МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

О.В. Глухов, О.О. Кравчук, Є.В. Левченко

Вивчення властивостей мікроконтролерів і електронних систем на базі платформи Ардуіно

навчальний посібник

Харків 2019

УДК 621.396.67

Глухов О.В., Кравчук О.О., Левченко Є.В. Вивчення властивостей мікроконтролерів і електронних систем на базі платформи Ардуіно: навч. посібник для студентів ВНЗ. Харків: ХНУРЕ, 2019. – 192 с.

ISBN 978-966-659-273-9

Посібник присвячений вивченню основних принципів моделювання та побудови електронних схем, програмуванню мікроконтролерів. У посібнику розглядаються питання розробки власних електронних схем, використовується широкий набір компонентів: від транзисторів і світлодіодів до сервомоторів і плат Arduino, які дозволяють запрограмувати інтелектуальне управління елементами схеми.

Призначено для студентів спеціальності 171 Електроніка та 153 Мікро- та наносистемна техніка.

Рецензенти:

О.Ю. Панченко, д-р фіз.-мат. наук, проф., завідувач кафедри проектування та експлуатації електронних апаратів Харківського національного університету радіоелектроніки;
І.І. Обод, д-р техн. наук, професор, професор кафедри мікропроцесорних технологій і систем

Іл.: 62. Бібліогр.: наймен.: 20.

ISBN 978-986-659-273-9

ЗМІСТ

ВСТУП

Цей навчальний посібник можна вважати першим кроком на шляху до освоєння «практичного» інженерного моделювання, на якому студенти ознайомляться з основними принципами моделювання та побудови електронних схем, програмування мікроконтролерів.

Посібник присвячений питанням конструювання електронних схем, а також програмуванню логіки роботи деяких її компонентів, як, наприклад, мікроконтролерів Arduino. Для створення простих електроних пристроїв і систем можна використовувати безкоштовну віртуальну лабораторію https://circuits.io/lab/.

Дана віртуальна лабораторія дозволяє створювати власні електронні схеми, використовуючи широкий набір компонентів: від транзисторів і світлодіодів до сервомоторів і плат Arduino, що дозволяють запрограмувати інтелектуальне управління елементами схеми.

Також даний посібник дозволяє студентам ознайомитися з низькорівневими питаннями робототехніки: робота з сенсорами, датчиками, серводвигунами, пристроями живлення, мікроконтролерами – з усім тим, із чого згодом будується складна, багатокомпонентна модель робота.

Завдання навчального посібника.

Навчальні:

– ознайомити учнів з основними принципами конструювання і програмування електроних систем, а також створення найпростіших роботів;

– сформувати і розвинути абстрактне і логічне мислення.

Розвиваючі:

– розвинути навички написання комп'ютерних програм для управління складними процесами та явищами (на прикладі систем Arduino);

– сформувати навчальну мотивацію і мотивацію до творчого пошуку;

– розвинути творчий і раціональний підхід до вирішення поставлених завдань;

– розвивати в учнів елементи технічного мислення, винахідливості, образне і просторове мислення;

– розвинути логічне мислення.

Вимоги до попередньої підготовки.

Для освоєння курсу студенти повинні мати базові навички програмування однією з мов високого рівня (C, C++). Посібник містить стислий опис мови Сі, що використовується в тексті, але цього не достатньо для людей, які не мають досвіду програмування іншими мовами.

Необхідно також мати початкові знання електротехніки у рамках загального курсу фізики:

– фізичні величини: сила струму, напруга та опір;

– прилади для вимірювання цих величин;

– закон Ома;

– правила Кірхгофа;

– послідовне та паралельне з'єднання опорів.

Для вивчення додаткової літератури і подальшого саморозвитку в даному напрямку бажано мати навички читання технічної літератури англійською мовою.

Цільова аудиторія і компетенції розвитку.

Цей навчальний посібник призначений для студентів спеціальності 171 Електроніка. Матеріал посібника може бути використаний для організації навчальної практики студентів початкових курсів, а також під час проведення лабораторних практикумів з електронними системами.

Даний навчальний посібник спрямований на розвиток таких компетенцій:

– здатність брати участь як виконавець в науково-дослідних розробках нових робототехнічних і механотронних системах;

– здатність застосовувати в професійній діяльності сучасні мови програмування і мови баз даних, методології системної інженерії, системи автоматизації проектування, електронні бібліотеки та колекції, мережні технології, бібліотеки та пакети програм, сучасні професійні стандарти інформаційних технологій;

– здатність до розробки алгоритмічних й програмних рішень в області системного та прикладного програмування, математичних, інформаційних та імітаційніх моделей, створення інформаційних ресурсів глобальних мереж, освітнього контенту, прикладних баз даних, тестів і засобів тестування систем і засобів на відповідність стандартам та вихідним вимогам;

– здатність використовувати сучасні інструментальні та обчислювальні засоби.

1 ВБУДОВАНІ СИСТЕМИ

1.1 Основні поняття вбудованих систем

Отже, що таке вбудовані системи? Можна дати просте визначення: вбудовані системи – це комп'ютерні системи, які зовсім не схожі на комп'ютери. У таких системах складність комп'ютера прихована від користувача.

Швидше за все у вас є комп'ютер, наприклад, ноутбук або щось подібне. І швидше за все у вас виникали складності в його використанні. Скажімо, ви хотіли встановити нове програмне забезпечення, але стався конфлікт, і ви не змогли цього зробити. Так іноді буває з комп'ютерними іграми. Ви купили нову гру, а вона не працює, тому що їй потрібна нова версія драйверів відеокарти. Ви встановлюєте нові драйвери, а у вас перестає працювати інша гра, яка працювала до цього, оскільки їй не підходять нові драйвера. Або нові драйвери можуть взагалі не працювати з вашою відеокартою і треба купити також й нову відеокарту. Всі ці проблеми виникають через універсальність наших «звичайних» комп'ютерів: ми використовуємо їх для вирішення дуже різних завдань, причому ми намагаємося їх вирішувати за допомогою одного і того ж обчислювального пристрою.

З вбудованими системами все не так. Як правило, вбудована система використовується для якоїсь однієї конкретної задачі. Якщо це камера, то вона повинна знімати зображення або відео. Якщо це автомобіль, то він повинен їздити як автомобіль. Усередині них може перебувати спеціальна комп'ютерна система, але тільки вона вбудована в пристрій і тому її складність прихована від користувача, який знає, як працювати з цими системами через простий інтерфейс.

Наприклад, цифрова камера використовує той самий інтерфейс, що і стара механічна камера – потрібно натиснути кнопку і вона зробить

фотографію. Зовні цифрова камера управляється так само, як механічна, але всередині вона влаштована набагато складніше, за рахунок чого користувач отримує додаткові переваги. Йому не потрібно більше проявляти плівку, він може відразу ж побачити результат і навіть більш того – саме тут на камері виконати його попередню обробку, яку раніше можна було зробити тільки на комп'ютері. Ще раз повторимо, що при цьому вся ця складність прихована від користувача за простим і зрозумілим інтерфейсом.

Вбудовані системи не завжди взаємодіють безпосередньо з користувачем: вони можуть робити це через інший пристрій. Що це означає? Давайте розглянемо USB накопичувач, який зберігає дані. Ця річ, як ми розуміємо, не взаємодіє безпосередньо з людиною, адже у нас немає розніму для підключення такого накопичувача до свого тіла або чогось на кшталт цього! Ми підключаємо його до комп'ютера, і потім взаємодіємо з ним через комп'ютер, щоб отримати доступ до файлів. І це не просто мікросхема пам'яті: в накопичувачі є мікрокомп'ютер, який управляє його роботою. Таким чином, це теж вбудована система, хоча вона безпосередньо не взаємодіє з людиною.

Те ж саме може бути і в інших пристроях, наприклад, антиблокувальна система всередині автомобіля. Людина натискає на педаль гальма, а педаль гальма, в свою чергу, пов'язана з антиблокувальною гальмівною системою, таким чином, людина взаємодіє з автомобілем, а в автомобілі взаємодіють один з одним його підсистеми.

Важливою особливістю вбудованих систем, що відрізняє їх від звичайних комп'ютерів, є те, що в їхній роботі ключовим фактором є ефективність. Це означає, що не завжди достатньо просто виконувати потрібну функцію, щоб вирішувати завдання. Можна сказати, що спосіб вирішення має бути нестандартним. Наприклад, він має працювати швидко, або він має працювати з низьким енергоспоживанням, або він повинен бути дешевим. І в цьому насправді полягає велика відмнність між розробкою вбудованих систем і, наприклад, традиційним дизайном програмного забезпечення.

Коли вивчають програмування, то як кінцева мета зазвичай ставиться вирішення якоїсь конкретної задачі. Достатньо написати код, який робить те, що потрібно, наприклад, сортує список. При цьому найчастіше не потрібно, щоб алгоритм використовував мінімально можливу кількість пам'яті, і щоб список з мільйона елементів сортувався за 0,01 секунди. Іноді такі обмеження важливі і студенти відповідних напрямів підготовки навіть вивчають цілі дисципліни, присвячені ефективним алгоритмам і структурам даних, але на практиці в звичайному програмуванні можуть бути вирішені завдання і не завжди висувають такі підвищені вимоги до ефективності – достатньо, щоб програма просто працювала.

Але для вбудованих систем ефективність дуже важлива. Недостатньо просто змусити таку систему працювати. Вона повинна робити це ефективним чином. Причиною цих обмежень є те, що для більшості цих пристроїв дуже важливо зниження їхньої собівартості, як, наприклад, для пристроїв споживчої електроніки. Або ж від їхньої роботи може залежати життя і здоров'я людей, якщо, наприклад, такі пристрої використовують лікарі або військові.

Розглянемо перший приклад: зі споживчим пристроєм. Припустимо, ви розробляєте новий телефон. Для таких пристроїв виробничі витрати, а також витрати на проектування, і час виходу на ринок є первинними. Ви хочете заробити гроші на пристрої, тому його вартість має бути низькою, адже ніхто не купуватиме новий телефон за занадто дорогою ціною, тим паче, що у конкурентів вона швидше за все буде менше. Крім того, важлива і швидкість виходу на ринок. Ті ж самі конкуренти зроблять все, щоб випустити свій телефон раніше вас і завоювати більшу частину ринку. Все це накладає жорсткі обмеження на «начинку» вашого телефону – його продуктивність, енергоспоживання, обсяг пам'яті, кількість додаткових функцій і т.д. Занадто «товста» програмна начинка витрачатиме більше процесорного часу, працювати повільніше, а батарейка сідатиме раніше. З іншого боку, занадто «худа» програмна начинка перетворить ваш пристрій в «цеглину», за яким в

кращому випадку можна буде тільки робити дзвінки, а про будь-які додаткові функції можна буде і не мріяти.

З іншого боку, розглянемо застосування в медицині. Від медичного обладнання залежить життя людей, тому тут ви більше дбаєте про продуктивність і надійність, а не про кінцеву ціну. Якщо електронний пристрій контролює роботу серця людини, краще, щоб він був надійним, навіть якщо через це він коштуватиме дорожче. Очевидно, що під надійністю розуміють не тільки фізичні характеристики пристрою. Програмне забезпечення подібних пристроїв теж має бути надійним. Наприклад, сучасний кардіостимулятор «слухає» серцевий ритм людини, аналізує його і, якщо в ньому виникає пауза або будь-яке інше порушення, починає генерувати імпульси.

Тепер уявіть, що буде, якщо програмне забезпечення кардіостимулятора почне «пригальмовувати» так само, як це іноді роблять програми на своєму комп'ютері, посилено «шурхати» при цьому твердим диском. Або ще гірше – візьме і зависне. Результат, думаю, буде очевидний.

Обговоримо відмінності в процесі розробки вбудованих пристроїв і звичайних комп'ютерів. Одним з найбільш важливих відмінностей є те, що на відміну від настільних і портативних комп'ютерів, які можуть запускати програми будь-якого типу, вбудовані пристрої, як правило, розробляються під конкретну задачу (або набір пов'язаних завдань). Сучасні телефони є винятком, тому що вони розмивають цю різницю. Телефони починалися як типові вбудовані системи, але сьогодні вони практично перетворилися в повноцінні універсальні комп'ютери. Тим не менш, більшість пристроїв, таких як камери, електроніка автомобіля, побутова техніка та інше зроблені для вирішення конкретних завдань. Наприклад, камера – щоб знімати. І вона робить все, що пов'язано з цією діяльністю камери, не роблячи інших речей. Ця властивість є загальним для більшості вбудованих систем. Це важливо, тому що це змінює спосіб розробки пристрою.

Таким чином, розробка фокусується на одному застосуванні, на відміну від систем загального призначення, таких як ноутбук або настільний комп'ютер. Універсальні ноутбуки та комп'ютери часто виявляються більш потужними, ніж необхідно для вирішення того завдання, для якої вони використовуються. Це відбувається тому, що вони мають бути готові працювати з будь-яким типом обчислювальних задач. Так, наприклад, візьмемо стандартний комп'ютер, скажімо з чотириядерним процесором. У вас може бути чотириядерний Core i7, який працює на частоті 4 ГГц. Як правило, ви не використовуєте увесь обчислювальний потенціал цієї машини. Вам зовсім не потрібно чотири процесори для запуску MS Word або PowerPoint, незалежно від того, що ви в них робите. Вам також не потрібно 4 ГГц для запуску цих програм. Комп'ютер встигає зробити мільйони різних операцій до того, як ви надрукуєте чергову букву. Відкрийте, заради інтересу, диспетчер задач свого ноутбука і подивіться на відсоток використання часу центрального процесора. Як правило (якщо ви не граєте в гру в даний момент і не рендеруете 3D-сцену), велика частина додатків спить і тільки одиниці споживають десяті частки відсотка (сам диспетчер задач, наприклад). Тобто 99% відсотків ресурсів центрального процесора не використовується!

Але іноді ця потужність все-таки потрібна. Наприклад, якщо ви обробляєте фотографії або відео. Такі додатки вимагають всю обчислювальну потужність, яка є в комп'ютері. Інший приклад – це ігри зі складною графікою, що вимагають величезну обчислювальну потужність. Тому, коли ви в них граєте, ви використовуєте всі можливості вашого комп'ютера. Але в решті часу ця потужність не використовується.

У цьому сенсі універсальні комп'ютери дуже неефективні. Якщо ви хочете грати в нові ігри, вам доведеться купити дорогий потужний комп'ютер з гарною відеокартою. Або, ви можете купити ігрову приставку, яка зможе запускати ті ж ігри, може бути навіть швидше, і вона буде дешевше комп'ютера. Приставку можна використовувати тільки для ігор, але вона робить це краще,

ніж комп'ютер. Тобто її архітектура більш ефективна для вирішення даного завдання. І більшу частину часу ви використовуватимете її, якщо не на повну потужність, то принаймні на більшу частину потужності, оскільки тільки гратимете на ній.

Таким чином, конкретне застосування дозволяє вплинути на процес розробки системи. Ви можете включити в неї тільки ті блоки, які дійсно необхідні. Завдяки цьому стає можливою більш висока ефективність конструкції. Це означає, що, якщо я знаю, що моє обладнання призначене тільки для ігор, то є певні компоненти, які мені не потрібні в моїй конструкції. Навіщо купувати те, що не потрібно? Розглянемо фотоапарат. Я знаю, що ця річ має робити деякі види обробки фото і відео. Я знаю, що потрібно управляти об'єктивом і спалахом. Але камері ніколи не доведеться рендерити 3D-графіку, роздавати оточуючим Wi-Fi, підключати до себе, обслуговувати десяток периферійних пристроїв і виконувати багато інших функцій. Тому, я можу просто включити тільки те програмне забезпечення і апаратні засоби, які мені потрібні для виконання моїх конкретних завдань. І це зробить пристрій дешевше і ефективніше. На жаль, але за все треба платити. Загальне правило нашого життя, яке виконується і для обчислювальних систем – чим більше універсальний пристрій, тим менше він ефективний, і навпаки. Звичайно, лікар загальної практики знає про зуби більше, ніж звичайна людина, але хіба ви підете до нього, якщо у вас заболить зуб? Ні, ви віддасте перевагу вузькоспеціалізованому, і від того більш ефективному для вирішення даного завдання зубного лікарю.

Ще однією великою відмінністю вбудованих систем порівняно зі звичайними настільними комп'ютерами і ноутбуками є те, що апаратне і програмне забезпечення, як правило, розробляється спільно. Якщо, скажімо, ви хочете купити Microsoft Word або іншу програму, ви, очевидно, можете придбати її окремо від комп'ютера. І їх зробили дві абсолютно різні фірми. Наприклад, ваш ноутбук може бути зроблений фірмою Asus, а Word – фірмою

Microsoft. Одна фірма робить обладнання, а інша – програми. Для вбудованих систем все навпаки. Саму TV-приставку і програмну оболонку для неї робить одна і та ж компанія. Ми говоримо «як правило», тому що, звичайно ж, з усього є винятки. Наприклад, компанія Apple робить все сама – і комп'ютери, і ноутбуки, і телевізори, і телефони, і операційні системи, і офісні додатки, іншими словами – весь спектр універсальних і вбудованих систем.

Чому зручно робити вбудовані системи повністю самому? Тому що я можу зробити обладнання саме таким, яке необхідно для запуску саме того програмного забезпечення, що потрібно мені. Я можу зробити їх відповідними один одному. Весь процес проектування стає більш ефективним. Ми купимо тільки ті компоненти, які потрібні, щоб побудувати систему, яку ми хочемо. Ми не говоритимемо, будь-яка система. Ми візьмемо тільки ті компоненти, які потрібні для нашої конкретної системи. І ми писатимемо код, який використовує тільки їх.

Це відрізняє вбудовані системи від систем загального призначення, в яких потрібно мати обладнання на всі випадки життя і програмне забезпечення для всіх випадків життя, навіть якщо це буде необхідно в цьому житті лише раз.

Питання для самоконтролю

1. Виберіть правильні твердження:

a) вбудована система використовується для однієї конкретної задачі;

b) вбудовані системи завжди взаємодіють безпосередньо з користувачем;

c) однією з особливостей вбудований систем є те, що для них дуже важлива ефективність.

2. Визначте, які з рис зазвичай притаманні вбудованим системам:

a) вбудовані пристрої зазвичай розробляються для конкретної задачі;

b) апаратне та програмне забезпечення розробляється окремо;

c) вбудовані системи багатофункціональні и універсальні.

1.2 Компоненти вбудованої системи

У цьому розділі ми поговоримо про структуру вбудованих систем. Конкретно, ми обговоримо структуру апаратних засобів, подивимось на її різні компоненти, причому основний компонент – мікроконтролер – обговоримо більш детально в наступному розділі.

На рис. 1.1 схематично зображена загальна схема. Вбудована система має отримувати дані з зовнішнього світу, обробляти їх і потім виводити дані в зовнішній світ. Так що, в першу чергу, вона має набір датчиків для прийому даних. Ці датчики можуть отримувати інформацію про зовнішній світ по-різному, тому існує багато різних типів подібних пристроїв.

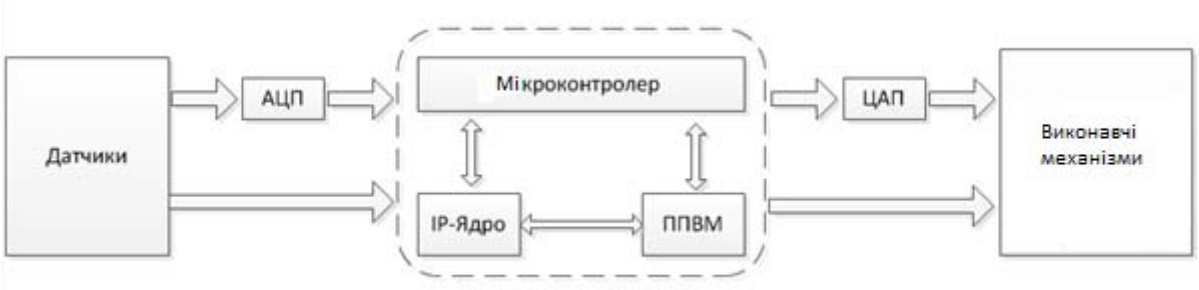


Рисунок 1.1 – Влаштування вбудованих систем

Найпростіший тип датчика – це кнопка або щось подібне, яка отримує дані в дуже простому вигляді: натиснута або не натиснута. Система може отримувати і звукову інформацію, використовуючи мікрофон. Відеокамери – це теж датчики, які отримують інформацію у вигляді зображення. Сенсорний екран дозволяє не тільки виводити, а й отримувати інформацію. І так далі – існує велика безліч найрізноманітніших типів датчіків, через які вбудована система може отримувати інформацію.

Отже, вхідна інформація надходить в систему і далі вона потрапляє в її ядро, яке займається її обробкою. В кінці цього процесу, коли система вирішила, що робити з інформацією або яке рішення потрібно прийняти на її основі, вона має видати якісь результати. Найчастіше це означає зробити якусь дію в зовнішньому світі. Це робиться за допомогою виконавчих механізмів або приводів, які показані в правій частині схеми на рис. 1.1.

Виконавчим механізмом може бути світлодіод, який може горіти або блимати. Например, він може сигналізувати, що відеокамера записує або що закінчується батарейка. Але є й інші виконавчі механізми. Наприклад, всередині камери є двигуни, за допомогою яких відбувається управління об'єктивом і наведення різкості. Ці двигуни є виконавчими механізмами, а їхній рух – це вихід системи.

Існує безліч різніх типів виконавчих механізмів: динаміки відтворюють звук, лампи дозволяють системі виводити світло, екран також виводить світло, але в більш детальному і «керованому» вигляді.

Отже, якщо ви зараз подивитесь на вбудовану систему, то побачите, що вона знаходиться посередині. Вона отримує дані від датчиків, щось з ними робить, а потім посилає відповідні сигнали на виконавчі пристрої, щоб змусити їх щось зробити в реальному світі у відповідь на дані, які вона отримала. У цьому сенсі вбудована система забезпечує зв'язок між датчиками і виконавчими механізмами.

У центрі вбудованої системи знаходиться мікроконтролер. Більш детально ми обговоримо його в наступних розділах. Крім мікроконтролерів можуть бути й інші компоненти. Два з тих, які зустрічаються досить часто, показані на рис. 1.1: це IP-ядра і ППВМ.

IP – це англійське скорочення від intellectual property, тобто інтелектуальний продукт. IP-ядро становить собою готові блоки для проектування пристроїв, наприклад, це може бути готова мікросхема, яка виконує одну функцію. Звичайно «одну функцію» не треба розуміти надто буквально –

IP-ядро може виконувати кілька функцій, але це тісно пов'язані функції, орієнтовані на якусь конкретну задачу. Тобто цей блок не є універсальною програмованою мікросхемою загального призначення. Навпаки, це мікросхема, яка просто вміє виконувати деякий невеликий набір пов'язаних між собою функцій. Такі пристрої можуть бути дуже корисні, і до того ж вони дешеві в ході виробництва у великих обсягах. Нехай у вас є досить загальна система, що має велику кількість різних підсистем, як, наприклад, мобільний телефон. Всі мобільні телефони мають виконувати певні алгоритми обробки звуку. Можна зробити спеціальну мікросхему, яка виконуватиме таку обробку і її можна буде продавати виробникам телефонів. Оскільки телефонів виробляють дуже багато, то і мікросхем буде потрібно багато. Щоб зробити одну таку мікросхему, буде потрібно багато часу, праці і грошей і це було б вкрай невигідно, але, якщо необхідно провести сотні тисяч або навіть мільйони подібних мікросхем, – то собівартість кожної з них стає дуже низькою.

IP-ядра роблять для поширених завдань – для тих, які виникають знову і знову і на які є великий попит. Якщо якась задача зустрічається тільки в одній конкретній системі, то для неї не має сенсу робити IP-ядро.

Розглянемо, наприклад, мережні контролери. Вони зустрічаються в багатьох системах. Будь-який пристрій, який підключається до мережі Інтернет, повинен мати відповідний пристрій. Таку річ можна реалізувати у вигляді IP-ядра – спеціалізованої частини апаратних засобів, що забезпечує тільки функцію взаємодії з комп'ютерною мережею. Інший приклад – аудіо і відеокодеки, які займаються кодуванням аудіо і відео, їхнім стисненням і розпакуванням. Ці операції виконуються у великій кількості вбудованих пристроїв, тому це якраз приклад такої функції, яку можна було б помістити в IP-ядрі.

Коли ви робите вбудовану систему, то буде корисно не «винаходити велосипед», а скористатися вже готовими IP-ядрами. Ви можете подивитися каталоги фірм, які роблять IP-ядра, наприклад, Texas Instruments. Такі фірми

випускають безліч спеціалізованих мікросхем, і ви зможете знайти те, що вам потрібно. Наприклад, якщо потрібно реалізувати стиснення за стандартом MPEG, то у Texas Instruments ви знайдете кілька десятків мікросхем, які його роблять, і вибрати відповідну.

IP-ядра мають взаємодіяти з мікроконтролером, який є центром всієї системи. Він керує IP-ядрами і командує, коли вони повинні починати роботу, передає їм інформацію і отримує результат. Наприклад, якщо IP-ядро виконує стиснення відео, то мікроконтролер каже йому, коли починати стискати дані і які дані стискати. Для цього мікроконтролер передає йому певну послідовність сигналів, а коли мікросхема закінчує свою роботу – вона посилає певні сигнали мікроконтролеру, повідомляючи, що результат готовий.

Інший вид компонентів, який застосовується у вбудованих системах, це вентильні матриці, або ППВМ. Ми не розглядатимемо ППВМ у цьому посібнику, тому що це досить складні, хоча і цікаві пристрої, але ви повинні знати, що це таке. ППВМ – це апаратні пристрої, а якщо бути точніше – апаратно програмовані мікросхеми. Що це означає? Звичайна мікросхема складається з величезної кількості побудованих на базі транзисторів логічних вентилів. Вентилі, в свою чергу, з'єднані між собою дуже складними численними зв'язками, за якими, власне, і подорожують керуючі та інформаційні сигнали. Схема об'єднання цих мікроскопічних пристроїв і обумовлює логіку виконуваних мікросхемою операцій, але тільки ця схема «друкується» на заводі.

У свою чергу ППВМ можна образно порівняти з набором юного мехатроніка, що складається з декількох десятків транзисторів, резисторів, лампочок, кнопок, проводів, клем, батарейок, коліщаток, приводів і інших елементів. Маючи цей набір, ви можете зібрати з нього апаратний пристрій, який виконуватиме якесь завдання – наприклад, машинку, яка вміє їздити по поверхні. Потім, якщо вам знадобиться інший пристрій, наприклад, човен з двигуном, який здатний плавати по воді, ви розберете машинку на складові

частини і перекомпонуєте їх у кораблик. Це зовсім образно можна уявити як процес апаратного програмування набору юного мехатроніка. ППВМ – це складна, хитро влаштована мікросхема, яка дозволяє змінювати малюнок взаємозв'язків своїх логічних компонентів, тобто по суті апаратну конфігурацію пристрою. Тут важливо ще раз підкреслити, що мова йде не про перепрошивку програмної начинки пристрою, а про зміну його апаратної структури.

Що це нам дає? Головна перевага ППВМ у тому, що вони працюють швидше, ніж звичайні мікропроцесори, які виконують ті ж функції, але програмним шляхом. ППВМ може замінити спеціалізовану мікросхему, таку як IP-ядро. З ППВМ не потрібна розробка і виробництво спеціальної мікросхеми, оскільки можна взяти ППВМ і запрограмувати її на виконання потрібного завдання, а це набагато дешевше. Спеціалізовані ж мікросхеми, в свою чергу, можуть працювати ще швидше, ніж ППВМ, і вони будуть дешевше, якщо їх виробляти у великих кількостях, тому що вони не включають в себе механізми переконфігурування, які є в ППВМ. Але це обгрунтовано тільки для випадку мікросхем з широко затребуваною логікою. Іншими словами, все, як завжди, залежить від розв'язуваної задачі, під яку ви підбираєте найбільш ефективну конфігурацію.

Питання для самоконтролю

1. Виберіть правильні твердження:

a) вбудована система сама генерує дані, проводить розрахунки і виводить їх у зовнішній світ;

b) вбудована система отримує дані з зовнішнього світу за допомогою набору датчиків;

c) світлодіод може виступати в ролі датчика;

d) сканер відбитків пальців може виступати в ролі датчика для вбудованої системи;

e) IP-ядра виготовляють під вузькоспеціалізовані завдання конкретних систем;

f) IP-ядра – готові блоки для проектування пристроїв;

g) аналогом програмування для ППВМ виступає зміна з'єднання елементів.

2. Що працює швидше: програма, що виконується на звичайному процесорі або ППВМ, що реалізує ту ж логіку, що і програма?

1.3 Мікроконтролери

У цьому параграфі ми продовжимо говорити про апаратні компоненти вбудованих систем, а саме про мікроконтролери.

Мікроконтролер є центром вбудованої системи, тому в цьому навчальному посібнику ми приділимо багато уваги саме роботі з ним. На рис. 1.2 ви бачите одну з плат Arduino. Це не мікроконтролер, це друкована плата з безліччю елементів, серед яких, придивившись, ви зможете побачити велику чорну прямокутну мікросхему. Ось це якраз і є мікроконтролер, який виконує програми, що керують пристроєм.

Отже, робота даної мікросхеми полягає в тому, щоб виконувати код, і це центр всієї системи. Він зчитує дані з інших компонентів, і він також керує іншими компонентами. Чим відрізняється мікроконтролер від мікропроцесора, який використовується в звичайних комп'ютерах? Мікроконтролер зазвичай менший і слабший, ніж мікропроцесор, тому, коли говорять про останній, то мають на увазі мікросхему від компанії Intel або AMD, що встановлюється в настільний комп'ютер або ноутбук, і яка працює на дуже великій швидкості.

Рисунок 1.2 – Плата Arduino

Мікроконтролер ж, як правило, істотно слабший. При цьому з точки зору функціонування він робить приблизно те ж саме, але тільки у вбудованих системах. З цим пов'язані і обмеження. Всі компоненти вбудованих систем мають бути енергоефективними і дешевими і для них, найчастіше, не потрібно продуктивність як у процесора з 6 ядрами і частотою 4 ГГц. Якщо потрібно реалізувати конкретну взаємодію з користувачем, то така продуктивність швидше за все і не потрібна, тому для вбудованих систем зазвичай потрібен дешевий процесор, що споживає мало енергії.

Яка продуктивність мікроконтролерів в абсолютному вираженні? Зараз досить часто використовується частота від 16 МГц, хоча може бути навіть і менше, наприклад, 8 або навіть 4 МГц. Це майже в кількасот разів повільніше, ніж процесор звичайного комп'ютера. Звичайно, можуть застосовувати і більш швидкі мікросхеми – до 500 МГц або навіть до 1 ГГц. Це потрібно, наприклад, для обробки відеозображень.

Інша відмінність від звичайних комп'ютерів в тому, що в звичайних комп'ютерах процесор і пам'ять – це окремі мікросхеми, причому зазвичай пам'ять становить кілька мікросхем. Ми можемо змінювати і додавати пам'ять незалежно від процесора. У мікроконтролерах початкового рівня все об'єднано в одну мікросхему, тобто за своєю суттю такий мікроконтролер – це міні-

комп'ютер, зроблений на одній мікросхемі, на якій знаходиться і мікропроцесор, і пам'ять, й інші необхідні компоненти. Зазвичай, пам'яті у них теж набагато менше, ніж в настільних комп'ютерах, але такий варіант може бути зручний, оскільки потрібно тільки мінімальну кількість зовнішніх компонентів, щоб мікропроцесор міг працювати. У найпростіших випадках потрібно тільки живлення.

У більш продуктивних мікроконтролерах використовують зовнішню пам'ять, яка встановлюється окремо. Це менш зручно, проте дозволяє обирати, скільки пам'яті нам потрібно, не переплачуючи за непотрібний в конкретному пристрої її обсяг. Зазвичай так роблять, коли мікроконтролеру для роботи потрібен досить великий обсяг пам'яті.

Отже, мікроконтролер – це інтегрована мікросхема, яка виконує програму. На рис. 1.3 мікроконтролер виглядає трохи по-іншому, ніж ті, що ми бачили раніше. На цьому рисунку ми заглянули всередину чорної оболонки і побачили те, як насправді виглядає мікросхема. Це маленький кремнієвий чіп, захований в захисний корпус.

Фотографія на рис. 1.3 показує мікропроцесор після того, як чіп був встановлений в корпус.

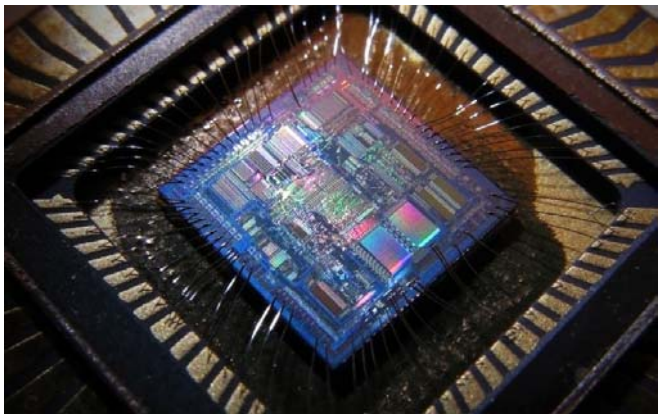


Рисунок 1.3 – Мікроконтролер

Сам корпус є чорним матеріалом, який оточує чіп. Він потрібен, перш за все, для захисту, а крім того, для охолодження мікросхеми, оскільки допомагає

більш ефективно відводити тепло. Це необхідно, тому що всі чіпи сильно нагріваються під час роботи. Настільки сильно, що якщо ви спробуєте торкнутися відкритого чіпа під час його роботи, то отримаєте такі ж відчуття, як і від розпеченої лампочки (за кількістю тепла, що виділяється, сучасні процесори можна порівняти зі звичайними 100-ватними лампочками).

Крім того, ви бачите, що по всьому периметру чіпа знаходяться контактні площадки. Це те, за допомогою чого мікросхема спілкується із зовнішнім світом. Вони з'єднані дротами з чіпом і виведені на маленькі шматочки металу на зовнішній стороні корпусу, так щоб можна було припаяти мікроконтролер до інших компонентів системи.

Зазвичай мікроконтролер ставиться на друковану плату, мідні доріжки якої з'єднують його з іншими компонентами системи. Дана мікросхема є центром системи, вона посилає команди інших компонентів і отримує від них дані. Щоб вона могла обробити ці дані, її потрібно запрограмувати. На відміну від ППВМ, апаратну архітектуру мікропроцесора міняти не можна. Зате в його пам'ять можна завантажити програму у вигляді набору інструкцій, написаних зрозумілою йому мовою, і попросити його її виконати.

Існує багато мов програмування, на яких можна писати подібні програми. В рамках цього навчального посібника ми використовуватимемо мову Сі для програмування мікроконтролера, встановлений на платі Arduino (рис. 1.2). Крім Сі використовують й інші мови. Перш ніж виконати програму, очевидно, треба десь зберегти. Отже, всередині мікроконтролера має бути пам'ять для програми. Зараз це, як правило, флеш-пам'ять, така ж, як в USB-накопичувачах.

Флеш-пам'ять – це незалежна пам'ять, тобто така, з якої дані не стираються після відключення живлення. Коли живлення з'являється знову, мікроконтролер починає виконання програми з самого початку.

Де, на чому і як пишуть програми для подібних пристроїв? Зазвичай це роблять на звичайному ноутбуці або настільному комп'ютері. Це пов'язано з тим, що як вже згадано раніше, мікроконтролери порівняно повільні і слабкі.

Тому програму пишуть і компілюють на потужному комп'ютері, а потім остаточна версія компільованої програми має бути записана в пам'ять мікроконтролера. Робиться це за допомогою спеціального пристрою, що називається програматором (рис. 1.4).

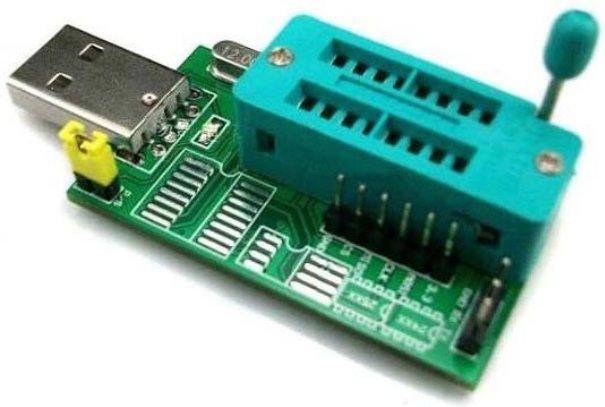


Рисунок 1.4 – Програматор

Це невелика плата, яка за допомогою USB приєднується до комп'ютера, а до розташованого на ній слоту підключається мікроконтролер. Є спеціальні програми на комп'ютері, які зв'язуються з цим пристроєм і записують через нього програму. Це один із способів запрограмувати мікроконтролер. З таким програматором мікроконтролер програмують окремо, а потім вже встановлюють в систему, де він працюватиме. Іноді буває так, що в пристрої вже передбачений спеціальний рознім, і тоді мікроконтролер програмують прямо в системі. Але в нашому випадку ми так не робитимемо, тому що ми використовуємо Arduino. Для неї не потрібен окремий програматор, оскільки він вже вбудований в плату. Можна просто взяти Arduino і, підключивши її до USB порту ноутбука або настільного ПК, програмувати її безпосередньо.

Питання для самоконтролю

1. Виберіть правильні твердження:

а) робота мікроконтролера полягає в тому, щоб виконувати код;

b) мікропроцесор менший і слабший за мікроконтролер;

c) процесор і пам'ять мікроконтролера початкового рівня знаходяться на різних мікросхемах;

d) для підвищення продуктивності мікроконтролера використовують зовнішню пам'ять.

2 Як називається пристрій, за допомогою якого остаточну версію програми переносять з комп'ютера на мікроконтролер?

1.4 Аналогові й дискретні сигнали

У цій частині ми обговоримо аналогові і цифрові сигнали. Раніше ми говорили про вбудовані системи і їхній інтерфейс, взаємодію з навколишнім світом і бачили, що на вході є набір сенсорів, які отримують інформацію ззовні. Сенсори можуть бути дуже простими, такими як кнопка, а можуть бути і складними, як, наприклад, мікрофон, датчик освітленості або камера. З іншого боку, у вбудованій системі є виконавчі механізми, якими можуть бути лампочки, світлодіоди, динаміки, екрани або щось ще. І ті, й інші забезпечують зв'язок вбудованої системи із зовнішнім фізичним світом. Вони вимірюють фізичні параметри або навпаки, виробляють якісь фізичні дії, а фізичні сигнали найчастіше є аналоговими. Водночас ядро системи – мікроконтролер – працює в дискретному оточенні – в світі «нулів і одиниць» і знати не знає про аналогові сигнали. Перш ніж інформація з зовнішнього світу надійде на обробку в мікропроцесор, вона має бути оцифрована, тобто переведена з аналогової форми в цифрову.

У чому ж відмінністть між цими формами подання сигналів? Можна навести таке порівняння – відмінність між аналоговими і цифровими сигналами така ж, як відмінністть між дійсними і цілими числами. Якщо ви пам'ятаєте, дійсні числа є безперервними, тобто на будь-якому інтервалі, який би ви не

взяли, існує нескінченне число таких чисел. З іншого боку, ви можете взяти як завгодно великий інтервал, але кількість цілих чисел на ньому буде звичайно. Власне, відмінність між аналоговими і цифровими сигналами, по суті, саме в цьому.

Давайте розглянемо який-небудь «аналоговий» приклад, наприклад, світло. Припустимо, на нас падає світло від лампи, і його яскравість можна регулювати. Як правило, яскравість світла – це аналоговий параметр. Можна взяти лампу, підключену через димер, і зробити її трохи яскравіше або трохи темніше. Залежно від того, наскільки точне управління, можна отримати дуже багато можливих значень між повністю вимкненим і повністю увімкненим світлом. Те ж саме зі звуком. Можна говорити голосно, можна тихо. Можна говорити з проміжною гучністю. Гучність можна плавно регулювати між максимально гучною і максимально тихою. Це теж аналогове явище. Всі безперервні явища нашого світу (принаймні саме так їх сприймає людина) – наприклад, час, відстань, маса, яскравість, сила, навіть смак – аналогові.

Фізики, звичайно, можуть заперечити, сказавши, що насправді це не так, що світ дискретний. Існуючі на субатомному рівні елементарні частинки можуть знаходитися тільки в суворо визначених станах, і це дискретний світ. Можливо, це так, але для наших людських почуттів світ є аналоговим. Ми не можемо сприймати стану окремих елементарних частинок, тому нам здається, що всі параметри можуть змінюватися плавно.

Але головна проблема навіть не в цьому, а в тому, що різних значень аналогового сигналу може бути дуже-дуже-дуже багато, настільки багато, що у сучасної обчислювальної системи просто не вистачить ресурсів працювати з усім його різноманіттям. З іншого боку, дискретне явище – це те, що може знаходитися тільки у фіксованому (порівняно, невеликому) числі різних станів, наприклад, «є» чи «ні», як вимикач світла на стіні – він або увімкнений, або вимкнений. У цьому випадку є всього два рівні яскравості.

Іншим прикладом може бути годинник. Бувають цифрові годинники, які показують цифри. А бувають годинники зі стрілками, і якщо стрілки рухаються повільно, то стрілка може перебувати в будь-якій проміжній позиції, в той час як цифровий годинник може показати тільки цілі години, хвилини, секунди.

Ми говоримо про це тому, що вбудовані системи взаємодіють з реальним світом, який, у рамках нашого сприйняття, багато в чому аналоговий. Нехай, наприклад, потрібен датчик, який вимірює яскравість світла, і нехай він буде аналоговим, щоб бути в змозі виміряти не тільки, що світло вимкнене або увімкнене, але і ступінь його яскравості. Проблема у тому, що, як уже було сказано вище, мікроконтролери є цифровими системами, тобто вони розуміють тільки цифрові дані, або якщо конкретніше – нулі і одиниці. Для того, щоб програма на мікроконтролері мала можливість використовувати інформацію від датчиків, потрібно щоб аналоговий сигнал був перетворений в цифрове значення. І це те, для чого потрібно аналого-цифрове перетворення.

Ми не обговорюватимемо детально сам процес, оскільки він є порівняно складним. Достатньо розуміти, що беруть аналогове дійсне значення і перетворять його в ціле число, яке є наближенням до вихідного аналогового значення. Завжди є якась помилка, але це нормально, тому що ми можемо зробити помилку настільки маленькою, щоб вона нас влаштовувала.

Отже, аналогово-цифрове перетворення (скорочено АЦП) потрібно на багатьох входах. На багатьох, але не на всіх, оскільки деякі з них можуть бути вже цифровими, наприклад, кнопки. Кнопки природним чином є цифровими: кнопка або натиснута, або відпущена.

На виході – з боку виконавчих механізмів, часто може знадобитися зворотне перетворення з цифрового сигналу в аналоговий. Наприклад, мікроконтролер має виводити звук через гучномовці. Акустичні системи є аналоговими пристроями, вони потребують аналоговий сигнал, а мікро-контролер виводить нулі і одиниці, тому тут потрібно перетворення цифрового сигналу в аналоговий. Так що дуже часто можна бачити перетворення

аналогового сигналу в цифровий на вході системи і цифрового в аналоговий на виході. Розглянемо приклад такого перетворення.

