МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет кораблебудування
імені адмірала Макарова

В. М. РЯБЕНЬКИЙ, О. О. УШКАРЕНКО

ПРОГРАМОВАНІ ЕЛЕКТРОННІ СИСТЕМИ КЕРУВАННЯ, ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ

Підручник

Рекомендовано Вченою радою НУК

Миколаїв • НУК • 2019

УДК 681.5, 004.4
ББК 32.84
Р 98

Автори: В. М. Рябенький, доктор технічних наук, професор, заслужений діяч науки і техніки України, завідувач кафедри програмованої електроніки, електротехніки і телекомунікацій
О. О. Ушкаренко, доктор технічних наук, доцент кафедри програмованої електроніки, електротехніки і телекомунікацій

Рецензенти: І. І. Коваленко, доктор технічних наук, професор;
А. В. Переверзев, доктор технічних наук, професор

Рекомендовано Вченою радою
Національного університету кораблебудування
імені адмірала Макарова
як підручник для студентів вищих навчальних закладів
(протокол № 12 від 27.12.2019 р.)

Рябенький В. М.
Програмовані електронні системи керування, збору та обробки інформації : підручник / В. М. Рябенький , О. О. Ушкаренко. – Миколаїв : НУК, 2019. – 490 с.

У підручнику описуються основи проектування та програмування електронних систем керування, збору та обробки інформації на базі 8-розрядних мікроконтролерів AVR. Розглянуті питання створення систем збору, обробки інформації з використанням апаратно-програмних засобів Arduino.

Велика увага приділяється особливостям використання вбудованих в мікроконтролер периферійних пристроїв. Детально описано роботу різних інтерфейсів та протоколів обміну даними (RS-232, I^2C, 1-Wire, SPI, USB, Ethernet) з датчиками фізичних величин та мікросхемами пам’яті. Пояснюються принципи отримання та обробки даних з датчиків, які найбільш часто використовуються в мікропроцесорних системах керування, збору та обробки інформації. Приводиться велика кількість прикладів програм для мікроконтролерів на мові Сі, а також прикладів програм з розвиненим графічним інтерфейсом для комп’ютера, розроблених у середовищі Visual Studio. Набір конкретних схемних та програмних рішень може бути використаний при розробці широкого спектра мікропроцесорних систем керування, збору та обробки інформації.

Підручник призначений для студентів спеціальностей 171 – «Електроніка», 172 – «Телекомунікації та радіотехніка» і суміжних спеціальностей, а також всіх тих, хто цікавиться комп'ютерною електронікою і програмуванням.

УДК 681.5, 004.4
ББК 32.84
Р 98

© Рябенький В.М., Ушкаренко О. О., 2019
© Національний університет кораблебудування імені адмірала Макарова, 2019

ЗМІСТ

Вступ 7
Перелік умовних скорочень 9
Розділ 1. Основи програмування мікропроцесорних систем керування, збору та обробки інформації 10
1.1. Апаратно-програмні засоби Arduino для побудови систем керування, збору та обробки інформації 10
1.2. Основи програмування та моделювання пристроїв Arduino 17
1.2.1. Керування портами введення/виведення мікроконтролера 19
1.2.2. Зчитування даних з портів введення/виведення мікроконтролера 25
1.2.3. Програмна реалізація базових логічних елементів 30
1.2.4. Принцип формування ШІМ-сигналу 34
1.2.5. Зчитування даних з аналогових датчиків 38
1.2.6. Використання послідовного інтерфейса для керування пристроями 43
1.3. Приклад програмної реалізації мультиплексора 48
1.4. Керування алфавітно-цифровим рідкокристалічним індикатором 51
1.5. Керування серводвигуном та кроковим двигуном 53
1.6. Використання функцій в програмах 59
1.7. Використання масивів в програмах 63
1.8. Керування напівпровідниковим семисегментним індикатором 65
Завдання до розділу 1 71

Розділ 2. Датчики фізичних величин, що використовуються в мікропроцесорних системах керування, збору та обробки інформації 72
2.1. Основні характеристики датчиків 72
2.2. Інфрачервоний датчик руху HC-SR501 75
2.3. Ультразвуковий датчик відстані HC-SR04 88
2.4. Оптичний інфрачервоний датчик FC-51 94
2.5. Датчик кольору TCS3200 98
2.6. Цифровий акселерометр ADXL-321 та датчики вібрації 102
2.7. Датчик звуку KY-037 107
2.8. Датчик газу MQ-2 112
2.9. Датчик полум’я KY-026 118
2.10. Датчик освітленості 124
2.11. Датчик вологості та температури DHT11 131
2.12. Датчики швидкості обертання FC-03 135
2.13. Датчик вологості ґрунту YL-38 139
2.14. Електронний компас GY271 146
2.15. Промислові датчики фізичних величин 153
2.16. Використання GSM-радіомодему для відправлення SMS-

повідомлень 157
2.17. Годинник реального часу DS1302 164
Завдання до розділу 2 171

Розділ 3. Програмна реалізація елементів цифрової електроніки з використанням мікропроцесорних систем 175
3.1. Основні визначення 175
3.2. Програмна реалізація базових логічних елементів 176
3.2.1. Логічний елемент І 178
3.2.2. Логічний елемент АБО 186
3.2.3. Логічний елемент ВИКЛ.АБО 188
3.3. Програмна реалізація комбінаційних автоматів 190
3.4. Мультиплексори 199
3.5. Дешифратори-демультиплексори 204
3.6. Програмна реалізація тригерів 217
3.6.1. RS-тригери 218
3.6.2. D-тригер 224
3.6.3. JK-тригер 229
3.6.4. Т-тригер 232
3.7. Програмна реалізація цифрових автоматів 236
3.7.1. Використання SWITCH-технології при реалізації автоматів Мура 237
3.7.2. Використання SWITCH-технології при реалізації автоматів Мілі 242
3.8. Лічильники імпульсів 245
Завдання до розділу 3 252

Розділ 4. Цифрові інтерфейси мікропроцесорних систем керування, збору та обробки інформації 256
4.1. Загальна характеристика цифрових інтерфейсів 256
4.2. Однопровідний інтерфейс 1-Wire 256
4.2.1. Загальна характеристика інтерфейсу 1-Wire 257
4.2.2. Блок-схема датчика температури DS18B20 259
4.2.3. Протокол обміну даними по шині 1-Wire 263
4.2.4. Сполучення датчика DS18B20 з контролером Arduino 265
4.2.5. Мікропроцесорна система контролю температури на базі датчика температури DS18B20 267
4.3. Послідовна асиметрична шина зв’язку I^2C 272
4.3.1. Загальна характеристика інтерфейсу I^2C 272
4.3.2. Діаграми передачі даних 273
4.3.3. Сполучення датчика DS1621 з контролером Arduino 275
4.3.4. Мікропроцесорна система контролю температури на базі датчика температури DS1621 277
4.3.5. Електронний годинник на базі DS1307 283
4.4. Послідовний інтерфейс SPI 291
4.4.1. Загальна характеристика інтерфейсу SPI 292

4.4.2. Особливості використання модуля SPI мікроконтролерів 294
4.4.3. Організація обміну даними в мультимікропроцесорній системі 296
4.4.4. Керування мікросхемами пам’яті SD/MMC по інтерфейсу SPI 302
4.4.5. Мікропроцесорна система збору та накопичення даних на карту пам’яті SD/MMC 307
4.4.6. Сполучення контролерів Arduino по інтерфейсу SPI 310
Завдання до розділу 4 313

Розділ 5. Апаратно-програмні засоби сполучення мікропроцесорних систем з комп’ютером 315
5.1. Послідовний інтерфейс RS-232 315
5.1.1. Загальна характеристика інтерфейсу RS-232 316
5.1.2. Формати передачі даних та режими зв’язку 323
5.1.3. Програмування COM-портів та реалізація протоколу обміну даними по інтерфейсу RS-232 325
5.1.4. Програмні засоби керування гусеничним транспортним роботом по послідовному інтерфейсу 341
5.1.5. Використання Bluetooth-радіомодему з інтерфейсом RS-232 для дистанційного керування мобільним роботом 347
5.2. Шина USB 354
5.2.1. Загальна характеристика інтерфейсу USB 354
5.2.2. Методи передачі даних по інтерфейсу USB 357
5.2.3. Спеціалізована мікросхема-перетворювач інтерфейсів FT232BM 359
5.2.4. Мікропроцесорна система збору даних з інтерфейсом USB на базі мікросхеми FT232BM 361
Завдання до розділу 5 374

Розділ 6. Принципи організації збору даних та дистанційного керування обладнанням по мережі Ethernet 375
6.1. Мережева модель OSI та протоколи обміну даними 375
6.2. Особливості використання Windows Sockets для організації обміну даними по мережі 378
6.3. Розробка програмних засобів обміну повідомленнями по мережі Ethernet 387
6.4. Дистанційне керування роботом-маніпулятором по мережі Ethernet 394
Завдання до розділу 6 411

Розділ 7. Мікропроцесорні системи керування, збору та обробки інформації різного призначення 412
7.1. Організація та застосування мікропроцесорних систем керування, збору та обробки інформації 412

7.2. Мікропроцесорна система охоронно-пожежної сигналізації з передачею SMS-повідомлень по мережі мобільного зв’язку 418
7.3. Мікропроцесорна система керування роботизованою платформою з чотирма роликонесучими колесами 435
7.4. Мікропроцесорна система керування рухом роботизованої мобільної платформи по заданій траєкторії 444
7.5. Мікропроцесорна система моніторингу параметрів дизель-генераторного агрегату 452
7.6. Апаратно-програмні засоби керування рухом мобільної платформи за допомогою смартфону з ОС Android 463
7.6.1. Апаратна частина системи радіокерування по інтерфейсу Bluetooth 463
7.6.2. Розробка програмних засобів дистанційного керування для ОС Android у середовищі Android Studio 471
Завдання до розділу 7

Список літератури 488

Вступ

Протягом останніх років на високотехнологічних підприємствах впроваджується все більша кількість мікропроцесорних систем керування, збору та обробки даних, які за своєю суттю є інформаційно-вимірювальними системами автоматизації процесів керування.

Інформаційно-вимірювальні і керуючі цифрові мікропроцесорні системи призначені для вимірювання, збору, обробки, зберігання і відображення інформації з реальних об'єктів. Такі системи використовуються практично у всіх галузях виробництва для контролю і керування різними технологічними процесами. В електронних та радіотехнічних системах, в техніці зв'язку такі системи використовуються для обробки сигналів, функціонального контролю каналів зв'язку, діагностування стану апаратури. Всі системи збору і контролю параметрів будуються за однаковими принципами. Перш за все, це вимірювання деякої фізичної величини за допомогою датчика. Первинна інформація надходить по каналах зв'язку у вигляді аналогової напруги або цифрового коду. Отримані дані обробляються в режимі реального часу з метою ідентифікації неприпустимих значень параметрів з відповідною індикацією та формуванням керуючого впливу, або зберігаються в пристрій пам'яті для подальшої їх обробки.

Типова схема системи керування, збору і обробки даних складається з апаратної частини, яка підключається до об'єкта керування, і програмної частини, яка реалізує отримання, обробку, відображення, запис і зберігання отриманих даних, а також обробку помилкових ситуацій.

Специфіка автоматизації процесів керування полягає в різноманітності об'єктів автоматизації, динамічності спектра розв'язуваних завдань, необхідності забезпечення унікальних характеристик вимірювального і керуючого обладнання, відносно швидкому моральному старінні автоматизованих систем.

При побудові масштабних систем керування, збору і обробки даних досить часто зустрічаються завдання, для яких не знаходиться оптимального рішення в рамках, запропонованих одним постачальником апаратно-програмних засобів. Тому виникають проблеми сумісності різнорідного устаткування і програмного забезпечення. Таким чином, найбільш ефективним є використання програмного забезпечення і устаткування, взаємодія між якими може здійснюватися з використанням відкритих стандартів.

Метою пропонованого підручника є ознайомлення студентів з принципами побудови мікропроцесорних систем керування, збору і обробки інформації. Побудова системи на одному з простих мікропроцесорних наборів дозволить в подальшому проектувати більш сучасні і швидкодіючі пристрої. Знайомство читачів з мікропроцесорними системами та їх програмуванням починається з апаратно-програмних засобів Arduino. Вивчивши принципи отримання та обробки даних з

датчиків, що найбільш часто використовуються, читачеві пропонується більш детально розібратися в процесах, що відбуваються всередині мікропроцесорних систем керування, збору і обробки даних, познайомитися з популярними цифровими інтерфейсами передачі даних.

Автоматизовані системи управління постійно вдосконалюються по мірі еволюції технічних засобів і програмного забезпечення, змінюється характер об'єктів і методів керування, засобів автоматизації та інших компонентів, які складають зміст сучасної системи управління. Між об'єктом і диспетчером наявний цілий ряд вимірювальних систем, виконавчих механізмів, засобів телемеханіки, мнемосхем та інших засобів відображення інформації. Активно розвиваються людино-машинні системи керування, методи і моделі дослідження операцій, системи диспетчерського управління на основі використання автоматичних інформаційних систем збору даних і сучасних обчислювальних комплексів. Диспетчер в багаторівневій автоматизованій системі керування технологічним процесом отримує інформацію з монітора комп'ютера або з електронної системи відображення інформації, і впливає на об'єкти, що знаходяться від нього на значній відстані, за допомогою телекомунікаційних систем, контролерів, інтелектуальних виконавчих механізмів. Основною необхідною умовою ефективної реалізації диспетчерського управління, що має яскраво виражений динамічний характер, стає робота з інформацією, тобто з процесами збору, передачі, обробки та відображення інформації. Слід також враховувати той факт, що велика кількість датчиків, мікросхем пам'яті виготовляються з цифровим виходом і стандартними протоколами низького рівня для забезпечення безпосереднього зв'язку з мікроконтролером. Тому в підручнику значна увага приділяється особливостям розробки програмного забезпечення для комп'ютера і його зв'язку з мікропроцесорними системами збору даних для вирішення завдань керування різноманітним обладнанням з використанням найбільш поширених комунікаційних інтерфейсів. Метою схем і прикладів програм, які приводяться в підручнику, є набуття студентами практичних навичок з проектування та програмування електронних систем керування, збору і обробки інформації різного призначення. Передбачається, що студенти володіють основами аналогової і цифрової схемотехніки, а також базовими знаннями в програмуванні на мові Сі.

Підручник призначений для студентів спеціальностей 171 – «Електроніка», 172 – «Телекомунікації та радіотехніка» і суміжних спеціальностей, а також всіх тих, хто цікавиться комп'ютерною електронікою і програмуванням.

Перелік умовних скорочень

АКД – апаратура каналів даних
АКТ – автоматичний керований тягач
АПД – апаратура передавання даних
АРМ – автоматизоване робоче місце
АСУ – автоматизована система управління
АСУ ТП – автоматизована система управління технологічним процесом
АЦП – аналого-цифровий перетворювач
ВР – виконавче реле
ГА – генераторний агрегат
ГРЩ – головний розподільний щит
ДАУ – дистанційне автоматичне управління
ДГ – датчик газу
ДГА – дизель-генераторний агрегат
ДД – дискретний датчик
ДЖ – джерело живлення
ДР – датчик руху
ДТ – датчик температури
ЕРС – електрорушійна сила
ІВС – інформаційно-вимірювальна система
ІТЗ – інженерно-технічні засоби
ІЧ – інфрачервоний
КСУ – комп'ютеризована система управління
ЛМІ – людино-машинний інтерфейс
ЛОМ – локальна обчислювальна мережа
ОЗП – оперативний запам'ятовуючий пристрій
ОПС – охоронно-пожежна сигналізація
ОС – операційна система
ПЕ – підсилювальний елемент
ПЗ – програмне забезпечення
ПЗП – постійний запам'ятовуючий пристрій
ПК – персональний комп'ютер
ПКП – приймально-контрольні прилади
ПЛК – програмований логічний контролер
РКІ – рідкокристалічний індикатор
СЕЕС – суднова електроенергетична система
СЗОІ – системи збору та обробки інформації
СУ – система управління
ТП – технологічний процес
ТТЛ – транзисторно-транзисторна логіка
ФАПЧ – фазове автопідлаштування частоти
ЦАП – цифро-аналоговий перетворювач
ШІМ – широтно-імпульсна модуляція

РОЗДІЛ 1. ОСНОВИ ПРОГРАМУВАННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ КЕРУВАННЯ, ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ

1.1. Апаратно-програмні засоби Arduino для побудови систем керування, збору та обробки інформації

Arduino – це інструмент для проектування електронних пристроїв та систем збору, обробки даних та керування (електронний конструктор), який щільно взаємодіє з навколишнім фізичним середовищем. Arduino застосовується для створення електронних пристроїв з можливістю прийому сигналів від різних цифрових і аналогових датчиків, які можуть бути підключені до нього, і управління різними виконавчими пристроями. Проекти пристроїв, засновані на Arduino, можуть працювати самостійно або взаємодіяти з програмним забезпеченням на комп'ютері. Для програмування Arduino використовується програмне забезпечення (ПЗ) з можливістю розширення – ПЗ Arduino випускається як інструмент, який може бути доповнений досвідченими користувачами. Мова програмування може доповнюватися бібліотеками C++. Користувачі, які бажають зрозуміти технічні нюанси, мають можливість перейти на мову AVR C на якому заснований C++. Відповідно, є можливість додати код з середовища AVR-C в програму для Arduino.

Середовище розробки Arduino складається з вбудованого текстового редактора програмного коду, області повідомлень, вікна виведення тексту (консолі), панелі інструментів з кнопками команд, які часто використовуються, і декількох меню. Для завантаження програм і зв'язку середовище розробки підключається до апаратної частини Arduino.

На сьогоднішній день за допомогою Arduino конструюють різні інтерактивні, навчальні, експериментальні, розважальні моделі, електронні й телекомунікаційні системи та пристрої. Інтерфейс середовища розробки Arduino IDE порівняно простий в користуванні, його основою є мова C++, тому освоїти інструментарій можуть навіть програмісти початківці.

Сама програма надається з відкритим вихідним кодом. Всі базові інструменти поширюються безкоштовно – достатньо вибрати версію та відповідну операційну систему. Останню версію програмного забезпечення (ПЗ) Arduino IDE можна знайти за цим посиланням:

http://www.arduino.cc/en/Main/Software

На рис. 1.1 представлена головна сторінка сайту, з якого можна завантажити ПЗ Arduino IDE, натиснувши на відповідне посилання (на рисунку показано стрілкою).

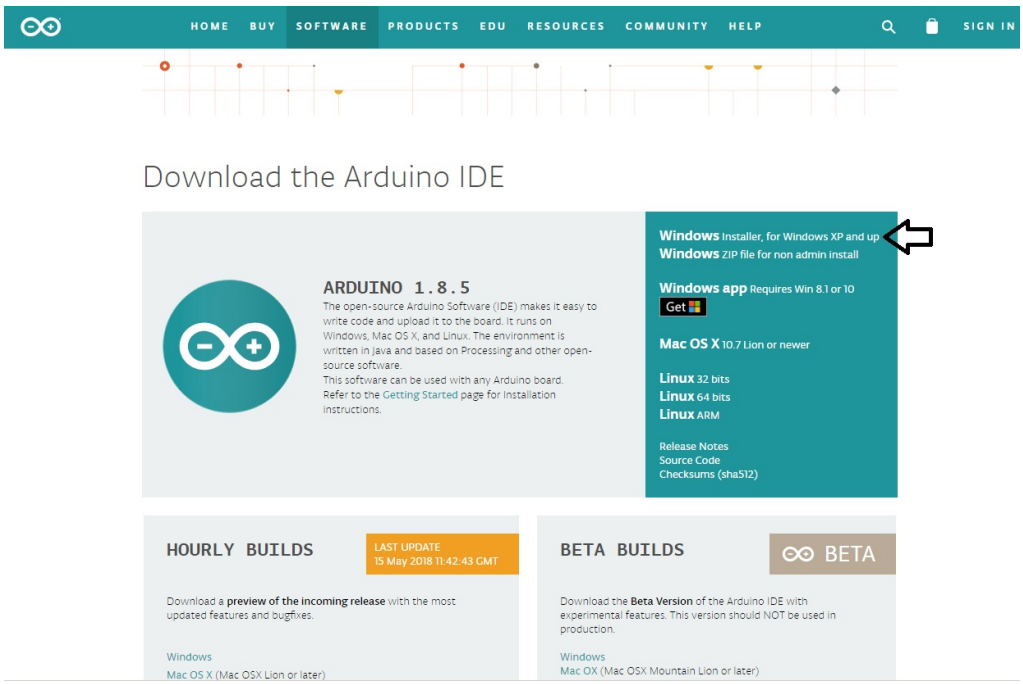


Рис. 1.1.

Після завантаження програми на комп'ютер, її потрібно установити. Процес установки програмного забезпечення такий самий, як для більшості програм для ОС Windows. Під час установки всі параметри рекомендується залишити за замовчуванням. Після завершення процесу установки і запуску програмного забезпечення Arduino IDE відкриється головне діалогове вікно програми, представлене на рис. 1.2.

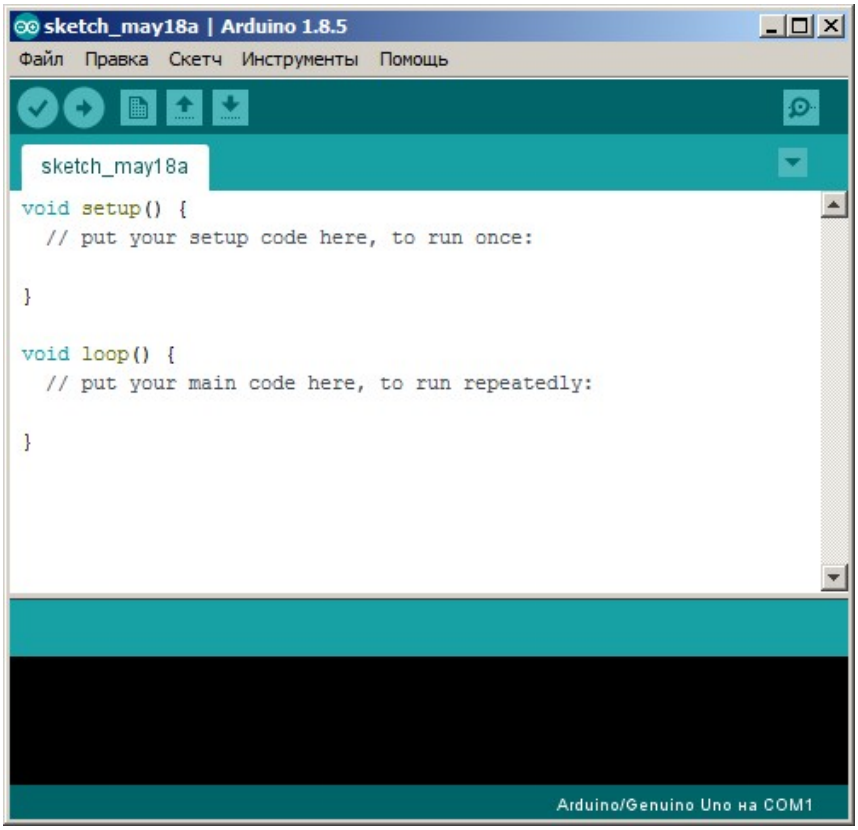


Рис. 1.2.

Програма, написана в середовищі Arduino, називається скетч. Скетч пишеться в текстовому редакторі, що має інструменти вирізки/вставки, пошуку/заміни тексту. Під час збереження і експорту проекту в області повідомлень з'являються пояснення. Також можуть відображатися наявні в програмі помилки. Вікно виведення тексту (консоль) показує повідомлення Arduino, що включають повні звіти про помилки та іншу інформацію. Кнопки панелі інструментів дозволяють перевірити і записати програму, створити, відкрити і зберегти скетч, відкрити моніторинг послідовної шини.

Середовищем Arduino використовується принцип блокнота: стандартне місце для зберігання програм (скетчів). Скетчі з блокнота відкриваються через меню File командою Sketchbook або кнопкою Open на панелі інструментів. При першому запуску програми Arduino автоматично створюється директорія для блокнота. Розташування блокнота змінюється через діалогове вікно налаштувань, викликати яке можна вибравши в меню «Файл» пункт «Настройки».

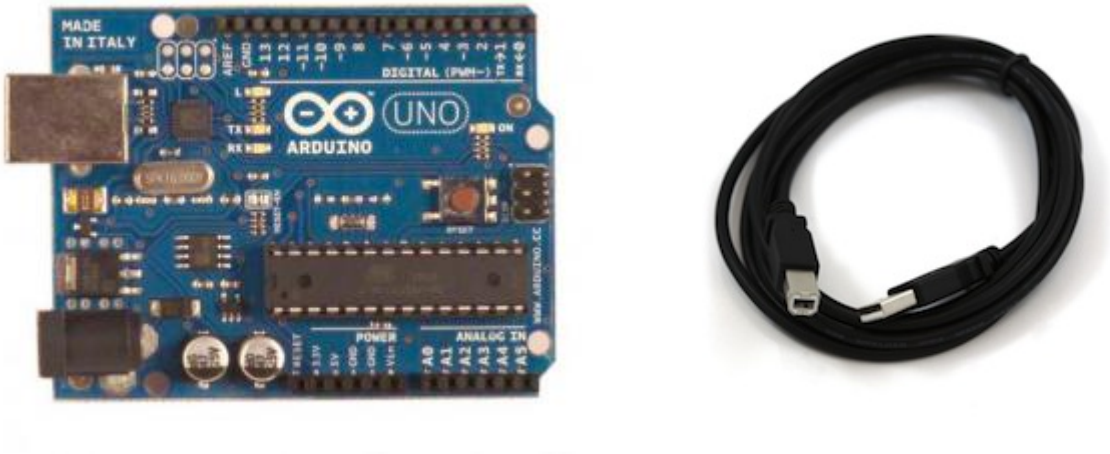
В табл. 1.1 приводиться опис кнопок головного вікна програми.

Таблиця 1.1.

 Перевірити	Перевірка програмного коду на помилки, компіляція.
 Завантаження	Компілює програмний код і завантажує його в пристрій Arduino. Опис завантаження наведено нижче.
 Новий	Створення нового скетчу.
 Відкрити	Відкриття меню доступу до всіх скетчам в блокноті. Відкривається натисканням в поточному вікні.
 Зберегти	Збереження скетчу.
 Монітор порта	Відкриття моніторингу послідовної шини (Serial monitor).

Існує кілька версій платформ Arduino. Версія Leonardo базується на мікроконтролері ATmega32u4. Uno, як і попередня версія Duemilanove побудовані на мікроконтролері Atmel ATmega328. Більш ранні версії платформи Diecimila і перша робоча Duemilanoves були розроблені на основі Atmel ATmega168, більш ранні версії використовували ATmega8. Arduino Mega2560, в свою чергу, побудована на мікроконтролері ATmega2560.

Для початку роботи потрібна плата Arduino Uno (рис. 1.3, а), або інша, та кабель для її підключення до комп’ютера (рис. 1.3, б). Потрібен кабель стандарту USB (з роз'ємами типу USB-A і USB-B).



а б

Рис. 1.3.

Arduino Uno, Mega, Duemilanove і Arduino Nano отримують живлення автоматично від будь-якого USB-підключення до комп'ютера або іншого джерела живлення. Підключіть плату Arduino до комп'ютера, використовуючи USB-кабель. На платі повинний засвітитися світлодіод живлення, позначений PWR.

Далі необхідно установити драйвер. Установка драйверів для Arduino Uno на Windows7, Vista або XP виконується наступним чином:

1. Підключіть плату і почекайте, поки Windows почне процес установки драйвера. Через деякий час, не дивлячись на всі її спроби, процес закінчиться безрезультатно.

2. Натисніть на кнопку ПУСК і відкрийте Панель управління. В панелі управління перейдіть на вкладку Система і безпека (System and Security). Потім виберіть Система. Коли відкриється вікно Система, виберіть Диспетчер пристроїв (Device Manager).

3. Зверніть увагу на вузол «Другие устройства». Ви побачите новий пристрій, позначений як USB2.0-Serial, як показано на рис. 1.4.

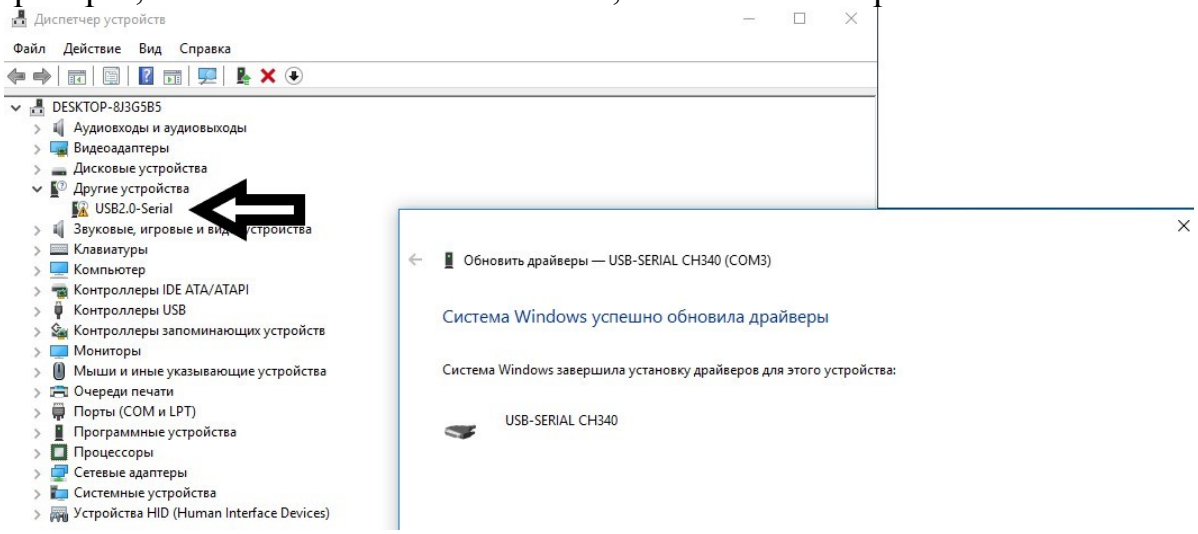


Рис. 1.4.

4. Натисніть на назві «USB2.0-Serial» правою кнопкою миші і виберіть опцію «Оновити драйвер» (Update Driver Software).

5. У діалоговому вікні, що з'явиться, виберіть опцію автоматичного пошуку та оновлення драйверів. Операційна система Windows закінчить установку драйвера.

6. Відкрийте вузол «Порты (COM и LPT)», і запам'ятайте номер послідовного порту, який виділено операційною системою для зв'язку з Arduino, як показано на рис. 1.5. В даному випадку це COM3. Цей порт потрібно буде вказати в середовищі Arduino IDE при програмуванні пристрою.

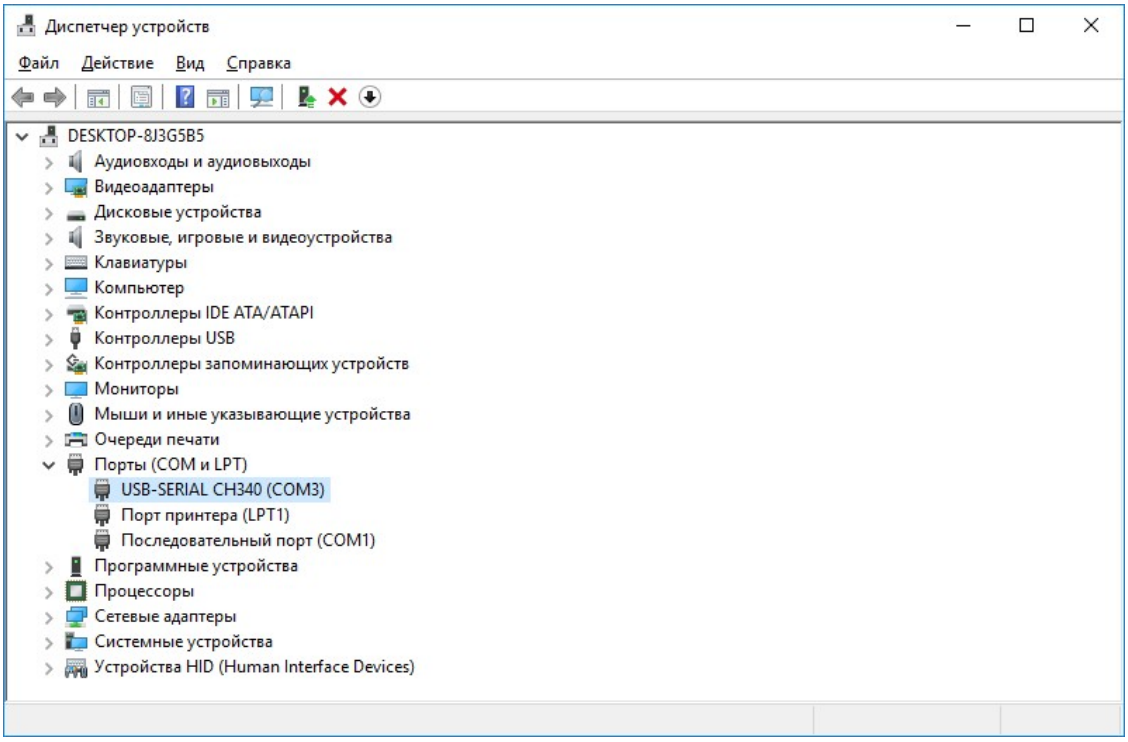


Рис. 1.5.

Перед завантаженням скетчу потрібно задати необхідні параметри в меню Tools пункт Board і меню Tools пункт Serial Port. В ОС Windows порти можуть позначатися як COM1 або COM2 (для плати послідовної шини) або COM3, COM4, COM5 і вище (для плати USB). Визначення порту USB проводиться в поле Послідовної шини USB Диспетчера пристроїв Windows.

Після вибору порту і платформи необхідно натиснути кнопку завантаження на панелі інструментів або вибрати в меню File пункт Upload to I/O Board. Сучасні платформи Arduino перезавантажуються автоматично перед завантаженням. На більшості плат під час процесу будуть блимати світлодіоди RX і TX. Середовище розробки Arduino виведе повідомлення про закінчення завантаження або про помилки.

При завантаженні скетчу використовується Завантажувач (Bootloader) Arduino – невелика програма, що завантажується в мікроконтролер на платі. Вона дозволяє завантажувати програмний код без

використання додаткових апаратних засобів. Завантажувач (Bootloader) активний протягом декількох секунд при перезавантаженні платформи і при завантаженні будь-якого з скетчів в мікроконтролер. Робота завантажувача (Bootloader) розпізнається по блиманню світлодіода (13 пін), наприклад, при перезавантаженні мікроконтролера на платі.

Для початку роботи відкриємо готовий приклад. Для цього в меню файл потрібно вибрати команду Приклади, як показано на рис. 1.6. Приклад демонструє блимання світлодіода, розташованого на платі, з періодом 1 секунда.

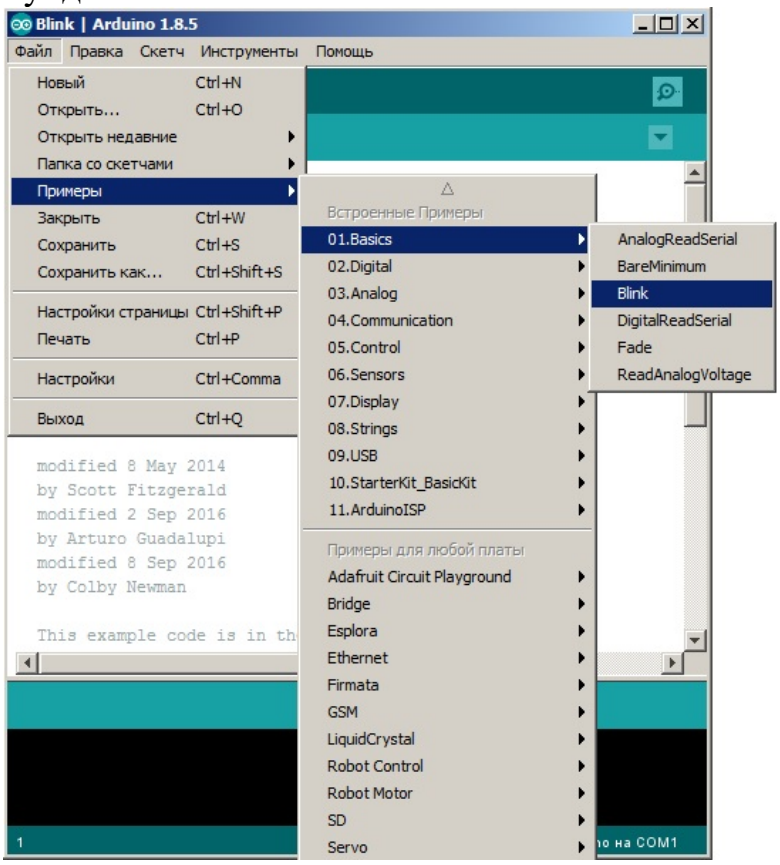


Рис. 1.6.

Після цього потрібно вибрати плату, в яку буде завантажено програму. В нашому випадку це Arduino UNO. На рис. 1.7 показано, як це можна зробити.

Виберіть пристрій послідовної передачі плати Arduino з меню Tools пункт Serial Port. Ймовірно, це буде COM3 або вище (COM1 і COM2 зазвичай резервуються для апаратних COM-портів). Щоб знайти потрібний порт, ви можете від'єднати плату Arduino і повторно відкрити меню. Пункт, який зник, і буде портом плати Arduino. Підключіть плату і виберіть послідовний порт (рис. 1.8).

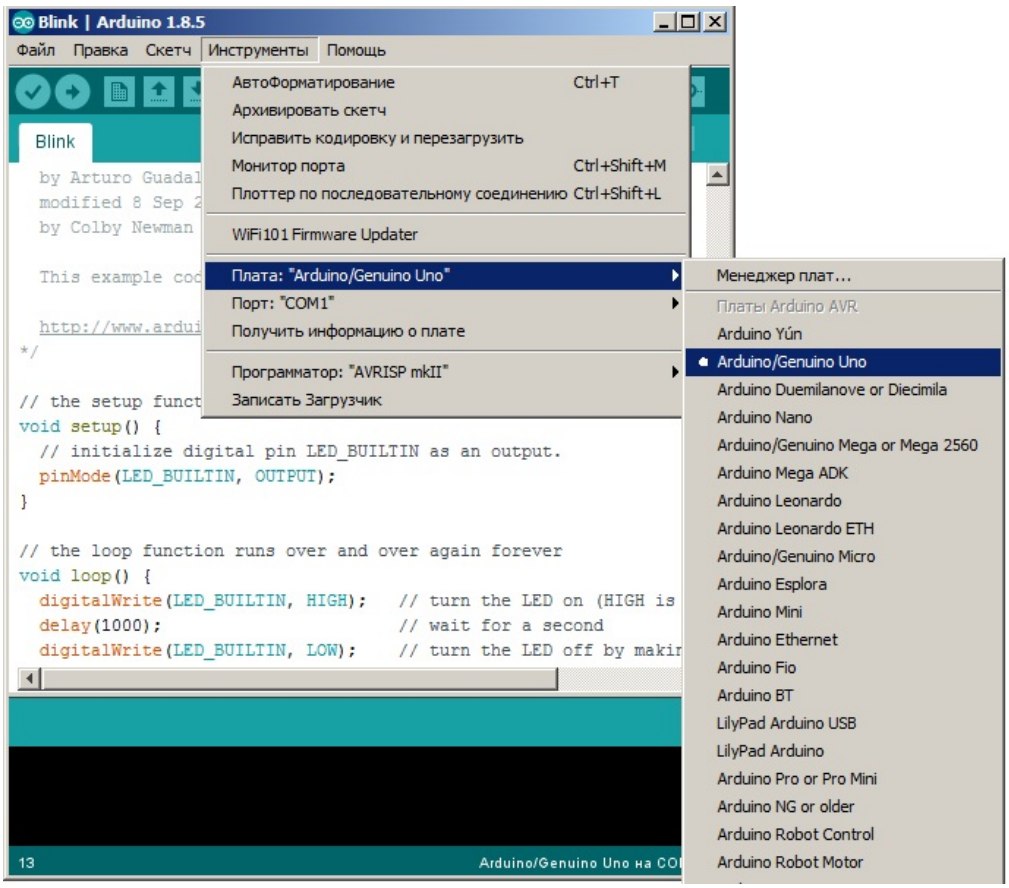


Рис. 1.7.

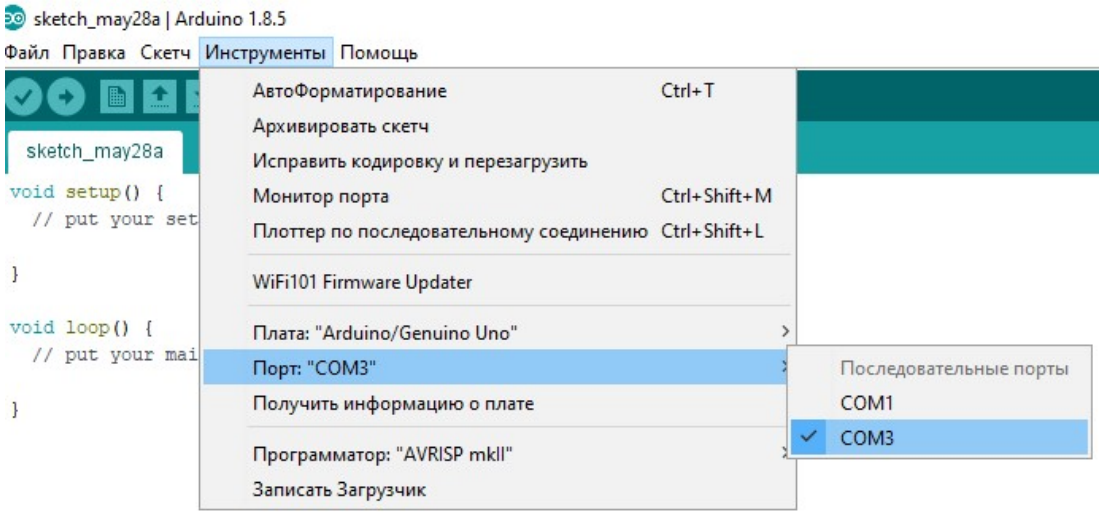


Рис. 1.8.

Тепер потрібно просто натиснути кнопку «Загрузить» в програмі – середовищі розробки Arduino IDE. Через декілька секунд можна побачити блимання світлодіодів RX і TX на платі. У разі успішного завантаження в рядку стану з'явиться повідомлення «Done uploading (Завантаження виконано)».

Через декілька секунд після закінчення завантаження можна побачите як світлодіод, підключений до виводу 13 (L) на платі почне

блимати помаранчевим кольором з періодом 1 секунда. Налаштування середовища розробки завершено.

З мовою програмування Arduino, операторами та функціями можна ознайомитися на сайті за посиланням: http://arduino.ru/Reference

1.2. Основи програмування та моделювання пристроїв Arduino

Мова програмування пристроїв Arduino заснована на C/C++. Вона проста в освоєнні, і на даний момент Arduino – це, мабуть, найпростіший спосіб програмування пристроїв на мікроконтролерах.

Програмувати Arduino можна без наявності самого Arduino. Для цього використовується середовище моделювання Proteus. В стандартних бібліотеках середовища моделювання електронних систем Proteus. відсутня бібліотека для роботи з Arduino. Тому перед початком роботи її слід завантажити з сайту:

https://www.theengineeringprojects.com/2015/12/arduino-uno-library-proteus.html

На рис. 1.9 представлено фрагмент сторінки цього сайту та позначено посилання, натиснувши на яке можна завантажити бібліотеку Arduino для Proteus.

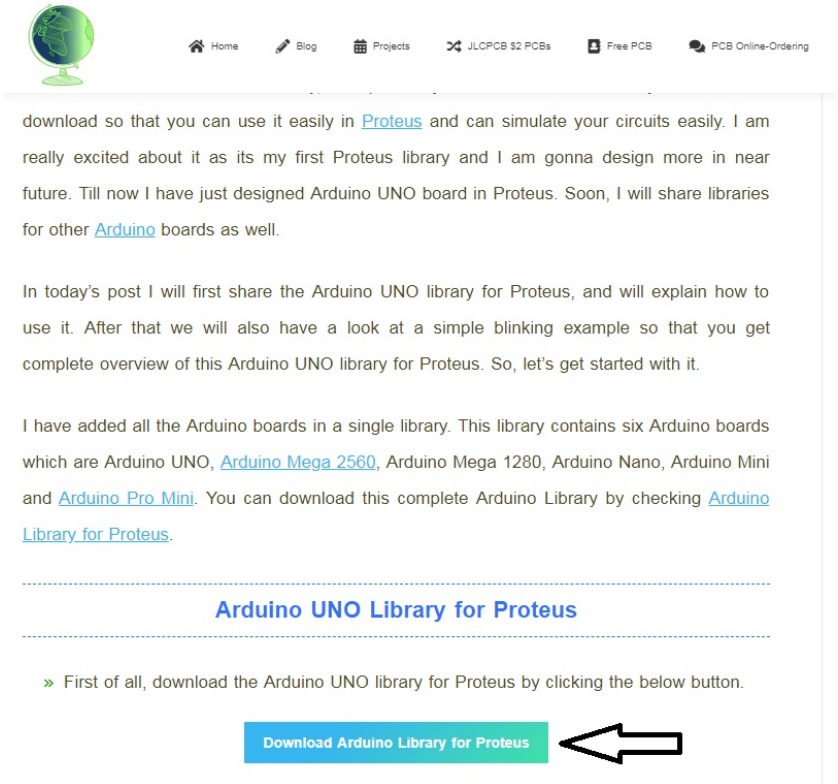


Рис. 1.9.

Також на цьому сайті можна завантажити бібліотеки різноманітних модулів та датчиків для Arduino: ультразвуковий та інфрачервоний сенсор, Bluetooth-модуль, приймач GPS, GSM-радіомодем, датчики газу, полум'я

та інші. Додаткові модулі розширюють функціональність Arduino для управління різними пристроями, отримання даних і т.д. Плата розширення WiFi використовується для з'єднання з бездротовими мережами стандарту 802.11. Плата розширення Xbee Shield забезпечує за допомогою модуля Maxstream Xbee Zigbee бездротовий зв'язок з кількома пристроями Arduino в радіусі до 35 метрів (в приміщенні) і до 90 метрів (поза приміщенням). Плата розширення Motor Shield забезпечує управління двигунами постійного струму і зчитування даних з датчиків положення. Плата розширення Ethernet Shield забезпечує підключення до Інтернету. На рис. 1.10 представлено деякі доступні моделі датчиків та модулів розширення для Proteus.

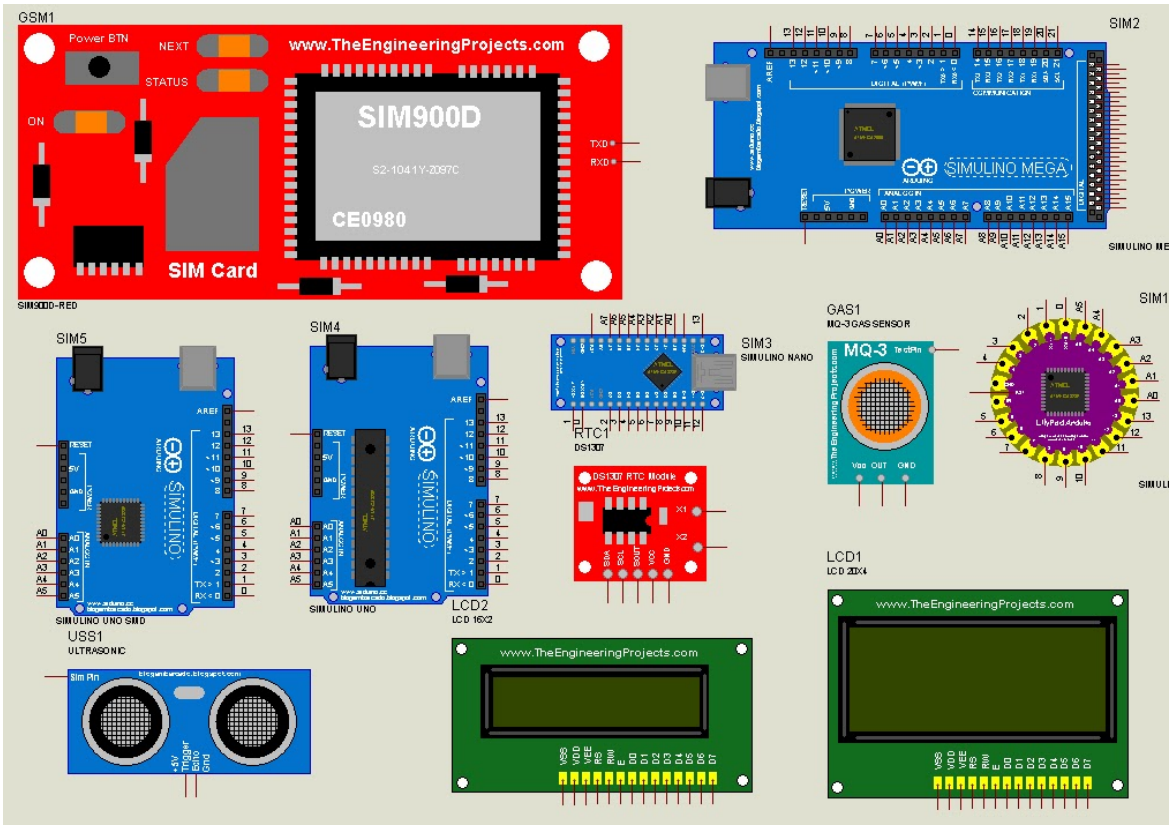


Рис. 1.10.

Після завантаження архіву з файлами моделей його необхідно розпакувати. В результаті в каталозі, в який відбувається розпаковка архіву, з’явиться 2 файли: ArduinoUnoTEP.LIB та ArduinoUnoTEP.IDX. Ці файли необхідно скопіювати в каталог з бібліотеками Proteus, який називається Library. Детальну інструкцію по підключенню в Proteus бібліотек різних модулів для роботи з Arduino можна знайти за посиланням: http://www.instructables.com/id/How-to-add-Arduino-Library-in-to-Proteus-7-8/

Велика кількість проектів Arduino для початківців доступна за посиланням:

https://www.theengineeringprojects.com/2017/07/arduino-projects-for-beginners.html

1.2.1. Керування портами введення/виведення мікроконтролера

Для моделювання роботи Arduino в середовищі Proteus спочатку потрібно скласти схему. Для цього необхідно запустити середовище моделювання Proteus, і в головному вікні програми натиснути кнопку з літерою «P» для додавання елемента в схему, як показано на рис. 1.11.



Рис. 1.11.

У вікні додавання електронного компонента, що з'явиться, в списку категорій ліворуч потрібно обрати категорію Arduino, а в списку праворуч модель пристрою, як показано на рис. 1.12. В даному випадку це Arduino UNO.

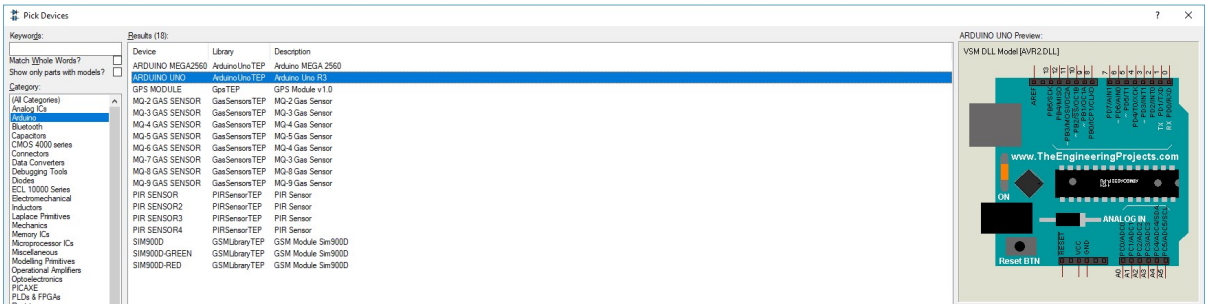


Рис. 1.12.

Аналогічне потрібно виконати для додавання в схему світлодіода та резистора. Пришвидшити процес пошуку елемента в бібліотеці можна шляхом введення його назви в поле Keywords у вікні додавання компонента (рис. 1.12). Це поле вводу знаходиться зверху ліворуч. Для пошуку резистора потрібно ввести RES, а для пошуку світлодіода потрібно ввести LED-GREEN.

Для додавання в схему джерела живлення та загального провідника потрібно на робочому полі середовища Proteus натиснути праву кнопку миші, і з контекстного меню, що з'явиться, вибрати пункт Place – Terminal – GROUND, як показано на рис. 1.13.

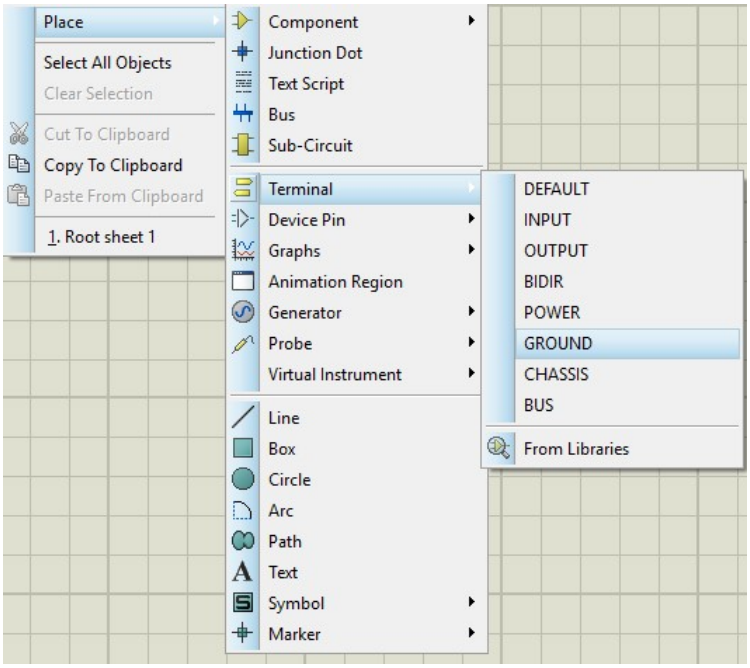


Рис. 1.13.

Далі ці елементи потрібно розташувати на робочому полі та з'єднати, як показано на рис. 1.14.

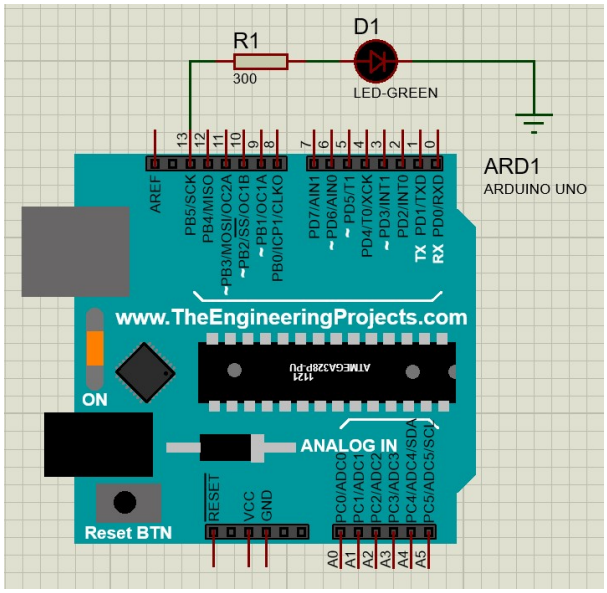


Рис. 1.14.

Наступний крок, який необхідно зробити, це вказати файл «прошивки» мікроконтролера з відкомпільованою програмою. Цей файл має розширення .hex. За замовчуванням при компіляції програми в середовищі Arduino такий файл не створюється. Тому потрібно дещо змінити налаштування програми. В меню Файл потрібно вибрати пункт Налаштування. У вікні, що з'явиться, потрібно встановити позначку Компіляція, як показано на рис. 1.15.

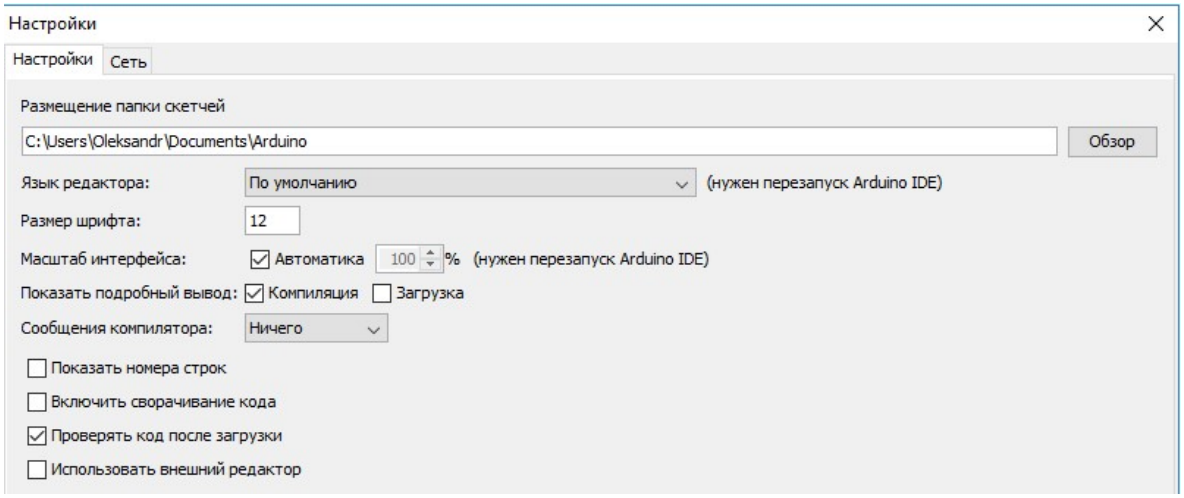


Рис. 1.15.

Тепер, після компіляції програми у програмному забезпеченні Arduino, у вікні повідомлень у нижній частині вікна програми з'явиться додаткова інформація, а саме інформація про місце розташування файлу «прошивки». В даному випадку це файл «Blink.ino.hex», що можна побачити на рис. 1.16.

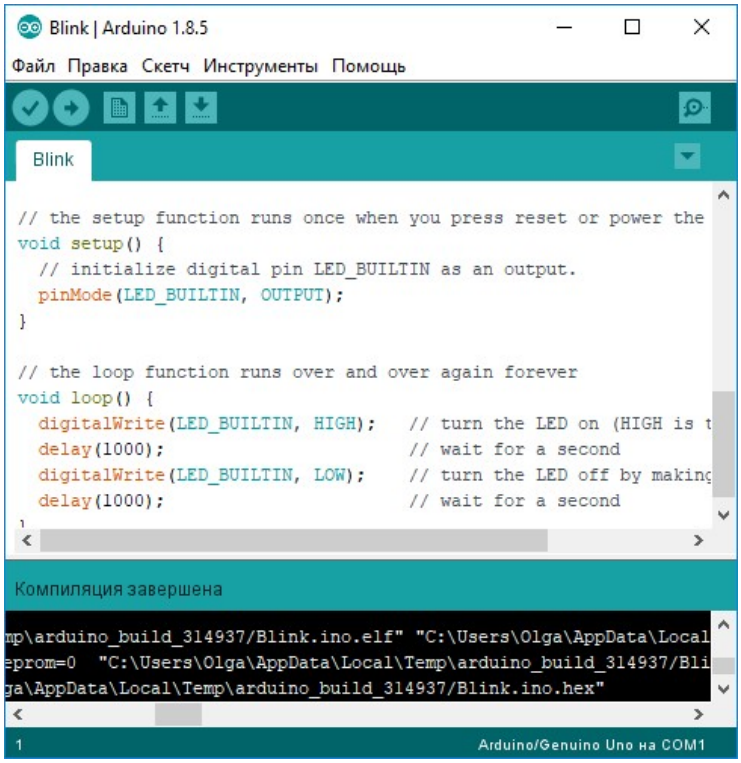


Рис. 1.16.

З цього вікна можна скопіювати весь рядок, який вказує шлях розташування файлу на диску, і вставити його в поле вводу Program File у вікні властивостей компонента, яке можна визвати подвійним натисканням лівої кнопки миші на самому компоненті схеми. На рис. 1.17 представлене вікно властивостей моделі плати Arduino UNO в середовищі Proteus, де потрібно вказати шлях до файлу «прошивки», що має розширення .hex.

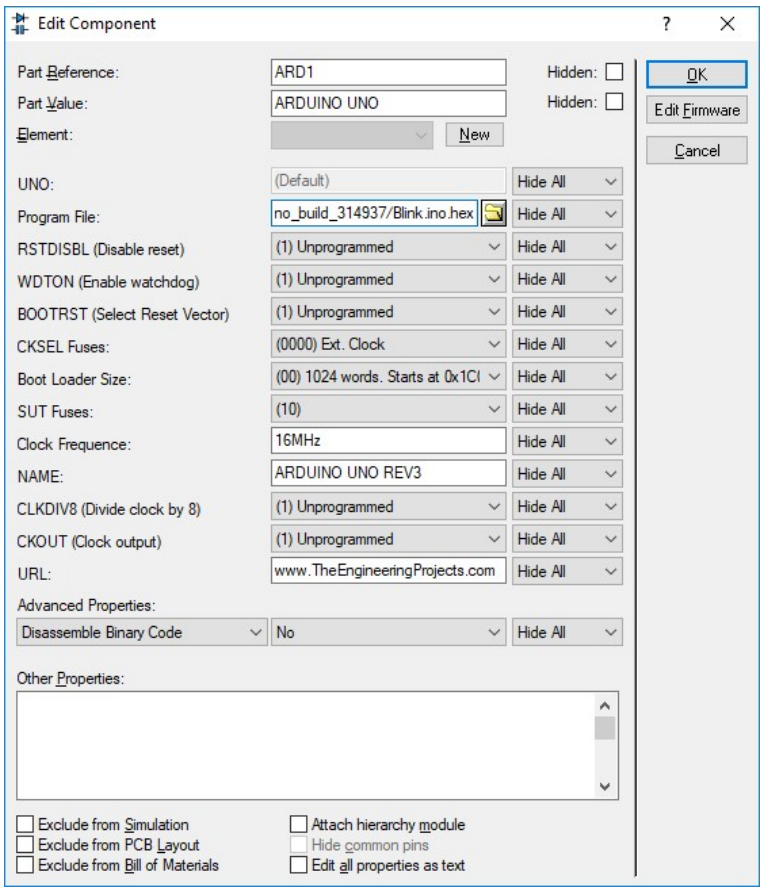


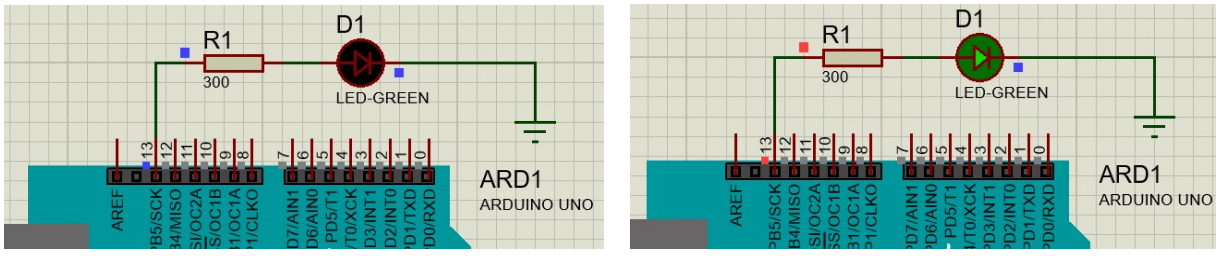
Рис. 1.17.

Тепер схему можна моделювати. Для цього потрібно натиснути на кнопку Run the simulation із зображенням стрілки, яка розташована в правому нижньому куті головного вікна програми (перша кнопка на рис. 1.18).



Рис. 1.18.

Після запуску процесу моделювання у вікні Proteus можна спостерігати зміну стану світлодіода, яка відбувається з періодом 1 секунда. На рис. 1.19, а, б, представлено два різних стани світлодіода в процесі моделювання роботи схеми.



а б

Рис. 1.19.

Текст програми, що виконує блимання світлодіодом з періодом 1 секунда, представлено нижче.

```
void setup() {
  // налаштування виводу LED_BUILTIN на вихід.
  pinMode(LED_BUILTIN, OUTPUT);
}

// цикл виконується весь час, доки є живлення пристрою
void loop() {
  // увімкнути світлодіод (на виводі високий рівень напруги HIGH)
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);              // виклик функції затримки 1 секунда = 1000 мілісекунд
  // вимкнути світлодіод (на виводі низький рівень напруги LOW)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);              // виклик функції затримки 1 секунда = 1000 мілісекунд
}
```

Для перевірки роботи програми на реальному пристрої потрібно зібрати схему, що представлена на рис. 1.20. Для завантаження програми в контролер потрібно натиснути кнопку «Загрузить», яка розташована на панелі інструментів середовища розробки Arduino IDE. Також не слід забувати в меню «Инструменты» головного вікна програми вказати послідовний комунікаційний порт, до якого підключено контролер Arduino. Це може бути порт COM3, COM4 або інший. На початку розділу було описано, яким чином можна дізнатися номер порту.



Рис. 1.20.

Функція setup() викликається, коли стартує скетч. Використовується для ініціалізації змінних, визначення режимів роботи виводів, запуску використовуваних бібліотек і т.д. Функція setup запускається тільки один раз, після кожної подачі живлення або скидання плати Arduino.

Після виклику функції setup(), яка ініціалізує і встановлює початкові значення, функція loop() робить те, що означає її назва – виконується в циклі, дозволяючи програмі здійснювати обчислення і реагувати на них. Її потрібно використовувати для управління платою Arduino. Саму в функції loop() розташовується основна логіка програми.

Функція digitalWrite() подає HIGH (високий рівень, логічна 1) або LOW (низький рівень, логічний 0) значення на цифровий вхід/вихід (pin).

Якщо вхід/вихід (pin) був встановлений в режим вихід (OUTPUT) функцією pinMode(), то для значення HIGH напруги на відповідному вході/виході (pin) буде 5В (3,3В для 3,3V плат), і 0В (земля) для LOW.

Якщо вхід/вихід (pin) був встановлений в режим вхід (INPUT), то функція digitalWrite() зі значенням HIGH буде активувати внутрішній 20K навантажувальний резистор. Подача LOW, в свою чергу, відключає цей резистор. Якщо світлодіод працює, але яскравість низька, можливо необхідно встановити режим «вихід» (OUTPUT) функцією pinMode().

Синтаксис: digitalWrite(pin, value);

Параметри: pin: номер вхід/вихід (pin); value: значення HIGH або LOW.

Функція delay() зупиняє виконання програми на задану в параметрі кількість мілісекунд (1000 мілісекунд в 1 секунді). Більшість активності плати зупиняється функцією delay(). Проте робота переривань не зупиняється, триває запис послідовно (serial) переданих даних на RX порту, ШІМ сигнал (analogWrite) продовжує генеруватися на портах.

Синтаксис ; (крапка з комою) використовується для позначення кінця оператора. Забута в кінці рядка крапка з комою призводить до помилки компіляції. Текст помилки може бути або видимим і посилатися на пропущену крапку з комою, або ні. Якщо зустрічається незрозуміла або схожа на нелогічну помилка компіляції, одним з перших дій повинна бути перевірка пропущених точок з комою в коді, безпосередньо в попередньому рядку, в якій компілятор видав попередження.

Фігурні дужки {} (також називаються просто «дужки») – важливий елемент мови програмування Сі. Вони використовуються в кількох різних конструкціях, наведених нижче, і це може іноді збивати з пантелику початківців.

Відкриваюча дужка "{" повинна завжди супроводжуватися закриваючою дужкою "}". Ця умова відома як парність (симетричність) фігурних дужок. Arduino IDE (інтегроване середовище розробника) включає відповідний інструмент для перевірки парності дужок. Досить виділити дужку, або навіть поставити курсор відразу ж за дужку, як буде підсвічена її логічна пара.

Програмісти-початківці або програмісти, які перейшли на Сі з Бейсика, часто вважають використання фігурних дужок лякаючим. Зрештою, одні і ті ж фігурні дужки замінюють оператор RETURN в підпрограмах (функціях), оператор ENDIF в умовних циклах і оператор NEXT в циклах FOR.

Оскільки використання фігурних дужок настільки багатогранно, рекомендованою практикою програмування буде друкувати закриваючу фігурну дужку відразу після того, як надрукована відкриваюча дужка, коли вставляється конструкція, для якої потрібно використовувати фігурні дужки. Потім курсор слід повернути в позицію між фігурними дужками і почати вводити оператори.

Непарні дужки можуть часто приводити до прихованих, незрозумілих помилок компіляції, які складно відстежити у великій програмі. Через їх всебічне використання, дужки також неймовірно важливі в синтаксичній правильності програми і переміщення дужки на один або два рядки часто призводять до значного впливу на логіку програми.

Коментарі – це рядки в програмі, які використовуються для інформування про те, як працює програма. Вони ігноруються компілятором і не експортуються в процесор, таким чином, вони не займають місце в пам'яті мікроконтролера.

Коментарі призначені тільки для того, щоб допомогти зрозуміти (або згадати), як працює програма, або пояснити це іншим. Є два способи позначити рядок як коментар:

```
x = 5;  // це коментар в одному рядку. Все після подвійного слеша – коментар
        // до кінця рядка
/ * Це багаторядковий коментар – використовуйте його для
коментування цілих фрагментів коду
if (gwb == 0) {
// коментар в рядку припустимий всередині багаторядкового коментаря
// але не інший багатостроковий коментар
}
// не слід забувати «закривати» коментарі – вони повинні бути парними
* /
```

Під час експериментів з кодом, коментування частин програми – відповідний спосіб видалення рядків, в яких можуть бути помилки. Так рядки в коді залишаються, але перетворюються в коментарі, і компілятор просто ігнорує їх. Це може бути особливо корисно при локалізації проблеми, або коли не виходить скомпілювати програму, а повідомлення про помилку при компіляції приховано або не містить змісту.

1.2.2. Зчитування даних з портів введення/виведення мікроконтролера

Наступний приклад демонструє процес зчитування інформації з виводу, до якого підключена дискретна кнопка, і обробку події натискання на кнопку. При натисканні на кнопку світлодіод гасне, а при відпусканні кнопки він загоряється. Цей приклад представляє собою інвертор сигналу, відомий з цифрової електроніки. Світлодіод підключений до виводу 13

плати Arduino UNO, а кнопка підключена до виводу номер 2. Схема пристрою представлена на рис. 1.21.

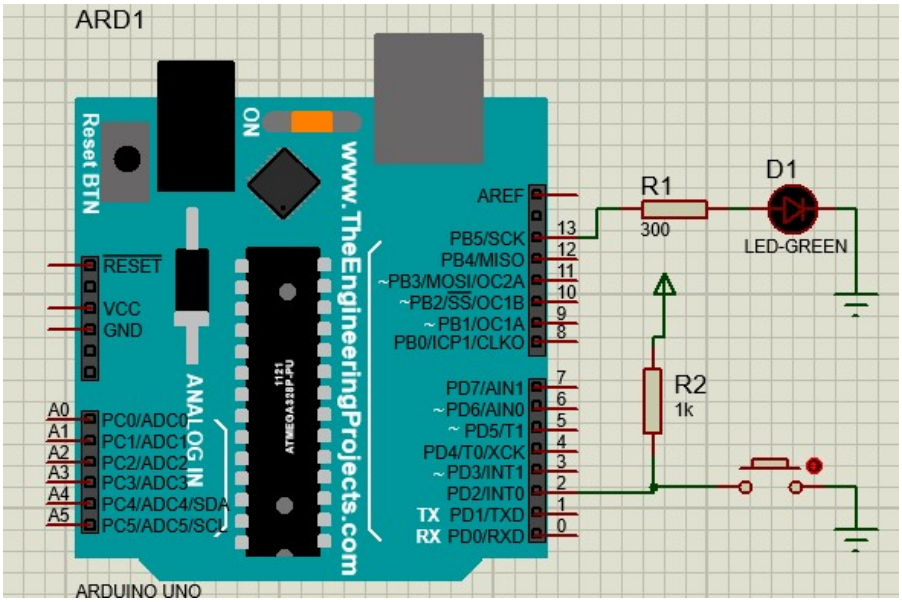


Рис. 1.21.

Якщо кнопка не натиснута, то на вхід номер 2 подається сигнал високого рівня (HIGH), оскільки цей вивід з'єднаний з лінією живлення через резистор опором 1 кОм. Якщо кнопка натиснута, то на цей вхід подається сигнал низького рівня (LOW). Світлодіод загориться у випадку. Коли напруга на його аноді буде більшою за напругу на катоді. Як видно зі схеми, світлодіод підключений до виводу номер 13 плати анодом. Тому для того. Щоб він світився. Потрібно встановити високий рівень сигналу на виводі номер 13. Нижче представлено текст програми.

```
const int buttonPin = 2;    // 2 - номер виводу, до якого підключена кнопка
const int ledPin =  13;      // 13 - номер виводу, до якого підключений світлодіод

// додаткова змінна, в яку буде записано стан кнопки
int buttonState = 0;

void setup() {
  // налаштування виводу, до якого підключено світлодіод, на вихід
  pinMode(ledPin, OUTPUT);
  // налаштування виводу, до якого підключена кнопка, на вхід
  pinMode(buttonPin, INPUT);
}

void loop() {
  // зчитування стану кнопки у змінну
  buttonState = digitalRead(buttonPin);

  // перевірка, чи натиснута кнопка. Якщо так, в змінній buttonState буде HIGH
  if (buttonState == HIGH) {
    // включити світлодіод
```

```
    digitalWrite(ledPin, HIGH);
  } else {
    // виключити світлодіод
    digitalWrite(ledPin, LOW);
  }
}
```

В наведеному вище прикладі використовувались змінні та константи. Константами в мові Arduino називають деякі постійні значення. Вони дозволяють зробити код програми більш легким для сприйняття. Константи поділяють на три групи:

1. Логічні константи – для представлення логічних значень істина і лож: true і false. Значення false визначається як 0, в логічному виразі. Зазвичай кажуть, що true – це 1, це коректно. Але також коректно те, що для integer будь-яке значення, відмінне від 0, буде також true в логічному виразі. Тобто -1, -2 і -200 буде також true в логічному виразі.

2. Завдання значення на вході/виході, HIGH і LOW. Можливі тільки два значення для зчитування або запису на цифровий вхід/вихід: HIGH і LOW. Значення HIGH може позначати дещо різне в залежності від налаштування режиму порту як INPUT або OUTPUT. Коли порт входу/виходу встановлений в режим INPUT за допомогою функції pinMode, і зчитується функцією digitalRead, мікроконтролер видасть значення HIGH при напрузі 3В або вище на зазначеному порту. Також порт може бути встановлений як вхідний (INPUT) функцією pinMode, і потім встановлений в значення HIGH функцією digitalWrite. Це підключить до порту внутрішній підтягуючий резистор опором 20K, що дозволить отримувати постійне значення HIGH при зчитуванні з цього порту, якщо тільки на нього не подається значення LOW (напруга менше 2В, зазвичай 0В – вивід підключений до загального провідника) зовнішнім ланцюгом, підключеним до цього порту.

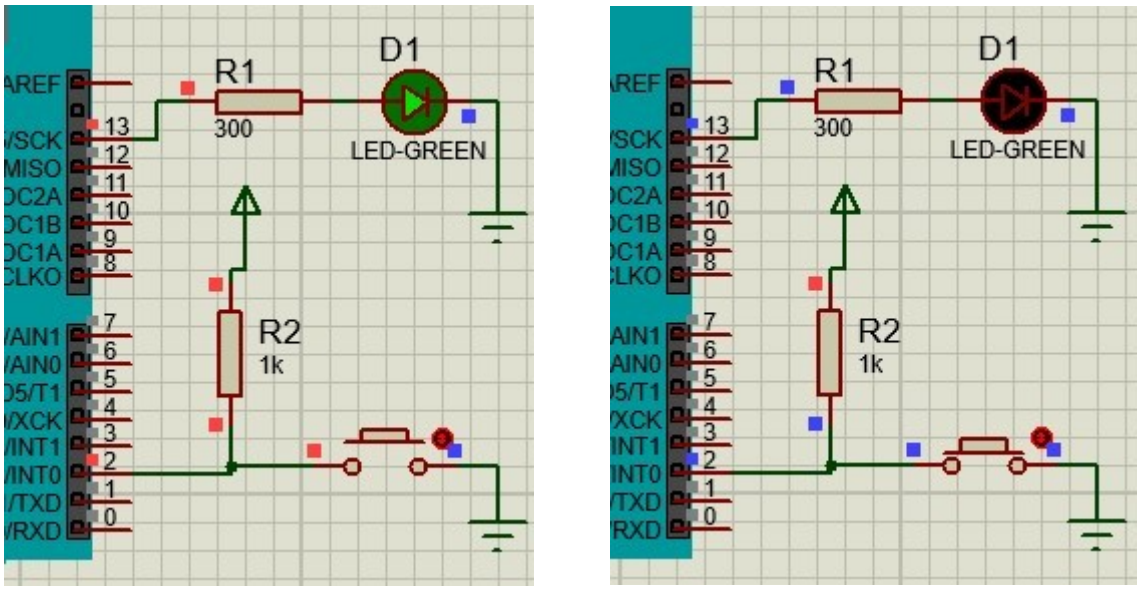
Коли порт входу/виходу налаштований як OUTPUT функцією pinMode, і встановлено значення HIGH функцією digitalWrite, на порту буде постійна напруга 5В (або 3,3В в залежності від напруги живлення схеми). Значення LOW також різне для режиму INPUT і OUTPUT. Коли порт налаштований як INPUT, і з нього зчитується значення функцією digitalRead, мікроконтролер поверне LOW якщо напруга на даному порту менше або дорівнює 2В. Якщо ж порт встановлений в OUTPUT і LOW, то напруга на виході порту буде 0 вольт. Зверніть увагу, що true і false пишуться малими літерами, на відміну від HIGH, LOW, INPUT і OUTPUT.

3. Завдання режиму порту входу/виходу, INPUT і OUTPUT. Цифрові порти введення/виведення можуть бути встановлені в режими роботи як вхід або вихід: INPUT або OUTPUT. Налаштування режиму роботи порту виконується функцією pinMode(). Порти Arduino встановлені в режим INPUT знаходяться в високоімпедансному стані. Це означає, що порт введення створює низьке навантаження на схему, в яку він включений. Це дозволяє використовувати виводи мікросхеми для підключення різних

датчиків. Якщо порт встановлений в режим виходу – OUTPUT, то це означає що він знаходиться в низькоімпедансному стані. Він може пропускати через себе досить великий струм, до 40 мА, достатній для живлення зовнішнього ланцюга, наприклад, світлодіода. У цьому стані порт може бути пошкоджений при замиканні на землю. Додаткову інформацію про цифрові порти мікроконтролера можна знайти за посиланням:

http://arduino.ru/Tutorial/DigitalPins

Після компіляції програми можна перевірити роботу схеми. На рис. 1.22, а, б, представлено зображення схеми в режимі моделювання при різних станах кнопки. Не слід забувати про необхідність вказувати у вікні властивостей моделі плати Arduino UNO в середовищі Proteus шлях до файлу «прошивки», який отримується в результаті компіляції програми.



а б

Рис. 1.22.

Оператор if використовується в поєднанні з операторами порівняння і перевіряє, чи досягнута істинність умови, наприклад, перевищує вхідне значення задане число. Формат оператора if наступний:

```
if (someVariable>50)
{
// виконувати дії
}
```

Програма перевіряє, значення someVariable більше ніж 50 чи ні. Якщо так, то виконуються певні дії. Інакше кажучи, якщо вираз в круглих дужках істинний, виконуються оператори всередині фігурних дужок. Якщо ні, програма пропускає цей код.

Вирази, які обчислюються всередині круглих дужок, можуть складатися з одного або декількох операторів. Оператори порівняння:

```
x==y   (x дорівнює y)
x!=y   (x не дорівнює y)
x<y    (x менше ніж y)
x>y    (x більше ніж y)
x<=y   (x менше або дорівнює y)
x>=y   (x більше або дорівнює y)
```

Слідкуйте, щоб випадково не використовувати простий знак рівності (наприклад, if (x = 10)). Знак рівності – це оператор присвоювання, і встановлює значення x рівним 10 (заносить значення 10 в змінну x). Замість цього використовуйте подвійний знак рівності (наприклад, if (x == 10)), який є оператором порівняння і перевіряє, x дорівнює 10 чи ні. Це пов'язано з тим, що C обчислює вираз if (x = 10) у такий спосіб: значення 10 присвоюється x (пам'ятаємо, що простий знак рівності – це оператор присвоювання), таким чином, x тепер дорівнює 10. Потім умовний if обчислює 10, яке вже має логічне значення ІСТИНА, так як будь-яке число, що не дорівнює 0, визначається як ІСТИНА. Тому if (x = 10) буде завжди мати логічне значення ІСТИНА, яке не є бажаним результатом, коли використовується оператор if. До того ж, змінній x буде присвоєно значення 10, що також не є бажаною дією.

Конструкція if..else надає більший контроль над процесом виконання коду, ніж базовий оператор if, дозволяючи здійснювати кілька перевірок, об'єднаних разом. Ключове слово else дозволяє робити відмінну від зазначеної в if перевірку, щоб можна було здійснювати відразу кілька взаємовиключних перевірок. Кожна перевірка дозволяє переходити до наступного за нею оператора не раніше, ніж отримає логічний результат ІСТИНА. Коли перевірка з результатом ІСТИНА знайдена, запускається вкладений в неї блок операторів, і потім програма ігнорує всі наступні рядки в конструкції if..else. Якщо жодна з перевірок не отримала результат ІСТИНА, за замовчуванням виконується блок операторів в else, якщо останній присутній, і встановлюється дія за замовчуванням.

Конструкція else if може бути використана з або без заключного else і навпаки. Допускається необмежене число таких переходів else if. Приклад використання конструкції else if представлено нижче.

```
if (pinFiveInput<500) // якщо значення pinFiveInput менше 500
{
 // виконувати дію A
}
else if (pinFiveInput> = 1000) // якщо pinFiveInput більше або дорівнює 1000
{
 // виконувати дію B
}
else // якщо жодна з попередніх умов не виконується, тоді виконується цей блок
```

```
{
 // виконувати дію C
}
```

1.2.3. Програмна реалізація базових логічних елементів

Наступний приклад демонструє реалізації елементу 2-І, добре відомого з цифрової електроніки. Цей елемент має 2 входи і один вихід. На виході елемента встановлюється рівень логічної одиниці (високий рівень сигналу), якщо на обох і лише на обох входах буде рівень логічної одиниці. В даному випадку до входів плати підключені 2 кнопки. Отже, в прикладі програми світлодіод загориться лише в тому випадку, коли обидві кнопки будуть натиснуті. На рис. 1.23 представлена схема, що реалізує роботу елемента 2-І.

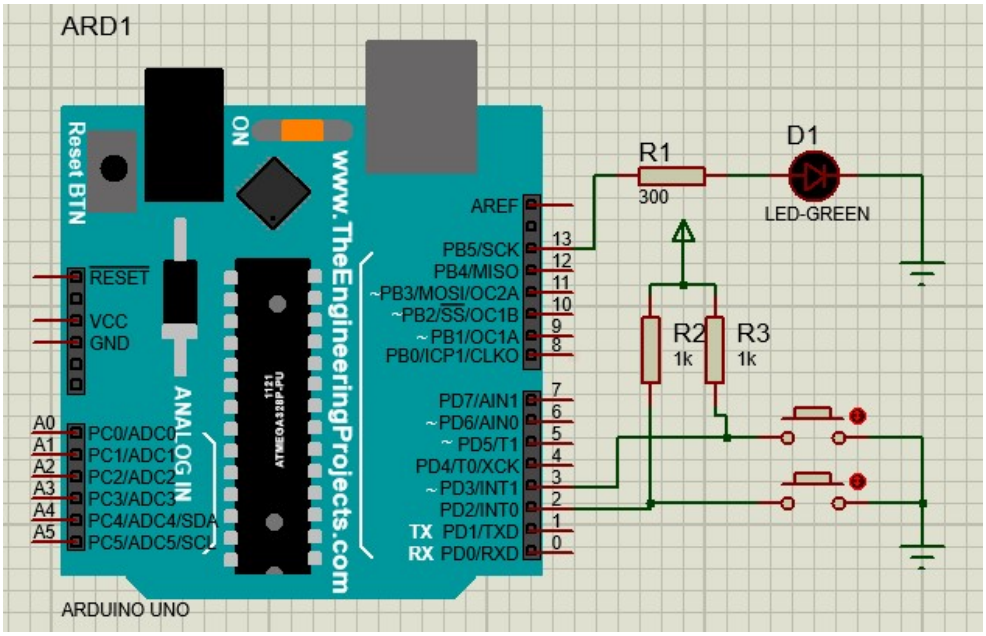


Рис. 1.23.

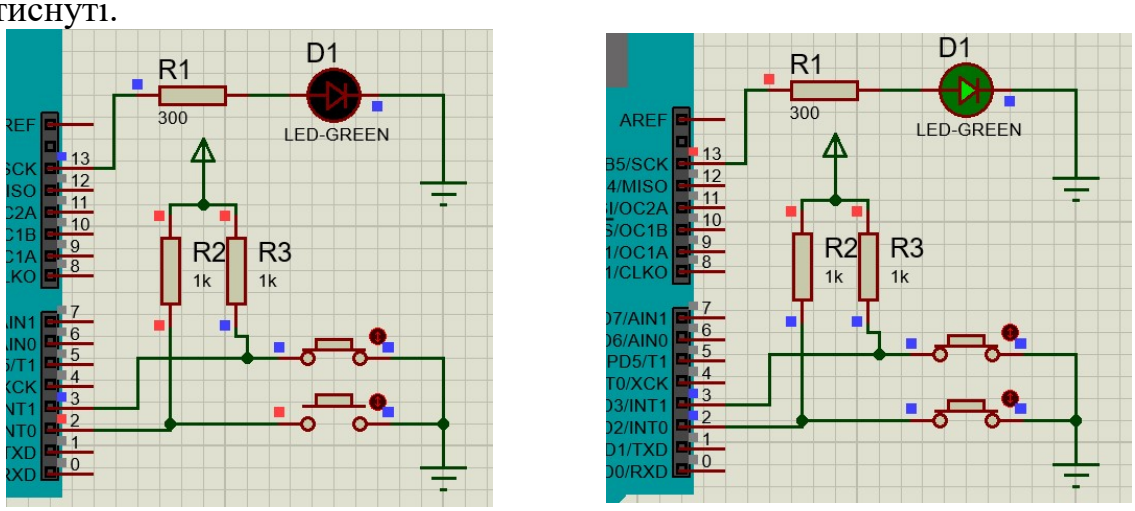
Слід зауважити, що на вхід схеми поступає високий рівень HIGH коли кнопка не натиснута, і низький рівень LOW коли кнопка натиснута. Текст програми представлено нижче.

```
// константи не змінюються і визначають номери виводів:
const int firstButtonPin = 2;     // номер виводу, до якого підключена перша кнопка
const int secondButtonPin = 3; // номер виводу, до якого підключена друга кнопка
const int ledPin =  13;           // номер виводу, до якого підключений світлодіод
// оголошення змінних, які будуть змінюватися
int firstButtonState = 0;         // змінна для зчитування стану першої кнопки
int secondButtonState = 0;     // змінна для зчитування стану другої кнопки

void setup() {
  // налаштування виводу, до якого підключений світлодіод, на вихід
  pinMode(ledPin, OUTPUT);
// налаштування виводів, до якого підключені кнопки, на вхід
  pinMode(firstButtonPin, INPUT);
```

```
  pinMode(secondButtonPin, INPUT);
}
void loop() {
  // зчитування стану кнопок в змінні
  firstButtonState = digitalRead(firstButtonPin);
  secondButtonState = digitalRead(secondButtonPin);
  // перевірка, чи натиснуті обидві кнопки
  if ((firstButtonState == LOW) && (secondButtonState == LOW)) {
    // увімкнути світлодіод
    digitalWrite(ledPin, HIGH);
  } else {
    // вимкнути
    digitalWrite(ledPin, LOW);
  }
}
```

Після компіляції програми можна промоделювати роботу схеми. На рис. 1.24, а, б, представлені результати моделювання. З рисунку видно, що світлодіод загоряється лише в тому випадку, коли обидві кнопки натиснуті.



а б

Рис. 1.24.

Наступний приклад демонструє реалізації елементу 2-АБО. Цей елемент також має 2 входи і один вихід. На виході елемента встановлюється рівень логічної одиниці (високий рівень сигналу), якщо на на будь-якому з входів або на обох входах буде рівень логічної одиниці. Схема для перевірки цієї програми залишається незмінною, як і у випадку з елементом 2-І, і представлена на рис. 1.23. Отже, в прикладі програми світлодіод загориться лише в тому випадку, коли будь-яка кнопка або відразу обидві кнопки будуть натиснуті. Текст програми представлено нижче.

```
// константи не змінюються і визначають номери виводів:
const int firstButtonPin = 2;     // номер виводу, до якого підключена перша кнопка
const int secondButtonPin = 3; // номер виводу, до якого підключена друга кнопка
const int ledPin =  13;           // номер виводу, до якого підключений світлодіод
```

```
// оголошення змінних, які будуть змінюватися
int firstButtonState = 0;         // змінна для зчитування стану першої кнопки
int secondButtonState = 0;     // змінна для зчитування стану другої кнопки

void setup() {
  // налаштування виводу, до якого підключений світлодіод, на вихід
  pinMode(ledPin, OUTPUT);
// налаштування виводів, до якого підключені кнопки, на вхід
  pinMode(firstButtonPin, INPUT);
  pinMode(secondButtonPin, INPUT);
}

void loop() {
  // зчитування стану кнопок в змінні
  firstButtonState = digitalRead(firstButtonPin);
  secondButtonState = digitalRead(secondButtonPin);

  // перевірка, чи натиснута хоча б одна кнопка
  if ((firstButtonState == LOW) || (secondButtonState == LOW)) {
    // увімкнути світлодіод
    digitalWrite(ledPin, HIGH);
  } else {
    // вимкнути
    digitalWrite(ledPin, LOW);
  }
}
```

Відмінність цієї програми від попередньої лише в одному рядку:

```
if ((firstButtonState == LOW) || (secondButtonState == LOW)) {…}
```

Замість логічного оператора І, який позначається &&, використовується логічний оператор АБО, який позначається ||. Логічні оператори найчастіше використовуються в перевірці умови оператора if. Базові відомості про логічні операції дивіться у Вікіпедії за посиланням: https://ru.wikipedia.org/wiki/Логическая_операция

Результатом логічної операції І є істина лише в тому випадку, якщо обидва значення є істинними. Результатом логічної операції АБО є істина в тому випадку, якщо будь-яке значення або обидва значення є істинними.

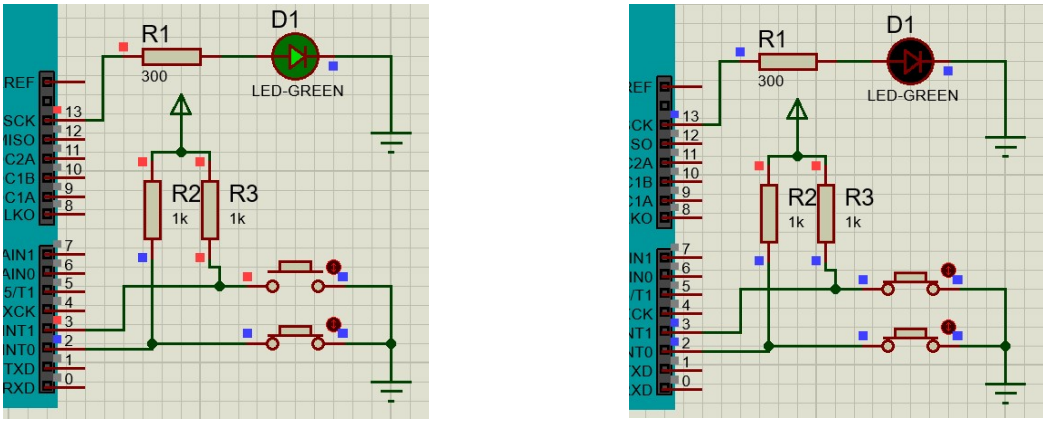
Існує ще логічний оператор НІ, який позначається знаком оклику «!». Він формує значення істина (true) якщо операнд має значення false. Наступний приклад демонструє використання всіх логічних операторів і представляє собою реалізацію базового логічного елемента ВИКЛ. АБО. Схема залишається незмінною та представлена на рис. 1.21. На виході елемента буде високий рівень (HIGH) лише в тому випадку, коли на одному і лише на одному з входів буде логічна одиниця. Тобто в даному прикладі, якщо користувач натисне лише одну кнопку, то світлодіод загориться. Якщо ж не буде натиснута жодна з кнопок, або будуть

натиснуті одночасно обидві кнопки, тоді світлодіод буде вимкнений. Текст програми представлено нижче.

```
// константи не змінюються і визначають номери виводів:
const int firstButtonPin = 2;     // номер виводу, до якого підключена перша кнопка
const int secondButtonPin = 3; // номер виводу, до якого підключена друга кнопка
const int ledPin =  13;            // номер виводу, до якого підключений світлодіод
// оголошення змінних, які будуть змінюватися
int firstButtonState = 0;         // змінна для зчитування стану першої кнопки
int secondButtonState = 0;     // змінна для зчитування стану другої кнопки

void setup() {
  // налаштування виводу, до якого підключений світлодіод, на вихід
  pinMode(ledPin, OUTPUT);
// налаштування виводів, до якого підключені кнопки, на вхід
  pinMode(firstButtonPin, INPUT);
  pinMode(secondButtonPin, INPUT);
}
void loop() {
  // зчитування стану кнопок в змінні
  firstButtonState = digitalRead(firstButtonPin);
  secondButtonState = digitalRead(secondButtonPin);
  // перевірка, чи натиснута лише одна кнопка
  if (((firstButtonState == LOW) && (secondButtonState != LOW)) || ((firstButtonState != LOW) && (secondButtonState == LOW))) {
    // увімкнути світлодіод
    digitalWrite(ledPin, HIGH);
  } else {
    // вимкнути
    digitalWrite(ledPin, LOW);
  }
}
```

На рис. 1.25, а, б, представлено результати моделювання роботи програми. Лише при натисканні однієї кнопки на виході буде значення логічної одиниці і світлодіод загориться. В іншому випадку світлодіод буде вимкненим.



а б

Рис. 1.25.

Відмінність цієї програми від попередніх, що реалізують базові логічні елементи, в наступній строчці:

```
if (((firstButtonState == LOW) && (secondButtonState != LOW)) || ((firstButtonState != LOW) && (secondButtonState == LOW))) {…}
```

Як видно з цього прикладу, умови в операторі if можна комбінувати за допомогою логічних операторів І, АБО та НІ.

1.2.4. Принцип формування ШІМ-сигналу

Наступний приклад демонструє формування широтно-імпульсно-модульованого (ШІМ) сигналу для зміни яскравості світлодіода. ШІМ-сигнал може використовуватися для керування освітленням в будинку в залежності від рівня освітленості, що буде продемонстровано трохи пізніше. Отже, в наступній програмі реалізований алгоритм плавного загоряння світлодіода при натисканні на кнопку та плавного згасання при її відпусканні. Цей проект також можна використовувати для керування освітленням в будинку для запобігання осліплення при раптовій зміні рівня освітлення (наприклад, вмикання світла в кімнаті вночі). Схема для перевірки програми представлена на рис. 1.26. За допомогою віртуального осцилографа можна спостерігати ШІМ-сигнал керування світлодіодом.

Частота ШІМ-сигналу приблизно 490 Гц. На більшості плат Arduino (на базі мікроконтролера ATmega168 або ATmega328) ШІМ підтримують порти 3, 5, 6, 9, 10 і 11, на платі Arduino Mega порти з 2 по 13. На більш ранніх версіях плат Arduino analogWrite() працював тільки на портах 9, 10 та 11.

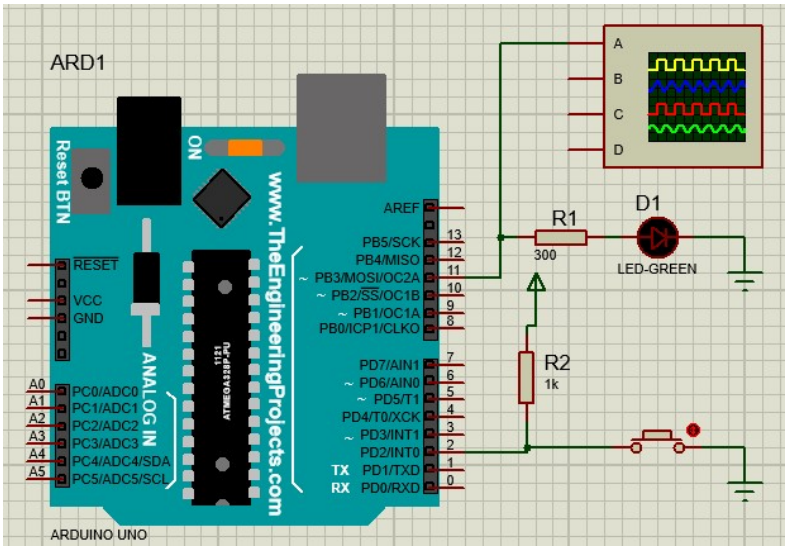


Рис. 1.26.

Текст програм представлено нижче.

```
const int ledPin = 11;    // світлодіод підключений до виводу номер 11
const int buttonPin = 2; // кнопка підключена до виводу номер 2
```

```
int buttonState = 0; // змінна для збереження стану кнопки – натиснута чи ні
bool isBrightnessMax = false; // логічна змінна в якій зберігається поточний стан
                              // світлодіода; true означає максимальна яскравість
                              // false означає що світлодіод вимкнений
void setup() {
 // для виводу, до якого підключений світлодіод, налаштування не потрібне
  pinMode(buttonPin, INPUT); // налаштування виводу номер 2 на вхід
}

void loop() {
// якщо кнопка натиснута, в змінну записується значення LOW, інакше HIGH
  buttonState = digitalRead(buttonPin); // зчитування стану кнопки
// перевірка умови якщо кнопка натиснута і світлодіод був вимкнений
  if((buttonState == LOW) && (isBrightnessMax==false))
  {
    // в циклі виконується поступове збільшення яскравості світлодіода
    // всього буде 255/5=51 ітерація
    for (int fadeValue = 0; fadeValue < 255; fadeValue += 5) {
      // встановити значення скважності на виході (діапазон від 0 до 255):
      analogWrite(ledPin, fadeValue);
      // затримка 30 мілісекунд щоб спостерігати ефект
      delay(30);
    }
    isBrightnessMax = true; // по завершенню виконання циклу в змінну записуємо true
                            // що означає світлодіод має максимальну яскравість
   while (buttonState == LOW) // цикл виконується доти, доки кнопка натиснута
   {
     buttonState = digitalRead(buttonPin);     // зчитуємо значення зі входу, до якого
                                               // підключена кнопка
   }
  }
 // перевірка умови якщо кнопка не натиснута і світлодіод був увімкнений
  else if((buttonState == HIGH) && (isBrightnessMax == true))
  {
    // в циклі виконується поступове зменшення яскравості світлодіода
    // всього буде 255/5=51 ітерація
    for (int fadeValue = 255 ; fadeValue > 0; fadeValue -= 5) {
      // встановити значення скважності на виході (діапазон від 0 до 255):
      analogWrite(ledPin, fadeValue);
      // затримка 30 мілісекунд щоб спостерігати ефект
      delay(30);
    }
    isBrightnessMax = false; // по завершенню виконання циклу в змінну записуємо false
                             // що означає світлодіод вимкнений
    while (buttonState == HIGH) // цикл виконується доти, доки кнопка не натиснута
    {
     buttonState = digitalRead(buttonPin);     // зчитуємо значення зі входу, до якого
                                               // підключена кнопка
    }
  }
}
```

Конструкція циклу for використовується для повторення блоку операторів, розташованих у фігурних дужках. Лічильник ітерацій зазвичай використовується для збільшення і завершення циклу. Оператор for підходить для будь-яких дій, що повторюються, і часто використовується в програмуванні. Цикл for складається з трьох частин:

```
for (initialization; condition; increment) {оператори виконуються в циклі}
```

Ініціалізація (Initialization) виконується найпершою і один раз. Кожен раз в циклі перевіряється умова (condition), і якщо вона вірна, виконується блок операторів і приріст (increment), потім умова перевіряється знову. Коли логічне значення умови стає хибним, цикл завершується.

Цикл while буде обчислювати в циклі безперервно і нескінченно доти, поки вираз в круглих дужках () не стане логічного помилковим. Тому треба змінювати значення змінної, яка перевіряється, інакше вихід з циклу while ніколи не буде досягнутий. Синтаксис циклу while має наступний вигляд:

```
while (вираз) {
// оператори
}
```

Вираз – булевий оператор, який повертає значення істина (true) або лож (false).

Функція analogWrite() видає аналогову величину (ШІМ-сигнал) на порт входу/виходу. Функція може бути корисна для управління яскравістю підключеного світлодіода або швидкістю електродвигуна. Після виклику функції analogWrite() на виході буде генеруватися постійна прямокутна хвиля із заданою шириною імпульсу до наступного виклику analogWrite (або виклику digitalWrite або digitalRead на тому ж порту входу/виходу). Ot Ще раз відмітимо, що частота ШІМ-сигналу приблизно 490 Гц. На більшості плат Arduino (на базі мікроконтролера ATmega168 або ATmega328) ШІМ підтримують порти 3, 5, 6, 9, 10 і 11, на платі Arduino Mega порти з 2 по 13. На більш ранніх версіях плат Arduino analogWrite() працював тільки на портах 9, 10 та 11.

Для виклику analogWrite() немає необхідності встановлювати тип входу/виходу функцією pinMode(). Функція analogWrite ніяк не пов'язана з аналоговими входами і з функцією analogRead.

Синтаксис функції: analogWrite (pin, value);

Параметри, що передаються в функцію: pin: порт вхід / виходу на який подаємо ШІМ сигнал; value: період робочого циклу значення між 0 (повністю вимкнено) і 255 (сигнал поданий постійно).

Функція не повертає ніякого значення.

Після запуску процесу моделювання автоматично з’явиться вікно віртуального осцилографа. Якщо вікно не з’явилось, потрібно зупинити

процес моделювання, і в меню Debug вибрати пункт Reset Debug Popup Windows. Після цього повторно запустити процес моделювання. Натиснувши на кнопку у вікні осцилографа можна спостерігати поступове збільшення скважності сигналу, як показано на рис. 1.27. Якщо кнопку відпустити, то скважність імпульсів поступово буде зменшуватися. Людське око не буде помічати мерехтіння світлодіода, лише поступове загоряння або згасання.

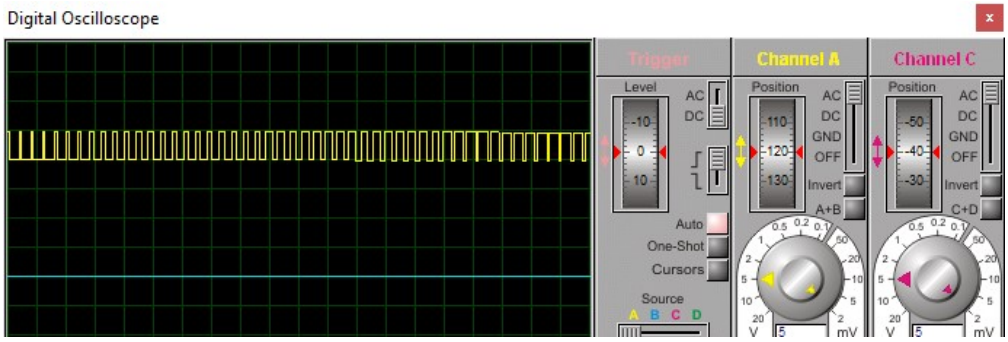


Рис. 1.27.

Широтно-імпульсна модуляція, або ШІМ – це операція отримання аналогового значення, що змінюється, за допомогою цифрових пристроїв. Пристрої використовуються для отримання прямокутних імпульсів – сигналу, який постійно перемикається між максимальним і мінімальним значеннями. Даний сигнал моделює напругу між максимальним значенням (5 В) і мінімальним (0 В), змінюючи при цьому тривалість часу включення 5 В відносно включення 0 В. Тривалість включення максимального значення називається шириною імпульсу. Для отримання різних аналогових величин змінюється ширина імпульсу. При досить швидкій зміні періодів включення-виключення можна подавати постійний сигнал між 0 і 5 В на світлодіод, тим самим керуючи яскравістю його свічення. Рис. 1.28 демонструє написане вище.

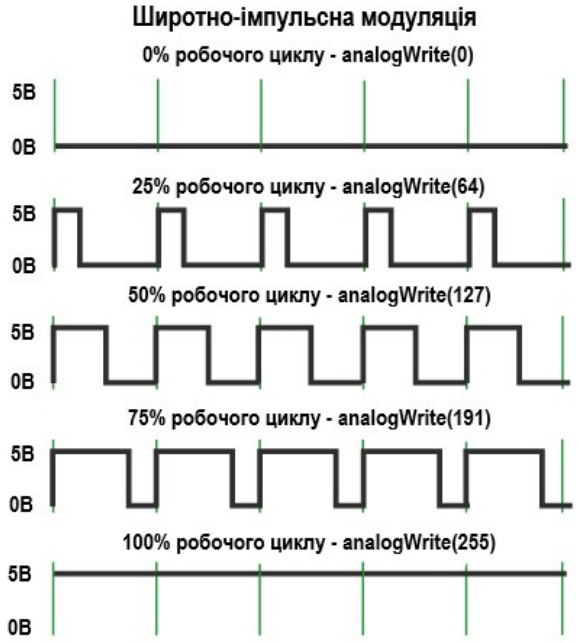


Рис. 1.28.

1.2.5. Зчитування даних з аналогових датчиків

Наступний приклад демонструє процедуру зчитування даних з аналогового датчика за допомогою аналого-цифрового перетворювача (АЦП). До входу A0 плати Arduino UNO підключений змінний резистор, який моделює роботу аналогового датчика. При зміні напруги, що подається на вхід A0, змінюється частота сигналу на виході номер 11, до якого підключений світлодіод. На рис. 1.29 представлено схему моделі.

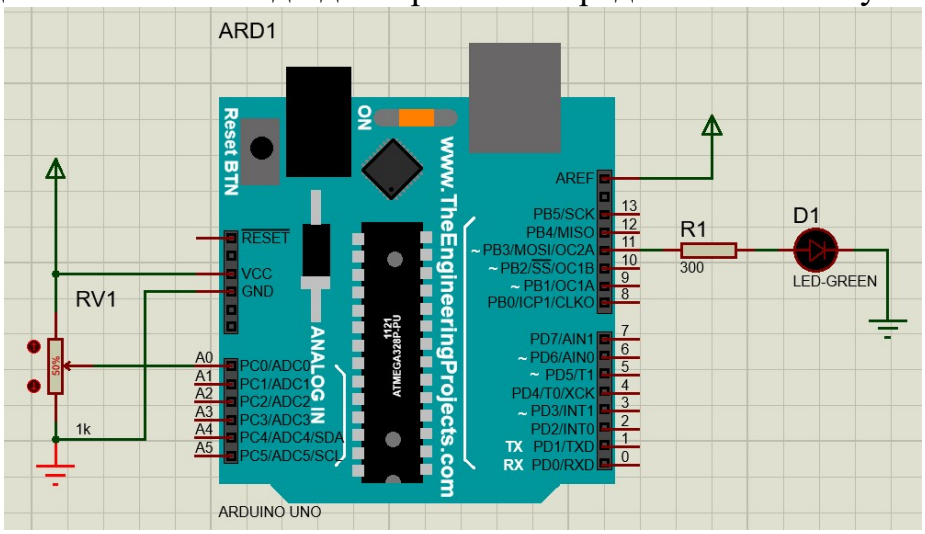


Рис. 1.29.

```
int sensorPin = A0;    // вибір входу, до якого підключено потенціометр – A0
int ledPin = 11;      // номер виводу, до якого підключений світлодіод – 11
int sensorValue = 0;  // змінна для зберігання зчитаного з аналогового входу значення

void setup() {
  // вивід, до якого підключений світлодіод, налаштовується на вихід
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // зчитування даних з аналогового датчика – потенціометра та запис зчитаного
  // значення у змінну  sensorValue; зчитане значення лежить в межах від 0 до 1023
  sensorValue = analogRead(sensorPin);
  // включити світлодіод встановивши на виході високий рівень сигналу
  digitalWrite(ledPin, HIGH);
  // зупинити виконання програми (затримка) на sensorValue мілісекунд
  delay(sensorValue);
  // виключити світлодіод встановивши на виході низький рівень сигналу
  digitalWrite(ledPin, LOW);
  // зупинити виконання програми (затримка) на sensorValue мілісекунд
  delay(sensorValue);
}
```

Функція analogRead зчитує значення з вказаного аналогового входу. Більшість плат Arduino мають 6 каналів (8 каналів у плати Mini та Nano, 16 у Mega) с 10-бітовим аналого-цифровим перетворювачем (АЦП). Напруга, що подається на аналоговий вхід, зазвичай лежить в межах від 0 до 5

вольт, і буде перетворена в значення від 0 до 1023. Це 1024 значення з кроком 0,0049 Вольт. Діапазон напруги і величина кроку може бути змінена функцією analogReference (). Зчитування значення з аналогового входу займає приблизно 100 мікросекунд (0,0001 сек), тобто максимальна частота зчитування приблизно 10000 раз в секунду.

Синтаксис функції має наступний вигляд: analogRead(pin);

Параметри функції: pin – номер порту аналогового входу з якого буде проводитися зчитування (0..5 для більшості плат, 0..7 для Mini і Nano і 0..15 для Mega).

Значення, що повертається, лежить в межах від 0 до 1023. Якщо аналоговий вхід не підключений, то значення, які повертаються функцією analogRead(), можуть приймати випадкові значення.

Велика кількість сучасних мікропроцесорних систем збору, обробки даних та керування після отримання виміряних значень з датчиків передають їх в комп'ютер або в інші пристрої по комунікаційному інтерфейсу. Один з найпопулярніших інтерфейсів, що використовується в системах збору даних та керування, є послідовний інтерфейс передачі даних RS-232C. Більш детально цей інтерфейс буде розглянуто в наступних розділах. В даному розділі розглянемо приклад передавання даних, отриманих в результаті аналого-цифрового перетворення, по послідовному інтерфейсу. Для перевірки роботи програми в Proteus потрібно скласти схему, представлену на рис. 1.30. Для спостереження за даними, що відправляються мікроконтролером по послідовному інтерфейсу, використовується віртуальний термінал.

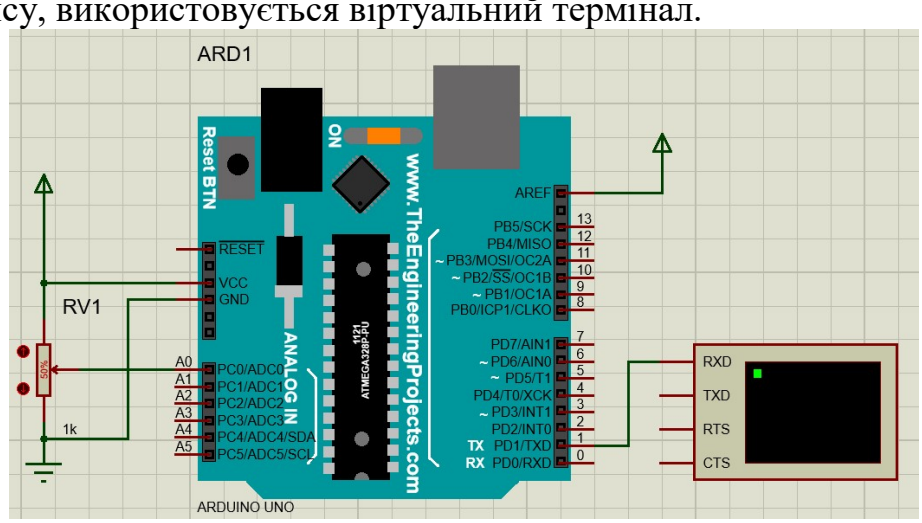


Рис. 1.30.

Текст програми, що виконує аналого-цифрове перетворення сигналу з датчику та передає його по послідовному інтерфейсу. Представлено нижче.

```
void setup() {
  // налаштування швидкості послідовної передачі даних 9600 біт/с
  Serial.begin(9600);
}
```

```
void loop() {
  // оцифровування аналогового сигналу, що подається на вхід A0
  // та передавання отриманого значення по послідовному інтерфейсу
  Serial.println(analogRead(A0));
  // формування затримки 20 мс після завершення кожного аналого-цифрового
  // перетворення
  delay(20);
}
```

Після компіляції програми та запуску схеми на моделювання відкриється вікно віртуального термінала, в якому буде відображатися числове значення, отримане з аналогового датчика в результаті аналого-цифрового перетворення (рис. 1.31). В розглянутому прикладі на вхід A0 поступала напруга 5 В, тому отримане значення є максимальним і дорівнює 1023.

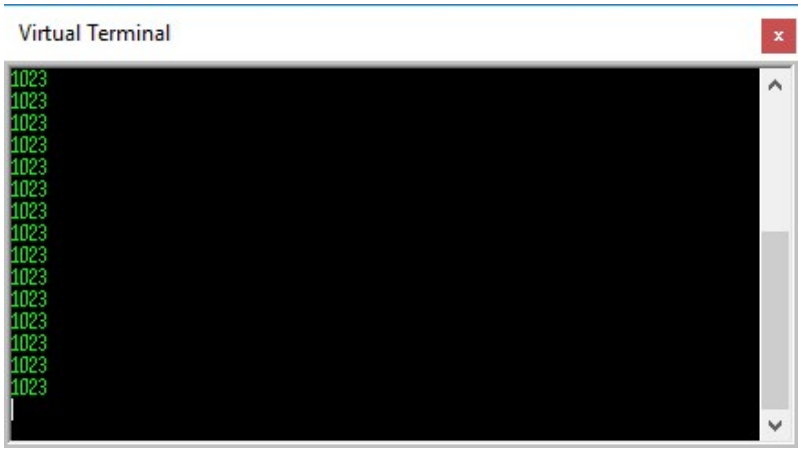


Рис. 1.31.

Набір функцій Serial служить для зв'язку пристрою Arduino з комп'ютером або іншими пристроями, що підтримують послідовний інтерфейс обміну даними. Всі плати Arduino мають хоча б один послідовний порт (UART, іноді називають USART). Для обміну даними Serial використовують цифрові порти введення/виведення 0 (RX) і 1 (TX), а також USB порт. Важливо враховувати, що якщо використовуються функції Serial, то не можна одночасно з цим використовувати порти 0 (RX) і 1 (TX) з іншою метою. Для перевірки роботи програми на реальному пристрої потрібно зібрати схему, як показано на рис. 1.32.

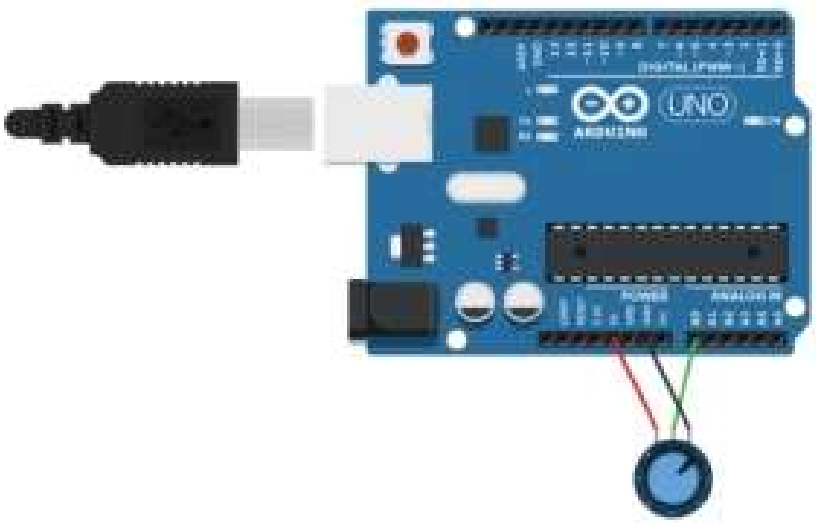


Рис. 1.32.

Для програмування пристрою потрібно натиснути на кнопку «Згрузка» у вікні середовища розробки програмного забезпечення Arduino IDE (рис. 1.33).

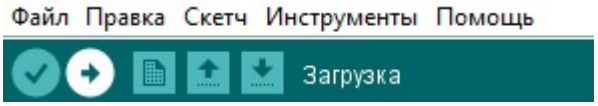


Рис. 1.33.

Функція Serial.begin() ініціює послідовне з'єднання і задає швидкість передачі даних в біт/с (бод). Для обміну даними з комп'ютером використовуйте наступні значення: 300, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 або 115200.

Синтаксис функції виглядає наступним чином:

```
Serial.begin(speed);
```

Додатково, тільки для Arduino Mega:

```
Serial1.begin (speed)
Serial2.begin (speed)
Serial3.begin (speed)
```

Параметри: speed – швидкість в біт/с (бод).

Середовище розробки Arduino має вбудований монітор послідовного інтерфейсу (Serial monitor). Для початку обміну даними необхідно запустити монітор натисканням кнопки Serial monitor і виставити ту ж швидкість зв'язку (baud rate), з якої викликана функція begin().

Функція Serial.println() передає дані через послідовне з'єднання як ASCII текст з наступним за ним символом розриву рядків (ASCII код символу 13, або '\r') і символом нового рядка (ASCII код символу 10, або '\n'). Параметри і типи даних для цієї функції такі ж, як і для Serial.print().

Функції має наступний синтаксис:

```
Serial.println (val);
Serial.println (val, format);
```

Параметри: val – дані для передачі через послідовне з'єднання; format – базис для цілих чисел або кількість знаків після коми для дробових чисел.

Плата Arduino Mega має три додаткових послідовних порту: Serial1 на портах 19 (RX) і 18 (TX), Serial2 на портах на портах 17 (RX) і 16 (TX), Serial3 на портах на портах 15 (RX) і 14 (TX). Щоб використовувати ці порти для зв'язку з комп'ютером знадобиться додаткові адаптери USB-to-serial, тому що вони не підключені до вбудованого адаптера плати Mega. Для зв'язку із зовнішнім пристроєм через послідовний інтерфейс з'єднайте TX порт вашого пристрою з RX портом зовнішнього пристрою і RX порт вашого пристрою з портом TX зовнішнього і з'єднайте "землю" на пристроях. (**Важливо! Не підключайте ці порти безпосередньо до COM-порту комп'ютера, це може пошкодити плату. Використовуйте спеціальні перетворювачі рівнів TTL-RS232**).

Для того, щоб відкрити вікно Монітору порту потрібно в меню Інструменти вибрати пункт Монітор порту (рис. 1.34).

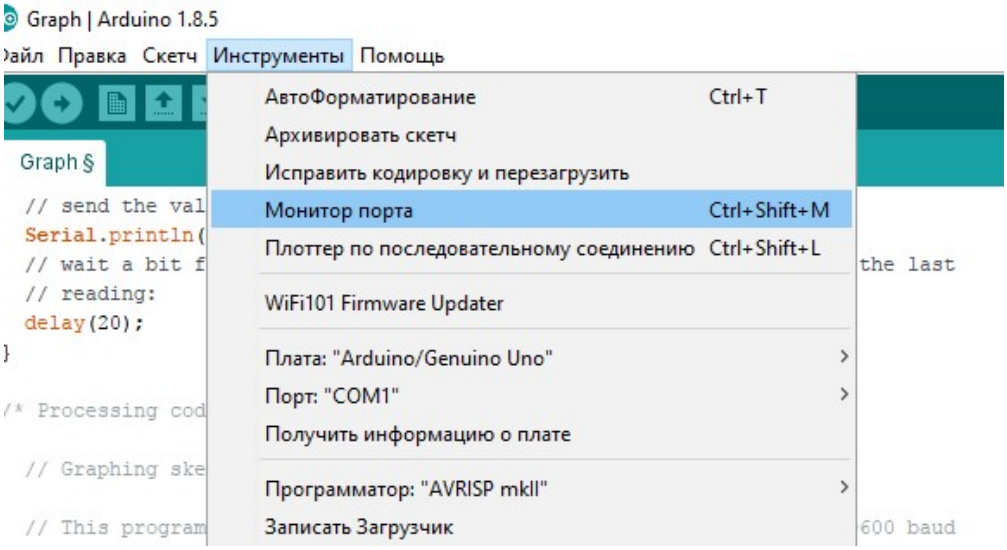


Рис. 1.34.

На рис. 1.35 показане головне вікно монітору порту, за допомогою якого можна перевіряти наявність передачі даних з реального пристрою.

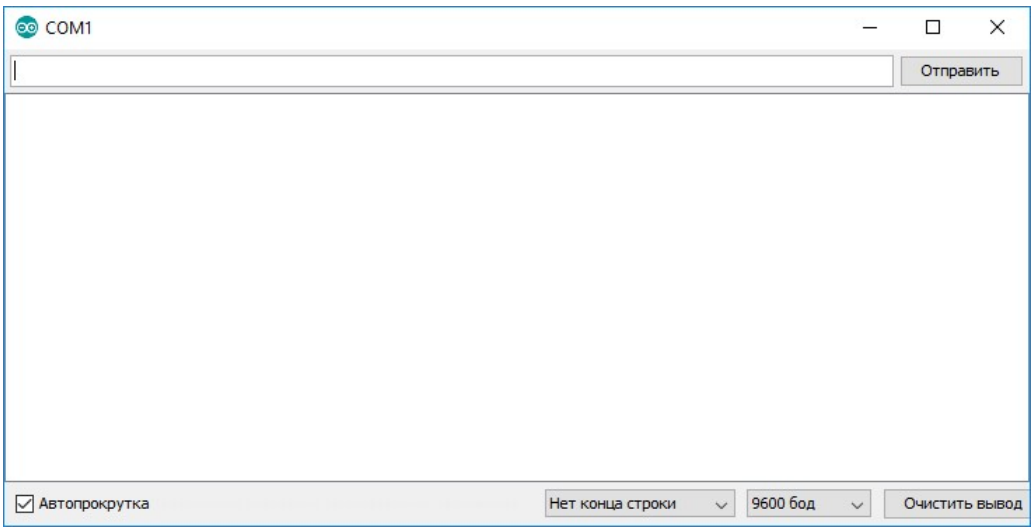


Рис. 1.35.

1.2.6. Використання послідовного інтерфейсу для керування пристроями

Наступний приклад демонструє керування освітленням за допомогою команд, що поступають в Arduino по послідовному інтерфейсу. Для перевірки програми та моделювання роботи пристрою потрібно скласти схему, що представлена на рис. 1.36. Віртуальний термінал дозволить передавати команди керування, а за допомогою віртуального осцилографа можна спостерігати, як буде змінюватися сигнал на виході, до якого підключений світлодіод.

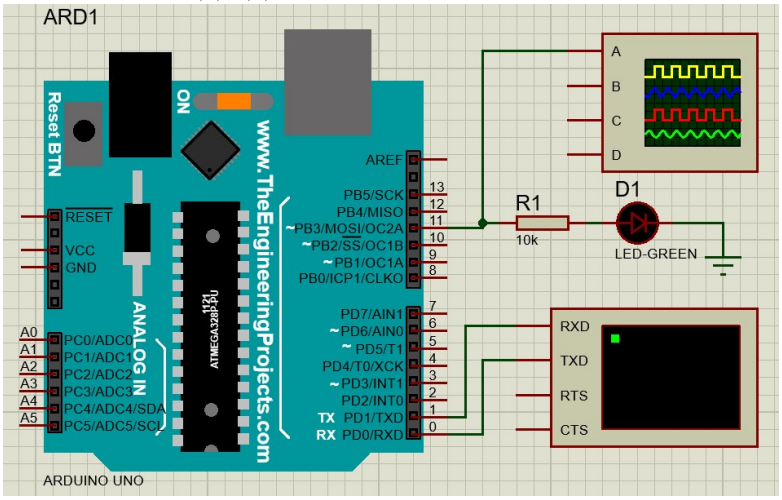


Рис. 1.36.

Команди керування (1 байт) представляють собою число від 0 до 255. При передачі значення 0 світлодіод буде вимкнений. При надходженні значення 255 яскравість світлодіода буде максимальною. За допомогою термінала такі команди можна надіслати шляхом відправлення будь-якого символу. Автоматично цей символ буде перетворено у відповідний ASCII-код. Наприклад при передачі символу ‘A’ буде передано значення його ASCII-код, що дорівнює 65. При передачі символу ‘z’ буде передано його ASCII-код, який дорівнює 122. Текст програми керування яскравістю

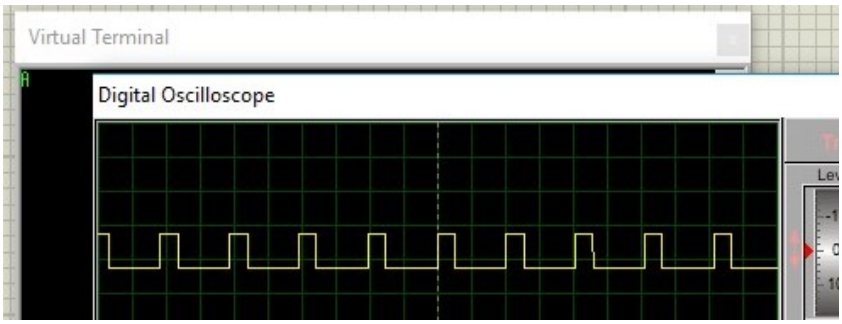
світлодіода за допомогою команд, що поступають по послідовному комунікаційному інтерфейсу, представлено нижче.

```
const int ledPin = 11;      // номер виводу, до якого підключений світлодіод

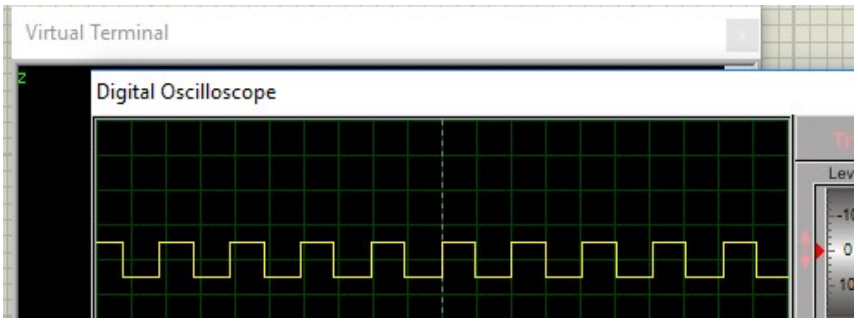
void setup() {
  // налаштування швидкості послідовного інтерфейсу
  Serial.begin(9600);
  // налаштування виводу, до якого підключено світлодіод. на вихід
  pinMode(ledPin, OUTPUT);
}

void loop() {
  byte brightness; // оголошення змінної, в яку зчитується отримане по послідовному
          // інтерфейсу значення
  // перевірка, чи були отримані дані, надіслані з комп'ютера
  if (Serial.available()) { // якщо так
    // зчитування отриманого значення (значення лежить в діапазоні від 0 до 255)
    brightness = Serial.read();
    // установка яскравості світлодіода за допомогою ШІМ-сигналу
    analogWrite(ledPin, brightness);
  }
}
```

Після компіляції програми можна перевірити роботу схеми. Запустивши процес моделювання у вікні Віртуального термінала можна вводити символи та спостерігати у вікні осцилографа як буде змінюватися тривалість імпульсів керування яскравістю світлодіода (рис. 1.37, а, б).



а



б

Рис. 1.37.

Функція Serial.available() отримує кількість байт (символів) доступних для зчитування з послідовного інтерфейсу зв'язку. Це ті байти, які вже надійшли і записані в буфер послідовного порту. Буфер може зберігати до 64 байт. Синтаксис має наступний вигляд:

```
Serial.available();
```

Для Arduino Mega:

```
Serial1.available();
Serial2.available();
Serial3.available();
```

Функція не має жодних параметрів. Значення, що повертається функцією представляє собою кількість байт, доступних для зчитування.

Функція Serial.read() зчитує черговий доступний байт з буфера послідовного з'єднання. Синтаксис має наступний вигляд:

```
Serial.read();
```

Для Arduino Mega:

```
Serial1.read();
Serial2.read();
Serial3.read();
```

Функція Serial.read() не потребує жодних параметрів. Значення, що повертається функцією – отриманий байт або -1, якщо буфер пустий.

Послідовний комунікаційний інтерфейс може використовуватися для підключення Bluetooth-модуля, наприклад, HC-05 або HC-06. Моделі для Proteus цих модулів можна завантажити з сайту за посиланням:

https://www.theengineeringprojects.com/2016/03/bluetooth-library-for-proteus.html

На рис. 1.38 представлена головна сторінка сайту, з якого можна завантажити бібліотеки, натиснувши на посилання, на яке вказує стрілка.

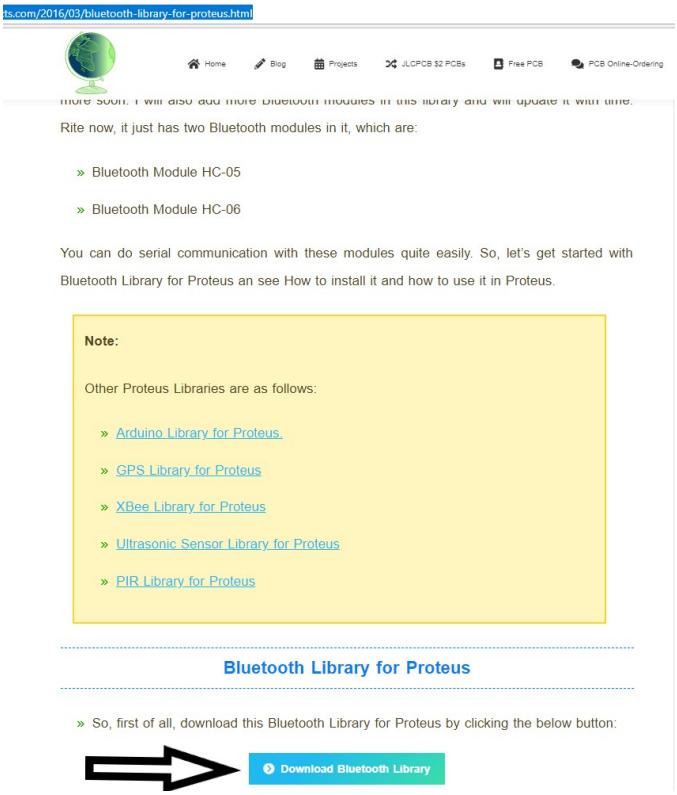


Рис.1.38.

Після завантаження архіву та його розпакування, файли BluetoothTEP.IDX та BluetoothTEP.LIB потрібно скопіювати в папку LIBRARY. Якщо при інсталяції Proteus шлях до каталогу установки не змінювався і був залишений за замовчуванням, тоді шлях до каталогу LIBRARY буде наступний: C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\LIBRARY.

На рис. 1.39 показаний вміст каталогу LIBRARY з файлами моделей Bluetooth-радіомодулей.

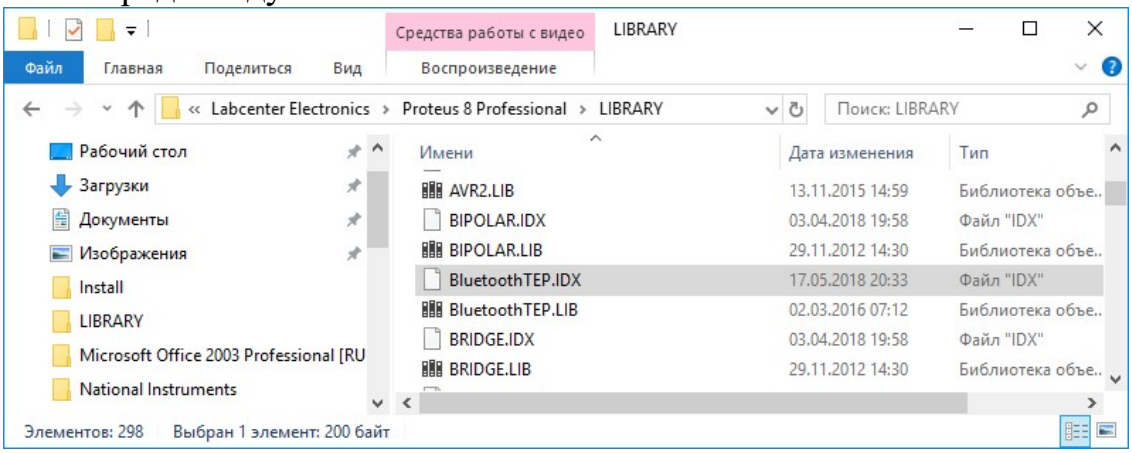


Рис. 1.39.

Аналогічним чином виконується підключення в Proteus сторонніх бібліотек різних модулів та датчиків.

Після підключення бібліотеки Bluetooth-модулей, вони доступні для використання в схемі. При додаванні нового елемента в схему у вікні Pick

devices, представленому на рис. 1.40, з’явиться категорія Bluetooth, в якій доступні 2 елементи – радіомодеми HC-05 та HC-06.

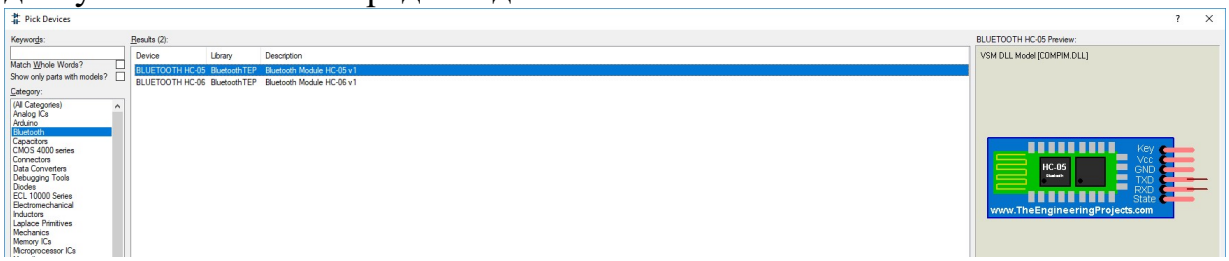


Рис. 1.40.

Схема підключення Bluetooth-радіомодуля HC-05 до плати Arduino UNO представлена на рис. 1.41.

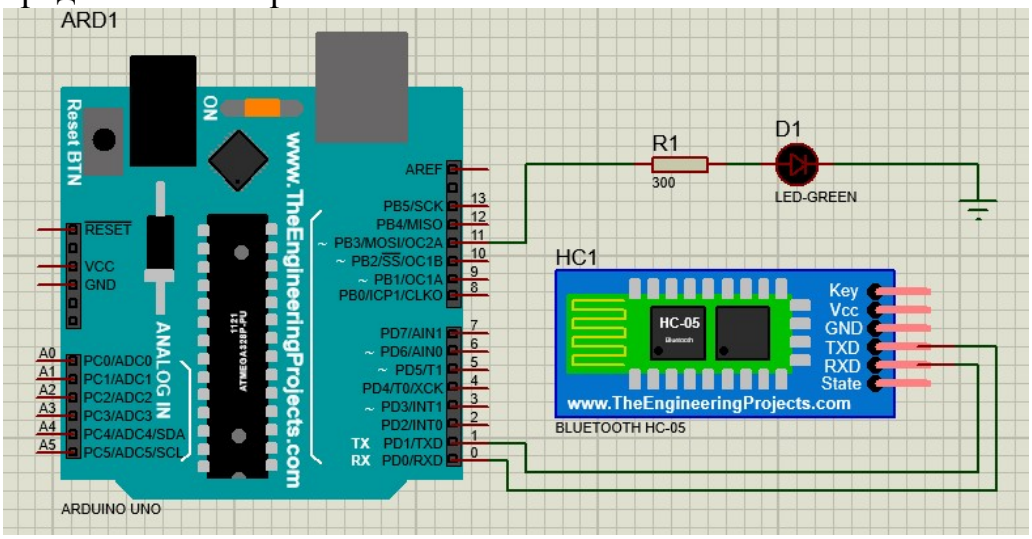


Рис. 1.41.

Слід мати на увазі, що при підключенні Bluetooth-радіомодуля до реального пристрою, його потрібно підключити до ліній живлення. При моделюванні це не обов’язково. На рис. 1.42 представлена схема підключення Bluetooth-радіомодуля до реального пристрою.

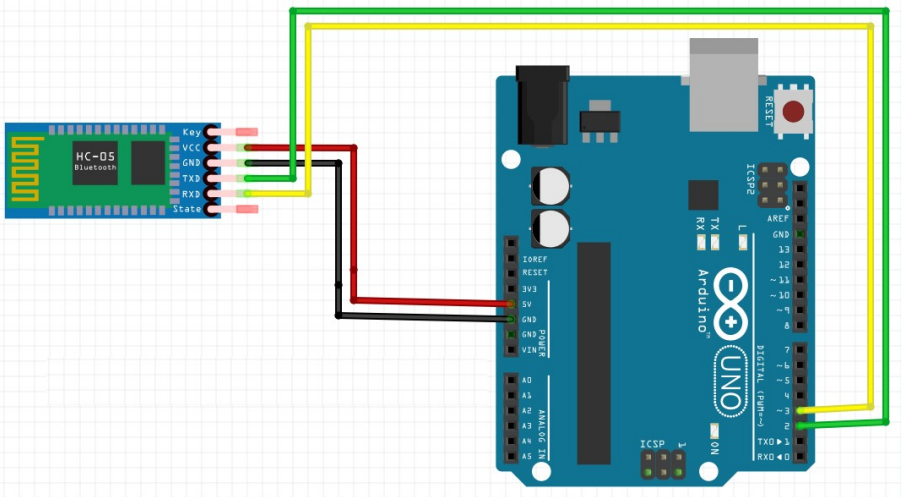


Рис. 1.42.

Наявність Bluetooth-радіомодуля, підключеного до Arduino, відкриває можливості для побудови мікропроцесорних систем збору, обробки даних та керування по радіоканалу з використанням смартфонів. Більш детально принципи побудови програмного забезпечення для комп'ютера для керування обладнанням по послідовному інтерфейсу, а також наявні програмні засоби передачі команд керування по Bluetooth для смартфонів будуть розглянуті в наступних розділах.

1.3. Приклад програмної реалізації мультиплексора

Наступний приклад демонструє реалізацію мультиплексора «4 в 1», який широко використовується в цифровій електроніці. На рис. 1.43 представлена схема для моделювання пристрою та перевірки алгоритму роботи програми.

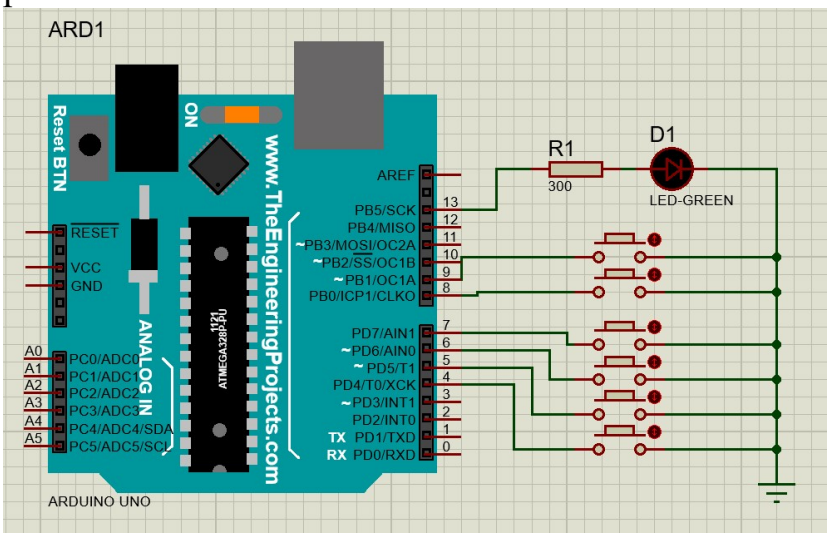


Рис. 1.43.

Входи з номерами 8 та 9, до яких підключені дискретні кнопки. Є адресними входами. Сигнал з одного з інформаційних входів, які мають номери 4, 5, 6 та 7, передається на один вихід 11, до якого підключений світлодіод, в залежності від двійкової комбінації на адресних входах. Текст програми, що реалізує роботу мультиплексора «4 в 1», представлено нижче. В схемі використовуються підтягуючі резистори, що вбудовані в мікроконтролер. Для їх підключення необхідно встановити високий рівень (HIGH) на всіх виводах, які налаштовано на вхід (INPUT). Тоді, при натисканні на кнопку з відповідного входу буде зчитано низький рівень (LOW), а у випадку, коли кнопка не натиснута, зчитуватися буде високий рівень сигналу (HIGH). Внутрішні підтягуючі резистори зручно використовувати щоб спростити схему та не використовувати додаткові резистори. Текст програми представлено нижче.

```
const int ledPin = 13;      // до виводу номер 13 підключений світлодіод
const int addressButton1 = 8; // перший адресний вхід
const int addressButton2 = 9; // другий адресний вхід
const int inputButton1 = 7; // перший інформаційний вхід
```

```
const int inputButton2 = 6; // другий інформаційний вхід
const int inputButton3 = 5; // третій інформаційний вхід
const int inputButton4 = 4; // четвертий інформаційний вхід

void setup() {
 pinMode(ledPin, OUTPUT); // налаштування виводу 13 на вихід
 pinMode(addressButton1, INPUT); // вивід 8 налаштовано на вхід
 digitalWrite(addressButton1, HIGH); // вмикання підтягуючого резистора
 pinMode(addressButton2, INPUT); // вивід 9 налаштовано на вхід
 digitalWrite(addressButton2, HIGH); // вмикання підтягуючого резистора
 pinMode(inputButton1, INPUT); // вивід 7 налаштовано на вхід
 digitalWrite(inputButton1, HIGH); // вмикання підтягуючого резистора
 pinMode(inputButton2, INPUT); // вивід 6 налаштовано на вхід
 digitalWrite(inputButton2, HIGH); // вмикання підтягуючого резистора
 pinMode(inputButton3, INPUT); // вивід 5 налаштовано на вхід
 digitalWrite(inputButton3, HIGH); // вмикання підтягуючого резистора
 pinMode(inputButton4, INPUT); // вивід 4 налаштовано на вхід
 digitalWrite(inputButton4, HIGH); // вмикання підтягуючого резистора
}

void loop() {
  byte address; // змінна для збереження значення адреси
  int state; // змінна для збереження стану інформаційної кнопки
  int buttonState1 = 0; // змінна для збереження стану першої адресної кнопки
  int buttonState2 = 0; // змінна для збереження стану другої адресної кнопки
  // зчитування стану першої адресної кнопки – якщо кнопка натиснута, функція
  // повертає LOW, інакше – HIGH, яке записується в змінну buttonState1
  buttonState1 = digitalRead(addressButton1);
 // зчитування стану другої адресної кнопки – якщо кнопка натиснута, функція
  // повертає LOW, інакше – HIGH, яке записується в змінну buttonState2
  buttonState2 = digitalRead(addressButton2);

  // якщо обидві адресні кнопки не натиснуті
  if((buttonState1 == HIGH) && (buttonState2 == HIGH))
  {
    address = 0;  // в змінну записується значення 0
  }
 // якщо перша адресна кнопка натиснута, а друга ні
  else if((buttonState1 == LOW) && (buttonState2 == HIGH))
  {
    address = 1;  // в змінну записується значення 1
  }
  // якщо перша адресна кнопка не натиснута, а друга натиснута
  else if((buttonState1 == HIGH) && (buttonState2 == LOW))
  {
    address = 2;  // в змінну записується значення 2
  }
  // якщо обидві кнопки натиснуті
  else if((buttonState1 == LOW) && (buttonState2 == LOW))
  {
    address = 3;  // в змінну записується значення 3
  }
```

```
// в залежності від значення змінної address виконується відповідний блок коду
 switch(address)
 {
  case 0: // якщо значення змінної address дорівнює 0
   state = digitalRead(inputButton1);
   // зчитування значення з першої інформаційної кнопки і вивід інвертованого
   // значення на вихід, до якого підключений світлодіод
   digitalWrite(ledPin, (state==LOW)?HIGH:LOW);
  break;
  case 1: // якщо значення змінної address дорівнює 0
   state = digitalRead(inputButton2);
   // зчитування значення з другої інформаційної кнопки і вивід інвертованого
   // значення на вихід, до якого підключений світлодіод
   digitalWrite(ledPin, (state==LOW)?HIGH:LOW);
  break;
  case 2: // якщо значення змінної address дорівнює 0
   state = digitalRead(inputButton3);
   // зчитування значення з третьої інформаційної кнопки і вивід інвертованого
   // значення на вихід, до якого підключений світлодіод
   digitalWrite(ledPin, (state==LOW)?HIGH:LOW);
  break;
  case 3: // якщо значення змінної address дорівнює 0
   state = digitalRead(inputButton4);
   // зчитування значення з четвертої інформаційної кнопки
   // і вивід інвертованого
   // значення на вихід, до якого підключений світлодіод
   digitalWrite(ledPin, (state==LOW)?HIGH:LOW);
  break;
  default: // якщо значення змінної address не дорівнює 0, 1, 2 або 3
   digitalWrite(ledPin, LOW); // встановлення на виході низького рівня
  break;
 }}
```

Після компіляції програми та моделювання схеми можна перевірити її роботу. На рис. 1.44 показаний випадок, коли обидві кнопки натиснуті. Отже, на вихід передається сигнал з 4 входу – якщо кнопка, підключена до цього входу натиснута, то світлодіод загоряється.

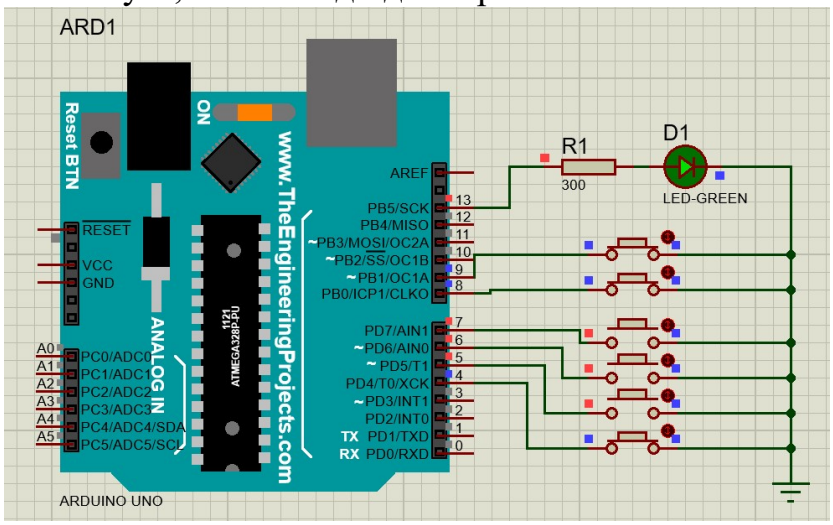


Рис. 1.44.

Подібно конструкції if, switch ... case керує процесом виконання програми, дозволяючи програмісту задавати альтернативний код, який буде виконуватися при різних умовах. Зокрема, оператор switch порівнює значення змінної зі значенням, визначеному в операторах case. Коли знайдений оператор case, значення якого дорівнює значенню змінної, виконується програмний код в цьому операторі.

Ключове слово break є командою виходу з оператора case і зазвичай використовується в кінці кожного case. Без оператора break оператор switch будет продовжувати обчислювати такі вирази, поки не досягне ключового слова break в програмі, або кінця оператора switch.

Конструкція switch…case має наступний синтаксис:

```
switch (var) {
 case label:
 // код для виконання
 break;
 case label:
 // код для виконання
 break;
 default:
 // код для виконання
}
```

Параметри: var – змінна, яка використовується для порівняння з варіантами в case; label – значення варіантів, з яким порівнюється значення змінної var.

1.4. Керування алфавітно-цифровим рідкокристалічним індикатором

Наступний приклад демонструє принцип використання алфавітно-цифрового рідкокристалічного індикатора (РКІ) для виводу тексту та числових даних. Схема підключення РКІ до Arduino представлена на рис. 1.45.

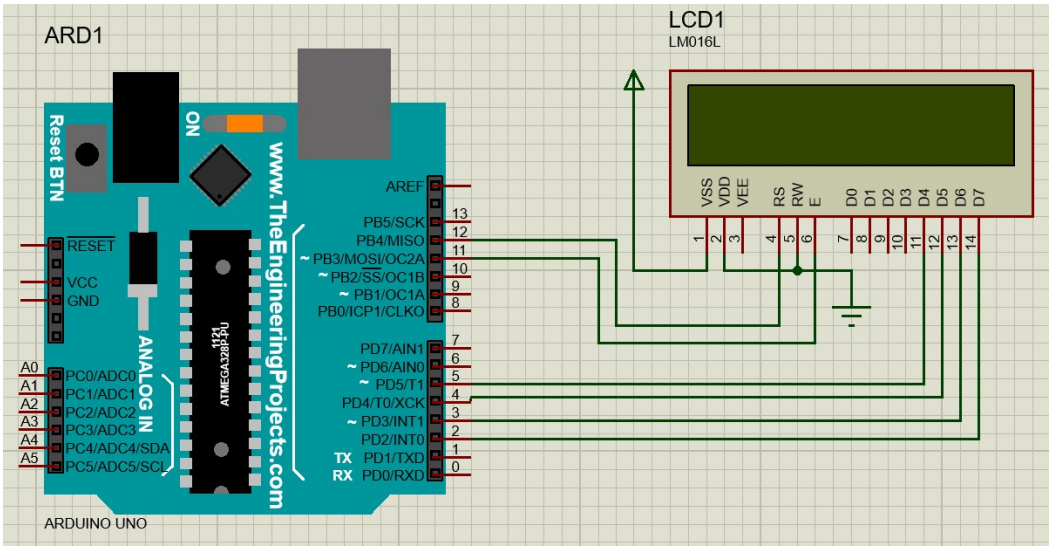


Рис. 1.45.

У верхній строчці РКІ виводиться текст «Hello, world!», а в нижній кількість секунд. Що минули з моменту вмикання пристрою (у випадку моделювання – з моменту початку моделювання роботи пристрою). Текст програми представлено нижче.

```
#include <LiquidCrystal.h> // підключення бібліотеки для роботи з РКІ

// ініціалізація РКІ шляхом вказівки, до яких виводів плати Arduino підключені
// відповідні виводи РКІ
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
// виклик функції ініціалізації РКІ
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

void setup() {
  // налаштування РКІ – режим виводу в 2 строки по 16 символів в кожній
  lcd.begin(16, 2);
  // вивід тексту "Hello, world!" на екран РКІ
  lcd.print("Hello, world!");
}

void loop() {
  // переведення курсору на другий рядок в початкову позицію
 // рядок 1 – це другий рядок. Оскільки нумерація починається з 0
  lcd.setCursor(0, 1);
  // вивід значення секунд з моменту вмикання пристрою
  lcd.print(millis() / 1000);
}
```

Після компіляції програми і запуску схеми на моделювання на екрані РКІ можна побачити відповідну інформацію, як показано на рис. 1.46.

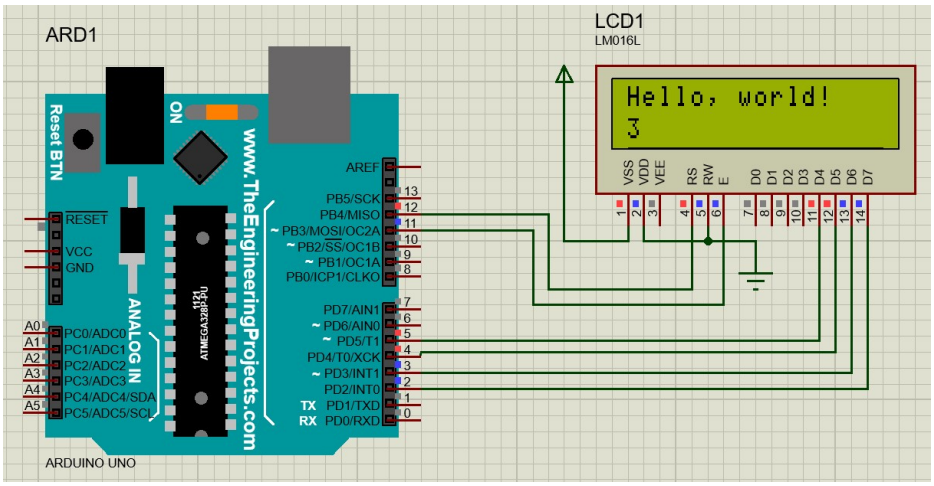


Рис. 1.46.

Детальний опис бібліотеки для роботи з рідкокристалічним індикатором з великою кількістю прикладів можна знайти в Інтернет за посиланнями:

http://developer.alexanderklimov.ru/arduino/liquidcrystal.php

http://arduino-diy.com/arduino-zhidkokristallicheskiy-displey-LCD-1602

https://doc.arduino.ua/ru/prog/LiquidCrystal

1.5. Керування серводвигуном та кроковим двигуном

Наступний приклад демонструє систему керування положенням ротора сервопривода в залежності від освітленості. Цей приклад може бути використаний для автоматичного керування положенням жалюзі у будинку в залежності від рівня освітленості в приміщенні. На рис. 1.47 представлено принципову схему. Для наочності в схемі також є вольтметр, на якому відображається значення напруги, що поступає на аналоговий вхід (вхід АЦП) Arduino. При зміні освітленості датчика, яким виступає фоторезистор. Його опір зменшується, і, відповідно, зменшується напруга, що надходить на вхід A0.

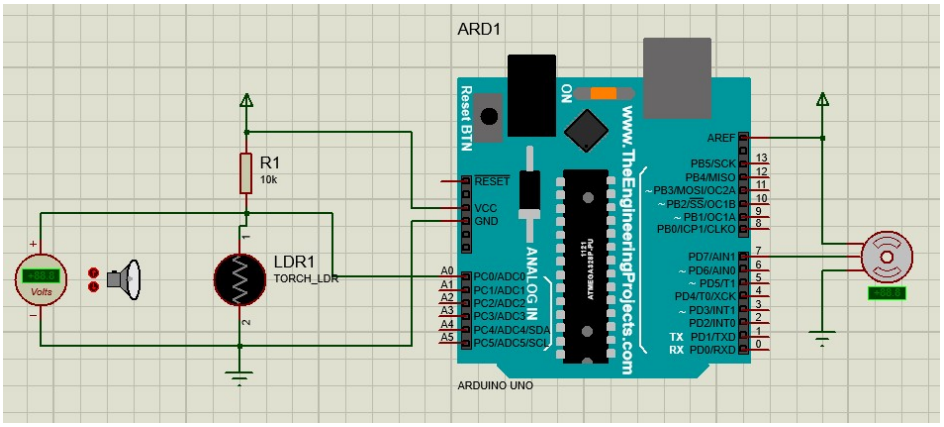


Рис. 1.47.

Текст програми керування положенням ротора сервопривода в залежності від освітлення представлено нижче.

```
#include <Servo.h> // підключення бібліотеки керування сервоприводом
const int servoPin = 7; // номер виводу, до якого підключено сервопривод
const int sensorPin = A0; // вхід, до якого підключений датчик освітлення

int sensorValue = 0; // змінна для зберігання даних, зчитаних з датчика
Servo myservo; // об’єкт для керування сервоприводом

void setup() {
  myservo.attach(servoPin); // ініціалізація сервопривода шляхом вказівки,
                            // до якого виводу плати Arduino він підключений
}

void loop() {
 int angle;       // змінна для зберігання кута повороту ротора сервопривода
 sensorValue = analogRead(sensorPin); // зчитування даних з датчика освітленості
 angle = sensorValue*0.176; // перетворення апаратних одиниць в кут повороту
 myservo.write(angle);  // встановлення сервоприводу в потрібне положення
                        // в залежності від освітленості датчика
```

```
 delay(50);                  // затримка 50 мс
}
```

В програмі використовується коефіцієнт 0,176 для перетворення апаратних одиниць, що отримуються після аналого-цифрового перетворення сигналу з датчику (максимальне значення 1023) в кут сервопривода (максимальне значення 180). Після компіляції програми можна виконати моделювання роботи схеми і перевірити коректність алгоритму. На рис. 1.48 представлено вигляд схеми під час моделювання за умови мінімальної освітленості датчика. Положення ротора сервоприводу складає 90 градусів.

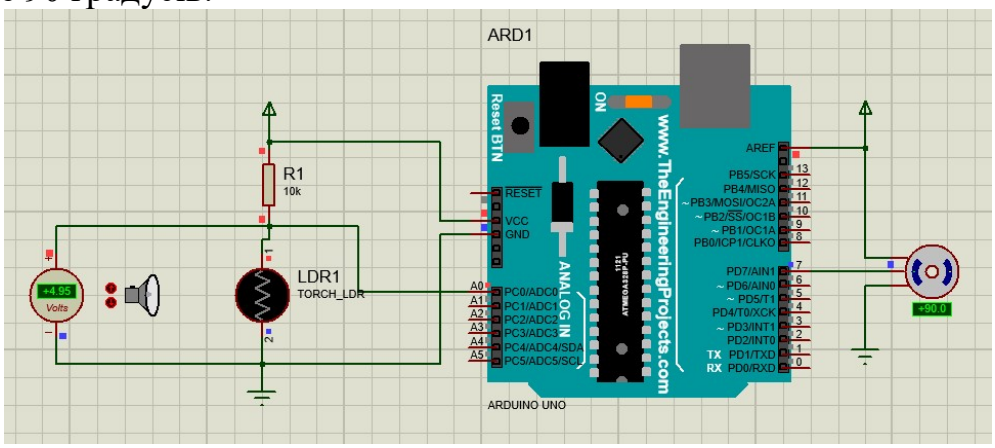


Рис. 1.48.

На рис. 1.49 представлена схема показаний випадок, коли освітленість датчика більша, ніж в попередньому випадку. Видно, що ротор сервоприводу змінив положення і зараз кут становить 50,7 градусів.

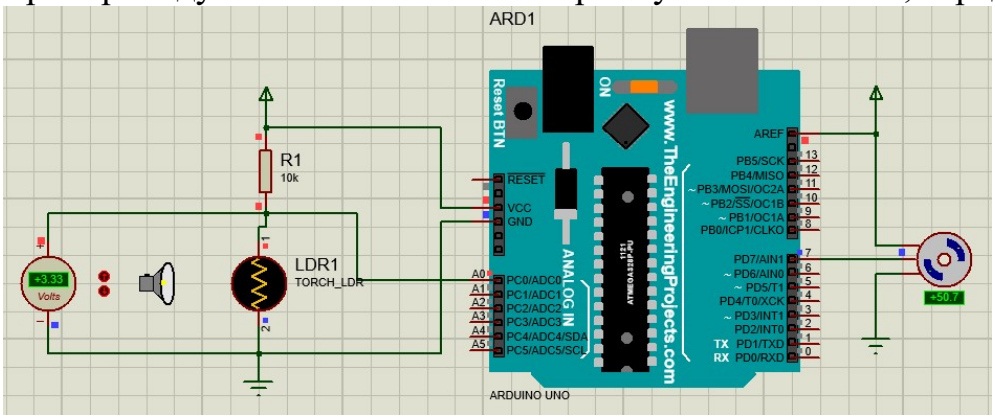


Рис. 1.49.

Бібліотека функцій Servo для Arduino контролера надає набір функцій для управління сервоприводами. Стандартні сервоприводи дозволяють повертати привід на визначений кут, зазвичай від 0 до 180 градусів. Деякі сервоприводи дозволяють здійснювати повні оберти на заданій швидкості.

Бібліотека Servo дозволяє одночасно управляти 12 сервоприводами на більшості плат Arduino і 48 на Arduino Mega. На контролерах, відмінних від Mega, використання бібліотеки виключає можливість використовувати

виходи 9 і 10 в режимі ШІМ навіть якщо привід не підключений до цих виводів. На платі Mega можуть бути використані до 12 сервоприводів без втрати функціоналу ШІМ.

У загальному випадку сервопривод підключається 3-ма проводами: живлення, земля і сигнальний. Зазвичай живлення – червоний провід і може бути підключений до виводу +5В на платі Arduino. Чорний провід (земля) підключається до виводу GND плати Arduino. Сигнальний провідник, який зазвичай має жовтий колір, підключається до цифрового виводу контролера Arduino. Слід зазначити, що потужні сервоприводи можуть створювати велике навантаження. В цьому випадку він повинен живитися окремо (не через вихід +5В контролера Arduino).

