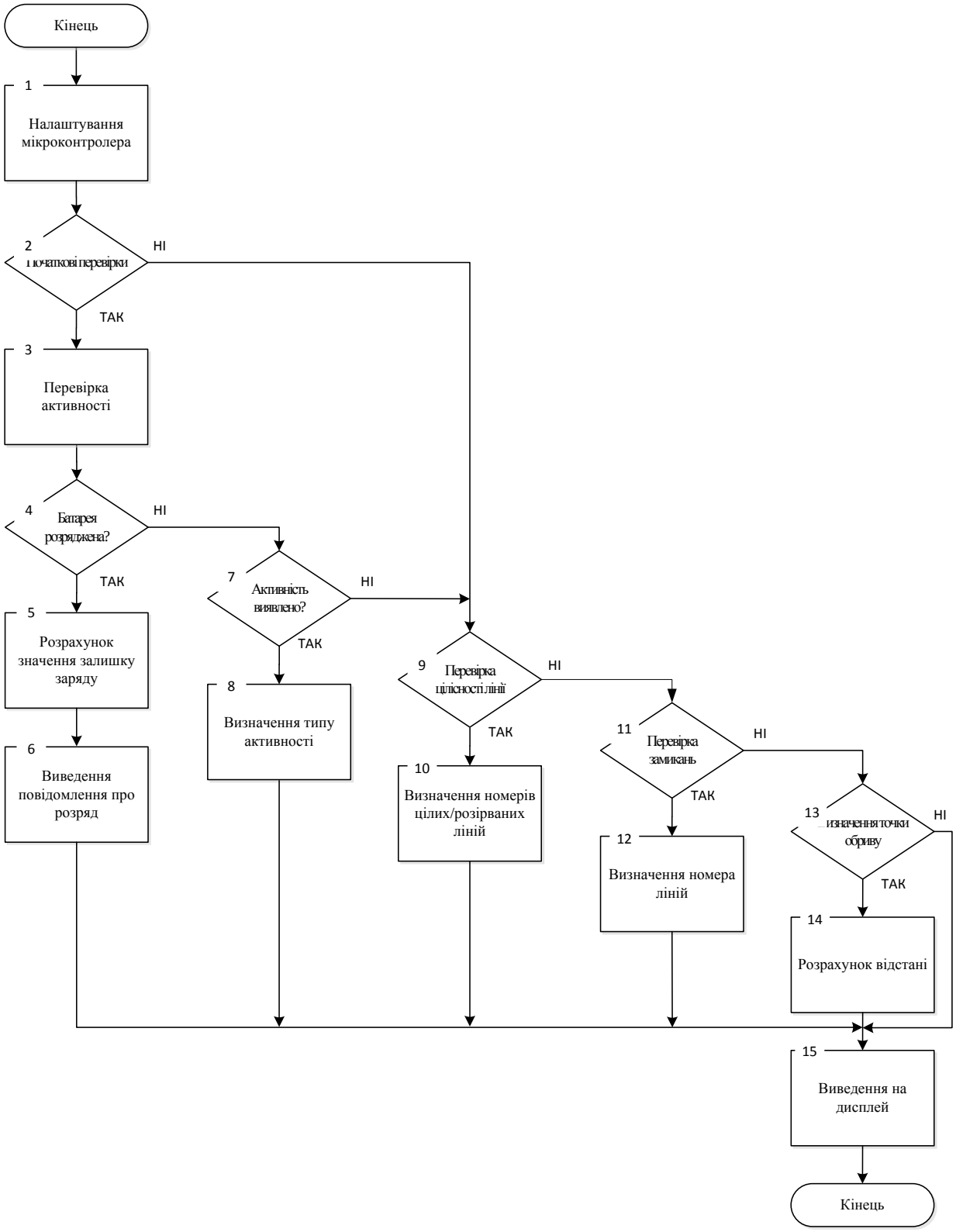


Рисунок 3.18 – Блок схема алгоритму роботи системи

Етап 3. З'ясування схеми кросування

Ось і підібралися до найцікавішого. Відсіявши все свідомо несправні лінії (перебиті і закорочені дроти), приступимо до вимірювання опорів залишилися ліній (нехай їх кількість N, 0 <= N <= 8).

Введемо позначення:

R_{xy} — опір між лініями x і y.

R_x — номінал опору, підключеного до лінії x.

Ясно, що $R_{xy} = R_x + R_y$.

Заміряючи опору між лініями, ми отримуємо систему лінійних рівнянь. Порівнявши отримані значення R_1 ... R_N з еталонними, ми з'ясуємо схему кросування.

Опір обчислити нескладно. Подамо на лінію X високий рівень, на лінію Y — низький, а інші лінії порту C залишимо в Hi-Z. У ланцюзі (див. Рис. 3) падіння напруги на відомому нам опорі, утвореному паралельним включенням R1.Y і R2.Y за схемою становить U_1, а на невідомому R_{xy} падає (U_2 - U_1). Значить, $R_{xy} = (R_1 \| R_2) * (U_2 - U_1) / U_1$. (рисунок 3. 19) .

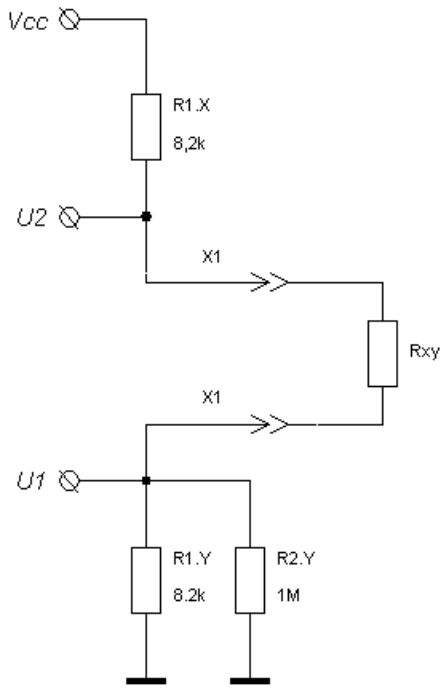


Рисунок 3.19 – Принцип вимірювання опору

Якщо N <3 — ми безсилі. Ми можемо зробити все одне вимірювання опору між ними, в той час, як маємо 2 невідомих — опір, підключений до кожної з них. Система, в якій число рівнянь менше числа невідомих, має безліч рішень. Доведеться показати користувачеві знаки питання на цих лініях — вони начебто справні, але з'ясувати схему кроссіровкі можливим не уявляється.

При N = 3 у нас є лише один можливий варіант. Вимірявши всі доступні опору R_{12} , R_{13} , R_{23} , ми отримаємо систему:

$R_1 + R_2 = R_{12}$

$R_1 + R_3 = R_{13}$

$R_2 + R_3 = R_{23}$

Легко показати, що:

$R_1 = 1/2 * (R_{12} + R_{13} - R_{23})$

$R_2 = R_{12} - R_1$

$R_3 = R_{13} - R_1$.

При більших значеннях N можемо складати систему рівнянь безліччю способів, проводячи заміри різних опорів R_{xy} . На перший погляд, різниці, як вибирати, які опору виміряти, немає. Проте варто враховувати дрібниці. Для прикладу при N = 8 можуть виникнути нюанси. У першій реалізації алгоритму вимірювання здійснюється так:

$R_1 + R_2 = R_{12}$

$R_1 + R_3 = R_{13}$

...

$R_1 + R_8 = R_{18}$

$R_2 + R_3 = R_{23}$

Склавши два перших рівняння і віднявши останнє, отримаємо те ж саме $2R_1 = R_{12} + R_{13} - R_{23}$, а всі інші опору знайдемо з рівнянь 1 - 7, де R_1 вже відомо.

Проблема криється в тому, що при деяких видах кросування значення R_1виявлялося велике (15 кОм і вище), а похибка вимірювання опору з його збільшенням зростає. У підсумку, виходило так, що малі щодо R_1 опору

номіналом 1-2 кОм вимірюються з похибкою в 70-80%! Очевидно, що для забезпечення гарної точності варто скласти систему так, щоб на місці R_1 виявилася інша невідома, мінімальна з усіх. Для цього нам доведеться виконати всі можливі вимірювання. Їх не так багато, в гіршому випадку 28). Фактично, отримано матрицю 8 х 8, симетричну відносно головної діагоналі (ясно, що $R_{xy} = R_{yx}$). Виберемо з усіх результатів мінімальний, нехай це $R_{ij} = R_i + R_j$. У рядку і знайдемо R_{ik} , таке, що $R_{ik} > R_{ij}$, але менше інших елементів рядка. Отримаємо:

$R_i + R_j = R_{ij}$

$R_i + R_k = R_{ik}$

$R_j + R_k = R_{jk}$

Вирішуємо і знаходимо серед R_i , R_j , R_k найменше (припустимо, їм виявилося R_i). залишилися невідомі R_x знаходимо з $R_x = R_{ix} - R_i$.

Етап 4. Визначення точки обриву, якщо така є

Розумні і дорогі залізяки вимірюють відстань до точки обриву за допомогою TDR . Але це складно і дорого.. У нас можливості куди скромніше, та й не так вже й часто потрібне знання положення обриву до сантиметрів — зазвичай розуміння в стилі «прямо біля мене», «на тому кінці», «посередині, де недавно стінку довбали» більш ніж достатньо. Так що обираємо вимір ємності кабелю.