Нехай є якийсь сигнал, який змінюється з часом, як показано на рис. 1.5. Наприклад, це звукова хвиля, що є зміною тиску з часом. Якщо це чистий тон, то тиск змінюватиметься за функцією синуса, як показано плавною лінією.

По осі Y задається тиск повітря, яке сприймається мікрофоном.

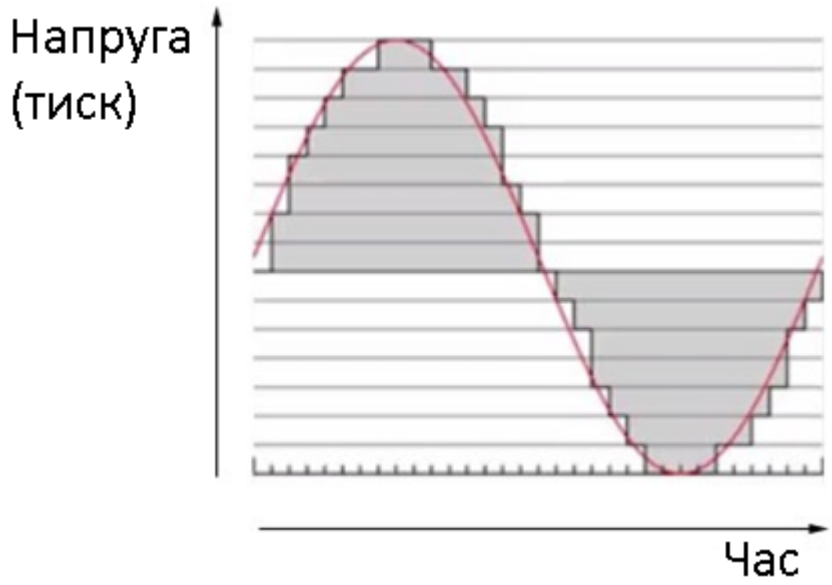


Рисунок 1.5 – Приклад аналогово-цифрового перетворення

Роль мікрофона в тому, щоб перетворити тиск в напругу, яку вже сприйматимуть електронні системи. Мікрофон сприймає тиск і видає напругу, яка змінюється так само, як змінюється тиск, тому вісь Y підписана ще і як напруга.

Але після мікрофона це все ще аналоговий сигнал, зображений у вигляді гладкої, безперервної, червоної лінії на графіку, що плавно змінюється в часі.

Значення цього сигналу – як і раніше дійсні числа, адже напруга може приймати будь-яке проміжне значення між максимумом і мінімумом.

На цій картинці можна бачити також графік ступеневого дискретного сигналу, який може приймати тільки певні значення, зображені у вигляді горизонтальних ліній. Ці значення відповідають цілим числам 0, 1, 2, 3 і т. д.

Власне, цифровий сигнал може приймати тільки такі значення, і не може приймати проміжні.

Таким чином, аналоговий сигнал з деякою похибкою наближається цифровим ступінчастим сигналом. Як це робиться? Для цього з певним інтервалом у часі записуються значення аналогового сигналу, а точніше найближчі дискретні значення, які у нас є. Саме це робить аналогово-цифрове перетворення.

У разі зворотного перетворення все відбувається навпаки – цифровий сигнал посилається на виконавчий механізм, наприклад, на динаміки, які і перетворюють його, в даному випадку в звукову хвилю.

Питання для самоконтролю

1. Оберіть датчики, з яких може надходити аналогова інформація:

a) кнопка.

b) датчик, що вимірює рівень шуму.

c) датчик освітленості.

d) трипозиційний перемикач.

2. Визначте правильні твердження:

a) мікроконтролери працюють як з цифровими, так і з аналоговими сигналами;

b) аналогово-цифрове перетворення наближає аналоговий сигнал до цифрового;

c) на виконавчий механізм (наприклад, динамік) надсилається сигнал, модифікований за допомогою аналогово-цифрового перетворювача.

2 АПАРАТНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Електронні компоненти

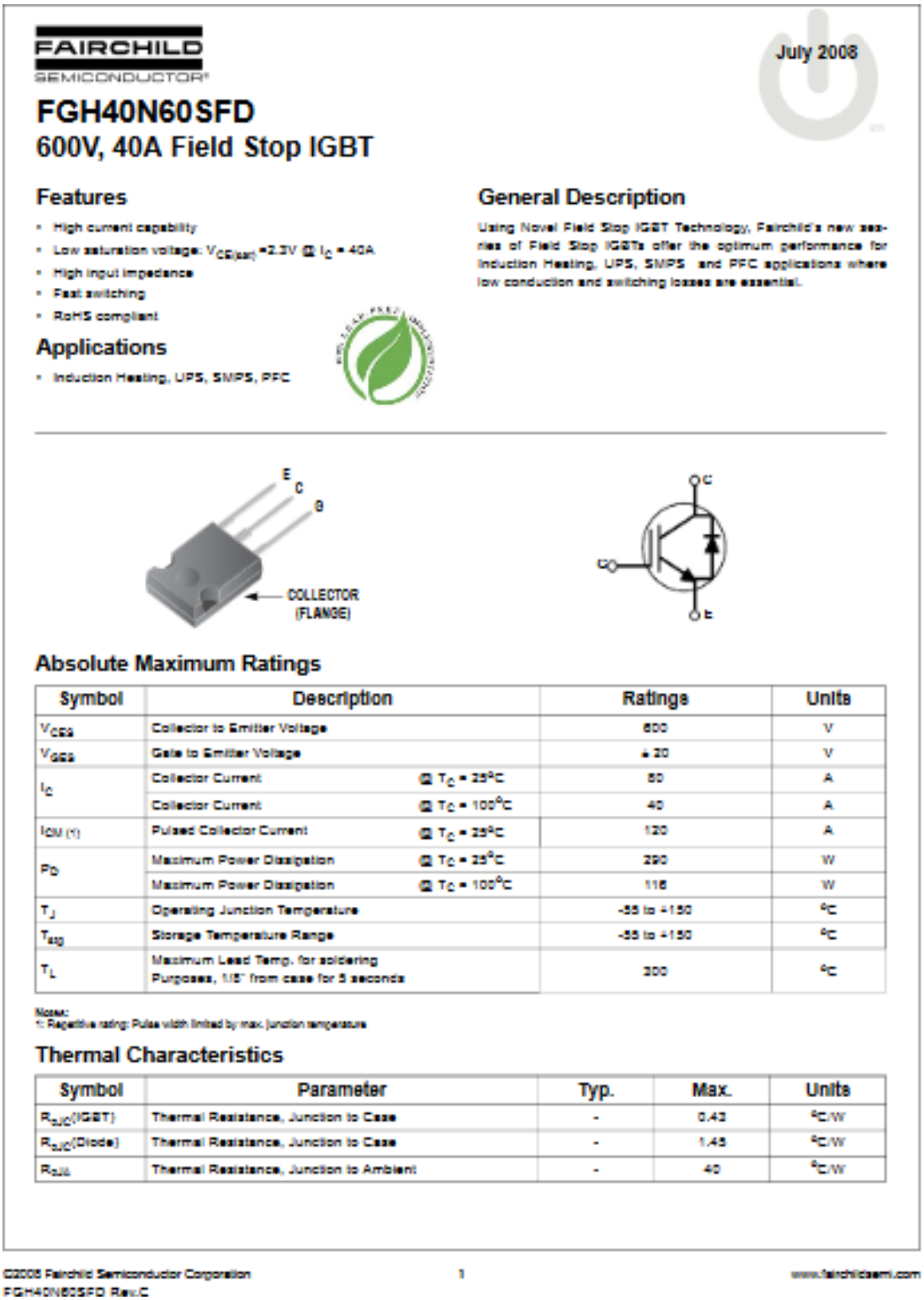
У цій темі ми поговоримо про документацію на електронні компоненти. Розглянемо, наприклад, ось цей транзистор:

Рисунок 2.1 – Транзистор

Якщо набрати в пошуковику назву цього транзистора, можна знайти його опис та технічну документацію. Перша сторінка опису може виглядати приблизно так, як зображено на рис. 2.2. У документації міститься дуже багато інформації, яку в перший час може бути складно зрозуміти.

До того ж багато компонентів виробляються за кордоном і опис є тільки англійською мовою. Іноді можна знайти його і українською мовою, але частіше за все він буде дуже стислим та міститиме тільки основні параметри. На рис. 2.3, наприклад, зображені пів сторінки документації на 14 різних транзисторів, а якщо подивитися в англійську версію, то там на кожен транзистор буде по 14 сторінок. Тому краще вчити технічну англійську мову, що, не дуже складно зробити, тому що кількість найчастіше використовуваних слів з області інформаційних технологій, як і в будь-якій іншій вузькоспеціалізованій сфері, порівняно невелика. Якщо ви тільки починаєте розбиратися з електронікою, то ви навряд чи зможете зрозуміти все, що

написано в документації. Але це не страшно, найчастіше розуміти все цілком навіть і не потрібно.



FAIRCHILD
SEMICONDUCTOR®

July 2008

FGH40N60SFD
600V, 40A Field Stop IGBT

Features

- High current capability
- Low saturation voltage: $V_{CE(sat)}$ =2.3V @ I_C = 40A
- High input impedance
- Fast switching
- RoHS compliant

Applications

- Induction Heating, UPS, SMPS, PFC

General Description

Using Novel Field Stop IGBT Technology, Fairchild's new series of Field Stop IGBTs offer the optimum performance for Induction Heating, UPS, SMPS and PFC applications where low conduction and switching losses are essential.

Absolute Maximum Ratings

Symbol	Description		Ratings	Units
V_{CES}	Collector to Emitter Voltage		600	V
V_{GES}	Gate to Emitter Voltage		± 20	V
I_C	Collector Current	@ T_C = 25°C	80	A
	Collector Current	@ T_C = 100°C	40	A
I_{CM} (1)	Pulsed Collector Current	@ T_C = 25°C	120	A
P_D	Maximum Power Dissipation	@ T_C = 25°C	290	W
	Maximum Power Dissipation	@ T_C = 100°C	116	W
T_J	Operating Junction Temperature		-55 to +150	°C
T_{stg}	Storage Temperature Range		-55 to +150	°C
T_L	Maximum Lead Temp. for soldering Purposes, 1/8" from case for 5 seconds		300	°C

Notes:
1: Repetitive rating: Pulse width limited by max. junction temperature

Thermal Characteristics

Symbol	Parameter	Typ.	Max.	Units
$R_{\theta JC}$(IGBT)	Thermal Resistance, Junction to Case	-	0.43	°C/W
$R_{\theta JC}$(Diode)	Thermal Resistance, Junction to Case	-	1.45	°C/W
$R_{\theta JA}$	Thermal Resistance, Junction to Ambient	-	40	°C/W

Рисунок 2.2 – Приклад сторінки з описом транзистора

Отже, яку інформацію ми можемо почерпнути з документації? Одна річ, яка завжди там є, це розміри корпусу. Оскільки різні компоненти вбудованих систем завжди потрібно з'єднувати разом, інженери повинні знати, який у них розмір, щоб можна було розробити відповідну схему плати.

IGBT ТРАНЗИСТОРИ

IGBT транзистори виконані по NTP технології, яка дозволила значно покращити робочі характеристики пристроїв.

IGBT транзистори випускаються в разнообразних корпусах. Технологія EmCon фірми Infeon дозволяє інтегрувати в одному корпусі транзистор і швидкодіючий зворотний діод.

Діапазон робочих струмів 2-30 А, робоча напруга до 1200 В.
Діапазон робочих температур: -55...+150°С.

СИСТЕМА ПОЗНАЧЕНЬ

	SG	P	30	N	120
	1	2	3		4

1. Тип транзистора: BUP - IGBT; SG - швидкодіючий IGBT; SK - швидкодіючий зі зворотним діодом
2. Тип корпуса: D - TO-252AA, I - TO262, B - TO-263AB, P(или 2) - TO-220AB, W - TO-247AC, 3 - TO-218AB
3. Ток коллектора, А
4. Напруга коллектор-емітер, В

Наименование	Частотный диапазон, кГц	Напряж. коллектор-эмиттер, В	Напряжение коллектор-эмиттер (откр.), В	Ток коллектора, А	Мощность рассеивания, Вт	Тип корпуса	Семейство
BUP212	10	1200	3.4	22	125	TO-220AB	IGBT
BUP213			3.3	32	200		
BUP313			3.4	32	200	TO-218AB	
BUP314			3.4	52	300		
BUP311D			3.4	20	125		IGBT с обратным диодом
BUP313D			3.4	32	200		
BUP314D			3.4	42	300		
SGP30N60	30	600	2.5	30	250	TO-220AB	быстродействующий IGBT
SGW30N60						TO-247AC	
SGP02N120		1200	3.1	2	62	TO-220AB	
SGP07N120		1200	3.1	7	125	TO-220AB	
SGW25N120		1200	3.1	25	313	TO-247AC	
SKP15N60		600	2.3	15	139	TO-220AB	быстродействующий IGBT с обратным диодом
SKW25N120		1200	3.7	25	313	TO-247AC	

ТИПИ КОРПУСІВ

TO-220AB

TO-247AC

TO-218AB

Рисунок 2.3 – Приклад документації по транзистору

Тому, в документації в обов'язковому порядку будуть вказані розміри, як, наприклад, у кресленні транзистора на рис. 2.4. Часто на кресленні розміри показані буквами, а поруч вказана таблиця, в якій написано, який букві який розмір відповідає. Як одиниці виміру зазвичай використовують і дюйми, і міліметри.

Габарити компонентів важливі, тому що, якщо, наприклад, використовувати макетну плату з кроком отворів, в які вставляються компоненти, в 2.54 мм, то з нею можна буде використовувати тільки ті компоненти, крок між ніжками яких дорівнює або кратний 2.54 мм, інакше їх просто не вийде вставити. Очевидно, що все це треба знати до покупки відповідних деталей.

По-друге, якщо потрібно спроектувати дуже компактний пристрій, то розміри вихідних елементів теж будуть дуже важливі, бо доведеться вибирати найменші компоненти, щоб вкластися у задані розміри.

У документації, звичайно, є й інша інформація. Там обов'язково є електричні і температурні параметри. Температурні параметри можуть бути важливі, якщо пристрою потрібно працювати взимку на вулиці або навпаки, він призначений для виробництва з високою температурою. Якщо пристрій не використовуватиметься в екстремальних умовах, то температурні характеристики не важливі.

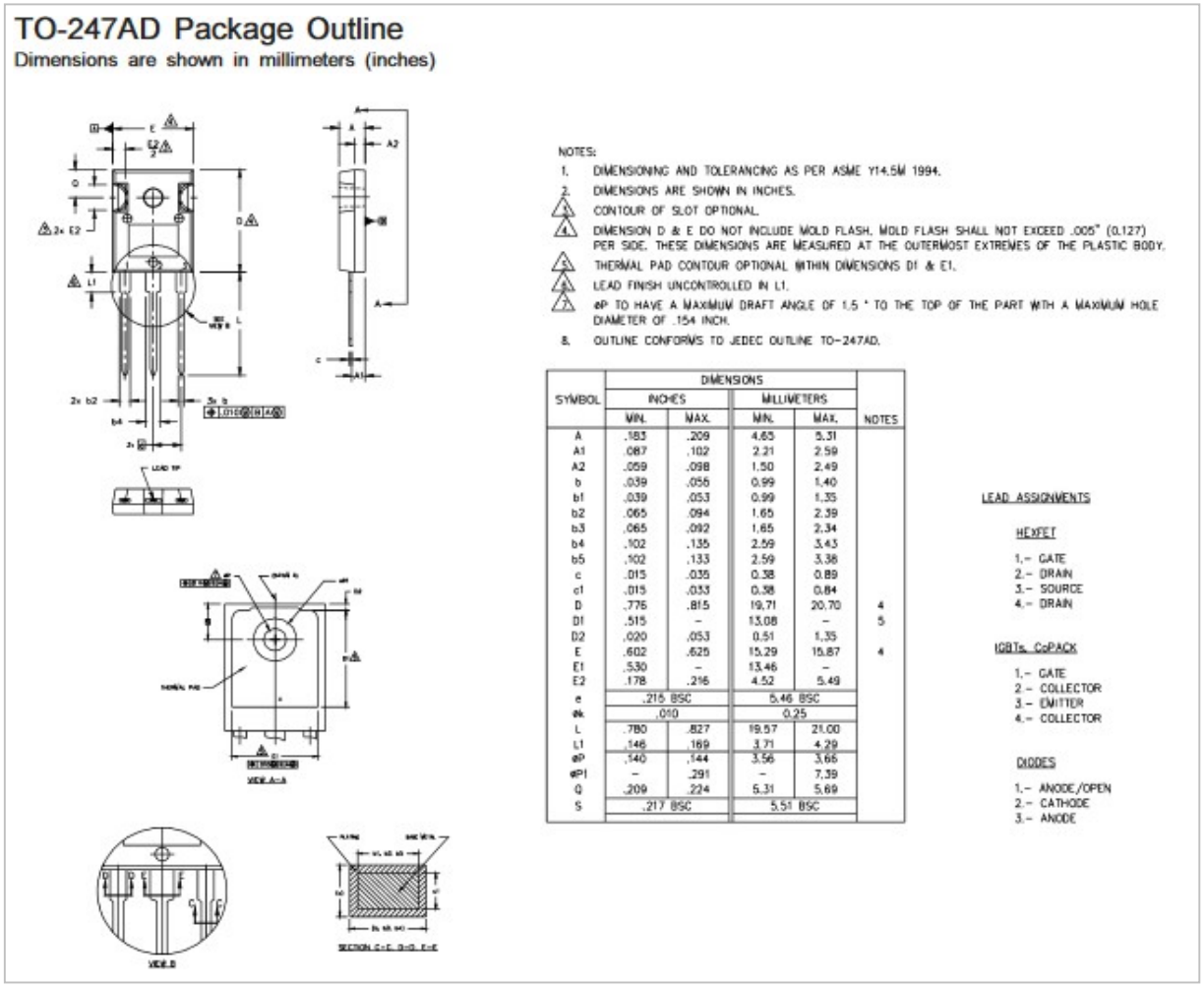


Рисунок 2.4 – Креслення транзистора

Електричні параметри включають такі важливі характеристики, як, наприклад, значення мінімальних і максимальних напружень і струмів, які

може витримати компонент, а також робочі діапазони цих величин. Це важливо, тому що потрібно бути впевненим, що всі компоненти сумісні по напрузі. Наприклад, мікроконтролер працює на напрузі 5 Вольт і, відповідно, видає всі сигнали з напругою 5 Вольт, а якийсь інший компонент, наприклад, IP-ядро, працює на рівні 3.3 Вольта. Такі компоненти неможна з'єднати безпосередньо, вони не зможуть один з одним працювати, і можуть взагалі «згоріти». В цьому випадку знадобляться спеціальні пристрої, які перево-дитимуть рівні напруги від 5 до 3.3 Вольт і назад. Або ж потрібно відразу підбирати компоненти, що працюють однаковою напругою.

Як уже було згадано, зазвичай в документації є два види електричних характеристик. Один, який буде зазначений на початку, це до максимальних значень (англійською «Absolute maximum ratings»). Ці характеристики точно не треба перевищувати, тому що компонент, швидше за все, відразу ж згорить. Наприклад, транзистор з рис. 2.1 не витримає, якщо подати більше 600 Вольт між однією з пар його виводів і більше 20 між іншими. І він не може пропускати більше 80 Ампер. 80 Ампер, це, звичайно, дуже багато, і мова йде про дуже потужний транзистор, але, якщо струм буде ще більше, то він швидко перегріється і згорить.

Другий вид електричних характеристик, зазначених у документації, це так звані робочі характеристики компонентів, тобто ті, з якими вони можуть працювати в штатному режимі.

Обсяги і складність технічної документації сильно залежать від самих компонентів, які можуть бути дуже різними. Деякі з них дуже прості, як наприклад, резистори, які можна описати всього декількома параметрами.

Але чим складніше компонент, тим більше у нього буде параметрів і тим складніше буде документація. Наприклад, мікросхеми можуть бути дуже складними, і документація на них може мати розмір у сотні сторінок. Зокрема мікроконтролери – це дуже складні мікросхеми, для опису яких потрібно не тільки вказати зовнішні параметри розмірів, напруги та інших характеристик,

але ще і повністю описати внутрішню логіку роботи мікросхеми, що іноді може займати більше місця, ніж всі зовнішні характеристики.

На щастя, в цьому навчальному посібнику нам не потрібно глибокого розуміння всіх деталей роботи мікроконтролера і тому нам не потрібно буде вивчати його об'ємну документацію. Ми використовуватимемо плату Arduino, а для неї є бібліотеки, що дозволяють простими засобами скористатися наявними можливостями встановлених на цих платах мікроконтролерів. Хоча, зазвичай, потрібно пам'ятати, що бібліотеки, приховуючи деталі реалізації, також приховують і всі потенційні можливості. Можуть бути такі завдання, які можна вирішити за допомогою мікроконтролера, але для яких немає бібліотеки, або бібліотека вирішує їх трохи не так, як потрібно. Буває, що в мікроконтролері якийсь компонент може працювати в різних режимах, а бібліотека дозволяє використовувати тільки один з них. Або бібліотека робить це повільніше, ніж, якщо керувати мікроконтролером безпосередньо. Тому, вам необхідно буде навчитися розуміти ці великі руководства, навчитися напряму працювати з мікроконтролером.

Питання для самоконтролю

1. Виберіть правильні твердження:

a) Зазвичай в документації є два види електричних характеристик: граничні і робочі;

b) у всіх компонентів крок виводів однаковий;

c) у документації міститься інформація про температурні і електричні параметри.

2. В яких випадках транзистор згорить:

a) подали 800 Вольт між однією з пар виводів;

b) температура навколишнього його середовища + 30^0C;

c) струм через транзистор склав 100 Ампер.

2.2 Мікроконтролери

У цьому розділі ми обговоримо характеристики мікроконтролерів. Коли розробляють вбудований пристрій, то одне з перших рішень, яке потрібно прийняти, це який мікроконтролер використовувати для управління системою.

Не вибравши мікроконтролер, не можна починати писати керуючу програму. Схему і друковану плату теж не можна починати робити, бо не зрозуміло навіть, яке живлення потрібно і який розмір контролера, і скільки у нього ніжок для підключення.

Для цього можна подивитися в документацію і спробувати зрозуміти, який мікроконтролер має ті характеристики і можливості, які потрібні. Як вже було сказано в попередньому розділі, нам у рамках цього навчального посібника такі рішення приймати не доведеться, тому що ми використо-вуватимемо плати Arduino, а на них вже є певний мікроконтролер, який ми і використовуватмемо. Але якщо ви розроблятимете справжні пристрої, то вам доведеться вирішувати і цю задачу, тому далі ми коротко обговоримо, які основні характеристики потрібно враховувати.

Одна з основних характеристик – це розрядність мікроконтролера. Що вона означає? Як і в більшості сучасних обчислювальних систем пам'ять, з якою працює мікроконтролер, логічно і схематично відділена від нього (хоч і може перебувати з ним на одному чіпі), і тому мікроконтролер не може безпосередньо виконувати операції з даними, які в ній записані (наприклад, складати). Для цього йому потрібно скопіювати ці дані в спеціальні осередки пам'яті, які як раз є безпосередньою частиною мікросхеми і називаються регістрами. Після того, як операція буде виконана, дані з регістрів, як правило, копіюються назад в пам'ять. Так зроблено тому, що реєстрова пам'ять, яка працює на тих самих швидкостях, що і мікроконтролер, є дуже дорогою, тому її не може бути багато. З іншого боку, основна пам'ять, винесена в свою власну

мікросхему, робиться за дещо іншою технологією, що дозволяє її істотно здешевити і, відповідно, зробити її більше, правда за все треба платити, тому подібна пам'ять, хоч її і багато, працює значно повільніше, ніж реєстрова – в десятки, а іноді і в сотні разів! Тому основний цикл роботи сучасного мікропроцесора виглядає так: скопіювати потрібні дані з основної пам'яті в регістри, виконати операцію над регістрами, скопіювати результат назад у зовнішню пам'ять або продовжити обчислення над значеннями,

Дані, які знаходяться в регістрах, як, власне, і всюди в комп'ютерах, подані у двійковому вигляді, тобто у вигляді нулів і одиниць, а регістри, в свою чергу, мають конкретний розмір, який визначає, скільки двійкових розрядів або скільки біт можна в нього записати. Власне, розрядність і є даний розмір регістра – кількість біт, які можна в нього записати.

Якщо взяти, наприклад, дуже простий мікроконтролер, то він може мати розрядність 8 біт (іноді кажуть, що він є восьмибітним). Це означає, що для обробки даних він використовує регістри розміром у 8 біт. У такий регістр можна записувати числа від 0 до 255, тому що 255 це найбільша кількість, яке можна записати в двійковому вигляді, використовуючи 8 розрядів. Так що такий процесор може працювати безпосередньо тільки з числами, значення яких не більше 255. Якщо потрібно працювати з числами більше, то навіть для найпростіших операцій доведеться писати програму. Наприклад, для складання ця програма складатиме спочатку молодші 8 біт числа, а потім, з огляду на можливе перенесення, старші 8 біт числа. Якщо ви не програмуєте на асемблері, а програмуєте, наприклад, мовою Сі, то вам, швидше за все, не доведеться самим писати таку програму, але все одно таке складання складатиметься з декількох дрібних операцій процесора і виконуватиметься повільніше, ніж коли він з самого початку мав би більшу розрядність і міг би вмістити числа цілком в свої регістри.

Інший поширений варіант розрядності – це 32 біта. 32-бітний процесор працює з числами, які можна уявити послідовністю з 32 нулів і одиниць і тому

він повинен мати такий же розмір регістрів. Він також повинен мати механізми, які дозволяють йому швидко передавати дані такого розміру в і з пам'яті. Такий процесор за інших рівних умов працюватиме швидше, якщо потрібно обробляти числа більше 255.

Отже, розрядність багато говорить про продуктивність процесора. Грубо кажучи, чим більше розрядність, тим вища продуктивність, тому що можна обробити більше інформації за один раз.

Розрядність це всього лише один з факторів, які визначають продуктивність мікроконтролерів і мікропроцесорів. Таких факторів дуже багато, що робить порівняння продуктивності складним завданням, на яку не завжди можна дати однозначну відповідь. Але є ще один важливий параметр, який впливає на продуктивність. Це тактова частота.

Тактова частота – це швидкість, з якою процесор виконує одну просту інструкцію. Правильніше сказати, це кількість простих інструкцій, які процесор виконує за одну секунду. Процесор з тактовою частотою 8 МГц виконує 8 мільйонів операцій в секунду, а якщо частота процесора 1 ГГц, то він виконує мільярд операцій у секунду. Тут принципово поняття простої операції. Це далеко не завжди відповідає одному оператору мови програмування. Це навіть не завжди відповідає одній операції мови асемблера. Навіть, якщо взяти найпростішу операцію – скопіювати значення з комірки пам'яті в регістр мікроконтролера, то цей процес може займати відразу кілька тактів його роботи. Це пов'язано з тим, що як вже було сказано вище, основна пам'ять працює істотно повільніше процесора і поки запит на інформацію дійде до неї і дані будуть передані назад, центральний процесор встигне зробити багато тактів. Або, наприклад, бувають досить складні інструкції, які виконуються не за один, а за кілька тактів – скажімо операції, що працюють відразу з масивом даних і т.д. Зазвичай, в сучасних архітектурах мікроконтролерів і мікропроцесорів використовується велика кількість хитрих і складних прийомів, що дозволяють підвищити продуктивність системи, незважаючи на подібний

перекіс у швидкостях роботи різних компонентів системи і складності виконуваних інструкцій. Тому можна вважати, що тактова частота безпосередньо впливає на продуктивність системи, і якщо потрібно, щоб інструкції виконувалися швидше, а значить, програма виконувалася швидше, то потрібна і більш висока тактова частота.

Отже, з точки зору продуктивності, з одного боку є прості восьмибітові мікроконтролери, що працюють на частотах, наприклад, 4, 8, 16 МГц. З іншого боку, є 32-бітні мікроконтролери, які працюють на частотах 500 МГц і більше, 1 ГГц. Це дуже велика різниця в продуктивності, але і, очевидно, в ціні і, що не менш важливо, в енергоспоживанні. Для кожного завдання потрібно вибирати відповідний контролер.

Входи і виходи мікросхеми теж дуже важливі, і вони часто є вузьким місцем – ресурсом, якого не вистачає. Може, ви чули, що є такий закон Мура. Цей закон стверджує, що з кожним роком мікросхеми стають все складніше і швидше. Однак, хоча продуктивність швидко зростає, число контактів, через які мікроконтролер спілкується із зовнішнім світом, залишається приблизно однаковим. Контакти (їх ще називають портами введення-виведення), є цінним ресурсом. Грубо кажучи, якщо мікроконтролер має бути пов'язаний ще з п'ятьма пристроями і кожне з них вимагає певну кількість контактів для зв'язку, то потрібно скласти їх число для всіх 5 пристроїв, і отримаємо мінімальне число контактів, яке має бути у мікросхеми. І це може відразу відсіяти багато варіантів. Таким чином, число входів і виходів визначає, наскільки зручно з'єднувати мікроконтролер з іншими пристроями і скільки таких зв'язків може бути. Це дуже важливий фактор.

Підтримка комунікаційних протоколів – інший важливий момент, який визначає повноваження зв'язку мікроконтролера з іншими компонентами. Для чого потрібні ці протоколи? Як вже було неодноразово сказано, мікроконтролер має обмінюватися даними з іншими компонентами. Вони передають один одному багато даних, і роблять це суворо певним чином. Найчастіше

недостатньо просто передати 0 або 1. Якщо потрібно обмінюватися великою кількістю даних (а крім самих даних є ще і керуючі, синхронизуючі, контрольні та інші сигнали), то зазвичай встановлюється порядок, у якому здійснюються комунікації. Цей порядок, або набір правил, і називається протоколом.

Існує багато протоколів: UART, I^2C, SPI, USB та ін. Ми розглянемо частину з них пізніше. Це загальновідомі протоколи, використовувані мікросхемами й іншими компонентами, причому очевидно, один пристрій не обов'язково підтримує їх всі. Наприклад, якась мікросхема для стиснення звуку може працювати з використанням тільки I^2C, і це потрібно знати і враховувати, тому що тоді і мікроконтролер має підтримувати I^2C, інакше він не зможе зв'язатися з цією мікросхемою.

Так що підтримка стандартних протоколів – це теж важливий параметр. Важливо не тільки те, скільки контактів для спілкування є, а й які протоколи реалізовані у всіх компонентах системи.

Ще одним важливим компонентом мікроконтролера є таймер. Він може використовуватися для того, щоб вимірювати інтервали часу або створювати затримку на певний час, для того, щоб генерувати спеціальні сигнали для управління приводами, але про це ми поговоримо пізніше. Один з варіантів використання таймера – оцифровка звуку. Наприклад, потрібно записувати рівні звуку (напруга з мікрофона) з частотою 4 КГц. Для цього необхідно, щоб хтось запускав вимір у фіксовані інтервали часу 4 тисячі разів у секунду. Цим якраз і займається таймер.

Цифро-аналогові і аналого-цифрові перетворювачі, про які ми говорили раніше в цьому розділі, також можуть бути вбудовані в мікроконтролер.

Ще один важливий аспект, який слід враховувати при виборі мікроконтролера – це режими низького споживання енергії. Вбудовані системи часто працюють від батарейок, і важливо, щоб пристрій споживав якомога менше енергії і працював якомога довше. Щоб скоротити споживання живлення використовують мікроконтролери з режимами зниженого споживання, в яких

підсистеми, які не використовуються, відключаються в ті моменти, коли вони не потрібні. Такі системи управління живленням можуть бути дуже складними і підтримувати кілька режимів (наприклад, звичайний, сон, глибокий сон і т. д.). Як правило, ці та інші основні характеристики описані на початку технічної документації, так що можна швидко подивитися і зрозуміти, є вони в цьому мікроконтролері чи ні. Для того, щоб зрозуміти, як саме вони працюють і як їх використовувати, потрібно вже більш детально вивчити документацію.

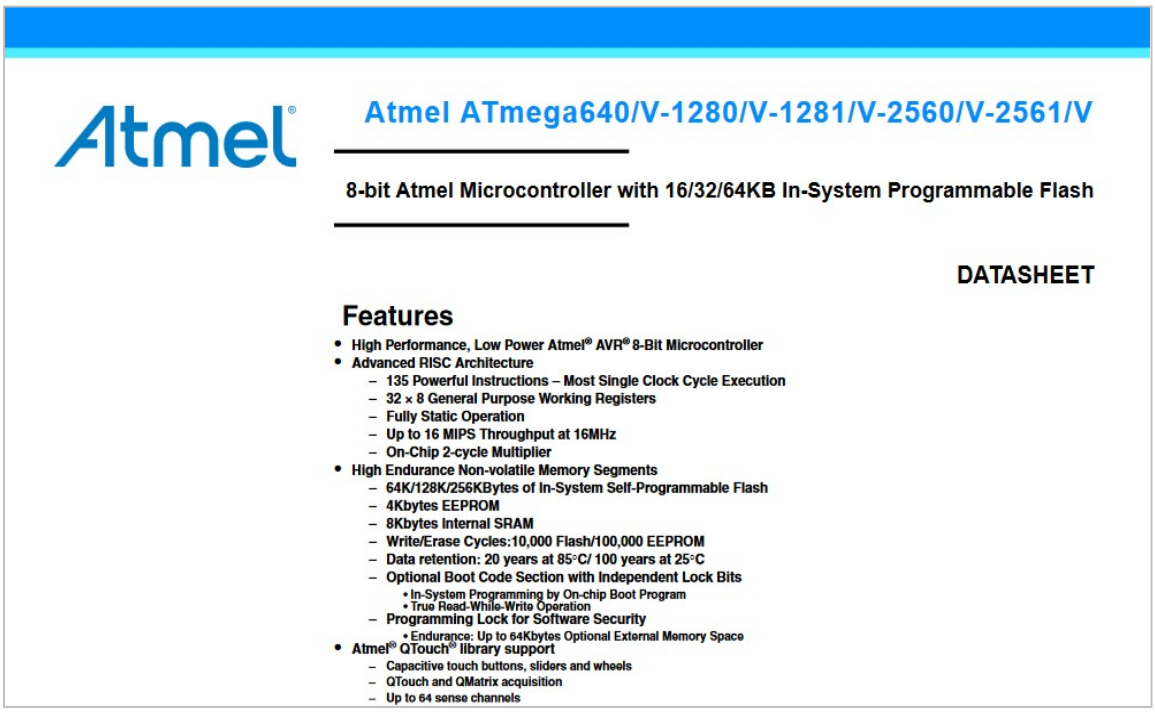
Можливо, один з найважливіших ресурсів будь-якого мікроконтролера – це пам'ять, щоб зберігати програми і дані. Розглянемо, які види пам'яті трапляються. На рис. 2.5 ми бачимо початок документації на один з мікроконтролерів ATmega фірми Atmel, які можуть використовуватися на платах Arduino.

Тут йдеться про те, що це восьмибітний мікроконтролер і що він може працювати на частоті до 16 МГц. Процесор вміє виконувати 135 різних інструкцій, велика частина з яких виконуються за один такт. Більшість інших параметрів описують види пам'яті, які в ньому є.

Чому буває так багато видів пам'яті? Одна з причин цього в необхідності балансу між ціною і обсягом. Є різні способи зробити пам'ять, але завжди виходить, що чим вона швидше, тим дорожче.

Найшвидша пам'ять – реєстрова. Процесор може за один такт прочитати і записати дані з декількох регістрів, наприклад, з двох або трьох. Наприклад, він може прочитати дані з двох регістрів, скласти їх і записати в третій регістр, і він це встигає все це зробити за один такт.

Регістри швидкі, але вони дорогі і тому їх мало. Наприклад, у цьому мікроконтролері ATMega 32 восьмибітних регістра загального призначення. Це ті регістри, де можна зберігати значення змінних, з якими працює програма. Є ще спеціальні регістри, що використовуються для того, щоб зберігати режими роботи процесора, адреса поточної виконуваної інструкції і т.д., але їх не можна використовувати безпосередньо для обробки даних.



Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V

8-bit Atmel Microcontroller with 16/32/64KB In-System Programmable Flash

DATASHEET

Features

- High Performance, Low Power Atmel® AVR® 8-Bit Microcontroller
- Advanced RISC Architecture
  - 135 Powerful Instructions – Most Single Clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16MHz
  - On-Chip 2-cycle Multiplier
- High Endurance Non-volatile Memory Segments
  - 64K/128K/256KBytes of In-System Self-Programmable Flash
  - 4Kbytes EEPROM
  - 8Kbytes Internal SRAM
  - Write/Erase Cycles:10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/ 100 years at 25°C
  - Optional Boot Code Section with Independent Lock Bits
    - In-System Programming by On-chip Boot Program
    - True Read-While-Write Operation
  - Programming Lock for Software Security
    - Endurance: Up to 64Kbytes Optional External Memory Space
- Atmel® QTouch® library support
  - Capacitive touch buttons, sliders and wheels
  - QTouch and QMatrix acquisition
  - Up to 64 sense channels

Рисунок 2.5 – Сторінка з документацією на контролер Arduino

Очевидно, що 32-х регістрів мало, щоб зберігати дані і програму. Їх вистачає лише для того, щоб організувати поточний процес обчислень, записуючи в них проміжні результати. Для зберігання основних даних використовується оперативна пам'ять (або її ще називають основною). У процесорі на рис. 2.5 вона називається SRAM. Її набагато більше – тут її цілих 8 кілобайт. Вона істотно дешевше, але і повільніше. Як вже говорилося раніше, сучасним процесорам, які використовуються в настільних комп'ютерах, потрібні сотні тактів, щоб прочитати значення з пам'яті. У повільних мікроконтролерах різниця не така велика, але все одно потрібно більше часу, щоб прочитати дані з SRAM, ніж з регістра.

Інша проблема з пам'яттю це енергонезалежність. Регістри і SRAM втрачають свій вміст, якщо вимкнути живлення. Очевидно, що в мікро-контролері має постійно зберігатися програма, яку він виконує, інакше пропадає будь-який сенс його використання, оскільки після включення мікропроцесор не знатиме, що йому робити. У звичайних комп'ютерах для вирішення цієї проблеми використовують так звану зовнішню пам'ять – тверді

диски. У мікроконтролерах з цією метою використовують флеш-пам'ять, яка є такою ж самою, як і в USB-накопичувачах. Правда, у неї є свої недоліки. По-перше, вона має обмежене число циклів перезапису, і може зіпсуватися, якщо в неї часто що-небудь записувати. По-друге, дані в неї записуються тільки великими фрагментами (це обумовлено її внутрішньою архітектурою), тому, вона підходить для зберігання програми, але не дуже добре підходить, якщо потрібно зберігати якісь дані, які часто змінюються під час роботи пристрою і мають зберігатися при виключенні живлення.

Тому є ще четвертий вид пам'яті, який називається EEPROM. Це теж незалежна пам'ять, але вона зроблена за іншою технологією. Її можна перезаписувати окремими байтами і у неї набагато більше робочих циклів запису, так що її можна міняти часто. Зокрема, з інструкції на рис. 2.5 ми бачимо, що флеш-пам'ять допускає 10000 циклів перезапису, а EEPROM – до 100000 циклів. Але вона дорожче і тому її менше, ніж флеш пам'яті.

Спочатку може бути складно зрозуміти, скільки буде потрібно пам'яті в майбутньому для конкретного пристрою. Хоча деякі речі можна оцінити. Нехай, наприклад, наша вбудована система оброблятиме звук тривалістю в 1 секунду і з частотою 4 КГц, то це означає, що потрібно 4000 байт, якщо для зберігання кожного відліку використовується 1 байт. Може бути потрібно два таких буфера – один обробляється, а в інший в цей же час йде запис нового звуку. Крім цього, буде потрібно пам'ять під різні додаткові змінні. Оцінити код програми складніше. Якщо писати програму мовою Сі, то скільки вона займе, заздалегідь незрозуміло. Якщо програма проста, то вона займе, наприклад, кілька кілобайт, а більш складна і велика програма може зайняти після компіляції суттєво більше. Це стане зрозуміло, коли вже більша частина програми написана.

Питання для самоконтролю

1. Виберіть правильні твердження:

a) при виборі мікроконтролера головну роль відіграє його тактова частота, тоді як розрядність – незначна деталь;

b) розрядність сильно впливає на продуктивність процесора;

c) продуктивність мікроконтролерів зростає повільно, а число контактів для спілкування з зовнішнім світом зростає швидко;

d) протоколи використовуються при обміні даними між мікроконтролером і іншими компонентами;

e) таймер, цифро-аналогові і аналого-цифрові перетворення, пам'ять – все це може входити в мікроконтролер.

2. Які з наступних видів пам'яті є незалежними:

a) оперативна пам'ять (SRAM);

b) флеш-пам'ять;

c) регістри;

d) EEPROM.

2.3 Програмування мікроконтролерів

У цьому розділі ми поговоримо про програмування мікроконтролерів. Точніше, як обробляється програма перед тим, як її виконає мікроконтролер.

Ключовим моментом тут є те, що мікроконтролер, як ви, напевно, здогадуєтеся, насправді не виконує буквально той код, який пишуть програмісти. Спочатку він повинен бути оброблений транслятором, а потім вже він виконується мікроконтролером.

Навіщо потрібен цей додатковий етап? Він потрібен тому, що мікроконтролер, як і будь-який інший процесор, не розуміє мов Сі, або С ++,

Java, Паскаль, Пітон і будь-які інші мови, якими зазвичай пишуться програми. Він розуміє і може виконувати тільки свою власну машинну мову.

Візьмемо процесор Intel. Він розуміє машинну мову X86. Інші процесори, наприклад, мікроконтролери Atmel, які стоять на платах Arduino, розуміють свою машинну мову. Різні сімейства процесорів використовують свою власну мову. Що вона собою являє? Це набір простих інструкцій, закодованих у двоїчному форматі за допомогою нулів і одиниць. Тому, якщо ви подивитеся безпосередньо на машинний код, то все, що ви побачите – це нулі і одиниці. Це, звичайно, рідко хто робить, але якщо і робить, то зазвичай код записується в шістнадцятковому форматі – так він займає менше місця і трохи більше зручний для сприйняття. Але в будь-якому випадку машинна мова не виглядає читаною для людини, хоча це те, що машина насправді виконує.

Для того, щоб зробити програми більш легкими для читання їх переводять мовою асемблера, які складаються з коротких і зрозумілих мнемонічних команд, на кшталт ADD, SUB, MUL, MOV, JMP і т.д. Ніякої виразної потужності асемблер не додає, оскільки по суті, це однозначне відображення з машинних кодів в слова. Він існує тільки заради зручності читання машинних кодів людиною. Наприклад, замість 10110110 10111001 01100110 01010101 (або в шістнадцятковому форматі – B6 B9 66 55) буде написано ADD R1, R2, R3, що означає скласти значення в регістрах R2 і R3 і помістити результат у R1.