Функція Servo.attach() підключає Servo до зазначеного виходу, з якого здійснюється управління приводом. Функція має наступний синтаксис:

```
servo.attach(pin);
servo.attach(pin, min, max);
```

Параметри функції: servo – змінна типу Servo; pin – номер виходу, до якого підключається servo і з якого здійснюється управління приводом; min (опціональний параметр) – ширина імпульсу в мікросекундах, що відповідає мінімальному (кут 0 градусів) положенню сервоприводу (за замовчуванням 544); max (опціональний параметр) – ширина імпульсу в мікросекундах, який відповідає максимальному (кут 180 градусів) положенню сервоприводу. Опціональний параметр означає, що вказувати значення цього параметра при виклику функції не обов'язково. Якщо опціональний параметр відсутній, то використовується значення за замовчуванням.

Функція Servo.write() передає значення для управління приводом. Для стандартного сервоприводу це кут повороту. Для приводу постійного обертання функція задає швидкість обертання (0 – для максимальної швидкості обертання в одну сторону, 180 – для максимальної швидкості в іншу сторону і 90 для нерухомого стану).

Функція має наступний синтаксис:

```
servo.write(angle);
```

Параметри функції: servo – змінна типу Servo; angle – значення кута повороту сервопривода від 0 до 180.

Бібліотека Stepper надає зручний інтерфейс управління біполярними і уніполярнимі кроковими двигунами. Для управління кроковим двигуном, в залежності від його типу (біполярний або уніполярний) і обраного способу підключення, знадобляться деякі додаткові електронні компоненти.

На рис. 1.50 представлено схему підключення уніполярного крокового двигуна до контролера Arduino. Безпосередньо підключати кроковий двигун до контролера неможна, оскільки навантажувальна здатність виводів недостатня для безпосереднього керування двигуном. В цій схемі використовується спеціальна мікросхема ULN200A, яка представляє собою набір транзисторів Дарлингтона та має виходи типу відкритий колектор. В даному випадку мікросхема відіграє роль підсилювача потужності. Сигнали керування, що поступають на вхід мікросхеми від контролера, змінюють режим роботи виводів мікросхеми і струм протікає по визначених обмотках крокового двигуна, тим самим обертаючи його ротор.

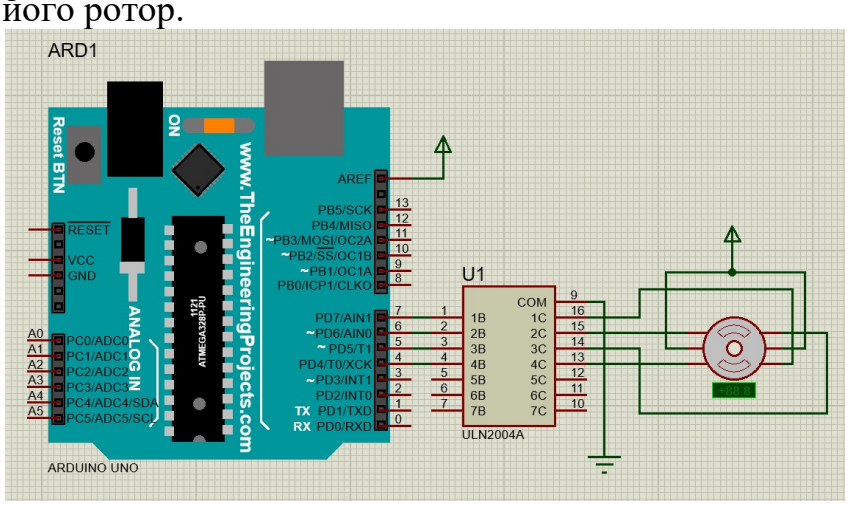


Рис. 1.50.

На рис. 1.51 представлено вікно налаштувань моделі крокового двигуна в Proteus. Напруга живлення крокового двигуна складає 5 В. Кількість кроків для здійснення одного повного оберту дорівнює 72, тобто один крок складає 5 градусів. Максимальна швидкість обертання ротора крокового двигуна складає 100 обертів на хвилину.

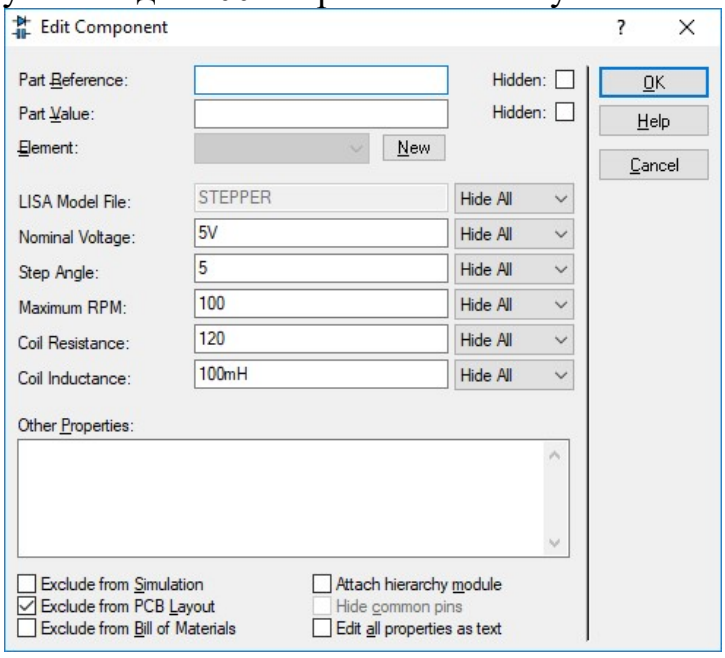


Рис. 1.51.

Текст програми, що забезпечує обертання крокового двигуна, представлений нижче.

```
#include <Stepper.h> // підключення бібліотеки для роботи з кроковим двигуном
// створення об’єкта для керування кроковим двигуном
// в якості параметрів передаються значення кількості кроків на 1 оберт та
// номери виводів контролера Arduino, до яких підключено обмотки двигуна
Stepper mystepper = Stepper(72, 4, 5, 6, 7);

void setup() {
 mystepper.setSpeed(1); // ця функція встановлює швидкість обертання 1 об/хв.
}

void loop() {
 mystepper.step(1);   // здійснити оберт на 1 крок
// затримка 100 мс щоб завершилися всі перехідні процеси в обмотках
 delay(100);
}
```

Функція-конструктор Stepper створює новий об'єкт класу Stepper. Цей об'єкт приєднаний до одного крокового двигуна, підключеного до контролера Arduino. Конструктор слід використовувати при оголошенні змінної класу Stepper, зазвичай на самому початку, поза setup() і loop(). Кількість параметрів залежить від способу підключення – 2 або 4 виводи використовується для керування кроковим двигуном. Синтаксис конструктора має наступний вигляд:

```
Stepper (steps, pin1, pin2);
Stepper (steps, pin1, pin2, pin3, pin4);
```

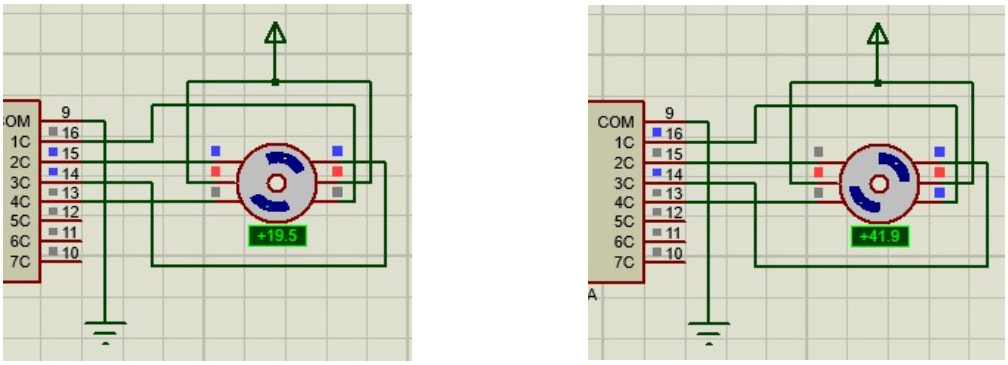
Параметри конструктора об’єкта Stepper наступні: steps – кількість кроків в повному обороті крокового двигуна, який використовується. Якщо в документації до двигуна вказано кут одного кроку, то слід розділити 360° на цей кут, що дасть кількість кроків. Так, наприклад, якщо один крок складає 3,6°, то кількість кроків на один повний оберт буде 100 (360° / 3,6° = 100 кроків); pin1, pin2: номери двох виходів, які використовуються для управління кроковим двигуном; pin3, pin4 (опціональні параметри) – два додаткові виходи для випадку використання 4-х провідної схеми підключення крокового двигуна до контролера. Значення, що повертається конструктором – новий екземпляр об'єкта класу Stepper.

Функція Stepper.step(steps) обертає крковий двигун на певну кількість кроків, на швидкості, заданою функцією setSpeed(). Ця функція є блокуючою, тобто вона очікує закінчення обертання двигуна, перш ніж передати управління в наступний рядок коду. Наприклад, якщо швидкість обертання встановлена на 1 RPM (1 оберт в хвилину), і викликана функція step(100) на двигуні з повним оборотом в 100 кроків, то виконання цієї

функції займе цілу хвилину. Правильніше організовувати управління так, щоб швидкість була висока, а за один виклик step() робилося лише кілька кроків, для уникнення тривалого блокування виконання коду скетчу. Функція має наступні параметри: steps (кількість кроків) – позитивне число викликає обертання в один бік, негативне значення – в протилежний. Функція не повертає жодного значення.

Функція Stepper.setSpeed(rpms) встановлює швидкість обертання в обертах на хвилину (RPMs – rotations per minute). Дана опція не примушує двигун обертатися, а лише встановлює швидкість обертання, яка буде використана при виконанні функції step(). Параметри функції: rpms – швидкість, з якою буде відбуватися обертання крокового двигуна, виражається в обертах на хвилину. Значення має бути позитивним цілим. Опис бібліотеки Stepper для керування кроковими двигунами можна знайти за посиланням: http://arduino.ru/Reference/Library/Stepper

Результати моделювання роботи системи керування кроковим двигуном представлено на рис. 1.52, а, б. На рисунках видно, що кут повороту крокового двигуна змінюється і поступово збільшується.



а б

Рис. 1.52.

На рис. 1.53 представлена осцилограма сигналів керування, що поступають на кроковий двигун і змушують його обертатися.

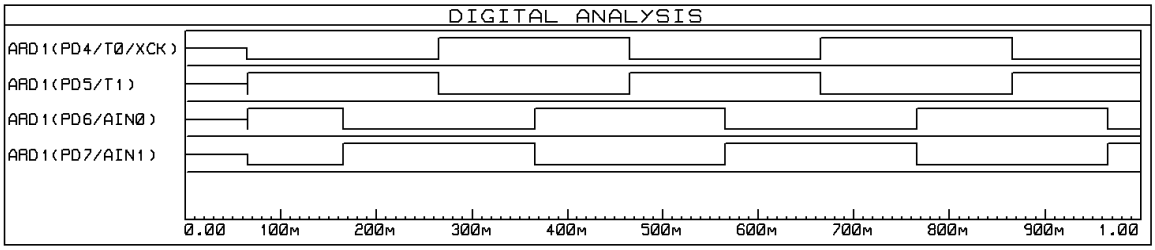


Рис. 1.53.

Ще один приклад схеми системи керування кроковим двигуном представлено на рис. 1.54.

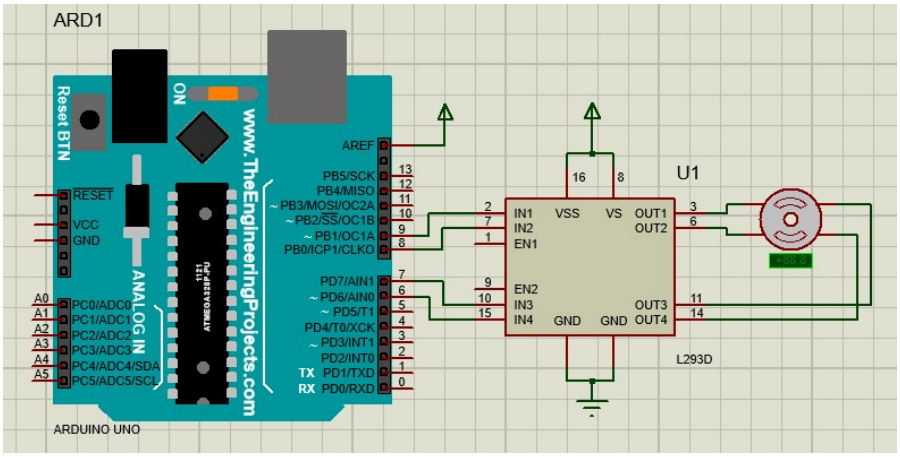


Рис. 1.54.

В даній схемі використовується спеціальна мікросхема-драйвер L293D, яка також виконує функцію підсилювача потужності, адже навантажувальна здатність виводів контролера Arduino недостатня для безпосереднього керування уніполярним крововим двигуном. Текст програми представлено нижче.

```
#include <Stepper.h> // підключення бібліотеки для роботи з крововим двигуном
// створення об’єкта для керування крововим двигуном
// в якості параметрів передаються значення кількості кроків на 1 оберт та
// номери виводів контролера Arduino, до яких підключено обмотки двигуна
Stepper mystepper = Stepper(72, 6, 7, 9, 8);

void setup() {
 mystepper.setSpeed(1); // ця функція встановлює швидкість обертання 1 об/хв.
}

void loop() {
 mystepper.step(1);   // здійснити оберт на 1 крок
 delay(100);    // затримка 100 мс щоб завершилися всі перехідні процеси в обмотках
}
```

1.6. Використання функцій в програмах

Розбиття на сегменти коду функціями дозволяє створювати частини коду, які виконують певні завдання. Після виконання відбувається повернення в те місце програми, звідки була викликана функція. Причиною створення функцій є необхідність виконувати однакову дію кілька разів. Для програмістів, які працюють або працювали з BASIC, функції в Arduino дозволяють використовувати підпрограми (GOSUB в BASIC).

Використання в коді функцій має ряд переваг:

1. Функції дозволяють організувати програму. Дуже часто функції допомагають заздалегідь скласти концепцію програми.

2. Функції кодують одну дію в одному місці програми. Далі необхідно тільки налагодити код функції.

3. Функції скорочують ймовірність появи помилки при необхідності зміни коду.

4. Функції скорочують текст скетчів і роблять його компактним, тому що деякі секції використовуються багато разів.

5. Функції полегшують використання коду в інших програмах роблячи його модульним. У цьому випадку функції мають ще одну невелику перевагу, роблячи код програми легким для читання та розуміння.

На рис. 1.55 представлено приклад опису функції та позначено основні елементи. З яких складається опис функції. Для виклику функції множення їй передаються параметри даних:

```
void loop ()
{
   int i = 2; // оголошення першої змінної і присвоювання їй значення 2
   int j = 3; // оголошення другої змінної і присвоювання їй значення 3
   int k; // оголошення змінної, в яку функція повертає значення
    k = myMultiplyFunction(i, j); // змінна k буде містити значення 6
 }
```

Створену функцію необхідно задекларувати поза дужками будь-якої іншої функції. Функція myMultiplyFunction() може бути описана вище або нижче функції loop().

Існують дві обов'язкові функції в скетчах Arduino. Це функції setup() і loop(). Інші функції повинні створюватися за дужками цих функцій. У наступному прикладі буде створена проста функція множення двох чисел.

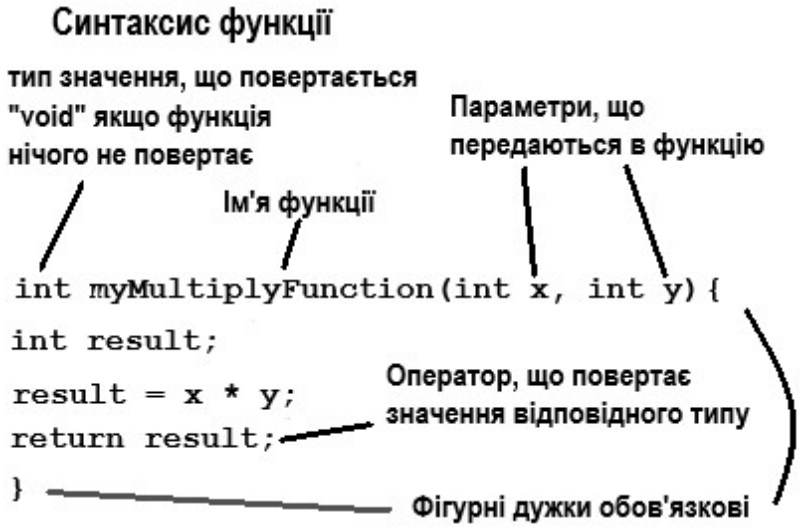


Рис. 1.55.

Весь скетч буде виглядати наступним чином:

```
void setup() {
// налаштування швидкості передачі даних по послідовному інтерфейсу
  Serial.begin(9600);
}
```

```
 void loop() {
   int i = 2; // оголошення змінної і присвоювання їй значення 2
   int j = 3; // оголошення ще однієї змінної і присвоювання їй значення 3
   int k; // оголошення змінної, в яку функція повертає значення
   // виклик функції перемноження двох чисел
   k = myMultiplyFunction (i, j); // змінна k буде містити значення 6
   Serial.println(k); // передавання значення змінної k по послідовному інтерфейсу
   delay (500); // затримка 500 мс
}
 // реалізація функції перемноження двох чисел
// в результаті перемноження двох чисел функція повертає результат – ціле число
// функція приймає два параметри – числа для перемноження
int myMultiplyFunction(int x, int y) {
   int result; // змінна для збереження результату перемноження x на y
   result = x * y; // виконання перемноження двох чисел, що передані в якості параметрів
   return result; // повернення результату перемноження в основну програму
}
```

На рис. 1.56 представлено схему для перевірки роботи програми. За допомогою віртуального термінала можна бачити. Що контролер передає значення 6 по послідовному комунікаційному інтерфейсу, яке є результатом перемноження чисел 2 та 3.

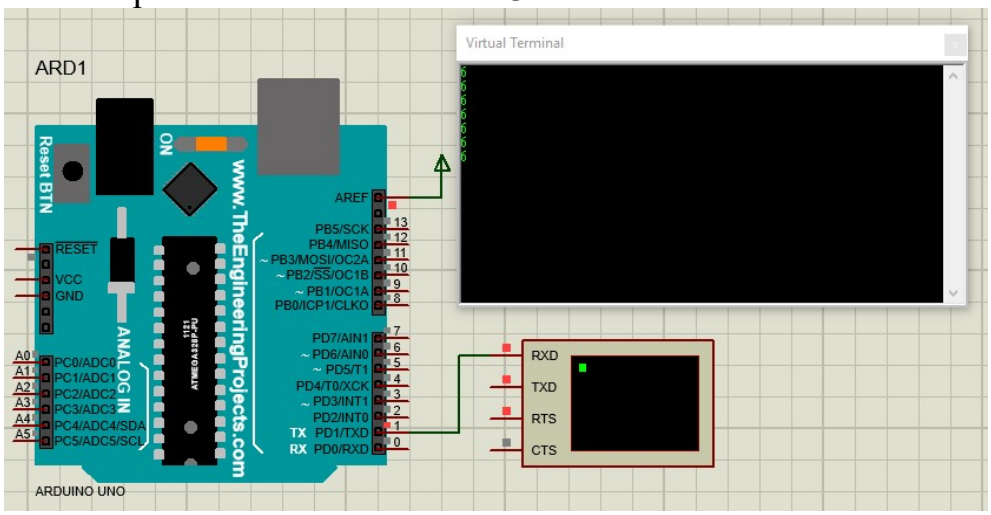


Рис. 1.56.

В наступному прикладі демонструється використання функції для усереднення значень, які зчитуються з аналогового датчика температури. На схемі, що представлена на рис. 1.57, використовується датчик температури, на виході якого формується аналоговий сигнал напруги, пропорційний температурі. В схемі також є вольтметр для того, щоб встановити відповідність між температурою та напругою на виході датчика. Зчитані з датчика та усереднені дані про температуру передаються по послідовному комунікаційному інтерфейсу, і за допомогою віртуального термінала можна спостерігати дані, що передаються.

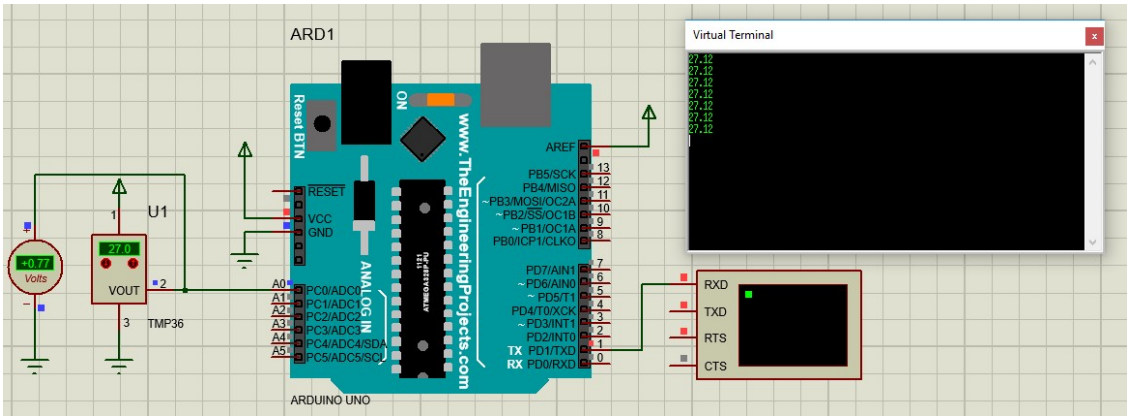


Рис. 1.57.

Скетч для Arduino представлено нижче.

```
void setup() {
// налаштування швидкості передачі даних по послідовному інтерфейсу
  Serial.begin(9600);
}

void loop() {
  double T; // оголошення змінної, в яку функція повертає значення температури
  // виклик функції зчитування та усереднення температури
  T = ReadTemperature(); // змінна T буде усереднене значення температури
  Serial.println(T); // передавання значення змінної T по послідовному інтерфейсу
  delay (500); // затримка 500 мс
}
// реалізація функції зчитування температури з аналогового датчика
// та розрахунку усередненого значення на основі 5 вимірювань
double ReadTemperature(){
 int i; // оголошення змінної для використання в циклі
// в якості лічильника ітерацій
 double avgT = 0; // оголошення змінної для зберігання даних про температуру
  for (i = 0; i < 5; i++)  // в циклі виконується 5 ітерацій
  {
  // зчитування даних з аналогового датчику та накопичення протягом
  // 5 вимірювань; ділення на коефіцієнт 5.82488889 потрібне для
  // перетворення з апаратних одиниць в значення температури, представленої
  // в градусах
   avgT = avgT + analogRead(A0)/5.82488889;
 }
  avgT = avgT / 5;     // розрахунок середнього значення температури
// отриманої з датчика за 5 вимірювань
  return avgT; // повернення результату в основну програму
}
```

На рис. 1.57 у вікні віртуального терміналу можна бачити усереднене значення температури. Це значення передається по послідовному комунікаційному інтерфейсу кожні 500 мс. Якщо реальний контролер Arduino підключити до комп’ютера, то за допомогою програми-термінала можна буде спостерігати за значенням температури в тому місці, де

розташовано датчик. Тобто це буде мікропроцесорна система моніторингу температури. В одному з наступних розділів розглядаються принципи побудови програмного забезпечення для візуалізації даних, отриманих по послідовному інтерфейсу, у графічному вигляді.

Тип даних double, на відміну від більшості мов програмування, має ту ж точність, що і тип float, і займає 4 байти в пам'яті. Тип double підтримується в Arduino для сумісності коду з іншими платформами.

Тип даних float служить для зберігання чисел з плаваючою комою. Цей тип часто використовується для операцій з даними, що зчитуються з аналогових входів. Діапазон значень – від -34028235E+38 до 3.4028235E+38. Змінна типу float займає 32 біта (4 байти) в пам'яті.

Тип float має точність 6-7 знаків (мається на увазі всі знаки, а не тільки мантиса). Зазвичай для збільшення точності використовують інший тип – double, але на платформі Arduino, double і float мають однакову точність.

Зберігання в пам'яті чисел з плаваючою точкою в двійковій системі обумовлює втрату точності. Так, наприклад, 6.0/3.0 не обов'язково дорівнює 2.0. Порівнюючи два числа з плаваючою точкою слід перевіряти не точну рівність, а різницю між цими числами, чи вона менша за деяке граничне значення. Слід також враховувати, що арифметичні операції над числами з плаваючою комою виконуються значно повільніше, ніж над цілими числами.

Тип даних int (від англ. integer – ціле число) один з найбільш часто типів даних, що використовуються для зберігання чисел. Змінна типу int займає 2 байта пам'яті, і може зберігати числа від -32768 до 32767 (від -2^{15} до $2^{15}-1$).

Для розміщення від'ємних значень int використовує так званий додатковий код представлення числа. Старший біт вказує на негативний знак числа, інші біти інвертуються з додаванням 1.

Arduino компілятор сам розміщує змінні в пам'яті і виконує представлення негативних чисел, тому арифметичні дії над цілими числами виконуються як зазвичай.

1.2.7. Використання масивів в програмах

Масиви Arduino – це елемент мови, що активно використовується програмістами для роботи з наборами однотипних даних. Масиви є практично у всіх мовах програмування, не виключенням є і Arduino, синтаксис якого сильно схожий на C++. Є багато прикладів, коли без використання масивів практично неможливо написати працездатну програму. Незнання нюансів і особливостей роботи з масивами в Arduino часто призводить до прихованих або явних помилок роботи скетчу.

Масив в будь-якій мові програмування – це деяка ділянка пам'яті, поділена на елементарні комірки, доступ до вмісту яких можливий за їх індексом. Зрозуміти масиви можна уявивши довгий коридор з дверима по одній стороні. Можна без помилки знайти потрібні двері, якщо знати її

номер від початку коридору. Просто почавши рахувати по порядку кожні двері на своєму шляху, можна дістатися до потрібної і увійти в неї. Все, що потрібно знати – це номер (індекс). По суті своїй, масиви – це великий набір змінних, у яких є загальне ім'я, а для пошуку кожної окремої змінної використовується просто її порядковий номер.

Всі основні операції з масивами можна розділити на наступні групи:

– створення та ініціалізація масиву;

– операції з елементами масиву (пошук, додавання, видалення);

– зміна довжини масиву;

– видалення масиву.

При створенні обов'язково потрібно вказувати тип масиву, назву масиву і спеціальні символи "[" та "]". Також обов'язково задається розмір масиву явним або неявним чином. Нижче показаний приклад створення масиву з 5 елементів типу int.

```
int sensorValues[5];
```

У цьому прикладі int – тип даних, які міститимуться в масиві; sensorValues – назва масиву; [5] – розмір масиву (компілятор виділять пам'ять під цей масив розміром 10 байтів – 5 елементів по 2 байта кожний).

При створенні масиву можна відразу вказувати його вміст. Це робиться за допомогою символів {}:

```
int arrPins[] = {3, 6, 4, 3, 7};
```

В даному випадку оголошуються елементи, а розмір не вказується. Розмір масиву розраховується компілятором автоматично (в наведеному прикладі визначено 5 елементів, отже розмір масиву дорівнює 5). Можна одночасно задати розмір масиву і його елементи:

```
int arrValues[6] = {5, 654, 120, 982, 15};
```

Отримати потрібний елемент масиву можна, вказавши його порядковий номер. Індексація масивів в Arduino, так само як в C++, починається з 0. Першим елементом масиву буде елемент з індексом 0.

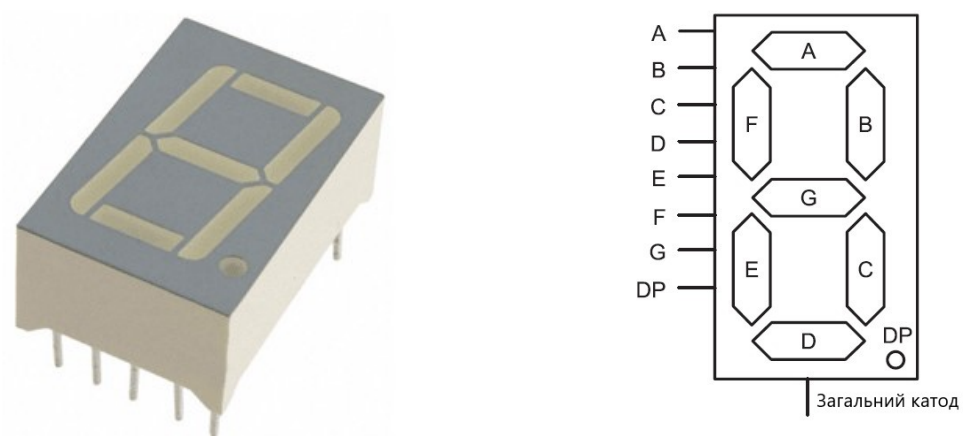
Цикли – зручний метод для роботи з масивами, вони застосовуються для повторення однакових дій, роблячи код програми компактним. Індексація кожного елемента в такому випадку здійснюється за допомогою змінної-лічильника циклу. Наприклад, для відображення на моніторі елементів масиву можна використовувати такий цикл for:

```
for (int i = 0; i < 6; i++) {
 Serial.println(arrValues[i]); // друкуємо кожен з 6 елементів масиву
}
```

За допомогою масивів в Arduino можна оперувати з великими наборами даних, наприклад, збирати інформацію з багатьох підключених даних або формувати сигнали для декількох підключених світлодіодів. Масиви використовують для зберігання рядків і досить складно організованих структур даних. Використання масивів у Arduino багато в чому схоже на C++ і вимагає розуміння принципів адресної арифметики, постійного контролю з боку програміста за розмірністю і типами. Але натомість масиви надають гнучкі можливості по зберіганню і динамічній зміни наборів даних. Без них написання складних програм практично неможливо.

1.8. Керування напівпровідниковим семисегментним індикатором

Розглянемо приклад використання масиву в програмі керування семисегментним індикатором. Семисегментний індикатор, як зрозуміло з його назви, складається з семи елементів індикації (сегментів), що включаються і виключаються окремо. Включаючи їх в різних комбінаціях, з них можна скласти зображення арабських цифр. Сегменти позначаються буквами від A до G; восьмий сегмент – десяткова кома, призначена для відображення дрібних чисел. На рис. 1.58, а, представлено зовнішній вигляд семисегментного індикатора, а на рис. 1.58, б, позначення його виводів.



а б

Рис. 1.58.

В табл. 1.2 наводиться інформація про сегменти. Які потрібно увімкнути для того, щоб сформувати відповідну цифру. Наприклад, для цифри 7 треба включити сегменти A, B і C, тобто видати на виводи, до яких підключені ці сегменти, значення HIGH, а на інші LOW.

Таблиця 1.2.

Цифра	Сегменти
0	ABCDEF
1	BC
2	ABDEG

Цифра	Сегменти
3	ABCDG
4	BCFG
5	ACDFG
6	ACDEFG
7	ABC
8	ABCDEFG
9	ABCDFG

На рис. 1.59 представлено схему для демонстрації програми керування семисегментним індикатором. В даному прикладі на індикатор послідовно виводяться цифри від 0 до 9 через проміжок часу 1 секунда.

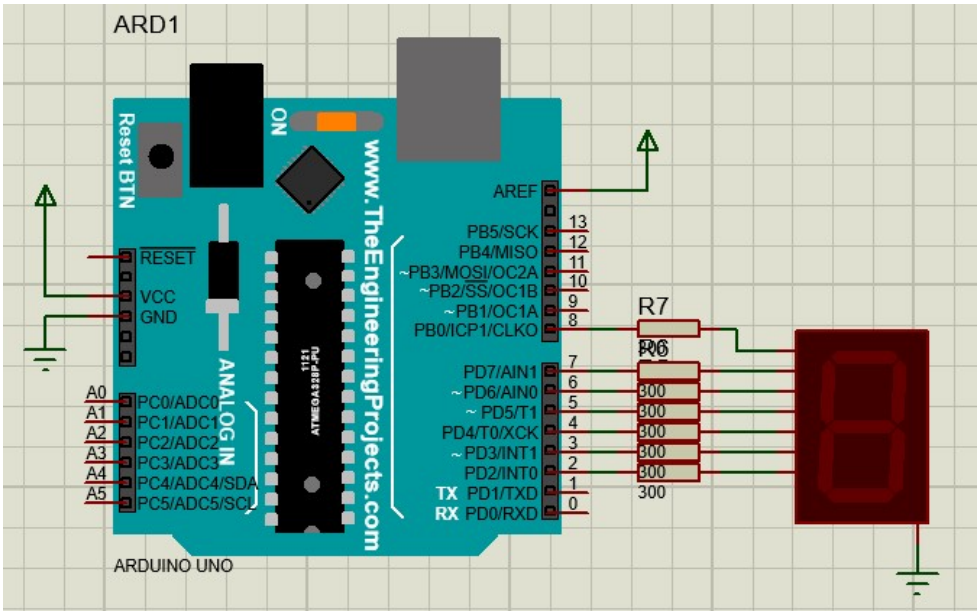


Рис. 1.59.

Текст програми представлено нижче.

```
// в масиві зберігаються номери виводів, до яких підключені сегменти
int pins[7]={8,7,6,5,4,3,2};
// в масиві зберігаються значення, що представляють кожну цифру
// якщо встановити виводи порту у відповідний стан 0 (LOW) або 1 (HIGH)
byte numbers[10] = { B11111100, B01100000, B11011010, B11110010, B01100110,
B10110110, B10111110, B11100000, B11111110, B11110110};
// змінна для зберігання значення
int number=0;

void setup() // функція ініціалізації контролера
{
// всі виводи контролера налаштовуються на вихід
// в циклі виконується послідовний доступ до кожного елемента масиву
for(int i=0;i<7;i++)
 pinMode(pins[i],OUTPUT); // налаштування відповідного виводу на вихід
}
```

```
void loop() // функція, в якій реалізовано основну логіку програми
{
 showNumber(number); // виклик функції відображення цифри на індикаторі
 delay(1000); // часова затримка 1 секунда
 number=(number+1)%10; // збільшення значення змінної на 1
 // залишок від ділення на 10 використовується для того, щоб не вийти за межі
 // масиву, адже залишок ділення на 10 не може перевищувати значення 9
}
// функція виводу цифри на семисегментний індикатор
void showNumber(int num)
{
 for(int i=0;i<7;i++)
 {
  if(bitRead(numbers[num],7-i)==HIGH) // якщо біт в масиві встановлений
  {
  digitalWrite(pins[i], HIGH); // увімкнути сегмент
}
else  // інакше
{
  digitalWrite(pins[i],LOW); // вимкнути сегмент
} // закриваюча дужка else
} // закриваюча дужка циклу for
} // закриваюча дужка тіла функції showNumber
```

В цій програмі було використано функцію bitRead(). Ця функція зчитує біт числа. Синтаксис функції:

```
bitRead(x, n);
```

Параметри, що передаються в функцію: x – номер біта, який зчитувати; n – який біт зчитувати, починаючи з 0 для найменш значущого (крайній праворуч) розряду. Функція повертає значення біта (0 або 1).

Наступний приклад демонструє обробку подій користувача. Схема, представлена на рис. 1.60. представляє собою лічильник. При натисканні на кнопку, яка позначена «+1» відбувається збільшення лічильника на 1, а при натисканні на кнопку, яка позначена «-1», відбувається зменшення значення лічильника на 1. При натисканні на кнопку «set 0» відбувається скидання лічильника і встановлення значення 0. Максимальне значення, яке може мати лічильник, складає 9, а мінімальне 0. При досягненні максимального або мінімального значення, натискання кнопки інкременту, і, відповідно, декременту, не буде мати ефекту. Також для запобігання появи на входах контролера невизначених значень використовується режим роботи виводів з увімкненими підтягуючими резисторами.

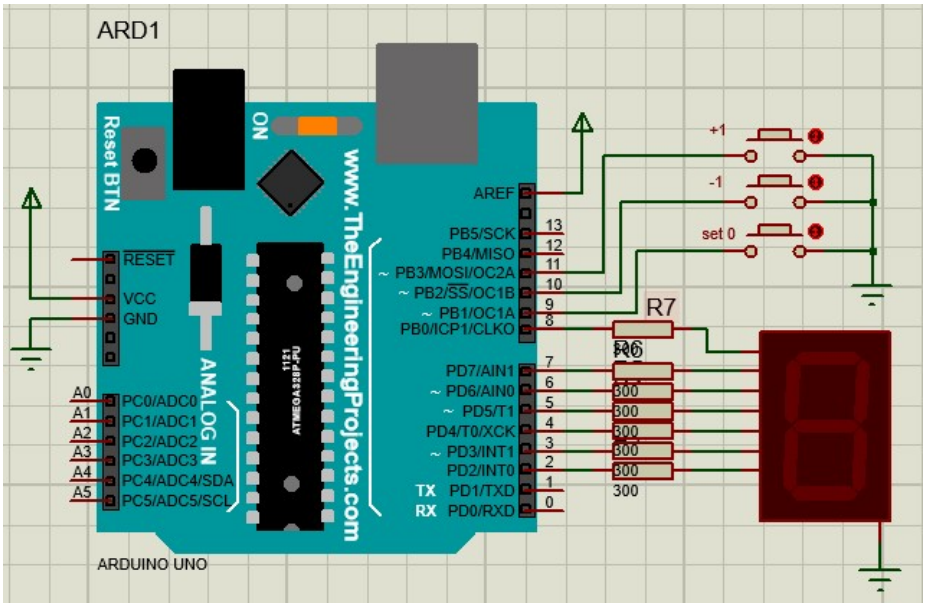


Рис. 1.60.

Текст програми представлено нижче.

```
// номер виводу, до якого підключена кнопка збільшення значення на 1
const int incButtonPin = 11;
// номер виводу, до якого підключена кнопка зменшення значення на 1
const int decButtonPin = 10;
// номер виводу, до якого підключена кнопка скидання лічильника в 0
const int resetButtonPin = 9;
        // в масиві зберігаються номери виводів, до яких підключені сегменти
int pins[7]={8,7,6,5,4,3,2};
// в масиві зберігаються значення, що представляють кожну цифру
// якщо встановити виводи порту у відповідний стан 0 (LOW) або 1 (HIGH)
byte numbers[10] = { B11111100, B01100000, B11011010, B11110010, B01100110,
B10110110, B10111110, B11100000, B11111110, B11110110};
// змінна для зберігання значення
int number=0;

void setup() // функція ініціалізації контролера
{
  pinMode(incButtonPin,INPUT); // налаштування виводу 11 на вхід
  digitalWrite(incButtonPin,HIGH); // вмикання підтягуючого резистора
  pinMode(decButtonPin,INPUT); // налаштування виводу 10 на вхід
  digitalWrite(decButtonPin,HIGH); // вмикання підтягуючого резистора
  pinMode(resetButtonPin,INPUT); // налаштування виводу 9 на вхід
  digitalWrite(resetButtonPin,HIGH); // вмикання підтягуючого резистора
// всі виводи контролера налаштовуються на вихід
// в циклі виконується послідовний доступ до кожного елемента масиву
for(int i=0;i<7;i++)
pinMode(pins[i],OUTPUT); // налаштування відповідного виводу на вихід
}

void loop() // функція, в якій реалізовано основну логіку програми
{
```

```
// якщо натиснута кнопка збільшення значення лічильника на 1, а також поточне
// значення лічильника менше 9
 if((digitalRead(incButtonPin) == LOW) && (number<9))
 {
  number++; // збільшення значення на 1 (інкремент)
  // очікування доки кнопка «+1» не буде відтиснута
  while(digitalRead(incButtonPin) == LOW);
 }
// якщо натиснута кнопка зменшення значення лічильника на 1, а також поточне
// значення лічильника більше 0
 else if((digitalRead(decButtonPin) == LOW) && (number>0))
 {
  number--; // зменшення значення на 1 (декремент)
  // очікування доки кнопка «-1» не буде відтиснута
  while(digitalRead(decButtonPin) == LOW);
 }
 // якщо натиснута кнопка скидання
 else if(digitalRead(resetButtonPin) == LOW)
 {
  number = 0; // запис в змінну значення 0
 }
showNumber(number); // виклик функції відображення цифри на індикаторі
delay(10);  // виклик функції формування часової затримки 10 мс
}

// функція виводу цифри на семисегментний індикатор
void showNumber(int num)
{
 for(int i=0;i<7;i++)
 {
  if(bitRead(numbers[num],7-i)==HIGH) // якщо біт в масиві встановлений
  {
  digitalWrite(pins[i], HIGH); // увімкнути сегмент
  }
else  // інакше
{
  digitalWrite(pins[i],LOW); // вимкнути сегмент
} // закриваюча дужка else
} // закриваюча дужка циклу for
} // закриваюча дужка тіла функції showNumber
```

На завершення першого розділу ще можна додати декілька слів про те, як запускається програма на Arduino. Коли контролер Arduino підключається до живлення, то всередині нього починається виконання вбудованих мікропрограм. Мікроконтролер налаштований так, що при запуску системи управління отримує програма-завантажувач. Перше, що робить завантажувач – перевіряє протягом 1-2 секунд чи не почнеться відправка нової програми від користувача. Якщо процес перепрограмування розпочато, то скетч завантажується в пам'ять і

управління віддається йому. Якщо нових програм немає, то завантажувач виконує раніше збережену програму.

Завантаживши програму, Arduino дає коду можливість взяти участь в ініціалізації системи. Для цього потрібно вказати мікроконтролеру команди, які він виконає в момент завантаження лише один раз. Тобто ці команди виконаються тільки один раз при старті системи. І саме з цією метою в потрібно виділити блок, в якому будуть зберігатися ці команди. Функція void setup(), а вірніше простір всередині фігурних дужок цієї функції, і є таким місцем всередині Arduino скетчу. Не слід забувати ставити фігурні дужки. Втрата хоча б однієї дужки відразу зробить всю програму незрозумілою для Arduino. Але і зайві дужки теж не потрібно ставити – це так само призведе до помилки.

Найчастіше всередині void setup вказуються такі інструкції:

– pinMode із зазначенням номера та типу виводу. Ця інструкція визначає режим роботи виводів Arduino;

– Serial.begin із зазначенням швидкості (найчастіше, 9600). Ця інструкція ініціалізує роботу з послідовним портом на зазначеній швидкості;

– інструкції підключення та ініціалізації різних об'єктів бібліотек Arduino. Наприклад, servo.atach(6) вкаже бібліотеці, що сервопривод було підключено до 6 виводу, і всі наступні дії код бібліотеки буде здійснювати саме з цим портом;

– ініціалізацію глобальних змінних, якщо з якихось причин не можемо зробити це при визначенні змінних в глобальному контексті.

– виконання інших налаштувань і початкових значень для змінних і об'єктів.

Функція loop() – це те місце, куди слід помістити команди, які будуть виконуватися весь час, поки подано живлення на плату Arduino. Почавши виконання з першої команди, мікроконтролер дійде до кінця і відразу ж перестрибне в початок, щоб повторити ту ж послідовність. І так безліч разів (до тих пір, поки на плату буде подане живлення). Найбільш доречний переклад англійського слова loop в даному випадку – це цикл (петля). За своєю суттю, void loop – це головна функція, точка входу в нашу програму.

Почавши виконання програми, Arduino виконує ряд рутинних операцій по ініціалізації та налаштуванню середовища оточення, і тільки потім приступає до виконання того самого коду, який міститься в ваших скетчах. Таким чином, Arduino позбавляє нас від необхідності пам'ятати всі деталі архітектури мікропроцесора і сконцентруватися на вирішенні самої задачі. Але це не означає, що не потрібно розуміти, що ж відбувається всередині мікроконтролера. В наступних розділах більш глибоко будуть розглянуті та розкриті питання побудови мікропроцесорних систем збору, обробки інформації та керування.

Завдання до розділу 1

1. Реалізувати ефект «бігучий вогник» на світлодіодних індикаторах.
2. Реалізувати 5 світлових ефектів. Вид ефекту вибирається натисканням відповідних кнопок.
3. Написати програму, яка на чотирьох семисегментних індикаторах виводить значення напруги, зчитане з АЦП.
4. При натисканні на кнопку запустити перетворення АЦП і вивести результат на чотирьох семисегментних індикаторах.
5. Написати програму, яка керує яскравістю / частотою мерехтіння світлодіодів в залежності від значення, зчитаного з АЦП.
6. Виміряти час, протягом якого значення напруги на вході АЦП перевищує деякий заданий рівень. Відобразити значення інтервалу часу на алфавітно-цифровому рідкокристалічному індикаторі.
7. Розробити вольтметр. У верхній частині РКІ відображати напругу на першому каналі АЦП, в нижньому рядку відображати значення напруги на другому каналі АЦП. Змінювати рівень напруги можна за допомогою змінних резисторів.
8. Написати програму, яка передає в комп'ютер номер натиснутої кнопки. Розробити відповідну схему.
9. Розробити схему та написати програму, яка на семисегментному індикаторі виводить прийняті значення (цифри від 0 до 9).
10. Написати програму, яка після прийому значення від комп'ютера (цифри від 0 до 9) встановлює відповідну яскравість/частоту мерехтіння світлодіодів.
11. Написати програму, яка виконує аналого-цифрове перетворення і відправляє отримане значення в комп'ютер.
12. Розробити схему та написати програму, що реалізує меню на екрані алфавітно-цифрового рідкокристалічного індикатора. Навігація по меню здійснюється за допомогою тактових кнопок, підключених до контролера.

РОЗДІЛ 2. ДАТЧИКИ, ЩО ВИКОРИСТОВУЮТЬСЯ В МІКРОПРОЦЕСОРНИХ СИСТЕМАХ КЕРУВАННЯ, ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ

2.1. Основні характеристики датчиків

Для більшості фізичних величин існує велика кількість технологій вимірювання, які характеризуються залежністю між сигналом, що виробляється, та величиною, яка вимірюється. Вимірюючий пристрій, або датчик, в загальному випадку складається з двох частин – вимірюючої головки та перетворювача.

Результат вимірювання – це реакція вимірюючої головки датчика, яка на виході перетворювача представляє собою електричну величину. В більшості електронних систем цей вихідний сигнал електричний, однак доволі часто зустрічаються й пневматичні датчики. Головна перевага електричних датчиків – це гнучкість та різноманітність способів обробки сигналу. Розрізняють три класи датчиків:

– аналогові датчики, тобто датчики що виробляють аналоговий сигнал;

– цифрові датчики, що генерують послідовність імпульсів або двійкове слово;

– бінарні датчики, що виробляють сигнал лише двох рівнів: 0 або 1.

Два важливих параметри – ширина смуги пропускання та рівень шуму – визначають спосіб передачі сигналів між елементами (підсистемами) електронної системи. Смуга пропускання є важливим параметром для багатьох систем – передача даних, системні шини, керування зі зворотнім зв’язком. Однак цей термін має різні значення. В системах передачі інформації та керування зі зворотнім зв’язком смуга пропускання означає діапазон частот, у межах котрого амплітудно-частотна характеристика залишається менше заданого значення (0,707 від максимального). Для системних шин смуга пропускання визначається терміном «пропускна здатність». При обробці сигналів керування та моніторингу смуга пропускання визначається як діапазон робочих частот датчика або виконуючого механізму – лише ті фізичні величини, робочі частоти яких лежать в межах смуги пропускання, можна належним чином виміряти та обробити. Чим ширша смуга пропускання, тим швидшою буде реакція датчика. Останнє не зажди є позитивним фактором, оскільки в такому випадку система буде більш чутливою до небажаних високочастотних збурень.

Будь-який виміряний сигнал спотворюється збуреннями та шумом як в процесі формування, так і в процесі передачі. Одна з основних проблем передачі сигналу – зменшення впливу шуму. Джерела шуму повинні бути ізольовані, або їх вплив повинен бути зменшений до мінімально можливого рівня.

Аналогові сигнали, що виробляються вимірювальними пристроями, необхідно перетворити для передачі. Наприклад, якщо сигнал у вигляді

напруги потрібно передати в комп'ютер, то він повинен бути підсилений так, щоб відповідати діапазону напруг інтерфейсу комп'ютера. Іноді рівень напруги датчика повинен бути зміщений. Ця процедура називається узгодженням сигналу.

На сигнал, що передається від датчика по електричному провіднику, може впливати шум середовища через наявність небажаних зв'язків резистивного, індуктивного або ємнісного характеру. Цей шум може спотворити сигнал. Одне з можливих рішень – перетворити аналоговий сигнал в послідовність імпульсів, частота або тривалість яких пов'язана з рівнем вхідного сигналу, а потім передавати перетворений сигнал. Послідовність імпульсів може передаватися або по електричному, або по оптичному світловоду.

Датчик повинен відтворювати фізичну величину максимально швидко і точно. Хоча найчастіше датчик обирають виходячи з надійності, а не зручності обслуговування, його точність, стабільність та повторюваність результатів залишаються найважливішими факторами. Основою роботи багатьох електронних систем керування, збору, обробки та відображення інформації є вхідна інформація, тому точні та надійні вимірювання – це необхідна умова якості функціонування електронних систем.

Більша частина характеристик датчика, що приводяться в технічній документації, – статичні параметри. Ці параметри не показують, наскільки швидко та точно датчик може виміряти сигнал, що змінюється з великою швидкістю. Властивості, що відображають роботу датчика в умовах вхідних впливів, що змінюються, називаються динамічними характеристиками. Вони суттєво впливають на роботу системи. Ідеальний датчик миттєво реагує на зміну фізичної величини. На практиці будь-якому датчику необхідний деякий час на відпрацювання нового вхідного сигналу. Для адекватного відображення реальних змін величини, за якою ведеться спостереження, час реакції датчика повинен бути якнайменшим. Однак часто потрібно шукати компроміс між швидкістю реакції датчика та його чутливістю к шуму.

Точність датчика визначає різницю між виміряною та дійсною величиною. Вона може бути віднесена до датчика в цілому або до конкретного його показання. Роздільна здатність – це найменше відхилення величини, що вимірюється, яке може бути зафіксовано датчиком. Точність залежить не лише від апаратної частини датчика, але й від інших елементів вимірювального комплексу. Похибка вимірювання визначається як різниця між виміряною та дійсною величинами. Оскільки дійне значення невідоме, оцінку точності можна виконати на основі еталонних вимірювань або поглибленого аналізу даних.

Помилки вимірювання можна класифікувати як детерміновані (систематичні) та випадкові (стохастичні). Детерміновані помилки пов'язані з несправністю датчика, порушенням умов його використання або процедури вимірювання. Ці помилки повторюються при кожному

вимірюванні. Типові систематичні помилки – це зсув показань. Систематичні помилки виправляються при повірках датчиків. Випадкові помилки можуть мати різноманітну природу. В більшості випадків – це вплив оточуючого середовища (температури, вологості та ін.). Якщо причини випадкових помилок відомі, то ці помилки можна компенсувати. Вплив випадкових збурень часто характеризують кількісно такими параметрами, як середня помилка (математичне очікування), середньоквадратична помилка або стандартне відхилення, розкид або похибка.

Динамічні властивості датчика характеризуються цілим рядом параметрів, які доволі рідко приводяться в технічному описі виробника. Динамічну характеристику датчика можна експериментально отримати як реакцію на стрибок вхідної величини, що вимірюється (рис. 2.1). Параметри, що описують реакцію датчика, дають уявлення про його швидкодію (час наростання, запізнення, досягнення першого максимуму), інерційних властивостях (відносне пере регулювання, час установлення) та точність (зсув).

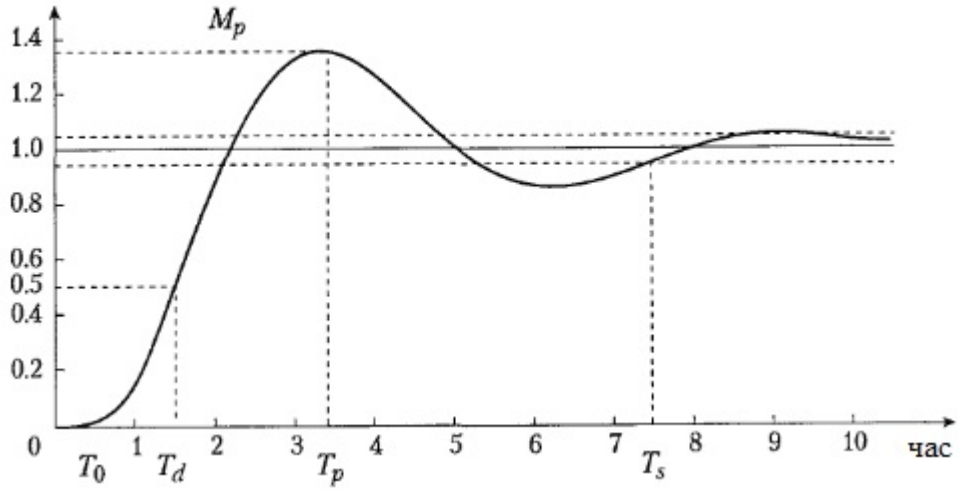


Рис. 2.1.

На рисунку позначено: T_0 – час проходження зони нечуттєвості; T_d – запізнення; T_P – час досягнення першого максимуму; T_s – час установлення; M_p – перерегулювання.

Слід намагатися мінімізувати наступні параметри.

Час проходження зони нечуттєвості – час між початком вимірювання фізичної величини та моментом реакції датчика, тобто моментом початку зміни вхідного сигналу.

Запізнення – проміжок часу, після якого показання датчика перший раз досягають значення 50% сталого значення.

Час наростання – проміжок часу, протягом якого вихідний сигнал збільшується від 10% до 90% сталого значення. Малий час наростання вказує на швидку реакцію.

Перерегулювання – час досягнення першого максимуму сигналу.

Час перехідного процесу, час установлення – проміжок часу, після завершення якого відхилення виходу датчика від сталого значення не перевищує деяке значення (наприклад ±5%).

Відносне перерегулювання – різниця між максимальним та сталим значеннями, віднесена до сталого значення (у відсотках).

Статична помилка – відхилення вихідної величини датчика від дійсного значення. Може бути усунена калібруванням датчика.

В реальних умовах деякі вимоги до датчиків у протирічять один одному, тому всі параметри неможливо мінімізувати одночасно.

Статичні характеристики датчика показують, наскільки коректно вихід датчика відображає виміряну величину через деякий час після її зміни, коли вхідний сигнал прийняв нове значення. Важливими статичними параметрами є: чутливість, дозволяюча здатність, лінійність, зсув нуля, робочий діапазон, повторюваність результатів.

Чутливість датчика визначається як відношення величини вихідного сигналу до одиничної вхідної величини.

Дозволяюча здатність – це найменша зміна вимірюваної величини, що може бути зафіксована та точно показана датчиком.

Лінійність визначається виходячи з залежності вихідного сигналу від вхідного при стаціонарних умовах. Близькість цієї залежності до прямої лінії визначає ступінь лінійності датчика. Максимальне відхилення від лінійної залежності виражається у відсотках.

Підсилення по постійному струму (статичне підсилення) це коефіцієнт підсилення датчика на низьких частотах.

Робочий діапазон датчика визначається допустимими верхньою та нижньою межами значень вхідної величини та рівня вихідного сигналу.

Повторюваність характеризується як відхилення між декількома послідовними вимірюваннями при заданому значенні вимірюваної величини в однакових умовах.

Багато датчиків мають властивість нелінійності. Наприклад, якщо датчик досягає верхньої межі робочого діапазону, проявляється ефект насичення, тобто вихідний сигнал обмежений, навіть якщо вхідна величина збільшується.

Об’єднання декількох елементів в систему може суттєво вплинути на поведінку кожного з них в порівнянні з їх незалежним використанням. Наприклад, підключення вольтметра змінює струм та напругу в колі, а термопара може спотворити температуру, яка вимірюється. Це називається ефектом навантаження. Такі помилки можуть переважати всі інші типи помилок. Їх потрібно враховувати при сполученні різних датчиків та систем перетворення та передачі інформації.

2.2. Інфрачервоний датчик руху HC-SR501

PIR (пасивні інфрачервоні) датчики дозволяють визначати наявність інфрачервоного випромінювання. Зокрема, такі датчики здатні вловлювати рух людини. Такі датчики дуже часто використовуються в системах

сигналізації. Ці датчики малі за габаритами, недорогі, споживають мало енергії, легкі в експлуатації. Крім PIR, подібні датчики називають піроелектричними і інфрачервоними датчиками руху. На рис. 2.2 представлено зовнішній вигляд інфрачервоного датчика руху.

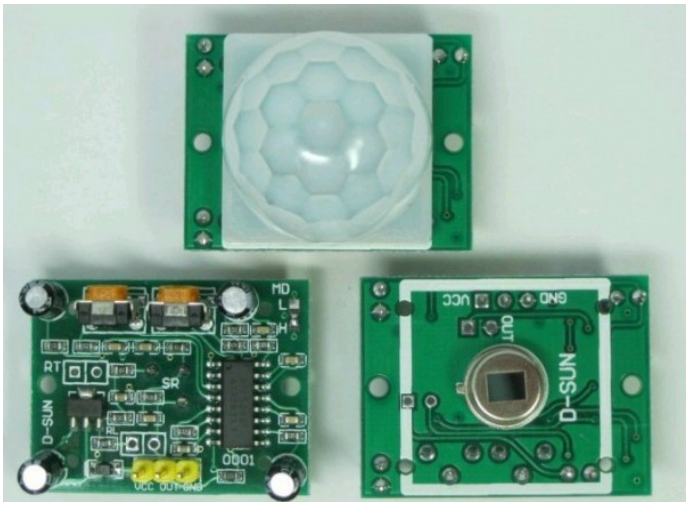


Рис. 2.2.

Інфрачервоні датчики руху по суті складаються з піроелектричного чутливого елемента (циліндрична деталь з прямокутним кристалом в центрі), який уловлює рівень інфрачервоного випромінювання. Все навколо випромінює невеликий рівень випромінювання. Чим більше температура, тим вище рівень випромінювання. Датчик фактично розділений на дві частини. Це обумовлено тим, що важливий не рівень випромінювання, а безпосередньо наявність рух в межах його зони чутливості. Дві частини датчика встановлені таким чином, що якщо одна половина вловлює більший рівень випромінювання, ніж інша, вихідний сигнал буде генерувати значення high або low.



Рис. 2.3.

Сам модуль, на якому встановлений датчик руху, складається також із додаткової електричної обв'язки: запобіжники, резистори і конденсатори. У більшості недорогих датчиків використовуються недорогі мікросхеми BISS0001 («Micro Power PIR Motion Detector IC»). Ця мікросхема сприймає зовнішнє джерело випромінювання і проводить мінімальну обробку сигналу для його перетворення з аналогового в цифровий вигляд. Новіші моделі PIR-датчиків мають додаткові виходи для додаткового налаштування і встановлені конектори для сигналу живлення і землі (рис. 2.3).

Більшість модулів з інфрачервоними датчиками руху мають три коннектора на задній частині. Один конектор підключений до землі, другий видає сигнал з сенсорів, третій – живлення. Напруга живлення зазвичай становить 3-5 В постійного струму. В результаті, коли PIR датчик виявить рух, на виході генерується сигнал HIGH, який відповідає рівню 3,3В.

У піроелектричного датчика руху є кілька налаштувань. На багатьох інфрачервоних датчиках руху встановлений невеликий потенціометр для настройки чутливості. Обертання потентенціометра за годинниковою стрілкою додає чутливість датчика.

Коли ми розглядаємо PIR датчики, важливі два проміжки часу «затримки». Перший відрізок часу – Tx: як довго на виході датчика буде присутнім сигнал високого рівня після виявлення руху. На багатьох піроелектричних модулях цей час регулюється вбудованим потенціометром. Другий відрізок часу – Ti: як довго на виході буде сигнал низького рівня, коли руху не було.

На датчиках є потенціометр, зазначений як TIME. Це змінний резистор з опором 1 МОм, який доданий до резистора на 10 кОм. Конденсатор C6 має ємність 0,01 мкФ, так що:

$$Tx = 24576 \times (10\ кОм + Rtime) \times 0{,}01\ мкФ$$

Коли потенціометр Rtime в «нульовому положенні» – повністю повернений проти годинникової стрілки (0 МОм):

$$Tx = 24576 \times (10\ кОм) \times 0{,}01\ мкФ = 2{,}5\ сек.$$

Коли потенціометр Rtime повністю повернений за годинниковою стрілкою (1 МОм):

$$Tx = 24576 \times (1010\ кОм) \times 0{,}01\ мкФ = 250\ сек.$$

У середній позиції RTime час буде складати близько 120 секунд (дві хвилини). Тобто, якщо потрібно відстежувати рух об'єкта з частотою раз в хвилину, треба повернути потенціометр на 1/4 оберти.

Датчики відмінно підходять для проектів, в яких необхідно визначати наявність або відсутність людини в межах певного робочого простору. Крім перерахованих вище переваг подібних датчиків, вони

мають велику зону чутливості. Однак піроелектричні датчики не нададуть інформації про те, скільки людей навколо і наскільки близько вони знаходяться до датчика.

PIR датчик не такі прості, як може здатися на перший погляд. Основна причина – велика кількість змінних, які впливають на його вхідний і вихідний сигнали. Щоб пояснити основи роботи датчиків, використовується рис. 2.4.

Піроелектричний датчик руху складається з двох основних частин. Кожна з частин включає в себе спеціальний матеріал, чутливий до інфрачервоного випромінювання. В даному випадку лінзи особливо не впливають на роботу датчика, так що ми бачимо дві ділянки чутливості всього модуля. Коли датчик знаходиться в стані спокою, обидва сенсори визначають однакову кількість випромінювання. Наприклад, це може бути випромінювання приміщення або навколишнього середовища на вулиці. Коли теплокровний об'єкт (людина або тварина), проходить повз, він перетинає зону чутливості першого сенсора, в результаті чого на модулі датчика генеруються два різних значення випромінювання. Коли людина залишає зону чутливості першого сенсора, значення вирівнюються. Саме зміни в показаннях двох датчиків реєструються і генерують імпульси HIGH або LOW на виході.

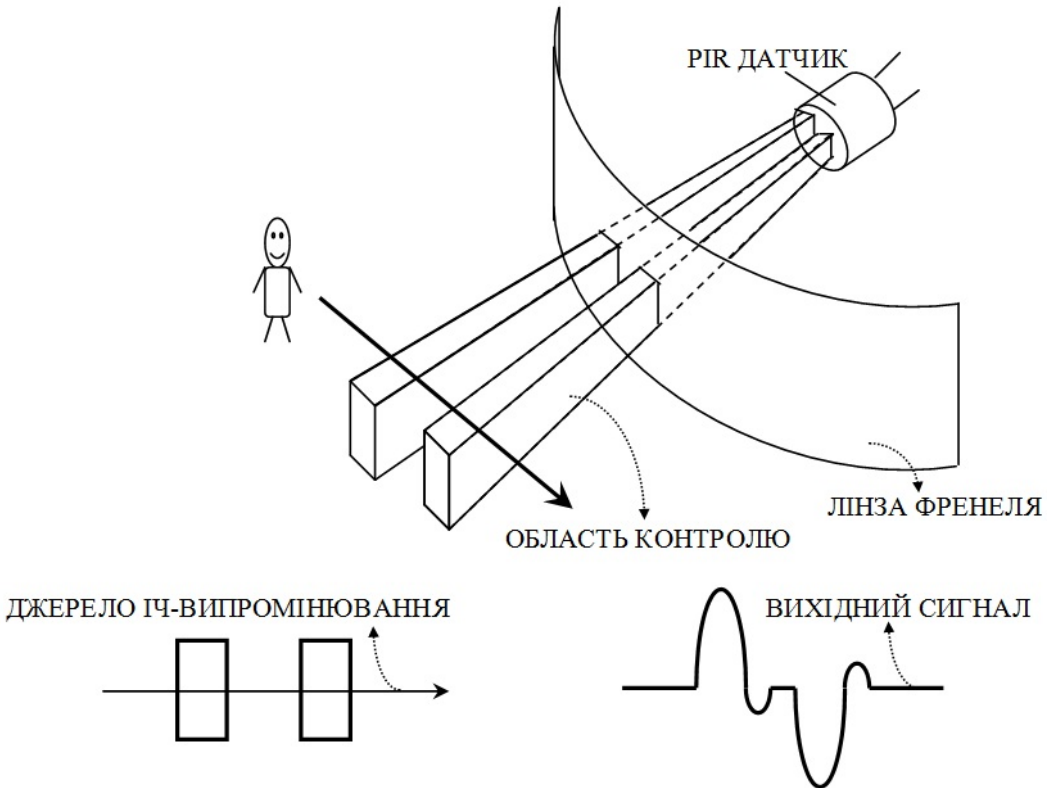


Рис. 2.4.

Чутливі елементи датчика встановлюються в металевий герметичний корпус, який захищає від зовнішніх шумів, перепадів температур і вологості. Прямокутник в центрі зроблений з матеріалу, який пропускає

інфрачервоне випромінювання (зазвичай це матеріал на основі силікону). За цією пластиною встановлюються два чутливих елемента (рис. 2.5).

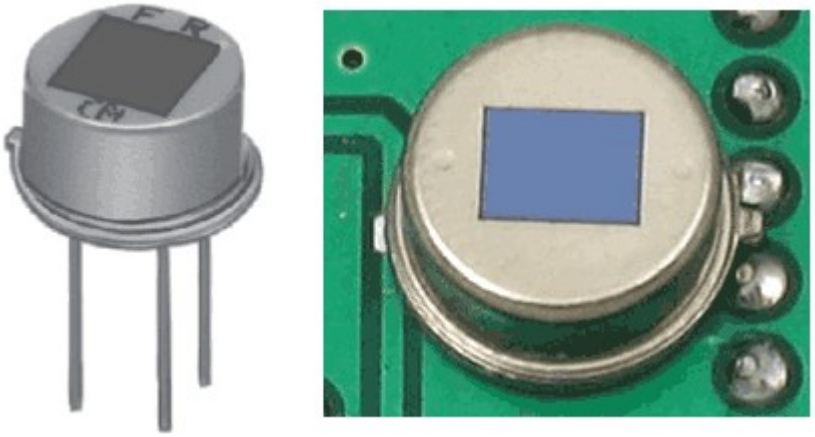


Рис. 2.5.

Інфрачервоні датчики руху практично однакові за своєю структурою. Основні відмінності – чутливість, яка залежить від якості чутливих елементів. При цьому значну роль відіграє оптика.

На рис. 2.6 наведено приклад лінзи з пластика. Це означає, що діапазон чутливості датчика представляє з себе два прямокутника. Але, як правило, потрібно забезпечити великі кути огляду. Для цього можна використовувати лінзи подібні до тих, які використовуються у фотоапаратах. При цьому лінза для датчика руху повинна бути маленька, тонка і виготовлятися з пластику, хоча він і додає шуми в вимірювання. Тому в більшості PIR датчиків використовуються лінзи Френеля (рис. 2.6 з Sensors Magazine):

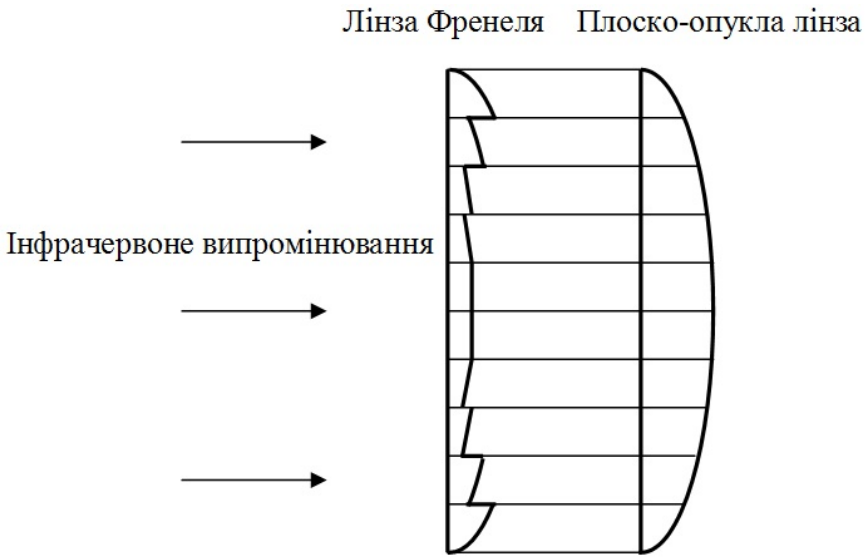


Рис. 2.6.

Лінзи Френеля концентрують випромінювання, значно розширюючи діапазон чутливості піродатчиків (рис. 2.7 з BHlens.com).

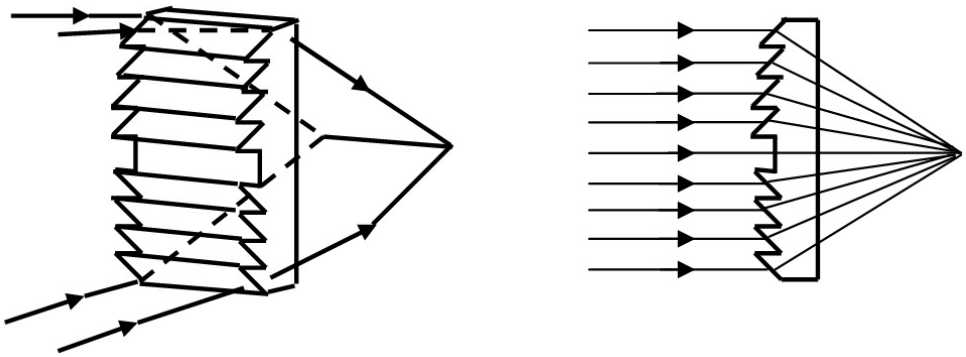


Рис. 2.7.

Рис. 2.8 (з Cypress appnote 2105) пояснює роботу лінзи Френеля.

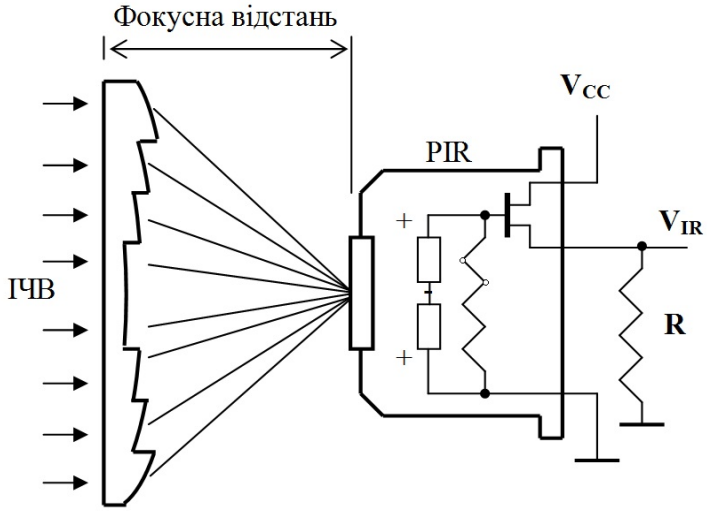


Рис. 2.8.

Таке рішення забезпечує значно більший діапазон чутливості. При цьому слід мати на увазі, що є два чутливих елемента і потрібні не стільки два великих прямокутника, скільки велика кількість маленьких зон чутливості. Для цього лінза розділяється на кілька секцій, кожна з яких представляє з себе окрему лінзу Френеля.

На рис. 2.9 можна побачити окремі секції – лінзи Френеля.



Рис. 2.9.

На рис. 2.10 видно, що фактура окремих лінз відрізняється.

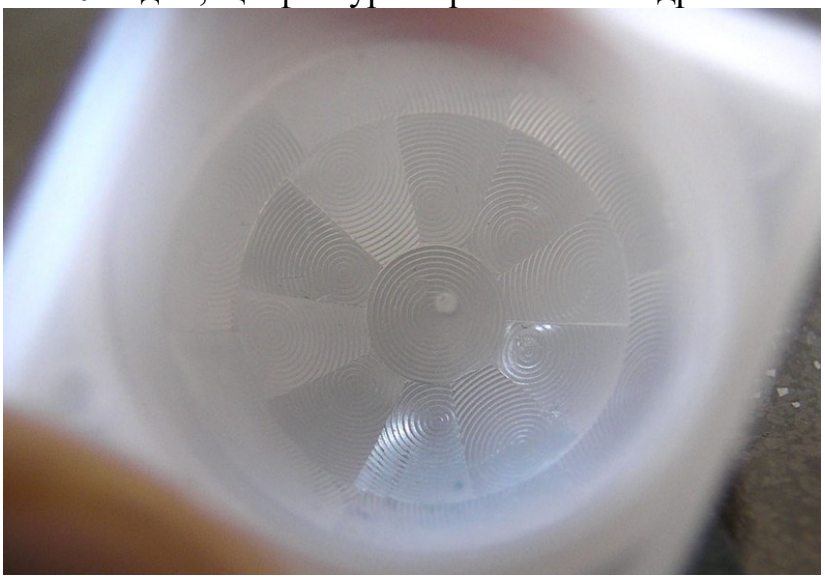


Рис. 2.10.

В результаті формується цілий набір чутливих ділянок, які взаємодіють між собою.

Для з'єднання з мікроконтролером, або безпосередньо з реле, у датчика HC-SR501 є три виводи – жавлення, загальний провідник та вихідний сигнал. Підключаєтья датчик до контролера Arduino за схемою, що представлена на рис. 2.11.

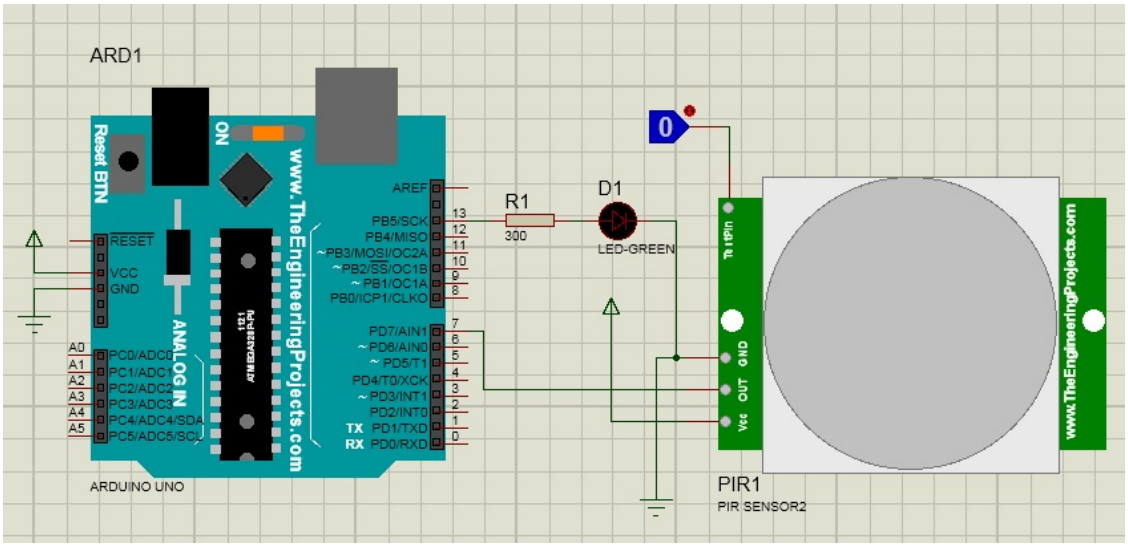


Рис. 2.11.

В схемі використовується модель контролера Arduino UNO та модель інфрачервоного датчика руху HC-SR501. Моделі для Proteus можна знайти на сайті за посиланням:

https://www.theengineeringprojects.com/2015/12/arduino-library-proteus-simulation.html

Завантаживши з сйту заархівовані файли моделей для Proteus, архіви потрібно розпакувати, і всі файли, які в них містилися, скопіювати в каталог LIBRARY, який, в свою чергу, розташований в каталозі, в який було втсновлено Protues. За замовчуванням шлях до цього каталогу наступний: C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\LIBRARY

Далі, запустивши середжовище моделювання Proteus, у вікні додавання компонента потрібно виділіти категорію Arduino і розмістити елементи на робочому полі. На рис. 2.12 показане вікно додавання компонента в схему.

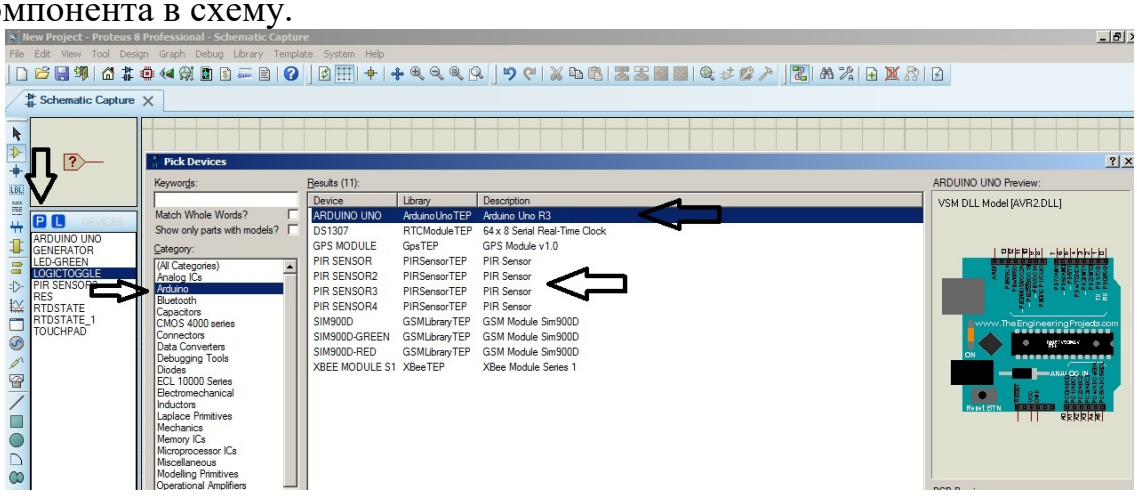


Рис. 2.12.

Також для моделювання роботи схеми потрібен світлодіод LED-GREEN, який буде загорятися при спрацюванні датчика, та струмообмежуючий резистор. Для моделювання виявлення руху модель датчика має додатковий вхід TestPin. Сигнал логічної одиниці на цьому вході призводить до спрацювання датчика. Для моделювання цього процесу в схемі використано елемент LOGICTOGGLE, який формує на виході лог.1 при натисканні на ньому при моделюванні роботи схеми в режимі реального часу. Також у вікні властивостей моделі інфрачервоного датчика руху потрібно вказати шлях до файлу PIRSensorTEP.hex, який також містився разом з файлом моделі в архіві при скачуванні. На рис. 2.13 показане вікно властивостей датчика руху. Шлях до файлу PIRSensorTEP.hex потрібно ввести в поле Program File.

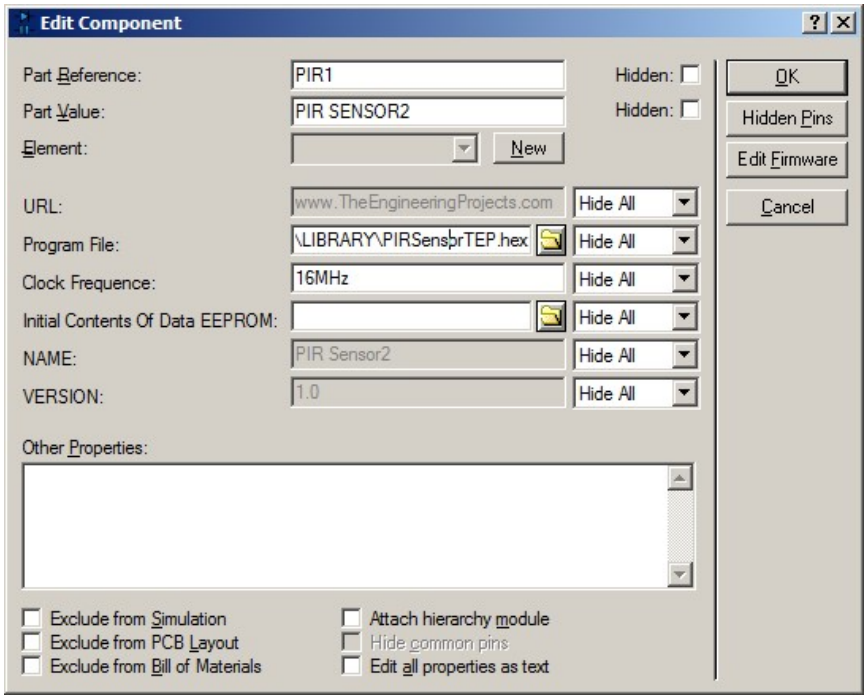


Рис. 2.13.

Скетч для перевірки роботи схеми представлено нижче.

```
const int movPin = 7; // номер виводу, до якого підключений датчик руху
const int ledPin = 13; // номер виводу, до якого підключений світлодіод

void setup() {
   pinMode(movPin, INPUT);        // налаштування виводу номер 7 на вхід
   pinMode(ledPin, OUTPUT);       // налаштування виводу номер 13 на вихід
}

void loop(){
   int value = digitalRead(movPin); // зчитування значення на вході 7 в змінну
   digitalWrite(ledPin, value); // запис вмісту змінної value на вихід 13
   delay(100); // часова затримка 100 мс
}
```

Для того, щоб після компіляції проекту було створено файл hex, в якому міститься «прошивка» мікроконтролера, у середовищі розробки програмного забезпечення Arduino IDE потрібно в меню Файл вибрати пункт налаштування. У вікні, що з’явиться, необхідно встановити позначку Компіляція, як показано на рис. 2.14.

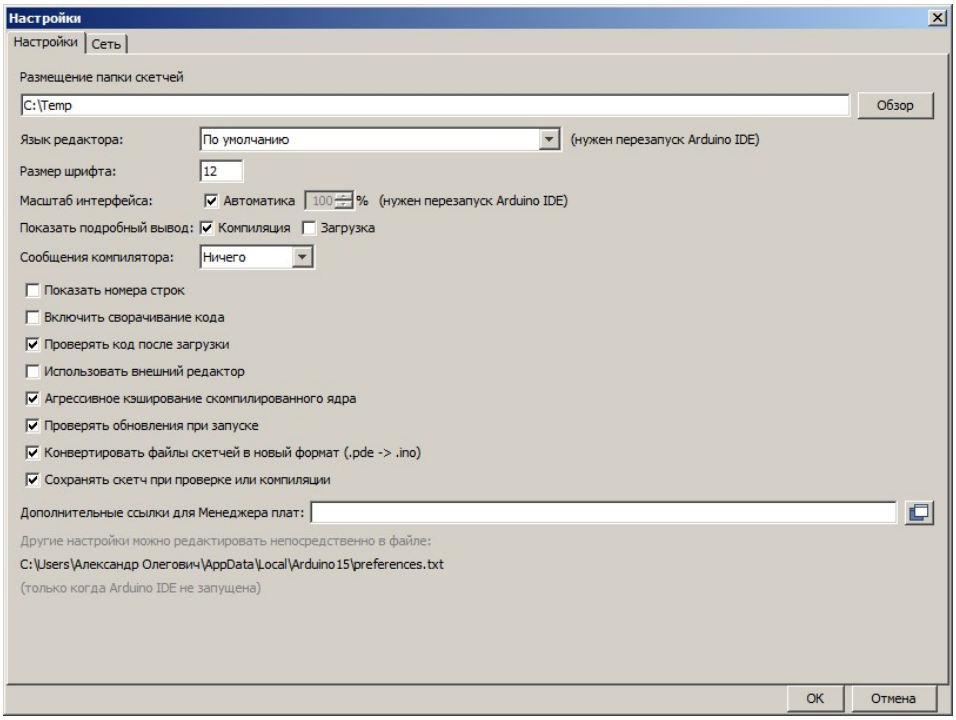


Рис. 2.14.

Тепер після компіляції проекту буде створено файл з розширенням hex. Шлях до цього файлу можна знайти у вікні повідомлень середовища Arduino (рис. 2.15). шлях до сього файлу потрібно скопіювати в буфер обміну та вставити у поле Program File у вікні властивостей моделі контролера Arduino в Proteus, як показано на рис. 2.16.

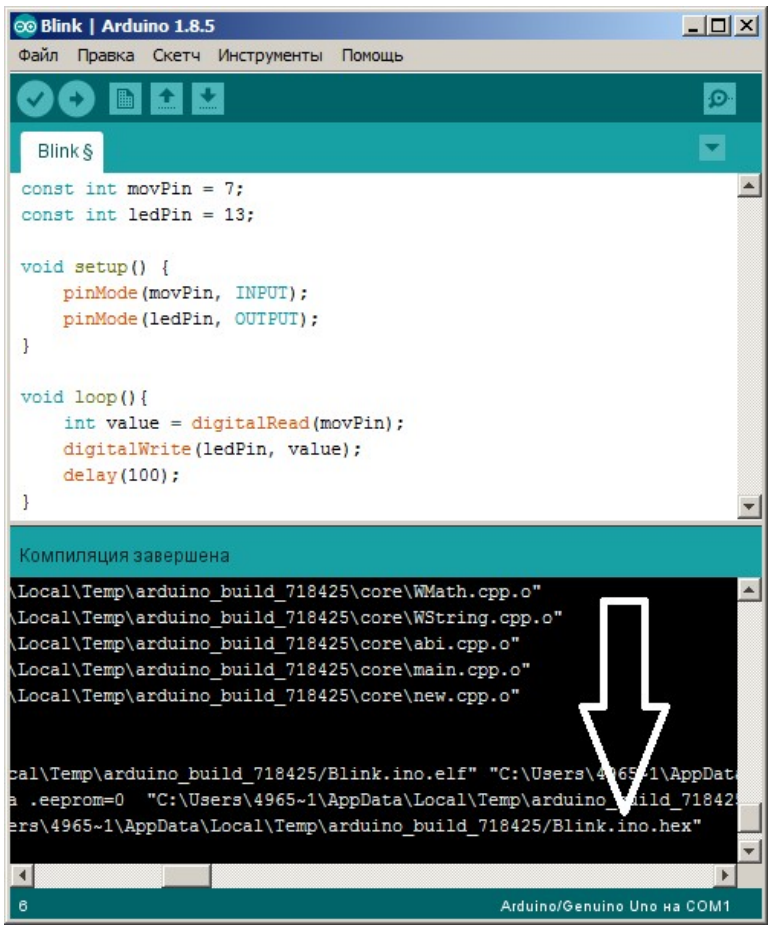


Рис. 2.15.

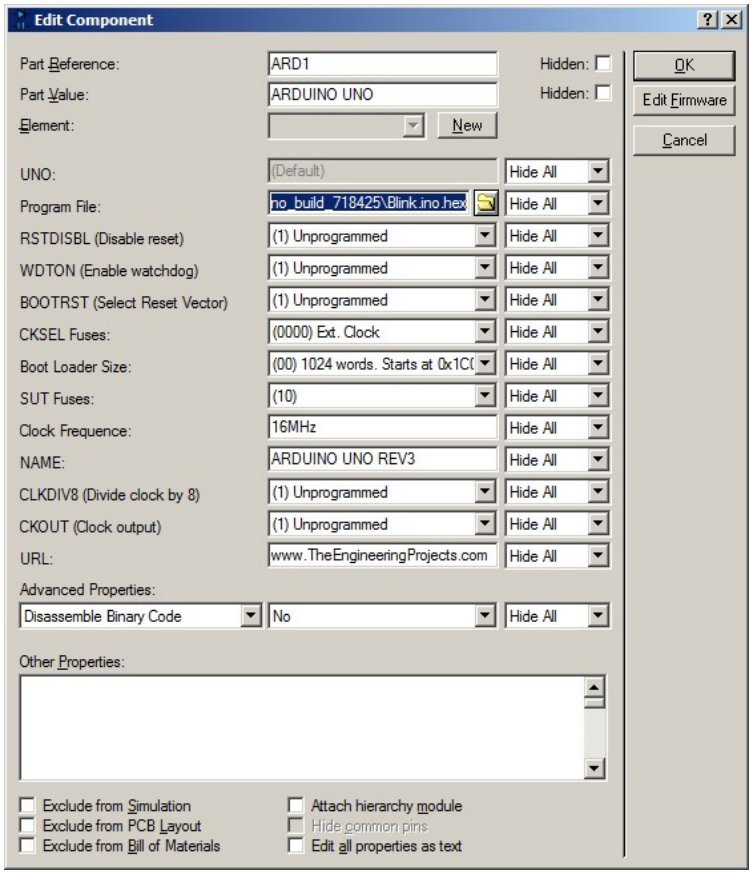


Рис. 2.16.

Запустивши процес моделювання можна перевірити роботу схеми та програми. Натиснувши один раз на елементі LOGICTOGGLE датчик спрацює, і на його виході з'явиться сигнал високого рівня. Цей сигнал буде зчитаний контролером та виведений на вивід, до якого підключений світлодіод. Тому світлодіод змінить свій стан і загориться. Через деякий проміжок часу світлодіод згасне. Рис. 2.17 демонструє схему в режимі моделювання.

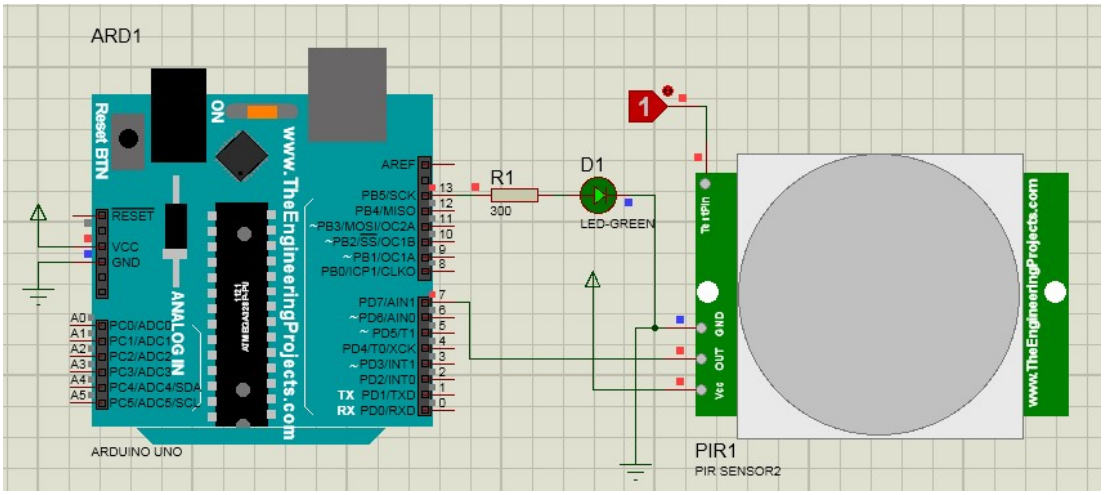


Рис. 2.17.

Для підключення датчика до реального пристрою з'єднати датчик та контролер Arduino потрібно так, як показано на рис. 2.18.

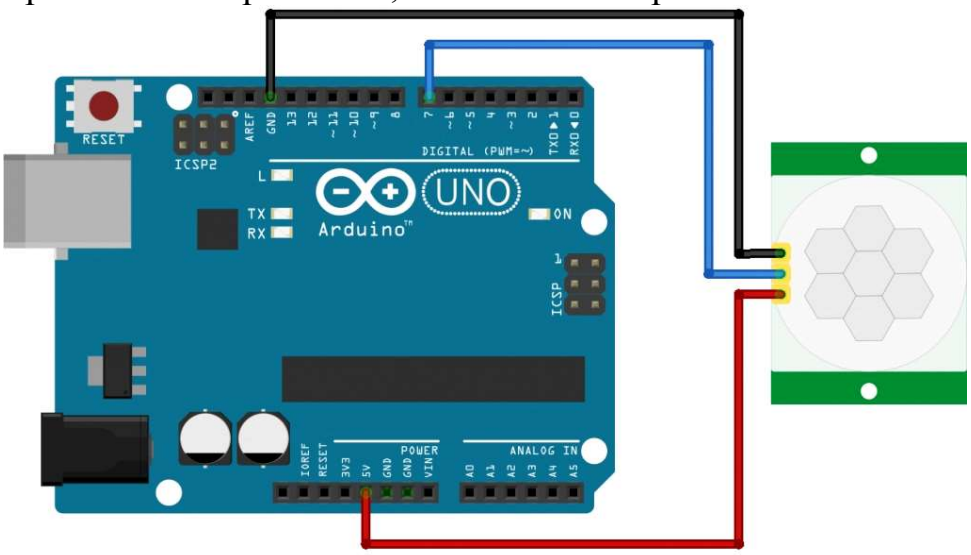


Рис. 2.18.

На платі Arduino UNO до виводу номер 13 підключений світлодіод, який розташований на самій платі (позначений літерою L). Тому немає необхідності підключати зовнішній світлодіод.

Цей проект можна розвинути, зробивши систему автоматичного вмикання світла при виявленні руху. На рис. 2.19 представлено схему керування освітленням. Оскільки лампа розжарювання живиться від 220 В, тому для керування нею використовується модуль реле. При виявленні

руху за допомогою інфрачервоного датчика контролер Arduino увімкне реле, що призведе до вмикання лампи розжарювання. Через деякий час, при відсутності руху в полі зору датчика, на його виході з'явиться сигнал низького рівня, і контролер Arduino вимкне реле, і, відповідно, освітлення.

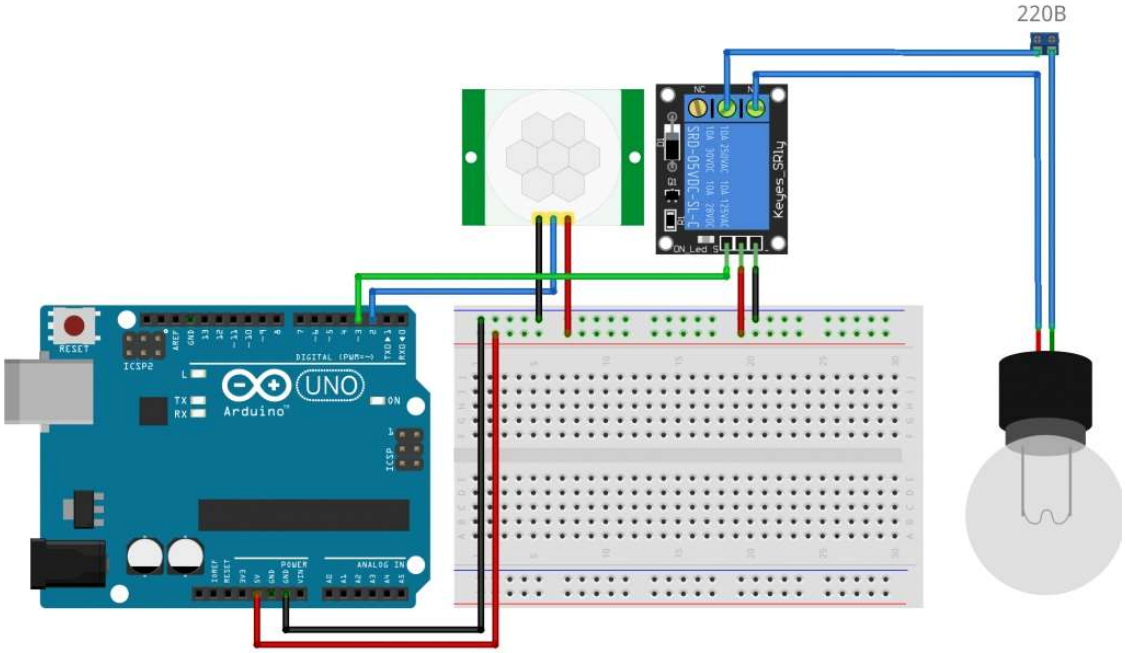


Рис. 2.19.

Скетч для системи автоматичного керування освітленням при виявлені руху представлено нижче.

```
const int sensorPin = 2; // номер виводу, до якого підключено датчик руху
const int relayPin = 3;// номер виводу, до якого підключено реле

void setup() {
   pinMode(sensorPin, INPUT); // налаштування виводу номер  2 на вхід
   pinMode(relayPin, OUTPUT); // налаштування виводу номер 3 на вихід
}

void loop(){
   int value = digitalRead(sensorPin); // зчитування дискретного сигналу з датчика
   if (value == HIGH) // якщо датчик формує високий рівень (є рух)
      digitalWrite(relayPin, HIGH);   // увімкнути реле
   else                               // інакше
      digitalWrite(relayPin, LOW);   // вимкнути реле
}
```

Для перевірки роботи програми в Proteus потрібно зібрати схему, представлену на рис. 2.10.

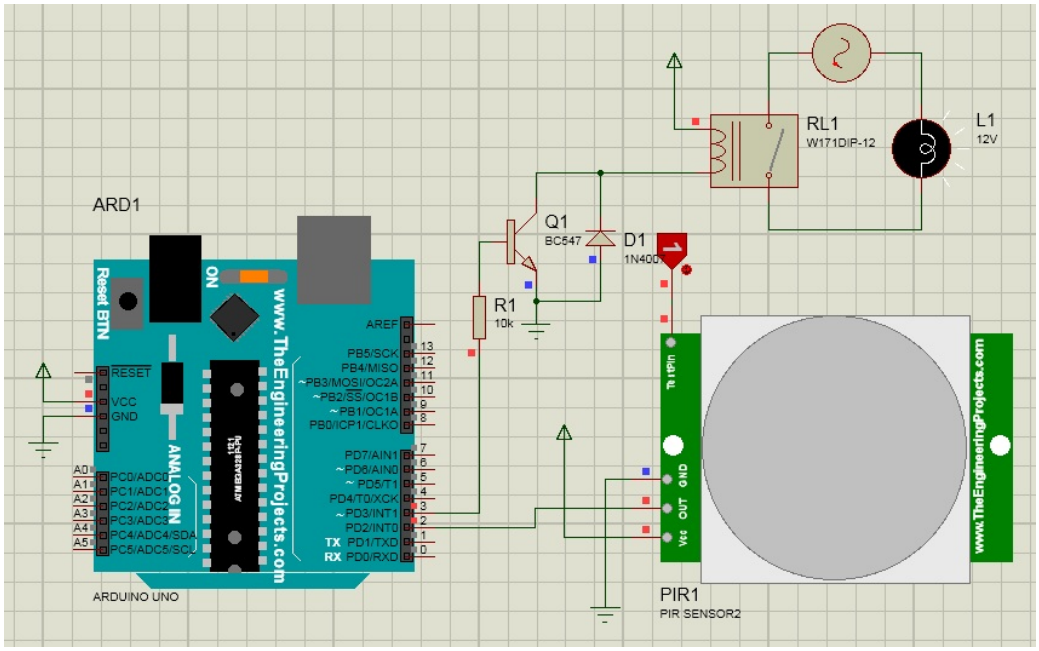


Рис. 2.20.

Запустивши процес моделювання роботи схеми і сформувавши високий рівень сигналу на вході TestPin датчика руху можна побачити, як лампа загориться. Моделювання дозволяє зробити висновки про коректність роботи схеми та програми.

2.3. Ультразвуковий датчик відстані HC-SR04

Ультразвуковий датчик HC-SR04, представлений на рис. 2.21, може служити датчиком відстані для робота, завдяки якому він зможе визначати дистанцію до об'єктів, об'їжджати перешкоди, або будувати карту приміщення. Його можна також використовувати в якості датчика для сигналізації, що спрацьовує при наближенні об'єктів.

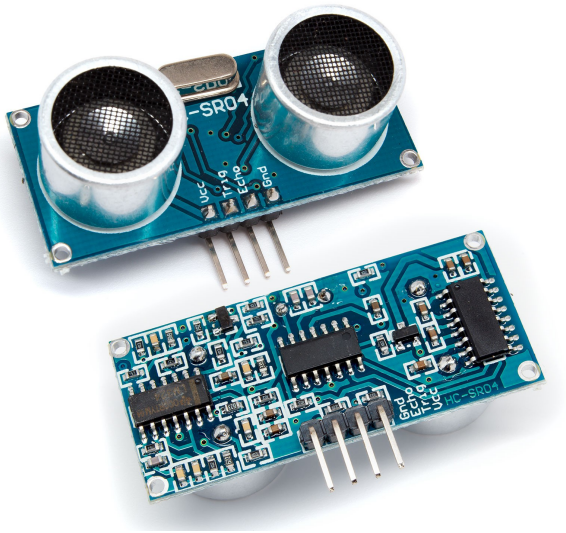


Рис. 2.21.

Ультразвуковий датчик визначає відстань до об'єктів таким самим чином, як це роблять дельфіни або кажани. Він генерує звукові імпульси

на частоті 40 кГц і приймає відлуння, як показано на рис. 2.22. За часом поширення звукової хвилі туди і назад можна однозначно визначити відстань до об'єкта.

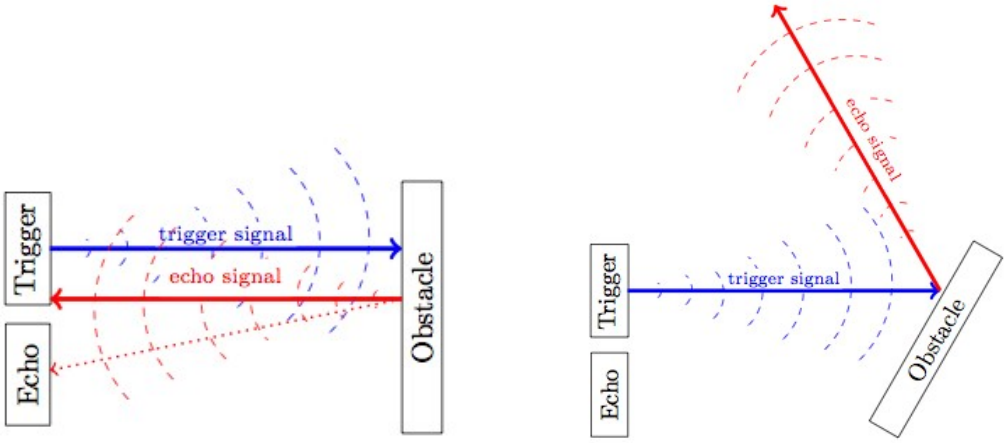


Рис. 2.22.

На відміну від інфрачервоних датчиків відстані, на покази ультразвукового далекоміра не впливають світлове випромінювання від сонця або колір об'єкта. Але можуть виникнути труднощі з визначенням відстані до пухнастих або дуже тонких предметів.

Опис виводів:

– Vcc – контакт живлення.

– Trig – цифровий вхід. Для запуску вимірювання необхідно подати на цей вхід логічну одиницю на 10 мкс. Наступне вимірювання рекомендується виконувати не раніше ніж через 50 мс.

– Echo – цифровий вихід. Після завершення вимірювання, на цей вихід буде подана логічна одиниця на час, пропорційний відстані до об’єкта.

– GND – від’ємний контакт живлення.

Роботу з датчиком можна умовно розділити на 4 етапи:

1. Подача імпульсу тривалістю 10 мкс на вхід Trig (рис. 2.23).

2. Всередені датчика вхідний імпульс перетворюється в 8 імпульсів частотою 40 кГц і відправляється вперед через випромінювач.

3. Відправлені імпульси відбиваються від перешкоди і приймаються приймачем. Отримується вихідний сигнал на виводі Echo.

4. Безпосередньо на стороні мікроконтролера отриманий сигнал перетворюється у відстань по формулі:

тривалість імпульсу(мкс)/58 = відстань (см),

тривалість імпульсу(мкс)/148 = відстань(дюйм).

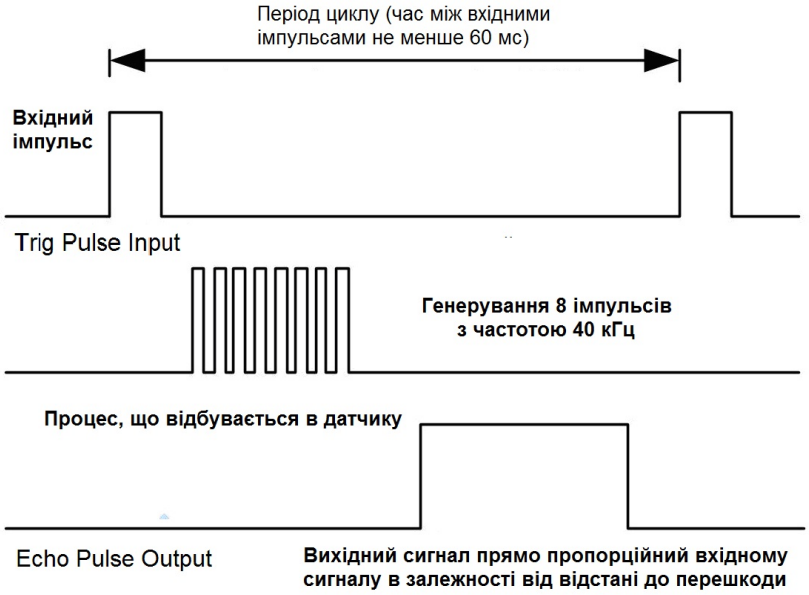


Рис. 2.23.

Характеристики ультразвукового датчика відстані:
– Напруга живлення: 5 В;
– Споживання струму в режимі тиші: 2 мА;
– Споживання струму при роботі: 15 мА;
– Діапазон відстаней: 2-400 см;
– Ефективний кут спостереження: 15°;
– Робочий кут спостереження (рис. 2.24): 30°.

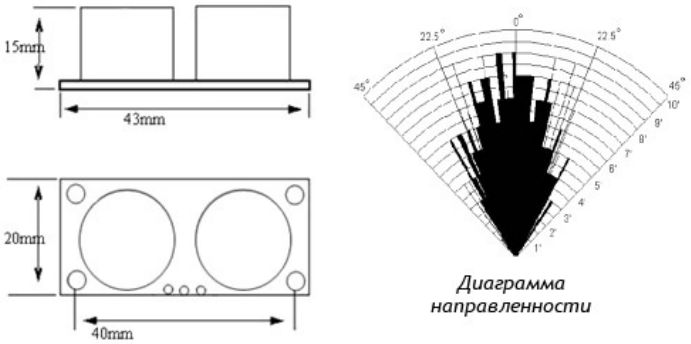


Рис. 2.24.

В даному прикладі ультразвуковий датчик HC-SR04 визначає відстань і виводить отримані значення у вікно серійного монітора в середовищі Arduino IDE. Якщо відстань до перешкоди менше 50 см, загоряється світлодіод. Основна мета даного прикладу – допомогти зрозуміти принцип дії датчика для подальшого використання в більш складних проектах. На рис. 2.25 представлена принципова схема системи вимірювання відстані до перешкоди.

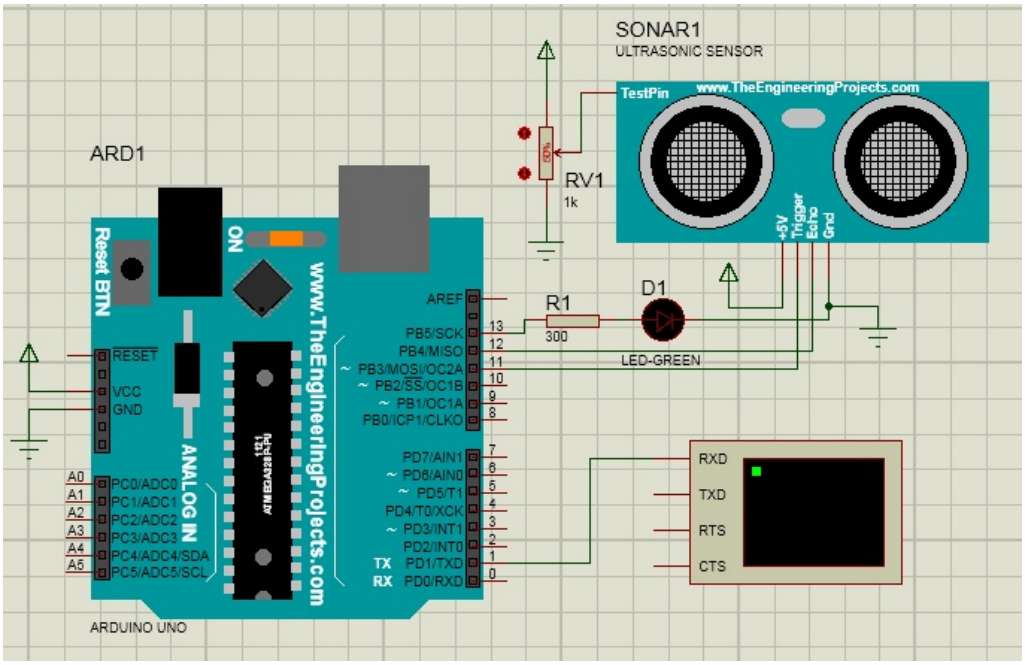


Рис. 2.25.

Текст програми представлено нижче.

```
int trigPin = 11;    // номер виводу, до якого підключено вхід Trig датчика
int echoPin = 12;  // номер виводу, до якого підключено вихід датчика
int ledPin = 13;    // номер виводу, до якого підключено світлодіод
// оголошення додаткових змінних для зберігання даних про виміряну відстань
long duration, cm, inches;

// у цій функції відбувається налаштування портів та швидкості передачі
// даних по послідовному інтерфейсу – ініціалізація контролера

void setup() {
Serial.begin (9600); // налаштування швидкості передачі даних 9600 біт/с
pinMode(trigPin, OUTPUT); // вивід номер 11 налаштовується на вихід
pinMode(echoPin, INPUT); // вивід номер 12 налаштовується на вхід
pinMode(ledPin, OUTPUT); // вивід номер 13 налаштовується на вихід
}

// в цій функції реалізовано основний алгоритм роботи програми
void loop()
{
// Датчик спрацьовує і генерує імпульси шириною 10 мкс або більше
// Генерується короткий LOW імпульс, щоб забезпечити імпульс HIGH:
digitalWrite(trigPin, LOW); // встановлення низького рівня на виході 11
delayMicroseconds(5); // формування часової затримки тривалістю 5 мкс
digitalWrite(trigPin, HIGH); // встановлення високого рівня на виході 11
delayMicroseconds(10); // формування часової затримки тривалістю 10 мкс
digitalWrite(trigPin, LOW); // встановлення низького рівня на виході 11
// Зчитуємо дані з ультразвукового датчика: значення HIGH, яке
// залежить від тривалості (в мікросекундах) між відправленням
// акустичної хвилі і її прийманням на локаторі відлуння.
pinMode(echoPin, INPUT); // вивід номер 12 налаштовується на вхід
```

```
duration = pulseIn(echoPin, HIGH); // зчитування тривалості сигналу на вході 12
cm = (duration/2) / 29.1; // перетворення значення виміряного часу у сантиметри
inches = (duration/2) / 74; // перетворення значення виміряного часу у дюйми
if(cm<50) // якщо відстань менше 50 см
{
 digitalWrite(ledPin, HIGH);  // встановити високий рівень на виході 13
}
else                              // інакше
{
 digitalWrite(ledPin, LOW);  // встановити низький рівень на виході 13
}
Serial.print(inches);    // передавання по послідовному порту значення дюймів
Serial.print("in, ");    // передавання текстового рядка “in”
Serial.print(cm);        // передавання по послідовному порту значення сантиметрів
Serial.print("cm");      // передавання текстового рядка “cm”
Serial.println();        // передавання символу переходу на новий рядок
delay(250);              // часова затримка 250 мс
}
```

Після компіляції програми можна промоделювати роботу схеми. На рис. 2.26 представлена схема в режимі моделювання. За допомогою віртуального термінала можна спостерігати значення відстані, що вимірюється датчиком. Модель датчика має додатковий вхід TestPin. Змінюючи напругу, яка подається на цей вхід, можна моделювати процес вимірювання датчиком різних відстаней. В даному випадку найпростіший спосіб змінювати напругу на цьому вході – це використовувати змінний опір. Якщо відстань стане менше 50 см, то загориться світлодіод.

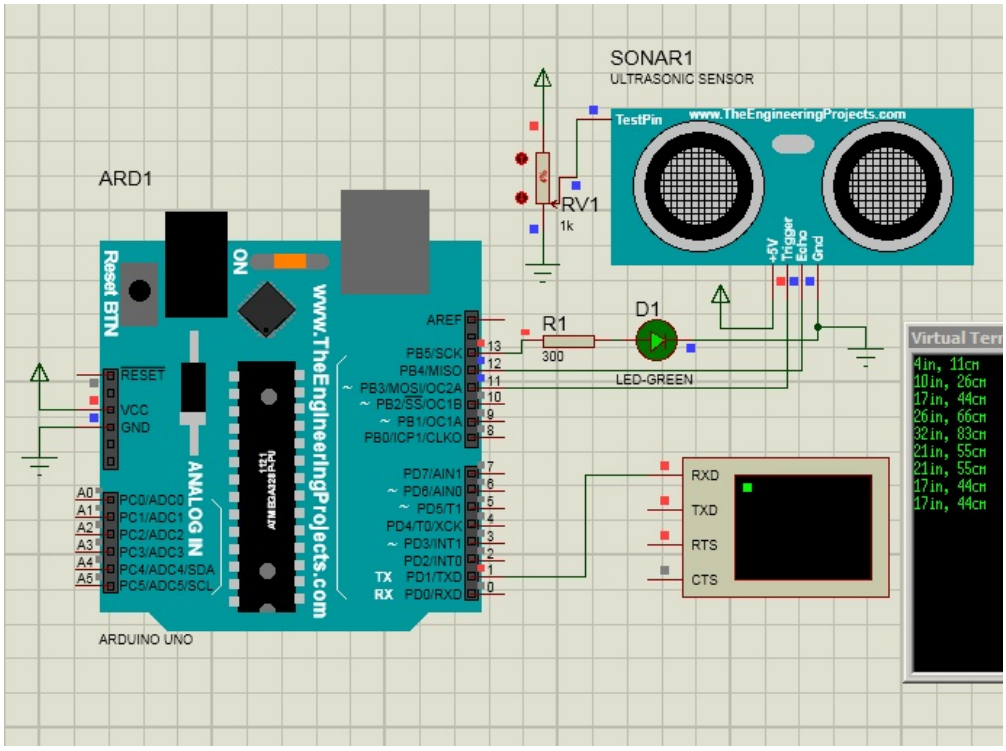


Рис. 2.26.

Зібравши схему, що представлена на рис. 2.27, можна перевірити роботу програми на реальному пристрої. Підключивши контролер Arduino до комп'ютера і завантаживши програму, у вікні термінала, вбудованого в середовище розробки Arduino, можна побачити результати вимірювання. Для того, щоб відкрити вікно термінала потрібно в меню Інструменти вибрати пункт Монітор порта. Вказавши потрібний порт та швидкість передачі даних 9600 біт/с, на екрані можна спостерігати в текстовому вигляді інформацію, отриману з контролера.

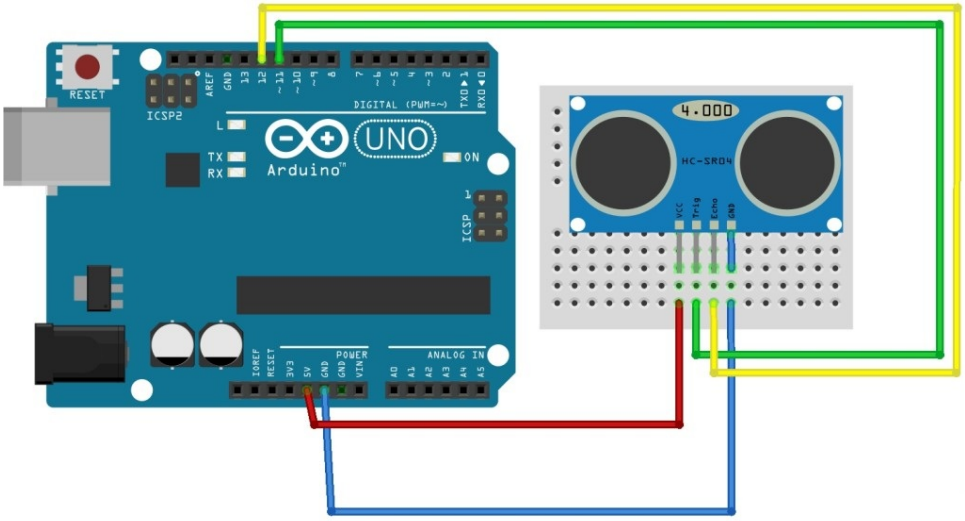


Рис. 2.27.

В програмі для вимірювання тривалості імпульсу було використано функцію pulseIn(). Ця функція зчитує довжину сигналу на заданому порту (HIGH або LOW). Наприклад, якщо виконується зчитування HIGH функцією pulseIn(), функція чекає поки на заданому порту не з'явиться високий рівень сигналу HIGH. Коли HIGH отримано, включається таймер, який буде зупинений коли на порту входу/виходу буде низький рівень сигналу LOW. Функція pulseIn() повертає довжину сигналу в мікросекундах. Функція повертає 0, якщо протягом заданого часу (таймаута) сигнал на відповідному виводі ні був зафіксований.

При використанні функції можливі деякі похибки в вимірювання довгих сигналів. Функція може вимірювати сигнали довжиною від 10 мікросекунд до 3 хвилин. Функція має наступний синтаксис:

```
pulseIn(pin, value);
pulseIn(pin, value, timeout);
```

Параметри, що передаються в функцію: pin – номер виводу, на якому буде очікуватися сигнал; value – тип очікуваного сигналу – HIGH або LOW; timeout (опціональний параметр) – час очікування сигналу (таймаут) в мікросекундах (за замовчуванням одна секунда).

Значення, що повертається – тривалість сигналу в мікросекундах або 0, якщо сигнал не був отриманий до закінчення часу очікування.

2.4. Оптичний інфрачервоний датчик FC-51

Оптичний інфрачервоний датчик можна застосувати для найрізноманітніших цілей:

– визначення наявності поверхні в роботах-пилососах;

– використання в якості датчика в енкодерах або в якості кінцевих датчиків;

– визначення обмежувальної лінії в змаганнях роботів;

– відстеження лінії в пошукових системах;

– використання модуля в якості найпростішого сенсора перешкод.

Для підсвічування можна використовувати будь-який потужний інфрачервоний світлодіод, проте зручніше для налагодження і контролю, якщо він буде випромінювати у видимому діапазоні. Фотоприймачі мають зазвичай досить широку смугу чутливості, тому світити можна різними кольорами – від синього до червоного.

Одним з найбільш поширених і часто використовуваних фотоелементів є фототранзистор. Фототранзистор дешевий, у нього дуже хороша чутливість і малий час спрацьовування. Однак у нього є один істотний недолік – нелінійність характеристики. Він занадто швидко переходить в режим насичення, не дозволяючи відстежити зміну освітленості.

На рис. 2.28 представлений зовнішній вигляд оптичного інфрачервоного датчика перешкод, а на рис. 2.29 схема, що пояснює принцип його роботи.

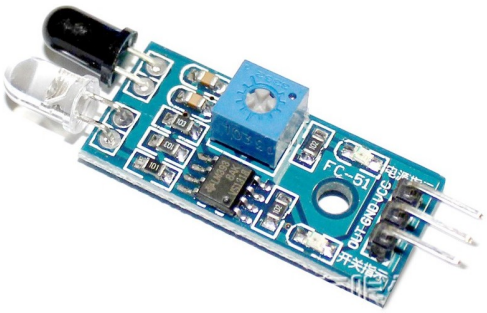


Рис. 2.28.

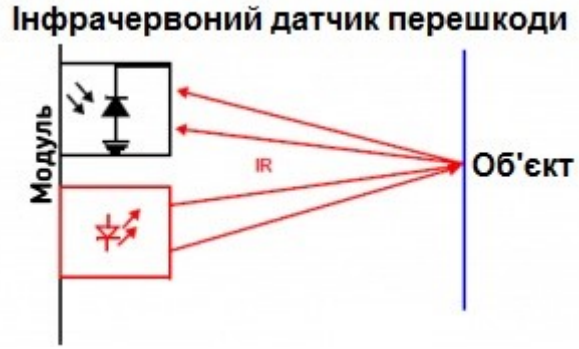


Рис. 2.29.

Принципова схема датчика представлена на рис. 2.30. Принцип роботи датчика досить простий. Компаратор порівнює опорну напругу і

напругу з колектора фототранзистора. При збігу напруг (або перевищенні еталонного) на виході компаратора логічний сигнал змінюється на протилежний.

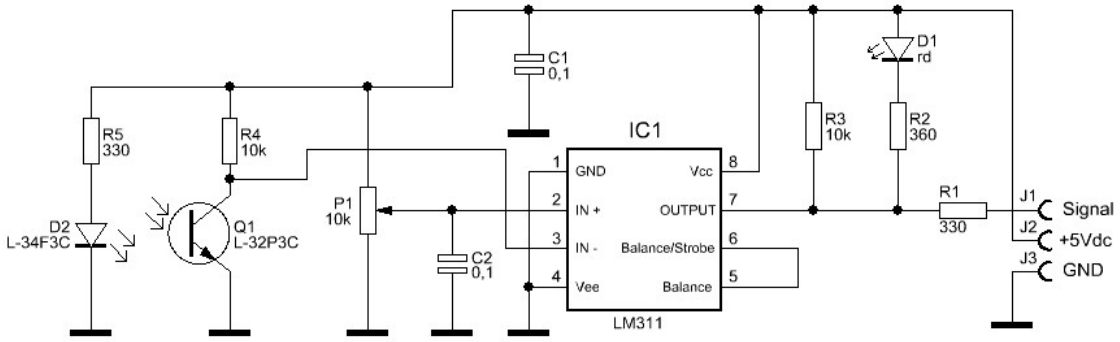


Рис. 2.30.

Модуль має можливість регулювання порогу спрацьовування. Для цього в схему включений змінний резистор, за допомогою якого можливе регулювання опорної напруги на компараторі.

Також в схемі присутній сигнальний світлодіод, який відображає поточний стан виходу компаратора:

– світлодіод горить – чорна (темна) поверхня, або її повна відсутність (на виході нуль);

– світлодіод не горить – біла (світла) поверхню або перешкода (на виході рівень логічної одиниці).

Діапазон дальності визначення перешкоди (лінії) досить широкий – від 1 мм до 15 мм і більше. Для перевірки працездатності модуля досить підключити живлення та відрегулювати поріг спрацьовування компаратора змінним резистором. Для початку необхідно підключити 5 В або 3,3 В до входу VCC і під’єднати загальний провід (земля). Загориться червоний світлодіод, розташований на датчику. Далі, перед датчиком потрібно поставити перешкоду на необхідній відстані для спрацьовування датчика і відрегулювати потенціометром необхідний рівень. При спрацьовуванні датчика на модулі загориться зелений світлодіод.

На рис. 2.31 представлена схема в Proteus для моделювання роботи датчика.

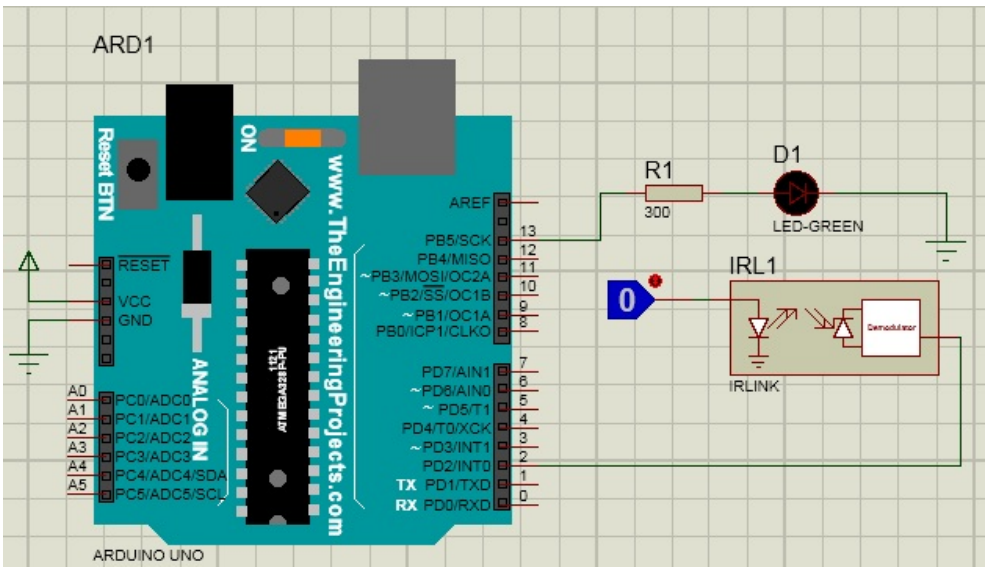


Рис. 2.31.

Для моделювання інфрачервоного датчика в Proteus було використано елемент IRLINK, вікно налаштувань якого представлено на рис. 2.32.

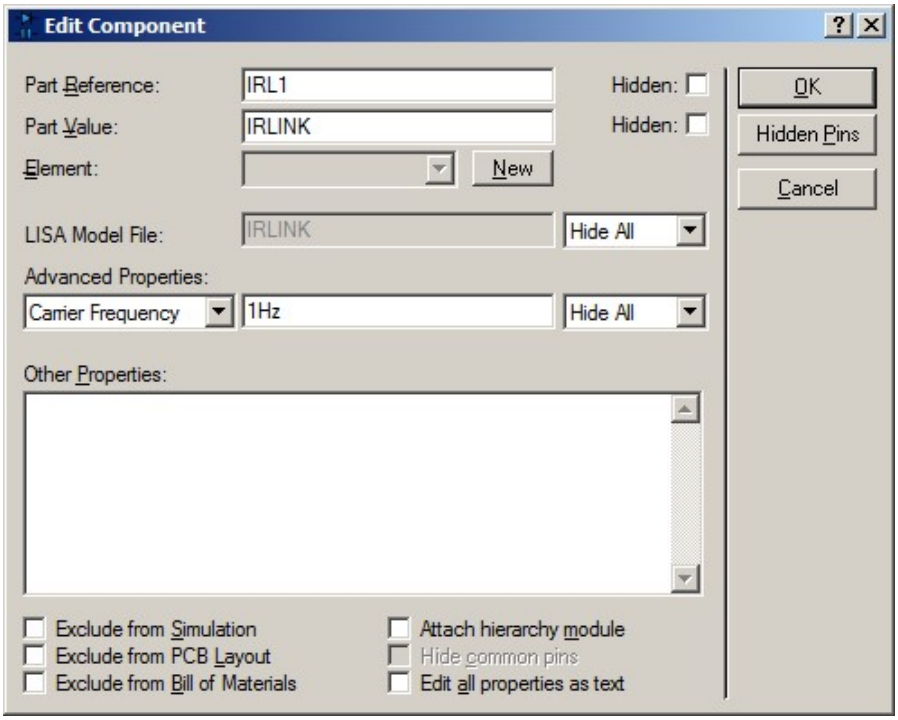


Рис. 2.32.

Програма, що представлена нижче, формує сигнал високого рівня на виході, до якого підключений світлодіод, якщо на виході датчика з'являється сигнал високого рівня, тобто виявлена перешкода.

```
int sensorPin = 2;   // номер виводу, до якого підключений вихід датчика
int ledPin = 13;     // номер виводу, до якого підключений світлодіод
// в цій функції виконується налаштування портів введення/виведення
контролера
void setup() {
```

```
pinMode(sensorPin, INPUT); // налаштування виводу номер 2 на вхід
pinMode(ledPin, OUTPUT); // налаштування виводу номер 13 на вихід
}

void loop()
{
int value; // оголошення змінної для збереження результату вимірювання
value = digitalRead(sensorPin); // зчитування сигналу зі входу номер 2
digitalWrite(ledPin, value); // виведення результатів вимірювання на світлодіод
delay(250); // формування часової затримки тривалістю 250 мс
}
```

Ще один приклад програми представлено нижче. Контролер Arduino виконує зчитування даних з датчика та виводить їх у послідовний порт в момент перетину кордону.

```
// номер виводу контролера, до якого підключений датчик
#define sensorPin 2
// запит стану виводу Далеко/Близько
// Далеко
#define AWAY 1
// Близько
#define CLOSE 0
void setup() {
  Serial.begin(9600); // налаштування швидкості передачі даних 9600 біт/с
  pinMode(sensorPin, INPUT); // налаштування виводу номер 2 на вхід
}

void loop() {
// зчитування даних зі входу, до якого підключений датчик
  if (digitalRead(sensorPin) == AWAY) // якщо перешкоди немає
  {
    Serial.println("Away"); // передавання рядка "Away" в комп'ютер
    while ( digitalRead(sensorPin) == AWAY){} // очікування доки не наблизиться
  }
else
{
   Serial.println("Close"); // передавання рядка "Close" в комп'ютер
   while ( digitalRead(sensorPin) == CLOSE){} // очікування доки не віддалиться
 }
}
```

На рис. 2.33 представлена схема моделі в Proteus в режимі моделювання. У вікні терміналу можна бачити рядки, що були відправлені контролером Arduino при спрацюванні датчика (при наближені та віддаленні від перешкоди).

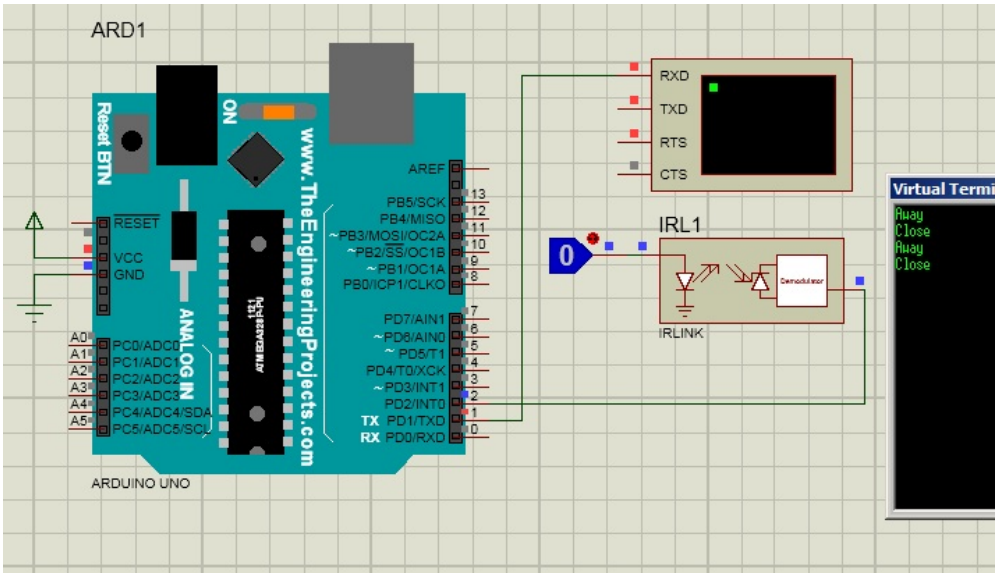


Рис. 2.33.

Інфрачервоний датчик перешкод застосовується часто в робототехніці, і його використання буде продемонстровано при реалізації системи керування рухом робота по заданій траєкторії.

2.5. Датчик кольору TCS3200

Датчик кольору TCS230, представлений на рис. 2.34, використовується для визначення кольору будь-якого об'єкта на відстані до 10 мм. Датчик кольору можна використовувати для розпізнавання кольорів оточуючих предметів в роботизованих системах.



Рис. 2.34.

Для підсвічування об'єктів на платі розташовані 4 білих світлодіода. Сам датчик TCS3200 складається з масиву фотоелементів з фільтром на червоний, зелений, синій і «безбарвний» кольори, як показано на рис. 2.35. Це дозволяє визначати кольори всього видимого спектру. Елементи розподілені рівномірно, що захищає від похибок, пов'язаних з їх розташуванням.

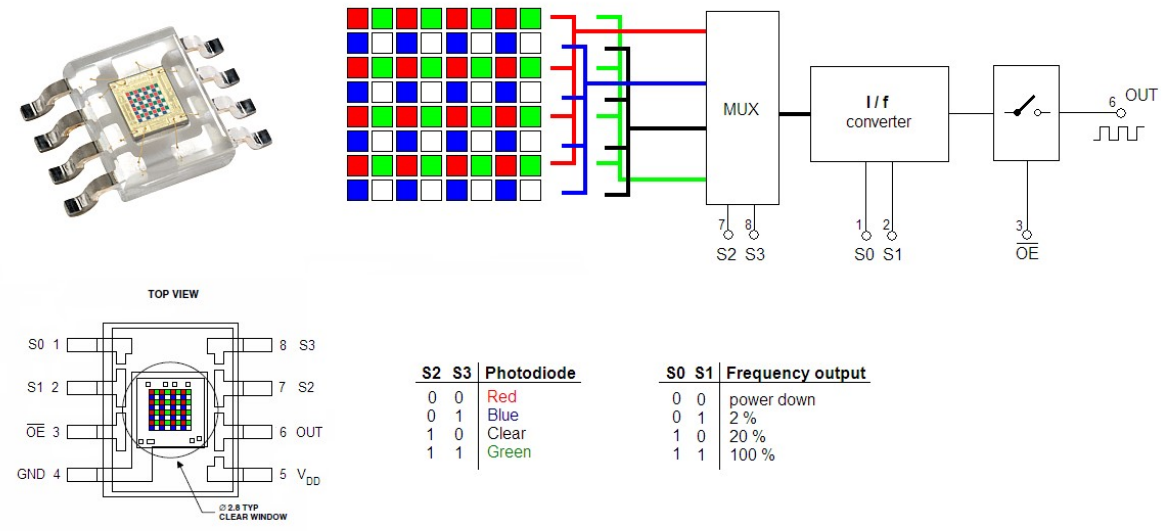


Рис. 2.35.

Датчик переводить інтенсивність світла, що проходить через фільтр заданого кольору, в цифровий сигнал з пропорційної частотою.

Плата сенсора підключається через 10 контактів, розташованих по краях.

Ліва колодка має контакти:

GND – загальний провід;

OE – контакт вмикання;

S1, S0 – вихідна частота з налаштуванням масштабування.

Права колодка має контакти:

S3, S2 – вхідний сигнал налаштування масштабування;

OUT – вихідна частота;

VCC – напруга живлення.

Характеристики датчика кольору:

– висока дозволяюча здатність конверсії світла в частоту;

– програмовані кольори і налаштування вихідної частоти;

– робота напряму з мікроконтролером;

– напруга живлення датчика: 2,7 – 5,5 В;

– функція автоматичного вимикання живлення;

– мала похибка вихідної частоти: 0,2%.

Табл. 2.1 показує значення вхідних сигналів для налаштування вихідної частоти.

Таблиця 2.1.

S0	S1	Масштаб вихідної частоти (f_0)
L	L	Режим енергозбереження
L	H	2%
H	L	20%
H	H	100%

В табл. 2.2 показані значення вхідних сигналів для налаштування датчика.

Таблиця 2.2.

S2	S3	Тип фотодіода
L	L	Червоний (Red)
L	H	Синій (Blue)
H	L	Без світлофільтра (Clear)
H	H	Зелений (Green)

Повномасштабна вихідна частота цього датчика кольору може бути розширена за допомогою одного з трьох попередньо встановлених значень, які подаються через два вхідних контакти S0 та S1 (рис. 2.36). TCS230 поставляються з цифровим входом і виходом, які дозволяють безпосередньо підключатися до мікроконтролера.



Рис. 2.36.

Діапазон частот на виході від 2 кГц до 500 кГц.

В якості прикладу використання датчика кольору пропонується схема, представлена на рис. 2.37. Контролер Arduino керує кольором RGB-світлодіода і встановлює такий колір, який було визначено за допомогою датчика кольору.

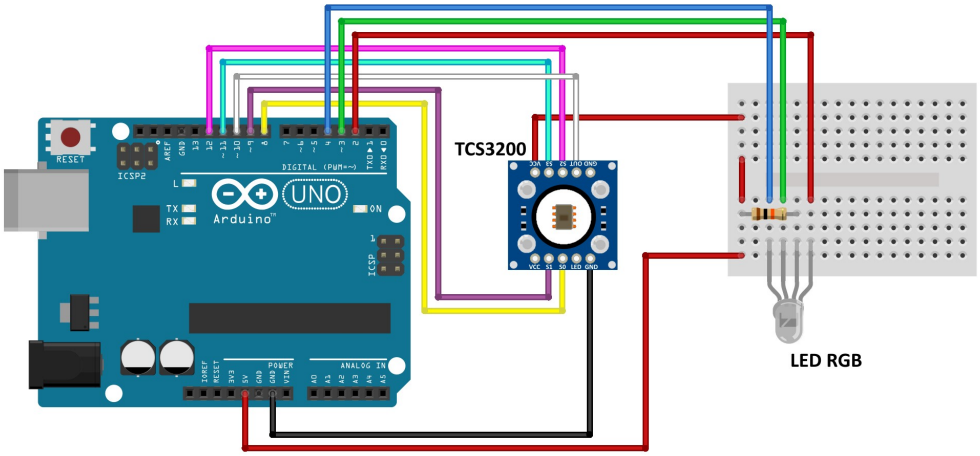


Рис. 2.37.

Текст програми представлено нижче.

```
const int s0 = 8; // номер виводу, до якого відключений вихід s0 датчика
const int s1 = 9; // номер виводу, до якого відключений вихід s1 датчика
const int s2 = 12; // номер виводу, до якого відключений вихід s2 датчика
const int s3 = 11; // номер виводу, до якого відключений вихід s3 датчика
const int out = 10; // номер виводу, до якого відключений вихід out датчика
int pinoledverm = 2; // номер виводу, до якого підключений червоний світлодіод
int pinoledverd = 3; // номер виводу, до якого підключений зелений світлодіод
int pinoledazul = 4; // номер виводу, до якого підключений синій світлодіод
// оголошення змінних, в які будуть записані результати вимірювань
int red = 0; // змінна для збереження червоної складової кольору
int green = 0; // змінна для збереження зеленої складової кольору
int blue = 0; // змінна для збереження синьої складової кольору

void setup()
{
 pinMode(s0, OUTPUT); // налаштування виводу 8 на вихід
 pinMode(s1, OUTPUT); // налаштування виводу 9 на вихід
 pinMode(s2, OUTPUT); // налаштування виводу 12 на вихід
 pinMode(s3, OUTPUT); // налаштування виводу 11 на вихід
 pinMode(out, INPUT); // // налаштування виводу 10 на вхід
 pinMode(pinoledverm, OUTPUT); // налаштування виводу 2 на вихід
 pinMode(pinoledverd, OUTPUT); // налаштування виводу 3 на вихід
 pinMode(pinoledazul, OUTPUT); // налаштування виводу 4 на вихід
 Serial.begin(9600); // налаштування швидкості передачі даних в комп’ютер
 digitalWrite(s0, HIGH); // встановлення високого рівня на виводі 8
 digitalWrite(s1, LOW); // встановлення низького рівня на виводі 9
}

void loop()
{
 color(); // визначення кольору
 Serial.print("Red :"); // передавання рідка "Red :" по послідовному інтерфейсу
 Serial.print(red, DEC); // передавання виміряного значення червоної складової
 Serial.print(" Green : "); // передавання рідка " Green : " по послідовному інтерфейсу
 Serial.print(green, DEC); // передавання виміряного значення зеленої складової
 Serial.print(" Blue : "); // передавання рідка " Blue : " по послідовному інтерфейсу
 Serial.print(blue, DEC); // передавання виміряного значення синьої складової
 Serial.println(); // відправлення команди переведення курсору на нову строку
  // Перевірка, чи виявлено червоний колір
 if (red < blue && red < green && red < 100)
 {
  Serial.println("Vermelho");
  digitalWrite(pinoledverm, LOW); // увімкнути червоний світлодіод
  digitalWrite(pinoledverd, HIGH);
  digitalWrite(pinoledazul, HIGH);
 }
 // Перевірка, чи було виявлено синій колір
 else if (blue < red && blue < green && blue < 1000)
 {
  Serial.println("Azul");
```

```
   digitalWrite(pinoledverm, HIGH);
   digitalWrite(pinoledverd, HIGH);
   digitalWrite(pinoledazul, LOW); // увімкнути синій світлодіод
  }
  // Перевірка, чи було виявлено зелений колір
  else if (green < red && green < blue)
  {
   Serial.println("Verde");
   digitalWrite(pinoledverm, HIGH);
   digitalWrite(pinoledverd, LOW); // увімкнути зелений світлодіод
   digitalWrite(pinoledazul, HIGH);
  }
  Serial.println(); // відправлення команди переведення курсору на нову строку
  // затримка очищення світлодіодів і перезапуск процесу
  delay(50); // затримка 50 мс
  digitalWrite(pinoledverm, HIGH);
  digitalWrite(pinoledverd, HIGH);
  digitalWrite(pinoledazul, HIGH);
}
 // функція визначення кольору
void color()
{
  digitalWrite(s2, LOW); // встановлення низького рівня сигналу на виході s2
  digitalWrite(s3, LOW); // встановлення низького рівня сигналу на виході s3
  // визначення червоної складової кольору
  red = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
  digitalWrite(s3, HIGH); // встановлення високого рівня сигналу на виході s3
  // визначення синьої складової кольору
  blue = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
  digitalWrite(s2, HIGH); // встановлення високого рівня сигналу на виході s2
   // визначення зеленої складової кольору
  green = pulseIn(out, digitalRead(out) == HIGH ? LOW : HIGH);
}
```

Цей модуль через вплив навколишнього середовища і зміни яскравості світла може визначати колір неправильно або з помилками. При використанні датчика кольору слід підтримувати постійну інтенсивність джерела світла в процесі тестування. Кожного разу при зміні джерела світла необхідно зробити корекцію балансу білого.

2.6. Цифровий акселерометр ADXL321 та датчики вібрації

Одним з розповсюджених акселерометрів є датчик ADXL321, виробником якого є компанія Analog Devices.

На рис. 2.38 представлена структурна схема акселерометра ADXL321.

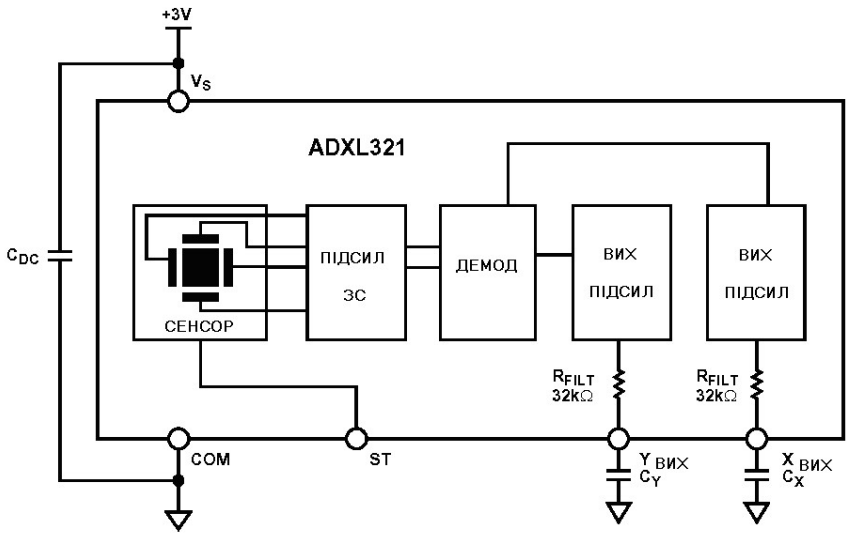


Рис. 2.38.

Основні параметри акселерометра ADXL321 представлені в табл. 2.3.

Таблиця 2.3.

Параметр	Одиниця виміру	Значення
Кількість осей вимірювання	одиниць	2
Максимальне значення вібрації по кожній осі	*g*	± 18
Чутливість при напрузі живлення V_S = 5 В по кожній з осей	мВ/g	100
Рівень нульового прискорення при напрузі живлення V_S = 5 В	В	2,5
Частотний діапазон вимірювання	кГц	0 – 2,5
Напруга живлення	В	2,6 – 6
Температура функціонування	°С	-20 – +70

Тип корпусу і фізичні розміри датчика представлено на рис. 2.39.

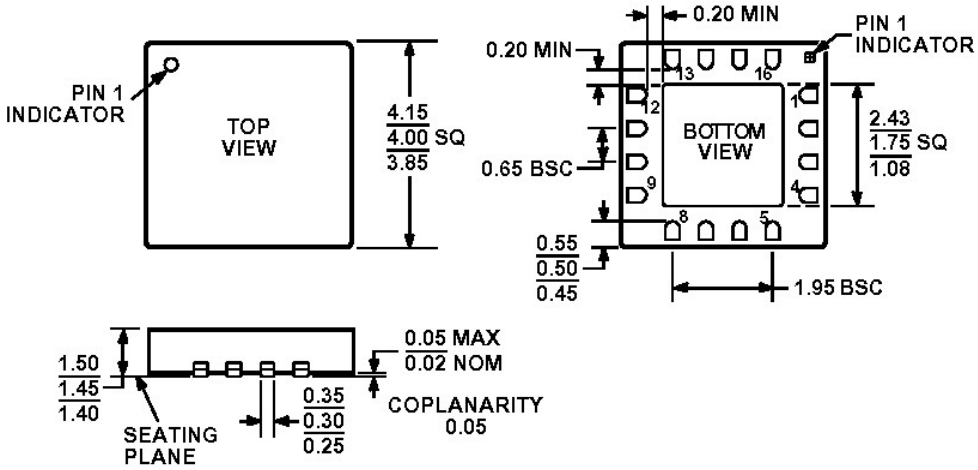
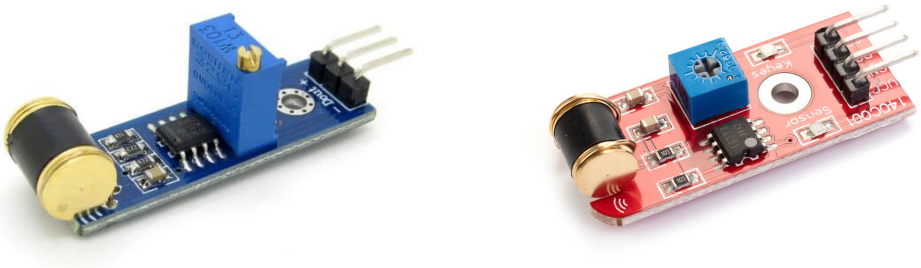


Рис. 2.39.

Датчики вібрації Arduino (їх ще іноді називають датчиками сигналізації) застосовуються для виявлення зовнішніх впливів вібраційного характеру і широко використовуються в протиугінних автомобільних системах, різних охоронних сигналізаціях, дозволяють детектувати вібрації при початку землетрусу.

Основа датчика вібрації – гнучка металева пружинка, всередині пластикової трубки, яка коливається від будь-яких впливів на неї. Далі сигнал подається для підсилення на операційний підсилювач LM386, а потім на аналоговий вихід. На рис. 2.40, а, б, представлено зовнішній вигляд вібродатчиків Logo sensors v1.5 та 140C001 відповідно.



а б

Рис. 2.40.

Кожен з цих датчиків має виводи GND, Vcc (живлення) і вихід аналогового сигналу A0. Налаштування чутливості датчика здійснюється потенціометром, який знаходяться на платі. Кожен з датчиків має світлодіод, що сигналізує про наявність напруги живлення. Датчик 140C001 має додатковий цифровий вихід D0, на якому при досягненні граничного значення величини вібрації видається логічний нуль. Межа спрацьовування регулюється потенціометром. Наявність цифрового виводу D0 дозволяє використовувати датчик 140C001 автономно, без

підключення до контролера. Датчики мають монтажний отвір для кріплення до поверхні.

Розглянемо використання датчика вібрацій в проекті охоронної сигналізації. Принципова схема моделі в Proteus представлено на рис. 2.41. При ударі по поверхні, до якої закріплений датчик, спрацьовує сигнал тривоги (звуковий сигнал на динамік). Додатково буде виводитися на світлодіодну шкалу відносне значення, що видається датчиком вібрацій (для налаштування сигналізації).

Бібліотеку для Proteus датчика вібрації можна знайти і завантажити за посиланням:

https://www.theengineeringprojects.com/2016/08/vibration-sensor-library-proteus.html

Файли бібліотеки потрібно скопіювати в каталог LIBRARY, який, в свою чергу, знаходиться в каталозі, куди було встановлено Proteus. В бібліотеці Proteus модель датчика вібрації можна знайти за ключовим словом VIBRATION SW-420. Також необхідно у вікні властивостей датчика вібрації вказати шлях до файлу VibrationSensorTEP.HEX. Він також повинен бути в каталозі LIBRARY разом з файлами бібліотеки датчика вібрації.

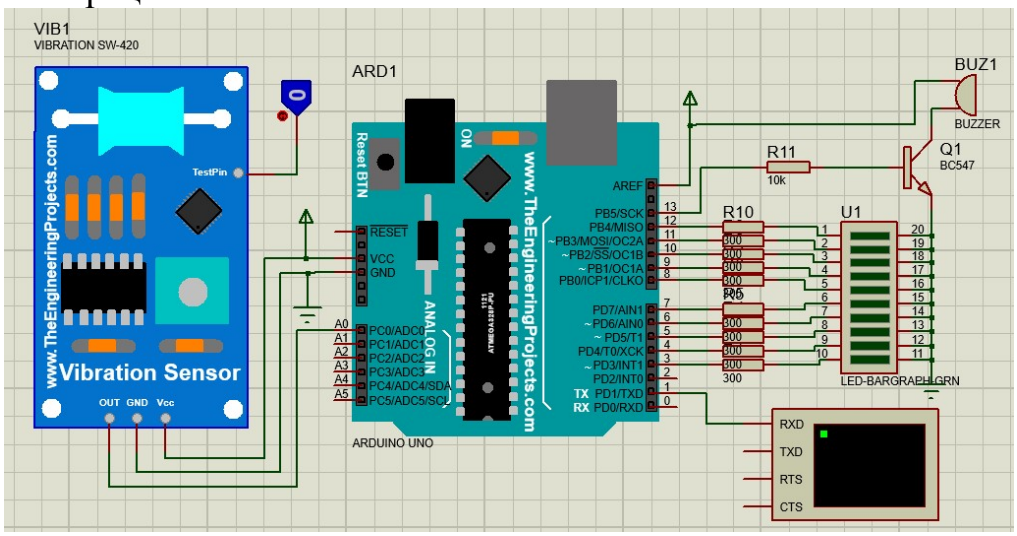


Рис. 2.41.

Текст програми представлено нижче.

```
int aPin=A0; // номер виводу, до якого підключений аналоговий вихід датчика
// номери виводів для підключення світлодіодної шкали
int ledPins[10]={3,4,5,6,7,8,9,10,11,12};
// оголошення змінної для зберігання значення,зчитаного з датчика
int avalue=0;
// змінна для зберігання кількості світлодіодів, які увімкнені
int countled=10;
// значення для вмикання сигналізації
int anxietyvalue=1000;
```

```
// номер виводу, до якого підключено реле (або звуковий сповіщувач)
int soundPin=13;
// частота звукового сигналу
int freq=587;

void setup()
 {
 // ініціалізація послідовного порту
 Serial.begin(9600);
 // налаштування виводів керування світлодіодами на вихід (в режим OUTPUT)
for(int i=0;i<10;i++)
 {
 pinMode(ledPins[i],OUTPUT);
 }
 }

void loop()
 {
 // отримання даних з аналогового виходу датчика
 avalue=analogRead(aPin);
// виведення значення в монітор послідовного порту Arduino
 Serial.print("avalue=");
 Serial.println(avalue);
 // масштабування значення на 10 світлодіодах
 countled=map(avalue,20,1000,0,9);
 // вивід значення в монітор послідовного порту Arduino
 Serial.print("countled=");
 Serial.println(countled);
 // індикація рівня вібрації
 for(int i=1;i<=10;i++)
 {
 if(i<=countled)
 digitalWrite(ledPins[i-1],HIGH); // увімкнення світлодіода
 else
 digitalWrite(ledPins[i-1],LOW); // вимикання світлодіода
 }
 // пауза перед наступним отриманням значення 1000 мс
 delay(100);
 // сигнал тривоги
 if(countled>8)
 {
 tone(soundPin,freq,10000);
 }
}
```

Для перевірки роботи програми на реальному пристрої потрібно скласти схему, що представлена на рис. 2.42.

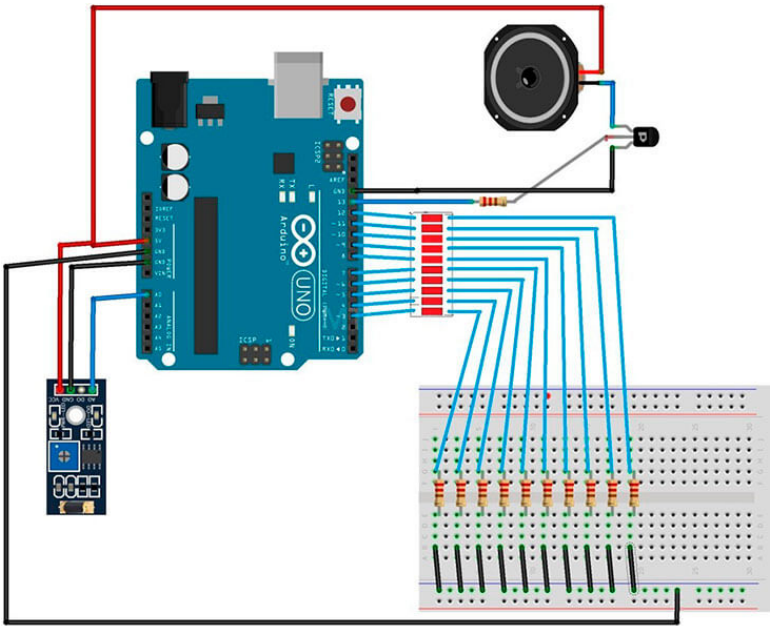


Рис. 2.42.

Аналоговий вихід датчика підключений до аналогового входу Arduino, який представляє собою 10-бітний аналого-цифровий перетворювач (АЦП), що дозволяє на виході отримувати значення від 0 до 1023. У стані спокою датчику на аналоговому вході Arduino буде фіксуватися значення від 0 (датчик 140C001) до 9-10 (датчик Logo sensors v1.5). У разі наявності вібрацій (наприклад, удар по столу, до якого прикріплений датчик) значення аналогового сигналу збільшується багаторазово, що видно за показниками світлодіодного шкального індикатора. При граничному значенні, який визначається експериментально, на динамік подається сигнал тривоги на 10 секунд. Також в режимі реального часу за допомогою Монітора порта в середовищі Arduino IDE (комбінація клавіш Ctrl+Shift+M) можна спостерігати значення, що надходить з датчика в контролер.

2.7. Датчик звуку KY-037

Звичайні електретні мікрофони – це дуже корисні сенсори. Окрім того, що вони можуть використовуватися за прямим призначенням – вловлювати звуки – вони також можуть бути додатковими джерелами інформації в інших ситуаціях, наприклад, працювати сенсорами удару. Мікрофони дешеві і споживають мало енергії. Вони займають небагато місця і надійні.

У деяких випадках, мікрофони навіть краще використовувати в якості тактильних датчиків, ніж звичайні механічні кнопки, оскільки маленькі контактні кнопки, зазвичай, дуже ненадійні. Мікрофони ж не містять видимих рухомих частин і, будучи обладнаними гумовими демпферами, можуть бути дуже надійними і чутливими. Більш того, вони дають більше інформації, ніж звичайна кнопка і ця інформація більш

схожа на ту, що видають сенсори у людини. Мікрофони не можуть реагувати на статичне навантаження, лише на зміну в навантаженні.

Для розпізнавання мови і звуків можливостей мікроконтролера ATmega16 недостатньо. Однак можна використовувати спрощені версії для розпізнавання окремих команд. Зовнішній вигляд датчиків звуку представлено на рис. 2.43, а, б, а принципова схема датчика звуку представлена на рис. 2.44.



Рис. 2.43.

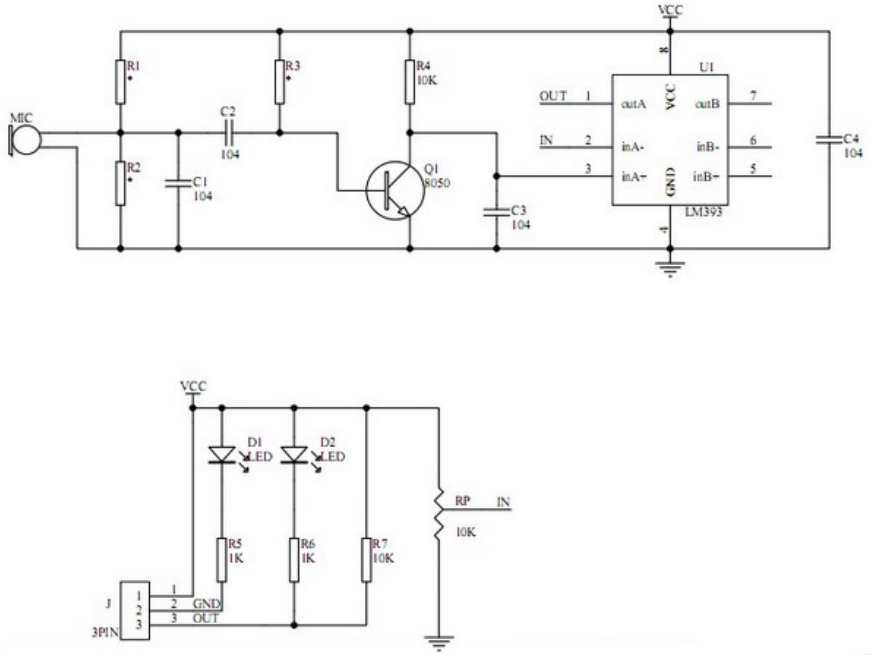


Рис. 2.44.

Потенціометр регулює коефіцієнт посилення підсилювача сигналу з мікрофона. Він буває корисний, якщо необхідно змінити умови спрацьовування мікропроцесорного пристрою без необхідності його перепрограмування. Чим вище чутливість модуля, тим вище частка перешкод в корисному сигналі датчика. Рекомендується починати роботу з модулем при середньому положенні потенціометра. В такому випадку чутливість модуля можна буде легко змінити в будь-яку сторону.

Особливості датчика:

1. Індикація вихідного сигналу.

2. Один канал вихідного сигнал.

4. У разі перевищення звукового порогу на виході встановлюється логічна одиниця.

5. Датчик може бути використаний для акустичного контролю світла, звукової і світлової сигналізації, виявлення звуку, шуму, звукової активності.

6. Регулювання потенціометром межі спрацьовування.

Датчик звуку вимірює рівень гучності звуку і призначений для визначення факту наявності звуку (цифровий вихід), але він не може визначити його рівень і звукові частоти. Датчик можна застосувати для управління роботом, в тому числі з розшифровкою кодової послідовності звуків (наприклад, три удари в долоні). Також датчик звуку можна використовувати для вмикання або вимикання освітлення в приміщенні. В наступному прикладі реалізовано систему керування освітленням в приміщенні за допомогою датчика звуку. Якщо світло вимкнене, то гучний звук, наприклад. удар в долоні, увімкне світло. Якщо світло увімкнене, то удар в долоні його вимикає. Фактично, програма керування представляє собою реалізацію Т-тригера, на вхід якого поступає сигнал з датчика, а вихідний сигнал керує освітленням. Принципова схема мікропроцесорної системи керування освітленням за допомогою звукових команд представена на рис. 2.45.

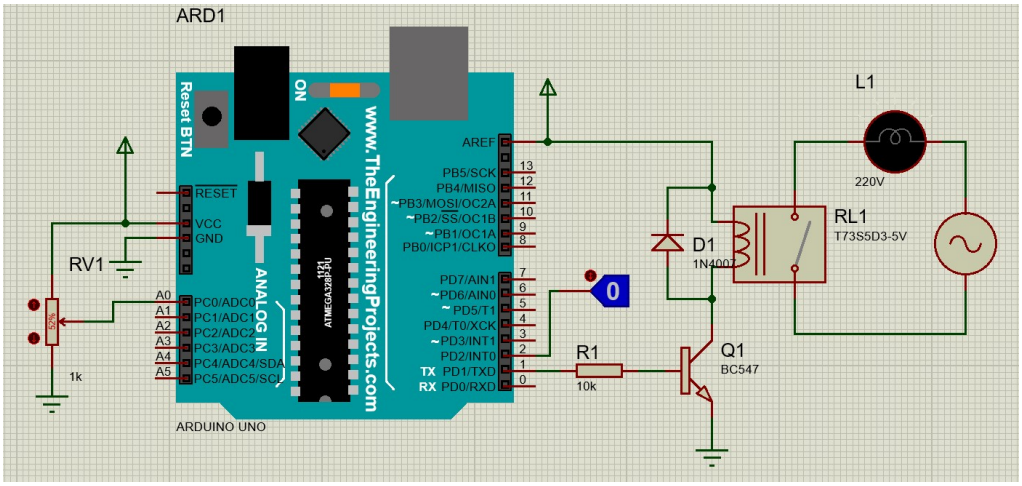


Рис. 2.45.