Для цього переводимо всі лінії порту С, крім тієї, яка підключена в тій жилі, де є обрив, в Hi-Z. Подаємо на жилу +5, заряджаючи її. Виміряємо напругу на ній, це буде наше початкове U_0 . Переводимо всі лінії в Hi-Z. Починається розряд кабелю через резистор R2.X опором 1 МОм. Зачекавши 1 мс, вимірюємо напруга на цій лінії U.

$$U = U_0 e^{-\frac{t}{RC}}$$

$$C = -\frac{t}{Rln\frac{U}{U_0}}$$

Не можна забувати, що ланцюга на платі, роз'єм і т.д. теж мають свою ємність, так що пристрій потрібно відкалібрувати на парі шматків кабелю різної довжини. Здебільшого при нульовій довжині 1710 пФ, і ємність кабелю 35 пФ/м. Практика використання показує, що навіть якщо є відхилення, то не сильно, процентів на 10.

3.4 Розробка та відлагодження програмного забезпечення

На основі алгоритму роботи та опису етапів функціонування, можна розробити програмне забезпечення. Програмне забезпечення розроблено на мові С++. Для МК виконується так послідовність дій:

— створення проекту в середовищі;

— набір коду скетчу;

— збір скетчу;

— налаштування:

— усунення синтаксичних помилок;

— усунення логічних помилок;

— заливка скетчу;

— перевірка скетчу на платі або МК.

Детально структура середовища, програмного забезпечення, їх функціональні можливості розглянуто в попередніх розділах і підрозділах.

Програмне забезпечення написано мовою С та наведено в додатках.

3.5 Принцип роботи та рекомендації щодо використання можливості

— визначення типу і підключення кабелю (100 Мбіт, 1Гбіт, відключений, несправний);

— попарно перевірка цілісності провідників, якості контактів, вимір довжини кабелю;

— пожильна перевірка відсутності коротких замикань і дефектів ізоляції (в тому числі на екран кабелю FTP);

— виявлення сторонньої напруги в кабелі і розряду батареї.

Всі вимірювання проводяться за постійним струмом, прилад ні чого не знає про частотні властивості кабелів, і не може їх визначити.

Прилад не призначений для точних вимірювань, його завдання — діагностика, і допомога в локалізації несправності.

Збір

Резистори R1 ... 16 і діоди VD1 ... 16 необхідно брати з однієї партії, для мінімізації розкиду параметрів.

З міркувань надійності, R18 після настройки можна замінити двома постійними резисторами.

Після пайки, плату необхідно ПОВНІСТЮ відмити від флюсу, чутливість приладу до дефектів ізоляції досить висока, найменші залишкові забруднення, можуть привести до неправдивого повідомлення про дефект перевіряється кабелю, або наявності сторонньої напруги.

Прошивка заливається через роз'єм J2 (10-піновий версія AVR-910).

Якщо після прошивки і подачі живлення, на лініях порту "B" не з`явилася логічна "1", це означає що контролер не запуститься, слід перевірити ланцюги живлення і кварцового резонатора, переконатися що правильно залито прошивку і виставлено фьюзи.

Якщо одиниця на лініях порту "B" з'явилася, але індикатор не працює – шукайте проблеми в ньому, і його ланцюгах (порт "D").

Повідомлення про сторонню напругу або несправності в кабелі, говорять про проблеми ланцюгів портів "A" і "C". Там же слід шукати проблему, якщо зі вставленою в роз'єм заглушкою, пристрій не пізнає підключений 1Гбіт кабель, довжиною близькою до нуля.

Калібрування

Унаслідок технологічного розкиду параметрів контролерів, показання різних примірників можуть відрізнятися на кілька відсотків. Так само, впливає різниця відстань скручування пар кабелю. Для підвищення точності, можна зробити калібрування, за допомогою вбудованих програмних засобів.

Для цього знадобляться заглушки "тато" і "мама" з чотирма перемичками, замикаючими пари (контакти 1-2, 3-6, 4-5, 7-8), і відрізок кабелю довжиною 100 або 305 метрів (прилад визначить довжину автоматично) .

Калібрування проходить в 4 стадії, кожна з яких запускається шляхом замикання контактів 9-10 роз'єму J2 в момент подачі живлення.

стадії:

"CAL. OPEN SHORT" — калібрування без кабелю.

"CAL. OPEN LONG" — калібрування по незамкнутому кабелю.

"CAL. LOOP SHORT" — калібрування по заглушці.

"CAL. LOOP LONG" — калібрування по кабелю замкнутому заглушкою.

Між стадіями можна (і потрібно!) Зробити звичайне вимір, для перевірки якості контактів перед калібруванням.

Пам'ятайте, що старі контакти (роз'єми на кабелі, перемички в заглушках), як правило далекі від ідеалу, через поступове окислення і деформацій міді. Так само, не завжди з першого разу, «вдало» стикуються роз'єми.

Експлуатація

У разі отримання повідомлення:

VOLTAGE DETECTED

REMOVE CABLE!

Негайно вимкніть перевіряється кабель від приладу, щоб уникнути пошкодження захисних ланцюгів або контролера!

Вимірювання кабелю займає кілька секунд, в процесі чого, верхній рядок екрана відображає назва приладу і версію прошивки, нижня "прогрес бар".

Після закінчення вимірювань, відображається тип кабелю і довжина попарно.

Для розімкнутих пар довжина вимірюється по електричній ємності, для замкнутих по активному опору.

Порядок пар (зліва направо): Помаранчева, Зелена, Синя, Коричнева.

Вимірювання чутливі до якості контактів, як правило, при поганому контакті результати по замкнутим парам завищуються, по розімкненим занижуються.

У загальному випадку, показання по розімкненим парам точніше.

Про неякісних контактах зазвичай свідчать:

— значна (20 і більше метрів) різниця показань по парам, або при вимірюванні з заглушкою і без заглушки;

— абсолютно нереальні свідчення або повідомлення що кабель пошкоджений.

Пам'ятайте що прилад розрахований на нормальний кабель, з мідною жилою діаметром 0,51мм.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності [1]. Результатом магістерської кваліфікаційної роботи є розробка системи перевірки провідних ліній зв'язку на базі Arduino.

Для проведення технологічного аудиту залучено трьох незалежних експертів: Снігур А.В. (к.т.н., доцент каф. ОТ ВНТУ), Богомолов С.В. (к.т.н., доцент каф. ОТ ВНТУ) та Гарнага В.А. (к.т.н., доцент каф. ОТ ВНТУ)

Оцінювання комерційного потенціалу здійснене за критеріями, що наведені в таблиці 4.1.

Таблиця 4.1 – Критерії оцінювання комерційного потенціалу розробки бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 4.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-тер.	0	1	2	3	4
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 4.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів	Необхідно отримання великої кількості дозвільних документів, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання комерційного потенціалу зведено в таблицю 4.2.

Таблиця 4.2 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	1 – Снігур	2 – Богомолов	3 – Гарнага
	Бали, виставлені експертами:		
1	3	3	3
Ринкові переваги (недоліки):			
2	3	3	3
3	4	4	4
4	4	3	4
5	3	4	3
Ринкові перспективи			
6	3	3	2
7	2	3	3
Практична здійсненність			
8	4	4	3
9	3	2	3
10	3	4	3
11	2	3	3
12	4	4	4
Сума балів	$СБ_1$=38	$СБ_2$=40	$СБ_3$=39
Середньоарифметична сума балів $\overline{СБ}$	39		

За даними таблиці 4.2 можна зробити висновок, щодо рівня комерційного потенціалу розробки. Зважимо на результат й порівняємо його з рівнями комерційного потенціалу розробки, що представлено в таблиці 4.3.

Таблиця 4.3 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Рівень комерційного потенціалу розробки, становить 39 балів, що відповідає рівню «вище середнього».

Аналоги:

1. Тестер кабеля RJ-45 + BNC (HT-C003) (TL-5248). Недоліки: Наявність декількох електронних блоків, лише можливість перевірки на цілісність, відсутність дисплея.

2. Тестер кабеля NCT-3 Gembird. Недоліки: Лише перевірка на цілісність, необхідність застосування заглушки на іншому кінці лінії.

3. Кабельний тестер Microscanner2 (FLN-MS2-100). Недоліки: складність використання, необхідність систем повірки, висока вартість.

Основними недоліками аналогів є їх недостатньо великий рівень точності, великі витрати при експлуатації та висока ціна.

У таблиці 4.4 наведені основні технічні показники аналога і нового програмного продукту

Таблиця 4.4 – Основні технічні показники аналога і нового програмного продукту

Показники	Кабельний тестер Microscanner2 (FLN-MS2-100).	Нова розробка	Відношення параметрів нової розробки до параметрів аналога
Функціональність	85%	95%	1,17
Надійність	70%	90%	1,28
Сумісність	середня	висока	середня-висока
Супровід	середній	високий	середня
Економія ресурсів і часу	середня	висока	середня-висока
Зрозумілість інтерфейсу	80%	100%	1,25

При порівнянні нової розробки з аналогом, можна відзначити, що нова розробка за функціональністю та надійністю перевищує аналог, оскільки має вищий рівень точності ніж у аналога та враховує індивідуальні характеристики клієнта, на відміну від аналогу. Нова розробка спроектована таким чином, що має швидке виявлення екстремальних областей у просторі розв'язків. Саме тому супровід нової системи краще ніж у аналогу. Щодо зручності використання, то нова розробка має доволі простий інтерфейс, на відміну від аналогу, адже для недосвідченого користувача інтерфейс аналогу є ускладненим.

Продукт не є новим на ринку, але застосовані в ньому моделі та методи є новими в даній предметній області. Відмінними рисами нового програмного продукту є: простота й зручність у використанні, висока якість та розвинений функціонал. Даний продукт планується реалізовувати на вітчизняному ринку. На даний час програмний продукт готовий і знаходиться на стадії дослідного зразка. Розробка потребує незначних фінансових ресурсів, а більшість матеріалів для її реалізації відомі і давно використовуються. На ринку праці наявні фахівці відповідної кваліфікації для обслуговування та підтримки нової розробки, регламентні обмеження відсутні і немає необхідності отримання дозвільних документів.