Хоч мову асемблера і придумали для полегшення читання і запису програм, але все-таки це дуже низький рівень. Асемблерна мова називається мовою з мітками, бо навпроти будь-якої інструкції можна поставити мітку, а потім перейти до неї за допомогою команд умовного і безумовного переходу (на кшталт goto у високорівневих мовах). Причому переходи по мітках – це єдиний спосіб організувати розгалуження і цикли, оскільки в подібних мовах немає ні циклів for або while, ні можливості об'єднати інструкції в блоки коду як в структурованих мовах програмування. Можна писати код мовою

асемблера, але це важко, і ми не робитимемо це в даному посібнику. Іноді це необхідно, щоб досягти максимальної продуктивності. Ми працюватимемо з мовами вищого рівня: Сі, C++, Python, Lua.

Таким чином, програма, написана мовою високого рівня, має бути переведена на машинну мову, перш ніж її зможе виконати мікроконтролер. Існує два способи, якими це можна зробити. Якщо використовується компільована мова, то програма цією мовою відразу цілком перекладається в машинний код. Після такого перекладу ви отримаєте те, що називається виконуваним файлом, що містить саме той код, який виконується щоразу при запуску програми. Приклади компільованих мов – Сі, С ++, Паскаль й інші. Коли ми використовуватимемо плати Arduino, ми працюватимемо із спрощеною мовою С ++.

З іншого боку, крім компіляції, є також інтерпретація. В інтерпретованому середовищі інструкції мовою високого рівня, перетворюються в машинний код щоразу під час виконання за допомогою інтерпретатора. Прикладами таких мов є Пітон, Java, Lua та ін. Недолік інтепретованих програм у тому, що вони працюють суттєво повільніше, адже процесор виконує не їх, а код інтерпретатора, який, в свою чергу, виконує вашу програму! Скомпільована ж програма, як вже було сказано раніше, займає центральний процесор цілком і повністю без будь-яких посередників. Але інтерпретація має і переваги, оскільки вона звільняє програміста від певних завдань. Так, наприклад, якщо в Сі ви хочете використовувати динамічний масив, вам потрібно викликати спеціальну функцію для отримання пам'яті, а потім не забути звільнити цю пам'ять. Це дуже складно відстежити і зробити коректно у великих програмах, тому часто виникають помилки. З іншого боку, в інтерпретованих мовах ви просто використовуєте готовий динамічний масив, а інтерпретатор вирішує всі завдання з керуванням пам'яттю. Тому яку мову використовувати, як завжди, залежить від розв'язуваної задачі.

Питання для самоконтролю

1. Розмістіть в порядку від низького до високого рівня такі способи написання коду:

a) код на Ci, Java, Python;

b) машинний код;

c) мова асемблера.

2. Виберіть правильні твердження:

a) Python є компільованою мовою;

b) інструкції компілювальної мови цілком переводяться в машинний код тільки один раз;

c) приклад компілювальної мови - C ++;

d) код, написаний компільованою мовою, виконується повільніше, ніж код, написаний інтерпретуючою мовою.

3 МОДЕЛЮВАННЯ ЕЛЕКТРОННИХ СХЕМ І ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ НА ПРИКЛАДІ ARDUINO

3.1 Платформа Arduino. Плата Arduino Uno

Arduino – це електронний конструктор і зручна платформа швидкої розробки електронних пристроїв для новачків і професіоналів. Платформа користується величезною популярністю в усьому світі завдяки зручності і простоті мови програмування, а також відкритої архітектури і програмного коду. Пристрій програмується через USB без використання програматорів.

Мікроконтролер на платі програмується за допомогою мови Arduino (заснований мовами Сі та С++) і свого середовища розробки, яка доступна для безкоштовного завантаження. Проекти пристроїв, засновані на Arduino, можуть працювати самостійно, або ж взаємодіяти з програмним забезпеченням на комп'ютері. Плати можуть бути зібрані користувачем самостійно або куплені вже в зборі, причому вихідні креслення схем є загальнодоступними, користувачі можуть застосовувати їх на свій розсуд.

Сьогодні випущено кілька плат з серії Arduino, деякі з яких зображені на рис. 3.1. У даному навчальному посібнику ми використовуватимемо Arduino Uno, але розглянуті приклади мають працювати на будь-якій з них, з урахуванням деяких особливостей використання виводів у різних платах, про які буде розказано пізніше.

Більшість плат Arduino побудовані на восьмибітних мікроконтролерах фірми Atmel. Процесори працюють на тактовій частоті 16 МГц. Плати містять всі необхідні для роботи мікроконтролера компоненти, включаючи схеми живлення і кварцовий резонатор. Робоча напруга в більшості випадків 5 Вольт. Програмування мікроконтролера можна здійснювати, просто підключивши його до комп'ютера через порт USB.

На платі є розніми, що дозволяють підключати зовнішні схеми до більшості

виходів мікроконтролера. Ці розніми дозволяють використовувати цифрові і аналогові входи та виходи, ШІМ генератори, різні цифрові інтерфейси. Для Arduino випускається безліч плат розширення, які дозволяють використовувати плату як основу для керування роботами, підключатися до комп'ютерних мереж через Ethernet або Wi-Fi, перетворювати її в цифровий фотоапарат і т. д..

Оскільки середовище Arduino дуже популярно, то багато розробників мікроконтролерів роблять свої плати, сумісними з нею. Це дозволить застосувати навички, отримані під час роботи з Arduino, з іншими, в тому числі набагато більш потужними мікроконтролерами.

Плата Arduino Uno, зображена на рис. 3.2, побудована на базі мікроконтролера ATmega328. Платформа має 14 цифрових входів/виходів (6 з яких можуть використовуватися як виходи ШІМ), 6 аналогових входів, кварцовий генератор 16 МГц, рознім USB, рознім живлення, рознім для вибору програм (ICSP) і кнопку перезавантаження. Для того щоб вона запрацювала, її необхідно підключити до комп'ютера за допомогою кабелю USB, або подати живлення за допомогою адаптера AC/DC або батареї.

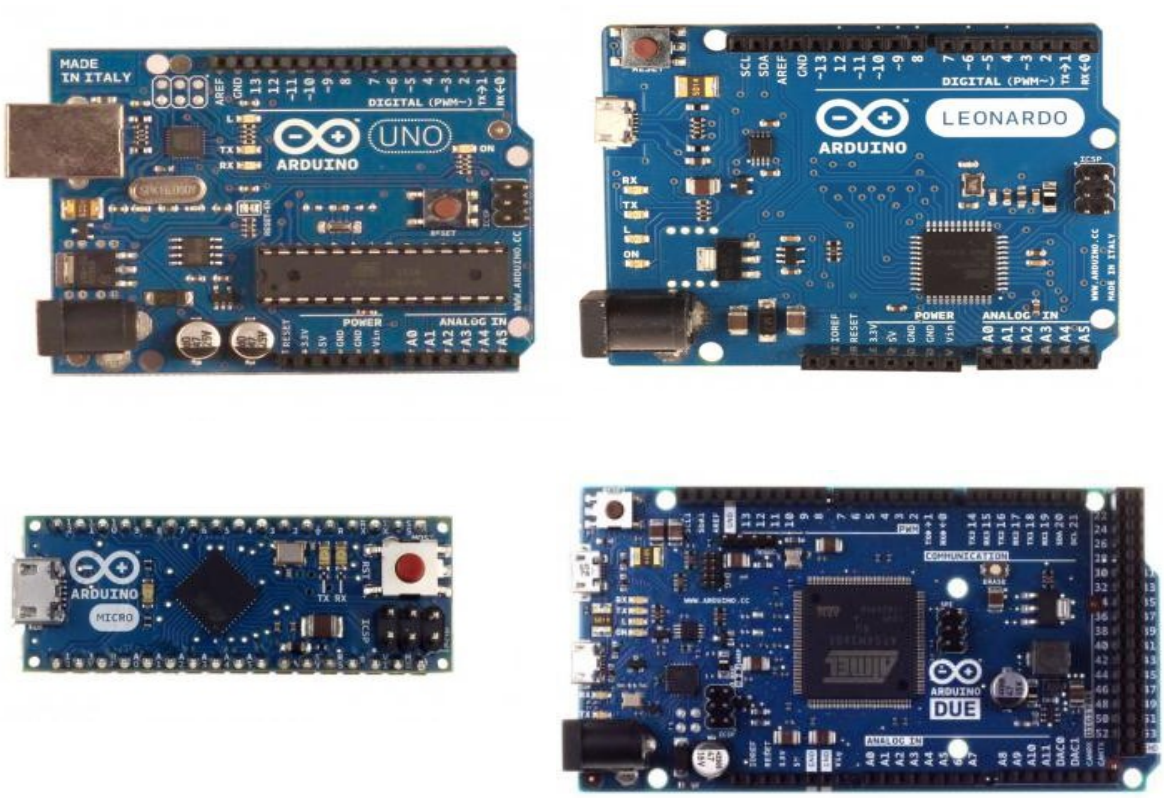


Рисунок 3.1 – Приклади плат Arduino

Рисунок. 3.2 – Плата Arduino Uno

3.2 Середовище розробки Arduino

У цьому розділі ми ознайомимося з середовищем розробки Arduino IDE (Integrated Development Environment – інтегроване середовище розробки). Це середовище потрібно використовувати, якщо у вас є реальна плата Arduino.

Рядок меню редактора (рис. 3.3) включає такі основні елементи: файл, правка, скетч, сервіс і довідка. Розглянемо докладніше кожен з них.

У пункті Файл можна знайти команди, що відповідають за створення нової програми, завантаження з диска вже існуючої, збереження її змін, а також команди для завантаження програми на мікроконтролер:

- «Створити» – створити нову програму (в даному середовищі програми називаються скетчами);
- «Відкрити» – відкрити існуючу програму;
- «Папка зі скетчами» – відкрити програму із заданої папки;
- «Приклади» – відкрити приклад програми;

– «Закрити» – закрити поточне вікно;
– «Зберегти» – зберегти зміни;
– «Зберегти як» – зберегти програму в новому файлі;
– «Завантажити» – завантажити програму в Arduino;
– «Завантажити за допомогою програматора» – завантажити програму за допомогою програматора;
– «Налаштування друку» – налаштування принтера;
– «Друк» – висновок на друк коду програми;
– «Налаштування» – налаштування редактора;
– «Вихід» – вихід з Arduino IDE.

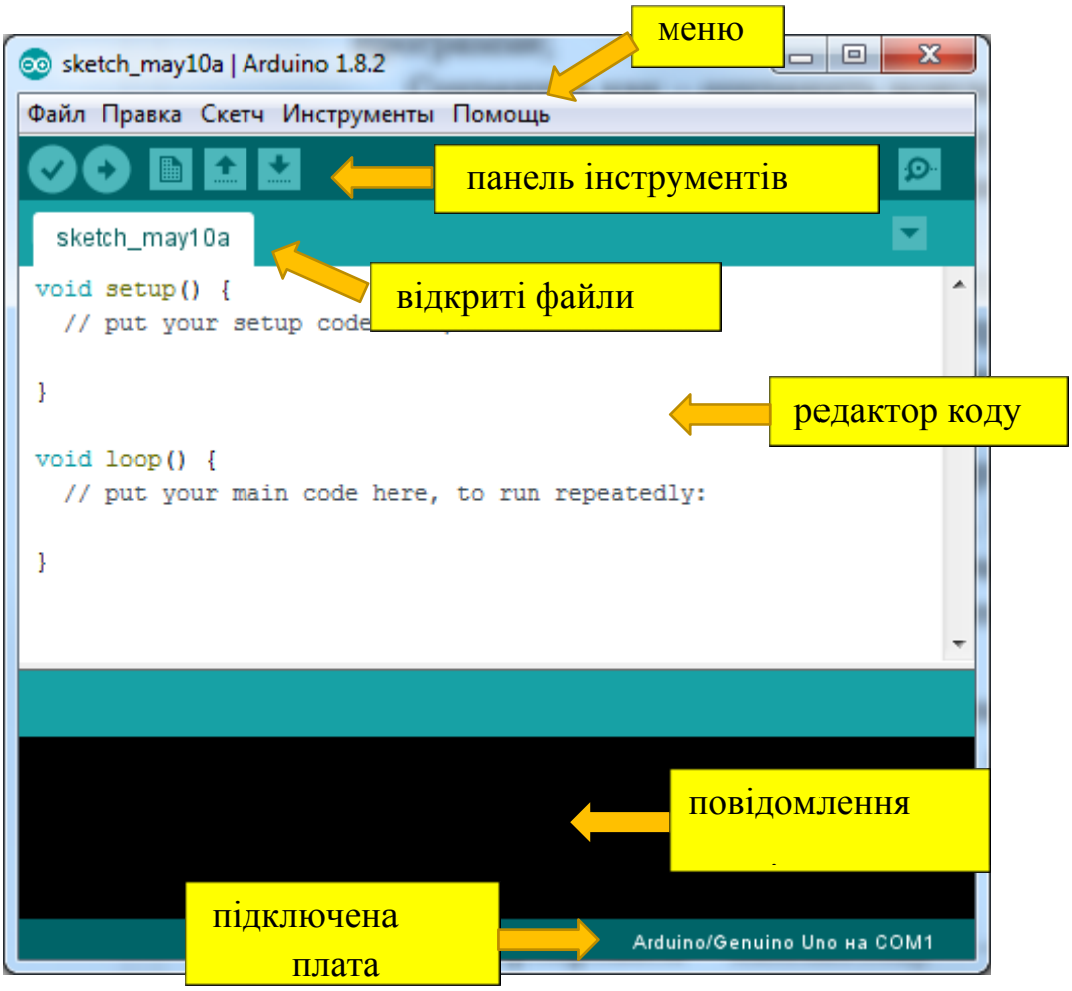


Рисунок 3.3 – Області вікна середовища розробки Arduino

Пункт меню «Правка» містить команди, пов'язані з редагуванням тексту програми, що включає в себе копіювання, вставку, налаштування відступів і пошук.

У розділі «Скетч» розміщуються команди для контролю за процесом компіляції програми:

- «Перевірити/Компілювати» – компілювати програму;
- «Показати папку скетчів» – відкрити системну папку з програмами;
- «Додати файл» – додати до проекту файл з даними або програмою;
- «Імпортувати бібліотеку» – підключити до програми бібліотеку зі списку встановлених;
- Пункт меню «Сервіс» включає в себе допоміжні функції для роботи з самим мікроконтролером;
- «Автоформатування» – автоматична розстановка відступів, переносів рядків тощо;
- «Архівувати скетч» – архівація папки з програмою, і збереження архіву у вказане місце;
- «Монітор порту» – відкрити вікно для обміну даними з мікроконтролером;
- «Плата» – вибір поточної плати (в нашому випадку Arduino Uno);
- «Послідовний порт» – вибір порту, до якого підключено пристрій;
- «Програматор» – вибір програм (нами в навчальному посібнику не використовується);
- «Записати завантажувач» – запис програми-завантажувача в мікроконтролер (також нами не використовується).

Нарешті, меню Довідка містить докладний опис всіх функцій самого редактора Arduino IDE, а також всілякі команди і прийоми роботи з платформою.

На панель інструментів винесені у вигляді кнопок найчастіше використовувані функції середовища. Їхнє призначення описано в таблиці 3.1.

Безпосередньо текст програми створюється і редагується у вікні редактора коду. Це вікно є типовим текстовим редактором з підсвічуванням синтаксису програми.

У нижній частині вікна Arduino IDE є область, яка служить для виведення повідомлень і повідомлень про помилки, що виникають в процесі компіляції програми або під час завантаження програми в мікроконтролер.

Таблиця 3.1 – Кнопки на панелі інструментів

Кнопка	**Опис**
	Перевірити / Компілювати програму
	Завантажити програму в Arduino
	Створити нову програму
	Відкрити існуючу програму
	Зберегти програму
	Монітор послідовного порту

3.3 Онлайн-емулятор Arduino

Для того, щоб виконувати вправи даного навчального посібника, не обов'язково купувати плату (хоча це і бажано, якщо у вас є така можливість). Ми використовуватимемо веб додаток Autodesk CIRCUITS, який дозволяє моделювати Arduino і різні електронні компоненти прямо у веб-браузері без необхідності завантаження і установки чогось собі на комп'ютер. Дана система абсолютно безкоштовна і все, що вам потрібно для роботи з нею – вихід в Інтернет.

Заходимо на сайт http://circuits.io і натискаємо у верхньому правому куті кнопку «Sign up for free». Можна зареєструватися безпосередньо в системі або

використовувати можливість входу за допомогою вже наявних облікових записів від Google, Facebook і інших сервісів.

Щоб створити нову схему, натискаємо кнопку «New Electronics Lab». Ви побачите редактор електронних моделей, в якому є одна порожня макетна плата.

Ця плата складається з чотирьох областей (див. рис. 3.4). Зверху і знизу йдуть по два ряди отворів, призначених для створення шин живлення. Всі отвори одного ряду пов'язані один з одним. Передбачається, що через один з отворів до цих шин підводять напругу живлення (позитивний або негативний контакт), а потім, через інші отвори до живлення можна буде підключати компоненти нашої системи.

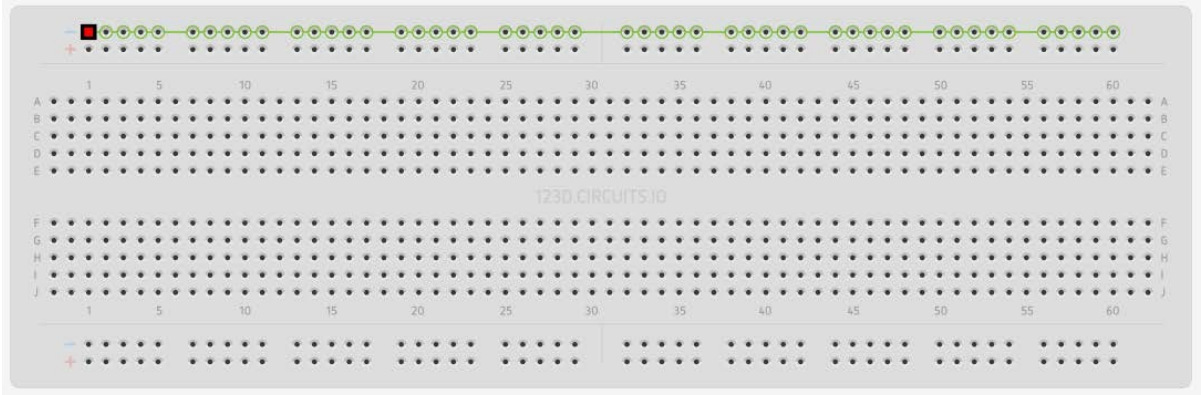


Рисунок 3.4 – Приклад плати

Отвори в кожній з двох центральних областей, навпаки, пов'язані вертикально: вони з'єднані в рядках ABCDE і FGHIJ. Ці області використовуються для установки і підключення електронних пристроїв.

Щоб створити схему, потрібно в першу чергу додати компоненти. Для цього натискаємо кнопку «+ Components» у верхньому правому кутку екрана (рис. 3.5).

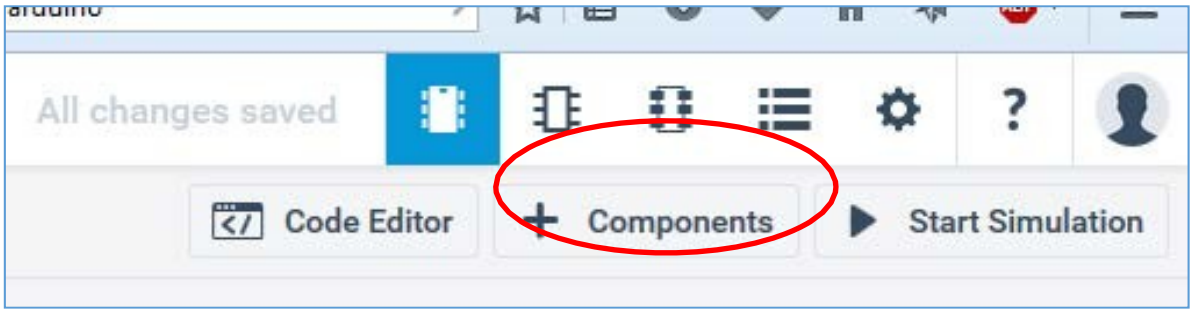


Рисунок 3.5 – Кнопка «+ Components»

Знизу з'явиться панель, в якій можна вибирати різні компоненти, підтримувані емулятором.

Давайте виберемо світлодіод і батарею і розмістимо їх на нашій макетній платі. Якщо виводи (контакти) компонентів співпадуть з отворами плати, то емулятор вважатиме, що вони туди вставлені і є контакт.

Якщо виділити мишкою компонент, то справа зверху з'явиться вікно з його властивостями. З його допомогою можна, наприклад, поставити колір світлодіода або поміняти опір резистора (рис. 3.6).

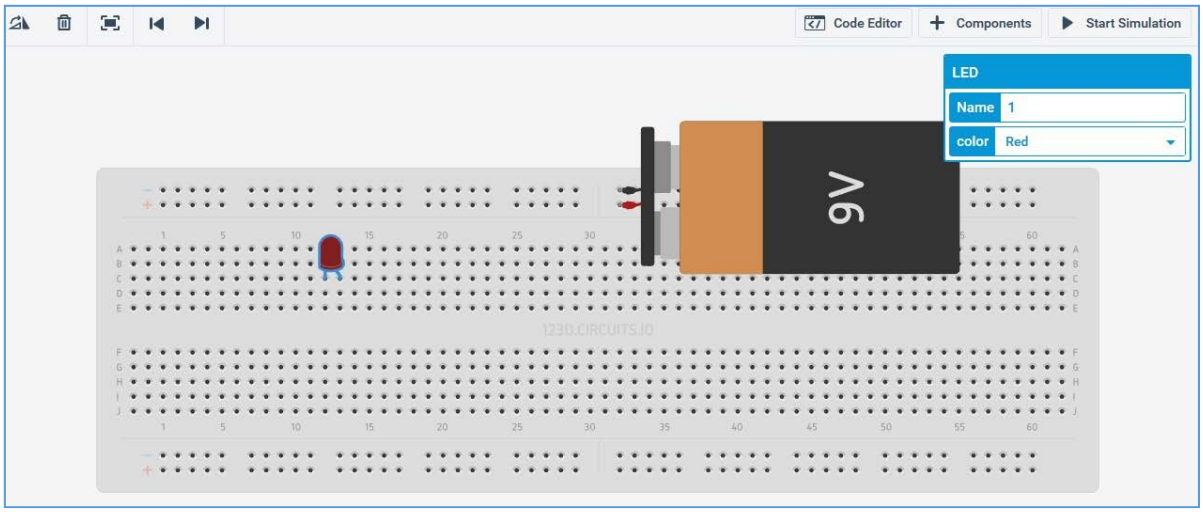


Рисунок 3.6 – Приклад вікна з властивостями

Розташовані в лівому верхньому кутку кнопки дозволяють керувати редактором моделі. Їхній опис наведено в таблиці 3.2. Щоб з'єднати компоненти проводами, достатньо клацнути мишкою на контактах, які треба

з'єднати. З'явиться зображення дрота. Якщо клацнути по виділеному дроту, то на ньому з'явиться точка, за допомогою якої його можна зігнути.

Таблиця 3.2 – Кнопки на панелі інструментів circuits.io

Кнопка	Опис
	Повертає компонент
	Вилучає компонент (можна також натиснути на клавішу <Delete> на клавіатурі)
	Зменшує масштаб так, щоб усі компоненти помістилися на екрані (збільшити масштаб можна)
	Дозволяє скасувати попередню дію і повернути її після скасування

Давайте підключимо світлодіод проводами до шини живлення. Щоб подивитися, як працює схема, натисніть на кнопку «Start Simulation» у правому верхньому кутку екрана (рис. 3.7).

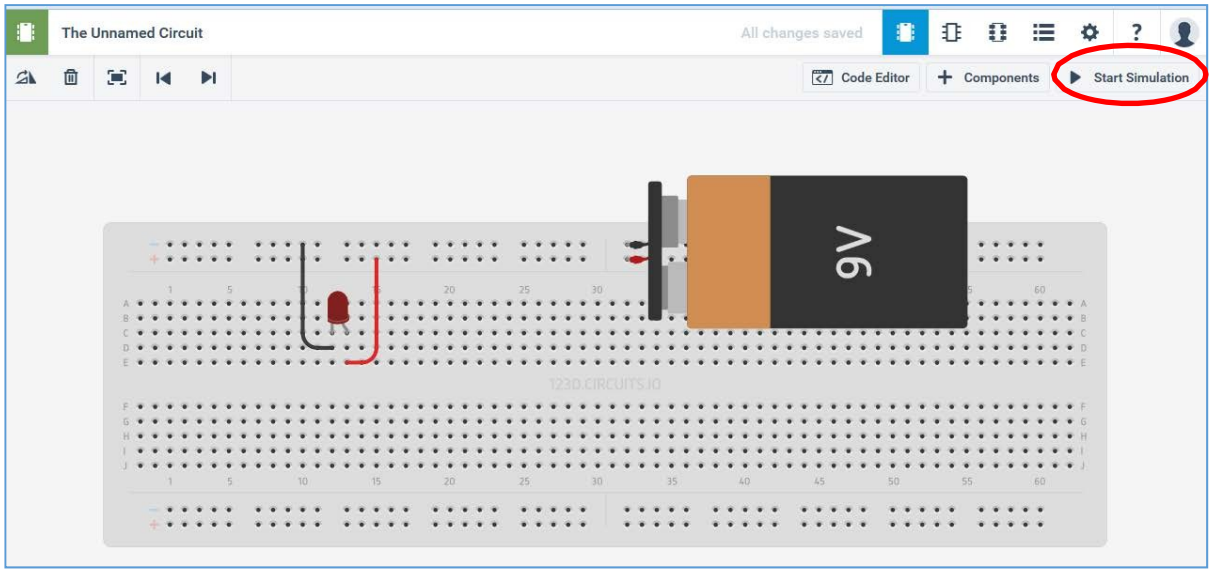


Рисунок 3.7 – Приклад підключення світлодіода

Що сталося? Симулятор показує, що наш світлодіод перегорів. Світлодіоди не можна безпосередньо підключати до джерела живлення, особливо з напругою 9 Вольт.

На цьому прикладі добре видно одну з корисних властивостей використання моделей – спалити віртуальний світлодіод можна абсолютно безкоштовно, і вам не потрібно бігти в магазин за новим. Моделювання дозволяє перевірити правильність схеми до того, як ми витратимо сили і час на її виготовлення. Можна перевірити, запрацює щось чи ні, поекспериментувати з різними компонентами і способами з'єднання, і вже після того, як ми переконаємося, що все працює як треба, можна купити компоненти і скласти реальну схему.

На жаль, моделі працюють недостатньо добре, або, краще сказати, що вони працюють дуже добре, і показують, як наш пристрій працював би в ідеальному світі. У реальному світі завжди є якісь шуми, завади, незначні відхилення від ідеальної поведінки, які в деяких випадках можуть вплинути на роботу пристрою, який до цього бездоганно працював в емуляторі.

Давайте виправимо нашу схему, щоб світлодіод не перегорав. Для цього треба додати в неї резистор на 430 Ом, послідовно з'єднаний з світлодіодом. Для зміни дротів, потрібно спочатку виділити дріт, а потім перетягнути одну з його ключових точок. Вилучити проміжну крапку можна виділивши її і натиснувши <Delete> на клавіатурі.

Запустимо схему. Вона працює нормально, світлодіод загорівся (рис 3.8).

У верхньому правому куті екрану є кнопки, які дозволяють перемикатися між різними режимами редагування моделі. Опис кнопок дано нижче в таблиці 3.3.

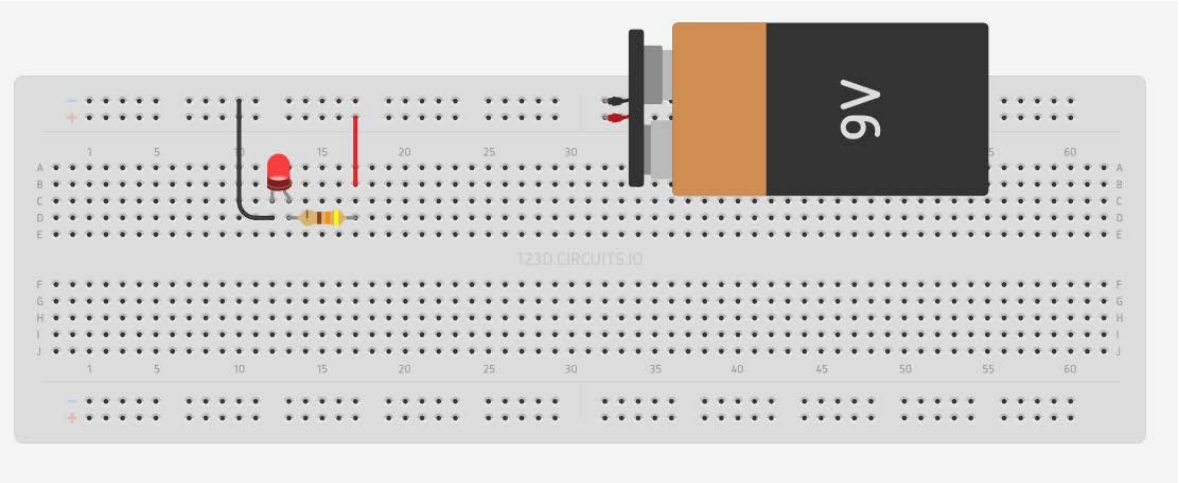


Рисунок 3.8 – Робота світлодіода

Таблиця 3.3 – Кнопки перемикання між режимами редагування

Кнопка	**Опис**
	Модель макетної плати (зараз ми працюємо в цьому режимі)
	Електрична принципова схема
	Редактор друкованої плати (можна зробити свою друковану плату)
	Список компонентів (можна роздрукувати перед походом в магазин)
	Властивості проекту (можна задати назву, опис і т.д.)
	Довідка за системою
	Ваша особиста сторінка в системі

Давайте заглянемо на сторінку з принциповою схемою. На ній ми бачимо схему, відповідну сполукам, які ми зробили на макетній платі (рис. 3.10). Компоненти можна обертати і переміщати мишкою. Якщо їх акуратно розкласти, то вийде схема, що цілком читається.

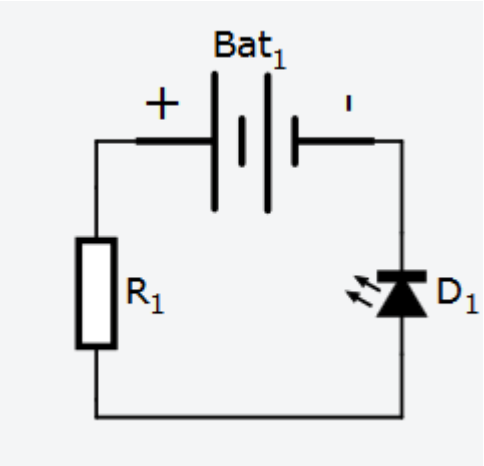


Рисунок 3.10 – Приклад схеми

Редагувати схему (додавати і вилучати компоненти і зв'язки між ними) в цьому режимі не можна. На жаль, не можна і перерозподілити руками з'єднання, вони завжди прокладаються автоматично, тому складні схеми можуть виходити досить неохайними.

Код для плат Arduino можна писати прямо в веб-інтерфейсі середовища Autodesk CIRCUITS. Для цього потрібно натиснути кнопку «Code Editor». Ця функція буде доступна тільки після додавання плати Arduino або інших програмованих компонентів у модель.

Написавши програму її можна запустити, натиснувши кнопку «Upload & Run» у верхній частині редактора коду. Там є й інші кнопки:

– Libraries – показує підтримувані емулятором бібліотеки Arduino і дозволяє їх підключити;

– Download Code –- скачати розроблений код на свій комп'ютер;

– Debugger – налагоджувач коду;

– Serial Monitor – показує повідомлення від Arduino.

Завдання 3.1

Складіть схему, яку розглянуто вище (рис. 3.10). Перевірте її роботу.

Завдання 3.2

Додайте в схему мультиметр. За допомогою кнопки встановіть режим амперметра і підключіть послідовно зі світлодіодом. Запустіть емуляцію і перевірте струм через світлодіод. Робочий струм звичайних світлодіодів не перевищує 20 мА.

3.4 Програмування мовою Сі

Програмування в середовищі Arduino здійснюється мовою Сі з мінімальним набором елементів C ++. Надалі, для стислості, називатимемо цю мову програмування мовою Сі.

Розглянемо основні особливості синтаксису цієї мови, знання яких необхідно для написання програм.

Кожен вираз закінчується символом крапки з комою.

Наприклад:

```
a = b + c;
```

Тіло функцій і складових операторів (if, else, for, while) виділяються фігурними дужками (аналогічно BeginEnd в мові Pascal), наприклад:

```
if (a> 0) {
b = a + 1;
}
```

Рядки виділяються подвійними лапками:

```
lcd.print ( "some text");
```

а символи виділяються одинарними лапками:

```
symbol = 'a';
```

Підключення бібліотек здійснюється за допомогою конструкції:

```
#include <math.h>
```

Коментарі в програмі починаються двома прямими слеш:

// це коментар

Оголошення змінної в мові Сі здійснюється за допомогою конструкції виду:

тип_змінної ім'я_змінної;

приклад:

int x, y; // оголошені змінні x і y, які мають цілий тип

У програмах для Arduino можна використовувати такі типи для числових даних:

– byte – ціле число від 0 до 255;
– int – ціле число від -32768 до +32767;
– word – ціле число від 0 до 65535;
– long – ціле число від -2147483648 до 2147483647;
– float і double – числа з плаваючою точкою, відповідають типу float для звичайних комп'ютерів, тип double в даний час еквівалентний float.

Масиви в мові Сі оголошуються конструкцією виду:

тип_елемента ім'я._массива [розмір];

приклад:

int values [10]; // масив з десяти цілих чисел

У мові Сі передбачений тип для зберігання символів і рядків, це тип char. Рядки є масивами типу char, в яких останній символ має код 0, що позначає кінець рядка.

Приклад:

char my_str [10]; // рядок не більше дев'яти символів довжиною

Тип boolean використовується для представлення логічних значень true (істина) і false (брехня). Крім того, будь-яке ціле значення може використовуватися як логічне, при цьому 0 означає брехня, а будь-яке нульове значення – істину.

Останній тип даних, який ми розглянемо – тип void. Це «порожній» тип, який використовується при оголошенні функцій, які не беруть аргументів або які не повертають результат (аналогічно процедурам в мові Pascal).

Вказівка слова const перед оголошенням змінної говорить про те, що її значення не може бути змінено.

У мові Сі використовується стандартний набір математичних операторів, що включає +, -, * і /. Для обчислення залишку від ділення використовується %.

Крім того, є їх спеціальні комбінації з оператором присвоювання, що дозволяють скоротити запис:

```
x + = 4;        // Або означає x = x + 4
```

Такі форми є для всіх операторів.

Щоб збільшити або зменшити числа на одиницю є інкремент і декременти:

```
x ++;    // Або означає x = x + 1
y--;       // Або означає y = y - 1
```

Набір операторів порівняння теж стандартний і включає оператори <,>, <=,> =. Рівність записується за допомогою двох знаків рівність (==), а нерівність записується як ! =. Логічні вирази можна обчислювати за допомогою операторів «і» - &&, «або» ||. Логічне заперечення записується за допомогою знаку оклику «!».

Умовний оператор має такий синтаксис:

```
if (умова)
код, що виконується, якщо умова істинна.
```

Якщо в разі істинності умови необхідно виконати декілька операторів, то вони об'єднуються в один блок за допомогою фігурних дужок:

```
if (умова) {
код, що виконується, якщо умова істинна
}
```

Повна форма умовного оператора включає блок else, код в якому виконується, якщо умова помилкова:

```
if (умова) {
код, що виконується, якщо умова істинна
}
else {
код, що виконується, якщо умова помилкова
}
```

Оператор switch дозволяє здійснити вибір однієї з гілок коду залежно від значення числової змінної, наприклад:

```
switch (mode) {
default:
case 0:
код для режиму 0
break;
case 1:
код для режиму 1
break;
case 2:
код для режиму 2
break;
}
```

При цьому значення змінної mode визначає, в яку точку перейде виконавець програми всередині тіла оператора. Далі виконуватимуться всі дії, якщо не зустрінеться break, який перерве подальше виконання коду всередині switch.

Якщо значення виразу не відповідає жодній з міток, то виконується код, починаючи з мітки default. У даному прикладі через відсутність break після default управління потрапить в «дії для режиму 0».

Цикл while дозволяє виконувати код до тих пір, доки умова правильна. Він має синтаксис:

while (умова)

код, що виконується в циклі.

Більш складним є цикл for: for (ініціалізація; умова; крок)

код, що виконується в циклі

У цьому циклі вираз, вказаний у блоці «ініціалізація», виконується один раз перед початком циклу. Зазвичай в ньому задають початкові значення для змінних циклу. Умова перевіряється щоразу на самому початку чергової ітерації циклу і якщо воно стане помилково, то цикл зупиняється. «Крок» виконується після виконання тіла циклу і зазвичай збільшує (або зменшує) змінні циклу. У блоках «ініціалізація» і «крок» можна змінити кілька змінних, якщо написати відповідні вирази через кому.

Приклад циклу, що обчислює суму арифметичної прогресії:

int i, sum;

for (i = 0, sum = 0; i <= 10; i ++)

sum + = i;

Код програми мовою Сі має бути обов'язково розбитий на функції. Для оголошення функції використовується такий синтаксис:

тип_функції ім'я_функції (параметри) {

код, що виконується в рамках функції

return результат_функції;

}

Тип функції повідомляє тип значення, яке повертатиме функція. Наприклад, стандартна функція cos, обчислює косинус, повертає значення типу float.

Ім'я функції може бути будь-який рядком, що починається з букви, і містить тільки літери, цифри і символи підкреслення.

Параметри – це перелік змінних, які приймають значення, передані у функцію. Змінні вказуються через кому, спочатку пишуть тип параметра, а потім – його назва.

Щоб повернути значення з функції використовується оператор return. Він негайно перериває виконання функції, а її результатом стає значення зазначеного після return виразу.

Якщо функція нічого не повертає, то тип функції вказується як void. Якщо функції не передається ніяких аргументів, то можна нічого не писати всередині круглих дужок або можна написати там слово void.

Нижче наведено приклад функції, що підсумовує два числа типу

int:

```
int sum (int a, int b) {
int result;
result = a + b;
return result;
}
```

Для виклику функції необхідно написати її ім'я і в круглих

дужках вказати через кому значення аргументів, наприклад:

```
r = sum (3, 10);
```

У програмах мовою Сі можна оголосити змінні за межами будь-яких функцій. Такі змінні називаються глобальними. Їх можна використовувати в будь-яких функціях, розташованих після оголошення змінної. Наприклад, у цій програмі змінна led використовується у функціях setup () і loop ().

```
const int led = 2;
void setup() {
pinMode(led, OUTPUT);
}

void loop() {
```

```
digitalWrite(led, HIGH;
delay(1000); digitalWrite(led,LOW); delay(1000);
}
```

Запитання для самоконтролю

1. Які з наступних рядків не відповідають синтаксису мови Сі:

a) int x;

b) int x, char u;

c) int x, y;

d) float int x.

2. Яке значення прийме змінна b після виконання цього коду:

```
int a = 2;
char b = '';
switch (a) {
default:
case 0:
b = 'a';
break;
case 1:
b = 'b';
break;
case 2:
b = 'c';
break;
 }
```

3. Чому дорівнюватиме значення змінної k у функції setup () в результаті виконання такого блоку коду:

```
int func (int b) {
int a = 0;
if (b% 2 == 0 || a> 0)
return a;
else
return (a + 1);
}
void setup () {
int k = func (5);
}
```

4. Скільки разів виконається тіло наступного циклу:

```
int i = 4;
while (i> = 0) {
i--;
}
```

3.5 Структура скетчу

У попередньому розділі ми говорили про мову програмування, яка використовується в Arduino. Ця мова дуже схожа на мову Сі, але є і деякі відмінності.

По-перше, коли пишуть програму для Arduino, вона обов'язково має бути написана в одному файлі, який називають скетч. Ви можете часто зустріти таке поєднання слів «скетч для Arduino». Це означає програма для Arduino.

Є відмінність в структурі скетчу від структури звичайної програми на Сі, і викликана вона відмінністю між роботою програми на звичайному комп'ютері

і у вбудованих системах. Коли виконують програму на звичайному комп'ютері, то вона запускається, виконує якісь дії, а потім завершується. Якщо програму пишуть мовою Сі, то за її виконання відповідає функція main ().

Під час запуску програми запускається саме ця функція. З неї програміст вже може викликати інші функції програми в тому порядку, в якому вони мають виконуватися. Коли функція main () повертає управління за допомогою return, програма завершується.

Але для програми, що управляє мікроконтролером, така поведінка не має сенсу. Поки на пристрій подано живлення, мікроконтролер має керувати пристроєм, а значить, він повинен виконувати свою програму. Ця програма ніколи не завершується сама, вона перестає працювати, тільки якщо вимкнути мікроконтролер.

Тому структура програми для Arduino відрізняється від структури звичайної програми мовою Сі. У програмі для Arduino не має бути функції main (), замість неї мають бути дві функції: setup () і loop ().

Перша з цих функцій, функція setup (), викликається один раз на самому початку виконання програми відразу після того, як мікроконтролер включили. Вона має підготувати мікроконтролер і всі пристрої до роботи.

Зазвичай у цій функції налаштовують мікроконтролер, наприклад, задають, які його виводи використовуватимуться як входи, а які – як виходи і т. д. У цій функції можна вивести на екран вітання для користувача (якщо у пристрої є екран). Або якимось іншим чином приготувати пристрій до роботи. Може бути, потрібно щось включити, або навпаки, переконатися, що нічого зайвого поки не включено.

Друга функція, функція loop (), виконується весь час, поки працює наш пристрій. Якщо вона завершиться, то середовище Arduino запустить її заново. У цій функції як раз і потрібно реалізувати алгоритм роботи пристрою. Зазвичай в ній опитують підключені до мікроконтролера сенсори, або самі входи

мікроконтролера, аналізують отриману інформацію і видають команди виконавчим пристроям.

Можна вважати, що десь в бібліотеці Arduino вже є функція main (), яка влаштована приблизно так:

```
void main () {
setup ();
while (true)
loop ();
}
```

Така функція там дійсно є, швидше за все вона влаштована більш складно, вона може робити ще якісь дії з керування мікроконтролером, але загальна структура саме така. А нам потрібно написати дві функції setup () і loop ().

Як і в звичайних програмах, ми можемо створювати і будь-які інші свої функції і викликати їх з setup () і loop (). Якщо ви робите щось просте, то можна написати весь код просто всередині setup () і loop (). Так слід робити, якщо весь код поміщається в кілька рядків. Якщо робити щось складне, то необхідно розбити код на окремі функції.