Для моделювання сигналу з аналогового виходу датчика в схемі використовується потенціометр, а для моделювання дискретного виходу датчика в схемі використовується елемент LOGITTOGGLE. Джерело змінної напруги можна знайти в бібліотеці Proteus за ключовим словом ALTERNATOR. У вікні налаштувань компонента лампи (LAMP) потрібно вказати її номінальну напругу 220 В. У вікні налаштувань джерела змінної напруги потрібно також вказати напругу 220 В, а частоту 50 Гц. Текст програми представлено нижче.

```
// номер виводу, до якого підключений аналоговий вихід датчика звуку
const int sensorPin = A0;
const int relayPin = 1; // номер виводу, який керує реле
```

```
int sensorState = 0; // оголошення змінної для запису в неї виміряного значення гучності
int relayState = 0; // оголошення змінної для збереження поточного стану реле

void setup() {
pinMode(relayPin, OUTPUT); // налаштування виводу 1 на вихід для керування реле
pinMode(sensorPin, INPUT); // налаштування виводу A0 на вхід
}

void loop(){
sensorState = analogRead(sensorPin); // зчитування сигналу з датчика (напруги з АЦП)
relayState = digitalRead(relayPin); // зчитування поточного стану реле
// якщо реле увімкнене та рівень гучності перевищує задану межу
if ((relayState == HIGH) && (sensorState > 200)) {
delay(250); // часова затримка 250 мс
digitalWrite(relayPin, LOW);  // вимикання реле та відключення освітлення
delay(250); // часова затримка 250 мс
}
// якщо реле вимкнено та рівень гучності перевищує задану межу
if ((relayState == LOW) && (sensorState > 200)) {
delay(250); // часова затримка 250 мс
digitalWrite(relayPin, HIGH); // вмикання реле та включення освітлення
delay(250); // часова затримка 250 мс
}
}
```

Після компіляції програми можна перевірити роботу схеми. На рис. 2.46 представлений вигляд схеми в режимі моделювання для випадку, коли освітлення увімкнене.

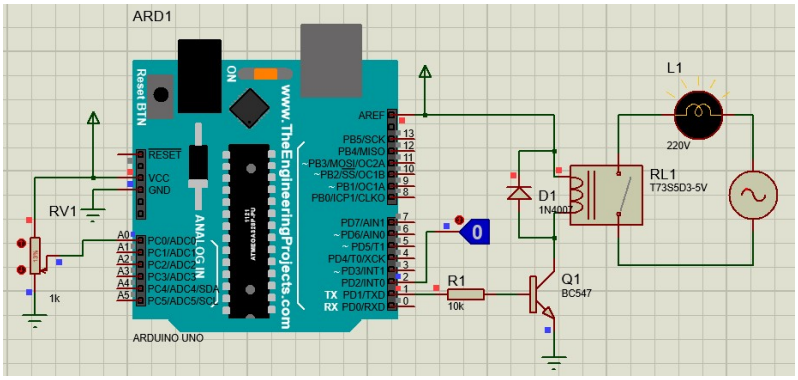


Рис. 2.46.

Змінивши рівень сигналу на вході контролера за допомогою змінного резистора можна впевнитися в коректності роботи схеми та програми. Для перевірки роботи програми на реальному пристрої потрібно зібрати схему, яку представлено на рис. 2.47.

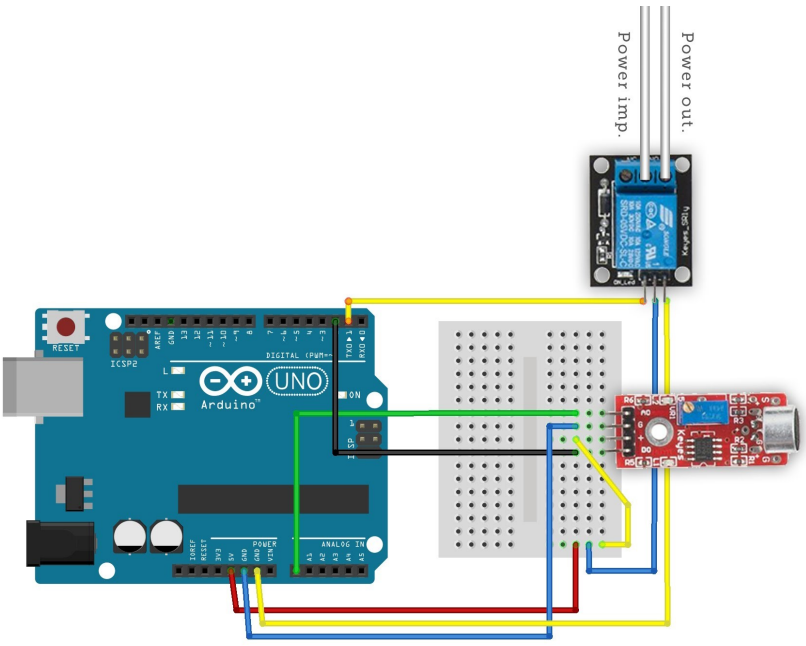


Рис. 2.47.

У випадку. Коли мікрофон недостатньо чутливий для надійного приймання звукового сигналу, потрібно налаштувати модуль (його чутливість) шляхом обертання потенціометра, розташованого на платі модуля. Для випадку, коли задіяний дискретний вихід датчика звуку, для керування освітленням можна використовувати схему, представлену на рис. 2.48.

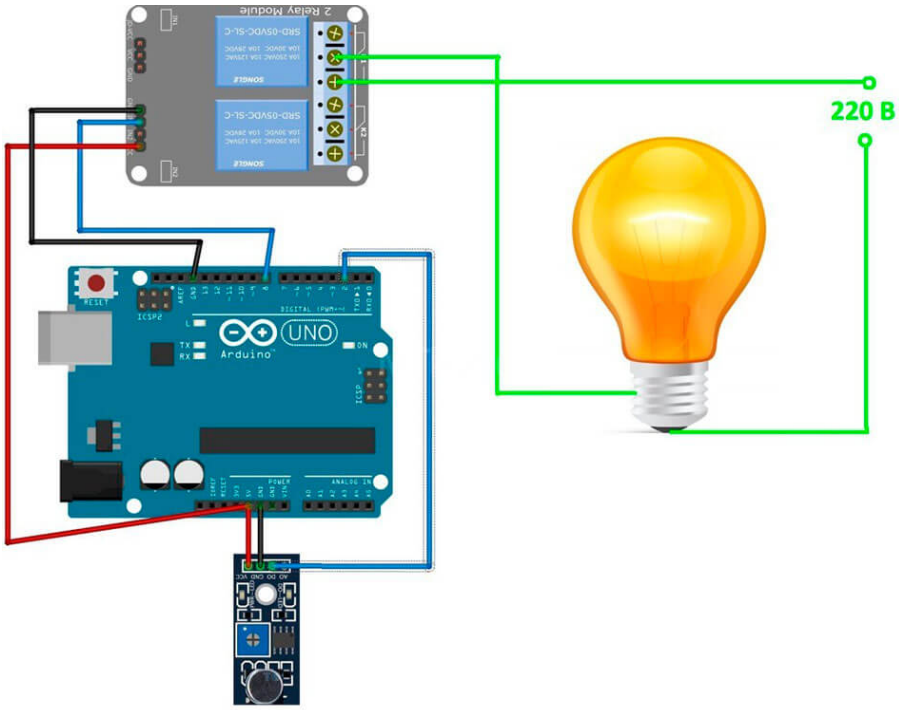


Рис. 2.48.

Текст програми представлено нижче.

```
int soundPin=2; // номер виводу, до якого підключений дискретний вихід датчик звуку
```

```
int relayPin=1; // номер виводу, до якого підключено реле
// оголошення змінної для збереження стану лампи
// true – включено, false – виключено
boolean statuslamp;

void setup()
{
pinMode(relayPin, OUTPUT); // налаштування виводу керування реле на вихід
statuslamp = false;     // початковий стан – лампа вимкнена
digitalWrite(relayPin, LOW); // встановлення на виводі номер 1 низького рівня
}

void loop()
{
// перевірка значення на виводі OUT датчика звуку
if(digitalRead(soundPin) == LOW) // реєстрація факту наявності звуку
{
// змінити статус лампи на протилежний
statuslamp = !statuslamp;
// перемкнути лампу (якщо лампа була вимкнена, то увімкнути, і навпаки)
digitalWrite(relayPin, statuslamp);
delay(250); // часова затримка щоб не реагувати на відлуння
}
}
```

В поєднанні з іншими датчиками, датчик звуку надає можливості побудови багатофункціональних систем керування з голосовим або іншим способом керування.

2.8. Датчик газу MQ-2

Датчики присутності газів потрібні для виявлення витоку природних і побутових газів. Датчики реагують на виникнення в повітрі домішок газу. При появі горючого газу провідність датчика зростає з ростом концентрації цього газу. Використовуючи просту електронну схему, можна перетворити провідність датчика в сигнал, пропорційний концентрації газу. Чутливий матеріал датчиків газу серії MQ – це високоактивний оксид металу, зазвичай SnO_2. Коли чутлива поверхня датчика нагрівається до певної температури, атоми кисню абсорбуються поверхнею напівпровідника, насиченого електронами.

Датчик газу (рис. 2.49), побудований на базі газоаналізатора MQ-2 дозволяє виявляти наявність в навколишньому повітрі вуглеводневих газів (пропан, метан, бутан), диму (зважені частинки, які є результатом горіння), водню.

Датчик можна використовувати для виявлення витоків промислового газу і задимлення. Вихідним результатом є аналоговий сигнал, пропорційний вмісту газів, до яких чутливий газоаналізатор.

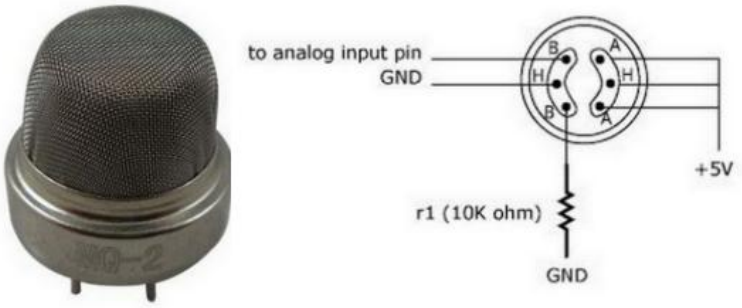


Рис. 2.49.

В газоаналізатор вбудований нагрівальний елемент, який необхідний для хімічної реакції. Тому під час роботи сенсор буде гарячим, це нормально. Для отримання стабільних показань новий сенсор необхідно один раз прогріти (залишити включеним) протягом 24 годин. Після цього стабілізація після включення буде займати близько хвилини.

Показання сенсора схильні до впливу температури і вологості навколишнього повітря. Тому в разі використання датчика газу в мінливому середовищі, при необхідності отримання точних показань, знадобиться реалізувати компенсацію цих параметрів. Схема вмикання датчика представлена на рис. 2.50.

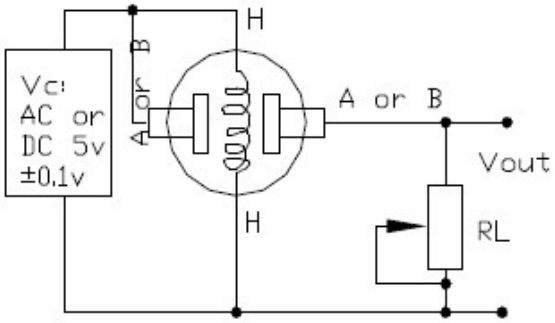


Рис. 2.50.

Характеристики датчика газу:
– Напруга живлення: 5 В;
– Споживаний струм: 160 мА.
Діапазон вимірювань:
– Пропан: 0,2 – 5 проміле;
– Бутан: 0,3 – 5 проміле;
– Метан: 5 – 2 проміле;
– Водород: 0,3 – 5 проміле.

Рис. 2.51 допомагає зрозуміти принцип роботи датчика. Напруга на виході датчика залежить від рівня концентрації газу в атмосфері.



Рис. 2.51.

Аналізуючи сигнал з виходу датчика можна зробити висновки про наявність небезпечної концентрації диму або відповідного газу у повітрі. На рис. 2.52 представлена принципова схема мікропроцесорної системи контролю концентрації газу у повітрі. Якщо концентрація газу невелика, горить зелений світлодіод. Якщо концентрація газу перевищує деяку задану межу, загоряється червоний світлодіод і вмикається звукове сповіщення. Також постійно відбувається передача даних про концентрацію газу по послідовному інтерфейсу. Підключивши комп'ютер до контролера Arduino за допомогою вбудованого в середовище розробки Монітору порту можна бачити результати вимірювань в режимі реального часу.

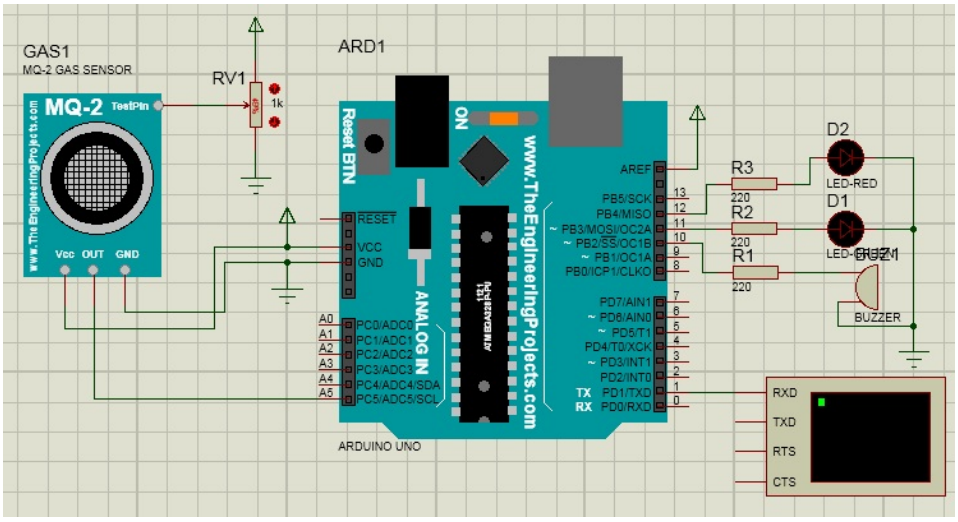


Рис. 2.52.

Модель датчика газу має додатковий вхід TestPin, на який потрібно подавати постійну напругу. Значення напруги на цьому вході є тим значенням напруги, яке формує датчик на своєму виході. Таким чином моделюється робота датчика газу. Текст програми (скетч) представлено нижче.

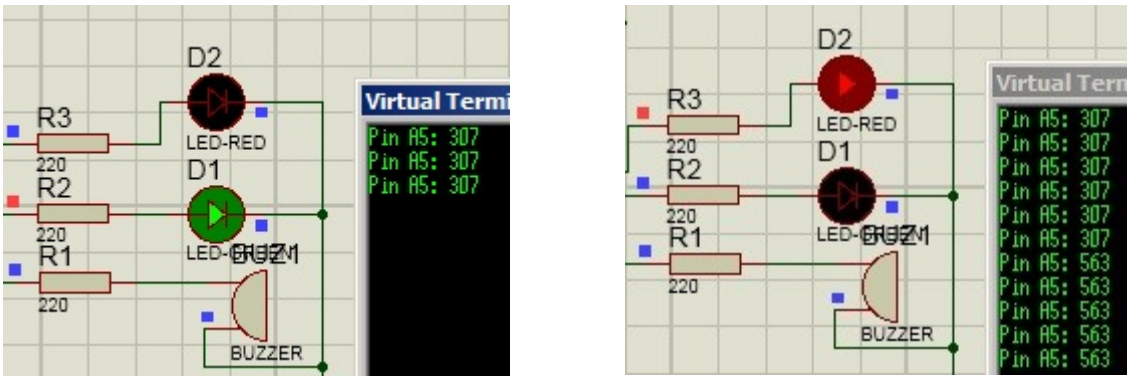
```
int redLed = 12; // номер виводу, до якого підключений червоний світлодіод
int greenLed = 11; // номер виводу, до якого підключений зелений світлодіод
int buzzer = 10; // номер виводу, до якого підключений звуковий сповіщувач
int smokeA5 = A5; // номер аналогового виводу, до якого підключений датчик
int sensorThres = 400; // значення граничної межі для спрацювання сповіщувача

void setup() {
// вивід, до якого підключений червоний світлодіод, налаштовується на вихід
  pinMode(redLed, OUTPUT);
// вивід, до якого підключений зелений світлодіод, налаштовується на вихід
  pinMode(greenLed, OUTPUT);
// вивід, до якого підключений звуковий сповіщувач, налаштовується на вихід
  pinMode(buzzer, OUTPUT);
// вивід, до якого підключений вихід датчика газу, налаштовується на вхід
  pinMode(smokeA5, INPUT);
  Serial.begin(9600); // налаштування швидкості передачі даних 9600 біт/с
}

void loop() {
  // зчитування даних з аналогового входу, до якого підключений датчик
  // в результаті аналого-цифрового перетворення в змінну записується число
  // в діапазоні від 0 до 1023, значення якого пропорційне концентрації газу
  int analogSensor = analogRead(smokeA5);
  // передавання текстового рядка по послідовному інтерфейсу
  Serial.print("Pin A5: ");
  Serial.println(analogSensor); // передавання виміряного значення з датчика
  // перевірка, чи не перевищує концентрація газу в повітрі заданої межі
  if (analogSensor > sensorThres) // якщо концентрація велика і перевищує межу
  {
    digitalWrite(redLed, HIGH); // встановити високий рівень на виводі, до
                                // якого підключений червоний світлодіод
    digitalWrite(greenLed, LOW); // встановити низький рівень на виводі
                                // до якого підключений зелений світлодіод
    tone(buzzer, 1000, 200);    // включити звукове сповіщення
  }
  else     // якщо концентрація газу низька і не перевищує межу
  {
    digitalWrite(redLed, LOW); // встановити низький рівень на виводі, до
                                // якого підключений червоний світлодіод
    digitalWrite(greenLed, HIGH); // встановити високий рівень на виводі
                                // до якого підключений зелений світлодіод
    noTone(buzzer);              // виключити звукове сповіщення
  }
  delay(100);
}
```

На рис. 2.53, а, представлено фрагмент схеми в режимі моделювання за умови низької концентрації газу. Горить червоний світлодіод, що говорить про безпечну концентрацію газу. У вікні віртуального терміналу можна бачити отримане значення з виходи датчика після аналого-

цифрового перетворення. На рис. 2.53, б, показано випадок, коли концентрація газу перевищила задану межу, яка складає 400 апаратних одиниць, і увімкнулися червоний світлодіод та звуковий сповіщувач.



а б

Рис. 2.53.

Бібліотеку датчика газу можна знайти та завантажити в Інтернет за посиланням: https://www.theengineeringprojects.com/2016/05/gas-sensor-library-proteus.html

Після завантаження архіву з файлами моделі датчика газу, архів потрібно розпакувати, а файли скопіювати в каталог LIBRARY. За замовчуванням шлях до каталогу LIBRARY наступний: C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\LIBRARY

В бібліотеці компонентів датчик газу можна знайти якщо ввести в поле Keywords назву – MQ-2, як показано на рис. 2.54.

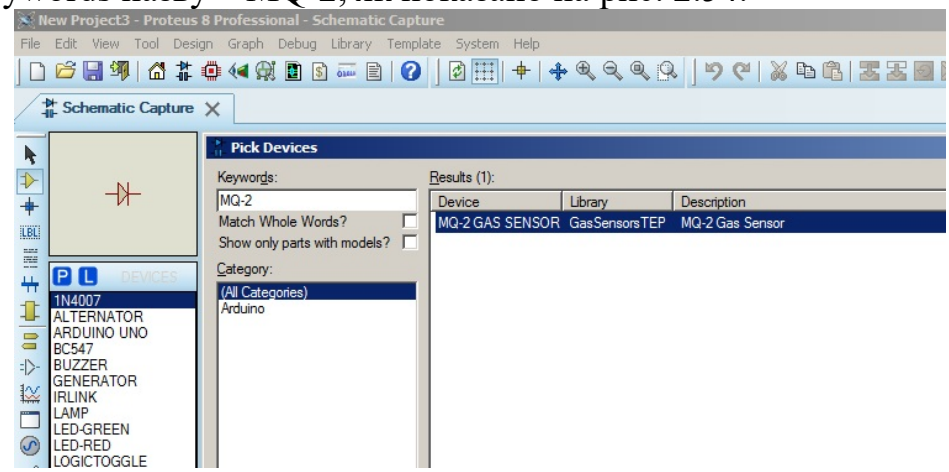


Рис. 2.54.

Також необхідно відкрити вікно властивостей датчика газу після того, як він буде доданий в схему, та вказати шлях до файлу GasSensorTEP.HEX, як показано на рис. 2.55. Цей файл також зручно розмістити в каталозі LIBRARY. Отже, повний шлях до цього файлу наступний:

C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\LIBRARY\GasSensorTEP.HEX

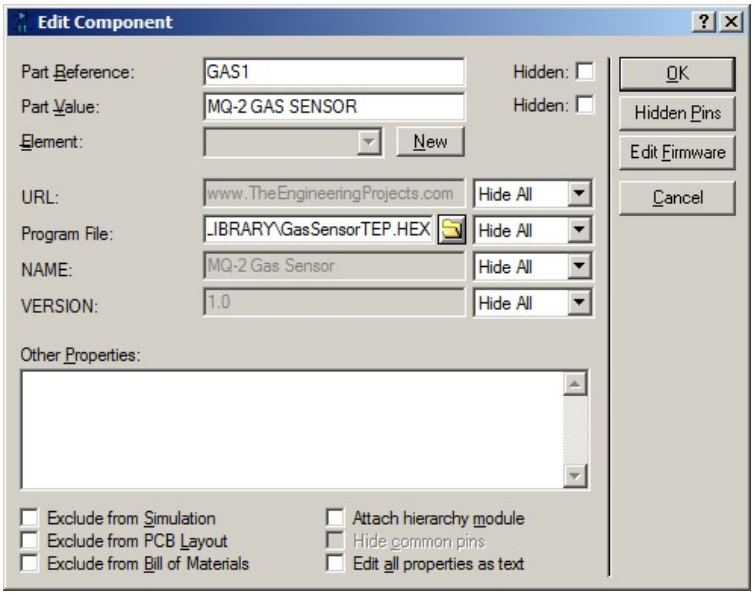


Рис. 2.55.

При бажанні виготовити реально діючу мікропроцесорну систему контролю концентрації газу у повітрі, потрібно зібрати схему, як показано на рис. 2.56.

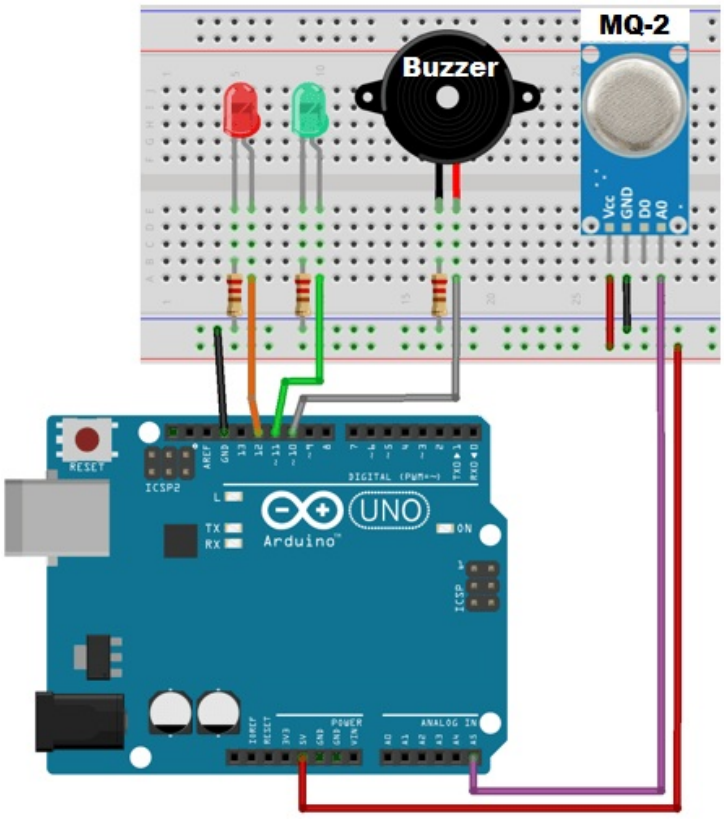


Рис. 2.56.

В програмі для керування звуковим сповіщувачем було використано функцію tone(). Ця функція генерує на порту введення/виведення сигнал – прямокутну "хвилю" заданої частоти і з 50% робочим циклом. Тривалість генерування сигналу може бути задана параметром, в іншому випадку сигнал генерується доти, поки не буде викликана функція noTone(). До

порту входу/виходу може бути підключений п’єзо динамік для відтворення сигналу.

Відтворюватися одночасно може тільки один сигнал. Якщо сигнал вже відтворюється на одному порту, то виклик функції tone() з номером іншого порту в якості параметра не буде мати жодного ефекту. Якщо ж функція tone() буде викликана з тим же номером порту, то буде встановлено нову частоту сигналу.

Використання функції tone() завадить використовувати ШІМ на портах входу/виходу з номерами 3 і 11 (крім плати Arduino Mega). Синтаксис функції має наступний вигляд:

```
tone(pin, frequency);
tone(pin, frequency, duration);
```

Параметри, що передаються при виклику функції: pin – номер порту входу/виходу, на якому буде генеруватися сигнал; frequency – частота сигналу в герцах; duration – тривалість сигналу в мілісекундах.

За посиланням https://playground.arduino.cc/Main/MQGasSensors можна знайти документацію на датчики різних газів: метану, бутану, диму (MQ-2), алкоголю, етанолу (MQ-3), метану (MQ-4), природний газ (MQ-5), бутан (MQ-6), монооксид вуглецю (MQ-7), водень (MQ-8) та багато інших.

2.9. Датчик полум’я KY-026

Датчик KY-026, зовнішній вигляд якого представлено на рис. 2.57, реагує на відкрите полум'я. Чутливим елементом датчика служить фотодіод, який отримує інфрачервоне випромінювання.

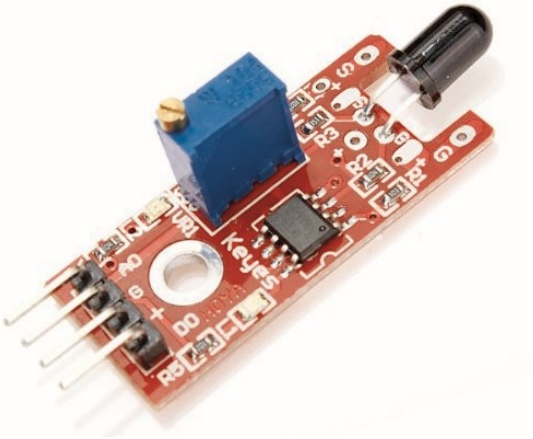


Рис. 2.57.

Датчик може використовуватися в газових котлах, бензинових пальниках, які мають електрозапалювання. В цьому випадку дуга , що створюється високою напругою, підпалює паливо, а мікропроцесорна керування отримує дані від датчика полум'я і визначає, загорілося паливо чи ні, і припиняє роботу електричного запалювання. Надалі відбувається управління подачею палива і контроль наявності полум'я. Цікавим є застосування модуля KY-026 для невеликого робота-пожежника.

Напруга живлення датчика складає 3,3 В або 5 В. Кут виявлення полум’я дорівнює 60^0.

На плату модуля KY-026 нанесено маркування контактів з'єднувача:

A0 – аналоговий вихід, вихідна напруга пропорційна рівню освітленості датчика ІЧ випромінюванням;

G – загальний провідник;

+ – живлення;

D0 – вихід сигналу має логічні рівні 0 (LOW, при відсутності полум’я) або 1 (HIGH, при виявленні полум’я).

На рис. 2.58 представлена принципова схема модуля KY-026.

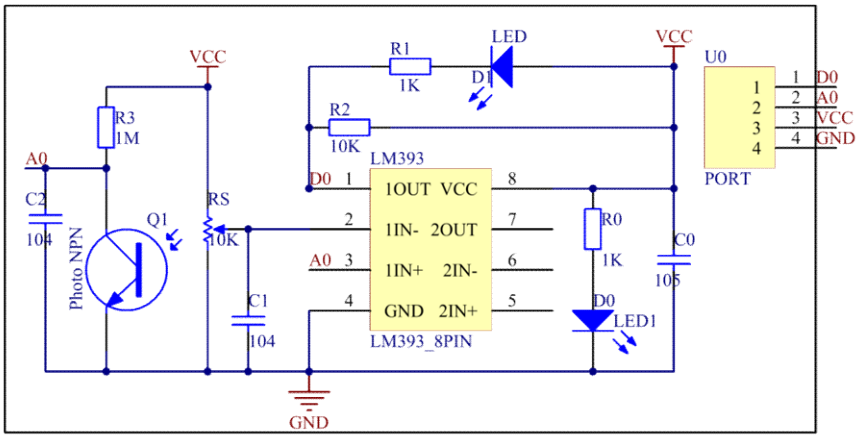


Рис. 2.58.

Принцип роботи модуля датчика вогню полягає в наступному. Фотодіод з'єднаний зі входом компаратора, виконаного на мікросхемі LM393. За допомогою змінного резистора виконується налаштування межі спрацьовування компаратора. Так встановлюється чутливість датчика вогню. При виявленні полум'я з яскравістю вище встановленої при налаштуванні межі, на виході D0 буде високий рівень напруги. Якщо вогню немає, або його яскравість мала, то на виході D0 буде низький рівень напруги. На аналоговий вихід надходить підсилений сигнал фотодіода. Світлодіод L1, розташований на датчику, показує включення живлення. Світлодіод L2 повідомляє про спрацювання датчика і формування на виході D0 високого рівня. Світлодіод L2 полегшує налаштування модуля KY-026.

Бібліотеку інфрачервоного датчика полум’я для Proteus можна завантажити за посиланням:

https://www.theengineeringprojects.com/2016/07/flame-sensor-library-proteus.html

Після завантаження з сайту архіву з файлами моделі датчика полум’я, архів потрібно розпакувати, і файли перемістити в каталог LIBRARY, який знаходиться у каталозі, куди було встановлено Proteus. За

замовчуванням шлях до цього каталогу наступний: C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\LIBRARY.

Наступний приклад демонструє використання датчика полум'я в мікропроцесорній системі пожежної сигналізації. При виявленні полум'я вмикається звукове сповіщення. В бібліотеці Proteus цей датчик можна знайти за ключовими словами FLAME SENSOR, як показано на рис. 2.59.

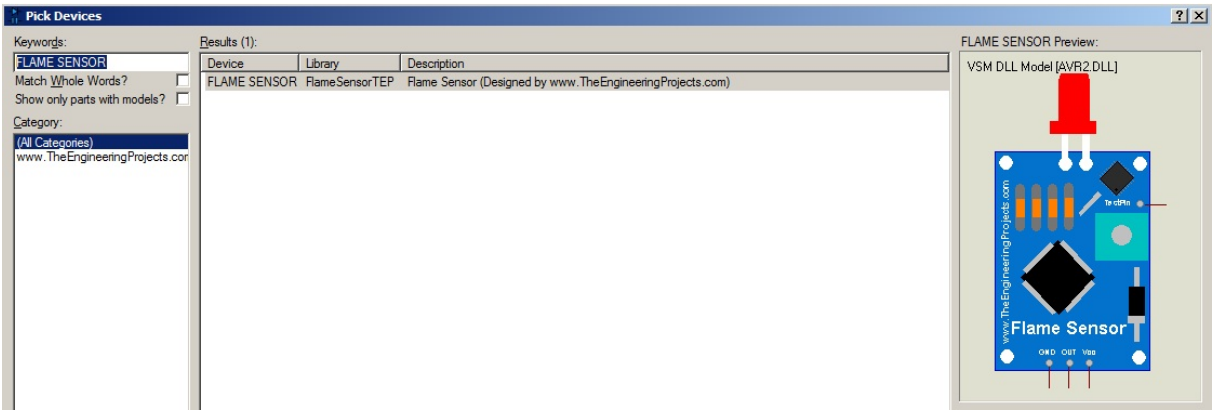


Рис. 2.59.

Принципова схема моделі системи в Proteus представлена на рис. 2.60.

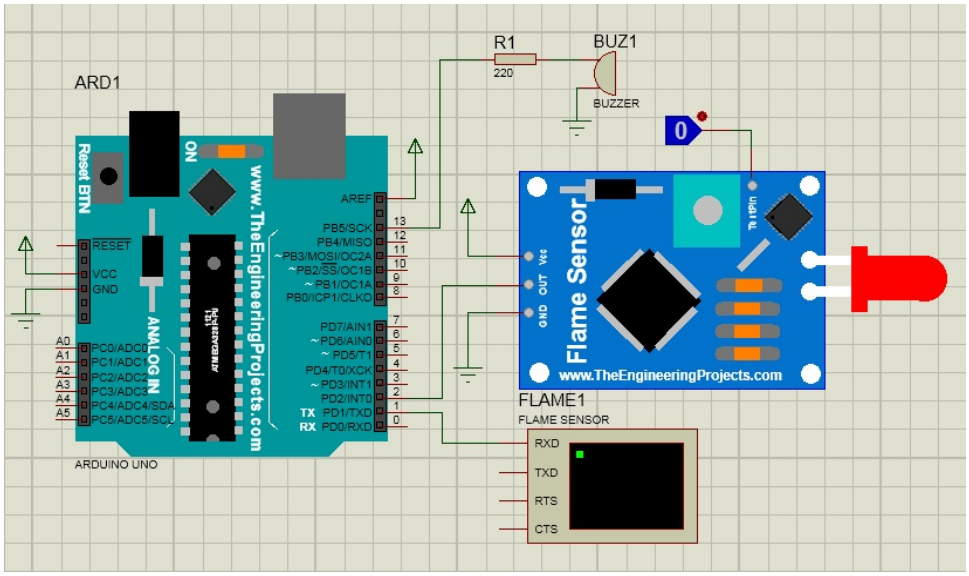


Рис. 2.60.

В даному прикладі виконується перевірка стану дискретного виходу датчика. Якщо на виході датчика контролером зафіксована наявність високого рівня сигналу, вмикається звукове сповіщення. Також в комп'ютер по послідовному комунікаційному інтерфейсу надсилається інформація про показання датчика. Текст програми представлено нижче.

```
int Buzzer = 13; // номер виводу, до якого підключений звуковий сповіщувач
int FlamePin = 2;  // номер виводу, до якого підключено вихід датчика
int Flame = HIGH;  // визначення рівня, при якому фіксується наявність полум'я
```

```
// функція налаштування контролера Aruino, виконується 1 раз при вмиканні пристрою
void setup() {
  pinMode(Buzzer, OUTPUT); // налаштування виводу номер 12 на вихід
  pinMode(FlamePin, INPUT); // налаштування виводу номер 2 на вхід
  Serial.begin(9600);  // налаштування швидкості передачі даних 9600 біт/с
}
// ця функція виконується доки не буде вимкнено пристрій
void loop() {
  Flame = digitalRead(FlamePin); // зчитування дискретного сигналу з датчика полум'я
  if (Flame== HIGH) // якщо на виході датчика високий рівень
  {
    // відправлення повідомлення ("HIGH FLAME" в комп'ютер
    Serial.println("HIGH FLAME");
    digitalWrite(Buzzer, HIGH);        // вмикання звукового сповіщувача
  }
  else    // якщо ж на виході датчика низький рівень
  {
    Serial.println("No flame"); // відправлення повідомлення "No flame" в комп'ютер
    digitalWrite(Buzzer, LOW); // вимикання звукового сповіщувача
  }
}
```

На рис. 2.61 представлено фрагмент схеми в режимі моделювання.

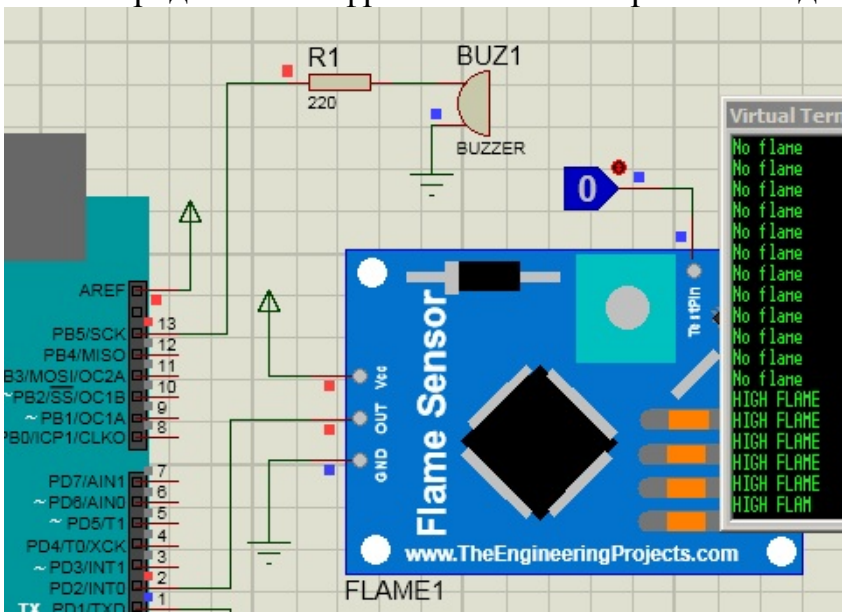


Рис. 2.61.

При відсутності полум'я звуковий сповіщувач вимкнений, і в комп'ютер відправляється повідомлення «No flame». Якщо датчик зафіксував наявність полум'я, вмикається звуковий сповіщувач та в комп'ютер відправляється повідомлення «HIGH FLAME». Для моделювання ситуації спрацювання датчика в його моделі є вхід TestPin. Підключивши до цього входу елемент LOGICTOGGLE можна в режимі реального часу моделювати ситуацію спрацьовування датчика полум'я.

Для створення реальної системи потрібно зібрати схему, як показано на рис. 2.62. Підключивши контролер Arduino до комп'ютера за

допомогою USB-кабеля, для завантаження програми потрібно натиснути на кнопку «Загрузить», яка розташована у верхній часині головного вікна середовища розробки Arduino IDE.

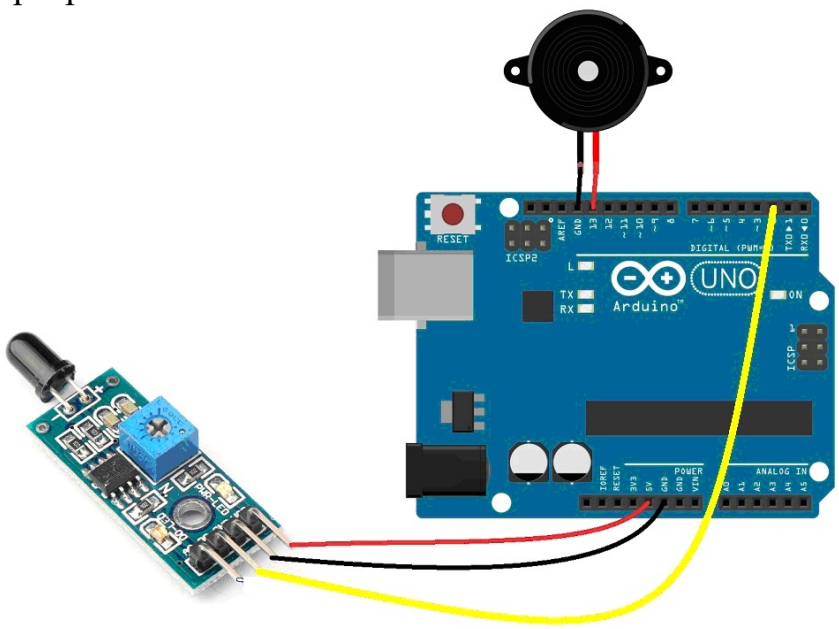


Рис. 2.62.

При необхідності до контролера Arduino можна підключити аналогових вихід датчика полум'я. В такому разі потрібно відстежувати рівень напруги на виході датчика за допомогою аналого-цифрового перетворення, і фіксувати момент, коли напруга перевищує деяку задану межу. Схема підключення датчика для такого варіанту системи представлена на рис. 2.63.

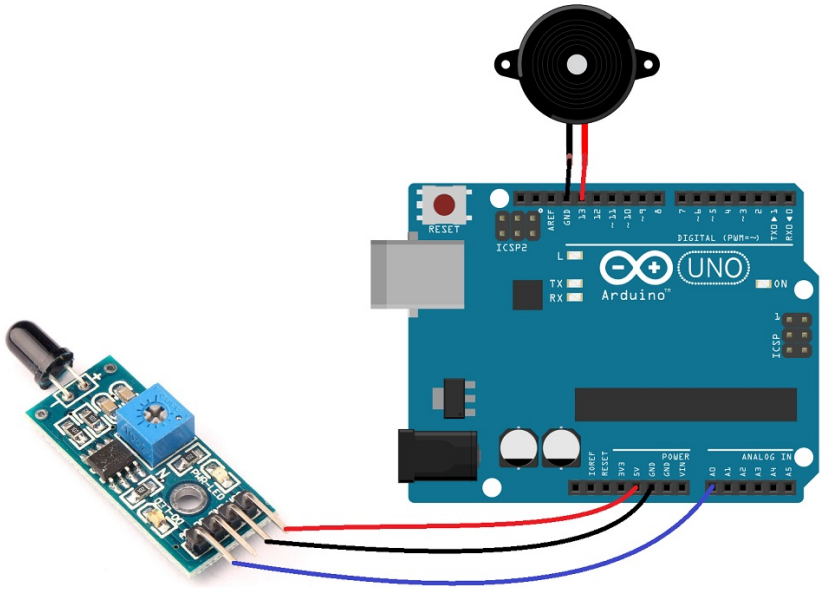


Рис. 2.63.

Для моделювання пристрою в середовищі Proteus і перевірки правильності роботи програми, замість датчика полум'я використовується резистор зі змінним опором, як показано на рис. 2.64. Змінюючи опір

резистора, на вхід A0 контролера поступає напруга, рівень якої визначає інтенсивність інфрачервоного випромінювання, зафіксованого датчиком.

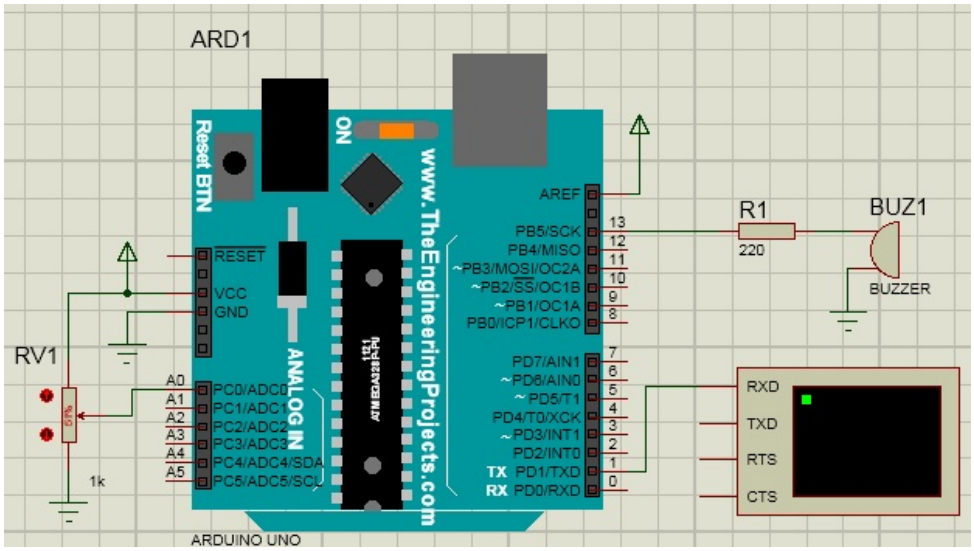


Рис. 2.64.

Текст програми представлено нижче.

```
// номер виводу, до якого підключений аналоговий вихід датчика
const int analogPin = A0;
const int BuzzerPin = 13;    // номер виводу, до якого підключений звуковий сповіщувач
// межа (в апаратних одиницях), при перевищенні якої спрацьовує звуковий сповіщувач
const int threshold = 512;   // при переведенні в вольти це значення складає 2,5 В
void setup() {

 pinMode(BuzzerPin, OUTPUT); // налаштування виводу номер 13 на вихід
 // ініціалізація і налаштування швидкості послідовної передачі даних
 Serial.begin(9600);
}

void loop() {
 // зчитування з аналогового входу, до якого підключено аналоговий вихід датчика
  int analogValue = analogRead(analogPin);
  Serial.println(analogValue); // передача отриманих значень в комп'ютер

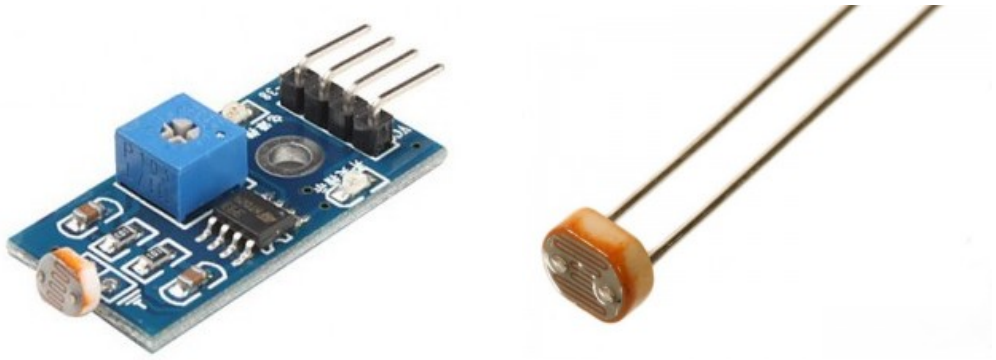
 if (analogValue > threshold)  // якщо виміряне значення перевищує задану межу
 {
   digitalWrite(BuzzerPin, HIGH); // вмикається звуковий сповіщувач
   Serial.print("High FLAME"); // в комп'ютер передається повідомлення "High FLAME"
 }
 // якщо значення на виході датчика лежить в межах від 200 до 400
 else if ((analogValue <= threshold) && (analogValue > threshold/2))
{
   Serial.print("Low FLAME"); // в комп'ютер передається повідомлення "Low FLAME"
   digitalWrite(BuzzerPin, HIGH); // вмикається звуковий сповіщувач
   delay(500);              // формування часової затримки 500 мс
   digitalWrite(BuzzerPin, LOW); // вимикається звуковий сповіщувач
 }
```

```
 else // якщо виміряне значення на виході датчика менше 200 апаратних одиниць
 {
   digitalWrite(BuzzerPin, LOW); // вимикається звуковий сповіщувач
   Serial.print("No flame");   // в комп'ютер передається повідомлення " No flame "
  }
  delay(1);       // формування часової затримки 1 мс
}
```

За допомогою Монітора послідовного порту, вбудованого в середовище розробки Arduino IDE, можна спостерігати за виміряними значеннями в режимі реального часу. Для запуску монітора порту потрібно в меню «Инструменты» вибрати пункт «Монитор порта», або натиснути комбінацію клавіш Ctrl+Shift+M.

2.10. Датчик освітленості

Модуль датчика освітленості, представлений на рис. 2.65, а, призначений для використання в автоматиці, яка керує включенням освітлення, або в приладах, що контролюють рівень освітленості. Світлочутливий елемент датчика – фоторезистор (рис. 2.65, б), який виступає за плату. При зміні освітленості від яскравого світла до темряви його опір змінюється від сотень Ом до декількох МОм. Це фіксує електроніка модуля і змінює стан виходів. Вихідний струм до 15 мА на контакті D0 (ТТЛ) дозволяє використовувати датчик освітленості спільно з модулем реле, який має на вході елементи управління реле і не потребують великих вхідних струмів керування.



а б

Рис. 2.65.

Модуль датчика освітленості має 4 виводи:

– VCC – живлення.

– GND – загальний провідник.

– D0 TTL – цифровий вихідний сигнал. Якщо рівень освітленості фоторезистора вище встановленої межі, то формується лог. 0.

– A0 – аналоговий вихід. Напруга на виході A0 пропорційна освітленості фоторезистора.

На рис. 2.66 представлена принципова схема модуля.

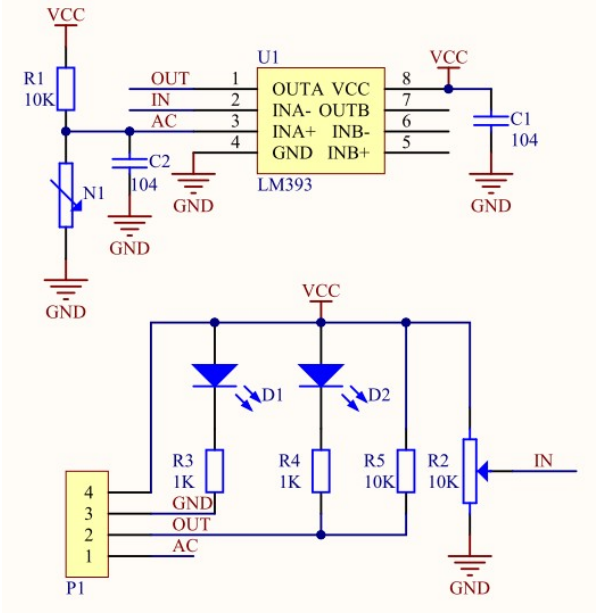


Рис. 2.66.

Світлодіод D1 (червоного кольору) використовується для індикації наявності напруги живлення модуля. При низькій освітленості світлодіод D2 (зеленого кольору) вимикається.

Фоторезистор на схемі не показаний. Основа модуля мікросхема компаратор LM393. Компаратор U1 порівнює рівні напруги на входах INA- і INA+. Встановлення межі спрацьовування проводиться змінним резистором N1. Залежно від результату порівняння на виході OUTA мікросхеми U1 формується логічний рівень, що поступає на вивід OUT з'єднувача P1.

При з'єднанні аналогового виходу A0 з входом АЦП мікроконтролера, під керуванням програми стає можливим контролювати роботу різних джерел світла. Наприклад, таке застосування можливе в парогенераторних котлах, газових нагрівачах води та інших приладах мають відкрите полум’я. Фотодатчик сприймає світло від полум’я при управлінні електропідпалюванням газу і контролює наявність полум'я.

Наступний приклад демонструє використання датчика освітленості для керування яскравістю світлодіода. Принципова схема пристрою представлена на рис. 2.67.

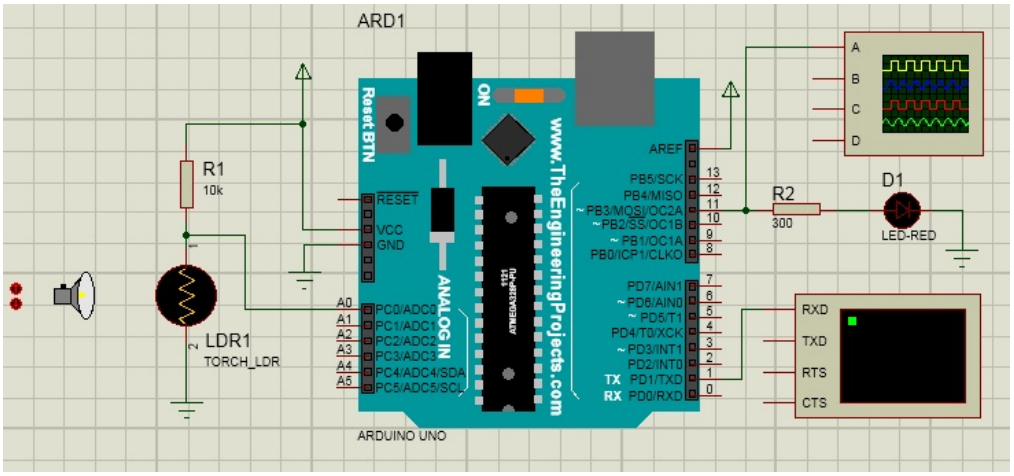


Рис. 2.67.

У цьому скетчі відбувається зчитування аналогового значення з виходу датчика освітленості для керування яскравістю світлодіода. Чим менша освітленість датчика, тим яскравіше буде світити світлодіод. Текст програми представлено нижче.

```
int photocellPin = A0; // датчик освітленості підключений до виходу A0 контролера
int photocellReading; // оголошення змінної для зчитування в неї даних з датчика
int LEDpin = 11; // номер виводу, до якого підключений світлодіод
int LEDbrightness; // оголошення змінної, в якій зберігатиметься значення яскравості
void setup(void) {
// налаштування швидкості передачі даних по послідовному інтерфейсу
// інформація передається з контролера в комп’ютер на швидкості 9600 біт/с
Serial.begin(9600);
}

void loop(void) {
photocellReading = analogRead(photocellPin); // зчитування даних з виходу датчика
Serial.print("Analog reading = "); // відправлення текстового рядка в комп’ютер
Serial.println(photocellReading); // відправлення виміряних значень в комп’ютер
// світлодіод світить яскравіше якщо рівень освітленості на датчику зменшується
// це означає, що потрібно інвертувати зчитане значення від 0-1023 до 1023-0
photocellReading = 1023 - photocellReading;
// тепер потрібно перетворити діапазон 0-1023 в 0-255, оскільки саме такий діапазон
// використовує функція analogWrite
LEDbrightness = map(photocellReading, 0, 1023, 0, 255);
analogWrite(LEDpin, LEDbrightness); // встановлення значення яскравості світлодіода
delay(100);     // часова затримка 100 мс
}
```

Після компіляції програми і запуску схеми на моделювання у вікні віртуального монітора можна бачити інформацію про поточне виміряне значення освітленості, а за допомогою осцилографа спостерігати ШІМ-сигнал керування яскравістю світлодіода. На рис. 2.68 представлена схема в режимі моделювання.

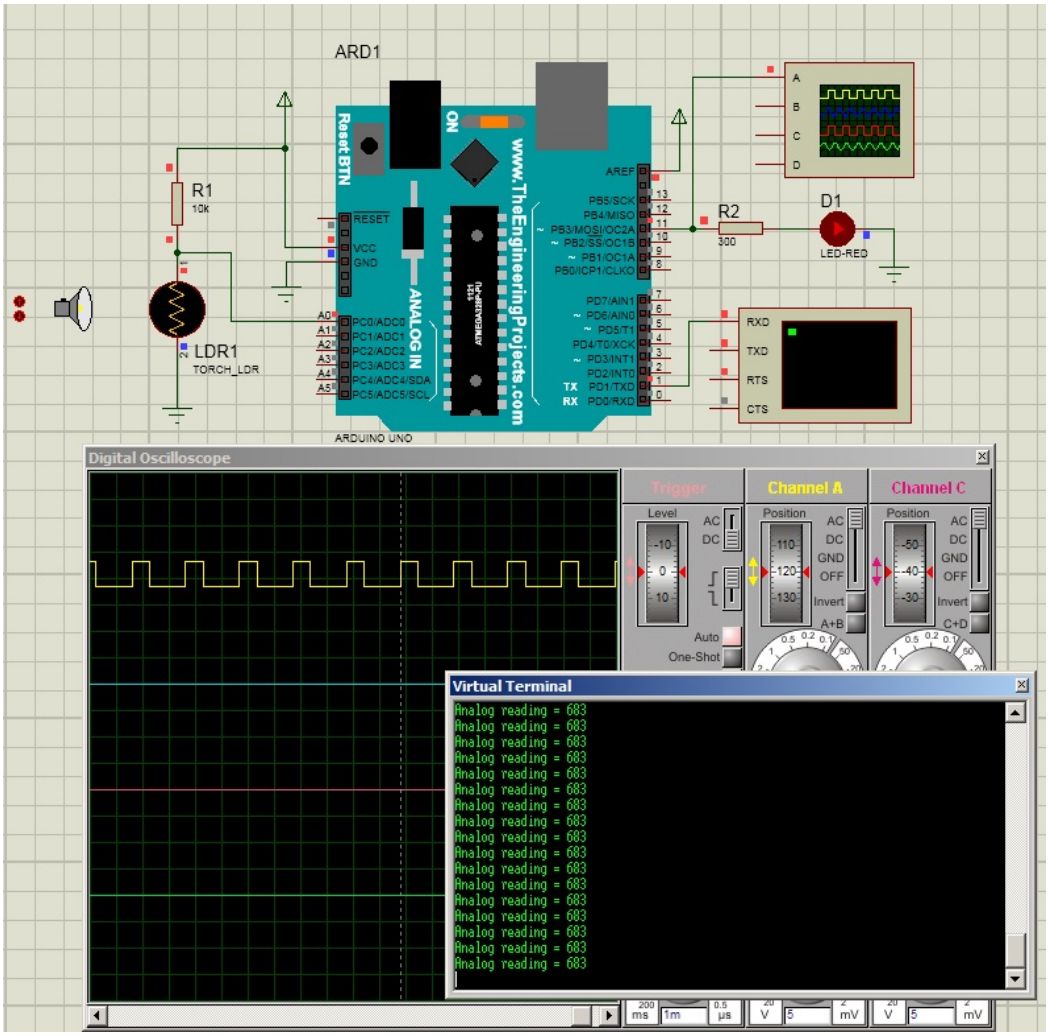


Рис. 2.68.

Фоторезистори досить активно застосовуються в найрізноманітніших системах. Найпоширеніший варіант застосування – ліхтарі вуличного освітлення. Якщо на вулиці темнішає, то вогні включаються автоматично. з Використовуючи фоторезистор можна зробити економну лампочку для будинку, що включається не за розкладом, а в залежності від освітлення. На базі датчика освітленості можна зробити навіть охоронну систему, яка буде спрацьовувати одразу після того, як закриту шафу або сейф відкрили і на датчик падає світло. На рис. 2.69 показана схема підключення фоторезистора до контролера Arduino. В прикладі автоматичного керування освітленням, що розглядається, опір фоторезистора складає 20 кОм, опір резистора на схемі складає 100 кОм.

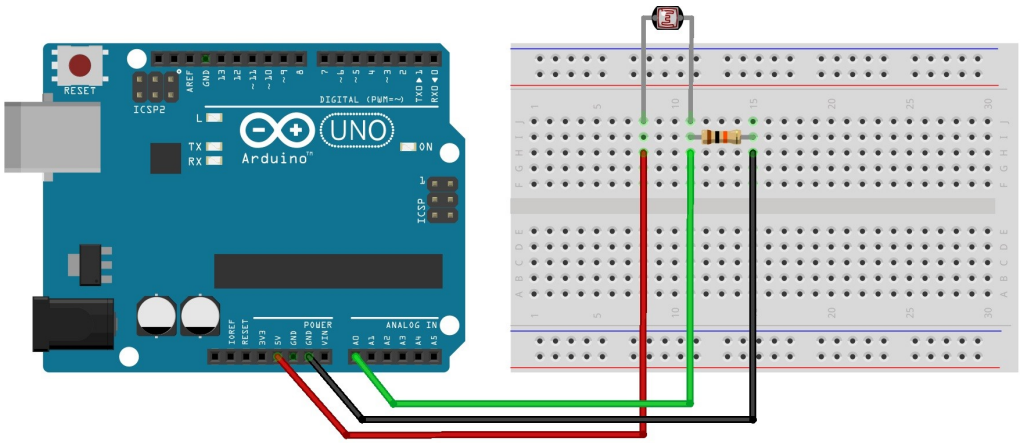


Рис. 2.69.

В схемі підключення датчик реалізований як дільник напруги. Опір одного плеча змінюється від рівня освітленості, з другого – подається напруга на аналоговий вхід A0. У мікросхемі контролера це напруга перетвориться в цифрові дані через АЦП. Оскільки опір датчика при попаданні на нього світла зменшується, то і значення напруги, що поступає в контролер Arduino, буде зменшуватися зі збільшенням освітленості. Залежно від того, в якому плечі дільника поставити фоторезистор, на аналоговий вхід буде подаватися або підвищена, або зменшена напруга. У тому випадку, якщо одна нога фоторезистора підключена до землі, то максимальне значення напруги буде відповідати темряві (опір фоторезистора максимальний, майже вся напруга падає на ньому), а мінімальне – гарному освітленню (опір близький до нуля, напруга мінімальна). Джерелом живлення в схемі є контролер Arduino. Фоторезистор приєднується однією ногою до землі, інша підключається до АЦП плати (в даному прикладі – A0).

Сам монтаж плати не повинен викликати труднощів. Оскільки фоторезистор не має полярності, підключити його можна будь-якою стороною, під'єднавши проводами за допомогою монтажної плати.

На рис. 2.70 представлено схему пристрою для автоматичного керування освітленням в приміщенні.

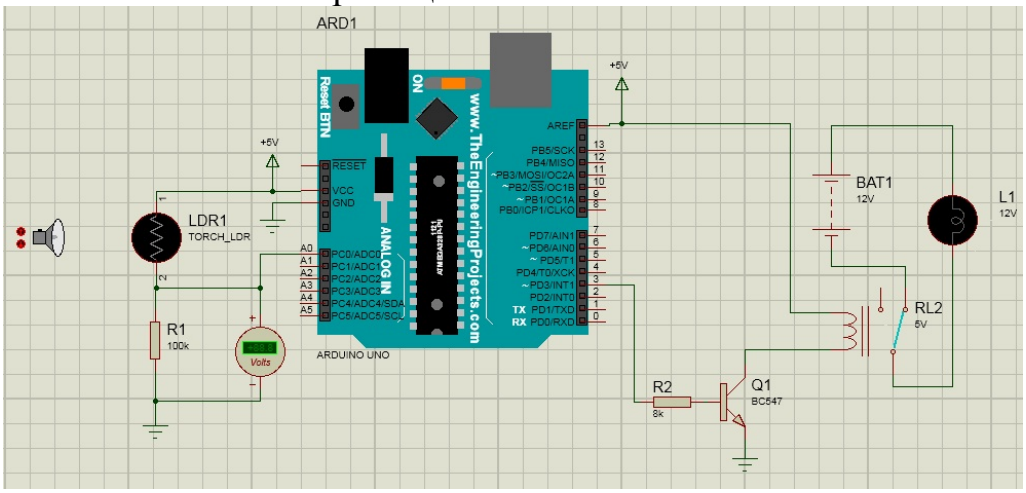


Рис. 2.70.

В схемі в якості датчика освітленості використовується елемент TORCH_LDR. На рис. 2.71 показано вікно додавання компонента в схему в середовищі Proteus, в якому показаний цей елемент.

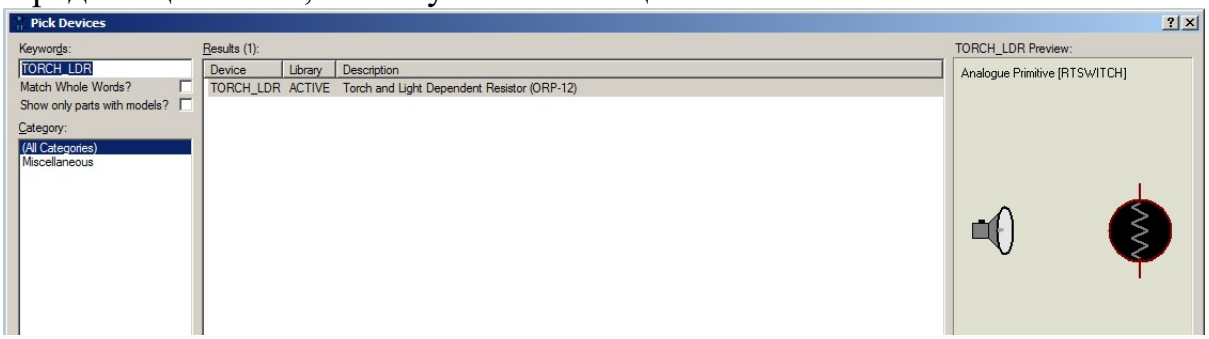


Рис. 2.71.

Також в схемі використано лампу розжарювання LAMP (рис. 2.72), реле QUAZ-SH-105D, транзистор BC547, джерело живлення BATTERY та струмообмежуючі резистори RES.

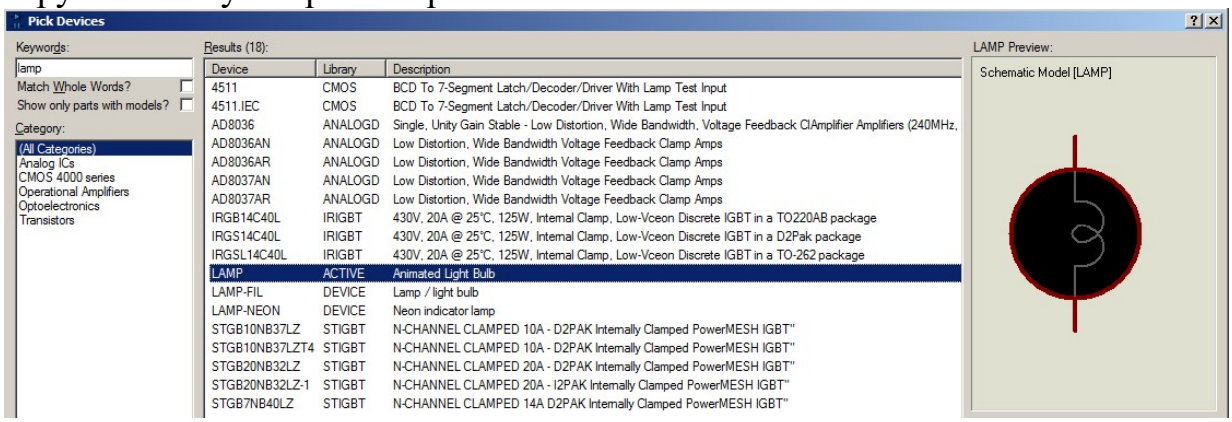


Рис. 2.72.

Написати скетч для датчика освітленості досить просто. Потрібно тільки зняти поточне значення напруги з того аналогового входу, до якого підключений датчик. Робиться це за допомогою відомої функції analogRead(). Потім можна виконувати якісь дії, в залежності від рівня освітленості. В даному випадку відбувається зміна стану цифрового виходу, що керує реле. В свою чергу, реле виконує керування станом лампи освітлення. Текст програми представлено нижче.

```
#define PIN_LED 13 // присвоюємо ідентифікатору PIN_LED число 13 – це номер виводу
// контролера, до якого підключений світлодіод, розташований на платі Arduino. В такому випадку використання вбудованого світлодіода допомагає виявити проблему – в
// апаратній частині, або в програмі узнати  в чому проблема (наприклад, якщо реле не
// перемикається при зміні освітлення). Світлодіод повинен світитися разом з лампою.

#define RELAY_PIN 3 // присвоюємо ідентифікатору RELAY_PIN число 3 – номер
// цифрового (дискретного) виводу контролера, за допомогою якого відбувається керування
// реле
```

```
#define PIN_PHOTO_SENSOR A0 // присвоюємо ідентифікатору PIN_PHOTO_SENSOR
// константу A0 – номер аналогового входу, до якого підключено датчик освітленості

void setup() {          // функція, в якій виконується початкове налаштування контролера
                        // Arduino. Ця функція виконується 1 раз при вмиканні живлення
                плати.
  Serial.begin(9600);  // налаштування швидкості передавання даних 9600 біт/с по
                        // послідовному інтерфейсу
// налаштування виводу номер 13, до якого підключений світлодіод, на вихід
pinMode(PIN_LED, OUTPUT);
// налаштування виводу номер 3, який керує реле, на вихід
pinMode(RELAY_PIN, OUTPUT);
}

// в цій функції, яка виконується циклічно, виконується реалізація алгоритму програми
void loop() {
int val; // оголошення змінної для зберігання значення, зчитаного з АЦП
// зчитування значення з аналогового входу, яке пропорційне освітленості, та запис
// зчитаного значення в змінну val
val = analogRead(PIN_PHOTO_SENSOR);
Serial.println(val);      // передавання зчитаного значення освітленості в комп'ютер
if (val > 700) {          // якщо зчитане значення більше 700, що означає високу
освітленість
// на виводі номер 3, що керує реле, встановлюється рівень лог.1, реле спрацьовує і
ланцюг,
// в який ввімкнено лампу, розривається, тобто лампа гасне
digitalWrite(RELAY_PIN, HIGH);
// на виводі номер 13, до якого підключений світлодіод, встановлюється рівень лог. 0
digitalWrite(PIN_LED, LOW);
} else {          // якщо зчитане значення в змінній val менше 700
// на виводі номер 3, що керує реле, встановлюється рівень лог.0, реле розмикається і
ланцюг,
// в який ввімкнено лампу,замикається, тобто лампа загоряється
digitalWrite(RELAY_PIN, LOW);
// на виводі номер 13, до якого підключений світлодіод, встановлюється рівень лог. 1,
тобто
// світлодіод вмикається
digitalWrite(PIN_LED, HIGH);
}
}
```

Отже, алгоритм роботи програми наступний:

– визначаємо рівень сигналу з аналогового входу A0;

– порівнюємо рівень з граничним значенням. Максимальне значення напруги на вході A0, і, відповідно, зчитаного з АЦП значення, буде відповідати максимальній освітленості, мінімальне – темряві. Граничне значення в даному прикладі вибрано рівним 700.

– якщо рівень освітленості більше граничного – темно, потрібно включати світлодіод та реле.

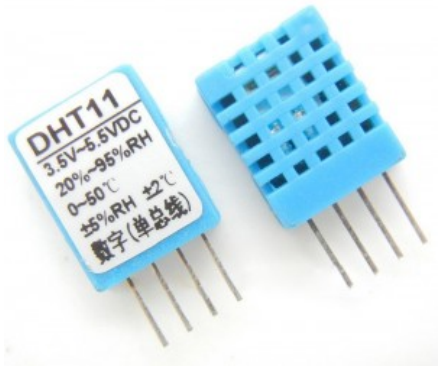
– інакше – світлодіод і реле вимикаються.

Прикриваючи фоторезистор (руками або світлонепроникним предметом), можна спостерігати включення і виключення світлодіода. Змінюючи в коді граничний параметр, можна змушувати вмикатися/вимикатися лампу при різному рівні освітлення.

Основним недоліком фоторезисторів є чутливість до спектру. Залежно від довжини хвилі падаючого світла опір може змінюватися на кілька порядків. До недоліків також відноситься низька швидкість реакції на зміну освітленості. Якщо освітленість не постійна, датчик може не встигати реагувати. Якщо ж частота зміни досить велика – фоторезистор взагалі перестане «бачити», що освітленість змінюється. До переваг датчика можна віднести простоту і доступність. Пряма залежність опору від інтенсивності світла, що падає на нього, дозволяє спростити електричну схему підключення.

2.11. Датчик вологості та температури DHT11

Датчики DHT складаються з двох основних частин: ємнісний датчик вологості і термістор. Також в корпусі встановлена мікросхема-перетворювач аналогового сигналу в цифровий. Конструктивно датчик може бути виконаний в пластиковому корпусі (рис. 2.73, а), або розташований на платі (рис. 2.73, б).



а



б

Рис. 2.73

Характеристики датчика:

– напруга живлення – 3-5 В;

– споживаний струм – 2,5 мА;

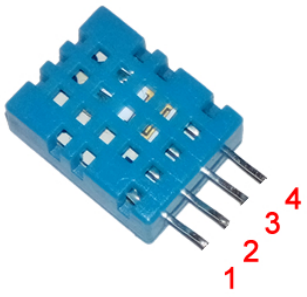
– діапазон вимірювань вологості – 20-80%, при точності 5%;

– діапазон вимірювань температури – 0-50 °С, при точності ± 2 °С;

– частота опитування – 1 Гц (один раз в секунду).

Крім DHT-11 можна зустріти аналогічні датчики: DHT-22, DHT-33 і DHT-44. Відрізняються ці датчики більш високим рівнем точності і діапазоном вимірювань. Наприклад, у датчика DHT-22 діапазон вимірювань вологості становить від 0 до 100%, а температури від -40 до 125 °С.

DHT11 цифровий датчик температури і вологості є складовим датчиком, який містить калібрований цифровий вихідний сигнал з показаннями температури і вологості. Він має високу надійність і чудову довготривалу стабільність роботи. Датчик включає в себе резистивний сенсор вологості і компоненти NTC структури для вимірювання температури. На рис. 2.74 представлено розташування та призначення виводів датчика.



1 - Vcc 3 - NC
2 - Data 2 - GND

Рис. 2.74.

Вивід номер 3, позначений на рис. 2.74 як NC, не використовується. На рис. 2.75 представлена схема підключення датчика до контролера Arduino. Рекомендований номінал резистора 10 кОм, якщо відстань від датчика до Arduino невелика. Для відстані більше 20 метрів, рекомендується резистор номіналом 5,1 кОм.

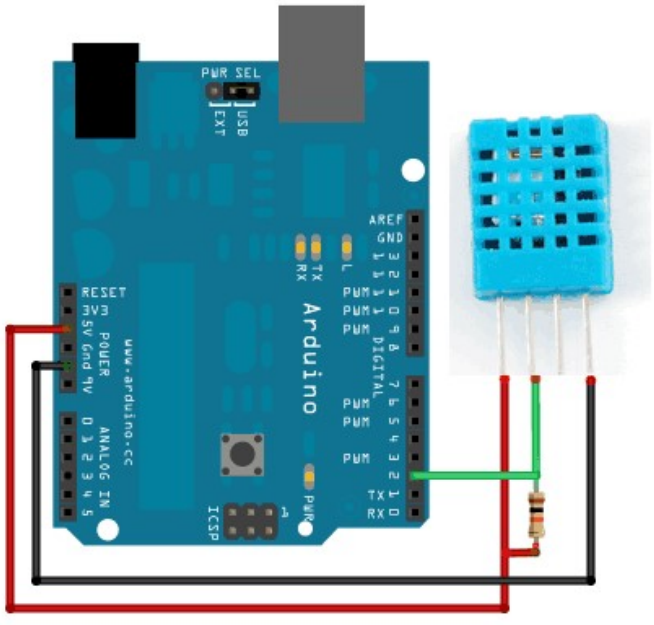


Рис. 2.75.

Для перевірки роботи програми в середовищі Proteus потрібно зібрати схему, представлену на рис. 2.76.

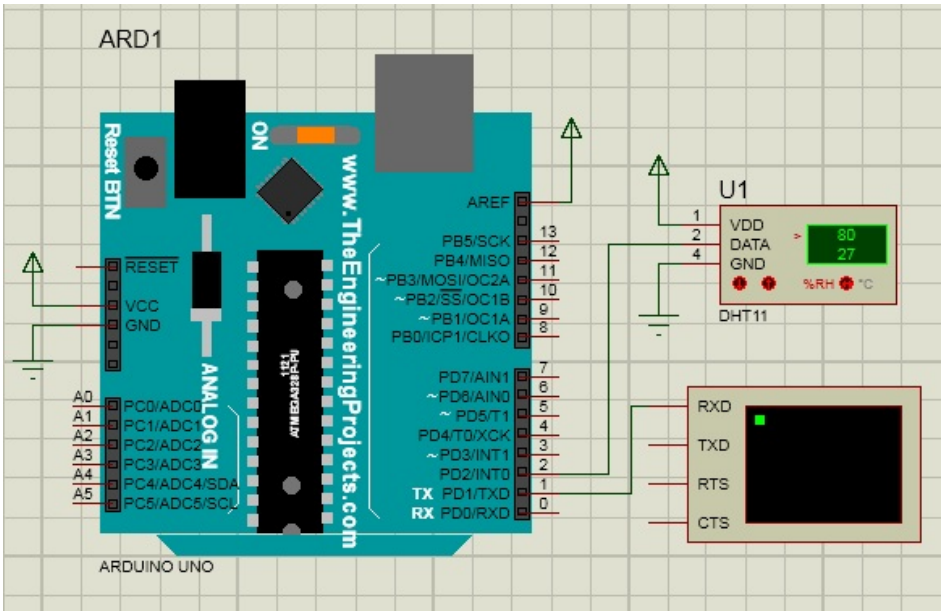


Рис. 2.76.

В наступній програмі відбувається зчитування даних з датчика та передача інформації про вологість та температуру в комп'ютер. В моделі схеми в Proteus у вікні терміналу можна побачити інформацію про виміряні значення. При роботі з реальним пристрої можна використовувати монітор порту, вбудований в середовище Arduino IDE.

Перше, що слід зробити перед початком розробки програми – встановити в Arduino IDE додаткову бібліотеку. Існує кілька бібліотек для роботи з DHT, але в даному прикладі використовується варіант бібліотеки з порталу adafruit. Завантажити архів з бібліотекою можна за наступним посиланням:

https://github.com/adafruit/DHT-sensor-library/archive/master.zip

Встановити бібліотеку можна безпосередньо з середовища Arduino IDE, використовуючи менеджер бібліотек (рис. 2.77).

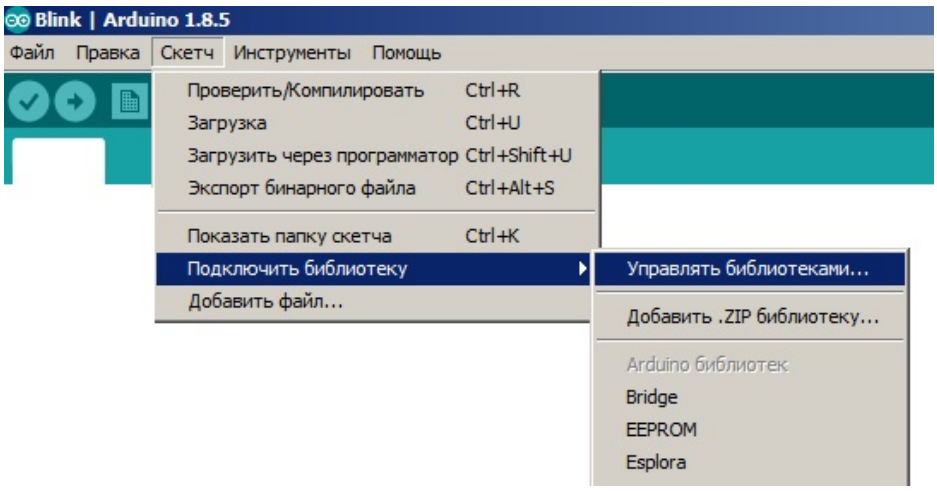


Рис. 2.77.

В менеджері бібліотек, вікно якого представлене на рис. 2.78, потрібно зі списку Тема вибрати категорію Сенсори, та ввести назву датчика DHT11. Вибравши версію бібліотеки, наприклад 1.2.3, потрібно натиснути на кнопку Установка. Всі необхідні файли бібліотеки будуть завантажені з Інтернет та доступні для використання в середовищі при розробці програми.

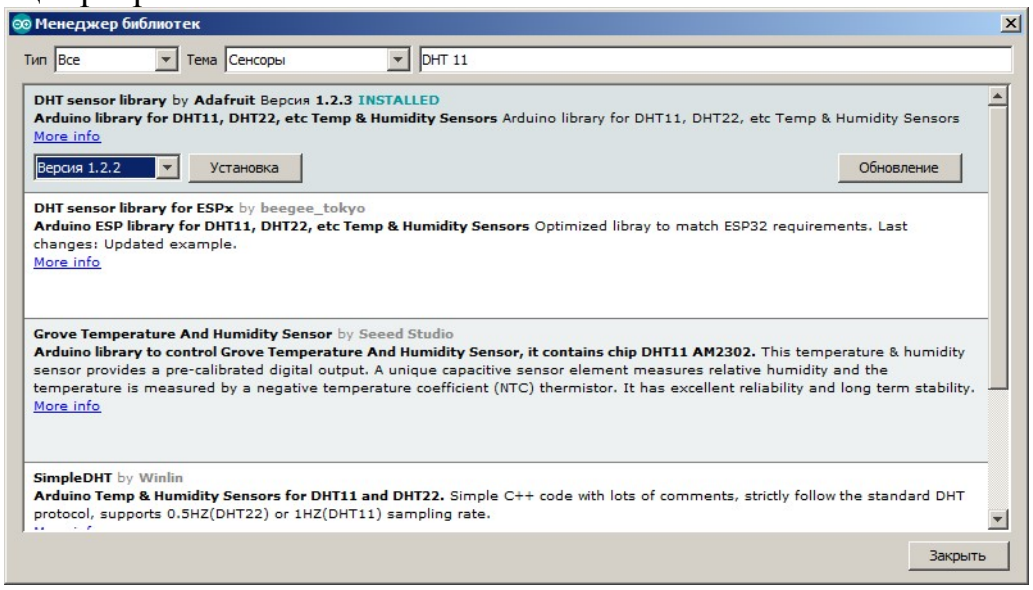


Рис. 2.78.

Текст програми представлено нижче.

```
#include "DHT.h" // підключення заголовочного файлу бібліотеки
const byte dhtPin = 2; // номер виводу, до якого підключений датчик
DHT dht(dhtPin, DHT11); // створення об’єкта для роботи з датчиком

void setup()
{
   Serial.begin(9600); // налаштування швидкості передачі даних в комп’ютер
   Serial.println("Test DHT11");  // відправлення повідомлення в комп’ютер
   dht.begin(); // ініціалізація процесу отримання даних з датчика
}
```

```
void loop()
{
   float h = dht.readHumidity(); // зчитування значення вологості
   float t = dht.readTemperature(); // зчитування значення температури

   if (isnan(t) || isnan(h))  // перевірка, чи отримані числа
   {
       // якщо ні, передача в комп’ютер відповідного повідомлення
      Serial.println("Failed to read from DHT");     }
  else // якщо отримані коректні числові дані
   {
      Serial.print("Humidity: ");  // передавання рядка "Humidity: " в комп’ютер
      Serial.print(h); // передавання значення вологості
      // передавання символу табуляції для зручного сприйняття інформації
      Serial.print(" %\t");
      Serial.print("Temperature: "); // передавання рядка "Temperature: " в комп’ютер
      Serial.print(t); // передавання значення температури
      Serial.println(" C"); // передавання символу «C» для позначення одиниць виміру
   }
}
```

Після компіляції програми і запуску процесу моделювання роботи схеми, у вікні терміналу можна побачити результати вимірювання вологості та температури (рис. 2.79).

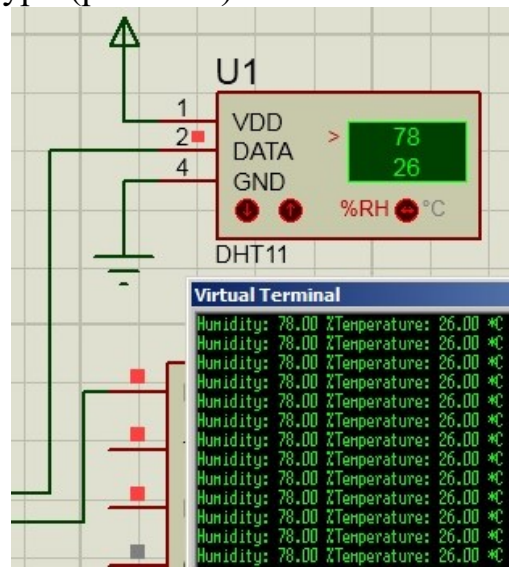


Рис. 2.79.

Запустивши вікно Монітора порта середовища розробки Arduino IDE в режимі реального часу можна спостерігати за значеннями вологості та температури у приміщенні.

2.12. Датчики швидкості обертання FC-03

Контролер (датчик) положення обертового об'єкта або по-іншому енкодер – це електромеханічний пристрій, за допомогою якого можна визначити положення вісі (вала), що обертається. У цьому пристрої

механічний рух перетворюється в електричні сигнали, що визначають положення об'єкта, дають інформацію про кут повороту вала, його положення і напрямок обертання. За допомогою енкодера також можна виміряти довжину і відстань.

Енкодери мають широку сферу застосування в промисловості, металообробці, ліфтах, автоматах для фасування, пакування і розливу, в випробувальних стендах, а також в роботах й інших машинах, які потребують точної реєстрації показників руху частин. Вони практично повністю замінили широко поширені раніше сельсини.

Інкрементальний енкодер – це пристрій, який визначає кут повороту обертового об'єкта, видаючи імпульсний цифровий код. Використовується для визначення швидкості обертання валу (вісі), коли немає потреби зберігати абсолютне кутове положення при відключенні живлення. Тобто, якщо вал нерухомий, передача імпульсів припиняється. Іншими словами, якщо включити енкодер цього типу, то відлік повороту кута почнеться з нуля, а не з кута на який він був виставлений до моменту виключення. Вісі об'єкта і енкодера з'єднуються між собою за допомогою спеціальної гнучкої перехідної муфти або жорсткої втулки, або енкодер може встановлюватися власне на сам вал. Основною перевагою інкрементальних енкодерів є їх простота, надійність і відносно низька вартість.

Конструкція оптичного енкодера (рис. 2.80) складається із спеціального оптичного диска, світловипромінюючого діода і фотодетектора. Диск з нанесеною оптичною шкалою (поверхня диска складається з прозорих і непрозорих ділянок) жорстко закріплюється на валу. При обертанні об'єкта спеціальний датчик зчитує інформацію і перетворює її в імпульси.

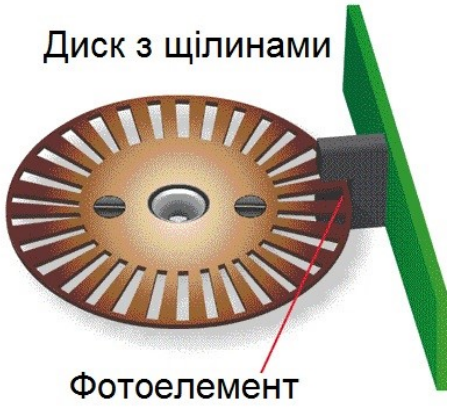


Рис. 2.80.

На рис. 2.81. представлена базова схема оптичного детектування руху: а – оптичний датчик; б – лінійний переривач; в – датчик кутової швидкості; г – аналог переривника.

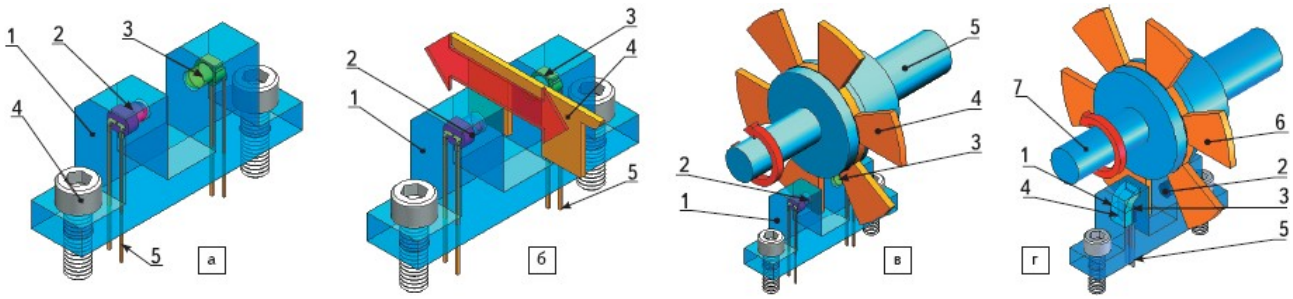


Рис. 2.81.

На рис. 2.81, а, позначені: 1 – корпус датчика; 2 – світлодіод; 3 – фоточуттєвий елемент (фототранзистор або фотодіод); 4 – елементи для монтажа корпуса; 5 – термінали для монтажу на друкованій платі.

На рис. 2.81, б, позначені: 1 – корпус датчика; 2 – світлодіод; 3 – фоточуттєвий елемент (фототранзистор або фотодіод); 4 – лінійний ротор; 5 – термінали для монтажа на друкованій платі.

На рис. 2.81, в, позначені: 1 – корпус датчика; 2 – світлодіод; 3 – фоточуттєвий елемент (фототранзистор або фотодіод); 4 – ротор, що обертається; 5 – вал, що обертається.

На рис. 2.81, г, позначені: 1 – корпус датчика; 2 – магніт; 3 –датчик Холла; 4 – магнітопровід; 5 – термінали для монтажу на друкованій платі; 6 –ротор, що обертається (крильчатка з лопастями з феромагнітного матеріала).

Рис. 2.81, а-в, демонструє принцип роботи типового щілинного оптичного перемикача-оптопереривача, або оптрона. У пластиковому корпусі розташовані світлодіод і фотодетектор, наприклад фототранзистор, розділені повітряним зазором в тілі корпусу. Світло від світлодіода потрапляє на фототранзистор, який переходить в стан насичення. Якщо в зазорі з'являється непрозорий елемент ротора, шлях світла між світлодіодом і фототранзистором блокується, що викликає перемикання виходу фототранзистора до високого рівня.

Випромінюючі світлодіоди можуть функціонувати як у видимому, так і в інфрачервоному спектрі. Для роботи датчика необхідно, щоб конструкція корпусу і ширина елементів ротора забезпечували чергування світлопередачі і перекриття оптичного каналу. Суттєво зменшити ширину оптичних кодових елементів і підвищити здатність пристрою дозволяють лазерні діоди. Відстань від датчика до об'єкта, який детектується, в межах ширини робочого зазору несуттєва, але якщо необхідний більший робочий діапазон, також застосовуються лазерні світлодіоди.

Датчик швидкості, представлений на рис. 2.82, на основі фотоелектричного елемента, який здатний фіксувати положення муфти колеса або електродвигуна, дає можливість керувати швидкістю обертання двигуна або колеса.

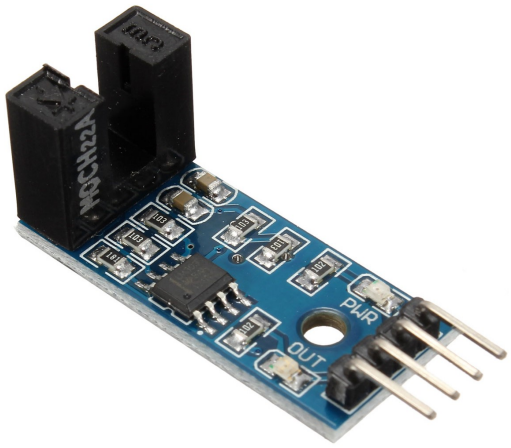


Рис. 2.82.

Принципова схема інфрачервоного датчика швидкості представлена на рис. 2.83.

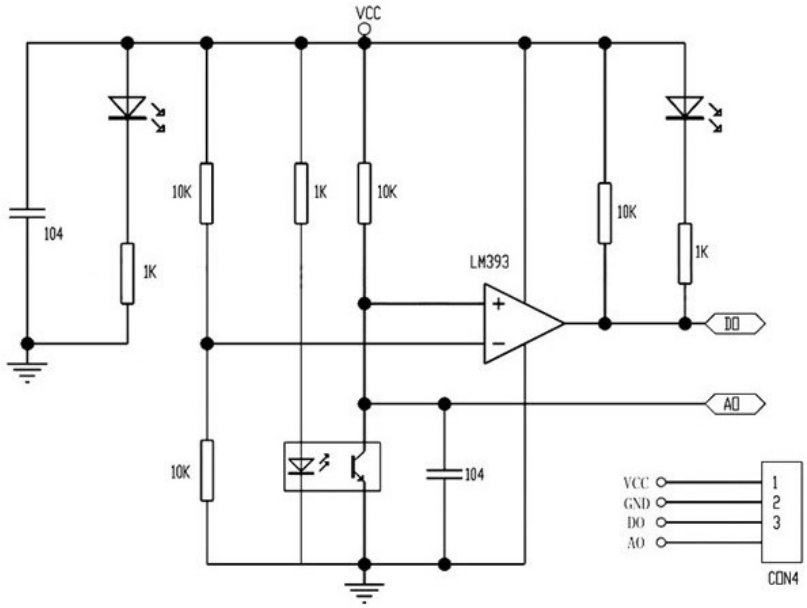


Рис. 2.83.

Опис виводів датчика:

VCC: напруга живлення 3,3-5 В;

GND: загальний провід;

DO: вихідний імпульсний сигнал, частота якого пропорційна обертам вала;

AO: аналоговий сигнал з фототранзистора.

Характеристики інфрачервоного датчика швидкості:

Тип датчика: щілинний;

Ширина паза датчика: 5 мм;

Компаратор: LM393;

Робоча напруга: 3,3...5В;

Індикатор стану виходу;

Вихідний формат: цифровий вихід (0 або 1);

Розмір пристрою: 39×15×13 мм.

При виборі енкодера слід звернути увагу на наступні параметри:

– число імпульсів на оборот (число біт у абсолютних енкодерів). Від даного показника залежить точність системи – чим більше імпульсів тим вище точність;

– вал, отвір під вал (і їх діаметр). Від цього залежить, яким чином на енкодер передаватиметься обертання, або об'єкт буде приєднуватися до отвору енкодера, або на вал енкодера буде передаватися обертання за допомогою, наприклад, зубчастої передачі або ременя;

– тип вихідного сигналу енкодера (HTL, TTL, RS422, двійковий код, код Грея, та ін.). Даний параметр впливає на зняття сигналу енкодера і подальшу його передачу;

– напруга живлення. Від цього показника залежить робота системи і точність зняття сигналу.

– довжина кабелю або тип роз'єму впливають на можливості установки робочої системи;

– інші вимоги за кріпленням (необхідність муфти, монтажного фланця, штанги та ін.). Даний параметр впливає на стійкість установки і на точність системи;

– важлива також ступінь захисту енкодера від проникнення пилу і вологи.

2.13. Датчик вологості ґрунту YL-38

На рис. 2.84 представлено принципову схему пристрою, що контролює вологість ґрунту, вологість повітря та температуру повітря, виводить інформацію про вказані параметри на екран алфавітно-цифрового рідкокристалічного індикатора, а також передає їх у текстовому вигляді по інтерфейсу RS232 в комп’ютер (або радіомодем Bluetooth, наприклад HC-05). Також за допомогою світлодіодів мікропроцесорна система видає сигнал про те, що вологість ґрунту низька і треба полити рослину.

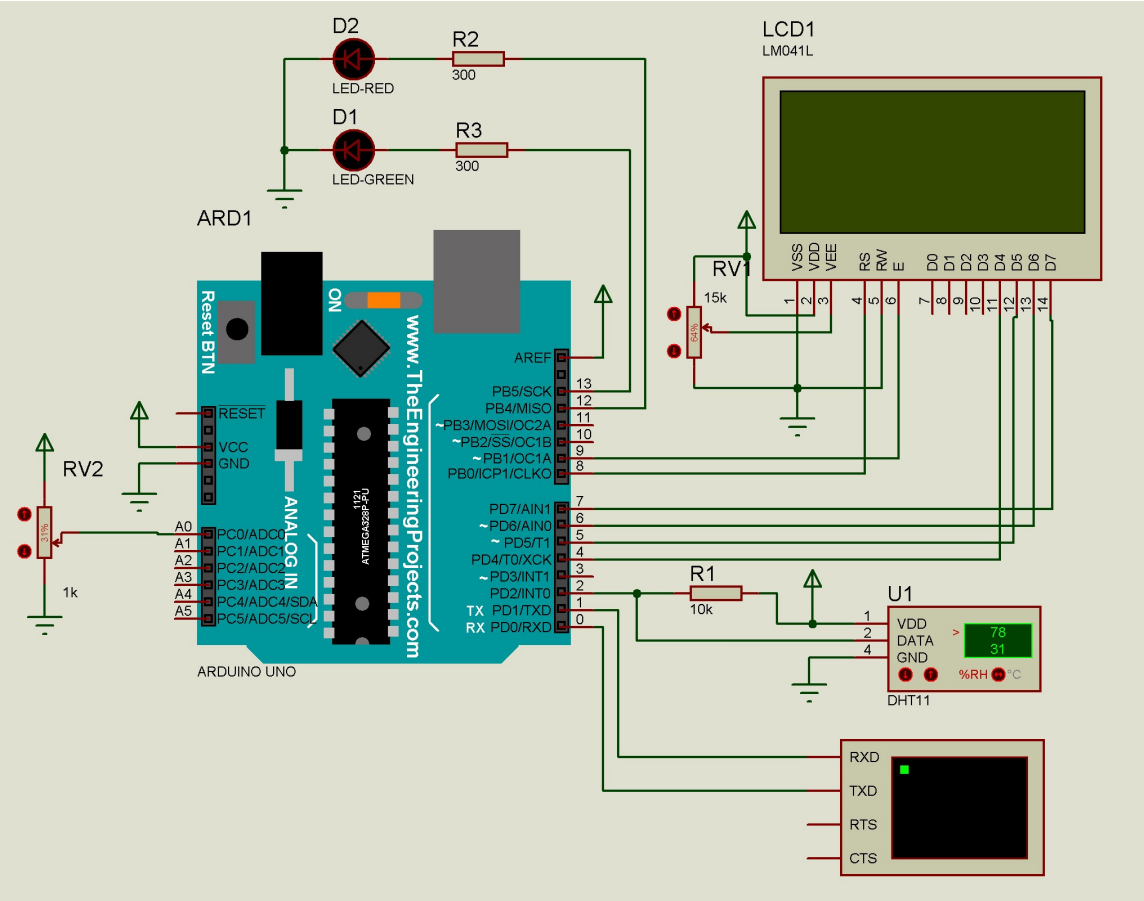


Рис. 2.84.

В даній схемі реальний датчик пологості ґрунту замінено резистором RV2 зі змінним опором, який підключений до аналогового входу A0 плати Arduino. З точки зору програмування немає ніякої різниці що саме є джерелом сигналу, оскільки з на виході датчика вологості також формується сигнал напруги в діапазоні від 0 до 5 В. Таким чином, в процесі моделювання роботи пристрою для налагодження програми іноді зручніше замінити реальний датчик еквівалентною схемою, як в даному прикладі. Для контролю вологості повітря та температури повітря використовується датчик DHT11, вихід якого підключено до виводу 2 контролера Arduino. Слід звернути увагу на те, що в схемі обов'язково потрібно підключити резистор з опором 10 кОм між лінією живлення та виходом датчика. Якщо цього не зробити, схема працювати не буде. Зазвичай цей опір вже встановлено на модулі, однак, на всяк випадок, потрібно переконатися в його наявності. Також для зручності використання системи в схемі використовуються 2 світлодіоди для інформування про рівень вологості ґрунту. Якщо ґрунт занадто сухий, то горить червоний світлодіод. Замість світлодіода може бути і насос. Якщо вологість ґрунту достатня, то горить зелений світлодіод. Рідкокристалічний дисплей, який дозволяє відображати виміряні параметри в зручному вигляді, підключений до контролера Arduino по чотирьохпровідній схемі (виводи D4-D7 рідкокристалічного індикатора підключено до виводів 4, 5, 6, 7 плати Arduino). Також для керування РКІ

виводи RS та E підключено до виводів 8 та 9 контролера. В реальному пристрої для керування контрастністю РКУ потрібно підключити резистор зі змінним опором, як показано на схемі. На рис. 2.85 представлено зовнішній вигляд датчика вологості ґрунту (гігрометр).

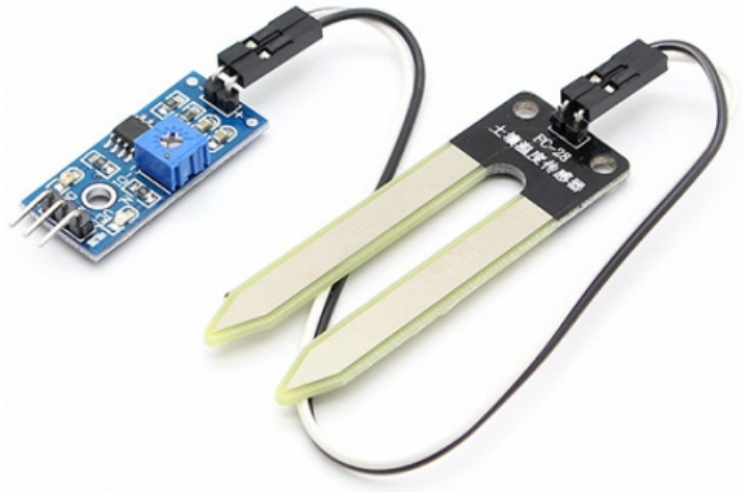


Рис. 2.85.

Датчик вологості ґрунту (гігрометр) для Arduino зібраний на мікросхемі LM393 і має окремий чутливий елемент і настройку чутливості. Між двома електродами датчика створюється невелика напруга. Якщо ґрунт сухий, опір великий і струм буде менше. Якщо земля волога – опір менше, і струм – більше. За підсумковим аналоговим сигналом можна судити про ступінь вологості. Датчик вологості ґрунту дає цифровому виході 1 або 0, залежно від того, чи достатньо вологий ґрунт. Для підключення живлення і мікроконтролера використовується 4-контактний штирьовий інтерфейс. Позначення контактів: VCC – напруга живлення; GND – загальний контакт; D0 – цифровий вихід; A0 – аналоговий вихід. На деяких модулях відсутній вихід аналогового сигналу. Чим вище вологість ґрунту, тим вище її електропровідність. Датчик занурюється в ґрунт на відстань до 45 мм і вимірює електропровідність ґрунту між своїми контактами. Напруга на виході датчика прямопропорційна рівню виміряної електропровідності. Чим вище вологість ґрунту, тим вище рівень сигналу на виході датчика. Залежність вихідної напруги датчика від вологості ґрунту представлено на рис. 2.86.

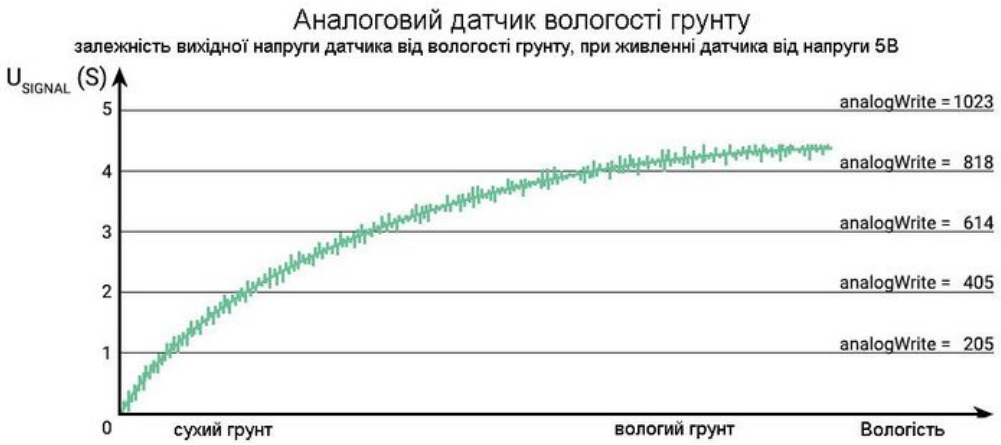


Рис. 2.86.

Показання датчика зчитуються викликом функції analogRead(номер_виходу); яка повертає число від 0 до 1023 (див. графік). На показання датчика впливають такі чинники:

– ступінь занурення датчика в ґрунт (чим глибше занурений датчик, тим вище показання датчика);

– тип ґрунту, його хімічні і фізичні властивості (чим щільніше ґрунт, тим вище показання датчика);

– наявність і кількість домішок у воді, якою поливається ґрунт.

Для контролю вологості та температури повітря в розглянутому прикладі використовується цифровий датчик DHT11. Передача даних здійснюється по одному дроту з використанням власного протоколу. Цей модуль складається з ємнісного датчика вологості і термістора. Також, датчик містить в собі АЦП для перетворення аналогових значень вологості і температури. На рис. 2.87 представлено зовнішній вигляд модуля DHT11.

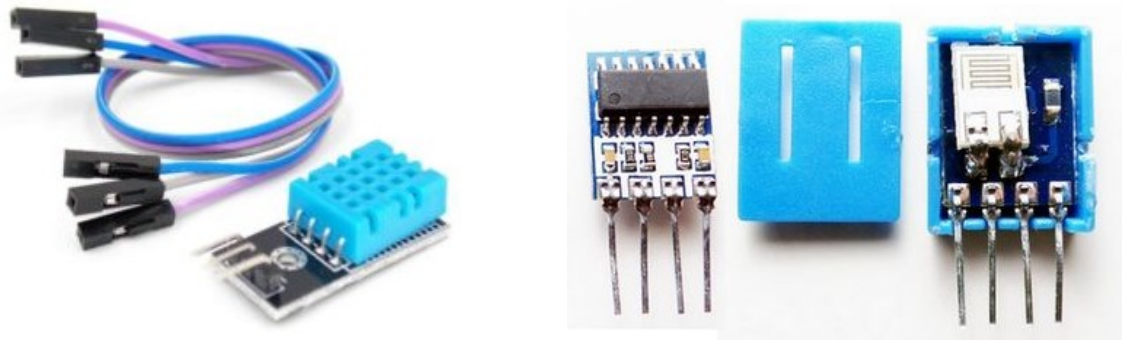


Рис. 2.87.

Для роботи з модулем DHT11 завантажити та підключити відповідну бібліотеку. Бібліотеку можна завантажити за посиланням: https://arduino.ua/docs/DHT-sensor-library.zip

Цей архів, в якому містяться файли бібліотеки для роботи з модулем DHT11, потрібно підключити до проекту. Для цього в середовищі Arduino IDE потрібно в меню «Скетч» вибрати підпункт «Подключить библиотеку», і у списку вибрати «Добавить .ZIP библиотеку», як показано на рис. 2.88.

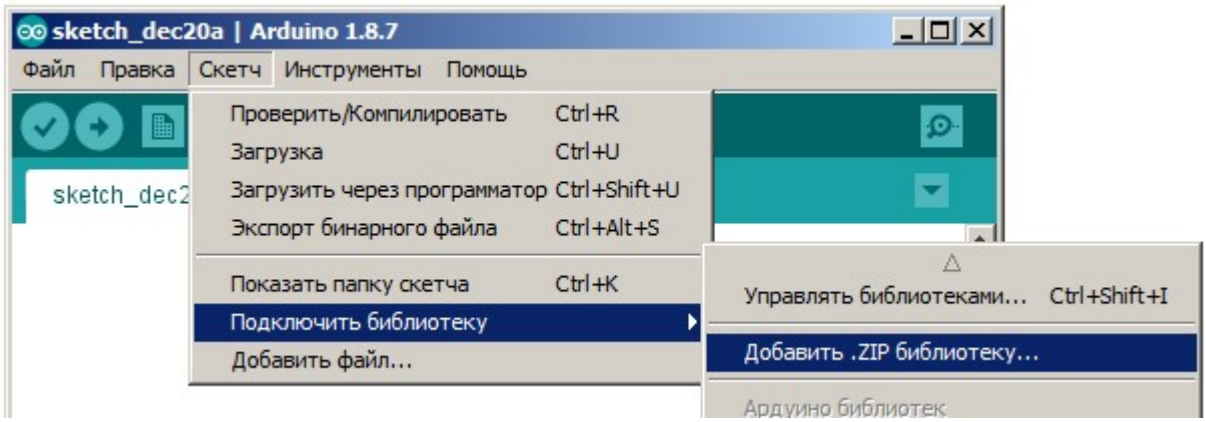


Рис. 2.88.

Після цього у вікні, що з'явиться, потрібно вказати шлях до архіву (файл DHT-sensor-library.zip), в якому містяться файли бібліотеки. Також потрібно підключити ще одну бібліотеку. В меню «Скетч» вибрати підпункт «Подключить библиотеку», і у списку вибрати «Управлять библиотеками» (рис. 2.89). У діалоговому вікні, що з'явиться, потрібно вибрати бібліотеку Adafruit Unified Sensor, і якщо вона не встановлена (відсутній напис INSTALLED), потрібно натиснути кнопку «Установить», яка буде розташована праворуч у куті.

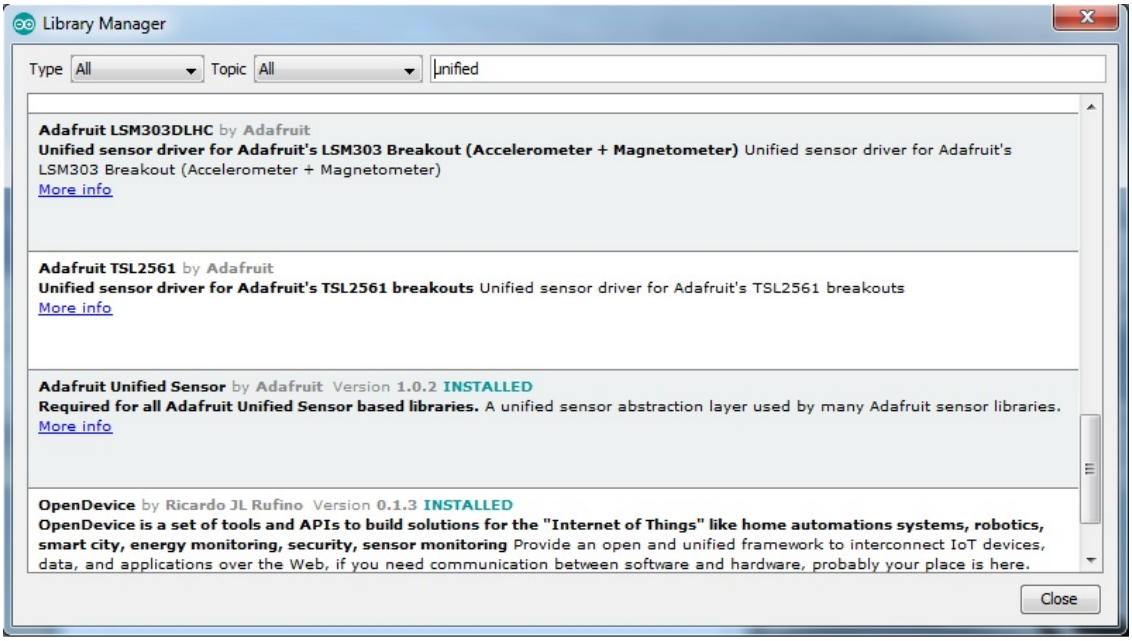


Рис. 2.89.

Після того, як бібліотеку підключено, можна писати текст програми, який представлено нижче.

```
// підключення бібліотеки для роботи з рідкокристалічним індикатором
#include <LiquidCrystal.h>
// підключення бібліотеки для роботи з модулем DHT11
#include "DHT.h"
```

```
#define DHTPIN 2     // ідентифікатору DHTPIN присвоюється значення 2 – номер виводу,
                     // до якого підключено датчик вологості та температури повітря DHT11

#define DHTTYPE DHT11 // ідентифікатору DHTTYPE присвоюється тип модуля DHT11

// створення об’єкта lcd класу LiquidCrystal для роботи з рідкокристалічним індикатором
// у дужках вказано номери виводів контролера, до яких підключено РКІ: RS – вивід 8
// контролера, E – вивід 9 контролера, D4 – вивід 4, D5 – вивід 5, D6 – вивід 6, D7 – вивід 7
LiquidCrystal lcd(8,9,4,5,6,7);

// створення об’єкта dht класу DHT для роботи з модулем DHT11
// DHTPIN – номер виводу контролера, до якого підключено вихід датчика (в даному
// прикладі 2), DHTTYPE – тип модуля (в даному прикладі DHT11)
DHT dht(DHTPIN, DHTTYPE);

// функція ініціалізації контролера; ця функція виконується один раз при вмиканні живлення
// контролера, і в ній виконується налаштування режимів роботи виводів контролера та
// інших вузлів, зокрема РКІ, модуля DHT11, комунікаційного вузла UART
void setup() {
  lcd.begin(16,4); // встановлення режиму відображення РКІ – 4 строки по 16 символів в кожній
  dht.begin();   // активація датчика температури та вологості повітря DHT11

  pinMode(12, OUTPUT);     // вивід 12 контролера, до якого підключено червоний світлодіод,
                           // налаштовується на вихід
  pinMode(13, OUTPUT);     // вивід 13 контролера, до якого підключено зелений світлодіод,
                           // налаштовується на вихід
  Serial.begin(9600);      // встановлення швидкості передавання даних по UART 9600 біт/с
}

// головна функція програми, в якій реалізовано алгоритм роботи;
// ця функція виконується циклічно
void loop() {
  // Зчитування температури та вологості повітря з датчика DHT11
  // кожна з цих дій триває протягом приблизно 250 мс
  float h = dht.readHumidity();          // зчитування значення вологості повітря у змінну h
  float t = dht.readTemperature();       // зчитування значення температури повітря у змінну t
  float sh=(1023-analogRead(0))/10.23; // зчитування значення вологості грунту з АЦП
  // та перетворення зчитанного значення з апаратних одиниць 0-1023  у відсотки 0-100%
  // перевірка, чи зчитані з датчика значення коректні
  // якщо значення не коректні, повертається значення NaN (not a number – не число)
```

```
  // що свідчить про те, що в процесі зчитування даних була помилка
  if (isnan(t) || isnan(h)) {
  // передавання текстового повідомлення по UART з інформацією про помилку
зчитування
    Serial.println("Failed to read from DHT");
  } else {  // якщо дані отримано і вони коректні, можна реалізувати логіку програми
    if(sh>60){                          // якщо вологість ґрунту перевищує 60%
     // вмикається зелений світлодіод шляхом встановлення лог.1 на виводі 13
    digitalWrite(13, HIGH);
    // вимикається червоний світлодіод шляхом встановлення лог.0 на виводі 12
    digitalWrite(12, LOW);
    }
    else{                               // якщо вологість ґрунту менше або дорівнює 60%
      // вимикається зелений світлодіод шляхом встановлення лог.0 на виводі 13
      digitalWrite(13, LOW);
      // вмикається червоний світлодіод шляхом встановлення лог.1 на виводі 12
      digitalWrite(12, HIGH);
    }

    lcd.setCursor(0,0); // установка курсора на екрані РКІ в перший рядок в початкову
позицію
    lcd.print("H(soil)=");        // виведення на екран РКІ текстового напису "H(soil)="
    lcd.print(sh);                // виведення на екран значення вологості ґрунту
    lcd.print(" %");      // виведення на екран символу «%» – одиниці вимірювання
вологості

    lcd.setCursor(0,2); // установка курсору на екрані РКІ в третій рядок в початкову
позицію
    lcd.print("H(air)=");         // виведення на екран РКІ текстового напису "H(air)="
    lcd.print(h);                 // виведення на екран значення вологості повітря
    lcd.print(" %");      // виведення на екран символу «%» – одиниці вимірювання
вологості

    lcd.setCursor(0,1); // установка курсора на екрані РКІ в другий рядок в початкову
позицію
    lcd.print("T(air)=");         // виведення на екран РКІ текстового напису "T(air)="
    lcd.print(t);                 // виведення на екран значення температури повітря
    lcd.print(" C");      // виведення на екран символу «C» – одиниці вимірювання
температури

    // передавання по послідовному інтерфейсу з використанням UART інформації про
    // виміряні параметри у текстовому вигляді
    Serial.print("Soil humidity: ");      // передавання строки "Soil humidity: " (вологість
грунту)
    Serial.print(sh);                     // передавання значення вологості ґрунту
    Serial.print(" %   ");                // передавання символу «%» – одиниці вимірювання
    Serial.print("Air humidity: ");       // передавання строки "Air humidity: " – вологість
повітря
    Serial.print(h);                      // передавання значення вологості повітря
    Serial.print(" %   ");                // передавання символу «%» – одиниці вимірювання
    Serial.print("Temperature: ");        // передавання строки "Temperature: " – температура
    Serial.print(t);                      // передавання значення температури повітря
```

```
  Serial.println(" C");                    // передавання символу «C» – одиниці вимірювання
}
// формування часової затримки 1000 мс
delay(1000);              // вимірювання параметрів відбувається один раз в секунду
}
```

Застосування датчика вологості ґрунту може бути найрізноманітнішим. Найбільш часто вони використовуються в системах автополиву і ручного поливу рослин: їх можна встановити в квіткових горщиках, якщо рослини чутливі до рівня води в ґрунті. Підключення до світлодіоду дозволить точно визначити, коли пора проводити полив. Вони незамінні для організації поливу рослин в теплиці. За аналогічним принципом також збираються датчики вологості повітря, які потрібні для запуску в роботу системи оприскування рослин. Все це дозволить автоматичним чином забезпечити полив рослин і нормальний рівень атмосферної вологості. На дачі використання датчиків дозволить не тримати в пам`яті час поливу кожної грядки, електроніка сама розповість про кількість води в ґрунті. Це дозволить не допустити надмірного поливу, якщо недавно пройшов дощ. Застосування датчиків дуже зручне і в деяких інших випадках. Наприклад, вони дозволять контролювати вологість ґрунту в підвалі і під будинком поблизу фундаменту. У квартирі його можна встановити під мийкою: якщо труба почне капати, про це відразу повідомить автоматика, і можна буде уникнути затоплення сусідів і подальшого ремонту. Простий пристрій датчика дозволить всього за кілька днів повністю обладнати системою оповіщення всі проблемні ділянки дому та саду. Якщо електроди досить довгі, з їх допомогою можна буде контролювати рівень води, наприклад, в штучному невеликому водоймищі.

2.14. Електронний компас GY271

Магнітометр являє собою пристрій для вимірювання інтенсивності однієї або декількох складових магнітного поля. Сьогодні ринок надає широкий вибір електронних компасів в інтегральному виконанні. Для більш повного розуміння принципу дії такого компаса розглянемо основні положення теорії магнетизму і принципи визначення напрямку вектора магнітного поля Землі.

Магнітне поле Землі в кожній точці простору характеризується вектором напруженості T, напрямок якого визначається трьома складовими по вісях X, Y і Z в прямокутній системі координат (рис. 2.90). Також магнітне поле Землі можна описати горизонтальною складовою напруженості H, магнітним схиленням D (кутом між H і площиною географічного меридіана) і магнітним нахилом I (кутом між T і площиною горизонту).

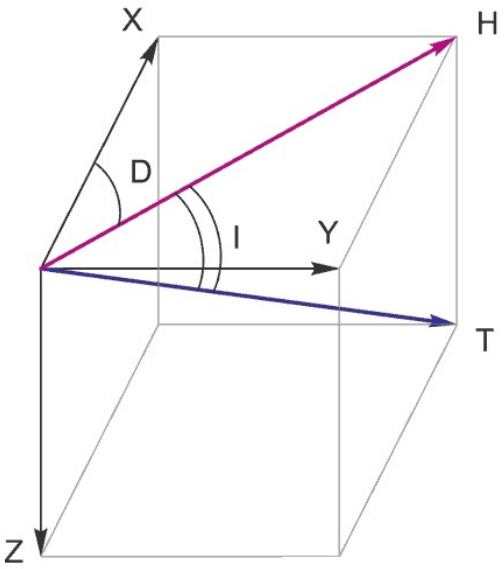


Рис. 2.90.

Основною характеристикою магнітного поля є магнітна індукція B, що представляє собою векторну величину. Напрямок вектора магнітної індукції збігається з напрямком сили, що діє на північний полюс магніту, розміщеного в даній точці магнітного поля. Величина B виражається одиницею виміру тесла (Тл або (Н/А·м)). Тесла є досить великою величиною магнітної індукції, тому для вимірювання слабких магнітних полів застосовують дрібну часткову одиницю – мікротесла (мкТл). Варто зауважити, що повний вектор магнітного поля Землі становить всього близько 50 мкТл. Але в документації на МЕМС-магнітометри зазвичай наводиться інша одиниця виміру, що характеризує магнітне поле – Гаусс (Гс). Гаусс є одиницею виміру магнітної індукції в системі СГС. При цьому справедливі наступні рівності:

$$1 \text{ Гс} = 100 \text{ мкТл},$$

$$1 \text{ Тл} = 10^4 \text{ Гс}.$$

Магнітна індукція пов'язана з напруженістю магнітного поля співвідношенням:

$$B=\mu\mu_0 H,$$

де μ – магнітна проникливість середовища, μ_0 – магнітна стала.

Виходячи з цього можна стверджувати, що $B \sim H$. В результаті, на практиці для визначення напрямку вектора магнітного поля Землі вимірюють дві його складові по вісі X та вісі Y (рис. 2.91), а потім розраховують кут φ на основі наступних формул:

$$tg\varphi = \frac{H_Y}{H_X} = \frac{H\sin\varphi}{H\cos\varphi},$$

$$\varphi = arctg(tg\varphi).$$

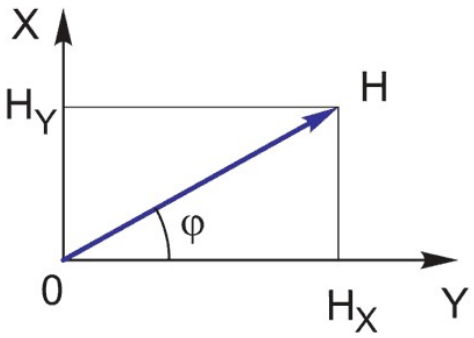


Рис. 2.91.

Розглянемо роботу датчика HMC5883L компанії Honeywell. Цей датчик (рис. 2.92) являє собою пристрій для вимірювання магнітного поля по вісях X, Y і Z. Він є типовим представником сімейства магнітометрів загального призначення, що застосовуються в мобільних телефонах, планшетах, автомобільних навігаційних системах, персональних навігаційних пристроях і інших електронних системах. Такі датчики за принципом своєї дії, методу зв'язку з керуючим пристроєм і навіть по структурі регістрів не сильно відрізняються один від одного. Для сполучення з мікроконтролером використовується стандартний I2C інтерфейс.

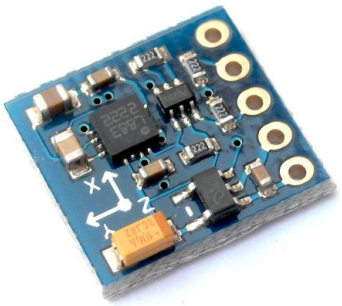


Рис. 2.92.

Принципова схема електронного компаса GY-271 представлена на рис. 2.93.

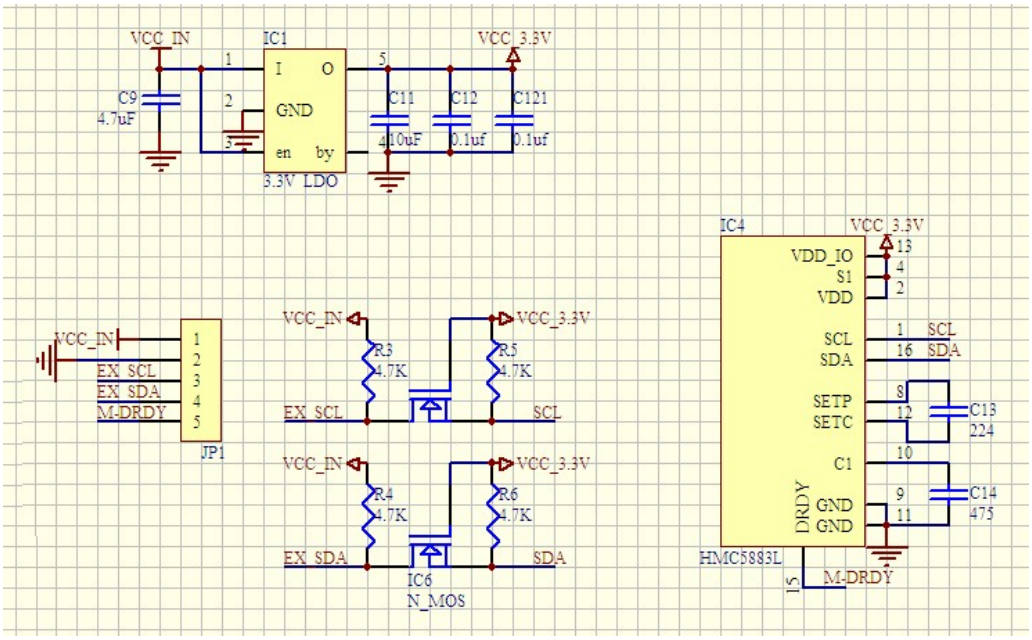


Рис. 2.93.

Як і будь-який пристрій, що працює по інтерфейсу I^2C, датчик має свою власну унікальну адресу для роботи. HMC5883L присвоєно адресу 0x1E. З восьмим бітом запису / зчитування адреса буде 0x3D для зчитування і 0x3C для запису. Карта регістрів датчика представлена в табл. 2.4.

Таблиця 2.4.

Адреса	Назва	Доступ
00	Регістр конфігурації A	Зчитування/Запис
01	Регістр конфігурації B	Зчитування/Запис
02	Регістр режиму	Зчитування/Запис
03	Старший байт виходу X	Зчитування
04	Молодший байт виходу X	Зчитування
05	Старший байт виходу Z	Зчитування
06	Молодший байт виходу Z	Зчитування
07	Старший байт виходу Y	Зчитування
08	Молодший байт виходу Y	Зчитування
09	Регістр статусу	Зчитування
10	Регістр ідентифікації A	Зчитування
11	Регістр ідентифікації B	Зчитування
12	Регістр ідентифікації C	Зчитування

Для налаштування світлового сенсора доступні регістри, що знаходяться за адресами 00 (configuration register A), 01 (configuration register B) і 02 (mode register).

Перший з доступних регістрів (регістр CRA) використовується для налаштування світлового сенсора (табл. 2.5). Кожен біт має своє ім'я для

того, щоб було видно, з яким ще бітом в регістрі він пов'язаний (наприклад, біти CRA6 і CRA5 мають загальне ім'я MA1 і MA0 відповідно, так як відповідають за один параметр).

Таблиця 2.5.

CRA7	CRA6	CRA5	CRA4	CRA3	CRA2	CRA1	CRA0
(0)	MA1(0)	MA0(0)	DO2(1)	DO1(0)	DO0(0)	MS1(0)	MS0(0)

CRA7 – зарезервований біт, не використовується

CRA6|CRA5 – встановлюють кількість вимірювань (вибірок) перед видачею результату вимірювання. Ці біти можуть приймати значення: 00 = 1(Default); 01 = 2; 10 = 4; 11 = 8.

CRA4|CRA3|CRA2 – встановлюють, з якою швидкістю будуть записуватися дані в регістри зчитування даних (Data Output Registers):

000 – 0,75 Гц
001 – 1,5 Гц
010 – 3 Гц
011 – 7,5 Гц
100 – 15 Гц (значення за замовчуванням)
101 – 30 Гц
110 – 75 Гц
111 – зарезервовано

CRA1|CRA0 – налаштовують тип вимірювання. Значення регістрів за замовчуванням (00 – нормальний режим вимірювання).

В регістрі B налаштовується чутливість датчика до магнітного поля. За ці установки відповідають біти CRB7|CRB6|CRB5. Решта 5 біт не використовуються, але для коректної роботи вони повинні бути заповнені нулями.

Залежність чутливості (Lsb/Gauss) датчика від встановлених бітів CRB7|CRB6|CRB5

000 – 1370
001 – 1090 (значення за замовчуванням)
010 – 820
011 – 660
100 – 440
101 – 390
110 – 330
111 – 230

Для встановлення максимальної чутливості датчика в регістр B потрібно записати двійкове значення 0b00000000.

В регістрі MR, опис бітів якого представлено в табл. 2.6, налаштовуються такі параметри, як швидкість роботи на шині I2C і режим роботи датчика.

Таблиця 2.6.

MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
HS(0)	(0)	(0)	(1)	(0)	(0)	MD1(0)	MD0(1)

Високошвидкісний режим роботи I2C (3400 кГц) встановлюється записом в біт MR7 одиниці.

MD1 | MD0 встановлюють режим роботи датчика:

00 – режим безперервного виміру.

01 – режим одиничного вимірювання (в цьому режимі датчик працює за замовчуванням). Після вимірювання і запису результату в регістр даних датчик переходить в режим очікування (Idle).

10 – переводить датчик в режим очікування. Споживання струму в цьому режимі знижується до 2 мкА.

11 – також переводить датчик в режим очікування.

Наприклад, для встановлення високошвидкісного з'єднання датчика з мікроконтролером на швидкості 3400кГц і запуску режиму безперервного виміру необхідно записати в регістр двійкове значення 0b10000000.

Сам по собі магнітометр – це прилад, який вимірює напруженість магнітного поля. На рис. 2.94 зображений простий випадок, коли компас розташований горизонтально поверхні землі на екваторі. Червоною стрілкою зазначено напрям до північного полюса.

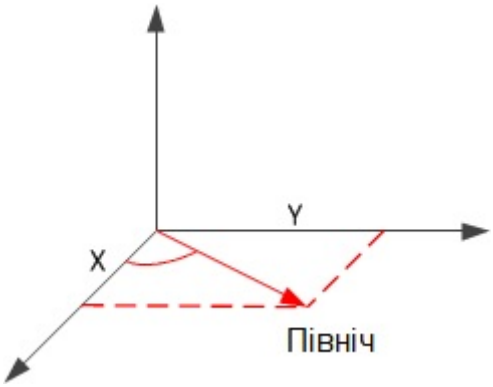


Рис. 2.94.

Для того, щоб показання компаса можна було використовувати, їх необхідно привести до вигляду, яке представляє собою число від -180 до 180, що визначає кут між віссю датчика і північним полюсом Землі. Виділимо три основні проблеми, які можуть перешкодити отримати цей значення цього кута:

1. Зсув і спотворення вектора магнітного поля Землі, внаслідок зовнішніх впливів.

2. Вплив крену датчика на виміряні значення.

3. Різниця між географічним і магнітним полюсами – магнітне схилення.

Якщо обертати магнітометр навколо вісі Z, при тому, що керуюча програма в цей час буде записувати показання датчика кожні 100 мс, і отримати 500 значень, то результат цього експерименту буде виглядати як показано на рис. 2.95.

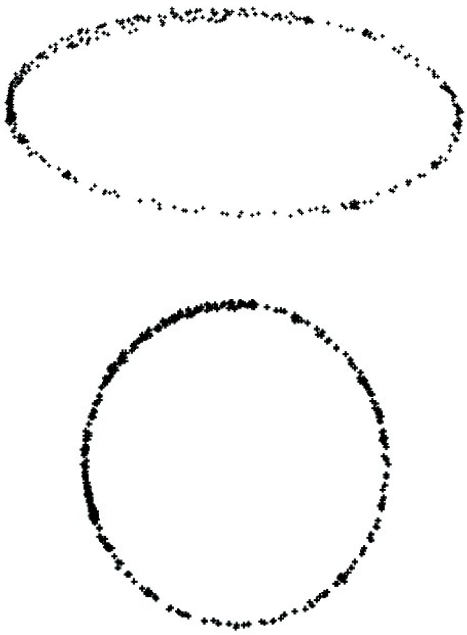


Рис. 2.95.

При одночасному обертанні навколо всіх вісей отримано результат, показаний на рис. 2.96. Додатково до основних 500 точок сфери, додані ще три масиву, по 500 точок в кожному. Кожна з доданих груп точок відповідає за обертання магнітометра навколо фіксованої вісі. Так, верхнє коло отримано обертанням приладу навколо вісі Z. Коло праворуч – обертанням навколо вісі Y. Нарешті, щільне кільце точок ліворуч відповідає за обертання магнітометра навколо вісі X.

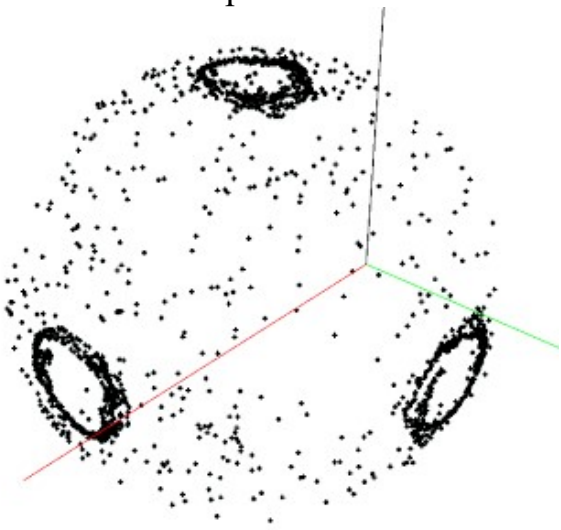


Рис. 2.96.

Оскільки при обертанні електронного компаса виникають ситуації ділення на нуль, для розрахунку кута доцільно користуватися табл. 2.7.

Таблиця 2.7.

X	Y	Азимут
0	<0	90
	>0	270
<0	будь-яке	$180 - arctg(\frac{Y}{X})\frac{180}{\pi}$
>0	<0	$- arctg(\frac{Y}{X})\frac{180}{\pi}$
	>0	$360 - arctg(\frac{Y}{X})\frac{180}{\pi}$

В результаті, використовуючи магнітометри загального призначення, можна створити досить точний електронний компас. Зв'язавши в єдину вимірювальну мережу такий магнітометр, акселерометр і гіроскоп, можна отримати повноцінну інерційну систему, що визначає своє місце розташування в просторі.

2.15. Промислові датчики фізичних величин

На рис. 2.97 представлені датчики температури з різним конструктивним виконанням: а – термопара ДТС-И (ДТП-И) з вбудованим нормуючим перетворювачем (4-20 мА); б – термоперетворювач опору ДТС ТСП-1088 (ТСМ-1088) з комутаційною головкою; в – термопари ТХА, ТХК.



а б в

Рис. 2.97.

Термопари типу ТХА, ТХК, ТПП складаються із двох спаяних на одному з кінців провідників, виготовлених з металів, що володіють різними термоелектричними властивостями. Спаяний кінець, який називається «робочим спаєм», занурюється у вимірюване середовище, а вільні кінці («холодний спай») підключаються до входу вимірюючих приладів, регуляторів. Якщо температури «робочого» і «холодного спаїв»

різні, то виробляється термо-ЕРС (електрорушійна сила), що і подається на прилад. Оскільки термо-ЕРС залежить від різниці температури двох спаїв датчика, то для одержання коректних показань необхідно знати температуру «холодного спаю», щоб компенсувати цю різницю в подальших обчисленнях.

Підключення термопар ТХА, ТХК (термоперетворювачів опору ДТС типу ТСП і ТСМ, термоелектричних перетворювачів) до датчика температури (термоперетворювача) повинне проводитися за допомогою спеціальних компенсаційних (термоелектродних) проводів, виготовлених з тих же матеріалів. Допускається використовувати провідники з металів з термоелектричними характеристиками, аналогічними характеристикам матеріалів електродів термопари в діапазоні температур 0..100 °С. При з'єднанні компенсаційних проводів з термопарами (термоелектричними перетворювачами, термоперетворювачами опору) і приладом необхідно дотримувати полярності. Щоб уникнути впливу перешкод на вимірювальну частину приладу рекомендується екранувати лінію зв'язку приладу з датчиком. При порушенні зазначених умов можуть мати місце значні погрішності при вимірі. Перед підключенням датчика обов’язково ознайомитися з технічною документацією на прилад, в якій уважно ознайомитися зі схемою підключення, характеристиками датчика та правилами монтажу.

Датчики тиску призначені для контролю тиску рідкого або газоподібного середовища. На рис. 2.98 представлені наступні датчики тиску: а – перетворювач надлишкового тиску ПД 100-ДИ; б – перетворювач диференціального тиску ПД200-ДД; в – перетворювач гідростатичного тиску ПД100-ДГ. Існують датчики абсолютного тиску (тиск виміряються відносно вакууму), надлишкового тиску (відносно атмосферного тиску) і диференціального тиску (різниця в тиску між двома точками гідро- або пневмосистеми). Сучасні датчики тиску можуть мати цифрові виходи, функцію програмування, пристрої індикації й цифрової візуалізації даних (дисплей) (ПД200). Часто до окремої групи датчиків тиску відносять датчики розрядження або вакууму.



а б в

Рис. 2.98.

Схеми підключення датчиків тиску ПД100 та ПД200 представлені відповідно на рис. 2.99 та рис. 2.100.

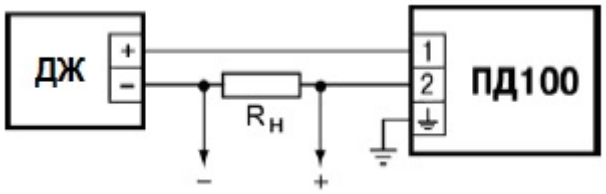


Рис. 2.99.

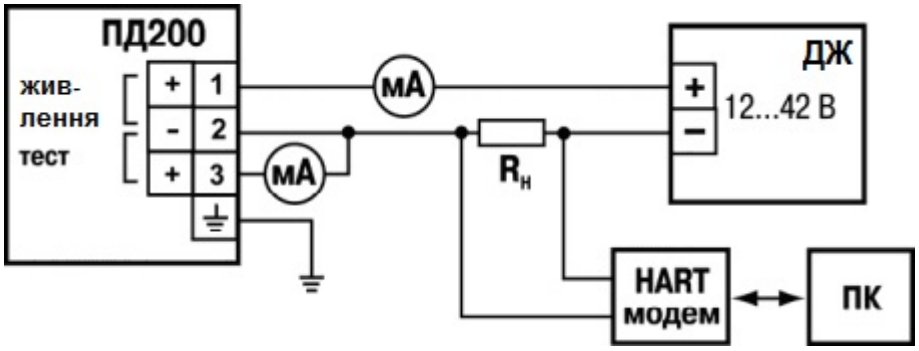


Рис. 2.100.

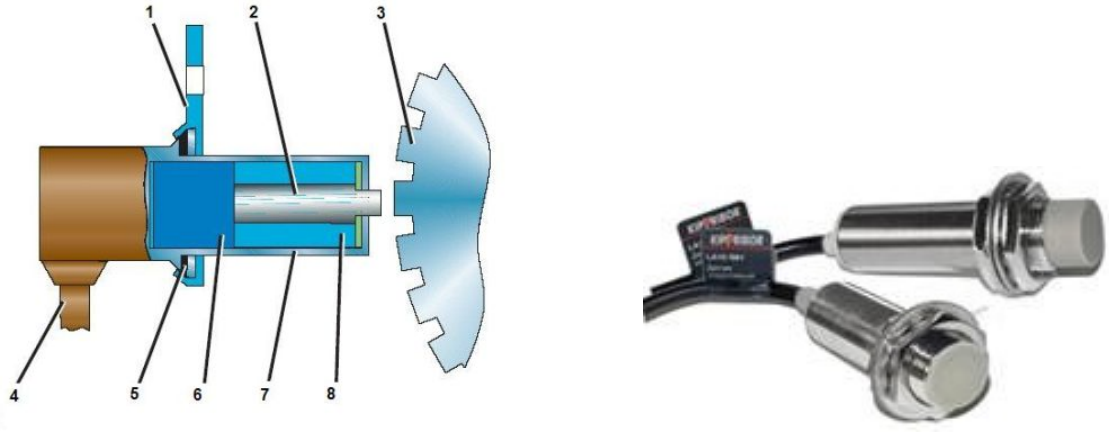
Датчики рівня призначені для телеметричного вимірювання рівня рідини, палива й роздільного шару. Поплавець із постійним магнітом переміщається разом з рівнем рідини по трубі ковзання, у якій перебуває ланцюг вимірювання. Ланцюг вимірювання складається з маленьких чипів з герконом і опором, припаяних до друкованої плати. Магнітне поле поплавця переключає геркони й ланцюг працює за схемою трьохпровідного потенціометра. Напруга, що формується, пропорційна висоті рівня рідини. Також існують ультразвукові датчики – призначений для вимірювання обсягу палива в баку транспортного засобу. Встановлюється або на місце штатного датчика, або безпосередньо в паливний бак. Є безконтактні датчики рівня дизельного палива, бензину, масла. Принцип дії таких датчиків ємнісний. Дві концентричні трубки утворюють обкладки конденсатора, ємність якого змінюється при зміні рівня рідини. Однак такі датчики не призначені для вимірювання рівня води. На рис. 2.101 представлені наступні датчики рівня рідини: а – одноелектродний датчик ДС.ПВТ; б – трьохелектродний датчик ДУ.3; в – поплавковий датчик рівня ПДУ-И.



а б в

Рис. 2.101.

Датчики частоти обертання призначені для видачі електричного сигналу із частотою, що пропорційна частоті обертання вала двигуна, на якому він встановлений. Принцип вимірювання частоти обертання показано на рис. 2.102, а, а зовнішній вигляд безконтактного датчика представлений на рис. 2.102, б.



а б

Рис. 2.102.

На рис. 2.102, а, позначено: 1 – кронштейн кріплення; 2 – магнітна серцевина; 3 – зубчастий ротор; 4 – коаксіальний провід; 5 – ущільнювач; 6 – магніт; 7 – корпус датчика частоти обертання; 8 – обмотка.

Датчики частоти обертання представляють собою генератор електричних імпульсів напруги. При обертанні ротора двигуна силові лінії поля постійного магніту перетинають витки обмотки датчика й у ній виникають електричні імпульси напруги, частота яких пропорційна частоті обертання ротора двигуна.

2.16. Використання GSM-радіомодема для відправлення SMS-повідомлень

GSM – це міжнародний стандарт для мобільних телефонів (абревіатура від «Global System for Mobile Communications» – глобальна система для мобільних комунікацій). Іноді його також називають стандартом 2G (оскільки це мережа другого покоління). Серед усього іншого, GSM дозволяє здійснювати вхідні та вихідні голосові дзвінки, відправляти та приймати текстові повідомлення (SMS – Simple Message System), а також обмінюватися даними (через GPRS).

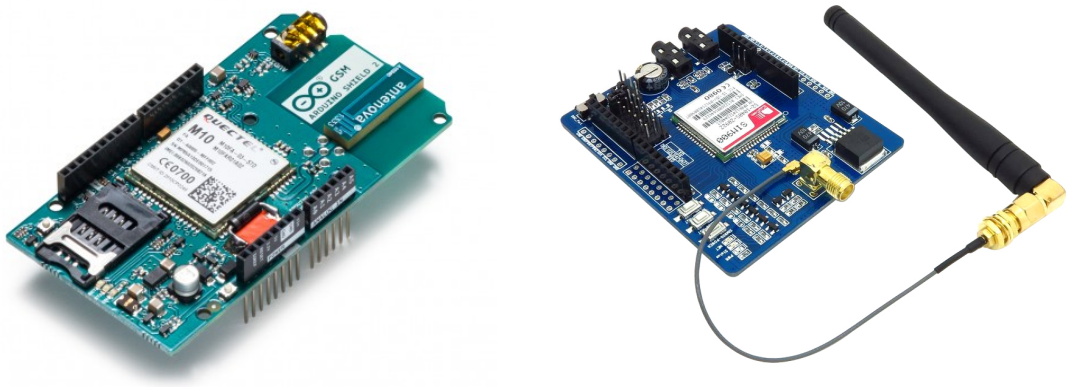
GPRS – це технологія пакетної передачі даних (абревіатура від «General Packet Radio Service»), яка може забезпечити швидкість передачі даних в діапазоні 56-114 кБіт в секунду.

Ряд інших технологій, таких, як SMS, базуються саме на технології GPRS. Плата розширення Arduino GSM дозволяє також використовувати пакетну передачу даних для доступу до мережі Інтернет. Бібліотека GSM дозволяє Arduino працювати як в якості клієнта, так і в якості сервера. При цьому для відправки або отримання веб-сторінки використовуються традиційні http-запити.

Для того, щоб можна було працювати в мобільній мережі, необхідно бути абонентом мобільного оператора, мати GSM-сумісний пристрою (наприклад, плату розширення Arduino GSM або мобільний телефон), а також SIM-карту (Subscriber Identity Module).

Крім плати розширення GSM і самого Arduino, також знадобиться SIM-карта. Взаємодія з провайдером зв'язку здійснюється за допомогою SIM-карти. Провайдер надає SIM-карту і забезпечує GSM-покриття мережі тієї області, де знаходиться абонент (або здійснює роумінг з тією компанією, у якої покриття мережі охоплює місце розташування абонента).

На рис. 2.103, а, представлений зовнішній вигляд плати розширення GSM для Arduino, а на рис. 2.103, б, представлено зовнішній вигляд GSM-радіомодему.



а б

Рис. 2.103.

Для початку роботи з платою розширення, необхідно вставити SIM-карту у відповідний роз'єм. Для завантаження програми в Arduino, контролер необхідно підключити до комп'ютера за допомогою USB-кабелю. Після цього для прошивки досить натиснути кнопку «Загрузить» в середовищі розробки програмного забезпечення Arduino IDE. Після того, як програма буде завантажена в Arduino, контролер можна відключити від комп'ютера і підключити до зовнішнього джерела живлення.

Взаємодія Arduino з GSM-платою розширення здійснюється за допомогою бібліотеки GSM. Велика частина її функцій служить для обробки даних, голосу і SMS-повідомлень. Також в ній передбачено кілька допоміжних функцій для роботи з PIN-кодом SIM-карти і отримання різної інформації про модем. Додаткову інформації та приклади можна знайти за посиланням: https://doc.arduino.ua/ru/prog/GSM

На рис. 2.104 представлена схема підключення GSM-радіомодема до контролера Arduino.

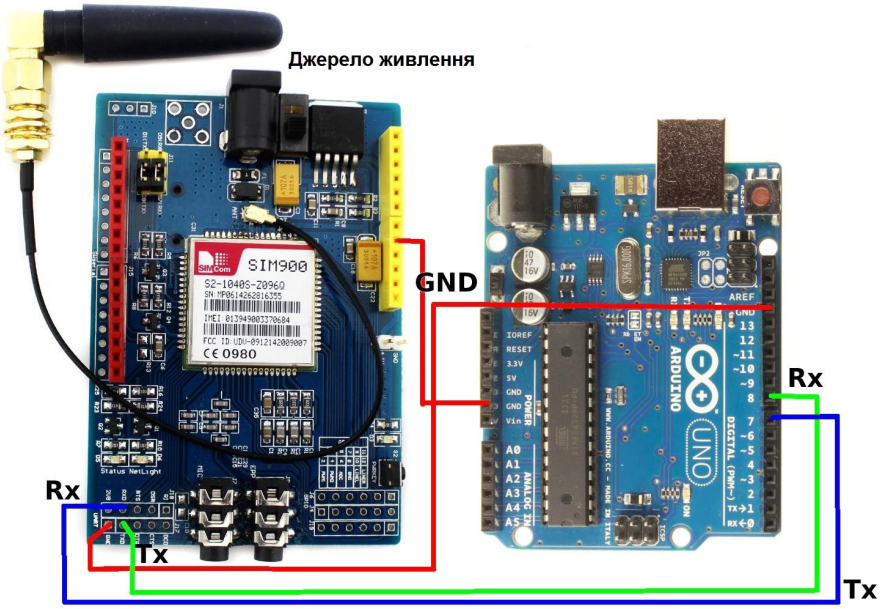


Рис. 2.104.

Управління роботою GSM-радіомодему здійснюється за допомогою AT – команд по послідовному інтерфейсу. Список найбільш широко використовуваних команд наведено в табл. 2.8. У системах зв'язку стандарту GSM передача даних може здійснюватися різними способами. Найпоширеніші – це передача даних по CSD, передача даних по GPRS і передача даних за допомогою SMS-повідомлень. Для передачі даних потрібно попередньо встановити з'єднання з віддаленим абонентом (крім передачі даних за допомогою SMS-повідомлень). При передачі даних за допомогою GPRS модем забезпечує підтримку TCP/IP і UDP протоколів. При передачі даних за допомогою SMS-повідомлень модем забезпечує підтримку текстового і PDU – режимів SMS-повідомлень.

Таблиця 2.8.

Найменування команди	Відповідь на команду	Опис команди
ATE0	OK	Відключити луна
ATE1	OK	Включити луна
AT+IPR=<швидкість>	OK	Налаштувати швидкість обміну по послідовному порту
AT&W	OK	Записати раніше зроблені налаштування в пам'ять
AT+CPIN?	+CPIN:SIM PIN OK	Потрібне введення PIN - коду
	+CPIN: READY OK	Не потрібне введення PIN - коду
AT+CPIN=<PIN-код>	OK	Введення PIN – коду
AT+CREG?	+CREG:<n>,<m>	Запросити інформацію про реєстрацію в мережі GSM . n може приймати значення : 0 - звіт про реєстрацію в мережі вимкнений 1 - звіт про реєстрацію в мережі включений m може приймати значення : 0 - не зареєстрований 1 - зареєстрований 2 - йде пошук мережі 3 - в реєстрації відмовлено 5 - зареєстрований в роумінгу
ATD<+380XXXXXXXXX>;	CONNECT<швидкість>	Установка CSD – з'єднання
AT+CMGS=<"номер"> <текст> Ctrl+Z	> <текст> OK	Відправлення SMS - повідомлення

Нижче наведено приклад використання модему для обміну даними з віддаленими абонентами GSM мережі, а саме передача даних за допомогою SMS-повідомлень. Для передачі даних за допомогою SMS-повідомлень в текстовому режимі необхідно відправити в модем наступні команди :

AT + CMGF = 1 – установка текстового режиму для SMS-повідомлень;

AT + CMGS = <"номер">

< Текст >

Ctrl + Z

Між введенням команди AT + CMGS = <"номер"> і введенням тексту необхідно дочекатися відповіді модему у вигляді символу ">".

Прийом SMS-повідомлення відображається звітом: + CMTI : < пам'ять>, <індекс>, де <пам'ять> – тип пам'яті, в якій збережено повідомлення, <індекс> – порядковий номер повідомлення в пам'яті.

Прочитати отримане повідомлення можна командою: AT + CMGR = <індекс>, 0.

Для спрощення коду програми бібліотека GSM для Arduino абстрагує низькорівневі функції, що працюють з модемом і SIM-картою. Кожна виконувана модемом команда, як правило, є фрагментом послідовності команд, спрямованих на виконання будь-якої функції. Бібліотека GSM спроектована з урахуванням цього, і здатна приймати / повертати інформацію на будь-якому етапі виконання послідовності команд.

На рис. 2.105 представлено схему підключення GSM-радіомодему до контролера Arduino в Proteus. За допомогою Proteus можна моделювати мікропроцесорні системи збору, обробки даних та керування, а також відлагоджувати програмне забезпечення без наявності самого пристрою.

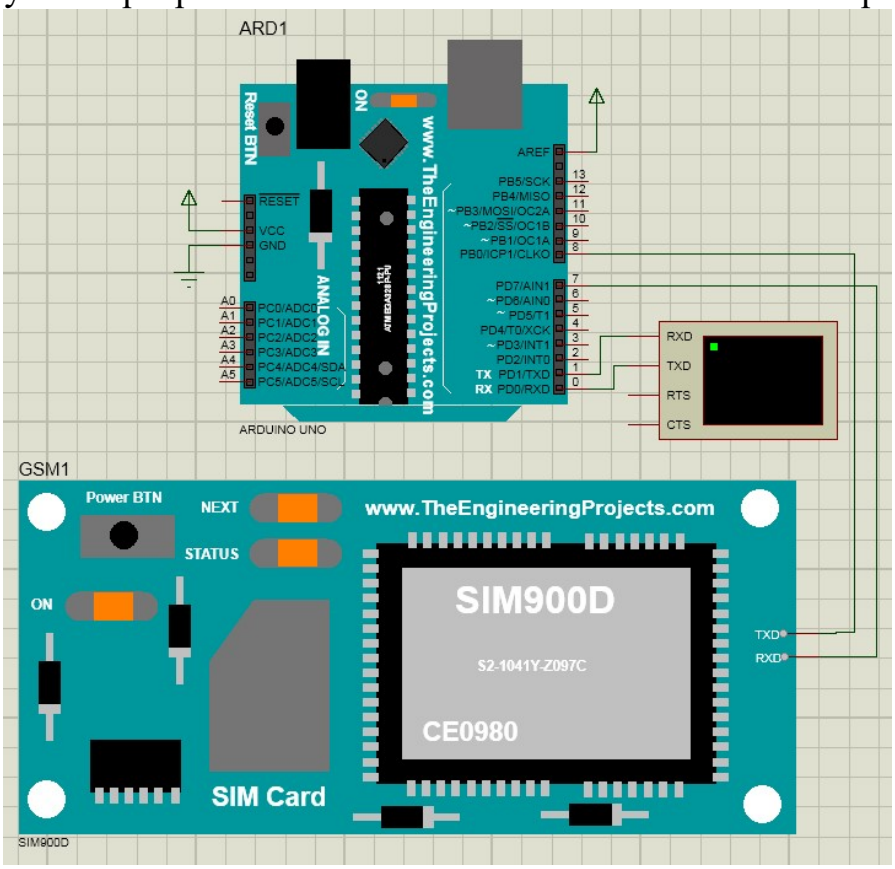


Рис. 2.105.

Бібліотеку для роботи з GSM-радіомодемом в Proteus можна знайти та завантажити на сайті за посиланням:

https://www.theengineeringprojects.com/2016/03/gsm-library-proteus.html

Після завантаження з цього сайту архіву з файлами, їх необхідно розпакувати в каталог LIBRARY програми Proteus. У вікні додавання

компонентів в схему модель радіомодему можна знайти за ключовим словом SIM900, як показано на рис. 2.106.

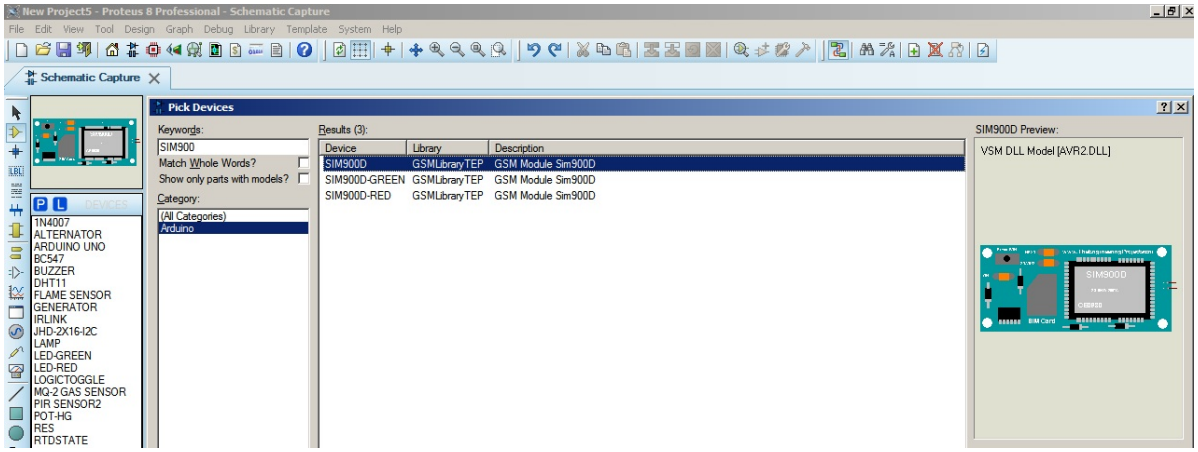


Рис. 2.106.

Додавши модуль GSM-радіомодему в схему, у вікні його властивостей потрібно вказати шлях до файлу GSMLibraryTEP.hex. Після цього модель готова до використання в схемі. На момент написання книги модель GSM-радіомодему не підтримувала функціональність, пов'язану з відправленям текстових повідомлень. Тому для перевірки наступної програми бажано мати в наявності контролер Arduino UNO та GSM-радіомодем.

Наступна програма демонструє відправлення SMS-повідомлення за домогогою GSM-радіомодема. Підключивши Arduino до комп'ютера, за допомогою Монітора порта можна ввести повідомлення, що буде відправлено. Для виклику вікна Монітору порта потрібно в середовищі розробки програмного забезпечення Arduino IDE в меню «Инструменты» вибрати пункт «Монитор порта», або натиснути комбінацію клавіш Ctrl+Shift+M.

```
#include <GSM.h> // підключення бібліотеки для роботи з радіомодемом
 #define PINNUMBER "" // визначення PIN-коду; якщо PIN-код встановлений,
// потрібно його вказати, наприклад "0000"
 // ініціалізація бібліотеки для роботи з GSM-радіомодемом
GSM gsmAccess; // для включення відлагодження необхідно вказати параметр 'true'
GSM_SMS sms;
 // масив символів представляє сбою телефонний номер для відправлення SMS
// змініть номер на інший, куди потрібно надіслати текстове повідомлення
char remoteNumber[20]= "+380641234567";
 // масив символів текстового повідомлення
char txtMsg[200]="Test";

void setup()
{
  // ініціалізація послідовного інтерфейсу
  Serial.begin(9600);
  Serial.println("SMS Messages Sender");
```

```
  // стан з’єднання
  boolean notConnected = true;
  // запуск GSM-радіомодему
  // якщо на SIM-карті встановлений PIN-код, необхідно вказати його
  // в якості параметра метода begin()
  while(notConnected)
  {
    if(gsmAccess.begin(PINNUMBER) == GSM_READY) // якщо вдале підключення
      notConnected = false;      // зміна значення змінної
    else
    {
      // якщо немає з’єднання, в Монітор порта виводиться відповідне повідомлення
      Serial.println("Not connected");      delay(1000);
    }
  }
  Serial.println("GSM initialized"); // передавання повідомлення про вдале підключення
  sendSMS();  // виклик функції відправлення SMS повідомлення
}

void loop()
{
        // в цій функції не повинно бути жодного програмного коду
}

void sendSMS() {      // реалізація функції відправлення текстового повідомлення
  // організація виводу інформації користувачу через Монітор порта
  // виведення повідомлення про номер телефону, куди буде відправлено SMS
  Serial.print("Message to mobile number: ");
  Serial.println(remoteNumber);

  // виведення в Монітор порта тексту повідомлення
  Serial.println("SENDING");
  Serial.println();
  Serial.println("Message:");
  Serial.println(txtMsg);

  // відправлення SMS-повідомлення
  sms.beginSMS(remoteNumber);
  sms.print(txtMsg);
  sms.endSMS(); // звершення передавання SMS-повідомлення
  Serial.println("\nCOMPLETE!\n");  // повідомлення про завершення операції
}
```

В наступній системі, схема якої представлена на рис. 2.107, виконується контроль наявності руху за допомогою інфрачервоного датчика руху, та у разі його виявлення відправляється текстове повідомлення абоненту. Наведений приклад представляє собою простий варіант охоронної системи зі сповіщенням у разі виявлення несанкціонованого проникнення.

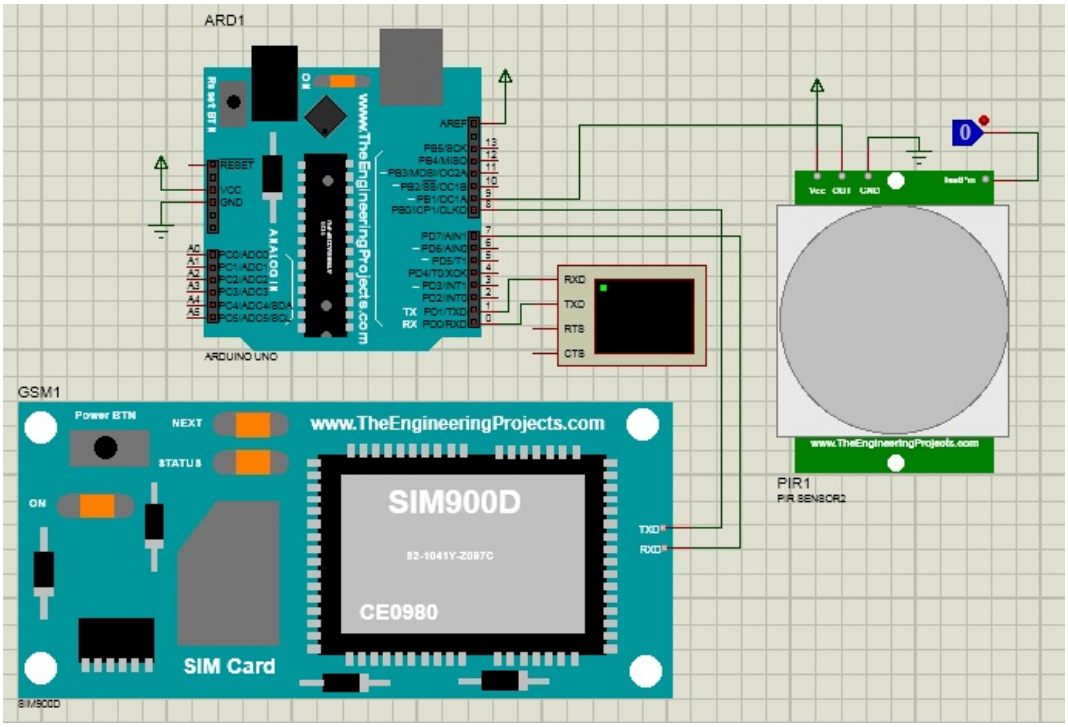


Рис. 2.107.