4.2 Прогнозування витрат на виконання наукової роботи та впровадження її результатів

Прогнозування витрат на виконання наукової роботи складається з таких етапів: 1. Розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи. 2. Розрахунок загальних витрат на виконання даної роботи. 3. Прогнозування загальних витрат на виконання та впровадження результатів даної роботи. Проведемо розрахунок витрат, які безпосередньо стосуються виконавців даного розділу роботи.

Обчислимо основну заробітну $З_о$, за формулою:

$$З_о = \frac{M}{T_р} \cdot t[грн], \tag{4.1}$$

де М — місячний посадовий оклад конкретного розробника, грн;

$Т_р$ — число робочих днів в місяці —22;

t — число днів роботи розробника.

Обчислимо заробітну плату розробника з місячним окладом 12600 грн і кількістю робочих днів у місяці — 22. Число днів роботи розробника 66.

$$З_о = \frac{12600}{22} \cdot 66 = 37800{,}00[грн].$$

Результати розрахунків зведемо до таблиці 4.5.

Таблиця 4.5 – Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Розробник	12600,00	572,72	66	37800,00
Керівник проекту	13000,00	590,91	3	1772,72
Всього				39572,72

Обчислимо додаткову заробітну плату розробників $З_д$. Розрахуємо її як 10% від основної заробітної плати:

$$З_д = З_о \cdot 0,10 \text{ [грн]}. \quad (4.2)$$

$$З_д = 39572,72 \cdot 0,10 = 3957,27 \text{ (грн)}.$$

Нарахування на заробітну плату складають 22% від суми основної та додаткової заробітної плати.

$$Н_з = (З_{о\ р} + З_д) \cdot 0,22 \text{ [грн]}. \quad (4.3)$$

$$Н_з = (39572,72+3957,27) \cdot 0,22 = 9576,60 \text{ (грн)}.$$

Обчислимо амортизацію обладнання, що використовувались для розробки. В спрощеному вигляді амортизаційні відрахування розраховується за формулою:

$$А = \frac{Ц \cdot Т}{12 \cdot Т_В} \text{ [грн]}, \quad (4.4)$$

де *Ц* — загальна балансова вартість всього обладнання, комп'ютерів, приміщень тощо, що використовувались для виконання даного етапу роботи, грн;

Т — фактична тривалість використання, міс;

$Т_В$ — термін використання обладнання, приміщень тощо, роки.

Розрахуємо амортизаційні витрати на системний блок та монітор. Балансова вартість блоку становить 10000 грн, а термін його використання — 5 років, а фактична тривалість використання 3 місяці.

$$A = \frac{20000 \cdot 3}{12 \cdot 5} = 1000(грн).$$

Інформацію про витрати на матеріали та комплектуючі, що використані при розробці внесено до таблиці 4.6 та 4.7.

Таблиця 4.6 – Матеріали, що використовуються під час виконання роботи

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено, шт.	Вартість витраченого матеріалу, грн
Папір	110,00	1	110,00
Ручка	12,50	2	25,00
Всього			135,00

Таблиця 4.7 – Комплектуючі, що використовуються під час виконання роботи

Компонент	Ціна за шт, грн	Кількість	Вартість
Arduino	150	1	150
Дисплей	140	1	140
Стабілізатор напруги	5	1	5
Резистори	0,2	20	4
Змінний резистор	1,5	1	1,5
Захистні діоди	1	16	16
Стабілітрон	1	1	1
Конденсатори електролітичні	2	5	10
Конденсатори керамічні	0,5	6	3
Роз’єми	7	2	14
Батарея живлення	30	1	30
Корпус	40	1	40
Витратні матеріали			
Припой	20	1	20
Флюс	10	1	10
Провідники	5	3	15
Гвинти	0,5	16	8
Всього			53,00

Обчислимо витрати на силову електроенергію за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п \, [грн], \qquad (4.5)$$

де В — вартість 1 кВт — години електроенергії, В = 1,69 грн/кВт – година;

П — встановлена потужність обладнання, кВт. П = 0,2 кВт;

Ф — фактична кількість годин роботи обладнання, годин. Ф = 528 годин;

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,4.

$$В_е = 1,69 \cdot 0,2 \cdot 528 \cdot 0,4 = 71,38 \text{ (грн)}.$$

Під час розробки програмного продукту використовувались лише безкоштовні програмні засоби.

Інші витрати ($В_{ін}$) охоплюють: витрати на управління організацією, Інтернет, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, тощо. Інші витрати приймаємо як 100% від суми основної заробітної плати розробників.

$$В_{ін} = З_о \cdot 100\% \text{ [грн]}. \quad (4.6)$$

$$В_{ін} = 39572,72 \cdot 1 = 39572,72 \text{ (грн)}.$$

Обчислимо витрати на виконання даної роботи, що є сумою всіх попередніх витрат.

$$В = З_о + З_р + З_д + Н_з + А + В_е + В_{ін} \text{ [грн]}.$$

В=39572,72+3957,27+9576,60 +1000+135,00+53+71,38 +39572,72=93867,71 (грн).

Загальна вартість всієї наукової роботи ($В_{заг}$) визначається за формулою:

$$В_{заг} = В/\alpha \text{[грн]}, \quad (4.7)$$

де α — частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відносних одиницях.

$$В_{заг} = 93867,71 / 1 = 93867,71 \text{ (грн)}.$$

Виконаємо прогнозування загальних витрат *(ЗВ)* на виконання та впровадження результатів виконаної наукової роботи за формулою:

$$ЗВ = В_{заг} / \beta \text{ [грн]}, \quad (4.8)$$

де β — коефіцієнт, який характеризує етап (стадію) виконання даної роботи. Якщо розробка знаходиться на стадії дослідного зразка, то β ≈ 0,5.

$$ЗВ = 93867,71 /0,5=187735,42 \text{ (грн)}.$$

Витрати на виконання наукової роботи та впровадження її результатів становитиме 187735,42 грн.

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Виконання даної наукової роботи та впровадження її результатів складає приблизно 1 рік. Позитивні результати від впровадження розробки очікуються вже в перші місяці після впровадження.

Проведемо детальніше прогнозування результатів та кількісне їх оцінювання по роках.

Обчислимо збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \cdot \Delta N)_i \text{ [грн]}, \quad (4.9)$$

де $\Delta П_я$ — покращення основного якісного показника від впровадження результатів розробки у даному році;

N — основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN — покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ — основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n — кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

Припустимо, що внаслідок впровадження результатів наукової розробки чистий прибуток підприємства збільшиться на 150 грн., а кількість одиниць реалізованої продукції збільшиться: протягом першого року — на 1000 од., протягом другого року — ще на 2000 од., протягом третього року — ще на 1000 од. Орієнтовно: реалізація продукції до впровадження результатів наукової розробки складала 1 шт., а прибуток, що його отримувало підприємство на одиницю продукції до впровадження результатів наукової розробки — 100 грн.

Потрібно спрогнозувати збільшення чистого прибутку підприємства від впровадження результатів наукової розробки у кожному році відносно базового.

Збільшення чистого прибутку підприємства $\Delta П_1$ протягом першого року складе:

$$\Delta П_1=150\cdot 1+(100+150)\cdot 1000=250150 \text{ (грн)}.$$

Обчислимо збільшення чистого прибутку підприємства $\Delta П_2$ протягом другого року:

$$\Delta П_2=150\cdot 1+(100+150)\cdot (1000+2000)=750150 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_3$ протягом третього року становитиме:

$$\Delta П_3=150\cdot 1+(100+150)\cdot (1000+2000+1000)=1000150{,}00 \text{ (грн)}.$$

Отже, розрахунки показують, що комерційний ефект від впровадження розробки виражається у значному збільшенні чистого прибутку підприємства.

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Щоб оцінити доцільність фінансування проекту, необхідно провести розрахунки ефективності вкладених інвестицій.

На першому етапі розрахуємо теперішню вартість інвестицій PV = ЗВ=188513,07 (грн).

На другому етапі розраховуємо очікуване збільшення прибутку $\Delta П_i$, що його отримає підприємство від впровадження результатів наукової розробки, для кожного із років, починаючи з першого року впровадження. Таке збільшення прибутку було розраховане раніше. Результати вкладених у наукову розробку інвестицій виявляться за перший рік після провадження у тому, що у 1 році підприємство отримає збільшення чистого прибутку на 250150 грн, у 2 році — збільшення чистого прибутку на 750150 грн, у 3 році — збільшення чистого прибутку на 1000150,00 грн.

На третьому етапі для спрощення подальших розрахунків будуємо вісь часу, на яку наносимо всі платежі (інвестиції та прибутки), що мають місце під час виконання науково-дослідної роботи та впровадження її результатів.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, наведений на рис. 4.1.