У програмі Arduino можна викликати стандартні функції, які є в бібліотеці. Для цього не потрібно додавати директив #include. У звичайній мові Сі це потрібно робити навіть для стандартних функцій. Одна з таких, яка часто використовуватиметься, це функція delay (час), що робить затримку на зазначений час. Час задається в мілісекундах, тому, щоб зачекати 1 секунду, треба написати:

```
delay (1000);
```

Є ще кілька стандартних функцій, які ми використовуватимемо, і які можна відразу писати в програмі. Ці функції керують базовими можливостями мікроконтролера, і ми їх розглянемо, коли вони нам знадобляться.

Великою перевагою середовища Arduino є її велика популярність. Багато виробників випускають плати розширення, які можна підключати до Arduino і використовувати разом з нею. Для того щоб такі плати було зручно використовувати, для них пишуть бібліотеки, що містять зручні функції для роботи з цими пристроями. Пишуть такі бібліотеки і просто для окремих компонентів, які можна підключити до плати Arduino. Ми використовував-тимемо, наприклад, бібліотеку для управління рідкокристалічним екраном.

Щоб використовувати бібліотеку, потрібно на початку програми помістити директиву #include:

#include <ім'я_бібліотеки.h>

Наприклад, бібліотека для роботи з дисплеєм називається

LiquidCrystall, і щоб її використовувати, треба спочатку написати

#include <LiquidCrystal.h>

У мові Arduino для роботи з бібліотеками використовують елементи мови C ++. Фактично, самі бібліотеки пишуть мовою C ++, але в скетчі можна використовувати тільки дуже прості можливості цієї мови. Мова C ++ дозволяє створювати класи. Класи – це призначені для користувача типи даних, що описують набір полів і операції, які з ними можна робити. За допомогою цього механізму в C ++ можна зробити, наприклад, клас «дисплей» і змінну такого типу. Такі змінні називають об'єктами. Потім можна працювати з цими «складними» змінними. Щоб зробити таку змінну, її треба оголосити, як оголошують звичайні змінні. Наприклад, для роботи з дисплеєм треба зробити змінну типу LiquidCrystall:

LiquidCrystal lcd (4, 5, 6, 7, 8, 9);

Тут LiquidCrystall – це назва типу, його пишуть в документації на бібліотеку. lcd – це назва нашої змінної, можна використовувати будь-яке ім'я, як зручно. Цифри – це параметри, які передаються об'єкту в ході створення. Про те, які параметри треба вказати, теж пишуть в документації. Для дисплея

це, насправді, номери висновків мікроконтролера, до яких він підключений. Пізніше ми розглянемо роботу з дисплеями більш докладно.

Після того, як змінна створена, ми можемо з нею працювати. Для цього використовують спеціальні функції, які доступні тільки для цієї змінної. Такі функції ще називають «методами класу». Викликають їх таким способом:

ім'я_змінної.ім'я_функції (параметри);

Наприклад, текст на дисплей виводить функція print, і її можна викликати таким чином:

lcd.print ("Hello!");

Оскільки ми вказуємо спочатку ім'я змінної, то функція print викликається саме для цієї змінної. Можна підключити кілька дисплеїв і зробити для кожного з них свою змінну.

Запитання для самоконтролю

1. Які функції обов'язково мають бути присутніми в скетчі для Arduino?

a) setup ();
b) main ();
c) startup ();
d) loop ();
e) init ().

2. Виберіть правильні твердження:

a) «клас» – це тип даних, а «об'єкт» – це значення такого типу;

b) для використання будь-якої стандартної функції в середовищі Arduino, треба на початку програми написати директиву #include і вказати ім'я бібліотеки;

c) бібліотеки пишуть для спрощення роботи з різними пристроями;

d) щоб працювати з двома однаковими пристроями, для яких є бібліотека, треба буде даічі підключити цю бібліотеку за допомогою #include;

e) щоб працювати з двома однаковими пристроями, для яких є бібліотека, треба буде один раз підключити бібліотеку за допомогою #include і створити два об'єкти відповідного класу.

4 ЦИФРОВИЙ ВХІД-ВИХІД

4.1 Цифрові виходи. Робота зі світлодіодом

Одне з найпростіших дій, яке може виконати мікроконтролер, це перемикання стану контактів цифрових входів-виходів. Будь-який з цифрових виходів мікроконтролера можна програмним способом переключити між високим рівнем (+5 V) і низьким (0 V). За допомогою такого перемикання відбувається управління простими пристроями.

Перемикання здійснюється функцією digitalWrite, виклик якої має вигляд:

digitalWrite (номер_контакту, рівень_сигналу);

Параметр рівень_сигналу може приймати два значення: HIGH (високий, +5 V) або LOW (низький). Параметр номер_контакту – це число, що відповідає номеру контакту на платі Arduino.

Зверніть увагу, що одні й ті ж виводи мікроконтролера можуть виступати як в ролі цифрових виходів, так і в ролі цифрових входів. Тому перед їхнім використанням необхідно вказати, в якій ролі використовуватиметься контакт: входу або виходу. Це робиться за допомогою функції pinMode:

pinMode (номер_контакту, режим_контакту);

Її аргумент режим_контакту може приймати значення: OUTPUT (вихід) і INPUT (вхід). Таким чином, щоб встановити на виході №2 високий рівень сигналу, достатньо виконати таку програму:

```
const int pin = 2;

void setup () {
pinMode (pin, OUTPUT);
digitalWrite (pin, HIGH);
}
```

```
void loop () {
}
```

Один з найбільш простих для підключення виконавчих пристроїв, яким можна керувати за допомогою цифрового виходу, – це світлодіод. Цей пристрій є напівпровідниковим приладом, здатним випромінювати світло під час проходження через нього електричного струму в прямому напрямку (від анода до катода). На рис. 4.1 показано зовнішній вигляд світлодіода і його позначення на електричній схемі.

Для того щоб правильно включити світлодіод в електричний ланцюг, необхідно відрізняти катод від анода. Зробити це можна за трьома ознаками:

1) анод світлодіода має довший провідник;
2) з боку катода корпус світлодіода трохи зрізаний;
3) всередині світлодіода катод ширше, ніж анод.

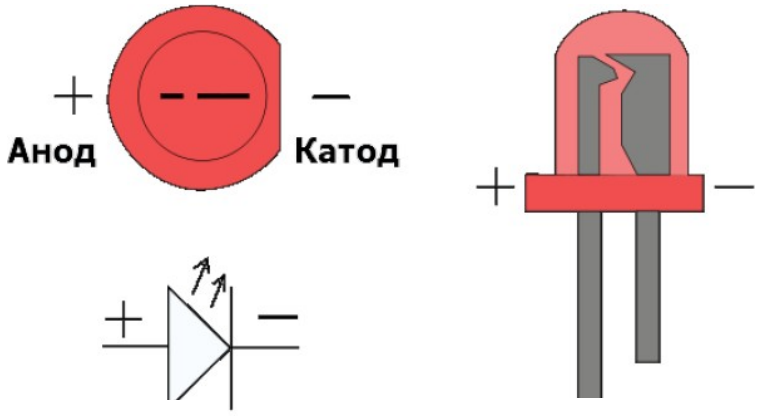
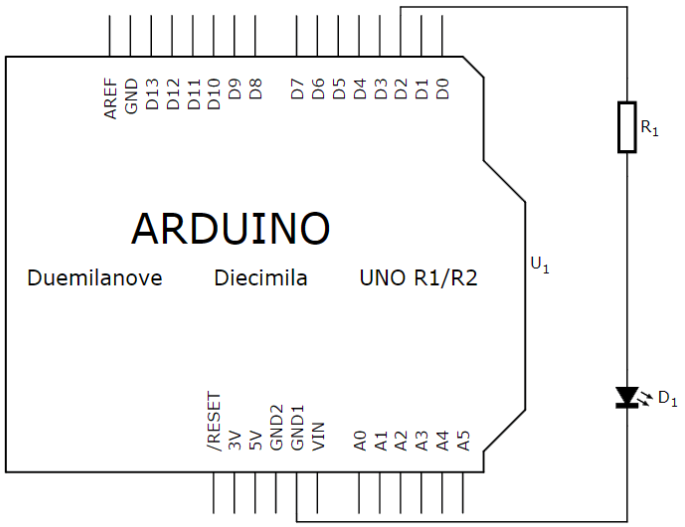


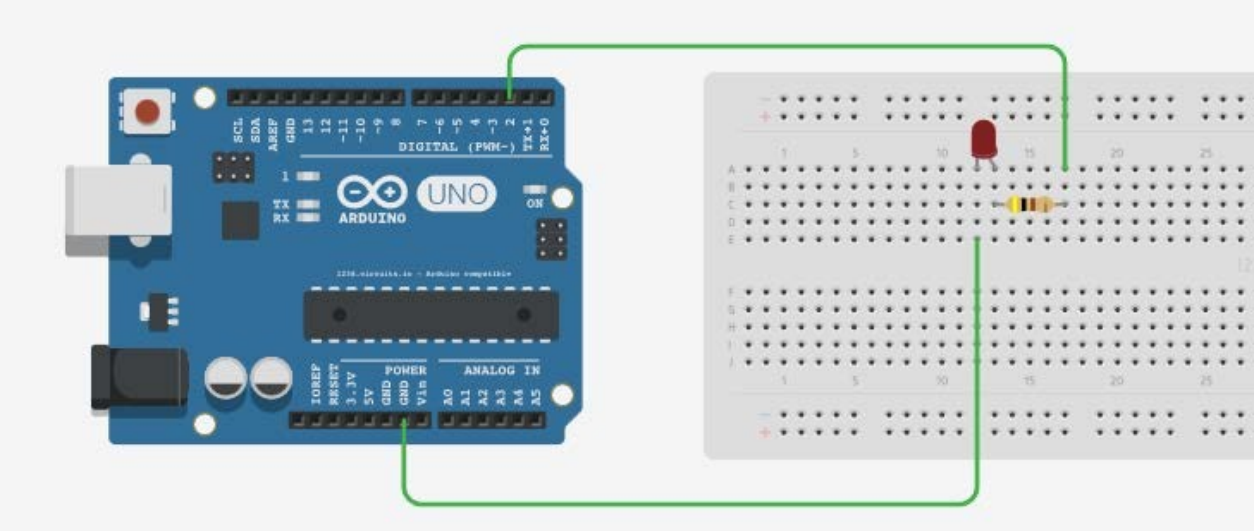
Рисунок 4.1 – Позначення на електричній схемі світлодіода (а) і його зовнішній вигляд (б)

У сучасній мікроелектроніці застосовуються мініатюрні світлодіоди для поверхневого монтажу. Такі індикатори, наприклад, є на Arduino Uno для інформування користувача про стан системи.

Важливо відзначити, що робоча напруга живлення світлодіода варіюється від 1.85 до 2.5 Вольт при рекомендованій силі струму не більше 20 мА. Щоб сила струму не перевищила це значення, при підключенні світлодіода до виходу мікроконтролера, який видає напругу 5 В, у ланцюг слід додати послідовний обмежуючий резистор опором від 200 до 500 Ом. В іншому випадку струм через світлодіод буде занадто великим, що може пошкодити як діод, так і вивід мікроконтролера. На рис. 4.2 наведена електрична схема



підключення світлодіода до Arduino Uno, а також модель складеного пристрою.



а

б

Рисунок 4.2 – Електрична схема підключення світлодіода (*а*) і модель складеного пристрою (*б*)

Спробуємо «моргнути» світлодіодом. Для цього ми послідовно запалюватимемо його, передаючи на ногу №2 сигнал HIGH, а потім гасити за допомогою сигналу LOW. Між включенням і вимиканням світлодіода обов'язково потрібно поставити затримку в кілька сотень мілісекунд, інакше ми не помітимо миготіння. Згадаймо, що контролер Arduino працює на частоті 16МГц, тому він може вмикати і вимикати світлодіод тисячі разів у секунду.

Отримуємо таку програму:

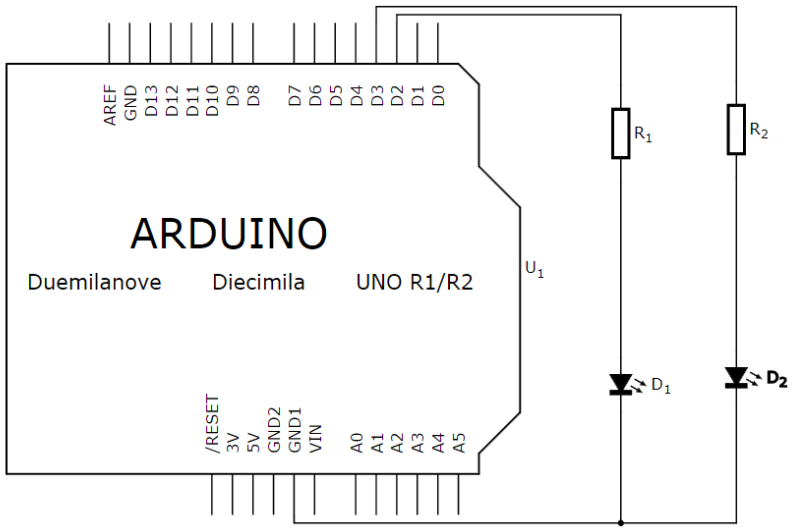
```
int led = 2;

void setup () {
pinMode (led, OUTPUT);
}

void loop () {
digitalWrite (led, HIGH);
delay (1000);
digitalWrite (led, LOW);
delay (1000);
}
```

Маємо такий код:

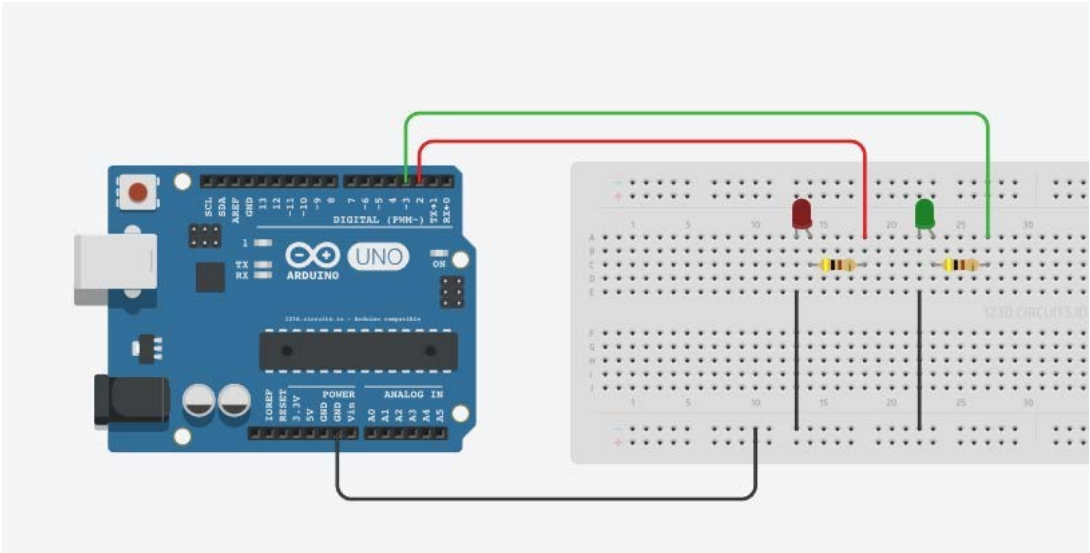
```
int led_r = 2;
int led_g = 3;

void setup () {
pinMode (led_r, OUTPUT);
pinMode (led_g, OUTPUT);
}

void loop () {
```

```
digitalWrite (led_r, HIGH);
digitalWrite (led_g, HIGH);
delay (1000); digitalWrite (led_r, LOW);
digitalWrite (led_g, LOW);
delay (1000);
}
```

Аналогічним чином можна підключити ще один світлодіод (рис. 4.3).



а



б

Рисунок 4.3 – Схема підключення двох світлодіодів (*а*) і модель зібраного пристрою (*б*)

Для підключення відразу двох світлодіодів до землі використовуємо шину живлення на макетній платі.

Завдання 4.1

Реалізуйте на базі розробленої схеми програму «Проблискові маячки». Світлодіоди повинні поперемінно мигати з інтервалом 0.3 с (рис. 4.3).

Завдання 4.2

Підключіть до Arduino три світлодіода (червоний, жовтий і зелений) і реалізуйте програму, яка змушує світлодіоди загорятися в режимі світлофора, повторюючи такий цикл:

1) зелений світлодіод горить 3 секунди;
2) зелений світлодіод блимає 3 секунди;
3) жовтий світлодіод запалюється на 1 секунду;
4) червоний світлодіод запалюється на 3 секунди.

4.2 Вивід інформації через послідовний порт

Arduino може надсилати текстові повідомлення на персональний комп'ютер. Для того щоб цим скористатися, необхідно в функції setup форматувати послідовний порт пристрою:

Serial.begin (9600);

Значення 9600 означає швидкість передачі даних у бітах за секунду. 9600 – це одна зі стандартних швидкостей послідовного порту. Безпосередньо для виведення короткого повідомлення використовується функція print:

Serial.print ("text");

Приклад програми:

```
void setup () {
Serial.begin (9600);
Serial.print ( "Hello!");
}
void loop () {
}
```

Щоб побачити ці повідомлення в Autodesk CIRCUITS включіть Serial Monitor на панелі управління над редактором коду.

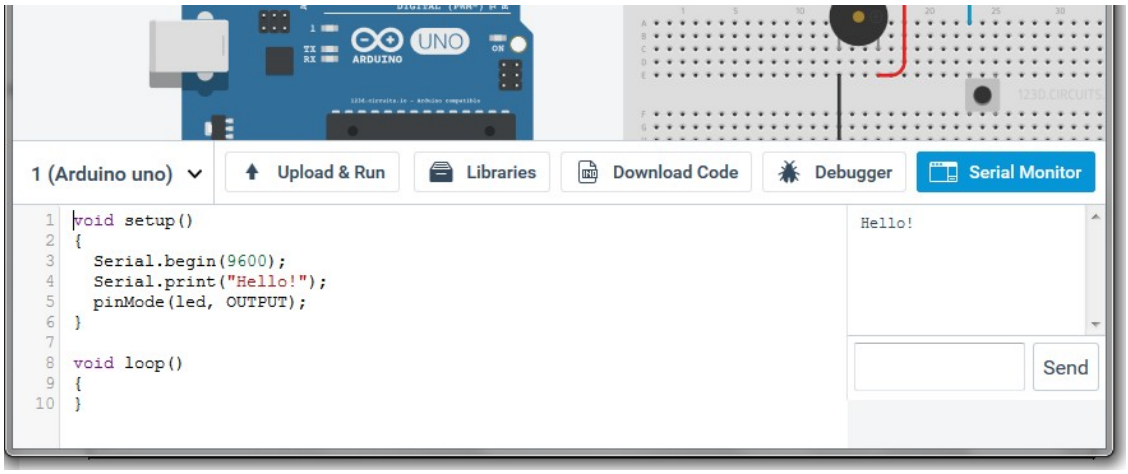


Рисунок 4.4 – Приклад роботи Serial Monitor

Якщо ви використовуєте справжню плату Arduino і редактор Arduino IDE, використовуйте наявне в ньому вікно «Монітор порту».

Завдання 4.3

Напишіть програму, яка щосекунди передаватиме через послідовний порт число, яке збільшується на одиницю.

4.3 Цифрові входи, підключення кнопок і вимикачів

Вимикач – це прилад, який дозволяє замикати і розмикати електричний ланцюг. Зміна стану вимикача може відбуватися різними способами, залежно від типу пристрою.

Механічні вимикачі використовуються безпосередньо людиною. До такого типу належать різні тумблери, кнопки, клавіші та рубильники. Електромагнітні та електронні, навпаки, застосовуються в автоматичних системах і управляються за допомогою електричних сигналів. Найвідомішим електромагнітним вимикачем є реле (relay). Прикладом електронного вимикача може служити транзистор (transistor).

У нашій роботі ми використовуватимемо простий механічний вимикач (pushbutton), який замикає два своїх контакти під час натискання. Для зручності монтажу їх часто роблять з чотирма контактами, у яких виходи з'єднані попарно.

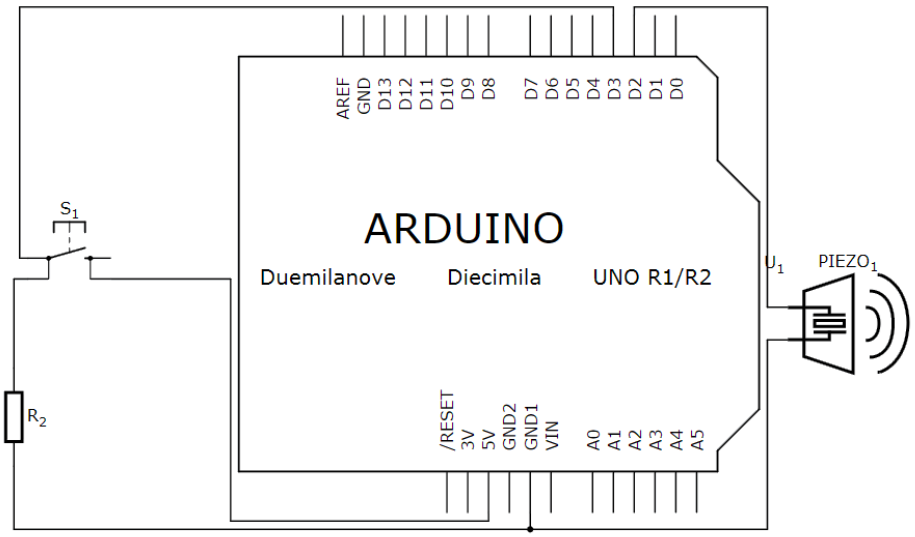
Для читання цифрового сигналу з одного з контактів Arduino, необхідно скористатися функцією digitalRead:

результат = digitalRead (номер_контакта);

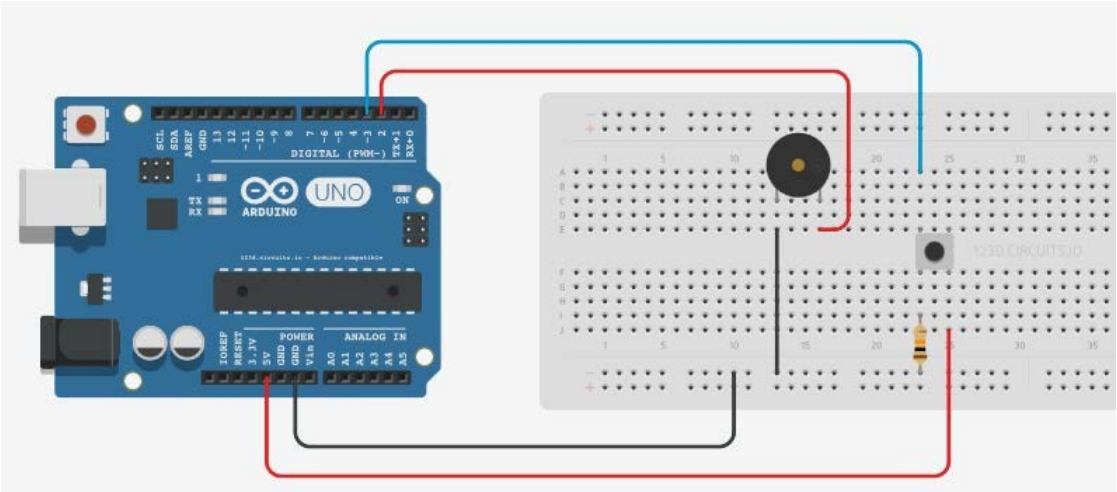
Після виклику цієї функції змінна результату зберігатиме рівень цифрового сигналу, що детектується на відповідному контакті.

Щоб читати стан кнопки, її потрібно підключити так, щоб під час її натискання на вході був високий рівень напруги (5 В), а під час відпускання – низький (0 В). Для цього підключимо послідовно між шиною живлення і землею кнопку і резистор, а вхід мікроконтролера підключимо посередині між ними (див. рис. 4.5). Тоді, якщо кнопка не була натиснута, то вихід мікроконтролера буде з'єднаний через резистор із землею і матиме низький рівень, а якщо кнопку натиснути, то вийде, що вхід з'єднаний з живленням і на ньому буде високий рівень.

За такою схемою підключення через резистор потече струм, який залежить від його опору. Щоб схема не споживала багато зайвої електрики, візьмемо резистор побільше, наприклад, 10 кОм.



а



б

Рисунок 4.5 – Приклад схеми підключення резистора (*а*) і складена модель (*б*)

Напишемо програму для виведення звукового сигналу після натискання кнопки. Для цього використовуємо ще один компонент – зумер, який видає звуковий сигнал, коли на нього подається високий рівень напруги. Після запуску програма починає постійно перевіряти стан кнопки. Якщо кнопка натиснута, то зумер вмикається, а якщо відпущена – вимикається:

```
const int btn = 3;
const int buzz = 4;
 byte val = 0;

void setup () {
// установка 3-го контакту в режим введення
pinMode (btn, INPUT);
// установка 4-го контакту в режим виведення
pinMode (buzz, OUTPUT);
}

void loop () {
val = digitalRead (btn);

// умова «натиснута кнопка»
if (val == HIGH)
// перехід зумера в активний стан
digitalWrite (buzz, HIGH);

else
// перехід зумера в неактивний стан
digitalWrite (buzz, LOW);
}
```

Тепер модифікуємо програму, щоб натискання на кнопку переключало звуковий сигнал, тобто натискаємо перший раз – зумер включається і працює, натискаємо вдруге – вимикається і т. д. Для цього введемо додаткову змінну state, в якій зберігатимемо поточний стан зумера.

Натискаємо кнопку один раз: у змінну стану записується 1, натискаємо другий – 0, третій – знову 1 і т.д. А зумер активуватимемо залежно від того, яке

значення зберігається в цій змінній:

```
const int btn = 3;
const int buzz = 4;
byte val = 0;
bool state = 0;

void setup () {
// установка 3-го контакту в режим введення
pinMode (btn, INPUT);
// установка 4-го контакту в режим виведення
pinMode (buzz, OUTPUT);
}

void loop () {
val = digitalRead (btn);

// умова зміни стану
if (val == HIGH) {
state =! state;
delay (200);
}
if (state == true) // умова активації зумера
// перехід зумера в активний стан
digitalWrite (buzz, HIGH);
else
// перехід зумера в неактивний стан
digitalWrite (buzz, LOW);
}
```

Зверніть увагу на затримку в 0,2 секунди після фіксації натискання на кнопку. Ця затримка потрібна, щоб виключити вплив так званого брязкоту контактів. Коли кнопка натискається, її контакти не відразу переходять в замкнутий стан, вони встигають кілька разів дуже швидко замкнутися і розімкнутися, що може бути сприйнято мікроконтролером як кілька різних натискань. Затримка дозволяє пропустити час, поки контакт не стабілізується.

Завдання 4.3

Напишіть програму, яка щосекунди передаватиме через послідовний порт число, яке збільшується на одиницю.

Завдання 4.4

Складіть схему, в якій до Arduino підключено дві кнопки, світлодіод і зумер. Напишіть програму так, щоб після натискання на одну з кнопок, світлодіод починав блимати, а зумер – видавати періодичний сигнал (лампочка і зумер мають працювати синхронно: коли лампочка горить, зумер видає сигнал, коли лампочка згасає, зумер теж вимикається). Обидва сигнали вимикаються після натискання другої кнопки.

Завдання 4.5

Складіть схему і напишіть програму для генератора сигналу SOS. У схемі мають бути присутніми одна кнопка, світлодіод і зумер. Після натискання на кнопку, зумер починає передавати сигнал SOS. Паралельно з зумером світлодіод дублює сигнал світловими імпульсами. Вимикається генератор за допомогою другого натискання кнопки.

Нагадування: сигнал SOS є послідовністю коротких і довгих сигналів, що чергуються в такий спосіб: три коротких, три довгих, три коротких, пауза, три коротких, три довгих, три коротких, пауза, ...

Завдання 4.6

Зробіть гру на реакцію. У схемі мають бути присутніми дві кнопки, два світлодіоди і зумер. Після запуску програми зумер починає видавати короткі імпульси через нерівні проміжки часу. Кожен гравець повинен якомога швидше натиснути на кнопку відразу після сигналу зумера. У гравця, натиснувши свою кнопку першим, на дві секунди запалюється світлодіод.

Завдання 4.7

Зробіть модель кодового замка. У схемі присутні три кнопки, зелений і червоний світлодіоди, а також зумер. Після запуску програми горить червоний світлодіод. Користувачеві необхідно натиснути три кнопки в правильній послідовності. Якщо кнопки натиснуті правильно, загоряється зелений світлодіод, в іншому випадку зумер видає п'ять довгих сигналів.

5 ІНДИКАЦІЯ

5.1 Семисегментний індикатор

Ми вже ознайомилися з світлодіодом, який є одним з найбільш часто використовуваних індикаторів. Звичайним світлодіодом легко інформувати користувача про якісь бінарні події, тобто події, про які важливо знати, настали вони чи ні: увімкнення або вимкнення будь-якого пристрою, про перевищення порогових значень і т. ін.

Але що, якщо потрібно повідомляти користувачеві більш складну інформацію, наприклад, числову? Іншими словами, що якщо прилад має повідомляти вимірювані значення явно, у вигляді чисел? Для цих цілей в електроніці часто використовується сегментний (або семисегментний) світлодіодний індикатор. Цей прилад є набором звичайних світлодіодів, розташованих у вигляді вісімки так, що, запалюючи деякі з них, можна отримати контури різних цифр.

Конструктивно, світлодіодні індикатори поділяються на прилади з загальним катодом і з загальним анодом. Загальний катод, наприклад, означає, що всі світлодіоди всередині індикатора пов'язані один з одним катодами, а їх аноди розведені по окремих контактах. Саме такий тип індикатора використовується в нашому навчальному посібнику.

Зовнішній вигляд і позначення на схемі сегментного індикатора наведені на рис. 5.1. Всі сегменти промарковані літерами латинського алфавіту, починаючи з a і закінчуючи g. Як правило, рахунок ведуть з найвищого сегмента за годинниковою стрілкою. Точка маркується окремо, словом dot (точка).

Крім контактів, з'єднаних з сегментами, на схемі видно два контакти com і один dot. Слово dot використовується для позначення точки, а com – це загальний катод (або анод). Він часто буває виведений в двох місцях на індикаторі.

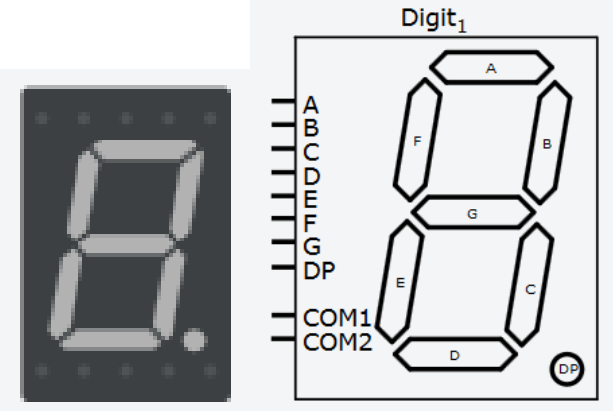
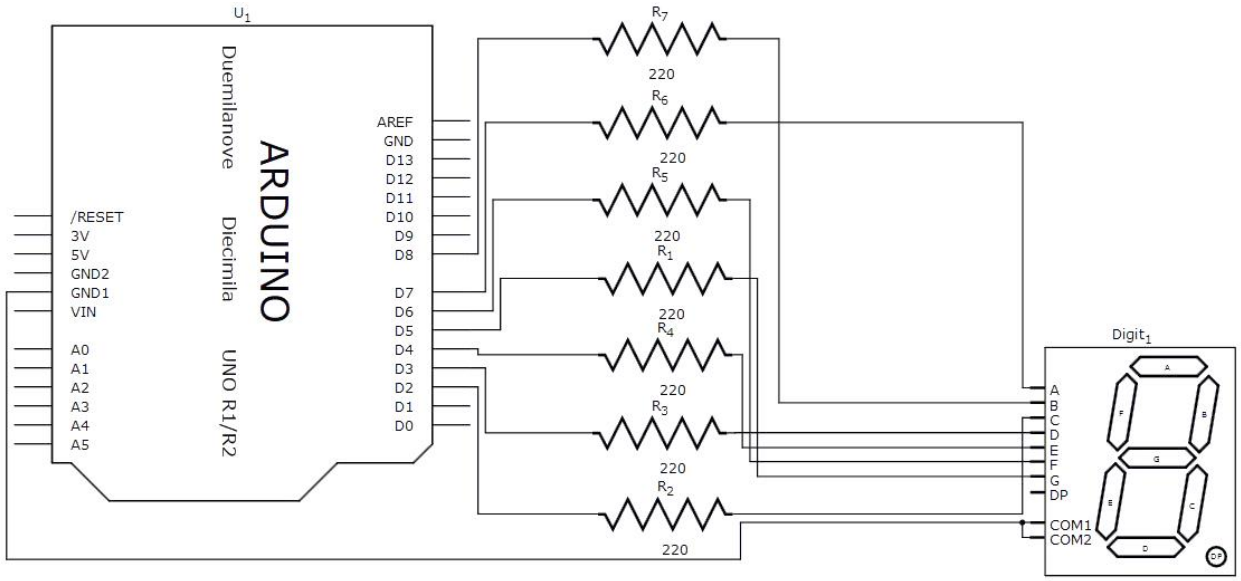
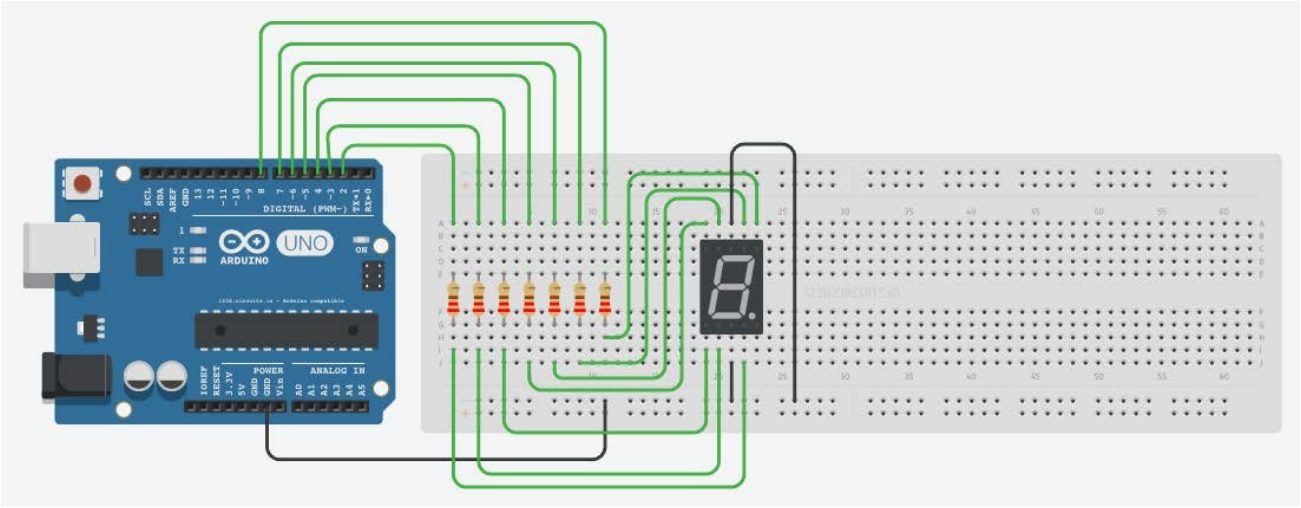


Рисунок 5.1 – Сегментний індикатор

Підключення семисегментного індикатора здійснюється так само, як і звичайного світлодіода. Оскільки в індикаторі присутні 7 діодів, потрібно використовувати 7 виводів мікроконтролера і 7 резисторів.



а



б

Рисунок 5.2 – Приклад підключення семисегментного індикатора (*а*) і модель складеного пристрою (*б*)

Програма, наведена нижче, відображає цифру 5, включаючи сегменти a, f, g, c, d.

```
int seg_a = 7;
int seg_b = 8;
int seg_c = 2;
int seg_d = 3;
int seg_e = 4;
int seg_f = 6;
int seg_g = 5;

void setup () {
pinMode (seg_a, OUTPUT);
pinMode (seg_b, OUTPUT);
pinMode (seg_c, OUTPUT);
pinMode (seg_d, OUTPUT);
pinMode (seg_e, OUTPUT);
```

```
pinMode (seg_f, OUTPUT);
pinMode (seg_g, OUTPUT);
digitalWrite (seg_a, HIGH);
digitalWrite (seg_f, HIGH);
digitalWrite (seg_g, HIGH);
digitalWrite (seg_c, HIGH);
digitalWrite (seg_d, HIGH);
}
void loop () {}
```

Завдання 5.1

Реалізуйте програму, яка здійснює анімацію сегментів. Після включення плати на індикаторі повинні по черзі загорятися і потухати сегменти в порядку: верхній, правий верхній, правий нижній, нижній, лівий нижній, лівий верхній і т.д.

Завдання 5.2

Реалізуйте світлофор із зворотним відліком. Додайте до створеного раніше в завданні 4.2 світлофор семисегментний індикатор, на якому виводитиметься кількість секунд, що залишилися до кінця чергової фази його роботи. Збільште тривалість фази до 9 секунд.

Завдання 5.3

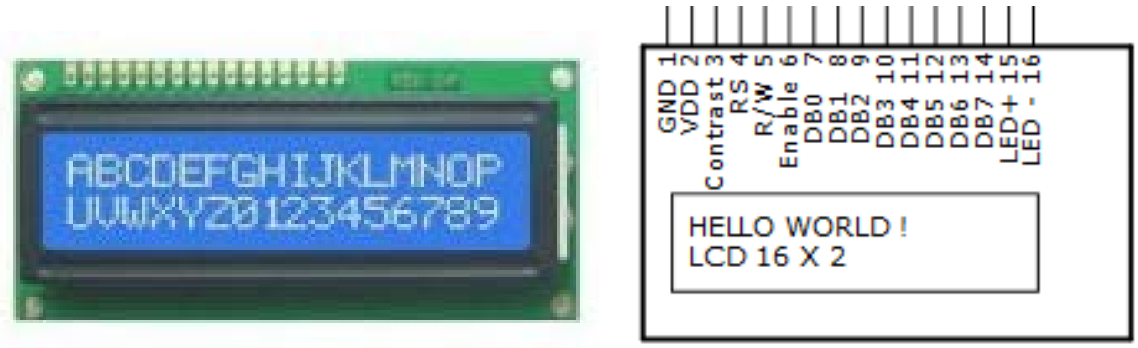
Реалізуйте секундомір. Для цього складіть схему з кнопкою і сегментним індикатором. При запуску плати на індикаторі висвічується 0. Після натискання на кнопку кожну секунду цифра збільшується на одиницю. Після 9 лічильник переходить в 0 і рахує далі. При другому натисканні на кнопку рахунок

зупиняється, а при третьому натисканні значення на індикаторі скидається на 0. Далі все повторюється заново.

5.2 Рідкокристалічний дисплей

Найінформативніший і водночас найскладніший вид індикаторів – це дисплеї. У нашому навчальному посібнику ми працюватимемо з рідкокристалічним індикатором, представників якого можна зустріти в багатьох електронних пристроях: електронний годинник, калькулятори, екрани старих стільникових телефонів – всі вони використовують подібні РК індикатори.

Ми використовуватимемо дисплей 16 x 2, що означає, що він вміє відображати 2 рядки по 16 символів. Зовнішній вигляд дисплея і його позначення на схемі показані на рис. 5.3.



а *б*

Рисунок 5.3 – Зовнішній вигляд дисплея (*а*) і його позначення на схемі (*б*)

Призначення контактів індикатора таке:

– GND – земля (мінус живлення);
– VDD – живлення +5 В;
– Contrast – регулятор контрасту дисплея;
– RS – вибір регістру;
– R / W – напрямок передачі даних (запис / читання);
– Enable – синхронізація;

– DB0 ... DB7 – шина даних;

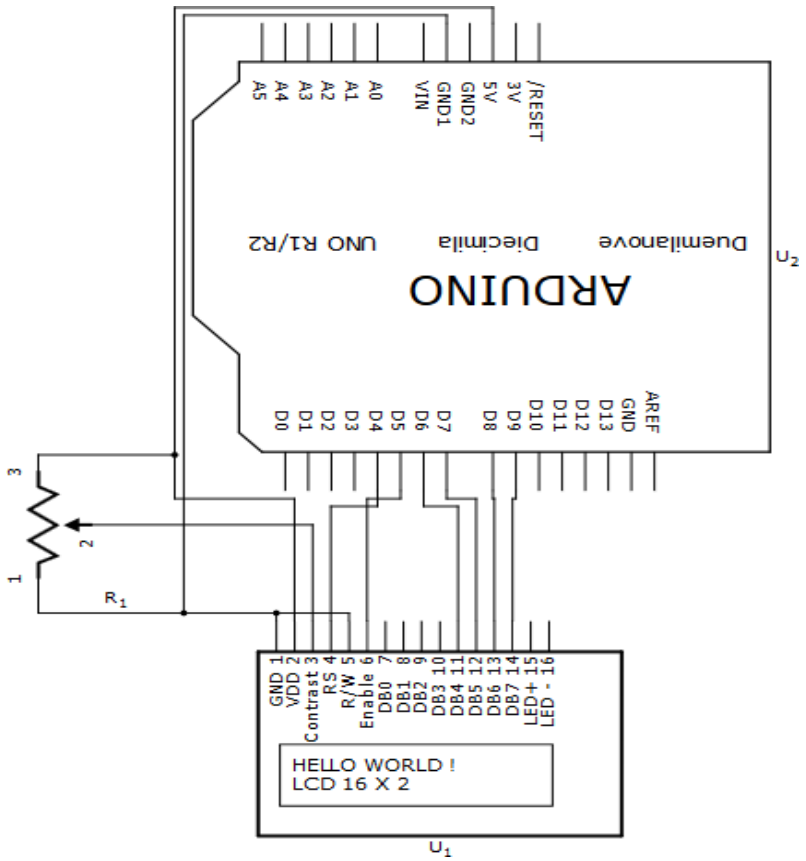
– Led- – катод підсвічування;

– Led + – анод підсвічування.

Схема і вид моделі підключення дисплея показані на рис. 5.4. Крім підключення інформаційних сигналів і живлення потрібно забезпечити регулювання контрасту дисплея. Для цього під'єднують потенціометр 10 кОм так, як показано на схемі. Детальніше про потенціометр мова піде в наступному розділі. Якщо потрібне підсвічування дисплея, то його підключають за допомогою контактів LED- і LED+ через резистор не менше 100 Ом.