Текст програми представлено нижче.

```
#include <GSM.h> // підключення бібліотеки для роботи з радіомодемом
#define PINNUMBER "" // визначення PIN-коду; якщо PIN-код встановлений,
// потрібно його вказати, наприклад "0000"
 // ініціалізація бібліотеки для роботи з GSM-радіомодемом
GSM gsmAccess; // для включення від лагодження необхідно вказати параметр 'true'
GSM_SMS sms;
 // масив символів представляє собою телефонний номер для відправлення SMS
// змініть номер на інший, куди потрібно надіслати текстове повідомлення
char remoteNumber[20]= "+380641234567";
 // масив символів текстового повідомлення
char txtMsg[200]="Alarm! Robber is here!";

int pirSensorPin = 9;

void setup()
{
  pinMode(pirSensorPin, INPUT);
  // ініціалізація послідовного інтерфейсу
  Serial.begin(9600);
  Serial.println("SMS Messages Sender");
  // стан з’єднання
  boolean notConnected = true;
  // запуск GSM-радіомодему
  // якщо на SIM-карті встановлений PIN-код, необхідно вказати його
  // в якості параметра метода begin()
  while(notConnected)
  {
    if(gsmAccess.begin(PINNUMBER) == GSM_READY) // якщо вдале підключення
```

```
    notConnected = false;  // зміна значення змінної
   else
   {
    // якщо немає з'єднання, в Монітор порта виводиться відповідне повідомлення
    Serial.println("Not connected");      delay(1000);
   }
  }
  Serial.println("GSM initialized"); // передавання повідомлення про вдале підключення

}

void loop()
{
 if(digitalRead(pirSensorPin) == HIGH)
  {
   sendSMS();  // виклик функції відправлення SMS повідомлення
   delay(10000);
  }
}

void sendSMS() {  // реалізація функції відправлення текстового повідомлення
  // організація виводу інформації користувачу через Монітор порта
  // виведення повідомлення про номер телефону, куди буде відправлено SMS
  Serial.print("Message to mobile number: ");
  Serial.println(remoteNumber);

  // виведення в Монітор порта тексту повідомлення
  Serial.println("SENDING");
  Serial.println();
  Serial.println("Message:");
  Serial.println(txtMsg);

  // відправлення SMS-повідомлення
  sms.beginSMS(remoteNumber);
  sms.print(txtMsg);
  sms.endSMS(); // звершення передавання SMS-повідомлення
  Serial.println("\nCOMPLETE!\n");  // повідомлення про завершення операції
}
```

Велику кількість прикладів роботи з GSM-радіомодемом та документацію на відповідну бібліотеку можна знайти за посиланням: https://doc.arduino.ua/ru/prog/GSM

2.17. Годинник реального часу DS1302

Суть годинника реального часу полягає в тому, що при наявності батарейки він може продовжувати працювати навіть якщо головний пристрій знеструмлено. Такі годинники постійно використовуються в комп'ютерах, ноутбуках, цифрових фотоапаратах та інших пристроях. Модуль годинника реального часу (рис. 2.108) зазвичай представляє собою невелику плату, що містить, як правило, одну з мікросхем DS1307,

DS1302, DS3231. Крім того, на платі розташовано механізм установки батарейки живлення. Такі модулі часто використовуються для обліку часу, дати, дня тижня та інших хронометричних параметрів, при цьому застосовуються зручні для людини одиниці вимірювання – години, хвилини секунди, день, місяць, рік. Модуль працює від автономного живлення – батарейки, і продовжує вести відлік часу навіть якщо на контролері Arduino відсутнє живлення.

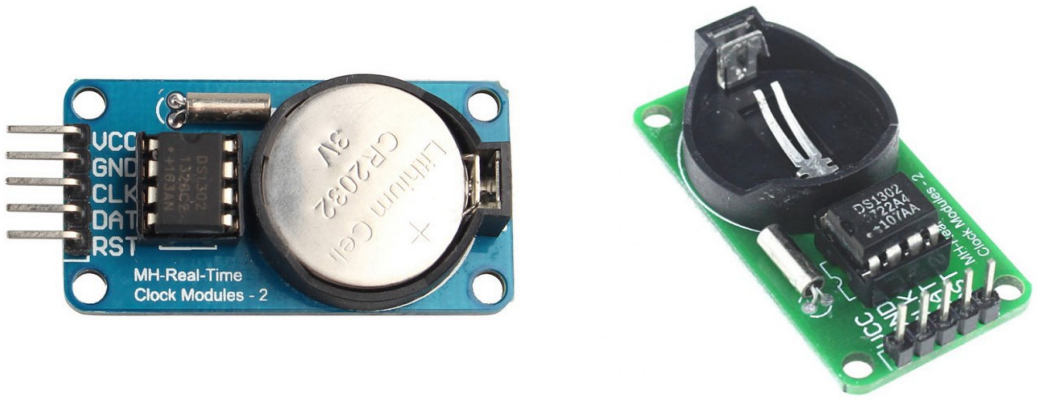


Рис. 2.108.

На більшості мікроконтролерів, в тому числі і Arduino, є вбудований лічильник часу, використання якого відбувається за допомогою функції millis(). Є і вбудовані в мікросхему таймери, які можуть відстежувати більш тривалі проміжки часу (хвилини або дні). Виникає питання, навіщо використовувати окремий модуль годинника реального часу? Основна проблема полягає в тому, що функція millis() відстежує час тільки з моменту подачі живлення на контролер Arduino. Тобто, як тільки на контролер перестає поступати живлення, таймер скидається в 0. Іншими словами, контролер Arduino не знає, що зараз, наприклад, вівторок або 17-е вересня. Все, що можна отримати від вбудованого лічильника – це інформацію про проміжок часу починаючи з деякого моменту, наприклад «минуло 5600 мілісекунд з моменту останнього включення». Якщо вимкнути живлення мікроконтролера, лічильник часу зіб'ється. У деяких проектах Arduino знадобиться надійний контроль часу без переривань. Саме в таких випадках використовується зовнішній модуль годин реального часу. Мікросхема, яка використовується в подібних годинниках, відстежує рік і навіть кількість днів у місяці (єдине, що зазвичай не враховується – це перехід на літній і зимовий час, тому що такі переходи різні в різних частинах світу).

На рис. 2.109 представлено принципову схему годинника з виводом інформації на алфавітно-цифровий рідкокристалічний індикатор. В схемі використовується модуль годинника реального часу на основі мікросхеми DS1302. Модуль годинника реального часу необхідно підключити до виводів SCL, SDA (на схемі позначений як I/O), що відносяться до шини I2C, та виводу RST. Також необхідно підключити лінії живлення (Vcc) і

землі (GND). Рідкокристалічний індикатор підключений до контролера Arduino по 4-х бітній паралельній шині даних (лінії D4-D7). Для налаштування контрастності зображення на екрані РКІ використовується резистор зі змінним опором, позначений на схемі як RV1. Для налаштування часу (збільшення та зменшення годин та хвилин) в схемі наявні 4 кнопки (елементи LOGICTOGGLE), підключені до виводів 10, 11, 12 та 13 контролера Arduino. Також для перевірки коректності передавання інформації про час з використанням UART в моделі використовується віртуальний термінал.

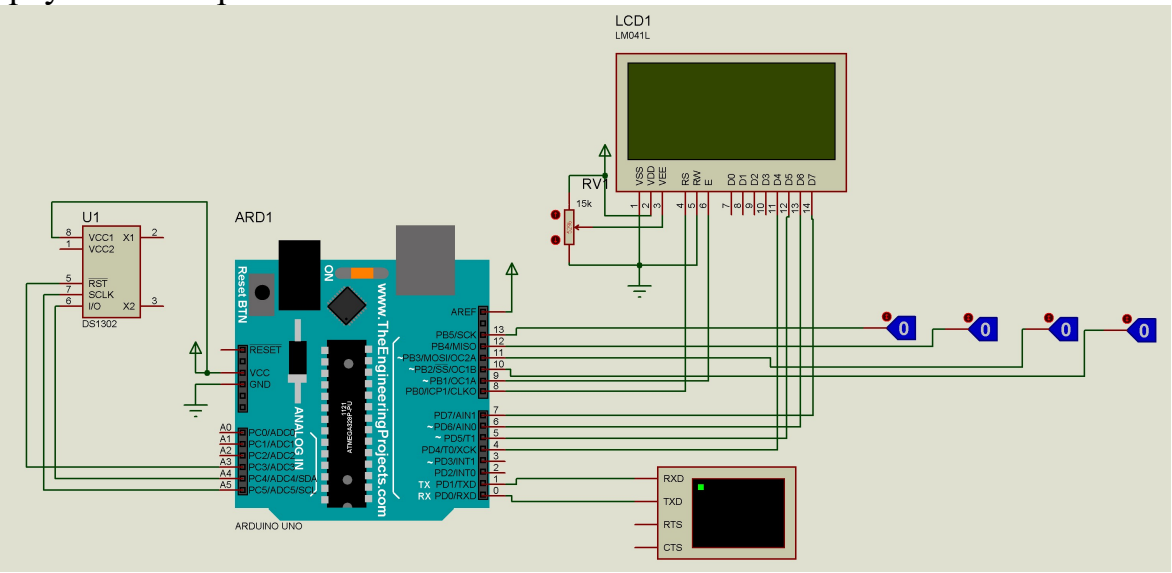


Рис. 2.109.

На годиннику реального часу для його підключення до контролера Arduino (або іншого) зазвичай наявні 5 виводів: 5V, GND, SCL, SDA і SQW:

– 5V використовується для живлення модуля годинника реального часу; якщо сигнал 5 В не надходить, мікросхема переходить в "сплячий" режим і живиться від батарейки чи акумулятора, що розташований на платі;

– GND – загальний провідник (земля);

– SCL – лінія сигналу синхронізації інтерфейсу I2C – необхідна для обміну даними з годинником реального часу;

– SDA – лінія даних інтерфейсу I2C, по якій передаються дані з годинника реального часу;

– SQW – дає можливість налаштувати вивід даних у вигляді прямокутних імпульсів (square-wave). У більшості випадків цей контакт не використовується.

Для роботи з годинником реального часу використовується бібліотека «iarduino_RTC.h». Ця бібліотека дозволяє зчитувати і записувати час RTC модулів на базі мікросхем DS1302, DS1307, DS3231 та інших. Перевагою даної бібліотеки є зручна реалізація методів отримання часу. Дана бібліотека може використовувати як апаратну, так і програмну реалізацію шини I2C. Призначення функцій і змінних бібліотеки:

#include <iarduino_RTC.h> // підключення бібліотеки

// створення об'єкта
iarduino_RTC Об'єкт (НАЗВА [, ВИВІД_RST [, ВИВІД_CLK [, ВИВІД_DAT]]]);

Функція begin (); // ініціалізація роботи RTC модуля

Функція settime (СЕК [, ХВИЛИНИ [, ГОДИНИ [, ДЕНЬ [, МІСЯЦЬ [, РІК [, ДЕНЬ ТИЖНЯ]]]]]]); // встановлення дати та/або часу
Функція gettime ([СТРОКА]); // зчитування дати та/або часу

Функція blinktime (ПАРАМЕТР [ЧАСТОТА]); // змушує функцію gettime «блимати» зазначеним параметром часу

Функція period (ХВИЛИНИ); // вказує мінімальний період звернення до модуля в хвилинах

Змінна seconds // повертає секунди від 0 до 59
Змінна minutes // повертає хвилини від 0 до 59
Змінна hours // повертає години від 1 до 12
Змінна Hours // повертає години від 0 до 23
Змінна midday // повертає 0 або 1 (0 – AM, 1 – PM)
Змінна day // повертає день місяця від 1 до 31
Змінна weekday // повертає день тижня від 0 до 6 (0 – неділя, 6 – субота)
Змінна month // повертає місяць від 1 до 12
Змінна year // повертає рік від 0 до 99

Завантажити бібліотеку для роботи з модулем годинника реального часу можна за посиланням: ttp://iarduino.ru/lib/3c1d8d2a7ff11ba50274fcca11f7fe83.zip

Для того, щоб підключити бібліотеку, в середовищі ArduinoIDE потрібно в меню «Скетч» вибрати підпункт «Подключить библиотеку», і у списку вибрати «Добавить .ZIP библиотеку», як показано на рис. 2.110.

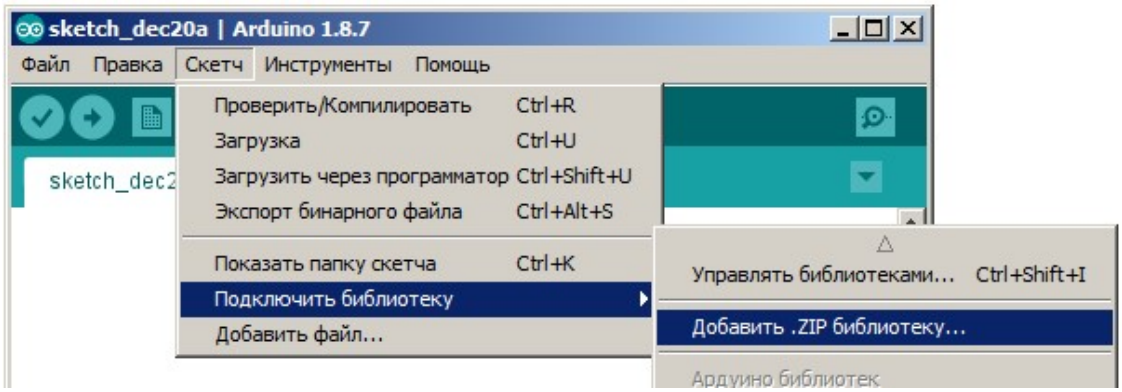


Рис. 2.110.

Після цього у вікні, що з’явиться, потрібно вказати шлях до архіву (файл 3c1d8d2a7ff11ba50274fcca11f7fe83.zip), в якому містяться файли бібліотеки.

Текст програми представлено нижче.

```
// підключення бібліотеки для роботи з рідкокристалічним індикатором
#include <LiquidCrystal.h>
// підключення бібліотеки для роботи з годинником реального часу
#include <iarduino_RTC.h>

// при створенні об’єкта для роботи з модулем  потрібно вказати, з яким саме модулем
// він буде працювати – в даному випадку це RTC_DS1302; також вказуються номери виводів
// контролера Arduino, до яких підключено модуль: 17 – вивід RST, 18 – вивід SDA,
// 19 – вивід SCK (див. принципову схему)
iarduino_RTC time(RTC_DS1302, 17, 19, 18);

// створення об’єкта для роботи з РКІ-модулем; при створенні об’єкта вказуються номери
// виводів контролера, до яких підключено рідкокристалічний індикатор: 8 – RS, 9 – E,
// 4 – D4, 5 – D5, 6 – D6, 7 – D7 (див. принципову схему)
LiquidCrystal lcd(8,9,4,5,6,7);

// перед створенням коду та реалізацією алгоритму роботи програми потрібно
// оголосити змінні
int m=0, h=0, s=0;      // змінні для збереження цілих значень годин, хвилин та секунд
// оголошення змінних для збереження значень про стани кнопок збільшення або зменшення
// годин та хвилин на одиницю
int h_up, h_down, m_up, m_down;

// ця функція виконується один раз при вмиканні живлення контролера і в ній відбувається
// початкова ініціалізація всіх необхідних модулів, налаштування ражимів роботи виводів
void setup() {
// виводи, до яких підключено кнопки для налаштування часу, налаштовуються на вхід
  pinMode(10, INPUT); // до виводу 10 підключена кнопка зменшення хвилин на 1
  pinMode(11, INPUT); // до виводу 11 підключена кнопка збільшеня хвилин на 1
  pinMode(12, INPUT); // до виводу 12 підключена кнопка зменшення годин на 1
  pinMode(13, INPUT); // до виводу 13 підключена кнопка збільшення годин на 1

  // ініціалізація РКІ та встановлення режиму виводу інформації 2 строки по 16 символів
  lcd.begin(16,2);
  // формування часової затримки 300 мс для того, щоб РКІ перейшов в потрібний режим
  delay(300);
  Serial.begin(9600); // налаштування швидкості передавання даних по UART 9600 біт/с
  time.begin(); // ініціалізація модуля годинника реального часу
  // в бібліотеці «iarduino_RTC» для роботи з датою і часом реалізовані дві функції:
  // settime() – встановлення дати/часу та gettime() – отримання інформації про дату/час
  // встановлення дати та часу; цю строчку коду після першого завантаження програми
```

```
  // необхідно закоментувати, і потім повторно завантажити програму в контролер
  time.settime(0,51,9,27,6,18,4);// 0  сек, 51 хв., 9 годин, 27 червня 2019 року, четвер
}

void loop() {
 if(millis()%1000==0){          // якщо пройшла 1 секунда
     // передавання у вигляді строки інформації про час в комп'ютер
     Serial.println(time.gettime("d-m-Y, H:i:s, D"));

     lcd.setCursor(0,0);          // встановлення курсору на екрані РКІ на початок першого
рядка
     lcd.print(time.gettime("H:i:s"));  // виведення часу на екран РКІ
     lcd.setCursor(0,1);          // встановлення курсору на екрані РКІ на початок другого
рядка
     lcd.print(time.gettime("d-m-Y, D")); // виведення дати та дня тижня на екран РКІ

     delay(1); // формування часової затримки тривалістю 1 мс
         // зчитування в змінні інформації про стан кнопок – натиснуті чи ні
     h_up=digitalRead(13);              // зчитування стану кнопки збільшення годин на 1
     h_down=digitalRead(12);          // зчитування стану кнопки зменшення годин на 1
     m_up=digitalRead(11);              // зчитування стану кнопки збільшення хвилин на 1
     m_down=digitalRead(10);          // зчитування стану кнопки зменшення хвилин на 1

         // якщо натиснута будь-яка кнопка, то виконується блок коду у фігурних дужках
     if((h_up == HIGH) || (h_down == HIGH) || (m_up == HIGH) || (m_down == HIGH))  {
      m=time.minutes;          // зчитування значення хвилин в змінну m з модуля RTC
      h=time.Hours;              // зчитування значення годин в змінну h з модуля RTC
      s=time.seconds;          // зчитування значення секунд в змінну s модуля RTC

         // перевірка, чи натиснута кнопка збільшення годин на 1
      if(h_up == HIGH) {       // якщо кнопка натиснута
         if(h<23) h++;          // якщо значення годин менше 23, відбувається додавання 1
         else h=0;                // інакше змінній присвоюється значення 0
        }

         // перевірка, чи натиснута кнопка зменшення годин на 1
      if(h_down == HIGH){   // якщо кнопка натиснута
        if(h>0) h--;              // якщо значення годин більше 0, відбувається віднімання 1
        else h=23;                // інакше змінній присвоюється значення 23
        }

         // перевірка, чи натиснута кнопка збільшення хвилин на 1
      if(m_up == HIGH){       // якщо кнопка натиснута
         if(m<59) m++;// якщо значення хвилин менше 59, відбувається додавання 1
         else m=0;       // інакше змінній присвоюється значення 0
        }

         // перевірка, чи натиснута кнопка зменшення хвилин на 1
      if(m_down == HIGH){  // якщо кнопка натиснута
         if(m>0) m--;   // якщо значення хвилин більше 0, відбувається віднімання 1
         else m=59;              // інакше змінній присвоюється значення 59
        }
```

```
      time.settime(s,m,h);       // встановлення оновленого часу
    }
  }
}
```

У бібліотеці «iarduino_RTC» доступні 9 числових змінних: seconds, minutes, hours, Hours, midday, day, weekday, month і year, значення яких оновлюються після кожного виклику функції gettime(). Ці змінні використовуються для зміни дати і часу. Програма працює за наступним алгоритмом:

1. Спочатку підключаються бібліотеки «LiquidCrystal.h» для роботи з алфавітно-цифровим рідкокристалічним індикатором та «iarduino_RTC.h» для роботи з модулем годинника реального часу.

2. Після цього створюються об'єкти time для роботи з годинником реального часу, та lcd для роботи з РКІ. Ці об'єкти містять поля (змінні) та методи (функції), за допомогою яких виконується запис та зчитування даних з модулів.

3. Далі оголошуються змінні m, h та s, яким відразу присвоюється значення 0, та які мають тип int, тобто змінні можуть зберігати лице цілі числа. Це очевидно, адже зазвичай години, хвилини та секунди мають цілі значення.

4. Для отримання даних з 4-х кнопок, за допомогою яких відбувається збільшення та зменшення годин та хвилин, оголошуються чотири змінні: h_up, h_down, m_up, m_down. Ці змінні також мають тип int, адже при зчитуванні значення з входу, до якого підключена відповідна кнопка, значення, що повертається функцією digitalRead() може приймати одне з двох значень: HIGH (якщо на вході присутній рівень лог.1) або LOW (якщо на вході присутній рівень лог.0). Слід відмітити, що в даній реалізації повертається значення саме типу int (цілого), а не bool (логічного), що пов'язано з особливостями реалізації цієї функції в середовищі ArduinoIDE.

5. В функції setup() виконується налаштування виводів, до яких підключені кнопки, на вхід, виконується ініціалізація модулів РКІ та годинника реального часу шляхом виклику метода begin() об'єктів lcd та time. Також ініціалізується послідовний інтерфейс UART для передавання даних про час в комп'ютер і встановлюється швидкість передавання даних 9600 біт/с. Для цього використовується спеціальний об'єкт Serial та метод begin().

6. В функції loop() циклічно з періодом 1 секунда виконується наступна послідовність дій:

– в комп'ютер передається текстова строчка з інформацією про час у форматі: «день-місяць-рік, години:хвилини:секунди, день тижня»;

– на екран РКІ у верхньому рядку виводиться інформація про час у форматі «години:хвилини:секунди», а у нижньому рядку виводиться інформація про дату та день тижня у форматі «день-місяць-рік, день тижня»;

– відбувається зчитування значень з кнопок у відповідні змінні за допомогою функції digitalRead();

– перевіряються отримані значення про стан кнопок, і якщо будь-яка з кнопок натиснута (хоча б в одній змінній h_up, h_down, m_up або m_down міститься значення HIGH), відбувається зчитування з датчика годин, хвилин та секунд у змінні h, m та s відповідно;

– якщо натиснута кнопка збільшення годин на одиницю, відбувається перевірка, чи не перевищує значення годин в змінній h величину 23, і якщо не перевищує, відбувається збільшення значення цієї змінної на одиницю. Якщо ж значення змінної h дорівнює 23, тоді при збільшенні годин на 1 наступним значенням буде 0. Аналогічні дії виконуються при натисканні кнопки зменшення годин на 1 з тією різницею, що зменшення відбувається лише в тому випадку, коли значення в змінній h більше 0. Якщо значення в змінній h дорівнює 0, то при натисканні кнопки зменшення годин на 1 наступним значенням буде 23, адже години можуть змінюватися в діапазоні від 0 до 23;

– якщо натиснута кнопка збільшення хвилин на одиницю, відбувається перевірка, чи не перевищує значення хвилин в змінній m величину 59, і якщо не перевищує, відбувається збільшення значення цієї змінної на одиницю. Якщо ж значення змінної m дорівнює 59, тоді при збільшенні хвилин на 1 наступним значенням буде 0. Аналогічні дії виконуються при натисканні кнопки зменшення хвилин на 1 з тією різницею, що зменшення відбувається лише в тому випадку, коли значення в змінній m більше 0. Якщо значення в змінній m дорівнює 0, то при натисканні кнопки зменшення хвилин на 1 наступним значенням буде 59, адже хвилини можуть змінюватися в діапазоні від 0 до 59;

– за допомогою функції settime() відбувається запис оновлених даних про час у годинник реального часу;

Модулі годинників реального часу використовуються в багатьох проектах. Вони потрібні для систем реєстрації даних, при створенні таймерів і керуючих пристроїв, які працюють за заданим розкладом, в побутових приладах. За допомогою широко поширених і дешевих модулів можна створити такі проекти як будильник або реєстратор даних з сенсорів, записуючи інформацію на SD-карту пам’яті або показуючи час на екрані дисплея, як було продемонстровано в наведеному прикладі.

Завдання до розділу 2

1. Розробити принципову схему парктроніка для автомобіля, в якому використовувати ультразвукові датчики відстані. Написати відповідну програму.
2. Розробити принципову схему автомобільної сигналізації. Якщо відчиняється хоча б одна з дверей чи багажник, або рівень вібрації автомобіля перевищує деякий заданий рівень (якщо хтось вибив скло), тоді спрацьовує звукова сигналізація.

3. Розробити спідометр для велосипеду з виведенням швидкості на рідкокристалічний індикатор. Для вимірювання швидкості обертання колеса використовувати геркон, який встановлюється на раму велосипеду, та магніт, який встановлюється на колесо.
4. Розробити принципову схему тахометра на базі Arduino та написати відповідну програму. Для вимірювання обертів використовувати світлодіод та фотодіод.
5. Розробити тестер ємності Li-ion акумуляторів. Принцип роботи наступний: через аналоговий вхід виконується вимірювання падіння напруги на навантажувальному резисторі. За законом Ома $I=U/R$. Кожну секунду, отримане значення ділиться на 3600 і додається для отримання ємності акумулятора в Ампер/годинах. Схема на рис. 2.111 пояснює описаний принцип вимірювання ємності акумулятора.

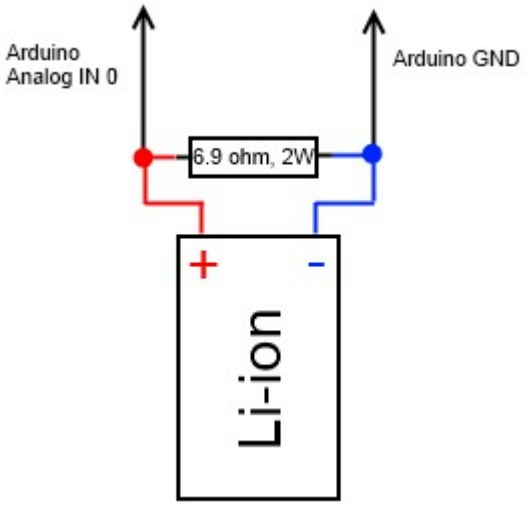


Рис. 2.111.

6. На базі контролера Arduino розробити вимірювач потужності, що споживається, для дому або квартири. Особливістю пристрою є те, що зчитування інформації про енергоспоживання відбувається через світлодіод лічильника, шляхом вимірювання тривалості проміжків часу між загораннями світлодіода. Відповідно, старі радянські дискові лічильники електроенергії не підійдуть для реалізації цього проекту.
7. На базі контролера Arduino розробити принципову схему та написати програму для вимірювання частоти напруги в мережі.
8. Розробити систему, що забезпечує керування обігрівачем (або кондиціонером) шляхом відправлення SMS-повідомлення з мобільного телефону.
9. Розробити портативний електронний компас з виведенням інформації на рідкокристалічний дисплей. Інформація на дисплеї представляє собою значення кута між напрямком на північ та орієнтацією приладу.

10. Розробити барометр. Значення атмосферного тиску відображати на екрані РКІ. В якості датчику використовувати модуль BMP180. Опис датчика можна знайти за посиланням:

https://iarduino.ru/shop/Sensory-Datchiki/datchik-temperatury-ds18b20-1-metr-vodonepronicaemyy.html

Бібліотека для роботи з однопровідним інтерфейсом 1-Wire доступна за посиланням:

https://iarduino.ru/file/51.html

Інструкцію по підключенню бібліотеки можна знайти за адресою:

https://wiki.iarduino.ru/page/Installing_libraries/

11. Розробити систему автоматичного освітлення. При наближенні людини світло вмикається на 30 секунд, після чого вимикається. Для реалізації проекту використовувати датчик HC-SR501.
12. Розробити систему «Розумний будинок». Вона може контролювати температуру зовні і всередині приміщення, фіксувати відкрито або закрито вікно, показувати, чи йде дощ, а також подавати сигнал, коли спрацює датчик руху PIR. На рис. 2.112 представлено схему системи, на якій позначені типи датчиків.

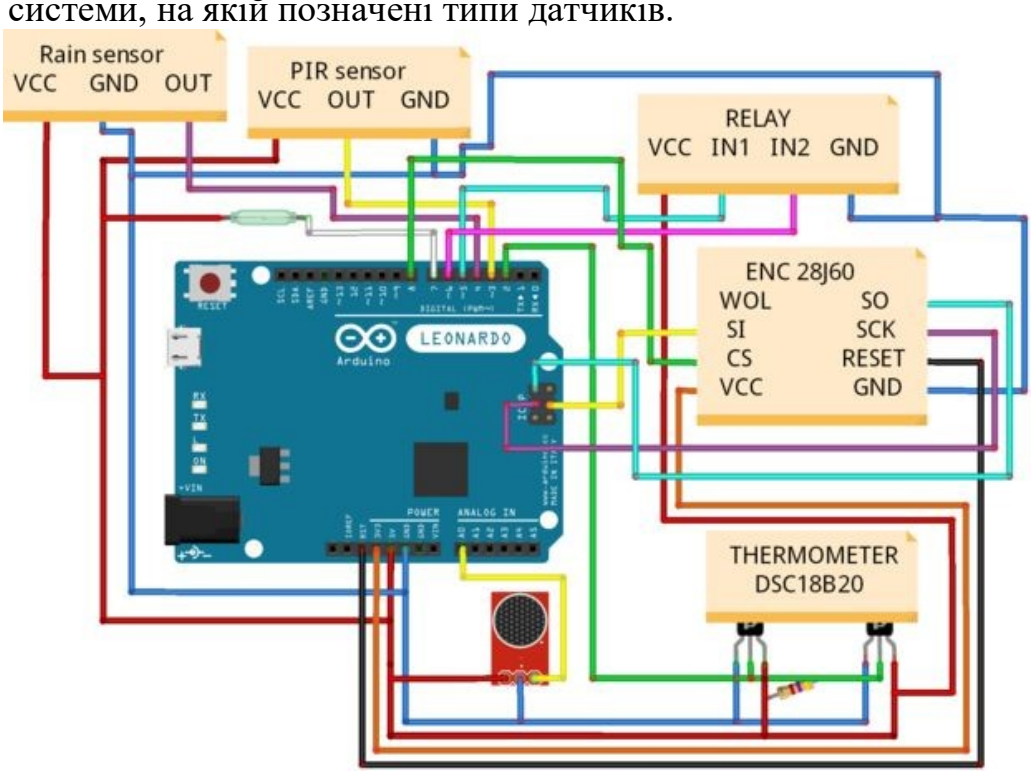


Рис. 2.112.

13. Розробити дзвоник з двома мелодіями. Схема повинна містити дві кнопки та динамік. При натисканні кнопки відбувається програмування заданої мелодії.
14. Розробити детектор забруднення повітря в навколишньому середовищі. Пристрій повинен виявляти основні забруднювачі атмосфери: озон, оксид вуглецю, діоксид сірки і оксид азоту. Також пристрій повинен включати детектор побутового газу, що дозволить попереджати користувачів про витік газу або присутності горючих газів.

Велику кількість прикладів та реальних проектів на базі контролера Arduino можна знайти за посиланням: http://cxem.net/arduino/arduino.php

РОЗДІЛ 3. ПРОГРАМНА РЕАЛІЗАЦІЯ ЕЛЕМЕНТІВ ЦИФРОВОЇ ЕЛЕКТРОНІКИ З ВИКОРИСТАННЯМ МІКРОПРОЦЕСОРНИХ СИСТЕМ

3.1. Основні визначення

У практиці інженерної діяльності часто мають місце ситуації, при яких має значення не рівень сигналів, що поступають з відповідних датчиків, а лише наявність чи відсутність таких сигналів. Наприклад, у системах охоронної сигналізації необхідно знати, замкнені чи не замкнені двері або вікна в приміщенні, що охороняється. У системах автоматики часто необхідно знати, чи не перевершує кількість рідини в цистерні заданий рівень, чи не є тиск у котлі нижчим визначеної межі, чи не перевершує температура в приміщенні задану величину і т. п.

Схеми, що дають можливість розв'язувати поставлені задачі, можуть описуватись виразами типу: "лампочка на пульті охоронної сигналізації горить, якщо всі вікна замкнені (точніше, замкнено перше **і** друге **і** третє **і**… вікно)". Або "лампочка не горить, якщо хоча б одне вікно відкрите (тобто може бути відкритим перше **або** друге **або** третє **або** перше **і** друге **або**...)". Такі вирази називаються *логічними*.

При проектуванні подібних систем задаються відповідним рівнем напруги живлення, і наявність чи відсутність її дає можливість одержувати відповіді на поставлені питання. Оскільки рівень напруги може бути різним і задаватись прийнятою елементною базою, то з метою формалізації опису подібних схем приймаються деякі умови. Як приклад, високий рівень напруги приймається за "**1**", низький – відповідно, за "**0**". У такому разі приведені вище вирази можуть бути формалізовані: якщо контакти, що фіксують положення вікон, позначити як аргументи x_1, x_2, …, x_n, які можуть приймати лише значення "**1**" або "**0**", то напругу на лампочці можемо розглядати як функцію y, яка теж приймає одне з двох аналогічних значень.

Математичний апарат, що оперує з аргументами та функціями, які набувають тільки двох значень – "**0**" та "**1**" – називається *двійковою* (*булевою*) *алгеброю* або *алгеброю логіки*. Такий математичний апарат для розв'язання задач формальної логіки розробив ірландський математик Дж. Буль.

Логічні змінні (аргументи), як і змінні звичайної алгебри, позначаються літерами латинського алфавіту з різними індексами – наприклад, x_0, x_1, x_2, x_3, … Індекс при змінній може одночасно означати розряд двійкового числа.

Якщо змінна x_i набуває значення $x_i = 1$, то таке її значення називають *істинним*. Протилежне $x_i = 0$ називають *хибним* і умовно позначають $\overline{x_i}$, що означає заперечення істинного значення аргументу.

Булева функція позначається літерою y і є двійковою функцією двійкових аргументів. Умовне її позначення

$$y = f(x_1, x_2, ..., x_n).$$

Булева функція, яка залежить від n аргументів, називається n-вимірною і є повністю визначеною, якщо вказані значення її для всіх двійкових наборів значень її аргументів. Кількість таких наборів дорівнює 2^n. *Тобто, областю визначеності функції* ***n*** *змінних є сукупність дискретних точок* ***n****-вимірного простору, причому кожна з*

точок є комбінацією значень цих змінних (*кодовою комбінацією*). Оскільки можливі 2^n різних комбінацій логічних змінних, то область визначення функції складається зі скінченої величини – 2^n точок. Це, в свою чергу, означає, що кожна функція може бути задана таблицею значень, які вона приймає в точках її області визначеності.

Усі можливі логічні функції *n* змінних можна створити за допомогою трьох основних операцій:

а) логічне заперечення (інверсія, операція **НІ**); позначається рискою над відповідною функцією або аргументом;

б) логічне додавання (диз'юнкція, операція **АБО**), яке позначається символами (V), (+);

в) логічне множення (кон'юнкція, операція **І**), яке позначається символами ($\wedge$), ($\cdot$), (&). Для позначення еквівалентності логічних виразів використовується знак (=).

Запереченням (*інверсією*) називається такий зв'язок між аргументом *x* та функцією *y*, при якому *y* істинна тоді і тільки тоді, коли значення *x* хибне, і навпаки.

Логічним множенням (*кон'юнкцією*) декількох змінних називається така функція, яка істинна тоді і тільки тоді, коли одночасно істинні всі логічні змінні.

Логічним додаванням (*диз'юнкцією*) декількох змінних називається така функція, яка хибна тоді і тільки тоді, коли одночасно хибні всі змінні, що додаються.

Слід пам'ятати, що операція кон'юнкції є старшою операцією і виконується раніше диз'юнкції.

Прикладом найпростіших функцій є наступні:

$$y_1 = \overline{x_1}; \qquad y_2 = x_1 \cdot x_2; \qquad y_3 = x_1 + \overline{x_2}.$$

На рис. 3.1 приведені умовні позначення, характерні для стандартів, прийнятих у країнах СНД: а – І; б – АБО; в – НІ; г – І-НІ; д – АБО-НІ; е – ВИКЛ.АБО. Використання приведених умовних позначень дає можливість будувати складні логічні та принципові схеми електронних пристроїв.

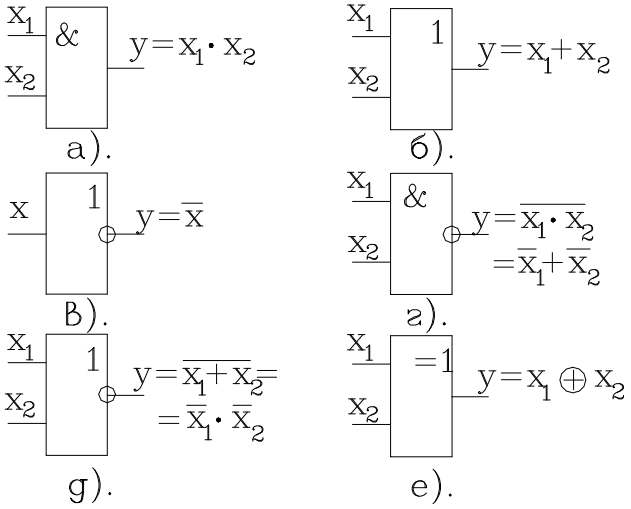


Рис. 3.1.

Кожен з цих логічних елементів може бути реалізований програмно на мікроконтролері. В наступних параграфах розглядаються особливості реалізації елементів цифрової схемотехніки на контролері Arduino.

3.2. Програмна реалізація базових логічних елементів

Логічний елемент – це електронний пристрій, що реалізує одну з логічних операцій. Логічні елементи являють собою електронні пристрої, у

яких оброблювана інформація закодована у вигляді двійкових чисел, відображуваних напругою (сигналом) високого і низького рівня. Всі базові логічні елементи можуть бути реалізовані на базі контролера Arduino. Для перевірки роботи програм, як і раніше, використовується система моделювання Proteus. Для реалізації прикладів, що наведено в цьому розділі, потрібно завантажити з мережі Internet бібліотеку контролера Arduino, як було описано в першому розділі. Якщо це не було ще зроблено, початком роботи слід завантажити бібліотеку з сайту:

https://www.theengineeringprojects.com/2015/12/arduino-uno-library-proteus.html

На рис. 3.2 представлено фрагмент сторінки цього сайту та позначено посилання, натиснувши на яке можна завантажити бібліотеку Arduino для Proteus.

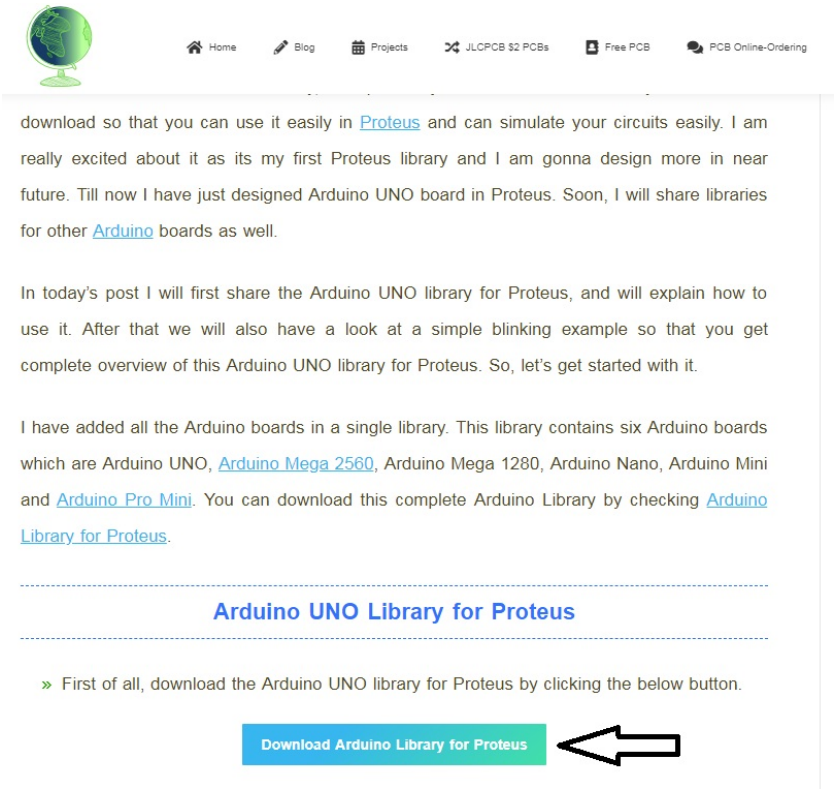


Рис. 3.2.

Після завантаження архіву з файлами моделей його необхідно розпакувати. В результаті в каталозі, в який відбувається розпаковка архіву, з'явиться 2 файли: ArduinoUnoTEP.LIB та ArduinoUnoTEP.IDX. Ці файли необхідно скопіювати в каталог з бібліотеками Proteus, який називається Library. На рис. 3.3 показано розташування каталогу Library та файли бібліотеки, які було переміщено в цей каталог.

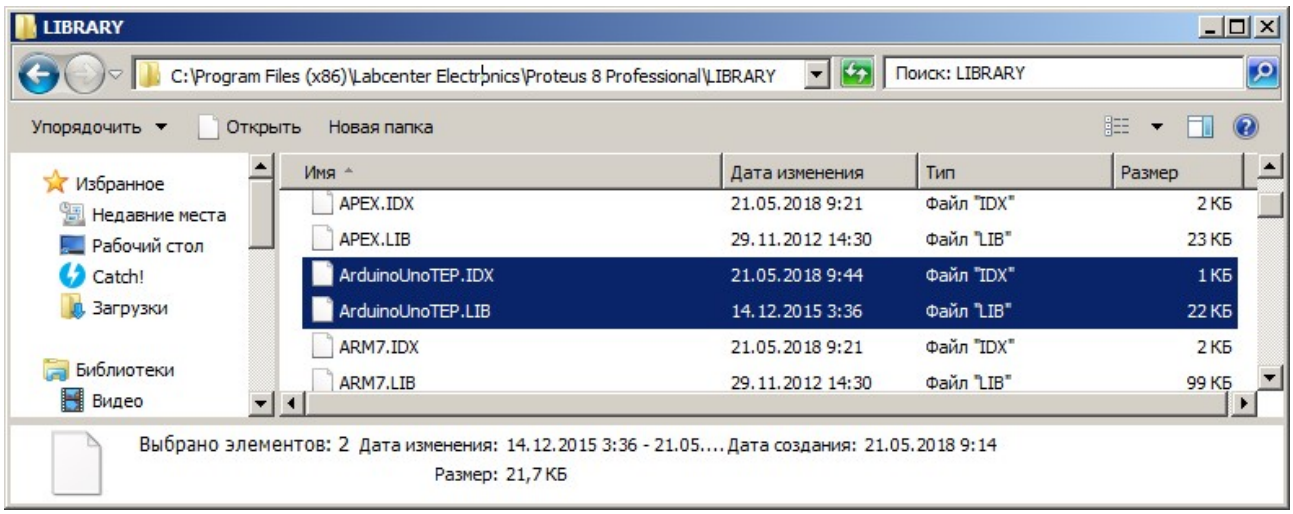


Рис. 3.3.

Детальну інструкцію по підключенню в Proteus бібліотек різних модулів для роботи з Arduino можна знайти за посиланням: http://www.instructables.com/id/How-to-add-Arduino-Library-in-to-Proteus-7-8/

3.2.1. Логічний елемент І

Для моделювання роботи Arduino в середовищі Proteus спочатку потрібно скласти схему. Для цього необхідно запустити середовище моделювання Proteus, і в головному вікні програми натиснути кнопку з літерою «P» для додавання елемента в схему, як показано на рис. 3.4.



Рис. 3.4.

У вікні додавання електронного компонента, що з’явиться, в списку категорій ліворуч потрібно обрати категорію Arduino, а в списку праворуч модель пристрою, як показано на рис. 3.5. В даному випадку це Arduino UNO.

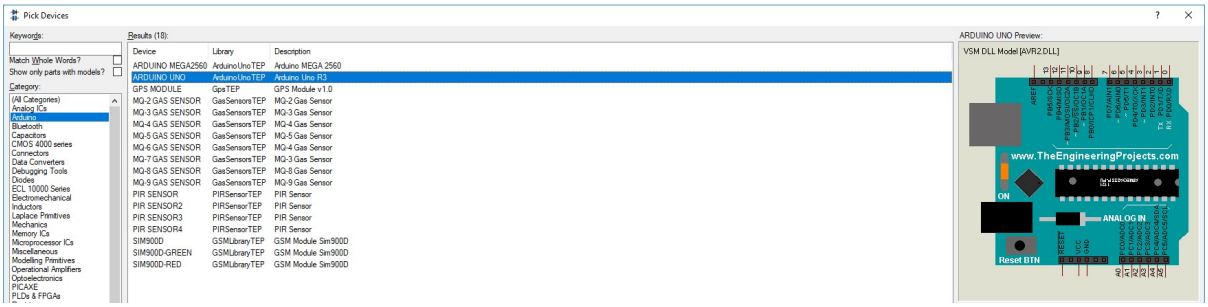


Рис. 3.5.

Аналогічне потрібно виконати для додавання в схему світлодіода та резистора. Пришвидшити процес пошуку елемента в бібліотеці можна шляхом введення його назви в поле Keywords у вікні додавання компонента (рис. 3.5). Це поле вводу знаходиться зверху ліворуч. Для пошуку резистора потрібно ввести RES, а для пошуку світлодіода потрібно ввести LED-GREEN. Для задання логічних рівнів на входах контролера Arduino в моделі використовується елемент LOGICTOGGLE (рис. 3.6).

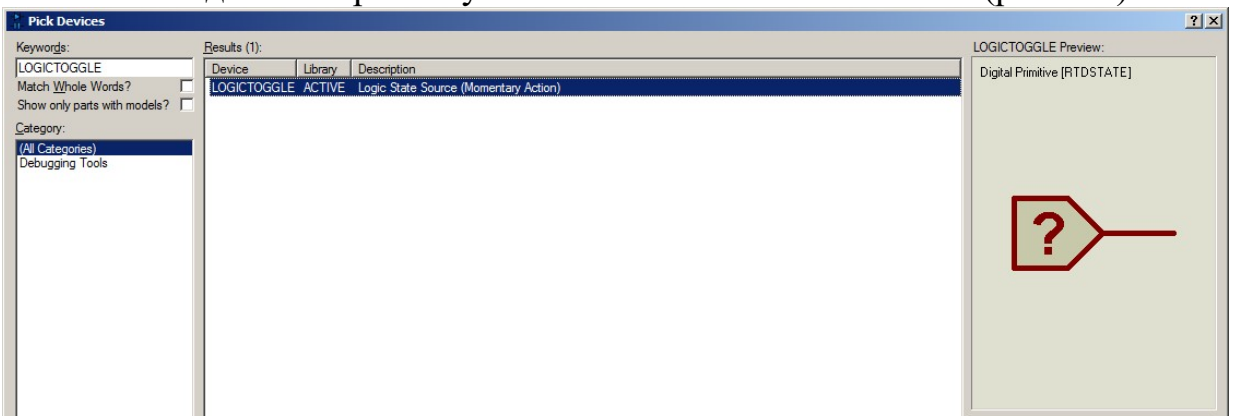


Рис. 3.6.

Для додавання в схему джерела живлення та загального провідника потрібно на робочому полі середовища Proteus натиснути праву кнопку миші, і з контекстного меню, що з’явиться, вибрати пункт Place – Terminal – GROUND, як показано на рис. 3.7.

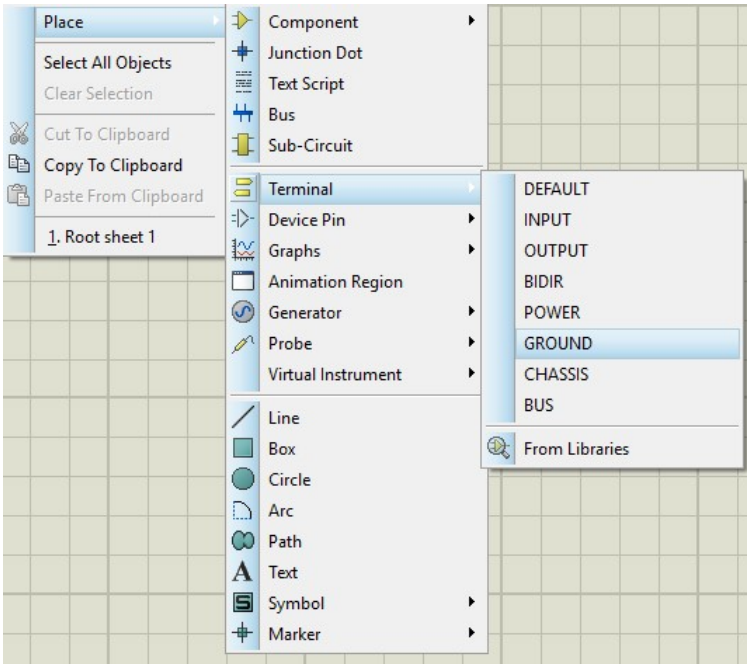


Рис. 3.7.

Далі ці елементи потрібно розташувати на робочому полі та з’єднати, як показано на рис. 3.8.

Логічний елемент, який реалізує функцію кон'юнкції, називається схемою збігу. Мнемонічне правило для *І* з будь-якою кількістю входів звучить так: На виході буде:

– «1» тоді і тільки тоді, коли на всіх входах діють «1»,

– «0» тоді і тільки тоді, коли хоча б на одному вході діє «0».

Словесно цю операцію можна виразити таким виразом: "Істина на виході може бути при істині на вході 1 та істині на вході 2".

На рис. 3.8 представлено схему, що використовується для дослідження базових логічних елементів.

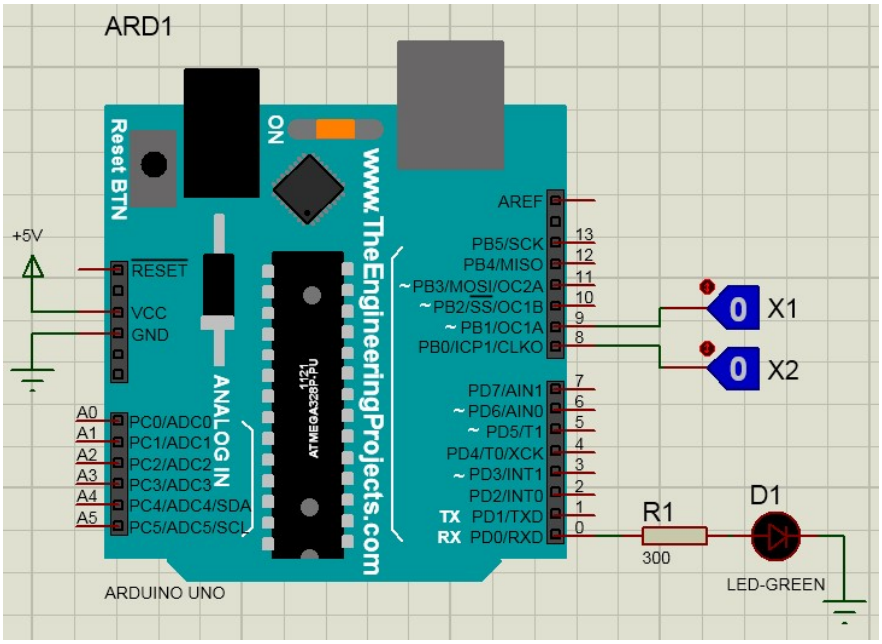


Рис. 3.8.

Програмна реалізація логічного елементу *І* представлена нижче.

```
// оголошення змінної, в якій зберігається номер виводу, до якого підключено
світлодіод
int led_Pin=0;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X1
int x1_Pin=9;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X2
int x2_Pin=8;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X1
int x1;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X2
int x2;
// оголошення змінної, в якій буде зберігатися результат виконання логічної операції
int y;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // вивід, до якого підключено світлодіод, налаштовується на вихід
  pinMode(led_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів (кнопки)
  // налаштовуються на вхід
  pinMode(x1_Pin, INPUT);
  pinMode(x2_Pin, INPUT);
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну x1 стану дискретного входу з номером 9,
  // що зберігається в змінній x1_Pin
  x1=digitalRead(x1_Pin);

  // зчитування в змінну x2 стану дискретного входу з номером 8,
  // що зберігається в змінній x2_Pin
  x2=digitalRead(x2_Pin);

  // перевірка значень, що зберігаються в змінних x1 та x2
  // якщо в обох змінних міститься значення HIGH, тобто на входи контролера
  // з номерами 8 та 9 подається лог.1 (в даному випадку напруга живлення +5В)
  if((x1==HIGH) && (x2==HIGH)) {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється високий рівень сигналу HIGH,
    // тобто на цьому виводі з'явиться напруга +5В, і світлодіод загориться
    digitalWrite(led_Pin, HIGH);
    }
    // в протилежному випадку, якщо хоча б на одному з входів є низький рівень сигналу
    else {
```

```
  // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
  // встановлюється низький рівень сигналу LOW,
  // тобто на цьому виводі з'явиться напруга 0В, і світлодіод згасне
      digitalWrite(led_Pin, LOW);
    }
  // далі відбувається циклічне виконання цієї функції з початку
}
```

Текст програми потрібно набрати в середовищі Arduino IDE та зберегти на диск під назвою «logic». Для компіляції програми в середовищі Arduino IDE потрібно натиснути на кнопку «Проверить». Її розташування показано стрілкою на рис. 3.9. Після компіляції програми у вікні повідомлень з’явиться інформація про розташування файлу-прошивки для мікроконтролера, який має розширення hex. За замовчуванням при компіляції програми в середовищі Arduino такий файл не створюється. Тому потрібно дещо змінити налаштування програми. В меню Файл потрібно вибрати пункт Налаштування. У вікні, що з’явиться, потрібно встановити позначку Компіляція, як показано на рис. 3.10.

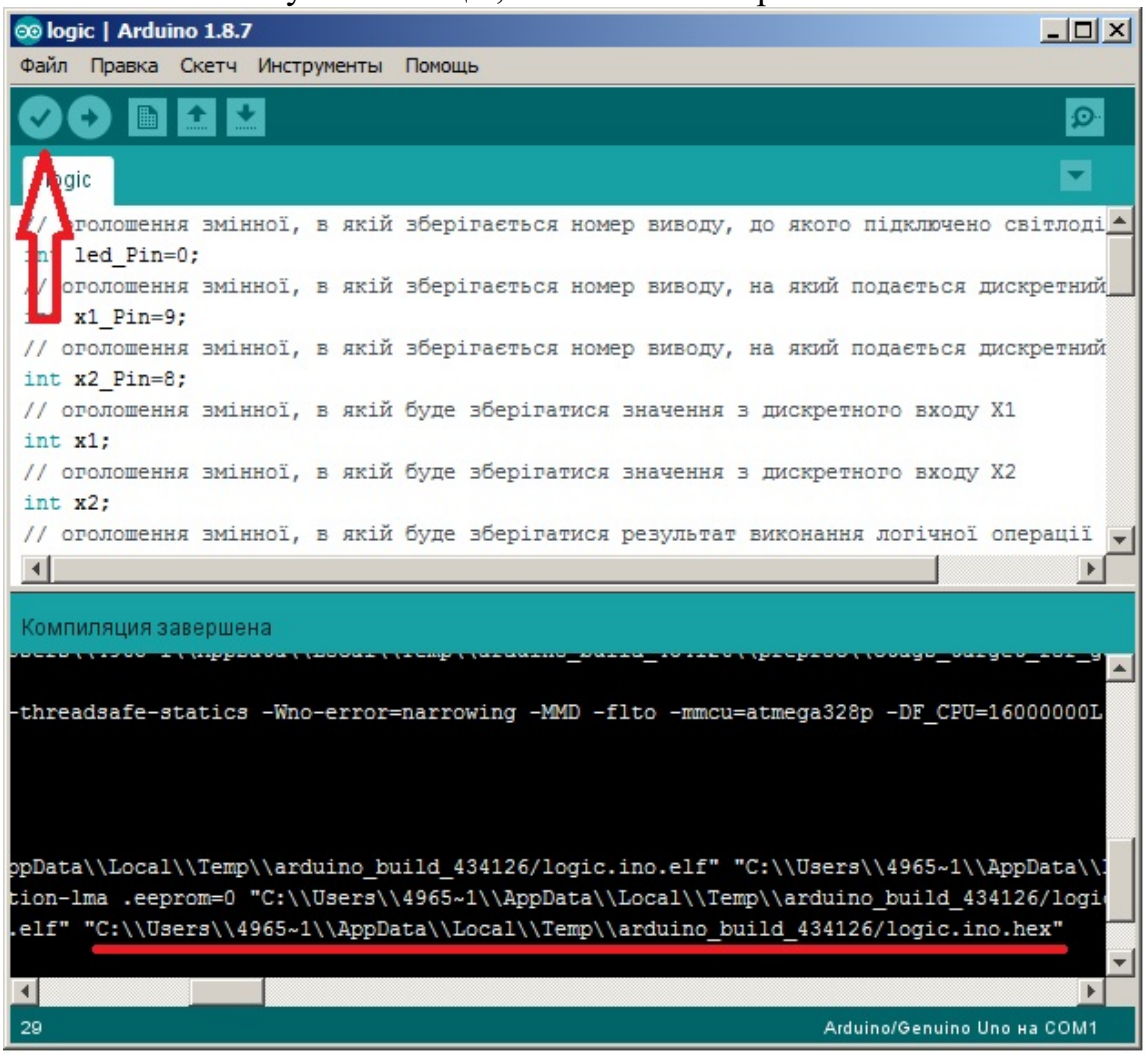


Рис. 3.9.

Тепер, після компіляції програми у програмному забезпеченні Arduino, у вікні повідомлень у нижній частині вікна програми з’явиться додаткова інформація, а саме інформація про місце розташування файлу прошивки. В даному випадку це файл «logic.ino.hex (на рис. 3.9 шлях до цього файлу підкреслено лінією).

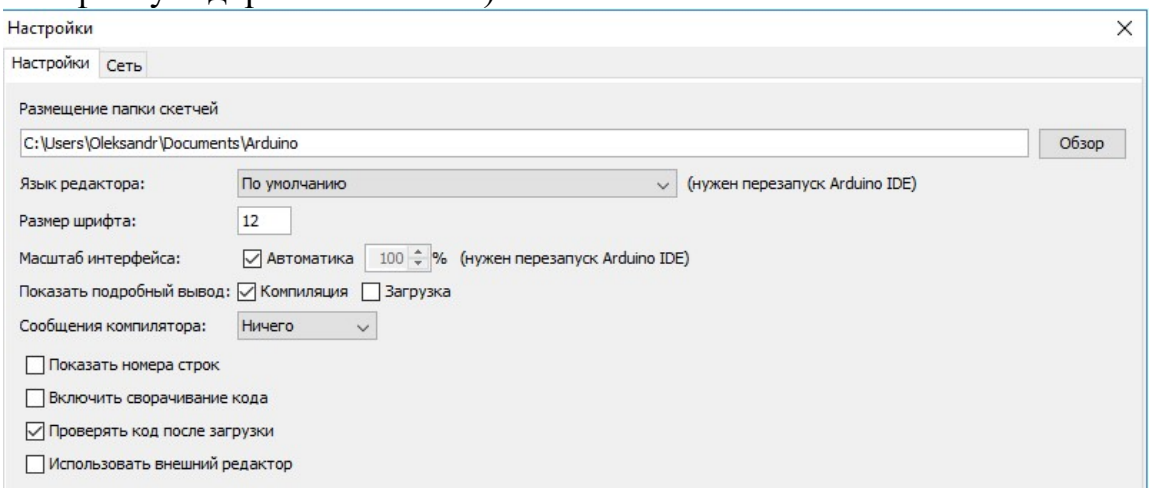


Рис. 3.10.

На рис. 3.11 представлене вікно властивостей моделі плати Arduino UNO в середовищі Proteus, де потрібно вказати шлях до файлу «прошивки», що має розширення .hex.

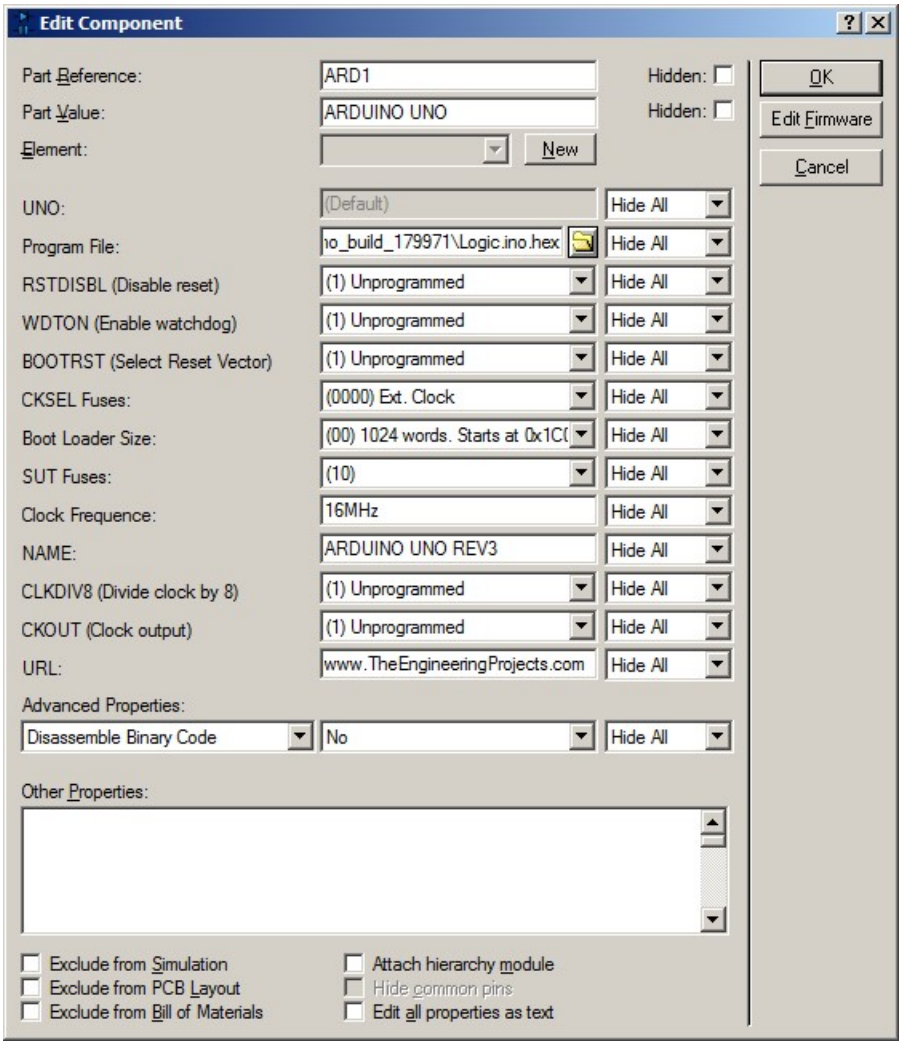


Рис. 3.11.

Тепер схему можна моделювати. Для цього потрібно натиснути на кнопку «Run the simulation» із зображенням стрілки, яка розташована в правому нижньому куті головного вікна програми (перша кнопка на рис. 3.12).



Рис. 3.12.

Після запуску процесу моделювання у вікні Proteus можна спостерігати зміну стану світлодіода при натисканні на кнопки X1 та X2, як показано на рис. 3.13. Моделювання роботи схеми відбувається в режимі реального часу. Рівні сигналів позначаються різними кольорами: низький рівень (лог. 0) – синій колір, високий рівень (лог. 1) – червоний колір.

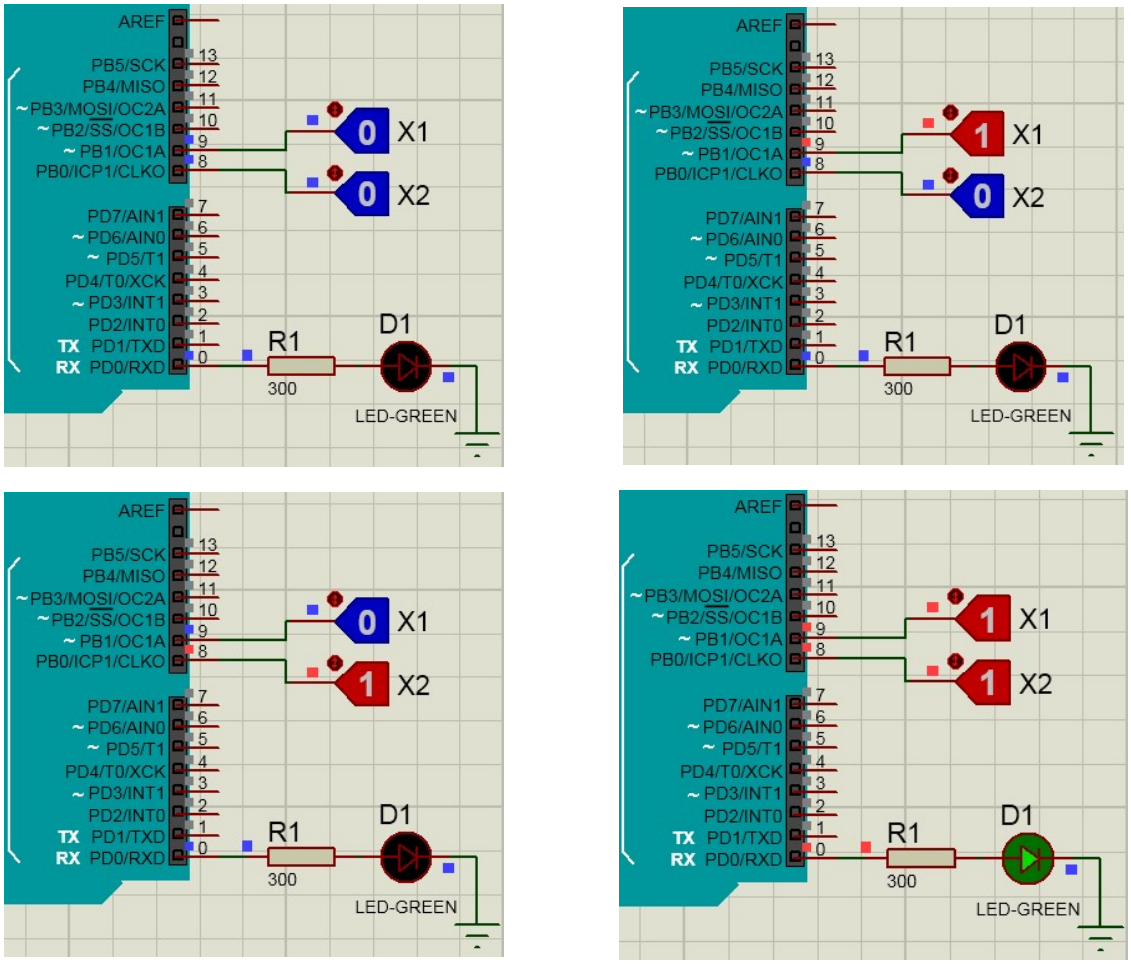


Рис. 3.13.

Як видно з рис. 3.13, на виході контролера Arduino з номером 0, до якого підключено світлодіод, з'являється високий рівень сигналу (лог. 1) лише в тому випадку, коли на обидва входи (з номерами 8 та 9) подаються сигнали високого рівня. Про це свідчить зміна стану світлодіода. В іншому випадку на виході встановлюється низький рівень сигналу (лог. 0). Результати моделювання свідчать про те, що розроблена програма дійсно реалізує логіку роботи базового логічного елемента *І*.

На рис. 3.14 представлено модель пристрою для дослідження програм, що реалізують функціональність базових логічних елементів, в якій замість елементів LOGICTOGGLE підключено кнопки BUTTON. Особливістю цієї схеми є те, що у стані, коли кнопки не натиснуті, на обидва входи контролера Arduino (з номерами 8 та 9) надходять сигнали високого рівня через те, що ці виводи «підтягнуті» до лінії живлення за допомогою резисторів R2 та R3. Саме тому світлодіод світиться. При натисканні кнопки на відповідний вхід подається сигнал низького рівня, і на виході контролера також з'явиться сигнал низького рівня.

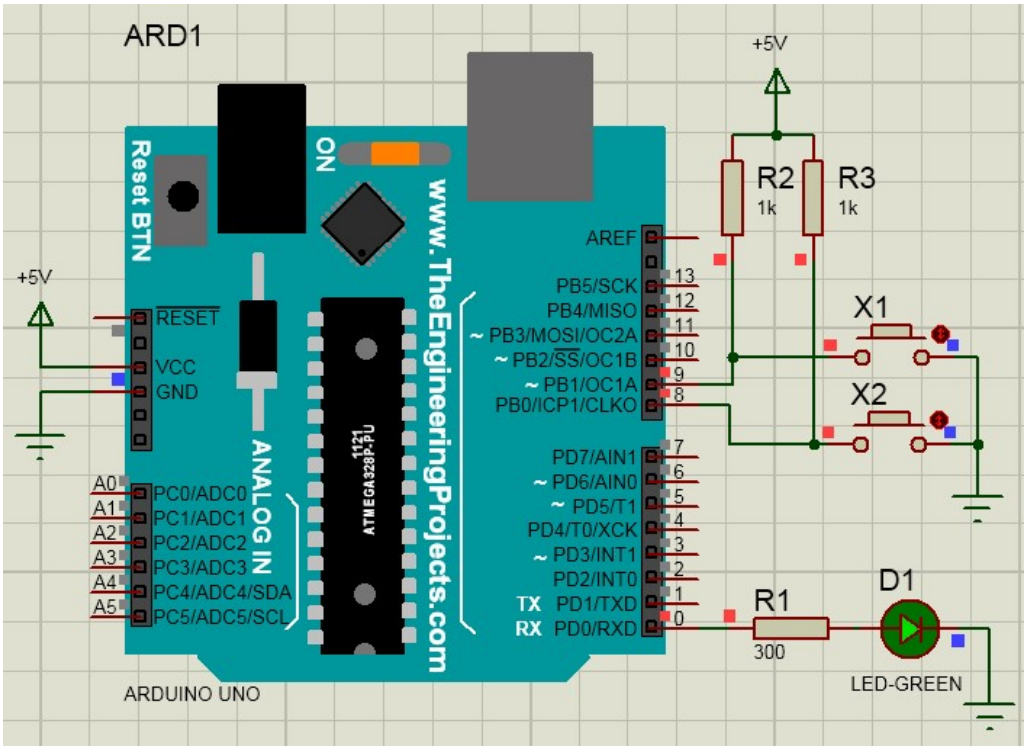


Рис. 3.14.

Для дослідження роботи програми на реальному пристрої, контролер Arduino потрібно підключити до комп'ютера за допомогою USB-кабеля, і натиснути на кнопку «Загрузить» в середовищі Arduino IDE.

3.2.2. Логічний елемент АБО

Мнемонічне правило для АБО з будь-якою кількістю входів звучить так: На виході буде:

– «1» тоді і тільки тоді, коли хоча б на одному вході діє «1»,

– «0» тоді і тільки тоді, коли на всіх входах діють «0».

Текст програми, що реалізує базовий логічний елемент 2-АБО представлено нижче.

```
// оголошення змінної, в якій зберігається номер виводу, до якого підключено світлодіод
int led_Pin=0;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний сигнал X1
int x1_Pin=9;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний сигнал X2
int x2_Pin=8;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X1
int x1;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X2
int x2;
// оголошення змінної, в якій буде зберігатися результат виконання логічної операції
int y;
```

```
// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
  // вивід, до якого підключено світлодіод, налаштовується на вихід
  pinMode(led_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів (кнопки)
  // налаштовуються на вхід
  pinMode(x1_Pin, INPUT);
  pinMode(x2_Pin, INPUT);
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну x1 стану дискретного входу з номером 9,
  // що зберігається в змінній x1_Pin
  x1=digitalRead(x1_Pin);

  // зчитування в змінну x2 стану дискретного входу з номером 8,
  // що зберігається в змінній x2_Pin
  x2=digitalRead(x2_Pin);

  // перевірка значень, що зберігаються в змінних x1 та x2
  // якщо хоча б в одній змінній міститься значення HIGH, або відразу в обох змінних,
  // міститься значення HIGH, тобто на входи контролера
  // з номерами 8 та 9 подається лог.1 (в даному випадку напруга живлення +5В)
  if((x1==HIGH) || (x2==HIGH)) {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється високий рівень сигналу HIGH,
    // тобто на цьому виводі з'явиться напруга +5В, і світлодіод загориться
    digitalWrite(led_Pin, HIGH);
    }
    // в протилежному випадку, якщо на обох входах присутній низький рівень сигналу
    else {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється низький рівень сигналу LOW,
    // тобто на цьому виводі з'явиться напруга 0В, і світлодіод згасне
        digitalWrite(led_Pin, LOW);
      }
    // далі відбувається циклічне виконання цієї функції з початку
}
```

Як і в попередньому прикладі, цю програму спочатку потрібно відкомпілювати, і в середовищі моделювання Proteus у вікні властивостей моделі контролера Arduino вказати шлях до отриманого файлу прошивки, що має розширення .hex. Результати моделювання роботи пристрою, що реалізує базовий логічний елемент 2-АБО, та перевірки його функціональності, представлено на рис. 3.15.

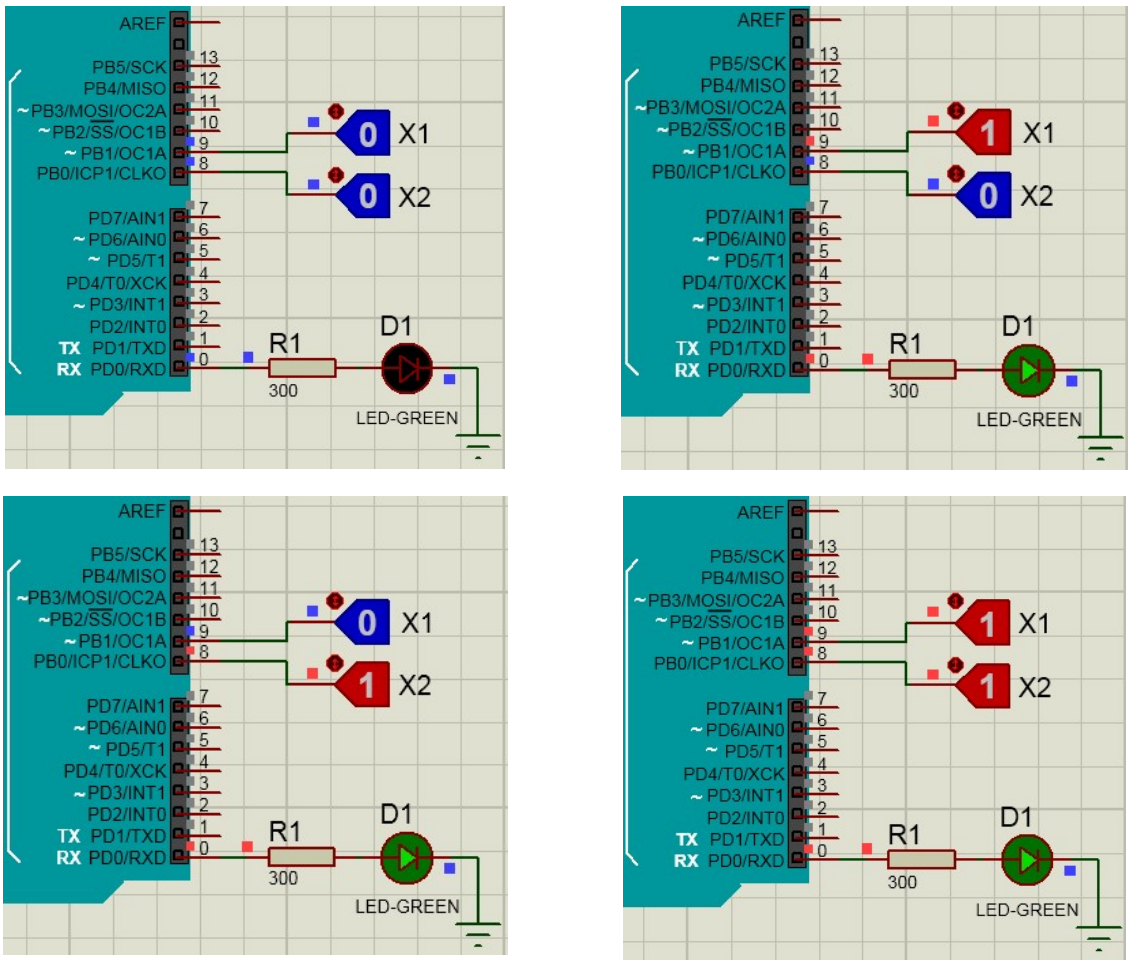


Рис. 3.15.

Аналіз результатів моделювання показує, що на виході контролера, до якого підключено світлодіод, з'являється високий рівень в тому випадку, коли хоча б на одному вході присутній високий рівень сигналу. Це дійсно відповідає логіці роботи базового логічного елемента АБО.

3.2.3. Логічний елемент ВИКЛ.АБО

Мнемонічне правило для ВИКЛЮЧНЕ_АБО з будь-якою кількістю входів звучить так: На виході буде:

– «1» тоді і тільки тоді, коли на вході діє непарна кількість,

– «0» тоді і тільки тоді, коли на вході діє парна кількість.

Словесний опис: «істина на виході – тільки при істині на вході 1, або тільки при істині на вході 2».

Текст програми, що реалізує базовий логічний елемент ВИКЛ.АБО представлено нижче.

```
// оголошення змінної, в якій зберігається номер виводу, до якого підключено світлодіод
int led_Pin=0;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний сигнал X1
int x1_Pin=9;
```

```
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний сигнал X2
int x2_Pin=8;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X1
int x1;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X2
int x2;
// оголошення змінної, в якій буде зберігатися результат виконання логічної операції
int y;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
  // вивід, до якого підключено світлодіод, налаштовується на вихід
  pinMode(led_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів (кнопки)
  // налаштовуються на вхід
  pinMode(x1_Pin, INPUT);
  pinMode(x2_Pin, INPUT);
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну x1 стану дискретного входу з номером 9,
  // що зберігається в змінній x1_Pin
  x1=digitalRead(x1_Pin);

  // зчитування в змінну x2 стану дискретного входу з номером 8,
  // що зберігається в змінній x2_Pin
  x2=digitalRead(x2_Pin);

  // перевірка значень, що зберігаються в змінних x1 та x2
  // якщо лише в одній змінній міститься значення HIGH, і тільки в одній,
  // тобто на входи контролера з номерами 8 та 9 подається лог.1
  // (в даному випадку напруга живлення +5В)
  if(((x1==HIGH) && (x2==LOW)) || (((x1==LOW) && (x2==HIGH)))) {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється високий рівень сигналу HIGH,
    // тобто на цьому виводі з'явиться напруга +5В, і світлодіод загориться
    digitalWrite(led_Pin, HIGH);
    }
    // в протилежному випадку, якщо на обох входах присутній низький рівень сигналу
    else {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється низький рівень сигналу LOW,
    // тобто на цьому виводі з'явиться напруга 0В, і світлодіод згасне
        digitalWrite(led_Pin, LOW);
      }
    // далі відбувається циклічне виконання цієї функції з початку
}
```

Результати моделювання роботи пристрою, що реалізує базовий логічний елемент ВИКЛ.АБО, та перевірки його функціональності представлено на рис. 3.16.

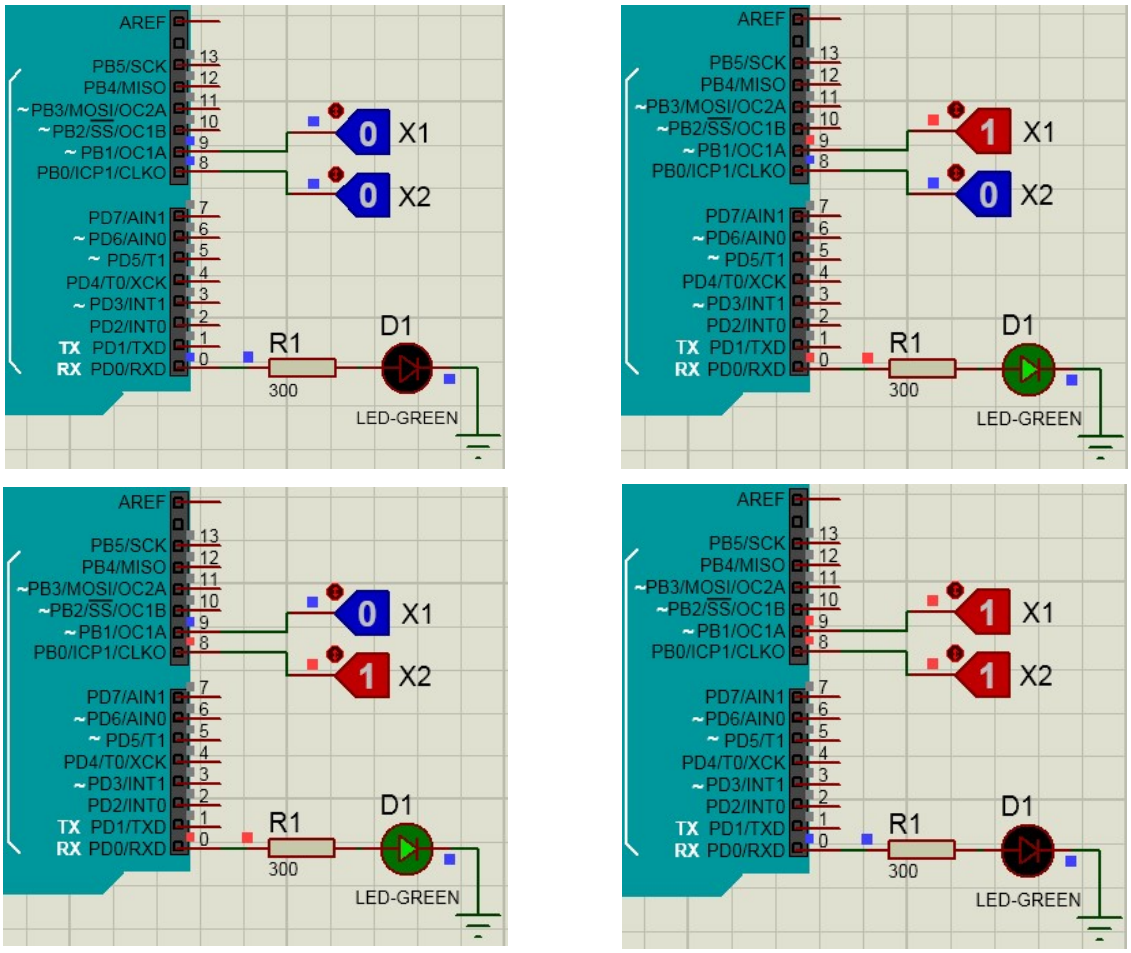


Рис. 3.16.

Як видно з рис. 3.16, при наявності лог.1 виключно на одному з двох входів, X1 або X2, на виході контролера, до якого підключено світлодіод, з'явиться високий рівень сигналу. Якщо на обидві входи контролера поступають сигнали однакового рівня, на виході формується низький рівень сигналу.

3.3. Програмна реалізація комбінаційних автоматів

Розглянемо наступний приклад. Необхідно записати вираз функції трьох змінних, яка приймає істинні значення за умови, що будь-яка пара змінних одночасно має істинні значення. Таке завдання може з'явитися при розробці системи голосування, де рішення приймається більшістю голосів.

Введемо умовні позначення змінних x_0, x_1, x_2.

У загальному плані функція матиме вигляд:

$$y = f(x_2, x_1, x_0).$$

Оскільки істинні значення функції визначаються будь-якою парою логічних змінних, тобто або x_0 і x_1 , або x_0 і x_2 , або x_1 і x_2 , то аналітична форма запису функції прийме вигляд:

$$y_1 = x_0 \cdot x_1 + x_0 \cdot x_2 + x_1 \cdot x_2 .$$

Отже, з аналізу цього виразу видно, що для реалізації цього завдання потрібно використовувати логічні операції І (символ « · ») та АБО (символ «+»).

Як вже було продемонстровано, для реалізації логічної операції І використовуються символи &&, а для реалізації логічної операції АБО використовуються символи ||.

На рис. 3.17 представлено схему моделі пристрою для перевірки роботи програми. Як і в попередніх прикладах, для моделювання зміни рівня сигналу на входах контролера Arduino використовуються елементи LOGICTOGGLE, підключені відповідно до входів з номерами 8, 9 та 10. При фізичній реалізації пристрою до цих входів підключається джерело вхідних сигналів, в якості якого може виступати інша електрона система.

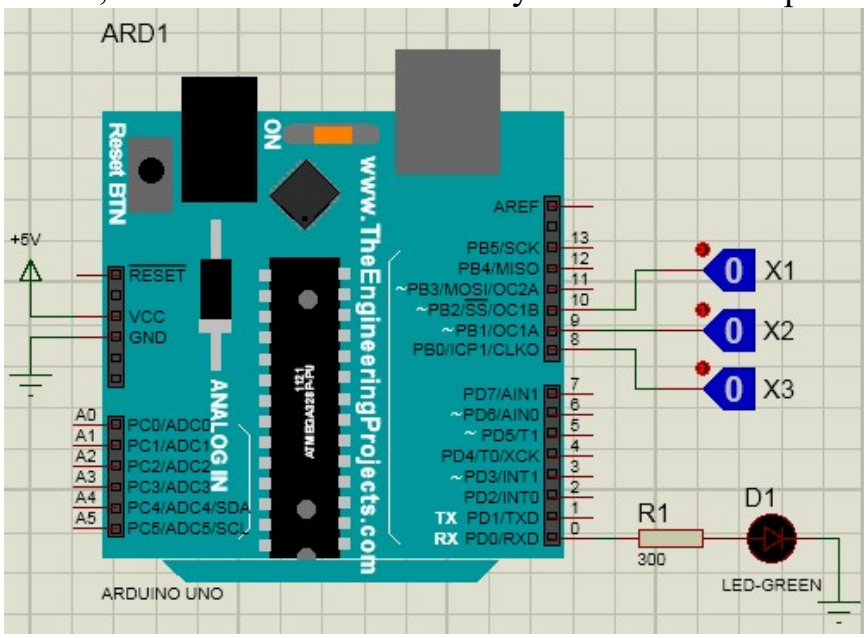


Рис. 3.17.

Текст програми, в якій вирішується поставлене завдання, представлено нижче.

```
// оголошення змінної, в якій зберігається номер виводу, до якого підключено
світлодіод
int led_Pin=0;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X1
int x1_Pin=10;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X2
int x2_Pin=9;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X3
```

```
int x3_Pin=8;

// оголошення змінної, в якій буде зберігатися значення з дискретного входу X1
int x1;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X2
int x2;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X3
int x3;
// оголошення змінної, в якій буде зберігатися результат виконання логічної операції
int y;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // вивід, до якого підключено світлодіод, налаштовується на вихід
  pinMode(led_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів (кнопки)
  // налаштовуються на вхід
  pinMode(x1_Pin, INPUT);
  pinMode(x2_Pin, INPUT);
  pinMode(x3_Pin, INPUT);
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну x1 стану дискретного входу з номером 10,
  // що зберігається в змінній x1_Pin
  x1=digitalRead(x1_Pin);

  // зчитування в змінну x2 стану дискретного входу з номером 9,
  // що зберігається в змінній x2_Pin
  x2=digitalRead(x2_Pin);

  // зчитування в змінну x3 стану дискретного входу з номером 8,
  // що зберігається в змінній x3_Pin
  x3=digitalRead(x3_Pin);

  // перевірка значень, що зберігаються в змінних x1, x2 та x3
  // якщо в будь-яких двох змінних міститься значення HIGH,
  // тобто хоча б на два з трьох входів контролера з номерами 8, 9 та 10 подається лог.1
  // (в даному випадку напруга живлення +5В)
  if(((x1==HIGH) && (x2==HIGH)) || ((x2==HIGH) && (x3==HIGH)) || ((x1==HIGH) &&
(x3==HIGH))) {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється високий рівень сигналу HIGH,
    // тобто на цьому виводі з'явиться напруга +5В, і світлодіод загориться
    digitalWrite(led_Pin, HIGH);
    }
    // в протилежному випадку
    else {
    // на виході 0 (змінна led_Pin), до якого підключено світлодіод,
```

```
  // встановлюється низький рівень сигналу LOW,
  // тобто на цьому виводі з'явиться напруга 0В, і світлодіод згасне
      digitalWrite(led_Pin, LOW);
    }
  // далі відбувається циклічне виконання цієї функції з початку
}
```

Після того, як програма буде відкомпільована в середовищі Arduino IDE, можна виконати моделювання роботи пристрою та перевірити його функціональність. Результати моделювання представлено на рис. 3.18.

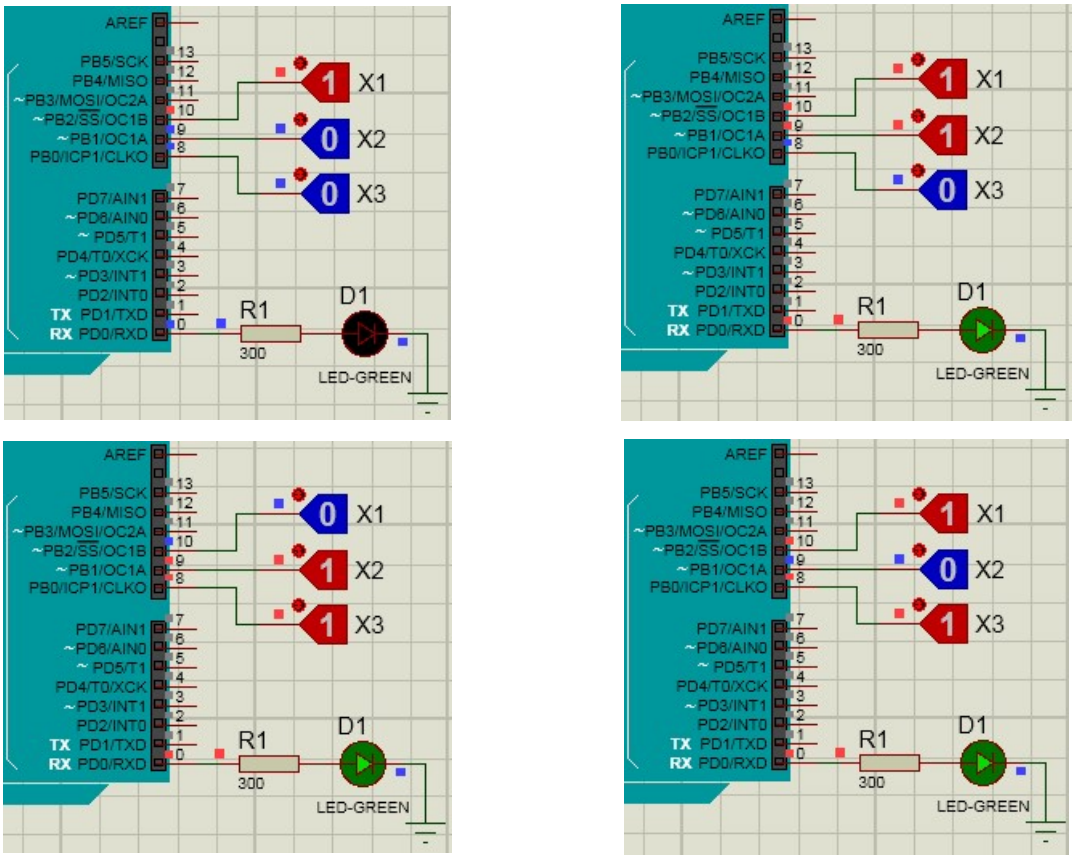


Рис. 3.18.

Отже, як видно з наведених рисунків, на виході контролера з’являється високий рівень, якщо хоча б на двох з трьох входів присутній сигнал високого рівня, що відповідає поставленому завданню.

Розглянемо наступний приклад. Змагання зі штанги судять три судді: головний, що знаходиться проти помосту, і два бокові. Якщо суддя вважає, що вага взята, він натискає кнопку, яка знаходиться на його столі. Для спортсмена вага вважається взятою, якщо загорається лампочка біля помосту. Умова загорання лампочки наступна: вона загоряється, якщо головний і, як мінімум, один з бокових суддів натиснули кнопки на своїх столах. Формалізувати умову загорання лампочки.

Приймемо за логічні змінні кнопки, що знаходяться на столах суддів: x_1 – кнопка головного судді; x_2 та x_3 – бокових. Приймемо за істинне

значення натиснуту кнопку x_i = 1. Тоді умова загорання лампочки формально може бути записана в вигляді:

$$y = x_1\ x_2\ x_3 + x_1\ x_2\ \overline{x_3} + x_1\ \overline{x_2}\ x_3 .$$

Для реалізації системи використовується принципова схема з попереднього прикладу. Програмна реалізація цієї функції має вигляд:

```
// оголошення змінної, в якій зберігається номер виводу, до якого підключено
світлодіод
int led_Pin=0;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X1
int x1_Pin=10;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X2
int x2_Pin=9;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X3
int x3_Pin=8;

// оголошення змінної, в якій буде зберігатися значення з дискретного входу X1
int x1;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X2
int x2;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X3
int x3;
// оголошення змінної, в якій буде зберігатися результат виконання логічної операції
int y;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // вивід, до якого підключено світлодіод, налаштовується на вихід
  pinMode(led_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів (кнопки)
  // налаштовуються на вхід
  pinMode(x1_Pin, INPUT);
  pinMode(x2_Pin, INPUT);
  pinMode(x3_Pin, INPUT);
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну x1 стану дискретного входу з номером 10,
  // що зберігається в змінній x1_Pin
  x1=digitalRead(x1_Pin);

  // зчитування в змінну x2 стану дискретного входу з номером 9,
  // що зберігається в змінній x2_Pin
  x2=digitalRead(x2_Pin);
```

```
  // зчитування в змінну x3 стану дискретного входу з номером 8,
  // що зберігається в змінній x3_Pin
  x3=digitalRead(x3_Pin);

  // перевірка значень, що зберігаються в змінних x1, x2 та x3
  // якщо в змінній x1 міститься значення HIGH, і в будь-якій з двох змінних x2 або x3
  // також міститься значення HIGH, тобто на цих входах присутній рівень лог.1
    if(((x1==HIGH) && (x2==HIGH) && (x3==HIGH)) || ((x1==HIGH) && (x2==HIGH)
&& (x3==LOW)) || ((x1==HIGH) && (x2==LOW) && (x3==HIGH))) {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється високий рівень сигналу HIGH,
    // тобто на цьому виводі з'явиться напруга +5В, і світлодіод загориться
    digitalWrite(led_Pin, HIGH);
    }
    // в протилежному випадку
    else {
    // на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється низький рівень сигналу LOW,
    // тобто на цьому виводі з'явиться напруга 0В, і світлодіод згасне
        digitalWrite(led_Pin, LOW);
      }
    // далі відбувається циклічне виконання цієї функції з початку
}
```

Після компіляції програми в середовищі Arduino IDE та моделюванні роботи системи можна зробити висновок, що алгоритм функціонування програми відповідає поставленому завданню.

Розглянемо ще один приклад. Формалізувати та записати у вигляді булевих функцій висловлення: лампочка охоронної сигналізації світиться, коли всі три двері приміщення зачинені.

Позначимо логічні змінні x_1, x_2, x_3 істинними, якщо відповідні двері зачинені. В такому випадку істинне значення функції (“лампочка сигналізації світиться”) визначається за формулою:

$$y_2 = x_1 \cdot x_2 \cdot x_3 .$$

Принципова схема для реалізації цієї системи така сама, як і в попередніх двох прикладах. Текст програмної реалізації наведеної логічної функції представлено нижче.

```
// оголошення змінної, в якій зберігається номер виводу, до якого підключено
світлодіод
int led_Pin=0;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X1
int x1_Pin=10;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X2
int x2_Pin=9;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X3
int x3_Pin=8;
```

```
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X1
int x1;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X2
int x2;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X3
int x3;
// оголошення змінної, в якій буде зберігатися результат виконання логічної операції
int y;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // вивід, до якого підключено світлодіод, налаштовується на вихід
  pinMode(led_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів (кнопки)
  // налаштовуються на вхід
  pinMode(x1_Pin, INPUT);
  pinMode(x2_Pin, INPUT);
  pinMode(x3_Pin, INPUT);
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну x1 стану дискретного входу з номером 10,
  // що зберігається в змінній x1_Pin
  x1=digitalRead(x1_Pin);

  // зчитування в змінну x2 стану дискретного входу з номером 9,
  // що зберігається в змінній x2_Pin
  x2=digitalRead(x2_Pin);

  // зчитування в змінну x3 стану дискретного входу з номером 8,
  // що зберігається в змінній x3_Pin
  x3=digitalRead(x3_Pin);

  // перевірка значень, що зберігаються в змінних x1, x2 та x3
  // якщо в усіх трьох змінних міститься значення HIGH
  // тобто на входах 8, 9 та 10 контролера присутній рівень лог.1
    if(((x1==HIGH) && (x2==HIGH) && (x3==HIGH))) {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється високий рівень сигналу HIGH,
    // тобто на цьому виводі з'явиться напруга +5В, і світлодіод загориться
    digitalWrite(led_Pin, HIGH);
    }
    // в протилежному випадку
    else {
    // на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється низький рівень сигналу LOW,
    // тобто на цьому виводі з'явиться напруга 0В, і світлодіод згасне
        digitalWrite(led_Pin, LOW);
```

```
    }
  // далі відбувається циклічне виконання цієї функції з початку
}
```

Результати моделювання роботи пристрою представлено на рис. 3.19. З рисунків видно, що лише за умови, коли на всіх трьох входах контролера Arduino (з номерами 8, 9 та 10) присутній рівень лог. 1, на виході контролера з номером 0 також формується високий рівень сигналу.

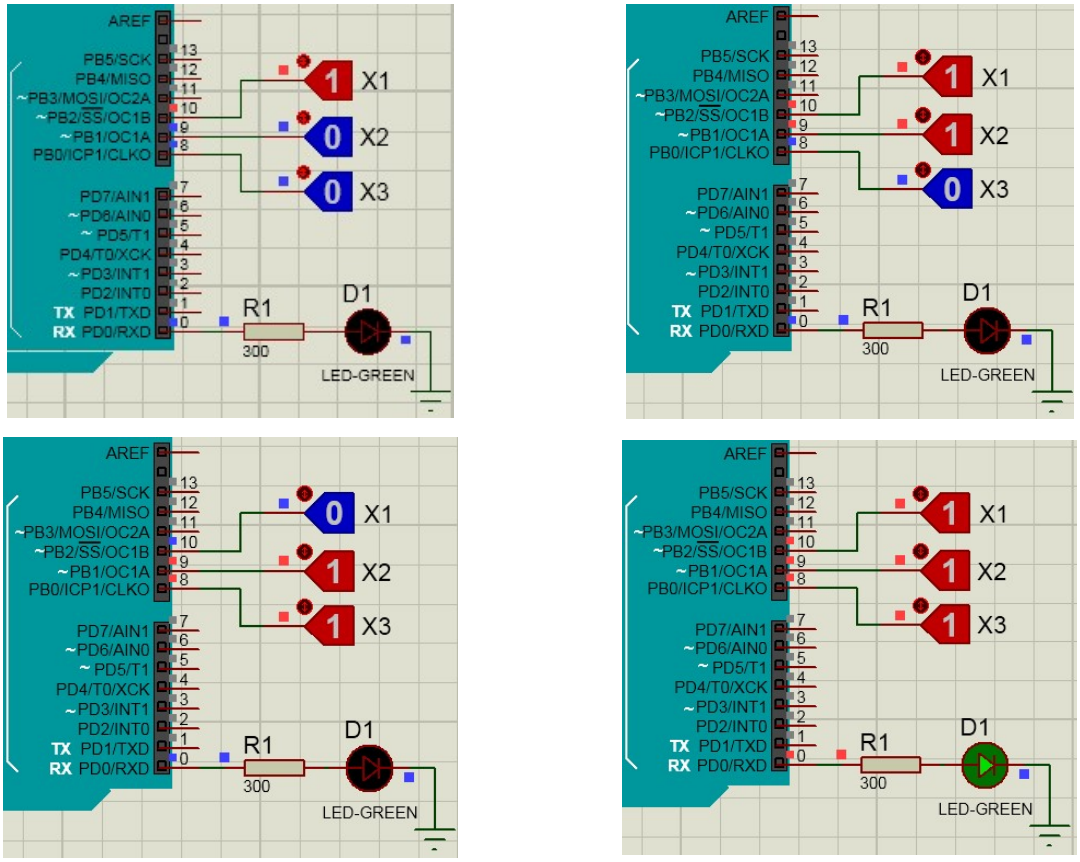


Рис. 3.19.

В наступному прикладі виконується програмна реалізація логічної функції, яка задана рівнянням:

$$y = x_1 \cdot \overline{x_2} + (\overline{x_2} + \overline{x_3}) \cdot \overline{x_1}.$$

Текст програми має вигляд:

```
// оголошення змінної, в якій зберігається номер виводу, до якого підключено
світлодіод
int led_Pin=0;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X1
int x1_Pin=10;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X2
int x2_Pin=9;
// оголошення змінної, в якій зберігається номер виводу, на який подається дискретний
сигнал X3
```

```
int x3_Pin=8;

// оголошення змінної, в якій буде зберігатися значення з дискретного входу X1
int x1;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X2
int x2;
// оголошення змінної, в якій буде зберігатися значення з дискретного входу X3
int x3;
// оголошення змінної, в якій буде зберігатися результат виконання логічної операції
int y;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // вивід, до якого підключено світлодіод, налаштовується на вихід
  pinMode(led_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів (кнопки)
  // налаштовуються на вхід
  pinMode(x1_Pin, INPUT);
  pinMode(x2_Pin, INPUT);
  pinMode(x3_Pin, INPUT);
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну x1 стану дискретного входу з номером 10,
  // що зберігається в змінній x1_Pin
  x1=digitalRead(x1_Pin);

  // зчитування в змінну x2 стану дискретного входу з номером 9,
  // що зберігається в змінній x2_Pin
  x2=digitalRead(x2_Pin);

  // зчитування в змінну x3 стану дискретного входу з номером 8,
  // що зберігається в змінній x3_Pin
  x3=digitalRead(x3_Pin);

  // перевірка значень, що зберігаються в змінних x1, x2 та x3
  // у відповідності до рівняння; якщо умова виконується
    if(((x1==HIGH) && (x2==LOW)) || (((x2==LOW) || (x3==LOW)) && (x1==LOW))) {
    // тоді на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється високий рівень сигналу HIGH,
    // тобто на цьому виводі з'явиться напруга +5В, і світлодіод загориться
    digitalWrite(led_Pin, HIGH);
    }
    // в протилежному випадку
    else {
    // на виході 0 (змінна led_Pin), до якого підключено світлодіод,
    // встановлюється низький рівень сигналу LOW,
    // тобто на цьому виводі з'явиться напруга 0В, і світлодіод згасне
        digitalWrite(led_Pin, LOW);
```

```
    }
  // далі відбувається циклічне виконання цієї функції з початку
}
```

Для перевірки роботи програми використовується модель пристрою з попереднього прикладу. Читачеві пропонується самостійно виконати моделювання схеми та перевірити алгоритм роботи програми.

3.4. Мультиплексори

При передачі цифрової інформації виникає велика кількість ситуацій, коли цифровий двійковий код необхідно перетворити з паралельного формату в послідовний з метою його запису або передачі по двопровідній лінії. Як приклад, пристрої запису інформації – накопичувачі на компакт-дисках, гнучкі та жорсткі диски – використовують послідовний формат. З послідовним форматом працюють COM-порти ПЕОМ, модеми, USB-шини.

Розглянемо умови і принцип перетворення двійкового коду з паралельного формату в послідовний.

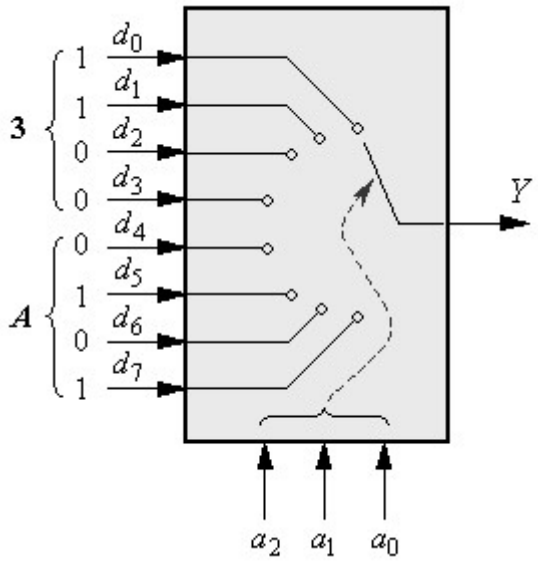


Рис. 3.20.

Таблиця 3.1.

N	a_2	a_1	a_0	Y	Код слова
0	0	0	0	d_0	1
1	0	0	1	d_1	1
2	0	1	0	d_2	0
3	0	1	1	d_3	0
4	1	0	0	d_4	0
5	1	0	1	d_5	1
6	1	1	0	d_6	0
7	1	1	1	d_7	1

Припустимо, що по однобайтовій шині даних до пристрою підводиться двійкове слово $d_7\,d_6\,d_5\,d_4\,d_3\,d_2\,d_1\,d_0 = \text{A3}_{16}$ (рис. 3.20), яке пристрій повинен зчитати і передати в послідовному форматі на вихід Y.

Щоб виконати таку операцію, необхідно послідовно звертатись (адресуватись) до кожного біта даних і по черзі приєднувати його до виходу Y. Зрозуміло, що для забезпечення такої адресації необхідно мати три розряди адресної шини. У механічному еквіваленті адресна шина визначає короткочасне положення бігунка перемикача, який по черзі з'єднує вхід шини даних з виходом Y пристрою.

Роботі пристрою, що зображений на рис. 3.20, відповідає таблиця станів (табл. 3.1).

Якщо вихідний код Y розглядати як функцію вхідних сигналів пристрою, то отримаємо наступну залежність:

$$Y = \overline{a_2}\ \overline{a_1}\ \overline{a_0}\ d_0 + \overline{a_2}\ \overline{a_1}\ a_0\ d_1 + \overline{a_2}\ a_1\ \overline{a_0}\ d_2 + \overline{a_2}\ a_1\ a_0\ d_3 +$$

$$+ a_2\ \overline{a_1}\ \overline{a_0}\ d_4 + a_2\ \overline{a_1}\ a_0\ d_5 + a_2\ a_1\ \overline{a_0}\ d_6 + a_2\ a_1\ a_0\ d_7\ .$$

Звідси витікає, що *мультиплексор* – це цифровий комбінаційний пристрій, який забезпечує перетворення інформації, що задана на входах $d_7 \ldots d_0$ у паралельному форматі, в послідовний формат на виході Y.

Умовне позначення мультиплексора в електронних схемах приводиться на рис. 3.21. Входи A називають адресними (адресна шина), а входи D – входами даних (шина даних). Допоміжний вхід V називають керуючим або *дозволяючим*. З його урахуванням робота мультиплексора, приведеного на рис. 3.21, описуватиметься наступною логічною функцією:

$$Y = \overline{v} \cdot (\overline{a_1}\ \overline{a_0}\ d_0 + \overline{a_1}\ a_0\ d_1 + a_1\ \overline{a_0}\ d_2 + a_1\ a_0\ d_3)\ .$$

Дозволяючий вхід V розширює можливості пристрою. Він дає можливість синхронізувати роботу мультиплексора з іншими схемами, а також використовується для нарощування розрядності адресних сигналів.

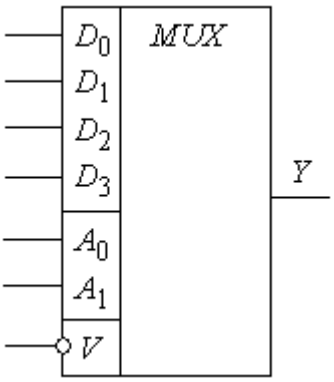


Рис. 3.21.

Наступний приклад демонструє реалізацію мультиплексора «з чотирьох в один» на базі контролера Arduino. Модель пристрою для дослідження роботи мультиплексора та перевірки програми представлено на рис. 3.22. Як і раніше, для зміни рівня сигналів на входах контролера використовуються елементи LOGICTOGGLE. Вхід, позначений як V, є дозволяючим входом. Входи A0 та A1 є адресними входами, а входи D0-D3 – входами даних. До виходу контролера з номером 0, який призначений як вихід мультиплексора, підключено світлодіод для того, щоб можна було спостерігати за зміною рівня сигналу при моделюванні роботи пристрою.

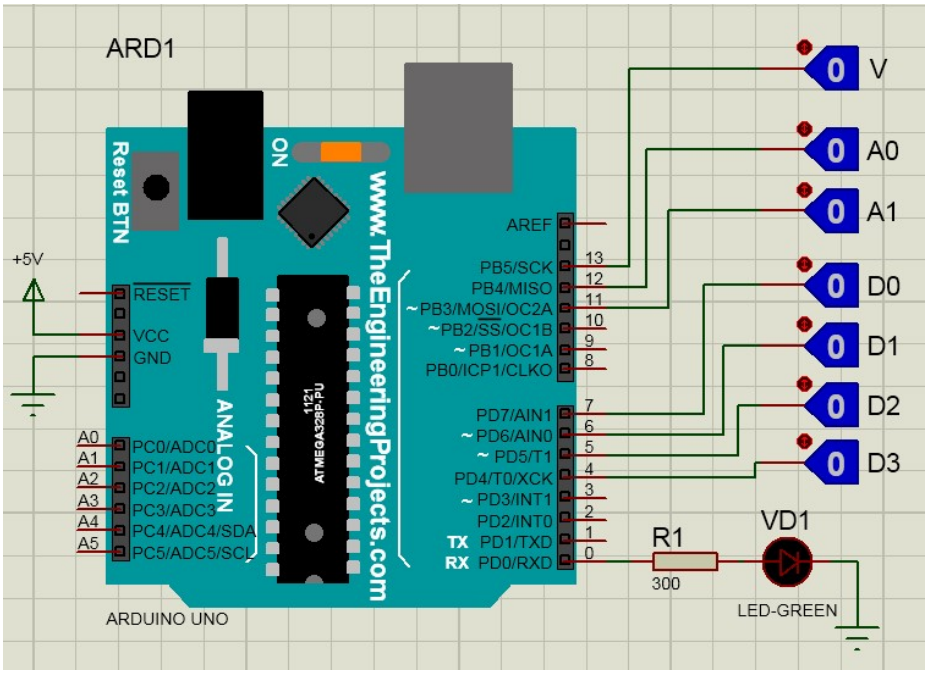


Рис. 3.22.

Текст програмної реалізації мультиплексора «з чотирьох в один» представлено нижче.

```
// оголошення змінної, в якій зберігається номер виводу, до якого підключено світлодіод
int led_Pin=0;
// оголошення змінної, в якій зберігається номер виводу D0
int D0_Pin=7;
// оголошення змінної, в якій зберігається номер виводу D1
int D1_Pin=6;
// оголошення змінної, в якій зберігається номер виводу D2
int D2_Pin=5;
// оголошення змінної, в якій зберігається номер виводу D3
int D3_Pin=4;
// оголошення змінної, в якій зберігається номер виводу A0
int A0_Pin=12;
// оголошення змінної, в якій зберігається номер виводу A1
int A1_Pin=11;
// оголошення змінної, в якій зберігається номер виводу V
int V_Pin=13;

// оголошення змінної, в якій буде зберігатися значення з входу D0
int D_0;
// оголошення змінної, в якій буде зберігатися значення з входу D1
int D_1;
// оголошення змінної, в якій буде зберігатися значення з входу D2
int D_2;
// оголошення змінної, в якій буде зберігатися значення з входу D3
int D_3;

// оголошення змінної, в якій буде зберігатися значення з адресного входу A0
```

```
int A_0;
// оголошення змінної, в якій буде зберігатися значення з адресного входу A1
int A_1;
// оголошення змінної, в якій буде зберігатися значення з дозволяючого входу V
int V;
// оголошення змінної, в якій буде зберігатися стан виходу мультиплексора
int Y;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // вивід, до якого підключено світлодіод (вихід мультиплексора), налаштовується на
вихід
  pinMode(led_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються
на вхід
  pinMode(D0_Pin, INPUT);
  pinMode(D1_Pin, INPUT);
  pinMode(D2_Pin, INPUT);
  pinMode(D3_Pin, INPUT);
  pinMode(A0_Pin, INPUT);
  pinMode(A1_Pin, INPUT);
  pinMode(V_Pin, INPUT);
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну D0 стану дискретного входу з номером 7,
  // що зберігається в змінній D0_Pin
  D_0=digitalRead(D0_Pin);

  // зчитування в змінну D1 стану дискретного входу з номером 6,
  // що зберігається в змінній D1_Pin
  D_1=digitalRead(D1_Pin);

  // зчитування в змінну D2 стану дискретного входу з номером 5,
  // що зберігається в змінній D2_Pin
  D_2=digitalRead(D2_Pin);

  // зчитування в змінну D3 стану дискретного входу з номером 4,
  // що зберігається в змінній D3_Pin
  D_3=digitalRead(D3_Pin);

  // зчитування в змінну A0 стану адресного входу з номером 12,
  // що зберігається в змінній A0_Pin
  A_0=digitalRead(A0_Pin);

  // зчитування в змінну A1 стану адресного входу з номером 11,
  // що зберігається в змінній A1_Pin
  A_1=digitalRead(A1_Pin);
```

```
  // зчитування в змінну V стану дозволяючого входу з номером 13,
  // що зберігається в змінній V_Pin
  V=digitalRead(V_Pin);

  // якщо на дозволяючому вході присутній рівень лог.1, то робота мультиплексора
дозволена
  if(V==LOW){
    // якщо на адресних входах A0 та A1 присутні значення лог.0, тобто адреса дорівнює
0
    if((A_0==LOW) && (A_1==LOW)){
      Y=D_0;  // в змінну Y записується значення з входу D0
      }
    // якщо на адресному вході A0 присутнє значення лог.1,
    // а на адресному вході A1 присутнє значення лог.0, тобто адреса дорівнює 1
    if((A_0==HIGH) && (A_1==LOW)){
      Y=D_1;  // в змінну Y записується значення з входу D1
      }
    // якщо на адресному вході A0 присутнє значення лог.0,
    // а на адресному вході A1 присутнє значення лог.1, тобто адреса дорівнює 2
    if((A_0==LOW) && (A_1==HIGH)){
      Y=D_2;  // в змінну Y записується значення з входу D2
      }
    // якщо на адресному вході A0 присутнє значення лог.1,
    // а на адресному вході A1 присутнє значення лог.1, тобто адреса дорівнює 3
    if((A_0==HIGH) && (A_1==HIGH)){
      Y=D_3;  // в змінну Y записується значення з входу D3
      }
    }
    // якщо на дозволяючому вході V присутній рівень лог.1, тоді
    // відбувається присвоювання змінній Y значення LOW,
    // яке відповідає низькому рівню сигналу, тобто лог.0
    else{
      Y=LOW;
      }
  digitalWrite(led_Pin, Y);
  // далі відбувається циклічне виконання цієї функції з початку
}
```

Результати моделювання роботи програмно реалізованого мультиплексора «з чотирьох в один» представлено на рис. 3.23.

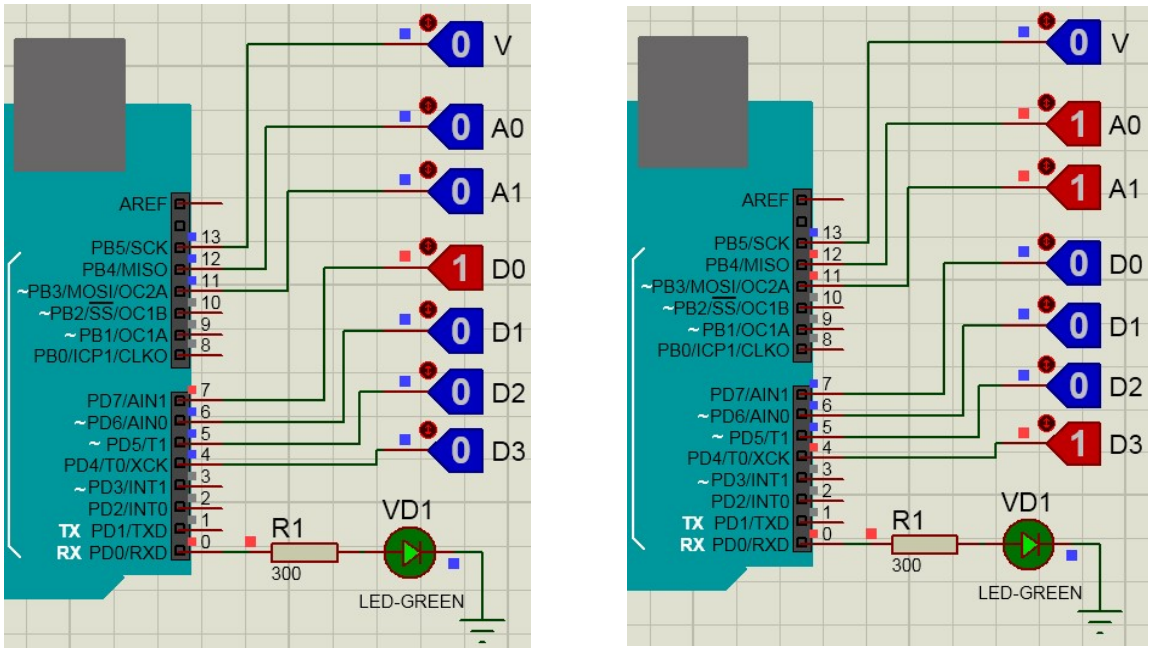


Рис. 3.23.

Робота мультиплексора дозволена, якщо на вході V присутній сигнал низького рівня. При формуванні на адресних входах A0 та A1 сигналів низького рівня, на виході мультиплексора сигнал приймає такий самий рівень, який присутній на вході D0, оскільки $00_2=0_{10}$. При формуванні на адресних A0 та A1 входах сигналів високого рівня, на виході мультиплексора сигнал приймає такий самий рівень, який присутній на вході D3, оскільки $11_2=3_{10}$.

3.5. Дешифратори-демультиплексори

Демультиплексори – це цифрові комбінаційні пристрої, функціональне призначення яких протилежне функціональному призначенню мультиплексорів. У них сигнали з одного інформаційного входу x розподіляються на 2^m виходів y_i, які комутуються m адресними входами.

У відповідності до схеми рис. 3.24, активними станами виходів $y_0 \div y_3$ є логічні “1”, активним станом дозволяючого входу V є логічний “0”.

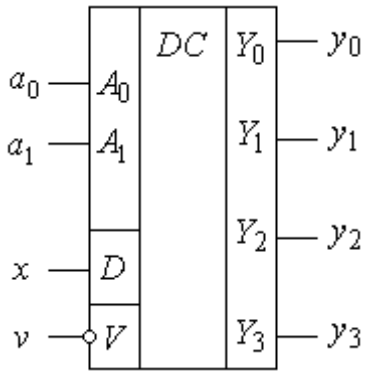


Рис. 3.24.

На рис. 3.25 представлено схему з контролером Arduino, на базі якої виконується програмна реалізація демультиплексора.

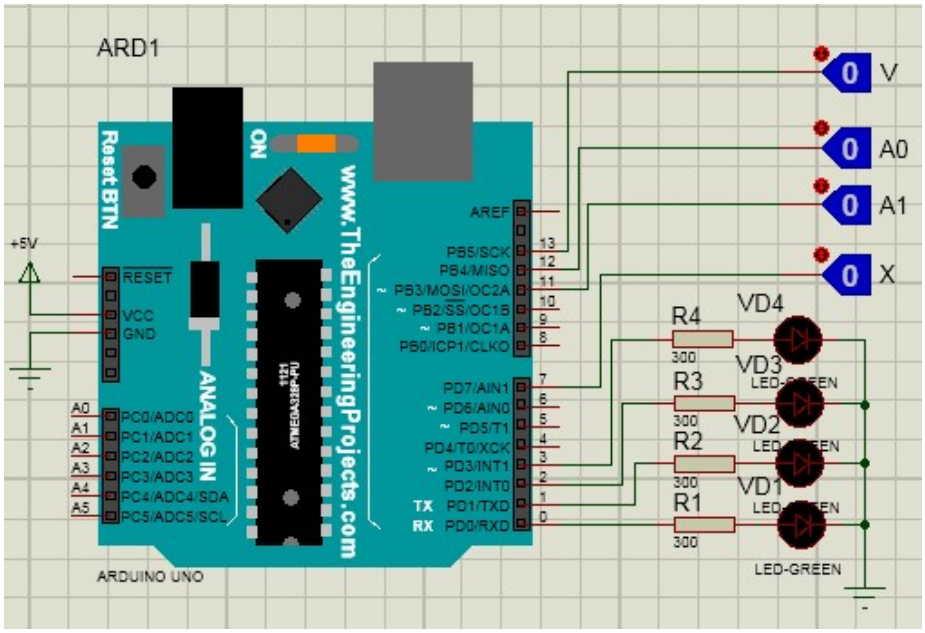


Рис. 3.25.

Текст програми, в якій реалізовано функціональність демультиплексора “з 1 на 4”, представлено нижче.

```
// оголошення змінної, в якій зберігається номер виводу Y0, до якого підключено світлодіод VD4
int Y0_Pin=3;
// оголошення змінної, в якій зберігається номер виводу Y1, до якого підключено світлодіод VD3
int Y1_Pin=2;
// оголошення змінної, в якій зберігається номер виводу Y2, до якого підключено світлодіод VD2
int Y2_Pin=1;
// оголошення змінної, в якій зберігається номер виводу Y3, до якого підключено світлодіод VD1
int Y3_Pin=0;

// оголошення змінної, в якій зберігається номер виводу X
int X_Pin=7;
// оголошення змінної, в якій зберігається номер виводу A0
int A0_Pin=12;
// оголошення змінної, в якій зберігається номер виводу A1
int A1_Pin=11;
// оголошення змінної, в якій зберігається номер виводу V
int V_Pin=13;

// оголошення змінної, в якій буде зберігатися значення з входу X
int X;
// оголошення змінної, в якій буде зберігатися значення з адресного входу A0
int A_0;
// оголошення змінної, в якій буде зберігатися значення з адресного входу A1
int A_1;
// оголошення змінної, в якій буде зберігатися значення з дозволяючого входу V
```

```
int V;
// оголошення змінної, в якій буде зберігатися стан виходу Y0 демультиплексора
int Y0;
// оголошення змінної, в якій буде зберігатися стан виходу Y1 демультиплексора
int Y1;
// оголошення змінної, в якій буде зберігатися стан виходу Y2 демультиплексора
int Y2;
// оголошення змінної, в якій буде зберігатися стан виходу Y3 демультиплексора
int Y3;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи демультиплексора), налаштовується на вихід
  pinMode(Y0_Pin, OUTPUT);
  pinMode(Y1_Pin, OUTPUT);
  pinMode(Y2_Pin, OUTPUT);
  pinMode(Y3_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються на вхід
  pinMode(X_Pin, INPUT);
  pinMode(A0_Pin, INPUT);
  pinMode(A1_Pin, INPUT);
  pinMode(V_Pin, INPUT);
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну X стану дискретного входу з номером 7,
  // що зберігається в змінній X_Pin
  X=digitalRead(X_Pin);

  // зчитування в змінну A0 стану адресного входу з номером 12,
  // що зберігається в змінній A0_Pin
  A_0=digitalRead(A0_Pin);

  // зчитування в змінну A1 стану адресного входу з номером 11,
  // що зберігається в змінній A1_Pin
  A_1=digitalRead(A1_Pin);

  // зчитування в змінну V стану дозволяючого входу з номером 13,
  // що зберігається в змінній V_Pin
  V=digitalRead(V_Pin);

    // якщо на дозволяючому вході присутній рівень лог.1, то робота мультиплексора дозволена
  if(V==LOW){
    // якщо на адресних входах A0 та A1 присутні значення лог.0, тобто адреса дорівнює 0
    if((A_0==LOW) && (A_1==LOW)){
```

```
    Y0=X;  // в змінну Y записується значення з входу X
    } else{
      Y0=LOW;
    }
  // якщо на адресному вході A0 присутнє значення лог.1,
  // а на адресному вході A1 присутнє значення лог.0, тобто адреса дорівнює 1
  if((A_0==HIGH) && (A_1==LOW)){
    Y1=X;  // в змінну Y записується значення з входу X
    } else{
      Y1=LOW;
    }
  // якщо на адресному вході A0 присутнє значення лог.0,
  // а на адресному вході A1 присутнє значення лог.1, тобто адреса дорівнює 2
  if((A_0==LOW) && (A_1==HIGH)){
    Y2=X;  // в змінну Y записується значення з входу X
    } else{
      Y2=LOW;
    }
  // якщо на адресному вході A0 присутнє значення лог.1,
  // а на адресному вході A1 присутнє значення лог.1, тобто адреса дорівнює 3
  if((A_0==HIGH) && (A_1==HIGH)){
    Y3=X;  // в змінну Y записується значення з входу X
    } else{
      Y3=LOW;
    }
  }
  // якщо на дозволяючому вході V присутній рівень лог.1, тоді
  // відбувається присвоювання змінним Y0, Y1, Y2 та Y3 значення LOW,
  // яке відповідає низькому рівню сигналу, тобто лог.0
  else{
    Y0=LOW;
    Y1=LOW;
    Y2=LOW;
    Y3=LOW;
    }
 // встановлення на виходах Y0_Pin, Y1_Pin, Y2_Pin та Y3_Pin демультиплексора
 // (номери виводів контролера 3, 2, 1 та 0)
 // рівнів сигналів, що відповідають вмісту змінних Y0, Y1, Y2 та Y3
 digitalWrite(Y0_Pin, Y0);
 digitalWrite(Y1_Pin, Y1);
 digitalWrite(Y2_Pin, Y2);
 digitalWrite(Y3_Pin, Y3);
 // далі відбувається циклічне виконання цієї функції з початку
}
```

Розглянемо приклад розробки цифрового комбінаційного автомату, що виконує перетворення чотирьохрозрядного двійкового коду в код “2 з 5”. Цей цифровий комбінаційний автомат представляє собою дешифратор.

Код “2 з 5” використовується для безпомилкової передачі цифрової інформації. В цьому п’ятирозрядному коді кожна допустима комбінація

містить дві одиниці та три нулі і, оскільки таких комбінацій десять, то кожна з них відповідає одній десятковій цифрі (табл. 3.2).

Таблиця 3.2.

Десят-кова цифра	4-розрядний двійковий код				Код “2 з 5”				
	x_3	x_2	x_1	x_0	a_4	a_3	a_2	a_1	a_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	1
2	0	0	1	0	0	0	1	0	1
3	0	0	1	1	0	0	1	1	0
4	0	1	0	0	0	1	0	0	1
5	0	1	0	1	0	1	0	1	0
6	0	1	1	0	0	1	1	0	0
7	0	1	1	1	1	0	0	0	1
8	1	0	0	0	1	0	0	1	0
9	1	0	0	1	1	0	1	0	0

Мінімізуючи функції $a_0 \div a_4$, отримаємо вирази для вихідних сигналів:

$$a_0 = (x_0 \oplus x_1)\bar{x}_2\bar{x}_3 + (x_0 \otimes x_1)x_2\bar{x}_3$$

$$a_1 = \bar{x}_1\bar{x}_2(x_0 \oplus x_3) + x_0\bar{x}_3(x_1 \oplus x_2)$$

$$a_2 = \bar{x}_0x_1\bar{x}_3 + x_0\bar{x}_2(x_1 \oplus x_3)$$

$$a_3 = x_2\bar{x}_3(\bar{x}_0\bar{x}_1 + x_0 \oplus x_1)$$

$$a_4 = x_0x_1x_2\bar{x}_3 + \bar{x}_1\bar{x}_2x_3$$

які легко реалізовуються програмно.

На рис. 3.26 представлено схему для перевірки роботи програмно реалізованого дешифратора, що виконує перетворення чотирьохрозрядного двійкового коду в код “2 з 5”.

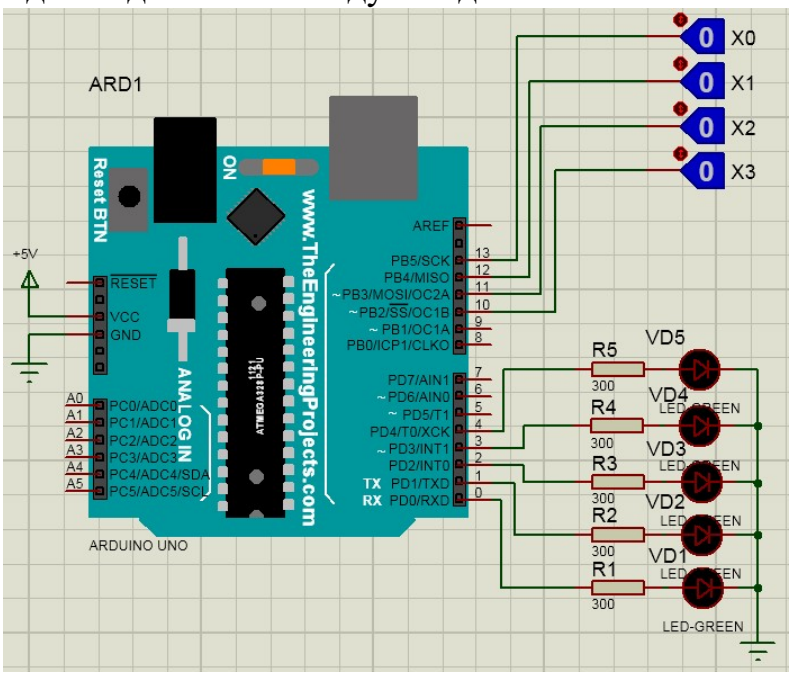


Рис. 3.26.

Програмна реалізація дешифратора, що виконує перетворення чотирьохрозрядного двійкового коду в код “2 з 5” має вигляд:

```
// оголошення змінної, в якій зберігається номер виводу A4, до якого підключено
світлодіод VD4
int A0_Pin=4;
// оголошення змінної, в якій зберігається номер виводу A3, до якого підключено
світлодіод VD4
int A1_Pin=3;
// оголошення змінної, в якій зберігається номер виводу A2, до якого підключено
світлодіод VD3
int A2_Pin=2;
// оголошення змінної, в якій зберігається номер виводу A1, до якого підключено
світлодіод VD2
int A3_Pin=1;
// оголошення змінної, в якій зберігається номер виводу A0, до якого підключено
світлодіод VD1
int A4_Pin=0;

// оголошення змінної, в якій зберігається номер виводу X0
int X0_Pin=13;
// оголошення змінної, в якій зберігається номер виводу X1
int X1_Pin=12;
// оголошення змінної, в якій зберігається номер виводу X2
int X2_Pin=11;
// оголошення змінної, в якій зберігається номер виводу X3
int X3_Pin=10;

// оголошення змінної, в якій буде зберігатися значення з входу X0
int X0;
// оголошення змінної, в якій буде зберігатися значення з входу X1
int X1;
// оголошення змінної, в якій буде зберігатися значення з входу X2
int X2;
// оголошення змінної, в якій буде зберігатися значення з входу X3
int X3;

// оголошення змінної, в якій буде зберігатися вхідний код дешифратора
int binary_code=0;
// оголошення масиву з 10 елементів, в якому кожен елемент представляє
// собою код "2 з 5", що відповідає порядковому номеру розташування елементу в
масиві
int result_code[10]={0b00000000, 0b00000011, 0b00000101, 0b00000110, 0b00001001,
0b00001010, 0b00001100, 0b00010001, 0b00010010, 0b00010100};
// оголошення змінної, в якій буде зберігатися результуючий код "2 з 5",
// який потім передається на вихід дешифратора
int output_code;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
```

```
  // виводи, до яких підключено світлодіоди (виходи дешифратора), налаштовується на
вихід
  pinMode(A0_Pin, OUTPUT);
  pinMode(A1_Pin, OUTPUT);
  pinMode(A2_Pin, OUTPUT);
  pinMode(A3_Pin, OUTPUT);
  pinMode(A4_Pin, OUTPUT);
  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються
на вхід
  pinMode(X0_Pin, INPUT);
  pinMode(X1_Pin, INPUT);
  pinMode(X2_Pin, INPUT);
  pinMode(X3_Pin, INPUT);
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінні X0-X3 станів дискретних входів з номерами 13, 12, 11, 10
  // що зберігаються в змінних X0_Pin, X1_Pin, X2_Pin, X3_Pin відповідно
  X0=digitalRead(X0_Pin);
  X1=digitalRead(X1_Pin);
  X2=digitalRead(X2_Pin);
  X3=digitalRead(X3_Pin);

 // якщо на вході X0_Pin низький рівень сигналу,
 // в змінній X0 буде міститися значення LOW, і умова виконується
  if(X0==LOW){
     // нульовий біт в змінній binary_code скидається (приймає значення 0)
     bitClear(binary_code, 0);
    }
    // інакше, якщо в змінній X0 буде міститися значення HIGH
    else{
      // нульовий біт в змінній binary_code встановлюється (приймає значення 1)
     bitSet(binary_code, 0);
    }

  // якщо на вході X1_Pin низький рівень сигналу,
  // в змінній X1 буде міститися значення LOW, і умова виконується
  if(X1==LOW){
     // перший біт в змінній binary_code скидається (приймає значення 0)
     bitClear(binary_code, 1);
    }
    // інакше, якщо в змінній X1 буде міститися значення HIGH
    else{
     // перший біт в змінній binary_code встановлюється (приймає значення 1)
     bitSet(binary_code, 1);
    }

  // якщо на вході X2_Pin низький рівень сигналу,
  // в змінній X2 буде міститися значення LOW, і умова виконується
  if(X2==LOW){
     // другий біт в змінній binary_code скидається (приймає значення 0)
```

```
    bitClear(binary_code, 2);
   }
   // інакше, якщо в змінній X2 буде міститися значення HIGH
   else{
    // другий біт в змінній binary_code встановлюється (приймає значення 1)
    bitSet(binary_code, 2);
   }

 // якщо на вході X3_Pin низький рівень сигналу,
 // в змінній X3 буде міститися значення LOW, і умова виконується
 if(X3==LOW){
    // третій біт в змінній binary_code скидається (приймає значення 0)
    bitClear(binary_code, 3);
   }
   // інакше, якщо в змінній X3 буде міститися значення HIGH
   else{
    // третій біт в змінній binary_code встановлюється (приймає значення 1)
    bitSet(binary_code, 3);
   }

   // запис в змінну output_code коду "2 з 5", що відповідає вхідному двійковому коду
   output_code=result_code[binary_code];

 // встановлення на виходах A0_Pin, A1_Pin, A2_Pin, A4_Pin та A5_Pin дешифратора
 // (номери виводів контролера 4, 3, 2, 1 та 0)
 // рівнів сигналів, що відповідають п'яти молодшим бітам змінної output_code
 digitalWrite(A0_Pin, bitRead(output_code, 0));
 digitalWrite(A1_Pin, bitRead(output_code, 1));
 digitalWrite(A2_Pin, bitRead(output_code, 2));
 digitalWrite(A3_Pin, bitRead(output_code, 3));
 digitalWrite(A4_Pin, bitRead(output_code, 4));
 // далі відбувається циклічне виконання функції loop() з початку
}
```

Результати моделювання роботи програмно реалізованого дешифратора чотирьохрозрядного двійкового коду в код “2 з 5” представлено на рис. 3.27. Якщо на входах X3-X0 формується двійкова комбінація 0001, на виходах контролера з’являється двійкова комбінація 00011, що відповідає таблиці 3.2. Нескладно перевірити й інші комбінації та підтвердити коректність функціонування програмно реалізованого дешифратора.

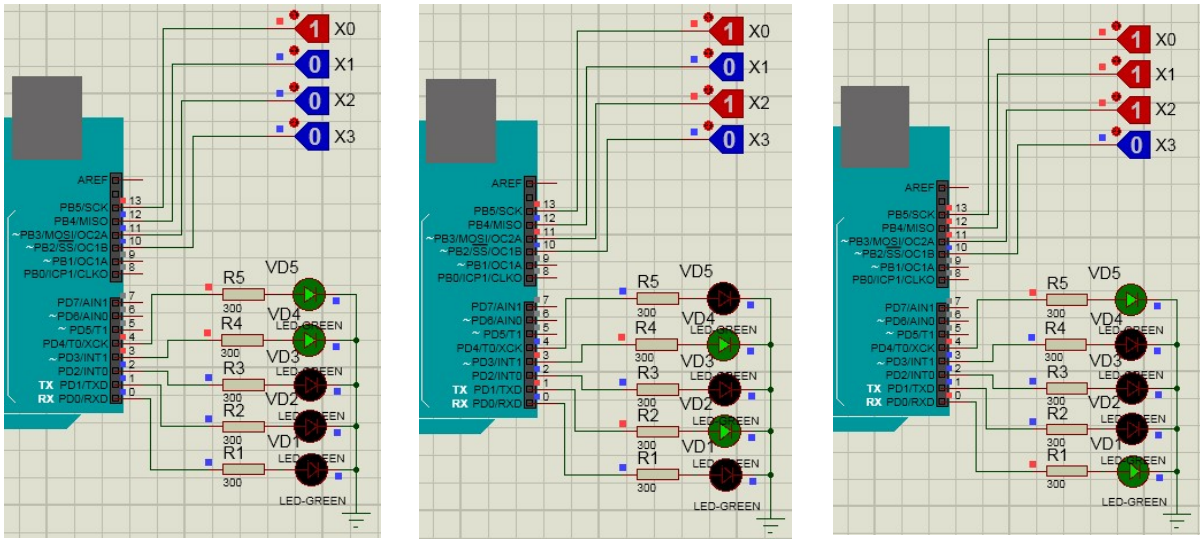


Рис. 3.27.

Розглянемо приклад розробки пристрою для перетворення двійкового однобайтного коду, заданого в послідовному форматі, в код, представлений у паралельному форматі, який фактично представляє собою за своєю функціональністю дешифратор К564ИД1.

Дешифратор К564ИД1 – це перетворювач чотирьохрозрядного двійкового коду в прямий десятковий. Він має чотири входи розрядів двійкового коду (1, 2, 4, 8) і 10 виходів розрядів десяткового коду (0 ÷ 9) (рис. 3.28). Табл. 3.3 є таблицею істинності мікросхеми і пояснює її роботу.

Таблиця 3.3.

Вхідний код Входи				Вихідний код Виходи									
8	**4**	**2**	**1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1
1	0	1	0	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0

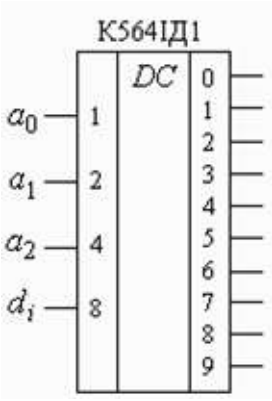


Рис. 3.28.

На рис. 3.29 представлено схему, яка використовується для перевірки роботи програми. Загалом, ця схема може бути використана для заміни мікросхеми-дешифратора К564ИД1. Нагадаємо, що в даному розділі для демонстрації роботи пристроїв та їх моделювання в середовищі Proteus використовуються в якості формувача сигналу елемент LOGICTOGGLE, а для відображення станів вихідних сигналів – світлодіоди. Це зроблено лише для наочності, адже при моделюванні системи в режимі реального часу можна змінювати стани цих елементів та спостерігати за допомогою світлодіодів зміну рівнів сигналів на виходах.

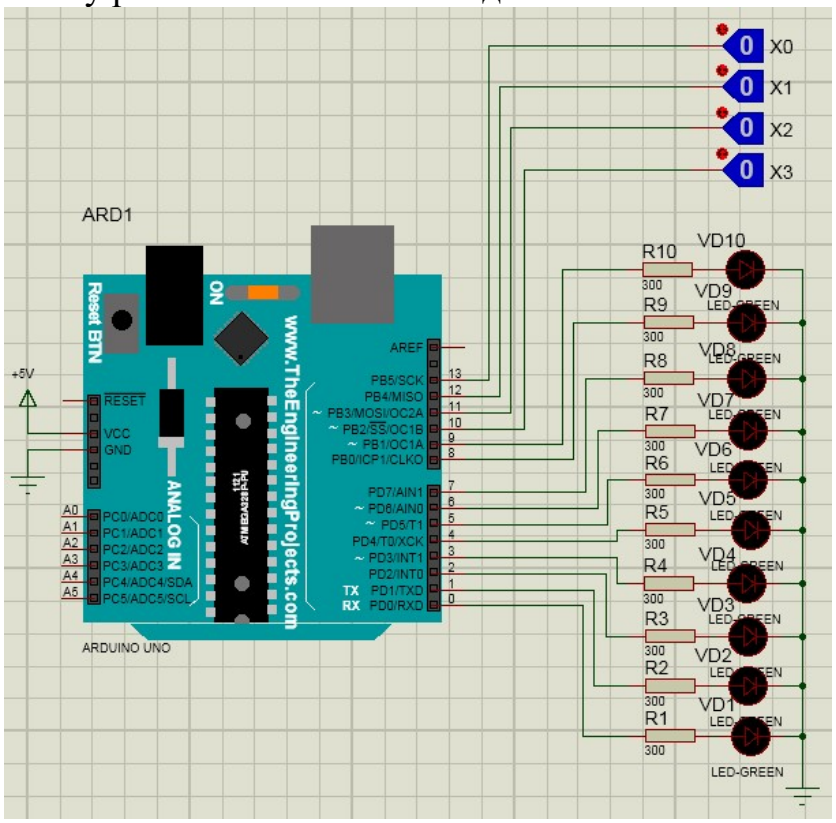


Рис. 3.29.

Текст програмної реалізації дешифратора К564ИД1 має вигляд:

```
// оголошення змінної, в якій зберігається номер виводу A9, до якого підключено світлодіод VD10
int A0_Pin=9;
// оголошення змінної, в якій зберігається номер виводу A8, до якого підключено світлодіод VD9
int A1_Pin=8;
// оголошення змінної, в якій зберігається номер виводу A7, до якого підключено світлодіод VD8
int A2_Pin=7;
// оголошення змінної, в якій зберігається номер виводу A6, до якого підключено світлодіод VD7
int A3_Pin=6;
// оголошення змінної, в якій зберігається номер виводу A5, до якого підключено світлодіод VD6
int A4_Pin=5;
```

```
// оголошення змінної, в якій зберігається номер виводу A4, до якого підключено
світлодіод VD5
int A5_Pin=4;
// оголошення змінної, в якій зберігається номер виводу A3, до якого підключено
світлодіод VD4
int A6_Pin=3;
// оголошення змінної, в якій зберігається номер виводу A2, до якого підключено
світлодіод VD3
int A7_Pin=2;
// оголошення змінної, в якій зберігається номер виводу A1, до якого підключено
світлодіод VD2
int A8_Pin=1;
// оголошення змінної, в якій зберігається номер виводу A0, до якого підключено
світлодіод VD1
int A9_Pin=0;

// оголошення змінної, в якій зберігається номер виводу X0
int X0_Pin=13;
// оголошення змінної, в якій зберігається номер виводу X1
int X1_Pin=12;
// оголошення змінної, в якій зберігається номер виводу X2
int X2_Pin=11;
// оголошення змінної, в якій зберігається номер виводу X3
int X3_Pin=10;

// оголошення змінної, в якій буде зберігатися значення з входу X0
int X0;
// оголошення змінної, в якій буде зберігатися значення з входу X1
int X1;
// оголошення змінної, в якій буде зберігатися значення з входу X2
int X2;
// оголошення змінної, в якій буде зберігатися значення з входу X3
int X3;

// оголошення змінної, в якій буде зберігатися вхідний код дешифратора
int binary_code=0;
// оголошення масиву з 10 елементів, в якому кожен елемент представляє
// собою код, що відповідає порядковому номеру розташування елементу в масиві
int result_code[10]={0b0000000001, 0b0000000010, 0b0000000100, 0b0000001000,
0b0000010000, 0b0000100000,
0b0001000000, 0b0010000000, 0b0100000000, 0b1000000000};
// оголошення змінної, в якій буде зберігатися результуючий код ,
// який потім передається на вихід дешифратора
int output_code;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи дешифратора), налаштовується на
вихід
  pinMode(A0_Pin, OUTPUT);
```

```
 pinMode(A1_Pin, OUTPUT);
 pinMode(A2_Pin, OUTPUT);
 pinMode(A3_Pin, OUTPUT);
 pinMode(A4_Pin, OUTPUT);
 pinMode(A5_Pin, OUTPUT);
 pinMode(A6_Pin, OUTPUT);
 pinMode(A7_Pin, OUTPUT);
 pinMode(A8_Pin, OUTPUT);
 pinMode(A9_Pin, OUTPUT);
 // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються
на вхід
 pinMode(X0_Pin, INPUT);
 pinMode(X1_Pin, INPUT);
 pinMode(X2_Pin, INPUT);
 pinMode(X3_Pin, INPUT);
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
 // зчитування в змінні X0-X3 станів дискретних входів з номерами 13, 12, 11, 10
 // що зберігаються в змінних X0_Pin, X1_Pin, X2_Pin, X3_Pin відповідно
 X0=digitalRead(X0_Pin);
 X1=digitalRead(X1_Pin);
 X2=digitalRead(X2_Pin);
 X3=digitalRead(X3_Pin);

// якщо на вході X0_Pin низький рівень сигналу,
// в змінній X0 буде міститися значення LOW, і умова виконується
 if(X0==LOW){
    // нульовий біт в змінній binary_code скидається (приймає значення 0)
    bitClear(binary_code, 0);
   }
   // інакше, якщо в змінній X0 буде міститися значення HIGH
   else{
     // нульовий біт в змінній binary_code встановлюється (приймає значення 1)
    bitSet(binary_code, 0);
   }

 // якщо на вході X1_Pin низький рівень сигналу,
 // в змінній X1 буде міститися значення LOW, і умова виконується
 if(X1==LOW){
    // перший біт в змінній binary_code скидається (приймає значення 0)
    bitClear(binary_code, 1);
   }
   // інакше, якщо в змінній X1 буде міститися значення HIGH
   else{
    // перший біт в змінній binary_code встановлюється (приймає значення 1)
    bitSet(binary_code, 1);
   }

 // якщо на вході X2_Pin низький рівень сигналу,
 // в змінній X2 буде міститися значення LOW, і умова виконується
```

```
 if(X2==LOW){
    // другий біт в змінній binary_code скидається (приймає значення 0)
    bitClear(binary_code, 2);
  }
  // інакше, якщо в змінній X2 буде міститися значення HIGH
  else{
    // другий біт в змінній binary_code встановлюється (приймає значення 1)
    bitSet(binary_code, 2);
  }

 // якщо на вході X3_Pin низький рівень сигналу,
 // в змінній X3 буде міститися значення LOW, і умова виконується
 if(X3==LOW){
    // третій біт в змінній binary_code скидається (приймає значення 0)
    bitClear(binary_code, 3);
  }
  // інакше, якщо в змінній X3 буде міститися значення HIGH
  else{
   // третій біт в змінній binary_code встановлюється (приймає значення 1)
    bitSet(binary_code, 3);
  }

  // запис в змінну output_code коду, що відповідає вхідному двійковому коду
  output_code=result_code[binary_code];

 // встановлення на виходах A0_Pin, A1_Pin, A2_Pin, A4_Pin та A5_Pin дешифратора
 // (номери виводів контролера 4, 3, 2, 1 та 0)
 // рівнів сигналів, що відповідають п'яти молодшим бітам змінної output_code
 digitalWrite(A0_Pin, bitRead(output_code, 0));
 digitalWrite(A1_Pin, bitRead(output_code, 1));
 digitalWrite(A2_Pin, bitRead(output_code, 2));
 digitalWrite(A3_Pin, bitRead(output_code, 3));
 digitalWrite(A4_Pin, bitRead(output_code, 4));
 digitalWrite(A5_Pin, bitRead(output_code, 5));
 digitalWrite(A6_Pin, bitRead(output_code, 6));
 digitalWrite(A7_Pin, bitRead(output_code, 7));
 digitalWrite(A8_Pin, bitRead(output_code, 8));
 digitalWrite(A9_Pin, bitRead(output_code, 9));
 // далі відбувається циклічне виконання функції loop() з початку
}
```

Результати моделювання роботи програмно реалізованого дешифратора представлено на рис. 3.30.

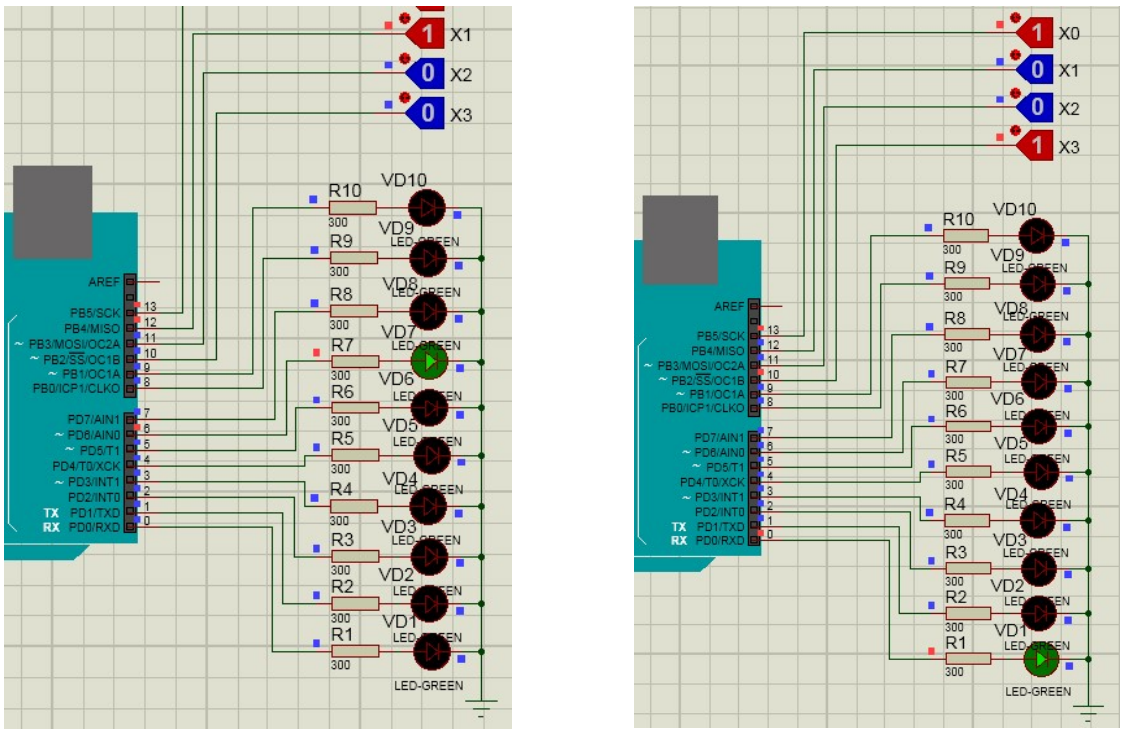


Рис. 3.30.

З рисунків видно, що вхідна двійкова комбінація задає номер виводу, на якому з'являється високий рівень сигналу. При формуванні двійкового коду 0011 на входах X3-X0, що відповідає значенню 3 в десятковому коді, та третьому виході дешифратора (вихід контролера з номером 6 до якого підключений світлодіод VD7) з'явиться високий рівень сигналу. Аналогічно, при формуванні двійкової комбінації 1001, що відповідає значенню 9, на дев'ятому виході контролера з'явиться сигнал високого рівня, про що свідчить зміна стану світлодіода VD1.

3.6. Програмна реалізація тригерів

Тригер – це запам'ятовуючий елемент з двома стійкими станами, зміна яких відбувається під дією вхідних сигналів. Як елемент комп'ютера, тригер призначений для зберігання одного біта інформації, тобто лог. 0 або лог. 1. Схема тригера забезпечує записування, зчитування, стирання та індикацію двійкової інформації, яка зберігається. На основі тригерів будують типові функціональні вузли комп'ютерів – регістри, лічильники, накопичувальні суматори, а також мікропрограмні автомати.

У цифрових пристроях використовується велика кількість різноманітних тригерів, але всі вони мають два виходи – прямий Q та інверсний $\overline{Q}$, котрі можуть приймати значення **1** або **0**. Логічні значення сигналів залежать від значень вхідних логічних змінних $x_1 \ldots x_n$, сигналів синхронізації $c_1 \ldots c_n$, станів виходів елемента пам'яті Q та $\overline{Q}$ і логіки роботи.

У зарубіжній інженерній практиці всі тригерні схеми розділяються на дві групи. Перша з них – *flip-flop* – характеризується тим, що вибірка

вхідних сигналів і відповідна зміна виходів визначається в моменти дії тактових часових сигналів (синхронні тригери). Особливість другої групи схем – *latch* – полягає в тому, що вони змінюють свій стан при зміні вхідних сигналів незалежно від наявності чи відсутності часових тактових сигналів.

Вхідні сигнали в залежності від призначення поділяють на: *інформаційні*, *дозволяючі*, *командні* або *синхронізуючі*. Відповідно до типів сигналів, поділяють і входи складних схем тригерів, які позначають літерами абетки згідно з табл. 3.4.

Таблиця 3.4.

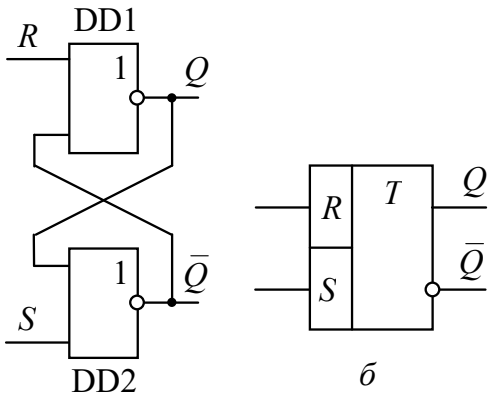
Позначення входу	Призначення
S	Вхід установки тригерів у стан $Q = 1$
R	Вхід установки тригерів у стан $Q = 0$
J	Вхід установки *JK*-тригерів у стан $Q = 1$
K	Вхід установки *JK*-тригерів у стан $Q = 0$
D	Вхід завантаження інформації у *D*-тригер
T	Тактовий вхід *T*-тригерів
V	Підготовчий вхід дозволу прийому інформації
C	Вхід синхронізації. Виконавчий вхід прийому інформації

Незалежно від функціональних властивостей, всі тригери за способом введення інформації поділяють на дві групи – *асинхронні* та *синхронні*.

Асинхронні тригери, подібно до розглянутого вище *RS*-тригера, змінюють свій стан безпосередньо після зміни інформаційних сигналів. Для спрацювання *синхронних тригерів* наявності інформаційних сигналів недостатньо. Допоміжно необхідна ще наявність синхроімпульсу (тактового імпульсу) на спеціальному тактовому *C*-вході. Тактові імпульси створюються за допомогою тактових генераторів і використовуються всім цифровим пристроєм.

3.6.1. RS-тригери

Схема *RS*-тригера, зібраного на логічних елементах **2АБО-НІ**, приведена на рис. 3.31, *а*. Завдяки симетричному вигляду, вона часто називається *симетричним RS-тригером* (але зовнішня симетрія не означає симетрію електричних режимів роботи логічних елементів DD1, DD2). На рис. 3.31, *б* приведено його умовне позначення.



а *б*

Рис. 3.31.

Стан тригера часто ототожнюється з сигналом на прямому виході Q.

Особливість тригерних схем, на відміну від комбінаційних, полягає в тому, що будь-який із станів є стійким при відсутності вхідних сигналів.

Таблиця станів *RS*-тригера (табл. 3.5) описує особливості його роботи. Звернемо увагу на те, що при $S_n = R_n = 1$, незалежно від Q_n, стан виходів тригера є невизначеним. Тому розглянута комбінація вхідних сигналів називається *невизначеною комбінацією*, а перехід від такої комбінації до нульової – *забороненим переходом*. Фактично це означає, що при проектуванні цифрових пристроїв необхідно приймати міри для виключення подібних ситуацій. При програмній реалізації *RS*-тригера це буде враховано.

Таблиця 3.5.

R_n	S_n	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	x
1	1	1	x

Зміна станів тригерних схем при подачі різних послідовностей вхідних сигналів зображується за допомогою графа переходів (рис. 3.32).

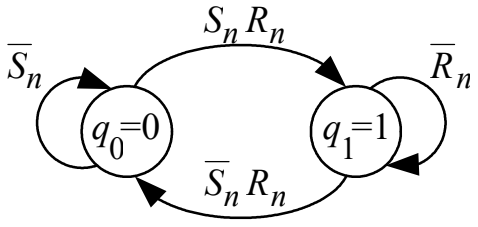


Рис. 3.32.

На рис. 3.33 представлено принципову схему, розроблену в середовищі Proteus, використання якої дозволяє дослідити роботу *RS*-тригера.

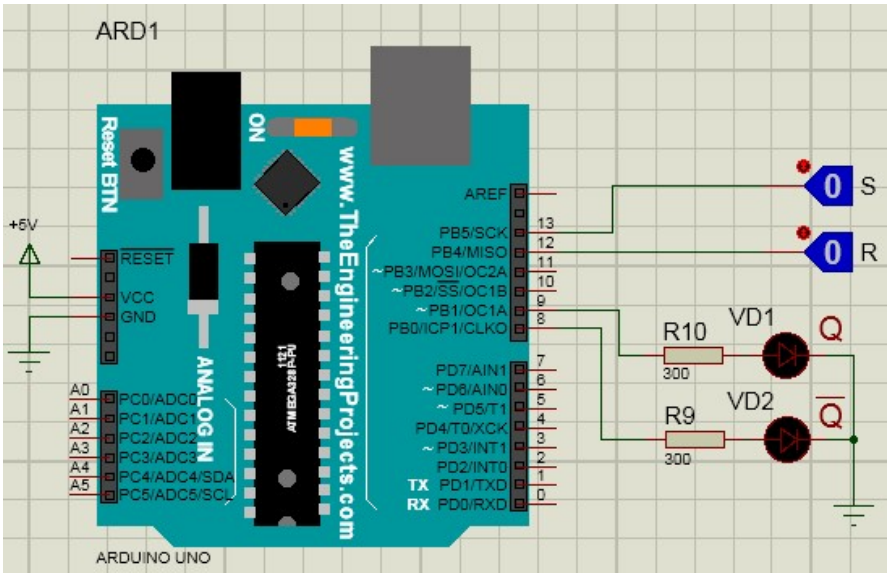


Рис. 3.33.