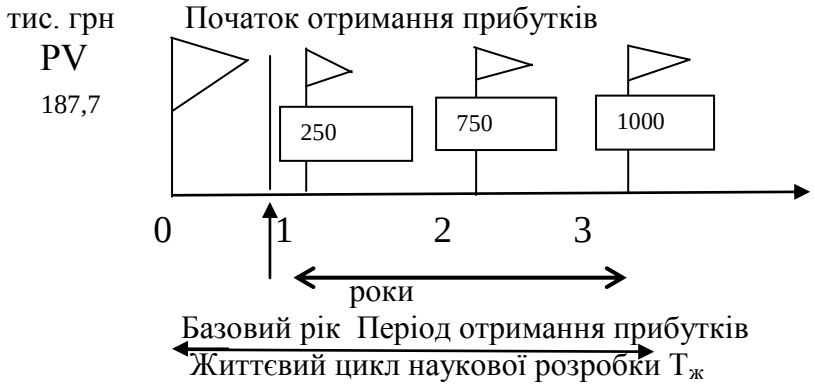


Рисунок 4.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

На четвертому етапі розраховуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ за формулою:

$$E_{абс}=(ПП-PV)[грн], \quad (4.10)$$

де ПП — приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн;

PV — теперішня вартість інвестицій PV = ЗВ=188513,07 (грн).

$$ПП = \sum_1^T \frac{\Delta П_i}{(1+\tau)^t} [грн], \quad (4.11)$$

де $\Delta П_i$ — збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т — період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t — період часу (в роках) від моменту отримання чистого прибутку до точки „0”.

Проведемо розрахунки.

$$ПП=250100 /(1+0,1)^1 + 750100/(1+0,1)^2 + 1000100,00 /(1+0,1)^3 = 1608712,82 (грн).$$

$$E_{абс} =1608712,82 - 187735,42 = 1420977,4 (грн).$$

Так як $E_{абс} > 0$, то результат від проведення наукових досліджень та їх впровадження принесе прибуток, але це ще не свідчить про те, що інвестор буде зацікавлений у фінансуванні даного проекту.

На п’ятому етапі розрахуємо відносну (щорічну) ефективність вкладених у наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$\mathrm{E_в} = \sqrt[Тж]{1 + \frac{\mathrm{E_{абс}}}{PV}} - 1 \qquad (4.12)$$

де $E_{абс}$ — абсолютна ефективність вкладених інвестицій, грн;

PV теперішня вартість інвестицій PV = ЗВ, грн;

$T_ж$ — життєвий цикл наукової розробки, роки.

$$\mathrm{E_в} = \sqrt[3]{1 + \frac{1420977,4}{187735,42}} - 1 = 1,05, \text{ або } 105\%.$$

Розраховану величину $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування визначається за формулою:

$$\tau = d + f, \qquad (4.13)$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; d =0,2;

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = 0,05.

$$\tau = 0,2 + 0,05 = 0,25$$

Оскільки $E_в$ = 105% > τ мін = 0,25 = 25%, то потенційний інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

На останньому етапі розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій.

Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ можна розрахувати за формулою:

$$T_{ок}=1/ E_в \text{ [роки]}, \quad (4.14)$$

$$T_{ок}=1/1,05 = 0,95 \text{ (року)}.$$

Термін окупності становить 0,95 року ($T_{ок}$ <3…5 років), що свідчить, що фінансування даної наукової розробки є доцільним.

ВИСНОВКИ

Магістерська кваліфікаційна робота присвячена ситемі перевірки провідних ліній зв’язку.

Для визначення напрямків дослідження оглянуто існуючі рішення, враховано їх переваги та недоліки.

За основу обрано мікропроцесорну платформу Arduino Nano, яку побудовано на базі мікроконтролера AVR ATMega168.

Вивчено детально роботу мікроконтролера ATMega168, описано всі його складові, що використовуються в роботі системи.

Обрано електронні елементи для повноцінної роботи системи.

Створено алгоритм роботи та програмне забезпечення на його основі могою програмування C++.

Визначено потребу споживачів, що дозволило створити систему з відповідними функціональними можливостями та технічними характеристиками.

Створено інструкцію щодо використання системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Arduino Mega 2560 [Електронний ресурс] // Arduino [Електронний ресурс] // Сайт користувачів«Arduino». — Режим доступу: http://arduino.ru/Hardware/ArduinoBoardMega2560
2. Arduino Nano [Електронний ресурс] // Arduino [Електронний ресурс] // Сайт користувачів «Arduino». — Режим доступу: http://arduino.ru/Hardware/ArduinoBoardNano
3. Arduino [Електронний ресурс] // Сайт програмного середовища і фірми «Arduino». — Режим доступу: https://www.arduino.cc
4. Програмування Arduino [Електронний ресурс] // Сайт користувачів«Arduino». — Режим доступу: http://arduino.ru/Reference
5. Переривання на мікроконтролері Arduino [Електронний ресурс] // Сайт по машинному навчання і програмування. — Режим доступу: http://robotosha.ru/arduino/arduino-interrupts.html
6. Аналогові вимірювання з Arduino [Електронний ресурс] // Сайт користувачів«Arduino». — Режим доступу: http://robotosha.ru/arduino/analog-measurements-arduino.html
7. Функції Arduino [Електронний ресурс] // Сайт http://codius.ru.— Режим доступу: http://codius.ru/articles. html
8. Проекты с использованием контролера Arduino. — СПб.: БХВ–Петербург, 2014. — 400 с.: — (Электроника)
9. Книга по программированию микроконтроллеров AVR [Електронний ресурс] http://cxem.net/mc/book.php
10. 10. Основи інформаційних систем: Навч. посібник. — Вид. 2-ге, перероб. и доп. / В. Ф. Ситник, Т. А. Писаревська, Н. В. Єрьоміна, О. С. Краєва; За ред. В. Ф. Ситника. — К .: КНЕУ, 2001. - 420 с.
11. 11. Новіков Ю.В., Карпенко Д.Г. Апаратура локальних мереж: функції, вибір, разработка.— М .: ЕКОМ, 1998.- 288 с.
12. 12. Кулаков Ю.А., Омельянскій С.В. Комп'ютерні мережі. Вибір, установка, використання і адміністрірованіе.— К .: Юніор, 1999.- 544с.

13. 13. Гук М. Апаратні засоби локальних мереж. Енциклопедія СПб: Пітер, 2000.- 576 с.
14. 14. Гусєва А.І. Технологія міжмережевих взаємодій. NetWare — Unix - Windows - Internet.- М .: Діалог-МІФІ, 1997.- 272 с.
15. 15. Книга по програмуванню мікроконтролерів AVR [Електронний ресурс] http://cxem.net/mc/book.php
16. 16. Букрєєв І.М. Мікроелектронні схеми цифрових пристроїв / І. Н. Букрєєв, В. І. Горячев, Б. М. Мансуров. — Москва: Техносфера, 2009. - 712 с.- ISBN 978-5-94836-197-0.
17. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

ДОДАТКИ

ДОДАТОК А

Міністерство освіти та науки України
Вінницький національний технічний університет
Інститут інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., професор Мартинюк Т. Б._____
(наук. ст., вч. зв., ініц. та прізв.) (підпис)
"____" _____20__р.

ТЕХНІЧНЕ ЗАВДАННЯ
на виконання магістерської кваліфікаційної роботи
Система перевірки провідних ліній зв'язку на базі Arduino
08–23.МКР.005.00.000.ТЗ

Науковий керівник: к.т.н., доц. Снігур А.В.
_____
(підпис)
студент групи КІ-18м
_____ Піщенко Д.В.
(підпис)

Вінниця 2020р.

1. Підстава для виконання магістерської кваліфікаційної роботи (МКР)

а) актуальність досліджень;

б) наказ про затвердження теми дипломної роботи.

2. Мета і призначення МКР

а) Мета полягає у реалізації системи перевірки провідних ліній зв'язку, використовуючи сучасні досягнення у напрямку мікропроцесорних систем, які дозволяють будувати повнофункціональні системи з можливістю адаптації функціональних можливостей під необхідні задачі.

б) призначення розробки – виконання магістерської кваліфікаційної роботи, виконання організаційно – технологічних та наукових досліджень.

3. Вихідні дані для виконання МКР

— технічні характеристики систем та пристроїв перевірки провідних ліній зв'язку;

— технічний опис платформ Arduino;

— технічний опис мікроконтролерів Atmel;

— середовище розробки ПЗ Arduino IDE.

4. Вимоги до виконання МКР

— огляд і аналіз методів перевірки провідних ліній зв'язку;

— класифікація методів перевірки провідних ліній зв'язку;

— дослідження способів та методів провідних ліній зв'язку;

— розробка функціональної та принципової схем системи;

— розробка алгоритму роботи та програмного забезпечення.

5. Етапи МКР та очікувані результати

№ етапу	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Пошук та огляд інформаційних джерел	11.02.19р.	02.03.19р.	Розділ 1
2	Дослідження способів побудови систем аутентифікації	03.03.19р.	30.03.19р.	Розділ 2
3	Побудова мікропроцесорного системи управління	31.03.19р.	27.04.19р.	Розділ 3
4	Економічна частина	28.04.19р.	25.05.19р.	Розділ 4

6. Матеріали, що подаються до захисту МКР

Пояснювальна записка МКР, графічні і ілюстративні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив опонента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, нормоконтроль про відповідність оформлення МКР діючим вимогам.

7. Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8. Вимоги до оформлення МКР

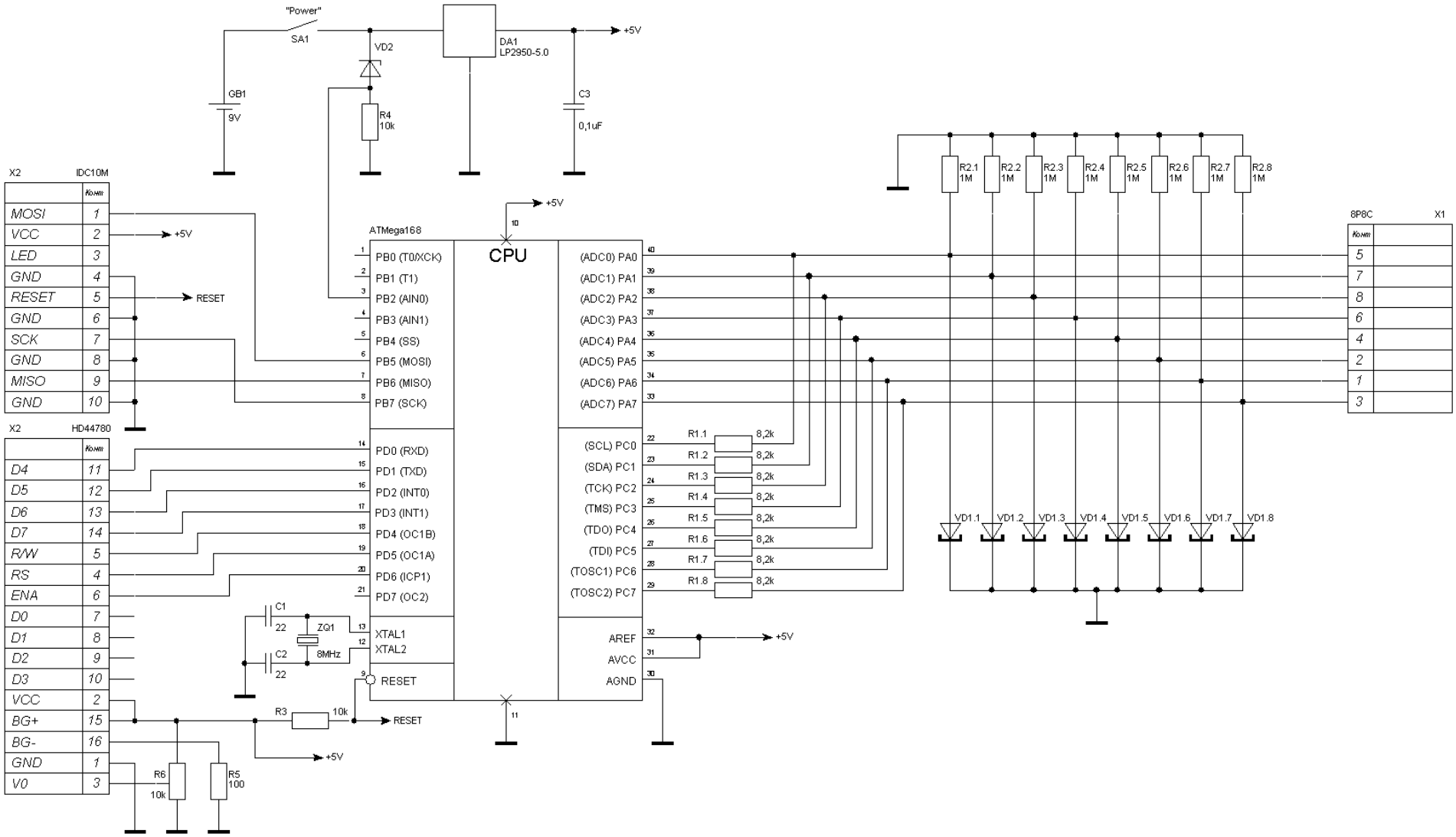
Вимоги викладені в МЕТОДИЧНИХ ВКАЗІВКАХ до дипломного проектування, ДСТУ_ 3008-95, ДСТУ 3974-2000 «Правила виконання дослідно-конструкторських робіт. Загальні положення» та діючого ГОСТ 2.114-95 ЕСКД.

9. Вимоги щодо технічного захисту інформації в МКР з обмеженим доступом

Відсутні.

ДОДАТОК Б

Електрична принципова схема



					08-23.МКР.005.00.001			
					Система перевірки провідних ліній зв'язку на базі Arduino	Літ.	Маса	Масштаб
Змн.	Арк.	№ докум.	Підпис	Дата				
Розробив		Піщенко Д.В.				Арк. 1	Аркушів 1	
Керівник		Снігур А.В.						
Рецензент		Дудатьєв А.В.						
					Схема електрична принципова	ВНТУ,гр. КІ – 18м		
Н. контроль		Швець С.І.						
Затверджую		Мартинюк Т.Б.						

ДОДАТОК В

Лістинг програмного забезпечення

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>
#include <avr/pgmspace.h>
#include <avr/sleep.h>
#include <avr/interrupt.h>
#include <avr/eeprom.h>

#include "hd44780.h"
#include "lan_tester.h"


#define SET_CHARGE { DDRC = 0xff; PORTC = 0xff; }
#define SET_DISCHARGE { DDRC = 0xff; PORTC = 0x00; }
#define SET_OPEN { DDRC = 0x00; PORTC = 0x00; }


const struct CAL_DATA CalDataDefault PROGMEM = {
	{ AC_MIN_CLOCKS, AC_MIN_CLOCKS, AC_MIN_CLOCKS,
AC_MIN_CLOCKS, AC_MIN_CLOCKS, AC_MIN_CLOCKS, AC_MIN_CLOCKS,
AC_MIN_CLOCKS },
	{ AC_CAPACITANCE_CONST, AC_CAPACITANCE_CONST,
AC_CAPACITANCE_CONST, AC_CAPACITANCE_CONST,
AC_CAPACITANCE_CONST, AC_CAPACITANCE_CONST,
AC_CAPACITANCE_CONST, AC_CAPACITANCE_CONST },
	{ ADC_MIN_DELTA, ADC_MIN_DELTA, ADC_MIN_DELTA,
ADC_MIN_DELTA, ADC_MIN_DELTA, ADC_MIN_DELTA, ADC_MIN_DELTA,
ADC_MIN_DELTA },
	{ ADC_RESISTANCE_CONST, ADC_RESISTANCE_CONST,
ADC_RESISTANCE_CONST, ADC_RESISTANCE_CONST,
ADC_RESISTANCE_CONST, ADC_RESISTANCE_CONST,
ADC_RESISTANCE_CONST, ADC_RESISTANCE_CONST },
	0x00,
	0x0000
};


const struct CAL_RANGES CalRanges[] PROGMEM = {
	{ 50, 0 },
	{ 200, 100 },
```

```
        { 400, 305 }
};


unsigned short usCapt;
unsigned short usVoltage[8];
unsigned short usResistance[8][8][2];
struct CAL_DATA CalData;


ISR(TIMER1_COMPA_vect) {
}


ISR(TIMER1_CAPT_vect) {

        usCapt = ICR1;
}


ISR(TIMER1_OVF_vect) {

        usCapt = USHRT_MAX;
}


ISR(ADC_vect) {
}


void halt() {

        set_sleep_mode(SLEEP_MODE_PWR_DOWN);
        sleep_cpu();
}


void pgm_read_block(const void *dst, const void *src, const size_t size) {

        size_t ct;

        for(ct = 0; ct < size; ct++) {
                *((unsigned char *) dst + ct) = pgm_read_byte(((unsigned char *) src
+ ct));
        }
}
```

```
unsigned short crc16(const void *pBuff, unsigned char n) {

	unsigned char i, carry;
	unsigned short crc16 = 0xffff;

	while(n) {
		crc16 ^= *(unsigned char *) pBuff;
		for(i = 0; i < 8; i++) {
			carry = crc16 & 1;
			crc16 >>= 1;
			if(carry) {
				crc16 ^= 0xA001;
			}
		}
		n--;
		pBuff++;
	}
	return crc16;
}


void sleep_ms(const unsigned short ms) {

	unsigned short ct;

	set_sleep_mode(SLEEP_MODE_IDLE);
	OCR1A = F_CPU / 1000;
	TCNT1 = 0;
	TIFR = (1 << OCF1A);   // Reset compare flag
	TIMSK = (1 << OCIE1A);		// Match Interrupt Enable
	TCCR1B = (1 << WGM12) | (1 << CS10);  // Start clock, CTC mode,
clkI/O/1 (No prescaling)
	for(ct = 0; ct < ms; ct++) {
		sleep_cpu();
	}
	TIMSK = 0; // Disable Timer/Counters interrupts
	TCCR1B = 0;		// Disable Timer/Counter1 clock
}


unsigned short f2u(float f) {

	if(f < 0.0f) {
		return(0);
	} else if(f > 999.0f) {
		return(999);
	}
```

```
	return(round(f));
}


void checkPower() {

	bool bLow;
	static char strLowBat1[] PROGMEM = "  BATTERY LOW!  ";
	static char strLowBat2[] PROGMEM = " Please replace.";

	ACSR = (1 << ACBG);
	SFIOR &= !(1 << ACME);
	sleep_ms(1);
	bLow = ACSR & (1 << ACO);
	ACSR = (1 << ACD);
	if(bLow) {
		lcdClear();
		lcdPutsFromFlash(strLowBat1);
		lcdGotoXY(1, 0);
		lcdPutsFromFlash(strLowBat2);
		sleep_ms(1000);
	}
}


void getVoltage(const unsigned char mask, const unsigned short cycles) {

	unsigned int ct;
	unsigned short bct;
	unsigned long ulAvg;

	set_sleep_mode(SLEEP_MODE_ADC);
	ADCSRA = (1 << ADIF);// Reset ADC flag
	ADCSRA = (1 << ADEN) | (1 << ADSC) | (1 << ADATE) | (1 << ADIE) | (1
<< ADPS2) | (1 << ADPS1) | (1 << ADPS0);
	for(ct = 0; ct < 8; ct++) {
		if((mask >> ct) & 1) {
			ulAvg = 0;
			ADMUX = ct;	// Channel select
			sleep_cpu();	// Skip first conversion
			for(bct = 0; bct < cycles; bct ++) {
				sleep_cpu();
				ulAvg += ADC;
			}
			usVoltage[ct] = (ulAvg << (SHRT_BITS - ADC_BITS)) / cycles;
		}
	}
	ADCSRA = 0;	// Disable ADC
```