Для роботи з РК дисплеями подібного типу різних розмірів у середовищі Arduino IDE є спеціальна бібліотека LiquidCrystal. Для того щоб скористатися цією бібліотекою в програмі, необхідно на початку коду вказати таку директиву:

#include <LiquidCrystal.h>



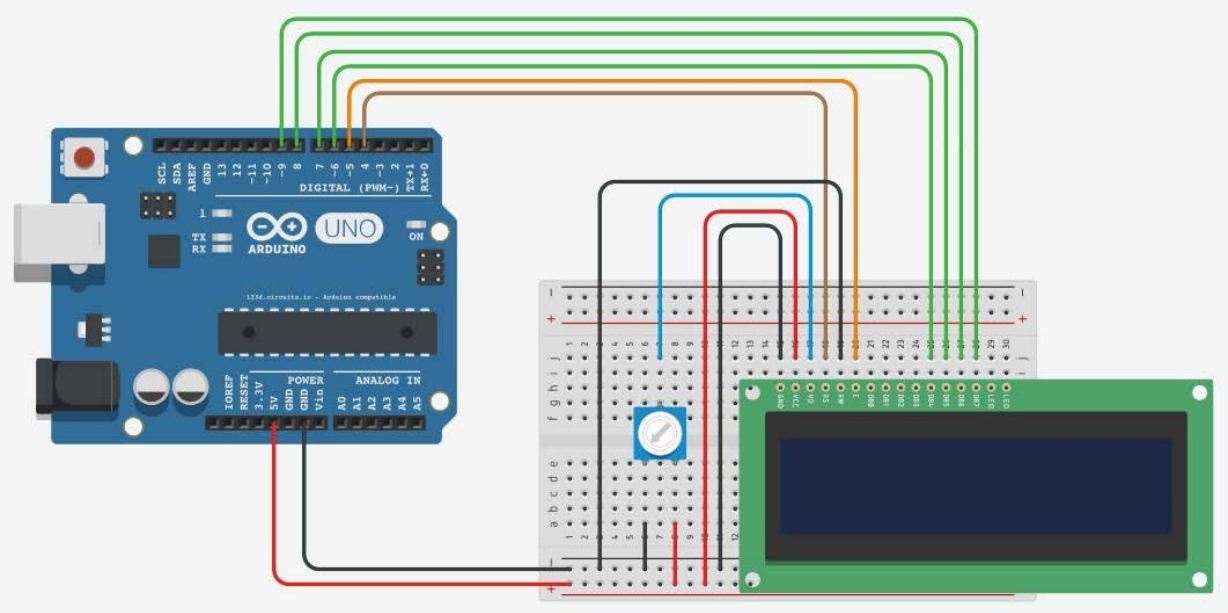


Рисунок 5.4 – Приклад підключення дисплея до загальної схеми

Після цього треба ініціалізувати дисплей, вказавши, які саме виводи використовуються для підключення до Arduino:

LiquidCrystal lcd (4, 5, 6, 7, 8, 9);

де перший аргумент – контакт RS, другий – Enable, а з третього по шостий – DB4…DB7. Інші виводи дисплея підключати до Arduino не потрібно.

Виведення тексту на дисплей здійснюється за допомогою функції print таким чином:

lcd.print (текст_або_змінна);

Як аргумент функції можна передавати текст, виділений подвійними лапками, або змінні будь-яких типів.

У бібліотеці передбачена функція для явної вказівки позиції виведення:

lcd.setCursor (номер_колонки, номер_рядка);

Так, для виведення символу у другому рядку потрібно викликати setCursor з такими значеннями аргументів:

```
// колонка з індексом 0, рядок з індексом 1
lcd.setCursor (0, 1);
```

Спробуємо вивести на дисплей повідомлення «hello, world!»:

```
#include <LiquidCrystal.h>

// ініціалізація дисплея
LiquidCrystal lcd (4, 5, 6, 7, 8, 9);

void setup () {
// параметри дисплея: 16 символів, 2 рядка
lcd.begin (16, 2);
// вивід на дисплей тексту
lcd.print ( "hello, world!");
}

void loop () {
}
```

Завдання 5.4

Реалізуйте секундомір з виводом часу на дисплей. Для цього додайте до схеми підключення дисплея дві кнопки. Напишіть програму, яка по одній кнопці запускає і зупиняє секундомір, а по іншій – скидає його показання в 0. Коли секундомір запущено, на дисплеї має відображатися час з моменту його запуску, час у форматі хвилини: секунди.

Завдання 5.5

Реалізуйте вивід рядка, що біжить "Hello, world!". Після подачі живлення в першому рядку дисплея має з'явитися зміщуючий справа наліво текст.

6 РОБОТА З АНАЛОГОВИМИ СИГНАЛАМИ

6.1 Сенсори. Резистивні сенсори

У попередніх розділах для обмеження сили струму через світлодіод ми використовували резистори. Як було тоді зазначено, існує безліч резисторів різного номіналу, розрахованих на різну потужність навантаження. Крім звичайних резисторів є аналогічні прилади, але тільки із змінним опором, які називаються потенціометрами (або реостатами).

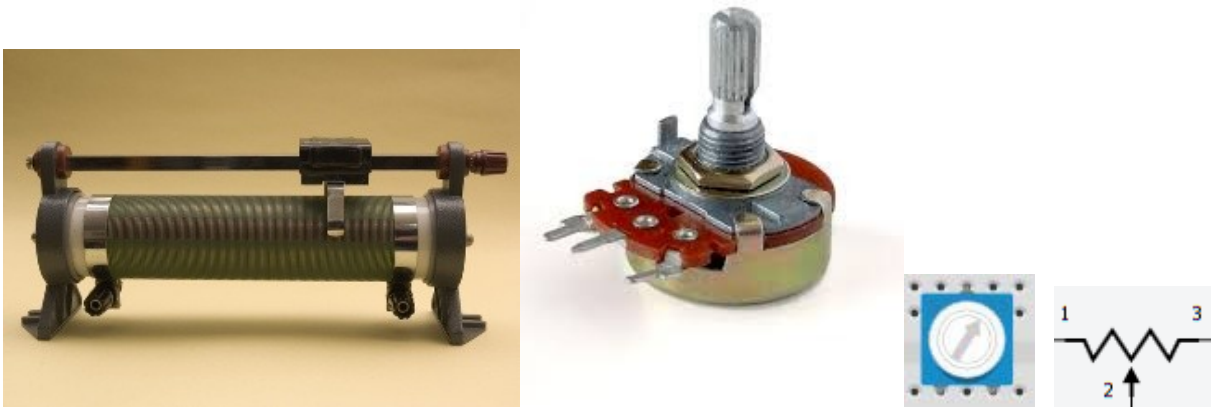


Рисунок 6.1 – Реостати

На рис. 6.1 показано кілька зображень реостатів, його позначення на електричних схемах і в моделях Autodesk CIRCUITS.

Для чого потрібен потенціометр? По-перше, виходячи з опису приладу, його можна використовувати для задання необхідного опору в разі, якщо немає відповідного резистора або потрібно відрегулювати або настроїти вже готовий пристрій. По-друге, за допомогою потенціометра можна зробити регульований дільник напруги – пристрій, який дозволяє передавати на навантаження тільки частину напруги від джерела. У цьому розділі ми використовуватимемо реостат саме в ролі дільника напруги, варіюючи напругу на одному з аналогових входів

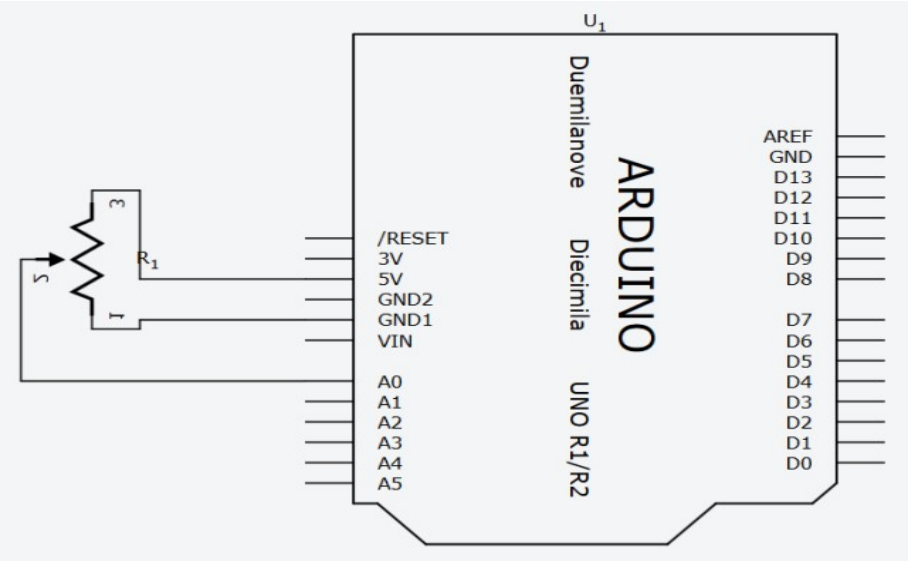
Arduino. Такий реостат є найпростішим аналоговим датчиком, який повідомляє кут повороту навколо своєї осі.

Під час роботи з кнопками ми вже ознайомилися з функцією digitalRead, яка вміє зчитувати цифровий сигнал з певного виводу контролера. У цій функції існує аналогова версія analogRead, яка може робити те ж саме, але тільки для аналогового сигналу:

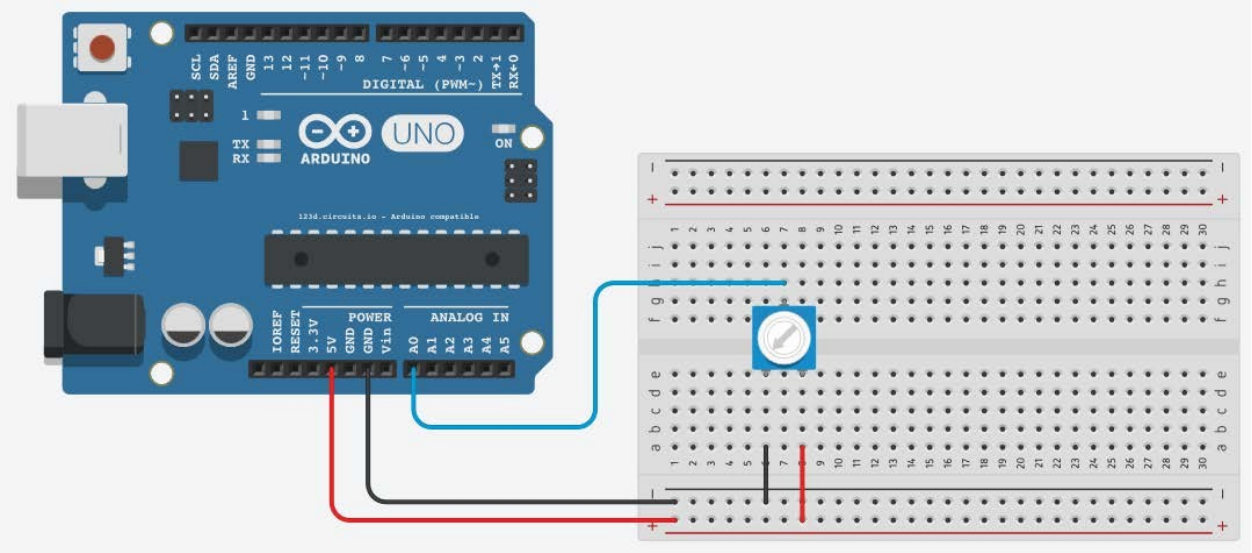
результат = analogRead (номер_контакта);

Після її виклику в програмі змінна_результат зберігатиме рівень аналогового сигналу, лічений з відповідного контакту. При цьому, на відміну від цифрового сигналу, функція analogRead повертає число від 0 до 1023, де 0 означає напругу 0 вольт, а 1023 – 5 вольт. Проміжним напруженням відповідатимуть проміжні значення. Підключати потенціометр треба до аналогових входів A0..A5 плати Arduino, як показано на рис. 6.2. Напишемо програму, яка вимірюватиме положення потенціометра раз у півсекунди і відправляти його через послідовно вальний порт:

```
const int analog_R = A0;
 void setup ()
 {
pinMode (analog_R, INPUT);
 Serial.begin (9600);
}
void loop ()
 {
int result = analogRead (analog_R);
Serial.println (result);
delay (500);
}
```



а



б

Рисунок 6.2 – Приклад підключення реостата (*а*) і модель складеного пристрою (*б*)

Зверніть увагу, що якщо ви використовуватимете кілька аналогових датчиків, то при послідовному читанні показання з них можуть зчитуватися нестійко. Це може відбуватися тому, що датчики впливають один на одного в момент перемикання АЦП між різними входами. В такому випадку можна показання з кожного входу зчитувати двічі з затримкою між вимірами і відкидати результати першого читання.

Завдання 6.1

Розробіть схему з потенціометром і рідкокристалічним екраном, яка б виводила на екран поточне значення, що читається з потенціометра.

Завдання 6.2

Розробіть схему з потенціометром і семисегментним індикатором, яка б відображала напругу на потенціометрі у вольтах. Потрібно показувати одну цифру напруги так, щоб показання 0 на аналоговому вході відображалося як «0» на індикаторі, а тисяча двадцять три – як «5».

6.2 Аналоговий датчик температури

Поширеним варіантом датчиків з аналоговим виходом є датчики температури. Ми використовуватимемо датчик TMP36, який зображений на рис. 6.3.

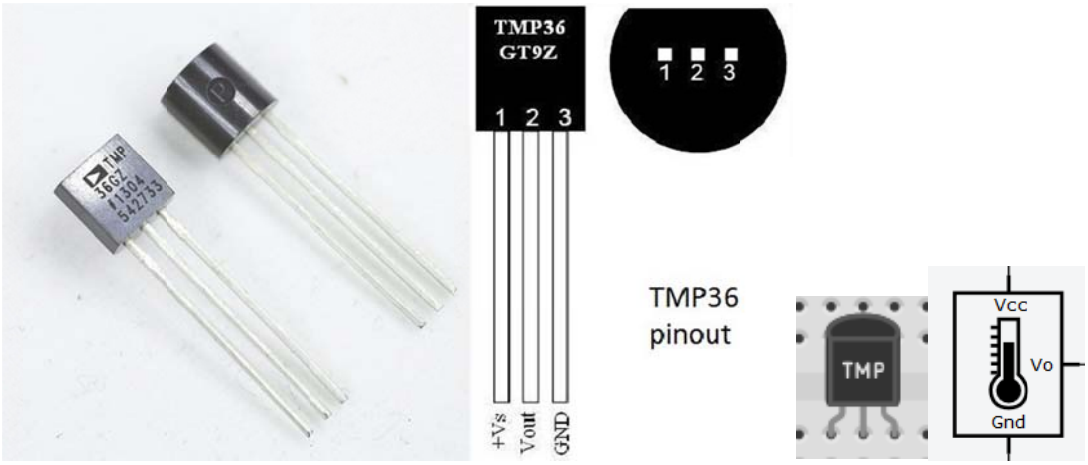
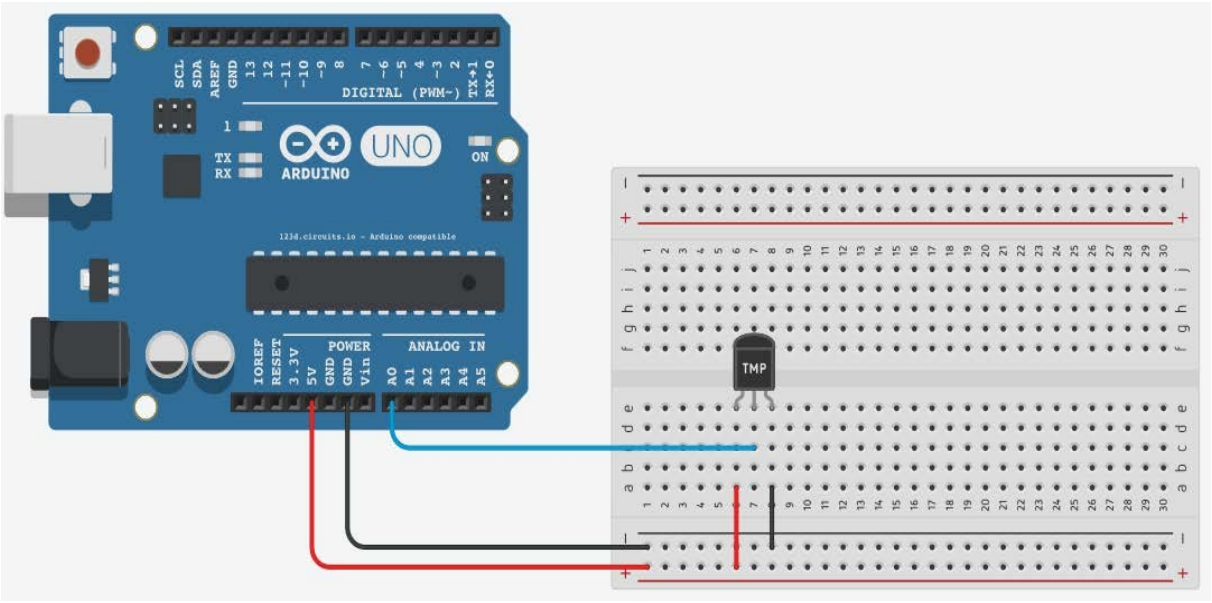


Рисунок 6.3 – Аналоговий датчик температури TMP36

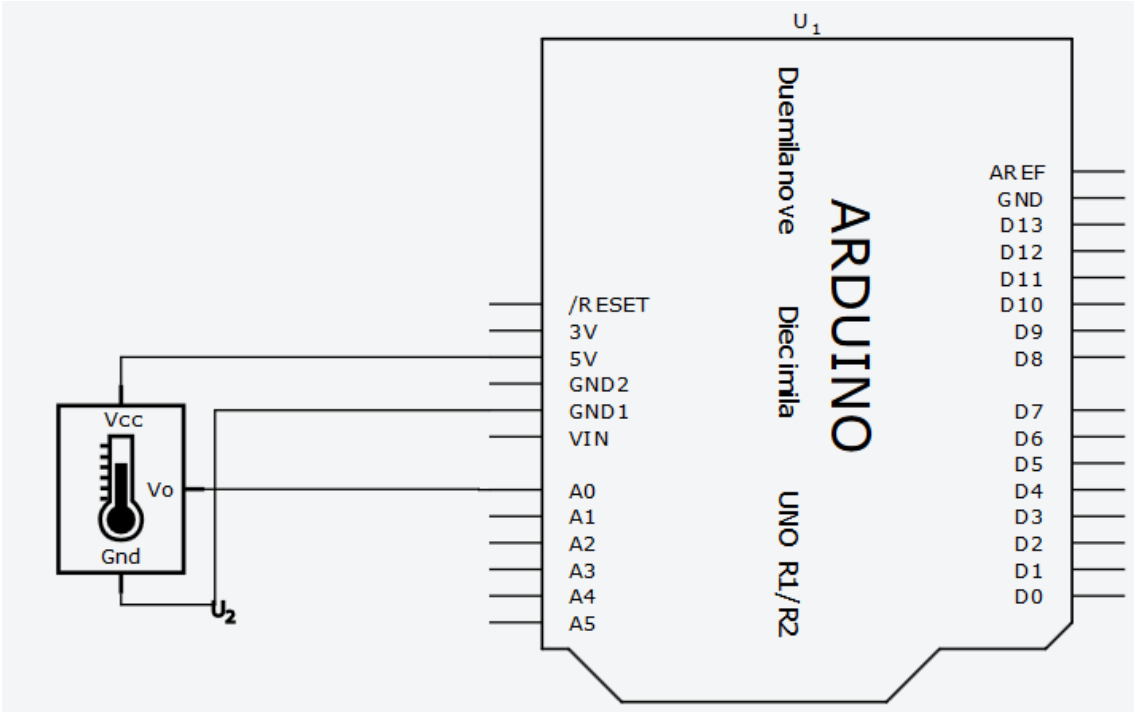
Призначення його контактів таке:

– Vs – живлення 2.7 - 5.5 В (у нашому випадку +5);
– Vout – вихідна напруга, залежна від температури;
– Gnd – земля.

Приклад підключення датчика зображений на рис. 6.4.



а



б

Рисунок 6.4 – Приклад підключення аналогового датчика температури (*а*) і вигляд на схемі (*б*)

Завдання 6.3

Реалізуйте програму, яка щосекунди виводить в послідовний порт температуру, виміряну аналоговим датчиком. Температура має виводитися в градусах Цельсія.

Рекомендація: для розрахунків використовуйте значення з плаваючою точкою (float). Для цього слід задавати константи у вигляді

«5000.0».

Завдання 6.4

Складіть схему, що включає світлодіод, зумер і датчик температури. При перевищенні температури заданого значення (наприклад, 25 С), має загорятися світлодіод і подаватися періодичні сигнали зумером.

6.3 Датчик світла

У цьому розділі ми розглянемо ще один аналоговий датчик – датчик світла, який називають фоторезистором (рис. 6.5).

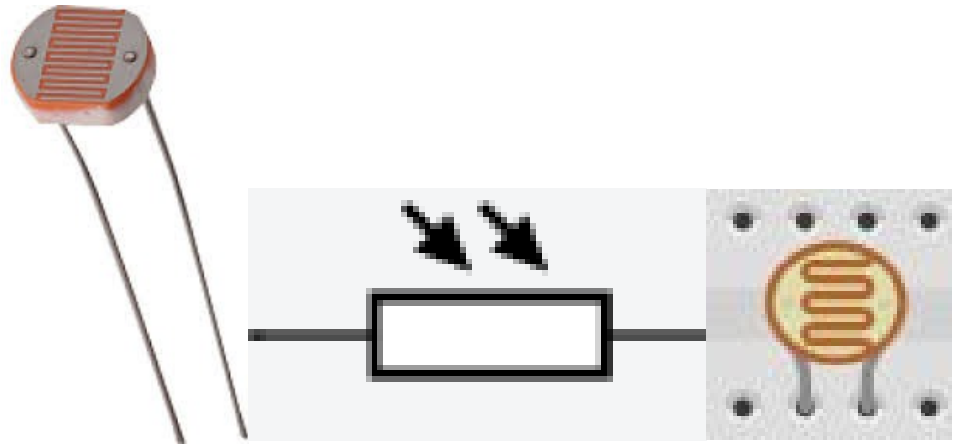
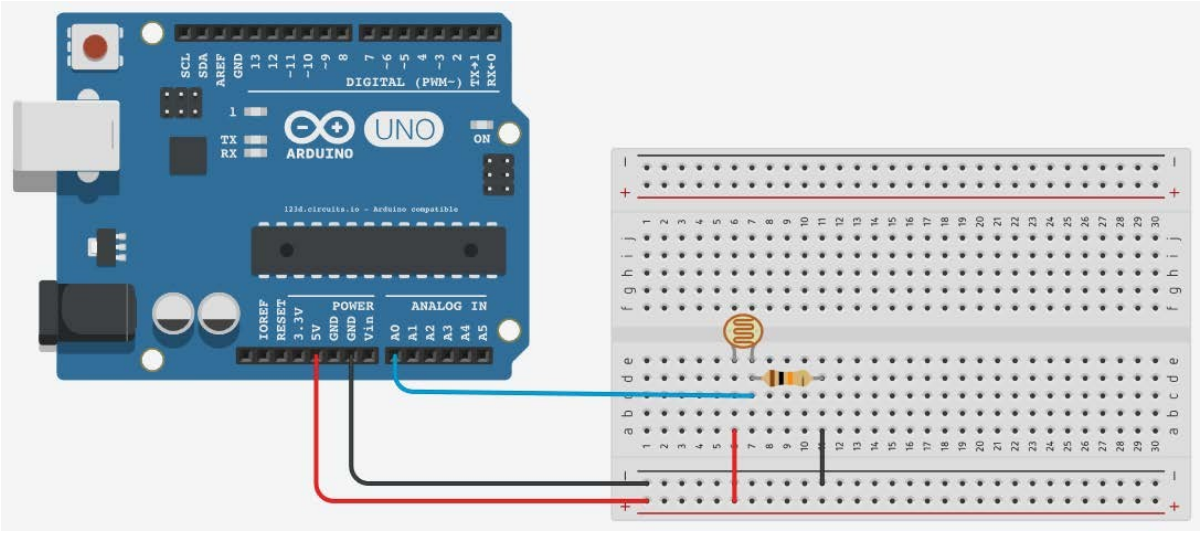


Рисунок 6.5 – Фоторезистор

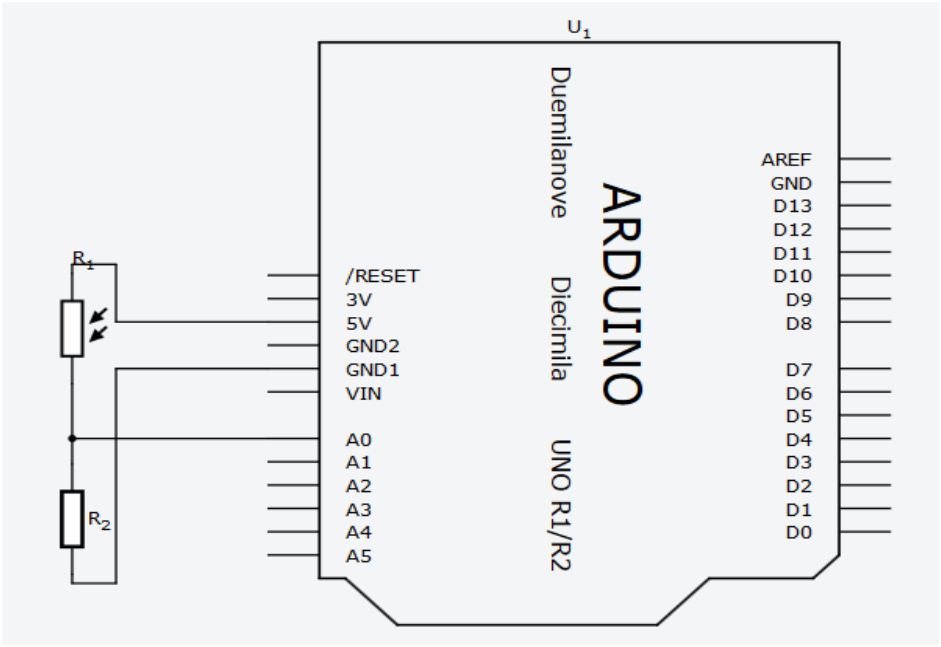
Фоторезистор – це простий компонент з двома входами, який змінює свій опір залежно від рівня освітленості. На відміну від датчика температури TMP36 фоторезистор не видає напругу, залежну від вимірюваного їм параметра. Він може тільки змінювати свій опір.

Щоб врахувати показання фоторезистора, треба спочатку перетворити зміну опору в зміну напруги. Для цього використовують схему, яка називається дільником напруги. З'єднаємо один вивід фоторезистора з напругою живлення, а до іншого підключимо звичайний резистор, з'єднаний з землею. Напруги на резисторах будуть пропорційні їхнім опорам. Наприклад, якщо опір фоторезистора 20 кОм, а звичайного резистора – 10 кОм, то напруга на фоторезисторі буде вдвічі більше, ніж на звичайному. Якщо, як в нашому випадку, напруга живлення 5 Вольт, то ці значення складуть 3.33 і 1.66 Вольта. При зміні опору фоторезистора напруги змінюватимуться. Поєднавши середню точку між резисторами з аналоговим входом плати Arduino можна вимірювати напругу на звичайному резисторі і робити висновки про освітленість.

Фоторезистори, які зазвичай йдуть в наборах з платами Arduino, як правило змінюють опір від 200 кОм (повна темрява) до 1 кОм (яскравість 10 люкс). Експериментуючи з датчиком, можна помітити, що свідчення на аналоговому вході змінюються не лінійно, це особливість роботи датчика.



а



б

Рисунок 6.6 – Приклад підключення фоторезистора (*а*) і модель пристрою (*б*)

Завдання 6.5

Реалізуйте автоматичний вимикач світла. Підключіть до плати Arduino фоторезистор і кілька світлодіодів. Запрограмуйте мікроконтролер так, щоб він включав світлодіоди при зниженні рівня освітлення і включав їх, якщо знову стає світліше.

У цьому завданні може бути розумно ввести два окремих порога освітленості – один, на якому освітлення вмикається, й інший, трохи більший, на якому вимикається, це допоможе уникнути миготіння світла в разі, якщо яскравість близька до порогової, оскільки свідчення реального датчика завжди містять невеликий шум.

7 УПРАВЛІННЯ ЕЛЕКТРОПРИВОДАМИ

7.1 Приводи. Аналогові приводи. PWM

Як нам вже відомо, мікроконтролер вміє працювати виключно з цифровими даними. Він легко може виконувати арифметичні операції над ними, приймати і передавати цифрові сигнали. Наприклад, ми можемо запалити світлодіод, подавши на нього позитивний сигнал, рівний напрузі живлення контролера. Для того щоб погасити світлодіод, просто відключимо його від живлення. Виходить, для управління ми використовуємо тільки нуль або одиницю, що і називається цифровим управлінням.

Але що робити, якщо нам потрібно запалити цей самий світлодіод тільки на половину яскравості? Або запустити двигун на 30% його потужності? Для того щоб це зробити, нам буде потрібно перетворити цифровий сигнал в якусь подобу аналогового сигналу з заданим рівнем напруги.

Для цієї мети можна використовувати спеціальний пристрій, що називається цифроаналоговим перетворювачем (ЦАП, DAC). ЦАП вміє генерувати необхідний рівень напруги, який задається мікроконтролером у цифровому вигляді. Однак, такий спосіб надлишковий для багатьох завдань. Крім того, якщо буде потрібно управляти потужним двигуном, доведеться використовувати дуже дорогий ЦАП.

Інший спосіб генерації сигналу з заданим рівнем напруги називається широтно-імпульсною модуляцією (ШІМ, PWM). Цей спосіб дозволяє обійтися чисто цифровими пристроями, що значно спрощує схему і тим самим значно здешевлює процес створення пристрою.

Ідея цього методу в тому, щоб подавати енергію на двигун (або джерело світла) не завжди, а з перервами: короткий час напруга є, а потім її немає, потім знову є і т. д. Якщо подати такий сигнал на лампу, то вона світитиметься тьмяніше, ніж якщо подавати на неї напругу постійно, адже частину часу вона

не отримує електрики. Те ж саме з двигуном – він крутитиметься повільніше. Регулюючи час подачі і відсутності напруги, можна змусити лампочку світитися з різною яскравістю або двигун крутитися з різною швидкістю.

На рис. 7.1 показано сигнали широтно-імпульсної модуляції. Одним з основних параметрів ШІМ є відсоток часу, протягом якого на вихід подається високий рівень напруги. На верхньому графіку цей відсоток дорівнює 0, і напруга завжди залишається на нульовому рівні. З таким сигналом лампочка не горітиме, а двигун стоятиме на місці. З іншого боку, сигнал із заповненням 100% робочого циклу – це постійно виведений високий рівень напруги. Він змусить підключений виконавчий механізм працювати на максимальній потужності. Проміжні значення дозволять регулювати, наскільки яскраво світитиметься лампочка або як швидко крутитиметься двигун.

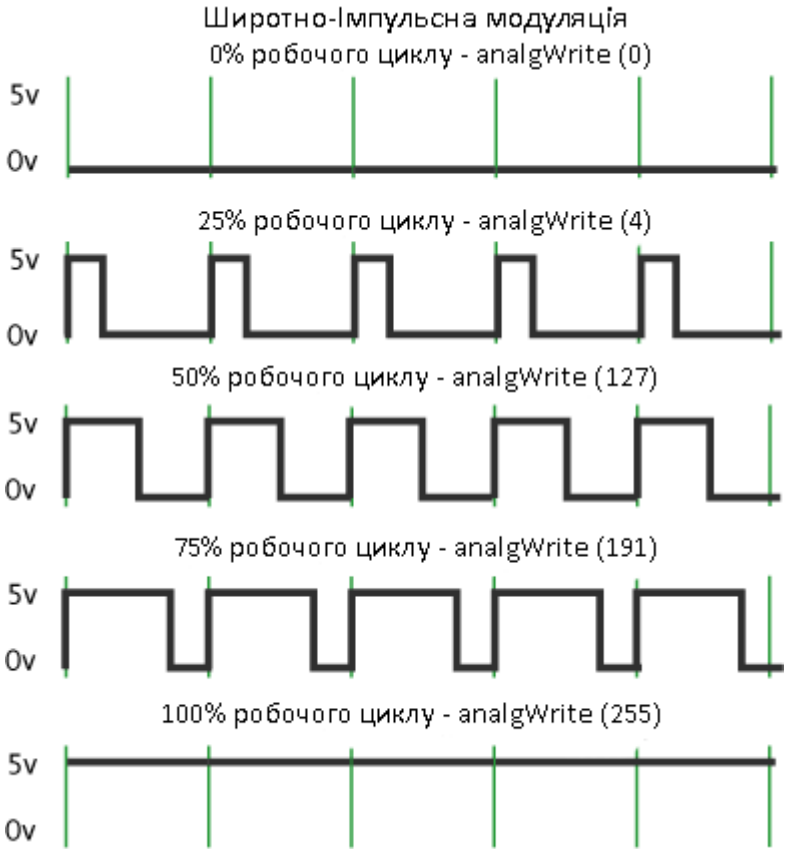


Рисунок 7.1 – Сигнали широтно-імпульсної модуляції

Інший важливий параметр ШІМ сигналу – це його частота. Вона визначає час, який займає один цикл зміни сигналу. На рис. 7.1 цей час показано зеленими лініями. Якщо ми управляємо світлодіодом і частота буде занадто низькою, то замість плавного регулювання яскравості ми побачимо миготливий світлодіод, оскільки світлодіоди дуже швидко реагують на включення і виключення живлення.

З лампою розжарювання ефект буде менш помітним, оскільки вона випромінює світло за рахунок нагріву нитки, а нитка остигає набагато повільніше, ніж вимикається світлодіод. Проте, такий ефект можна буде помітити і на звичайних лампах розжарювання, якщо сильно зменшити частоту.

В Arduino частота ШІМ дорівнює 500 кГц, що означає, що інтервал часу між двома зеленими лініями дорівнює 2 мілісекунди.

Щоб видати ШІМ сигнал на один з портів Arduino, використовують функцію analogWrite:

analogWrite (номер_контакта, рівень_сигналу);

Після виклику analogWrite на виході генеруватиметься ШІМ сигнал із заданою шириною імпульсу до наступного виклику analogWrite (або виклику digitalWrite або digitalRead на тому ж порту входу / виходу).

Рівень сигналу задається числом від 0 до 255, де 0 відповідає 0% заповнення, а 255–100%.

Необхідно враховувати, що генерація ШІМ сигналу підтримується не на всіх виходах мікроконтролера. На більшості плат Arduino (на базі мікроконтролера ATmega168 або ATmega328) ШІМ підтримують порти 3, 5, 6, 9, 10 і 11, а на платі Arduino Mega порти 2-13. На більш ранніх версіях плат Arduino analogWrite працював тільки на портах 9, 10 і 11. Виводи, які підтримують цей режим, на схемі плати позначають абревіатурою ШІМ або PWM.

Зазначимо, що функція analogWrite ніяк не пов'язана з аналоговими входами і з функцією analogRead. У даному випадку збіг є тільки в назві. Для

виклику analogWrite немає необхідності встановлювати тип входу/виходу функцією pinMode.

Розглянемо програму, яка збільшить яскравість світлодіода. Ця програма використовується з такою ж схемою, яка використовувалася в найпершому прикладі підключення світлодіода.

```
int led = 2;

void setup () {
analogWrite (led, 0);
delay (1000);
analogWrite (led, 32);
delay (1000);
analogWrite (led, 64);
delay (1000);
analogWrite (led, 128);
delay (1000);
analogWrite ( led, 255);
delay (1000);
}

void loop () {
}
```

Зверніть увагу, що емулятор Autodesk CIRCUITS не вказує включення світлодіода на низьких рівнях заповнення ШІМ сигналу.

Завдання 7.1

Реалізуйте програму, яка забезпечуватиме плавне загоряння, а потім плавне загасання світлодіода з періодом в одну секунду.

Завдання 7.2

Складіть схему з світлодіодом і потенціометром та реалізуйте програму, яка регулюватиме яскравість світлодіода на підставі положення потенціометра.

7.2 Управління електричним двигуном

Нам вже відомо, що мікроконтролер-пристрій, який оперує малими струмами і напругами. Так більшість сучасних мікроконтролерів працюють з напругою 5 Вольт, 3.3 Вольта або ще менше. При цьому сила струму, що проходить через виводи мікроконтролерів, як правило, не має перевищувати 20...40 міліампер.

Такого струму достатньо, щоб запалити світлодіод або включити зумер. Однак для управління електричним двигуном потужності мікроконтролера недостатньо. Навіть для невеликого двигуна потрібна напруга 5–6 Вольт і сила струму в сотні міліампер. Таке навантаження неодмінно знищить будь-який мікроконтролер за долі секунди.

Тому для управління двигуном потрібно використовувати додаткові пристрої, які б дозволили оперувати великими струмами, що проходять через двигун. У найпростішому випадку в ролі такого ключа можна використовувати потужні транзистори. Однак зручніше використовувати спеціальні мікросхеми – драйвери двигунів. Такі мікросхеми містять всі необхідні компоненти для

комутації струму і дозволяють керувати напрямком і швидкістю обертання двигуна.

Ми використовуватимемо мікросхему L293D (рис. 7.2), яка дозволяє управляти відразу двома двигунами з напругою живлення від 4.5 до 36 Вольт і постійним струмом до 600 мА. Цього достатньо, щоб управляти двигунами невеликого робота.

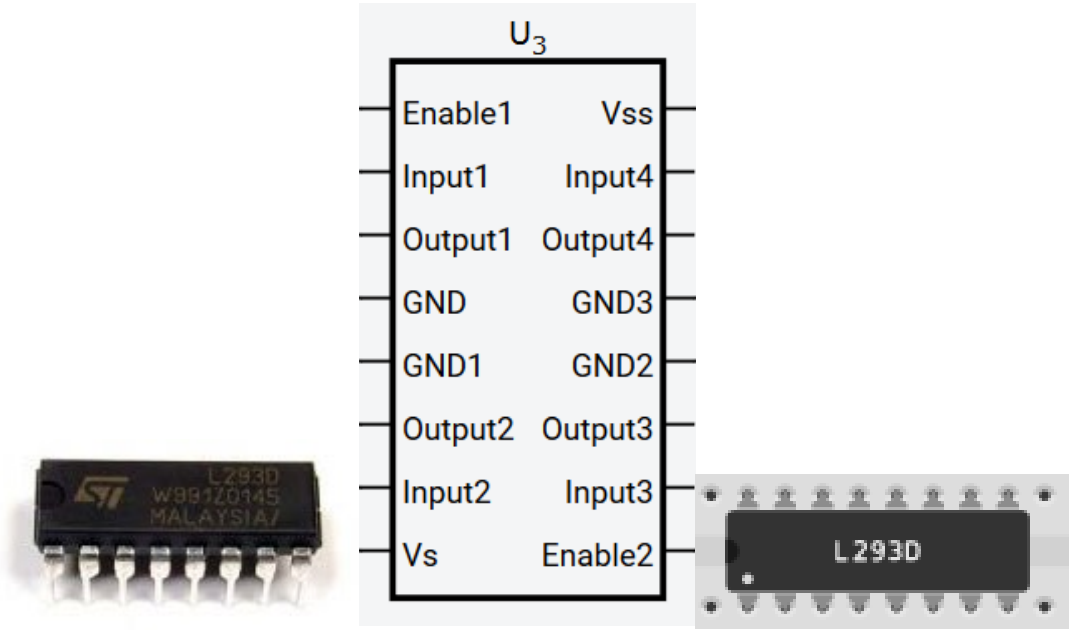


Рисунок 7.2 – Мікросхема L293D

Мікросхема дозволяє управляти відразу двома двигунами, при цьому її виходи розподілені так, що з одного боку знаходиться управління одним двигуном, а з іншого – іншим. Їхня робота абсолютно ідентична, тому ми розглянемо тільки одну її сторону.

Безсумнівним плюсом даної мікросхеми є роздільне живлення її логічної частини, напруга якого лежить в межах 4.5–5 Вольт (контакт VSS), і силової частини живлення двигунів (контакт VS). Справа в тому, що потужні двигуни створюють завади в ланцюгах живлення, які можуть призводити до збоїв і перезапуску мікроконтролера. Тому бажано розділяти живлення мікро-контролера і двигунів.

Розберемо тепер виводи для управління першим (лівим на схемі) двигуном. Висновок ENABLE1 – це головний вхід управління лівим каналом.

Без сигналу одиниці (або ШИМ) на ньому, двигун не працюватиме незалежно від того, що подається на входи INPUT1 і INPUT2.

Виводи INPUT1 і INPUT2 задають напрямок обертання мотора. Щоб змусити двигун обертатися в одну сторону, треба подати логічну одиницю на висновок INPUT1, а на INPUT2 подати логічний нуль. Для зміни напрямку потрібно поміняти місцями: на INPUT1 подати 0, а на INPUT2 - 1.

При поданні однакових сигналів на INPUT1 і INPUT2 двигун не обертатиметься, отже, обертання можна зупинити або подачею логічного нуля на висновок ENABLE1 при будь-якій конфігурації INPUT1 і INPUT2, або однаковими сигналами на INPUT1 і INPUT2, не змінюючи рівня сигналу на виводі ENABLE1.

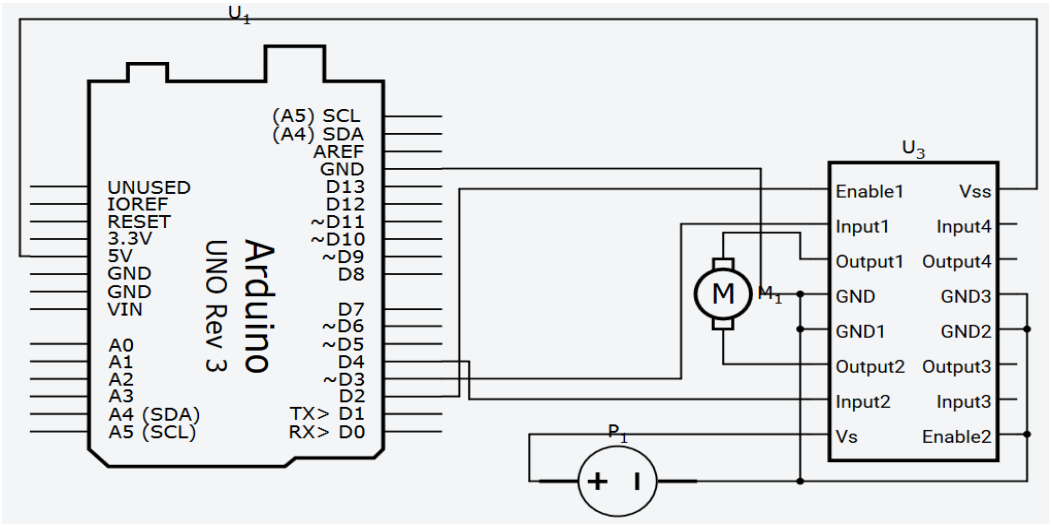
Виводи GND з'єднуються з негативним полюсом джерела живлення (земля). Решта виводів OUTPUT1 і OUTPUT2 служать безпосередньо для підключення мотора.