Текст програмної реалізації RS-тригера представлено нижче.

```
// оголошення змінної, в якій зберігається номер виводу Q,
// до якого підключено світлодіод VD1
int Q_Pin=9;
// оголошення змінної, в якій зберігається номер виводу
// NQ (інверсний вихід тригера), до якого підключено світлодіод VD2
int NQ_Pin=8;

// оголошення змінної, в якій зберігається номер виводу
// S - вхід установки тригера у стан Q=1
int S_Pin=13;
// оголошення змінної, в якій зберігається номер виводу
// R - вхід установки тригера у стан Q=0
int R_Pin=12;

// оголошення додаткових змінних, в яких зберігаються стани
// входів R та S контролера (виводи 12 та 13 відповідно)
int R, S;
// оголошення змінної, в якій зберігається поточний стан тригера
int Q;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи тригера), налаштовується на вихід
  pinMode(Q_Pin, OUTPUT);
  pinMode(NQ_Pin, OUTPUT);
```

```
  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються
на вхід
  pinMode(S_Pin, INPUT);
  pinMode(R_Pin, INPUT);

  // Початкове значення змінної Q дорівнює 0 - це початкове значення стану тригера
  Q=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінні S та R станів дискретних входів з номерами 13 та 12
  // що зберігаються в змінних S_Pin та R_Pin відповідно
  S=digitalRead(S_Pin);
  R=digitalRead(R_Pin);

  // Якщо тригер знаходиться в стані Q=0, виконується перевірка станів вхідних
сигналів
  if(Q==0)
  {
    // якщо на вході S високий рівень сигналу (лог. 1, HIGH), І (логічний оператор &&)
на вході R низький рівень (лог.0, LOW)
    if((S==HIGH) && (R==LOW))
    {
      Q=1; // тригер переходить в стан Q=1
    }
    // якщо на вході S низький рівень сигналу, І (логічний оператор &&) на вході R
    // високий АБО (логічний оператор ||) низький рівень сигналу
    else if((S==LOW) && ((R==LOW) || (R==HIGH)))
    {
      Q=0;  // тригер переходить в стан Q=0
    }
  }
  // Якщо тригер знаходиться в стані Q=1, також виконується перевірка станів вхідних
сигналів
  else if (Q==1)
  {
    // якщо на вході S низький рівень сигналу (лог. 0, LOW), І (логічний оператор &&) на
вході R високий рівень (лог.1, HIGH)
    if((S==LOW) && (R==HIGH))
    {
      Q=0;  // тригер переходить в стан Q=0
    }
    // якщо на вході R низький рівень сигналу, І (логічний оператор &&) на вході S
    // високий АБО (логічний оператор ||) низький рівень сигналу
    else if((R==LOW) && ((S==LOW) || (S==HIGH)))
    {
      Q=1;  // тригер переходить в стан Q=1
    }
  }

  // зміна станів виводів контролера у відповідності зі станом тригера
```

```
  digitalWrite(Q_Pin, Q);
  // значення на виході NQ (інверсний вихід тригера) є протилежним до Q
  digitalWrite(NQ_Pin, !Q);

  // далі відбувається циклічне виконання функції loop() з початку
}
```

Наведений вище код програми не є оптимальним. Аналіз програми та графу переходів RS-тригера показує, що деякі оператори умови (if) є надлишковими, і їх наявність не впливає на роботу програми. Наприклад, в стані Q=0 тригера відбувається перевірка станів входів, і за умови S=0 незалежно від значення на вході R, стан тригера не змінюється, тобто тригер залишається в стані Q=0. Це означає, що цей фрагмент коду можна взагалі видалити з програми, і це ніяким чином не вплине на логіку роботи програми. Після оптимізації програмна реалізація RS-тригера матиме вигляд, як показано далі.

```
// оголошення змінної, в якій зберігається номер виводу Q, до якого підключено
світлодіод VD1
int Q_Pin=9;
// оголошення змінної, в якій зберігається номер виводу NQ (інверсний вихід тригера),
до якого підключено світлодіод VD2
int NQ_Pin=8;

// оголошення змінної, в якій зберігається номер виводу S - вхід установки тригера у
стан Q=1
int S_Pin=13;
// оголошення змінної, в якій зберігається номер виводу R - вхід установки тригера у
стан Q=0
int R_Pin=12;

// Оголошення додаткових змінних, в яких зберігаються стани входів R та S контролера
(виводи 12 та 13 відповідно)
int R, S;
// Оголошення змінної, в якій зберігається поточний стан тригера
int Q;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи тригера), налаштовується на вихід
  pinMode(Q_Pin, OUTPUT);
  pinMode(NQ_Pin, OUTPUT);

  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються
на вхід
  pinMode(S_Pin, INPUT);
  pinMode(R_Pin, INPUT);
```

```
  // Початкове значення змінної Q дорівнює 0 - це початкове значення стану тригера
  Q=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінні S та R станів дискретних входів з номерами 13 та 12
  // що зберігаються в змінних S_Pin та R_Pin відповідно
  S=digitalRead(S_Pin);
  R=digitalRead(R_Pin);

  // Якщо тригер знаходиться в стані Q=0, виконується перевірка станів вхідних
сигналів
  if(Q==0)
  {
    // якщо на вході S високий рівень сигналу (лог. 1, HIGH), I (логічний оператор &&)
на вході R низький рівень (лог.0, LOW)
    if((S==HIGH) && (R==LOW))
    {
      Q=1; // тригер переходить в стан Q=1
    }
  }
  // Якщо тригер знаходиться в стані Q=1, також виконується перевірка станів вхідних
сигналів
  else if (Q==1)
  {
    // якщо на вході S низький рівень сигналу (лог. 0, LOW), I (логічний оператор &&) на
вході R високий рівень (лог.1, HIGH)
    if((S==LOW) && (R==HIGH))
    {
      Q=0;  // тригер переходить в стан Q=0
    }
  }

  // зміна станів виводів контролера у відповідності зі станом тригера
  digitalWrite(Q_Pin, Q);
  // значення на виході NQ (інверсний вихід тригера) є протилежним до Q
  digitalWrite(NQ_Pin, !Q);

  // далі відбувається циклічне виконання функції loop() з початку
}
```

На рис. 3.34 показано фрагмент схеми в режимі моделювання роботи RS-тригера.

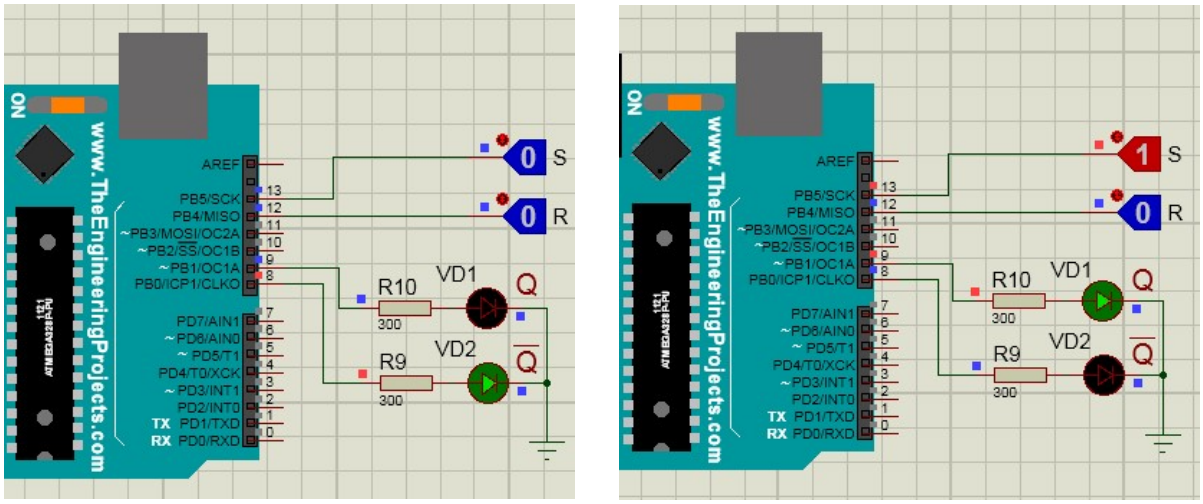


Рис. 3.34.

В початковому стані на прямому виході Q тригера присутній рівень лог.0, тоді як на інверсному виході присутній сигнал лог.1. При формуванні лог.1 на вході S, виходи тригера змінюють свій стан на протилежний, та потім залишаються незмінними до тих пір, поки на вхід R не буде подано сигнал високого рівня (за тієї умови, що на вході S буде низький рівень сигналу). З аналізу програми видно, що при подачі на обидва входи сигналів високого рівня, що є забороненою комбінацією, невизначеності не виникає, і тригер залишає свій стан незмінним.

3.6.2. D-тригер

Функціональна особливість тригерів цього типу полягає в тому, що сигнал на виході Q в $(n+1)$-ому такті повторює значення сигналу на вході D в n-му такті. Роботу D-тригера пояснює таблиця його станів (табл. 3.6), з якої витікає, що при $C_n = 0$ значення $Q_{n+1} = Q_n$, а при $C_n = 1$ $Q_{n+1} = D_n$.

Таблиця 3.6.

C_n	D_n	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Закон функціонування D-тригера виражається формулою:

$$Q_{n+1} = C_n\, D_n .$$

Граф переходів у відповідності до табл. 3.6 зображений на рис. 3.35.

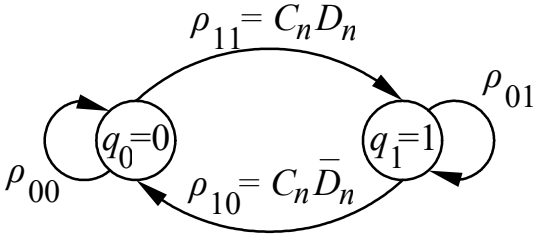


Рис. 3.35.

Перехід зі стану $q_0 = 0$ в стан $q_1 = 1$ відбувається лише при $\rho_{11} = C_n\, D_n$, а зворотній – тільки при $\rho_{10} = C_n\, \overline{D_n}$. При $q_0 = 0$ стан тригера не змінюється при:

$$\rho_{00} = \overline{C_n}\ \overline{D_n} + C_n\ \overline{D_n} + \overline{C_n}\ D_n = \overline{D_n} + \overline{C_n}.$$

При q_1=1 стан тригера залишається незмінним за умови:

$$\rho_{01} = \overline{C_n}\ \overline{D_n} + C_n\ D_n + \overline{C_n}\ D_n = \overline{C_n} + D_n.$$

При практичному використанні *D*-тригерів у них фіксується значення сигналу *D* у момент відповідної зміни сигналу *C*. Тому в літературі такі тригери часто називають *фіксаторами* (*Latch*). З точки зору логіки роботи *D*-тригера, він затримує проходження сигналу, що поступає на *D*-вхід, на один такт періоду синхросигналу (*Delay* – затримка).

Друге призначення *D*-тригера – зберігати дані (*Datas*), що надходять по *D*-входу. Синхросигнали в цьому випадку відіграють роль команди "Запис у тригер". Особливість схеми проявляється в режимі $C = 1$. У цьому випадку будь-які зміни (t_5, t_6, t_7, t_8) на вході *D* приведуть до відповідних змін на виході *Q*. Тобто схема виконуватиме просто функцію повторювача сигналів, що присутні на вході *D*. На рис. 3.36 представлено схему для дослідження *D*-тригера.

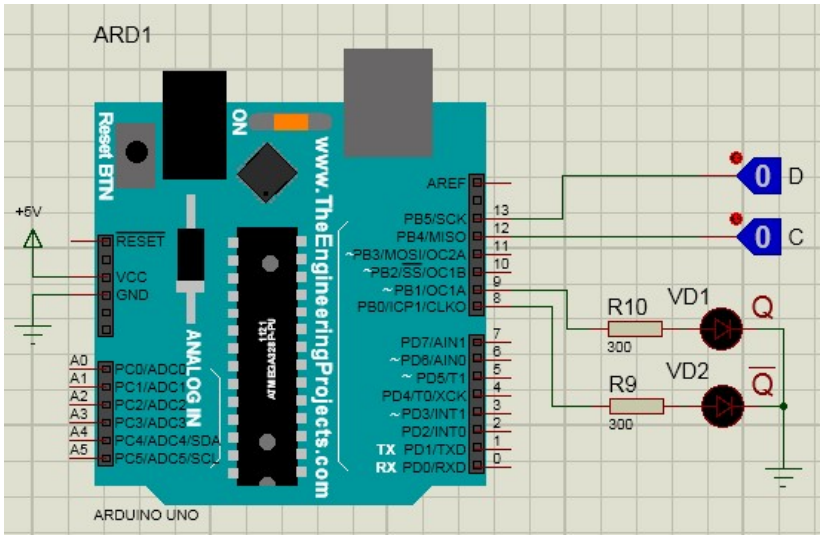


Рис. 3.36.

Програмна реалізація статичного *D*-тригера на базі наведеної схеми має вигляд:

```
// оголошення змінної, в якій зберігається номер виводу Q, до якого підключено світлодіод VD1
int Q_Pin=9;
// оголошення змінної, в якій зберігається номер виводу NQ (інверсний вихід тригера), до якого підключено світлодіод VD2
int NQ_Pin=8;

// оголошення змінної, в якій зберігається номер виводу D - вхід завантаження інформації у D-тригер
int D_Pin=13;
// оголошення змінної, в якій зберігається номер виводу C - вхід синхронізації
int C_Pin=12;

// Оголошення додаткових змінних, в яких зберігаються стани входів C та D контролера (виводи 12 та 13 відповідно)
int C, D;
// Оголошення змінної, в якій зберігається поточний стан тригера
int Q;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи тригера), налаштовується на вихід
  pinMode(Q_Pin, OUTPUT);
  pinMode(NQ_Pin, OUTPUT);

  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються на вхід
  pinMode(D_Pin, INPUT);
  pinMode(C_Pin, INPUT);

  // Початкове значення змінної Q дорівнює 0 - це початкове значення стану тригера
  Q=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінні D та C станів дискретних входів з номерами 13 та 12
  // що зберігаються в змінних D_Pin та C_Pin відповідно
  D=digitalRead(D_Pin);
  C=digitalRead(C_Pin);

  // тільки якщо на вході C присутній високий рівень, відбувається зміна стану тригера
  if(C==1)
  {
  // Якщо на вході низький рівень сигналу
  if(D==0)
  {
    Q=0; // тригер переходить в стан Q=0
  }
```

```
 // Якщо на вході D високий рівень сигналу
 else if (D==1)
 {
   Q=1;  // тригер переходить в стан Q=1
 }
 }

 // зміна станів виводів контролера у відповідності зі станом тригера
 digitalWrite(Q_Pin, Q);
 // значення на виході NQ (інверсний вихід тригера) є протилежним до Q
 digitalWrite(NQ_Pin, !Q);

 // далі відбувається циклічне виконання функції loop() з початку
}
```

Наведену програму також можна оптимізувати і не використовувати зайві оператори перевірки умови. Враховуючи те, що сигнал на виході Q в (n +1)-ому такті повторює значення сигналу на вході D в n-му такті, фрагмент програми:

```
 // тільки якщо на вході C присутній високий рівень, відбувається зміна стану тригера
 if(C==1)
 {
 // Якщо на вході D високий рівень сигналу
 if(D==0)
 {
  Q=0; // тригер переходить в стан Q=0
 }
 // Якщо на вході D високий рівень сигналу
 else if (D==1)
 {
   Q=1;  // тригер переходить в стан Q=1
 }
 }
```

можна замінити на:

```
 // тільки якщо на вході C присутній високий рівень, відбувається зміна стану тригера
 if(C==1)
 {
  // стан тригера визначається рівнем сигналу на вході D
  Q=D;
 }
```

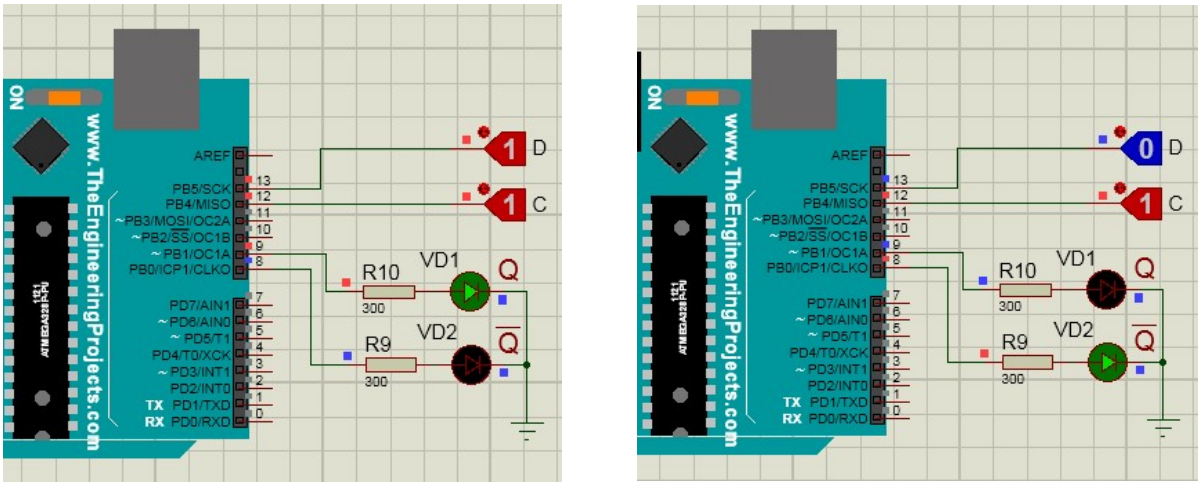


Рис. 3.37.

Для того, щоб зробити динамічний *D*-тригер, тобто стан виходу тригера змінювався лише при зміні сигналу синхронізації *C*, в програму потрібно внести невеликі зміни, як показано нижче.

```
// тільки якщо на вході C присутній високий рівень, відбувається зміна стану тригера
  if(C==1)
  {
   Q=D;
   while(C==1)
   {
    C=digitalRead(C_Pin);
   }
  }
```

Якщо на вході *C* був високий рівень сигналу, умова виконується, і на виході тригера встановлюється значення, що присутнє на вході *D*. Після цього виконання програми призупиняється в циклі while доти, поки на вході *C* залишається високий рівень. Тобто, для того, щоб стан тригера змінився, обов'язковою умовою є повернення сигналу на вході *C* в низький рівень.

3.6.3. JK-тригер

Цей тип тригерів за логікою роботи подібний до *RS*-тригерів, але, на відміну від них, не має невизначених переходів. Скорочена таблиця станів (табл. 3.7) пояснює наведену різницю, якщо прийняти, що *S*-вхід відповідає *J*-входу, а *R*-вхід – відповідно, *K*-.

Таблиця 3.7.

J_n	K_n	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	$\overline{Q}_n$

З таблиці витікає, що поява комбінації $J = K = 1$ у кожному такті призводить до зміни стану тригера на протилежний.

На відміну від *RS*-тригерів, схеми *JK*-тригерів виготовляються тільки синхронними. Для синхронного *JK*-тригера є справедливим рівняння:

$$Q_{n+1} = C(J_n \overline{Q}_n + \overline{K}_n Q_n) + \overline{C} Q_n .$$

На рис. 3.38 представлено схему для дослідження роботи JK-тригера.

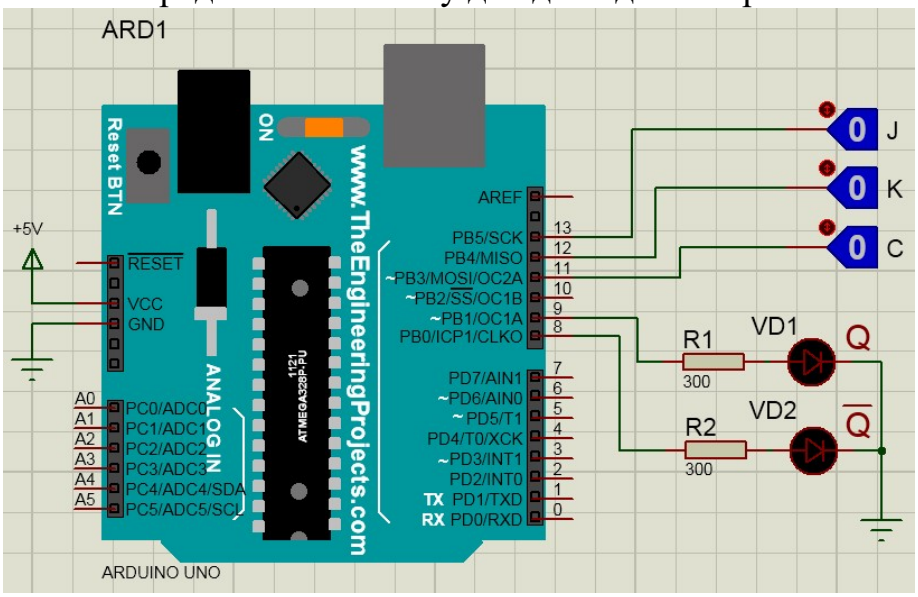


Рис. 3.38.

Текст програмної реалізації *JK*-тригера наведено нижче.

```
// оголошення змінної, в якій зберігається номер виводу Q, до якого підключено світлодіод VD1
int Q_Pin=9;
// оголошення змінної, в якій зберігається номер виводу NQ (інверсний вихід тригера), до якого підключено світлодіод VD2
int NQ_Pin=8;

// оголошення змінної, в якій зберігається номер виводу J - вхід установки JK-тригерів у стан Q = 1
int J_Pin=13;
// оголошення змінної, в якій зберігається номер виводу K - вхід установки JK-тригерів у стан Q = 0
int K_Pin=12;
// оголошення змінної, в якій зберігається номер виводу C - вхід синхронізації
int C_Pin=11;
```

```
// оголошення додаткових змінних, в яких зберігаються стани входів J, K та C
контролера (виводи 13, 12 та 11 відповідно)
int J, K, C;
// оголошення змінної, в якій зберігається поточний стан тригера
int Q;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи тригера), налаштовується на вихід
  pinMode(Q_Pin, OUTPUT);
  pinMode(NQ_Pin, OUTPUT);

  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються
на вхід
  pinMode(J_Pin, INPUT);
  pinMode(K_Pin, INPUT);
  pinMode(C_Pin, INPUT);

  // Початкове значення змінної Q дорівнює 0 - це початкове значення стану тригера
  Q=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінні J, K та C станів дискретних входів з номерами 13, 12 та 11
  // що зберігаються в змінних J_Pin, K_Pin та C_Pin відповідно
  J=digitalRead(J_Pin);
  K=digitalRead(K_Pin);
  C=digitalRead(C_Pin);

  // тільки якщо на вході C присутній високий рівень, відбувається зміна стану тригера
  if(C==1)
  {
     // умова виконується, якщо на вході J високий рівень сигналу (лог.1),
     // і на вході K низький рівень сигналу (лог.0)
     if((J==1) && (K==0))
     {
       Q=1;  // тригер переходить в стан Q=1
     }
     // умова виконується, якщо на вході J низький рівень сигналу (лог.0),
     // і на вході K високий рівень сигналу (лог.1)
     else if((J==0) && (K==1))
     {
       Q=0;  // тригер переходить в стан Q=0
     }
     // умова виконується, якщо на вході J високий рівень сигналу (лог.1),
     // і на вході K високий рівень сигналу (лог.1)
     else if((J==1) && (K==1))
```

```
    {
      Q=!Q;  // стан тригера змінюється на протилежний
    }

   // в циклі відбувається опитування стану входу C до тих пір,
   // поки на ньому не з'явиться низький рівень (лог.0)
   while(C==1)
   {
     C=digitalRead(C_Pin);
   }
 }

 // зміна станів виводів контролера у відповідності зі станом тригера
 digitalWrite(Q_Pin, Q);
 // значення на виході NQ (інверсний вихід тригера) є протилежним до Q
 digitalWrite(NQ_Pin, !Q);

 // далі відбувається циклічне виконання функції loop() з початку
}
```

Результати моделювання роботи *JK*-тригера представлено на рис. 3.39.

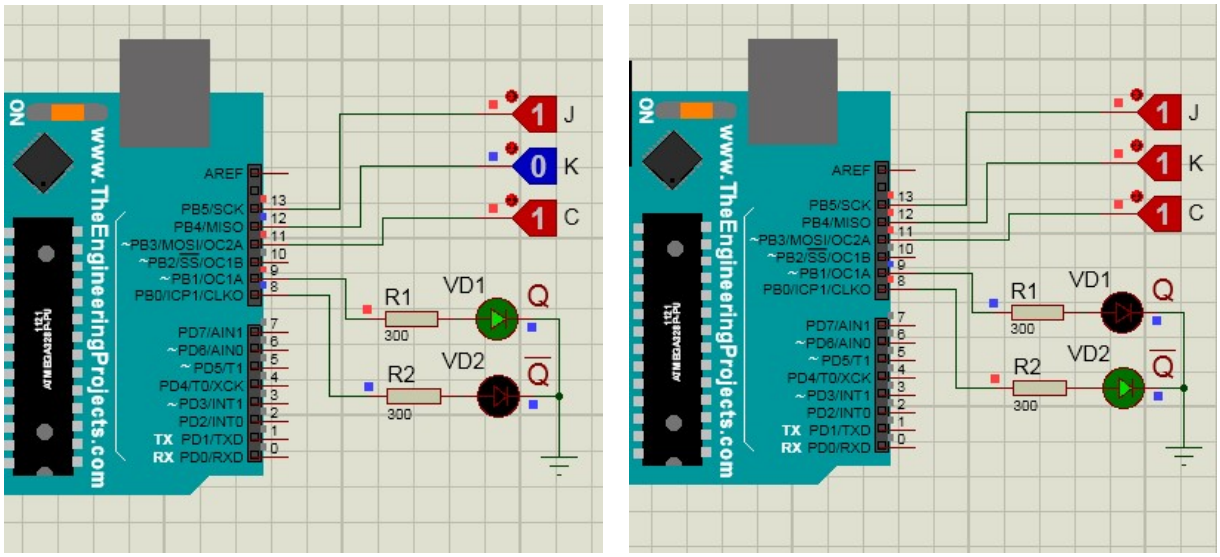


Рис. 3.39.

Якщо на вході J встановити сигнал високого рівня, а потім сформувати тактовий сигнал C, то на прямому виході Q тригера з’явиться високий рівень сигналу. Зрозуміло, на інверсному виході тригера в цей час буде протилежне значення – сигнал низького рівня. Якщо на обидві входи тригера, J та K, подавати лог.1, та формувати тактовий сигнал C, то кожного разу при зміні тактового сигналу (в даному випадку по зрізу сигналу C), вихід тригера буде змінювати свій стан на протилежний, що повністю відповідає логіці роботи *JK*-тригера.

3.6.4. Т-тригер

До тригерів *T*-типу відносяться такі схеми, які за сигналом на *T*-вхід переключаються у протилежний стан. Це зазвичай тригери з динамічним *T*-входом або з динамічним *C*-входом і статичним *T*-входом.

У табл. 3.8 приведений перелік можливих станів *T*-тригера. З таблиці витікає, що у *T*-тригерах, на відміну від раніше розглянутих тригерів, стан виходу Q_{n+1} залежить не від значень інформаційних сигналів, а визначається тільки їх станом у попередньому такті.

Таблиця 3.8.

T-тригер	
T_n	Q_n
0	Q_n
⌠	$\overline{Q}_n$
⌡	Q_n
1	Q_n

Рівняння роботи асинхронного тактованого *T*-тригера має вигляд:

$$Q_{n+1} = Q_n \overline{T_n} + \overline{Q_n}\, T_n .$$

Для синхронного *T*-тригера маємо:

$$Q_{n+1} = (Q_n \overline{T} + \overline{Q_n}\, T)\, C + \overline{C}\, Q_n .$$

На рис. 3.40 представлено схему для дослідження роботи *T*-тригера з динамічним *C*-входом і статичним *T*-входом, на базі якої виконується програмна реалізація *T*-тригера.

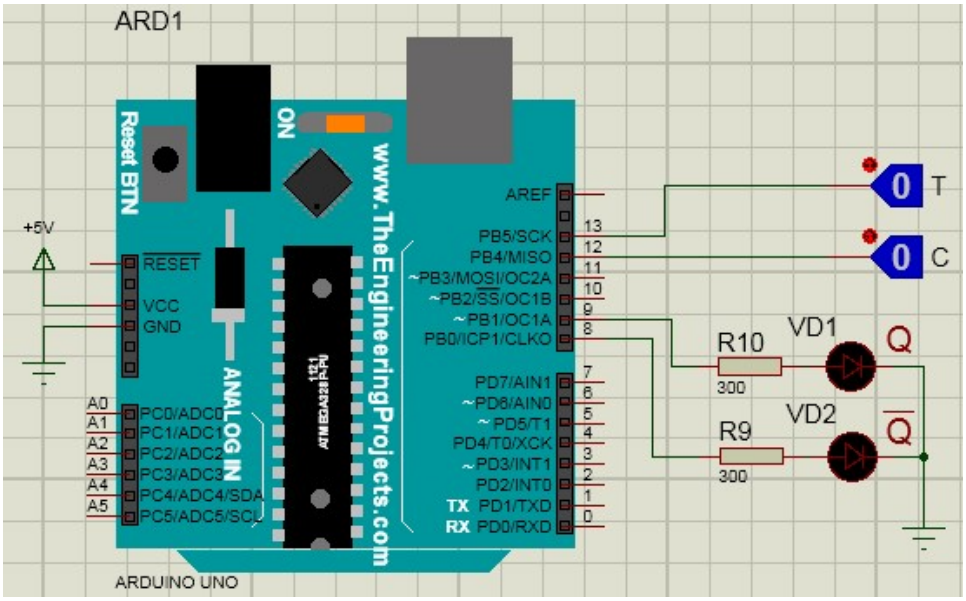


Рис. 3.40.

Текст програми, що реалізує на базі наведеної схеми роботу *T*-тригера з динамічним *C*-входом та статичним *T*-входом, наведено нижче.

```
// оголошення змінної, в якій зберігається номер виводу Q, до якого підключено світлодіод VD1
int Q_Pin=9;
// оголошення змінної, в якій зберігається номер виводу NQ (інверсний вихід тригера), до якого підключено світлодіод VD2
int NQ_Pin=8;

// оголошення змінної, в якій зберігається номер виводу T - тактовий вхід T-тригерів
int T_Pin=13;
// оголошення змінної, в якій зберігається номер виводу C - вхід синхронізації
int C_Pin=12;

// оголошення додаткових змінних, в яких зберігаються стани входів C та T контролера (виводи 12 та 13 відповідно)
int C, T;
// оголошення змінної, в якій зберігається поточний стан тригера
int Q;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи тригера), налаштовується на вихід
  pinMode(Q_Pin, OUTPUT);
  pinMode(NQ_Pin, OUTPUT);

  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються на вхід
  pinMode(T_Pin, INPUT);
  pinMode(C_Pin, INPUT);

  // Початкове значення змінної Q дорівнює 0 - це початкове значення стану тригера
  Q=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінні T та C станів дискретних входів з номерами 13 та 12
  // що зберігаються в змінних T_Pin та C_Pin відповідно
  T=digitalRead(T_Pin);
  C=digitalRead(C_Pin);

  // тільки якщо на вході C присутній високий рівень, відбувається зміна стану тригера
  if(C==1)
  {
    if(T==1) // якщо на вході T присутній високий рівень сигналу (лог.1)
    {
      Q=!Q;  // стан тригера змінюється на протилежний
    }
```

```
  // в циклі відбувається опитування стану входу C до тих пір,
  // поки на ньому не з'явиться низький рівень (лог.0)
  while(C==1)
  {
    C=digitalRead(C_Pin);
  }
 }

 // зміна станів виводів контролера у відповідності зі Q станом тригера
 digitalWrite(Q_Pin, Q);
 // значення на виході NQ (інверсний вихід тригера) є протилежним до Q
 digitalWrite(NQ_Pin, !Q);

 // далі відбувається циклічне виконання функції loop() з початку
}
```

Схема для дослідження роботи динамічного *T*-тригера представлена на рис. 3.41.

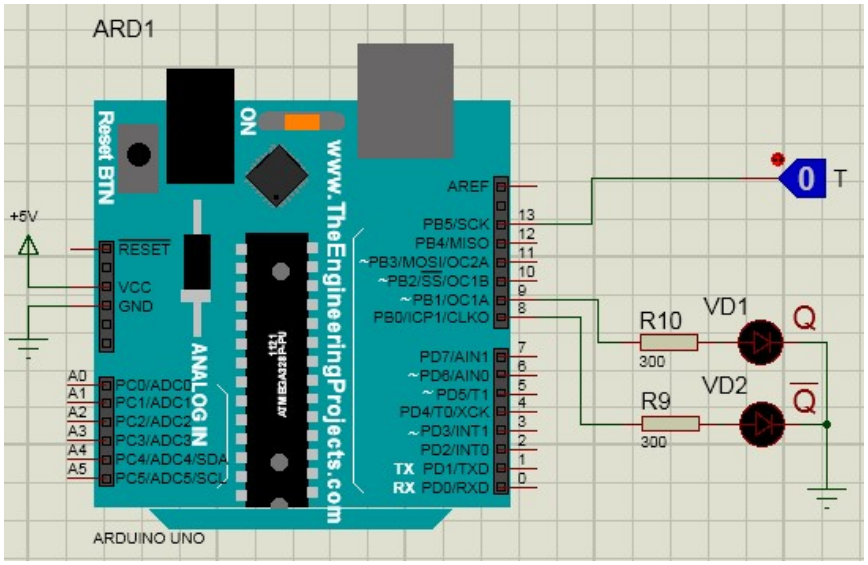


Рис. 3.41.

Текст програми, в якій реалізовано логіку роботи *T*-тригера з динамічним *T*-входом наведено далі.

```
// оголошення змінної, в якій зберігається номер виводу Q, до якого підключено світлодіод VD1
int Q_Pin=9;
// оголошення змінної, в якій зберігається номер виводу NQ (інверсний вихід тригера), до якого підключено світлодіод VD2
int NQ_Pin=8;

// оголошення змінної, в якій зберігається номер виводу T - тактовий вхід T-тригерів

int T_Pin=13;

// оголошення додаткових змінних, в яких зберігаються стани входів C та T контролера (виводи 12 та 13 відповідно)
```

```
int T;
// оголошення змінної, в якій зберігається поточний стан тригера
int Q;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при
перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи тригера), налаштовується на вихід
  pinMode(Q_Pin, OUTPUT);
  pinMode(NQ_Pin, OUTPUT);

  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються
на вхід
  pinMode(T_Pin, INPUT);

  // Початкове значення змінної Q дорівнює 0 - це початкове значення стану тригера
  Q=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну T стану дискретного входу з номером 13
  // що зберігається в змінній T_Pin
  T=digitalRead(T_Pin);

  // тільки якщо на вході T присутній високий рівень, відбувається зміна стану тригера
  if(T==1)
  {
     Q=!Q;  // стан тригера змінюється на протилежний
    // в циклі відбувається опитування стану входу T до тих пір,
    // поки на ньому не з'явиться низький рівень (лог.0)
    while(T==1)
    {
      T=digitalRead(T_Pin);
    }
  }

  // зміна станів виводів контролера у відповідності зі станом тригера
  digitalWrite(Q_Pin, Q);
  // значення на виході NQ (інверсний вихід тригера) є протилежним до Q
  digitalWrite(NQ_Pin, !Q);

  // далі відбувається циклічне виконання функції loop() з початку
}
```

Після компіляції програми нескладно перевірити її роботу у середовищі Proteus. Змінюючи стан входу T можна спостерігати за зміною стану виходів тригера (прямого та інверсного).

3.7. Програмна реалізація цифрових автоматів

У загальному плані *цифровим автоматом* називається пристрій, який формує ряд вихідних дискретних сигналів у відповідності із вхідною бінарною комбінацією сигналів. Таким чином, однією з особливостей цифрових автоматів є те, що вони мають свої внутрішні стани, від яких залежить реакція на вхідні сигнали. Одним з найпростіших прикладів цифрових автоматів є кодовий замок, реакція на черговий сигнал якого залежить від попередніх сигналів.

Скінченні автомати можуть бути *синхронними*, зміна станів яких відбувається в тактові моменти часу, що задаються зовнішнім генератором. В *асинхронних* автоматах зміна станів відбувається внаслідок дії вхідних сигналів без затримок.

Математичною моделлю цифрового автомата є абстрактний автомат, в якому враховуються його вхідні та вихідні сигнали, а також внутрішні стани. Будемо розглядати детерміновані автомати, які завжди мають початковий стан Q_0, з якого починають працювати при дії вхідних сигналів і при повторі перебору вхідних сигналів повторюють послідовність станів і вихідних сигналів.

Абстрактний автомат задається *множиною внутрішніх станів* $Q = \{ Q_1, Q_2, \ldots, Q_m \}$ (*алфавітом станів*), *множиною вхідних сигналів* $X = \{ X_1, X_2, \ldots, X_p \}$ (*вхідним алфавітом*), *множиною вихідних сигналів* $Y = \{ Y_1, Y_2, \ldots, Y_k \}$ (*вихідним алфавітом*) і *початковим станом* Q_0. Перехід з одного стану в інший визначається функцією переходів f_p, що визначає стан автомата Q_s, в який він переходить з попереднього стану Q_m при дії сигналу X_p:
$Q_s = f_p (Q_m, X_p)$.

Значення виходів автомата задається функцією виходів λ, що залежить від стану автомата Q_m і вхідного сигналу X_p:
$Y_k = \lambda (Q_m, X_p)$.

Абстрактний автомат працює в дискретному часі, який задається цілими позитивними числами $t = 0, 1, 2, \ldots$

У кожний момент дискретного часу t, який зветься *тактом*, автомат перебуває у деякому стані $Q(t)$ з множини станів автомата Q.

У початковий момент часу ($t = 0$) він завжди знаходиться в стані $Q(0) = Q_0$. Вважається, що реакція автомата на вхідні сигнали не залежить від інтервалів часу між тактовими моментами часу.

У момент t, знаходячись у стані $Q(t)$, автомат сприймає на своєму вході сигнали, які називаються *буквами* (*літерами*) *вхідного алфавіту* $X_p \in X$. У відповідності до функції переходів f_p, він перейде у стан $Q_s(t)$, що описується алфавітом станів, тобто $Q_s(t) \in Q$.

Аналогічно, у відповідності з функцією виходів λ, на виході отримається сигнал Y_k, де $Y_k \in Y$.

Як мови алгоритмізації та програмування в системах логічного керування залежно від типів керуючих обчислювальних пристроїв застосовуються: алгоритмічні мови високого рівня (C, Pascal), алгоритмічні мови низького рівня (асемблери), і спеціалізовані мови (наприклад, на базі ступінчатих і функціональних схем).

Крім того існує два підходи алгоритмізації систем цього класу. У першому з них вважається, що відомий алгоритм функціонування об'єкта керування і потрібно по ньому синтезувати алгоритм логічного керування, що забезпечує задане поводження. У другому підході враховується інформація про стан об'єкта керування. Таким чином, перша стратегія базується на понятті «стан», а друга – на понятті «подія».

Ні той, ні інший різновид керування в загальному випадку не є вичерпним, однак в SWITCH-технології пропонується зробити поняття «стан» первинним, а як мова специфікації для опису алгоритмів застосовувати графи переходів – пропонується представляти програму як систему взаємодіючих скінченних автоматів, що описуються графами переходів. Інакше кажучи, спочатку описується поводження керуючого пристрою, а не його структура, а побудова алгоритмів і програм починається з формування станів.

Об'єкти керування, як і система керування, можуть бути реалізовані за допомогою розглянутої технології.

3.7.1. Використання SWITCH-технології при реалізації автоматів Мура

Закон функціонування автомата Мура описується рівняннями:

$$\begin{cases} Q(t+1) = f_p\left[Q(t),\ X(t)\right]; \\ Y_k(t) = \lambda\left[Q(t)\right]; \qquad\qquad t = 0,\ 1,\ 2,\ \ldots \end{cases}$$

З другого рівняння видно, що вихідний сигнал автомата Мура залежить тільки від внутрішнього стану.

Нехай автомат Мура заданий графом переходів на рис. 3.42.

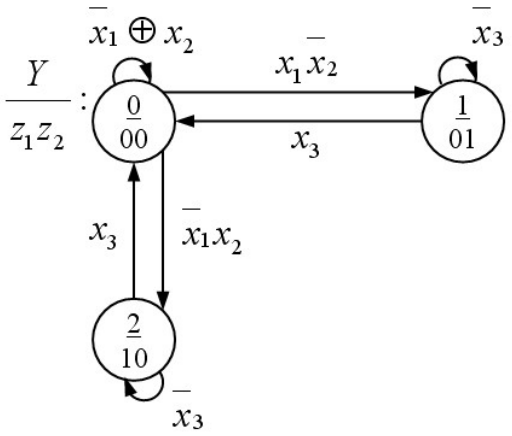


Рис. 3.42.

Реалізуємо принцип програмної реалізації цифрового автомату, застосовуючи дві конструкції switch:

```
switch(Y)
{
case 0: z1=0; z2=0; break;
case 0: z1=0; z2=1; break;
case 0: z1=1; z2=0; break;
}
switch(Y)
{
case 0: if ( x̄1 ⊕ x2) Y=0;
  if (x1 & x̄2 ) Y=1;
  if ( x̄1 & x2) Y=2;
break;
case 1: if (x3) Y=0;
            if ( x̄3 ) Y=1;
break;
case 2: if (x3) Y=0;
  if ( x̄3 ) Y=2;
break;
}
```

Відзначимо, що в цьому фрагменті програми для спрощення використовується позначення інверсії, що відрізняється від прийнятих в мові Сі.

На рис. 3.43 представлено схему, що використовується для перевірки роботи програмно реалізованого цифрового автомату у відповідності до наведеного вище графу переходів.

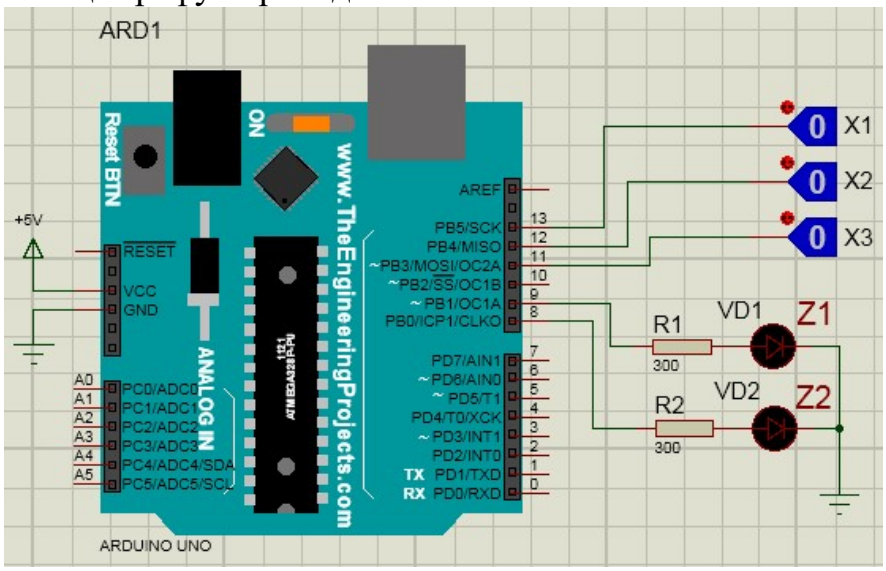


Рис. 3.43.

Текст програмної реалізації цифрового автомату наведено нижче.

```
// оголошення змінної, в якій зберігається номер виходу автомата Z1, до якого
підключено світлодіод VD1
```

```
int Z1_Pin=9;
// оголошення змінної, в якій зберігається номер виходу автомата Z2, до якого підключено світлодіод VD2
int Z2_Pin=8;

// оголошення змінної, в якій зберігається номер виводу X1
int X1_Pin=13;
// оголошення змінної, в якій зберігається номер виводу X2
int X2_Pin=12;
// оголошення змінної, в якій зберігається номер виводу X3
int X3_Pin=11;

// оголошення додаткових змінних, в яких зберігаються стани входів X1, X2 та X3 контролера (виводи 13, 12 та 11 відповідно)
int X1, X2, X3;
// оголошення додаткових змінних, в яких зберігаються стани виходів Z1 та Z2
int Z1, Z2;
// оголошення змінної, в якій зберігається поточний стан автомата
int Y;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи автомата), налаштовується на вихід
  pinMode(Z1_Pin, OUTPUT);
  pinMode(Z2_Pin, OUTPUT);

  // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються на вхід
  pinMode(X1_Pin, INPUT);
  pinMode(X2_Pin, INPUT);
  pinMode(X3_Pin, INPUT);

  // початкове значення змінної Y дорівнює 0 - це початкове значення стану автомата
  Y=0;
  // початкове значення станів виходів автомата Z1 та Z2 дорівнює 0
  Z1=0;
  Z2=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінні X1, X2 та X3 станів дискретних входів з номерами 13, 12 та 11
  // що зберігаються в змінних X1_Pin, X2_Pin та X3_Pin відповідно
  X1=digitalRead(X1_Pin);
  X2=digitalRead(X2_Pin);
  X3=digitalRead(X3_Pin);

  // цей оператор вибору однієї з альтернатив switch використовується
  // для керування станом виходів автомата
  switch(Y) // перевірка значення в змінній Y (поточний стан автомату)
```

```
  {
   case 0: // якщо автомат знаходиться в стані 0
     Z1=0; // запис у змінну Z1 значення 0
     Z2=0; // запис у змінну Z2 значення 0
   break;
   case 1: // якщо автомат знаходиться в стані 1
     Z1=0; // запис у змінну Z1 значення 0
     Z2=1; // запис у змінну Z2 значення 1
   break;
   case 2: // якщо автомат знаходиться в стані 2
     Z1=1; // запис у змінну Z1 значення 1
     Z2=0; // запис у змінну Z2 значення 0
   break;
   default: // якщо стан автомату невизначений
     Z1=0;  // запис у змінну Z1 значення 0
     Z2=0;  // запис у змінну Z2 значення 0
   break;
  }

  // цей оператор вибору однієї з альтернатив switch використовується
  // для керування переходами автомату в різні стани
  switch(Y)
  {
   case 0: // якщо автомат знаходиться в стані 0
     if(((X1==0) && (X2==0)) || ((X1==1) && (X2==1))) Y=0; // при виконанні умови
автомат залишається в стані 0
     if((X1==1) && (X2==0)) Y=1; // при виконанні умови автомат переходить в стан 1
     if((X1==0) && (X2==1)) Y=2; // при виконанні умови автомат переходить в стан 2
   break;
   case 1: // якщо автомат знаходиться в стані 1
     if(X3==1) Y=0;  // при виконанні умови автомат переходить в стан 0
     if(X3==0) Y=1;  // при виконанні умови автомат залишається в стані 1
   break;
   case 2: // якщо автомат знаходиться в стані 2
     if(X3==1) Y=0; // при виконанні умови автомат переходить в стан 0
     if(X3==0) Y=2; // при виконанні умови автомат залишається в стані 2
   break;
   default: // якщо стан автомату невизначений
     Y=0;   // автомат переходить в стан 0
   break;
  }

  // зміна станів виводів контролера у відповідності зі станом автомата
  // значення на виході Z1
  digitalWrite(Z1_Pin, Z1);
  // значення на виході Z2
  digitalWrite(Z2_Pin, Z2);

  // далі відбувається циклічне виконання функції loop() з початку
}
```

На рис. 3.44 представлено результати моделювання роботи цифрового автомату. При появі на вході X1 високого рівня сигналу, а на вході X2 низького рівня сигналу, автомат переходить в стан Y=1. В цьому стані на виході Z1 встановлюється значення лог.0, а на виході Z2 встановлюється значення лог.1. При появі на вході X3 високого рівня, автомат перейде в початковий стан Y=0, і на обох виходах встановиться низький рівень сигналу. Далі, сформувавши на виході X2 високий рівень сигналу, автомат переходить в стан Y=2, в якому на виході Z1 встановлюється високий рівень, а на виході Z2 низький рівень сигналу.

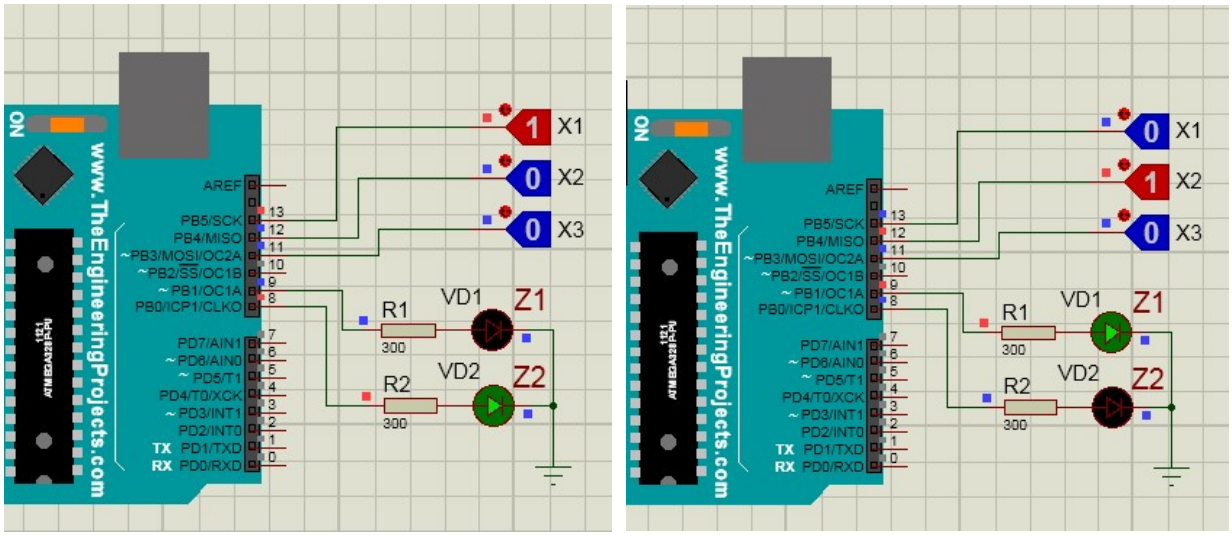


Рис. 3.44.

Текст цієї програми може бути спрощений за рахунок виключення по другій конструкції switch рядків, що відповідають за збереження попереднього стану:

```
 // цей оператор вибору однієї з альтернатив switch використовується
 // для керування переходами автомату в різні стани
 switch(Y)
 {
  case 0: // якщо автомат знаходиться в стані 0
  /*   if(((X1==0) && (X2==0)) || ((X1==1) && (X2==1))) Y=0; // при виконанні умови
автомат залишається в стані 0
*/
    if((X1==1) && (X2==0)) Y=1; // при виконанні умови автомат переходить в стан 1
    if((X1==0) && (X2==1)) Y=2; // при виконанні умови автомат переходить в стан 2
   break;
   case 1: // якщо автомат знаходиться в стані 1
    if(X3==1) Y=0;  // при виконанні умови автомат переходить в стан 0
   /* if(X3==0) Y=1;  // при виконанні умови автомат залишається в стані 1*/
   break;
   case 2: // якщо автомат знаходиться в стані 2
    if(X3==1) Y=0; // при виконанні умови автомат переходить в стан 0
   /* if(X3==0) Y=2; // при виконанні умови автомат залишається в стані 2 */
```

```
  break;
  default: // якщо стан автомату невизначений
    Y=0;   // автомат переходить в стан 0
  break;
}
```

Ця програмна модель повністю ізоморфна графу переходів, по якому вона побудована. У приведеній програмі значення вихідних змінних, що не змінюються під час переходу, закоментовані, і в об'єктний код не транслюються.

3.7.2. Використання SWITCH-технології при реалізації автоматів Мілі

Закон функціонування автомата Мілі задається рівняннями:

$$\begin{cases} Q(t+1) = f_p\left[Q(t), X(t)\right]; \\ Y_k(t) = \lambda\left[Q(t), X(t)\right]; \qquad t = 0,\ 1,\ 2,\ \ldots \end{cases}$$

В автоматах Мілі значення вихідного сигналу залежить від внутрішнього стану і від вхідного сигналу.

Нехай автомат Мілі заданий графом переходів на рис. 3.45.

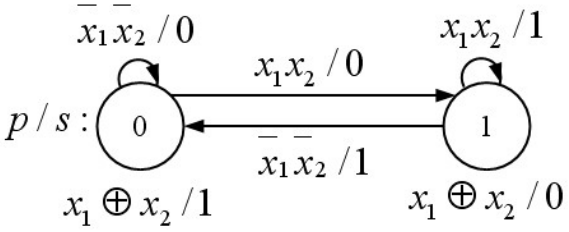


Рис. 3.45.

Програмна модель, що відповідає цьому графу переходів, має вигляд:

```
switch (P)
{
case 0: if ( x̄1 & x̄2 ) S=0;
          if (x1 & x2) S=0;
          if (x1 ⊕ x2) S=1;
break;
case 1: if ( x̄1 & x̄2 ) S=1;
          if (x1 & x2) S=1;
          if (x1 ⊕ x2) S=0;
break;
}
switch (P)
{
case 0: if ( x̄1 & x̄2 ) P=0;
          if (x1 ⊕ x2) P=0;
          if (x1 & x2) P=1;
```

```
break;
case 1: if ( x̄2  &  x̄2 ) P=1;
          if (x1 ⊕ x2) P=1;
          if (x1 & x2) P=0;
break;
}
```

Для реалізації цифрового автомата Мілі використовується схема, наведена на рис. 3.46.

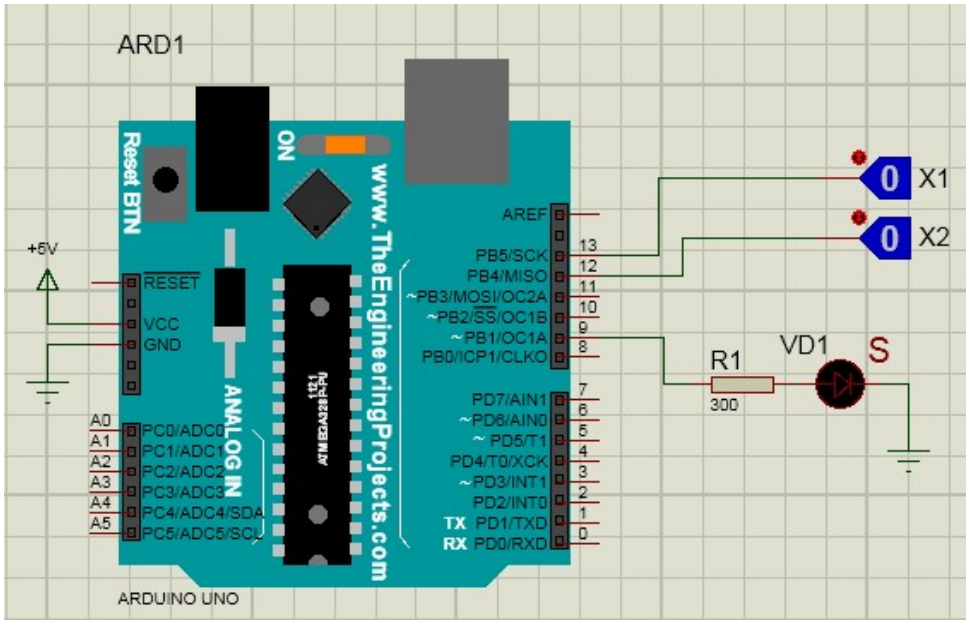


Рис. 3.46.

Текст програмної реалізації автомата Мілі представлено нижче.

```
// оголошення змінної, в якій зберігається номер виходу автомата S, до якого підключено світлодіод VD1
int S_Pin=9;

// оголошення змінної, в якій зберігається номер виводу X1
int X1_Pin=13;
// оголошення змінної, в якій зберігається номер виводу X2
int X2_Pin=12;

// оголошення додаткових змінних, в яких зберігаються стани входів X1, X2 та X3 контролера (виводи 13, 12 та 11 відповідно)
int X1, X2;
// оголошення додаткової змінної, в якій зберігається стан виходу S
int S;
// оголошення змінної, в якій зберігається поточний стан автомата
int P;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
```

```
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
 // вивід, до якого підключено світлодіод (вихід автомата), налаштовується на вихід
 pinMode(S_Pin, OUTPUT);

 // виводи, до яких підключено джерела вхідних дискретних сигналів налаштовуються на вхід
 pinMode(X1_Pin, INPUT);
 pinMode(X2_Pin, INPUT);

 // початкове значення змінної Y дорівнює 0 - це початкове значення стану автомата
 P=0;
 // початкове значення стану виходу автомата S дорівнює 0
 S=0;
}

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
 // зчитування в змінні X1 та X2 станів дискретних входів з номерами 13 та 12
 // що зберігаються в змінних X1_Pin та X2_Pin відповідно
 X1=digitalRead(X1_Pin);
 X2=digitalRead(X2_Pin);

 // цей оператор вибору однієї з альтернатив switch використовується
 // для керування станом виходів автомата
 switch(P) // перевірка значення в змінній P (поточний стан автомату)
 {
  case 0: // якщо автомат знаходиться в стані 0
    if((X1==0) && (X2==0)) S=0;   // перевірка істинності виразу  x̄1x̄2
    if((X1==1) && (X2==1)) S=0;   // перевірка істинності виразу  x1x2
        // перевірка істиності виразу  x1 ⊕ x2
    if(((X1==0) && (X2==1)) || ((X1==1) && (X2==0))) S=1;
  break;
  case 1: // якщо автомат знаходиться в стані 1
    if((X1==0) && (X2==0)) S=1;    // перевірка істинності виразу  x̄1x̄2
    if((X1==1) && (X2==1)) S=1;   // перевірка істинності виразу  x1x2
        // перевірка істинності виразу  x1 ⊕ x2
    if(((X1==0) && (X2==1)) || ((X1==1) && (X2==0))) S=0;
  break;
  default: // якщо стан автомату невизначений
    S=0;   // змінній, що визначає стан виходу автомата, присвоюється значення 0
  break;
 }

 // цей оператор вибору однієї з альтернатив switch використовується
 // для керування переходами автомату в різні стани
 switch(P) // перевірка значення в змінній P (поточний стан автомату)
 {
  case 0: // якщо автомат знаходиться в стані 0
    if((X1==0) && (X2==0)) P=0;
```

```
    if(((X1==0) && (X2==1)) || ((X1==1) && (X2==0))) P=0;
    if((X1==1) && (X2==1)) P=1;
   break;
   case 1: // якщо автомат знаходиться в стані 1
    if((X1==0) && (X2==0)) P=0;
     if(((X1==0) && (X2==1)) || ((X1==1) && (X2==0))) P=1;
    if((X1==1) && (X2==1)) P=1;
   break;
   default: // якщо стан автомату невизначений
    P=0;   // автомат переходить в стан 0
   break;
  }

  // зміна станів виводів контролера у відповідності зі станом автомата
  // значення на виході S
  digitalWrite(S_Pin, S);

  // далі відбувається циклічне виконання функції loop() з початку
}
```

Ця модель ізоморфна графу переходів, по якому вона будувалася, і тому її доцільно застосовувати для реалізації графу переходів автоматів Мілі.

3.8. Лічильники імпульсів

У загальному плані *лічильниками* називаються цифрові пристрої (автомати), призначені для підрахунку і фіксації кількості імпульсів, що подаються на їх інформаційні входи або синхровходи. Назва *"лічильники"* використовується до будь-яких послідовнісних цифрових пристроїв із замкнутим циклом діаграми станів.

Робота найпростішого двійкового лічильника пояснюється даними, приведеними у табл. 3.9 і часовими діаграмами, зображеними на рис. 3.47.

Таблиця 3.9.

Вхідні імпульси (N)	Q_8	Q_4	Q_2	Q_1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

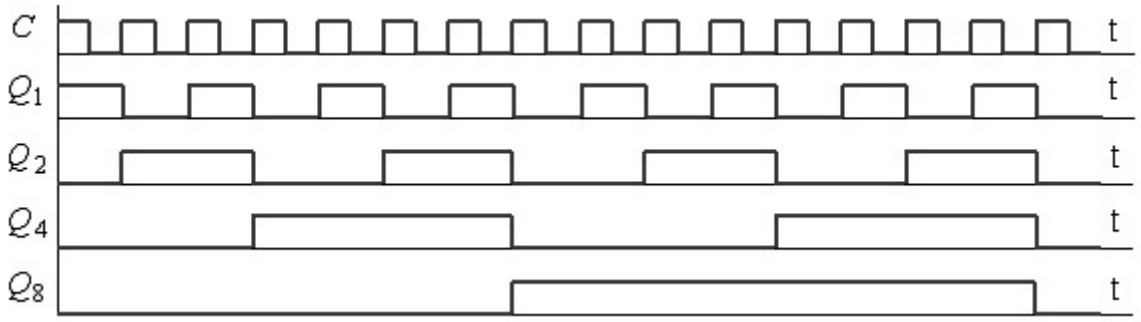


Рис. 3.47.

Якщо на вхід лічильника подано N імпульсів, то їх кількість, підрахована за допомогою лічильника, відповідає формулі:

$$N = K\ M + a_0\ 2^0 + a_1\ 2^1 + \ldots + a_{n-1}\ 2^{n-1} = n\ M + \sum_{i=0}^{n-1} a_i\ 2^i\ ,$$

де K – кількість вихідних імпульсів лічильника; $a_i \in 1, 0$ – рівень сигналу на i-ому виході лічильника; 2^i – вагові коефіцієнти кожного прямого виходу. Для урахування вагових коефіцієнтів виходи лічильника нумерують відповідними індексами. Це дає можливість з послідовності значень виходів $Q_8\ Q_4\ Q_2\ Q_1$, рівних, наприклад, 1011, одразу ж зчитувати вміст лічильника, тобто кількість імпульсів, яка менша M.

Зазвичай код, що визначається вихідними станами тригерів, змінюється у зростаючому напрямку (додаючі лічильники). Операція збільшення вмісту лічильника на одиницю називається інкрементуванням.

При зворотній зміні станів лічильники звуться віднімаючими, а якщо напрямок рахунку може змінюватися, то лічильник має назву реверсивного. Операція зменшення вмісту лічильника на одиницю називається декрементуванням.

На рис. 3.48 представлено схему для дослідження роботи двійкового лічильника. Лічильник має лише один вхід, на який подаються імпульси. До виходів Q3-Q0 лічильника (це виводи 6, 7, 8 та 9 контролера Arduino) підключено світлодіоди, за допомогою яких можна спостерігати значення кількості імпульсів, що поступили на вхід лічильника, представлене в двійковому коді.

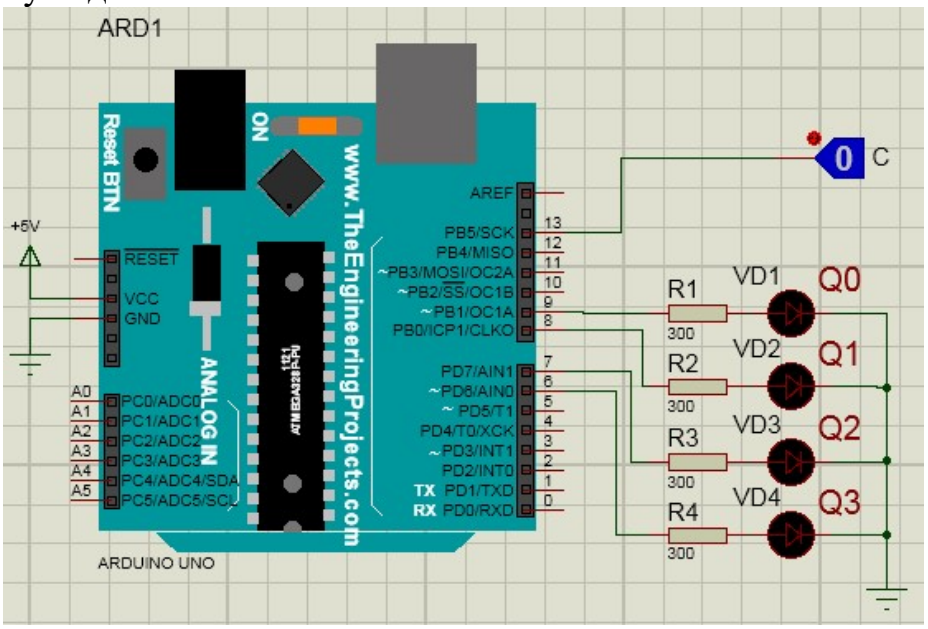


Рис. 3.48.

Текст програмної реалізації двійкового лічильника з коефіцієнтом перерахунку 15 на базі наведеної схеми має вигляд:

```
// оголошення змінної, в якій зберігається номер виходу Q0, до якого підключено світлодіод VD1
int Q0_Pin=9;
// оголошення змінної, в якій зберігається номер виходу Q1, до якого підключено світлодіод VD2
int Q1_Pin=8;
// оголошення змінної, в якій зберігається номер виходу Q2, до якого підключено світлодіод VD3
int Q2_Pin=7;
// оголошення змінної, в якій зберігається номер виходу Q3, до якого підключено світлодіод VD4
int Q3_Pin=6;

// оголошення змінної, в якій зберігається номер виводу C
int C_Pin=13;

// оголошення додаткової змінної, в яких зберігається стан входу C (вивід 13)
int C;
```

```
// оголошення додаткової змінної, в якій зберігається значення лічильника
int value;

// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи лічильника), налаштовується на вихід
  pinMode(Q0_Pin, OUTPUT);
  pinMode(Q1_Pin, OUTPUT);
  pinMode(Q2_Pin, OUTPUT);
  pinMode(Q3_Pin, OUTPUT);

   // вивід, до якого підключено джерело вхідного тактового сигналу, налаштовується на вхід
  pinMode(C_Pin, INPUT);

  // початкове значення змінної value дорівнює 0 - це початкове значення лічильника
  value=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну C стану дискретного входу з номером 13
  // що зберігається в змінній C_Pin
  C=digitalRead(C_Pin);

  if(C==1)       // якщо на тактовому вході C присутній високий рівень сигналу
  {
    if(value<15)          // якщо значення змінної value менше 15
    {
      value++;   // збільшення значення в змінній value на 1 (інкремент)
    }
    else             // якщо значення змінної value дорівнює 15, або більше 15
    {
      value=0;   // змінній value присвоюється значення 0
    }
    while(C==1)          // цикл виконується поки на вході C залишається високий рівень сигналу
    {
          // в циклі відбувається «опитування» тактового входу C
      C=digitalRead(C_Pin);
    }
  }

  // зміна станів виводів контролера у відповідності зі значенням лічильника
  // значення на виході Q0
  digitalWrite(Q0_Pin, bitRead(value, 0));
  // значення на виході Q1
  digitalWrite(Q1_Pin, bitRead(value, 1));
  // значення на виході Q2
```

```
  digitalWrite(Q2_Pin, bitRead(value, 2));
  // значення на виході Q3
  digitalWrite(Q3_Pin, bitRead(value, 3));

  // далі відбувається циклічне виконання функції loop() з початку
}
```

В наведеній програмі було використано функцію bitRead(…) для отримання значення потрібного біту змінної value та зміни стану відповідного виходу лічильника. Синтаксис цієї функції має вигляд:

bitRead (x, n);

Параметри функції:

x: число, біт якого слід зчитати

n: номер біту, який слід зчитати, починаючи з молодшого (правого).

Функція повертає стан біту (0 або 1).

Розподілювачами називають функціональні вузли, які циклічно розподіляють потік імпульсів по декількох каналах. Прикладом розподілювача може служити схема двійкового лічильника, на вхід якого подаються імпульси для підрахунку, а виходи приєднані до дешифратора. Збільшуваний або зменшуваний вихідний двійковий код лічильника дешифрується дешифратором по ряду каналів, створюючи в кожному з них послідовність імпульсів, що задається циклом роботи схеми. Функцію розподілювача може виконувати цифровий автомат. У літературі такі схеми, у залежності від характеру зворотних зв'язків, називають кільцевими лічильниками, лічильниками Джонсона, лічильниками Мебіуса.

Часова діаграма лічильника при подачі серії синхроімпульсів має вигляд, приведений на рис. 3.49.

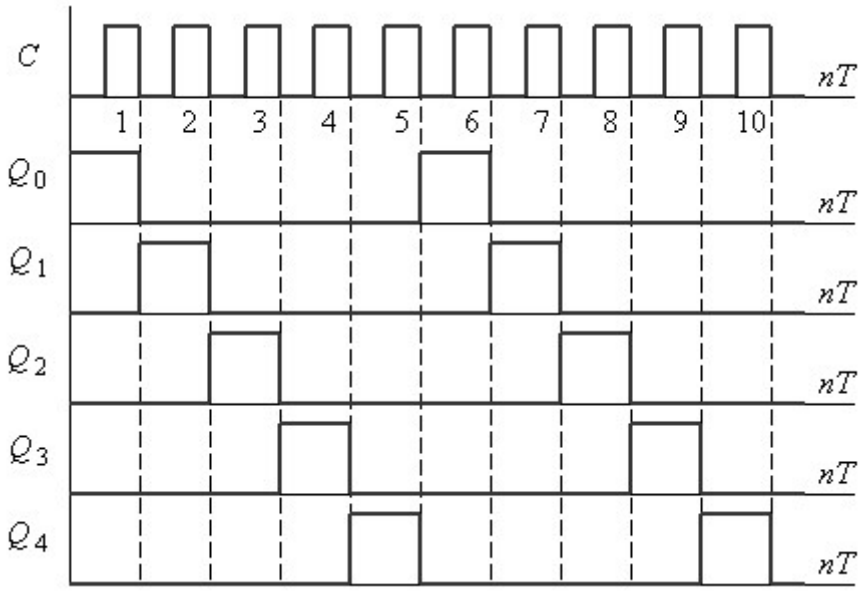


Рис. 3.49.

Схема для дослідження роботи кільцевого лічильника представлена на рис. 3.50. Ця схема принципово не відрізняється від звичайного

двійкового лічильника, різниця полягає лише в алгоритмі роботи програми.

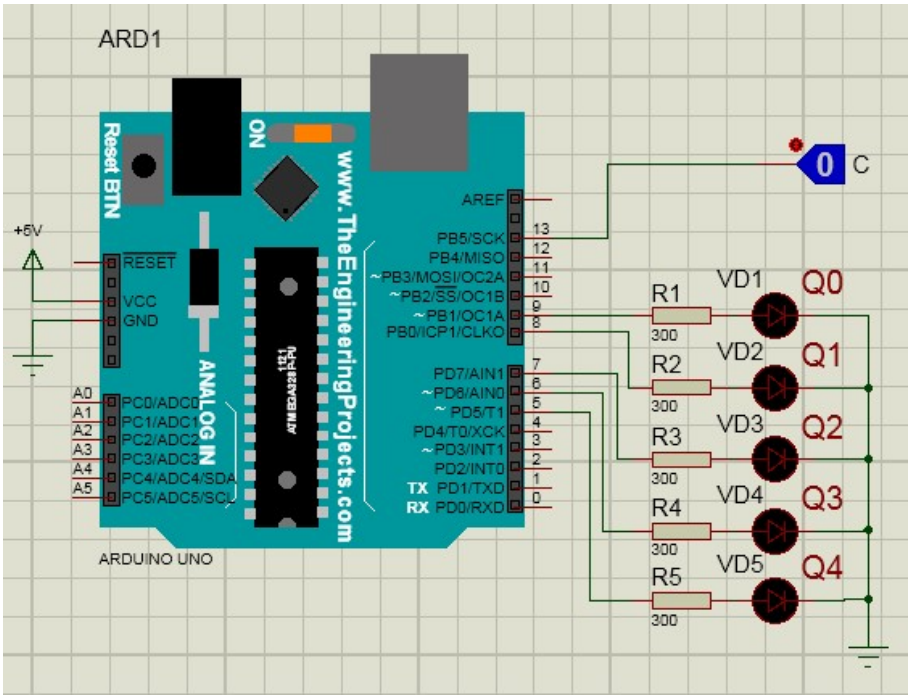


Рис. 3.50.

Текст програми, що реалізує розподілювач імпульсів (кільцевий лічильник), представлено нижче.

```
// оголошення змінної, в якій зберігається номер виходу Q0, до якого підключено світлодіод VD1
int Q0_Pin=9;
// оголошення змінної, в якій зберігається номер виходу Q1, до якого підключено світлодіод VD2
int Q1_Pin=8;
// оголошення змінної, в якій зберігається номер виходу Q2, до якого підключено світлодіод VD3
int Q2_Pin=7;
// оголошення змінної, в якій зберігається номер виходу Q3, до якого підключено світлодіод VD4
int Q3_Pin=6;
// оголошення змінної, в якій зберігається номер виходу Q4, до якого підключено світлодіод VD5
int Q4_Pin=5;
// оголошення змінної, в якій зберігається номер виводу C
int C_Pin=13;

// оголошення додаткової змінної, в яких зберігається стан входу C (вивід 13)
int C;
int i;
// оголошення масиву з 5 елементів, в якому зберігається значення вихідних сигналів в кожному з 5 станів кільцевого лічильника
int value[5]={0b00000001, 0b00000010, 0b00000100, 0b00001000, 0b00010000};
```

```
// в цій функції відбувається налаштування виводів контролера на вхід або на вихід;
// функція setup() виконується один раз при вмиканні живлення контролера, або при перезавантаженні
void setup() {
  // виводи, до яких підключено світлодіоди (виходи лічильника), налаштовується на вихід
  pinMode(Q0_Pin, OUTPUT);
  pinMode(Q1_Pin, OUTPUT);
  pinMode(Q2_Pin, OUTPUT);
  pinMode(Q3_Pin, OUTPUT);
  pinMode(Q4_Pin, OUTPUT);

  // вивід, до якого підключено джерело вхідного тактового сигналу, налаштовується на вхід
  pinMode(C_Pin, INPUT);

  // початкове значення змінної i дорівнює 0
  i=0;
 }

// ця функція виконується циклічно увесь час, поки на контролер подається живлення
void loop() {
  // зчитування в змінну C стану дискретного входу з номером 13,
  // що зберігається в змінній C_Pin
  C=digitalRead(C_Pin);

  if(C==1) // якщо на тактовому вході C присутній високий рівень сигналу
  {
    if(i<4) // якщо значення змінної менше 4
    {
     i++;// відбувається збільшення на 1 вмісту змінної
    }
    else  // інакше
    {
      i=0;          // змінній присвоюється значення 0
    }
          // цикл виконується поки на вході C залишається високий рівень сигналу
    while(C==1)
    {
          // в циклі відбувається «опитування» тактового входу C
      C=digitalRead(C_Pin);
    }
  }

  // зміна станів виводів контролера у відповідності зі значенням лічильника
  // значення на виході Q0
  digitalWrite(Q0_Pin, bitRead(value[i], 0));
  // значення на виході Q1
  digitalWrite(Q1_Pin, bitRead(value[i], 1));
  // значення на виході Q2
  digitalWrite(Q2_Pin, bitRead(value[i], 2));
  // значення на виході Q3
```

```
  digitalWrite(Q3_Pin, bitRead(value[i], 3));
  // значення на виході Q4
  digitalWrite(Q4_Pin, bitRead(value[i], 4));
  // далі відбувається циклічне виконання функції loop() з початку
}
```

Результати моделювання роботи кільцевого лічильника наведено на рис. 3.51.

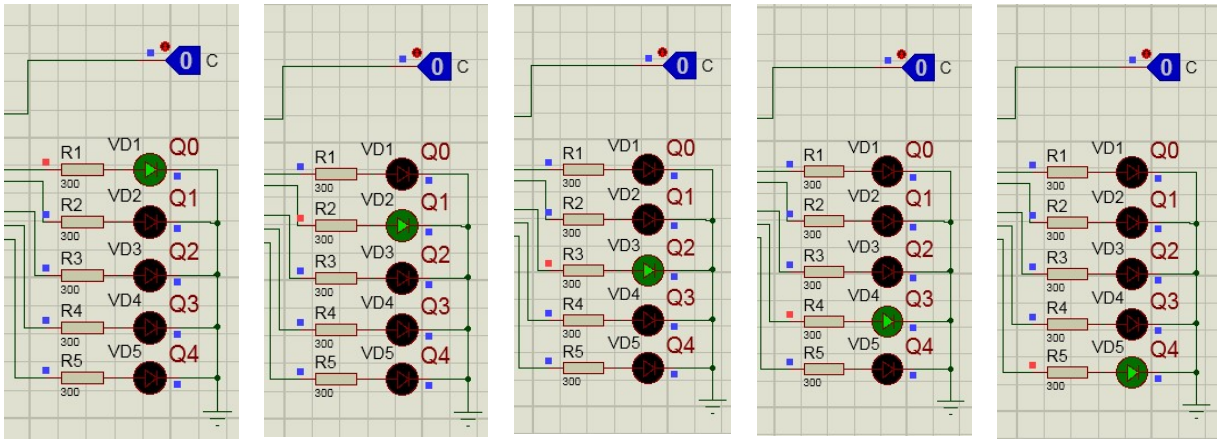


Рис. 3.51.

При послідовній зміні станів на вході С схеми, за допомогою світлодіодів можна спостерігати зміну станів виходів, яка повністю відповідає часовій діаграмі, яку було розглянуто раніше.

Завдання до розділу 3

1. Реалізувати програмно наступні логічні висловлювання: *а*) “якщо *A*, то *B*”; *б*) “якщо *B*, то *C* і *D*”; *в*) “якщо *A*, то *B* або *C*”; *г*) “*C* тільки при *A* і *B*”; *д*) “*D* при будь-яких двох *A*, *B*, *C*”; *е*) “*A* при *B* або *C*”.
2. Розробити програму, що дозволяє реалізувати на контролері Arduino логічну функцію, що реалізується схемою на рис. 3.52.

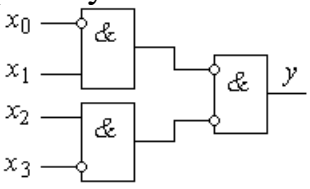


Рис. 3.52

3. На рис. 3.53 приведена логічна схема цифрового пристрою. Побудувати таблицю станів, записати логічні функції по відношенню до її виходів, пояснити призначення цього пристрою. Розробити принципову схему системи з використанням контролера Arduino та написати програму, що реалізує логіку роботи цього пристрою.

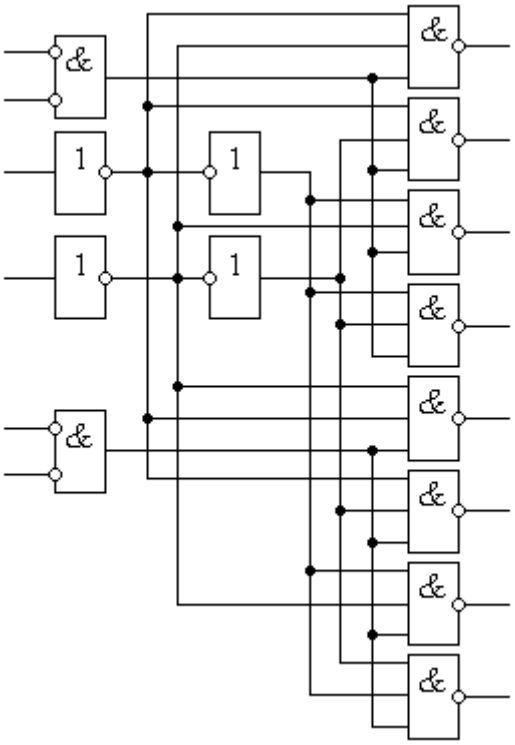


Рис. 3.53

4. Розробити пристрій зсуву однобайтного слова на базі контролера Arduino, та написати відповідну програму. Обґрунтувати організацію адресних сигналів з метою забезпечення у першому розряді будь-якого біта вхідного слова.
5. Спроектувати селектор-мультиплексор для комутації трьох п’ятибітних слів на загальну п’ятибітну шину.
6. Розробити пристрій на базі контролера Arduino для реалізації системи логічних функцій:

$$y_1 = \bigvee_{0}^{15} 2, 4, 6, 14;$$

$$y_2 = \bigvee_{0}^{15} 0, 1, 2, 3, 5, 7, 11, 13;$$

$$y_3 = \bigvee_{0}^{15} 1, 3, 4, 5, 12, 14, 15.$$

7. Спроектувати схему пристрою перевірки на чотири входи $x_3 \ldots x_0$ і один вихід *Y*. Вихід *Y* має одиничне значення, якщо два і більше входів дорівнюють одиницям, і нульове, якщо або на всіх входах сигнал дорівнює нулю або лише на одному з них наявний одиничний сигнал. Написати відповідну програму.
8. Розробити схему дешифратора “з 10 на 4” для перетворення десяткових цифр від 0 до 9 у двійковий код і написати відповідну програму для контролера Arduino.

9. Виконати програму реалізацію дешифратора, алгоритм функціонування якого описується приведеною таблицею відповідності (табл. 3.10).

Таблиця 3.10.

N	S_2	S_1	S_0	Виходи
0	0	0	0	***A***
1	0	0	1	***B***
2	0	1	0	***A***
3	0	1	1	***C***
4	1	0	0	***A***
5	1	0	1	***D***
6	1	1	0	***A***
7	1	1	1	***E***

10. Написати програму, що виконує додавання 2-х чисел. Результат виводити в двійковому коді на світлодіодні індикатори.
11. Реалізувати двійковий лічильник. Значення лічильника відображати на світлодіодних індикаторах. Інкремент здійснюється автоматично з періодом 200 мс. При досягненні значення 100 лічильник починає рахувати у зворотньому напрямку до 0. Потім цикл повторюється.
12. Реалізувати лічильник з коефіцієнтом перерахунку M = 147.
13. Написати програму – кодовий замок.
14. Розробити реверсивний лічильник. При натисканні на одну кнопку лічильник інкрементується, при натисканні на іншу кнопку значення лічильника зменшується. Натискання третьої кнопки обнуляє лічильник.
15. Розробити суматор напівбайтних слів. Старша тетрада додається до молодшої тетради, результат виводиться на семисегментні індикатори.
16. Виконати програмну реалізацію тактованого фронтом двовходового тригера, що функціонує відповідно до табл. 3.11.

Таблиця 3.11

x_1	x_0	Q_{n+1}
0	0	0
0	1	Q_n
1	0	$\overline{Q_n}$
1	1	1

17. Побудувати граф переходів для кінцевого автомату, що описується таблицею переходів (табл. 3.12), та виконати його програмну реалізацію.

Таблиця 3.12.

Q \ X	$x_1\ x_0$				Y
	00	01	11	10	
Q_0	Q_1	Q_1	Q_2	Q_2	0
Q_1	Q_3	Q_3	Q_2	Q_2	0
Q_2	Q_1	Q_1	Q_4	Q_4	0
Q_3	Q_3	Q_3	Q_4	Q_2	1
Q_4	Q_1	Q_3	Q_4	Q_4	1

18. Побудувати граф-схему автомату, заданого таблицею переходів (табл. 3.13). Виконати його програмну реалізацію.

Таблиця 3.13.

Q \ X	$x_1\ x_0$				Y
	00	01	11	10	
Q_0	Q_0	Q_1	Q_2	Q_1	1
Q_1	Q_1	Q_2	Q_3	Q_2	0
Q_2	Q_2	Q_3	Q_0	Q_3	0
Q_3	Q_3	Q_0	Q_1	Q_0	0

РОЗДІЛ 4. ЦИФРОВІ ІНТЕРФЕЙСИ МІКРОПРОЦЕСОРНИХ СИСТЕМ КЕРУВАННЯ, ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ

4.1. Загальна характеристика цифрових інтерфейсів

Зв’язок пристроїв в мікропроцесорних системах керування, збору та обробки інформації один з одним здійснюють за допомогою сполучень, які називаються інтерфейсами. Інтерфейс – це сукупність інформаційно-логічних і конструктивних засобів і правил, які забезпечують оптимальний алгоритм взаємодії всіх модулів в мікропроцесорній системі.

Інформаційно-логічні вимоги визначають структуру в складі ліній і сигналів, способи кодування і формати даних, адрес і команд, протоколів обміну для різних режимів роботи. Вони безпосередньо впливають на пропускну здатність інтерфейсу, надійність обміну даними та апаратурні витрати.

Інтерфейси мають забезпечувати:

– побудову мікропроцесорних систем із змінним складом обладнання (змінною конфігурацією, відкритістю архітектури);

– паралельне у часі виконання програм і процедур введення-виведення;

– збільшення швидкості обміну інформацією;

– спрощення і стандартизацію програмування операцій введення-виведення та їхню незалежність від особливостей периферійних пристроїв;

– автоматичне розпізнавання і реакцію мікропроцесора на різноманітні ситуації периферійного пристрою (готовність пристрою, відсутність носія даних, порушення нормальної роботи).

Термін «інтерфейс» застосовують як до апаратури мікропроцесорної системи, так до її програмного забезпечення.

Інтерфейси розподіляють на:

– Внутрішні – внутрішньо платні, між платні, системні.

– Зовнішні – для периферійних пристроїв, локальних мереж, розподілених систем керування.

Інтерфейси також класифікують за:

– напрямком обміну інформації;

– структурою зв’язків між абонентами;

– способом передачі даних тощо.

В даному розділі розглядаються внутрішні інтерфейси мікропроцесорних систем керування, збору та обробки інформації.

4.2. Однопровідний інтерфейс 1-Wire

Однопровідний інтерфейс 1-Wire, розроблений наприкінці 90-х років фірмою Dallas Semiconductor Corp., регламентований розробниками для застосування в трьох основних сферах:

– прилади в спеціальних корпусах MicroCAN для рішення проблем ідентифікації, переносу або перетворення інформації (технологія iButton);

– програмування вбудованої пам'яті інтегральних компонентів;

– системи автоматизації (технологія мереж 1-Wire).

Переваги 1-Wire:

- просте й оригінальне рішення адресації абонентів;
- зручний протокол;
- проста структура лінії зв'язку;
- невелика кількість необхідних компонентів;
- легка зміна конфігурації мережі;
- значна довжина ліній зв'язку;
- невелика вартість всієї технології в цілому.

4.2.1. Загальна характеристика інтерфейсу 1-Wire

Мережа 1-Wire представляє собою інформаційну мережу, що використовує для здійснення цифрового зв'язку одну лінію даних та один загальний провід. Таким чином, для реалізації середовища обміну цієї мережі можуть бути застосовані доступні кабелі, що містять неекрановану кручену пару, і навіть звичайний телефонний провід. Такі кабелі при їхній прокладці не вимагають наявності якого-небудь спеціального устаткування, а обмеження максимальної довжини однопровідної лінії регламентовано розробниками на рівні 300 м. Основою архітектури мереж 1-Wire є топологія загальної шини, коли кожний із пристроїв підключений безпосередньо до єдиної магістралі, без яких-небудь каскадних з'єднань або розгалужень. При цьому в якості базової використовується структура мережі з одним ведучим, або майстром, і багатьма веденими. Хоча існує ряд специфічних прийомів організації роботи однопровідних систем у режимі мультимастера. Конфігурація будь-якої 1-Wire-мережі може довільно змінюватися в процесі її роботи, не створюючи перешкод подальшої експлуатації та працездатності всієї системи в цілому, якщо при цих змінах дотримуються основні принципи організації однопровідної шини. Ця можливість досягається завдяки присутності в протоколі 1-Wire-інтерфейсу спеціальної команди пошуку ведених пристроїв (пошук ПЗП), що дозволяє швидко визначити нових учасників інформаційного обміну. Стандартна швидкість відпрацьовування такої команди становить ~75 вузлів мережі в секунду.

Завдяки наявності в складі будь-якого пристрою з мережною версією 1-Wire-інтерфейсу унікальної індивідуальної адреси (відсутність збігу адрес для приладів, що випускаються компанією Dallas Semiconductor Corp, гарантується самою фірмою-виробником), така мережа має практично необмежений адресний простір. При цьому кожний з однопровідних приладів відразу готовий до використання в складі 1-Wire-мережі, без яких-небудь додаткових апаратно-програмних модифікацій. Однопровідні компоненти є самотактованими напівпровідниковими пристроями, в основі обміну інформацією між якими лежить керування зміною тривалості часових інтервалів імпульсних сигналів в однопровідному середовищі, та їх вимірювання. Передача сигналів, для 1-

Wire-інтерфейсу, асинхронна й напівдуплексна, а вся інформація, що циркулює в мережі, сприймається абонентами або як команди, або як дані. Команди мережі генеруються майстром і забезпечують різні варіанти пошуку й адресації ведених пристроїв, визначають активність на лінії навіть без безпосередньої адресації окремих компонентів, керують обміном даними в мережі й т.д.

Стандартна швидкість роботи 1-Wire-мережі, яка становить 15,4 кБіт/с, була обрана, по-перше, з урахуванням забезпечення максимальної надійності передачі даних на великі відстані, і по-друге, з урахуванням швидкодії найбільш широко розповсюджених типів мікроконтролерів, які в основному повинні використовуватися при реалізації ведучих пристроїв однопровідної шини. Це значення швидкості обміну може бути зменшене до будь-якого можливого значення завдяки введенню примусової затримки між передачею в лінію окремих бітів даних (розтягуванню часових слотів протоколу), або збільшено за рахунок переходу на спеціальний прискорений режим обміну (швидкість Overdrive – до 125 кБіт/с), що допускається для окремих типів однопровідних компонентів на невеликий відстані, не перевантаженої іншими приладами лінії зв'язку.

Окремі види адаптерів, які дозволяють наділити будь-який персональний комп'ютер можливістю обслуговувати як майстер 1-Wire-мережі, випускаються самою фірмою Dallas Semiconductor Corp. До них відносяться адаптери для паралельного порту типу DS1410E, для COM-порту типу DS9097E та DS9097U, для USB-порту типу DS9490R. Ці прилади мають різні функціональні можливості й конструктивні особливості, що забезпечує розробнику максимальну зручність при конструюванні.

Часто як ведучий однопровідної шини виступає не комп'ютер, а найпростіший універсальний мікроконтролер. Для організації його сполучення з 1-Wire-мережею використовуються різні програмно-апаратні методи.

Чотирьохканальний однопровідний АЦП типу DS2450 та двоканальний однопровідний лічильник, сполучений з буферною пам'яттю, типу DS2423 дозволяють вирішувати завдання, пов'язані з оцифровуванням аналогових та імпульсно-часових сигналів. Найбільш незамінними при автоматизації є універсальні здвоєні транзисторні ключі типу DS2406P. На базі цих пристроїв можуть бути реалізовані вузли контролю логічних станів (рівнів) і схеми обслуговування датчиків "сухого контакту", а також різноманітні ключові схеми. Саме завдяки використанню цих компонентів здійснюється збір дискретної інформації з територіально розосереджених датчиків (моніторів дверей, контакторів положення арматури, будь-яких датчиків, що мають вихід ТАК/НІ – датчики положення, проходу, присутності, пожежної та охоронної сигналізації та ін.). Подібні ж прилади забезпечують керування перемиканням будь-яких видів силового устаткування, які мають два робочих стани: включене/виключене (нагрівачів, кондиціонерів, моторів,

вентиляторів і т.д.). Значно розширює можливості однопровідних мереж по аналоговому керуванню розосередженим, у тому числі силовим, устаткуванням цифровий потенціометр DS2890 укомплектований мережним 1-Wire-інтерфейсом. Використовуючи цей прилад можна створювати найрізноманітніші системи керування, завдяки можливості плавної зміни аналогового регулюючого сигналу по 256 градаціям.

4.2.2. Блок-схема датчика температури DS18B20

Найбільш популярними компонентами, на базі яких реалізована найбільша кількість пристроїв, є цифрові термометри типу DS18S20. Розглянемо принципи роботи й програмування пристроїв з 1-Wire на прикладі датчика температури DS18B20 і мікроконтролера AT90S8535.

Широко розповсюджена мікросхема цифрового термометра DS18B20 забезпечує вимір температури в діапазоні –55...+125°С. Точність перетворення температури може бути від 9 до 12 розрядів (налаштовується). DS18B20 допускає напругу живлення від +3 до +5,5 В. У режимі очікування споживаний струм близький до нуля (менше 1мкА), а під час перетворення температури він дорівнює приблизно 1 мА. Процес перетворення триває максимум 750 мс. Температурні датчики дозволяють не тільки здійснювати безпосередній моніторинг температури в режимі реального часу, але й завдяки наявності вбудованої енергонезалежної пам'яті температурних уставок, можуть забезпечувати пріоритетну сигналізацію в лінію про факт виходу контрольованого параметра за межі заданих значень.

Принцип дії такого цифрового датчика температури заснований на підрахунку кількості імпульсів, що виробляються генератором з низьким температурним коефіцієнтом у часовому інтервалі, який формується генератором з більшим температурним коефіцієнтом. Лічильник ініціюється значенням, що відповідає –55 °С (мінімальній вимірюваній температурі). Якщо лічильник досягає нуля перед тим, як закінчується часовий інтервал (це означає, що температура більша –55 °С), то регістр температури, що також ініційований значенням –55 °С, інкрементується. Одночасно в лічильник завантажується нове значення, що задається схемою формування нахилу характеристики. Ця схема потрібна для компенсації параболічної залежності частот генераторів від температури. Лічильник знову починає працювати, і якщо він знову досягає нуля, коли інтервал ще не закінчений, процес повторюється знову. Схема формування нахилу завантажує в лічильник значення, які відповідають кількості імпульсів генератора на один градус для кожного конкретного значення температури. По закінченню процесу перетворення регістр температури буде містити значення температури. На рис. 4.1 показані основні вузли датчика.

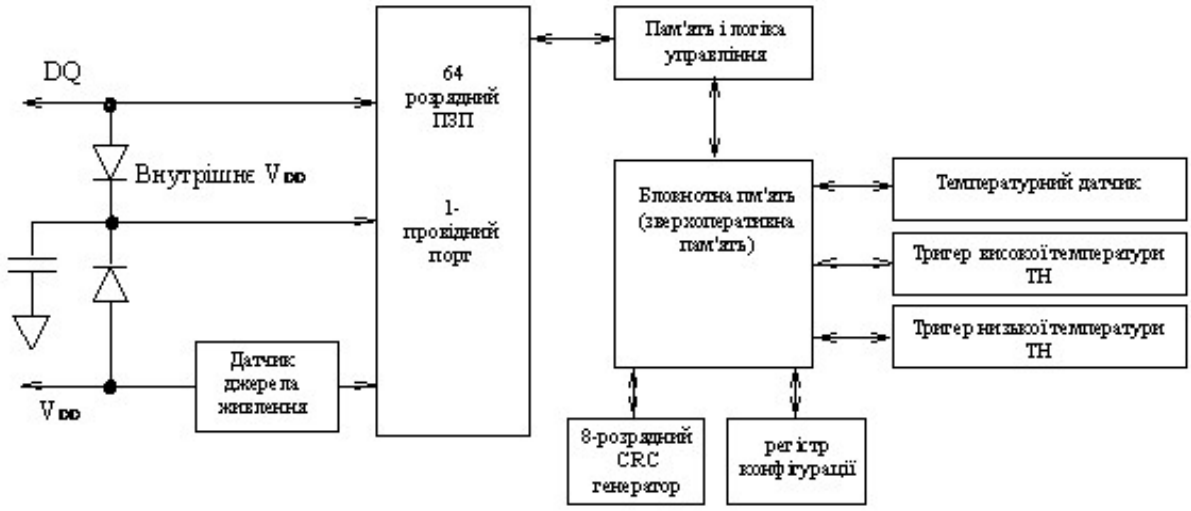


Рис. 4.1.

Кожен датчик містить унікальний 48-бітний серійний номер, записаний при виробництві, тому на одну шину можуть бути підключені кілька датчиків. Це дозволяє розмістити датчики в різних місцях і збирати інформацію по простому 2-х провідному кабелю. Крім серійного номера в ПЗП міститься код сімейства (для DS18B20 це 10h) і контрольна сума. Також DS18B20 має температурний датчик, регістр конфігурації, енергозалежні температурні сигнальні тригери TH та TL, проміжний ОЗП обсягом 8 байт (рис. 4.2).

Ï ðî ì ³æí å Ï ÇÏ

Ì î ëî äø èé áàéò òåì ï åðàòóðè	0
Ñòàðø èé áàéò òåì ï åðàòóðè	1
TH / áàéò êî ðèñòóâà÷à 1	2
TH / áàéò êî ðèñòóâà÷à 2	3
Çàðåçåðâî âàí î	4
Çàðåçåðâî âàí î	5
COUNT_REMEIN	6
COUNT_PER_C	7

Рис. 4.2.

Байти TH та TL представляють собою температурні пороги, з якими зрівнюються 8 біт кожного вимірюваного значення температури (молодший біт відкидається). За допомогою спеціальної команди можна організувати сигналізацію виходу температури за межі цих порогів. Якщо

така функція не потрібна, байти TH та TL можна використати для зберігання будь-яких даних користувача.

Зчитування значення виміряної температури, а також передача команди початку перетворення та інших команд, виконується за допомогою 1-провідного інтерфейсу (1-Wire). На основі цього інтерфейсу фірма DALLAS створила мережу, що називається MicroLAN. Для роботи в цій мережі випускається цілий ряд пристроїв, таких як адресовані ключі, АЦП, термометри, годинники реального часу, цифрові потенціометри.

Протокол, що використовується 1-провідним інтерфейсом, досить простий. У будь-який момент часу на 1-провідній шині можна виділити пристрій-майстер, яким може бути мікропроцесор або комп'ютер, і підлеглий пристрій. Якщо на шині присутні тільки майстер і всього один підлеглий пристрій, можна опустити все те, що пов'язане з адресацією пристроїв. В результаті потрібно знати лише протокол передачі даних, які можуть бути командами або даними.

На рис. 4.3 показана апаратна конфігурація інтерфейсної частини DS18B20.

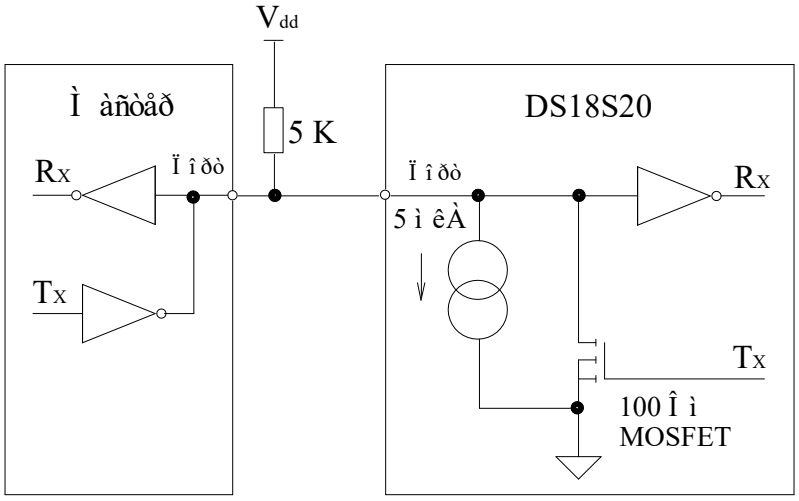


Рис. 4.3.

У кожного однопровідного пристрою до шини підключений вхід приймача й вихід передавача з відкритим стоком. Відкритий стік дозволяє підключати до шини безліч пристроїв, забезпечуючи «монтажне або». Генератор струму 5 мкА забезпечує на вході 1-провідного пристрою низький логічний рівень, коли шина не підключена. Оскільки лінія тактового сигналу відсутня, обмін є асинхронним. Це означає, що в процесі обміну потрібно досить точно витримувати необхідні часові співвідношення.

4.2.3. Протокол обміну даними по шині 1-Wire

Однопровідна шина оперує з TTL-рівнями, тобто логічна одиниця представлена рівнем напруги близько 5 В, а логічний нуль – напругою

поблизу 0 В. У вихідному стані на лінії присутній рівень логічної одиниці, що забезпечується резистором номіналом близько 5 кОм.

Ініціатором обміну по однопровідній шині завжди виступає майстер. Всі пересилання починаються із процесу ініціалізації. Ініціалізація виконується в наступній послідовності (рис. 4.4):

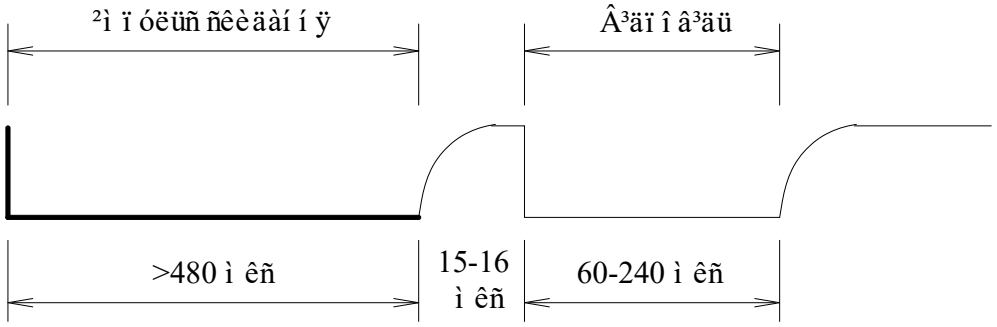


Рис. 4.4.

Майстер посилає імпульс скидання (reset pulse) – сигнал низького рівня тривалістю не менше 480 мкс. За імпульсом скидання відбувається відповідь підлеглого пристрою (presence pulse) – сигнал низького рівня тривалістю 60 – 240 мкс, що генерується через 15 – 60 мкс після завершення імпульсу скидання.

Відповідь підлеглого пристрою для головного пристрою означає, що на шині присутній термометр і він готовий до обміну даними. Після того, як майстер виявив відповідь, він може передати термометру одну з команд. Передача ведеться шляхом формування майстром спеціальних часових інтервалів (time slots). Кожен часовий інтервал служить для передачі одного біта. Першим передається молодший біт. Інтервал починається імпульсом низького рівня, тривалість якого лежить у межах 1 – 15 мкс. Оскільки перехід з одиниці в нуль менш чутливий до ємності шини (він формується відкритим транзистором, у той час як перехід з нуля в одиницю формується резистором), саме цей перехід використовують однопровідні пристрої для синхронізації з майстром. У підлеглому пристрої запускається схема часової затримки, що визначає момент зчитування даних. Номінальне значення затримки дорівнює 30 мкс, однак воно може коливатися в межах 15 – 60 мкс. За імпульсом низького рівня видається черговий біт для передачі. Він повинен утримуватися майстром на шині протягом 60 – 120 мкс від початку інтервалу. Часовий інтервал завершується переводом шини в стан високого рівня на час не менший 1 мкс. Слід відзначити, що обмеження на цей час не накладається. Аналогічним чином формуються часові інтервали для всіх інших біт (рис. 4.5):

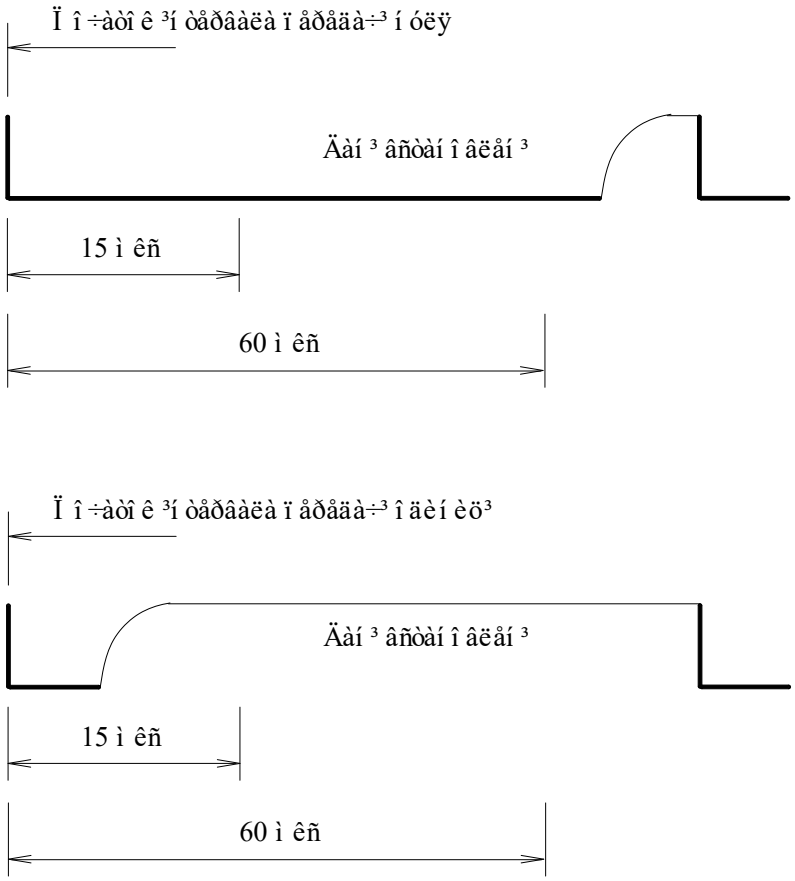


Рис. 4.5.

Першою командою, яку повинен передати майстер для DS18B20 після ініціалізації, є одна з команд функцій ПЗП. Усього DS18B20 має 5 команд функцій ПЗП:

- Read ROM [33h]. Ця команда дозволяє прочитати вміст ПЗП. У відповідь на цю команду DS18B20 передає 8-бітний код сімейства (10h), потім 48-бітний серійний номер, а потім 8-бітне значення CRC для перевірки правильності прийнятої інформації.
- Match ROM [55h]. Ця команда дозволяє адресувати на шині конкретний термометр. Після цієї команди майстер повинен передати потрібний 64-бітний код, і тільки той термометр, що має такий код, буде «відгукуватися» до наступного імпульсу скидання.
- Skip ROM [CCh]. Ця команда дозволяє пропустити процедуру порівняння серійного номера, і тим самим заощадити час у системах, де на шині є всього один пристрій.
- Search ROM [F0h]. Ця досить складна у використанні команда дозволяє визначити серійні номери всіх термометрів, що присутні на шині.
- Alarm Search [ECh]. Ця команда аналогічна попередній, але «відгукуватися» будуть тільки ті термометри, у яких результат

останнього виміру температури виходить за встановлені межі TH та TL.

Оскільки в нашому прикладі всього один пристрій, найбільш підходящою є функція Skip ROM. Крім неї ще може бути корисною функція Read ROM, що дозволяє ідентифікувати підключений на шину пристрій по його коду сімейства та серійному номеру.

При прийомі даних від підлеглого пристрою часові інтервали для прийнятих біт також формує майстер. Інтервал починається імпульсом низького рівня тривалістю 1 – 15 мкс. Потім майстер повинен звільнити шину, щоб дати можливість термометру вивести біт даних. По переходу з одиниці в нуль DS18B20 виводить на шину біт даних і запускає схему часової затримки, що визначає, як довго біт даних буде присутній на шині. Цей час лежить у межах 15 – 60 мкс. Для того щоб дані на шині, яка завжди має деяку ємність, гарантовано встановилися, потрібний деякий час. Тому момент зчитування даних майстром повинен відстояти якнайдалі, але не більше ніж на 15 мкс від початку часового інтервалу (рис. 4.6):

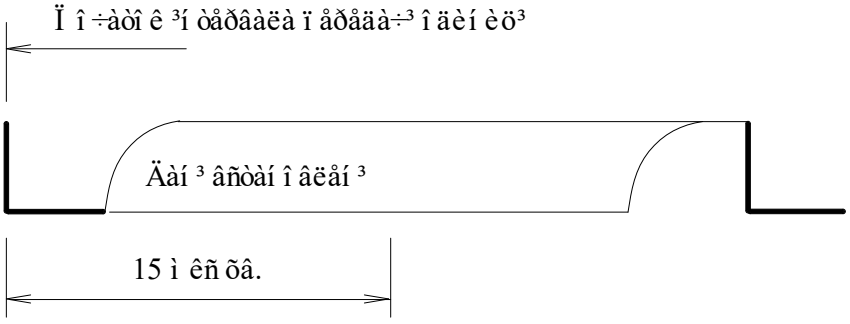


Рис. 4.6.

Прийом байта починається з молодшого біта. Спочатку передається байт коду сімейства. За кодом сімейства передається 6 байт серійного номера, починаючи з молодшого. Потім передається байт контрольної суми (CRC). В обчисленні байта контрольної суми беруть участь перші 7 байт, або 56 переданих біт. Після прийому даних майстер повинен обчислити контрольну суму та порівняти значення з CRC. Якщо ці значення збігаються – прийом даних пройшов без помилок.

Після обробки однієї з команд функцій ПЗП, DS18B20 здатний сприймати ще кілька команд:

- Write Scratchpad [4Eh]. Ця команда дозволяє записати дані в проміжний ОЗП DS18B20.
- Read Scratchpad [BEh]. Ця команда дозволяє зчитувати дані із проміжного ОЗП.
- Copy Scratchpad [48h]. Ця команда копіює байти TH і TL із проміжного ОЗП в енергонезалежну пам'ять. Ця операція вимагає близько 10 мс.

– Convert T [44h]. Ця команда запускає процес перетворення температури.
– Recall E2 [B8h]. Ця команда діє зворотним чином відносно команди Copy Scratchpad, тобто вона дозволяє зчитувати байти TH і TL з енергонезалежної пам'яті в проміжний ОЗП. При включенні живлення ця команда виконується автоматично.
– Read Power Supply [B4h]. Ця команда дозволяє перевірити, чи використає DS18B20 паразитне живлення. Справа в тому, що DS18B20 можна підключати всього за допомогою двох проводів. У цьому випадку для живлення використовується лінія даних.

При використанні DS18B20 тільки для виміру температури потрібні всього дві із цих команд: Convert T і Read Scratchpad. Послідовність дій при вимірі температури повинна бути наступною:

1. Посилаємо імпульс скидання і приймаємо відповідь термометра.
2. Посилаємо команду Skip ROM [CCh].
3. Посилаємо команду Convert T [44h].
4. Формуємо затримку мінімум 750 мс.
5. Посилаємо імпульс скидання і приймаємо відповідь термометра.
6. Посилаємо команду Skip ROM [CCh].
7. Посилаємо команду Read Scratchpad [BEh].
8. Читаємо дані із проміжного ОЗП (8 байт) і CRC.
9. Перевіряємо CRC, і якщо дані зчитані вірно, обчислюємо температуру.

4.2.4. Сполучення датчика DS18B20 з контролером Arduino

Схема підключення датчика температури DS18B20 до контролера Arduino по однопровідному інтерфейсу 1-Wire представлена на рис. 4.7.

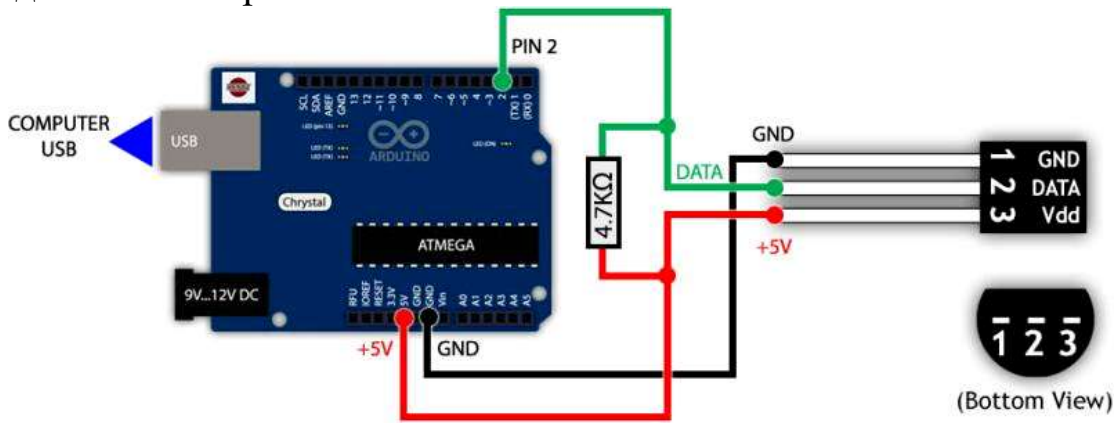


Рис. 4.7.

Для моделювання роботи пристрою в Proteus потрібно скласти схему, як показано на рис. 4.8. За допомогою Віртуального терміналу в

режимі реального часу можна спостерігати значення температури, яке зчитується контролером Arduino з датчика.

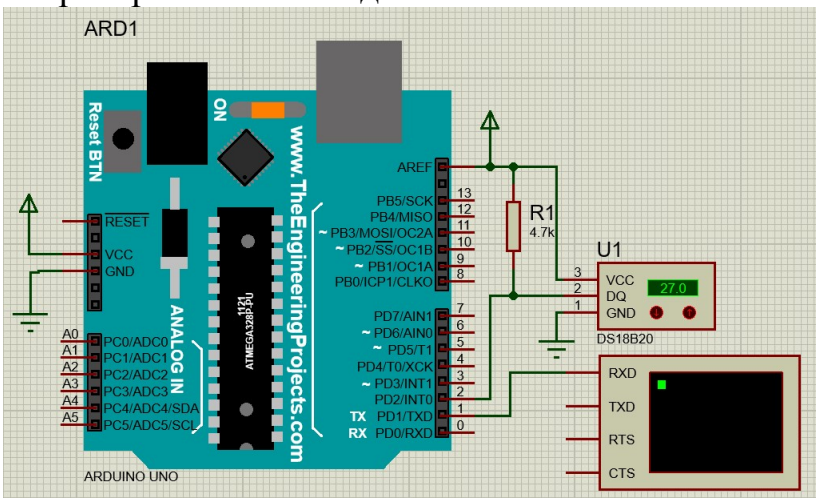


Рис. 4.8.

Текст програми, яка виконує зчитування температури з датчика та передавання отриманих даних в текстовому вигляді за допомогою USART в комп’ютер (в моделі – у вікно терміналу), представлений нижче.

```
// підключення бібліотеки функцій для роботи з однопровідним інтерфейсом
#include <OneWire.h>
// створення об’єкта для роботи з датчиком, який відключений до виводу 2 контролера
OneWire ds(2);

void setup()                    // функція ініціалізації контролера
{
   Serial.begin(9600);          // налаштування швидкості передавання даних 9600 біт/с
}

void loop()                     // блок коду в тілі цієї функції виконується циклічно
{
   byte i;                      // оголошення додаткової змінної
   byte data[12];               // масив з 12 байт для зчитування даних
   byte addr[8];                // масив з 12 байт для зберігання адреси датчика
   float celsius;               // змінна для збереження виміряного значення температури

   // пошук датчика та визначення його адреси
   if ( !ds.search(addr)) {     // якщо виклик функції пошуку датчика завершився невдало
      ds.reset_search();
      delay(250);               // часова затримка 250 мс
      return;                   // вихід з функції
   }
   ds.reset();                  // переведення об’єкту в початковий стан
   ds.select(addr);             // вибір датчика з визначеною адресою
   ds.write(0x44, 1);           // команда початку вимірювання температури
   delay(1000);                 // часова затримка 1 с
   ds.reset();                  // переведення об’єкту в початковий стан
   ds.select(addr);             // вибір датчика з вказаною адресою
   ds.write(0xBE);              // команда зчитування виміряної температури
```

```
  // зчитуємо показники температури з внутрішньої пам’яті датчика
  for ( i = 0; i < 9; i++) {
     data[i] = ds.read();
  }
  int16_t raw = (data[1] << 8) | data[0];
  // датчик може бути налаштований на різну точність, тому потрібно її з’ясувати
  byte cfg = (data[4] & 0x60);
  if (cfg == 0x00) raw = raw & ~7;             // точність 9-розрядів, 93,75 мс
  else if (cfg == 0x20) raw = raw & ~3;      // точність 10-розрядів, 187,5 мс
  else if (cfg == 0x40) raw = raw & ~1;      // точність 11-розрядів, 375 мс
  // перетворення показань датчика в градуси Цельсія
  celsius = (float)raw / 16.0;
  Serial.print("t=");
  Serial.println(celsius);       // передавання даних по USART в комп’ютер
}
```

Після компіляції проекту та запуску процесу моделювання схеми в Proteus, у вікні віртуального терміналу можна побачити значення виміряної температури. Якщо підключити реальний пристрій до комп’ютера, тоді за допомогою Монітора порта, вікно якого відкривається натисканням комбінацію клавиш Shift+Ctrl+M, на екрані комп’ютера у текстовому вигляді можна бачити значення температури, яке буде оновлюватись кожну секунду.

4.2.5. Мікропроцесорна система контролю температури на базі датчика температури DS18B20

На рис. 4.9 представлена схема підключення датчика температури DS18B20 до мікроконтролера AT90S8535. Мікроконтролер підключається до комп'ютера по інтерфейсу RS232 і передає значення температури. Для узгодження рівнів напруг COM-порту комп'ютера і мікроконтролера використана мікросхема-перетворювач рівнів MAX232.

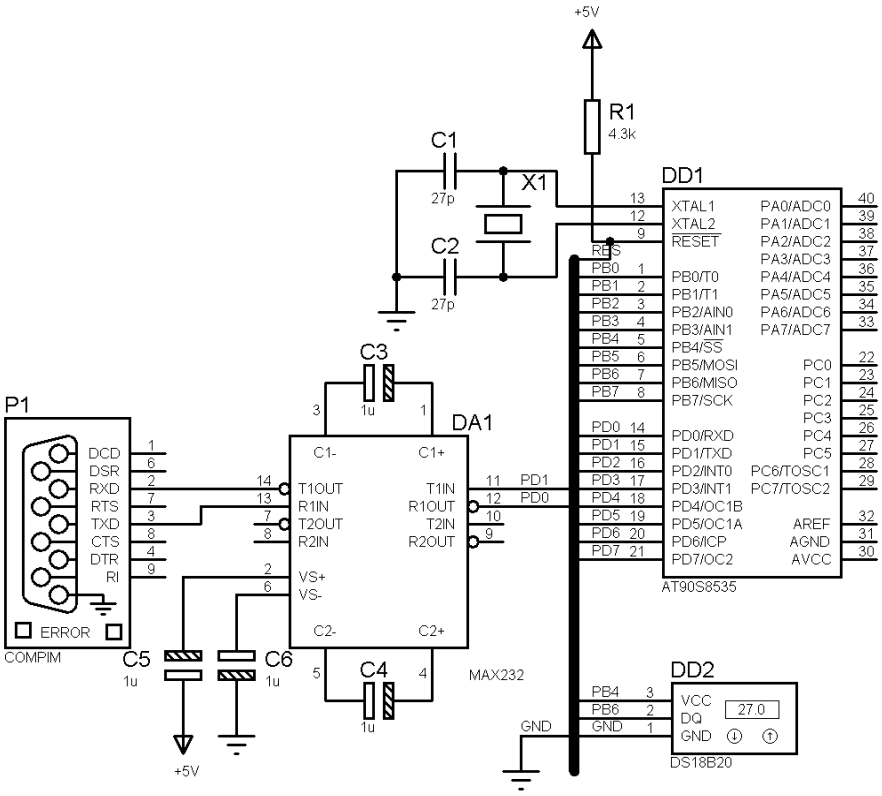


Рис. 4.9.

Для написання програми для мікроконтролера AT90S8535, що виконує зчитування температури з датчика DS18B21 по інтерфейсу 1-Wire і передає значення температури по COM-порту потрібно виконати наступні дії:

1. Запускаємо середовище розробки CodeVision і створюємо новий проект, вибравши з меню File команду New. У вікні, що з'явилося (рис. 4.10) відзначити позицію Project.

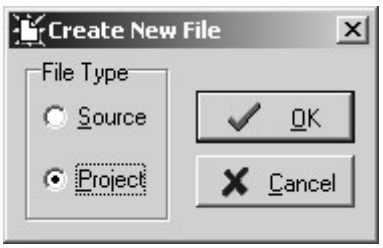


Рис. 4.10.

У вікні, що з'явилося, із пропозицією використання майстра проекту (CodeWizard), натиснути на кнопку «NO». Далі потрібно ввести назву проекту (наприклад TestWire).

2. У вікні налаштувань проекту встановити значення, як показано на рис. 4.11.

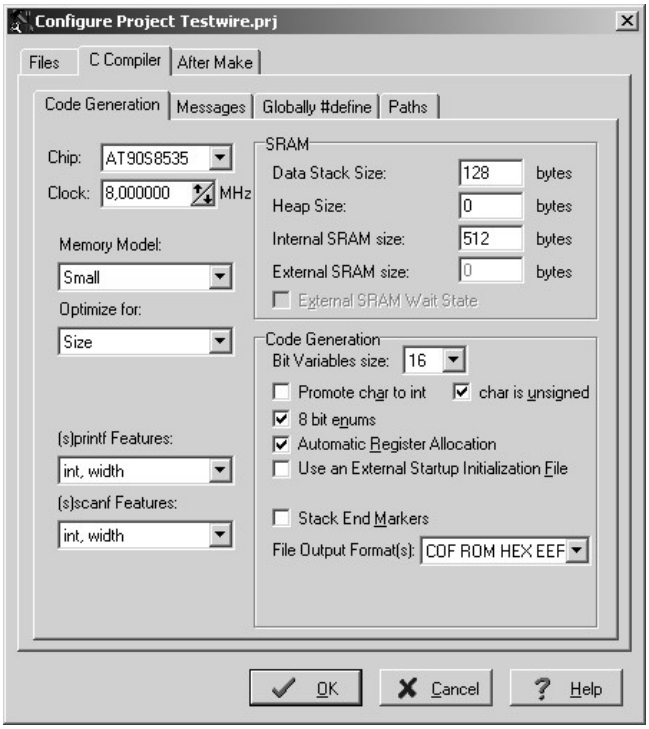


Рис. 4.11.

3. Додаємо в проект файл із вихідним кодом. Для цього в меню File вибрати команду New. Програма для мікроконтролера AT90S8535, створена в компіляторі CodeVision, представлена нижче.

```
#asm                                    // вказівка виводів, до яких підключається датчик
   .equ __w1_port=0x18
   .equ __w1_bit=6                      // лінія даних на виводі 6 порту B
#endasm
#include "1wire.h"                      // підключення заголовних файлів
#include "90s8535.h"
#include "delay.h"
#include "math.h"
unsigned char delay;                    // затримка опитування датчика
bit start_stop;                         // якщо біт =1, то почати перетворення
// оголошення прототипів функцій
void main(void);                        // основна функція програми
void init(void);                        // ініціалізація мікроконтролера
void temp(void);                        // опитування датчика температури
// змінює байт конфігурації датчика
void set_config(unsigned char config);
// підпрограма обробки переривання
// викликається по прийому символу з UART
interrupt [UART_RXC] void uart_rxc(void)
{
        unsigned char temp;             // оголошення додаткової змінної
        temp = UDR;                     // зчитати прийнятий байт
        // якщо прийнято команду 0xBC, почати перетворення
        if(temp==0xBC)        start_stop = 1; }
// функція опитування датчика температури
```

```
void temp()
{
        unsigned char buf[9];           // буфер зчитаних даних
        unsigned char i;                // додаткові змінні
        unsigned char k;
        if(w1_init())                   // виклик функції ініціалізації датчика
        {
                w1_write(0xCC);         // пропуск ПЗП
                w1_write(0x44);         // перетворення температури
                // очікування кінця перетворення температури
                delay_ms(delay);
                // цикл зчитування блокнотної пам'яті датчика
                for (k=0; k<3; k++)
      {
                w1_init();              // ініціалізація
                w1_write(0xCC);         // пропуск ПЗП
                w1_write(0xBE);         // зчитування блокнотної пам'яті
                for (i=0; i<9; i++)     // зчитати дев'ять байт
                {
                buf[i] = w1_read();     // запис зчитаного байта в буфер
                }
                // якщо помилка, то повторити цикл
                if(buf[8]==w1_dow_crc8(&buf[0], 8)) break;
                }
                // вийти, якщо датчик не відповідає
                if(buf[8]!=w1_dow_crc8(&buf[0], 8)) return;
                else                    // передача двох байт температури в комп'ютер
                {
                while(!USR.5);          // якщо біт UDRE == 1
                        UDR = buf[0]; // передати байт
                while(!USR.5);
                        UDR = buf[1];
        }
        }
        start_stop=0;                   // скидання прапора дозволу перетворення
        return;
}

// функція установки байта конфігурації
void set_config(unsigned char config)
{
        while(!w1_init());              // виклик функції ініціалізації датчика
                                        // повторювати поки датчик не відповість
        switch(config)
        {
                case 0:                 // 9 біт
                        config = 0b00011111;
                        delay = 100;
                        break;
                case 1:                 // 10 біт
                        config = 0b00111111;
                        delay = 188;
```

```
                break;
            case 2:             // 11 біт
                config = 0b01011111;
                delay = 375;
                break;
            case 3:             // 12 біт
                config = 0b01111111;
                delay = 750;
                break;
            default:            // якщо помилка
                return;         // вихід з функції
        }
        // перепрошивання байта конфігурації датчика 9-12 розрядів
        w1_write(0xCC);         // пропуск ПЗП
        w1_write(0x4E);         // запис у блокнотну пам'ять

        w1_write(0x00);         // запис TH
        w1_write(0x00);         // запис TL
        w1_write(config);       // запис байта конфігурації
}

// функція ініціалізації мікроконтролера
void init_cpu(void)
{
        DDRB = 0b00010100;      // підключення DS18B20:6 - DQ, 4 - UCC
        PORTB = 0xFF;           // вивести одиницю на Port.4 - живлення
        // ініціалізація UART
        DDRD = 0b10000010;      // налаштування виводів UART
        UCR  = 0b10011000;      // дозвіл прийому/передачі
        UBRR = 51;              // швидкість обміну 9600 біт/с при 8 МГц
        // конфігурація за замовчуванням
        delay = 100;            // затримка
        start_stop = 0;         // прапор дозволу перетворення
        #asm("sei")             // дозволити переривання
}

void main(void)                 // головна функція
{
        init_cpu();             // виклик функції ініціалізації мікроконтролера
        set_config(0);          // установка 9-бітного режиму перетворення T
        while(1)                // вічний цикл
        {
            if(start_stop!=0) temp();       // якщо біт дозволу
            // перетворення  температури встановлений,
            // викликати функцію перетворення температури
        }
}
```

Зберігши файл на диск із назвою TestWire.c, проект можна компілювати. Для цього в меню Project вибрати команду Compile, або натиснути клавішу F9. У каталозі, де розташовані файли проекту, буде створений файл для програмування мікроконтролера TestWire.hex.

4.3. Послідовна асиметрична шина зв’язку I^2C

Шина I^2C широко використовується в побутовій електроніці, передачі даних і промисловій електроніці. Розроблена фірмою Philips проста двонапрямлена 2-провідна шина спочатку для ефективного керування та взаємодії різних блоків телевізорів, стала застосовуватися для зв'язку між собою однокристальних мікроконтролерів, РКІ-індикаторів, мікросхем пам'яті, аналого-цифрових і цифро-аналогових перетворювачів, мікросхем реального часу та ін.

4.3.1. Загальна характеристика інтерфейсу I^2C

Для здійснення процесу обміну інформацією по шині I^2C використовуються всього два сигнали – лінія даних SDA та лінія синхронізації SCL. Для забезпечення реалізації двонапрямленості шини без застосування складних арбітрів шини вихідні каскади пристроїв, підключених до шини, мають відкритий стік або відкритий колектор для забезпечення функції монтажного "І". Як показано на рис. 4.12, обидві лінії шини підключені до живлення через резистори.

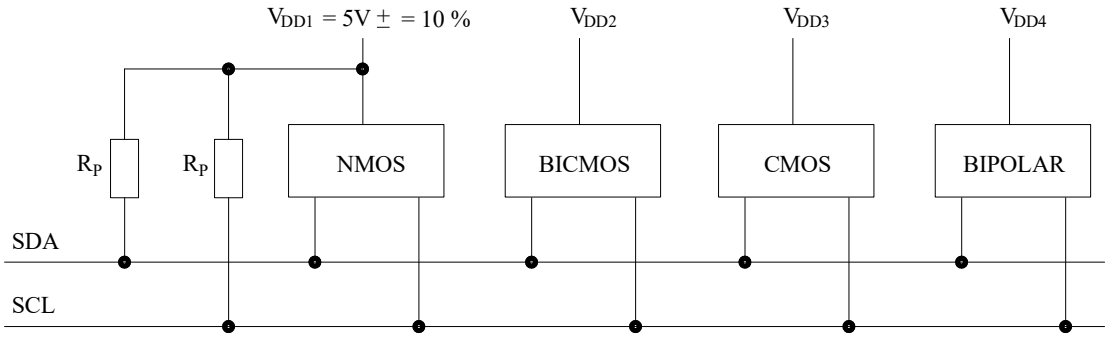


Рис. 4.12.

Проста двохпровідна послідовна шина I^2C мінімізує кількість з'єднань між інтегральними схемами. Як результат – друковані плати стають більш простими та технологічними при виготовленні. Інтегрований I2C-протокол усуває необхідність у дешифраторах адреси та іншій зовнішній логіці узгодження. Максимальна припустима кількість мікросхем, приєднаних до однієї шини, обмежується максимальною ємністю шини 400 пФ.

Шина I^2C має наступні переваги:

– Тільки дві лінії – послідовна лінія даних (SDA) і послідовна лінія синхронізації (SCL).

– Кожен елемент, з'єднаний із шиною, є програмно адресованим своєю унікальною адресою. При цьому взаємодія між ними може бути побудовано по простому принципу master/slave.

– Послідовна 8-розрядна передача даних може проводитися зі швидкістю від 0 до 100 кБіт/с у стандартному режимі та 400 кБіт/с у швидкому режимі. При розробці власних пристроїв можливе застосування і більш високих швидкостей.

– Низьке енергоспоживання, висока завадостійкість, широкий діапазон напруг живлення.

Всі абоненти шини діляться на два класи – «Master» та «Slave». Пристрій «Master» генерує тактовий сигнал (SCL), і тому є ведучим. Він може самостійно виходити на шину й адресуватися до будь-якого «Slave» – пристрою з метою передачі або прийому інформації. Всі «Slave» – пристрої «слухають» шину на предмет виявлення власної адреси, і, розпізнавши її, виконують вказані дії. Крім того, можливий режим «Multi-master» – режим, коли на шині встановлено кілька ведучих («Master») пристроїв, які або спільно розділяють загальні «Slave» – пристрої, або почергово є ведучими пристроями, коли самі ініціюють обмін інформацією. Режим «Multi-master» вимагає арбітражу та розпізнавання конфліктів. Він складніше реалізується і тому рідше використовується в пристроях.

4.3.2. Діаграми передачі даних

В початковий момент часу (у режимі очікування) лінії SDA і SCL перебувають у стані логічної одиниці. У режимі передачі (рис. 4.13) біт даних SDA синхронізується позитивним імпульсом SCL. Зміна інформації на лінії SDA виконується при нульовому стані лінії SCL. «Slave» – пристрій може утримувати лінію SCL у нульовому стані на час обробки чергового прийнятого байта, при цьому «Master» – пристрій повинен дочекатися звільнення лінії SCL, перш ніж продовжити передачу інформації.

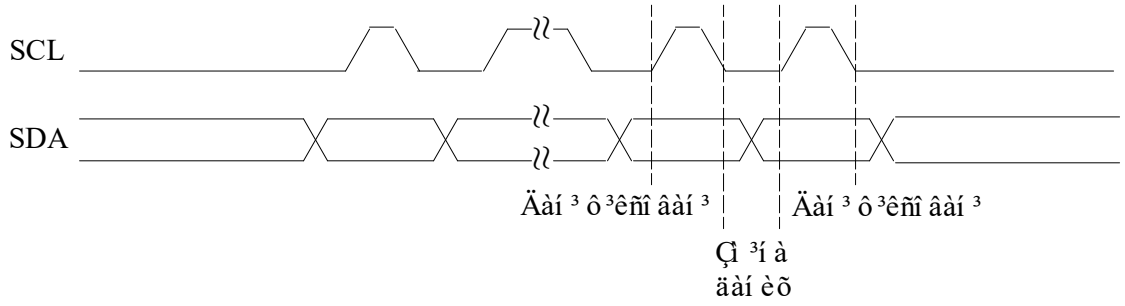


Рис. 4.13.

Для синхронізації пакетів шини I2C розрізняють дві умови – «Start» та «Stop», які обмежують початок і кінець інформаційного пакета (рис. 4.14). Для кодування цих умов використовується зміна стану лінії SDA при одиничному стані лінії SCL, що неприпустимо при передачі даних. «Start» – умова утворюється при зрізі сигналу SDA, коли лінія SCL перебуває в одиничному стані. «Stop» – умова утвориться фронтом сигналу на лінії SDA при одиничному стані лінії SCL.

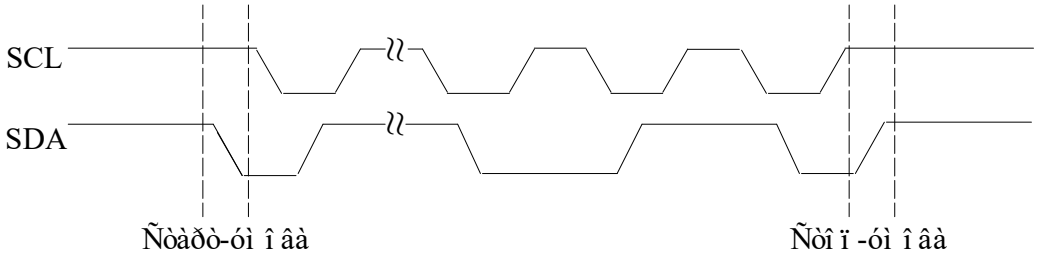


Рис. 4.14.

Стани «Старт» та «Стоп» завжди виробляються ведучим. Вважається, що шина зайнята після появи стану «Старт». Шина вважається звільненою через деякий час після появи умови «Стоп».

Передача даних починається при першому позитивному імпульсі на лінії SCL, яким синхронізується старший біт першого інформаційного байта. Кожен інформаційний байт містить 9 тактових періодів лінії SCL. У дев'ятому такті пристрій-одержувач видає підтвердження (#ACK) – негативний імпульс, що свідчить про прийом даних (рис. 4.15). Відсутність підтвердження інтерпретується як помилка.

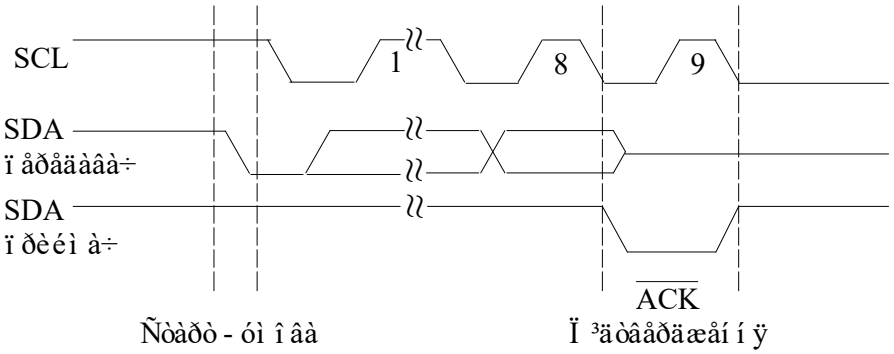


Рис. 4.15.

При прийомі останнього байта в серії, замість сигналу АСК майстер може виставити стан «Стоп», при цьому підлеглий передавач повинен звільнити лінію.

Кожен пристрій, що підключений до шини, може бути програмно адресований по унікальній адресі. Для вибору приймача повідомлення ведучий використовує унікальний адресний компонент у форматі посилки. При використанні однотипних пристроїв, інтегральні схеми часто мають додатковий селектор адреси, який може бути реалізований як у вигляді додаткових цифрових входів селектора адреси, так і у вигляді аналогового входу. При цьому адреси таких однотипних пристроїв виявляються рознесені в адресному просторі пристроїв, підключених до шини. У звичайному режимі використовується 7-бітна адресація. Всі інтегральні мікросхеми, що підтримують роботу в стандарті шини I2C, мають набір фіксованих адрес, перелік яких зазначений виробником в описах контролерів.

Щоб почати операцію обміну пристрій «Master» видає на шину «Start» – умову, за яким передається байт із адресою веденого пристрою, що складається з 7-бітної адреси пристрою та одного біта – прапора операції «R/W», який визначає напрямок обміну даними, причому 0 означає передачу від ведучого до веденого, а 1 – зчитування читання з веденого пристрою (рис. 4.16). Всі біти по шині I2C передаються в порядку старший – молодший, тобто першим передається 7-й біт, останнім 0-й біт. Після того, як адреса передана, кожен пристрій у системі порівнює перші сім бітів після сигналу «Старт» зі своєю адресою. При збігу пристрій вважається вибраним як приймач або передавач, залежно від біта напрямку. Адреса веденого може складатися з фіксованої та програмованої частини.

За адресою можуть передаватися один або декілька інформаційних байтів (у напрямку, встановленому прапором «R/W»), біти яких синхронізуються сигналом SCL з «Master» – пристроєм.

При виконанні операції зчитування «Master» – абонент повинен супроводжувати прочитаний байт сигналом #ACK якщо необхідно прочитати наступний байт і не видавати сигнал #ACK якщо потрібно закінчити зчитування пакета (рис. 4.17, рис. 4.18). Логічна реалізація протоколів на шині I2C може бути довільною для кожної конкретної інтегральної схеми.

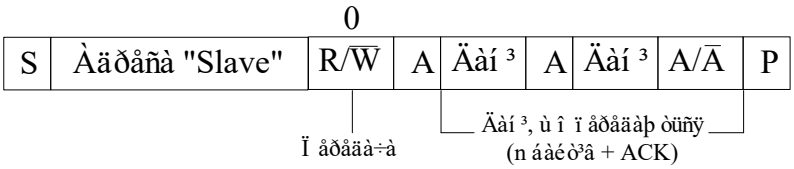


Рис. 4.16.

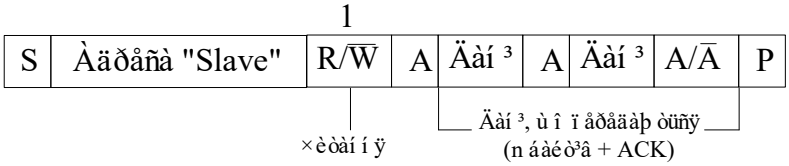


Рис. 4.17.

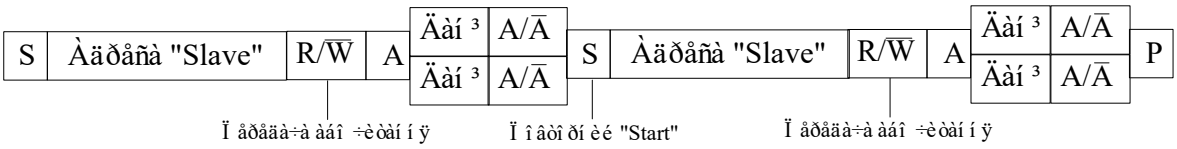


Рис. 4.18.

На представлених діаграмах «S» – умова «Start», «A» – біт підтвердження прийому #ACK, «P» – умова «Stop».

4.3.3. Сполучення датчика DS1621 з контролером Arduino

Схема підключення датчика температури DS1621 до контролера Arduino по інтерфейсу I^2C представлена на рис. 4.19.

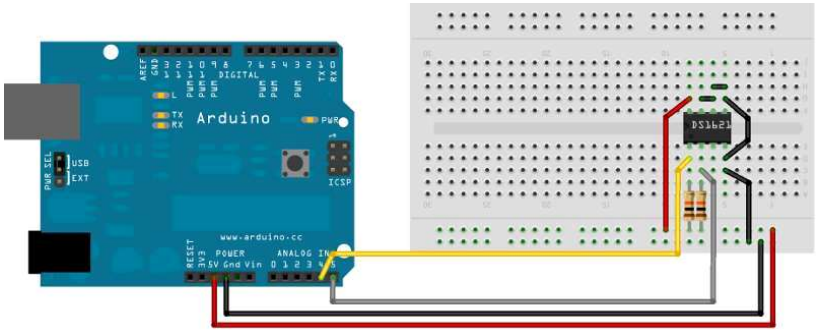


Рис. 4.19.

Для моделювання роботи пристрою в Proteus потрібно скласти схему, як показано на рис. 4.20. За допомогою Віртуального терміналу в режимі реального часу можна спостерігати значення температури, яке зчитується контролером Arduino з датчика. Також в схемі присутній аналізатор протоколу I^2C, за допомогою якого можна спостерігати та аналізувати команди та дані, що передаються між контролером та датчиком.

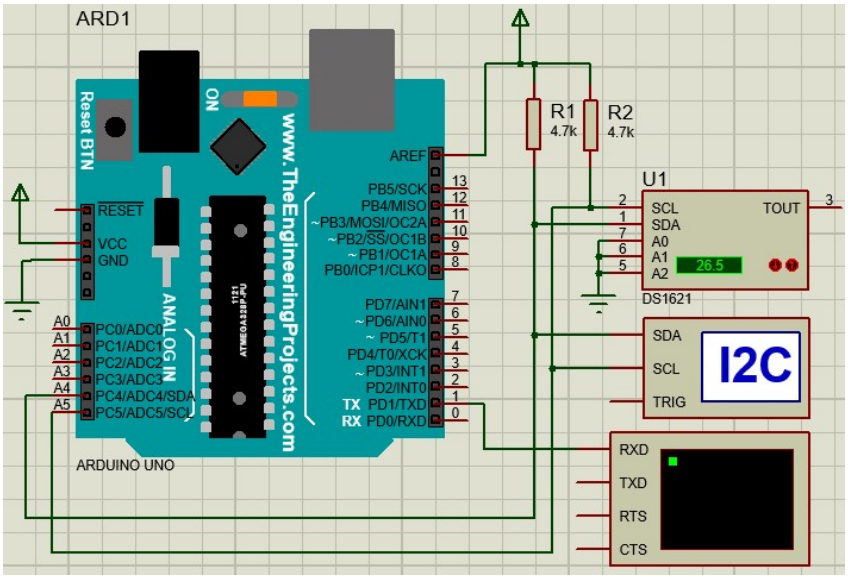


Рис. 4.20.

Текст програми, яка виконує зчитування температури з датчика та передавання отриманих даних в текстовому вигляді за допомогою USART в комп’ютер (в моделі – у вікно терміналу), представлений нижче.

```
// підключення бібліотеки функцій для роботи з інтерфейсом  I2C
#include <Wire.h>
// зсув на 1 розряд праворуч необхідний для роботи з бібліотекою wire.h
#define DEV_ID 0x90 >> 1
void setup()    // функція ініціалізації контролера
{
   Serial.begin(9600);// налаштування швидкості передавання даних 9600 біт/с
   Wire.begin();          // формування умови «старт»
   Wire.beginTransmission(DEV_ID); // підключення до DS1621
   Wire.write(0xAC);             // доступ до регістра конфігурації
```

```
  Wire.write(0x02);            // налаштування режиму безперервного вимірювання
  Wire.beginTransmission(DEV_ID);  // «повторний старт»
  Wire.write(0xEE);            // початок перетворення
  Wire.endTransmission();   // формування умови 2стоп»
}

void loop()     // основна програма в тілі цієї функції виконується циклічно
{
  int8_t firstByte;     // оголошення змінних для збереження зчитаних даних
  int8_t secondByte;
  float temp = 0; // змінна для збереження розрахованого значення температури
  delay(1000);          // часова затримка 1 секунда для завершення вимірювання T
  Wire.beginTransmission(DEV_ID);        // початок обміну даними
  Wire.write(0xAA);  // команда зчитування температури з датчика
  Wire.endTransmission(); // завершення обміну даними
  // запит 2 байт з DS1621 (точність вимірювання 0,5 град.)
  Wire.requestFrom(DEV_ID, 2);
  firstByte = Wire.read();              // зчитати перший байт
  secondByte = Wire.read();             // зчитати другий байт

  temp = firstByte;
  if (secondByte)           // якщо різниця 0,5 град.
    temp += 0.5;
  Serial.println(temp); // передавання даних про температуру в комп’ютер
}
```

Після компіляції проекту та запуску процесу моделювання схеми в Proteus, у вікні віртуального терміналу можна побачити значення виміряної температури. Якщо підключити реальний пристрій до комп’ютера, тоді за допомогою Монітора порта, вікно якого відкривається натисканням комбінацію клавіш Shift+Ctrl+M, на екрані комп’ютера у текстовому вигляді можна бачити значення температури, яке буде оновлюватись кожну секунду.

4.3.4. Мікропроцесорна система контролю температури на базі датчика температури DS1621

На рис. 4.21 представлена схема мікропроцесорної системи контролю температури. У схемі використовується датчик DS1621, з'єднаний з мікроконтролером AT90S8535 по шині I2C.

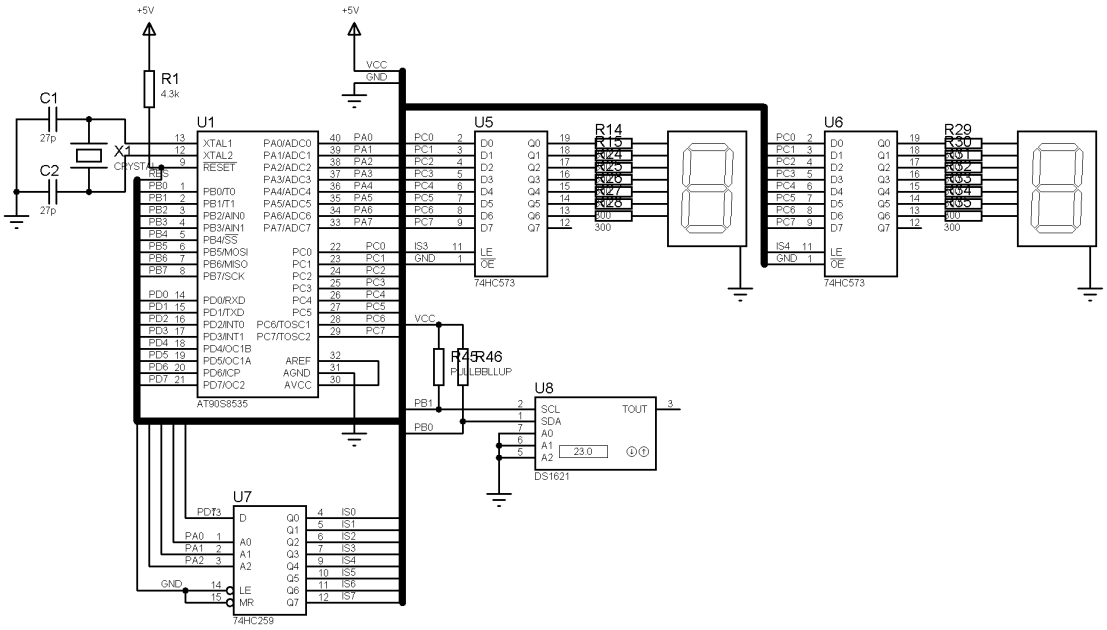


Рис. 4.21.

Мікросхема DS1621 – це цифровий термометр і термостат. Прилад є закінченим конвертером класу «температура – цифра» і не вимагає при використанні ніяких зовнішніх компонентів. Негативні значення температури представлені в коді з доповненням до 2-х. Прилад складається з п'яти основних компонентів:

– прецизійний датчик температури;

– аналого-цифровий перетворювач;

– електроніка двохпровідного інтерфейсу;

– регістр даних;

– компаратор термостата.

Ведучий шини може періодично зчитувати значення з температурного регістра, що містить результат попереднього перетворення.

Перший байт, переданий ведучим пристроєм після умови «Старт», містить адресу пристрою. Для DS1621 це значення 1001. Далі передаються 3 біти вибору пристрою A2, A1, A0. Якщо використовується тільки один пристрій на шині I2C, то значення цих біт може бути 0 за умови, що однойменні виводи мікросхеми заземлені. Останній біт – біт зчитування/запису – визначає напрямок передачі даних. Після цього команда, що визначає подальші дії по організації обміну даними. Набір команд для DS121 приводиться в табл. 4.1.

Таблиця 4.1.

Команда	Опис	Код команди
Зчитування температури	Зчитує останнє значення температури з регістра температури	0xAA
Зчитування лічильника	Зчитує значення відліку, що залишається в лічильнику	0xA8

Зчитування TH	Зчитування збереженого значення межи високої температури з регістра TH, R/W=0	0xA1
Зчитування TL	Зчитування збереженого значення межи низької температури з регістра TL, R/W=0	0xA2
Запис TH	Запис значення межи високої , температури в регістр TH, R/W=1	0xA1
Запис TL	Запис значення межи низької температури в регістр TL, R/W=1	0xA2
Запис конфігурації	Запис даних конфігурації в регістр конфігурації, R/W=1	0xAC
Зчитування конфігурації	Зчитування даних з регістра конфігурації, R/W=0	0xAC
Почати перетворення	Початок перетворення температури, R/W=0	0xEE
Зупинити перетворення	Зупинка перетворення температури в безперервному режимі, R/W=0	0x22

Стан датчика DS1621, а також його режим роботи, визначається за допомогою регістра стану й керування:

DONE	THF	TLF	NVB	1	0	POL	1SHOT

DONE – біт завершення перетворення температури: «1» – перетворення закінчене, «0» – перетворення виконується.
THF – цей біт встановлюється якщо температура перевищує значення, встановлене в регістрі TH. Біт можна скинути шляхом запису 0 у цю позицію, або відключенням живлення датчика.
TLF – цей біт встановлюється якщо температура менше значення, встановленого в регістрі TL. Біт можна скинути шляхом запису 0 у цю позицію, або відключенням живлення датчика.
NVB – прапор зайнятості пам'яті: «1» – виконується запис у комірку пам'яті.
POL – біт полярності вихідного сигналу TOUT: «1» – активний високий рівень, «0» – активний низький рівень.
1SHOT – якщо значення біта «1», перетворення температури виконується 1 раз. Якщо значення біта «0» – перетворення температури виконується безупинно.

У табл. 4.2 представлена відповідність значення температури цифровому коду.

Таблиця 4.2.

Температура, ^{0}C	Код (BIN)	Код (HEX)
+ 125	01111101 00000000	7B00h
+ 25	00011001 00000000	1900h
+ 0,5	00000001 00000000	0080h
0	00000000 00000000	0000h

– 0,5	11111111 10000000	FF80h
– 25	11100111 00000000	E700h
– 55	11001001 00000000	C900h

В наступному прикладі розглянуто програму для мікроконтролера AT90S8535, що виконує зчитування температури з датчика DS1621 по інтерфейсу I2C і виводить значення температури на 2 семисегментних індикатори. Текст програми для мікроконтролера, створеної з використанням компілятора ImageCraft, представлений нижче.

```
#include <io8535v.h>                          // підключення заголовних файлів
 #include <macros.h>
 #define SETSDA() PORTB|=0b00000001           // уст. сигналу SDA
 #define SETSCL() PORTB|=0b00000010           // уст. сигналу SCL
 #define RSTSDA() PORTB&=~0b00000001          // скидання сигналу SDA
 #define RSTSCL() PORTB&=~0b00000010          // скидання сигналу SCL

// функція ініціалізації контролера
 void init_cpu()
 {
        DDRA=0x07;                  // 3 молодших біти порту A налаштовані на вихід
        DDRC=0xFF;                  // порт C налаштований на вихід
        DDRD=0x80;                  // налаштування виводу 7 порту D на вихід
        PORTD=0x80;                 // установка 1 на виводі 7 порту D
 }

 // функція затримки; параметр – затримка в мс
 void delay_ms(int ms)
 {
        int c1, c2;                 // оголошення додаткових змінних
        for(c1=0;c1<ms;c1++)        // у циклі виконується «порожня»
        for(c2=0;c2<970;c2++) NOP();       // операція
 }

// функція затримки; параметр - кількість тактів затримки
void delay_us(int st)
 {
        int c1;                     // оголошення додаткової змінної
        for(c1=0;c1<st;c1++) NOP();// у циклі виконується «порожня» операція
}

// функція перевірки «зайнятості» шини
 char i2c_isbusy()
  {
        char st;                // додаткова змінна
        DDRB=0x00;                  // налаштування порту B на вхід
        PORTB=0x00;                 // відключення резисторів, що підтягують
     if((PINB&0x03)!=0x03) st=1;    // перевірка стану ліній
        else st=0;                  // функція повертає 0 якщо шина вільна
        return st;                  // інакше повертається значення 1
  }
```

```
// функція формування сигналу початку передачі даних
void i2c_start()
 {
        DDRB=0x03;              // налаштування 2-х біт порту B на вихід
        SETSDA();               // установка сигналу SDA
        SETSCL();               // установка сигналу SCL
        delay_us(2);            // затримка
        RSTSDA();               // скидання сигналу SDA
        delay_us(2);            // затримка
        RSTSCL();               // скидання сигналу SCL
        delay_us(2);            // затримка
 }

// функція формування сигналу завершення обміну даними
void i2c_stop()
 {
        DDRB=0x03;              // налаштування 2-х біт порту B на вихід
        RSTSDA();               // скидання сигналу SDA
        SETSCL();               // установка сигналу SCL
        delay_us(2);            // затримка
        SETSDA();               // установка сигналу SDA
        delay_us(2);            // затримка
 }

// функція посилки 1 байта; параметр - переданий байт
unsigned char i2c_sendbyte(char d)
 {
  unsigned char i=8,acknowledge=0; // додаткові змінні
  do {                          // у циклі виконується посилка 8 біт
   if((d & 0x80)!=0) SETSDA();  // установка сигналу SDA якщо
                                // старший біт дорівнює 1
        else RSTSDA();          // інакше скидання сигналу SDA
   d =d << 1;                   // зсув переданого байта на 1 розряд ліворуч
   delay_us(2);                 // формування затримки
                                // формування тактового імпульсу
        SETSCL();               // установка сигналу SCL
        delay_us(2);            // затримка
        RSTSCL();               // скидання сигналу SCL
        delay_us(2);            // затримка
  }
  while (--i);                  // цикл виконується 8 разів
  // формування тактового імпульсу для біта підтвердження
  DDRB=0x02;                    // переведення лінії SDA у третій стан
  RSTSDA();                     // скидання сигналу SDA
  SETSCL();                     // установка сигналу SCL
  delay_us(2);                  // затримка
  if ((PINB&0x01)==0) acknowledge=1; // якщо на лінії SCL низький
  // рівень, біт підтвердження прийнятий
  RSTSCL();                     // скидання сигналу SCL
```

```
 delay_us(2);                    // затримка
 DDRB=0x03;                      // налаштування 2-х біт порту B на вихід
 return acknowledge;             // повертається біт підтвердження
}

// функція зчитування одного байта
unsigned char i2c_readbyte(unsigned char acknowledge)
 {
 unsigned char i=8,i2c_data=0;   // додаткові змінні
 char d;
 DDRB=0x02;                      // переведення лінії SDA у третій стан
 do {
  i2c_data<<=1;                  // зсув на 1 розряд ліворуч
  SETSCL();                      // установка сигналу SCL
       delay_us(2);              // затримка
       d=PINB;                   // зчитування стану порту B
       d=d&0x01;                 // виділення 0 біта
  i2c_data|=d;                   // копіювання 0 біта в змінну
  RSTSCL();                      // скидання сигналу SCL
       delay_us(2);              // затримка
 }
 while (--i);                    // цикл виконується 8 разів
 if (acknowledge)                // якщо необхідний біт підтвердження
       {
       DDRB=0x03;                // налаштування 2-х біт на вихід
       RSTSDA();                 // скидання сигналу SDA
       }
                                 // формування тактового імпульсу
  SETSCL();                      // установка сигналу SCL
  delay_us(2);                   // затримка
  RSTSCL();                      // скидання сигналу SCL
  DDRB=0x02;                     // переведення лінії SDA у третій стан
  return i2c_data;               // функція повертає прийнятий байт
}

 // головна функція
 int main ()
 {
 int c1=0,c2=0;                  // оголошення додаткових змінних
 int t1, t2;
 char seg[10]={0b00111111, 0b00000110, 0b01011011, 0b01001111,
 0b01100110, 0b01101101, 0b01111101, 0b00000111, 0b01111111,
 0b01101111};                    // дані з кодами символів для 7-сегм. індикатора
 init_cpu();                     // виклик функції ініціалізації контролера
 while(1)                        // вічний цикл
 {
 while (i2c_isbusy()!=0;         // чекаємо поки лінія не звільниться
 DDRB=0x03;                      // налаштування 2-х молодших біт порту B на вихід
 i2c_start();                    // функції формування сигналу початку передачі
 i2c_sendbyte(0b10010000);       // відправлення байта 0x90 (адреса)
 !i2c_sendbyte(0xEE);            // відправлення команди 0xEE
 i2c_stop();                     // формування сигналу закінчення передачі
```

```
i2c_start();                    // початок передачі даних
i2c_sendbyte(0b10010000);       // відправлення байта 0x90 (адреса)
i2c_sendbyte(0xAA);             // відправлення команди 0xAA
i2c_stop();                     // завершення передачі даних
i2c_start();                    // повторний старт
i2c_sendbyte(0b10010001);       // відправлення команди зчитування
c1=i2c_readbyte(1);             // зчитування старшого байта температури
c2=i2c_readbyte(0);             // зчитування молодшого байта температури
i2c_stop();                     // звільнення лінії I2C
t1=c1/10;                       // формування значення температури (десятки)
t2=c1-t1*10;                    // формування значення температури (одиниці)
PORTA=4;                        // включення лівого індикатора
PORTC=seg[t2];                  // вивід значення температури на індикатор
PORTA=3;                        // включення правого індикатора
PORTC=seg[t1];                  // вивід значення температури на індикатор
delay_ms(1000);                 // затримка 1 секунда
}
}
```

На рис. 4.22 представлена діаграма роботи інтерфейсу I2C. Тактовий сигнал SCL формує мікроконтролер на виводі PB1. На лінії даних SDA (вивід PB0) встановлюються рівні лог.1 та лог.0 відповідно до переданого байта. Після 8 тактових імпульсів, на 9-му відбувається перевірка біта підтвердження, що формує підлеглий пристрій (датчик температури DS1620).

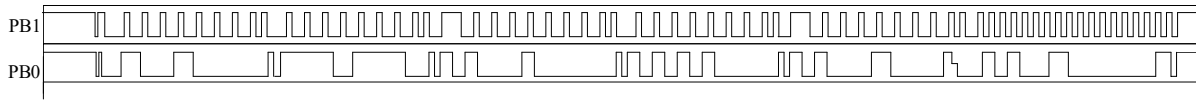


Рис. 4.22.

4.3.5. Електронний годинник на базі DS1307

Годинники реального часу (RTC – Real Time Clock) з послідовним інтерфейсом DS1307 – це двійково-десятковий годинник-календар, який містить 56 байт енергонезалежної статичної пам’яті. Адреси та дані передаються послідовно по двохпровідній двонапрямленій шині. Годинник-календар відраховує секунди, хвилини, години, день, дату, місяць і рік. Остання дата місяця автоматично коригується для місяців з кількістю днів менше 31, включаючи корекцію високосного року. Годинники працюють як в 24-годинному, так і в 12-годинному режимах з індикатором AM/PM. DS1307 має вбудовану схему спостереження за живленням, що виявляє перебої живлення і автоматично перемикається на живлення від батареї.

DS1307 на послідовній шині працює як підлеглий («Slave») пристрій. Доступ до нього виконується установкою умови START і передачею пристрою ідентифікаційного коду, за яким передається адреса регістра. До наступних за ним регістрів доступ здійснюється послідовно, поки не буде

виконана умова STOP. Блок-схема на рис. 4.23 показує основні елементи RTC з послідовним інтерфейсом.

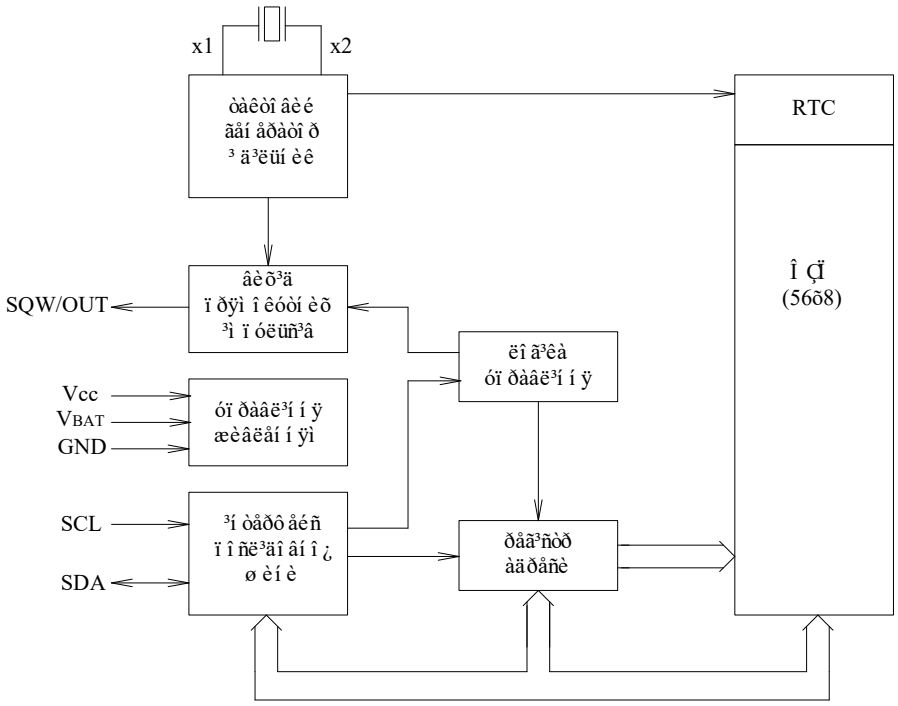


Рис. 4.23.

Типова схема підключення DS1307 представлена на рис. 4.24.

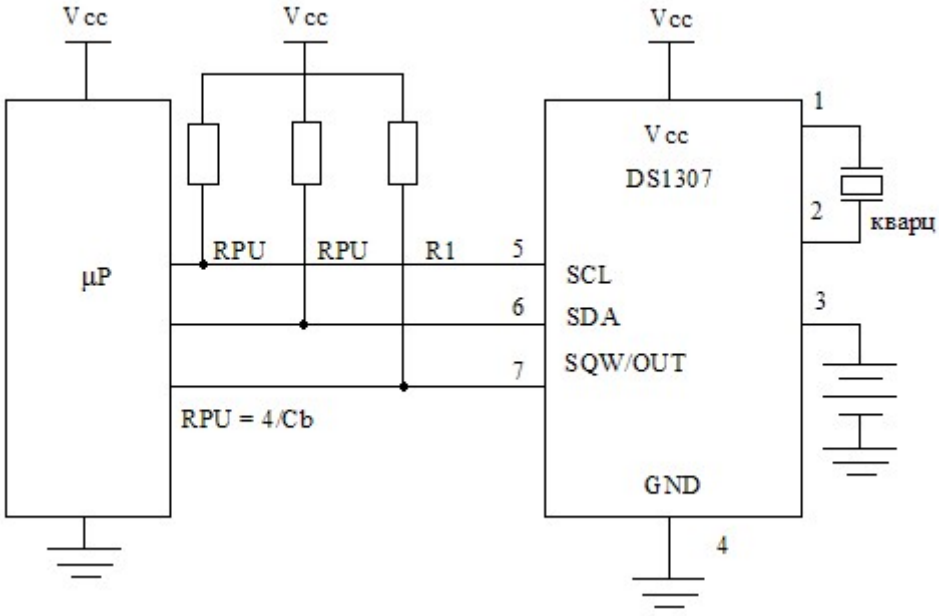


Рис. 4.24.

Карта адреси для RTC і регістрів ОЗП представлена на рис. 4.25. Регістри RTC розташовані в комірках пам’яті з адресами від 00h до 07h. Регістри ОЗП розташовані в комірках з адресами від 08h до 3Fh. У процесі багатобайтового доступу, коли вказівник адреси досягає 3Fh (кінець

простору ОЗП), він переміщується на комірку 00h – початок простору RTC.