```
}

void charge(const char state, const unsigned char mask, const unsigned short
delay) {

	bool bEnd = false;
	unsigned int ct;
	unsigned short usOldVoltage[8];

	DDRC = mask;
	PORTC = state;

	for(ct = 0; ct < 8; ct++) {
		if((mask >> ct) & 1) {
			if((state >> ct) & 1) {		// 1
				usOldVoltage[ct] = 0;
			} else {			// 0
				usOldVoltage[ct] = USHRT_MAX;
			}
		}
	}

	while(!bEnd) {
		bEnd = true;
		getVoltage(mask, ADC_LINETEST_CYCLES);
		for(ct = 0; ct < 8; ct++) {
			if((mask >> ct) & 1) {
				if((state >> ct) & 1) {		// 1
					if(usVoltage[ct] > usOldVoltage[ct]) {
						usOldVoltage[ct] = usVoltage[ct];
						bEnd = false;
					}
				} else {			// 0
					if(usVoltage[ct] < usOldVoltage[ct]) {
						usOldVoltage[ct] = usVoltage[ct];
						bEnd = false;
					}
				}
			}
		}
		if(!bEnd) {
			sleep_ms(delay);
		}
	}
}
```

```
void checkVoltage() {

	unsigned char ct, ct2;
	static char strVoltage1[] PROGMEM = "VOLTAGE DETECTED";
	static char strVoltage2[] PROGMEM = " REMOVE CABLE!  ";

	charge(0, 0xff, CHARGE_LINETEST_DELAY);

	for(ct = 0; ct < CHECK_VOLTAGE_TRYES; ct++) {
		sleep_ms(CHECK_VOLTAGE_DELAY);
		getVoltage(0xff, ADC_LINETEST_CYCLES);
		for(ct2 = 0; ct2 < 8; ct2++) {
			if(usVoltage[ct2] > CHECK_VOLTAGE_SENSE) {
				lcdClear();
				lcdPutsFromFlash(strVoltage1);
				lcdGotoXY(1, 0);
				lcdPutsFromFlash(strVoltage2);
				halt();
			}
		}
	}
}


void getResistance() {

	unsigned char ct, ct2;
	unsigned char mask;

	for(ct = 0; ct < 8; ct++) {
		for(ct2 = 0; ct2 < 8; ct2++) {
			if(ct2 != ct) {
				mask = (1 << ct) | (1 << ct2);
				charge(1 << ct, mask, CHARGE_LINETEST_DELAY);
				getVoltage(mask, ADC_LINETEST_CYCLES);
				usResistance[ct][ct2][0] = usVoltage[ct2];
				usResistance[ct][ct2][1] = usVoltage[ct];
			}
		}
	}
	SET_OPEN;
}


bool checkOpen2(const int pin1, const int pin2) {

	if(
		usResistance[pin1][pin2][0] > BROKEN_SENSE ||
```

```
		usResistance[pin1][pin2][1] < (USHRT_MAX - BROKEN_SENSE) ||
		usResistance[pin2][pin1][0] > BROKEN_SENSE ||
		usResistance[pin2][pin1][1] < (USHRT_MAX - BROKEN_SENSE)
	) {
		return(false);
	}

	return(true);
}


bool checkOpen(const int pin) {

	unsigned char ct;

	for(ct = 0; ct < 8; ct++) {
		if(ct != pin) {
			if(!checkOpen2(pin, ct)) {
				return(false);
			}
		}
	}

	return(true);
}


bool checkOpenAll() {

	unsigned char ct;

	for(ct = 0; ct < 7; ct++) {
		if(!checkOpen(ct)) {
			return(false);
		}
	}

	return(true);
}


bool checkLoop(const int pin1, const int pin2) {

	if(
		usResistance[pin1][pin2][0] < LOOP_SENSE ||
		usResistance[pin1][pin2][1] > (USHRT_MAX - LOOP_SENSE) ||
		usResistance[pin2][pin1][0] < LOOP_SENSE ||
		usResistance[pin2][pin1][1] > (USHRT_MAX - LOOP_SENSE)
```

```
        ) {
                return(false);
        }

        return(true);
}


bool check_conn_1000() {

        if(
                !checkLoop(JPIN1, JPIN2) ||
                !checkLoop(JPIN3, JPIN6) ||
                !checkLoop(JPIN4, JPIN5) ||
                !checkLoop(JPIN7, JPIN8)
        ) {
                return(false);
        }

        return(true);
}


bool check_conn_100() {

        if(
                !checkLoop(JPIN1, JPIN2) ||
                !checkLoop(JPIN3, JPIN6) ||
                !checkOpen(JPIN4) ||
                !checkOpen(JPIN5) ||
                !checkOpen(JPIN7) ||
                !checkOpen(JPIN8)
        ) {
                return(false);
        }

        return(true);
}


short getShortRaw(const int pin1, const int pin2) {

        unsigned char mask;

        mask = (1 << pin1) | (1 << pin2);
        charge(1 << pin1, mask, CHARGE_PRECISION_DELAY);
        getVoltage(mask, ADC_PRECISION_CYCLES);
        SET_OPEN;
```

```
	return(usVoltage[pin1] - usVoltage[pin2]);
}


unsigned short getShortLength(const int pin1, const int pin2) {

	float f1, f2;

	f1 = getShortRaw(pin1, pin2);
	f1 -= CalData.AdcMinDelta[pin1];
	f1 *= R1_16 * 2 + RDS_ON * 2;
	f1 /= USHRT_MAX;
	f1 /= CalData.AdcMultipler[pin1];

	f2 = getShortRaw(pin2, pin1);
	f2 -= CalData.AdcMinDelta[pin2];
	f2 *= R1_16 * 2 + RDS_ON * 2;
	f2 /= USHRT_MAX;
	f2 /= CalData.AdcMultipler[pin2];

	return(f2u((f1 + f2) / 2));
}


unsigned short getOpenRaw(const int pin) {

	charge(1 << pin, 0xff, CHARGE_PRECISION_DELAY);

	ACSR = (1 << ACBG) | (1 << ACIC) | (1 << ACO);
	SFIOR = (1 << ACME);
	ADMUX = pin;

	set_sleep_mode(SLEEP_MODE_IDLE);
	TCNT1 = 0;
	TIFR = (1 << TOV1) | (1 << ICF1);    // Reset overflow and capture flags
	TIMSK = (1 << TICIE1) | (1 << TOIE1);      // Enable overflow and capture
interrupts
	TCCR1B = (1 << CS10) | (1 << ICNC1) | (1 << ICES1); // Start clock,
Normal mode, clkI/O/1 (No prescaling)
	SET_DISCHARGE;
	sleep_cpu();

	TIMSK = 0;          // Disable Timer/Counters interrupts
	TCCR1B = 0;                 // Disable Timer/Counter1 clock
	ADCSRA = 0;                 // Disable ADC
	ACSR = (1 << ACD);     // Disable Analog comparator
	SET_OPEN;
```

```
	return(usCapt);
}


unsigned short getOpenLength(const int pin1, const int pin2) {

	float f1, f2;

	f1 = getOpenRaw(pin1);
	f1 -= CalData.AcMinClocks[pin1];
	f1 *= CalData.AcMultipler[pin1];
	f2 = getOpenRaw(pin2);
	f2 -= CalData.AcMinClocks[pin2];
	f2 *= CalData.AcMultipler[pin2];

	return(f2u((f1 + f2) / 2));
}


void writeConfig() {

	CalData.crc = crc16(&CalData, sizeof(CalData) - sizeof(unsigned short));
	eeprom_write_block(&CalData, 0, sizeof(CalData));
}


void progress() {

	static char strProgress[] PROGMEM = "##";

	lcdPutsFromFlash(strProgress);
}


void calibrate_error() {

	static char strCalBad[] PROGMEM = "ERROR, BAD CABLE";

	lcdPutsFromFlash(strCalBad);
	halt();
}


void calStage1(const unsigned short usMeters) {

	unsigned char ct, ct2;
	unsigned long ul;
```

```
	for(ct = 0; ct < 8; ct++) {
		ul = 0;
		for(ct2 = 0; ct2 < CALIBRATE_CAP_CYCLES; ct2++) {
			ul += getOpenRaw(ct);
		}
		CalData.AcMinClocks[ct] = ul / CALIBRATE_CAP_CYCLES;  //
Round to min
		progress();
	}
}


void calStage2(const unsigned short usMeters) {

	unsigned char ct, ct2;
	float fRes;

	for(ct = 0; ct < 8; ct++) {
		fRes = 0.0f;
		for(ct2 = 0; ct2 < CALIBRATE_CAP_CYCLES; ct2++) {
			fRes += getOpenRaw(ct);
			fRes -= CalData.AcMinClocks[ct];
		}
		fRes /= CALIBRATE_CAP_CYCLES;
		fRes /= usMeters;
		fRes = 1 / fRes;
		CalData.AcMultipler[ct] = fRes;
		progress();
	}
}


void calStage3b(const int pin1, const int pin2) {

	unsigned char ct;
	long l;

	l = 0;
	for(ct = 0; ct < CALIBRATE_ADC_CYCLES; ct++) {
		l += getShortRaw(pin1, pin2);
	}
	CalData.AdcMinDelta[pin1] = l / CALIBRATE_ADC_CYCLES;		//
Round to min
}


void calStage3(const unsigned short usMeters) {
```

```
        calStage3b(JPIN1, JPIN2);
        progress();
        calStage3b(JPIN2, JPIN1);
        progress();
        calStage3b(JPIN3, JPIN6);
        progress();
        calStage3b(JPIN6, JPIN3);
        progress();
        calStage3b(JPIN4, JPIN5);
        progress();
        calStage3b(JPIN5, JPIN4);
        progress();
        calStage3b(JPIN7, JPIN8);
        progress();
        calStage3b(JPIN8, JPIN7);
        progress();
}

void calStage4b(const int pin1, const int pin2, const unsigned short usMeters) {

        unsigned char ct;
        long l;
        float f;

        l = 0;
        for(ct = 0; ct < CALIBRATE_ADC_CYCLES; ct++) {
                l += getShortRaw(pin1, pin2);
        }
        f = l;
        f /= CALIBRATE_ADC_CYCLES;
        f -= CalData.AdcMinDelta[pin1];
        f *= R1_16 * 2 + RDS_ON * 2;
        f /= USHRT_MAX;
        f /= usMeters;
        CalData.AdcMultipler[pin1] = f;
}

void calStage4(const unsigned short usMeters) {

        calStage4b(JPIN1, JPIN2, usMeters);
        progress();
        calStage4b(JPIN2, JPIN1, usMeters);
        progress();
        calStage4b(JPIN3, JPIN6, usMeters);
        progress();
```