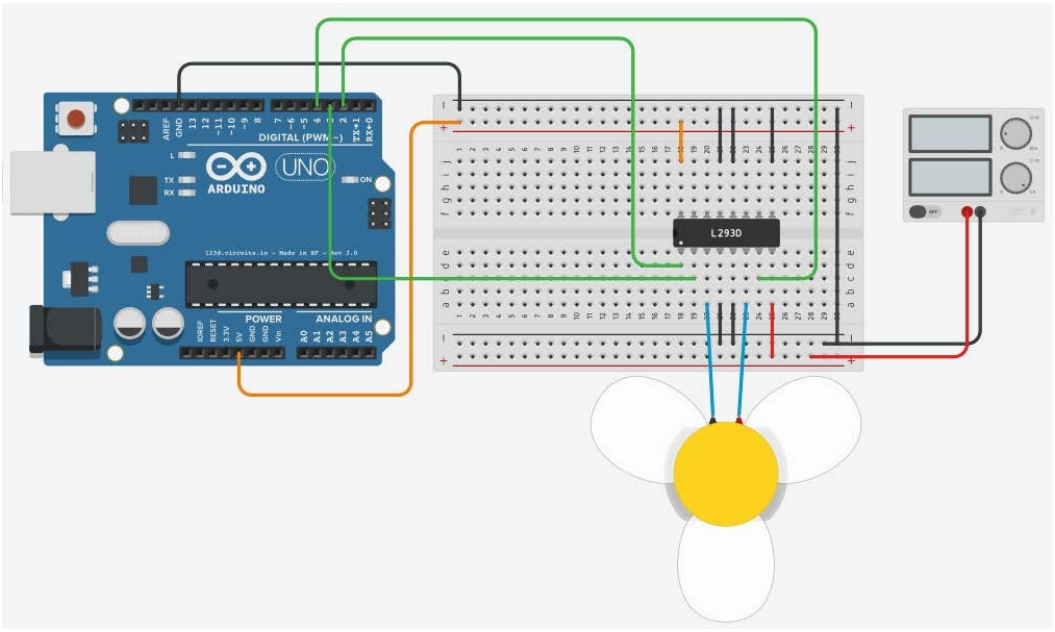
Складемо схему, в якій підключимо один двигун за допомогою мікросхеми L392D (рис. 7.3). Зверніть увагу, що для двигуна використовується окреме джерело живлення, при цьому «земля» обох джерел з'єднана. Вхід ENABLE2, керуючий другим мотором, відключений завдяки підключенню його до землі, оскільки в цій схемі другий двигун не використовується.

Наступна програма демонструє включення двигуна за допомогою подачі ШІМ сигналу на вхід ENABLE1:

```
const int m_enable = 2; // вихід сигналу ENABLE1
const int m_dir1 = 3; // вихід вибору напрямку INPUT1
const int m_dir2 = 4; // вихід вибору напрямку INPUT2

void setup () {
// конфігурація контактів на режим виходів
pinMode (m_enable, OUTPUT);
pinMode (m_dir1, OUTPUT);
```

pinMode (m_dir2, OUTPUT);





а

б

Рисунок 7.3 – Приклад підключення мікросхеми L293D

// вибір напрямку обертання

digitalWrite (m_dir1, HIGH);

digitalWrite (m_dir2, LOW);

```
// управління обертанням за допомогою
PWM analogWrite (m_enable, 150);
}
void loop () {
}
```

При різкому включенні двигунів мікросхема L293D починає дуже сильно нагріватися через те, що при старті працює відразу на повну потужність, для того щоб зрушити з місця, двигун споживає струм, хоч і короткочасно, але мінімум у 2–5 разів більше установленого робочого значення. При різкій зміні напряму обертання з'являється ще більший стрибок струму, оскільки доводиться не тільки зрушити якір з місця, як це було в першому випадку, але і побороти його інерцію руху в іншому напрямку.

Для зниження цих факторів слід розганяти двигуни плавно – чим довше, тим краще (в розумних межах). При зміні напрямку обертання слід дати проміжок часу для самостійної зупинки двигуна (частки секунди), а потім знову плавно розганяти. Приклавши палець до мікросхеми, можна відчути відмінністть. Експериментуйте і знайдіть золоту середину між нагріванням і швидкістю реагування для вашого завдання.

Завдання 7.3

Змініть програму, написану вище, щоб двигун поперемінно обертався то в одну сторону, то в іншу. Період зміни напряму – п'ять секунд.

Завдання 7.4

Реалізуйте плавне управління швидкістю роботи двигуна. При включенні двигун повинен почати плавно збільшувати обороти і за 5 секунд досягти

максимальної швидкості. Потім він повинен також плавно зупинитися і почати обертатися в іншу сторону.

Завдання 7. 5

Реалізуйте схему для управління рухом робота, яка має забезпечити управління двома двигунами і містити 4 кнопки, натисканням на які визначається напрямок руху:

– перша кнопка – рухатися вперед (обидва мотори вперед);
– друга – поворот направо (правий мотор вперед, лівий – назад);
– третя – поворот наліво (лівий мотор вперед, правий назад);
– четверта – рухатися назад (обидва мотори обертаються назад).

7.3 Управління сервоприводами

Ще одним електромеханічним виконавчим механізмом, часто застосовується в робототехніці і моделюванні, є серводвигун (сервопривід, сервомашинка), зображений на рис. 7.4.

На відміну від звичайних двигунів, серводвигун вміє повертати вал суворо на заданий кут. Цю корисну властивість сервомашинок часто використовують в авіамоделюванні для управління елеронами, кермом висоти тощо. Потужні серводвигуни можна використовувати для створення механічних рук-маніпуляторів.

Влаштований такий двигун досить складно. У верхній частині приладу розміщується шестерінчастий редуктор, який дозволяє значно збільшити крутний момент двигуна постійного струму за рахунок зниження швидкості його обертання. Нижче розташований потенціометр, завдання якого – визначати, на який кут повернути вихідний вал редуктора. Нарешті, в глибині

корпусу знаходиться невелика плата управління, яка і робить серводвигун розумним. Ця плата постійно відстежує поточний стан вала і коригує його в разі, якщо вал намагається піти з заданої позиції.

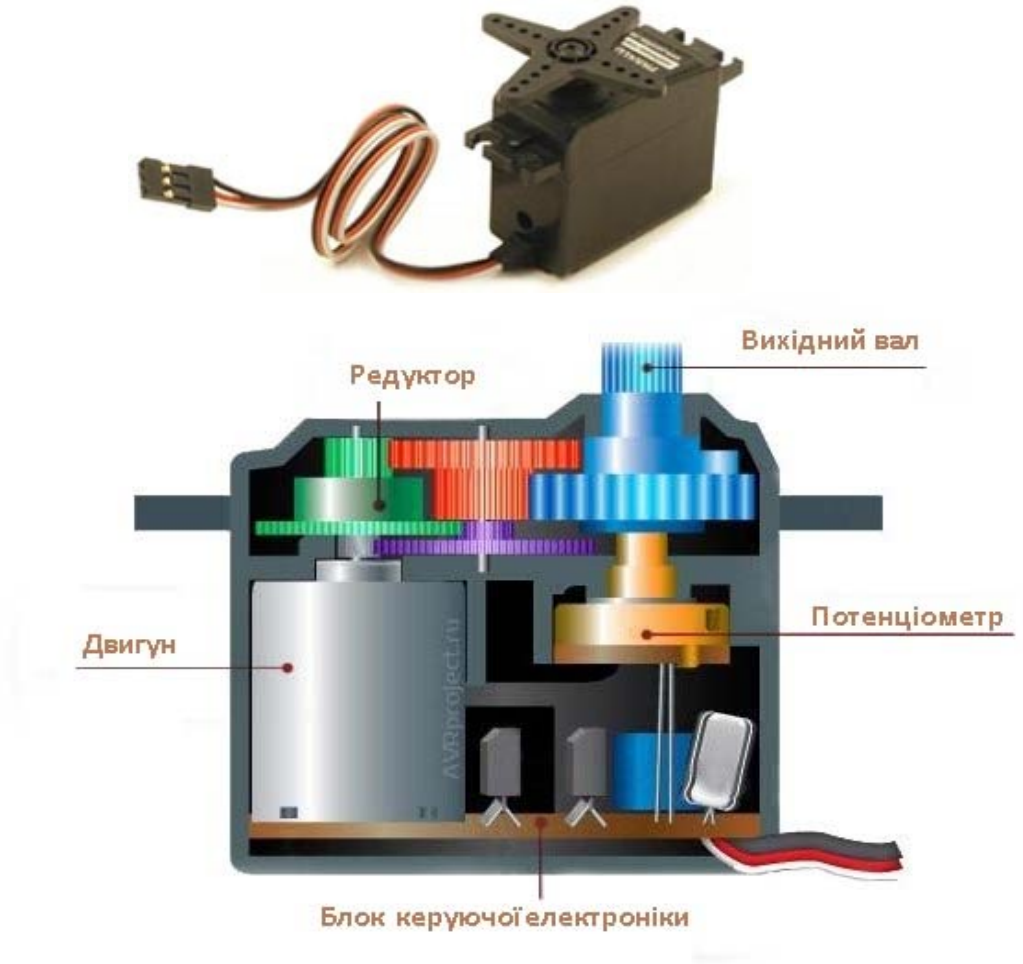


Рисунок 7.4 – Серводвигун

Управляється серводвигун за допомогою ШІМ сигналу. Але, на відміну від звичайного двигуна постійного струму, тут рівень ШІМ задає не швидкість обертання, а кут повороту. Крім того, оскільки серводвигун містить всю необхідну силову електроніку, керувати ним можна безпосередньо з порту мікроконтролера.

В Arduino IDE є спеціальна бібліотека для роботи з серводвигунами – Servo. Для підключення двигуна необхідно створити відповідний об'єкт:

Servo myservo;

Потім у функції setup виконати його ініціалізацію:

myservo.attach (номер_контакта);

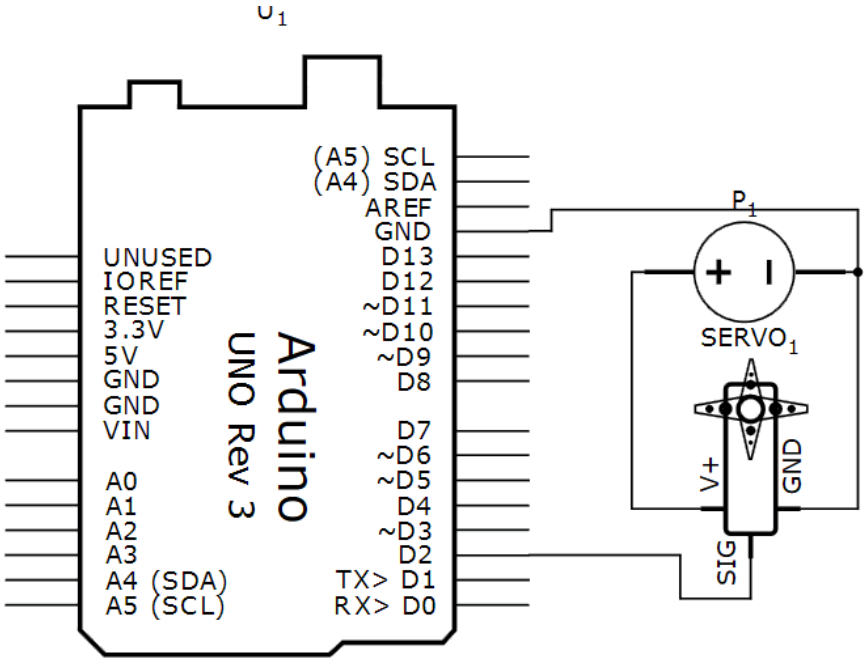
Тут номер_контакта – номер ніжки мікроконтролера, до якого підключений керуючий вхід серводвигуна. Цей контакт має підтримувати роботу в режимі PWM. Поворот вала на кут здійснюється функцією write:

myservo.write (кут);

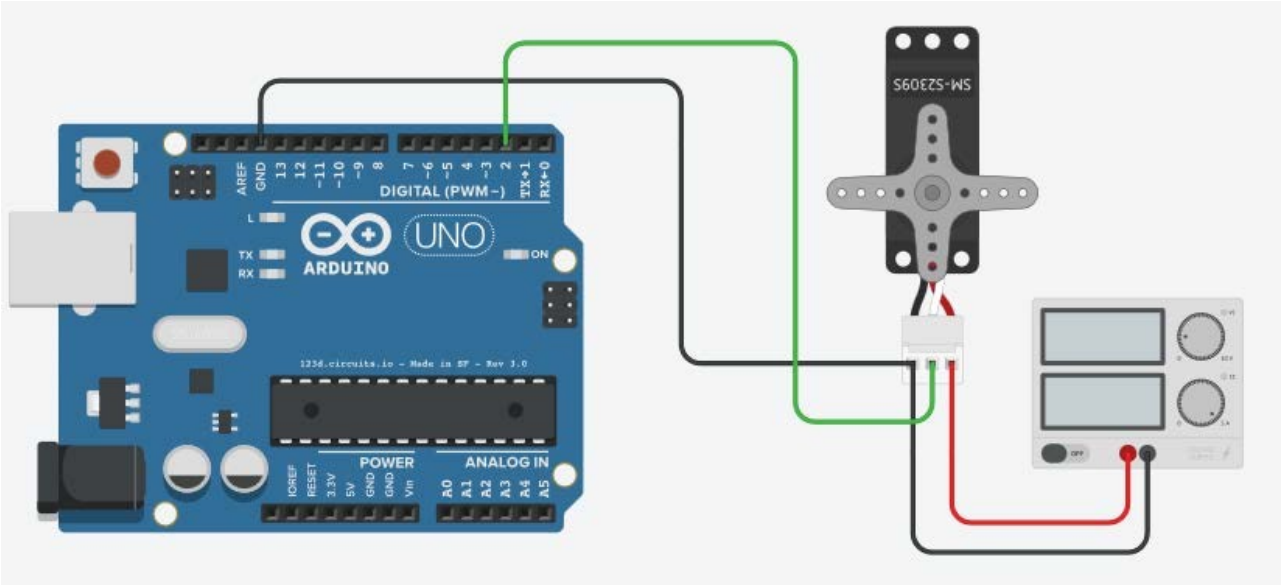
Кут задається в градусах і може приймати значення від 90° до +90° залежно від моделі серводвигуна.

Схема підключення серводвигуна показана на рис 7.5. У даному випадку теж краще використовувати окреме живлення для приводу.

```
void setup () {
// повідомляємо, куди підключений привід
myservo.attach (servo_pwm);
// повертаємо вал приводу на +90 градусів
myservo.write (90);
}

void loop () {
}
```



а



б

Рисунок 7.5 – Приклад підключення серводвигуна (*а*) і складена модель пристрою (*б*)

Нижче наведено приклад програми, яка демонструє настройку бібліотеки Servo і поворот осі приводу на 90 градусів:

```
#include <Servo.h>

Servo myservo;
int servo_pwm = 2;
```

Завдання 7.6

Реалізуйте програму, що забезпечує плавний поворот серводвигуна між крайніми положеннями за 5 секунд.

Завдання 7.7

Складіть схему з серводвигуном і датчиком температури. Реалізуйте стрілковий термометр – серводвигун має займати положення, що задається показанням термодатчика.

8 I²C І БІБЛІОТЕКА WIRE

8.1 Шина I²C

I²C це скорочення від «Inter-Integrated Circuit». У далеких 1970-х роках підрозділ з виробництва напівпровідникових компонентів фірми Philips (тепер це фірма NXP) зрозуміла необхідність розробки і стандартизації простого протоколу для передачі даних між мікросхемами. В результаті вони розробили протокол, який зараз називають I²C або TWI (Two-Wire Interface). Цей протокол дозволяє обмінюватися інформацією, використовуючи тільки два дроти, що позначаються SDA і SCL (рис. 8.1). По першому передаються дані, а другий використовується для сигналу синхронізації.

Шина I²C (рис. 8.1) дозволяє з'єднувати відразу кілька пристроїв. У мережі має бути хоча б один провідний пристрій (Master), який ініціалізує передачу даних і генерує сигнали синхронізації. Інші пристрої є веденими (Slave), вони передають дані за запитом ведучого. У кожного відомого пристрою є унікальна адреса, за якою ведучий і звертається до нього. Адреса пристрою вказується в документації. До однієї шини I²C може бути підключено до 127 пристроїв, у тому числі кілька ведучих. До шини можна підключати пристрої в процесі роботи, вона підтримує так зване «гаряче підключення».

Як видно з рис. 8.1, лінії SCL і SDA мають бути з'єднані з живленням VCC через резистори. Це необхідно для нормальної роботи I²C. На платах Arduino (точніше, на встановлених у них мікроконтролерах) такі резистори вже присутні, тому у нас не буде необхідності додавати їх у схему.

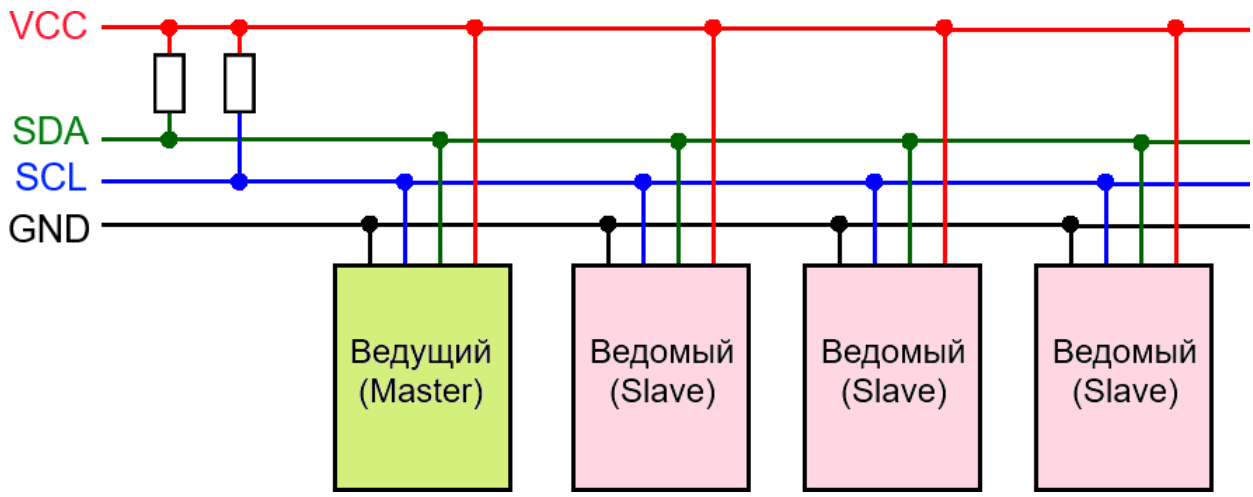


Рисунок 8.1 – Шина I^2C

Arduino використовує для роботи по інтерфейсу I^2C два порти, які можуть перебувати на різних платах у різних місцях. Найбільш поширений варіант, коли аналоговий порт A4 відповідає SDA, аналоговий порт A5 відповідає SCL. Така схема використовується, наприклад, в Arduino UNO і Arduino Nano. Для інших моделей розташування цих портів показано в таблиці 8.1.

Таблиця 8.1 – Розташування портів SDA і SCL

Плата	**Пін SDA**	**Пін SCL**
Arduino Uno, Nano, Pro і Pro Mini	A4	A5
Arduino Mega	20	21
Arduino Leonardo	2	3
Arduino Due	20, SDA1	21, SCL1

Для роботи з протоколом I^2C у Arduino є штатна бібліотека Wire, що дозволяє взаємодіяти з I^2C/TWI-пристроями як в режимі майстра, так і в режимі

веденого. Для ініціалізації бібліотеки використовується функція begin, яка може викликатися в двох варіантах:

Wire.begin ();

або

Wire.begin (адреса);

Тут адреса – цифрова адреса відомого пристрою, якщо Arduino підключається до шини в ролі відомого. Якщо адресу не вказувати, то Arduino працюватиме як майстер.

Для того щоб майстер міг передати відомому дані, він повинен викликати три функції. Спочатку він викликає функцію

beginTransmission (адреса);

аргументом якої є номер веденого. Потім він передає самі дані:

write (значення);

Цю функцію можна викликати кілька разів. У кінці передачі майстер викликає функцію endTransmission, у момент виклику якої якраз і відбувається реальна відправка всіх даних.

Наприклад, щоб відправити пристрою з адресою 55 байт val, треба виконати такий код:

```
Wire.beginTransmission (55);  // передача для пристрою 55
Wire.write (val);                    // відправка байта val
Wire.endTransmission ();         // завершення передачі
```

Цей майстер може запросити відомого передати йому дані. Для цього майстер спочатку викликає функцію requestFrom (адреса, розмір); якої передає адресу веденого і число байт, які очікує прийняти. Після цього майстер може отримати по черзі всі байти відповіді, викликаючи функцію read. Функція available повертає число байт, які доступні для читання.

Таким чином, звичайний код читання даних від веденого виглядає таким чином:

```
// запитуємо 6 байт від веденого # 55
Wire.requestFrom (55, 6);
// ведений може надіслати менше, ніж просили
while (Wire.available ()) {
// отримуємо один байт як символ
char c = Wire.read ();
// відправляємо прийнятий байт на ПК
Serial.print (c);
}
```

Робота з бібліотекою Wire в режимі веденого відбувається іншим чином. Майстер знає, коли він хоче щось передати або отримати і він може просто викликати необхідні функції. Однак ведений заздалегідь не знає, в який момент він отримає дані або запит від майстра.

Тому, під час написання програми для веденого необхідно спочатку написати функцію, яка реагуватиме на дії майстра, а потім зареєструвати її в бібліотеці Wire. Бібліотека сама викличе нашу функцію в той момент, коли зафіксує дії майстра.

Таких функцій може бути дві. Одна використовується для прийому даних (якщо він передав їх за допомогою beginTransmission / send / endTransmission), а друга – якщо майстер запросив дані за допомогою requestFrom. Такі функції називаються функціями-оброблювачами.

Розглянемо спочатку перший варіант. Насамперед треба написати функцію, яка прийматиме дані. Ця функція може мати будь-яке ім'я, але вона має отримувати один аргумент – число байт, що передаються майстром, і не повинна повертати ніяких значень. Всередині неї можна читати дані за допомогою функцій available і read, наприклад:

```
void receiveEvent (int howMany) { // ім'я може бути іншим
while (Wire.available ()) {// поки є дані
char c = Wire.read (); // читаємо один байт
Serial.print (c); // передаємо його на комп'ютер


}
}
```

Оброблювач слід зареєструвати у функції setup програми за допомогою onRecieve таким чином:

```
// треба вказати ім'я оброблювача
Wire.onReceive (receiveEvent);
```

Другий варіант, коли ведений повинен відправити дані за запитом майстра, реалізується схожим чином. Спочатку потрібно написати оброблювач, який в даному випадку матиме порожній список параметрів. Всередині нього досить просто викликати функцію send для відправки даних. Бажано відправити стільки байт, скільки від нас очікує майстер:

```
void requestEvent () {
Wire.write ( "data12"); // відповідь довжиною 6 байт
}
```

Реєструється такий оброблювач запиту за допомогою функції

onRequest: Wire.onRequest (requestEvent);

Сьогодні випускається безліч пристроїв, що працюють по шині I^2C. Це можуть бути цифрові датчики температури, мікросхеми пам'яті, електронні компаси, рідкокристалічні дисплеї, мікросхеми – радіоприймачі та багато іншого. Всі ці компоненти можна підключити до мікроконтролера, використовуючи всього два виводи, що дуже зручно. Для багатьох з них є

бібліотеки для Arduino, які зроблять їх використання ще більш легким, ніж робота через бібліотеку Wire.

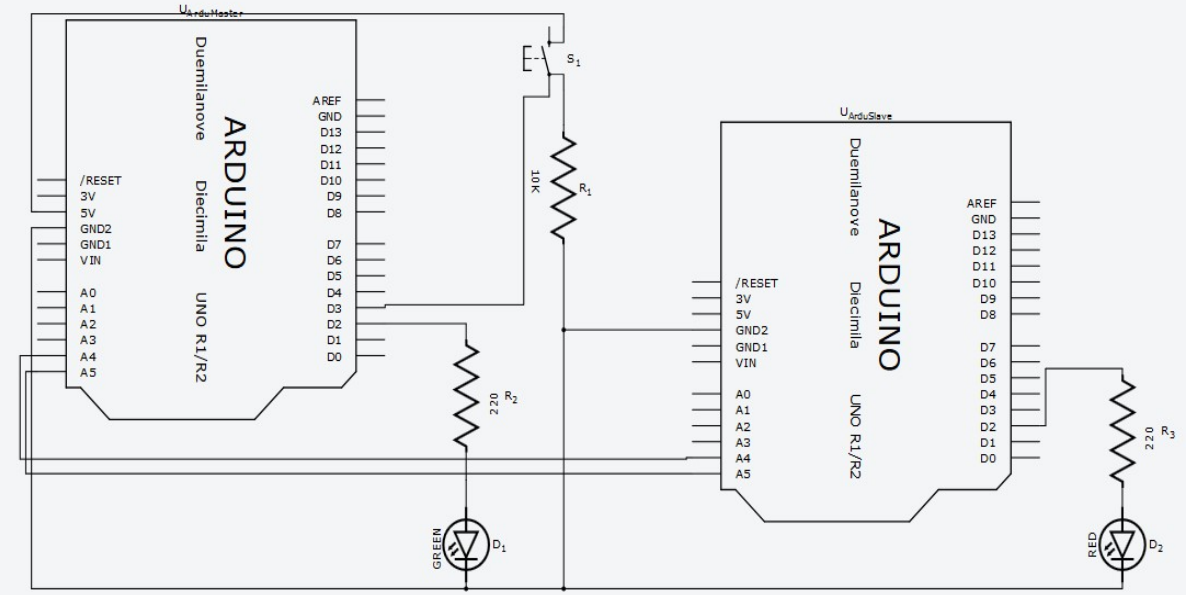
8.2 Приклад використання I^2C в Autodesk CIRCUITS

На жаль, в емуляторі Autodesk CIRCUITS немає моделей для I^2C пристроїв. Тому в нашому навчальному посібнику ми використовуватимемо цю шину для організації зв'язку між двома платами Arduino. Це теж дуже цікава і корисна можливість, яка дозволить продемонструвати роботу як в режимі майстра, так і в режимі веденого.

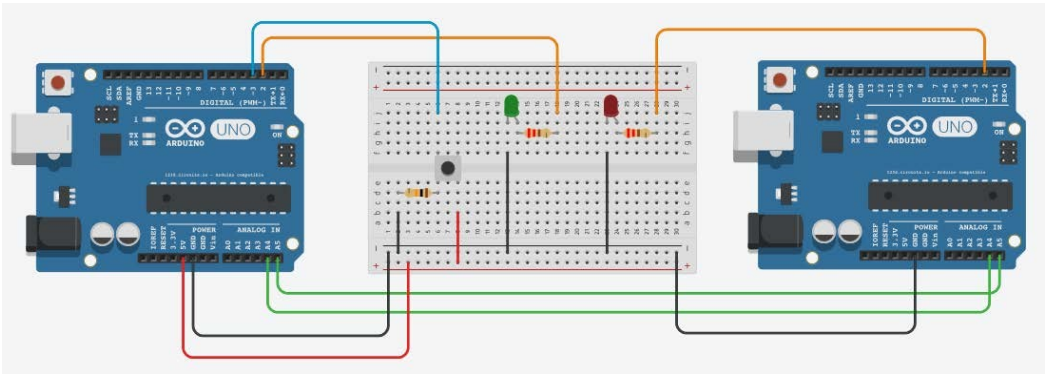
Ми використовуватимемо просту схему, показану на рис. 8.2. У цій схемі є дві плати Arduino, до яких підключено по світлодіоду. До однієї з плат, яка виступає в ролі майстра, підключена кнопка. Обидві плати з'єднані шиною I^2C через виводи A4 і A5. Земля живлення плат також об'єднана. Оскільки в платах Arduino використовуються вбудовані резистори для шини I^2C, то немає необхідності з'єднувати лінії шини з живленням окремими резисторами.

Під час натискання на кнопку майстер запалюватиме свій світлодіод, і відправлятиме команду, що складається з одного символу «H» відомому. При відпуску кнопки майстер вимикає світлодіод і відправляє команду «L». Ведений запалюватиме і вимикатиме свій світлодіод по командах від майстра.

В ході програмування двох плат у середовищі Autodesk CIRCUITS потрібно вибирати плату, для якої пишеться програма, за допомогою списку, що знаходиться в лівій верхній частині редактора коду (див. рис. 8.3).



а



б

Рисунок 8.2 – Схема підключення шини I2C (*а*) і зібрана модель пристрою(*б*)

Розглянемо код для майстра:

```
#include <Wire.h>

const int SLAVE_ADDR = 5;     // адреса веденого на шині I2C
const int PIN_BUTTON = 3;     // вивід з кнопкою
const int PIN_LED = 2;        // вивід із світлодіодом
const char BTN_PRESSED = 'H';     // повідомлення увімкнути діод
const char BTN_RELEASED = 'L';    // повідомлення вимкнути діод
```

```
int buttonState = 0;                    // стан кнопки
void setup () {
Wire.begin (); // ця плата налаштовується майстром
pinMode (PIN_BUTTON, INPUT); // настройка виведення для кнопки
pinMode (PIN_LED, OUTPUT); // настройка виведення для діода
}

void loop () {
// читаємо стан кнопки
buttonState = digitalRead (PIN_BUTTON);

if (buttonState == HIGH) {
// вмикаємо свій діод
digitalWrite (PIN_LED, HIGH);
// посилаємо команду відомому
Wire.beginTransmission (SLAVE_ADDR);
Wire.write (BTN_PRESSED);
Wire.endTransmission ();
} else {
// вимикаємо діод
digitalWrite (PIN_LED, LOW);
// посилаємо команду відомому
Wire.beginTransmission (SLAVE_ADDR);
Wire.write (BTN_RELEASED);
Wire.endTransmission ();
}
delay (30);
}
```

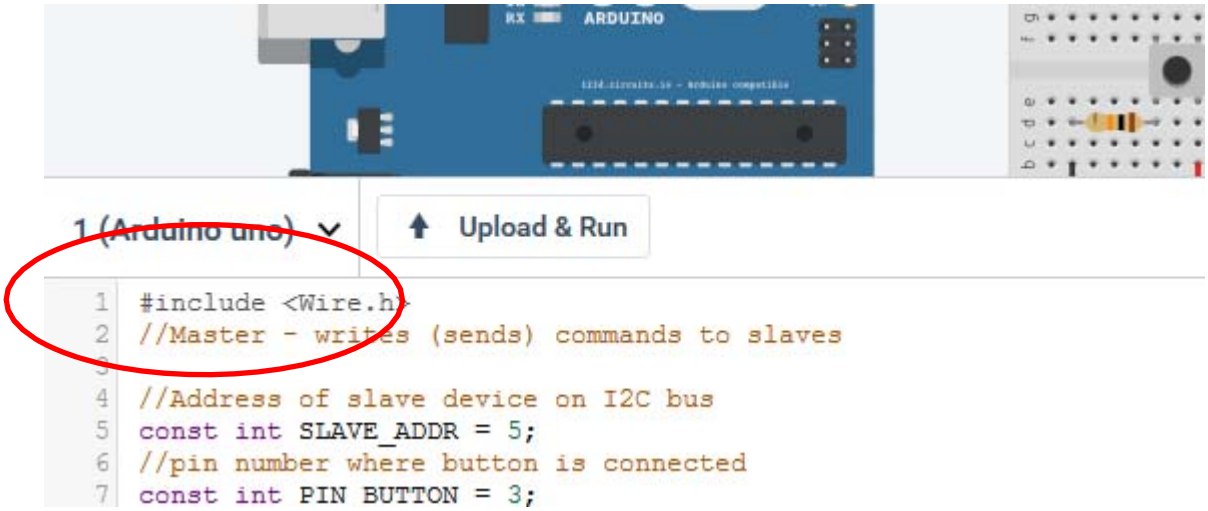


Рисунок 8.3 – Вибір плати в середовищі Autodesk CIRCUITS

Тепер розглянемо код веденого:

```
#include <Wire.h>

const int LED = 2;                  // вивід з світлодіодом
const int SLAVE_ADDR = 5;     // адреса веденого на шині I2C
void setup () {
Wire.begin (SLAVE_ADDR); // ця плата буде веденою
Wire.onReceive (receiveEvent); // реєстрація обробника
pinMode (LED, OUTPUT); // налаштовуємо вивід світлодіода
digitalWrite (LED, LOW);
}

void loop () {
// Тут нічого не відбувається.
// Вся робота в обробнику далі.
}

// Ця функція викликається, коли приходять дані по I2C
```

```
void receiveEvent (int howMany) {
// цикл читання даних
while (Wire.available ()) {
char c = Wire.read ();                  // читаємо один байт
if (c == 'H')                           // розбираємо команду
digitalWrite (LED, HIGH); // включаємо діод
else if (c == 'L')
digitalWrite (LED, LOW);                        // вимикаємо діод
}
}
```

Завдання 8.1

Складіть схему, в якій дві плати Arduino з трьома світлодіодами (червоний, жовтий і зелений) були з'єднані шиною I^2C. Реалізуйте роботу в режимі світлофора – одна плата має показувати сигнали для однієї дороги, а інша – для перпендикулярної (якщо на одній платі червоний, на другий – зелений і навпаки). Реалізуйте управління одного світлофора іншим через відправку команд по шині I^2C. Обмін має проводитися в режимі «майстер пише, ведений читає».

Завдання 8.2

Складіть схему з двох плат Arduino. До однієї плати підключений рідкокристалічний екран, а до іншої – датчик температури. Плати з'єднані по шині I^2C. Плата з екраном виступатиме в ролі майстра, який запитує дані (температуру) у веденого і відображає її на екрані.

Відповіді до запитань для самоконтролю

1.1

1) 1, 3

2) 1

1.2

1) 2, 4, 6, 7

2) ППВМ

1.3

1) 1, 4

2) термостат

1.4

1) 2, 3

2) 2

2.1

1) 1, 3

2) 1, 3

2.2

1) 2, 4, 5

2) 2, 4

2.3

1) 3-2-1

2) 2, 3, 4

3.3

1) 2, 4

2) b = 'c'

3) k = 1

4) 5

3.4

1) 1, 4

2) 1, 3, 5

ДОДАТОК А

Рекомендації і вказівки для виконання лабораторних робіт

1 ДОСЛІДЖЕННЯ АВТОНОМНОГО ПРИСТРОЮ ВИМІРЮВАННЯ ТЕМПЕРАТУРИ НА БАЗІ ДАТЧИКА ТЕМПЕРАТУРИ Dallas 18b20

1.1 Мета роботи

Вивчити основні принципи прийому даних з датчика температури, їх обробки та індикації

1.2 Методичні вказівки щодо організації самостійної роботи студентів

Останнім часом у техніці все частіше необхідно розробляти пристрої, які не мають бути позв’язані з комп’ютером. Наряду з тим, необхідно проводити зчитування з датчиків, отримані значення, виконувати їхню обробку та виводити інформацію не крізь послідовний порт у комп’ютер, а, наприклад, на РК-дисплей. Також можлива ситуація, коли необхідно отримати деяку реакцію на зовнішній подразник. Яскравим прикладом може бути система підігріву, яка автоматично вмикається/вимикається при конкретних значеннях температури, або звичайна сигналізація. Такі нескладні системи можуть бути реалізовані на базі МК, наприклад, Arduino Uno.

Arduino Uno може отримувати живлення не тільки від USB, але і від зовнішнього пристрою живлення. Зовнішнє живлення може буди подане з перетворювача напруги AC/DC (блок живлення) або безпосередньо з акумуляторної батареї. Перетворювач напруги підключається за допомогою розніму 2.1 мм з центральним позитивним полюсом. Дроти від батареї підключаються до рознімів Gnd та Vin – рознімів живлення.

Платформа Arduino може працювати від зовнішнього живлення від 6 до 20 В. Однак при напрузі живлення нижче 7В, вивід 5В може видавати напругу меншу за 5В, завдяки цьому платформа може працювати нестабільно. В ході використання напруги вище 12 В регулятор напруги може перегріватися й пошкодити плату.

Рекомендована напруга живлення – від 7 до 12 В.

Калібрування датчика Dallas 18b20. Калібрування будь-яких вимірювальних пристроїв полягає у встановленні залежності між показниками пристрою та реальними значеннями вимірювальної (вхідної) величини. Під калібруванням, найчастіше, розуміють процес підстроювання показників вихідної величини або індикації вимірювального інструменту до моменту досягнення узгодження між еталонною величиною на вході та результатом на виході (з урахуванням оговореної точності). Наприклад, калібруванням медичного термометру, що відображає температуру здорової людини T=36.6 ^{o}C та результатом на дисплеї T=36.3 ^{o}C, буде додаток T=0.3 ^{o}C.

Калібрування буває первинним, вторинним та позаплановим. На етапі розробки пристрою проводиться первинне калібрування, яке полягає в дослідженні показника датчика та порівнянні результатів, відображених на ньому, з реальним значенням. Після цього в схему пристрою або програму вносяться відповідні зміни. Найчастіше результатом калібрування є або внесення відповідного коефіцієнта, на який примножується значення датчика, або введення додаткового доданка.

Датчик Dallas 18b20. Датчик Dallas 18b20 дозволяє вимірювати температуру з великою точністю. Він працює по шині One-Wire. Для пристроїв працюючих з цією шиною, в середовищі Arduino використовується бібліотека OneWire.

Таблиця 1.1 – Характеристики датчика Dallas 18b20

Показник	DS18b20
Дозволений діапазон t, ^{o}C	-55..+125
Похибка вимірів r, min	± 0.5 ^{o}C при t=10..+85 ^{o}C
Похибка вимірів r, max	± 2 ^{o}C
Роздільна здатність шкали t , ^{o}C	0.5 / 0.25 / 0.125 / 0.0625

Датчик може знаходитися у металевому корпусі, який заізольовано термоусадковою трубкою. Завдяки цьому його можна занурювати у воду.

Зовнішній вигляд та внутрішня структура датчика наведений на рис. 1.1.

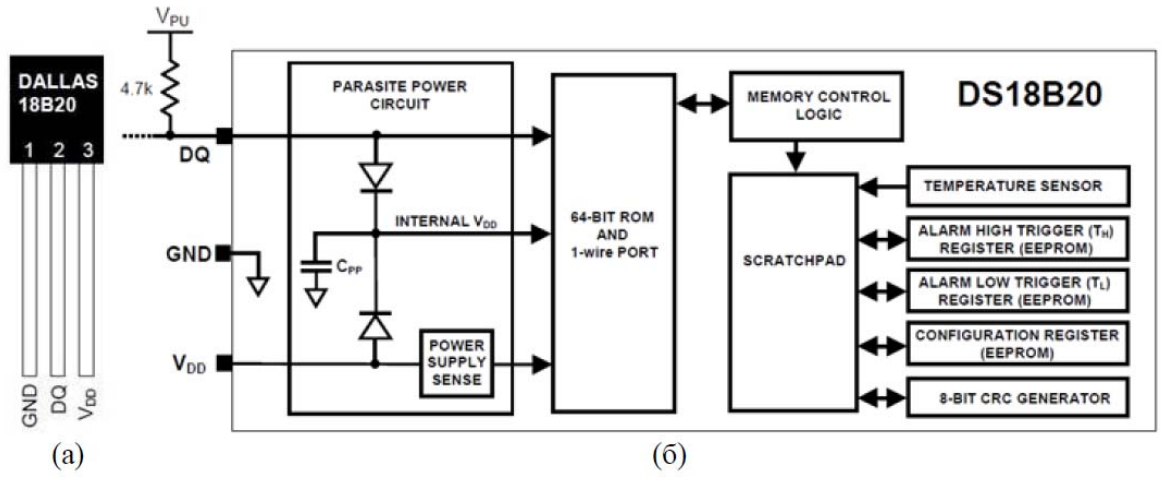


Рисунок 1.1 – Датчик Dallas 18b20 (а) та його внутрішня структура (б)

Датчик містить унікальний 64-бітний ROM код, який складається з 8 біт, що визначає код серії, яка складається з 9 байт:

– 1 та 2 байти зберігають інформацію про температуру.

– 3 та 4 байти зберігають відповідно верхню та нижню межі температури.

– 5 та 6 байти зарезервовані.

– 7 та 8 байти використовуються для надточного вимірювання температури.

– 9 байт зберігає завадостійкий CRC код попередніх 8 байт.

Підключення датчика до плати має свої особливості (рис. 1.2). Він використовує так зване «паразитне живлення». Паразитне живлення – це живлення електронного пристрою напругою будь-яких сигналів без використання спеціально виділеної шини живлення. Сигнальна шина даних, з якої отримується паразитне живлення, може бути спеціально призначена для такого режиму. Але можливий також випадок, коли сигнальна шина не призначена для надання паразитного живлення, при цьому відібрана потужність має бути достатньо малою, щоб не порушити роботу сигнальної лінії за її прямим призначенням.

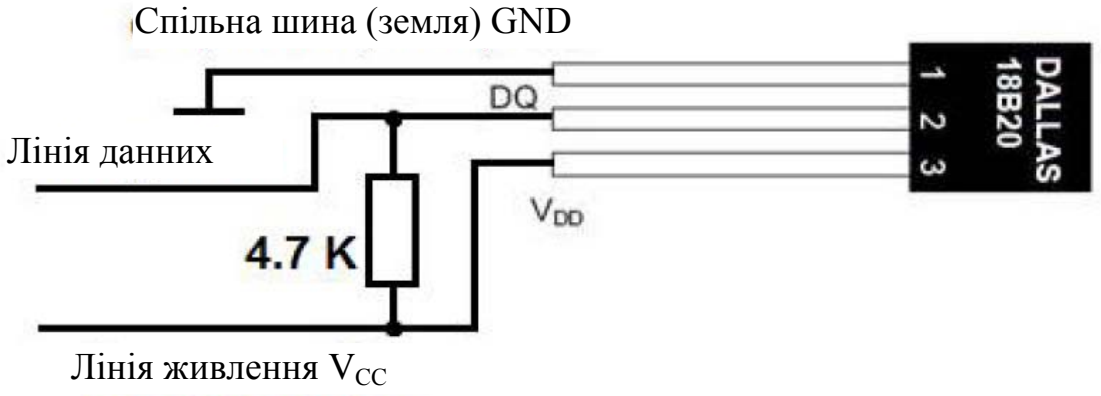


Рисунок 1.2 – Схема підключення датчика

Логічна лінія може бути використана як джерело живлення, якщо на ній гарантовано присутній сигнал високого рівня впродовж досить тривалого часу. Пристрій, що одержує паразитне живлення, може деякий час заряджатися від лінії, а після цього по цій же лінії передати деяку інформацію.

Інтерфейс 1-Wire. Цей інтрейфейс використовує паразитне живлення.

1-Wire – двонаправлена шина зв’язку для пристроїв з низькошвидкісною передачею даних (зазвичай 15.4 Кбіт/с, максимум 125 Кбіт/с), в якій дані передаються по ланцюгу живлення (тобто усього використовується 2 дроти – один для заземлення, а інший – для живлення та даних; у деяких випадках використовують й окремий дріт живлення).