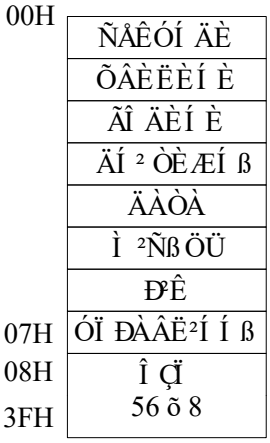


Рис. 4.25.

Регістри RTC показані на рис. 4.26. Час і календар встановлюються або ініціюються записом відповідних байтів регістра. Вміст регістрів годинника та календаря має двійково-десятковий формат. Біт 7 регістра 0 – це біт зупинки годинника (clock halt – CH). Коли цей біт встановлений в 1, тактовий генератор виключений. Коли скинутий в 0 – тактовий генератор включений.

DS1307 може працювати як в 12-годинному, так і в 24-годинному режимах. Біт 6 регістра годин відповідає за вибір 12- або 24-годинного режиму. Коли він встановлений в 1, встановлюється 12-годинний режим. У цьому режимі біт 5 – це біт AM/PM, при цьому високий логічний рівень означає PM. В 24-годинному режимі біт 5 – це біт другого десятка годин (20 – 23 години).

З появою на двохпровідній шині умови START, значення поточного часу копіюється в другий набір регістрів. Інформація про час зчитується із цих допоміжних регістрів, і годинник може продовжувати працювати. Це усуває необхідність повторного зчитування вмісту регістрів DS1307 у випадку оновлення їхнього вмісту під час операції зчитування.

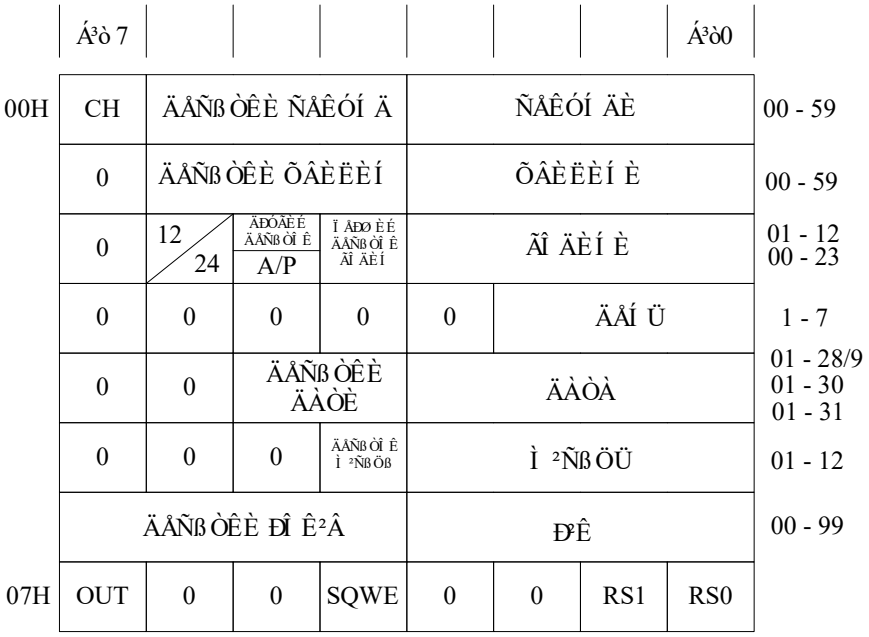


Рис. 4.26.

Керуючий регістр використовується для керування роботою виводу SQW/OUT.

БІТ 7	БІТ 6	БІТ 5	БІТ 4	БІТ 3	БІТ 2	БІТ 1	БІТ 0
OUT	0	0	SQWE	0	0	RS1	RS0

OUT (Output control – керування виходом) – цей біт керує логічним рівнем на виводі SQW/OUT, коли вихід сигналу із прямокутними імпульсами відключений. Якщо SQWE = 0, то логічний рівень на виводі SQW/OUT дорівнює 1, якщо OUT = 1, і 0, якщо OUT = 0.

SQWE (Square Wave Enable – включення сигналу із прямокутними імпульсами) – коли цей біт встановлений в 1, включається генерація прямокутних імпульсів. Їхня частота залежить від значення бітів RS0 та RS1. (Якщо частота вихідного сигналу встановлена в 1 Гц, регістри годин оновлюються по задньому фронту цих імпульсів).

RS (Rate Select – вибір частоти) – ці біти керують частотою прямокутних імпульсів, коли біт SQWE=1. У табл. 4.3 наведені частоти, які можна вибрати за допомогою бітів RS.

Таблиця 4.3.

RS1	RS0	Частота
0	0	1 Гц
0	1	4,096 кГц
1	0	8,192 кГц
1	1	32,768 кГц

В наступному прикладі виконується розробка принципової схеми годинника з виводом інформації на алфавітно-цифровий

рідкокристалічний індикатор, з використанням мікросхеми DS1307 і мікроконтролера ATMEGA128. Написати програму для мікроконтролера, що виконує зчитування дати та часу по інтерфейсу I2C. Використати вбудований периферійний вузол TWI мікроконтролера.

Принципова схема годинника представлена на рис. 4.27.

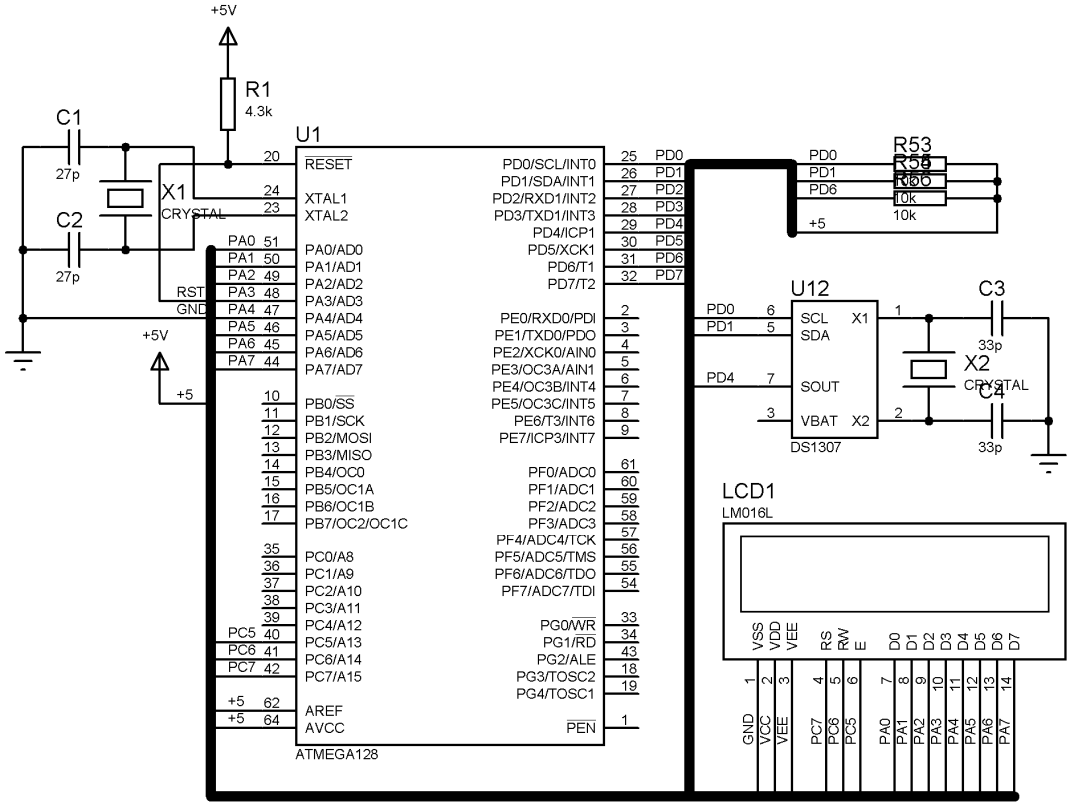


Рис. 4.27.

Текст програми для мікроконтролера, створеної з використанням компілятора ImageCraft, представлений нижче.

```
#include <iom128v.h>
#include <macros.h>
#define RS() PORTC|=0b10000000          // встановити сигнал команда/дані
#define NRS() PORTC&=~0b10000000        // скинути сигнал команда/дані
// встановити сигнал читання/запис (біт 6 порту А)
#define RW() PORTC|=0b01000000
// скинути сигнал читання/запис
#define NRW() PORTC&=~0b01000000
#define E() PORTC|=0b00100000           // встановити сигнал стробу
#define NE() PORTC&=~0b00100000         // скинути сигнал стробу
#define LCD PORTA                       // РКІ модуль підключений до порту С
#define DDLCD DDRA                      // даємо назву регістру напрямку даних

 void init_cpu()                        // функція ініціалізації мікроконтролера
 {
 DDRC=0b11100000;                       // керування дешифратором шини даних
 DDRA=0xFF;                             // шина даних
 PORTC=0;
```

```
PORTA=0xFF;                          // налаштування порту А на вихід
PORTC=5;
PORTA=0;
NRS();                               // скидання сигналу команда/дані
NRW();                               // скидання сигналу зчитування/запис
NE();                                // скидання тактового сигналу
}

void pause()                         // функція затримки
{
int tmp;                             // додаткова змінна для використання в циклі
// контролер виконує пусті операції
for(tmp=0;tmp<450;tmp=tmp+1) NOP();}

void init_lcd()                      // функція ініціалізації РКІ
{
DDLCD=0xFF;                          // налаштування порту на вихід
NRS();                  // скидання сигналу команда/дані - передаються команди
NRW();                               // скидання сигналу запис/зчитування - запис
LCD=0x38;                            // вивід команди 0x38 на порт контролера
write();                             // запис команди в модуль
LCD=0x0C;                            // вивід команди 0x0C на порт контролера
write();                             // запис команди в модуль
LCD=0x06;                            // вивід команди 0x06 на порт контролера
write();                             // запис команди в модуль
LCD=0x02;                            // вивід команди 0x02 на порт контролера
write();                             // запис команди в модуль
LCD=0x00;                            // вивід значення 0 на порт контролера
DDLCD=0x00;                          // перевід порту в стан високого імпедансу
}

void write()                         // функція формування тактового сигналу
{
 E();                                // установка тактового сигналу
 pause();                            // часова затримка
 NE();                               // скидання тактового сигналу
 pause();                            // часова затримка
}

void delay_ms(int ms)                // функція затримки
{
        int c1, c2;
        for(c1=0;c1<ms;c1++)
        for(c2=0;c2<970;c2++);
}

void fi2c_init(void)
{
 TWCR= 0x00;                         // відключити TWI
 TWBR= 0x02;                         // установка значення в регістрі дільника
 TWSR= 0x02;                         // установка значення дільника частоти
 TWCR= 0x04;                         // дозвіл роботи TWI
```

```
}

unsigned char fi2c_start(void)                  //функція формування start-умови
// повертається 1 якщо шина вільна, або 0 якщо шина зайнята
{
 // формування start-умови
 TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWSTA);
 // очікування установки прапора TWINT
 while (!(TWCR & (1<<TWINT)));
 // при відсутності помилок повертається значення 1
 if (((TWSR&0x08)==0x08) || ((TWSR&0x08)==0x10)) return 1;
 // якщо виникла помилка при формуванні start-умови, повертається 0
 return 0;
}

void fi2c_stop(void)                            // функція формування STOP-умови
{
 TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWSTO);
}

                                                // функція зчитування байта з шини
unsigned char fi2c_read(unsigned char ack) // параметр функції:
// 0 - сигнал підтвердження ack не формується, 1 - формується
{
  unsigned char data;                       // додаткова змінна
  // якщо параметр переданий у функцію дорівнює 0
  if (ack == 0) TWCR = (1<<TWEN)|(1<<TWINT);   // біт TWEA не встановлюється
  // інакше біт TWEA установлений
  else  TWCR = (1<<TWEN)|(1<<TWINT)|(1<<TWEA);
  // очікування установки біта TWINT
  while (!(TWCR & (1<<TWINT)));
  data = TWDR;                              // зчитування даних з регістра
  return data;                              // функція повертає зчитане значення
}
// функція передачі байта
unsigned char fi2c_write(unsigned char data)
{
 TWDR = data;                               // запис значення в регістр даних
 TWCR &= 0xCF;                              // біти формування умов start і stop скинуті
 TWCR |= 0x84;                              // установка біт TWINT і TWEN в 1
 // очікування завершення передачі даних
 while (!(TWCR & (1<<TWINT)));
 //$40 SLA+R відправлений, $18 SLA+W відправлений, $28 дані
 // відправлені й ACK отриманий
 if (((TWSR&0xF8)==0x18) || ((TWSR&0xF8)==0x28) || ((TWSR&0xF8)
 ==0x40)) return 1;
 // $20 SLA+W відправлений, $30 - дані відправлені,  ACK не
 // отриманий, $48 SLA+R відправлений
 if (((TWSR&0xF8)==0x20) || ((TWSR&0xF8)==0x30) || ((TWSR&0xF8)
 ==0x48)) return 0;
```

```
 return 0;
 }

void main ()
{
 char h, m, s, y, mon, day, d, i;
 char txt[6]="Time: ";
 char txt1[6]="Date: ";
 init_cpu();      // виклик функції ініціалізації мікроконтролера
 init_lcd();
 while(1)                                    // вічний цикл
 {
  fi2c_init();                               // ініціалізація TWI
  fi2c_start();                              // формування start-умови
  fi2c_write(0b11010000);                    // відправлення адреси пристрою
  fi2c_write(0);                             // установка вказівника адреси в комірку 0
  fi2c_stop();                               // формування stop-умови
  delay_ms(1);                               // пауза 1 мс
  fi2c_start();                              // формування start-умови
  fi2c_write(0b11010001);                    // відправлення адреси пристрою
  s=fi2c_read(1);                            // зчитування значення секунд
  m=fi2c_read(1);                            // зчитування хвилин
  h=fi2c_read(1);                            // зчитування годин
  day=fi2c_read(1);                          // зчитування номера дня
  d=fi2c_read(1);                            // зчитування дати
  mon=fi2c_read(1);                          // зчитування номера місяця
  y=fi2c_read(0);                            // зчитування року
  fi2c_stop();                               // формування stop-умови
  NRS();                                     // у РКІ-модуль передаються команди
  LCD=0x01;                                  // очищення екрана
  write();                                   // запис команди в РКІ-модуль
  RS();                                      // установка сигналу передачі даних
  delay_ms(1);                               // пауза 1 мс
  for(i=0;i<6;i++)                           // у циклі по буквах передається текст
  {
   LCD=txt[i];                               // вивід коду символу в порт
   write();                                  // запис даних у модуль
  }
   LCD=((h>>4)&0x07)+48;                     // вивід десятків годин
   write();                                  // запис даних у модуль
   LCD=(h&0x0F)+48;                          // вивід одиниць годин
   write();
   LCD=':';                                  // вивід двокрапки на екран
   write();
   LCD=((m>>4)&0x07)+48;                     // вивід десятків хвилин
   write();
   LCD=(m&0x0F)+48;                          // вивід одиниць хвилин
   write();
   LCD=':';                                  // вивід двокрапки на екран
   write();
   LCD=((s>>4)&0x07)+48;                     // вивід десятків секунд
   write();
```

```
  LCD=(s&0x0F)+48;                  // вивід одиниць секунд
  write();
  NRS();                            // установка сигналу передачі команд
  delay_ms(1);                      // пауза 1 мс
  LCD=0b10101000;                   // команда установки курсору на нижній рядок
  write();                          // запис команди в РКІ
  RS();                             // установка сигналу передачі даних
  delay_ms(1);                      // пауза 1 мс
  for(i=0;i<6;i++)                  // в циклі по буквах виводиться текст
  {
   LCD=txt1[i];                     // вивід коду символу в порт
   write();                         // запис даних у модуль
  }
   LCD=((d>>4) &0x07)+48;           // вивід дати на екран РКІ
   write();
   LCD=(d&0x0F)+48;
   write();
   LCD='.';                         // вивід крапки на екран
   write();
   LCD=((mon>>4)&0x07)+48;          // вивід номера місяця
   write();
   LCD=(mon&0x0F)+48;
   write();
   LCD='.';                         // вивід крапки на екран
   write();
   LCD=((y>>4)&0x07)+48;            // вивід значення року
   write();
   LCD=(y&0x0F)+48;
   write();
   delay_ms(500);                   // пауза 500 мс
  }
}
```

4.4. Послідовний інтерфейс SPI

Схеми послідовного периферійного інтерфейсу забезпечують синхронний послідовний зв'язок, в якому можлива передача даних у зовнішні пристрої в режимі ведучого й веденого. Якщо в системі присутній зовнішній процесор, за допомогою цього інтерфейсу може бути забезпечений міжпроцесорний зв'язок.

Основні характеристики інтерфейсу SPI:

– повнодуплексний 3-провідний синхронний обмін даними;

– режим роботи ведучий або ведений;

– обмін даними з переданими першими старшим або молодшим бітами;

– чотири програмовані швидкості обміну даними;

– переривання по закінченню передачі;

– вихід з режиму зниженого споживання енергії (тільки в режимі веденого).

4.4.1. Загальна характеристика інтерфейсу SPI

Апаратна частина послідовного периферійного інтерфейсу представляє собою регістр зсуву, що послідовно передає біти даних в аналогічні інтерфейси інших пристроїв. Під час процесу передачі один із пристроїв працює як ведучий, забезпечуючи контроль над потоком даних, у той час як інші, ведені пристрої, приймають або передають дані під керуванням ведучого. Функції провідного пристрою інтерфейсу можуть передаватися від одного процесора іншому (мультимастерний протокол у порівнянні з одномастерним, коли тільки один процесор завжди працює в якості ведучого, а всі інші в якості ведених), і ведучий пристрій може вести передачу даних декільком веденим пристроям одночасно.

З'єднання між ведучим й веденим пристроями показане на рис. 4.28.

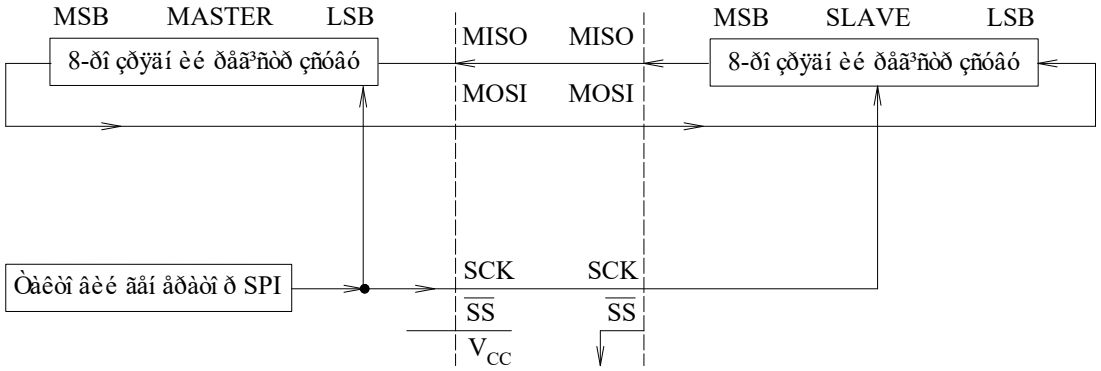


Рис. 4.28.

Два регістри зсуву ведучого і веденого можна розглядати як один рознесений 16-розрядний циклічний регістр зсуву. При зсуві даних із ведучого мікроконтролера у ведений одночасно відбувається зсув даних з веденого мікроконтролера у ведучий, тобто протягом одного циклу зсуву відбувається обмін даними між ведучим й веденим мікроконтролерами.

Вивід SCK є виходом тактового сигналу ведучого мікроконтролера і входом тактового сигналу веденого. При записі ведучим мікроконтролером даних у регістр SPI починає працювати тактовий генератор, і записані дані передаються з виходу MOSI ведучого мікроконтролера на вхід MOSI веденого. Після передачі одного байта генератор зупиняється, встановлюючи прапор завершення передачі SPIF. Якщо в регістрі SPCR встановлений прапор дозволу переривання (SPIE), то відбудеться запит переривання. Вхід вибору веденого SS, для вибору індивідуального SPI пристрою в якості веденого, встановлюється на низький рівень. При установці високого рівня на виводі SS порт SPI деактивується. Режим ведучий/ведений може бути встановлений програмним способом установкою або очищенням біта MSTR у регістрі керування SPI. На рис. 4.29 представлена блок-схема SPI.

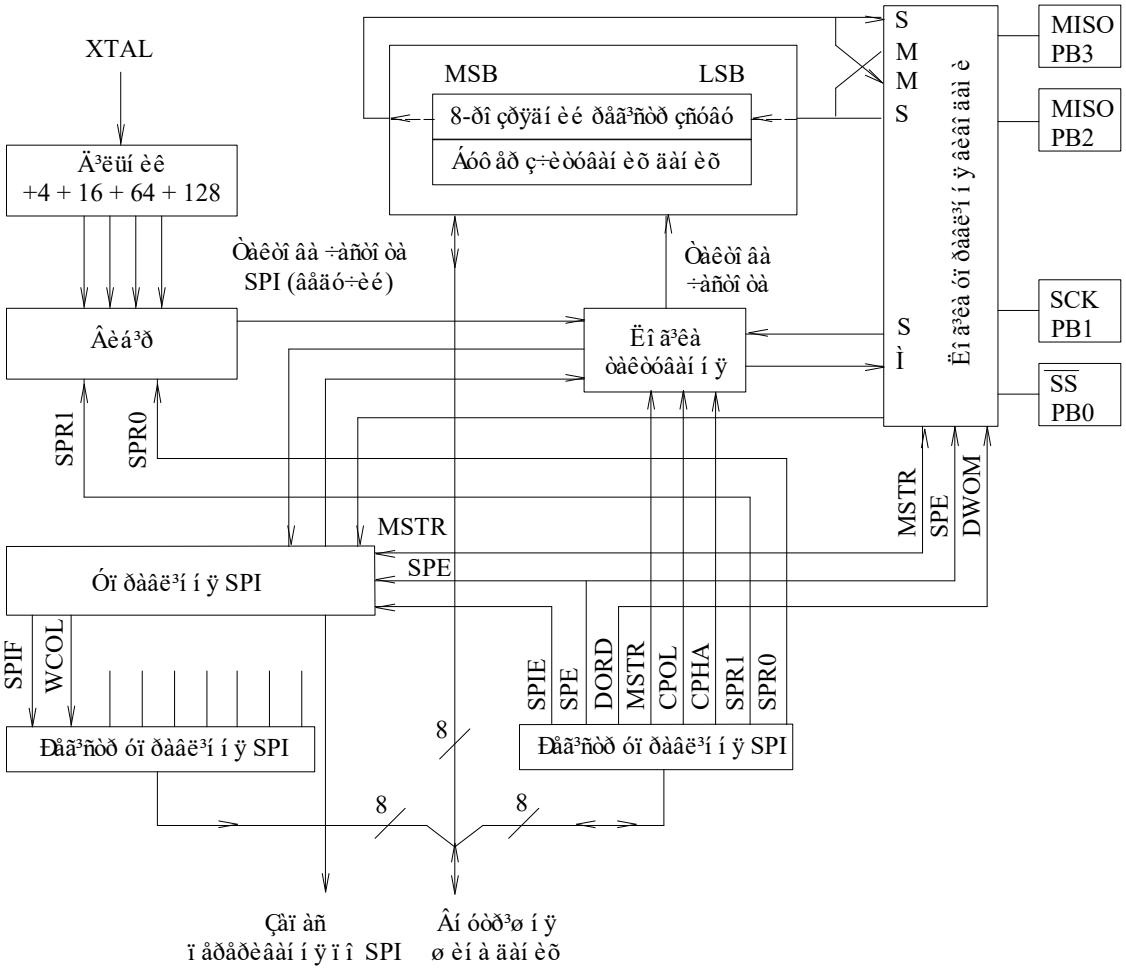


Рис. 4.29.

При роботі SPI ведучим (біт MSTR регістра SPCR встановлений), користувач має можливість установити напрямок роботи виводу #SS. Якщо вивід #SS налаштований як вихід, то вивід є виводом загального призначення і він не активується системою SPI. Якщо ж вивід #SS налаштований як вхід, то для забезпечення роботи ведучого SPI він повинен утримуватися на високому рівні. Якщо в режимі ведучого вивід #SS є входом, і зовнішньою периферійною системою на нього поданий низький рівень, то SPI сприйме його як звернення іншого провідного SPI до себе, як до веденого. Щоб уникнути конфліктної ситуації на шині, система SPI виконує наступні дії:

1. Біт MSTR у регістрі SPCR очищається й SPI система стає веденою. Результатом цього є те, що виводи MOSI та SCK стають входами.
2. Встановлюється прапор SPIF регістра SPSR, і якщо дозволено переривання SPI, починається виконання підпрограми обробки переривання.

Якщо SPI працює в режимі веденого, то вивід #SS постійно працює на вхід. Якщо на вивід #SS поданий низький рівень, то SPI активується і вивід MISO стає виходом. Всі інші виводи є входами. Якщо вивід #SS

утримується на високому рівні, то всі виводи є входами, SPI пасивний, і це означає, що він не буде одержувати вхідних даних.

Існує чотири варіанти комбінації фази й полярності SCK відносно послідовних даних, які встановлюються керуючими бітами CPHA та CPOL. Формати передачі даних показані на рис. 4.30 і рис. 4.31.

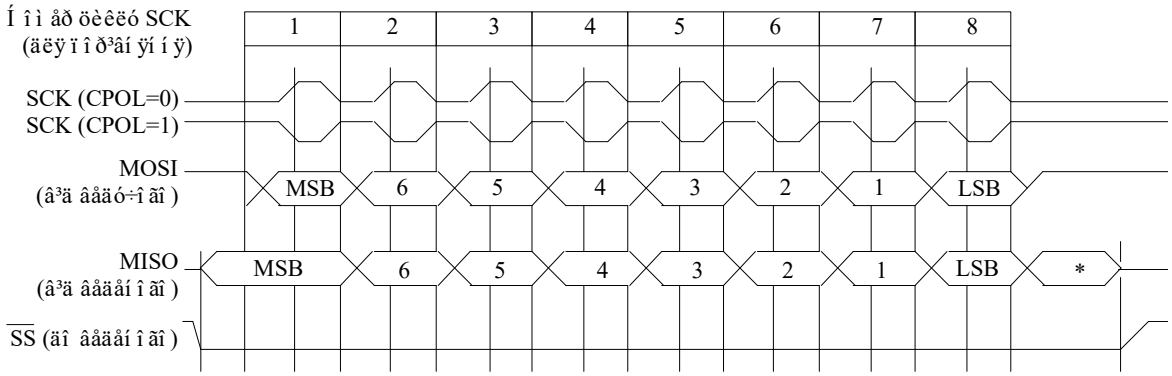


Рис. 4.30.

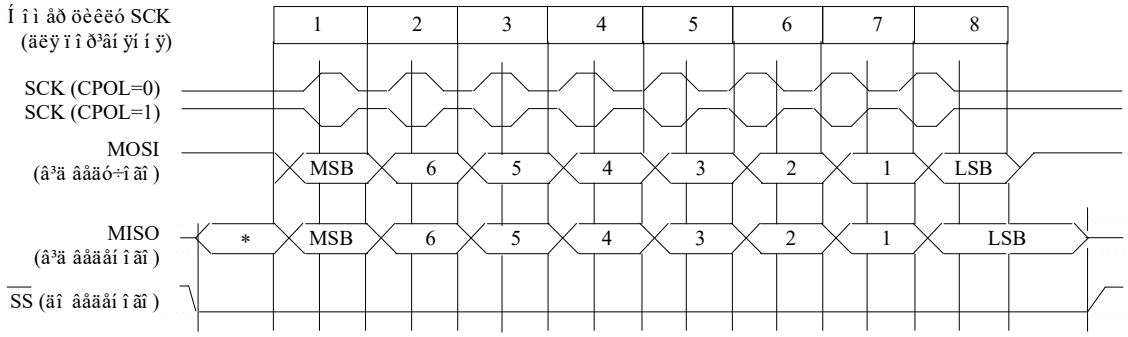


Рис. 4.31.

4.4.2. Особливості використання модуля SPI мікроконтролерів

Після переходу SPI у режим веденого вивід SS завжди працює як вхід. У цьому випадку SPI активується, якщо на вхід SS подати низький рівень, а вивід MISO стає виходом. Всі інші виводи працюють як входи. Якщо на вхід SS подати високий рівень, то всі виводи стануть входами і SPI перейде в пасивний стан, у якому блокується прийом вхідних даних. Логіка SPI скидається, як тільки на вивід SS подається високий логічний рівень.

Опис регістрів SPI для контролерів AVR приводиться нижче.

Регістр SPCR – регістр керування:

Розряд	7	6	5	4	3	2	1	0
	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.
Початкове значення	0	0	0	0	0	0	0	0

SPIE – дозвіл переривання SPI.

SPE – SPI включений. Якщо встановлено цей біт, то виводи #SS, MISO, MOSI та SCK працюють як виводи SPI, інакше – як прості виводи порту.

DORD – напрямок передачі даних. Якщо цей біт встановлено, то передача починається з молодшого біта, якщо скинуто – зі старшого.

MSTR – якщо встановлено, контролер працює як Master, якщо скинуто – як Slave. Керується також виводом #SS, якщо він налаштований на вхід – при подачі "0" на #SS біт MSTR скидається.

CPOL – визначає рівень на виході SCK у режимі очікування – SCK = CPOL.

CPHA – якщо встановлено, передача і прийом біта відбуваються по спаду сигналу, якщо скинуто – по фронту.

SPR1, SPR0 – дільник тактової частоти (F – частота генератора контролера).

SPR1	SPR0	Частота
0	0	F/4
0	1	F/16
1	0	F/64
1	1	F/128

Регістр SPSR – регістр стану:

Розряд	7	6	5	4	3	2	1	0
	SPIF	WCOL	–	–	–	–	–	SPI2X
Зчитування /запис	Зч.	Зч.	Зч.	Зч.	Зч.	Зч.	Зч.	Зч./Зап.
Початкове значення	0	0	0	0	0	0	0	0

SPIF – прапор переривання по закінченню обміну даними. Встановлюється й скидається апаратно. Якщо біт SPIE у регістрі SPCR встановлений, і дозволене глобальне переривання, генерується сигнал переривання. Біт SPIF може бути очищений також при першому зчитуванні регістра стану.

WCOL – прапор помилки при записі. Встановлюється, коли прийнятий байт накладається на ще не зчитаний з регістра даних. Скидається зчитуванням регістра стану.

Біти 5..1 – зарезервовані біти. При зчитуванні завжди покажуть 0.
SPI2X – біт подвоєння швидкості SPI.

Регістр SPDR – регістр даних:

Розряд	7	6	5	4	3	2	1	0
	Ст. розр.							Мол. розр.
Зчитування /запис	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.	Зч. /Зап.

Регістр даних SPI представляє собою регістр із можливістю зчитування/запису і призначений для пересилання даних. Запис у регістр SPDR ініціює передачу даних, зчитування регістра призводить до зчитування даних із регістра зсуву приймача. Передача та прийом тактуються по лінії SCK відповідно до конфігурації, що задана в регістрі керування. Після того, як байт переданий, тактування припиняється, у регістрі стану встановлюється прапор SPIF і викликається переривання по закінченню передачі, якщо воно дозволено. Запис у регістр даних передаючого контролера неможливий, поки не звільнений регістр зсуву SPI.

В SPI установка й передача даних відбувається по протилежних фронтах тактових імпульсів. Такий поділ операцій зсуву та передачі має певні переваги, дозволяючи уникати жорсткого часового режиму між цими двома операціями. Це дозволяє полегшити задачу синхронізації при розробці інтегральних схем або плат. Але, з іншого боку, через комбінації полярності й фази синхроімпульсів виникає чотири режими роботи, і ведучому пристрою доводиться налаштовуватися на той режим, що використовується веденим.

4.4.3. Організація обміну даними в мультимікропроцесорній системі

Зараз SD-карти пам'яті (Secure Digital Memory Card) є найбільш популярними картами пам'яті для мобільних пристроїв. SD-карти (далі SDC) були розроблені як сумісні з MMC-картами (Multi Media Card) (далі MMC), при цьому SDC-сумісні пристрої також можуть використовувати MMC, але з деякими застереженнями. Також існують зменшені версії, такі як RS-MMC, miniSD й microSD, з аналогічними функціями. MMC/SDC має вбудований мікроконтролер, всі засоби керування flash-пам'яттю (стирання, зчитування, запис і захист від помилок) повністю зосереджені усередині карти. За замовчуванням дані передаються між картою й ведучим контролером блоками по 512 байт, так що з погляду прикладних програм її можна розглядати як звичайний жорсткий диск. На рис. 4.32 представлений зовнішній вигляд карт пам'яті SDC та MMC.



Рис. 4.32.

На рис. 4.33, показані контактні поверхні SDC й MMC. MMC має сім контактних площадок, а в SDC їх дев'ять, дві з яких додані до семи площадок MMC. Три контакти в кожній картці зайняті як виводи живлення, тому кількість робочих сигналів становить відповідно чотири або шість. Обмін даними між ведучим контролером і картою здійснюється у вигляді послідовної синхронної передачі.

Діапазон робочої напруги живлення зазначений у регістрі OCR. Його необхідно зчитати, щоб переконатися в правильності обраної напруги. Однак живляча напруга може приймати фіксоване значення, оскільки MMC/SDC працюють у діапазоні напруг від 2,7 до 3,6 В. Оскільки струм споживання може досягати декількох десятків міліампер, ведучий контролер повинен бути здатний забезпечити струм 100 мА.

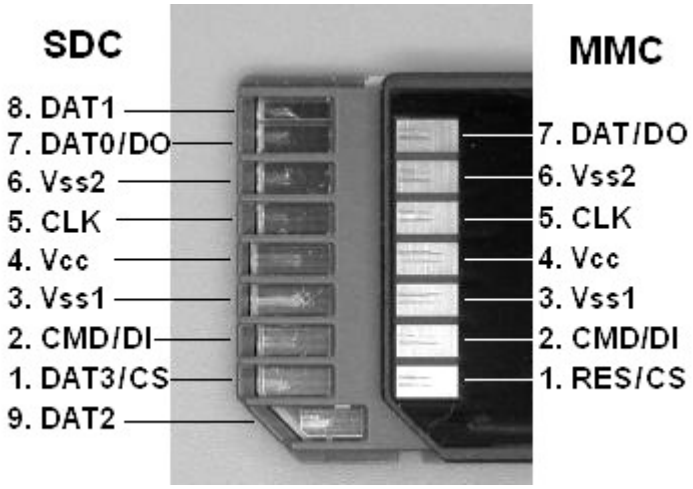


Рис. 4.33.

Режим SPI є альтернативним режимом роботи з MMC/SDC, більш простим у порівнянні з використанням їхнього рідного інтерфейсу. MMC/SDC можна підключити через універсальний порт SPI або порт GPIO (універсальний порт введення/виведення), вбудований у більшість мікроконтролерів. Найбільш підходящим режимом для роботи з MMC/SDC по SPI є режим "SPI 0" (позитивний синхроімпульс, передача по фронту синхроімпульсу, зсув по зрізу). Режим "SPI 3" теж підходить для цих цілей.

У режимі SPI напрямок даних на сигнальній лінії фіксований, і дані передаються послідовно по байтах. Кадр команди, переданий від ведучого контролера до карти, має фіксовану довжину в 6 байтів і формат, представлений на рис. 4.34 Оскільки вся передача даних керується послідовними синхроімпульсами, що генеруються ведучим, останній повинен продовжувати зчитування байтів, поки не одержить дійсну відповідь. Час відповіді на команду (NCR) становить від 0 до 8 байтів для SDC, і від 1 до 8 байтів для MMC.

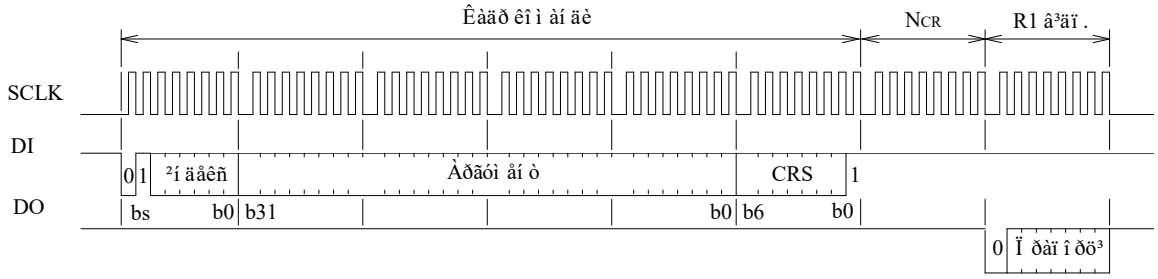


Рис. 4.34.

Сигнал CS повинен утримуватися в стані низького рівня протягом всієї транзакції (команда, відповідь і передача даних, якщо такі є). Поле CRC в режимі SPI не обов'язкове, але його присутність необхідна, щоб скласти кадр команди.

Кожна команда в табл. 4.4 представлена у вигляді абревіатури (наприклад, GO_IDLE_STATE), або у вигляді позначення CMD#, де # – номер індексу команди, що може приймати значення від 0 до 63. У наведеній нижче таблиці описана тільки група команд, що використовується для зчитування/запису та ініціалізації карти. Більш детальну інформацію із всіх команд ви можете знайти в специфікаціях від MMCA та SDCA.

Таблиця 4.4.

Індекс команди	Аргумент	Відп.	Дані	Абревіатура	Опис
CMD0	Hi(0)	R1	Hi	GO_IDLE_STATE	Програмне скидання.
CMD1	Hi(0)	R1	Hi	SEND_OP_COND	Ініціювати процес ініціалізації.
ACMD41 (*1)	Hi(0)	R1	Hi	APP_SEND_OP_COND	Тільки для SDC. Ініціювати процес ініціалізації.
CMD9	Hi(0)	R1	Так	SEND_CSD	Зчитати регістр CSD.
CMD10	Hi(0)	R1	Так	SEND_CID	Зчитати регістр CID.
CMD12	Hi(0)	R1b	Hi	STOP_TRANSMISSION	Запинити зчитування даних.
CMD17	Адреса [31:0]	R1	Так	READ_SINGLE_BLOCK	Зчитати блок.
CMD18	Адреса [31:0]	R1	Так	READ_MULTIPLE_BLOCK	Зчитати декілька блоків.
CMD23	Кількість блоків [15:0]	R1	Hi	SET_BLOCK_COUNT	Тільки для MMC. Визначити кількість блоків для передачі з наступною командою багатоблокового зчитування/запису.

ACMD23 (*1)	Кількість блоків [15:0]	R1	Ні	SET_WR_BLOCK_ERASE_COUNT	Тільки для SDC. Визначити кількість блоків для попереднього стирання для наступної багато блокової команди запису.
CMD24	Адреса [31:0]	R1	Так	WRITE_BLOCK	Записати блок.
CMD25	Адреса [31:0]	R1	Так	WRITE_MULTIPLE_BLOCK	Записати декілька блоків.
CMD55 (*1)	Ні(0)	R1	Ні	APP_CMD	Команда, що визначається програмою.
CMD58	Ні(0)	R3	Ні	READ_OCR	Зчитати OCR.
*1: ACMD означає послідовність команд CMD55, CMD.					

У стані очікування карта приймає тільки команди CMD0, CMD1 та CMD58, будь-які інші команди будуть відхилені. Коли карта виявляє команду CMD1, вона починає ініціалізацію. Щоб визначити, чи завершена ініціалізація, ведучий контролер повинен повторювати посилку команди CMD1 і перевіряти відповідь. Як тільки карта успішно була ініційована, біт "In Idle State" у відповіді R1 буде скинутий (R1 = 0x00). Процес ініціалізації може зайняти кілька сотень мілісекунд (більші за об’ємом карти мають тенденцію до більших витрат часу). Після закінчення ініціалізації будуть прийматися й команди зчитування/запису. У цей час можна зчитати OCR й CID для підтвердження діапазону робочої напруги, ємності карти або будь-якої іншої характеристики.

У транзакції з передачею даних після відповіді на команду може бути переданий/отриманий один або більше блоків даних. Блок даних передається як пакет даних, що складається з маркера (Token), блоку даних (Data Block) і контрольної суми (CRC). Формат пакета даних показаний на рис. 4.35. Існує три маркери даних. Один з них – маркер "Stop Tran", який означає кінець багатоблокового запису, використовується у вигляді одиночного байта без блоку даних та CRC.

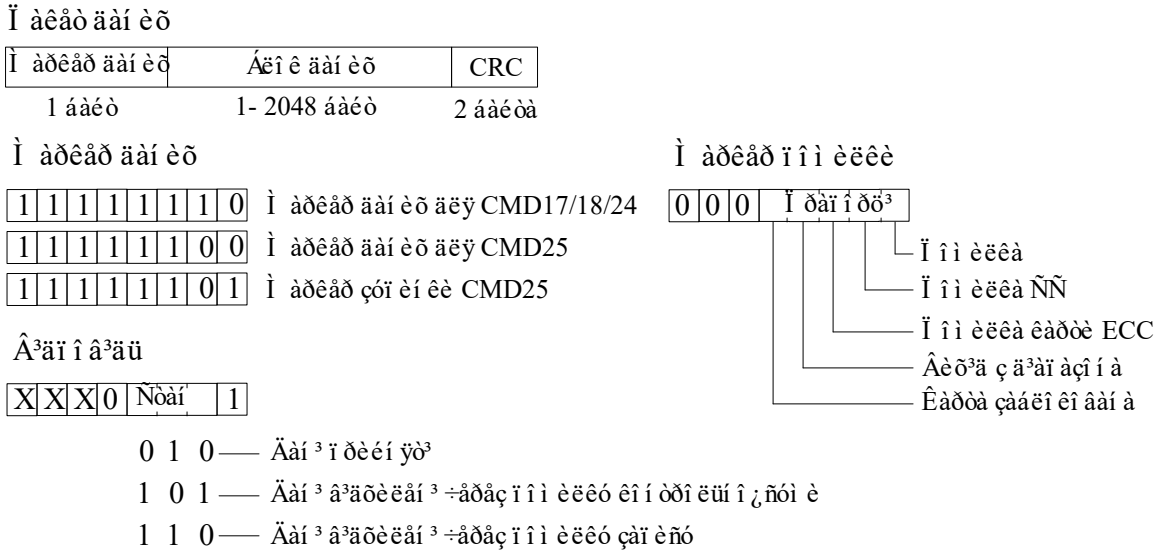


Рис. 4.35.

Одноблокове зчитування (рис. 4.36) ініціюється командою CMD17. Її аргумент задає комірку, з якої потрібно почати зчитування. Читання здійснюється по байтах. Адреса повинна бути байт-байт-орієнтованим, тобто якщо ви вказуєте адресу сектора, що задається декількома старшими бітами, вам необхідно доповнити його нулями для одержання повних чотирьох адресних байтів.

У відповідь на команду CMD17 карта передає ведучому контролеру пакет даних. Після виявлення правильного маркера даних ведучий контролер приймає наступний за ним блок даних і два байти CRC, які необхідно прийняти, навіть якщо CRC не використовується. По замовчуванню розмір блоку дорівнює 512 байт, але його можна змінити командою CMD16. Якщо під час операції зчитування відбулася яка-небудь помилка, замість пакета даних буде повернутий маркер помилки (рис. 4.32).

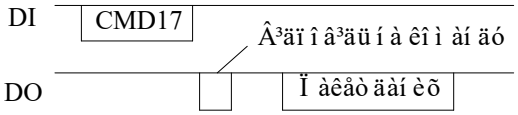


Рис. 4.36.

За допомогою команди багатоблокового зчитування CMD18 можна прочитати з карти послідовність із декількох блоків, починаючи із заданої адреси (рис. 4.37). Якщо перед цією командою за допомогою команди CMD23 (тільки для MMC) не було задане число переданих блоків, буде ініційоване необмежене багатоблокове зчитування, тобто операція зчитування буде тривати, поки ведучий контролер не перерве її командою CMD12.

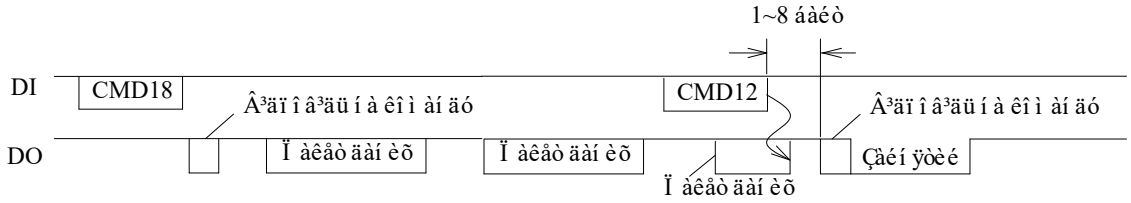


Рис. 4.37.

Одноблоковий запис демонструє часова діаграма, що представлена на рис. 4.38. Байт, що отримується відразу після передачі CMD12, є заповнюючим, і його не потрібно враховувати. Після цього байта приймається відповідь на команду.

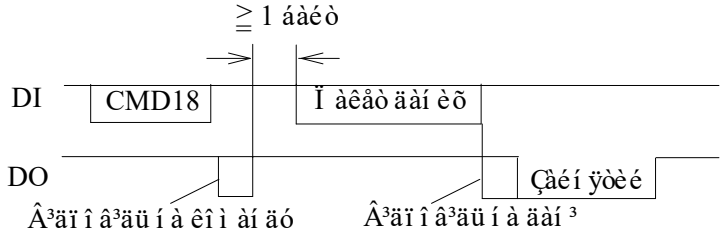


Рис. 4.38.

Після того як карта прийняла команду запису CMD24, ведучий контролер після деякого проміжку (один або більше байтів) передає в карту пакет даних. Формат пакета такий самий, як у команди блокового читання. Після передачі пакету, карта відразу ж видає відповідь на дані (Data Response), за яким передається прапор зайнятості. Більшість карт не можуть змінювати розмір блоку, що записується, він є фіксованим і становить 512 байт.

За правилами режиму SPI сигнал CS повинен перебувати в активному рівні протягом всієї транзакції, однак є виключення із цього правила. Коли карта зайнята, ведучий контролер може зняти сигнал CS, щоб звільнити шину SPI для якого-небудь іншого SPI-пристрою. Якщо ж знову вибрати карту в той час, коли вона зайнята виконанням внутрішнього процесу, карта знову встановить сигнал DO у низький рівень. Тому, щоб скоротити час очікування, краще виконувати перевірку на зайнятість безпосередньо перед видачею команди та пакета даних, а не очікувати звільнення карти після посилки команди. Крім того, внутрішній процес ініціюється через байт після відповіді даних, тобто необхідно видати 8 тактових імпульсів, щоб ініціювати внутрішню операцію запису. Стан сигналу CS під час цих восьми тактових імпульсів не враховується, тому можна сполучити цю ініціацію із процесом звільнення шини.

Багатоблоковий запис демонструє часова діаграма, представлена на рис. 4.39. За допомогою команди багатоблокового запису CMD25 можна записати послідовність із декількох блоків, починаючи із заданої адреси.

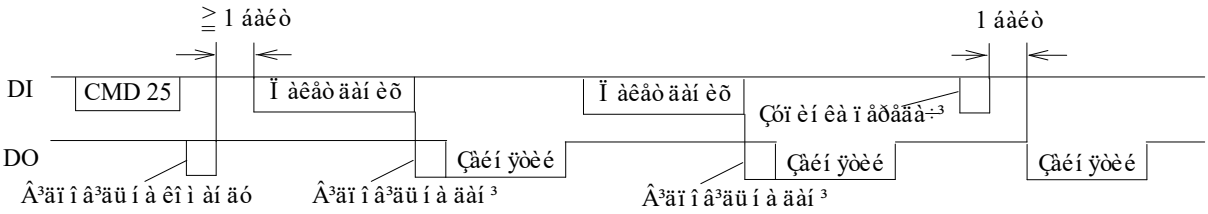


Рис. 4.39.

Якщо перед цією командою число переданих блоків не було задано за допомогою команди CMD23 (тільки для MMC) або ACMD23 (для SDC), транзакція буде ініційована як необмежений багато блоковий запис, тобто операція запису буде тривати, поки ведучий контролер не перерве її передачею маркера "Stop Tran". Що ж стосується SDC, то транзакція багатоблокового запису повинна перериватися маркером "Stop Tran" незалежно від того, чи є вона визначеною або необмеженою.

4.4.4. Мікропроцесорна система збору та накопичення даних на карту пам'яті SD/MMC

На рис. 4.40 представлена принципова схема пристрою зчитування/запису карт пам'яті, створена в Proteus.

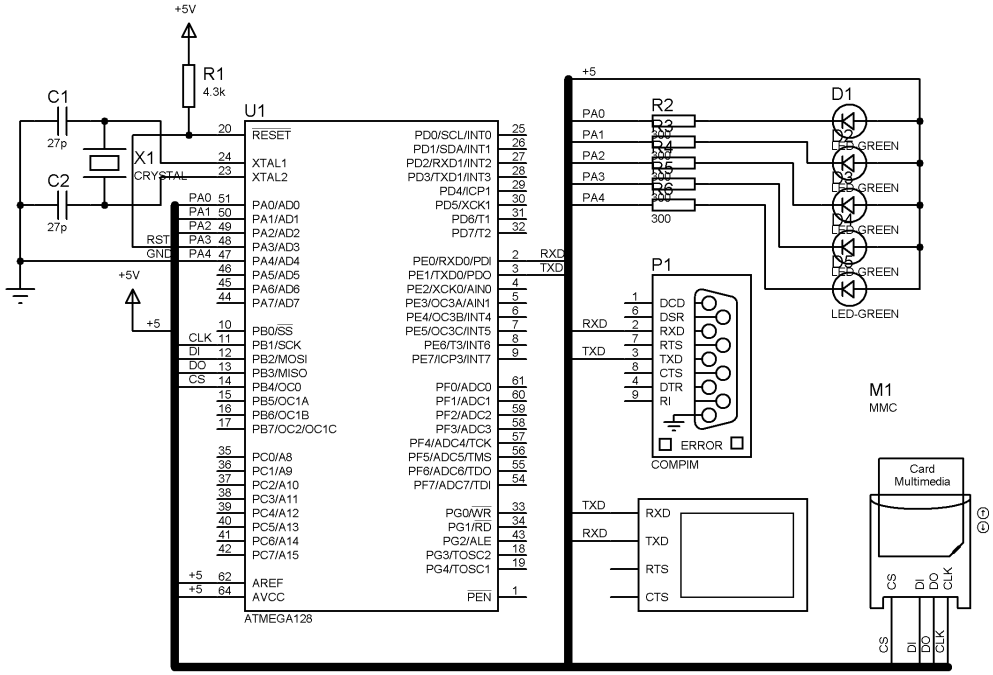


Рис. 4.40.

В наступному прикладі виконується розробка програми для мікроконтролера, що виконує запис 512 байт даних у заданий сектор карти пам'яті, а після цього зчитує ці дані й передає по UART. При кожному звертанні до карти пам'яті, по UART відправляється повідомлення, що інформує користувача про результат виконання дій. За допомогою

світлодіодних індикаторів інформувати користувача про помилки звертання до карти пам'яті.

Текст програми для мікроконтролера ATMega128 представлений нижче.

```
#include <iom128v.h>                    // підключення заголовочних файлів
#include <macros.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define SPIDDR DDRB                     // присвоєння регістрам альтернативних імен
#define SPIPORT PORTB
#define SPIPIN PINB
#define CS 0x10
#define RST0() PORTA|=0x01
#define SET0() PORTA&=~0x01
#define RST1() PORTA|=0x02
#define SET1() PORTA&=~0x02
#define RST2() PORTA|=0x04
#define SET2() PORTA&=~0x04
#define RST3() PORTA|=0x08
#define SET3() PORTA&=~0x08
#define RST4() PORTA|=0x10
#define SET4() PORTA&=~0x10

#define DISABLE_MMC() PORTB|=CS
#define ENABLE_MMC() PORTB&=~CS

void delay(int ms);                     // функція часової затримки
void init_cpu(void);                    // функція ініціалізації контролера
// функція передачі строки по UART
void uart_puts(char * s, int size);
// функція передачі байта даних по SPI
char sspi(char d);
// функція передачі команди в карту пам'яті
char MMC_Command(char bef, unsigned long addr, char bef);
char MMC_Init();                // функція ініціалізації карти пам'яті
// функція запису сектору
char MMC_WriteSector(unsigned long Sector, char * buff);
// функція зчитування сектору
char MMC_ReadSector(unsigned long Sector, char *buff);
char arr[512]; // масив з 512 елементів
// оголошення строкових констант для контролю операцій з картою
char err0[]="MMC Initialization error!";
char err1[]="Command 'writes a single block' (0x58) error!";
char err2[]="Invalid response! MMC write data error!";
char err3[]="Command 'read single block' (0x51) error!";
char msg0[]="Initialization...";
char msg1[]="Initialization OK!";
char msg2[]="Begin write data to MMC...";
char msg3[]="Write data OK!";
char msg4[]="Begin read data from MMC...";
char msg5[]="Read data OK!";
```

```
// функція часової затримки
void delay(int ms)
{
int c1, c2;                         // оголошення додаткових змінних
for(c1=0;c1<ms;c1++)                // в циклі виконується порожня операція
 for(c2=0;c2<970;c2++) NOP();
}

// функція ініціалізації мікроконтролера
void init_cpu(void)
{
// налаштування UART
UCSR0C = 0x06;
UBRR0L = 0x33;                      // швидкість передачі даних 9600 біт/с
UCSR0B = 0x18;                      // дозвіл роботи приймача й передавача
DDRB=0b00010111;                    // налаштування виводів порту B
// налаштування SPI
SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR1) | (1 << SPR0);
}

// функція передачі даних по UART
void uart_puts(char * s, int size)
{
int i;                              // додаткова змінна
for(i=0;i<size-1;i++)
{
 UDR0=s[i];  // передача даних
 // очікування завершення передачі даних
 while(!(UCSR0A&(1<<UDRE0)));
}
 UDR0=13;                           // передача символу нового рядка
 while(!(UCSR0A&(1<<UDRE0)));
}

// функція передачі байта даних по SPI
char sspi(char d)
{
char r=0;
SPDR=d;                             // запис значення в регістр даних
// очікування завершення процесу передачі
while(!(SPSR&(1<<SPIF)));
r=SPDR; // зчитування з регістра прийнятого байта
return r;
}

// функція передачі команди в карту MMC
char MMC_Command(char bef, unsigned long addr, char bef)
{
sspi(0xFF);                         // передача значення 0xFF
sspi(bef);                          // передача коду команди
sspi((char)(addr >> 24));           // передача адреси
```

```
sspi((char)(addr  >> 16));
sspi((char)(addr >> 8));
sspi((char)addr);
sspi(bef);
sspi(0xFF);
// повертається останнє зчитане значення
return sspi(0xFF);
}

// функція ініціалізації карти MMC
char MMC_Init()
{
char i;
DISABLE_MMC();                  // відключити MMC
// очікування 10*8=80 тактових імпульсів
for(i=0; i < 10; i++) sspi(0xFF);
ENABLE_MMC();                   // активувати MMC

// відправити повідомлення по UART
uart_puts(msg0,sizeof(msg0));
if (MMC_Command(0x40,0,0x95)!=1)
{
 SET0();
 // відправити повідомлення по UART
 uart_puts(err0,sizeof(err0));
 return 1;                      // виникла помилка при роботі з MMC
}
 // якщо карта MMC відсутня то програма зациклюється
  while (MMC_Command(0x41,0,0xFF)!=0);
 // відправити повідомлення по UART
 uart_puts(msg1,sizeof(msg1));
 RST0();
 return 0;
}
// функція запису сектора в пам'ять
char MMC_WriteSector(unsigned long Sector, char * buff)
{
int i;
ENABLE_MMC(); // активувати карту MMC
// передача команди CMD24
if (MMC_Command(0x58,(Sector<<9),0xFF) !=0)
{
 SET1();
 // відправити повідомлення по UART
 uart_puts(err1,sizeof(err1));
 DISABLE_MMC();                 // деактивувати MMC
 return 1; // вихід з функції
}
RST1();
// відправити повідомлення по UART
uart_puts(msg2,sizeof(msg2));
// формування пакетів даних у відповідності до протоколу
```

```
sspi(0xFF);
sspi(0xFF);
sspi(0xFE);
// передача 512 байт для запису
for (i=0; i<512; i++) sspi(buff[i]);
sspi(0xFF);
sspi(0xFF);
// перевірка коду стану
if((sspi(0xFF)& 0x1F)!=0x05)
{
 SET2();
 // відправити повідомлення по UART
 uart_puts(err2,sizeof(err2));
 DISABLE_MMC();                    // деактивувати MMC
 return 2;                         // вихід з функції
}
RST2();
// очікування поки карта не стані вільною
while(sspi(0xFF) != 0xFF);
// відправити повідомлення по UART
uart_puts(msg3,sizeof(msg3));
DISABLE_MMC();                     // деактивувати MMC
return 0;                          // вихід з функції
}
// функція зчитування сектора (512 байт) з карти MMC
char MMC_ReadSector(unsigned long Sector, char *buff)
{
int i;                             // додаткова змінна
ENABLE_MMC();                      // активувати карту пам'яті
// передача команди CMD17
if(MMC_Command(0x51,(Sector<<9),0xFF)!=0)
{
 SET3();
 // відправити повідомлення по UART
 uart_puts(err3,sizeof(err3));
 DISABLE_MMC();                    // деактивувати карту MMC
 return 1;                         // MMC: помилка зчитування
}
RST3();

uart_puts(msg4,sizeof(msg4));
// очікування значення 0xFE - кінець передачі
while(sspi(0xFF)!=0xFE);
// зчитування даних з карти пам'яті
for(i=0; i<512; i++) buff[i]=sspi(0xFF);
sspi(0xFF);
sspi(0xFF);
// відправити повідомлення по UART
uart_puts(msg5,sizeof(msg5));
DISABLE_MMC();                     // деактивувати MMC
return 0;                          // вихід з функції
}
```

```
// головна функція програми
void main()
{
int i;                                  // оголошення додаткової змінної
// формування даних для запису
for(i=0;i<512;i++) arr[i]=(char)(i>>1);
init_cpu();                             // виклик функції ініціалізації
MMC_Init();                             // ініціалізація карти пам'яті
MMC_WriteSector(0,arr);                 // запис даних у сектор 0
MMC_ReadSector(0,arr);                  // зчитування даних з сектору 0
// передача даних по UART
for(i=0;i<512;i++)
{
   UDR0=arr[i]; //
   while(!(UCSR0A&(1<<UDRE0)));
}
while(1);                               // зациклення програми
}
```

Після компіляції програми й моделювання, за допомогою віртуального термінала (рис. 4.41) можна бачити результат всіх операцій звертання до пам'яті.

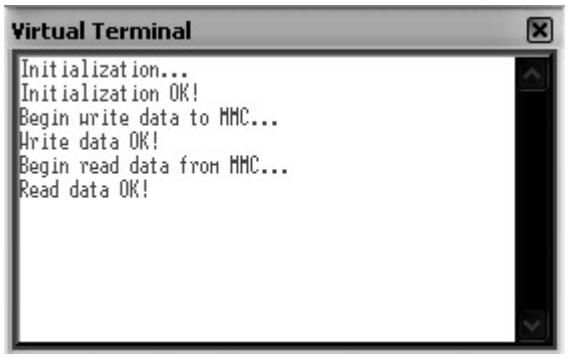


Рис. 4.41

4.4.5. Організація обміну даними в мультимікропроцесорній системі по інтерфейсу SPI

На сьогоднішній день при побудові автоматизованих систем управління виробничими процесами все частіше постають завдання, які не можуть бути ефективно вирішені системами з одним мікроконтролером або мікропроцесором, що зумовлює зацікавленість замовників у створенні багатопроцесорних систем з розподілом виконуваних функцій. Найпривабливішими для виробництва залишаються мікроконтролерні мережі типу “ведучий – ведений”. Більшість таких мереж мають відкритий протокол як на фізичному, так і на канальному рівні, відрізняються простотою його реалізації, і, що саме головне, дані протоколи легко адаптуються до різних застосувань за рахунок наявності в подібних системах властивостей нарощуваності та адаптованості.

Інтерфейс SPI часто використовується для обміну даними між мікроконтролерами. Структурна схема типового застосування представлена на рис. 4.42.

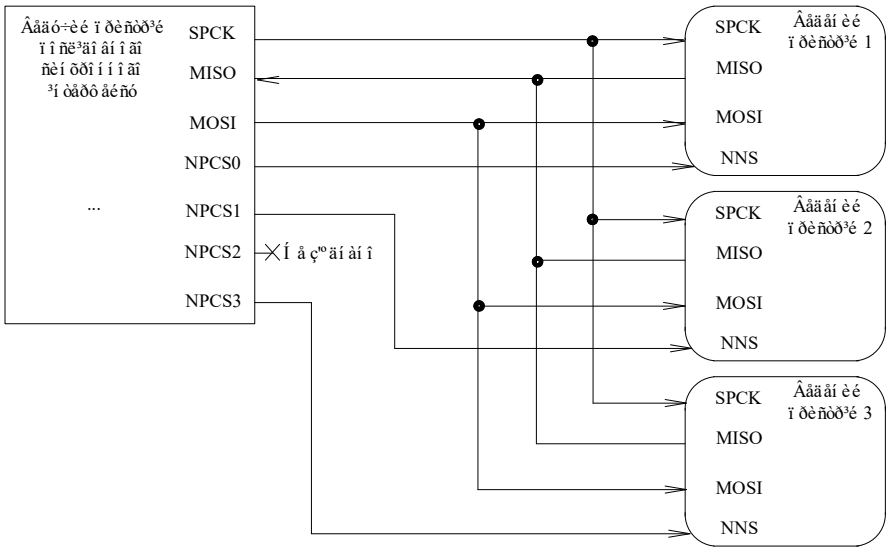


Рис. 4.42.

На рис. 4.43 представлена схема сполучення 2-х мікроконтролерів AT90S8535 по інтерфейсу SPI.

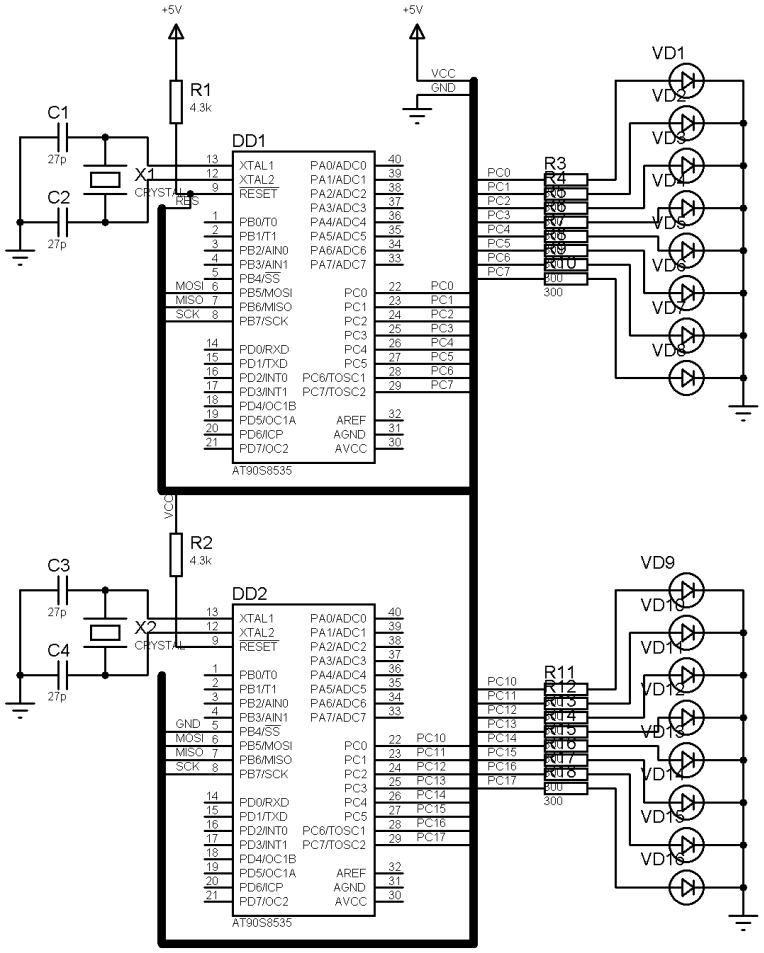


Рис. 4.43.

В наступному прикладі виконується розробка програм для ведучого і веденого мікроконтролерів AT90S8535, які реалізують наступний алгоритм: ведучий мікроконтролер (DD1) виконує інкремент змінної-лічильника, і кожні 200 мс передає значення цієї змінної по інтерфейсу SPI. Прийняте значення виводиться в порт С. Ведений мікроконтролер виводить у порт С прийняте по SPI значення. Прийняте число інвертується і передається назад ведучому мікроконтролеру.

Запускаємо середовище розробки ImageCraft і створюємо новий проект із назвою Master. У налаштуваннях проекту вибираємо контролер AT90S8535. Текст програми для мікроконтролера DD1, що працює в режимі ведучого, представлений нижче.

```
 #include <io8535v.h>          // підключення заголовних файлів
 #include <macros.h>
// функція ініціалізації мікроконтролера
 void init_cpu()
{
 DDRA=0x00;                    // порт А налаштований на вхід
 DDRB=0b10110000;              // лінії SS, MOSI, SCK налаштовані на вихід
 DDRC=0xFF;                    // порт С налаштований на вихід
 DDRD=0x00;                    // порт D налаштований на вхід
 SPCR=0x53;                    // налаштування SPI: SPI включений у режимі
                               // ведучого, дільник тактової частоти 128
 }

 void delay_ms(int ms)         // функція затримки
 {
  int c1, c2;                  // додаткові змінні
  for(c1=0;c1<ms;c1++)   // контролер виконує «порожню» операцію
  for(c2=0;c2<970;c2++) NOP();
 }

 void main()                   // головна функція
 {
  char counter=0;              // оголошення змінної-лічильника
  char rdata;                  // додаткова змінна
  init_cpu();                  // виклик функції ініціалізації контролера
  while(1)                     // вічний цикл
  {
   SPDR=counter;               // передача значення змінної counter
   while(!(SPSR&0x80));        // очікування завершення передачі
   rdata=SPDR;                 // зчитування з регістра даних SPI
   PORTC=rdata;                // вивід зчитаного значення в порт С
   counter++;                  // інкремент значення змінної
   delay_ms(200);              // виклик функції затримки
  }
 }
```

Після компіляції проекту буде створений файл прошивки мікроконтролера. Створюємо новий проект із назвою Slave. Текст

програми для мікроконтролера DD2, що працює в режимі веденого, представлений нижче.

```
#include <io8535v.h>          // підключення заголовних файлів
#include <macros.h>
// підпрограма обробки переривання
#pragma interrupt_handler spi_stc_isr:11
void spi_stc_isr(void)
{
char data;                    // оголошення додаткової змінної
data=SPDR;                    // зчитування прийнятого значення з регістра
                              // даних SPI
PORTC=data;                   // вивід даних у порт C
SPDR=~data;                   // запис у регістр даних інвертованого
                              // значення
}
// функція ініціалізації мікроконтролера
void init_cpu()
{
DDRA=0;                       // порт A налаштований на вхід
DDRB=0b01000000;              // вивід MISO налаштований на вихід
DDRC=0xFF;                    // порт C налаштований на вихід
DDRD=0;                       // порт D налаштований на вхід
SPCR=0xC3;                    // налаштування SPI: SPI включений в режимі
                              // веденого, дільник тактової частоти 128
                              // дозволене переривання по прийому
SEI();                        // дозвіл обробки переривань
}
void main()                   // головна функція
{
char c;                  // оголошення додаткової змінної
init_cpu();                   // виклик функції ініціалізації контролера
while(1);                     // вічний цикл
}
```

4.4.6. Сполучення двох контролерів Arduino по інтерфейсу SPI

Схема сполучення двох контролерів Arduino по інтерфейсу SPI представлена на рис. 4.44.

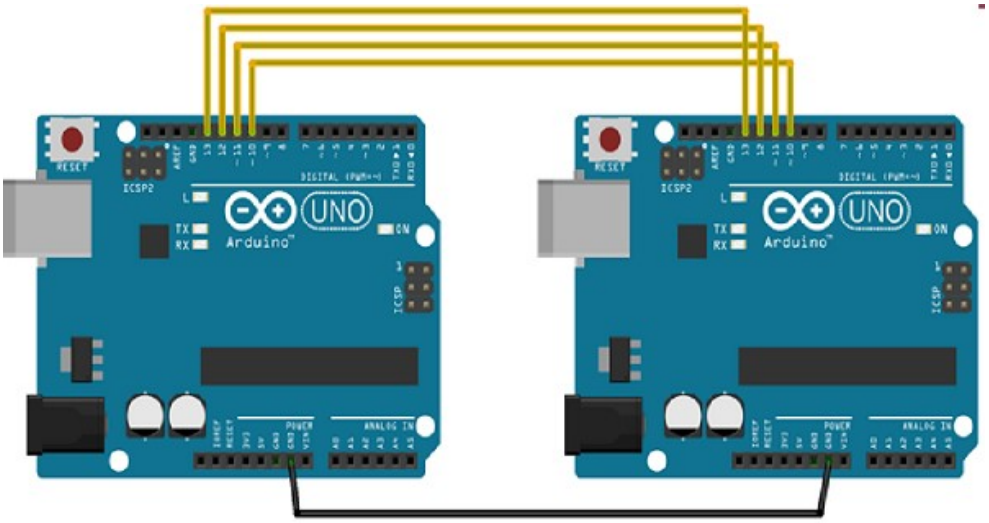


Рис. 4.44.

Один з контролерів виступає в ролі ведучого, а другий – в ролі веденого. Ведучий контролер відправляє по інтерфейсу SPI текстовий рядок «Hello, world!», а ведений контролер відображає прийняті дані у вікні терміналу. Схема моделі в Proteus для перевірки роботи пристрою представлена на рис. 4.45. За допомогою Віртуальних терміналів, які підключені до виходів USART кожного з контролерів Arduino, в режимі реального часу можна спостерігати дані, що відправляються ведучим контролером і приймаються веденим контролером. Як видно зі схеми, контролери з'єднані між собою однойменними виводами інтерфейсу SPI. При тестуванні на реальних пристроях не слід забувати про необхідність з'єднання загального провідника обох контролерів.

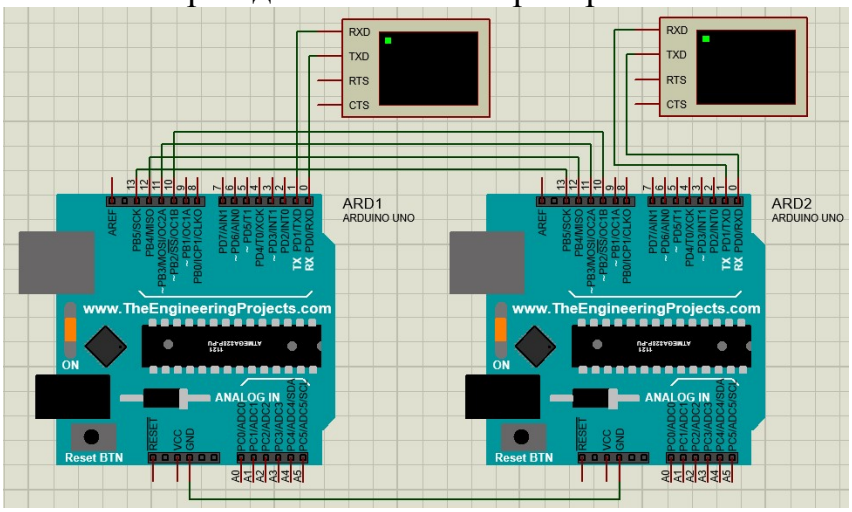


Рис. 4.45.

Текст програми для ведучого (Master) контролера представлено нижче.

```
// підключення бібліотеки функцій для роботи з інтерфейсом SPI
#include <SPI.h>
void setup (void)                    // функція ініціалізації контролера
{
```

```
  Serial.begin(9600); // налаштування швидкості передавання даних 9600 біт/с
  digitalWrite(SS, HIGH);    // ведений пристрій не обрано
  SPI.begin ();          // початок обміну даними
  SPI.setClockDivider(SPI_CLOCK_DIV8);         // дільник частоти дорівнює 8
}
// в циклі виконується відправлення даних у ведений контролер
void loop (void)
{
  char c;         // оголошення додаткової змінної
  // вибір веденого пристрою (на лінії SS низький рівень)
  digitalWrite(SS, LOW);
  // відправлення тестового рядка
  for (const char * p = "Hello, world!\r" ; c = *p; p++) {
    SPI.transfer (c);    // передавання символу по інтерфейсу SPI
    Serial.print(c); // передавання символу по інтерфейсу USART
  }
  // ведений пристрій не обрано (на лінії SS високий рівень)
  digitalWrite(SS, HIGH);
  delay(2000);// часова затримка 2 секунди
}
```

Текст програми для веденого (Slave) контролера представлено нижче.

```
          // підключення бібліотеки функцій для роботи з інтерфейсом SPI
#include <SPI.h>
char buff [50];                    // масив з 50 байт
volatile byte indx;                // оголошення додаткової змінної
volatile boolean process;          // оголошення змінної логічного типу

void setup (void)                  // функція ініціалізації контролера
{
  Serial.begin (9600);             // налаштування швидкості передавання даних
  pinMode(MISO, OUTPUT);           // налаштування виводу MISO на вихід
  SPCR |= _BV(SPE);                // переведення SPI в режим веденого
  indx = 0;                        // буфер пустий
  process = false;
  SPI.attachInterrupt();           // увімкнути обробку переривань
}

ISR (SPI_STC_vect)                 // функція-обробник переривання SPI
{
  byte c = SPDR;                    // зчитати байт з регістру даних SPI
  if (indx < sizeof buff)
  {
    buff [indx++] = c;             // зберегти дані в масиві за вказаним індексом
    if (c == '\r')                 // перевірити чи отримано символ кінця рядка
    process = true;                // записати значення true в змінну-прапорець
  }
}

void loop (void)                   // в циклі відбувається перевірка, чи прийняті дані
{
```

```
  if (process)                  // якщо отримано дані по SPI
{
    process = false;            // записати значення false в змінну-прапорець
    Serial.println (buff);      // відправити вміст масиву по USART в комп’ютер
    indx= 0;                    // обнуління індексу
  }
}
```

Відкомпілювавши обидві програми, та вказавши у налаштуваннях контролера Arduino шлях до файлу з розширенням .hex, можна перевірити роботу схеми. Після запуску процесу моделювання у вікнах Віртуальних терміналів можна побачити рядок «Hello, world!», що свідчить про правильність роботи обох програм.

Завдання до розділу 4

1. Розробити систему підтримання заданої температури в приміщенні. Якщо температура падає нижче заданого рівня, вмикається нагрівач, якщо перевищує заданий рівень, вмикається кондиціонер. Передбачити можливість налаштування нижньої та верхньої меж температури.
2. В мікросхему пам'яті необхідно записувати значення, зчитане з АЦП, з інтервалом 1 с. За запитом дані зчитуються з пам'яті і передаються в комп'ютер. Після цього пам'ять очищається.
3. Розробити схему пристрою та написати програму, яка зчитує значення температури по шині I2C з мікросхеми DS1621, і виводить її на семисегментні індикатори.
4. Написати програму, що працює з мікросхемою пам'яті. У комірку пам'яті записуються дані, потім зчитуються і виводяться на семисегментні індикатори. Реалізувати алгоритм двома способами: використовуючи вбудований інтерфейс TWI та інтерфейс, реалізований програмно.
5. Написати програму – годинник. В якості початкового значення часу прийняти поточний час. Час має бути представлено у форматі: ГГ: ХХ: СС: МС і виводитися у верхньому рядку алфавітно-цифрового рідкокристалічного індикатора.
6. Розробити принципову схему сполучення двох мікроконтролерів по інтерфейсу SPI. На вхід АЦП першого мікроконтролера, що є головним, подається напруга від 0 до 5В. До другого мікроконтролера підключені чотири семисегментних індикатори. Написати програми для обох мікроконтролерів, що виконують наступну задачу: перший мікроконтролер зчитує дані з АЦП та передає їх другому мікроконтролеру, який відображає отримане значення на семисегментних індикаторах.
7. Розробити програму, що зчитує значення з АЦП із частотою дискретизації 100 Гц, і записує дані в карту пам'яті SD/MMC. При

отриманні запиту від комп'ютера дані зчитуються з карти пам'яті і передаються в комп'ютер.

8. Розробити програвач звукових файлів. Програма записує на карту пам'яті дані з WAV-файлу, а потім виконується програвання цього файлу (дані зчитуються з карти пам'яті і виводяться на ЦАП).
9. Розробити мережу, що складається з 3-х мікроконтролерів. Обмін даними між мікроконтролерами відбувається по інтерфейсу SPI. До порту першого мікроконтролера, який в мережі є головний, підключені тактові кнопки. На двох інших мікроконтролерах, які в мережі є веденими, відображати стани кнопок – на першому МК молодшу тетраду, на другому МК – старшу тетраду.
10. Розробити мережу MicroLAN, яка складається з чотирьох датчиків температури DS18B20. Написати програму для мікроконтролера, задачею якого є зчитування значення температури з кожного датчика та вивід температури на алфавітно-цифровий РКІ.
11. Використовуючи мікросхему аналого-цифрового перетворювача DS2450 з інтерфейсом 1-Wire розробити чотирьохканальний вольтметр з виведенням інформації на семисегментні індикатори. Номер каналу вибирається шляхом натискання тактової кнопки, що підключена до мікроконтролера. Документацію на мікросхему АЦП можна знайти в мережі Інтернет за адресою: http://www.maxim-ic.com.
12. Використовуючи таймер реального часу DS2415 з інтерфейсом 1-Wire, розробити принципову схему електронного годинника. Написати програму для мікроконтролера, що зчитує значення часу з мікросхеми і виводить його на екран алфавітно-цифрового РКІ. Документацію на мікросхему таймера реального часу можна знайти в мережі Інтернет за адресою: http://www.maxim-ic.com.
13. Розробити мікропроцесорну мережу, що складається з 2-х мікроконтролерів. Обмін даними між мікроконтролерами виконується з використанням інтерфейсу 1-Wire, який потрібно реалізувати програмно.
14. Розробити систему контролю температури в приміщенні. В якості датчика температури використовувати мікросхему DS1621 або DS18B20. Система повинна підключатися до комп’ютера по інтерфейсу USB. Розробити інтерфейс програми для комп’ютера для спостереження за температурою.

РОЗДІЛ 5. АПАРАТНО-ПРОГРАМНІ ЗАСОБИ СПОЛУЧЕННЯ МІКРОПРОЦЕСОРНИХ СИСТЕМ З КОМП'ЮТЕРОМ

5.1. Послідовний інтерфейс RS-232

Інтерфейс RS-232C призначений для підключення апаратури передавання або прийому даних (АПД) до апаратури каналів даних (АКД). У ролі АПД може виступати комп'ютер, принтер, плоттер та інше периферійне обладнання. Цим пристроям відповідає абревіатура DTE – Data Terminal Equipment. У ролі АКД звичайно виступає модем. Цим пристроям відповідає абревіатура DCE – Data Communication Equipment. Кінцевою метою підключення є з'єднання двох пристроїв. Інтерфейс дозволяє виключити канал зв'язку разом з парою пристроїв DTE (модемів), шляхом з'єднання пристроїв безпосередньо за допомогою нуль-модемного кабелю.

Стандарт описує керуючі сигнали інтерфейсу, пересилки даних, електричний інтерфейс і типи з'єднувачів. Стандарт описує асинхронний і синхронний режими обміну, але COM-порти підтримують тільки асинхронний режим. Розташування контактів 9-вивідного COM-порту наведено на рис. 5.1. Табл. 5.1 пояснює призначення його контактів.

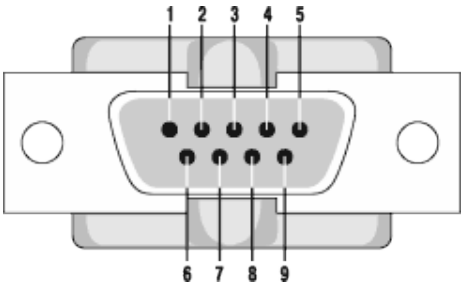


Рис. 5.1.

Таблиця 5.1.

Контакт	Сигнал
1	Детектування несучої (Data Carrier Detect, DCD)
2	Прийом даних (Received Data, Rx)
3	Передача даних (Transmitted Data, Tx)
4	Готовність термінала (Data Terminal Ready, DTR)
5	Сигнальна «земля» (Signal Ground)
6	Готовність даних (Data Set Ready, DSR)
7	Запит на передачу (Request to Send, RTS)
8	Готовність до передачі (Clear to Send, CTS)
9	Індикатор виклику (Ring Indicator, RI)

Послідовний або комунікаційний порт комп'ютера є промисловим стандартом для підключення до комп'ютера модему, а також може використовуватися для багатьох інших цілей – підключення другого комп'ютера для обміну файлами по нуль-модемному кабелю або будь-яких

інших «нестандартних» електронних пристроїв, яким необхідно обмінюватися даними з персональним комп'ютером. COM-порт часто використовують як канал для обміну інформацією в промислових системах керування та збору даних – наприклад, для обміну даними з вимірювальними мікроконтролерами. Більшість провідних фірм-виробників електронних приладів і пристроїв серійно випускають численні вимірювальні та технологічні пристрої, що підключаються до комп'ютера через послідовний інтерфейс.

5.1.1. Загальна характеристика інтерфейсу RS-232

Послідовний інтерфейс для передачі сигналу в одну сторону використовує одну сигнальну лінію, по якій інформаційні біти передаються один за одним послідовно. Такий спосіб передачі визначає назву інтерфейсу і порту, що його реалізує. Ці назви відповідають англійським термінам Serial Interface і Serial Port (іноді в невдалому перекладі їх називають «серійними», що звучить досить дивно). Послідовна передача даних може здійснюватися як в асинхронному, так і синхронному режимах.

При асинхронній передачі кожному байту передує стартовий біт, що сигналізує приймачу про початок чергової посилки, за яким ідуть біти даних і, можливо, біт паритету (контролю парності). Завершує посилку стоповий біт, що гарантує певний інтервал між сусідніми посилками (рис. 5.2). Стартовий біт наступного посланого байту може бути відправлений у будь-який момент після закінчення стопового біту, тобто між передачами можливі паузи довільної тривалості.



Рис. 5.2.

Стартовий біт, що має завжди строго визначене значення (лог.0), забезпечує простий механізм синхронізації приймача за сигналом від передавача. Передбачається, що приймач і передавач працюють на одній швидкості обміну, що виміряється у кількості переданих біт за секунду. Внутрішній генератор синхронізації приймача використовує лічильник-дільник опорної частоти, який скидається у момент прийому початку стартового біта. Цей лічильник генерує внутрішні строби, за якими приймач фіксує наступні прийняті біти. В ідеальному випадку ці строби розташовуються в середині бітових інтервалів, що забезпечує можливість прийому даних і при деякій неузгодженості швидкостей приймача та

передавача. Неважко помітити, що при передачі 8 біт даних, одного контрольного та одного стопового біта гранично припустима неузгодженість швидкостей, при якому дані будуть вірно розпізнані, не може перевищувати 5%. З урахуванням фазових перекручувань (пологих фронтів сигналу) і дискретності роботи внутрішнього лічильника синхронізації, реально припустиме менше відхилення частот. Чим менше коефіцієнт ділення опорної частоти внутрішнього генератора (чим вище швидкість передачі), тим більша похибка прив'язки стробів до середини бітового інтервалу, і вимоги до узгодженості частот більш строгі. Також, чим вище частота передачі, тим більший вплив робить перекручування фронтів на фазу прийнятого сигналу. Така дія цих двох факторів призводить до підвищення вимог узгодженості частот приймача і передавача з ростом частоти обміну.

Формат асинхронної посилки дозволяє виявляти можливі помилки передачі:

– якщо прийнято перепад, що сигналізує про початок посилки, а за стробом стартового біту зафіксований рівень логічної одиниці, стартовий біт вважається помилковим і приймач знову переходить у стан очікування. Про цю помилку приймач може і не повідомляти.

– якщо впродовж часу, відведеного під стоповий біт(и), виявлений рівень логічної одиниці, фіксується помилка стопового біта.

– якщо застосовується контроль парності (паритету), то після посилки біт даних, перед стоповим бітом передається контрольний біт. Цей біт доповнює кількість одиничних біт даних до парного або непарного, залежно від прийнятої угоди. Прийом байта з невірним значенням контрольного біту при включеному контролі парності, призводить до фіксації помилки прийнятих даних.

Контроль формату дозволяє виявляти обрив лінії: при цьому звичайно приймається логічний нуль, що спочатку трактується як стартовий біт і нульові біти даних, але потім спрацює контроль стопового біта.

Для асинхронного режиму прийнятий ряд стандартних швидкостей обміну: 50, 75, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600 і 115200 біт/с. Іноді замість одиниці вимірювання «біт/с» використовують «бод», але в цьому випадку, при розгляді двійкових переданих сигналів, це некоректно. У бодах прийнято вимірювати частоту зміни стану лінії, а при недвійковому способі кодування (широко застосовуваному в сучасних модемах) у тому ж самому каналі зв'язку, швидкості передачі біт (біт/с) і зміни сигналу (бод) можуть відрізнятися в кілька разів.

Кількість біт даних може становити 5, 6, 7 або 8 (5- і 6-бітні формати мало поширені). Кількість стопових біт може бути 1, 1,5 і 2 (під «півтора біта» мається на увазі тільки тривалість стопового інтервалу).

Асинхронний обмін у персональному комп’ютері реалізується за допомогою COM-порту з використанням протоколу RS-232C.

Синхронний режим передачі припускає постійну активність каналу зв'язку. Посилка починається із синхробайту, за яким відразу ж передається потік інформаційних біт. Якщо в передавача немає даних для передачі, він заповнює паузу безперервною посилкою байтів синхронізації. Очевидно, що при передачі великих масивів даних, витрати на синхронізацію в даному режимі обміну будуть нижче, ніж в асинхронному. Однак у синхронному режимі необхідна зовнішня синхронізація приймача з передавачем, оскільки навіть мале відхилення частот призведе до швидкого накопичення помилки і спотворенню прийнятих даних. Зовнішня синхронізація можлива або за допомогою окремої лінії для передачі сигналу синхронізації, або шляхом використання самосинхронізуючого кодування даних (наприклад, манчестерський або код NRZ), при якому на прийомній стороні із прийнятого сигналу можуть бути виділені імпульси синхронізації. В кожному разі, синхронний режим вимагає або дорогих ліній зв'язку, або дорогого обладнання (а може, і того, і іншого). Існують спеціальні плати – адаптери SDLC (досить дорогі), які підтримують синхронний режим обміну. Вони використовуються в основному для зв'язку з мейнфреймами (mainframes) IBM і зараз мало розповсюджені (їх витиснули менш дорогі і більш ефективні засоби комунікацій).

Послідовний інтерфейс на фізичному рівні може мати різні реалізації, які розрізняються способами передачі електричних сигналів. Існує ряд родинних міжнародних стандартів: RS-232C, RS-423A, RS-422A и RS-485. На рис. 5.3 приводяться схеми з'єднання приймачів і передавачів і показані їхні обмеження на довжину лінії (L) і максимальну швидкість передачі даних (V).

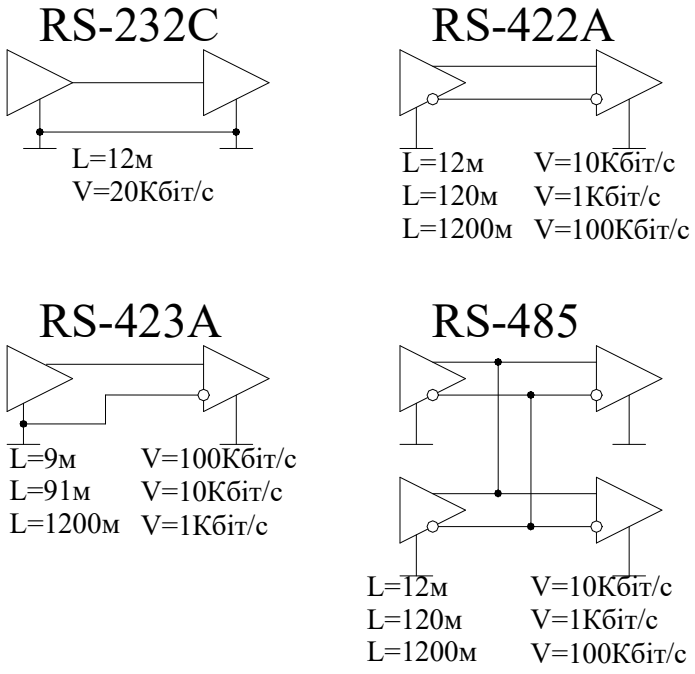


Рис. 5.3.

Несиметричні лінії інтерфейсів RS-232C и RS-423A мають найнижчу захищеність від синфазної перешкоди, хоча диференціальний вхід приймача RS-423A небагато зм'якшує ситуацію. Кращі параметри має двохточковий інтерфейс RS-422A и його магістральний (шинний) аналог RS-485, які працюють на симетричних лініях зв'язку. У них для передачі кожного сигналу використовуються диференціальні сигнали з окремою (витою) парою проводів. Існують відносно нескладні перетворювачі сигналів для узгодження всіх цих інтерфейсів. На рис. 5.4 представлена повна схема з'єднання по RS232, а на рис. 5.5 – схема з'єднання по RS232 нуль-модемним кабелем.

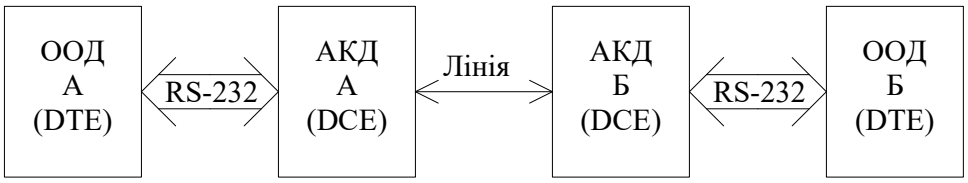


Рис. 5.4.

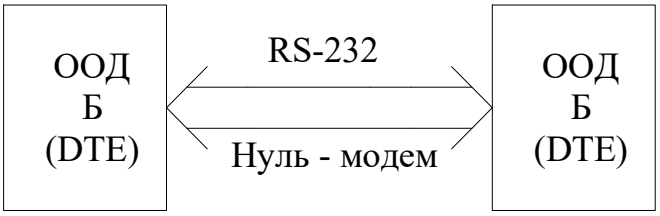


Рис. 5.5.

Стандарт RS-232C використовує несиметричні передавачі і приймачі – сигнал передається по загальному проводу – схемній землі (симетричні диференціальні сигнали використовуються в інших інтерфейсах, наприклад RS-422). Інтерфейс не забезпечує гальванічної розв'язки пристроїв. Логічній одиниці відповідає рівень напруги на вході приймача в діапазоні -12...-3 В. Для ліній керуючих сигналів цей стан називається ON («включене»), для ліній послідовних даних називається MARK («відзначене»). Логічному нулю відповідає напруга в діапазоні +3...+12 В. Для ліній керуючих сигналів цей стан називається OFF («виключене»), для ліній послідовних даних називається SPACE. Між рівнями -3...+3 В є зона нечутливості, що забезпечує гістерезис приймача: стан лінії буде вважатися зміненим тільки після перетинання відповідного порогу (рис.3.6). Рівні сигналів на виходах передавачів повинні перебувати в діапазонах -12...-5В и +5...+12В для передачі одиниці та нуля відповідно. Різниця потенціалів між схемними землями (SG) пристроїв які з'єднуються, повинна бути менше 2В. При більш високій різниці потенціалів можливий невірний прийом сигналів. Інтерфейс допускає наявність захисного заземлення для пристроїв, що з'єднуються, якщо вони обидва живляться від мережі змінного струму і мають мережні фільтри.

Підключення та відключення інтерфейсних кабелів пристроїв з автономним живленням повинне виконуватися при відключеному живлення. У противному випадку різниця не вирівняних потенціалів пристроїв у момент комутації (приєднанні або від'єднанні роз'ємів) може виявитися прикладеною до вихідних або вхідних ланцюгів інтерфейсу і вивести мікросхему з ладу. Діаграма, представлена на рис. 5.6, демонструє приймання сигналів по інтерфейсу RS-232C.

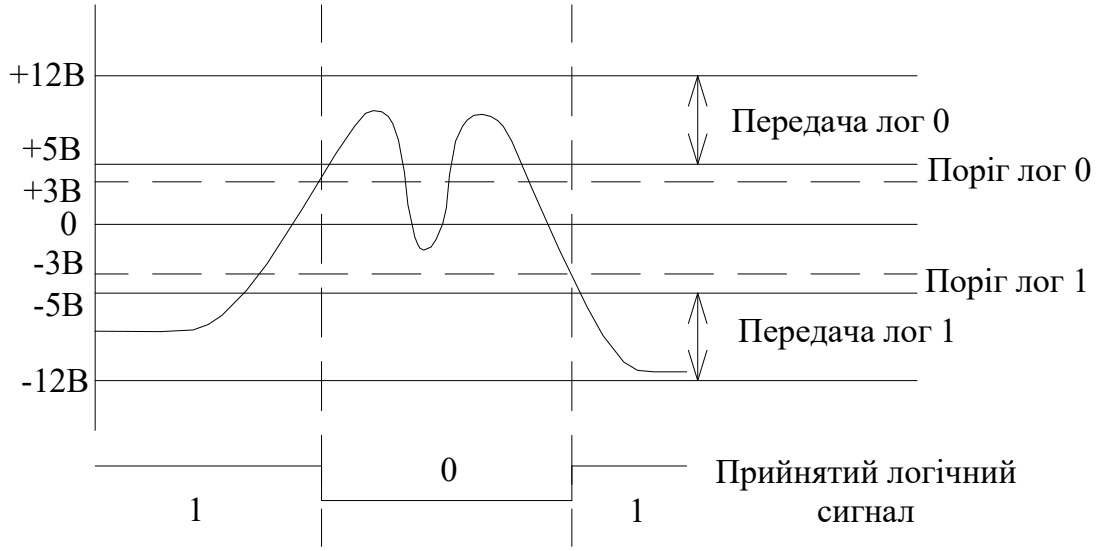
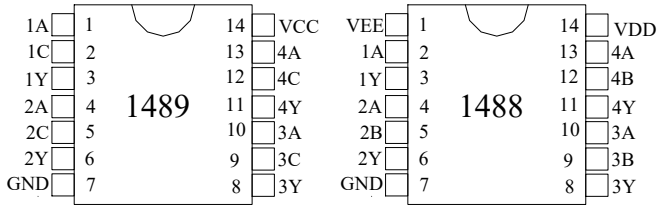


Рис. 5.6.

Мікросхеми формувачів сигналів RS-232C представлені на рис. 5.7: а - приймач; б - передавач; с - таблиця станів виходів передавача



a b

A	B*	Y
0	0	VDD
0	1	VDD
1	0	VDD
1	1	VEE

c

Рис. 5.7.

Для інтерфейсу RS-232C спеціально випускаються буферні мікросхеми приймачів (з гістерезисом) і передавачів двохполярного сигналу. При недотриманні правил заземлення і комутації включених пристроїв вони безумовно є першими жертвами. Іноді їх встановлюють в «панельках», що значно полегшує заміну. Розташування виводів популярних мікросхем формувачів сигналів RS-232C приводиться на рис. 5.7. Часто буферні схеми входять прямо до складу інтерфейсних БІС. Це робить пристрій дешевим, заощаджує місце на платі, але у випадку поломки звичайно обертається більшими фінансовими втратами. Вихід з ладу інтерфейсних мікросхем замиканням сигнальних ланцюгів

малоймовірний, оскільки струм короткого замикання передавачів звичайно обмежений на рівні 20 мА.

Стандарт RS-232C регламентує типи застосовуваних роз'ємів, які забезпечують високий рівень сумісності апаратури різних виробників.

На пристроях DTE (у тому числі, і на COM-портах) прийнято встановлювати штекери DB25-P, або більш компактний варіант DB9-P.

Дев'ятиштиркові роз'єми не мають контактів для додаткових сигналів, необхідних для синхронного режиму (у більшості 25-штиркових роз'ємів ці контакти не використовуються).

На пристроях DCE (модемах) встановлюють розетки DB25-S або DB-9S. Це правило допускає, що роз'єми DCE можуть підключатися до роз'ємів DTE безпосередньо (якщо дозволяє геометрія конструктиву) або через перехідні «прямі» кабелі з розеткою та вилкою, у яких контакти з'єднуються «один в один». Перехідні кабелі можуть бути і перехідниками з 9 на 25–штиркові роз'єми (рис. 5.8).

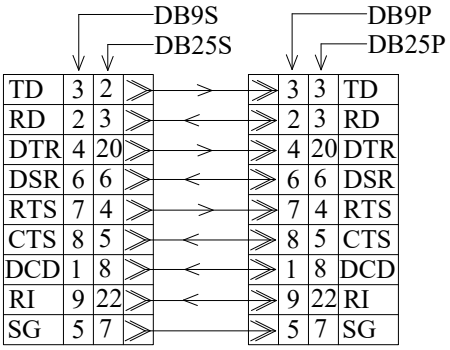


Рис. 5.8.

Якщо пристрої з'єднується без модемів, то роз'єми пристроїв з'єднуються між собою нуль - модемним кабелем, що має на обох кінцях роз'єми, контакти яких з'єднуються за однією зі схем, наведених на рис. 5.9.

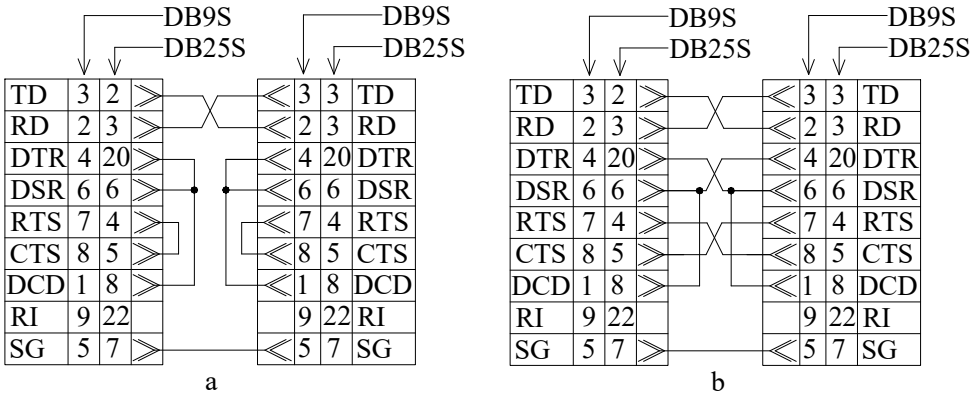


Рис. 5.9.

Якщо на якому-небудь пристрої DTE (принтер, плоттер) встановлена розетка – це майже стовідсоткова ознака того, що до іншого пристрою (комп'ютеру) воно повинне підключатися прямим кабелем,

аналогічним кабелем підключення модему. Розетка встановлюється звичайно на ті пристрої, у яких підключення через модем не передбачене.

У табл. 5.2 приводиться призначення контактів роз'ємів COM-портів (і будь-яких інших пристроїв DTE).

Таблиця 5.2.

Позначення ланцюга		Контакт роз'єму		Провід шлейфа виносного роз'єму PC				Напрямок	Назва ланцюга
RS232	Стик 2	DB25S	UB9S	1*	У	у	4*	1/0	
PG	101	1	–	(10)	(10)	(10)	1	–	Protect Ground – Захистна земля
TD	103	2	3	3	5	3	3	0	Transmit Data – Передані дані
RD	104	3	9	2	3	4	5	1	Receive Data – Прийняті дані
RTS	105	4	7	7	4	8	7	0	Request To Send – Запит на передачу
CTS	100	5	8	8	0	7	9	1	Clear To Send – Готовність модему до прийому даних для передачі
DSR	107	6	6	6	2	9	11	1	Data Set Ready – Готовність модему до роботи
SG	102	7	5	5	9	1	13		Signal Ground – Схемна земля
DCD	109	8	1	1	1	5	15		Data Carrier Detected – Несуча виявлена
DTR	108/2	20	4	4	7	2	14	0	Data Terminal Ready – готовність термінала (PC) до роботи
RI	125	22	9	9	8	6	18	1	Ring Indicator – Індикатор виклику

1* – шлейф 8-бітних мультикарт.

V - шлейф 16-бітних мультикарт і портів на системних платах.

З* – варіант шлейфу портів на системних платах.

4* – широкий шлейф до 25-контактному роз'єму.

Підмножина сигналів RS-232C, які належать до асинхронного режиму, розглянемо з погляду COM - порту PC, що є по термінології RS-232C терміналом даних (DTE). Варто пам'ятати, що активному стану сигналу («включене») відповідає негативний потенціал (нижче –3В) сигналу інтерфейсу, а стану «виключено» і логічному нулю - позитивний (вище +3В). Сигнали інтерфейсу мають наступне призначення:

PG – Захисна земля, з'єднується з корпусом пристрою та екраном кабелю.

SG – Сигнальна (схемна) земля, відносно якої діють рівні сигналів.

TD – Послідовні дані – вихід передавача.

RD – Послідовні дані – вхід приймача.

RT – Вихід запиту передачі даних: стан «включено» повідомляє модем про наявність у термінала даних для передачі. У напівдуплексному

режимі використовується для керування напрямком - стан «включено» є сигналом модему на перемикання в режим передачі.

CT – Вхід дозволу термінала передавати дані. Стан «виключений» апаратно забороняє передачу даних. Сигнал використовується для апаратного керування потоками даних.

DT – Вихід сигналу готовності терміналу до обміну даними. Стан «включений» підтримує канал, що комутується, у стані з'єднання.

DSR – Вхід сигналу готовності від пристроїв передачі даних (модем у робочому режимі підключений до каналу і закінчив дії за узгодженням із пристроями на протилежному кінці каналу).

DC – Вхід сигналу виявлення вилученого модему.

RI – Вхід індикатора виклику (дзвінка). У каналі, що комутується, цим сигналом модем сигналізує про прийняття виклику.

5.1.2. Формати передачі даних та режими зв'язку

Для керування потоком даних (Flow Control) можуть використовуватися два варіанти протоколу – апаратний і програмний. Іноді керування потоком плутають із квотуванням, але це різні методи досягнення однієї мети – узгодження темпу передачі та прийому. Квотування (Handshaking) має на увазі посилку повідомлення про одержання елементу, у той час як керування потоком допускає посилку повідомлення про неможливість наступного прийому даних.

Апаратний протокол керування потоком RTS/CT (Hardware Flow Control) використовує сигнал CT, що дозволяє зупинити передачу даних, якщо приймач не готовий до їхнього прийому. Роботу цього протоколу ілюструє рис. 5.10. Передавач «випускає» черговий байт тільки при включеному стані лінії CT. Байт, що вже почав передаватися, затримати сигналом CT неможливо (це гарантує цілісність посилки). Апаратний протокол забезпечує швидку реакцію передавача на стан приймача. Звичайно мікросхеми асинхронних приймачів-передавачів мають не менш двох регістрів у прийомної частини. Це дозволяє реалізувати обмін з апаратним протоколом без втрати даних, не прибігаючи до програмної буферизації.

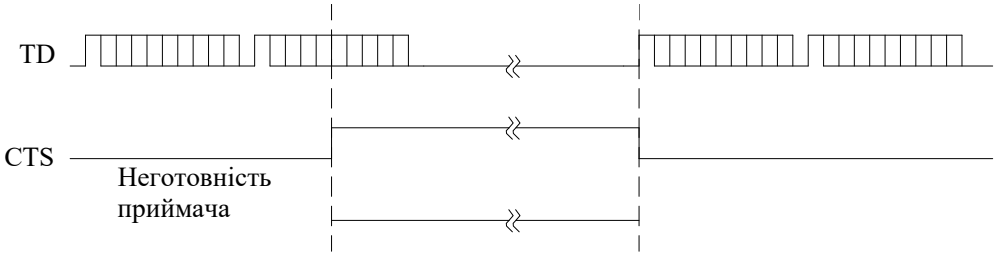


Рис. 5.10.

Апаратний протокол зручно використовувати при підключенні принтерів і плотерів, якщо вони підтримують цей режим (рис. 5.11). При безпосередньому (без модемів) з'єднанні двох комп'ютерів апаратний протокол вимагає перехресного з'єднання ліній RT - CT.

Якщо апаратний протокол не використовується, то при безпосередньому з'єднанні в терміналі, що передає, повинне бути забезпечене стан «включено» на лінії CT (звичайно з'єднанням власних ліній RT - CT). У противному випадку передавач буде мовчати.

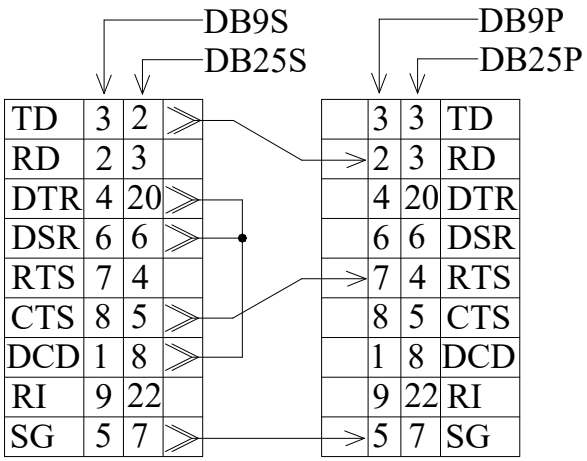


Рис. 5.11.

Програмний протокол керування потоком XO/XOFF допускає наявність двонапрямленого каналу передачі даних. Працює він у такий спосіб: якщо пристрій, що приймає дані, виявляє причини, через які воно не може їх далі приймати, воно по зворотному послідовному каналу посилає XOFF (13h). Протилежний пристрій, прийнявши цей символ, припиняє передачу. Далі, коли приймаючий пристрій знову стає готовим до прийому даних, воно посилає символ XON (11h), прийнявши який протилежний пристрій відновлює передачу. Час реакції передавача на зміну стану приймача в порівнянні з апаратним протоколом збільшується принаймні на час передачі символу (XON або XOFF) плюс час реакції програми передавача на прийом символу (рис. 5.12). Із цього витікає, що дані без втрат можуть прийматися тільки приймачем, що має додатковий буфер прийнятих даних. Схема нуль-модемного кабелю представлена на рис. 5.13.

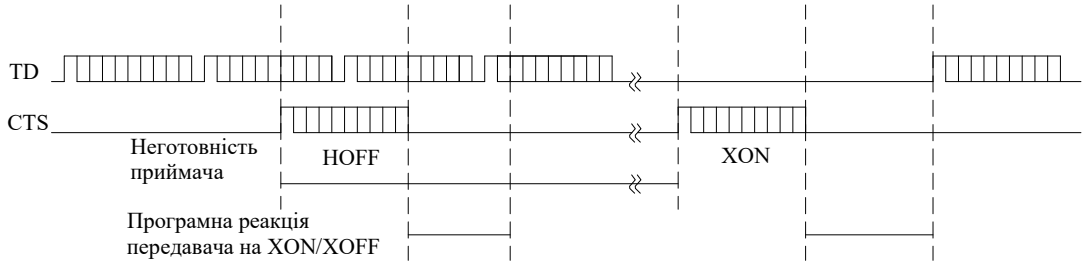


Рис. 5.12.

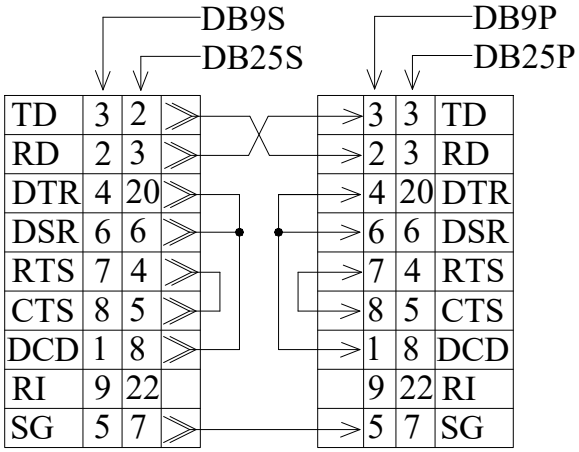


Рис. 5.13.

Перевага програмного протоколу при безпосередньому з'єднанні пристроїв складається у відсутності необхідності передачі керуючих сигналів інтерфейсу – мінімальний кабель для двостороннього обміну може мати тільки 3 провідники (див. рис. 5.13). Недоліком, крім вимоги наявності буфера і більшого часу реакції (що знижує і загальну продуктивність каналу через очікування проходження сигналу XON), є складність реалізації повнодуплексного режиму обміну. У цьому випадку з потоку прийнятих даних повинні виділятися (і оброблятися) символи керування потоком, що обмежує набір переданих символів.

Крім цих двох розповсюджених стандартних протоколів, що підтримуються і пристроями, і операційними системами, існують і деякі інші. Наприклад, деякі плоттери з послідовним інтерфейсом використовують програмне керування, але посилають не стандартні символи XON/XOFF, а слова (ASCII - речення). Такий обмін на рівні системної підтримки протоколу практично не підтримується (ці плоттери розраховані на прямий діалог із прикладною програмою). Кабель для підключення такого пристрою збігається з наведеним на рис. 5.13.

5.1.3. Програмування COM-портів та реалізація протоколу обміну даними по інтерфейсу RS-232

З послідовними портами в Win32 працюють як з файлами. Отже, починати треба з відкриття порту як файлу, використовуючи функцію CreateFile. У якості одного з параметрів функції передається ім'я порту (наприклад, COM1), з яким буде виконуватися робота. Спроба створити файл із таким ім'ям призводить до перенаправлення інформації на відповідний порт. Прототип функції має такий вигляд:

```
HANDLE CreateFile(
LPCTSTR lpFileName,
DWORD dwDesiredAccess,
DWORD dwShareMode,
LPSECURITY_ATTRIBUTESlpSecurityAttributes,
```

```
    DWORD dwCreationDispostion,
    DWORD dwFlagsAndAttributes,
    HANDLEhTemplateFile
  );
```

Опис параметрів функції приводиться нижче.

lpFileName

У вигляді строчки передається ім'я порту, який відкривається, (наприклад, “COM1”).

dwDesiredAccess

Визначає режим доступу до порту: доступ на зчитування, запис, зчитування/запис даних із/в порт. Цей параметр може бути GENERIC_READ – зчитування, GENERIC_WRITE – запис, GENERIC_READ | GENERIC_WRITE – можливість виконання операцій зчитування і запису.

dwShareMode

Визначає режим доступу до порту між різними процесами. До порту одночасно можуть звернутися кілька процесів. Цей параметр повинен дорівнювати 0. Це означає, що після відкриття порту його не можна буде повторно відкрити доти, поки він не буде закритий програмою, яка його використовує.

lpSecurityAttributes

При роботі з портами цей параметр не використовується і повинен приймати значення NULL.

dwCreationDispostion

Задає спосіб створення файлу. Для комунікаційних портів у це поле повинне бути занесене значення OPEN_EXISTING, що вказує на те, що відкривати треба існуючий порт.

dwFlagsAndAttributes

Задає атрибути створюваного файлу, а також керує різними режимами обробки. У випадку виконання синхронних операцій зчитування/запису цей параметр приймає значення FILE_ATTRIBUTE_NORMAL.

hTemplateFile

Задається дескриптор файлу-шаблона, що у цьому випадку не використовується. Цей параметр повинен бути рівним NULL.

При успішному відкритті порту функція CreateFile поверне в основну програму його дескриптор (HANDLE). Якщо ж за якимось причинами відкрити порт не вдалося, цій змінній привласнюється значення `INVALID_HANDLE_VALUE`. Таким чином, перевірка ідентифікатора порту на його рівність `INVALID_HANDLE_VALUE` дозволяє виконати обробку

виняткових ситуацій (наприклад, відсутність порту, який відкривається, на материнській платі або його використання операційною системою).

Відкривши порт, ми одержали його у своє розпорядження. Тепер з портом може працювати тільки наша програма. Однак, перш ніж організовувати обмін даними, необхідно налаштувати порт. Це стосується тільки послідовних портів, для яких повинна бути задана швидкість обміну, параметри парності, формат даних та інше. Крім того, існує трохи специфічних для Windows параметрів. Мова йде про тайм-аути, які дозволяють контролювати як інтервал між прийнятими байтами, так і загальний час прийому повідомлення. Є можливість керувати станом сигналів керування модемом.

Послідовність кроків, які необхідно виконати в процесі роботи з портом, приводиться на рис. 5.14.



Рис. 5.14.

Основні параметри послідовного порту описуються структурою DCB:

```
typedef struct _DCB {
   DWORD DCBlength;
   DWORD BaudRate;
   DWORD fBinary:1;
   DWORD fParity:1;
   DWORD fOutxCtsFlow:1;
   DWORD fOutxDsrFlow:1;
   DWORD fDtrControl:2;
   DWORD fDsrSensitivity:1;
   DWORD fTXContinueOnXoff:1;
   DWORD fOut:1;
   DWORD fIn:1;
   DWORD fErrorChar:1;
   DWORD fNull:1;
   DWORD fRtsControl:2;
   DWORD fAbortOnError:1;
   DWORD fDummy2:17;
```

```
    WORD  wReserved;
    WORD  XonLim;
    WORD  XoffLim;
    BYTE  ByteSize;
    BYTE  Parity;
    BYTE  StopBits;
    char  XonChar;
    char  XoffChar;
    char  ErrorChar;
    char  EofChar;
    char  EvtChar;
    WORD  wReserved1;
  } DCB;
```

Ця структура містить майже всю керуючу інформацію, що у реальності розташовується в різних регістрах послідовного порту. Тепер розберемося, що означає кожне з полів найважливішої структури:

DCBlength

Задає довжину, у байтах, структури DCB. Використовується для контролю коректності структури при передачі її адресу у функцію налаштування порту.

BaudRate

Швидкість передачі даних. Можливі зазначення: CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800, CBR_9600, CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600, CBR_115200, CBR_128000, CBR_256000. Як бачимо, ці константи відповідають всім стандартним швидкостям обміну. Насправді, це поле містить числове значення швидкості передачі, а константи просто є символічними іменами. Тому можна вказувати, наприклад, і CBR_9600, і просто 9600. Однак рекомендується вказувати символічні константи.

fBinary

Включає двійковий режим обміну. Win32 не підтримує не двійковий режим, тому дане поле завжди повинне дорівнювати 1, або логічній константі TRUE (що є переважніше).

fParity

Включає режим контролю парності. Якщо це поле дорівнює TRUE, то виконується перевірка парності, при помилці в програму видається відповідний код завершення.

fOutxCtsFlow

Включає режим спостереження за сигналом CTS. Якщо це поле дорівнює TRUE і сигнал CTS скинутий, передача даних припиняється до установки сигналу CTS. Це дозволяє підключеному до комп'ютера приладу

призупинити потік переданої в нього інформації, якщо він не встигає її обробляти.

fOutxDsrFlow

Включає режим спостереження за сигналом DSR. Якщо це поле дорівнює TRUE і сигнал DSR скинутий, передача даних припиняється до установки сигналу DSR.

fDtrControl

Задає режим керування обміном для сигналу DTR. Це поле може приймати наступні значення:

DTR_CONTROL_DISABLE - Забороняє використання лінії DTR

DTR_CONTROL_ENABLE - Дозволяє використання лінії DTR

DTR_CONTROL_HANDSHAKE - Дозволяє використання механізму „рукостискання” для виходу з помилкових ситуацій. Цей режим використовується, зокрема, модемами при відновленні в ситуації втрати зв'язку.

fDsrSensitivity

Задає чутливість комунікаційного драйвера до стану лінії DSR. Якщо це поле дорівнює TRUE, то всі прийняті дані ігноруються драйвером (комунікаційний драйвер розташований в операційній системі) за винятком тих, які приймаються при встановленому сигналі DSR.

fTXContinueOnXoff

Задає, чи припиняється передача при переповненні прийомного буфера і передачі драйвером символу XoffChar. Якщо це поле дорівнює TRUE, то передача триває, незважаючи на те, що прийомний буфер містить більше XoffLim символів і близький до переповнення, а драйвер передав символ XoffChar для призупинення потоку прийнятих даних. Якщо поле дорівнює FALSE, то передача не буде продовжена доти, поки в прийомному буфері не залишиться менше XonLim символів і драйвер не передасть символ XonChar для поновлення потоку прийнятих даних. Таким чином, це поле вводить залежність між керуванням вхідними і вихідними потоками інформації.

fOut

Задає використання XON/XOFF керування потоком при передачі. Якщо це поле дорівнює TRUE, то передача зупиняється при прийманні символу XoffChar і відновляється при прийманні символу XonChar.

fIn

Задає використання XON/XOFF керування потоком при прийманні. Якщо це поле дорівнює TRUE, то драйвер передає символ XoffChar, коли в прийомному буфері перебуває більше XoffLim, і XonChar, коли в прийомному буфері залишається менш XonLim символів.

fErrorChar

Вказує на необхідність заміни символів з помилкою парністю на символ, що задається полем ErrorChar. Якщо це поле дорівнює TRUE, і поле fParity дорівнює TRUE, то виконується заміна.

fNull

Визначає дії, які виконуються при прийманні нульового байта. Якщо це поле TRUE, то нульові байти відкидаються при передачі.

fRtsControl

Задає режим керування потоком для сигналу RTS. Якщо це поле дорівнює 0, то за замовчуванням мається на увазі RTS_CONTROL_HANDSHAKE. Поле може приймати одне з наступних значень:

RTS_CONTROL_DISABLE - забороняє використання лінії RTS

RTS_CONTROL_ENABLE - дозволяє використання лінії RTS

RTS_CONTROL_HANDSHAKE – дозволяє використання RTS рукостискання. Драйвер установлює сигнал RTS коли прийомний буфер заповнений менш ніж на половину, і скидає, коли буфер заповнюється більш ніж на три чверті.

RTS_CONTROL_TOGGLE - Задає, що сигнал RTS установлений, коли є дані для передачі. Коли всі символи з передавального буфера передані, сигнал скидається.

fAbortOnError

Задає ігнорування всіх операцій зчитування/запису при виникненні помилки. Якщо це поле дорівнює TRUE, драйвер припиняє всі операції зчитування/запису для порту при виникненні помилки. Продовжувати працювати з портом можна буде тільки після усунення причини помилки і виклику функції ClearCommError.

fDummy2

Зарезервовано і не використовується.

wReserved

Не використовується, повинне бути встановлене в 0.

XonLim

Задає мінімальне число символів у прийомному буфері перед посилкою символу XON.

XoffLim

Визначає максимальну кількість байт у прийомному буфері перед посилкою символу XOFF. Максимально припустима кількість байт у

буфері обчислюється вирахуванням даного значення з розміру прийомного буфера в байтах.

ByteSize
Визначає число інформаційних біт у переданих і прийнятих байтах.

Parity
Визначає вибір схеми контролю парності. Дане поле повинне містити одне з наступних значень:
EVENPARITY – доповнення до парності
MARKPARITY – біт парності, завжди 1
NOPARITY – біт парності відсутній
ODDPARITY – доповнення до непарності
SPACEPARITY – біт парності завжди 0
StopBits
Задає кількість стопових біт. Поле може приймати наступні значення:
ONESTOPBIT – один стоповий біт
ONE5STOPBIT – півтора стопових біта
TWOSTOPBIT – два стопових біти

XonChar
Задає символ XON використовується як для прийому, так і для передачі.

XoffChar
Задає символ XOFF використовується як для прийому, так і для передачі.

ErrorChar
Задає символ, що використовується для заміни символів з помилковою парністю.

EofChar
Задає символ, що використовується для сигналізації про кінець даних.

EvtChar
Задає символ, що використовується для сигналізації про подію.

wReserved1
Зарезервовано і не використовується.

Оскільки поля структури DCB використовуються для конфігурування мікросхем портів, на них накладаються деякі обмеження. Розмір байта повинен бути 5, 6, 7 або 8 біт. Комбінація з п'яти бітного слова і двох стопових біт є неприпустимою. Так само як і комбінація із шести, семи або восьми бітного слова і півтора стопових біт.

Розглянута структура DCB найбільша із всіх, що використовуються для налаштування послідовних портів. Але вона і найважливіша. Заповнення всіх полів цієї структури може викликати ускладнення, тому що треба дуже чітко представляти, як працює послідовний порт. Тому установку полів власноруч можна порекомендувати досвідченим програмістам. Один зі способів заповнення структури – скористатися функцією BuildCommDCB, яка дозволяє заповнити поля структури DCB на основі строчки, по синтаксису аналогічному строчці команди mode. Прототип цієї функції виглядає в такий спосіб:

```
BOOL BuildCommDCB(LPCTSTR lpDef, LPDCB lpDCB);
```

Функція має всього два параметри:

lpDef

Вказівник на строку з конфігураційною інформацією у форматі команди mode. Наприклад, наступна строчка задає швидкість 1200, без парності, 8 біт даних і 1 стоповий біт:

```
baud=1200 parity=N data=8 stop=1
```

lpDCB

Вказівник на структуру DCB, що заповнюється. При цьому структура повинна бути вже створена і заповнена нулями, крім поля DCBlength, в якому повинне міститися коректне значення. Можливо так само використання вже заповненої структури DCB, наприклад отриманої викликом однієї з функцій зчитування параметрів порту.

У випадку вдалого завершення, функція BuildCommDCB повертає не нульове значення. У випадку помилки повертається значення 0.

Другий спосіб припускає виконання такої послідовності кроків:

1. Зчитати поточні значення полів структури DCB, встановлені операційною системою за замовчуванням;
2. Внести необхідні зміни в поля цієї структури;
3. Призначити знову заповнену структуру для подальшого використання.

Заповнити поля структури DCB даними про поточний стан порту (точніше, про його налаштування) дозволяє функція GetCommState, прототип якої має вигляд:

```
BOOL GetCommState(
HANDLE hFile,
LPDCB lpDCB
);
```

Функція дуже проста і має всього два параметри:

hFile

Дескриптор відкритого файлу комунікаційного порту. Цей дескриптор повертається функцією CreateFile. Отже, перш ніж одержати параметри порту, Ви повинні його відкрити. Для функції BuildCommDCB це не було потрібно.

lpDCB

Вказівник на структуру DCB. Для структури DCB повинен бути виділений блок пам'яті. При успішному завершенні функція повертає ненульове значення. При помилці повертається нуль. Одержати параметри порту можна в будь-який момент, а не тільки при початковому налаштуванні.

Заповнивши поля структури DCB, можна приступати до конфігурування порту. Це робиться за допомогою функції SetCommState, прототип якої має вигляд:

```
BOOL SetCommState(
  HANDLE hFile,
  LPDCB  lpDCB
);
```

Ця функція має точно такі ж параметри, як GetCommState. Відрізняється тільки напрямком передачі інформації. GetCommState зчитує інформацію із внутрішніх керуючих структур і регістрів порту, а SetCommState навпаки заносить її. Варто бути обережним при виклику функції SetCommState, оскільки вона змінить параметри навіть у тому випадку, якщо черги прийому/передачі не порожні, що може викликати перекручування потоку переданих або прийнятих даних. У випадку успішного завершення вертається відмінне від нуля значення, а у випадку помилки - нуль.

Наступною важливою керуючою структурою є COMMTIMEOUTS. Вона визначає параметри часових затримок при прийманні і передачі даних. Значення, що задаються полями цієї структури, дуже впливають на роботу функцій зчитування/запису.

```
typedef struct _COMMTIMEOUTS {
  DWORD ReadIntervalTimeout;
  DWORD ReadTotalTimeoutMultiplier;
  DWORD ReadTotalTimeoutConstant;
  DWORD WriteTotalTimeoutMultiplier;
  DWORD WriteTotalTimeoutConstant;
} COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```

Поля структури COMMTIMEOUTS мають наступні значення:

ReadIntervalTimeout

Максимальний час, у мілісекундах, припустимий між двома послідовними символами, які зчитуються з комунікаційної лінії. Під час операції зчитування, часовий проміжок починає відлічуватися з моменту

прийому першого символу. Якщо інтервал між двома послідовними символами перевищить задане значення, операція зчитування завершується і всі дані, накопичені в буфері, передаються в програму. Нульове значення даного поля означає, що даний тайм-аут не використовується. Значення MAXDWORD, разом з нульовими значеннями полів ReadTotalTimeoutConstant і ReadTotalTimeoutMultiplier, означає негайне повернення з операції зчитування з передачею вже прийнятого символу, навіть якщо жодного символу не було отримано з лінії.

ReadTotalTimeoutMultiplier

Задає множник, у мілісекундах, який використовується для обчислення загального тайм-ауту операції зчитування. Для кожної операції зчитування даного значення множиться на кількість символів, що вказані для зчитування.

ReadTotalTimeoutConstant

Задає константу, у мілісекундах, яка використовується для обчислення загального тайм-ауту операції зчитування. Для кожної операції зчитування дане значення додається до результату множення ReadTotalTimeoutMultiplier на кількість запитаних для зчитування символів. Нульове значення полів ReadTotalTimeoutMultiplier і ReadTotalTimeoutConstant означає, що загальний тайм-аут для операції зчитування не використовується.

WriteTotalTimeoutMultiplier

Задає множник, у мілісекундах, який використовується для обчислення загального тайм-ауту операції запису. Для кожної операції запису дане значення множиться на кількість символів, що записуються.

WriteTotalTimeoutConstant

Задає константу, у мілісекундах, яка використовується для обчислення загального тайм-ауту операції запису. Для кожної операції запису дане значення додається до результату множення WriteTotalTimeoutMultiplier на кількість символів, що записуються. Нульове значення полів WriteTotalTimeoutMultiplier і WriteTotalTimeoutConstant означає, що загальний тайм-аут для операції запису не використовується.

Якщо не використовуються тайм-аути для синхронної операції зчитування і зовнішній пристрій, вийшовши з ладу, припинив передачу, то програма буде вічно очікувати завершення цієї операції (інакше кажучи, вона «зависне»). Якщо ж тайм-аути використовуються, то операція зчитування завершиться коректно – тільки кількість фактично зчитаних байт буде менше кількості байт, запитаних для зчитування, але це не обов'язково свідчить про помилку. Наприклад, програма може по тайм-ауту визначати кінець чергового блоку даних. Аналогічно і для операції

запису, з тією лише різницею, що неповна передача даних з буфера, швидше за все, буде свідчити про проблему в зовнішньому пристрої, тобто буде вважатися помилкою.

Як і для заповнення структури DCB, для COMMTIMEOUTS існує функція зчитування встановлених у системі значень. Це функція GetCommTimeouts:

```
BOOL GetCommTimeouts(
HANDLE hFile,
LPCOMMTIMEOUTS lpCommTimeouts
);
```

В якості першого параметру передається дескриптор відкритого порту. Другий параметр – вказівник на структуру COMMTIMEOUTS. При успішному виконанні функції повертається ненульове значення.

Заповнивши структуру COMMTIMEOUTS, можна викликати функцію установки тайм-аутів порту. Це функція називається SetCommTimeouts:

```
BOOL SetCommTimeouts(
HANDLE hFile,
LPCOMMTIMEOUTS lpCommTimeouts
);
```

Установку тайм-аутів можна робити як до установки параметрів порту, так і після. Послідовність виклику функцій SetCommState і SetCommTimeouts не має ніякого значення. Головне, що б всі налаштування були завершені до початку введення/виведення інформації.

Для розуміння тайм-аутів припустимо наступне. Нехай ми зчитуємо 50 символів з порту зі швидкістю 9600 біт/с. При цьому використовується 8 біт на символ, доповнення до парності і один стоповий біт. Таким чином, на один символ у фізичній лінії припадає 11 біт, включаючи стартовий. 50 символів на швидкості 9600 біт/с будуть прийматися 50 11/9600 = 0.0572916 секунд, або приблизно 57.3 мілісекунди, за умови нульового інтервалу між прийомом послідовних символів. Якщо інтервал між символами становить приблизно половину часу передачі одного символу, тобто 0.5 мілісекунди, то час прийому буде 50 11/9600+49 0.0005=0.0817916 секунд, або приблизно 82 мілісекунди. Якщо в процесі зчитування пройшло більше 82 мілісекунд, то ми вправі припустити, що відбулася помилка в роботі зовнішнього пристрою і припинити зчитування, уникнувши тим самим зависання програми. Це і є загальний тайм-аут операції зчитування. Аналогічно існує і загальний там-аут операції запису. Формула для обчислення загального тайм-ауту операції, наприклад зчитування, виглядає так:

$$\text{NumOfChar} \cdot \text{ReadTotalTimeoutMultiplier} + \text{ReadTotalTimeoutConstant},$$

де NumOfChar – це число символів, запитаних для операції зчитування.
Тепер невеликий приклад. ReadTotalTimeoutMultiplier = 2, ReadTotalTimeoutConstant = 1, ReadIntervalTimeout = 1, зчитується 250 символів. Якщо операція зчитування завершиться за 250 · 2 + 1 = 501 мілісекунду, то буде зчитане все повідомлення. Якщо операція зчитування не завершиться за 501 мілісекунду, то вона однаково буде завершена. При цьому будуть повернуті символи, прийом яких завершився до закінчення тайм-ауту операції. Інші символи можуть бути отримані наступною операцією зчитування. Якщо між початками двох послідовних символів пройде більше 1 мілісекунди, то операція зчитування так само буде завершена.

Прийом і передача даних виконується функціями ReadFile і WriteFile, тобто тими ж самими, які використовуються для роботи з дисковими файлами. Ось як виглядають прототипи цих функцій:

```
BOOL ReadFile(
  HANDLE       hFile,
  LPVOID       lpBuffer,
  DWORD        nNumOfBytesToRead,
  LPDWORD      lpNumOfBytesRead,
  LPOVERLAPPED lpOverlapped
);
BOOL WriteFile(
  HANDLE       hFile,
  LPVOID       lpBuffer,
  DWORD        nNumOfBytesToWrite,
  LPDWORD      lpNumOfBytesWritten,
  LPOVERLAPPED lpOverlapped
);
```

Опис параметрів функцій:

hFile

Дескриптор відкритого файлу комунікаційного порту.

lpBuffer

Адреса буфера. Для операції запису дані із цього буфера будуть передаватися в порт. Для операції зчитування в цей буфер будуть поміщатися прийняті дані.

nNumOfBytesToRead, nNumOfBytesToWrite

Число очікуваних для прийому або призначених для передачі байт.

nNumOfBytesRead, nNumOfBytesWritten

Число фактично прийнятих або переданих байт. Якщо прийнято або передано менше даних, ніж запитано, то для дискового файлу це свідчить про помилку, а для комунікаційного порту зовсім не обов'язково. Причина в тайм-аутах.

lpOverlapped

Адреса структури OVERLAPPED, що використовується для асинхронних операцій. Для синхронних операцій даний параметр повинен бути рівним NULL.

Іноді потрібно терміново передати символ, що має певне спеціальне значення, а в черзі передавача вже є дані, які не можна втрачати. У цьому випадку можна скористатися функцією:

```
BOOL TransmitCommChar(
  HANDLE hFile,
  char   cChar
);
```

Дана функція передає один (і тільки один) позачерговий байт у лінію, не дивлячись на наявність даних у черзі передавача, і перед цими даними. Перший параметр – дескриптор відкритого порту. Другий параметр - символ для передачі.

В наступному прикладі виконується розробка мікропроцесорної система збору даних, підключена до ПК через COM-порт, виконує контроль температури деякого об'єкту. Написати програму, що приймає значення температури і відображає її у вигляді графіку. Для одержання даних необхідно відправити запит – команда 0xBC. Періодичність запиту значення температури – 100 мс. Швидкість передачі становить 9600 біт/с. Формат посилки – 8 біт даних в інформаційному блоці, 1 стоповий біт, перевірка на парність відсутня.

Порядок дій:

1. Запускаємо середовище розробки Visual Studio .NET. Відкриваємо проект, створений у прикладі 5, який беремо за основу.
2. Змінюємо вигляд діалогового вікна. Виносимо на форму елемент керування «Static Text» коректуємо його ID на IDC_STATIC3. Також очищаємо поле «Caption» щоб не відображався текст. Потім необхідно розтягти цей елемент керування так, щоб по ширині він зайняв всю форму, а по висоті – приблизно 70%. У полі «Border» на сторінці властивостей установити значення «True». В області, яку зайняв елемент керування «Static Text» буде відображатися графік зміни температури. Також необхідно прив'язати до цього елементу керування змінну з ім'ям m_graph. На рис. 5.15 приводиться вигляд діалогового вікна.

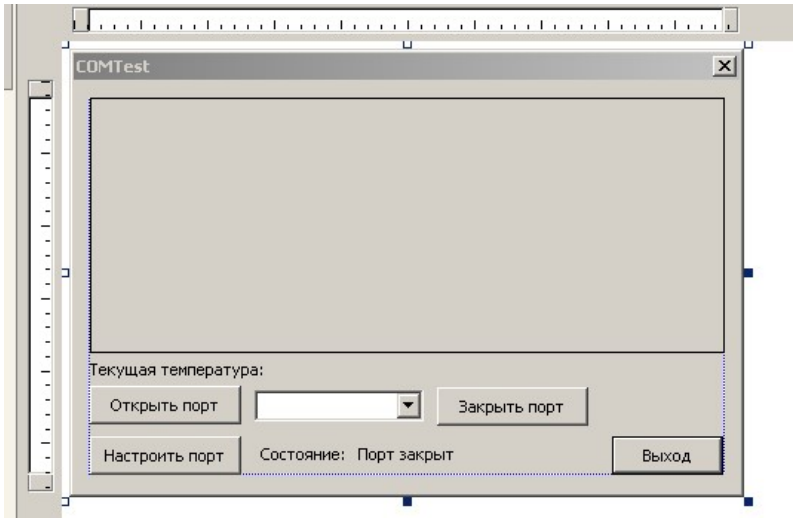


Рис. 5.15.

Крім того, необхідно додати елемент керування «Static Text» для відображення поточної температури. Змінити ID на IDC_STATIC4. До цього елементу керування прив'язати змінну з ім'ям m_currtemp.

3. Для зберігання отриманих даних про температуру використовується змінна m_temp яка має тип CByteArray. Ця змінна - член класу CCOMTestDlg. Для того, щоб додати змінну, у вікні «Class View» виділити клас «CCOMTestDlg», викликати контекстне меню і з підменю «Add» вибрати команду «Add Variable». Діалогове вікно заповнити так, як показано на рис. 5.16.

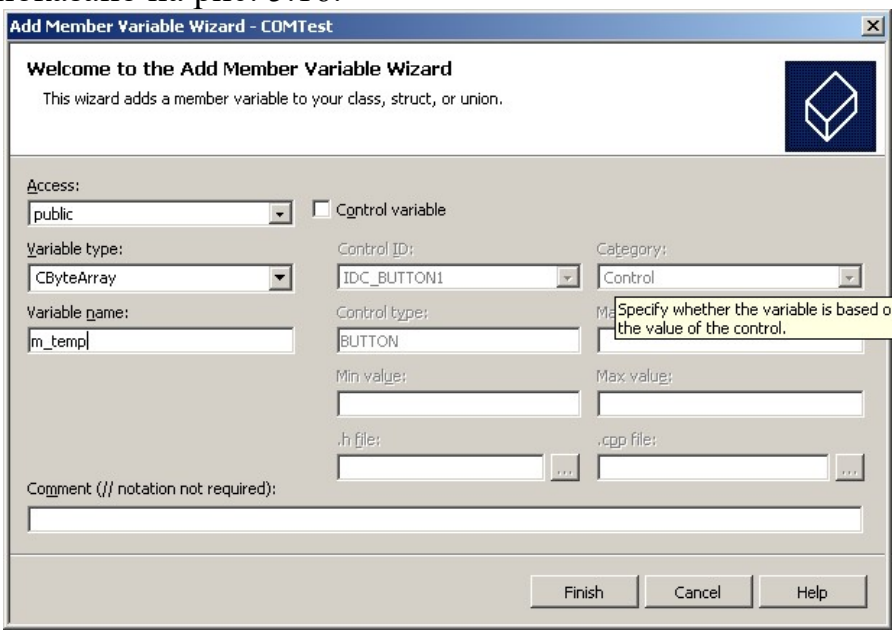


Рис. 5.16.

4. Додаємо оброблювач події таймера. Для цього зі сторінки властивостей (Properties) викликати вікно «Messages» і навпроти позиції «WM_TIMER» вибрати «OnTimer» (рис. 5.17).

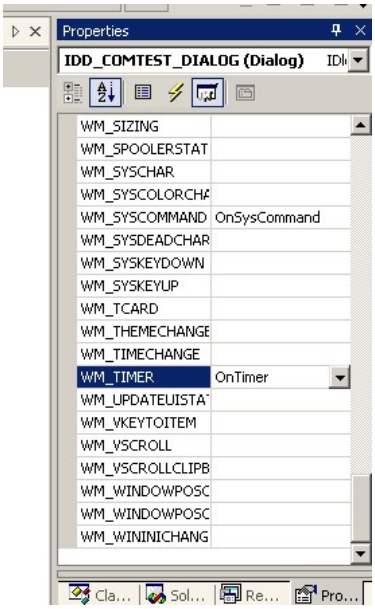


Рис. 5.17.

5. Додаємо у функцію – оброблювач події таймера код, як показано нижче:

```
void CCOMTestDlg::OnTimer(UINT nIDEvent)
{
	TransmitCommChar(m_hCom,(char)0xBC); // передача значення 0xBC
	BYTE rdata;
	DWORD size;
	ReadFile(m_hCom,&rdata,1,&size,NULL);// зчитування 1 байта з порту
	if(size!=1)	// якщо 1 байт не зчитаний
	{
		// видаємо повідомлення
		SetWindowText("Відсутні дані про температуру!");
		return; // вихід з функції
	}
	m_temp.Add(rdata); // додаємо дані в масив
	CString s;
	// форматування рядка
	s.Format("Поточна температура: %d град.", rdata);
	m_currtemp.SetWindowText(s);
	DrawGraph(); // оновлення графіка на екрані
	CDialog::OnTimer(nIDEvent);
}
```

У цій функції запит на одержання даних про температуру відправляється за допомогою функції TransmitCommChar. Потім відбувається зчитування з порту 1 байту даних Якщо дані були отримані, значення змінної size (кількість зчитаних байт) буде дорівнює 1. Якщо ж це значення відрізняється від 1, видається повідомлення про те, що дані про температуру відсутні. Завдяки налаштуванням тайм-аутів, програма не «зависне» через відсутність прийнятих даних.

6. Реалізуємо функцію void DrawGraph(void). Для цього її необхідно зробити членом класу CCOMTestDlg. Реалізація функції має такий вигляд:

```
void CCOMTestDlg::DrawGraph(void)
{
	CClientDC dc(&m_graph);   // створення контексту для малювання
	CPen p(PS_SOLID,2,RGB(200,0,0));// створення пера червоного кольору
	CPen pp(PS_DOT,1,RGB(0,0,0));
	CRect rect;
	CRgn rgn;
	CPen* op;
	m_graph.RedrawWindow();  // перемальовування вікна малювання графіка
	m_graph.GetClientRect(&rect);
	// одержання розмірів області малювання
	rgn.CreateRectRgn(0, 0, rect.Width(), rect.Height());
	// створення регіону для обмеження області малювання
	op=dc.SelectObject(&pp);    // установка пера
	for(int x=20;x<rect.Width();x=x+20)          // малювання сітки
	{
		dc.MoveTo(x,0);       // переміщення початкової позиції
		dc.LineTo(x,rect.Height());   // малювання лінії
	}
	dc.SetTextColor(RGB(50,50,50));    // установка кольору тексту
	dc.SetBkColor(RGB(210,210,210)); // установка кольору фону
	for(int y=20;y<rect.Height();y=y+20)
	{
		dc.MoveTo(0,rect.Height()-y);
		dc.LineTo(rect.Width(), rect.Height()-y);
		CString s;
		s.Format("%d",y);      // форматування рядка
		dc.TextOut(rect.Width()-30, rect.Height()-y,s);
		// вивід текстових міток
	}
	dc.SelectObject(&p); // вибір червоного пера в контекст
	dc.SelectClipRgn(&rgn);       // установка регіону
	if(rect.Width()<=m_temp.GetCount()) m_temp.RemoveAll();
	// якщо значення елементів у масиві перевищує розмір області
	// малювання, то знищуються всі елементи масиву
	for(int i=1;i<m_temp.GetCount();i++)
	{
		// малювання графіка температури
		dc.MoveTo(i,rect.Height()-m_temp.GetAt(i));
		dc.LineTo( i-1,rect.Height()-m_temp.GetAt( i-1));
	}
	dc.SelectObject(op);
}
```

7. Коректуємо оброблювач натискання на кнопку «Відкрити порт». Додаємо в цей оброблювач рядок

```
SetTimer(1,200,NULL);
```

При натисканні на цю кнопку, у випадку успішного відкриття порту, запускається таймер, що буде спрацьовувати через кожні 200 мс. При цьому буде викликатися функція OnTimer, у якій реалізований код по запиту даних про температуру і їхнє відображення.

8. Коригуємо оброблювач події натискання на кнопку «Закрити порт». Оскільки при закритті порту дані неможливо буде зчитати, необхідно зупинити таймер. Для цього в функцію-оброблювач події потрібно додати наступний рядок:

```
KillTimer(1);
```

9. Відкомпілювавши проект, з'явиться діалогове вікно, як показано на рис. 5.18 (за умови, що дані приймаються через порт).

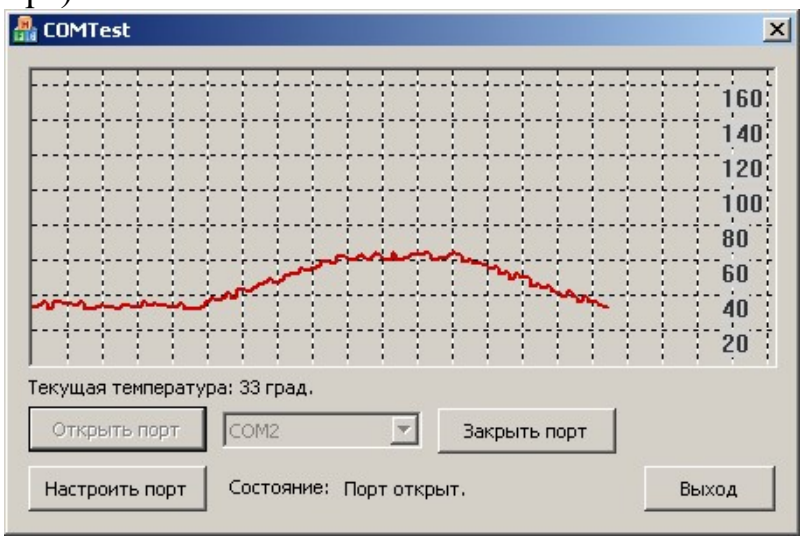


Рис. 5.18.

5.1.4. Програмні засоби керування гусеничним транспортним роботом по послідовному інтерфейсу

Транспортні роботи призначені для автоматизованого транспортування об'єктів, а також для управління різними транспортними системами. Дослідження і розробки по створенню транспортних роботів інтенсивно ведуться в усьому світі. При цьому виділяються чотири принципово різних типи – наземні, повітряні, водоплавні і підземні.

Гусеничні транспортні роботи застосовуються для наукових досліджень в умовах важкодоступної місцевості і гірських розробок, а також для виконання виробничих операцій (наприклад, робототехнічні комплекси).

Колісні транспортні роботи знаходять широке застосування на промислових підприємствах. Умовно ці промислові роботи можна розділити на наступні типи:

– автоматичні тягачі (АТ), що забезпечують транспортування вантажлнесучих візків, які підчіплюють до них;

– транспортні візки (ТТ), що забезпечують крім буксирування автономне перевезення вантажів;

– маніпуляційні візки (МТ), що забезпечують крім буксирування і автономного перевезення вантажів, ще й виконання ряду технологічних операцій, таких, як взяття вантажу, його штабелювання, орієнтування, сортування, установка-зняття при обслуговуванні обробного обладнання і т.п. Для цього вони оснащуються відповідними механізмами: підйомниками, маніпуляторами і різними типами навантажувачів.

Найбільший розвиток і поширення набули колісні і гусеничні транспортні роботи, які використовуються досить широко в промислових автоматизованих транспортно-складських системах і гнучких автоматизованих виробництвах у вигляді мобільних автоматичних кранів, автоматичних керованих візків (АКТ), робокари та ін., які оснащуються в багатьох випадках різними маніпуляційними пристроями .

Для керування транспортними роботами використовуються також оптичні, ультразвукові та інші системи автоматичного завдання і стабілізації курсу. На рис. 5.19 представлено зовнішній вигляд гусеничного транспортного робота.



Рис. 5.19.

Гусеничний привід дуже схожий на диференційний привід з тією лише різницею, що колеса контактують з землею не в одній точці, а площа контакту розтягнута по довжині гусениці. Це відразу зменшує прослизання при прямолінійному русі, збільшує зчеплення і прохідність на м'якому ґрунті, але вимагає великих витрат енергії в поворотах для компенсації бічного опору гусениць. Якщо сила тертя буде занадто великою, гусениці можуть бути навіть зірвані з коліс.

Опис команд керування роботом представлено нижче.

Параметри зв'язку:
Швидкість: 9600 біт/с
Стоп-біт: 1
Біт паритету: відсутній

Формат керуючого пакету даних:
Розмір пакету 4 байта

Байт1	Байт2	Байт3	Байт4
0x40	*cmd1*	*cmd2*	0x21

Байт1 і Байт4 використовуються для ідентифікації пакету даних. Якщо значення Байт1 дорівнює 64 (0x40, ASCII-символ '@'), а значення Байт4 дорівнює 33 (0x21, ASCII-символ '!'), то біти поля *cmd1* (Байт2) визначають команду, що виконується. Поле *cmd2* (Байт3) не використовується, значення завжди дорівнює 0.

Значення біт поля *cmd1*:

M1		M2		M3		Не використовуються	
Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
B1	B0	B1	B0	B1	B0	0	0

Значення біт поля *cmd2* не використовуються і завжди дорівнюють 0:

Не використовуються, завжди дорівнюють 0							
Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
0	0	0	0	0	0	0	0

M1 – мотор правої гусениці
M2 – мотор лівої гусениці
M3 – мотор підіймача

Значення біт полів M1-M3:
{B1, B0}
0 0 – зупинка мотора
0 1 – обертання мотора назад (для M3 – підіймач опускається)
1 0 – обертання мотора вперед (для M3 – підіймач піднімається)
Відповідь – завжди 1 байт. Якщо команда керування вдало прийнята, повертається значення 36 (0x24, ASCII-символ '$').

Приклади команд:
Обертання робота праворуч на місці. Відправляється пакет 4 байта:
(ліва гусениця вперед, права гусениця назад)

Байт1	Байт2	Байт3	Байт4
0x40	0x60	0	0x21

Обертання робота ліворуч на місці. Відправляється пакет 4 байта:
(ліва гусениця назад, права гусениця вперед)

Байт1	Байт2	Байт3	Байт4
0x40	0x90	0	0x21

Зупинка всіх моторів: Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0	0	0x21

Рух вперед: Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0xA0	0	0x21

Опустити підіймач: Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0x50	0	0x21

Для управління роботом може використовуватися програма, інтерфейс якої представлений на рис. 5.20.

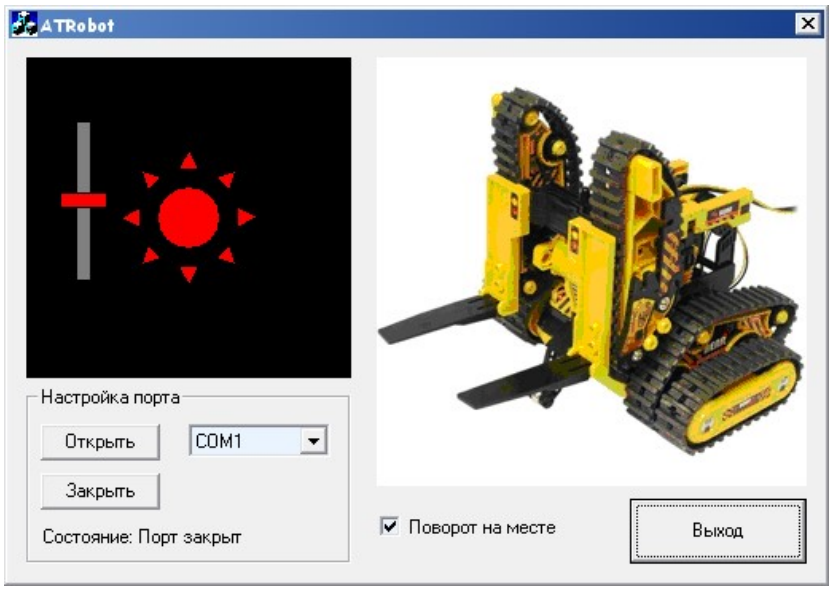


Рис. 5.20.

Функція-обробник натискання на кнопку «Открыть»:

```
void CATRobotDlg::OnButton1()
{
        CString portname;
        m_port.GetLBText(m_port.GetCurSel(),portname);          // зчитування значення
        // з вибраного послідовного порту
        DCB* pDCB;                  // структура для налаштування параметрів порту
        COMMTIMEOUTS ct;        // структура для налаштування тайм-аутів
        pDCB=new DCB;               // динамічне  виділення пам’яті для структури
        memset(pDCB,0,sizeof(DCB));// встановлення значення полів структури в 0
```

```
	m_hCom=CreateFile(portname,GENERIC_READ |
GENERIC_WRITE,0,NULL,OPEN_EXISTING,0,NULL);// відкриття порту
	if(m_hCom==INVALID_HANDLE_VALUE)
	{
		MessageBox("Неможливо відкрити послідовний
порт!","Помилка!",MB_OK | MB_ICONERROR);	// якщо неможливо відкрити
порт, видається повідомлення
		m_ps.SetWindowText("Помилка порта.");
		m_isopen=false;
		return;		// завершуємо роботу функції
	}
	else	// якщо порт відкритий
	{
		m_port.EnableWindow(false);	// список з портами не доступний
		m_isopen=true;			// встановлюємо логічну змінну в стан 1
		m_bop.EnableWindow(false);
		m_bcl.EnableWindow(true);
		m_ps.SetWindowText("Стан: порт відкритий");
		SetTimer(1,300,NULL);
	}
	GetCommState(m_hCom,pDCB);	// отримуємо поточні налаштування
					// послідовного порту
	GetCommTimeouts(m_hCom, &ct);// і значення тайм-аутів

	pDCB->DCBlength=sizeof(DCB);
	pDCB->BaudRate=CBR_9600;	// швидкість обміну 9600 біт/с
	pDCB->ByteSize=8;		// в слові 8 біт даних
	pDCB->StopBits=ONESTOPBIT;	// один стоповий біт
	pDCB->Parity=NOPARITY;		// перевірка на парність відсутня

	ct.ReadIntervalTimeout=10;  //  встановлення значення тайм-аутів на приймання і
передачу даних
	ct.ReadTotalTimeoutConstant=10;
	ct.ReadTotalTimeoutMultiplier=2;
	ct.WriteTotalTimeoutConstant=100;
	ct.WriteTotalTimeoutMultiplier=2;

	SetCommState(m_hCom,pDCB);	// встановлення нових налаштувань порту
	SetCommTimeouts(m_hCom,&ct);
}
```

Функція-обробник натискання на кнопку «Закрыть»:

```
void CATRobotDlg::OnButton2()
{
if(m_isopen==true)
	{
		KillTimer(1);			// зупинка таймера
		CloseHandle(m_hCom);
		m_hCom=NULL;			// обнуління дескриптора порта
		m_isopen=false;
		m_port.EnableWindow(true); // розблокування елементів керування
```

```
            m_bop.EnableWindow(true);
            m_bcl.EnableWindow(false);
            m_ps.SetWindowText("Стан: порт закритий");
        }
}
```

Функція-обробник події таймера, в якій відбувається визначення положення рукоятки джойстика, формування пакета даних і його відправка в систему управління:

```
void CATRobotDlg::OnTimer(UINT nIDEvent)
{
        if(nIDEvent==1)
        {
            CString s;
            int x=m_joy.GetXPos();      // отримання інформації про положення
            int y=m_joy.GetYPos();      // джойстика
            int z=m_joy.GetZPos();

            if(m_isopen!=true) return;
            DWORD wb,rb;
            unsigned char wbuff[4]={0,0,0,0}, rbuff[2]={0,0};
            wbuff[0]='@';               // перший символ команди керування
            wbuff[3]='!';               // останній символ команди керування

            if(m_ch.GetCheck())
            {
            if(x>10)
            {
                wbuff[1]|=0x20;         // формування команди обертання
                wbuff[1]|=0x40;         // за часовою стрілкою
            }
            else if(x<-10)
            {
                wbuff[1]|=0x10;         // формування команди обертання
                wbuff[1]|=0x80;         // проти часової стрілки
            }
            }
            else
            {
            if(x>10)
            {
                wbuff[1]|=0x20;         // команда увімкнення мотора лівої гусениці
            }
            else if(x<-10)
            {
                wbuff[1]|=0x80;         // команда увімкнення мотора правої гусениці
            }
            }

            if(y>10)
            {
```

```
                wbuff[1]|=0x10;         // формування команди руху вперед
                wbuff[1]|=0x40;
            }
            else if(y<-10)
            {
                wbuff[1]|=0x20;         // формування команди руху назад
                wbuff[1]|=0x80;
            }

            if(z>10) wbuff[1]|=0x04;        // формування команди підняття підіймача
            else if(z<-10)wbuff[1]|=0x08; // формування команди опускання підіймача