```
        calStage4b(JPIN6, JPIN3, usMeters);
        progress();
        calStage4b(JPIN4, JPIN5, usMeters);
        progress();
        calStage4b(JPIN5, JPIN4, usMeters);
        progress();
        calStage4b(JPIN7, JPIN8, usMeters);
        progress();
        calStage4b(JPIN8, JPIN7, usMeters);
        progress();
}


const struct CAL_STAGES CalStages[] PROGMEM = {
        { calStage1, T_OPEN, false, "CAL. OPEN SHORT " },
        { calStage2, T_OPEN, true, " CAL. OPEN LONG " },
        { calStage3, T_CONN_1000, false, "CAL. LOOP SHORT " },
        { calStage4, T_CONN_1000, true, " CAL. LOOP LONG " }
};



void calibrate(const enum ETYPE eType, const unsigned short usResult0, const
unsigned short usResult1, const unsigned short usResult2, const unsigned short
usResult3) {

        unsigned char ct;
        unsigned short usMeters;
        void (* pFunc)(unsigned short);
        char strResult[LCD_LINE_SIZE + 1];

        if(CalData.cStage >= (sizeof(CalStages) / sizeof(struct CAL_STAGES))) {
                CalData.cStage = 0;
        }

        lcdPutsFromFlash(CalStages[CalData.cStage].strMessage);
        lcdGotoXY(1, 0);

        if(eType != pgm_read_byte(&CalStages[CalData.cStage].eType)) {
                calibrate_error();
        }

        usMeters = 999;
        for(ct = 0; ct < (sizeof(CalRanges) / sizeof(struct CAL_RANGES)); ct++) {
                if(
                        usResult0 < pgm_read_word(&CalRanges[ct].max) &&
                        usResult1 < pgm_read_word(&CalRanges[ct].max) &&
                        usResult2 < pgm_read_word(&CalRanges[ct].max) &&
```

```
                usResult3 < pgm_read_word(&CalRanges[ct].max)
            ) {
                usMeters = pgm_read_word(&CalRanges[ct].len);
                break;
            }
        }

        if(usMeters == 999) {
            calibrate_error();
        }

        if(pgm_read_byte(&CalStages[CalData.cStage].bLong)) {
            if(usMeters == 0) {
                calibrate_error();
            }
        } else {
            if(usMeters > 0) {
                calibrate_error();
            }
        }

        pFunc = (void *) pgm_read_word(&CalStages[CalData.cStage].func);
        pFunc(usMeters);

        CalData.cStage++;
        writeConfig();
        lcdGotoXY(1, 0);
        sprintf(strResult, "    OK, %3um    ", usMeters);
        lcdPutsFromRAM(strResult);
        halt();
}


void readConfig() {

        static char strBad1[] PROGMEM = " NOT CALIBRATED ";
        static char strBad2[] PROGMEM = "DEFAULTS LOADED ";

        eeprom_read_block(&CalData, 0, sizeof(CalData));
        if(crc16(&CalData, sizeof(CalData) - sizeof(unsigned short)) !=
CalData.crc) {
            pgm_read_block(&CalData, &CalDataDefault, sizeof(struct
CAL_DATA));
            lcdPutsFromFlash(strBad1);
            lcdGotoXY(1, 0);
            lcdPutsFromFlash(strBad2);
            writeConfig();
            halt();
```

```
        }
}

int main() {

        bool bCalibrate;
        enum ETYPE eType;
        char strResult[LCD_LINE_SIZE + 1];
        unsigned short usResult[4];
        static char strMotd[] PROGMEM = "LAN-tester V1.00";
        const char *pstrType[] = {
                "     OPEN     ",
                " CONNECTED 1000 ",
                " CONNECTED 100  ",
                "    BROKEN    "
        };

        DDRA = 0x00;            // Input
        DDRB = 0x00;            // Input
        DDRC = 0xff;            // Output
        DDRD = 0xff;            // Output
        PORTA = 0x00;
        PORTB = 0xf7;           // PullUp On, expect AIN1
        PORTC = 0x00;
        PORTD = 0x00;
        ACSR = (1 << ACD);      // Disable Analog comparator
        sei();                  // Enable interrupts
        sleep_enable();
        sleep_ms(100);
        bCalibrate = !((PINB >> PINB6) & 1);

        lcdInit();
        lcdClear();
        readConfig();
        lcdPutsFromFlash(strMotd);
        lcdGotoXY(1, 0);

        while(true) {
                checkPower();
                progress();
                checkVoltage();
                progress();
                getResistance();
                progress();

                if(checkOpenAll()) {
                        eType = T_OPEN;
```

```
} else if(check_conn_1000()) {
        eType = T_CONN_1000;
} else if(check_conn_100()) {
        eType = T_CONN_100;
} else {
        eType = T_BROKEN;
}
progress();

switch(eType) {
        case T_OPEN:
                usResult[0] = getOpenLength(JPIN1, JPIN2);
                progress();
                usResult[1] = getOpenLength(JPIN3, JPIN6);
                progress();
                usResult[2] = getOpenLength(JPIN4, JPIN5);
                progress();
                usResult[3] = getOpenLength(JPIN7, JPIN8);
                break;
        case T_CONN_100:
                usResult[0] = getShortLength(JPIN1, JPIN2);
                progress();
                usResult[1] = getShortLength(JPIN3, JPIN6);
                progress();
                usResult[2] = getOpenLength(JPIN4, JPIN5);
                progress();
                usResult[3] = getOpenLength(JPIN7, JPIN8);
                break;
        case T_CONN_1000:
                usResult[0] = getShortLength(JPIN1, JPIN2);
                progress();
                usResult[1] = getShortLength(JPIN3, JPIN6);
                progress();
                usResult[2] = getShortLength(JPIN4, JPIN5);
                progress();
                usResult[3] = getShortLength(JPIN7, JPIN8);
                break;
        default:
                if(checkLoop(JPIN1, JPIN2)) {
                        usResult[0] = getShortLength(JPIN1, JPIN2);
                } else {
                        usResult[0] = getOpenLength(JPIN1, JPIN2);
                }
                progress();
                if(checkLoop(JPIN3, JPIN6)) {
                        usResult[1] = getShortLength(JPIN3, JPIN6);
                } else {
                        usResult[1] = getOpenLength(JPIN3, JPIN6);
```

```
                        }
                        progress();
                        if(checkLoop(JPIN4, JPIN5)) {
                                usResult[2] = getShortLength(JPIN4, JPIN5);
                        } else {
                                usResult[2] = getOpenLength(JPIN4, JPIN5);
                        }
                        progress();
                        if(checkLoop(JPIN7, JPIN8)) {
                                usResult[3] = getShortLength(JPIN7, JPIN8);
                        } else {
                                usResult[3] = getOpenLength(JPIN7, JPIN8);
                        }
                        break;
                }
                progress();

                lcdClear();
                if(bCalibrate) {
                        calibrate(eType, usResult[0], usResult[1], usResult[2],
usResult[3]);
                }
                lcdPutsFromRAM(pstrType[eType]);
                lcdGotoXY(1, 0);
                sprintf(strResult, "%3u %3u  %3u %3u", usResult[0], usResult[1],
usResult[2], usResult[3]);
                lcdPutsFromRAM(strResult);
        }
}
```

ДОДАТОК Г

Лістинг бібліотеки hd44780.h

```
#include "hd44780.h"
#include <avr/pgmspace.h>

//-------------------------------------------------------------
void lcdSendNibble(const char byte, const char state) {

	LCDCONTROLDDR |= LCD_CONTROL_MASK;
	LCDDATADDR |= LCD_DATA_MASK;

	LCDCONTROLPORT &= ~(1 << LCD_RW);

	if(state) {
		LCDCONTROLPORT |= (1 << LCD_RS);
	} else {
		LCDCONTROLPORT &= ~(1 << LCD_RS);
	}

	LCDCONTROLPORT |= (1 << LCD_E);

	LCDDATAPORT &= ~LCD_DATA_MASK;

	if(byte & (1 << 3)) { LCDDATAPORT |= (1 << LCD_D7); }
	if(byte & (1 << 2)) { LCDDATAPORT |= (1 << LCD_D6); }
	if(byte & (1 << 1)) { LCDDATAPORT |= (1 << LCD_D5); }
	if(byte & (1 << 0)) { LCDDATAPORT |= (1 << LCD_D4); }

	_delay_us(LCD_STROBEDELAY_US);

	LCDCONTROLPORT &= ~(1 << LCD_E);
}

//-------------------------------------------------------------
char lcdGetNibble(const char state) {

	char temp = 0;