Зазвичай інтерфейс 1-Wire використовують для зв’язку з простими пристроями, такими як, наприклад, цифрові термометри та вимірювачі параметрів зовнішнього середовища.

Схема для проведення експерименту в лабораторній роботі наведена на рис. 1.3.

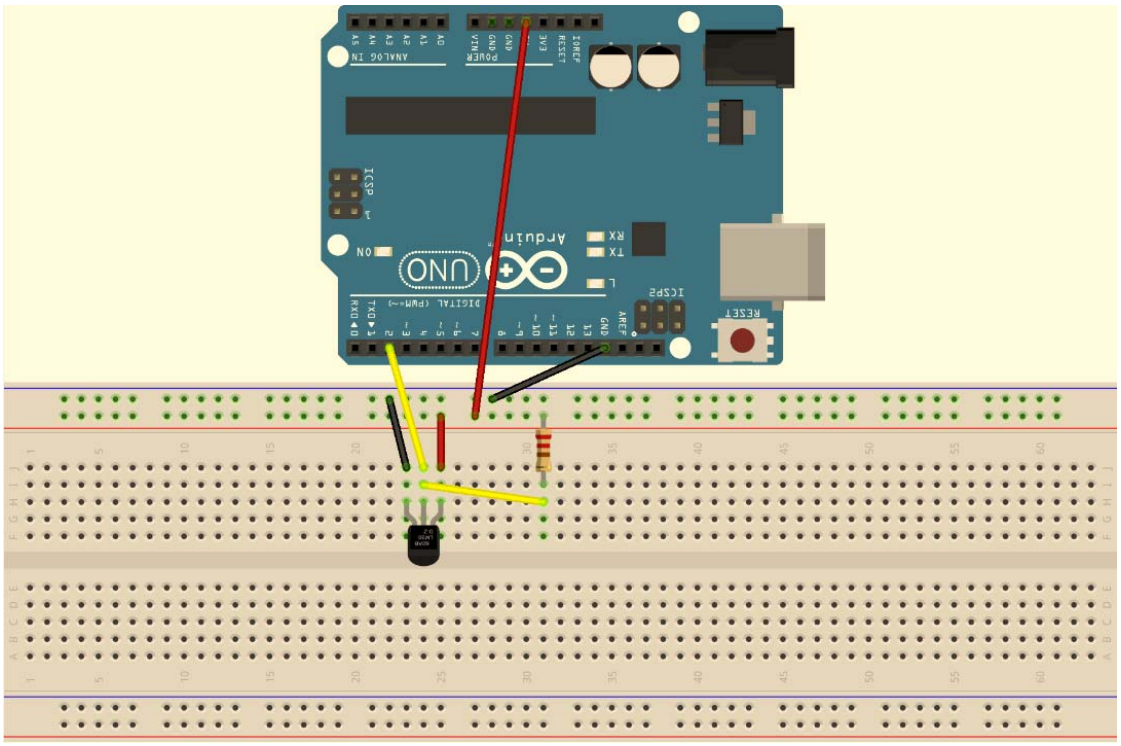


Рисунок 1.3 – Схема дослідження датчика Dallas 18b20

1.3 Опис лабораторної установки

До лабораторної установки належать:

– Arduino Uno
– DS18b20
– Резистор
– Плата розширення
– Дроти для з’єднання плат

Зокрема, для роботи датчика, потрібно не лише правильного підключення останнього до плати Arduino Uno, а й введення коду програми, що відповідає за програмний зв'язок Arduino та DS18b20.

– Скетч програмного коду для дослідження наведений нижче.

```
#include <OneWire.h> // including library 1 - wire
#include <DallasTemperature.h>// including Dallas`s library

//--------------------------------------------------------
//Defining pins on bus 1 - wire
//--------------------------------------------------------
#define ONE_WIRE_BUS_1 2
#define ONE_WIRE_BUS_2 4

OneWire oneWire_in(ONE_WIRE_BUS_1);
OneWire oneWire_out(ONE_WIRE_BUS_2);

DallasTemperature sensor_inhouse(&oneWire_in);
DallasTemperature sensor_outhouse(&oneWire_out);

void setup(void)
{
  Serial.begin(9600);
  Serial.println("Dallas Temperature Control Library Demo - TwoPin_DS18B20");

  sensor_inhouse.begin();
  sensor_outhouse.begin();
}
```

```
void loop(void)
{
  Serial.print("Requesting temperatures...");
  sensor_inhouse.requestTemperatures();
  sensor_outhouse.requestTemperatures();
  Serial.println(" done");

  Serial.print("Inhouse: ");
  Serial.println(sensor_inhouse.getTempCByIndex(0));

  Serial.print("Outhouse: ");
  Serial.println(sensor_outhouse.getTempCByIndex(0));
}
```

1.4 Порядок виконання роботи та методичні вказівки

1. Ознайомитися з теоретичними відомостями.

2. Отримати у викладача необхідні матеріали/початкові дані для проведення лабораторної роботи.

3. Запустити на комп’ютері середовище розробки Arduino.

4. Імпортувати та підключити до проекту необхідні бібліотеки (бібліотеки OneWire, DallasTemperature).

5. Розробити та спроектувати схему пристрою.

6. Написати програму, яка виконує поставлену задачу.

7. Після завершення написання програмного коду (скетчу) натиснути кнопку «Verify» та переконатися, що у нижній частині робочого вікна з’явився напис «Done Compiling». Цей напис означає, що у програмі не знайдено помилок.

8. Підключити плату Arduino через USB до ПК.

9. Скласти розроблену схему.

10. Провести такі налаштування: вибрати в «Tools»-> «Board» тип плати, з якою виконується робота. Перевірити, чи правильно обрано USB-порт за допомогою «Tools»-> «Serial port». Після закінчення натиснути на кнопку «Upload».

11. Якщо внизу з’явився надпис «Done uploading» – процес запису пройшов успішно.

12. Оформити звіт з лабораторної роботи згідно з пунктом 1.5

1.5 Зміст звіту

Звіт з лабораторної роботи має містити:

- номер та тему лабораторної роботи;
- мету роботи;
- стислі теоретичні відомості;
- порядок виконання роботи;
- постановку задачі;
- схему спроектованого пристрою з поясненнями;
- алгоритм роботи схеми;
- текст програми, який містить необхідні компоненти та пояснення;
- висновки з лабораторної роботи.

1.6 Контрольні запитання

1. Які особливості необхідно враховувати при живленні плати Arduino від зовнішніх джерел живлення?
2. Що таке калібрування датчика? Як проводиться калібрування датчика та для чого воно потрібне?

3. Перелічіть характеристики датчика Dallas 18b20 та підключення датчика до плати.

4. Що таке «паразитне живлення» та де воно використовується?

5. Які особливості інтерфейсу 1-Wire? Де використовується даний інтерфейс та які його переваги?

2 УПРАВЛІННЯ СВІТЛОВИПРОМІНЮЮЧИМ ДІОДОМ ЗА ДОПОМОГОЮ ТАКТОВОЇ КНОПКИ

2.1 Мета роботи

Набути навички управління світловипромінюючим діодом за допомогою тактових кнопок

2.2 Методичні вказівки щодо організації самостійної роботи студентів

Під час проведення лабораторної роботи студент має використовувати такі деталі як світловипромінюючий діод (СВД), тактова кнопка, плата для прототипуванння, резистори, комутуючі дроти та плата Arduino Uno.

Кнопка Arduino. Кнопка (або кнопочний перемикач) – найпростіший та найдоступніший з усіх видів датчиків. Під час натискання подається МК сигнал, який потім призводить до будь-яких таких процесів: вмикання СВД, видавання звуків, запуск моторів та ін.

Тактові кнопки та кнопки-перемикачі. Кнопка – це дійсно простий пристрій, який замикає або розмикає електричну мережу. Виконувати розмикання/замикання мережі (ланцюга) можливо у різних режимах, при цьому можливо фіксувати або не фіксувати свій стан. Виходячи з цього, кнопки можна поділити на дві великі групи:

– Кнопки-перемикачі з фіксацією. Вони повертаються до первинного стану після того, як їх відпустили. Залежно від первинного стану їх ділять на нормально-замкнуті та нормально-розімкнуті кнопки.

– Кнопки без фіксації (тактові кнопки). Вони фіксуються та знаходяться в тому положенні, яке їм було надано після останньої взаємодії.

Кнопка – один із найрозповсюдженіших видів електронних компонентів. Деякі з них наведені на рис. 2.1.

Рисунок 2.1 – Приклади існуючих кнопок

У даній лабораторній роботі буде використана тактова кнопка із чотирма ніжками. Кнопка – це перемикач з двома парами контактів. Контакти в одній парі з’єднані між собою. Виходячи з цього, більш ніж один вимикач у схемі реалізувати не вдасться, але є можливість управляти водночас двома паралельними сегментами.

Залежно від ситуації, є можливість будувати як схеми з нормально замкнутими, так і з нормально розімкнутими контактами – для цього необхідно буде лише відповідним чином виконати з’єднання в схемі.

Підключення кнопки до Arduino. Почнемо з найпростішого способу підключення тактової кнопки.

На рис. 2.2 наведена схема, в якій джерело живлення – плата Arduino. Схема містить світловипромінюючий діод (СВД), обмежувальний резистор, номіналом від 200 до 500 Ом та кнопку, яка замикатиме та розмикатиме з’єднання.

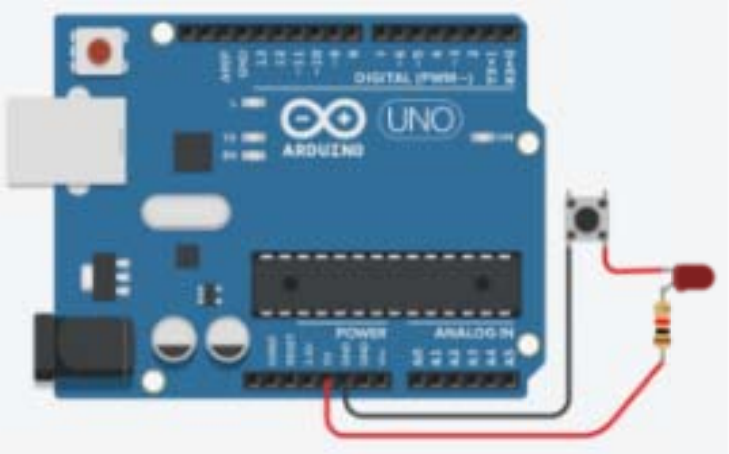


Рисунок 2.2 – Схема підключення тактової кнопки

При підключенні кнопки з двома парами ніжок важливо правильно обрати розмикаючі контакти. На рис. 2.3 наведена кнопка з двома парами ніжок.

Рисунок 2.3 – Тактова кнопка з 2 парами ніжок

На рис. 2.3 помітно, що пари ніжок розташовані з двох боків кнопки. Сама кнопка квадратна, але відстані між двома контактами помітні: можна виділити одразу два контакти на одній стороні та два на іншій. Саме між однією «парою» на стороні й буде реалізований вимикач. Для включення до схеми

необхідно з’єднати два контакти, між якими мінімальна відстань. Друга пара контактів просто дублює першу.

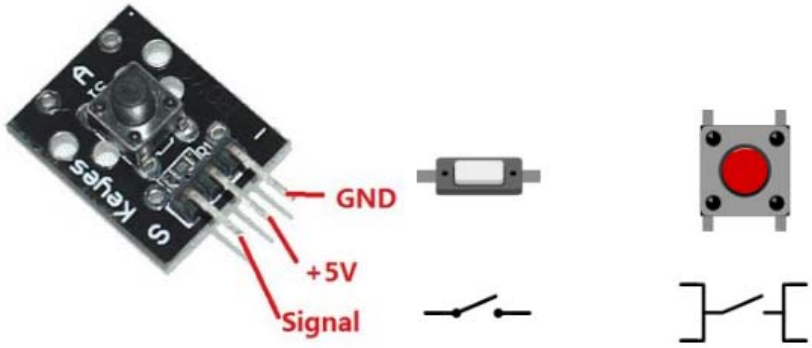


Рисунок 2.4 - Види кнопок

Найнадійніший спосіб визначення правильних ніжок – це продзвонити контакти тестером (мультиметром).

Сама схема з кнопкою, СВД та контролером Arduino не потребує особливих пояснень. Кнопка розриває контакт, СВД не горить. Під час натискання контакт замикається, СВД вмикається.

Брязкіт кнопки Arduino. В процесі роботи із кнопками можна зіткнутися з дуже неприємним явищем. Як випливає з самої назви, це явище обумовлюється брязкітом контактів усередині кнопкового перемикача. Металеві пластини стикаються одна з одною не миттєво (хоча і занадто швидко для наших очей), тому на короткий час в зоні контакту виникають скачки і провали напруги. Якщо ми не передбачимо з’явлення таких «сміттєвих» сигналів, то реагуватимемо на них щоразу та можемо привести наш проект до хаосу.

Для усунення брязкіту використовують програмні та апаратні рішення. Наведемо декілька методів заглушення брязкіту:

– Додамо до скетчу паузу 10–50 мілісекунд між отриманням значень з піну Arduino;

– Якщо ми використовуємо переривання, то програмний метод використовуватись не може і необхідно формувати апаратний захист. Найпростіший з них це – RC фільтр з конденсатором й опором.

– Для більш точного заглушення брязкіту використовується апаратний фільтр з використанням трігера Шмідта. Цей варіант дозволить отримати на вході в Arduino сигнал майже ідеальної форми.

Підключення СВД. СВД вмикається в схему через струмообмежувальний резистор 200 Ом до виходу Arduino. Номінал резистора може становити 200 Ом – 500 Ом, від цього змінюватиметься струм, який проходить крізь СВД й відповідно змінюватиметься яскравість СВД. Якщо підключити СВД безпосередньо до виводу Arduino, то кріз СВД піде занадто високий струм, внаслідок чого або сам СВД або вихід Arduino вийде з ладу. Також необхідно пам’ятати що СВД, це також діод, унього є полярність.

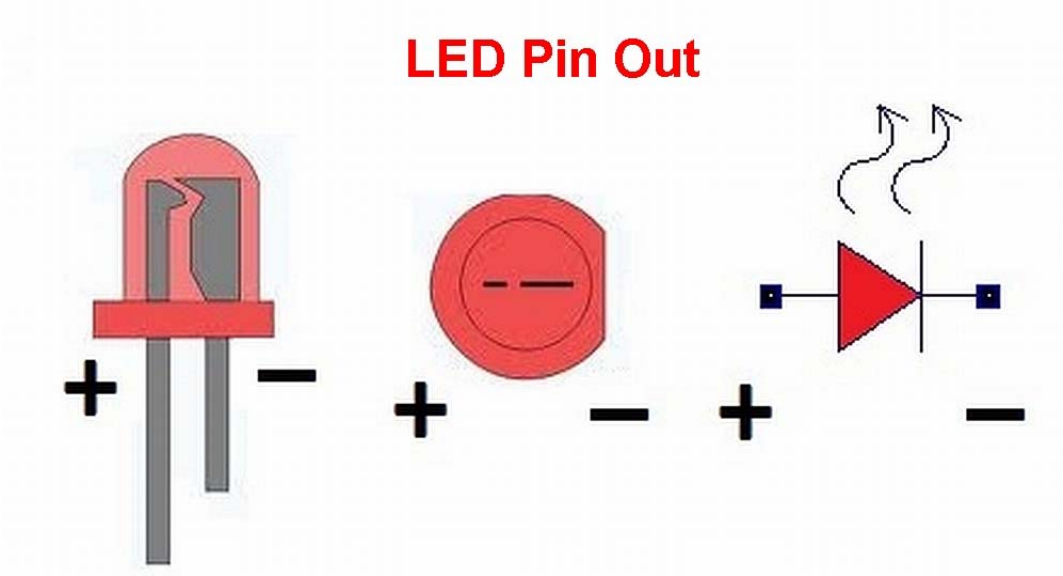


Рисунок 2.5 – Полярність СВД

Якщо підключити СВД неправильно, він не загориться, тому що струм кріз нього не піде (він буде зачинений). Визначити полярність СВД дуже просто, коротка нога СВД – це мінус (нуль або GND), а довга – це «+».

Для роботи зі світлодіодом, включеним у схему з Arduino, необхідно володіти такими функціями і константами:

– оператор setup();
– оператор loop();
– функція pinMode()
– функція digitalWrite();
– функція delay();
– константи OUTPUT, HIGH, LOW.

2.3 Опис лабораторної установки

До лабораторної установки належать:

– Arduino Uno
– Кнопка
– Світлодіод
– Резистор
– Плата розширення
– Дроти для з’єднання плат

Схема для виконання лабораторної роботи наведена на рис. 2.6.

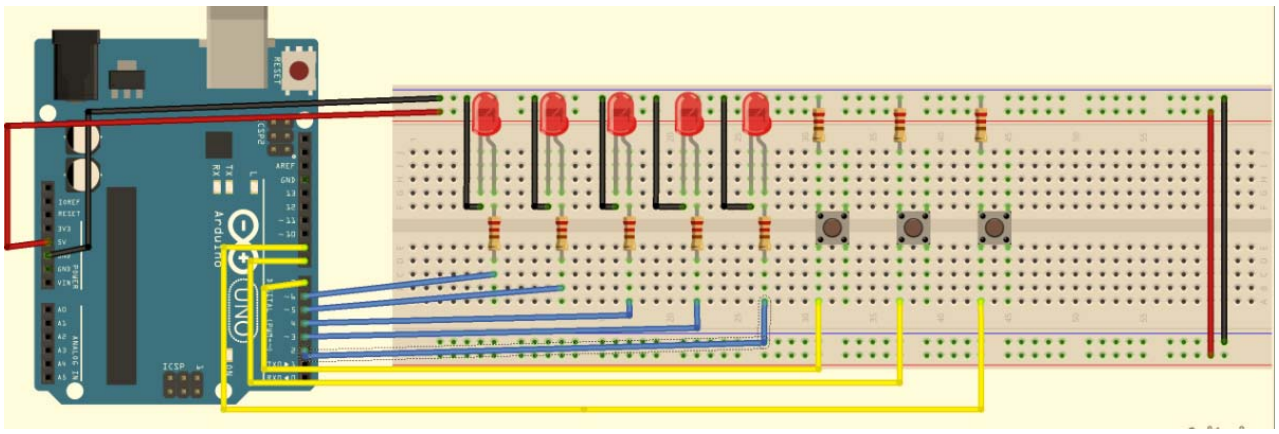


Рисунок 2.6 – Схема дослідження

Скетч програми для лабораторної роботи:

```
#define LED_1 2
#define LED_2 3
#define LED_3 4
#define LED_4 5
#define LED_5 6
/*--------------------------------
Defined number of pin to buttons
--------------------------------*/
#define Button_1 7
#define Button_2 8
#define Button_3 9
/*--------------------------------
Pleace to prototype users functions
--------------------------------*/
void setup() {
  int counter = 0;
/*--------------------------------
init pins on output! to connect leds.
--------------------------------*/
  pinMode(LED_1, OUTPUT);
  pinMode(LED_2, OUTPUT);
  pinMode(LED_3, OUTPUT);
  pinMode(LED_4, OUTPUT);
  pinMode(LED_5, OUTPUT);
/*--------------------------------
init pins to connect buttons on input.
--------------------------------*/
```

```
  pinMode(Button_1, INPUT);
  pinMode(Button_2, INPUT);
  pinMode(Button_3, INPUT);
/*------------------------------
init Serial port
------------------------------*/
Serial.begin(9600);//9600 - speed work of Serial port
}
void loop() {
/*------------------------------
place to create functions
------------------------------*/
/*------------------------------
Test leds
------------------------------*/
for(int counter = 0; counter<6; counter++){
  digitalWrite(counter-1,LOW);
  digitalWrite(counter,HIGH);//state can be LOW it is depend of connect
  delay(1000);
}
/*------------------------------
Test buttons
------------------------------*/
digitalRead(Button_1);
digitalRead(Button_2);
digitalRead(Button_3);
Serial.print(Button_1);
Serial.print(Button_2);
Serial.print(Button_3);
```

```
/*--------------------------------
Test buttons&LEDs
--------------------------------*/
digitalRead(Button_1);
digitalRead(Button_2);
digitalRead(Button_3);
if(Button_1 == HIGH){
  digitalWrite(LED_1,HIGH);
}
if(Button_2 == HIGH){
  digitalWrite(LED_2,HIGH);
}
if(Button_3 == HIGH){
  digitalWrite(LED_3,HIGH);
}
}
```

2.4 Порядок виконання роботи та методичні вказівки

1. Ознайомитися з теоретичними відомостями.

2. Отримати у викладача необхідні матеріали для проведення лабораторної роботи.

3. Запустити на комп'ютері середовища розробки Arduino.

4. Розробити та спроектувати схему пристрою, зображеного на рис.2.6.

5. Написати програму, яка виконує поставлену задачу (Текст програми наведений у вказівках до організації самостійної роботи студента).

6. Натиснути кнопку «Verify» й переконатися, що у нижній частині вікна з’явився надпис «Done Compiling». Це означає, що у написаній програмі не знайдено помилок.

7. Підключити плату Arduino через USB до ПК.

8. Обрати у «Tools»-> «Board» ваш тип плати. Перевірити, чи правильно обрано USB-порт в «Tools»-> «Serial port». Після цього натиснути на кнопку «Upload».

9. Якщо знизу з’явився надпис «Done uploading» – процес запису пройшов успішно.

10. Зробити звіт з виконаної роботи згідно з пунктом 2.5.

2.5 Зміст звіту

Звіт з лабораторної роботи має містити:

- номер та тему лабораторної роботи;
- мету роботи;
- стислі теоретичні відомості;
- порядок виконання роботи;
- постановку задачі;
- схему спроектованого пристрою з поясненнями;
- алгоритм роботи схеми;
- текст програми, який містить необхідні компоненти та пояснення;
- висновки з лабораторної роботи.

2.6 Контрольні запитання

1. Що таке кнопка? Види кнопок.
2. Які існують способи підключення кнопки у схему з платою Arduino?
3. Що таке брязкіт кнопки та як його знешкодити?

4. Що таке світлодіод? Як правильно його підключити до плати Arduino?

5. Перерахуйте та поясніть основні функції та константи, що використовуються під час роботи зі світлодіодом та Arduino.

3 СВІТЛОДІОДНА МАТРИЦЯ З ДРАЙВЕРОМ MAX7219

3.1 Мета роботи

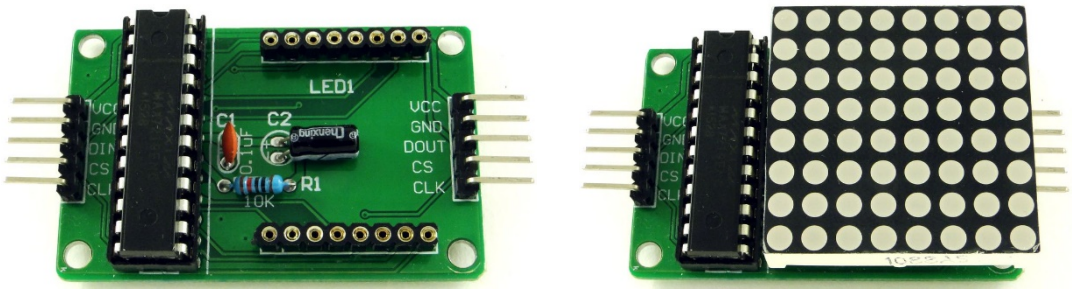
Набути навички управління модулем матриці за допомогою Arduino.

3.2 Методичні вказівки з організації самостійної роботи студентів

Під час виконання лабораторної роботи студент має ознайомитися з принципами підключення світлодіодної матриці, виводом пікселей, виводом тексту на матрицю.

Світлодіодна матриця (світлодіодний індикатор – СІД) – це графічний індикатор, який можна використовувати для виведення простих зображень, літер та цифр. У даній лабораторній роботі буде розглянуто матричний модуль з мікросхемою MAX7219.

Модуль становить плату з мікросхемою, обв'язкою та матричним індикатором. Зазвичай індикатор не впаюють у плату, а вставляють до розніму. Це використовується для того, щоб групу модулів можна було закріпити на якійсь поверхні гвинтами, а після цього вставити у них матриці.



а б

Рисунок 3.1 – Модуль на мікросхемі MAX7219 без СІД матриці (*а*)та із СІД матрицею (*б)*

Модуль має по 5 виводів з кожної сторони. З однієї сторони дані входять в модуль, з іншої сторони дані виходять з модуля та передаються до наступного. Це дозволяє з’єднувати модулі у ланцюги.

Вхідний рознім:

– VCC, GND – живлення;

– DIN – вхід даних;

– CS – вибір модуля (chip select);

– CLK – синхроімпульс.

Вихідний роз’єм:

– VCC, GND – живлення;

– DIN – вихід даних;

– CS – вибір модуля (chip select);

– CLK – синхроімпульс.

Працює модуль від напруги 5В.

Підключення модуля до Arduino. Підключення матричного модуля до контролера Arduino Uno слід виконувати за такою схемою (рис.3.2):

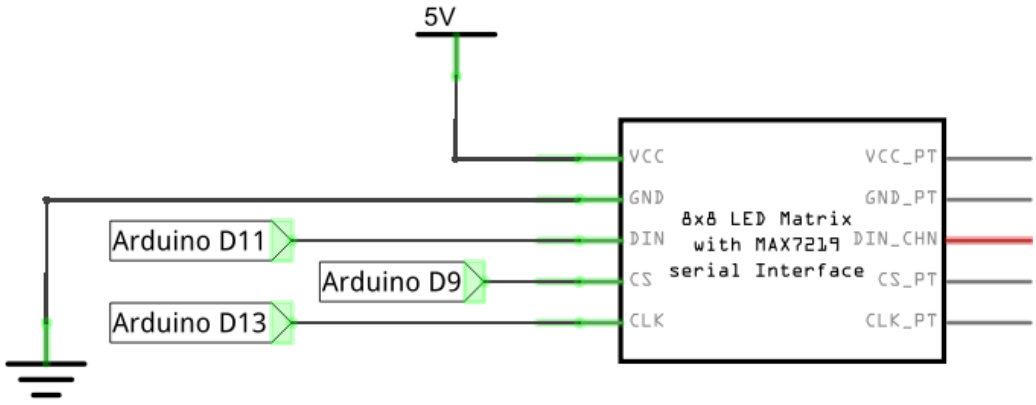


Рисунок 3.2 – Принципова схема з’єднання матричного модуля та плати Arduino Uno

Таблиця 3.1 – Відповідні розніми для з’єднання матричного модуля та контролера

Світлодіодна матриця 8x8 з MAX7219	VCC	GND	CIN	CS	CLK
Arduino Uno	+5V	GND	11	9	13

Зовнішній вигляд макета підключення матричного модуля і котролера наведено на рис 3.3.

3.3 Опис лабораторної установки

До лабораторної установки належать:

– Arduino Uno;

– MAX7219;

– плата розширення;

– дроти для з’єднання плат.

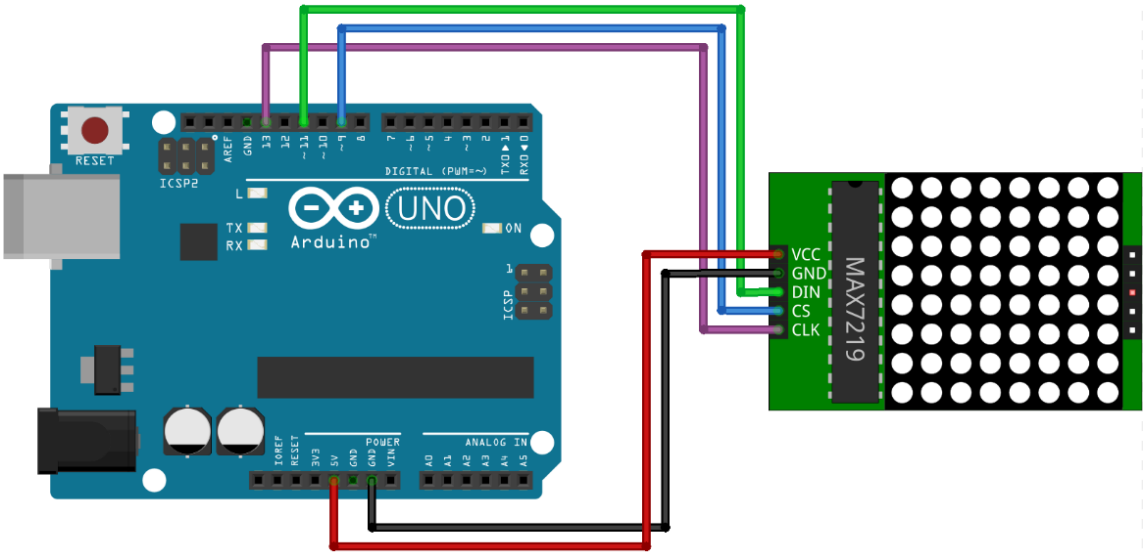


Рисунок 3.4 – Зовнішній вигляд макета

Під час виконання лабораторної роботи для роботи з мікросхемою MAX7219 необхідно буде використовувати бібліотеки SPI.h, Adafruit_GFX.h, Max72xxPanel.h. Для прикладу використання мікросхеми нижче буде наведена програма для виводу однієї точки на дисплей з координатою x=3 та y=4. Точка блиматиме з періодом 600 мс.

Текст скетча:

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Max72xxPanel.h>

int pinCS = 9;
int numberOfHorizontalDisplays = 1; // кількість матриць по горизонталі
int numberOfVerticalDisplays = 1; кількість матриць по вертикалі

Max72xxPanel matrix = Max72xxPanel(pinCS, numberOfHorizontalDisplays, numberOfVerticalDisplays);

void setup() {
  matrix.setIntensity(4); // яскравість від 0 до 15
}

void loop() {
  matrix.drawPixel(3, 4, HIGH); // запалюємо піксель з координатами {3,4}
  matrix.write(); // вивід всіх пікселів на матрицю
  delay(300);
  matrix.drawPixel(3, 4, LOW); // гасимо піксель
  matrix.write();
  delay(300);
```

```
}
```

Важливо відмітити, що після включення і виключення пікселів за допомогою функції «drawPixel», необхідно викликати функцію «write». Без функції «write», пікселі не висвітяться на матриці!

Виведення тексту за допомогою бібліотеки Adafruit _GFX – Library.

Аналогічним чином можна виводити на матрицю і будь-який інший символ, наприклад, букву. Але щоб мати можливість відображати будь-яку букву англійського алфавіту, нам необхідно буде визначити в програмі цілих 26 восьмибайтних масивів! У популярній бібліотеці Adafruit_GFX - Library окрім функцій для роботи з графікою і текстом, є і база латинських букв у верхньому і нижньому регістрах, а також усі розділові знаки й інші службові символи.

Відобразити символ на матриці можна за допомогою функції «drawChar». «drawChar» (x, y, символ, колір, фон, розмір). Перші два параметри функції відповідають за координати верхнього лівого кута символу. Третій параметр – це сам символ. Колір символу в нашому випадку буде рівний 1 або HIGH, оскільки матриця двоколірна. Фон рівний 0 або LOW. Останній параметр "розмір" зробимо рівним 1.

У бібліотеці Adafruit_GFX є безліч функцій для роботи з графікою. Наприклад, «drawCircle» (3, 3, 2, HIGH) накресліть коло з центром {3,3} і радіусом 2. Останній параметр - колір, але у разі монохромної матриці він рівний 1 або HIGH. Функція «drawLine» (0, 0, 3, 6, HIGH) накресліть відрізок між точками {0,0} і {3,6}.

Для виконання лабораторної роботи необхідно підключити матрицю до Arduino, та написати скетч:

```
#include <SPI.h>
#include <Adafruit_GFX.h>
```

```
#include <Max72xxPanel.h>


int pinCS = 9;
int numberOfHorizontalDisplays = 6;
int numberOfVerticalDisplays = 1;

Max72xxPanel matrix = Max72xxPanel(pinCS, numberOfHorizontalDisplays,
numberOfVerticalDisplays);

int wait = 200;

int spacer = 1;
int width = 5 + spacer;

void setup() {
  Serial.begin(9600);
    matrix.setIntensity(7);
    matrix.setRotation( 0, 1 );
    matrix.setRotation( 1, 1 );
    matrix.setRotation( 2, 1 );
    matrix.setRotation( 3, 1 );
}

void loop() {
      String tape = "qscmnbkf743";

      for ( int i = 0; i < width * tape.length() + matrix.width() - 1 - spacer; i++ )
{
```

```
    matrix.fillScreen(LOW);
    int letter = i / width;
    int x = (matrix.width() - 1) - i % width;
    int y = (matrix.height() - 8) / 2; // center the text vertically

    while ( x + width - spacer >= 0 && letter >= 0 ) {
      if ( letter < tape.length() ) {
        matrix.drawChar(x, y, tape[letter], HIGH, LOW, 1);
      }
      letter--;
      x -= width;
    }

    matrix.write();
    delay(wait);
  }
```

3.4 Порядок виконання роботи та методичні вказівки

1. Ознайомитися з теоретичними відомостями.

2. Отримати у викладача необхідні матеріали для проведення лабораторної роботи.

3. Запустити на комп’ютері середовище розробки Arduino.

4. Розробити та спроектувати схему пристрою.

5. Написати програму, яка виконує поставлену задачу.

6. Натиснути кнопку «Verify» й переконатися, що у нижній частині вікна з’явився надпис «Done Compiling». Це означає, що у написаній програмі не знайдено помилок.

7. Підключити плату Arduino через USB до ПК.

8. Скласти розроблену схему.

9. Обрати у «Tools»-> «Board» ваш тип плати. Перевірити, чи правильно обрано USB-порт в «Tools»-> «Serial port». Після цього натиснути на кнопку «Upload».

10. Якщо внизу з’явився надпис «Done uploading» – процес запису пройшов успішно.

11. Зробити звіт з виконаної роботи згідно з пунктом 3.5.

3.5 Зміст звіту

Звіт з лабораторної роботи має містити:

- номер та тему роботи;
- мету роботи;
- стислі теоретичні відомості;
- порядок виконання роботи;
- постановку задачі;
- схему спроектованого пристрою з поясненнями;
- алгоритм роботи схеми;
- текст програми, який містить необхідні компоненти та пояснення;
- висновки з лабораторної роботи.

3.6 Контрольні запитання

1. Що таке світлодіодна матриця? Як вона працює?

2. Підключення СІД до Arduino Uno.

3. Бібліотеки SPI.h, Max72xxPanel.h. – навіщо вони та які функції виконують?

4. Adafruit_GFX.h – що це та які функції виконує?

4 УПРАВЛІННЯ СВІТЛОДІОДНОЮ МАТРИЦЕЮ ЗА ДОПОМОГОЮ ДАНИХ З МАНІПУЛЯТОРА

4.1 Мета роботи

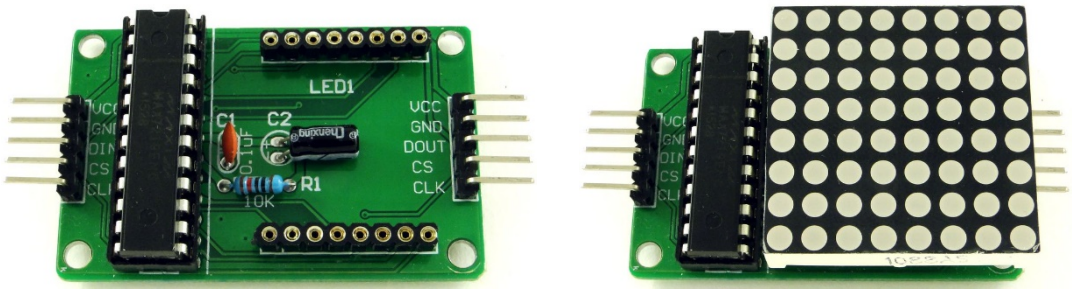
Набути навички управління світлодіодною матрицею за допомогою даних з маніпулятора

4.2 Методичні вказівки з організації самостійної роботи студентів

Під час виконання лабораторної роботи студент ознайомиться з принципами управління світлодіодною матрицею (СДМ) та роботою з маніпулятором.

Світлодіодна матриця (СДМ) – це графічний індикатор, який можна використовувати для виводу простих зображень, літер і цифр. У даній лабораторній роботі буде розглянуто матричний модуль з мікросхемою MAX7219.

Модуль є платою з мікросхемою, необхідною для неї обв'язкою та матричним індикатором. Зазвичай індикатор вставляють до розніму. Це використовується для того, щоб закріпити групу модулів на будь-якій поверхні гвинтами, а після цього вставити в них матриці.



а *б*

Рисунок 4.1 – Модуль на мікросхемі MAX7219 без СІД матриці (*а*) та з СІД матрицею (*б*)

Модуль має по 5 виводів з кожної сторони. З однієї сторони дані входять в модуль, з іншої сторони дані виходять з модуля та передаються до наступного. Це дозволяє з’єднувати модулі у ланцюги.

Вхідний рознім:

– VCC, GND – живлення;
– DIN – вхід даних;
– CS – вибір модуля (chip select);
– CLK – синхроімпульс.

Вихідний рознім:

– VCC, GND – живлення;
– DIN – вихід даних;
– CS – вибір модуля (chip select);
– CLK – синхроімпульс.

Працює модуль від напруги 5В.

Підключення модуля до Arduino. Підключення матричного модуля до контролера Arduino Uno слід виконувати за схемою 4.1.

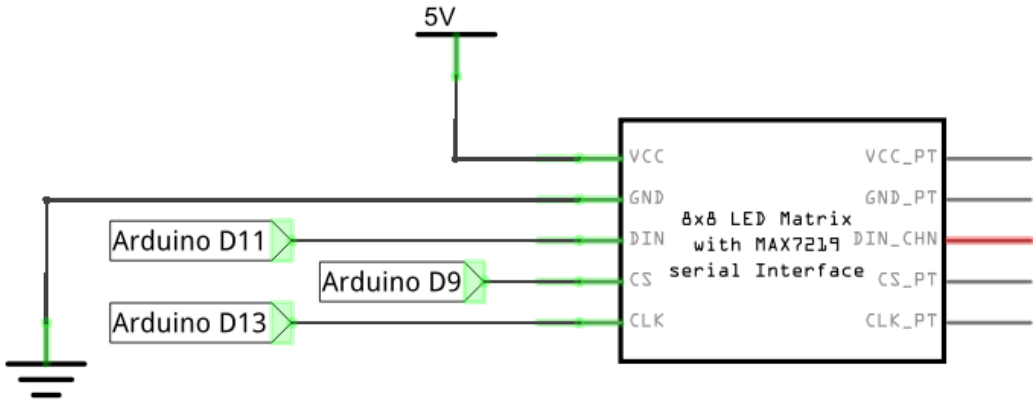


Рисунок 4.1 – Принципова схема з’єднання матричного модуля та плати Arduino Uno

Таблиця 4.1 – Відповідні розніми для з’єднання матричного модуля та контролера

Світлодіодна матриця 8x8 з MAX7219	VCC	GND	CIN	CS	CLK
Arduino Uno	+5V	GND	11	9	13

Для плат Arduino існують модулі аналогових джойстиків (маніпуляторів). Як правило, вони мають вісь X, Y і кнопку – вісь Z. Джойстик дозволяє плавніше і найбільш точно відстежувати міру відхилення від нульової точки. А окрім зручності порівняно з кнопками, він дозволяє реалізовувати досконаліші інтерфейси. Наприклад, при зміні будь-якого значення в меню, можна написати програму так, що чим сильніше відхилена вісь джойстика, тим швидше змінюється значення змінної. Припустимо, що нам необхідно змінити значення від 0 до 2000 з кроком в 1. Уявіть, скільки разів вам потрібно було б натискати кнопку або писати спеціальний алгоритм, скажемо при тривалості натиснення більше 3 с, або додавати/змінювати крок на 10 або 100. А під час використання джойстика це можна реалізувати набагато простіше.

Приклад джойстиків наведено на рис. 4.2.

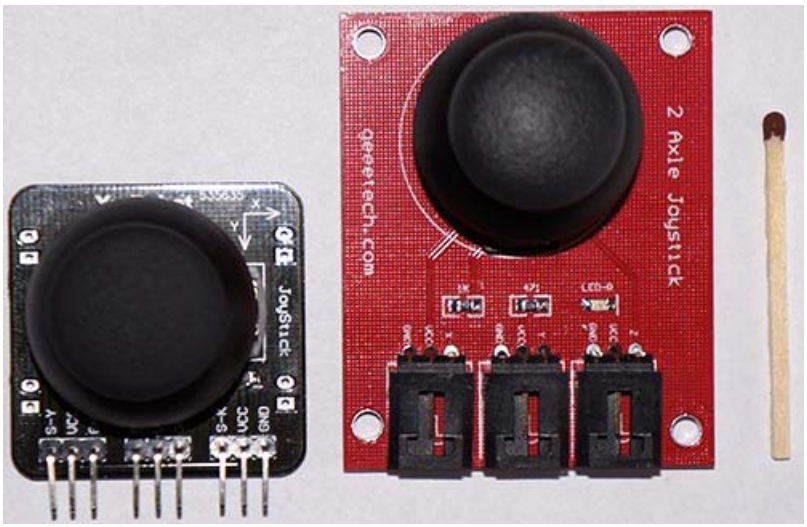


Рисунок 4.2 – Зовнішній вигляд маніпулятора (джойстика)

Кількість виводів виготовлено з урахуванням універсальності та зручності використання. Контакти Vcc і GND між усіма трьома групами контактів сполучені. Тобто для підключення треба 5 дротів: вісь X, вісь Y, кнопка Z, живлення Vcc і загальний GND. Джойстики пасивні модулі і не споживають енергію від плати Arduino. Живлення Vcc потрібне тільки для резисторів з підтяжкою. Бувають модулі без резисторів з підтяжкою, у такому разі, необхідно виведення підключення кнопки підтягнути до +Vcc через резистор 1–10 кОм.

Приклад підключення маніпулятора до Arduino наведений на рисунку 4.3.

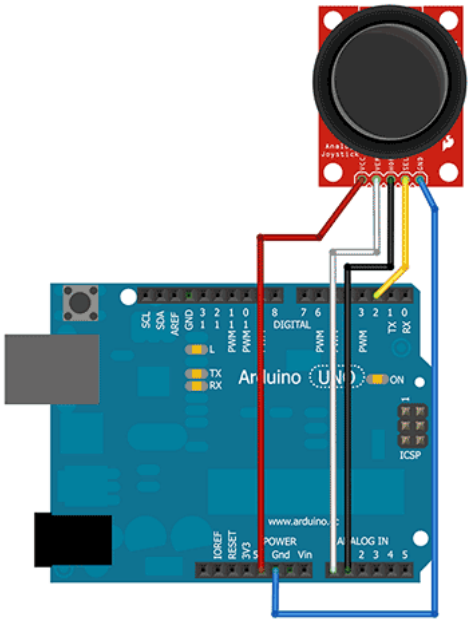


Рисунок 4.3 – Приклад схеми підключення маніпулятора до Arduino

4.3 Опис лабораторної установки

До лабораторної установки належать:

– Arduino Uno;
– матричний модуль MAX7219;
– маніпулятор;
– плата розширення;
– дроти для з’єднання плат.

Для перевірки роботи такої схеми включення потрібно завантажити пробний скетч:

```
#define axis_X 0    // Вісь X підключена до Analog 0
#define axis_Y 1    // Вісь Y підключена до Analog 1
#define axis_Z 2    // Вісь Z (кнопка джойстика) підключена до Digital 2

int value_X, value_Y, value_Z = 0;    // Змінні для  зберігання значень осей

void setup() {
  pinMode(axis_Z, INPUT);    // Задаємо як вхід
  Serial.begin(9600);
}

void loop() {
  value_X = analogRead(axis_X);    // Зчитуємо аналогові значення для осі X
  Serial.print("X:");
  Serial.print(value_X, DEC);      // Виводимо значення в Serial Monitor

  value_Y = analogRead(axis_Y);    // // Зчитуємо аналогові значення для ociY
  Serial.print(" | Y:");
  Serial.print(value_Y, DEC);      // Виводимо значення в Serial Monitor

  value_Z = digitalRead(axis_Z);   // Зчитуємо цифрове значення  для осі Z (кнопка)
  value_Z = value_Z ^ 1;           // Інвертуємо значення
  Serial.print(" | Z: ");
  Serial.println(value_Z, DEC);    // Виводимо значення в Serial Monitor
```

```
  delay(250);                    // Затримка 250 мс
}
```

На початку ми визначемо вхідні піни для осей (define), а потім у головному циклі прочитаємо значення з пінів і виведемо їх у Serial Monitor. У результаті, побачимо таку картину (рис 4.4):

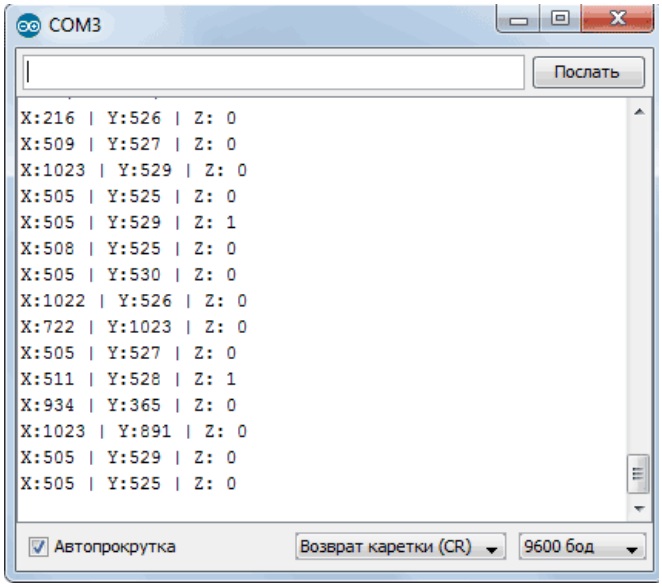


Рисунок 4.4 – Результат роботи скетчу

Для виконання лабораторної роботи необхідно підключити матричний модуль та маніпулятор водночас до котролера.

Зовнішній вигляд макета підключення матричного модуля і котролера наведено на рисунку 4.5.

Також необхідно написати скетч для роботи макету, наведеного на рисунку 4.5:

```
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Max72xxPanel.h>

int x, y;
int pinCS = 9;
```

```
int numberOfHorizontalDisplays = 1;
int numberOfVerticalDisplays = 4;
```

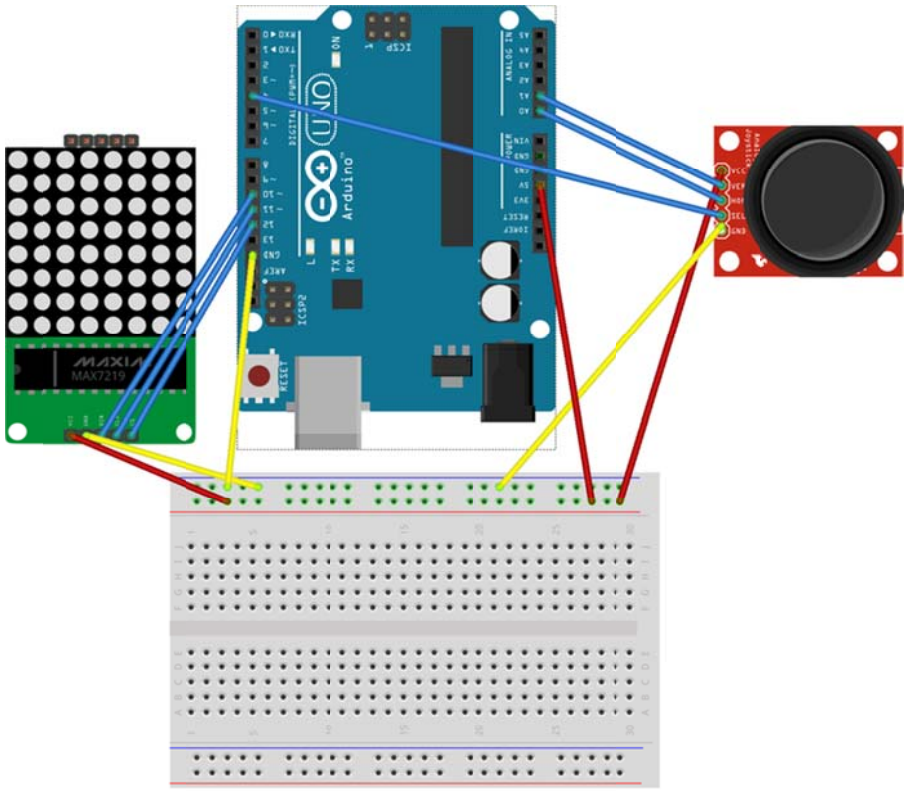


Рисунок 4.5 – Макет для дослідження

```
Max72xxPanel matrix = Max72xxPanel(pinCS, numberOfHorizontalDisplays, numberOfVerticalDisplays);
int pos_x = 0, pos_y = 0;
void setup() {
   pinMode(A0,INPUT);
   pinMode(A1,INPUT);
   Serial.begin(9600);
   matrix.setIntensity(4);

}
```

```
void loop() {
   x = analogRead(A0);
   y = analogRead(A1);
   Serial.println(x);
   Serial.println(y);
   if(x >= 1000 && pos_x>=1)pos_x -=1;
   if(x <= 150 && pos_x<=6)pos_x +=1;
   if(y >= 1000 && pos_y<=30)pos_y +=1;
   if(y <= 150 && pos_y>=1)pos_y -=1;
   matrix.drawPixel( pos_x, pos_y, HIGH);
   matrix.write();
   delay(100);
   matrix.drawPixel( pos_x, pos_y, LOW);
   matrix.write();
   delay(100);
  }
```

4.4 Порядок виконання роботи та методичні вказівки

1. Ознайомитися з теоретичними відомостями.

2. Отримати у викладача необхідні матеріали для проведення лабораторної роботи.

3. Запустити на комп’ютері середовище розробки Arduino

4. Розробити та спроектувати схему пристрою.

5. Написати програму, яка виконує поставлену задачу.

6. Натиснути кнопку «Verify» й переконатися, що у нижній частині вікна з’явився надпис «Done Compiling». Це означає, що у написаній програмі не знайдено помилок.

7. Підключити плату Arduino через USB до ПК.

8. Скласти розроблену схему.

9. Обрати у «Tools»-> «Board» ваш тип плати. Перевірити, чи правильно обрано USB-порт у «Tools»-> «Serial port», натиснути на кнопку «Upload».

10. Якщо внизу з’явився надпис «Done uploading» – процес запису прошов успішно.

11. Зробити звіт з виконаної роботи згідно з пунктом 4.5.

4.5 Зміст звіту

Звіт з лабораторної роботи має містити:

- номер та тему роботи;
- мету роботи;
- стислі теоретичні відомості;
- порядок виконання роботи;
- постановку задачі;
- схему спроектованого пристрою з поясненнями;
- алгоритм роботи схеми;
- текст програми, який містить необхідні компоненти та пояснення;
- висновки з лабораторної роботи.

4.6 Контрольні запитання

1. Що таке світлодіодна матриця? Як вона працює?
2. Що таке маніпулятор (джойстик)?
3. Навіщо маніпулятору використовувати вісь X, Y, Z?
4. Пояснити скетч.

5 ДОСЛІДЖЕННЯ АНАЛОГОВОГО ТЕРМОДАТЧИКА LM35 ПІД УПРАВЛІННЯМ ARDUINO

5.1 Мета роботи

Вивчити основні принципи підключення та управління датчиком LM35 за допомогою мікроконтролера, отримання даних з датчика температури LM35, їх обробки та індикації.

5.2 Вказівки з організації самостійної роботи.

Датчик температури LM35 може використовуватися у багатьох простих проектах. Розглянемо аналоговий термодатчик LM35. Характеристики термодатчика LM35:

- живлення: 2,7-5,5 Вольт;
- споживаний струм: 50 mkA;
- діапазон температур: 10°C - 125°C;
- похибка: 2 градуси.

Замість LM 35 можна використати будь-який інший датчик температури, наприклад, TMP35, TMP37, LM335. Виглядає датчик як транзистор і тому його легко сплутати з іншими датчиками, тому завжди уважно треба читати маркування на радіоелементах. У межах лабораторної роботи буде розглянуто датчик LM35, який має три виводи.

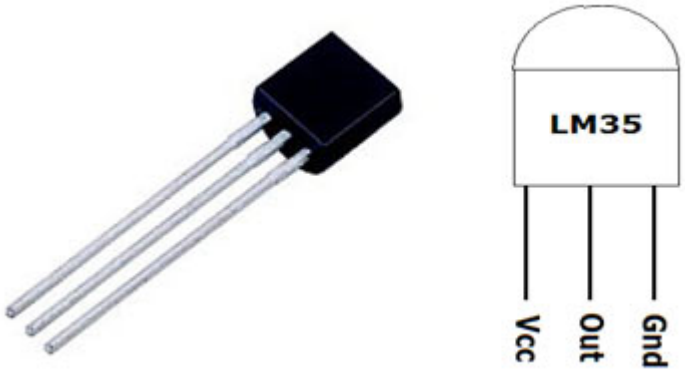


Рисунок 5.1 – Зовнішній вигляд досліджуваного датчика

Якщо подивитися на температурний сенсор LM35 з боку, то ліворуч буде позитивний контакт для живлення 2,7–5,5 Вольт, контакт по центру – це вихід, а справа – негативний контакт живлення (GND) – рис. 5.1.

5.3 Опис лабораторної установки

До лабораторної установки належать:

– Arduino Uno;
– термодатчик LM35;
– резистор;
– плата розширення;
– дроти для з’єднання плат.

Підключення датчика в даній лабораторній роботі буде реалізовано як на рисунку 5.2.

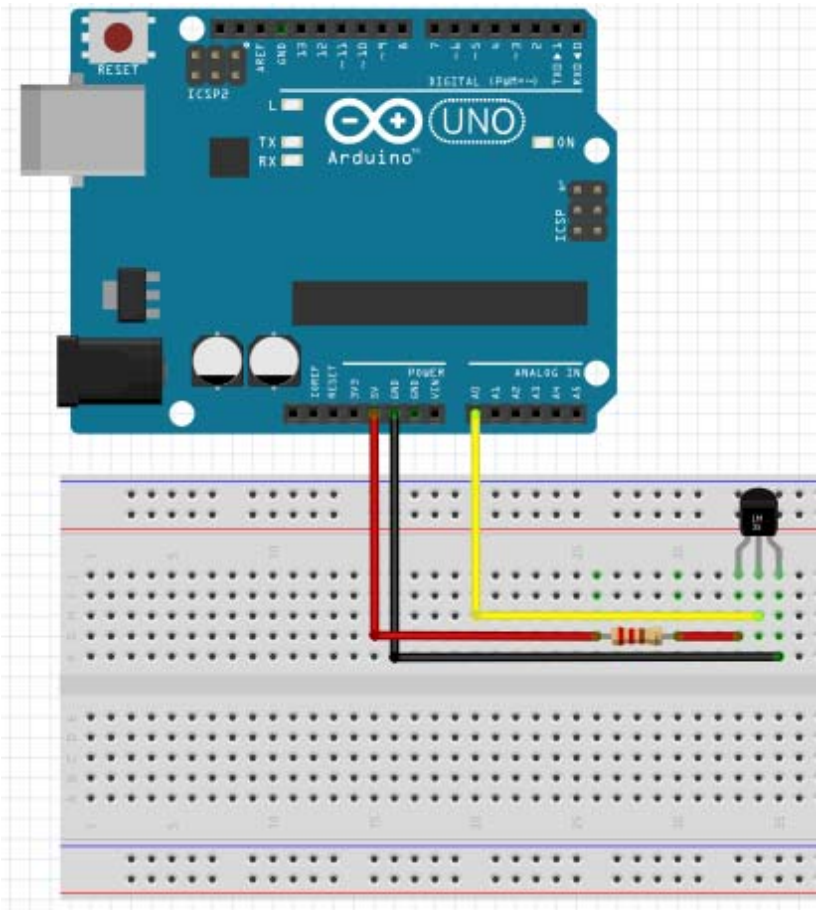


Рисунок 5.2 – Схема підключення термодатчика до Arduino Uno

Цей датчик аналоговий, тому на виході ми маємо значення не 0 або 1, а безперервну зміну напруги в діапазоні від 0 до 5 Вольт. Отже, ми повинні підключити датчик LM 35 аналогових портів A0 - A5 Arduino за схемою, зображеною на рис. 5.2. Після складання схеми завантажте простий скетч для зняття значень з аналогових датчиків і виведення їх у послідовний порт. Текст скетчу наведений нижче:

```
void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);              //inicialization serial port on speed 9600
  pinMode( A0, INPUT );            //init input mode on A0 pin
}
```

```
void loop() {

  // put your main code here, to run repeatedly:
  int res, res2;                                  // init constats
  res = analogRead(0);                            //first sensor
  res2 = analogRead(1);                           //second sensor
  temp = ( res/1023.0 )*5*1000/10;  //mathematic formuls
  Serial.println(res);
  Serial.println(res2);
  Serial.println(temp);
  delay(100);                                     //delay temp
}
```

5.4 Порядок виконання роботи та методичні вказівки

1. Ознайомитися з теоретичними відомостями.

2. Отримати у викладача необхідні матеріали для проведення лабораторної роботи.

3. Запустити на комп'ютері середовища розробки Arduino.

4. Імпортувати та підключити до проекту необхідні бібліотеки. Якщо це необхідно.

5. Розробити та спроектувати схему пристрою.

6. Написати програму, яка виконує поставлену задачу.

7. Натиснути кнопку «Verify» й переконатись, що у нижній частині вікна з'явився надпис «Done Compiling». Це означає, що у написаній програмі не знайдено помилок.

8. Підключити плату Arduino через USB до ПК.

9. Скласти розроблену схему.

10. Обрати у «Tools»-> «Board» ваш тип плати. Перевірити, чи правильно обрано USB-порт в «Tools»-> «Serial port», натиснути на кнопку «Upload».

11. Якщо внизу з’явився надпис «Done uploading» – процес запису пройшов успішно.

12. Зробити звіт з виконаної роботи.

5.5 Зміст звіту

Звіт з лабораторної роботи має містити:

- номер та тему роботи;
- мету роботи;
- стислі теоретичні відомості;
- порядок виконання роботи;
- постановку задачі;
- схему спроектованого пристрою з поясненнями;
- алгоритм роботи схеми;
- текст програми, який містить необхідні компоненти та пояснення;
- висновки з лабораторної роботи.

5.6 Контрольні запитання та завдання

1. Як працює LM35?
2. Наведіть схему включення LM35.
3. Поясніть скетч.
4. Які бібліотеки використовувались в лабораторній роботі під час написання скетча?

6 МОНІТОРИНГ ТЕМПЕРАТУРИ ЗА ДОПОМОГОЮ ARDUINO І ДАТЧИКА LM35

6.1 Мета роботи

Побудувати монітор температури, використовуючи LM35, 16x2 LCD дисплей і плату Arduino Uno

6.2 Методичні вказівки з організації самостійної роботи студентів

LM35 – це ідеальний температурний датчик для вимірювання температури довкілля. Він забезпечує лінійний вихід, пропорційний температурі, де 0 В відповідає температурі 0 градусів Цельсія, а зміна вихідної напруги на 10 мВ відповідає зміні температури на один градус Цельсія. Датчики LM35 простіше у використанні порівняно з термісторами і термопарами, тому що вони дуже лінійні і не вимагають ніякої обробки сигналу. Вихід LM35 може бути підключений безпосередньо до аналогового входу Arduino. Оскільки аналого-цифровий перетворювач (АЦП, ADC) Arduino має дозвіл 1024 біта, а опорна напруга складає 5 В, для обчислення температури з виміряного значення АЦП використовуватиметься така формула:

$$T = \frac{5.0 \cdot ana\log \mathrm{Re}\, ad(temperaturePin)}{1024} \cdot 100.0$$

Для відображення температури ми використовуватимемо рідкокристалічний дисплей (LCD).

6.3 Опис лабораторної установки

Для проведення експерименту необхідні такі комплектуючі:

– 1 x Arduino Uno;

– 1 х LCD дисплей 1602;
– 1 х потенціометр;
– 1 х макетна плата;
– 1 х датчик температури LM35;
– 1 х резистор на 1 кОм;
– Перемички.

Схема з’єднання наведена на рисунку 6.1.

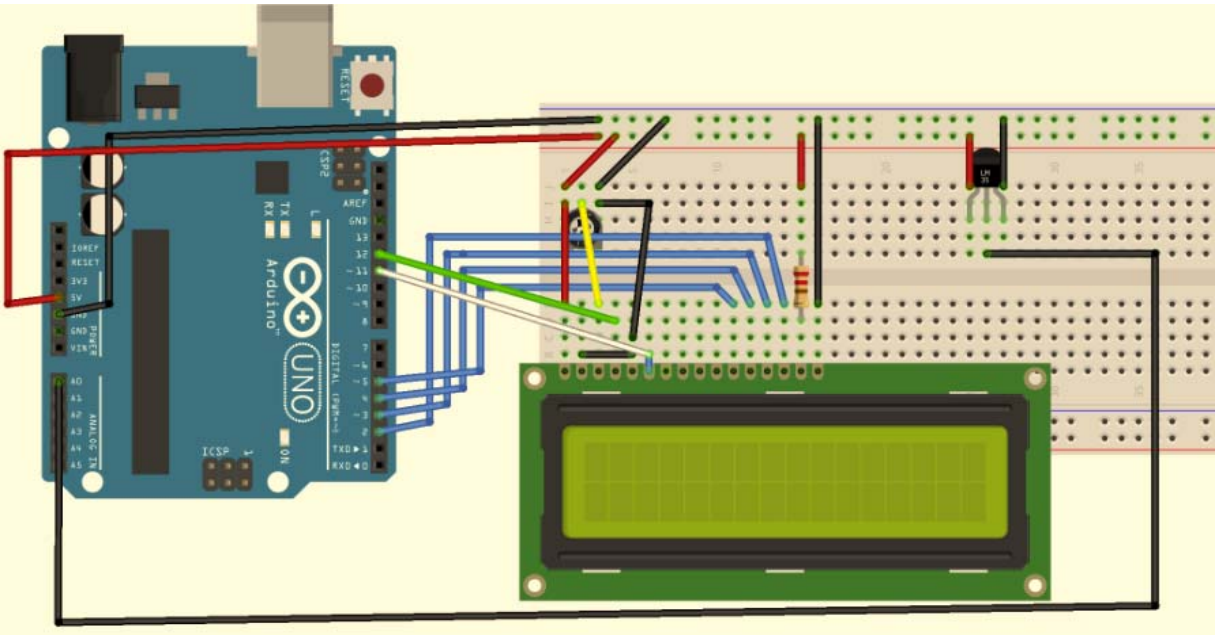


Рисунок 6.1 – Схема дослідження і виводу інформації на дисплей

Скетч:

```
#include <LiquidCrystal.h>

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
int raw;
float temp;
void setup() {
  // set up the LCD's number of columns and rows:
```

```
  pinMode(A0,INPUT);
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("Temperature at");
}

void loop() {
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  lcd.setCursor(0, 0);
  // print the number of seconds since reset:
  raw = analogRead(A0);
  temp = ( raw/1023.0 )*5.0*1000/10;
  lcd.print(temp);
  lcd.print(" C");
  lcd.print("  home");
}
```

6.4 Порядок виконання роботи та методичні вказівки

1. Ознайомитися з теоретичними відомостями.

2. Отримати у викладача необхідні матеріали для проведення лабораторної роботи.

3. Запустити на комп’ютері середовища розробки Arduino.

4. Імпортувати та підключити до проекту необхідні бібліотеки, якщо це необхідно.

5. Розробити та спроектувати схему пристрою.

6. Написати програму, яка виконує поставлену задачу.

7. Натиснути кнопку «Verify» й переконатися, що у нижній частині вікна з’явився надпис «Done Compiling». Це означає, що у написаній програмі не знайдено помилок.

8. Підключити плату Arduino через USB до ПК.

9. Скласти розроблену схему.

10. Обрати у «Tools»-> «Board» ваш тип плати. Перевірити, чи правильно обрано USB-порт в «Tools»-> «Serial port». Після цього натиснути на кнопку «Upload».

11. Якщо внизу з’явився надпис «Done uploading» – процес запису пройшов успішно.

12. Зробити звіт з виконаної роботи.

6.5 Зміст звіту

Звіт з лабораторної роботи має містити:

- номер та тему роботи;
- мету роботи;
- стислі теоретичні відомості;
- порядок виконання роботи;
- постановку задачі;
- схему спроектованого пристрою з поясненнями;
- алгоритм роботи схеми;
- текст програми, який містить необхідні компоненти та пояснення;
- висновки з лабораторної роботи.

6.6 Контрольні запитання

1. Як працює LM35?
2. Наведіть схему включення LM35.

3. Поясніть скетч.

4. Які бібліотеки використовувались в лабораторній роботі під час написання скетча?

5. Навіщо в схемі використовується потенціометр?

6. Як підключається в схему LCD-дисплей?

СЛОВНИК ТЕРМІНІВ

AC/DC (від англ. «alternating current/direct current» «змінний струм/постійний струм») – термін, який застосовується як для позначення характеру струму, так і для позначення режиму роботи пристрою, який відповідно підтримує режим роботи за змінним або постійним струмом.

DDRx – регістр порту, який виконує настройку розрядів порту x на вхід або вихід.

PINx – регістр порту, який виконує читання логічних рівнів розрядів порту x.

PORTx – регістр порту, який керує станом виходів порта x (якщо відповідний розряд налаштований як вихід) або підключенням внутрішнього pull-up резистора (якщо відповідний розряд налаштовано як вхід).

USB-порт – універсальна послідовна шина, призначена для з'єднання периферійних пристроїв обчислювальної техніки.

Алгоритм – послідовність команд, яку містить код, у результаті виконання якої мікроконтролер вирішує поставлену задачу.

Аналоговий сигнал – сигнал (напруга, струм тощо), неперервний на всьому проміжку часу.

Архітектура мікроконтролера – це ретельний підбір таких комбінацій кодів операцій, які можна було б виконати за один такт тактового генератора.

Байт – одиниця вимірювання обсягу даних, яка дорівнює 8-ми бітам.

Бібліотека – це програмний код, що зберігається не в скетчі, а в зовнішніх файлах, які можна підключити до проекту.

Біт – мінімальна одиниця кількості інформації, яка дорівнює одному двійковому розряду, який може бути рівним одному з двох значень/станів (0

або 1), застосовуваних для подання даних у двійковій системі числення.

Налагоджувач коду – комп'ютерна програма, призначена для пошуку помилок в інших програмах, ядрах операційних систем та інших видах коду.

Динамічна оперативна пам'ять (DRAM) — один із видів комп'ютерної пам'яті із довільним доступом (RAM), найчастіше використовується як ОЗП сучасних комп'ютерів

Динамічний масив – масив, розмір якого можна змінювати під час виконання програми.

Друкована плата – пластина, виконана з діелектрика, на якій сформавані шари з провідними доріжками.

Закон Мура – закон, сформульований Г. Муром, який говорить, що кількість транзисторів на кристалі мікросхеми подвоюватиметься кожні 24 місяці, з чого випливає, що потужність обчислювальних пристроїв експоненційно зросте протягом відносно короткого проміжку часу.

Змінна – об'єкт програми, що має ім'я та значення.

Інтерпретація – призначення змісту символам формальної мови.

Коментар – пояснення до вихідного тексту програми, що знаходиться безпосередньо всередині коментуючого коду.

Компілятор – комп'ютерна програма, що перетворює вихідний код, написаний певною мовою програмування, на еквівалентний код в іншій мові програмування, який зазвичай необхідний для виконання програми машиною, наприклад, комп'ютером.

Комунікаційний протокол – це набір обумовлених наперед семантичних і синтаксичних правил передачі даних між двома пристроями.

Масив – структура даних, що зберігає набір значень, ідентифікованих за індексом або набору індексів, які приймають цілі значення з деякого заданого безперервного діапазону.

Машинна мова – це штучна мова, створена для передачі команд машинам, а саме для створення програм, які контролюють поведінку машин, та

запису алгоритмів.

Мова асемблер – мова програмування низького рівня для програмованої обчислювальної системи (мікропроцесора, мікроконтролера, комп'ютера або іншого програмованого пристрою), в якій існує сувора відповідність між операторами мови та машинними командами.

Оперативна пам'ять (SRAM) – напівпровідникова оперативна пам'ять, в якій кожен двійковий розряд зберігається в схемі з додатним зворотним зв'язком, що не потребує регенерації, необхідної в динамічній пам'яті (DRAM).

Оператор – найменша автономна частина мови програмування; команда або набір команд.

Пам'ять EEPROM – постійний запам'ятовувальний пристрій, що програмується та очищується за допомогою електрики, один з видів енергонезалежної пам'яті.

Пам'ять даних – пам'ять, призначена для зберігання змінних у процесі виконання програми (ОЗП).

Пам'ять програм – це постійна пам'ять (ПЗП), призначена для зберігання програмного коду (команд) і констант.

Плата розширення – плата, яку поміщають в слот розширення материнської плати комп'ютерної системи з метою додавання додаткових функцій.

Послідовний порт – двонаправлений послідовний інтерфейс, призначений для обміну байтовою інформацією.

Процесор мікроконтролера – пристрій, що має функціонал комп’ютера, знаходиться на одному напівпровідниковому кристалі; призначений для обробки даних, що надходять з вхідних периферійних пристроїв і передачі оброблених даних на вихідні периферійні пристрої.

Регістр – пристрій для запису, зберігання і зчитування n-розрядних двійкових даних і виконання інших операцій над ними.

Регістри МК – пам'ять, яка включає в себе внутрішні регістри процесора і регістри, які служать для управління периферійними пристроями (регістри спеціальних функцій).

Розрядність – кількість розрядів (бітів) електронного (зокрема, периферійного) пристрою або шини, які одночасно обробляються цим пристроєм або передаються по цій шині.

Середовище розробки – комплексне програмне рішення для розробки програмного забезпечення.

Скетч – це програма, написана для платформи Arduino і має певну структуру.

Такт – сигнал, що забезпечує координацію роботи деякої кількості цифрових мікросхем.

Тактова частота – основна одиниця вимірювання частоти тактів у синхронних колах, що визначає кількість елементарних операцій (тактів), що виконуються системою за одну секунду.

Тип даних – характеристика, яку надано об'єкту (змінній, функції, полю масиву тощо), яка визначає множину припустимих значень, формат їхнього збереження, розмір виділеної пам'яті та набір операцій, які можна робити над даними.

Умовний оператор – оператору, в ході застосування якого програма перевіряє поставлену умову і під час її виконання (істинності) виконує перший оператор, а у випадку не виконання умови(хибності) програма виконує інший оператор або він просто пропускається і продовжується виконання програми.

Флеш-пам'ять – це незалежна пам'ять, тобто така, з якої дані не стираються після відключення живлення.

Функція – частина програми, яка реалізує певний алгоритм і дозволяє звернення до неї з різних частин загальної (головної) програми.

Цикл – різновид конструкції у високорівневих мовах програмування, призначений для організації багаторазового виконання набору команд.

Чіп – електронна схема, яка виготовлена на напівпровідниковій структурі, на поверхні якої сформовані контактні площини.

Шина – підсистема, що слугує для передачі даних між функціональними блоками комп'ютера.

Цифровий сигнал – сигнал, який можна подати у вигляді послідовності дискретних значень.

СЛОВАРЬ ТЕРМИНОВ

AC/DC (от англ. «Alternating current/direct current» «переменный ток/постоянный ток») – термин, который применяется как для обозначения характера тока, так и для обозначения режима работы устройства, что, соответственно, поддерживает режим работы по переменному или постоянному току.

DDRx – регистр порта, который выполняет настройку разрядов порта x на вход или выход.

PINx – регистр порта, который выполняет чтение логических уровней разрядов порта x.

PORTx – регистр порта, который руководит состоянием выходов порта x (если соответствующий разряд настроек как выход) или подключением внутреннего pull-up резистора (если соответствующий разряд настроен как вход).

USB-порт – универсальная последовательная шина, предназначенная для соединения периферийных устройств вычислительной техники.

Алгоритм – последовательность команд, которую содержит код, в результате выполнения которой микроконтроллер решает поставленную задачу.

Аналоговый сигнал – сигнал (напряжение, ток и т.д.), непрерывный на всем промежутке времени.

Архитектура микроконтроллера – это тщательный подбор таких комбинаций кодов операций, которые можно было бы выполнить за один такт тактового генератора.

Байт – единица измерения объема данных, равная 8-ми битам.

Библиотека – это программный код, который хранится не в скетче, а во внешних файлах, которые можно подключить к проекту.

Бит – минимальная единица количества информации, равная одному двоичному разряду, который может быть равным одному из двух значений/состояний (0 или 1), применяемых для представления данных в двоичной системе исчисления.

Динамическая оперативная память (DRAM) – один из видов компьютерной памяти с произвольным доступом (RAM), чаще всего используется как ОЗУ современных компьютеров.

Динамический массив – массив, размер которого можно изменять во время выполнения программы.

Закон Мура – закон, сформулированный Г.Муром, который гласит, что количество транзисторов на кристалле микросхемы будет удваиваться каждые 24 месяца, из чего следует, что мощность вычислительных устройств экспоненциально возрастет в течение относительно короткого промежутка времени.

Интерпретация – назначение содержания символам формального языка.

Комментарий – пояснения к исходному тексту программы, находящиеся непосредственно внутри комментируемого кода.

Компилятор – компьютерная программа, которая преобразует исходный код, написанный на определенном языке программирования, в эквивалентный код в другом языке программирования, который обычно необходим для выполнения программы машиной, например, компьютером.

Коммуникационный протокол – это набор обусловленных заранее семантических и синтаксических правил передачи данных между двумя устройствами.

Массив – структура данных, которая хранит набор значений, идентифицированных по индексу или набора индексов, которые принимают целые значения из некоторого заданного непрерывного диапазона.

Машинный язык – это искусственный язык, созданный для передачи команд машинам, а именно для создания программ, которые контролируют поведение машин, и для записи алгоритмов.

Оперативная память (SRAM) – полупроводниковая оперативная память, в которой каждый двоичный разряд сохраняется в схеме с положительной обратной связью, не нуждается в регенерации, необходимой динамической памяти (DRAM).

Оператор – наименьшая автономная часть языка программирования; команда или набор команд.

Отладчик кода – компьютерная программа, предназначенная для поиска ошибок в других программах, ядрах операционных систем и других видах кода.

Память EEPROM – постоянное запоминающее устройство, программируемое и очищаеемое с помощью электричества, один из видов энергонезависимой памяти.

Память данных – память, предназначенная для хранения переменных в процессе выполнения программы (ОЗУ).

Память программ – это постоянная память (ПЗУ), предназначенная для хранения программного кода (команд) и констант.

Переменная – объект программы, который имеет имя и значение.

Печатная плата – пластина, выполненная из диэлектрика, на которой сформаваны слои с проводящими дорожками.

Плата расширения – плата, которую помещают в слот расширения материнской платы компьютерной системы с целью добавления дополнительных функций.

Последовательный порт – двунаправленный последовательный интерфейс, предназначенный для обмена байтовой информацией.

Процессор микроконтроллера – устройство, которое имеет функционал компьютера, находящееся на одном полупроводниковом кристалле; предназначено для обработки данных, поступающих с входных периферийных

устройств и передачи обработанных данных на выходные периферийные устройства.

Разрядность – количество разрядов (битов) электронного (в частности, периферийного) устройства или шины, которые одновременно обрабатываются устройством или передаются по этой шине.

Регистр – устройство для записи, хранения и считывания n-разрядных двоичных данных и выполнения других операций над ними.

Регистры МК – память, которая включает в себя внутренние регистры процессора и регистры, которые служат для управления периферийными устройствами (регистры специальных функций).

Скетч – это программа, написанная для платформы Arduino, имеющая определенную структуру.

Среда разработки – комплексное программное решение для разработки программного обеспечения.

Такт – сигнал, который обеспечивает координацию работы некоторого количества цифровых микросхем.

Тактовая частота – основная единица измерения частоты тактов в синхронных кругах, определяющая количество элементарных операций (тактов), выполняемых системой за 1 секунду.

Тип данных – характеристика, которая предоставлена объекту (переменной, функции, полю массива и т.п.), которая определяет множество допустимых значений, формат их хранения, размер выделенной памяти и набор операций, которые можно производить над данными.

Условный оператор – оператор, при применении которого программа проверяет поставленное условие и при его выполнении (истинности) исполняет первый оператор, а в случае невыполнения условия (ложности) программа выполняет второй оператор, или он просто пропускается, и продолжается выполнение программы.

Цикл – разновидность конструкции в высокоуровневых языках программирования, предназначенная для организации многократного выполнения набора команд.

Цифровой сигнал – сигнал, который можно представить в виде последовательности дискретных значений.

Флэш-память – это энергонезависимая память, то есть такая, с которой данные не стираются после отключения питания.

Функция – часть программы, которая реализует определенный алгоритм и позволяет обращения к ней из разных частей общей (главной) программы.

Чип – электронная схема, которая изготовлена на полупроводниковой структуре, на поверхности которой сформированы контактные плоскости.

Шина – подсистема, которая служит для передачи данных между функциональными блоками компьютера.

Язык ассемблер – язык программирования низкого уровня для программируемой вычислительной системы (процессора, микроконтроллера, компьютера или другого программируемого устройства), в которой существует строгое соответствие между операторами языка и машинными командами.

GLOSSARY

AC/DC (alternating current/direct current) – a term that is used to denote the nature of the current, and to denote the operation mode of the device, which supports the operation on alternating or direct current.

Algorithm – a chain of commands that contains the code, as a result of which the microcontroller solves the task.

An analog signal – a signal (voltage, current, etc.) that is continuous throughout the entire time span.

An array – a data structure that stores a set of values identified by an index or a set of indexes that take integer values from some given range.

Assembyк language – a low-level programming language for a programmable computing system (processor, microcontroller, computer, or other programmable device), in which there is a strict accordance between language operators and machine instructions.

A bit – the minimum unit of the amount of information equal to one binary digit, which can be equal to one of two values/states (0 or 1) used to represent data in binary calculus.

Bus – a subsystem that serves to transfer data between the functional blocks of the computer.

Byte – a unit of measurement of data equal to 8 bits.

A chip – an electronic circuit that is fabricated on a semiconductor structure, on the surface of which contact planes are formed.

A clock – a signal that coordinates the operation of a number of digital circuits.

Clock frequency – the basic unit of measure for the frequency of clock cycles in synchronous circles, which determines the number of elementary operations (clock cycles) performed by the system in 1 second.

Comment – explanations to the source code of the program, located directly inside the commented code.

A communication protocol – a set of predetermined semantic and syntax rules for transferring data between two devices.

A compiler – a computer program that converts source code written in a specific programming language into an equivalent code in another programming language that is usually necessary for the execution of a program by a machine, for example, a computer.

A conditional statement – an operator which when applied makes the program check the set condition and when it is executed (truth) the first statement is executed, and if the condition (falsehood) is not met, the second statement is executed, or it is simply skipped and the program continues.

A cycle – a type of construction in high-level programming languages designed to organize multiple execution of a set of commands.

Data memory – a memory designed to store variables in the course of program execution (RAM).

Data type – a characteristic that is provided to an object (variable, function, array field, etc.), which determines the set of acceptable values, the storage format, the size of the allocated memory, and the set of operations that can be performed on the data.

DDRx – the port register that configures the bits of port x for input or output.

Development environment – a comprehensive software solution for software development.

Digit capacity – the number of bits of an electronic (in particular, peripheral) device or bus that processed at the same time by the device or transmitted via this bus.

A digital signal – a signal that can be represented as a sequence of discrete values.

Dynamic array – an array which size can be changed during program execution.

Dynamic random access memory (DRAM) – one of the types of computer random access memory (RAM), most often used as the RAM of modern computers.

EEPROM memory – a permanent storage device, programmable and cleaned with electricity, one of the types of nonvolatile memory.

Expansion board – a board that is placed in the expansion slot of the motherboard of a computer system in order to add additional functions.

Flash memory – non-volatile memory, where data is not erased after a power outage.

A function – a part of a program that implements a specific algorithm and allows access to it from different parts of the main program.

Interpretation – the assignment of content to symbols of a formal language.

A library – a program code that is not stored in a sketch, but in external files that can be connected to a project.

Machine language – an artificial language created for transmitting commands to machines, to create programs that control the behavior of machines and to write algorithms.

A microcontroller processor – a device that has a computer functionality located on a single semiconductor chip; designed for processing data from input peripheral devices and transferring processed data to output peripheral devices.

Moore's Law – a law formulated by G. Moore, which states that the number of transistors on a chip will double every 24 months, which means that the power of computing devices will exponentially increase in a relatively short period of time.

Operator – the smallest autonomous part of the programming language; command or set of commands.

PINx – the port register that reads the logical levels of the bits of port x.

PORTx – the port register that controls the status of the outputs of port x (if the digit set as an output) or by connecting the internal pull-up resistor (if the appropriate digit is configured as an input).

A printed circuit board – a plate made of a dielectric on which layers conducting paths are molded.

Program memory – a permanent memory (ROM), designed to store program code (commands) and constants.

Random access memory (SRAM) – a semiconductor random access memory, in which each binary bit is stored in a circuit with positive feedback, does not need to be regenerated, unlike the necessary dynamic memory (DRAM).

Register - a device for recording, storing and reading n-bit binary data and performing other operations on them.

Registers of the MC – memory, which includes the internal registers of the processor and registers, which are used to control peripheral devices (registers of special functions).

A sketch – a program written for the Arduino platform that has a specific structure.

The architecture of the microcontroller – a careful selection of such combinations of operation codes that could be performed in a single clock cycle.

The code debugger – a computer program designed to find errors in other programs, operating systems and other types of code.

The serial port – a bidirectional serial interface designed for the exchange of byte information.

The USB port – a universal serial bus designed for connecting computer peripherals.

A variable – a program object that has a name and a value.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дмитрів В.Т., Шиманський В.М. Електроніка і мікросхемотехніка: навч. посібник. – Львів: Афіша, 2004. -175 с.

2. Прищепа М.М., Погребняк В.П. Мікроелектроніка: В 3 ч. Ч. 1: Елементи мікроелектроніки: навч. посібник. – К.: Вища шк., 2004. – 431 с.

3. Топильский В.Б. Схемотехника аналого-цифровых преобразователей. Учебное издание. – М.: Техносфера, 2014. – 288 с.

4. Сумик Маркиян. Основи теорії радіотехнічних систем: навч. посібник. – Львів: Видавництво Національного університету «Львівська політехніка», 2005. – 240 с.

5. Ткачук В. Електромеханотроніка: Підручник. – Львів: Видавництво Національного університету «Львівська політехніка», 2006. – 440 с.

6. Кузьмин И.В., Кедрус В.А. Основи теории информации и кодирования. – Киев: Вища школа, 1986. – 430 с.

7. Лесовой Л.В. Електроніка та мікропроцесорна техніка. Частина 2. Лабораторний практикум: навч. посібник. – Львів: Видавництво Львівської політехніки, 2014. – 268 с.

8. Бобало Ю.Я. Математичні моделі та методи аналізу надійності радіоелектронних, електротехнічних та програмних систем: монографія / Ю.Я. Бобало, Б.Ю. Волочій та ін. – Львів: Видавництво Львівської політехніки, 2013 – 300 с.

9. Петин В.А. Проекты с использованием контроллера Arduino. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2015. – 464с.

10. Блум Дж. Изучаем Arduino: инструменты и методы технического волшебства: пер. с англ. / Дж. Блум. – Санкт-Петербург: БХВ-Петербург, 2015. – 336 с.

11. Иго Т. Arduino, датчики и сети для связи устройств: пер. с англ. / Т. Иго. – 2-е изд. – Санкт-Петербург: БХВ-Петербург, 2015. – 544 с.

12. Электроника для всех [Электронный ресурс]. – http://easyelectronics.ru.

13. Официальный сайт сообщества разработчиков встроенных систем [Электронный ресурс]. – http://embedders.org.

14. Wolf W.H. Hardware-Software Co-Design of Embedded Systems [Электронный ресурс]. – http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=3602704FA35174F9A8FFBDEB C59F7932?doi=10.1.1.135.1147&rep=rep1&type=pdf

15. Сорокин С.В., Солдатенко И.С. С65 Основы разработки и программирования робототехни- ческих систем: учеб. пособие. – Тверь: Твер. гос. ун-т, 2017. – 157 с.

16. Мамичев Д.И. Простые роботы своими руками или несерьёзная электроника [Текст]. – М.: Солон-Пресс, 2016. – 144 с.

17. Боксел Д. Изучаем Arduino. 65 проектов своими руками [Текст]. – СПб.: Питер, 2017. – 400 с.

18. Шилдт Г. Полный справочник по C . – 4-е изд. – М.: Издательский дом «Вильямс», 2004. – 704 с.

19. Поляков К. Язык программирования Си. [Электронный ресурс].- http://kpolyakov.spb.ru/school/c.htm.

20. Официальная документация проекта Arduino [Электронный ресурс]. – https://www.arduino.cc/en/Guide/ HomePage

Навчальне видання

ГЛУХОВ Олег Вікторович
КРАВЧУК Ольга Олександрівна
ЛЕВЧЕНКО Євгеній Васильович

ВИВЧЕННЯ ВЛАСТИВОСТЕЙ МІКРОКОНТРОЛЕРІВ І ЕЛЕКТРОННИХ СИСТЕМ НА БАЗІ ПЛАТФОРМИ АРДУІНО

Навчальний посібник

Відповідальний випусковий І.М. Бондаренко

Редактор Б.П. Косіковська

Комп'ютерна верстка Г.М. Голоднікова

План 2019 (друге півріччя), поз. 7.
Підп. до друку 26.06.2019. Формат 60х 84 $^{1}/_{16}$. Спосіб друку – ризографія.
Умов. друк. арк. 11,16. Облік.вид.арк. 10,2. Тираж 50 прим.
Зам. № 1-7. Ціна договірна.

ХНУРЕ, 61166, Харків, просп. Науки, 14

Віддруковано у редакційно-видавничому відділі ХНУРЕ
Харків, просп. Науки 14