            // виклик функції запису команди в систему керування роботом
            WriteFile(m_hCom,wbuff,4,&wb,NULL);
            // зчитування у масив отриманої відповіді від системи керування роботом
            ReadFile(m_hCom,rbuff,1,&rb,NULL);

            if(rb==1)
            {

            }
        }
        CDialog::OnTimer(nIDEvent);
}
```

5.1.5. Використання Bluetooth-радіомодему з інтерфейсом RS-232 для дистанційного керування мобільним роботом

Bluetooth – це мабуть найпоширеніший тип зв'язку для коротких дистанцій, яким користуються більшість сучасних електронних пристроїв. Телефонні гарнітури, навушники, клавіатури і мишки, принтери і багато інших пристроїв використовують Bluetooth. Деякий час назад Bluetooth був поширений як засіб передачі файлів між телефонами. В даний час в зарубіжних і вітчизняних магазинах можна знайти велику кількість різних Bluetooth Serial модулів для DIY-пристроїв. Ціна на них не висока і складає 5-10 $ за модуль (з урахуванням безкоштовної доставки в будь-яку країну). Їх легко можна знайти на широко відомому сайті AliExpress ввівши в пошуковий рядок одне із словосполучень: Bluetooth RS232, Bluetooth Serial, HC-03, HC-04, HC-05, HC-06. На рис. 5.21 показано зовнішній вигляд Bluetooth-радіомодемів HC-05.

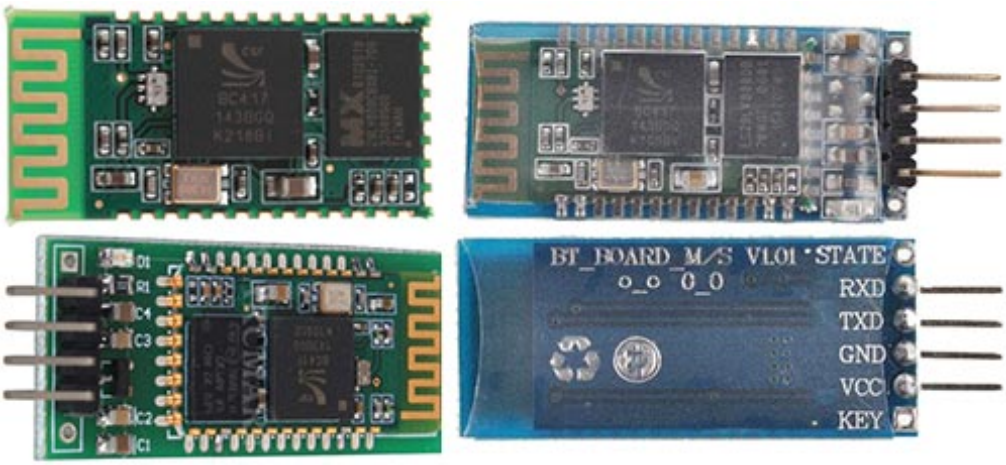


Рис. 5.21.

В більшості модулів використовується мікросхема BC417, а також Flash-пам’ять. Модуль підтримує специфікацію Bluetooth v2.0 + EDR, AT-команди, може працювати в режимі Master або Slave (модулі HC-03/HC-5, може встановлюватися AT-командою), підтримує швидкість обміну від 2400 до 1382400 біт/с.

Призначення виводів у всіх варіантів модулів:

VCC – живлення +5В;

GND – земля;

TXD, RXD – UART інтерфейс для сполучення з мікроконтролером;

STATE – індикатор стану;

KEY – вивід для входу в режим AT-команд.

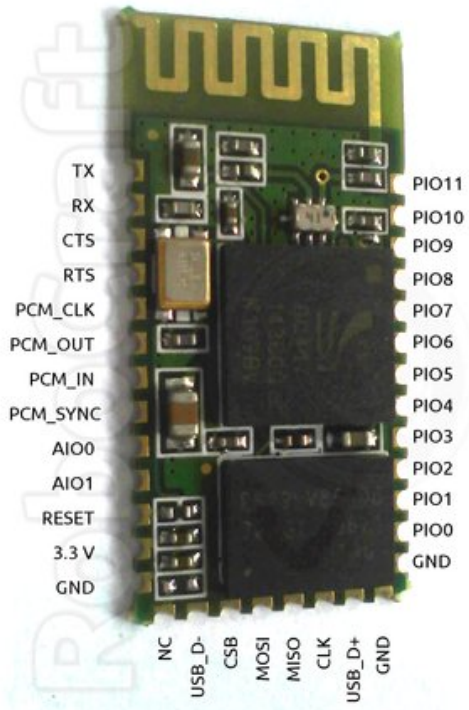


Рис. 5.22.

TX, RX, CTS, RTS – лінії UART

PCM_CLK, PCM_OUT, PCM_IN, PCM_SYNC – лінії для приймання-передачі звуку

AIO0, AIO1 – лінії I/O загального призначення

RESET – лінія скидання (активується логічним нулем)

3.3 V, GND – живлення, земля

NC – не підключений (Not Connected)

USB_D+, USB_D- – лінії даних USB

CSB, MOSI, MISO, CLK – лінії SPI (CSB – це SS, Slave Select), використовуються для перепрограмування мікросхеми

PIO0 – дозвіл/заборона RX

PIO1 – дозвіл/заборона TX

PIO2-PIO7, PIO10 – лінії I/O загального призначення

PIO8 – для світлодіода, який показує стан модуля: світлодіод блимає з різною швидкістю в залежності від того, чим зайнятий модуль – опитуванням Bluetooth-пристроїв, очікуванням або іншим завданням

PIO9 – для світлодіода, який показує статус з'єднання: горить, якщо встановлено з'єднання з іншим Bluetooth-пристроєм

PIO11 – для керування режимом роботи: за замовчуванням режим простого подовжувача UART, а якщо подати на нього логічну 1 – режим AT-команд

Модуль живиться від 3,3 В, але його лінії I/O можуть працювати з 5-вольтовою логікою.

Характеристики Bluetooth-модуля HC-05:

– протокол зв'язку Bluetooth Specification v2.0+EDR

– частота GFSK(Gaussian Frequency Shift Keying)

– потужність передавача ≤4дБм, Class 2

– потужність приймача ≤-84дБм

– швидкість асинхронна 2,1МБіт/с (максимальна) / 160 кБіт/с ,
синхронна 1МБіт/1МБіт

– безпека Authentication and encryption

– профіль Bluetooth serial port

– живлення +5В, 50мА

– робочі температури -20 ~ +75 ^{0}C

– розміри 26,9мм × 13мм × 2,2мм

Іноді зручно якусь величину прийняти за еталон (нульовий рівень) і відносно неї вимірювати рівень вже в децибелах. Так, якщо взяти за нульовий рівень 1мВт і відносно його вимірювати, то з'являється така одиниця виміру як дБм (1 мВт = 0 дБм). Вона вже має цілком вагомий фізичний зміст, на відміну від децибелів, dBm – це міра потужності. У ній вимірюють рівень слабких сигналів, чутливість приймачів, потужність передавачів і т.п. Наприклад, рівень в 50 мкв на 50-омному вході приймача відповідає рівню потужності $5 \cdot 10^{-8}$ мВт або -73 дБм. Вимірювати чутливість в одиницях потужності більш зручно, ніж в одиницях напруги, оскільки доводиться мати справу з сигналами різної форми, в тому числі шумовими. До того ж, немає необхідності кожного разу уточнювати, який

вхідний опір приймача. Наприклад, гранична потужність більшості "свистків", при якій вони ще з'єднуються з базовою станцією близько -110 dBm. Потужність передавача теж можна вимірювати в dBm. Наприклад, потужність WiFi роутера в 100 мВт дорівнює 20 dBm. dBm має пріоритет як міра потужності сигналу. Наприклад: рівень на вході приймача (dBm) = Потужність передавача (dBm) + Посилення антен (dBi) – Ослаблення сигналу (dB).

Після включення модуля HC-05 повинен блимати світлодіод червоного (або іншого) кольору на модулі з частотою близько 4 рази в секунду без пауз. Це індикація включеного модуля без наявності підключення до якогось пристрою. Тепер розглянемо налаштування Bluetooth в смартфоні (Android 4.1.2).

В Google Play потрібно знайти додаток Bluetooth Terminal (https://play.google.com/store/apps/details?id=Qwerty.BluetoothTerminal) і встановити на смартфон. Далі потрібно включити Bluetooth на телефоні і знайти нові Bluetooth пристрої. Знаходимо пристрій з ім'ям HC-05 і підключаємося до нього. При запиті пароля слід ввести пароль 1234. Далі потрібно відкрити меню налаштувань і вибрати Connect a device – Secure (на малюнку виділено червоним), як показано на рис. 5.23.

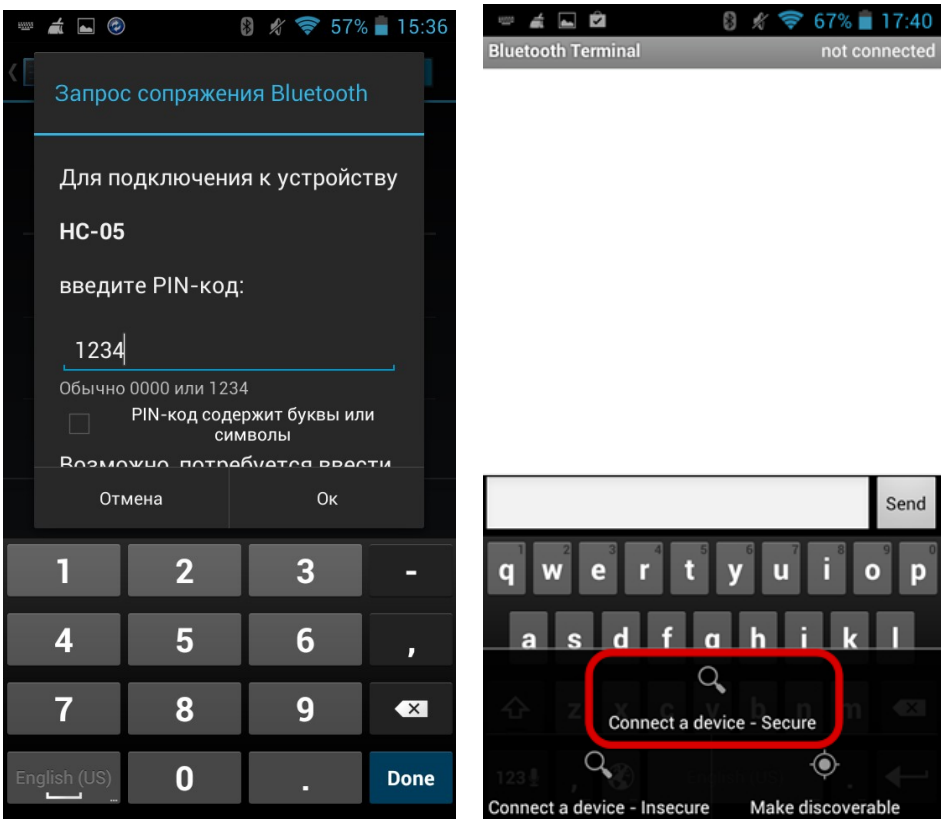


Рис. 5.23.

У вікні термінала (рис. 5.24), що з’явиться, потрібно вибрати Bluetooth модуль HC-05.

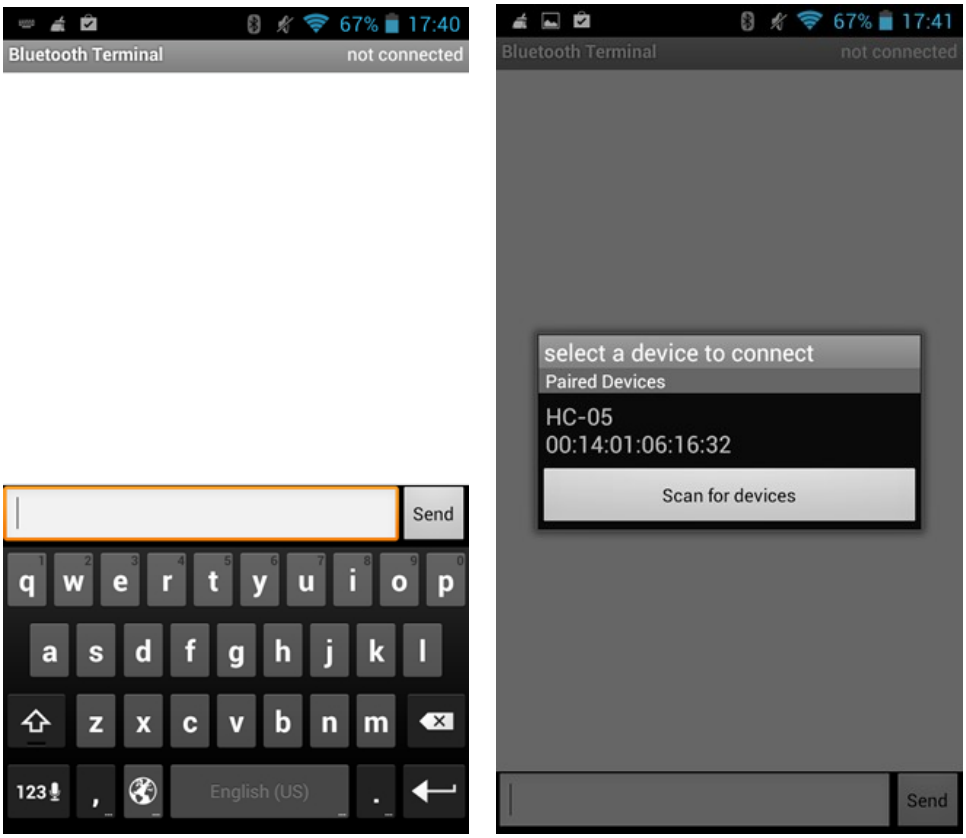


Рис. 5.24.

На комп'ютері в якості Bluetooth модуля може використовуватися Bluetooth USB адаптер (рис. 5.25).



Рис. 5.25.

Після того, як адаптер підключається до комп'ютера, встановлюються необхідні драйвера. Натиснувши на панелі задач на значок потрібно додати новий пристрій у вікні, що з’явиться (рис. 5.26).

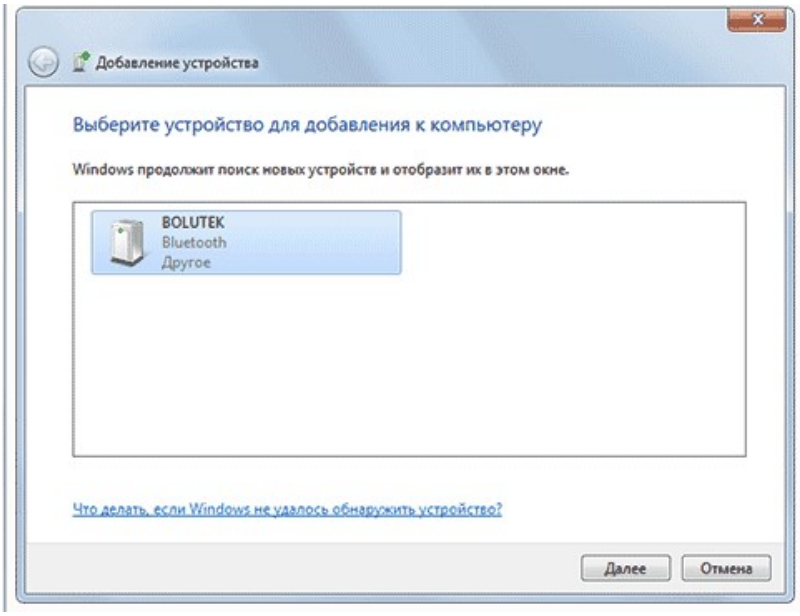


Рис. 5.26.

Натискаємо "Далее" и переходимо до вікна, де необхідно ввести пароль для пристрою.

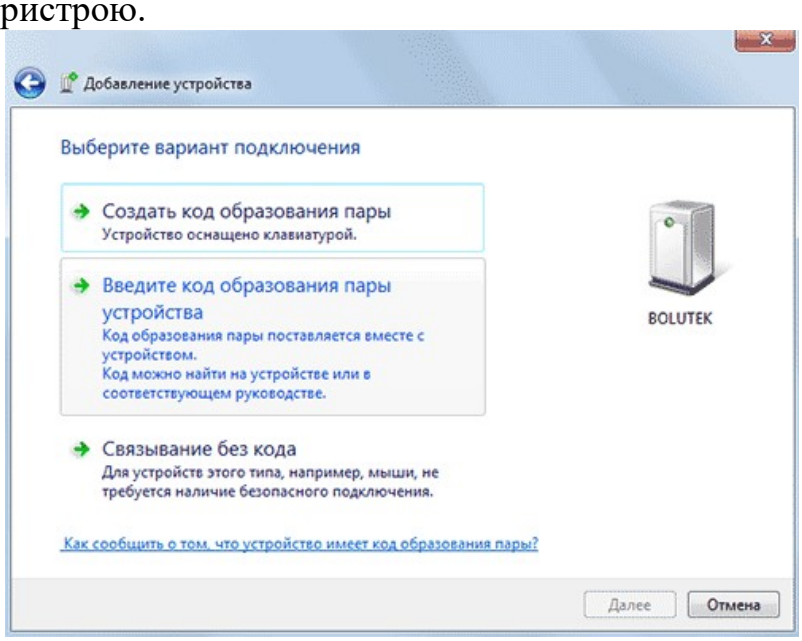


Рис. 5.27.

Слід вибрати другий пункт: "Введите код образования пары устройства". Виробником за замовчуванням встановлений пароль 1234. Цього і слід ввести. Програма налаштує і додасть в систему віртуальні COM-порти. Діалогове вікно з інформацією про установку драйверів представлено на рис. 5.28.

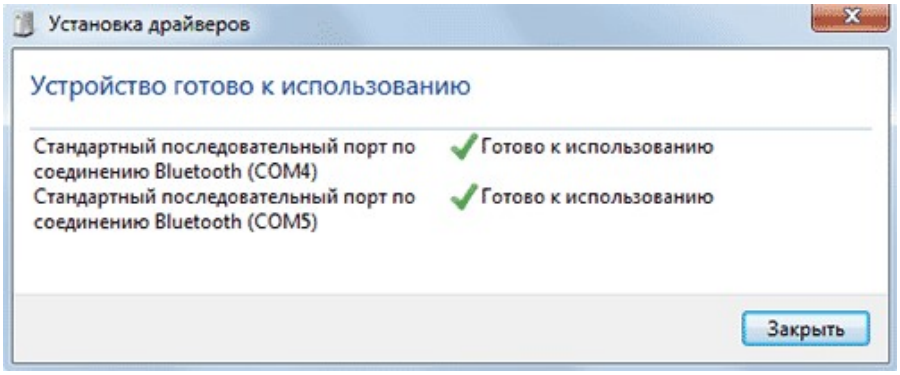


Рис. 5.28.

Можна подивитися на який COM-порт було призначено модуль і його параметри. Для цього слід натиснути правою кнопкою миші по назві пристрою, вибрати "Властивості" і вкладку "Обладнання". Там можна подивитися і змінити параметри віртуального COM-порту, як показано на рис. 5.29.

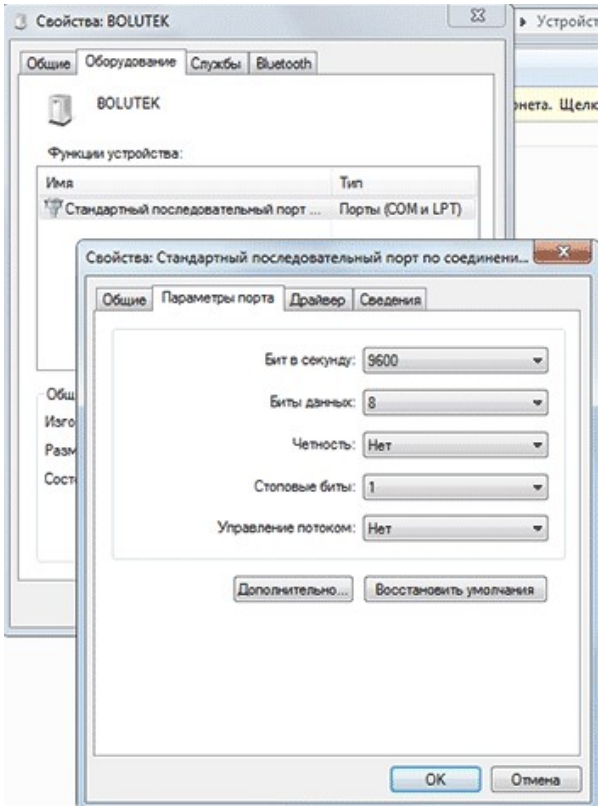


Рис. 5.29.

Тепер, для прийому і передачі даних на віртуальний COM порт можна використовувати будь-яку програму-термінал. При кожному запуску програми пропонується вибір з'єднання. Потрібно перемикнути прапорець на Serial і вибрати COM порт, до якого підключений Bluetooth модуль.

5.2. Шина USB

USB (Universal Serial Bus – універсальна послідовна шина) є промисловим стандартом розширення архітектури PC, орієнтованим на інтеграцію з телефонією та пристроями побутової електроніки. Спочатку (у версіях 1.0 і 1.1) шина забезпечувала дві швидкості передачі інформації: повну швидкість FS (Full speed) – 12 Мбіт/с і низьку швидкість LS (Low Speed) – 1,5 Мбіт/с. У версії 2.0 визначена ще і висока швидкість HS (High Speed) – 480 Мбіт/с, що дозволяє істотно розширити коло пристроїв, що підключаються до шини. В одній і тій же системі можуть бути присутнім і одночасно працювати пристрої з усіма трьома швидкостями. Шина дозволяє з'єднувати пристрої, віддалені від комп'ютера на відстань до 25 м (з використанням проміжних хабів).

5.2.1. Загальна характеристика інтерфейсу USB

Фізичний інтерфейс USB складається із чотирьох провідників: 2 для живлення зовнішнього пристрою (VCC та GND) і 2 сигнальних провідника (DATA+ й DATA–). Через провідники живлення передається постійна напруга 5 В с навантажувальною здатністю максимум 500 мА. Мікроконтролер AVR живиться через виводи Vcc і GND. Сигнальні провідники називаються DATA+ й DATA– і управляють зв'язком між комп'ютером і пристроєм. Сигнали в цих провідниках є двонапрямленими. Рівні напруги – диференціальні: коли DATA+ має високий рівень, тоді DATA– перебуває на низькому рівні. Однак, є деякі випадки, коли DATA+ й DATA– мають однаковий рівень, наприклад, при команді EOP (кінець пакета). На рис. 5.30 представлена осцилограма сигналів при передачі даних низькошвидкісним драйвером.

У відповідності зі стандартом USB високий рівень на сигнальних провідниках повинен становити 3,0...3,6 В, при цьому, напруга живлення Vcc шини USB, що надходить від головного (комп'ютера) становить 4,4...5,25 В. Таким чином, якщо мікроконтролер живиться безпосередньо від шини USB, то лінії даних повинні пройти через каскад перетворення рівнів для компенсації рівнів диференціальної напруги. Іншим рішенням може бути використання стабілізатора напруги, що понизить напругу Vcc до рівня 3,3 В, при цьому мікроконтролер буде працювати із цією зниженою напругою і, відповідно, генерувати знижені рівні напруг. В AT90USB1286 дані рівні реалізовані апаратно.

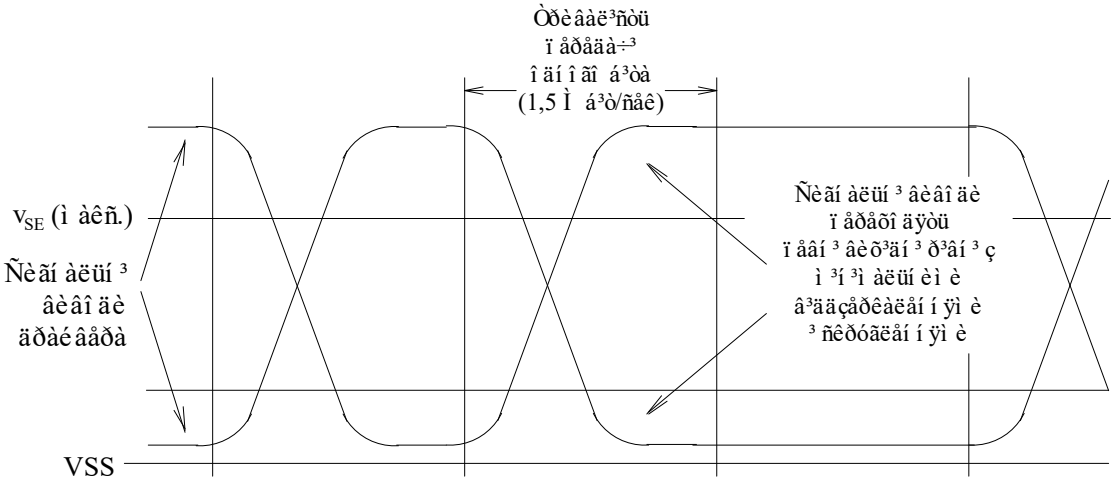


Рис. 5.30.

Дані по шині USB передаються пакетами, по декілька байт у кожному. На рис. 5.31 представлені осцилограми, що демонструють формування рівнів при передачі пакета даних. Розмір пакета визначається кожним пристроєм, але його граничний розмір обмежений. Для низькошвидкісних пристроїв максимальний розмір пакета дорівнює 8 байтам. Даний 8-байтний пакет разом з початковим і кінцевим полем повинні бути прийняті в буфер пристрою за одну USB-передачу.

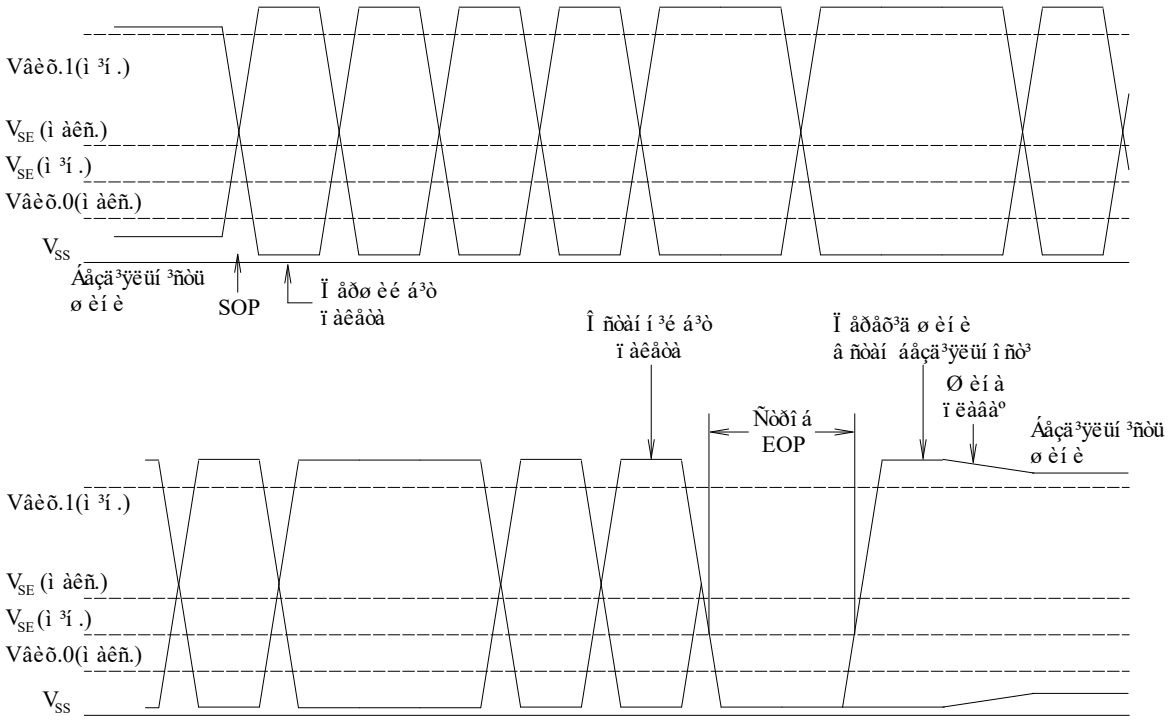


Рис. 5.31.

Після визначення нового пристрою головний починає зв'язок відповідно до фізичного протоколу USB. Протокол USB, на відміну від UART, заснований на синхронній передачі даних. Синхронізація передавача й приймача необхідна для здійснення зв'язку. Синхронізація

виконується шляхом передачі невеликого заголовка "зразкова синхронізація", що передує передачі даних (рис. 5.32). Даний заголовок представляє собою прямокутні імпульси (101010), за ними передаються два 0, а потім дані.

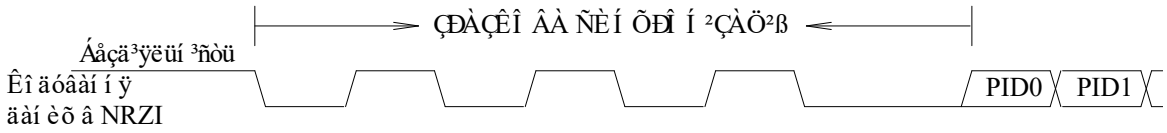


Рис. 5.32.

Зразкова синхронізація присутня в кожному пакеті. В апаратно-реалізованому USB-приймачі дана синхронізація гарантується цифровою системою ФАПЧ (фазове автопідстроювання частоти). У даній реалізації період перетворення даних повинен бути синхронізований зі зразковою синхронізацією, потім очікується два нулі, а потім починається процес прийому даних.

Прийом даних повинен задовольняти вимозі можливості засинхронізувати приймач і передавач у будь-який момент часу. Таким чином, не дозволяється передавати безперервний потік нулів або одиниць по лініях даних. Протокол USB гарантує синхронізацію за рахунок заповнення бітами. Це означає, що, після 6 безперервних одиниць або нулів на лініях даних, вставляється одна зміна (один біт). Сигнали по лініях USB передаються в коді NRZI (Non return to zero inverting – код без повернення до одиниці, інверсний). У коді NRZI кожен '0' представляється шляхом зсуву поточного рівня сигналу, а кожна '1' шляхом утримання поточного рівня (рис. 5.33). На рівні бітового заповнення це означає, що кожен нульовий біт вставляється в потік логічних даних після 6 безперервних логічних 1.

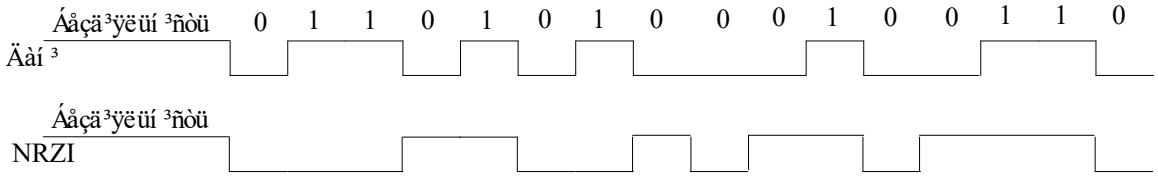


Рис. 5.33.

На рис. 5.34 представлена осцилограма, яка демонструє послідовність кодування даних.

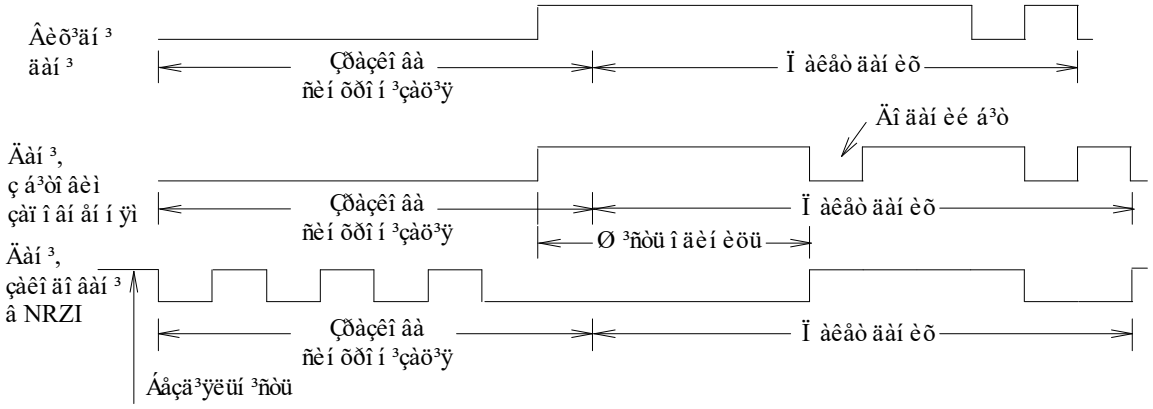


Рис. 5.34.

Повідомлення про закінчення передачі даних виконується за допомогою передачі сигналу "кінець пакета" (EOP). EOP передається шляхом установки низьких рівнів на обох лініях даних DATA+ й DATA–. EOP передається нетривалий час (мінімум два періоди швидкості даних). Це демонструє осцилограма на рис. 5.35. Після цього, виконується наступна транзакція.

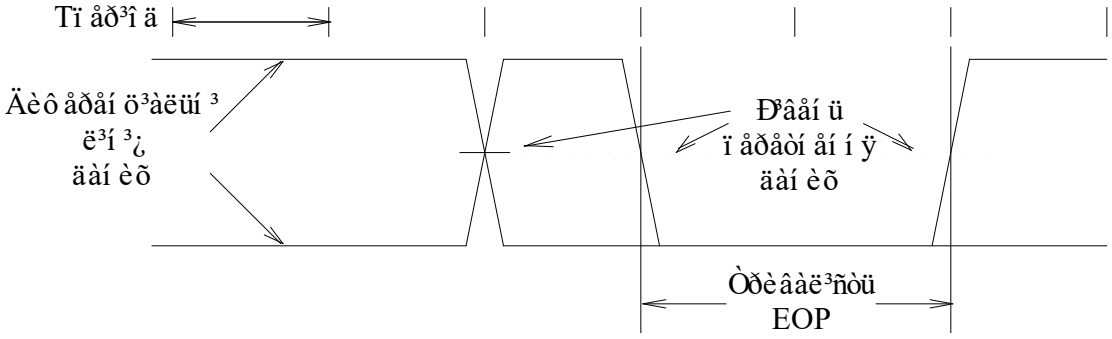


Рис. 5.35.

5.2.2. Методи передачі даних по інтерфейсу USB

Існує 3 основних методи реалізації передачі даних по USB-протоколу. Перший метод полягає в реалізації програмного моста типу RS-232 → USB, з використанням для цього проміжного контролера. Проміжний контролер зберігає в собі програму перетворення пакетів RS-232 в USB. Необхідно лише подавати дані на UART проміжного контролера. Швидкість передачі транзакцій даного проміжного контролера відповідає мінімальній швидкості передачі по специфікації USB і становить 1,5 Мбіт/с. Однак загальна швидкість передачі даних у цьому випадку обмежена швидкістю протоколу RS-232. Даний метод зручний своєю простотою реалізації та низкою вартістю кінцевого продукту. Використання даного методу викликане тим, що на сьогоднішній день складно знайти персональний комп'ютер з вбудованим COM-портом, а різного роду датчики, що використовують RS-232, продовжують випускатися (тому що реалізація RS-232 у такому випадку більше

виправдана, ніж USB). Існує ряд вже готових розробок моста RS-232 → USB, і так само розроблених до них драйверів і програм-терміналів. Дані розробки мають ATMEL, Microchip, Philips, FTDI та інші компанії.

Другий метод полягає в повністю програмній реалізації протоколу безпосередньо в самому контролері. У цьому випадку відпадає необхідність використання проміжного контролера. При такому методі ускладнюється розробка кінцевого програмного продукту, оскільки розробнику необхідно досконально вивчити протокол обміну даними USB. Також розробнику необхідно вирішити ряд питань, пов'язаних з реалізацією всього програмного циклу, оскільки в цьому випадку необхідно встигнути обробити вхідний сигнал за 8-10 тактів роботи самого контролера. За цей час необхідно синхронізуватися з комп'ютером, зчитати дані із проміжних буферів, обробити їх, очистити проміжні буфери та сформувати підтвердження прийнятих даних. Програмна реалізація протоколу забирає основний час роботи та ресурси контролера, що обмежує виконання контролером інших операцій з обробки даних (зчитування даних з датчиків, їх обробка та підготовка до передачі по USB). Так само розробнику необхідно реалізувати драйвер для операційних систем і програму-термінал для обміну даними між комп’ютером та пристроєм. Швидкість передачі транзакцій так само становить 1,5 Мбіт/с, але швидкість передачі даних перебуває в межах 1,5 Мбіт/с, що приблизно в 10 разів швидше, ніж швидкість моста RS-232 → USB.

Третій метод є найбільш використовуваним і полягає в програмно-апаратній реалізації протоколу USB. У цьому випадку розробнику необхідно програмно описати протокол, що підготовлює дані, а передачу-прийом даних, синхронізацію, проміжні буфери, надати інтегрованим пристроям контролера. Швидкість передачі даних залежить від типу контролера і його властивостей. Найчастіше швидкість лежить у межах 1,5 Мбіт/с – 200 Мбіт/с. У деяких випадках швидкість обміну даними становить ~400 Мбіт/с.

У шині використовується диференціальний спосіб передачі сигналів по двох проводах D+ і D–. Сигнали синхронізації і дані кодуються по методу NRZI. У цьому кодуванні логічна 1 представляє собою зміну рівня на протилежний (фронт) протягом бітового інтервалу. Для низькошвидкісних і повношвидкісних пристроїв диференціальна 1 передається шляхом підтяжки лінії D– до напруги, більшою за 2,8 В, а лінії D+ до напруги, меншою за 0,3 В. При цьому лінії D+ і D– терміновані на стороні хоста (вхідного потоку) резисторами 15кОм, підключеними до землі. Диференціальний нуль передається шляхом підтяжки лінії D– до напруги менше 0,3 В, а лінії D+ до напруги більше 2,8 В. Приймач визначає диференціальну одиницю тільки в тому випадку, коли напруга на лінії D+ більше на 200 мВ, ніж на лінії D–, а диференціальний 0, коли напруга на лінії D+ менше на 200мВ, ніж на лінії D–.

Швидкість, яка використовується пристроєм, підключеним до конкретного порту, визначається хабом по рівнях сигналів на лініях D+ і D–, що зміщуються навантажувальними резисторами приймачів-передавачів: пристрої з низькою швидкістю «підтягують» до високого рівня лінію D–, з повною – D+. Передача по двох проводах у USB не обмежується диференціальними сигналами. Крім диференціального приймача кожен пристрій має лінійні приймачі сигналів D+ і D–, а передавачі цих ліній керуються індивідуально. Це дозволяє розрізняти більше двох станів лінії, що використовуються для організації апаратного інтерфейсу. Швидкість передачі даних вибирається розробником периферійного пристрою відповідно до потреб цього пристрою.

5.2.3. Спеціалізована мікросхема-перетворювач інтерфейсів FT232BM

Мікросхеми FT8U232AM, FT232BM – це перетворювачі USB у традиційний послідовний інтерфейс – можна встановлювати в USB-модемах, перетворювач COM-USB, сканерах штрих-коду, вимірювальній апаратурі – фактично в будь-яких пристроях, що раніше використовували порівняно повільні інтерфейси RS232, RS422, RS485. Мікросхема FT8U232AM здатна передавати дані в обидва боки зі швидкістю до 2000 кБіт/с, причому користувачеві не потрібно ніяких знань про пристрій і роботу USB. Програмні драйвера, що постачаються компанією FTDI, створюють враження, що обмін йде через звичайний COM-порт.

На рис. 5.36 представлена одна з можливих схем підключення мікросхеми FT232BM до USB. Живлення пристрою автономне.

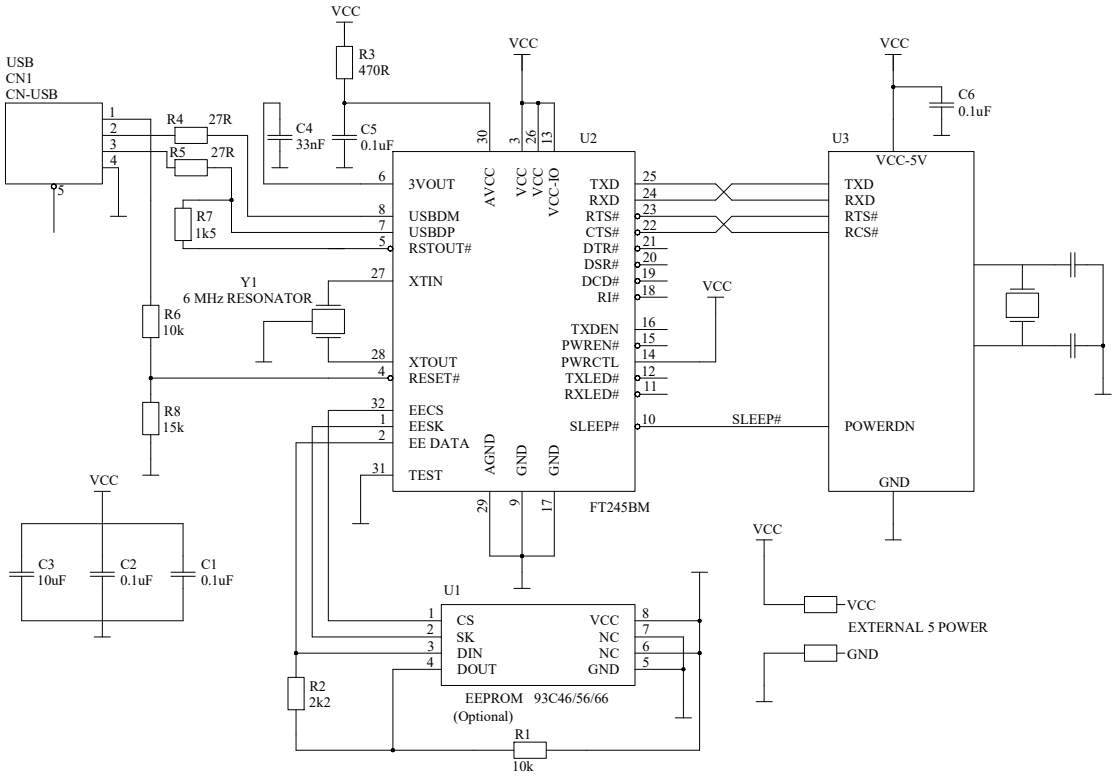


Рис. 5.36.

У даному пристрої немає необхідності стежити за струмом споживання, який у випадку живлення від USB не повинен перевищувати 500мА. У цьому прикладі зручно підключити вивід SLEEP# до одного з входів мікроконтролера, оскільки це дозволить довідатися про те, що контролер у режимі очікування і не здатний почати обмін даними. Вхід PWRCTL підтягнуть до живлення, що означає, що пристрій має автономне живлення.

На рис. 5.37 представлена схема підключення мікросхеми FT232BM до USB. Пристрій живиться від шини USB.

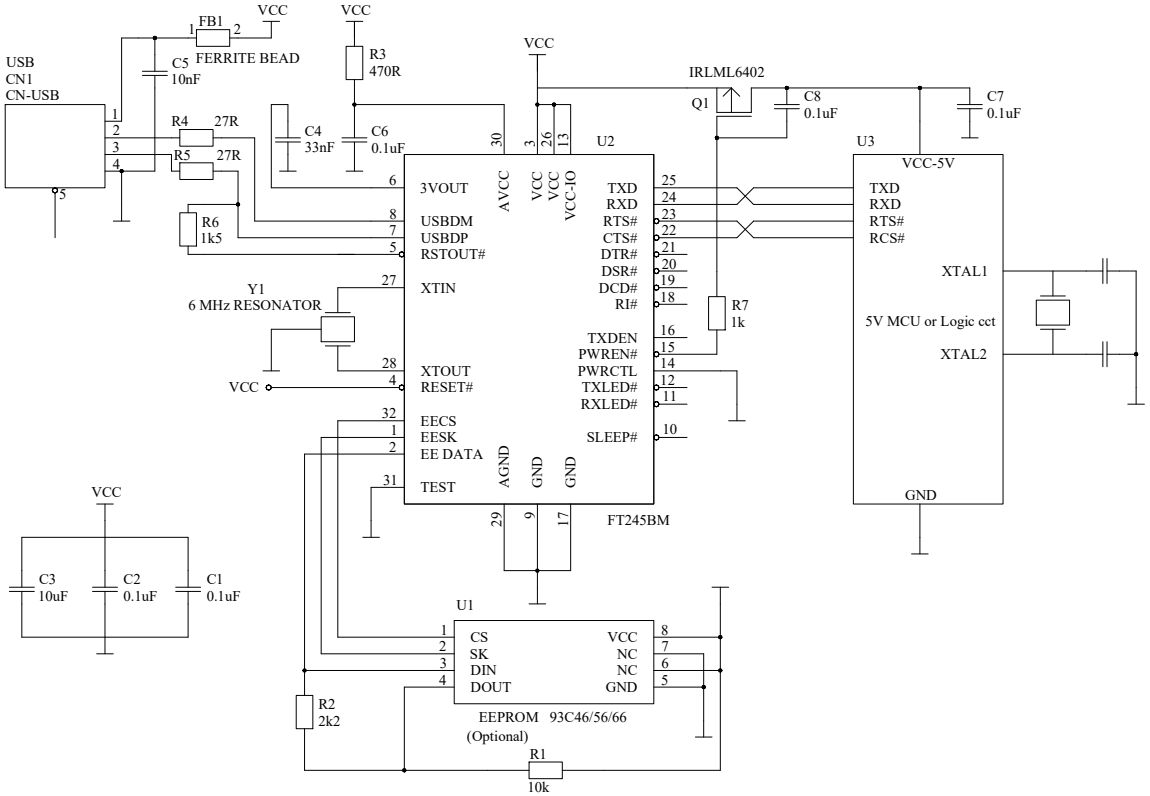


Рис. 5.37.

Вихід передавача UART мікросхеми (Tx) з'єднується з входом Rx мікроконтролера. Вихід передавача мікроконтролера з'єднується з входом приймача мікросхеми. У багатьох програмах досить цих двох з'єднань. Мікросхема FT232BM також має виходи, які є на стандартному COM-порту. При необхідності їх можна використовувати. Рівні напруг на виходах мікросхеми відповідають рівням ТТЛ, тому для сполучення з мікроконтролером немає необхідності використовувати спеціальні мікросхеми-перетворювачі рівнів.

Наявність виводів як у COM-порту дозволяє використовувати мікросхему для підключення модемів, принтерів, інших пристроїв до комп'ютера по шині USB. При установці спеціальних драйверів робота із шиною USB не буде відрізнятися від роботи із COM – портом. У наступному розділі докладно описані принципи програмування USB з використанням бібліотеки FTD2XX.

5.2.4. Мікропроцесорна система збору даних з інтерфейсом USB на базі мікросхеми FT232BM

Для того щоб система USB почала працювати, необхідно, щоб були завантажені драйвери хост-контролера (або контролерів, якщо їх декілька). При підключенні пристрою до шини USB, ОС Windows видає повідомлення «Виявлений новий пристрій» і, якщо пристрій підключається вперше, пропонує завантажити для нього драйвери (рис. 5.38).

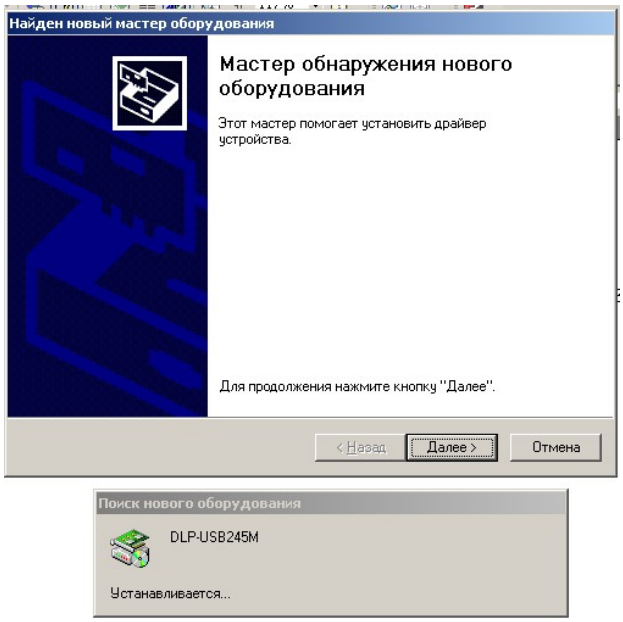


Рис. 5.38.

Натиснувши кнопку «Далі» можна перейти до наступного етапу установки драйвера (рис. 5.39 – рис. 5.42).

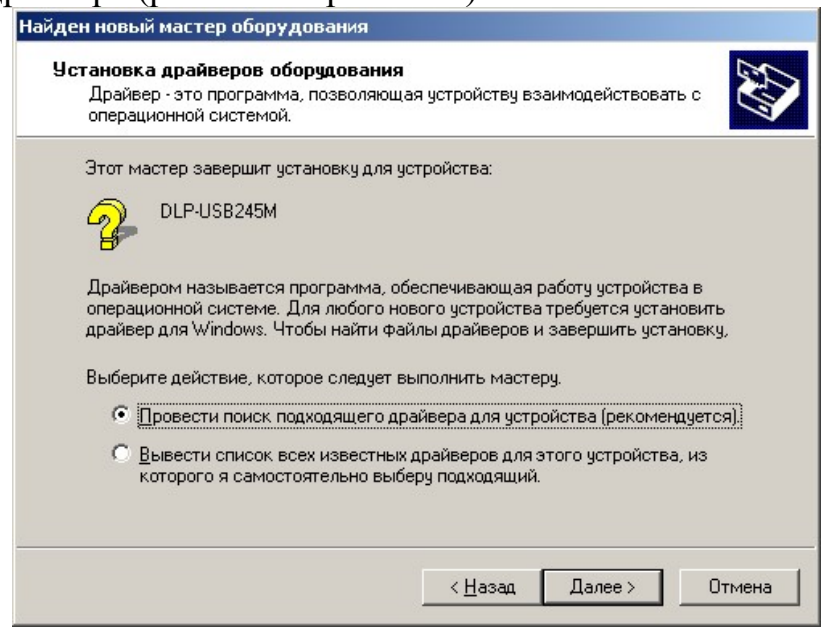


Рис. 5.39.

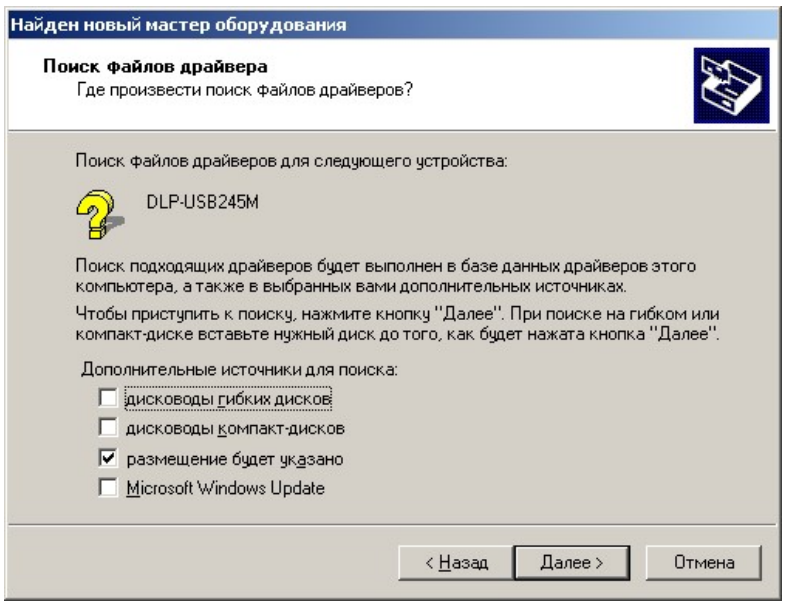


Рис. 5.40.

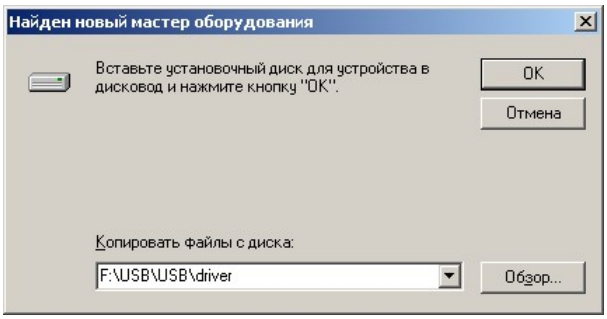


Рис. 5.41.

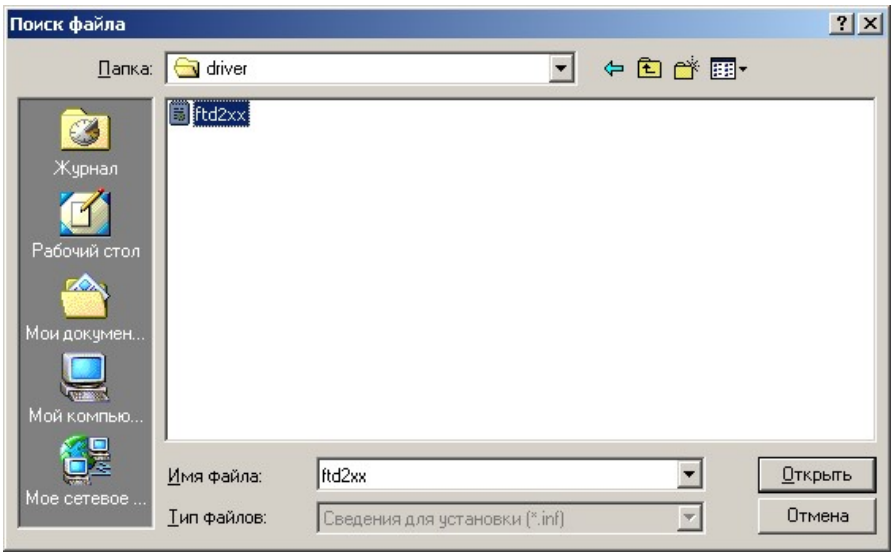


Рис. 5.42.

Натиснувши на кнопку «Далі» виконається установка драйвера, про що повідомить діалогове вікно на рис. 5.43.

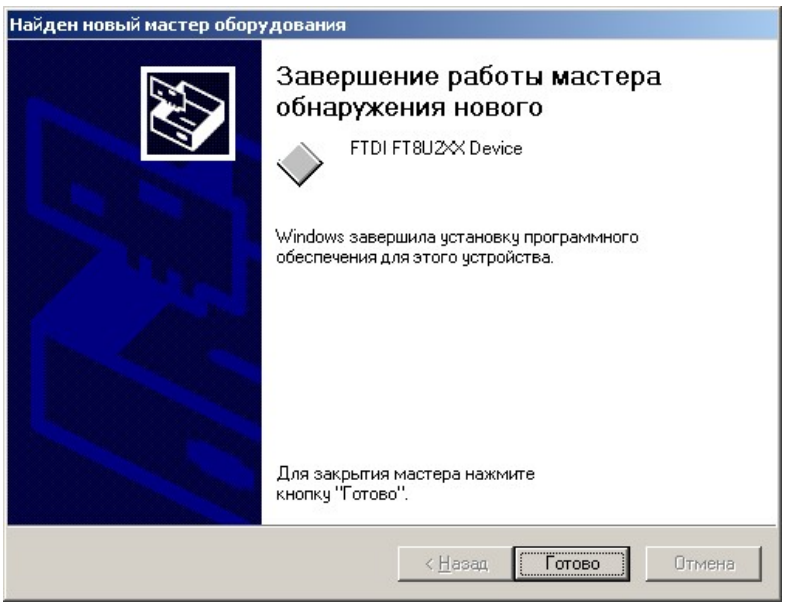


Рис. 5.43.

У диспетчері пристроїв з'явиться новий пункт (рис. 5.44).

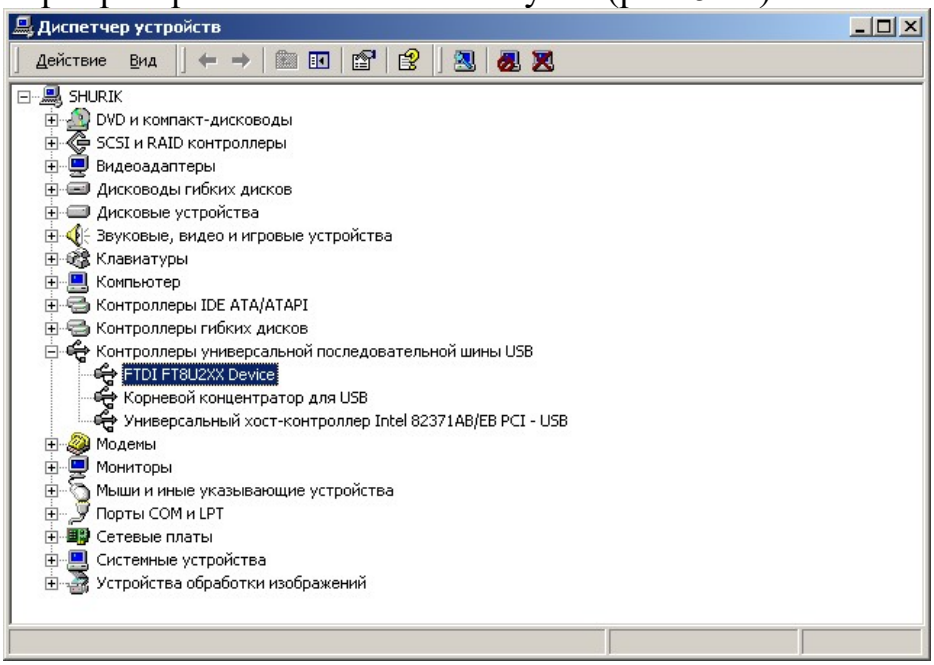


Рис. 5.44.

Разом із драйвером поставляються наступні файли (рис. 5.45).

Имя	Размер	Тип
FTD2XX	15 KB	C compiler header file
FTD2XX	16 KB	C compiler library file
D2XX Release Info	2 KB	Text file
D2XX_EX	20 KB	Документ Microsoft Word
FTD2XX.dll	68 KB	Компонент приложения
FTD2XXUN	1 KB	Параметры конфигурации
FTD2XXUN	397 KB	Приложение
ftd2xx	3 KB	Сведения для установки
FTD2XX.sys	25 KB	Системный файл

Рис. 5.45.

Компанія FTDI пропонує рішення, що не вимагає драйверів, які емулюють послідовний порт. Архітектура, названа її авторами D2XX, заснована на технології WDM. Програмування пристрою ведеться через стек USB і динамічну бібліотеку драйвера.

Для одержання інформації про підключений пристрій використовується функція, прототип якої має вигляд:

```
FT_STATUS FT_ListDevices(
PVOID pvArg1,
PVOID pvArg2,
DWORD dwFlags
);
```

ДО USB одночасно можуть бути підключені кілька пристроїв, і за допомогою цієї функції є можливість одержати кількість приєднаних пристроїв, їхні серійні номери і опис. При успішному виконанні функції повертається значення FT_OK. При виникненні помилки повертається код помилки. В залежності від значення прапорів (параметр dwFlags), через параметри pvArg1 і pvArg2 може повертатися різна інформація.

У найпростішому варіанті виклику цієї функції повертається кількість підключених пристроїв. У цьому випадку параметр dwFlags повинен приймати значення FT_LIST_NUMBER_ONLY. Перший параметр (pvArg1) інтерпретується як вказівник на змінну типу DWORD, у якій буде зберігатися значення кількості приєднаних пристроїв, а другий параметр повинен приймати значення NULL. Якщо параметр dwFlags має значення FT_LIST_BY_INDEX, то в параметрі pvArg1 передається номер пристрою, а в параметрі pvArg2 передається вказівник на буфер, у який буде поміщена інформація про відповідний пристрій. Яка саме інформація буде розміщена в буфері, вибирається установкою одного з прапорів (параметр dwFlags):

FT_OPEN_BY_SERIAL_NUMBER – повертається серійний номер пристрою;

FT_OPEN_BY_DESCRIPTION – повертається опис пристрою.

Якщо був переданий неправильний номер пристрою, функція поверне значення FT_DEVICE_NOT_FOUND.

Функція FT_ListDevices важлива тим, що дозволяє одержати інформацію про всі пристрої, підключених у даний момент. Це дозволяє вибрати пристрій, з яким буде організований обмін даними. Це важливо, оскільки часто користувач може підключити кілька однотипних пристроїв. При цьому виникнуть труднощі звернення до одного з них. Якщо ж у програмі передбачити можливість вибору визначеного пристрою (наприклад, за допомогою списку), то проблема легко вирішується. Нумерація пристроїв починається з 0. Тому для одержання інформації про кожне з підключених пристроїв досить збільшувати індекс на 1 і повторно викликати цю функцію. Другий спосіб полягає в тому, що як прапор

передається значення FT_LIST_ALL. У цьому випадку перший параметр (pvArg1) інтерпретується як масив вказівників на буфери, у яких буде розміщена інформація про підключені пристрої. Важливо, щоб останній елемент у масиві вказівників на буфери був рівним NULL. Другий параметр – вказівник на змінну типу DWORD, у якій буде зберігатися кількість підключених пристроїв. Наступний приклад показує різні способи використання функції FT_ListDevices.
Для роботи з пристроєм необхідно одержати його дескриптор. Це можна виконати за допомогою функції FT_Open, прототип якої має вигляд:

```
FT_STATUS FT_Open(
int iDevice,
FT_HANDLE *ftHandle
);
```

Першим параметром (iDevice) у функцію передається номер пристрою, що відкривається. Якщо підключений тільки один пристрій, значення цього параметра повинне бути рівне 0. Другий параметр (ftHandle) – вказівник на змінну типу FT_HANDLE, у якій буде зберігатися дескриптор відкритого пристрою. Через цей дескриптор буде здійснюватися доступ до пристрою. При успішному виконанні функції повертається значення FT_OK. Функція FT_Open може використовуватися для відкриття одного з декількох пристроїв шляхом установки параметра iDevice у 0, 1,2 і т.д. Однак немає можливості відкрити визначений пристрій, оскільки номери призначаються довільним образом. Для рішення цієї проблеми варто використовувати функцію FT_OpenEx, прототип якої має вигляд:

```
FT_STATUS FT_OpenEx(
PVOID pvArg1,
DWORD dwFlags,
FT_HANDLE *ftHandle
);
```

Значення першого переданого параметра (pvArg1) залежить від прапорів (другий параметр dwFlags). У будь–якому випадку цей параметр інтерпретується як вказівник на рядок, обмежений нулем. Другий параметр (dwFlags) може приймати одне з двох значень: FT_OPEN_BY_SERIAL_NUMBER – означає, що в параметрі pvArg1 міститься серійний номер пристрою; FT_OPEN_BY_DESCRIPTION – у параметрі pvArg1 міститься опис пристрою. Дескриптор пристрою повертається через вказівник типу FT_HANDLE (третій параметр ftHandle). Цей дескриптор буде використовуватися для доступу до пристрою. Для відкриття визначеного пристрою як параметр можна передати серійний номер або опис пристрою, попередньо отриманий за допомогою функції FT_ListDevices.

Після того, як робота з пристроєм закінчена, його необхідно закрити. Для цього використовується функція FT_Close, прототип якої має вигляд:

```
FT_STATUS FT_Close(FT_HANDLE ftHandle);
```

Єдиний параметр, переданий у функцію (ftHandle) – дескриптор відкритого пристрою. При успішному виконанні повертається значення FT_OK.

Ще одна функція, що дозволяє одержати інформацію про пристрій – це FT_GetDeviceInfo. Прототип цієї функції має вигляд:

```
FT_STATUS FT_GetDeviceInfo(
FT_HANDLE fth,
FT_DEVICE *ftd,
DWORD *id,
PVOID snum,
PVOID desc,
NULL);
```

Параметри функції:

fth – дескриптор відкритого пристрою, інформацію про який необхідно одержати;

ftd – вказівник на змінну типу FT_DEVICE, у якій зберігається тип пристрою;

id – вказівник на змінну типу DWORD, у якій зберігається номер пристрою;

snum – вказівник на буфер, у якому розташовується серійний номер пристрою;

desc – вказівник на буфер, у якому розташовується опис пристрою.

До основних операцій обміну даними відносяться запис даних у пристрій і зчитування даних із пристрою. Для передачі даних використовується функція FT_Write, прототип якої має такий вигляд:

```
FT_STATUS FT_Write(
FT_HANDLE ftHandle,
LPVOID lpBuffer,
DWORD dwBytesToWrite,
LPDWORD lpdwBytesWritten
);
```

Параметри функції мають наступні значення:

ftHandle – дескриптор відкритого пристрою, отриманий за допомогою функції FT_Open або FT_OpenEx;

lpBuffer – вказівник на буфер даних, що будуть передані в пристрій;

dwBytesToWrite – кількість байт для передачі;

lpdwBytesWritten – вказівник на змінну типу DWORD, у якій повертається кількість переданих байт.

Якщо функція виконалася успішно, повертається значення FT_OK, у противному випадку – код помилки.

Для зчитування даних із пристрою використовується функція FT_Read. Прототип цієї функції має такий вигляд:

```
FT_STATUS FT_Read(
FT_HANDLE ftHandle,
LPVOID lpBuffer,
DWORD dwBytesToRead,
LPDWORD lpdwBytesReturned
);
```

Параметри, що передаються у функцію:

ftHandle – дескриптор пристрою, з якого буде відбуватися зчитування даних;

lpBuffer – вказівник на буфер, у якому будуть розташовані зчитані дані;

dwBytesToRead – кількість байт, які повинні бути зчитані з пристрою;

lpdwBytesReturned – вказівник на змінну типу DWORD, у якій повернеться кількість зчитаних із пристрою байт.

Якщо функція виконалася успішно, повертається значення FT_OK, у протилежному випадку – код помилки.

Особливість функції FT_Read у тому, що вона не поверне значення доти, поки не буде зчитано кількість байт, зазначене в параметрі dwBytesToRead. Це може призвести до зависання програми, якщо підключений до комп'ютера пристрій виявиться непрацюючим. Є кілька способів вирішення цієї проблеми. Перший з них полягає у використанні функції FT_GetQueueStatus. За допомогою цієї функції можна довідатися кількість байт у черзі прийому. Прототип функції має вигляд:

```
FT_STATUS FT_GetQueueStatus(
FT_HANDLE ftHandle,
LPDWORD lpdwAmountInRxQueue
);
```

Перший параметр ftHandle – дескриптор пристрою, з якого виконується зчитування даних. Другий параметр lpdwAmountInRxQueue – вказівник на змінну типу DWORD, у якій повертається кількість байт у черзі прийому. Перед тим, як викликати функцію FT_Read, необхідно знати кількість байт у черзі прийому. Тому попередньо варто викликати функцію FT_GetQueueStatus, яка поверне кількість байт у черзі прийому, і це значення передати у функцію FT_Read.

Ще одна функція, яка може бути використана для одержання стану пристрою – FT_GetStatus. Прототип цієї функції має вигляд:

```
FT_STATUS FT_GetStatus(
FT_HANDLE ftHandle,
```

```
LPDWORD lpdwAmountInRxQueue,
LPDWORD lpdwAmountInTxQueue,
LPDWORD lpdwEventStatus
);
```

Параметри, що передаються у функцію:

ftHandle – дескриптор пристрою, інформацію про стан якого необхідно одержати;

lpdwAmountInRxQueue – вказівник на змінну типу DWORD, у якій повертається значення кількості байт у черзі прийому;

lpdwAmountInTxQueue – вказівник на змінну типу DWORD, у якій повертається значення кількості байт у черзі передачі;

lpdwEventStatus – вказівник на змінну типу DWORD, у якій повертається поточна подія.

Останній параметр може використовуватися для виявлення однієї з наступних подій:

FT_EVENT_RXCHAR – подія, зв'язана з прийомом даних у чергу передачі пристрою;

FT_EVENT_MODEM_STATUS – подія, зв'язана зі зміною сигналів стану модему (для мікросхеми FT232B(M)).

Використовуючи середовище розробки Visual Studio .NET, створимо програму, що виконує контроль температури деякого технологічного процесу. Дані про температуру приймаються через USB-порт і повинні відображатися у виді гістограми. Прийняте значення відповідає температурі. Для одержання даних необхідно відправити запит – команда 0xFF.

1. Запускаємо середовище розробки Visual Studio .NET. Створюємо новий проект із назвою TempControl. У нашому прикладі програма створюється на базі діалогового вікна.
2. У каталог, де розташовуються файли створеного проекту, скопіювати наступні файли:
   FTD2XX.h
   FTD2XX.lib
   FTD2XX.dll
   На початку файлу TempControlDlg.h, для підключення бібліотеки для роботи з USB необхідно додати наступні два рядки:

```
#include "FTD2XX.h"
#pragma comment(lib,"FTD2XX.lib")
```

3. Проектуємо діалогове вікно. На форму необхідно винести елементи керування Static Text, у якому буде відображатися значення температури в графічному виді; дві кнопки (Button) для підключення і відключення пристрою; список (Combo Box), у

якому будуть відображатися номери підключених пристроїв. На рис. 5.46 представлений зовнішній вигляд діалогового вікна.

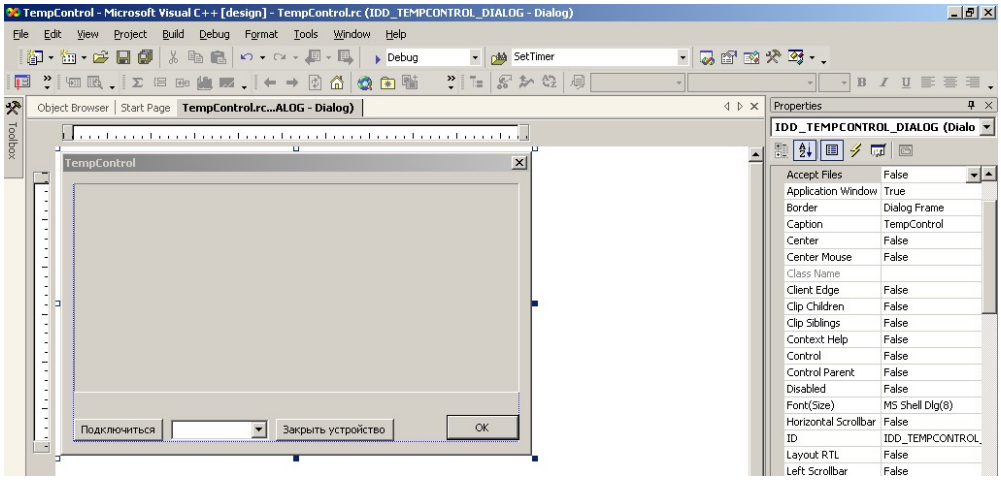


Рис. 5.46.

Для елемента керування Static Text (текстове поле) установити поле Sunken на сторінці властивостей у значення True. Навколо елемента керування з'явиться рельєфна рамка. Також необхідно змінити ID цього елемента керування на IDC_STATIC1 і сформувати необхідний розмір вікна. Прив'язуємо до елемента керування текстове поле (IDC_STATIC1) змінну m_graph. Для цього необхідно виділити елемент керування і з контекстного меню вибрати команду «Add Variable». З'явиться діалогове вікно, показане на рис. 5.47. У полі «Variable name» задати ім'я змінної.

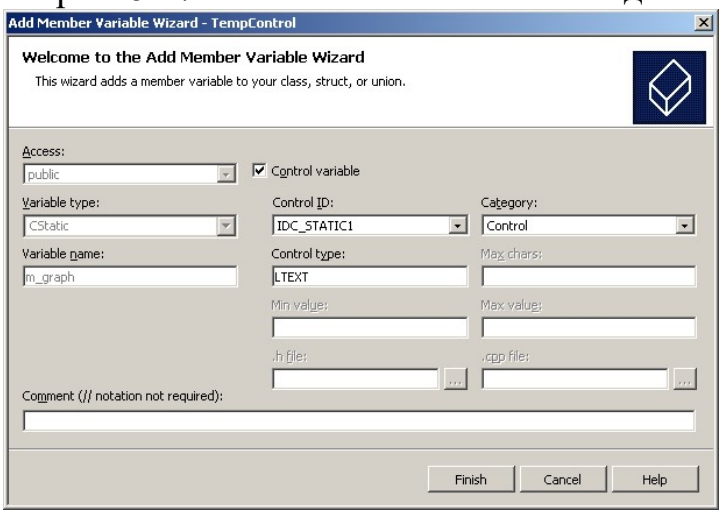


Рис. 5.47.

4. На сторінці властивостей елемента керування «Combo Box» (IDC_COMBO1) установити в поле «Type» значення «Drop List». Прив'язуємо до цього елемента керування змінну m_devices.
5. Додаємо змінну – член класу CTempControlDlg, у якій буде зберігатися дескриптор порту. Для цього у вікні «Class View» виділяємо клас «CTempControlDld» і з контекстного меню, у підменю «Add» вибираємо «Add Variable» (рис. 5.48).

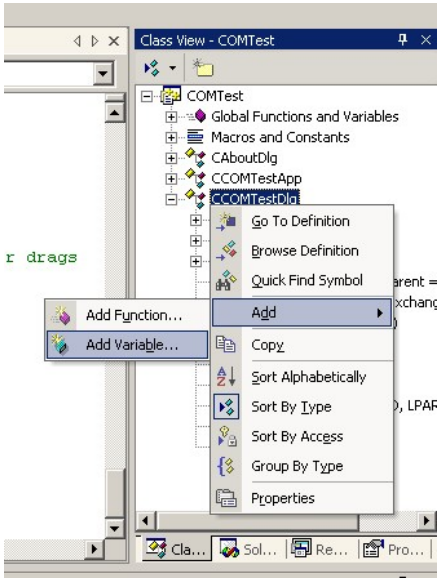


Рис. 5.48.

Ім'я змінної повинне бути m_hUSB, тип змінної – FT_HANLDE, тип доступу (Access) – public.

6. Для збереження отриманих даних про температуру використовується змінна m_temp, яка має тип CByteArray. Ця змінна – член класу CCOMTestDlg. Для того, щоб додати змінну, у вікні «Class View» виділити клас «CCOMTestDlg», викликати контекстне меню і з підменю «Add» вибрати команду «Add Variable».
7. При запуску програми в список необхідно додати всі підключені до USB пристрою, щоб надалі вибрати необхідне. Для цього наприкінці функції OnInitDialog класу CTemControlDlg, що викликається перед появою діалогу на екрані, додати наступний код:

```
FT_STATUS fts;
DWORD num;
// одержання кількості підключених пристроїв
fts=FT_ListDevices(&num, NULL, FT_LIST_NUMBER_ONLY);
if(fts!=FT_OK)
{
        MessageBox("Неможливо знайти пристрій!");
        return FALSE;
}
for(int i=0;i<num;i++)          //
{
        char desc[16];
   // у циклі одержуємо інформацію про кожен підключений пристрій
   fts=FT_ListDevices((PVOID)i, desc, FT_LIST_BY_INDEX |
FT_OPEN_BY_DESCRIPTION);
        if(fts==FT_OK)
        {
        CString str(desc);
```

```
            // додавання в список інформації про підключені пристрої
            m_devices.AddString(str);
            }
            else MessageBox("Пристрій не знайдений!");
        }
        m_devices.SetCurSel(0);
        m_hUSB=NULL;
        m_temp.RemoveAll();          // очищення масиву
```

8. Додаємо оброблювач події натискання на кнопку «Підключитися». Необхідно відкрити пристрій, номер якого обраний у списку. Функція – оброблювач події має вигляд:

```
void CTempControlDlg::OnBnClickedButton1()
{
    FT_STATUS fts;
    char desc[16];
    m_devices.GetLBText(m_devices.GetCurSel(), desc);
    // одержання опису пристрою з виділеного елемента списку
    fts=FT_OpenEx(desc, FT_OPEN_BY_DESCRIPTION, &m_hUSB);
    // відкриття пристрою по його опису
    if(fts!=FT_OK) m_status.SetWindowText("Стан: Неможливо відкрити
обраний пристрій!");
    else
    {
        m_status.SetWindowText("Стан: Пристрій відкритий!");
        SetTimer(1,200,NULL);       // запуск таймера
    }
    m_devices.EnableWindow(0);
}
```

9. Додаємо оброблювач події натискання на кнопку «Закрити пристрій». Функція – оброблювач події має вигляд:

```
void CTempControlDlg::OnBnClickedButton2()
{
    if(m_hUSB!=NULL)
    {
        FT_Close(m_hUSB); // закриття пристрою
        m_hUSB=NULL;
        m_status.SetWindowText("Стан: Пристрій закритий!");
        m_devices.EnableWindow(1);
        KillTimer(1);  // зупинка таймера
    }
    else m_status.SetWindowText("Стан: Пристрій уже закритий!");
}
```

10. Додаємо оброблювач події таймера. Для цього зі сторінки властивостей (Properties) викликати вікно «Messages» і напроти позиції «WM_TIMER» вибрати «OnTimer» (рис. 5.49).

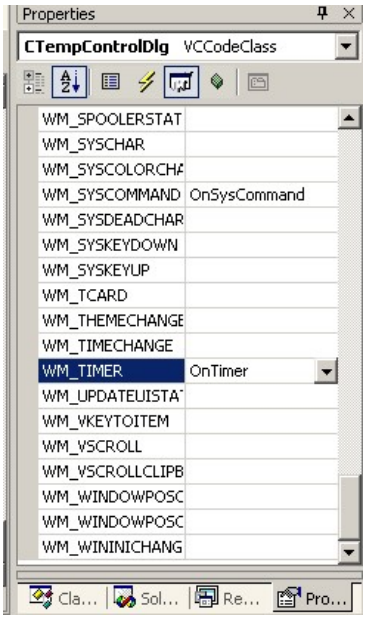


Рис. 5.49.

11. Додаємо у функцію – оброблювач події таймера код, як показано нижче:

```
void CTempControlDlg::OnTimer(UINT nIDEvent)
{
 FT_STATUS fts;
 DWORD sb,rb=0;
 unsigned char buff[2];
 buff[0]=0xFF;
 fts=FT_Write(m_hUSB, buff,1,&sb);          // запис даних у пристрій
 if((fts!=FT_OK) || (sb!=1))
 {
          m_status.SetWindowText("Стан: Помилка при передачі даних!");
          return;
 }
          // одержання кількості байт у черзі прийому
          fts=FT_GetQueueStatus(m_hUSB,&rb);
          if(fts!=FT_OK)
          {
 m_status.SetWindowText("Стан: Стан черги не визначено!");
          return;
          }
          if(rb<1)          // якщо в черзі прийому немає даних
          {
 m_status.SetWindowText("Стан: Дані про температуру відсутні!");
          // виводимо повідомлення
          return;
          }
 fts=FT_Read(m_hUSB, buff, 1, &rb);          // зчитування даних із пристрою
 if((fts!=FT_OK) | (rb!=1))
 {
          m_status.SetWindowText("Стан: Помилка при прийомі даних!");
```

```
        return;
    }
    m_temp.Add(buff[0]);          // додавання отриманих даних у масив
    CString str;
    str.Format("Поточна температура: %d", buff[0]);
    m_status.SetWindowText(str);
    DrawGraph();          // виклик функції малювання гістограми
    CDialog::OnTimer(nIDEvent);
}
```

12. Реалізуємо функцію void DrawGraph(void). Для цього її необхідно зробити членом класу CTempControlDlg. Реалізація функції має такий вигляд:

```
void CTempControlDlg::DrawGraph(void)
{
    CClientDC dc(&m_graph);   // створення контексту для малювання
    CPen p(PS_SOLID,2,RGB(200,0,0));          // створення пера
    CPen pp(PS_SOLID,1,RGB(0,100,0));
    CRect rect;
    CRgn rgn;
    CPen* op;
    CBrush br(RGB(180,180,180));       // створення кисті сірого кольору
    m_graph.RedrawWindow();            // перемальовування вікна
    m_graph.GetClientRect(&rect);      // одержання розмірів вікна
    rgn.CreateRectRgn(0,0,rect.Width(),rect.Height());
    op=dc.SelectObject(&pp);    // вибір пера в контекст
    dc.FillRect(&rect,&br);        // заповнення області малювання сірим кольором
    for(int x=20;x<rect.Width();x=x+20)
    {
        // малювання вертикальних ліній
        dc.MoveTo(x,0);
        dc.LineTo(x,rect.Height());
    }
    dc.SetTextColor(RGB(0,0,0)); // установка кольору тексту
    dc.SetBkColor(RGB(180,180,180)); // установка кольору фону
    br.DeleteObject();
    br.CreateSolidBrush(RGB(0,0,160));          // створення синьої кисті
    for(int y=20;y<rect.Height();y=y+20)
    {
        // малювання горизонтальних ліній
        dc.MoveTo(0,rect.Height()–y);
        dc.LineTo(rect.Width(),rect.Height()–y);
        CString s;
        s.Format("%d",y);
        // малювання текстових міток
        dc.TextOut(rect.Width()–30,rect.Height()–y,s);
    }
    dc.SelectObject(&p);
    dc.SelectClipRgn(&rgn);      // обмеження області малювання
    if(rect.Width()<=m_temp.GetCount()) m_temp.RemoveAll();
    for(int i=0;i<m_temp.GetCount();i++)
    {
        CRect r(i*5,rect.Height(),i*5+3,rect.Height()–m_temp.GetAt(i));
```

```
            dc.FillRect(&r,&br);  // малювання гістограми
        }
        dc.SelectObject(op);
    // якщо елементів у масиві більше, ніж розмір області перемальовування
    // то всі дані з масиву знищуються
        if(rect.Width()/5<m_temp.GetCount()) m_temp.RemoveAll();
    }
```

Після компіляції проекту і підключення до пристрою, з'явиться діалогове вікно, представлене на рис. 5.50.

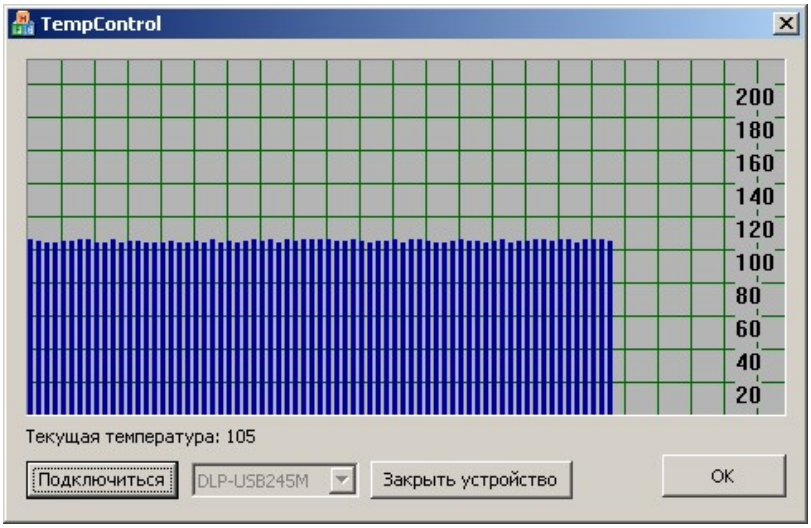


Рис. 5.50.

Завдання до розділу 5

1. Написати програму, яка посилає в комп'ютер номер натиснутої кнопки.
2. Написати програму, яка на семисегментних індикаторах виводить прийняті значення (цифри від 0 до 9). Після прийому чергового байта, він виводиться на наступний семисегментний індикатор.
3. Написати програму, яка формує синусоїдальний сигнал і відправляє значення в комп'ютер.
4. Написати програму, яка після прийому значення з комп'ютера (від 0 до 9) встановлює відповідну яскравість / частоту мерехтіння світлодіодів.
5. Написати програму, яка виконує аналого-цифрове перетворення і відправляє отримане значення в комп'ютер.
6. Написати програму, яка дозволяє виміряти швидкість реакції людини. Після натискання на першу кнопку всі світлодіоди гаснуть, і через випадковий проміжок часу (не більше 5 секунд) вони спалахують. Користувач натискає другу кнопку й на семисегментних індикаторах відображається час (в мілісекундах) від моменту запалювання світлодіодів до моменту натискання на кнопку.

РОЗДІЛ 6. ПРИНЦИПИ ОРГАНІЗАЦІЇ ЗБОРУ ДАНИХ ТА ДИСТАНЦІЙНОГО КЕРУВАННЯ ОБЛАДНАННЯМ ПО МЕРЕЖІ ETHERNET

6.1. Мережева модель OSI та протоколи обміну даними

В більшості програм використовуються схожі принципи роботи в мережах, які однакові для Internet та локальних мереж. Одна з програм очікує доки інша встановить з нею зв'язок по мережі. Одночасно інша програма, яка знаходиться на другому комп'ютері (хоча і не обов'язково) намагається встановити зв'язок з першою. Після встановлення зв'язку між двома програмами вони можуть обмінюватися повідомленнями в прямому та зворотному напрямку (рис. 6.1).

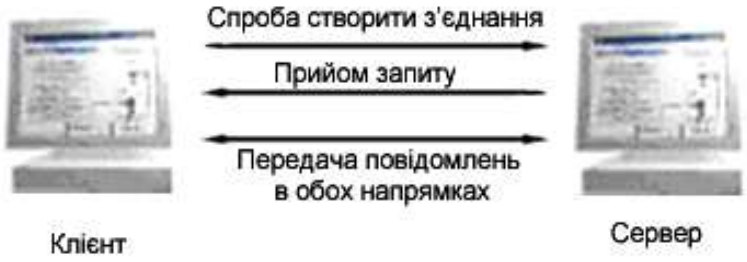


Рис. 6.1.

Таке з'єднання представляє собою двосторонній канал зв'язку, в якому обидві сторони можуть приймати та передавати інформацію. Коли одна зі сторін завершує діалог, вона розриває з'єднання. Інша сторона може виявити цей факт та розірвати зв'язок на своєму кінці. Описаний процес є ілюстрацією базової схеми мережевого з'єднання між двома або декількома програмами. Такий принцип мережевих комунікацій використовується в рамках протоколу TCP/IP, який є основним протоколом зв'язку в мережі Internet. Багато протоколів є варіаціями описаної схеми. В деяких протоколах (наприклад UDP) не відбувається встановлення каналу зв'язку між програмами, а отже не контролюється достовірність передачі даних. Одна програма посилає повідомлення, а інша програма повинна прийняти всі повідомлення і виконати пов'язані з ними дії.

У мережі може працювати різноманітне за технологією обладнання, від якого залежать можливості мережі: продуктивність, надійність і т.п., і різноманітне програмне забезпечення: мережеві операційні системи, програми, від яких також залежать можливості мережі (прозорість, безпека). Мережа – це з'єднання поєднання різного обладнання, різного програмного забезпечення, а отже проблема сумісності (сумісності різних технологій обладнання, або різних типів операційних систем) залишається однією з найбільш гострих.

Завдання ускладнюється необхідністю прийняти безліч «угод» між цими вузлами. Це угоди про рівні і формі електричних сигналів, способі

визначення довжини інформації, методах контролю її на помилки і достовірності. Іншими словами, правила повинні бути прийняті для всіх рівнів, починаючи від найнижчого – рівня передачі бітів інформації – до самого високого, що реалізовує сервіс користувача мережі.

Використовуючи прийом декомпозиції, можна розбити загальну задачу на кілька модулів, чітко визначити функції як кожного модуля, який вирішує якусь цілком певну задачу, так і інтерфейсів між цими модулями, який забезпечує їх взаємодію. Для декомпозиції використовують, як правило, багаторівневий підхід. Набори функцій всіх рівнів та інтерфейсів між цими рівнями чітко визначені і, за рахунок цього, вони відносно незалежні. Саме тому такі моделі мають дуже важливу перевагу – при заміні того чи іншого модуля, це мало позначиться на загальній роботі системи в цілому. Модулі нижнього рівня можуть вирішувати всі проблеми передачі сигналів між двома сусідніми вузлами, а модулі більш високого рівня можуть організовувати транспортування повідомлень в межах всієї мережі, користуючись засобами нижнього рівня.

Над створенням єдиної, стандартної моделі, якої б дотримувалися всі виробники мережевого устаткування, на початку 80-х років працювали такі провідні організації по стандартизації, як ISO, ITU-T та інші. Рнзультатом роботи стало створення еталонної моделі, яка отримала назву Open System Interconnection Reference Model, або модель OSI.

У моделі OSI під відкритою системою розуміється мережевий пристрій, готовий взаємодіяти з іншими мережевими пристроями з використанням стандартних правил, що визначають формат, зміст і значення повідомлень які приймаються і відправляються. Якщо дві мережі побудовані з дотриманням принципів відкритості, то це дає такі переваги:

– можливість побудови мережі з апаратних і програмних засобів різних виробників, що дотримуються однакових стандартів;

– можливість заміни окремих компонентів мережі іншими, більш досконалими, що дозволяє мережі розвиватися з мінімальними витратами;

– можливість легкого об'єднувати одну мережу з іншою;

– простота освоєння і обслуговування мережі.

Рішення проблем організації зв'язку вимагає розбиття одного завдання на кілька менших, для збільшення швидкості обробки і передачі даних. У моделі OSI засоби взаємодії поділяють на сім рівнів:

7-й – прикладний (Application),

6-й – представлення (Presentation),

5-й – сеансовий (Session),

4-й – транспортний (Transport),

3-й – мережевий (Network),

2-й – канальний (Data Link),

1-й – фізичний (Physical).

Таблиця 6.1.

Комп’ютер 1		Обмін даними	Комп’ютер 1	
7	Прикладний	↔	7	Прикладний
6	Представлення	↔	6	Представлення
5	Сеансовий	↔	5	Сеансовий
4	Транспортний	↔	4	Транспортний
3	Мережевий	↔	3	Мережевий
2	Канальний	↔	2	Канальний
1	Фізичний	Середовище передачі ↔	1	Фізичний

Кожен рівень відповідає за певну сторону взаємодії мережевих пристроїв. Передача даних від одного комп'ютера до іншого починається з сьомого рівня – дані передаються з рівня на рівень (табл. 6.1). При прийомі дані передаються вгору від першого рівня (фізичного) до сьомого (прикладного).

Модель OSI описує тільки системні засоби взаємодії, що реалізовуються операційною системою, системними утилітами, системними апаратними засобами.

Фізичний (Physical) рівень відповідає за передачу бітів по фізичних каналах зв'язку, таким, наприклад, як коаксіальний кабель, кручена пара, оптичне волокно або цифровий територіальний канал. До цього рівня відносяться такі характеристики фізичних середовищ передачі даних, як смуга пропускання, перешкодозахищеність, хвильовий опір та ін. На цьому ж рівні визначаються характеристики електричних сигналів, що передають інформацію, наприклад, вимоги до крутизни фронтів імпульсів, рівні напруги або струму, тип кодування, швидкість передачі сигналів.

Канальний рівень перевіряє доступність середовища передачі і реалізує механізми виявлення й коригування помилок. Для роботи на канальному рівні біти групуються в набори – кадри. Спеціальна послідовність бітів поміщається в початок і кінець кожного кадру для його виділення. Всі байти обробляються певним способом, і до кадру додається контрольна сума. У такому вигляді кадр приходить по мережі до одержувача, який знову обчислює контрольну суму отриманих даних, порівнює результат з контрольною сумою з кадру. Якщо вони збігаються, то кадр правильний і одержувач його приймає.

Мережевий рівень (Network layer) служить для утворення єдиної транспортної системи, що об'єднує декілька мереж. При цьому ці мережі можуть використовувати абсолютно різні принципи передачі інформації і бути організованими абсолютно довільно за структурою.

Транспортний рівень забезпечує додаткам або верхнім рівням (прикладному і сеансовому) моделі OSI передачу даних із рівнем надійності, яка їм потрібна. Модель OSI визначає п'ять класів сервісу, які надає транспортний рівень. Всі ці види відрізняються якістю своїх послуг: терміновістю, можливістю відновлення перерваного зв'язку, наявністю

засобів мультиплексування, а головне – здатністю до виявлення і виправлення помилок передачі, таких як спотворення, втрата і дублювання пакетів.

Сеансовий рівень працює з діалогом мережі (сесіями), і забезпечує управління ними: фіксує, яка сторона мережевих "переговорів" є активною в даний момент, надає засоби синхронізації. Ці засоби дозволяють вставляти контрольні точки в довгі передачі, щоб у разі відмови можна було б повернутися назад до останньої контрольної точки і не починати все з початку. Сеансовий рівень відповідає за організацію і підтримку з'єднань між сесіями, адміністрування і безпеку мережі.

Представницький рівень відповідає за форму подання переданої по мережі інформації, не змінюючи при цьому її змісту. За рахунок рівня представлення інформація, що передається прикладним рівнем однієї системи, завжди зрозуміла прикладному рівню іншої системи. За допомогою засобів даного рівня протоколи прикладних рівнів можуть подолати синтаксичні відмінності в представленні даних, або ж відмінності в кодах символів, наприклад кодів ASCII і EBCDIC. На цьому рівні може виконуватися шифрування і дешифрування даних, завдяки якому секретність обміну даними забезпечується відразу для всіх прикладних служб.

Прикладний рівень відповідає за доступ додатків в мережу. Завданнями цього рівня є копіювання файлів, обмін поштовими повідомленнями і керування мережею.

6.2. Особливості використання Windows Sockets для організації обміну даними по мережі

Специфікацією Windows Sockets (Winsock) визначається інтерфейс бібліотеки, що динамічно завантажується, файл якої, як правило, називається winsock.dll або wsock32.dll. Функції цієї бібліотеки реалізуються розробниками. Цей інтерфейс забезпечив розробників програм загальним підходом для реалізації мережевих комунікацій, незалежно від характеру мережі, яка використовується, та програмного забезпечення.

При програмуванні з використанням Winsock важливим поняттям є гніздо (socket), яке є базовим об’єктом, що використовується в програмах для виконання більшості мережевих з’єднань. Кожному комп’ютеру привласнена числовий адрес, що називається IP-адресом, який записується у вигляді чотирьох чисел, розділених крапками. Нехай, наприклад, один комп’ютер має адрес 192.168.7.62. Програми, що працюють на ньому, хочуть за допомогою Winsock з’єднатися з іншим комп’ютером. Виникає питання: якщо по адресу 192.168.7.62 надходить запит, яка програма повинна його обробляти?

Кожен запит, що надходить у комп’ютер, місить номер порту – число від 1024 і вище, яке вказує, якій програмі призначений запит. Деякі номери портів зарезервовані для стандартного використання. Наприклад, порт 80

традиційно використовується серверами Web для отримання запитів на документи від таких програм, як Internet Explorer та інших.

Робота з Winsock основана на з'єднанні: дві програми встановлюють зв'язок за допомогою гнізд на кожному кінці, а потім передають та приймають дані по цьому з'єднанню. Деякі програми можуть передавати дані без встановлення з'єднання, але в цьому випадку немає гарантії, що дані будуть передані за призначенням.

Технологія гнізд дозволяє організувати зв'язок між програмами по аналогії з процесами запису та зчитування файлів. Однак для організації з'єднання між програмами необхідна наявність деякої іншої інформації, ніж для відкриття файлу. Для відкриття файлу потрібно знати його назву та розташування. Для відкриття гніздового з'єднання потрібно знати IP – адрес комп'ютера, на якому виконується програма, з якою потрібно встановити зв'язок, а також адрес порту, який прослуховується цією програмою. Схема мережевих комунікацій представлена на рис. 6.2.

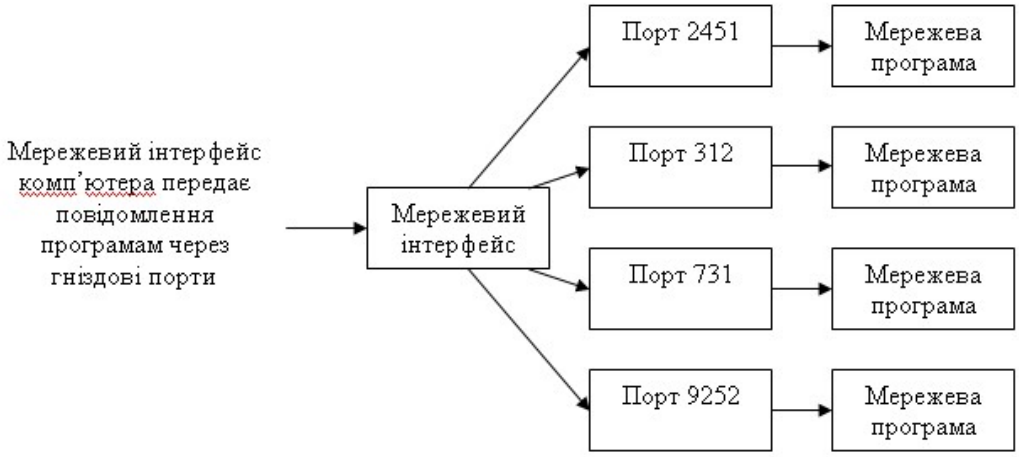


Рис. 6.2.

Зауважимо, що лише одна програма може прослуховувати повідомлення, які надходять на заданий порт деякого комп'ютера. Хоча на одному і тому ж комп'ютері відразу декілька програм можуть очікувати запитів на встановлення зв'язку, однак кожна з них буде прослуховувати окремий порт.

При створенні програм в середовищі Visual Studio .NET достатньо організувати в ньому підтримку мережевих комунікацій за допомогою класів інтерфейсу Winsock. Базовий клас цього інтерфейсу, CAsyncSocket, забезпечує повну підтримку гніздового зв'язку, що керується повідомленнями. Можна створити свій власний похідний клас, який буде перехоплювати відповідні повідомлення і виконувати відповідні дії. Клас CSocket є похідним від CAsyncSocket. Він інкапсулює в собі та спрощує деякі функціональні можливості базового класу. В табл. 6.2 представлені методи класу CAsyncSocket та їх призначення.

Таблиця 6.2.

Метод	Призначення
Accept	Обробляє запит на з'єднання, який надходить на гніздо, заповнює його інформацією про адрес
AsyncSelect	Організує посилку повідомлення Windows при переході гнізда в стан готовності
Attach	Зв'язує дескриптор гнізда з екземпляром класу CAsyncSocket, щоб мати можливість сформувати з'єднання з іншим комп'ютером
Bind	Асоціює адрес з гніздом
Close	Закриває гніздо
Conect	Підключає гніздо до віддаленого адресу та порту
Create	Завершує процес ініціалізації, початий конструктором
Detach	Розриває з'єднання з гніздом із заданим дескриптором
FromHandle	Повертає вказівник на об'єкт класу CAsyncSocket дескриптору, з яким він пов'язаний
GetLastError	Повертає код помилки гнізда. Виклик цього методу виконується після виявлення помилки, щоб з'ясувати її причину
GetPeerName	Повертає IP адрес та номер порту віддаленого гнізда, з яким пов'язаний об'єкт, або заповнює цією інформацією структуру гнізда
GetSockName	Повертає IP адрес та номер порту об'єкту this або заповнює цією інформацією структуру гнізда
GetSockOpt	Повертає поточні параметри гнізда
Listen	Примушує гніздо слідкувати за запитами на з'єднання
OnAccept	Обробляє повідомлення Windows, яке формується при прийомі гніздом запиту на з'єднання. Часто перевизначається в похідних класах
OnClose	Обробляє повідомлення Windows, яке формується при закритті гнізда. Часто перевизначається в похідних класах
OnConnect	Обробляє повідомлення Windows, яке формується при встановленні з'єднання, або при невдачі його встановлення. Часто перевизначається в похідних класах
OnRecieve	Обробляє повідомлення Windows, яке формується при появі даних, які можна зчитати за допомогою функції Receive(). Часто перевизначається в похідних класах
OnSend	Обробляє повідомлення Windows, яке формується при готовності гнізда прийняти дані, що передаються за допомогою Send(). Часто перевизначається в похідних класах
Receive	Зчитує дані з віддаленого гнізда, до якого підключене задане гніздо
RecieveFrom	Зчитує дейтаграму (пакет в мережі передачі даних, що передається через мережу без встановлення логічного з'єднання) з віддаленого гнізда без встановлення зв'язку
Send	Передає дані віддаленому гнізду
SendTo	Передає дейтаграму без встановлення зв'язку
SetSockOpt	Встановлює параметри гнізда
ShutDown	Залишає гніздо відкритим, але попереджає подальші виклики Send() або Receive()

Перед використанням будь-якого із класів інтерфейсу Winsock необхідно ініціалізувати відповідне середовище для своєї програми. Ця операція виконується за допомогою функції AfxSocketInit при ініціалізації самої програми. Ця функція приймає в єдиного необов'язкового аргументу структуру WSADATA. Якщо передати цю структуру при виклику функції AfxSocketInit, вона буде заповнена інформацією про поточну версію інтерфейсу Winsock. Якщо інформація, що передається в структурі, не потрібна, можна не задавати її в якості параметру при виклику функції AfxSocketInit:

```
BOOL CTestNetworkApp::InitInstance()
{
        ...
        if (!AfxSocketInit())   // ініціалізація Winsock
        {
                AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
                // повідомлення про помилку
                return FALSE;           // вихід з програми
        }
        ...
}
```

При використанні майстрів Visual C++ для створення оболонки проекту (рис. 6.3), в яких буде вказана необхідність підтримки інтерфейсу Winsock, функція AfxSocketInit буде автоматично додана в програмний код.

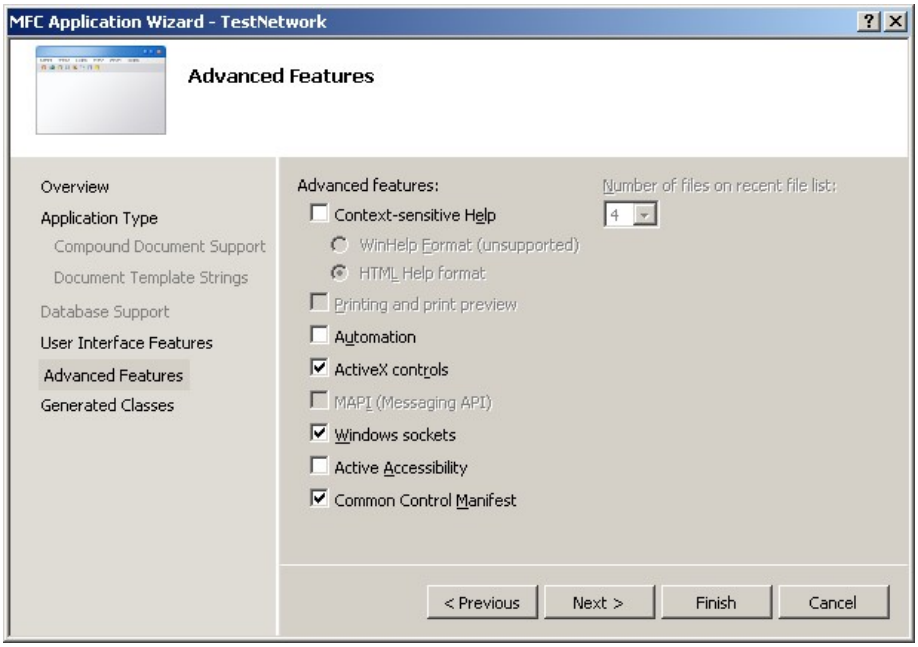


Рис. 6.3.

Для створення гнізда, яке буде використовуватися в програмі, необхідно перш за все створити змінні-члени класу CAsyncSocket.

```
class CTestNetworkDlg : public CDialog
```

```
{
public:
        …
        CAsyncSocket m_mySock;
…
}
```

Перед тим, як використовувати гніздо, потрібно викликати метод Create. Цей метод виконує створення гнізда та підготовку його до роботи. Синтаксис виклику функції Create визначається конкретними потребами програми. Якщо гніздо буде використовуватися для зв'язку з іншими програмами, які виступають ініціаторами з'єднання (програмами-клієнтами), при виклику методу Create не потрібно передавати ніяких параметрів.

```
if(m_mySock.Create())
{
        // продовження
}
else // обробка помилок
```

Однак якщо гніздо призначене для прослуховування каналу зв'язку в очікуванні виклику зі сторони іншої програми (сервера), при створенні гнізда необхідно вказати номер порта, який буде прослуховуватися.

```
if(m_mySock.Create(7000))
{
        // продовження
}
else // обробка помилок
```

Виклик методу Create може супроводжуватися іншими параметрами, наприклад, вказівником типу створюваного гнізда, повідомленнями, на яке гніздо повинне реагувати, а також адресом порту, який повинен прослуховуватися. Можна створювати два типи гнізд: потокові, або TCP-гнізда, та дейтаграмні, або UDP-гнізда. Потокові гнізда основані на створенні логічного з'єднання і містять вбудовані функції, які забезпечують гарантовану доставку повідомлень. Гнізда-дейтаграми не створюють логічного з'єднання. Для створення потокового гнізда необхідно при виклику методу Create задати в якості другого аргументу параметр SOCK_STREAM. Для створення гнізда-дейтаграми в якості другого аргументу слід задати параметр SOCK_DGRAM. Якщо створюється програма, яка буде працювати на комп'ютері з декількома мережними картами, при створенні гнізда потрібно вказати адрес порту, який буде прослуховуватися. Адрес порту задається в якості четвертого параметра для класу CAsyncSocket, або третього для класу CSocket. Адрес передається у вигляді строки, наприклад «192.168.7.62».

Після створення гнізда можна відкрити з ним з'єднання. Цей процес складається з трьох етапів. Два з них виконуються сервером (програмою, яка очікує на з'єднання), а один – клієнтом (програмою, яка виконує виклик).

З точки зору клієнта відкриття з'єднання полягає у виклику функції Connect. При цьому, програма-клієнт повинна передати методу Connect два параметри: ім'я комп'ютера або його адрес, а також номер порту, з яким потрібно встановити зв'язок.

```
if(m_mySock.Connect("192.168.7.62",5321)
{
        // продовження
}
else  // обробка помилок
```

або

```
if(m_mySock.Connect("ім'я комп'ютера",5321)
{
        // продовження
}
else  // обробка помилок
```

Після того, як з'єднання встановлено, у випадку використання класу CAsyncSocket або похідного від нього, генерується повідомлення про встановлення з'єднання, або про виникнення помилки. При роботі з класом CSocket функція не повертає результат доти, доки не буде встановлене з'єднання, або не виникне помилка. Це може призвести до зависання програми.

З точки зору сервера, програма спочатку повинна відправити гнізду команду на прослуховування викликів, що надходять. Це виконується за допомогою метода Listen. Цей метод приймає один параметр, який не обов'язково задавати. Цей параметр визначає кількість відкладених з'єднань, які можуть знаходитися в черзі повідомлень в очікуванні завершення попередніх операцій. Виклик методу Listen виглядає наступним чином:

```
if(m_mySock.Listen())
{
        // продовження
}
else // обробка помилок
```

Якщо інша програма намагається встановити з'єднання з програмою, яка прослуховує повідомлення, що надходять, клас CAsyncSocket (або похідний від нього) генерує повідомлення, яке інформує про надходження запиту на встановлення з'єднання. Програма, яка прослуховує, повинна

прийняти запит шляхом виклику методу Accept. Цей метод потребує використання другого класу CAsyncSocket, який зв'язується з програмою.

```
if(m_mySock.Accept(m_otherSock))
{
        // продовження
}
else // обробка помилок
```

При надходженні запиту на з'єднання, гніздо, що прослуховує, створює інше гніздо, яке підключається до іншої програми. При цьому метод Create для другого гнізда не викликається, оскільки воно створюється методом Accept. Функція Accept для класу CSocket не завершує своєї роботи доти, доки запит на встановлення з'єднання не буде отриманий.

Гнізда можуть використовуватися для передачі будь-якої інформації, причому зовсім немає значення який тип мають дані, оскільки функції, які виконують передачу та прийом даних, очікують передачі в них вказівника на буфер. У випадку передачі даних цей буфер повинен містити інформацію, яка буде передана. При отриманні даних, вони будуть скопійовані в цей буфер. При відправці текстових даних, діалог з буфером доить простий і виконується за допомогою змінних класу CString.

Для відправки повідомлення через гніздове з'єднання використовується метод Send. Цей метод потребує задання двох обов'язкових параметрів, а також має третій необов'язковий параметр, який може використовуватися для керування відправленням повідомлення. Перший параметр представляє собою вказівник на буфер, який містить дані для передачі. Якщо повідомлення є змінною типу CString, то для його передачі можна використати функцію LPCTSTR. В якості другого параметру передається довжина буферу. Метод повертає значення, що вказує на об'єм переданих даних. При виникненні помилки повертається значення SOCKET_ERROR. Метод Send викликається наступним чином:

```
CString str("Повідомлення для передачі");          // строка для передачі
int len;          // змінна для визначення довжини повідомлення
int send;         // змінна для визначення кількості переданих байт
len=str.GetLength();  // визначення довжини строки
send=m_mySock.Send((LPCTSTR)str,len); // передача даних
if(send==SOCKET_ERROR)                    // перевірка на помилки
{
        // обробка помилок
}
else    // продовження
```

При готовності прийому даних від іншої програми, програма, яка виконує прийом даних, генерує повідомлення. Щоб прийняти повідомлення необхідно викликати метод Receive. Перший параметр, що

передається в функцію, представляє собою вказівник на буфер, в який буде скопійоване повідомлення. Другий параметр – це розмір буферу. При наявності помилки метод Receive також повертає значення SOCKET_ERROR. Якщо прийняте текстове повідомлення, воно може бути безпосередньо скопійовано в змінну типу CString. Метод Receive може бути використаний наступним чином:

```
char rbuff[1024];       // буфер прийому
int buffsize=1023;      // розмір буферу
int recieved;           // змінна для визначення кількості прийнятих байт
CString str;            // строкова змінна
recieved=m_mySock.Receive(rbuff,buffsize);      // прийом даних
if(recieved==SOCKET_ERROR)                      // перевірка на помилки
{
        // обробка помилок
}
else
{
        rbuff[ercieved]='\0';           // обмеження строки
        str=rbuff;                      // копіювання вказівника на буфер
}
```

При отриманні текстових повідомлень слід записувати обмежувач ‘\0’ після останнього прийнятого символу. Якщо цього не зробити, то символи, що залишилися в буфері, помилково будуть прийняті як частина отриманого повідомлення.

Для класу CSocket функція Receive не завершить свою роботу доти, доки не будуть прийняті дані. При використанні дейтаграмних гнізд можна використовувати альтернативні версії двох останніх методів: SendTo та RecieveFrom.

Коли програма завершить обмін даними, вона може закрити з’єднання, викликав метод Close. Цей метод не використовує вхідних параметрів і синтаксис його виклику виглядає наступним чином:

```
m_mySock.Close();
```

Перед закриттям гнізда його можна відключити. Це виконується за допомогою методу ShutDown. Єдиний параметр, що передається в цю функцію, вказує на необхідність відключення прийому або передачі даних для даного гнізда: 0 – завершує прийом пакетів; 1 – завершує передачу пакетів; 2 – завершує і прийом, і передачу даних через гніздо.

Основна причина, по якій слід створювати похідні класи від CAsyncSocket та CSocket полягає в тому, що вони дозволяють перехоплювати повідомлення, які генеруються при отриманні даних, встановленні або розірванні зв’язку та ін. Клас CAsyncSocket містить набір функцій, які викликаються у відповідь на кожне з повідомлень. В похідних

класах ці функції повинні бути переписані. В табл. 6.3 приводяться ці функції та їх призначення.

Таблиця 6.3.

Функція	Описання
OnAccept	Викликається для гнізда, що прослуховує, щоб повідомити, що очікується запит на з'єднання, який надійшов від іншої програми.
OnClose	Викликається щоб повідомити гнізду, що програма на другому кінці з'єднання закрила гніздо зі своєї сторони, або про те, що з'єднання розірване. Це повідомлення повинне супроводжуватися закриттям гнізда.
OnConnect	Викликається для гнізда щоб просигналізувати йому, що з'єднання завершене і програма може приймати та передавати дані.
OnOutOfBandData	Викликається у випадку прийому службових даних.
OnRecieve	Сигналізує про надходження даних по мережевому з'єднанню, які доступні для зчитування методом Receive.
OnSend	Сигналізує про готовність гнізда для передачі даних. Викликається відразу після завершення з'єднання.

Клас CAsyncSocket викликає всі функції, наведені в табл. 6.3, а клас CSocket – жодну з них. Для того, щоб викликалася частина з наведених функцій, необхідно при виклику методу Create передати в якості третього параметру набір прапорців, які визначають, які з повідомлень будуть оброблені. Інший спосіб полягає у виклику функції AsyncSelect, яка виконує фільтрацію повідомлень:

```
if(m_mySock.AsyncSelect(FD_READ | FD_CONNECT |
FD_CLOSE)==SOCKET_ERROR)
{
        // обробка помилок
}
else
{
        // все добре
}
```

Значення, які можуть передаватися в функцію AsyncSelect, представлені в табл. 6.4.

Таблиця 6.4.

FD_READ	При надходженні даних викликається функція OnRecieve, що інформує про наявність даних для зчитування
FD_WRITE	Викликається функція OnSend, що інформує про готовність до передачі даних
FD_OOB	Викликається функція OnOutOfBandData при надходженні службових даних

FD_ACCEPT	Викликається функція OnAccept, що інформує програму про те, що на гніздо надійшов запит на вхідне з'єднання
FD_CONNECT	Викликається функція OnConnect, яка інформує програму про завершення запиту на з'єднання. Це повідомлення супроводжується викликом функції OnSend.
FD_CLOSE	Викликається функція OnClose, яка інформує програму про те, що гніздо було закрите.

Іноді програмі необхідна інформація про стан гнізд, наприклад адреси порту на іншому кінці з'єднання, а також про те, знаходиться гніздо в заблокованому стані в очікуванні закінчення деякої операції, чи ні. При наявності гнізд, підключеного до іншої програми, можна визначити адрес комп'ютера, на якому ця програма виконується. Це виконується за допомогою методу GetPeerName з передачею йому вказівника об'єкту CString та змінної цілого типу. Після виконання функції GetPeerName ці параметри заповнюються адресом і номером порту комп'ютера, з яким встановлений зв'язок.

```
CString addr;           // оголошення змінних
UINT port;
if(m_mySock.GetPeerName(&addr,&port)!=SOCKET_ERROR)
{
        MessageBox(addr);    // вивід повідомлення з адресом віддаленої машини
}
else
{
        // обробка помилок
}
```

Аналогічним чином можна отримати дані про гніздо, яке прослуховує повідомлення. Для цього необхідно викликати функцію GetSockName з такими самими параметрами, як і для функції GetPeerName.

6.3. Розробка програмних засобів обміну повідомленнями по мережі Ethernet

В цьому параграфі виконується розробка програми, яка буде функціонувати як в ролі клієнта, так і в ролі серверу. Після того, як програма встановить з'єднання з іншою програмою по мережі, користувач може вводити текстові повідомлення і передавати їх по мережі на інший комп'ютер. Кожне прийняте повідомлення записується в список прийнятих повідомлень.

Створюємо проект на базі діалогового вікна. При створенні проекту включити підтримку Windows Sockets (рис. 6.3). На діалоговому вікні повинен бути розташований перемикач, який визначає режим роботи програми – клієнт чи сервер. Для цього зручно використовувати елемент керування CheckBox. Також повинні міститися елементи керування для вводу адресу комп'ютера, з яким відбувається з'єднання, та номеру порту.

Для запису прийнятих повідомлень на формі слід розташувати елемент керування ListBox, а для вводу повідомлень – елемент керування EditBox. Повідомлення буде відправлене при натисканні на кнопку „Відправити”. На рис. 6.4 представлений вигляд спроектованого інтерфейсу програми.

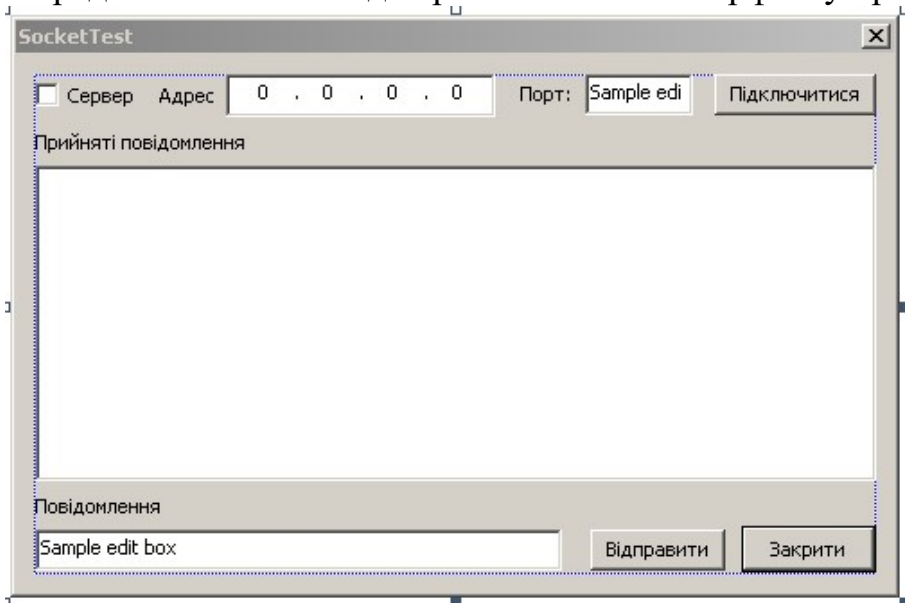


Рис. 6.4.

Зв’язуємо з елементами керування змінні. Для цього з контекстного меню, яке з’являється при натисканні правої кнопки миші на елементі керування, вибрати команду Add Variable (рис. 6.5).

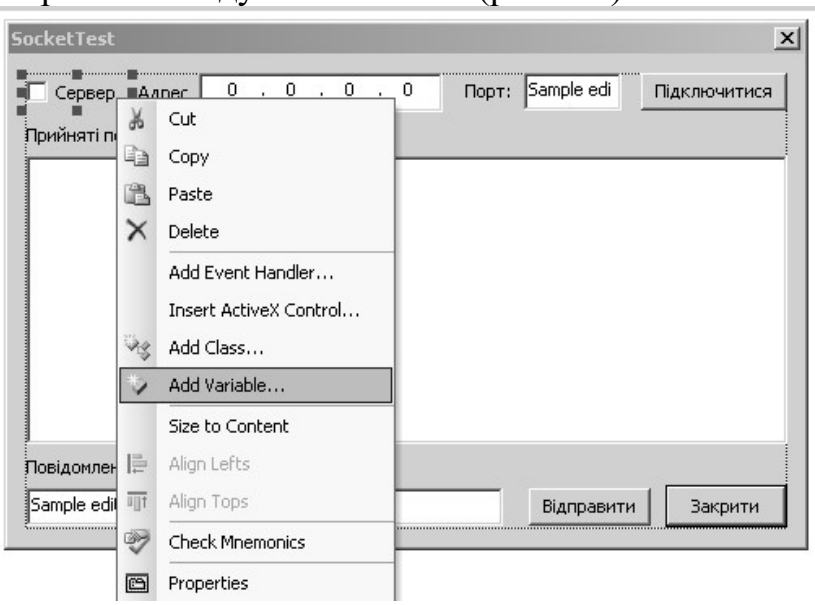


Рис. 6.5.

Для елементу керування CheckBox назва змінної m_serv (рис. 6.6). З елементом керування IP Address зв’язана змінна m_ipaddr, з EditBox – m_port, з LisBox – m_messages, з другим EditBox – m_smsg. При додаванні кожної змінної прапорець Control Variable повинен бути встановлений (рис. 6.6).

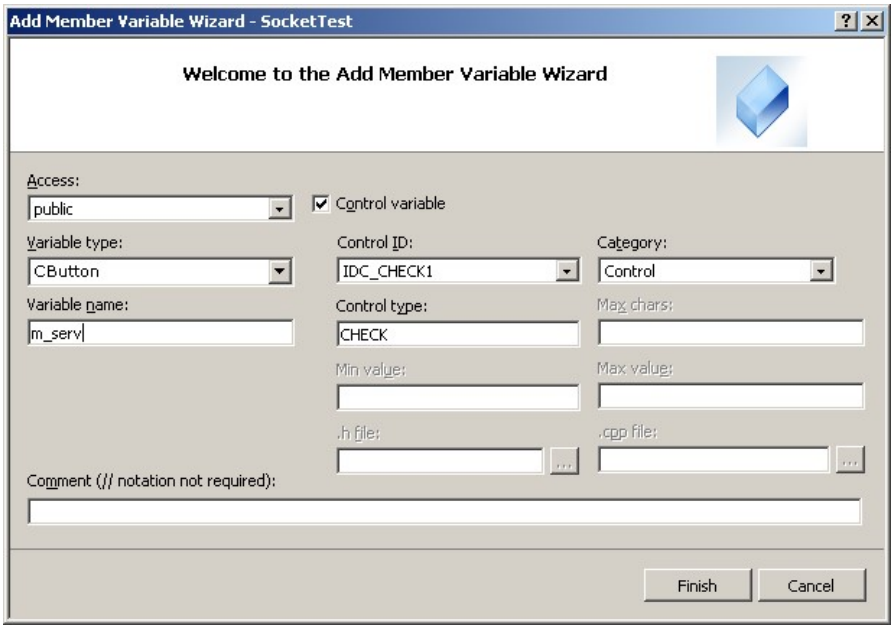


Рис. 6.6.

Для того, щоб мати можливість перехоплювати повідомлення, пов’язані з гніздом, та реагувати на них, потрібно створити власний клас, похідний від CAsyncSocket. Цей клас буде містити власні версії процедур обробки подій, а також власні засоби передачі цих повідомлень діалоговому вікну. Для передачі всіх повідомлень класу на рівні діалогового вікна, потрібно додати вказівник класу діалогового вікна у вигляді змінної-члену класу гнізда. Цей вказівник буде використовуватися для виклику функцій обробки подій, пов’язаних з гніздом. Для створення класу з меню Project вибрати команду Add Class. У діалоговому вікні, що з’явиться, слід вибрати MFC Class. Потім ввести назву нового класу (CMySocket), та вибрати базовий клас (CAsyncSocket). На рис. 6.7 представлене діалогове вікно створення нового класу.

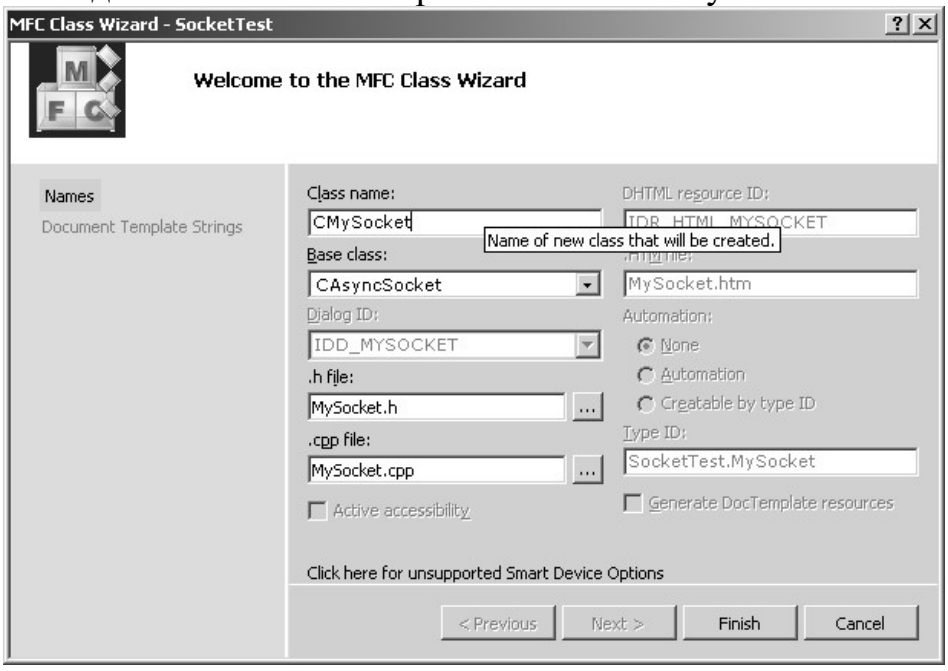


Рис. 6.7.

Після створення класу додамо в нього змінну-член, яка буде вказівником на діалогове вікно. Тип змінної CDialog*, а назва – m_pWnd. Крім того, необхідно додати метод, призначений для встановлення значення вказівника. Тому в клас CMySocket додаємо ще один метод:

```
void SetParent(CDialog* pdlg);
```

В тілі функції повинна виконуватися операція присвоєння вказівника на діалогове вікно. Загалом, реалізація цієї функції виглядає наступним чином:

```
void CMySocket::SetParent(CDialog* pdlg)
{
        m_pWnd=pdlg;        // копіювання вказівника
}
```

Вводимо в клас гнізда функції обробки подій, які будуть викликати однойменні функції класу діалогового вікна. Програмний код функції OnAccept має вигляд:

```
void CMySocket::OnAccept(int err)
{
        if(err==0) ((CSocketTestDlg*)m_pWnd)->OnAccept();
        // викликається однойменна функція класу діалогового вікна
}
```

Аналогічно створюємо функцію обробки події OnRecieve, яка призначена для виклику однойменних функцій класу діалогового вікна.

```
void CMySocket::OnReceive(int err)
{
  ((CSocketTestDlg *)m_pWnd)->OnReceive(this);
  // викликається однойменна функція класу діалогового вікна
}
```

Додавши цю функцію, також необхідно підключити заготовочний файл. Для цього на початку файлу MySocket.cpp додаємо строчку: #include "SocketTestDlg.h".

Для режиму сервера в класі діалогового вікна повинні бути дві змінні: перша з них буде прослуховувати запити на з'єднання, а друга – забезпечувати зв'язок з іншою програмою. Тип обох змінних повинен відповідати типу класу гнізда, тобто CMySocket. Додаємо дві змінні в клас CSocketTestDlg. Для цього у файл SocketTestDlg.h додати наступні рядки:

```
CMySocket m_socket;          // гніздо для зв'язку
CMySocket m_listensock;      // гніздо для прослуховування запитів
```

Також необхідно виконати ініціалізацію цих змінних. Для цього в кінці функції CSocketTestDlg::OnInitDialog() додаємо наступні строчки:

```
m_socket.SetParent(this);      // передача вказівника на об’єкт діалогового вікна
m_listensock.SetParent(this);
```

Коли користувач встановлює прапорець вибору режиму сервера, потрібно виконати створення гнізда. Додаємо оброблювач події натискання на елемент керування CheckBox, двічі натиснувши на ньому ліву кнопку миші. Відредагуємо функцію-оброблювач події наступним чином:

```
void CSocketTestDlg::OnBnClickedCheck1()
{
        if(m_serv.GetCheck()!=0)    // якщо програма працює в режимі клієнта
        {
        int iport;                  // змінна для зберігання номера порту
        CString sport;              // строкова змінна для зчитування даних
        // номера порту з елемента керування
        m_port.GetWindowTextW(sport);   // зчитування номеру порту
        iport=StrToInt(sport);          // перетворення строки в число
        m_listensock.Create(iport);  // створення гнізда для прослуховування
        m_listensock.AsyncSelect(FD_ACCEPT|FD_READ);
        // задається маска подій, що будуть оброблені
        m_listensock.Listen();                  // початок прослуховування порту
        CString s("Server is running");         // повідомлення про запуск сервера
        MessageBox(s);                          // видача повідомлення
        }
        else                        // якщо програма працює в режимі клієнта
        {
        CString s("Server is stopped");
        MessageBox(s);              // видача повідомлення про зупинку сервера
        m_listensock.Close();       // закриття гнізда прослуховування
        }
}
```

Функція Create викликається для відповідної змінної-члену класу. Для запуску сервера викликається функція Listen. Як параметр в цю функцію передається номер порту, з якого очікуються повідомлення. Номер порту вводиться користувачем у відповідному полі на діалоговому вікні. Введене значення перетворюється зі строки на число за допомогою функції StrToInt. Для того, щоб при компіляції не виникло помилки, слід підключити заготовочний файл: #include "shlwapi.h"

Додаємо оброблювач події натискання на кнопку „Підключитися”. Функція-оброблювач події натискання виглядає наступним чином:

```
void CSocketTestDlg::OnBnClickedButton1()
{
        if(m_serv.GetCheck()!=0)    // якщо режим сервера
        {
```

```
        // видача повідомлення
        MessageBox((LPCTSTR)"Програма працює в режимі серверу.");
        return; // вихід з функції
    }
    CString sip, sport;     // додаткові строкові змінні
    int iport;
    if(m_socket.Create()==SOCKET_ERROR) // створення гнізда
    {
        CString s("Error create socket!");
        MessageBox(s);              // видача повідомлення про помилку
        return; // вихід з функції
    }
    m_ipaddr.GetWindowTextW(sip);   // отримання адресу
    m_port.GetWindowTextW(sport);   // отримання номера порту
    iport=StrToInt(sport);// перетворення строки в число
    // спроба підключитися до сервера
    if(m_socket.Connect(sip,iport)==SOCKET_ERROR)
    {
        // якщо підключення не вдалося, видача повідомлення
        // про неможливість встановити з'єднання
        CString s("Connection is impossible!");
        MessageBox(s);
        return;
    }
    else    // якщо підключення вдалося
    {
        // видається відповідне повідомлення
        CString s("Connection was succesfull!");
        MessageBox(s);
    }
    }
```

Для встановлення з'єднання викликається функція Connect для відповідної змінної. Параметри, а саме адрес віддаленого комп'ютера та номер порту, вводяться користувачем за допомогою елементів керування на діалоговому вікні.

Додаємо в клас діалогового вікна CSocketTestDlg функції OnAccept та OnRecieve для обробки відповідних подій. Ці функції викликаються однойменними процедурами класу гнізда. Тип обох функцій – void. Отже, функція OnAccept виглядає так:

```
void CSocketTestDlg::OnAccept()
{
    m_listensock.Accept(m_socket);      // при надходженні
    // запиту на з'єднання відбувається встановлення каналу зв'язку
    CString str("Connection was accepted!");
    MessageBox(str);
}
```

При встановленні з’єднання видається відповідне повідомлення. Функція OnReceive виглядає наступним чином:

```
void CSocketTestDlg::OnReceive(CMySocket* sock)
{
	char buff[1024];                    // буфер для прийнятих даних
	int rsize;
	rsize=m_socket.Receive(buff,1023); // зчитування прийнятих даних
	if(rsize==SOCKET_ERROR)
	{
		MessageBox((LPCTSTR)"Receive error!");
		return;
	}
buff[rsize]='\0';
CString str(buff);              // формування рядка
m_messages.AddString(str);  // повідомлення додається в список
}
```

Кожне повідомлення, що буде прийняте, додається в список. Додаємо оброблювач події натискання на кнопку „Відправити”. При натисканні на цю кнопку текстове повідомлення у полі вводу передається на віддалений комп’ютер. Функція-обробник події натискання на кнопку виглядає наступним чином:

```
void CSocketTestDlg::OnBnClickedButton2()
{
	CString s;	// додаткова строкова змінна
	int len;
	// зчитування повідомлення з елементу керування
	m_smsg.GetWindowTextW(s);
	len=s.GetLength();                  // визначення довжини повідомлення
	m_socket.Send((LPCTSTR)s,5);	// передача даних по мережі
}
```

Тепер можна компілювати проект. Для перевірки роботи програми необхідно запустити дві його копії (якщо тестування відбувається на одному і тому ж самому комп’ютері). Одна з програм працює в режимі серверу. Перед тим, як встановити позначку „Сервер”, в полі „Порт” необхідно ввести номер порту, який буде прослуховуватися (рис. 6.8). У разі вдалого виконання процедури запуску сервера з’явиться відповідне повідомлення.

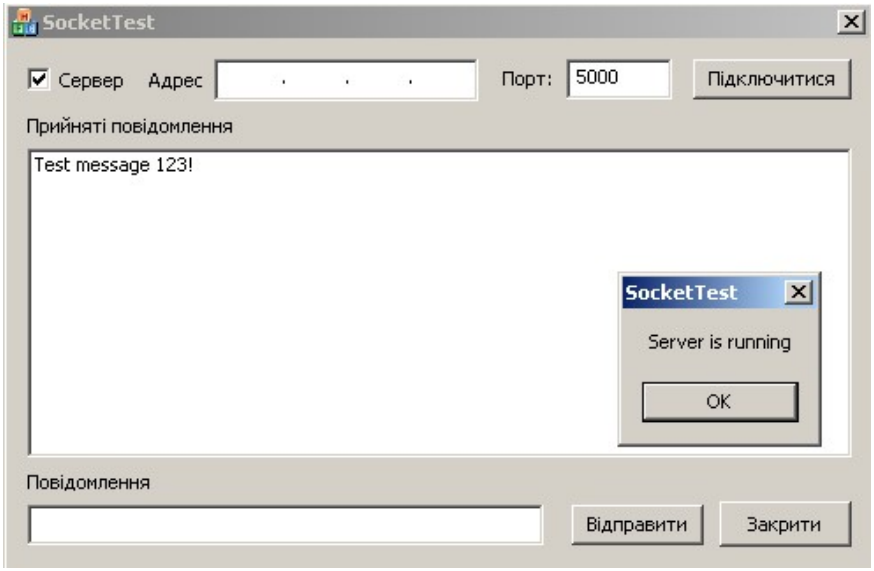


Рис. 6.8.

Для встановлення з'єднання необхідно вказати IP-адрес комп'ютера, з яким встановлюється зв'язок, а також номер порту. При вдалому підключенні з'явиться відповідне повідомлення. На рис. 6.9 представлене діалогове вікно програми при роботі в режимі клієнта.

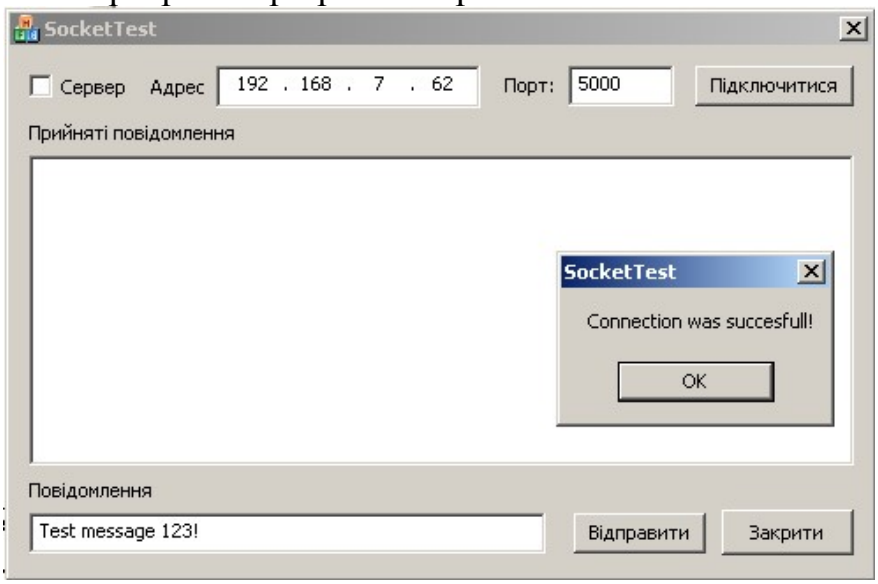


Рис. 6.9.

Наведений приклад демонструє основні принципи організації зв'язку між комп'ютерами по локальній мережі з використанням класів бібліотеки MFC. Програма дозволяє передавати повідомлення від в обох напрямках: від клієнта на сервер і навпаки.

6.4. Дистанційне керування роботом-маніпулятором по мережі Ethernet

Компанія Velleman створила набір-конструктор Robotic Arm KSR10 для бажаючих взяти участь в управлінні роботом маніпулятором

(рис. 6.10). Рухомі частини робота переміщаються п'ятьма моторами, забезпеченими редукторами.

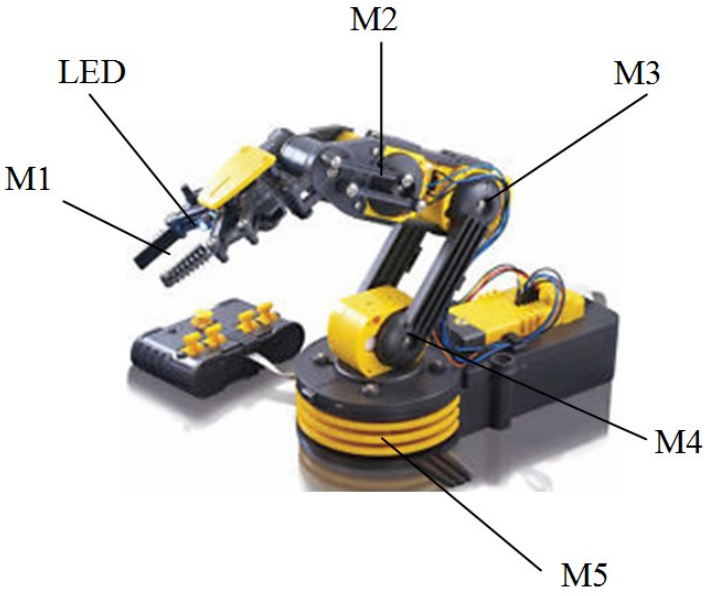


Рис. 6.10.

По кожній ступені свободи маніпулятор обертається через редуктор з дуже великим передавальним числом, яке не менше ніж 1:300. Редуктори-черв'ячно-циліндричний, багатоступінчастий, з трещіткою обмежувача моменту.

Живлення приладу виконується стабілізованими двополярною напругою 3,3В. Двополярна напруга використовується для живлення двигунів постійного струму для забезпечення їх реверсу. Проаналізувавши існуючі схеми випрямних пристроїв, було обрано однофазну мостову схему. Ця схема характеризується гарним використанням потужності трансформатора. Зворотна напруга на діодах при цій схемі в два рази менша, ніж при одно- і двохнапівперіодних схемах випрямляча.

Мостова схема містить однофазний трансформатор, діодний міст, згладжуючи фільтри і стабілізатори напруги. Прилад живиться від мережі 220В/50Гц. На рис. 6.11 зображена принципова схема джерела живлення системи керування роботом-маніпулятором.

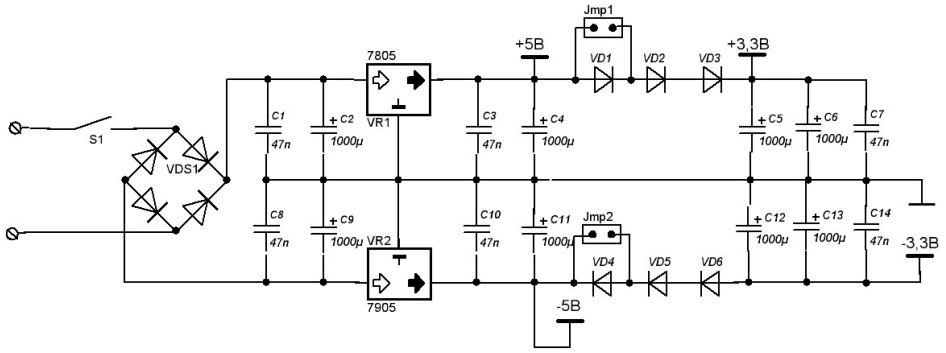


Рис. 6.11.

На рис. 6.12 представлено принципову схему силової частини системи керування роботом-маніпулятором. Для керування двигунами постійного струму з можливістю реверса було використано біполярні транзистори різних типів. При розробці програмного забезпечення слід забезпечити паузу між перемиканням сигналів керування для запобігання наскрізних струмів через транзистори. Якщо цього не зробити, то транзистори можуть вийти з ладу.

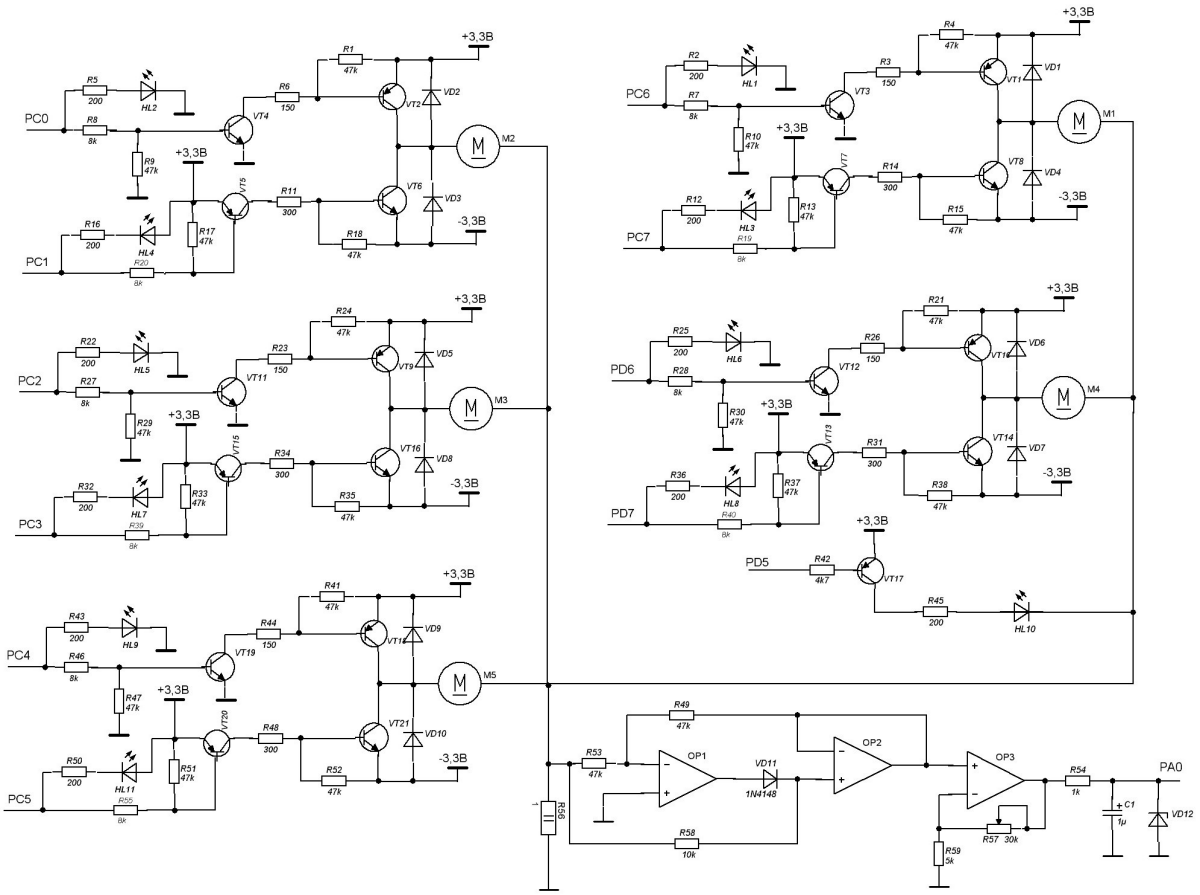


Рис. 6.12.

Текст програми для мікроконтролера системи керування роботом-маніпулятором представлено нижче.

```
// підключення заголовочних файлів та бібліотеки макросів
#include "iom16v.h"
#include "macros.h"
// призначення ідентифікатора рядку символів
#define SET_R1() PORTD|=0x10
#define RST_R1() PORTD&=~0x10
#define SET_R2() PORTD|=0x20
#define RST_R2() PORTD&=~0x20

#define CHANGE_R1() PORTD^=0x10

#define SET_LED() PORTA&=~0b00100000
#define RST_LED() PORTA|=0b00100000
```

```
// оголошення прототипів функцій керування маніпулятором
void move_robot(void);
void set_timer0(void);          // функція запуску таймера-лічильника
void stop_timer0(void);         // функція зупинки таймера-лічильника
void delay(int ms);             // функція часової затримки
void M1_left(void);             // функція вмикання обертання мотору M1 ліворуч
void M1_right(void);            // функція вмикання обертання мотору M1 праворуч
void M1_stop(void);             // функція зупинки мотору M1
void M2_left(void);             // функція вмикання обертання мотору M2 ліворуч
void M2_right(void);            // функція вмикання обертання мотору M2 праворуч
void M2_stop(void) ;            // функція зупинки мотору M2
void M3_left(void);             // функція вмикання обертання мотору M3 ліворуч
void M3_right(void);            // функція вмикання обертання мотору M3 праворуч
void M3_stop(void);             // функція зупинки мотору M3
void M4_left(void);             // функція вмикання обертання мотору M4 ліворуч
void M4_right(void);            // функція вмикання обертання мотору M4 праворуч
void M4_stop(void);             // функція зупинки мотору M4
void M5_left(void);             // функція вмикання обертання мотору M5 ліворуч
void M5_right(void);            // функція вмикання обертання мотору M5 праворуч
void M5_stop(void);             // функція зупинки мотору M5
void stop(void);                // функція зупинки всіх моторів
void init_cpu(void);            // функція ініціалізації мікроконтролера
unsigned int read_adc(unsigned char channel);

char state;
char b1, b2;

void move_robot()               // реалізація функції руху робота
{
 if((b1&0b11000000)==0b10000000) M1_left();              // керування мотором M1
 else if((b1&0b11000000)==0b01000000) M1_right();
 else if(((b1&0b11000000)==0) || ((b1&0b11000000)==0b11000000)) M1_stop();

 if((b1&0b00110000)==0b00100000) M2_left();              // керування мотором M2
 else if((b1&0b00110000)==0b00010000) M2_right();
 else if(((b1&0b00110000)==0)  || ((b1&0b00110000)==0b00110000)) M2_stop();

 if((b1&0b00001100)==0b00001000) M3_left();              // керування мотором M3
 else if((b1&0b00001100)==0b00000100) M3_right();
 else if(((b1&0b00001100)==0)  || ((b1&0b00001100)==0b00001100)) M3_stop();

 if((b1&0b000011)==0b00000010) M4_left();                // керування мотором M4
 else if((b1&0b00000011)==0b00000001) M4_right();
 else if(((b1&0b00000011)==0)  || ((b1&0b00000011)==0b00000011)) M4_stop();

 if((b2&0b11000000)==0b10000000) M5_left();              // керування мотором M5
 else if((b2&0b11000000)==0b01000000) M5_right();
 else if(((b2&0b11000000)==0)  || ((b2&0b11000000)==0b11000000)) M5_stop();

 if((b2&0b00100000)!=0) SET_LED();                       // керування світлодіодом
 else RST_LED();
```

```
}

void set_timer0()               // реалізація функції запуску таймера
{
 TCCR0=0;                       // зупинка таймера-лічильника
 TCNT0=0x64;                    // встановлення початкового значення
 TCCR0=0x05;                    // налаштування поділювача частоти і запуск таймера
 RST_R2();
}

void stop_timer0()
{
 TCCR0 = 0;                     // зупинка таймера-лічильника
}
// функція-обробник переривання по переповненню таймера
#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
 state=0;
 SET_R2();
 stop();
}
// функція-обробник переривання по прийманню даних по USART
#pragma interrupt_handler uart0_rx_isr:12
void uart0_rx_isr(void)
{
 char d;                    // оголошення одаткових змінних
 int v;
 d=UDR;                     // зчитування отриманих даних в змінну
 switch(state)
 {
  case 0:
            if(d=='@')    // якщо отримано символ '@'
            {
                  state=1;
                  set_timer0();   // запуск таймера-лічильника
                  SET_R1();
            }
  break;
  case 1:
            b1=d;
            state=2;
  break;
  case 2:
            b2=d;
            state=3;
  break;
  case 3:
            state=0;
            if(d=='#') // якщо отримано символ '#'
             {
                  stop_timer0(); // зупинка таймера-лічильника
```

```
                    // зчитування даних з АЦП про струм споживання
                    v=read_adc(0);
                    UDR=(char)v;          // відправлення даних в комп'ютер
                    RST_R1();
                    move_robot();
                   }
 break;
 default:
          state=0;
          if(d==255) stop();
 break;
 }
}
// реалізація функції часової затримки
void delay(int ms)
{
 int c1, c2;
 for(c1=0;c1<ms;c1++)
  for(c2=0;c2<970;c2++) NOP();      // в циклі виконується «пуста» операція
}

void M1_stop()                  // реалізація функції зупинки мотора M1
{
 PORTB&=~0b00100000;    // скидання біту номер 5 порта B
 PORTB|=0b00010000;     // встановлення біту номер 4 порта B
 delay(10);             // часова затримка 10 мс
}

void M1_left()                  // реалізація функції обертання мотора M1 ліворуч
{
 PORTB|=0b00010000;     // встановлення біту номер 4 порта B
 delay(10);
 PORTB|=0b00100000;     // встановлення біту номер 5 порта B
}

void M1_right()                 // реалізація функції обертання мотора M1 праворуч
{
 PORTB&=~0b00100000;    // скидання біту номер 5 порта B
 delay(10);
 PORTB&=~0b00010000;    // скидання біту номер 4 порта B
}

void M2_stop()                  // реалізація функції зупинки мотора M2
{
 PORTB&=~0b00001000;    // скидання біту номер 3 порта B
 PORTB|=0b00000100;     // встановлення біту номер 2 порта B
 delay(10);             // часова затримка 10 мс
}

void M2_left()                  // реалізація функції обертання мотора M2 ліворуч
{
 PORTB|=0b00000100;     // встановлення біту номер 2 порта B
```

```
 delay(10);
 PORTB|=0b00001000;        // встановлення біту номер 3 порта B
}

void M2_right()            // реалізація функції обертання мотора M2 праворуч
{
 PORTB&=~0b00001000;       // скидання біту номер 3 порта B
 delay(10);
 PORTB&=~0b00000100;       // скидання біту номер 2 порта B
}

void M3_stop()             // реалізація функції зупинки мотора M3
{
 PORTB&=~0b00000010;       // скидання біту номер 1 порта B
 PORTB|=0b00000001;        // встановлення біту номер 0 порта B
 delay(10);                // часова затримка 10 мс
}

void M3_left()             // реалізація функції обертання мотора M3 ліворуч
{
 PORTB|=0b00000001;        // встановлення біту номер 0 порта B
 delay(10);                // часова затримка 10 мс
 PORTB|=0b00000010;        // встановлення біту номер 1 порта B
}

void M3_right()            // реалізація функції обертання мотора M3 праворуч
{
 PORTB&=~0b00000010;       // скидання біту номер 1 порта B
 delay(10);                // часова затримка 10 мс
 PORTB&=~0b00000001;       // скидання біту номер 0 порта B
}

void M4_stop()             // реалізація функції зупинки мотора M4
{
 PORTA&=~0b00010000;       // скидання біту номер 4 порта A
 PORTA|=0b00001000;        // встановлення біту номер 3 порта A
 delay(10);                // часова затримка 10 мс
}

void M4_left()             // реалізація функції обертання мотора M4 ліворуч
{
 PORTA|=0b00001000;        // встановлення біту номер 3 порта A
 delay(10);                // часова затримка 10 мс
 PORTA|=0b00010000;        // встановлення біту номер 4 порта A
}

void M4_right()            // реалізація функції обертання мотора M4 праворуч
{
 PORTA&=~0b00010000;       // скидання біту номер 4 порта A
 delay(10);                // часова затримка 10 мс
 PORTA&=~0b00001000;       // скидання біту номер 3 порта A
}
```

```
void M5_stop()                  // реалізація функції зупинки мотора M5
{
 PORTA&=~0b00000100;    // скидання біту номер 2 порта A
 PORTA|=0b00000010;     // встановлення біту номер 1 порта A
 delay(10);                     // часова затримка 10 мс
}

void M5_left()                  // реалізація функції обертання мотора M5 ліворуч
{
 PORTA|=0b00000010;     // встановлення біту номер 1 порта A
 delay(10);                     // часова затримка 10 мс
 PORTA|=0b00000100;     // встановлення біту номер 2 порта A
}

void M5_right()                 // реалізація функції обертання мотора M5 праворуч
{
 PORTA&=~0b00000100;    // скидання біту номер 2 порта A
 delay(10);                     // часова затримка 10 мс
 PORTA&=~0b00000010;    // скидання біту номер 1 порта A
}

void test()
{
  M1_left();                    // виклик функції обертання мотора M1 ліворуч
  delay(1000);                  // часова затримка 1 с
  M1_right();                   // виклик функції обертання мотора M1 праворуч
  delay(1000);                  // часова затримка 1 с
  M1_stop();                    // виклик функції зупинки мотора M1
  delay(1000);                  // часова затримка 1 с

  M2_left();                    // виклик функції обертання мотора M2 ліворуч
  delay(1000);                  // часова затримка 1 с
  M2_right();                   // виклик функції обертання мотора M2 праворуч
  delay(1000);                  // часова затримка 1 с
  M2_stop();                    // виклик функції зупинки мотора M2
  delay(1000);                  // часова затримка 1 с

  M3_left();                    // виклик функції обертання мотора M3 ліворуч
  delay(1000);                  // часова затримка 1 с
  M3_right();                   // виклик функції обертання мотора M3 праворуч
  delay(1000);                  // часова затримка 1 с
  M3_stop();                    // виклик функції зупинки мотора M3
  delay(1000);                  // часова затримка 1 с

  M4_left();                    // виклик функції обертання мотора M4 ліворуч
  delay(1000);                  // часова затримка 1 с
  M4_right();                   // виклик функції обертання мотора M4 праворуч
  delay(1000);                  // часова затримка 1 с
  M4_stop();                    // виклик функції зупинки мотора M4
  delay(1000);                  // часова затримка 1 с
```

```
 M5_left();                  // виклик функції обертання мотора M5 ліворуч
 delay(1000);                // часова затримка 1 с
 M5_right();                 // виклик функції обертання мотора M5 праворуч
 delay(1000);                // часова затримка 1 с
 M5_stop();                  // виклик функції зупинки мотора M5
 delay(1000);                // часова затримка 1 с

 SET_LED();                  // увімкнути світлодіод
 delay(1000);                // часова затримка 1 с
 RST_LED();                  // вимкнути світлодіод
 delay(1000);                // часова затримка 1 с
}

void stop()                  // реалізація функції зупинки моторів
{
 // встановлення високих (1) та низьких (0) рівнів на виводах порта B
 PORTB=0b00010101;
 // встановлення високих (1) та низьких (0) рівнів на виводах порта A
 PORTA=0b00001010;
 RST_LED();                  // вимкнути світлодіод
 delay(100);                 // часова затримка 100 мс
}

void init_cpu()              // реалізація функції ініціалізації мікроконтролера
{
 char i;
 DDRD=0b00110000; // налаштування виводів портів на вхід (0) та вихід (1)
 PORTB=0b00010101;
 DDRB=0b00111111;
 PORTA=0b00001010;
 DDRA=0b00111110;
 RST_LED();
 ADCSRA=0x86;      // налаштування аналого-цифрового перетворювача
 UCSRB = 0x00;
 UCSRA = 0x00;
 UBRRL = 51;       // налаштування швидкості передачі даних 9600 біт/с
 UBRRH = 0x00;
 UCSRB = 0x98;     // дозвіл роботи приймача та передавача, та переривання

 for(i=0;i<4;i++)  // в циклі виконується вмикання та вимикання світлодіода
 {
  SET_R1();        // увімкнути світлодіод
  delay(250);      // часова затримка 250 мс
  RST_R1();        // вимкнути світлодіод
  SET_R2();        // увімкнути світлодіод
  delay(250) ;     // часова затримка 250 мс
  RST_R2();        // вимкнути світлодіод
 }
 TIMSK = 0x01;     // дозвіл переривання по переповненню таймера-лічильника
 SEI();            // дозвіл обробки будь-яких переривань
}
```

```
// реалізація функції аналого-цифрового перетворення
unsigned int read_adc(unsigned char channel)
{
 unsigned int adcw;            // оголошення додаткових змінних
 char a;
 char b;
 ADMUX=channel;                // вибір каналу АЦП
ADCSRA|=0x40;                  // запуск процесу перетворення
while ((ADCSRA & 0x10)==0); // очікування завершення перетворення
 ADCSRA&=~0x10;
 a=ADCL;                       // зчитування отриманих даних
 b=ADCH;
 b&=0x03;
 adcw=b;                       // формування результату
 adcw=adcw<<8;                 // зсув на 8 розрядів ліворуч
 adcw=adcw+a;
 return adcw;                  // повернення результату в головну програму
}

void main()                    // головна функція програми
{
 delay(5000);                  // часова затримка 5 секунд
 init_cpu();                   // виклик функції ініціалізації мікроконтролера
 state=0;                      // встановлення значення початкового стану
 while(1);                     // вічний цикл – робота ведеться по перериванням
}
```

Для вимірювання струму, що споживається роботом, використана схема підсилювача на операційному підсилювачі. Вихідний сигнал подається на вхід аналого-цифрового перетворювача, вбудованого в мікроконтролер.

Опис протоколу обміну даними з системою керування роботом-маніпулятором представлено нижче.

Формат керуючого пакету даних:

Розмір пакету 4 байти:

Байт1	Байт2	Байт3	Байт4
0x40	*cmd1*	*cmd2*	0x23

Байт1 та Байт4 використовуються для ідентифікації пакету даних. Якщо значення Байт1 дорівнює 64 (0x40, ASCII-символ ‘@’), а значення Байт4 дорівнює 35 (0x23, ASCII-символ ‘#’), то біти полів *cmd1* (Байт2) й *cmd2* (Байт3) визначають команду, що виконується.

Значення біт поля *cmd1*:

M1		M2		M3		M4	
Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
B1	B0	B1	B0	B1	B0	B1	B0

Значення біт поля *cmd2*:

M5		LED		Не використовуються, завжди дорівнюють 0			
Біт 7	Біт 6	Біт 5	Біт 4	Біт 3	Біт 2	Біт 1	Біт 0
B1	B0	B1	B0	0	0	0	0

Значення біт полів M1-M5:
{B1, B0}
0 0 – зупинка мотора
0 1 – обертання мотора за годинниковою стрілкою
1 0 – обертання мотора проти годинникової стрілки

Біти поля LED використовуються для керування станом світлодіода:
{B1, B0}
0 0 – виключити світлодіод
1 0 – включити світлодіод

Відповідь – 2 байти – значення споживаного струму:

Байт1	Байт2
current	0

Формула перетворення отриманого значення струму в ампери:

$$I = current/175 \text{ (A)}.$$

Поворот робота праворуч. Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0	0x40	0x23

Відповідь: 2 байти з інформацією про споживаний струм.

Поворот робота ліворуч. Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0	0x80	0x23

Відповідь: 2 байти з інформацією про споживаний струм.

Зупинка всіх моторів: Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0	0	0x23

Відповідь: 2 байти з інформацією про споживаний струм.

Захватити предмет: Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0x80	0	0x23

Відповідь: 2 байти з інформацією про споживаний струм.

Відпустити предмет: Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0x40	0	0x23

Відповідь: 2 байти з інформацією про споживаний струм.

Включити світлодіод: Відправляється пакет 4 байта:

Байт1	Байт2	Байт3	Байт4
0x40	0	0x20	0x23

Відповідь: 2 байти з інформацією про споживаний струм.

Розробка програмного забезпечення для управління роботом починається зі створення проекту в середовищі Visual Studio. Далі, на діалоговому вікні необхідно розмістити елементи управління. З елементами управління, розташованими на діалоговому вікні, необхідно пов'язати змінні. На рис. 6.13 представлено діалогове вікно серверної частини програми.



Рис. 6.13.

Для того, щоб задіяти функціональність роботи з мережею, в проект необхідно додати клас, що наслідується від класу CAsyncSocket. Назва класу має бути CMySocket. В цей клас необхідно додати зміну-член:

```
CDialog* m_dlg;
```

Також потрібно перевизначити наступні віртуальні функції базового класу:

```
void CMySocket::OnAccept(int nErrorCode)  // обробник приймання підключення
{
	((CRobotServerDlg*)m_dlg)->OnAccept(this); // виклик функції діалогового вікна
	CAsyncSocket::OnAccept(nErrorCode);
}

void CMySocket::OnReceive(int nErrorCode) // обробник події отримання пакету даних
{
	((CRobotServerDlg*)m_dlg)->OnReceive(this); // виклик функції діалогового вікна
```

```
	CAsyncSocket::OnReceive(nErrorCode);
}
```

В заголовочний файл класу діалогу необхідно додати наступний код:

```
HANDLE m_hCom;		// оголошення змінної для збереження дескриптора порту
CMySocket m_socket;
CMySocket m_listenSock;
void OnAccept(CAsyncSocket*);
void OnReceive(CAsyncSocket*);
```

Програмний код класу діалога представлено нижче.

```
BOOL CRobotServerDlg::OnInitDialog()
{
	CDialog::OnInitDialog();
	ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
	ASSERT(IDM_ABOUTBOX < 0xF000);

	CMenu* pSysMenu = GetSystemMenu(FALSE);
	if (pSysMenu != NULL)
	{
		CString strAboutMenu;
		strAboutMenu.LoadString(IDS_ABOUTBOX);
		if (!strAboutMenu.IsEmpty())
		{
	 pSysMenu->AppendMenu(MF_SEPARATOR);
	 pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
		}
	}

	SetIcon(m_hIcon, TRUE);
	SetIcon(m_hIcon, FALSE);
	m_listenSock.m_dlg=this;		// ініціалізація сокетів
	m_socket.m_dlg=this;
	m_status.SetWindowText("Server: stopped");
	m_combo.SetCurSel(0);
	return TRUE;
}

void CRobotServerDlg::OnReceive(CAsyncSocket * socket)
{
	char buff[4];				// оголошення масиву для збереження команди
	int size;				// змінна для збереження розміру пакету даних
	size=socket->Receive(buff,4);		// зчитати отримані даних в масив
	CString str;
	str.Format("Command: 0x%x 0x%x, size=%d",buff[1],buff[2],size);
	m_list.AddString(str);
	if(size!=4) return;			// вихід із функції якщо отримано не 4 байти
	DWORD rb,wb;
	// передавання команди в систему керування роботом
```

```
	WriteFile(m_hCom,buff,size,&wb,NULL);
	ReadFile(m_hCom,buff,2,&rb,NULL);	// зчитування відповіді
}

void CRobotServerDlg::OnAccept(CAsyncSocket * socket)
{
	m_listenSock.Accept(m_socket);	// прийняти підключення
}

void CRobotServerDlg::OnButton1()
{
	UpdateData(true);
	m_listenSock.Create(m_port); // створення сокета, що прослуховує вказаний порт
	m_listenSock.m_dlg=this;	// ініціалізація поля m_dlg об'єкта сокета
	m_listenSock.AsyncSelect(FD_READ|FD_ACCEPT);
	m_listenSock.Listen();		// почати прослуховування порта
	m_status.SetWindowText("Server: running");	// оновлення тексту на діалозі

	CString port;
	// зчитування назви обраного користувачем посліовного порта
	m_combo.GetLBText(m_combo.GetCurSel(), port);
	// відкриття послідовного порта для запису та зчитування даних
	m_hCom=CreateFile(port,GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_EX
ISTING,0,NULL);

	DCB dcb;		// структура для налаштування параметрів порта
	GetCommState(m_hCom,&dcb);
	dcb.BaudRate=CBR_9600;	// налаштування швидкості передавання даних
	dcb.StopBits=ONESTOPBIT;	// один стоповий біт
	dcb.Parity=NOPARITY;		// перевірка на парність відсутня
	dcb.ByteSize=8;			// розмір одного слова даних 8 біт
	SetCommState(m_hCom,&dcb);
	COMMTIMEOUTS ct;		// структура для налаштування тайм-аутів
	ct.ReadIntervalTimeout=10;	// тайм-аут для зчитування даних
	ct.ReadTotalTimeoutConstant=100;
	ct.ReadTotalTimeoutMultiplier=2;
	ct.WriteTotalTimeoutConstant=100;// тайм-аут на запис даних
	ct.WriteTotalTimeoutMultiplier=2;
	SetCommTimeouts(m_hCom,&ct);
}

void CRobotServerDlg::OnButton2()
{
	m_listenSock.Close();		// закрити об'єкт сокета
	m_listenSock.ShutDown();
	m_status.SetWindowText("Server: stopped");
	CloseHandle(m_hCom);		// закрити ослідовний порт
}
```

На рис. 6.14 представлено діалогове вікно клієнтської частини програмного забезпечення.

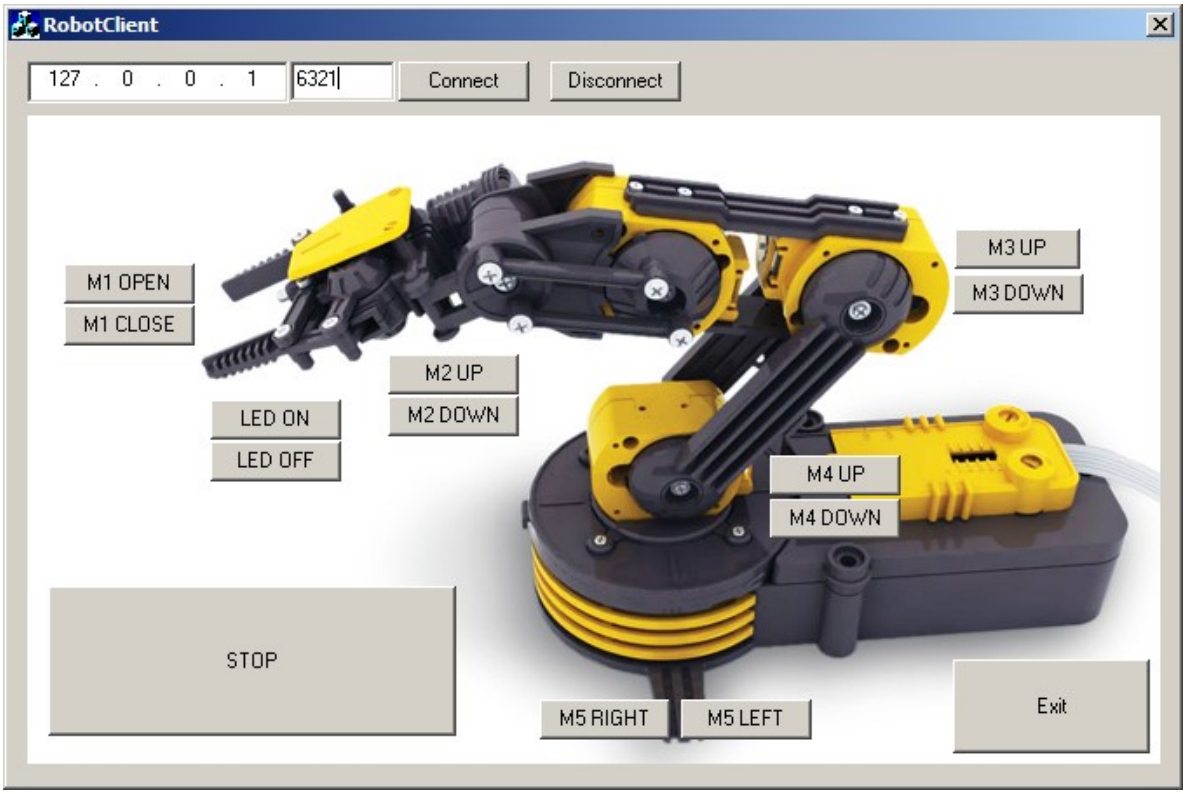


Рис. 6.14.

Обробники подій натискання на кнопки інтерфейсу користувача представлено нижче.