	LCDCONTROLDDR |= LCD_CONTROL_MASK;
	LCDCONTROLPORT |= (1 << LCD_RW);

	if(state) {
		LCDCONTROLPORT |= (1 << LCD_RS);
	} else {
		LCDCONTROLPORT &= ~(1 << LCD_RS);
```

```
        }

        LCDCONTROLPORT |= (1 << LCD_E);

        LCDDATADDR &= ~LCD_DATA_MASK;
        LCDDATAPORT |= LCD_DATA_MASK;

        _delay_us(LCD_STROBEDELAY_US);

        LCDCONTROLPORT &= ~(1 << LCD_E);

        if(LCDDATAPIN & (1 << LCD_D7)) { temp |= (1 << 3); }
        if(LCDDATAPIN & (1 << LCD_D6)) { temp |= (1 << 2); }
        if(LCDDATAPIN & (1 << LCD_D5)) { temp |= (1 << 1); }
        if(LCDDATAPIN & (1 << LCD_D4)) { temp |= (1 << 0); }

        return(temp);
}

//-------------------------------------------------------------
char lcdRawGetByte(const char state) {

       char temp = 0;

        temp |= lcdGetNibble(state);
        temp = temp << 4;
        temp |= lcdGetNibble(state);

        return temp;
}

//-------------------------------------------------------------
void lcdRawSendByte(const char b, const char state) {

        char byte = b;

        #ifdef lcdEnableRussian
                if(state) {
                        byte = toRussian(byte);
                }
        #endif

        lcdSendNibble((byte>>4), state);
        lcdSendNibble(byte, state);
}

//-------------------------------------------------------------
char lcdIsBusy() {
```

```
        if(lcdRawGetByte(LCD_COMMAND) & (1<<7)) {
                return(0xff);
         } else {
                return(0x00);
         }
}

//-------------------------------------------------------------
void lcdInit() {

        _delay_us(LCD_INIT_DELAY1_US);
        lcdSendNibble(0x30, LCD_COMMAND);
        _delay_us(LCD_INIT_DELAY2_US);
        lcdSendNibble(0x30, LCD_COMMAND);
        _delay_us(LCD_INIT_DELAY3_US);
        lcdSendNibble(0x30, LCD_COMMAND);

        while(lcdIsBusy());
        lcdSendNibble(0b0010, LCD_COMMAND);             // 4bit mode

        while(lcdIsBusy());
        lcdRawSendByte(0b00101000, LCD_COMMAND);        // 4bit mode, N-flag

        while(lcdIsBusy());
        lcdRawSendByte(0b00000001, LCD_COMMAND);        // Set AC to
DDRAM

        while(lcdIsBusy());
        lcdRawSendByte(0b00000110, LCD_COMMAND);        // Shift direction

        while(lcdIsBusy());
        lcdRawSendByte(0b00001100, LCD_COMMAND);        // Display mode
}

//-------------------------------------------------------------
void lcdSetCursor(const char cursor) {

        while(lcdIsBusy());
        lcdRawSendByte((0b00001100 | cursor), LCD_COMMAND);
}

//-------------------------------------------------------------
void lcdSetDisplay(const char state) {

        while(lcdIsBusy());
         lcdRawSendByte((0b00001000 | state), LCD_COMMAND);
}
```

```
//-------------------------------------------------------------
void lcdClear() {

	while(lcdIsBusy());
	lcdRawSendByte(0b00000001, LCD_COMMAND);
}

//-------------------------------------------------------------
void lcdGotoXY(const char str, const char col) {

	char offset[4] = { LCD_ROW1_OFFSET, LCD_ROW2_OFFSET,
LCD_ROW3_OFFSET, LCD_ROW4_OFFSET };

	while(lcdIsBusy());
	lcdRawSendByte((0b10000000 | (offset[(unsigned char) str] + col)),
LCD_COMMAND);
}

//-------------------------------------------------------------
void lcdDisplayScroll(const char position, char dir) {

	char pos = position;

	while(pos) {
		while(lcdIsBusy());
		lcdRawSendByte((0b00011000 | dir), LCD_COMMAND);
		pos--;
	}
}

//-------------------------------------------------------------
void lcdPutsFromRAM(const char *string) {

	const char *str = string;

	while(*str) {
		while(lcdIsBusy());
		lcdRawSendByte(*str++, LCD_DATA);
	}
}

//-------------------------------------------------------------
void lcdPutsFromFlash(const char *string) {

	const char *str = string;

	while(pgm_read_byte(str)) {
```

```
            while(lcdIsBusy());
            lcdRawSendByte(pgm_read_byte(str++), LCD_DATA);
        }
}

//-------------------------------------------------------------
void lcdLoadCharacterFromRAM(const char code, const char *pattern) {

        char ct;
        const char *pat = pattern;

        while(lcdIsBusy());
        lcdRawSendByte((code << 3) | 0b01000000, LCD_COMMAND);

       for(ct = 0; ct <= 7; ct++) {
            while(lcdIsBusy());
            lcdRawSendByte(*pat++, LCD_DATA);
        }

        while(lcdIsBusy());
        lcdRawSendByte(0b10000000, LCD_COMMAND);
}

//-------------------------------------------------------------
void lcdLoadCharacterFromFlash(const char code, const char *pattern) {

        char ct;
        const char *pat = pattern;

        while(lcdIsBusy());
        lcdRawSendByte((code << 3) | 0b01000000, LCD_COMMAND);

        for(ct = 0; ct <= 7; ct++) {
            while(lcdIsBusy());
            lcdRawSendByte(pgm_read_byte(pat++), LCD_DATA);
        }

        while(lcdIsBusy());
        lcdRawSendByte(0b10000000, LCD_COMMAND);
}

//-------------------------------------------------------------
#ifdef lcdEnableRussian
char toRussian(const char in) {

        char byte;

        switch(in) {
```

```
case 0xC0: byte = 0x41; break; // А
case 0xC1: byte = 0xA0; break; // Б
case 0xC2: byte = 0x42; break; // В
case 0xC3: byte = 0xA1; break; // Г
case 0xC4: byte = 0xE0; break; // Д
case 0xC5: byte = 0x45; break; // Е
case 0xA8: byte = 0xA2; break; // Ё
case 0xC6: byte = 0xA3; break; // Ж
case 0xC7: byte = 0xA4; break; // З
case 0xC8: byte = 0xA5; break; // И
case 0xC9: byte = 0xA6; break; // Й
case 0xCA: byte = 0x4B; break; // К
case 0xCB: byte = 0xA7; break; // Л
case 0xCC: byte = 0x4D; break; // М
case 0xCD: byte = 0x48; break; // Н
case 0xCE: byte = 0x4F; break; // О
case 0xCF: byte = 0xA8; break; // П
case 0xD0: byte = 0x50; break; // Р
case 0xD1: byte = 0x43; break; // С
case 0xD2: byte = 0x54; break; // Т
case 0xD3: byte = 0xA9; break; // У
case 0xD4: byte = 0xAA; break; // Ф
case 0xD5: byte = 0x58; break; // Х
case 0xD6: byte = 0xE1; break; // Ц
case 0xD7: byte = 0xAB; break; // Ч
case 0xD8: byte = 0xAC; break; // Ш
case 0xD9: byte = 0xE2; break; // Щ
case 0xDA: byte = 0xAD; break; // Ъ
case 0xDB: byte = 0xAE; break; // Ы
case 0xDC: byte = 0x62; break; // Ь
case 0xDD: byte = 0xAF; break; // Э
case 0xDE: byte = 0xB0; break; // Ю
case 0xDF: byte = 0xB1; break; // Я
case 0xE0: byte = 0x61; break; // а
case 0xE1: byte = 0xB2; break; // б
case 0xE2: byte = 0xB3; break; // в
case 0xE3: byte = 0xB4; break; // г
case 0xE4: byte = 0xE3; break; // д
case 0xE5: byte = 0x65; break; // е
case 0xB8: byte = 0xB5; break; // ё
case 0xE6: byte = 0xB6; break; // ж
case 0xE7: byte = 0xB7; break; // з
case 0xE8: byte = 0xB8; break; // и
case 0xE9: byte = 0xB9; break; // й
case 0xEA: byte = 0xBA; break; // к
case 0xEB: byte = 0xBB; break; // л
case 0xEC: byte = 0xBC; break; // м
case 0xED: byte = 0xBD; break; // н
```

```
                case 0xEE: byte = 0x6F; break; // о
                case 0xEF: byte = 0xBE; break; // п
                case 0xF0: byte = 0x70; break; // р
                case 0xF1: byte = 0x63; break; // с
                case 0xF2: byte = 0xBF; break; // т
                case 0xF3: byte = 0x79; break; // у
                case 0xF4: byte = 0xE4; break; // ф
                case 0xF5: byte = 0x78; break; // х
                case 0xF6: byte = 0xE5; break; // ц
                case 0xF7: byte = 0xC0; break; // ч
                case 0xF8: byte = 0xC1; break; // ш
                case 0xF9: byte = 0xE6; break; // щ
                case 0xFA: byte = 0xC2; break; // ъ
                case 0xFB: byte = 0xC3; break; // ы
                case 0xFC: byte = 0xC4; break; // ь
                case 0xFD: byte = 0xC5; break; // э
                case 0xFE: byte = 0xC6; break; // ю
                case 0xFF: byte = 0xC7; break; // я
                default: byte = in; break;
        }
        return(byte);
}
#endif
```

ДОДАТОК Д

Список Fuse бітів

SPI Enable

Ext XTAL, High frequecy

Startup: 64ms + 258 CK

BOD enabled, 4.0V

Disable JTAG

ДОДАТОК Е

Друкована плата