```
void CRobotClientDlg::OnButton1()          // обробник натискання на кнопку Connect
{
        UpdateData(true);
        m_socket.Create();                 // створення об’єкта сокета
        CString sip;
        m_ipaddress.GetWindowText(sip);    // отримання ip-адреси
        m_socket.Connect(sip,m_port);      // підключення до віддаленого комп’ютера
}

void CRobotClientDlg::OnButton2()          // обробник натискання на кнопку Disconnect
{
        m_socket.Close();                  // закрити сокет
}

void CRobotClientDlg::OnButton3() // обробник натискання на кнопку STOP
{
        char buff[4];                      // оголошення масиву даних для зберігання команди
        buff[0]=0x40;                      // перший символ команди
        buff[3]=0x23;                      // останній символ команди
        buff[1]=0;                         // команда зупинки всіх двигунів
        buff[2]=0;
        m_socket.Send(buff,4);             // відправлення команди на сервер
}

void CRobotClientDlg::OnButton4() // обробник натискання на кнопку M1 OPEN
```

```
{
	char buff[4];			// оголошення масиву даних для зберігання команди
	buff[0]=0x40;			// перший символ команди
	buff[3]=0x23;			// останній символ команди
	buff[1]=0x40;			// команда керування мотором M1
	buff[2]=0;			// формування команди відповідно до протоколу
	m_socket.Send(buff,4);		// відправлення команди на сервер
}

void CRobotClientDlg::OnButton9()// обробник натискання на кнопку M1 CLOSE
{
	char buff[4];			// оголошення масиву даних для зберігання команди
	buff[0]=0x40;			// перший символ команди
	buff[3]=0x23;			// останній символ команди
	buff[1]=0x80;			// формування команди відповідно до протоколу
	buff[2]=0;			// формування команди відповідно до протоколу
	m_socket.Send(buff,4);		// відправлення команди на сервер
}

void CRobotClientDlg::OnButton5()// обробник натискання на кнопку M2 UP
{
	char buff[4];			// оголошення масиву даних для зберігання команди
	buff[0]=0x40;			// перший символ команди
	buff[3]=0x23;			// останній символ команди
	buff[1]=0x20;			// формування команди відповідно до протоколу
	buff[2]=0;			// формування команди відповідно до протоколу
	m_socket.Send(buff,4);		// відправлення команди на сервер
}

void CRobotClientDlg::OnButton10()// обробник натискання на кнопку M2 DOWN
{
	char buff[4];			// оголошення масиву даних для зберігання команди
	buff[0]=0x40;			// перший символ команди
	buff[3]=0x23;			// останній символ команди
	buff[1]=0x10;			// формування команди відповідно до протоколу
	buff[2]=0;			// формування команди відповідно до протоколу
	m_socket.Send(buff,4);		// відправлення команди на сервер
}

void CRobotClientDlg::OnButton6()// обробник натискання на кнопку M3 UP
{
	char buff[4];			// оголошення масиву даних для зберігання команди
	buff[0]=0x40;			// перший символ команди
	buff[3]=0x23;			// останній символ команди
	buff[1]=0x08;			// формування команди відповідно до протоколу
	buff[2]=0;			// формування команди відповідно до протоколу
	m_socket.Send(buff,4);		// відправлення команди на сервер
}

void CRobotClientDlg::OnButton11()// обробник натискання на кнопку M3 DOWN
{
	char buff[4];			// оголошення масиву даних для зберігання команди
```

```
	buff[0]=0x40;		// перший символ команди
	buff[3]=0x23;		// останній символ команди
	buff[1]=0x04;		// формування команди відповідно до протоколу
	buff[2]=0;		// формування команди відповідно до протоколу
	m_socket.Send(buff,4);	// відправлення команди на сервер
}

void CRobotClientDlg::OnButton7()// обробник натискання на кнопку M4 UP
{
	char buff[4];		// оголошення масиву даних для зберігання команди
	buff[0]=0x40;		// перший символ команди
	buff[3]=0x23;		// останній символ команди
	buff[1]=0x02;		// формування команди відповідно до протоколу
	buff[2]=0;		// формування команди відповідно до протоколу
	m_socket.Send(buff,4);	// відправлення команди на сервер
}

void CRobotClientDlg::OnButton12()// обробник натискання на кнопку M4 DOWN
{
	char buff[4];		// оголошення масиву даних для зберігання команди
	buff[0]=0x40;		// перший символ команди
	buff[3]=0x23;		// останній символ команди
	buff[1]=0x01;		// формування команди відповідно до протоколу
	buff[2]=0;		// формування команди відповідно до протоколу
	m_socket.Send(buff,4);	// відправлення команди на сервер
}

void CRobotClientDlg::OnButton8()// обробник натискання на кнопку M5 LEFT
{
	char buff[4];		// оголошення масиву даних для зберігання команди
	buff[0]=0x40;		// перший символ команди
	buff[3]=0x23;		// останній символ команди
	buff[1]=0;		// формування команди відповідно до протоколу
	buff[2]=0x80;		// формування команди відповідно до протоколу
	m_socket.Send(buff,4);	// відправлення команди на сервер
}

void CRobotClientDlg::OnButton13()// обробник натискання на кнопку M5 RIGHT
{
	char buff[4];		// оголошення масиву даних для зберігання команди
	buff[0]=0x40;		// перший символ команди
	buff[3]=0x23;		// останній символ команди
	buff[1]=0;		// формування команди відповідно до протоколу
	buff[2]=0x40;		// формування команди відповідно до протоколу
	m_socket.Send(buff,4);	// відправлення команди на сервер
}

void CRobotClientDlg::OnButton14()// обробник натискання на кнопку LED ON
{
	char buff[4];		// оголошення масиву даних для зберігання команди
	buff[0]=0x40;		// перший символ команди
	buff[3]=0x23;		// останній символ команди
```

```
	buff[1]=0;			// формування команди відповідно до протоколу
	buff[2]=0x20;			// формування команди відповідно до протоколу
	m_socket.Send(buff,4);		// відправлення команди на сервер
}

void CRobotClientDlg::OnButton15()// обробник натискання на кнопку LED OFF
{
	char buff[4];			// оголошення масиву даних для зберігання команди
	buff[0]=0x40;			// перший символ команди
	buff[3]=0x23;			// останній символ команди
	buff[1]=0;			// формування команди відповідно до протоколу
	buff[2]=0x10;			// формування команди відповідно до протоколу
	m_socket.Send(buff,4);		// відправлення команди на сервер
}
```

Перед початком роботи слід запустити серверну частину програмного забезпечення на комп'ютері, до якого підключений робот. Після цього потрібно відкрити порт та запустити режим приймання команд.

З будь-якого віддаленого комп'ютера за допомогою клієнтського програмного забезпечення можна подавати команди керування. Для підключення до сервера потрібно ввести адрес комп'ютера та номер порту.

Завдання до розділу 6

1. Розробити клієнт-серверний додаток, що виконує функції чату: передавання повідомлень всім користувачам, які підключені до сервера.
2. Розробити програму, що виконує функції серверу, та на запит клієнта передає значення системного часу. Використовувати протокол UDP.
3. Написати програму, яка дозволяє передавати по мережі файли. Використовувати протокол TCP/IP.
4. Написати програму, що дозволяє спостерігати за екраном монітору віддаленого комп'ютера. При реалізації програми використовувати протокол TCP/IP.
5. Написати програму, використання якої дозволяє дистанційно вимкнути комп'ютер. При реалізації проекту використовувати протокол UDP.
6. Написати програму, яка транслює отримані по локальній мережі дані в послідовний порт (COM-порт) комп'ютера.

РОЗДІЛ 7. МІКРОПРОЦЕСОРНІ СИСТЕМИ КЕРУВАННЯ, ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ РІЗНОГО ПРИЗНАЧЕННЯ

7.1. Організація та застосування мікропроцесорних систем керування, збору та обробки інформації

Система збору та обробки інформації (СЗОІ) призначена для інтеграції систем інженерно – технічних засобів (ІТЗ) в єдиний комплекс з метою підвищення ефективності їх використання та комплексного надання інформації стосовно роботи систем ІТЗ оперативному черговому, відповідальним посадовцям і керівництву. Особливо ефективним є застосування СЗОІ на територіально рознесених об'єктах, що мають декілька будівель або філій. У цьому випадку СЗОІ дозволяє створити в організації єдиний інформаційний простір безпеки, що в будь-який момент часу дозволяє мати актуальну інформацію про стан систем безпеки об'єкта та оперативно реагувати на події, які відбуваються у системі.

Обробка інформації полягає в отриманні одних «інформаційних об'єктів» з інших «інформаційних об'єктів» шляхом виконання деяких алгоритмів і є однією з основних операцій, здійснюваних над інформацією, і головним засобом збільшення її обсягу і різноманітності.

На самому верхньому рівні можна виділити числову і нечислову обробку. У зазначені види обробки вкладається різне трактування змісту поняття «дані». При числовий обробці використовуються такі об'єкти, як змінні, вектори, матриці, багатовимірні масиви, константи і т.д. При нечисловій обробці об'єктами можуть бути файли, записи, поля, ієрархії, мережі, відносини і т.д. Інша відмінність полягає в тому, що при числовій обробці зміст даних не має великого значення, в той час як при нечисловій обробці нас цікавлять безпосередні відомості про об'єкти, а не їх сукупність в цілому.

З точки зору реалізації на основі обчислювальної техніки виділяють наступні види обробки інформації:

- послідовна обробка, яка виконується одним мікропроцесором;
- паралельна обробка, яка застосовується при наявності декількох мікропроцесорів;
- конвеєрна обробка, пов'язана з використанням в архітектурі мікропроцесорної системи одних і тих же ресурсів для вирішення різних завдань, причому якщо ці завдання тотожні, то це послідовний конвеєр, якщо завдання однакові – векторний конвеєр.

Інформаційно – вимірювальна система (ІВС) – це сукупність програмно-керованих засобів вимірів і обчислень, що виконують загальне завдання одержання інформації про об'єкт за єдиним алгоритмом. Об'єктом виміру ІВС є звичайно сукупність багатьох величин і залежностей між ними. Засоби інформаційно-вимірювальної техніки з'єднують в єдині системи за допомогою сигналів різного призначення – керування, зв'язку, синхронізації, зразкових, вимірювальних та ін. Сигнали для забезпечення

можливості з'єднання ланок системи повинні мати сумісність за родом, розміром та ін. Поняття «сигнал» є дуже широким. У загальному випадку під сигналом мають на увазі фізичний процес – носій відомостей. Сигнал має різні параметри та характеристики і представляється математичною моделлю.

Типова система збору та обробки інформації будується на базі локальної обчислювальної мережі (ЛОМ) і включає в себе наступне обладнання:

- серверні блоки для отримання і обробки в реальному часі інформації про роботу систем ІТЗ;
- серверні блоки для управління обладнанням СЗОІ, обробки інформації від різних систем, виявлення типових (штатних і позаштатних) ситуацій, вироблення реакції системи на виникнення типових ситуацій;
- серверні блоки для зберігання архівної інформації про події систем ІТЗО (оперативний і довготривалий архіви);
- АРМ адміністратора для контролю працездатності, налаштування і конфігурації СЗОІ;
- АРМ операторів для перегляду інформації СЗОІ в реальному режимі часу і в архівах, оперативного управління системою;
- джерела безперебійного живлення для забезпечення безперервної роботи системи;
- серверні блоки діагностики обладнання СЗОІ;
- мережеве обладнання;
- кабельні та бездротові лінії зв'язку.

СЗОІ отримує інформацію про стан засобів ІТЗО і може реагувати на реєстровані події. Якщо засоби ІТЗ допускають зовнішнє управління, то спеціалізовані контролери СЗОІ перетворять цифрові команди СЗОІ у формат даних засобів. Іноді зворотний зв'язок із засобами ІТЗ об'єкта здійснюється на рівні баз даних. СЗОІ дозволяє здійснити часткове або повне управління функціями засобів ІТСО як ручне, так і автоматичне – на рівні сценаріїв.

Для збору інформації та управління окремими функціями систем ІТЗ використовуються різні методи підключення інтерфейсів і передачі даних.

Архітектура системи із загальною шиною. Архітектура розподіленої системи промислової автоматизації на основі загальної шини показана на рис. 7.1. Для того, щоб отримати дані з модуля або контролера, комп'ютер (або контролер) посилає в шину його адресу і команду запиту даних. Мікропроцесор, що входить до складу кожного модуля або контролера, звіряє адреси на шині з його власною адресою, записаною в ПЗП, і, якщо адреси співпадають, виконує наступну за адресою команду. Команда дозволяє зчитувати дані, що надходять на вхід пристрою, або встановити необхідні дані на його виході.

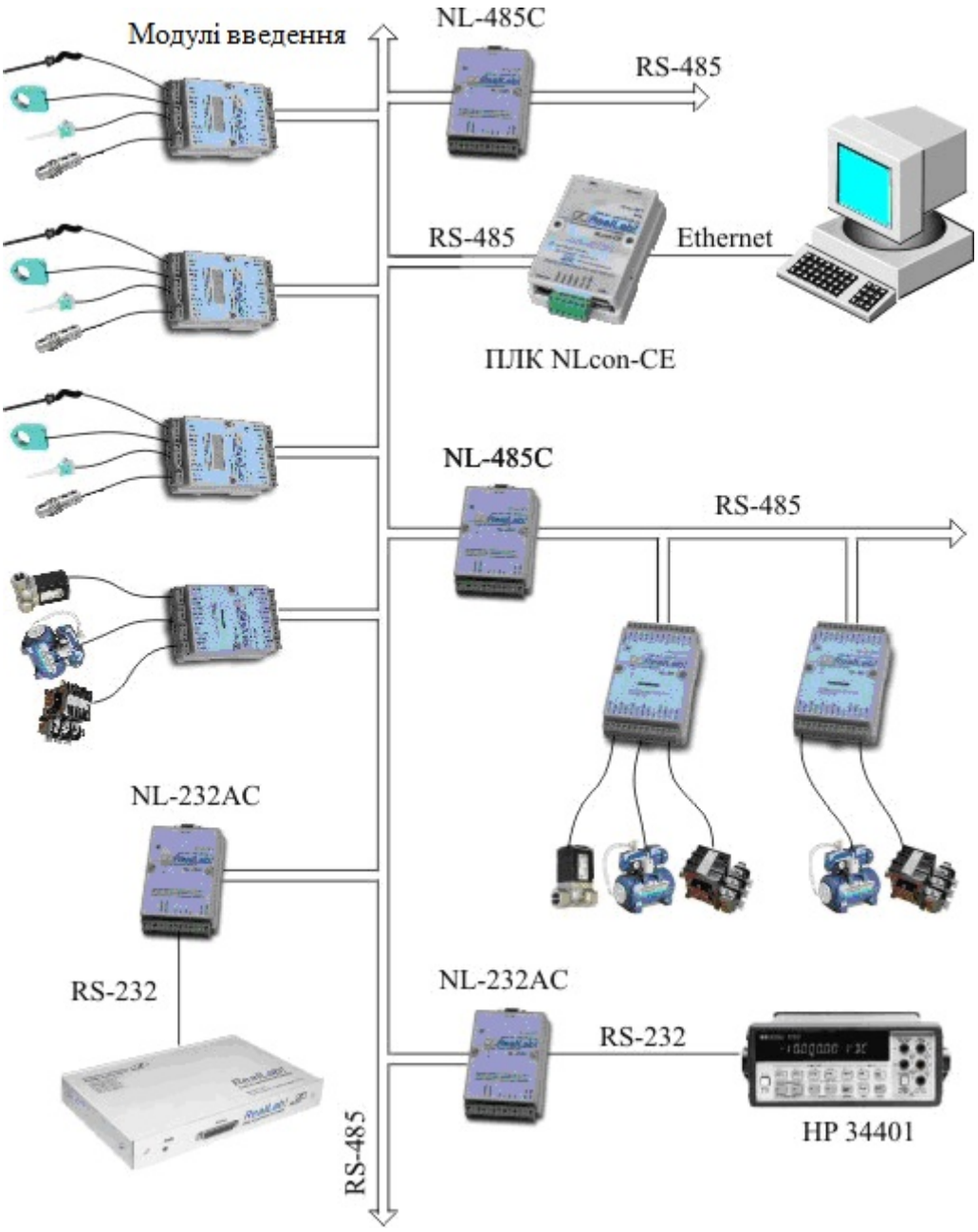


Рис. 7.1.

Недолік архітектури системи із загальною шиною:

– необхідність адресації пристроїв;
– необхідність очікування в черзі.

Багаторівнева архітектура. Промислова мережа може бути підключена не тільки до одного комп'ютера, як показано на рис. 7.1, але і до мережі комп'ютерів, наприклад, до локальної мережі Ethernet (рис. 7.2) або глобальної мережі Internet. Така архітектура автоматизованої системи зручна при колективній роботі з системою автоматизації або для зв'язку технологічного рівня АСУ з управлінським.

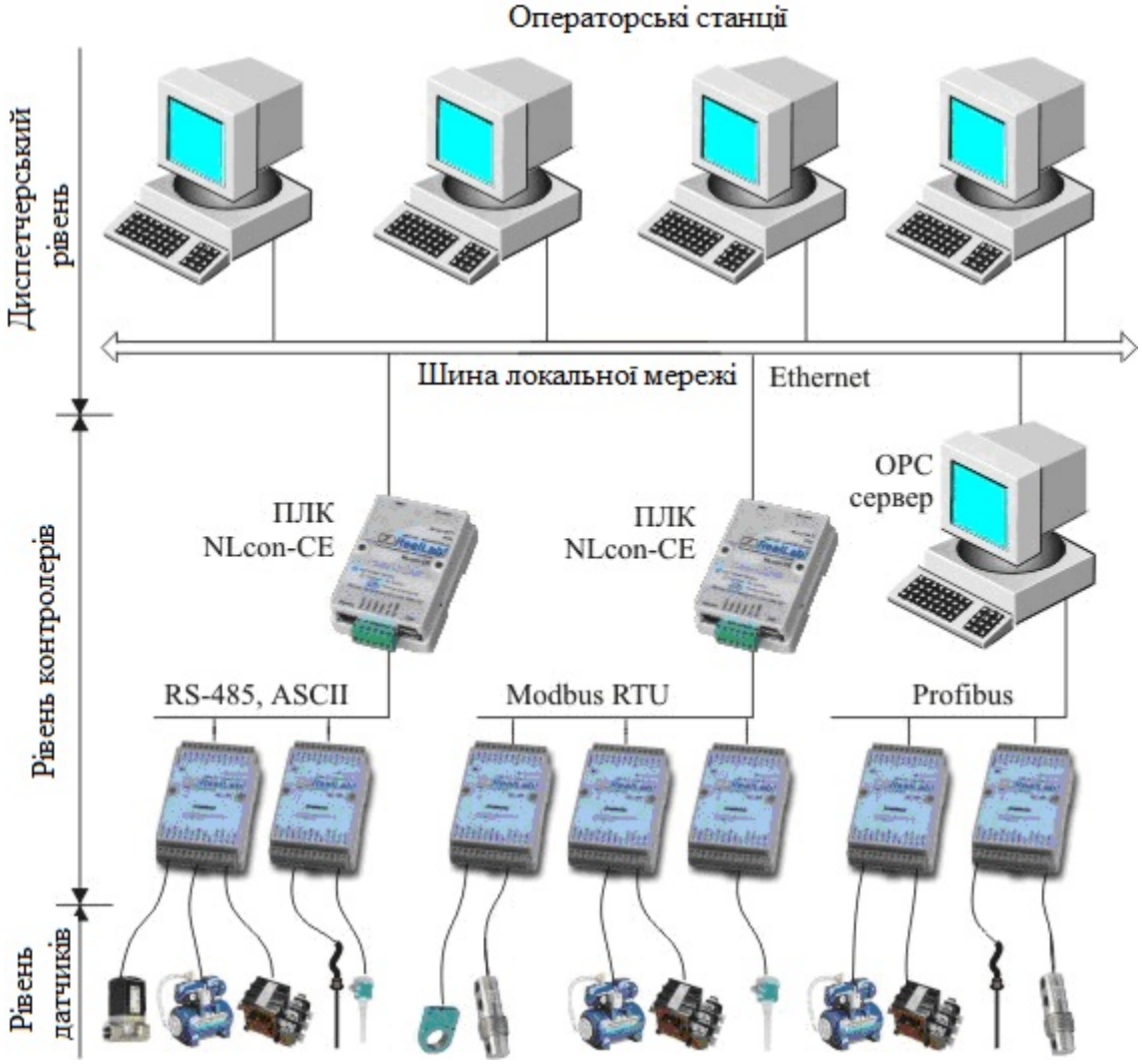


Рис. 7.2.

Системи управління з багаторівневою архітектурою зазвичай будуються по об'єктному принципу, коли структура системи обирається подібній структурі об'єкта автоматизації, а кожна підсистема є локальною, тобто всі зворотні зв'язки замкнуті в межах цієї підсистеми. Кожна локальна підсистема виконує окрему функцію, що задається логікою функціонування всієї системи. Об'єктний принцип побудови дозволяє спростити проектування багаторівневої системи і забезпечити її структурну (архітектурну) надійність.

Аналіз складних систем управління дозволяє виділити в них кілька однорідних рівнів ієрархії.

Нижчий (нульовий) рівень включає в себе датчики та виконавчі пристрої (актуатори): датчики температури, тиску, кінцеві вимикачі, дискретні датчики наявності напруги, вимірювальні трансформатори, реле – пускачі, контактори, електромагнітні клапани, електроприводи та ін. Датчики і актуатори можуть мати інтерфейси типу AS-інтерфейс (ASI), 1-Wire або CAN, HART та інші.

Перший рівень складається з програмованих логічних контролерів і модулів аналого-цифрового і дискретного вводу-виводу, які обмінюються

інформацією з промислової мережі (Fieldbus) типу Modbus RTU, Modbus TCP, Profibus та ін. Іноді модулі вводу – виводу виділяють в окремий рівень ієрархії.

Другий (диспетчерський) рівень складається з робочих станцій – комп'ютерів з людино-машинним інтерфейсом (ЛМІ, HMI – Human Machine Interface), найбільш поширеними варіантами якого є SCADA – пакети.

Третій рівень (управління цехом) з'являється як засіб інтеграції системи АСУ ТП з АСУП – автоматизованою системою управління підприємством. АСУП, залежно від розмірів корпорації, може включати ще більш високий (четвертий) рівень і забезпечувати інтеграцію з вищим керівництвом.

Управління через Інтернет. Взаємодія між клієнтською і серверною частиною SCADA на рис. 7.3 виглядає наступним чином. Коли користувач хоче отримати дані від контролера через Інтернет, він натискає деяку командну кнопку в вікні веб-браузера. Цей запит надсилається з сервера через Інтернет у форматі SOAP-повідомлення. Коли веб-сервер отримує SOAP-повідомлення через TCP порт 80, запит направляється оброблювачу скрипта ASP.NET, який також розташовується на веб-сервері. Веб-сервіс створює запитувані дані або передає управління програмі, яка спілкується з контролером через OPC сервер. Отримані таким чином дані викладаються на веб-сторінку, яку бачить користувач за допомогою веб-браузера.

У разі віддаленого управління на основі WAP користувач отримує доступ до керованої системи через мобільний телефон (GSM-модем). WAP підтримується стандартами GSM, TDMA, CDMA, GPRS.

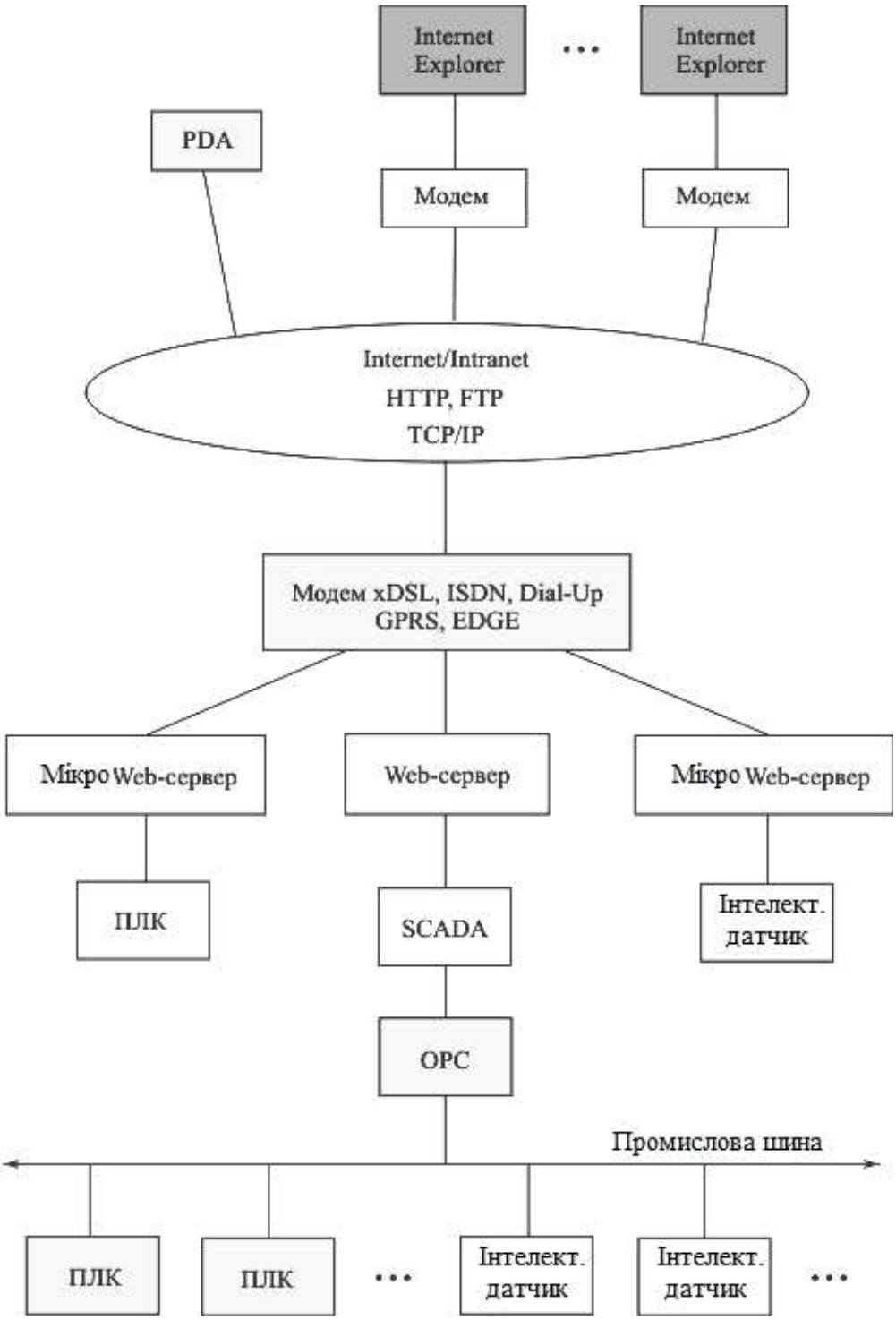


Рис. 7.3.

Відмінною особливістю сучасних систем керування, збору і обробки інформації є те, що в них в єдину систему інтегровані підсистеми забезпечення безпеки виробництва різних компаній. При цьому інтегрувати доводиться не тільки сучасне цифрове обладнання, а й аналогові системи.

7.2. Мікропроцесорна система охоронно-пожежної сигналізації з передачею SMS-повідомлень по мережі мобільного зв'язку

Система охоронно-пожежної сигналізації (ОПС) може бути простою, і працювати за рахунок використання реле, а може мати спеціальне програмне забезпечення, і входити в загальну систему безпеки об'єкта.

Основними елементами ОПС є:

1. Сповіщувачі, до яких відносяться:
   - магнітоконтактні датчики, які встановлюються в дверних і віконних отворах;
   - акустичні датчики забезпечують захист скла і реагують на звук скла, що розбивається (встановлюються на стінах або стелях);
   - датчики обсягу контролюють захист внутрішнього об'єму приміщення шляхом реакції на переміщення.

Для вирішення конкретних завдань встановлюються різні датчики: оптичні, теплові, радіохвильові, ультразвукові. Вони можуть мати різну форму і володіти різною порогом чутливості. Розрізняють датчики пасивні і активні. Активні датчики генерують світловий, акустичний або радіосигнал, і контролюють його параметри. Пасивні датчики відстежують зміни в стані навколишнього середовища (температуру, тиск та ін.).

2. Приймально-контрольні прилади (ПКП) – це система пристроїв, які реагують на сигнали сповіщувачів і передають інформацію про виникнення тривожної ситуації. Сюди відносяться центральні станції, контрольні панелі та інші прилади. Для невеликих приміщень частіше використовуються неадресні оповіщувачі, а для більш великих – адресні та адресно-аналогові. Вони забезпечують більш надійний захист від порушення основних комунікацій. ПКП приводять в дію звукові, світлові оповіщувачі, забезпечують автодозвон по заданих телефонах, забезпечують підключення до пульту централізованої охорони.

Всі пристрої, крім оповіщувачів, відносяться до периферійних пристроїв, які можуть розрізнятися:

- кількістю шлейфів, що підключаються;
- їх призначенням (пожежна або охоронно-пожежна сигналізація);
- способом підключення (адресне або пряме);
- можливістю нарощування і підключення до комп'ютерної мережі.

Далі наводиться перелік необхідних функцій, що покладаються на охоронно-пожежну сигналізацію:

- вимірювання температури в контрольованому приміщені;
- контроль наявності осередку загоряння за допомогою датчиків;
- аналіз складу повітря на наявність небезпечного газу;
- оцінка основних динамічних характеристик системи в різних критичних ситуаціях;
- контроль стану дверей та вікон приміщення, з метою перевірки факту проникнення;
- формування сигналів на включення світлової аварійно-попереджувальної сигналізації;

– формування сигналів на включення звукової аварійно-попереджувальної сигналізації;
– включення вентиляторів при збільшені рівня небезпечного газу в приміщенні;
– формування сигналів тривоги оператору;
– відображення параметрів даних на рідкокристалічному індикаторі;
– формування АТ-команд на GSM-модуль для передачі інформації абоненту на мобільний телефон;
– прийом команд від GSM-модему та їх виконання.

Наступним етапом після визначення функцій системи охоронно-пожежної сигналізації, що проектується, є розробка її структурної схеми. Структурна схема – це сукупність елементарних ланок об'єкта і зв'язків між ними, один з видів графічної моделі. Під елементарною ланкою розуміють частину об'єкта, яка реалізує елементарну функцію. В системі охоронно-пожежної сигналізації умовно можна виділити п’ять підсистем:

1. Набір аналогових та дискретних датчиків, за допомогою який контролюється стан зовнішнього середовища.

2. Блок відображення інформації, що представляє собою алфавітно-цифровий рідкокристалічний індикатор та дискретні світлодіоди (рідкокристалічний індикатор відображає значення температури, стан дискретних датчиків, рівень загазованості приміщення та наявність інфрачервоного випромінювання, а світлодіоди відображають стан виконавчих реле).

3. Мікроконтролер, в якому програмно реалізована логіка роботи системи, виконується прийняття рішень, оптимізуючих динамічні характеристики системи, що знижують ймовірність помилкових спрацьовувань.

4. Блок виконавчих реле, за допомогою якого виконується керування виконавчими механізмами (ВР1 – керування насосом для гасіння пожежі, ВР2 – керування світловою сигналізацією, ВР3 – керування звуковою сигналізацією, ВР4 – керування вентилятором, що вмикається при виявленні за допомогою датчика небезпечної концентрації газу на контрольованому об’єкті).

5. Радіомодем (GSM-модуль), який виконує передачу повідомлення на стільниковий телефон по команді, яка надходить з мікроконтролера (для керування радіомодемом потрібно перетворення фізичних рівнів сигналів за допомогою спеціального перетворювача).

Структурна схема визначає основні функціональні частини пристрою, їх призначення та взаємозв'язок, і служить для загального ознайомлення з системою. На структурній схемі розкривається не принцип роботи окремих функціональних частин виробу, а тільки взаємодія між ними. Тому складові частини системи охоронно-пожежної сигналізації зображають спрощено у вигляді прямокутників довільної форми. На рис. 7.4 представлена структурна схема системи збору та обробки інформації.

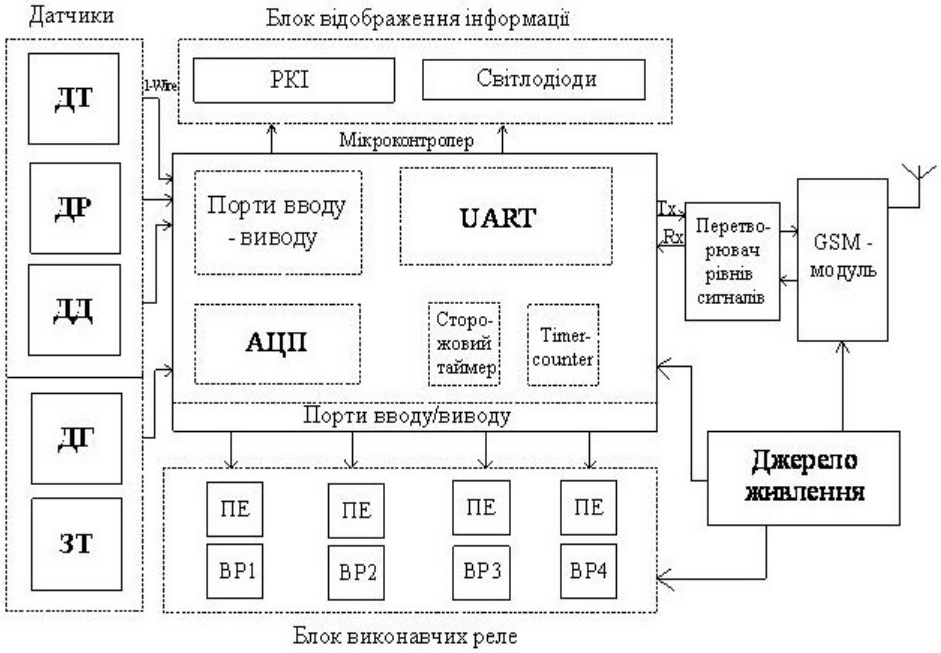


Рис. 7.4.

На рисунку використано наступні позначення: ДТ – датчик температури; ДР – датчик руху; ДГ – датчик газу; ДД – дискретні датчики; ЗТ – задавач температури; АЦП – аналого-цифровий перетворювач; РКІ – рідкокристалічний індикатор; ПЕ – підсилювальний елемент; ВР1-ВР4 – виконавче реле; UART – універсальний асинхронний приймач/передавач.

Метою проведення процедур збору й обробки інформації є одержання достовірної інформації про стан об’єкта і раціональна організація процесів керування, контролю, виміру з високими якісними характеристиками. Збір даних виконується за допомогою блоку датчиків.

Блок датчиків є джерелом інформації для розроблювального пристрою. Він складається з двох частин: блок аналоговий датчиків та блок цифрових датчиків.

Сигнал з блоку аналогових датчиків надходить на АЦП, де він оцифровується. АЦП, який вбудований в мікроконтролер, має розрядність 10 біт, що цілком достатньо для виконання функцій контролю та керування, оскільки в системі потрібно виявити не точне значення концентрації небезпечного газу в приміщенні, а факт перевищення концентрації деякої межі, що буде слугувати сигналом на вмикання сигналізації та вентилятора системи витяжки.

Сигнал з блоку цифрових датчиків надходить на порти вводу-виводу мікроконтролера. Дискретними датчиками можуть виступати як датчики типу «сухий контакт», які встановлюються на вікна та двері, та дозволяють виявити факт несанкціонованого проникнення в приміщення, так і датчики диму для контролю оптичної щільності повітря, теплові порогові сповіщувачі параметрів фізичних величин, інтелектуальні датчики газу або інші підсистеми, що виконують охоронні функції. Мікроконтролер виконує обробку сигналів з усіх

датчиків у відповідності з закладеним алгоритмом роботи системи, виводить інформацію на рідкокристалічний індикатор. З температурного датчику сигнал надходить до мікроконтролера по інтерфейсу 1-Wire. Перевагою цього інтерфейсу є наявність різноманітних цифрових датчиків фізичних параметрів середовища (температури, тиску, вологості та ін.), простий інтерфейс зв'язку з ведучим мікроконтролером, велика довжина ліній зв'язку (до 300 м), велика кількість пристроїв, що підключаються, можливість легкої реконфігурації мережі.

У випадку виявлення пожежі або несанкціонованого проникнення на об’єкт, інформація про стан в приміщенні передається до GSM-модуля за допомогою АТ-команди, і далі відбувається передача повідомлення на стільниковий телефон абонента.

При проектуванні системи необхідно враховувати набір системних принципів, які допоможуть досягнути мети функціонування системи. Системні принципи, які враховуються, при проектуванні системи охоронно-пожежної сигналізації:

1. Принцип причинності – один із самих загальних фізичних принципів, що встановлює допустимі межі впливу подій один на одного. При виявленні пожежі в контрольованій системою зоні (причина), виконується формування сигналів керування на виконавчі механізми (наслідки) з метою локалізації та ліквідації осередку загоряння.
2. Принцип «чорного ящику» – систему або її підсистеми можна розглядати у вигляді непрозорого «ящика», виділеного з навколишнього середовища. В система охоронно-пожежної сигналізації у вигляді «чорного ящику» розглядається радіомодем. В даному випадку не є принципово важливим його внутрішня будова. При використанні радіомодему звернення до нього виконується за допомогою АТ-команд (вхідний сигнал). Радіомодем реагує на цю команду тим, що передає сигнал на мобільний телефон через мережу стільникового зв’язку (вихідний сигнал). Отже, внутрішня будова радіо модему невідома, але чітко визначені його функції, вхідні та вихідні сигнали.
3. Принцип різноманіття – дозволяє використовувати безліч кібернетичних та математичних моделей для опису окремих елементів і системи в цілому. Використання цього принципу робить систему більш надійною та зменшує ймовірність помилкового спрацювання. Так, при виході з ладу, наприклад, датчику температури, осередок загоряння може бути виявлено інфрачервоним датчиком або датчиком газу.
4. Принцип розвитку – це врахування змінності системи, її здатності до розвитку, адаптації, розширенню, заміні частин, накопиченню інформації. В системі охоронно-пожежної сигналізації цей принцип враховано шляхом використання цифрового інтерфейсу 1-Wire, що дозволяє підключати при необхідності додаткові датчики без

необхідності принципової зміни структури системи, а також наявності декількох дискретних входів, до яких можна підключити дискретні датчики, об’єднавши їх по схемі АБО.

5. Принцип узгодження мети – для визначення якості та ефективності функціонування системи треба представити її як частину більш загальної і проводити оцінку зовнішніх властивостей досліджуваної системи щодо цілей і завдань суперсистеми. Наявність цифрових інтерфейсів дозволяє інтегрувати розроблену систему у суперсистему, якою може виступати автоматизована система управління деяким технологічним процесом, або охоронна система підприємства. У такому випадку розроблена система виступає як окремий елемент іншої системи, при цьому не порушується мета та завдання функціонування системи.

На основі викладеного можна сказати, що система збору й обробки інформації є складний технічний пристрій, що у загальному випадку здійснює автоматичне обчислення сукупних параметрів керування, контролю, виміру об'єктів, різних по своїй фізичній сутності, і реєстрацію результатів обчислень. Тобто пристрої подібного роду є глобальними, багатофункціональними системами введення-виведення інформації.

На рис. 7.5 зображена принципова схема модуля протипожежного захисту, яку було розроблено у програмному середовищі Proteus.

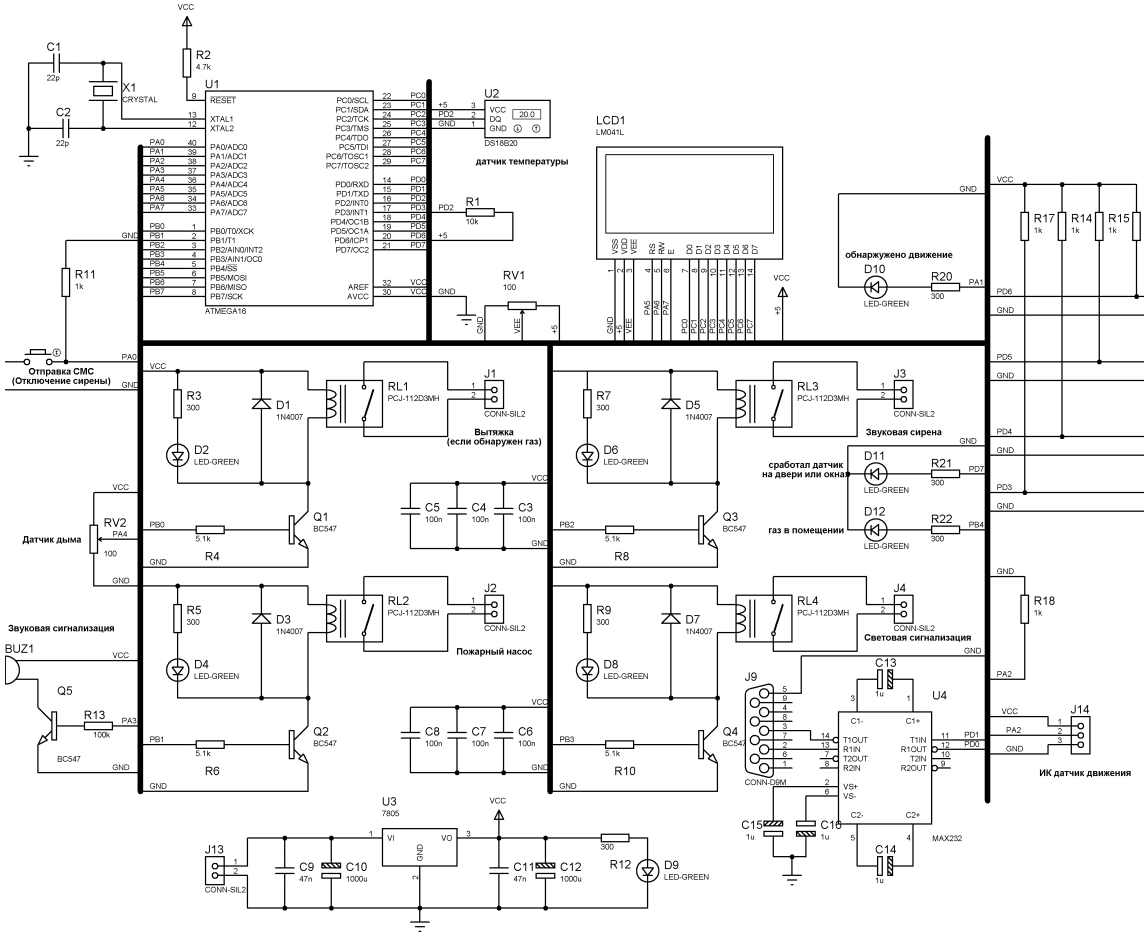


Рис. 7.5.

В системі використовуються мікроконтролер ATMega16, температурний датчик DS18B20, датчик газу HX-2008EX-A (який на схемі зображено у вигляді потенціометра, оскільки на виході датчика газу формується сигнал напруги, яка пропорційна концентрації газу), дискретні датчики, які будуть визначати стан – відкрито або зачинено для вікон та дверей, датчик руху HC-SR501, алфавітно-цифровий рідкокристалічний індикатор (РКІ) та виконавчі реле.

Сигнал з блоку цифрових датчиків надходить на порти введення/виведення мікроконтролера. Дискретні датчики можуть виступати як датчики типу «сухий контакт». Для того, щоб на вході мікроконтролера не виникало завад, кожен вивід, до якого підключено дискретний датчик, має підтягуючий резистор до шини живлення. Саме завдяки використанню цих компонентів здійснюється збір дискретної інформації з територіально розосереджених датчиків (моніторинг стану дверей та вікон). Подібні ж прилади (виконавчі реле) забезпечують керування перемиканням будь-яких видів силового устаткування, які мають два робочих стани: включення/виключення (монітор, вентилятор, сирена, насос та ін.). Сигнал з чотирьох дискретних датчиків та інфрачервоного датчику руху відображається на РКІ, що дозволяє відстежити ситуацію в приміщенні. Також на РКІ представлена інформація: виміряна температура з датчику DS18B20, покази датчику газу. Вся інформація на РКІ допомагає оператору відстежити та проконтролювати ситуацію в приміщенні.

При перших проявах надзвичайної ситуації п’єзоелектричний випромінювач повідомить персонал звуковим сигналом. П’єзоелектричний випромінювач є активним приладом, з вбудованою системою керування. Для його вмикання достатньо подати живлення на виводи. Струм, що споживає випромінювач при роботі складає близько 30 мА, а, отже, потрібно використовувати транзисторний ключ для того, щоб керувати випромінювачем від мікроконтролера. Для керування засобами гасіння пожежі використовується виконавчі реле, до яких підключається системи керування насосом, світловою та звуковою сигналізацією.

Також в схемі є дві дискретні кнопки, які дозволяють оператору власноруч ініціювати процес відправки повідомлення про виникнення пожеже-небезпечної ситуації, а також перевести систему у початковий стан після її спрацювання, якщо пожежу було локалізовано та зупинено.

Головним вузлом схеми є мікроконтролер ATMega16 – 8-розрядний високопродуктивний мікроконтролер з архітектурою AVR, з малим енергоспоживанням. Логіка керування організує взаємодію всіх вузлів процесора, перенаправляє дані, синхронізує роботу процесора із зовнішніми сигналами, а також реалізує процедури вводу і виводу інформації. Тактова частота роботи мікроконтролера складає 8 МГц. Ця частота задається зовнішнім кварцовим резонатором. У відповідності до технічної документації на мікроконтролер, кожен вивід кварцового резонатора повинен бути підключений через ємність 22 пФ до загального

провідника. Також потрібно підключити вивід Reset до шини живлення через резистор опором 4,3 кОм.

Конденсатори в схемі використовуються для зменшення високочастотних завад, що виникають при комутації реле, та їх впливу на мікроконтролер.

Оскільки в схемі використовуються виконавчі реле, для роботи яких потрібен струм близько 40 мА, було використано транзисторні ключі, що виконують підсилення сигналу керування. Безпосередньо до виводу мікроконтролера неможна підключити реле, адже максимальний струм виводу мікроконтролера складає близько 20 мА, а цього недостатньо для спрацювання реле. Для захисту від проти-ЕРС при комутації обмотки реле в схемі присутні діоди 1Т4007. Для індикації стану реле використовуються світлодіоди зі струмообмежуючими резисторами.

Для підключення до системи GSM-радіомодему використовується інтерфейс RS232.

Програма збору та передачі даних складається з декількох спеціальних модулів (рис. 7.6): головний модуль програми, модуль вимірювальних параметрів, модуль відправлення повідомлення.

Головний модуль в свою чергу включає наступні модулі: визначення температури, визначення концентрації газу, опрацювання дискретних сигналів, формування сигналів керування та відображення інформації.

В головній функції програми main() виконується циклічний виклик модулів визначення температури, визначення концентрації газу за допомогою аналого-цифрового перетворення напруги на виході датчику газу, аналіз станів дискретних входів, до яких підключені дискретні датчики. Після цього виконується виклик підпрограми аналізу виміряних параметрів та прийняття рішення у відповідності до правил, який з виконавчих механізмів включати.

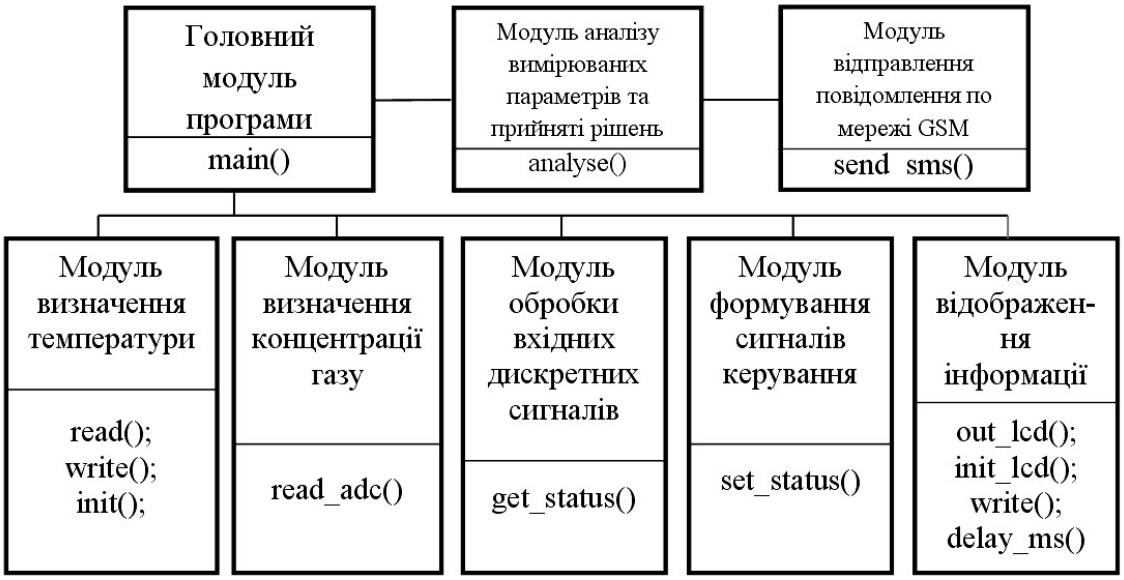


Рис. 7.6.

Якщо температура перевищує поріг в 50 ^{0}C, виконується відправка повідомлення на мобільний телефон, вмикається світлова та звукова сигналізація, та пожежний насос. Для цього використовується програмний модуль формування сигналів керування. При виявленні небезпечної концентрації газу формується сигнал керування вентилятором, а також звукова та світлова сигналізація. При спрацюванні дискретних датчиків у разі виявлення факту несанкціонованого проникнення в приміщення вмикається сигналізація та відправляється повідомлення на мобільний телефон.

Текст програми мікроконтролера системи охоронно-пожежної сигналізації представлено нижче.

```
// підключення необхідних бібліотек, функції з яких використовуються в програмі
#include "iom16v.h"
#include "macros.h"
#include "string.h"

// визначення ідентифікаторів та послідовності символів, на який він буде заміщений
// при компіляції програми
#define WIRE_DDR DDRD
#define WIRE_PIN PIND
#define WIRE_PORT PORTD
#define PIN_NUM 2
#define RS() PORTA|=0b00100000          // встановлення лог. 1 на виводі RS РКІ
#define NRS() PORTA&=~0b00100000        // встановлення лог. 0 на виводі RS РКІ
#define RW() PORTA|=0b01000000          // встановлення лог. 1 на виводі RW РКІ
#define NRW() PORTA&=~0b01000000        // встановлення лог. 0 на виводі RW РКІ
#define E() PORTA|=0b10000000           // встановлення лог. 1 на виводі E РКІ
#define NE() PORTA&=~0b10000000         // встановлення лог. 0 на виводі E РКІ
#define LCD PORTC
#define DDLCD DDRC

// оголошення глобальних змінних для використання в програмі
char data[3];
char ready; // змінна-прапорець

// обробник переривання таймера-лічильника
#pragma interrupt_handler timer0_ovf_isr:iv_TIM0_OVF
void timer0_ovf_isr(){
TCCR0=0; // зупинка таймера
ready=1; // запил значення 1 в змінну
}

// оброюник переривання отримання даних по послідовному інтерфейсу USART
#pragma interrupt_handler uart0_rx_isr:12
void uart0_rx_isr(void){
char d;
d=UDR; // зчитування отриманих даних в змінну
}
```

```
// реалізація функції формування часової затримки в мікросекундах
void delay_us(char us){
ready=0;
TCNT0=255-us; // встановлення початкового значення таймера/лічильника
TCCR0=0x02; // запуск таймера з дільником частоти 8
while(ready==0) NOP(); // очікування доки не завершиться часовий інтервал
}

// реалізація функції затримки в мілісекундах
void delay(int ms)
{
int c1,c2;
for(c1=0; c1<ms; c1++)
for(c2=0; c2<970; c2++) NOP(); // в циклі виконується пуста операція
}

// функція ініціалізації датчика з інтерфейсом 1-Wire
char reset()
{
char rs;
WIRE_DDR|=(1<<PIN_NUM); // налаштування виводу на вихід
delay_us(245);                  // формування часової затримки 490 мкс
delay_us(245);
WIRE_DDR&=~(1<<PIN_NUM); // налаштування виводу на вхід
delay_us(65);                   // формування часової затримки 65 мкс
if((WIRE_PIN&(1<<PIN_NUM))!=0) rs=1; // аналіз стану лінії
else rs=0;
delay_us(245);                  // формування часової затримки 425 мкс
delay_us(180);
return rs; // повернення із функції значення, що показує тан лінії
}

// реалізація функції відправлення 1 біта по інтерфейсу 1-Wire
void send_bit(char bit)
{
WIRE_DDR|=(1<<PIN_NUM); // налаштування виводу на вихід
delay_us(2);                    // формування часової затримки 2 мкс
if(bit==0){                     // якщо передається 0 то формується часова затримка
delay_us(65);                   // формування часової затримки 65 мкс
}
else                            // якщо передається 1, тоді
{
WIRE_DDR&=~(1<<PIN_NUM); // встановлення лог.1 на виводі PIN_NUM
delay_us(65);                   // формування часової затримки 65 мкс
}
WIRE_DDR&=~(1<<PIN_NUM); // скидання лог.1 на виводі PIN_NUM
delay_us(5);                    // формування часової затримки 5 мкс
}

// реалізація функції зчитування 1 біта по інтерфейсу 1-Wire
char receve_bit()
{
```

```
char state;                    // оголошення додаткової змінної
WIRE_DDR|=(1<<PIN_NUM); // налаштування вивоу на вихід
delay_us(2);                   // формування часової затримки 2 мкс
WIRE_DDR&=~(1<<PIN_NUM); // налаштування виводу на вхід
delay_us(7);                   // формування часової затримки 7 мкс
if((WIRE_PIN&(1<<PIN_NUM))!=0) state=1; // аналіз стану лінії в запис 1
else state=0;                  // запис 0 в змінну якщо на виводі PIN_NUM рівень лог.0
delay_us(50);                  // формування часової затримки 50 мкс
return state;                  // повернення значення змінної із функції
}

// реалізація функції відправлення байта даних по інтерфейсу 1-Wire
void send_byte(char data){
char i;
for(i=0; i<8; i++){
if((data&0x01)!=0) send_bit(1);      // в циклі послідовно виконується аналіз бітів та їх
else send_bit(0);                    // відправлення по інтерфейсу 1-Wire
data=(data>>1);                      // зсув вмісту змінної на 1 розряд праворуч
}
}

// реалізація функції зчитування байта даних по інтерфейсу 1-Wire
char read_byte(){
char i;
char data=0;
for(i=0; i<8; i++){                  // в циклі послідовно виконується аналіз бітів та їх
if(receve_bit()==1) data|=(1<<i);    // та їх запис в змінну
}
return data;                         // поверненні отриманого байта в основну програму
}

// функція початкового налаштування мікроконтролера
void init_cpu(){
DDRA=0b11101010;          // налаштування виводів порта A на вхід (0) або вихід (1)
DDLCD=0xFF;               // порт, до якого підключено РКІ, налаштовується на вихід
DDRB=0b00011111;          // налаштування виводів порта B на вхід (0) або вихід (1)
DDRD=0b10000000;          // налаштування 7 бта порта D на вихід
ADCSRA=0x86;              // вмикання АЦП та налаштування дільника частоти

UBRRL=0x33;               // встановлення швидкості передачі даних 9600 біт/с
UBRRH=0;
UCSRB=0x98;               // дозвіл роботи передавача та приймача USART

SEI();                    // дозвіл обробки в програмі будь-яких переривань
NRS();                    // лінії керування РКІ передено в стан низького рівня
NRW();
NE();
}

// реалізація функції зчитування даних з РКІ
unsigned int read_adc(char channel){
char a;                  // оголошення додаткових змінних
```

```
char b;
int val=0;
ADMUX=channel;              // вибір каналу для аналого-цифрового перетворення
ADCSRA|=0x40;               // запуск перетворення
while((ADCSRA&0x10)==0); // очікування готовності даних
a=ADCL;                     // зчитування отриманих даних з регістрів
b=ADCH;
val=b;                      // формування 10-бітного результату
val=(val<<8);
val=val+a;
return val;                 // повернення результату АЦП в основну програму
}

// реалізація функції формування тактового сигналу для запису даних в РКІ
void write()
{
E();                        // встановити рівень лог. 1 на лінії E РКІ
delay(1);                   // формування часової затримки 1 мс
NE();                       // встановити рівень лог. 0 на лінії E РКІ
delay(1);                    // формування часової затримки 1 мс
}

// реалізація функції початкового налаштування РКІ
void init_lcd(){
DDLCD=0xFF;         // порт, до якого підключено РКІ, налаштовується на вихід
LCD=0x38;           // формування команди вмикання 8-бітної шини керування
write();            // запис команди
LCD=0x0C;           // команда налаштування кількості рядків та розміру шрифту
write();            // запис команди
LCD=0x06;           // налаштування режиму відображення курсору
write();            // запис команди
LCD=0x02;           // команда встановлення напрямку зсуву курсору
write();            // запис команди
LCD=0x01;           // команда очищення дисплею
write();            // запис команди
}

// реалізація функції відправлення повідомлення про температуру
void send_sms_temp(){
char i;
char sms[]=" Warning! Temperature too high!";      // повідомлення для відправлення
char CMD[]="AT+CMGS==\"+380612231234\""; // номер телефону
CMD[8]="";                          // додаткові масиви для збереження значень
CMD[19]="";
for(i=0; i<strlen(CMD); i++){
UDR=CMD[i];                         // передавання даних по USART в GSM-модем
while(!(UCSRA&(1<<UDRE)));      // очікування завершення передавання даних
}

UDR=0x0D;                           // передавання команди завершення даних
while(!(UCSRA&(1<<UDRE)));      // очікування завершення передавання даних
delay(50);                          // формування часової затримки 50 мс
```

```
for(i=0; i<strlen(sms); i++){
UDR=sms[i];
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}
UDR=0x1A;
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}

// реалізація функції відправлення повідомлення про виявлення витоку газу
void send_sms_gas(){
char sms[]=" Warning! Gas in room!";
char CMD[]="AT+CMGS=\"+380612231234\"";   // команда та номер телефону
char i;
for(i=0; i<strlen(CMD); i++){
UDR=CMD[i];                    // передавання даних по USART в GSM-модем
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}
UDR=0x0D;
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
delay(50);                     // формування часової затримки 50 мс
for(i=0; i<strlen(sms); i++){
UDR=sms[i];                     // передавання повідомлення в GSM-модем
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}
UDR=0x1A;
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}

// реалізація функції відправлення повідомлення про тривогу
void send_sms_alarm(){
char sms[]=" Alarm!";
char CMD[]="AT+CMGS==\"+380612231234\""; // команда та номер телефону
char i;
for(i=0; i<strlen(CMD); i++){
UDR=CMD[i];                    // передавання даних по USART в GSM-модем
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}
UDR=0x0D;                      // передавання команди завершення даних
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
delay(50);                     // формування часової затримки 50 мс
for(i=0; i<strlen(sms); i++){
UDR=sms[i];                    // передавання повідомлення в GSM-модем
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}
UDR=0x1A;
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}

// реалізація функції виклику абонента
void ring(){
char CMD[]="ATD+380612231234;"; // команда виклику абонента
char i;
```

```
for(i=0; i<strlen(CMD); i++){
UDR=CMD[i];                    // передавання даних по USART в GSM-модем
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
}
UDR=0x0D;                      // передавання команди завершення даних
while(!(UCSRA&(1<<UDRE)));     // очікування завершення передавання даних
delay(50);                     // формування часової затримки 50 мс
}

// головна функція програми
void main()
{
// оголошення додаткових змінних для використання в прорамі
char gasFlag=0;
char tempFlag=0;
char switchFlag=0;
char inputs;           // змінна для зберігання значення, отриманого з дискретних входів
int gassensor;         // змінна для берігання значення, зчитаного з датчика газу
char th, tl;
int T;                 // змінна для берігання значення, зчитаного з датчика температури
char k;
char active=0;
char state=0;
// оголошення рядків для виведення на екран РКІ
char pir1[]="Alarm!";
char pir2[]="Normal";
char on[]="ON  ";
char off[]="OFF ";
char s1[]="Temperature=";
char gas[]="Gas: ";
int i;
delay(2000);           // формування часової затримки 2 с
init_cpu();            // виклик функції ініціалізація мікроконтролера
init_lcd();            // виклик функції ініціалізації РКІ

TIMSK=0x01;            // дозвіл переривання по переповненню таймера-лічильника
SEI();                 // дозвіл обробки будь-яких переривань
while(1)               // головний цикл програми
{
reset();               // ініціалізація датчика температури з інтерфейсом 1-Wire
send_byte(0xcc);       // відправлення команди початку вимірювання в датчик
send_byte(0x44);
delay(1000);           // формування часової затримки 1 с
reset();               // ініціалізація датчика температури з інтерфейсом 1-Wire
send_byte(0xcc);       // відправлення команди зчитування даних з датчика
send_byte(0xbe);
tl=read_byte();        // зчитування молодшого байта температури
th=read_byte();        // зчитування старшого байта температури
T=th;                  // формування результату про отримане значення температури
T=(T<<8);              // зсув вмісту змінної на 8 розрядів ліворуч
T=T+tl;
T=(T>>4);              // зсув вмісту змінної на 4 розрядів праворуч
```

```
data[0]=T;              // запис значення температури в 0 елемент масиву

NRS();                  // переведення РКІ в режим передавання команд
delay(1);               // формування часової затримки 1 мс
LCD=0x01;
write();

RS();                   // переведення РКІ в режим передавання даних
delay(1);               // формування часової затримки 1 мс
for(i=0; i<strlen(s1); i++){
LCD=s1[i];              // виведення елементу масива на шину даних
write();                // виклик функції формування тактового сигналу РКІ
}
LCD=data[0]/10+'0';
write();                // виклик функції формування тактового сигналу РКІ
LCD=data[0]%10+'0';
write();                // виклик функції формування тактового сигналу РКІ
LCD='C';
write();                // виклик функції формування тактового сигналу РКІ

inputs=PIND;
data[1]=inputs>>3;
if((PINA&0b00000100)!=0) data[1]|=0b00010000;
else data[1]&=~0b00010000;

NRS();                  // переведення РКІ в режим передавання команд
LCD=0b11000000;
write();                // виклик функції формування тактового сигналу РКІ
RS();                   // переведення РКІ в режим передавання даних
delay(1);               // формування часової затримки 1 мс
if((data[1]&0b00001000)==0)
{
for(i=0; i<strlen(on); i++){
LCD=on[i];              // виведення елементу масива на шину даних
write();                // виклик функції формування тактового сигналу РКІ
}
}
else
{
for(i=0; i<strlen(off); i++){
LCD=off[i];             // виведення елементу масива на шину даних
write();                // виклик функції формування тактового сигналу РКІ
}
}

if((data[1]&0b00000100)==0)
{
for(i=0; i<strlen(on); i++){
LCD=on[i];              // виведення елементу масива на шину даних
write();                // виклик функції формування тактового сигналу РКІ
}
}
```

```
else
{
for(i=0; i<strlen(off); i++){
LCD=off[i];              // виведення елементу масива на шину даних
write();                 // виклик функції формування тактового сигналу РКІ
}
}
if((data[1]&0b00000010)==0){
for(i=0; i<strlen(on); i++){
LCD=on[i];               // виведення елементу масива на шину даних
write();                 // виклик функції формування тактового сигналу РКІ
}
}
else
{
for(i=0; i<strlen(off); i++)
{
LCD=off[i];              // виведення елементу масива на шину даних
write();                 // виклик функції формування тактового сигналу РКІ
}
}
if((data[1]&0b00000001)==0)
{
for(i=0; i<strlen(on); i++){
LCD=on[i];               // виведення елементу масива на шину даних
write();                 // виклик функції формування тактового сигналу РКІ
}
}else{
for(i=0; i<strlen(off); i++)
{
LCD=off[i];              // виведення елементу масива на шину даних
write();                 // виклик функції формування тактового сигналу РКІ
}
}
gassensor=read_adc(4);
data[2]=(gassensor>>2)/2.6;
NRS();                   // переведення РКІ в режим передавання команд
delay(1);                // формування часової затримки 1 мс
LCD=0x90;
write();                 // виклик функції формування тактового сигналу РКІ
RS();                    // переведення РКІ в режим передавання даних
delay(1);                // формування часової затримки 1 мс
for(i=0; i<strlen(gas); i++)
{
LCD=gas[i];              // виведення елементу масива на шину даних
write();                 // виклик функції формування тактового сигналу РКІ
}
LCD=data[2]/10+'0';
write();                 // виклик функції формування тактового сигналу РКІ
LCD=data[2]%10+'0';
write();                 // виклик функції формування тактового сигналу РКІ
LCD='%';
```

```
write();                // виклик функції формування тактового сигналу РКІ

NRS();                  // переведення РКІ в режим передавання команд
delay(1);               // формування часової затримки 1 мс
LCD=0x80|0x50;
write();                // виклик функції формування тактового сигналу РКІ
RS();                   // переведення РКІ в режим передавання даних
delay(1);               // формування часової затримки 1 мс
if((data[1]&0b00010000)!=0)
{
for(i=0; i<strlen(pir1); i++)
{
LCD=pir1[i];            // виведення елементу масива на шину даних
write();                // виклик функції формування тактового сигналу РКІ
}
}else{
for(i=0; i<strlen(pir2); i++)
{
LCD=pir2[i];            // виведення елементу масива на шину даних
write();                // виклик функції формування тактового сигналу РКІ
}
}
if((data[1]!=0b00001111)&& (active==1))
{
PORTB|=0b00010000; // встановлення біту 4 порта в в лог.1
if(switchFlag==0){
PORTA|=0b00001000; // встановлення біту 3 порта А в лог.1
send_sms_alarm();   // виклик функції відправлення повідомлення
switchFlag=1;
}
}else{
PORTA&=~0b00001000;     // встановлення біту 3 порта А в лог.0
PORTB&=~0b00010000;     // встановлення біту 4 порта В в лог.0
switchFlag=0;
}
if((data[0]>60) && (active==1))
{
PORTB|=0b00000010;
ring();                 // виклик функції телефонування абоненту
delay(10000);           // формування часової затримки 10 с
}
else if((data[0]>40) && (active==1))
{
PORTB|=0b00000010; // встановлення біту 1 порта В в лог.1
if(tempFlag==0){
send_sms_temp();    // виклик функції відправлення повідомлення
tempFlag=1;
delay(1000);            // формування часової затримки 1 с
}
}
else
{
```

```
PORTB&=~0b00000010;     // встановлення біту 1 порта B в лог.0
tempFlag=0;
}

if((data[2]>50) && (active==1))
{
PORTB|=0b00000001;      // встановлення біту 0 порта B в лог.1
if(gasFlag==0)
{
send_sms_gas();         // виклик функції відправлення повідомлення
delay(5000);            // формування часової затримки 5 с
gasFlag=1;              // запис значення 1 в змінну
if(data[2]>80){
ring();                 // виклик функції телефонування абоненту
delay(10000);           // формування часової затримки 10 с
}
}
}
else
{
PORTB&=~0b00000001;     // встановлення біту 0 порта B в лог.0
gasFlag=0;
}
if(((data[1]&0x0F)!=0x0F) && (active==1))
{
PORTB|=0b00001000;      // встановлення біту 3 порта B в лог.1
}else{
PORTB&=~0b00001000;     // встановлення біту 3 порта B в лог.0
}
if(((data[1]&0b00010000)!=0) && (active==1))
{
PORTB|=0x04;            // встановлення біту 2 порта B в лог.1
}else{
PORTB&=~0x04;           // встановлення біту 2 порта B в лог.0
}
if(((PINA&0x01)==0) && (active==0))
{
 PORTA|=0x02;           // встановлення біту 1 порта A в лог.1
 active=1;              // запис значення 1 в змінну
 }
else if(((PINA&0x01)==0) && (active==1))
{
 PORTA&=~0x02;          // скидання другого біту в порту A
 active=0;              // запис значення 0 в змінну active
 gasFlag=0;             // запис значення 0 в змінну gasFlag
 tempFlag=0;            // запис значення 0 в змінну tempFlag
 switchFlag=0;          // запис значення 0 в змінну switchFlag
}
}
}
```

7.3. Мікропроцесорна система керування роботизованою платформою з чотирма роликонесучими колесами

Специфіка операцій, умови експлуатації та функціональне призначення мобільного робота визначають конструктивні особливості робота, ступінь складності системи управління, масогабаритні характеристики і склад спеціального.

До мобільному роботу пред'являються наступні загальні вимоги:

– робот повинен мати високі рухливість і прохідність в міських умовах, всередині будівель і споруд, в зонах руйнувань, на пересіченій місцевості, як на твердих гладких покриттях, так і на ґрунтових, що деформуються;

– робот повинен надійно діяти як в непідготовлених природних умовах, так і в середовищі, спеціально пристосованому для проживання людини (всередині будинків, в транспортних комунікаціях), вписуватися в міські транспортні потоки або рухатися в складі транспортних колон;

– конструкція робота повинна забезпечувати його високу мобільність і швидке розгортання при виконанні спецоперацій.

Колесо є найпопулярнішим механізмом переміщення в мобільній робототехніці і в транспортних засобах в цілому. Колесо дозволяє досягати високої ефективності в багатьох додатках при відносно простій механічній реалізації.

Існує чотири основних типи коліс:

1. Стандартне колесо (рис. 7.7, а): два ступені свободи. Обертається навколо колісної осі (моторизованої) і точкою дотику з поверхнею.

2. Поворотний колесо (рис. 7.7, б): два ступені свободи. Обертається навколо зміщеного колісного шарніра.

3. Шведське колесо (рис. 7.7, в, г): три ступені свободи. Обертається навколо (моторизованої) колісної вісі, навколо осей роликів і навколо точки дотику. Ролики можуть бути розміщені двома різними способами – 90° варіант (рис. 7.7, в) і 45° варіант (рис. 7.7, г).

4. Шарове або сферичне колесо (рис. 7.7, д): технічно складно піддається реалізації.

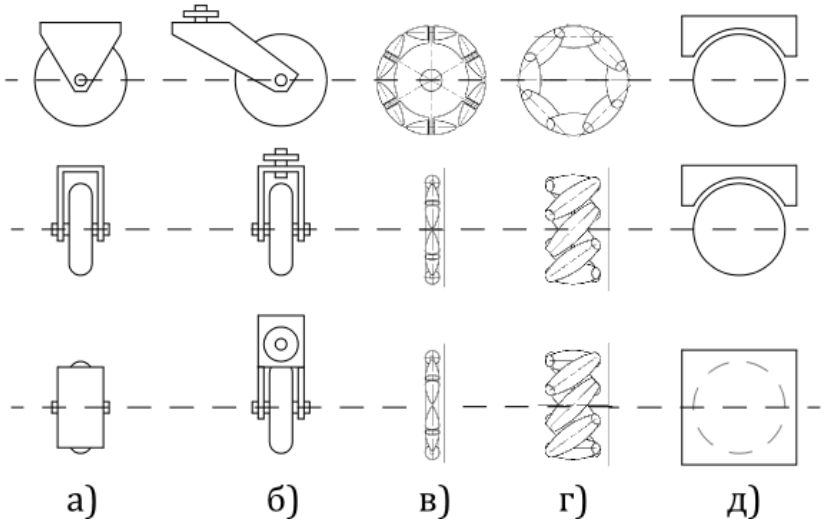


Рис. 7.7.

Всі ці типи коліс сильно відрізняються своєю кінематикою, і тому той чи інший тип значно впливає на всю кінематику мобільного робота.

Звичайне колесо і поворотне колесо мають основну вісь обертання і, таким чином, є строго спрямованими. Для руху в іншому напрямку колесо повинно бути спочатку розгорнуто вздовж вертикальної осі. Основна відмінність між цими двома типами коліс полягає в тому, що звичайне колесо може виконати цей поворот без побічного ефекту, тому що центр обертання знаходиться в точці контакту з поверхнею, в той час, як поворотне колесо обертається навколо зміщеної осі, в результаті чого, вихідне зусилля буде рухати шасі під час руління.

Збільшення цін на промислові площі сприяє підвищенню щільності товарів в складському приміщенні, в зв'язку з цим потрібні нові високоманеврені вантажно-транспортні пристрої. Раніше для цих цілей успішно застосовувалися машини з електричним приводом і кінематичною схемою, що включає в себе чотири стандартних колеса. Їхня основна перевага – простота конструкції і відсутність необхідності складного інтелектуального управління. Основний їх недолік – низька маневреність.

На рис. 7.8 представлені різні варіанти конструкцій роликонесучих коліс.



Рис. 7.8.

Класична чотирьохколісна кінематична схема лежить в основі практично всіх сучасних транспортних роботів на будь-якому підприємстві. Її основними перевагами є:

– простота і надійність конструкції;

– проста система управління;

– висока здатність навантаження;
– гарне зчеплення з поверхнею;
– висока стійкість, статична стійкість конструкції.
Її основними недоліками є:
– відносно велика величина зовнішнього радіуса розвороту.

Шведське колесо (Swedish wheel) або, як його ще називають, колесо Ілона, винайшов інженер з шведської компанії Mecanum AB Бенгт Ілон в 1973 році – звідси і варіанти назв, відповідно. Конструкція таких коліс дозволяє обертатися на місці при мінімальній силі тертя і низькому обертальному моменті.

На рис. 7.9 представлена мобільна роботизована платформа з чотирма роликонесучими колесами.

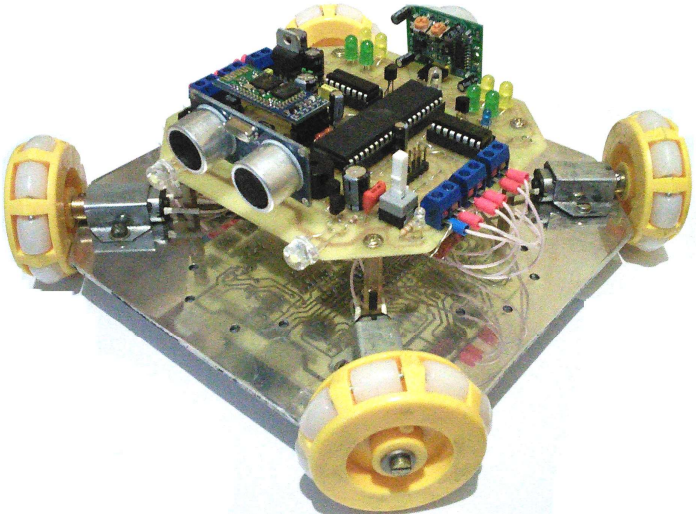


Рис. 7.9.

На рис.7.10 представлені можливі напрямки руху платформи при різному напрямку обертання коліс.

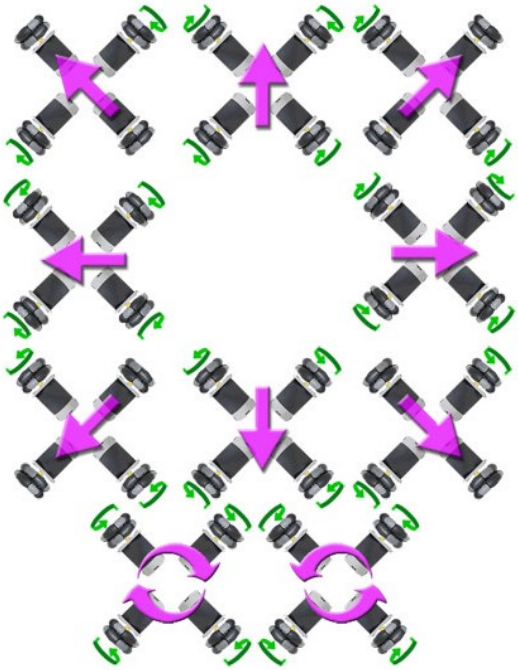


Рис. 7.10.

На рис. 7.11 представлена принципова схема мікропроцесорної системи керування мобільною роботизованою платформою.

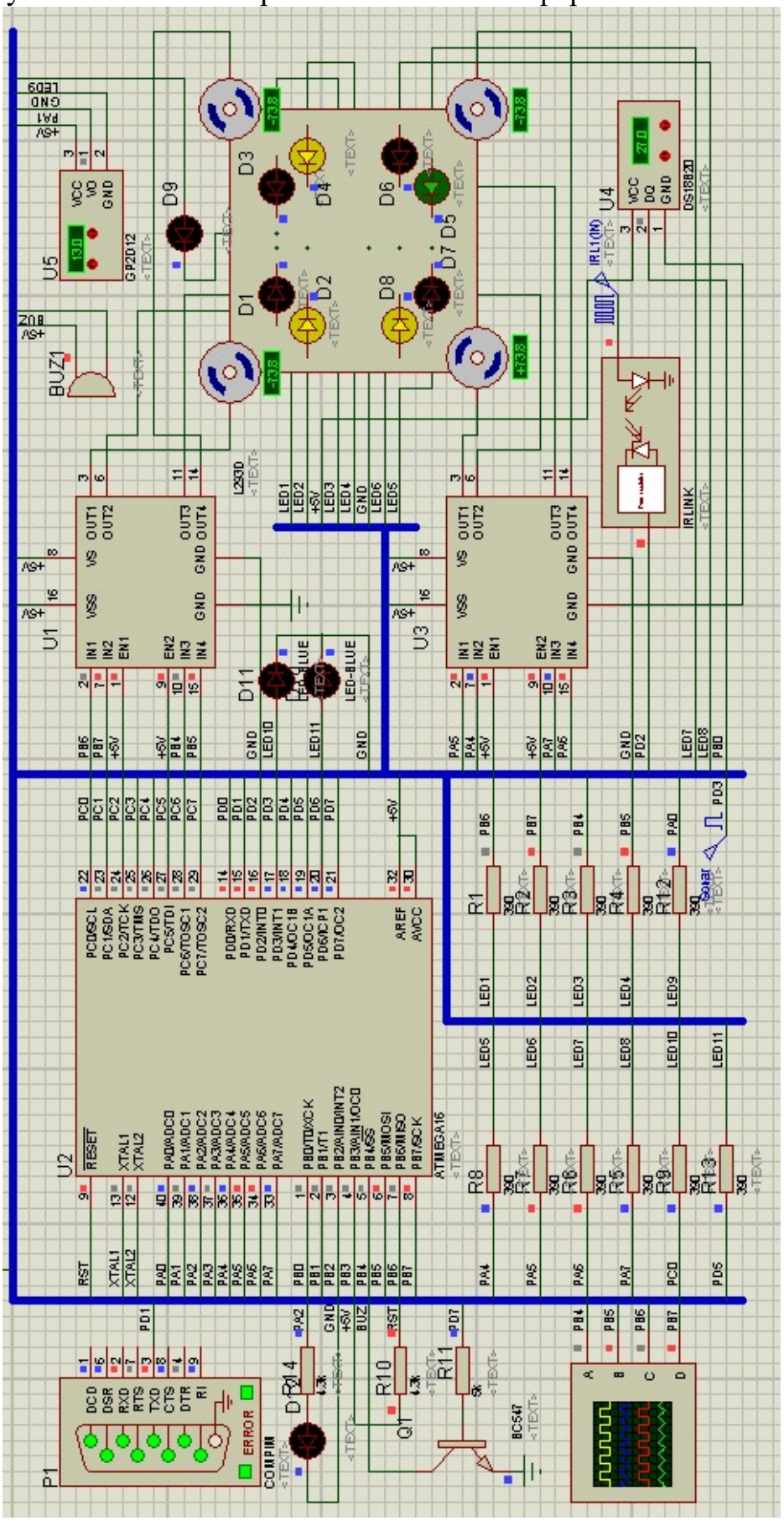


Рис. 7.11.

Приклад програми управління роботизованою платформою представлений нижче. У програмі реалізовані функції управління рухом робота в різні боки і демонструється їх використання.

```
// підключення бібліотек
#include "iom16v.h"
#include "macros.h"

// визначення ідентифікатора та послідовності символів, якою буде заміщено
// ідентифікатор при компіляції програми
// ідентифікатор для керування двигуном M2
#define M2_STOP() PORTB&=~0b11000000
#define M2_F() PORTB|=0b01000000
#define M2_R() PORTB|=0b10000000
// ідентифікатор для керування двигуном M1
#define M1_STOP() PORTB&=~0b00110000
#define M1_F() PORTB|=0b00100000
#define M1_R() PORTB|=0b00010000
// ідентифікатор для керування двигуном M3
#define M3_STOP() PORTA&=~0b11000000
#define M3_F() PORTA|=0b10000000
#define M3_R() PORTA|=0b01000000
// ідентифікатор для керування двигуном M4
#define M4_STOP() PORTA&=~0b00110000
#define M4_F() PORTA|=0b00100000
#define M4_R() PORTA|=0b00010000
// ідентифікатор для керування звуковим сповіщувачем
#define BUZZER_ON() PORTD|=0x80
#define BUZZER_OFF() PORTD&=~0x80
// ідентифікатор для керування кольоровим світлодіодом
#define COLOR_ON() PORTA|=0b00000001
#define COLOR_OFF() PORTA&=~0b00000001
// ідентифікатор для керування освітленням
#define LED_ON() PORTA|=0b00000100
#define LED_OFF() PORTA&=~0b00000100
#define CHANGE_LED() PORTA^=0b00000100
#define DEFAULT 0
#define FIRE 1
// оголошення додаткових змінних
char blink_status;
char blink_mode;
// реалізація функції вмикання світла
void LightON()
{
PORTC|=0x01;
PORTD|=0b00100000;
}
// реалізація функції вимикання світла
void LightOFF()
{
PORTC&=~0x01;
PORTD&=~0b00100000;
```

```
}
// реалізація функції руху мобільної платформи вперед
void MoveN()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_F();        // увімкнути двигун M3 (обертання вперед)
 M4_F();        // увімкнути двигун M4 (обертання вперед)
}
// реалізація функції руху мобільної платформи назад
void MoveS()
{
 M1_R();        // увімкнути двигун M1 (обертання назад)
 M2_R();        // увімкнути двигун M2 (обертання назад)
 M3_R();        // увімкнути двигун M3 (обертання назад)
 M4_R();        // увімкнути двигун M4 (обертання назад)
}
// реалізація функції руху мобільної платформи ліворуч
void MoveW()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M2_R();        // увімкнути двигун M2 (обертання назад)
 M3_R();        // увімкнути двигун M3 (обертання назад)
 M4_F();        // увімкнути двигун M4 (обертання вперед)
}
// реалізація функції руху мобільної платформи праворуч
void MoveE()
{
 M1_R();        // увімкнути двигун M1 (обертання назад)
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_F();        // увімкнути двигун M3 (обертання вперед)
 M4_R();        // увімкнути двигун M4 (обертання назад)
}
// реалізація функції руху мобільної платформи вперед праворуч
void MoveNW()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M4_F();        // увімкнути двигун M4 (обертання вперед)
}
// реалізація функції руху мобільної платформи вперед ліворуч
void MoveNE()
{
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_F();        // увімкнути двигун M3 (обертання вперед)
}
// реалізація функції руху мобільної платформи назад праворуч
void MoveSW()
{
 M2_R();        // увімкнути двигун M2 (обертання назад)
 M3_R();        // увімкнути двигун M3 (обертання назад)
}
```

```
// реалізація функції руху мобільної платформи назад ліворуч
void MoveSE()
{
 M1_R();        // увімкнути двигун M1 (обертання назад)
 M4_R();        // увімкнути двигун M4 (обертання назад)
}
// реалізація функції обертання мобільної платформи за часовою стрілкою
void RotateCW()
{
 M1_R();        // увімкнути двигун M1 (обертання назад)
 M2_R();        // увімкнути двигун M2 (обертання назад)
 M3_F();        // увімкнути двигун M3 (обертання вперед)
 M4_F();        // увімкнути двигун M4 (обертання вперед)
}
// реалізація функції обертання мобільної платформи проти часової стрілки
void RotateCCW()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_R();        // увімкнути двигун M3 (обертання назад)
 M4_R();        // увімкнути двигун M4 (обертання назад)
}
// реалізація функції зупинки платформи
void Stop()
{
 PORTB&=~0b11110000;    // встановлення рівня лог. 0 на 4 старших розрядах порта B
 PORTA&=~0b11110000;    // встановлення рівня лог. 0 на 4 старших розрядах порта A
}
// реалізація функції часової затримки
void delay(int ms)
{
 int c1, c2;
 for(c1=0;c1<ms;c1++)
 {
  for(c2=0;c2<970;c2++) NOP();      // в циклі виконується пуста операція
 }
}
// реалізація функції налаштування портів мікроконтролера
void init_cpu()
{
 DDRA=0b11110101;           // налаштування виводів порта A на вхід (0) та вихід (1)
 DDRB=0b11110000;           // налаштування виводів порта B на вхід (0) та вихід (1)
 DDRC=0b00000001;           // налаштування виводів порта C на вхід (0) та вихід (1)
 DDRD=0b11110000;           // налаштування виводів порта D на вхід (0) та вихід (1)

 PORTA=0;                   // встановлення на всіх виводах порта A низького рівня
 PORTB=0;                   // встановлення на всіх виводах порта B низького рівня
 PORTC=0;                   // встановлення на всіх виводах порта C низького рівня
 PORTD=0;                   // встановлення на всіх виводах порта D низького рівня

 blink_status=0;            // запис значення 0 в змінну blink_status
 blink_mode=0;              // запис значення 0 в змінну blink_mode
```

```
}
// реалізація функції блимання світлодіодами
void Blink()
{
if(blink_mode==0)
{
 blink_mode=1;
 LeftLED_ON();       // увімкнути світлодіоди з лівого боку
 delay(50);          // часова затримка 50 мс
 LeftLED_OFF();      // вимкнути світлодіоди з лівого боку
 delay(50);          // часова затримка 50 мс
 LeftLED_ON();       // увімкнути світлодіоди з лівого боку
 delay(50);          // часова затримка 50 мс
 LeftLED_OFF();      // вимкнути світлодіоди з лівого боку
}
else if(blink_mode==1)
{
 blink_mode=0;
 RightLED_ON();      // увімкнути світлодіоди з правого боку
 delay(50);          // часова затримка 50 мс
 RightLED_OFF();     // вимкнути світлодіоди з правого боку
 delay(50);          // часова затримка 50 мс
 RightLED_ON();      // увімкнути світлодіоди з правого боку
 delay(50);          // часова затримка 50 мс
 RightLED_OFF();     // вимкнути світлодіоди з правого боку
}
}
// реалізація функції вмикання світлодіода з лівого боку
void  LeftLED_ON()
{
PORTC|=0x01;         // встановлення рівня лог. 1 на виводі PC0
}
// реалізація функції вимикання світлодіода з лівого боку
void  LeftLED_OFF()
{
PORTC&=~0x01;       // встановлення рівня лог. 0 на виводі PC0
}
// реалізація функції вмикання світлодіода з правого боку
void  RightLED_ON()
{
 PORTD|=0b00100000; // встановлення рівня лог. 1 на виводі PD5
}
// реалізація функції вимикання світлодіода з правого боку
void  RightLED_OFF()
{
 PORTD&=~0b00100000; // встановлення рівня лог. 0 на виводі PD5
}
// головна функція програми
void main()
{
 init_cpu();     // виклик функції ініціалізації мікроконтролера та налаштування портів
 while(1)        // головний цикл програми
```

```
{
  MoveN();          // виклик функції руху вперед
  delay(3000);      // формування часової затримки 3 секунди
  Stop();           // виклик функції зупинки мобільної платформи

  MoveE();          // виклик функції руху праворуч
  delay(3000);      // формування часової затримки 3 секунди
  Stop();           // виклик функції зупинки мобільної платформи

  MoveS();          // виклик функції руху назад
  delay(3000);      // формування часової затримки 3 секунди
  Stop();           // виклик функції зупинки мобільної платформи

  MoveW();          // виклик функції руху ліворуч
  delay(3000);      // формування часової затримки 3 секунди
  Stop();           // виклик функції зупинки мобільної платформи

  RotateCW();       // виклик функції обертання платформи за часовою стрілкою
  delay(3000);      // формування часової затримки 3 секунди
  Stop();           // виклик функції зупинки мобільної платформи

  RotateCCW();      // виклик функції обертання платформи проти часової стрілки
  delay(3000);      // формування часової затримки 3 секунди
  Stop();           // виклик функції зупинки мобільної платформи

  LightON();        // виклик функції вмикання світлодіодів
  delay(200);       // формування часової затримки 200 мс
  LightOFF();       // виклик функції вимикання світлодіодів
  delay(200);       // формування часової затримки 200 мс

  LightON();        // виклик функції вмикання світлодіодів
  delay(200);       // формування часової затримки 200 мс
  LightOFF();       // виклик функції вимикання світлодіодів
  delay(200);       // формування часової затримки 200 мс

  LightON();        // виклик функції вмикання світлодіодів
  delay(200);       // формування часової затримки 200 мс
  LightOFF();       // виклик функції вимикання світлодіодів
  delay(200);       // формування часової затримки 200 мс
  BUZZER_ON();      // виклик функції формування звукового сигналу
  delay(200);       // формування часової затримки 200 мс
  BUZZER_OFF();     // виклик функції вимкнення звукового сигналу
  delay(100);       // формування часової затримки 100 мс
  BUZZER_ON();      // виклик функції формування звукового сигналу
  delay(200);       // формування часової затримки 200 мс
  BUZZER_OFF();     // виклик функції вимкнення звукового сигналу
  delay(2000);      // формування часової затримки 2 секунди
}
```

З більш ніж трьома колесами, як правило, потрібна система підвіски для підтримки постійного контакту колеса з поверхнею. Один з найпростіших підходів в реалізації підвіски є створення гнучкості

(еластичності) самого колеса. Наприклад, в разі деяких чотириколісних роботів для приміщень, які використовують поворотні колеса, виробники використовують в колесі шини з м'якої гуми, що деформуються, щоб створити спрощену підвіску. Звичайно, це обмежене рішення не може конкурувати зі складною системою підвіски в тих випадках, коли робот потребує більшої позитивної динаміки підвісці для сильно пересіченої місцевості.

7.4. Мікропроцесорна система керування рухом роботизованої мобільної платформи по заданій траєкторії

Роботизована платформа призначена для автономної роботи у закритому приміщенні та повинна виконувати функції руху по заданій траєкторії з метою переміщення невеликих вантажів у задану точку. Для виконання цих функції на роботизованій платформі повинні бути встановлені датчики, що забезпечують рух по заданій траєкторії та датчики для запобігання зіткнення платформи з перешкодами.

Керування роботом можливе з мобільного телефону або іншого пристрою по інтерфейсу Bluetooth.

Виходячи з зазначеного можна сформулювати наступні вимоги до системи керування:

– керування трьома двигунами постійного струму з можливістю їх реверсування для забезпечення руху платформи в будь-якому напрямку в умовах обмеженого простору;

– прийом та передача даних по інтерфейсу Bluetooth по захищеному каналу передачі;

– контроль руху по заданій траєкторії;

– контроль відстані до перешкоди з використанням ультразвукового датчику HC-05 та виконання маневрів для обходу перешкоди;

– світлова індикація режимів роботи двигунів за допомогою напівпровідникових світлодіодів;

– наявність засобів для освітлення навколишнього простору з використанням ультраяскравих світлодіодів;

– звукове сповіщення при виявленні пожежі або натраплянні на перешкоду.

На рис. 7.12 представлено структурну схему системи керування роботизованою платформою. Використання мікроконтролера ATMega16 дозволило реалізувати логіку керування роботом та регулятор на програмному рівні, що дозволило зменшити масо-габаритні показники системи керування та її енергоспоживання, що дуже важливо при автономній роботі платформи. Для керування чотирма двигунами постійного струму зручно використовувати спеціалізований інтегральний драйвер L293D, який має в собі елементи захисту двигуна від перевантаження, короткого замикання, запобігає протіканню прямих струмів через силові транзистори схеми, має вбудовані зворотні діоди та захист від перегріву. Для індикації режиму роботи кожного з двигунів

використовуються напівпровідникові діоди жовтого та зеленого кольорів. Для освітлення навколишнього середовища використовуються ультра яскраві світлодіоди білого кольору.

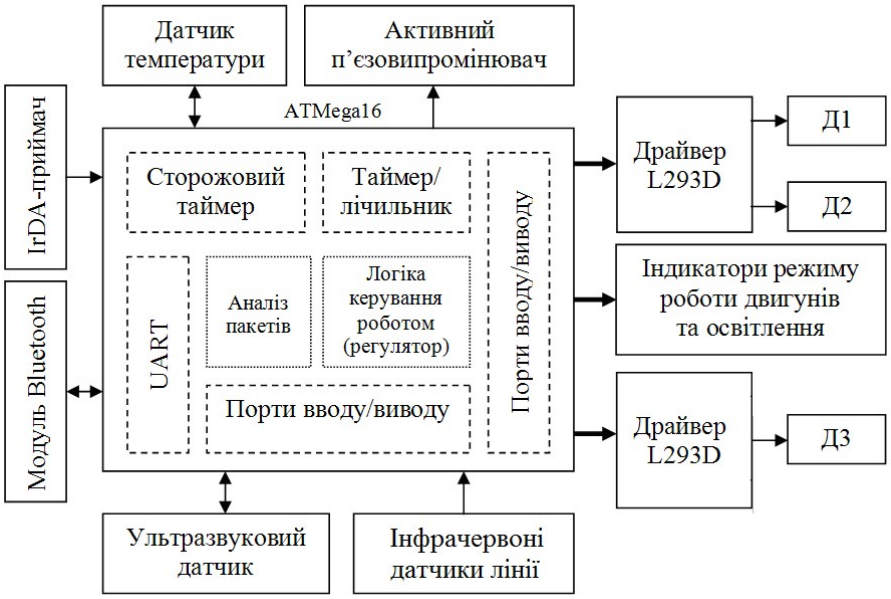


Рис. 7.12.

Керування драйверами двигунів постійного струму виконується за допомогою портів введення/виведення мікроконтролера. Для сполучення мікросхеми-драйвера з мікроконтролером не потрібно використовувати додаткових елементів, оскільки входи драйвера толерантні до напруги рівня ТТЛ (0-5В).

Для визначення роботом відстані до перешкоди використовується ультразвуковий датчик, який запускається шляхом подачі на його вхід імпульсу тривалістю 10 мкс, який формує мікроконтролер. Тривалість імпульсу, що формує ультразвуковий датчик у відповідь, пропорційна відстані до перешкоди. За допомогою таймера/лічильника, вбудованого в мікроконтролер, відбувається вимірювання тривалості імпульсу, після чого отримане значення використовується для розрахунку відстані.

Інфрачервоні датчики використовується для визначення наявності лінії в зоні спостереження. При спрацюванні датчика на його виході формується сигнал логічної одиниці, який надходить до порту мікроконтролера. Чутливість інфрачервоного датчика налаштовується елементами керування, що розташовані на ньому.

Для звукового сповіщення про наявність перешкоди, або при спрацюванні інфрачервоного датчику, використовується активний п'єзовипромінювач, який формує різні звукові сигнали.

Керування роботом відбувається по радіоканалу (протокол Bluetooth), для чого використовується спеціалізований трансивер HC-05 та мобільний телефон або планшет. Також керування роботом можливе по інфрачервоному каналу з використанням протоколу RC5 за допомогою пульта дистанційного керування.

На рис. 7.13 представлено принципову схему системи керування роботизованою мобільною платформою.

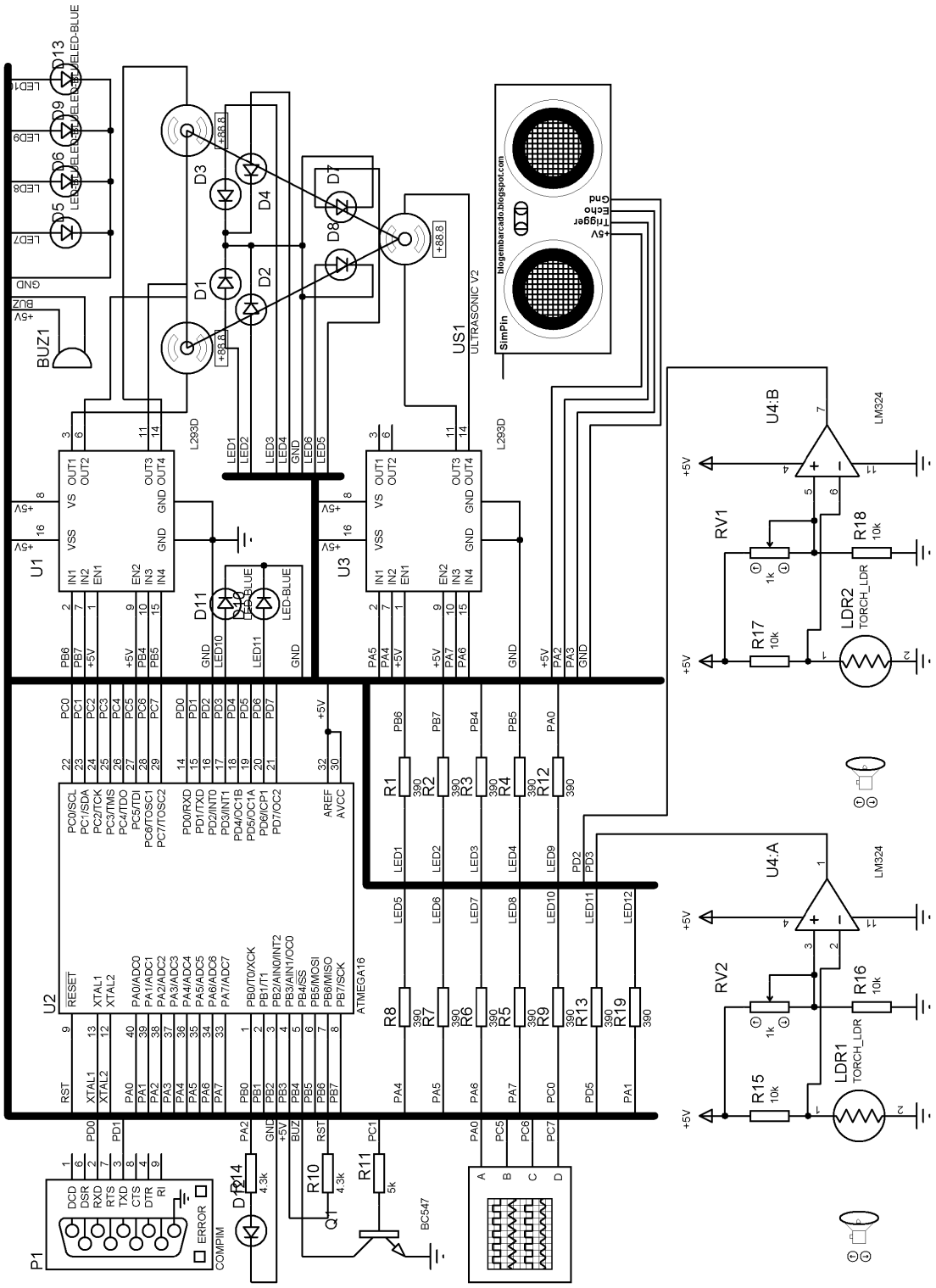


Рис. 7.13.

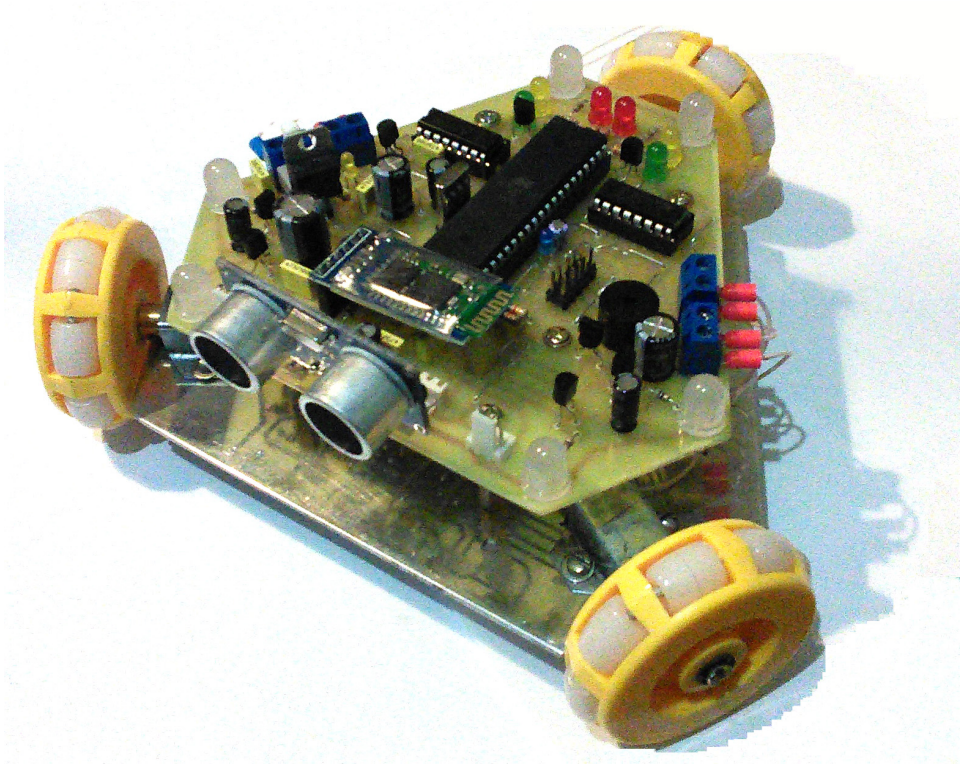


Рис. 7.14.

Методів, алгоритмів і схем рішення задачі руху по простій лінії існує безліч. Вони розраховані на самі різні системи використовуваних датчиків смуги – від відеокамери до найпростішого граничного датчика освітленості. Розглянемо лише деякі з них, зокрема ті, які використовують розташування пари датчиків освітленості по краях чорної лінії. Характеристики різних регуляторів для управління рухом робота по лінії представлені на рис. 7.15.

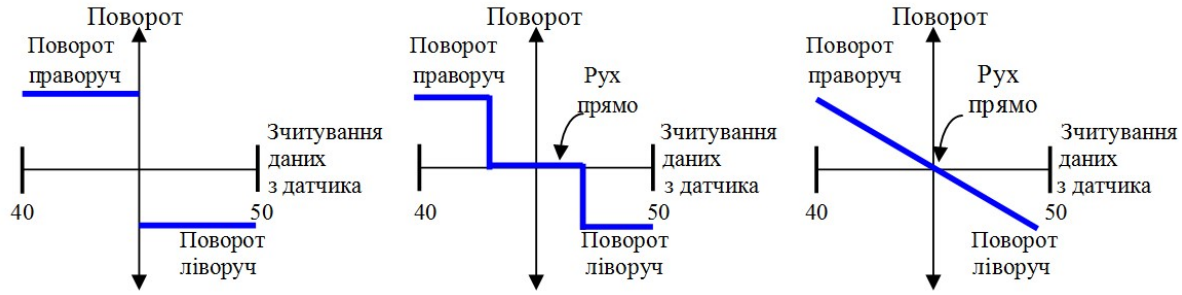


Рис. 7.15.

Можливість отримання фактичних сенсорних даних від реально діючого рухомого об'єкту має важливе значення для розуміння особливостей сприйняття технічними пристроями зовнішнього світу і що виникають при цьому проблем зашумленості, надмірності і невизначеності інформації, необхідної для автономного функціонування складних систем.

Приклад програми керування рухом робота по лінії представлено нижче.

```
// підключення необхідних бібліотек, функції з яких використовуються в програмі
#include "iom16v.h"
#include "macros.h"
// визначення ідентифікатора та послідовності символів, якою буде заміщено
// ідентифікатор при компіляції програми
// ідентифікатор для керування двигуном M1
#define M1_STOP() PORTA&=~0b00110000
#define M1_F() PORTA|=0b00010000
#define M1_R() PORTA|=0b00100000
// ідентифікатор для керування двигуном M2
#define M2_STOP() PORTB&=~0b11000000
#define M2_F() PORTB|=0b10000000
#define M2_R() PORTB|=0b01000000
// ідентифікатор для керування двигуном M3
#define M3_STOP() PORTA&=~0b11000000
#define M3_F() PORTA|=0b01000000
#define M3_R() PORTA|=0b10000000
// ідентифікатор для керування звуковим сповіщувачем
#define BUZZER_ON() PORTC|=0x02
#define BUZZER_OFF() PORTC&=~0x02
// ідентифікатор для світлодіода 1
#define L1_ON() PORTC|=0b00000001
#define L1_OFF() PORTC&=~0b00000001
// ідентифікатор для світлодіода 2
#define L2_ON() PORTD|=0b10000000
#define L2_OFF() PORTD&=~0b10000000
// ідентифікатор для світлодіода 3
#define L3_ON() PORTD|=0b00100000
#define L3_OFF() PORTD&=~0b00100000
// ідентифікатор для світлодіода 4
#define L4_ON() PORTD|=0b00010000
#define L4_OFF() PORTD&=~0b00010000
// ідентифікатор для світлодіода 5
#define L5_ON() PORTB|=0b00000010
#define L5_OFF() PORTB&=~0b00000010
// ідентифікатор для світлодіода 6
#define L6_ON() PORTA|=0b00000010
#define L6_OFF() PORTA&=~0b00000010
// ідентифікатор для кольорового світлодіода
#define LED_ON() PORTC&=~0b01000000
#define LED_OFF() PORTC|=0b01000000
#define CHANGE_LED() PORTC^=0b01000000
// реалізація функції руху вперед
void MoveN()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_STOP();
}
// реалізація функції руху назад
void MoveS()
{
```

```
 M1_R();        // увімкнути двигун M1 (обертання назад)
 M2_R();        // увімкнути двигун M2 (обертання назад)
 M3_STOP();
}
// реалізація функції руху ліворуч
void MoveW()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_R();        // увімкнути двигун M3 (обертання назад)
}
// реалізація функції руху праворуч
void MoveE()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_F();        // увімкнути двигун M3 (обертання вперед)
}
// реалізація функції руху вперед ліворуч
void MoveNW()
{
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_F();        // увімкнути двигун M3 (обертання вперед)
}
// реалізація функції руху вперед праворуч
void MoveNE()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M3_R();        // увімкнути двигун M3 (обертання назад)
}
// реалізація функції руху назад ліворуч
void MoveSW()
{
 M1_R();        // увімкнути двигун M1 (обертання назад)
 M3_F();        // увімкнути двигун M3 (обертання вперед)
}
// реалізація функції руху назад праворуч
void MoveSE()
{
 M2_R();        // увімкнути двигун M2 (обертання назад)
 M3_R();        // увімкнути двигун M3 (обертання назад)
}
// реалізація функції обертання за часовою стрілкою
void RotateCW()
{
 M1_F();        // увімкнути двигун M1 (обертання вперед)
 M2_R();        // увімкнути двигун M2 (обертання назад)
 M3_F();        // увімкнути двигун M3 (обертання вперед)
}
// реалізація функції обертання проти часової стрілки
void RotateCCW()
{
```

```
 M1_R();        // увімкнути двигун M1 (обертання назад)
 M2_F();        // увімкнути двигун M2 (обертання вперед)
 M3_R();        // увімкнути двигун M3 (обертання назад)
}
// реалізація функції зупинки всіх двигунів і, відповідно, мобільної платформи
void Stop()
{
 PORTB&=~0b11000000;    // встановлення низького рівня на виводах PB7 та PB6
 PORTA&=~0b11110000;    // встановлення низького рівня на виводах PB7-PB4
}
// реалізація функції часової затримки
void delay(int ms)
{
 int c1, c2;
 for(c1=0;c1<ms;c1++)
 {
  for(c2=0;c2<970;c2++) NOP();      // в циклі виконується пуста операція
 }
}
// реалізація функції часової затримки
void pause(int ms)
{
 int c1, c2;
 for(c1=0;c1<ms;c1++)
 {
  for(c2=0;c2<50;c2++) NOP();       // в циклі виконується пуста операція
 }
}
// реалізація функції ініціалізації мікроконтролера
void init_cpu()
{
 DDRA=0b11110010;           // налаштування виводів порта A на вхід (0) та вихід (1)
 DDRB=0b11010010;           // налаштування виводів порта B на вхід (0) та вихід (1)
 DDRC=0b01000011;           // налаштування виводів порта C на вхід (0) та вихід (1)
 DDRD=0b11110000;           // налаштування виводів порта D на вхід (0) та вихід (1)
 PORTA=0;                   // встановлення на всіх виводах порта A низького рівня
 PORTB=0;                   // встановлення на всіх виводах порта B низького рівня
 PORTC=0;                   // встановлення на всіх виводах порта C низького рівня
 PORTD=0;                   // встановлення на всіх виводах порта D низького рівня
 LED_OFF();                 // вимкнення світлодіода
}
// головна функція програми
void main()
{
char line_mode=1;           // оголошення та ініціалізація додаткових змінних
char m3_run=0;
char motor_run=1;
init_cpu();                 // виклик функції ініціалізації мікроконтролера
while(1)                    // головний цикл
{
if(line_mode==1)
 {
```

```
// аналіз стану датчиків
if(((PIND&0b00000100)!=0) && ((PIND&0b00001000)==0))
{
  if(m3_run==2)
  {
   M3_STOP();                    // зупинка мотора M3
   delay(50);                    // часова затримка 50 мс
  }
  if(motor_run==1) M2_STOP();    // зупинка мотора M2
  M3_F();
  m3_run=1;
}
// аналіз стану датчиків
 else if(((PIND&0b00001000)!=0)&& ((PIND&0b00000100)==0))
{
  if(m3_run==1)
  {
   M3_STOP();                    // зупинка мотора M3
   delay(50);                    // часова затримка 50 мс
  }
  if(motor_run==1) M1_STOP();
  M3_R();                       // обертання мотора M3 назад
  m3_run=2;
}
 else if((((PIND&0b00000100)!=0) && ((PIND&0b00001000)!=0)) ||
(((PIND&0b00000100)==0) && ((PIND&0b00001000)==0)))
 {
  M3_STOP();                    // зупинка мотора M3
  if(motor_run==1) MoveN();     // виклик функції руху платформи вперед
  m3_run=0;
 }
 }
}
}
```

Розглянута вище система з двох датчиків, розташованих по зовнішніх краях лінії, є лише одним з можливих варіантів. Цей варіант простий, однак у нього є свої недоліки і обмеження. Справа в тому, що все це добре працює лише при малих швидкостях, коли робот відхиляється від лінії настільки, що система управління може встигнути зреагувати на це відхилення, коли основний режим роботи датчиків – це відстеження кордону лінії. Якщо ж потрібно максимально швидкісне проходження, то зазвичай використовується лінійка з декількох датчиків.

Іноді використовують установку датчиків не по зовнішніх, а по внутрішніх краях смуги. Таке розташування також має свої плюси і мінуси. В цьому випадку вкрай небезпечним стає перерегулювання, що приводить до виходу обох датчиків за межі лінії. Вузька зона регулювання дещо обмежує розмах поперечних коливань робота під час руху і зовні поведінка робота виглядає більш плавною. Проте, небезпечним є відсутність деякого зазору між датчиками і межею, що відслідковується.

7.5. Мікропроцесорна система моніторингу параметрів дизель-генераторного агрегату

Тривалість і безпека експлуатації дизель-генераторів на суднах та в промислових автономних електроенергетичних системах в значній мірі залежить від своєчасного й повного контролю параметрів двигунів. Перспективи розвитку систем діагностування дизельних агрегатів пов'язані з розробкою нових методів і засобів раннього попередження несправностей для того, щоб виключити аварійну зупинку двигуна. Необхідність забезпечення високої надійності експлуатації генеруючих агрегатів поставили завдання розробки систем діагностування стану двигуна. Такі системи розрізняються видом технічного діагностування – тестовим і функціональним. При тестовому діагностуванні на досліджуваний об'єкт (дизель) подаються спеціальні тестові впливи, склад і послідовність яких вибирається у відповідності з методикою діагностування. Функціональне технічне діагностування здійснюється в процесі безпосереднього використання дизеля за призначенням, коли на неї надходять тільки робочі впливу. Лабораторний стенд дозволяє змоделювати процес тестового діагностування і за результатами тесту зробити висновки про працездатність програмного забезпечення системи моніторингу параметрів.

У загальному випадку завдання моніторингу параметрів зводяться до дослідження дизеля як об'єкта діагностування, розробці відповідних методів і засобів, вирішення питань побудови та організації системи моніторингу. Вирішення цих завдань включає в себе:

– Аналіз процесів, що протікають в двигуні в режимі експлуатації;

– Вибір необхідного числа діагностичних параметрів і розробку методів їх визначення;

– Розробку методів оцінки та прогнозування динаміки зміни параметрів стану;

– Розробку системи збору, зберігання та аналізу діагностичної інформації;

– Розробку системи технічного діагностування;

– Визначення місця технічної діагностики в системі експлуатації з метою запобігання аварійних відмов, повного використання ресурсних можливостей дизельного агрегату, скорочення витрат на технічне обслуговування і т.п.

Стратегія створення таких систем моніторингу полягає в об'єднанні вимірювальних датчиків, апаратури збору даних, їх обробки, а також засобів програмного забезпечення в єдину інформаційно-вимірювальну систему, засновану на сучасних комп'ютерних технологіях.

В табл. 7.1 представлені основні параметри дизельного агрегату, які необхідно контролювати.

Таблиця 7.1.

№ п/п	Параметр	Діапазон значень	Тип сигналу
1	Тиск змащувального масла на вході в двигун	0 - 0,5 кгс/см^2	аналоговий 4-20 мА, 0-10В
2	Температура змащувального масла на вході в двигун	30 - 125 ºС	аналоговий 4-20 мА, 0-10В
3	Температура охолоджувальної рідини на вході системи	0 - 50 ºС	аналоговий 4-20 мА, 0-10В
4	Температура охолоджувальної рідини на виході системи	0 - 50 ºС	аналоговий 4-20 мА, 0-10В
5	Температура вихлопних газів	0 - 800 ºС	аналоговий 4-20 мА, 0-10В
6	Рівень палива	0 – 100 %	аналоговий 4-20 мА, 0-10В
7	Частота обертання валу двигуна	0 – 2000 об/хв	цифровий
8	Вміст кисню у вихлопних газах	0,8 – 1,3	аналоговий 4-20 мА, 0-10В
9	Тиск повітря у компресорі	0 - 2 кгс/см^2	аналоговий 4-20 мА, 0-10В
10	Рівень палива у паливному баку	0-100%	аналоговий 4-20 мА, 0-10В

Частотний діапазон сигналів, кількість вимірювальних каналів і динамічний діапазон сигналів на вході визначають технічні вимоги до системи:

– Кількість вхідних каналів не менше 6 (у розглянутій системі 8);

– Частотний діапазон вхідних сигналів: 10-1000 Гц;

– Розрядність АЦП / ЦАП: 10-16 біт;

– Час перетворення АЦП: 100-1000 мкс;

– Порт введення/виведення: 8 біт ТТЛ;

– Динамічний діапазон по входу: 60-80 дБ.

Для вимірювання вказаних параметрів використовуються різні типи датчиків. Датчик температури – це пристрій, що безпосередньо приймає і перетворює вимірювану фізичну величину в сигнал напруги або струму для наступної передачі його на вимірюючий прилад. До датчиків температури відносять термоелектричні перетворювачі, термоперетворювачі опору, термопари.

Критерії вибору датчика температури. Необхідно враховувати наступні фактори:

– вимірюваний діапазон температури й припустимі відхилення по точності вимірів;

– місце розташування датчика всередині вимірюваного середовища або об'єкта;

– умови експлуатації (нормальні, підвищена вологість, високоокислювальна атмосфера, пожеженебезпечні, сейсмонебезпечні й т.п.);

– час стабільності й можливість демонтажу датчика для періодичної перевірки;

– можливість індивідуальної градуювання або взаємозамінності датчиків;

– вигляд отриманого обмірюваного сигналу на виході – градуси, опір, напруга, струм, можливість самостійного перерахунку в температуру.

Як було зазначено вище, всі промислові датчики мають уніфікований вихід, на якому формується сигнал напруги (0-5В, 0-10В) або струму (4-20 мА). Сигнал струму перетворюється за допомогою спеціальних мікросхем в сигнал напруги 0-5В для того, щоб його можна було вимірювати за допомогою АЦП, вбудованого в мікроконтролер. Незалежно від типу параметра, що вимірюється (тиск, температура, рівень рідини), на виході відповідного датчика формується сигнал напруги або струму. Перетворення до фізичних величин після вимірювання сигналу, отримане значення помножується на деякий коефіцієнт, який для кожного з датчиків унікальний та може бути визначений по діаграмі, яка для будь-якого промислового датчика наводиться в технічній документації. З точки зору програміста при розробці алгоритму не має принципового значення який саме параметр вимірюється. Це впливає лише на вибір масштабуючого коефіцієнту. Тому на стенді встановлені змінні резистори, за допомогою яких при відлагодженні програмних засобів можна задати будь-які значення сигналів, які можуть бути на виході датчиків. Це зручно, оскільки в реальній системі неможливо перевірити всі можливі стани, наприклад ситуацію, коли температура охолоджуючої рідини дизельного агрегату складатиме 90 ^{0}C, або тиск масла дорівнює 0. Якщо це станеться в реальній системі, то велика вірогідність виходу дизельного агрегату з ладу. Саме тому дуже важливим є моделювання та відлагодження програмних засобів до того, як вони будуть використовуватися на реальній системі. Разом з тим, технічно дуже складно в лабораторії створити умови, які є в реальній системі. Так, наприклад, для вимірювання температури вихлопних газів потрібно джерело з температурою кілька сотень градусів, для перевірки датчиків тиску потрібен компресор, що формує тиск декілька атмосфер. Навіть при наявності всього цього, принципово при розробці та відлагодженні програмних засобів нічого не зміниться.

Принципова схема системи зображена на рис. 7.16. Основну функцію по збору та обробці даних виконує мікроконтролер. Він має 8 входів аналого-цифрового перетворювача (АЦП), на входи яких поступають сигнали з різних датчиків. В залежності від типу вихідного сигналу (напруга, струм) використовується зовнішній перетворювач інтерфейсів (на схемі не показаний) для того, щоб виконати гальванічне розв'язання ланцюгів, а також нормування сигналів та приведення їх до рівня 0-5 В.

Дискретні сигнали (S6-S9) поступають з датчиків, що мають лише 2 можливих рівні – лог. 0 та лог. 1. Датчики мають відкриті ("сухі") контакти (виходи реле), тому немає необхідності виконувати гальванічне розв'язання за допомогою оптопар. Мікроконтролер також виконує функції

захисту – при відхиленні параметрів від заданих замикається одне з чотирьох реле. Оскільки струм вивода мікроконтролера не перевищує 20 мА, що недостатньо для спрацювання реле, тому в схемі використані транзистори, що працюють в ключовому режимі.

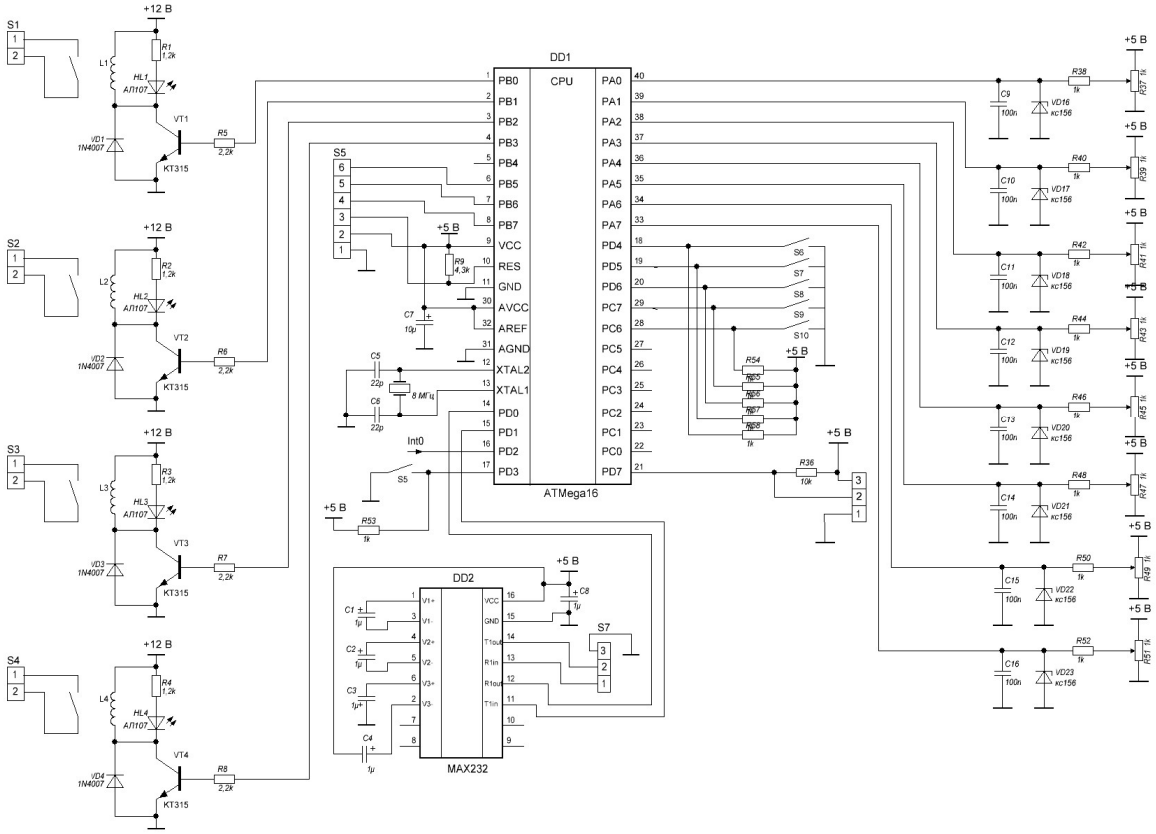


Рис. 7.16.

В системі збору даних про параметри дизельного агрегату реалізований наступний протокол обміну даними:

Передача пакету на запит даних від системи (3 байта):

Байт 1	Байт 2	Байт 3
‘Y’	‘N’	‘G’

Відповідь (20 байт):

Байт 1	Байт 2	Байт 3	Байт 4	Байт 5	Байт 6	Байт 7	Байт 8	Байт 9
‘D’	H	L	H	L	H	L	H	L
заголовок пакету	канал 0		канал 1		канал 2		канал 3	

Байт 10	Байт 11	Байт 12	Байт 13	Байт 14	Байт 15	Байт 16	Байт 17	Байт 18	Байт 19	Байт 20
H	L	H	L	H	L	H	L	C	*freq*	‘!’
канал 4		канал 5		канал 6		канал 7		стан кнопок	частота	

Значення, отримане з кожного каналу 10-розрядного АЦП, що вбудований в мікроконтролер, передається двома байтами – старшим (H, 2 старших біти) та молодшим (L, 8 біт). Мінімальне значення дорівнює 0,

максимальне дорівнює 1023 (2^{10}–1). Це значення помножується на деякий коефіцієнт для того, щоб отримати результат вимірювання у фізичних одиницях (градусах Цельсія для температури, паскалях або кг/м^2 для датчиків тиску, і т.п.). Значення цього коефіцієнту визначається типом датчика.

В змінній *freq* передається значення частоти напруги, по якому можна визначити частоту обертів валу дизеля:

$$w = \frac{1}{freq} \cdot 46800 \text{ (об/хв).}$$

При частоті напруги генератора 50 Гц значення змінної *freq*=78. Тоді значення обертів дизеля дорівнює 600 об/хв.

Стан кнопок (байт 18 відповіді):

X	X	PIN6	PIN5	PIN4	PIN3	PIN2	PIN1

Якщо значення біта дорівнює 1, то відповідному вході присутній сигнал лог.1.

Команда зміни стану реле (4 байта):

Байт 1	Байт 2	Байт 3	Байт 4							
‘Y’	‘N’	‘R’	X	X	X	X	R4	R3	R2	R1

Логічна «1» в позиціях R4-R1 вмикає відповідне реле, логічний «0» вимикає реле.

Призначення бітів:

PIN1 – немає води в системі охолодження
PIN2 – немає масла в змащувальній системі
PIN3 – дизель готовий до пуску
PIN4 – збудження генератора увімкнене
PIN5 – небезпечна концентрація газів
PIN6 – зарезервовано
R1 – запуск дизеля
R2 – зупинка дизеля
R3 – аварійний тиск масла
R4 – перегрів дизеля
0 – температура масла
1 –тиск масла
2 – температура води на вході системи
3 – температура води на виході системи
4 – тиск води в системі охолодження
5 – температура вихлопних газів
6 – витрати палива
7 – зарезервовано

Під користувацьким інтерфейсом (КІ) програми розуміється сукупність елементів, що дозволяють користувачеві програми управляти її роботою й отримувати необхідні результати. Фактично, КІ – це канал, по якому здійснюється взаємодія користувача й програми.

В Labview для роботи з послідовним портом використовується стандартний інтерфейс програмного забезпечення, який не залежить від протоколу або шини апаратної частини, заснований на архітектурі програмного забезпечення віртуального приладу – VISA. VISA є стандартним програмним інтерфейсом програми (API), що здійснює, введення/вивід, для програмування контрольно-вимірювального обладнання VISA може управляти багатьма типами приладів (VXI, GPIB, PXI або з послідовною передачею), викликаючи відповідні драйвери, які залежать від типу використовуваного приладу. Ім'я ресурсу VISA аналогічно каналу збору даних. Він говорить функціям VISA, з яким приладом або ресурсом ви взаємодієте.

Вид палітри функцій послідовної комунікації (Serial) наведений на рис. 7.17. Дані функції є спеціалізованими функціями VISA для послідовних портів. До складу палітри входять функції конфігурування послідовного порту VISA, читання, запису й закриття VISA.

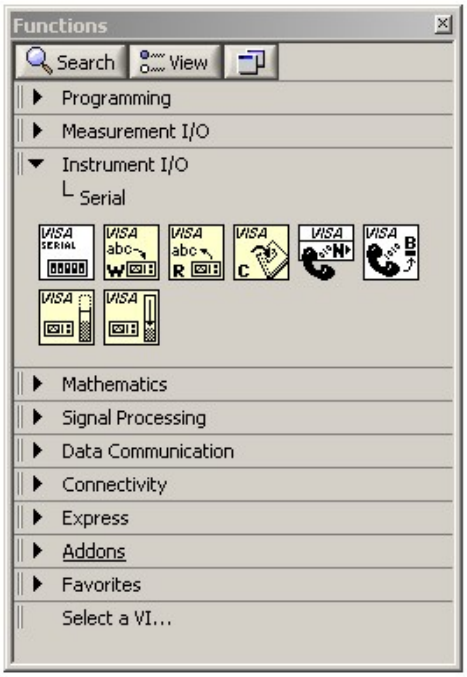
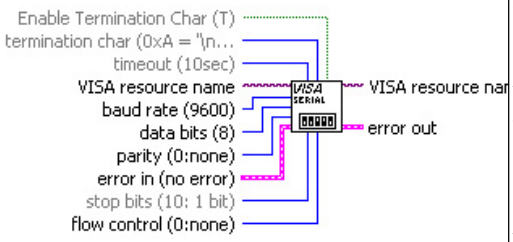
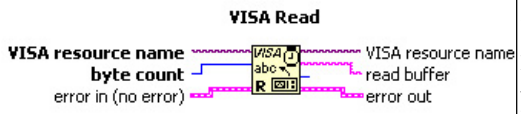
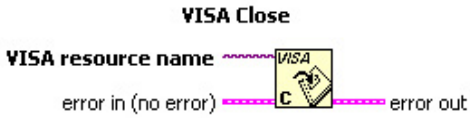
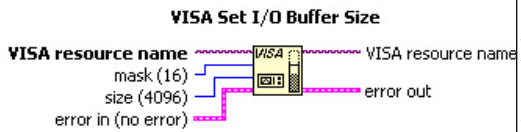
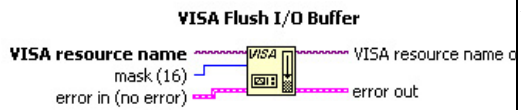


Рис.7.17.

В табл. 7.2 наводиться опис блоків для роботи з послідовним комунікаційним інтерфейсом. Опис кластера помилки представлено в табл. 7.3.

Таблиця 7.2.

Блок	Опис параметрів
VISA Configure Serial Port  Конфігурування послідовного порту VISA	Функція ініціалізує послідовний порт, що визначається за допомогою входу «ім'я ресурсу VISA» (VISA resource name), роблячи певні установки. При подачі на вхід «дозволити символ завершення» (Enable Termination Char) значення «ІСТИНА» (за замовчуванням) послідовний пристрій підготовляється до розпізнавання символу завершення (termination char) А якщо ні, то воно не розпізнає такий символ. Символ із входу «символ завершення» (termination char) викликається для завершення операції читання. Операція читання завершується при зчитуванні символу завершення з послідовного пристрою 0x0A є шістнадцятирічним еквівалентом символу «переведення рядка» (linefeed character (\n)) Для рядків повідомлень, які завершуються символом «повернення каретки» (carriage return (\r)), на цьому вході необхідно встановити значення 0x0D. Значення на вході «ліміт часу очікування» (timeout) установлює припустиме значення інтервалу очікування виконання операцій запису або читання. Вхід «ім'я ресурсу VISA» (VISA resource name) визначає ресурс, що відкривається. Цей елемент керування також визначає сесію й клас. Вхід «біти даних» (data bits) визначає число бітів у даних, що надходять,. Значення змінюється в діапазоні від 5 до 8 біт. За замовчуванням число біт дорівнює 8. Вхід «парність» (parity) визначає спосіб перевірки парності, використовуваний для кожного переданого або прийнятого кадру. Вхід «стоп-біти» (stop bits) визначає число стопових бітів, використовуваних для індикації кінця кадру (10 – 1 стоп-біт, 15 – 1.5 стоп-біта, 20 – 2 стоп-біта). Вхід «контроль потоку» (flow control) установлює тип контролю, використовуваного механізмом передачі. Вхід error in – кластер помилки, що описує стан помилки перед виконанням цього блоку. За замовчуванням помилки немає.
VISA Write  Запис	Функція записує дані з буфера запису (write buffer) у пристрій або інтерфейс, визначені за допомогою імені ресурсу VISA (VISA resource name). Вихід «підрахунок повернення» (return count) містить дійсне число записаних байтів.

 Читання	Функція зчитує задане число байтів із пристрою або інтерфейсу, визначеного за допомогою імені ресурсу VISA (VISA resource name), і повертає дані в буфері читання (read buffer). Для установки синхронного читання необхідно в контекстному меню функції вибрати опцію «виконувати введення/вивід» синхронно (Do I/O Synchronously). Операція повертається тільки після завершення передачі.
 Закрити	Функція закриває сесію пристрою або об'єкт події, визначені за допомогою імені ресурсу VISA (VISA resource name). Кожна відкрита сесії VISA повинна бути закрита при завершенні роботи з нею.
 Байти послідовного порту	Функція повертає число байтів вхідного буфера заданого послідовного порту. Властивість «число байтів послідовного порту» (Number of Bytes at Serial Port) визначає число байтів, доступних у поточний час у послідовному порту, використовуваному в даній сесії. Функція «Байти послідовного порту» VISA є «Вузлом властивостей» (Property Node). За допомогою вузла властивостей указаного класу можуть бути програмно встановлені або зчитані всі параметри послідовного порту.
	Функція встановлює розмір буфера введення/виводу, визначеного за допомогою імені ресурсу VISA (VISA resource name). Параметр «Маска» (Mask) визначає, для якого буфера встановлюється розмір: 16 – для вхідного буфера, 32 – для вихідного, 48 – для обох. Параметр «Розмір» (Size) визначає розмір буфера (у байтах).
	Функція очищає буфер уведення/виводу, визначеного за допомогою імені ресурсу VISA (VISA resource name). Параметр «Маска» (Mask) визначає, який з буферів буде очищений: 16 – вхідний буфер, 32 – вихідний буфер, 48 – обидва.

Таблиця 7.3.

status 	Значення «ІСТИНА» (TRUE) сигналізує про наявність помилки. Нормальний стан «НЕПРАВДА» (FALSE) – немає помилки.
code 	Код помилки. Нормальне значення 0. Якщо параметр **status** приймає значення «ІСТИНА», у цьому параметрі перебуває код помилки.
source 	Містить назву блоку, у якому виникла помилка. За замовчуванням – порожній рядок.

На рис. 7.18 представлена блок-діаграма програми збору даних про стан газодизельного агрегату, яку було створено в середовищі LabView. Блок-діаграма виконує алгоритм роботи автоматизованого робочого місця.

При запуску програми перша дія, що виконується – ініціалізація та відкриття послідовного порта. Ініціалізація – це налаштування таких параметрів, як швидкість обміну (9600 біт/с), тривалість тайм-ауту (запобігає зависанню програми), кількість біт даних, кількість стопових біт, наявність перевірки на парність. Після цього починає виконуватися головний цикл програми. Він виконується доти, доки користувач не натисне кнопку "Stop" чи у разі виникнення проблем з роботою послідовного порта.

В систему відправляється команда "YNG", при отриманні якої виконується вимірювання всіх параметрів в системи. В комп'ютер передається пакет розміром 20 байт у відповідності до описаного вище протоколу. Перевіряється розмір пакету та наявність початкового і кінцевого символів: D!. У разі відсутності цих символів формується логічний сигнал, який інформує оператора про помилку формату.

Якщо помилки формату немає, то виконується блок Case Structure, в якому відбувається розшифрування отриманого інформаційного пакету. Оскільки позиції даних про параметри дизельного агрегату, що отримані, відомі, то за допомогою блоків роботи з масивами та простих арифметичних операцій отримані дані переводяться у фізичні величини, що відповідають значенням виміряних параметрів. Ці дані відображаються на інтерфейсі користувача у зручній формі.

На рис. 7.19 представлений користувацький інтерфейс програми збору даних про стан газодизельного агрегату.

Розроблений інтерфейс користувача (автоматизоване робоче місце) має стилізовані прилади для відображення параметрів дизельного агрегату. Всі елементи керування розташовані у чотирьох групах: параметри системи змащення, стан системи охолодження, керування та додаткові параметри. На екрані оператор має можливість контролювати температуру та тиск масла, а також наявність масла у системі. Якщо температура масла виходить за припустимі межі, то загоряється відповідний індикатор (змінює колір з зеленого на червоний).

Також контролюється температура охолоджувальної рідини на вході і на виході системи. На вході системи температура води не повинна перевищувати 32 градуси. Якщо це не виконується, загоряється відповідний індикатор.

Кнопки керування, при їх натисканні, формують команди на замикання відповідних реле: Пуск дизеля, зупинка дизеля. Крім того, ще 2 реле замикаються якщо температура води перевищує максимально припустимий рівень або тиск масла впав нижче мінімального. Для зручності параметри температури також дублюються у вигляді графіку.

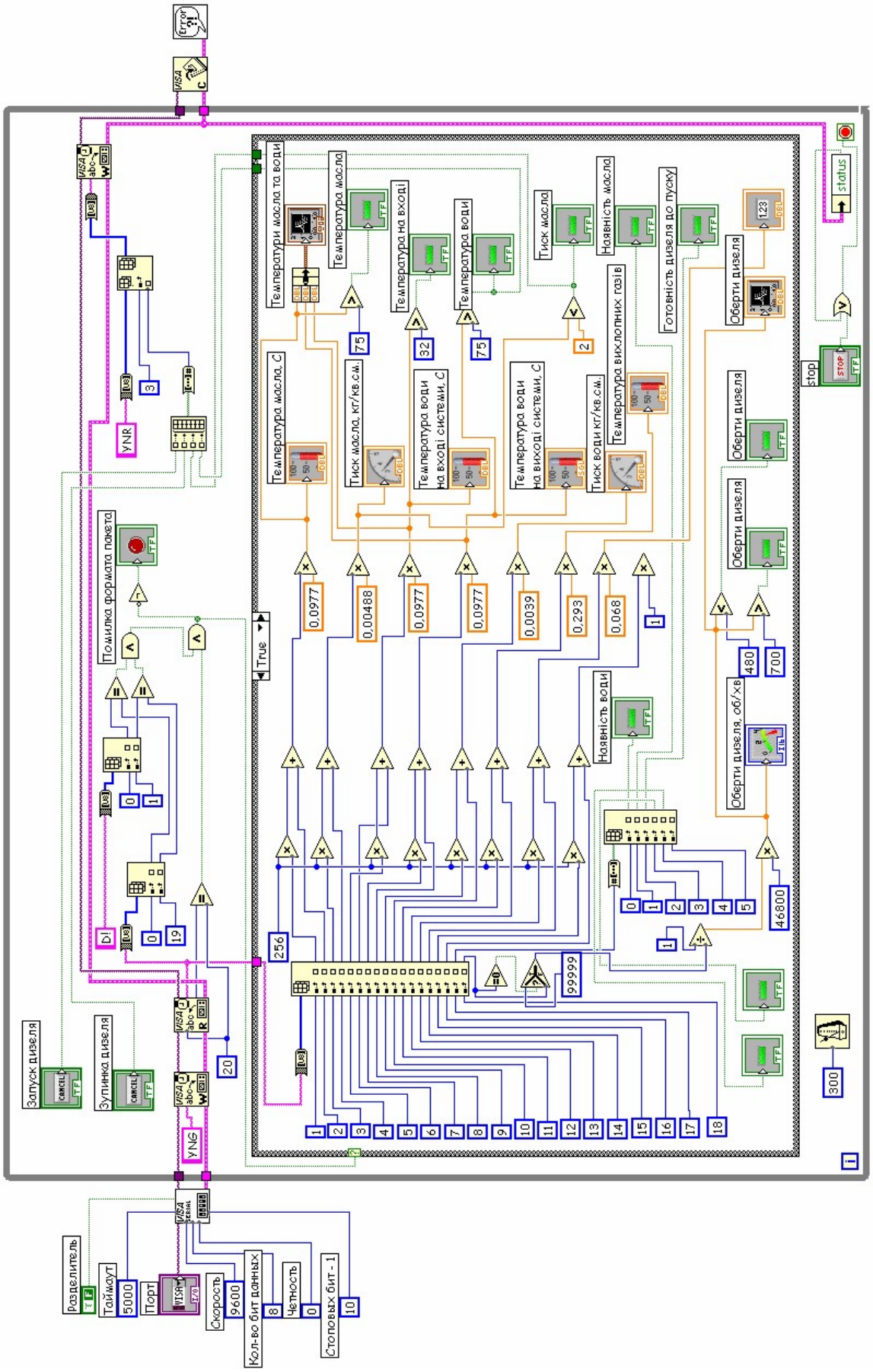

Температури масла та води
Температура масла
Температура на вході
Температура води
Тиск масла
Наявність масла
Готовність дизеля до пуску
Оберти дизеля
Температура масла, С
Тиск масла, кг/кв.см.
Температура води на вході системи, С
Температура води на виході системи, С
Тиск води кг/кв.см.
Температура вих.лопних газів
Оберти дизеля
Оберти дизеля
Оберти дизеля
Оберти дизеля, об/хв
Наявність води
Помилка формата пакета
Запуск дизеля
Зупинка дизеля
True
stop
status
YNR
YNG
0,0977
0,00488
0,0977
0,0977
0,0039
0,293
0,068
480
700
46800
99999
256
300
20
Розділитель
Тайм-аут
5000
Порт
Скорость
9600
Кол-во бит данных
8
Четность
0
Стоповых бит - 1
10

Рис. 7.18.

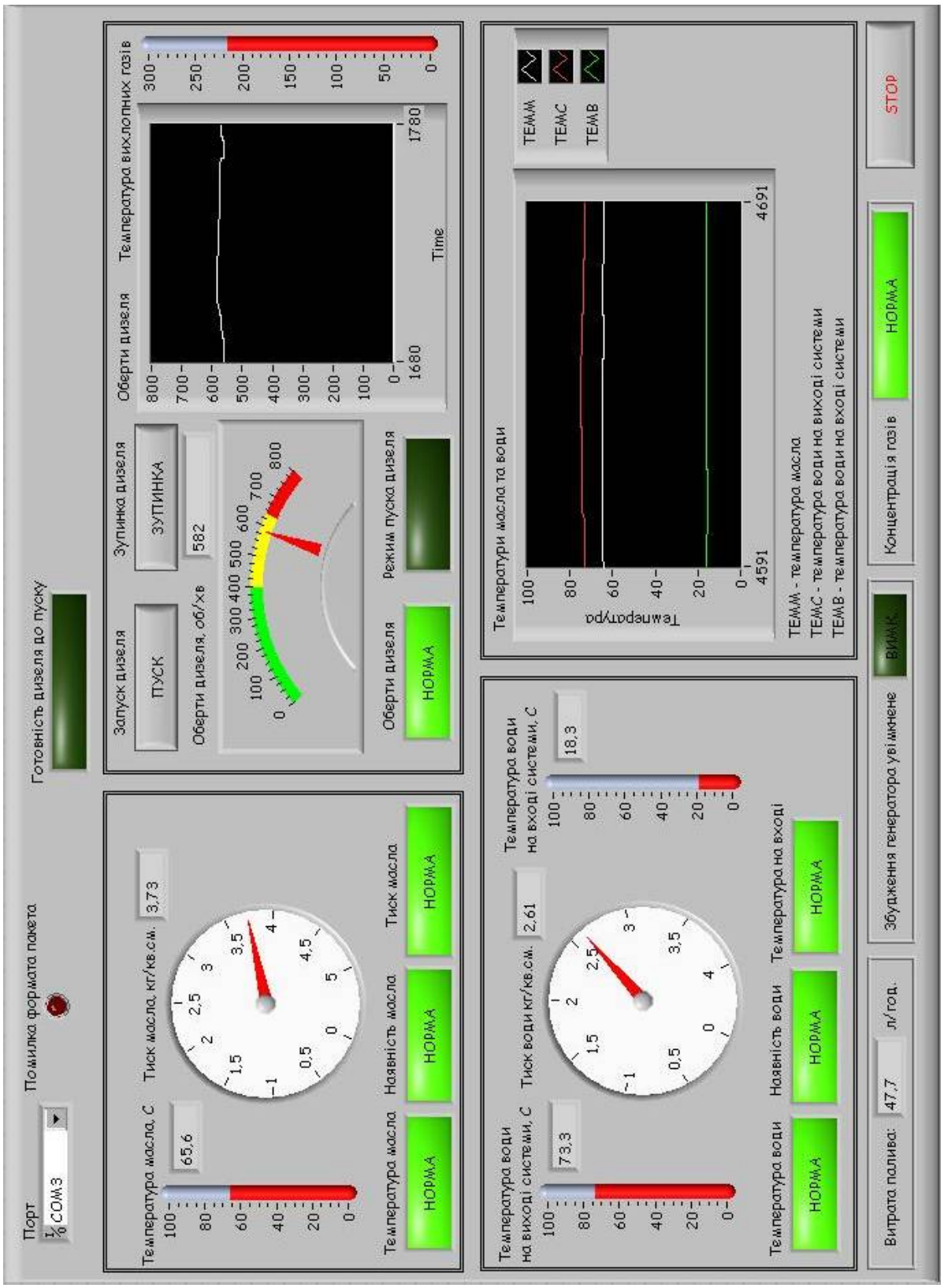


Рис. 7.19.

Запропонована структура мікропроцесорної системи забезпечує виконання основних вимог до системи моніторингу та його функціональним завданням. Елементи системи пов'язані між собою на фізичному і (або) логіко-функціональному рівні. Введення даних у систему

реалізується апаратними засобами підсистеми збору даних, а керує процесом збору користувач, використовуючи екранні форми інтерфейсу.

Система відображає виміряні і розраховані сигнали в графічному вигляді, розраховані параметри і показники в текстовому вигляді на екрані комп'ютера. Взаємодія з оператором ведеться за допомогою зручного графічного інтерфейсу.

7.6. Апаратно-програмні засоби керування рухом мобільної платформи за допомогою смартфону з ОС Android

7.6.1. Апаратна частина системи радіокерування по інтерфейсу Bluetooth

Керування електронними пристроями та системами можна здійснювати за допомогою смартфона з використанням модуля Bluetooth. На рис. 7.20 наведено принципову схему підключення Bluetooth-радіомодему до контролера Arduino.

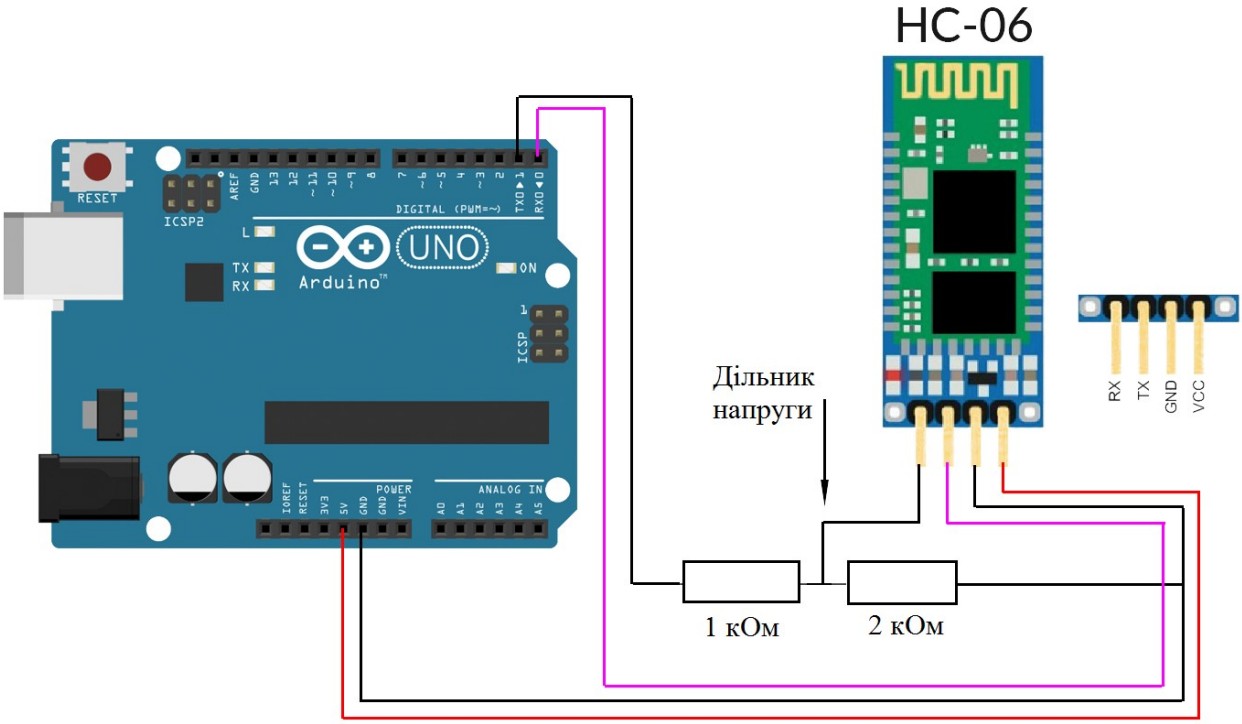


Рис. 7.20.

Для підключення радіомодуля Bluetooth використовується лише чотири провідники, два з яких це напруга живлення, а інші два – лінія передавання та приймання даних. Радіомодуль підключається до контролера по інтерфейсу Bluetooth.

В якості об’єкту керування в даному прикладі виступає рухома двохколісна платформа, зовнішній вигляд якої представлено на рис. 7.21.

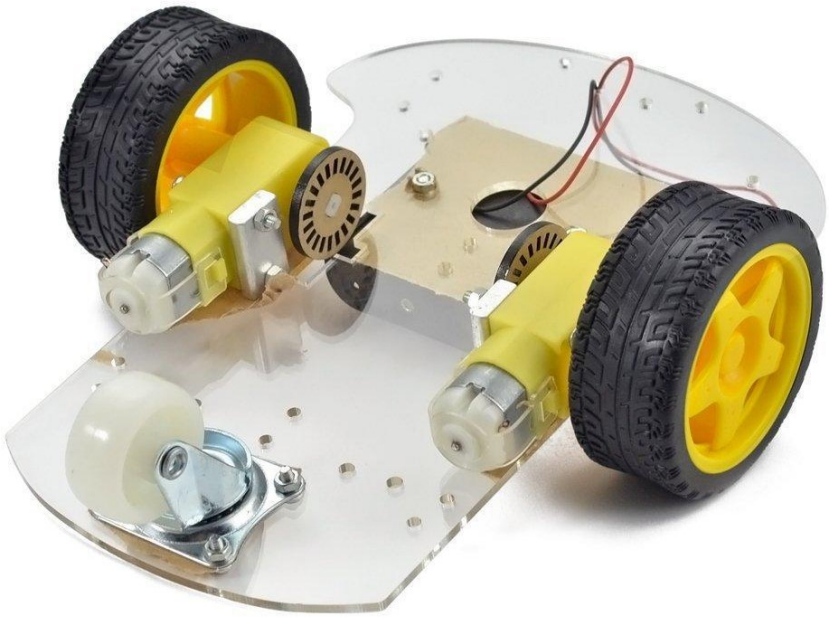


Рис. 7.21.

Ця платформа приводиться в рух двома двигунами постійного струму. Для керування цими двигунами за допомогою Arduino потрібно використовувати спеціальний силовий драйвер. Одним з популярних драйверів двигунів є модуль L298, зовнішній вигляд якого та схему підключення представлено на рис. 7.22.

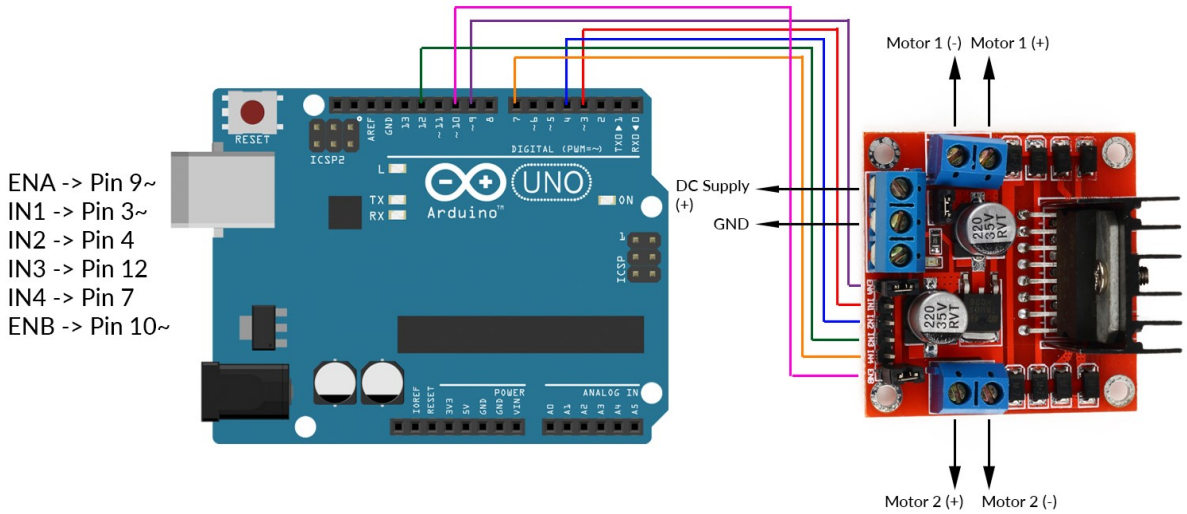


Рис. 7.22.

Всі ці елементи (радіомодем Bluetooth HC-06, Arduino UNO, модуль L298) монтуються на рухомій платформі, яка живиться від чотирьох акумуоляторів АА напругою 1,2В кожний. Повна схема системи керування рухомою платформою за допомогою смартфона шляхом передавання команд по Bluetooth отримується поєднанням схем, представлених на рис. 7.20 та рис. 7.22, та буде наведено далі.

Для створення схеми системи керування та її подальшого моделювання в середовищі Proteus спочатку необхідно завантажити з

мережі Internet та встановити бібліотеки контролера Arduino UNO, Bluetooth та драйвера двигунів L298. Процедура завантаження та встановлення бібліотеки Arduino UNO детально описана в першому розділі. Для завантаження бібліотеки з компонентом драйвера двигунів L298 потрібно перейти за посиланням:

https://www.theengineeringprojects.com/2017/09/l298-motor-driver-library-proteus.html

На головній сторінці цього сайту потрібно натиснути на кнопку «Download Proteus Library», як показано на рис. 7.23.

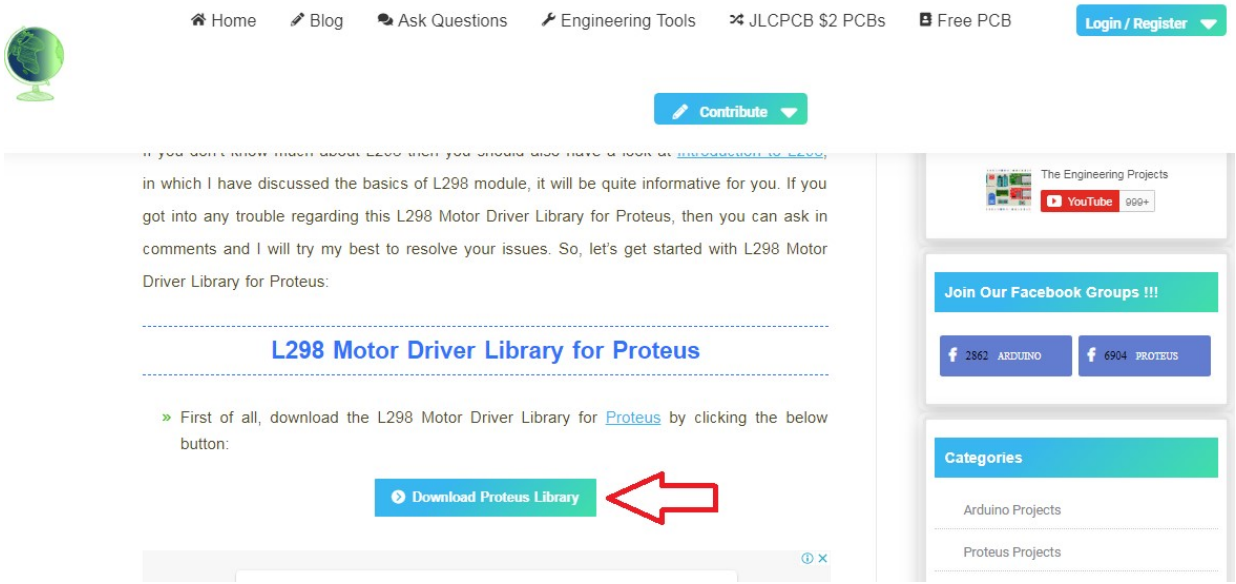


Рис. 7.23.

Після того, як архів з файлами бібліотеки буде завантажено, його необхідно розпакувати, а файли L298MotorDriverTEP.IDX та L298MotorDriverTEP.LIB скопіювати в каталог з бібліотеками Proteus, адреса розташування якого наступна:

C:\Program Files (x86)\Labcenter Electronics\Proteus 8 Professional\LIBRARY

На рис. 7.24 показано вікно провідника Windows, в якому файли бібліотеки розміщено за вказаною адресою.

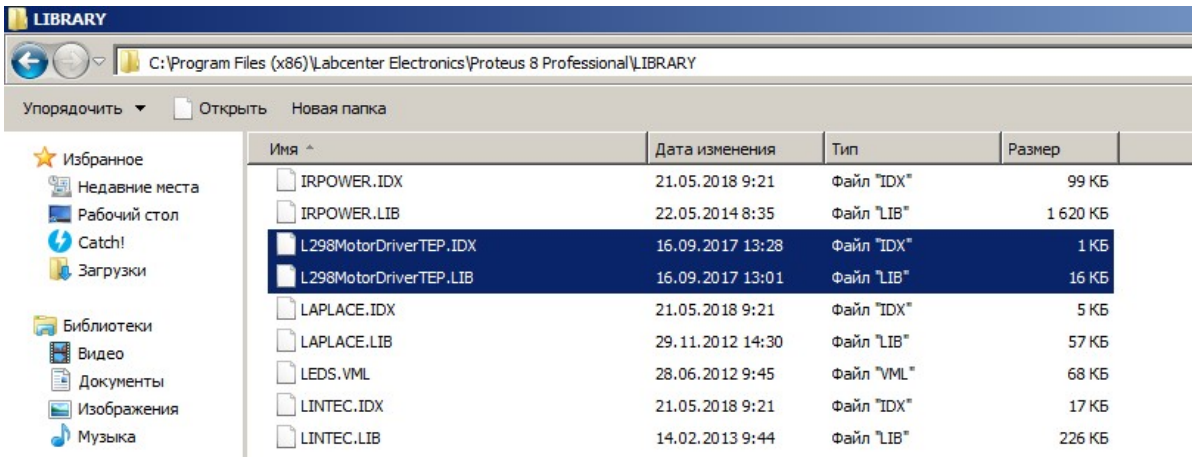


Рис. 7.24.

Після цього можна запустити середовище проектування та моделювання електронних схем Proteus та розробити принципову схему системи керування рухом платформи з можливістю керування по Bluetooth за допомогою смартфона. На рис. 7.25 представлено принципову схему системи керування.

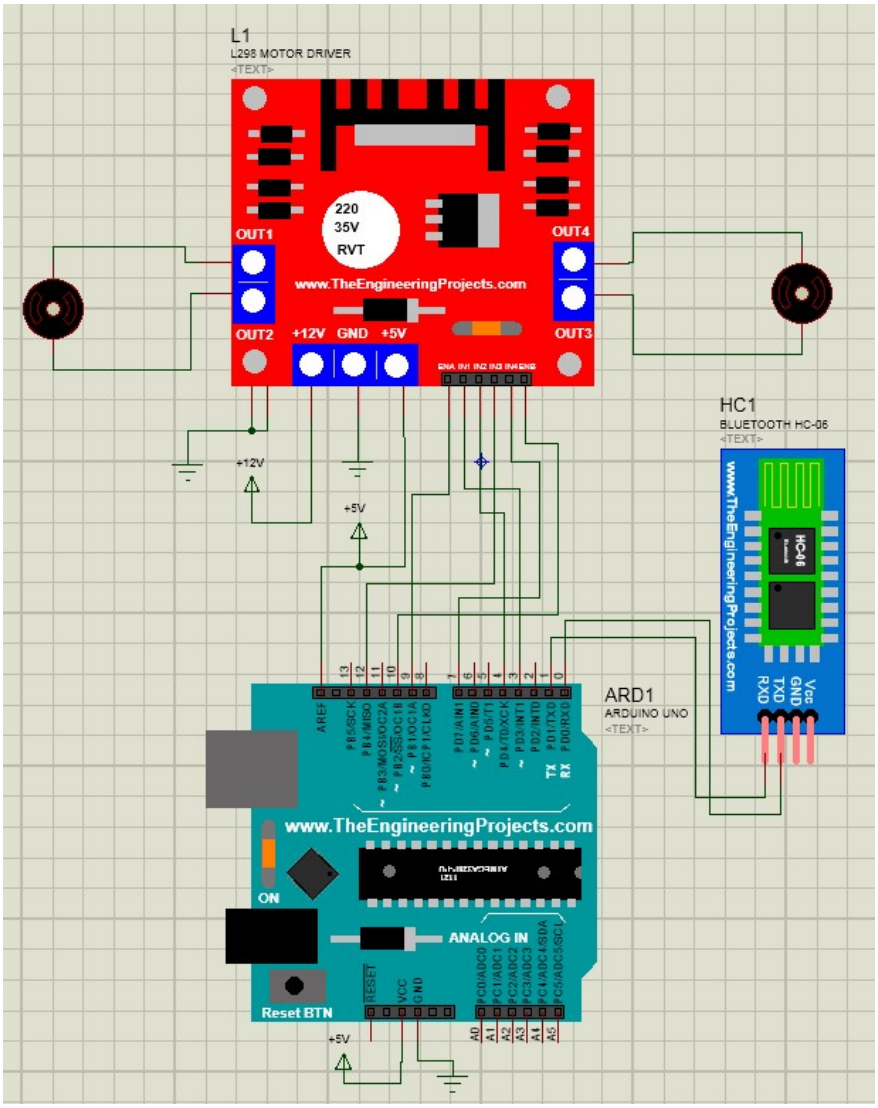


Рис. 7.25.

Чергування різнойменних сигналів (високий логічний рівень або низький) на парах виводів IN1, IN2 і IN3, IN4 задають напрямок обертання моторів. Виводи ENA, ENB (ENA прив'язаний до IN1, IN2; ENB прив'язаний до IN3, IN4) відповідають за роздільне управління каналами. Ці канали можуть використовуватися в двох режимах:

1. Умовно «активному» режимі, коли ними буде керувати контролер – високий логічний рівень сигналу дозволяє обертання моторів, низький рівень сигналу забороняє незалежно від стану виводів «IN». Для регулювання швидкості моторів, на EN-виводи подається ШІМ (PWM) сигнал.

2. Умовно «пасивному» режимі, підтягнувши виводи «EN» до високого рівня (+5В). Для цього на платі, поруч з виводами ENA і ENB знаходяться роз’єми, з'єднані з +5В. Замкнути виводи можна за допомогою джамперів. В даному режимі немає можливості регулювання швидкості двигунів, і вони будуть завжди обертатися на повну швидкість. Напрямок обертання буде задаватися як і раніше, а для зупинки в даному варіанті, стан виводів буде вже грати роль. Для зупинки потрібно буде подавати однойменні сигнали на виводи «IN».

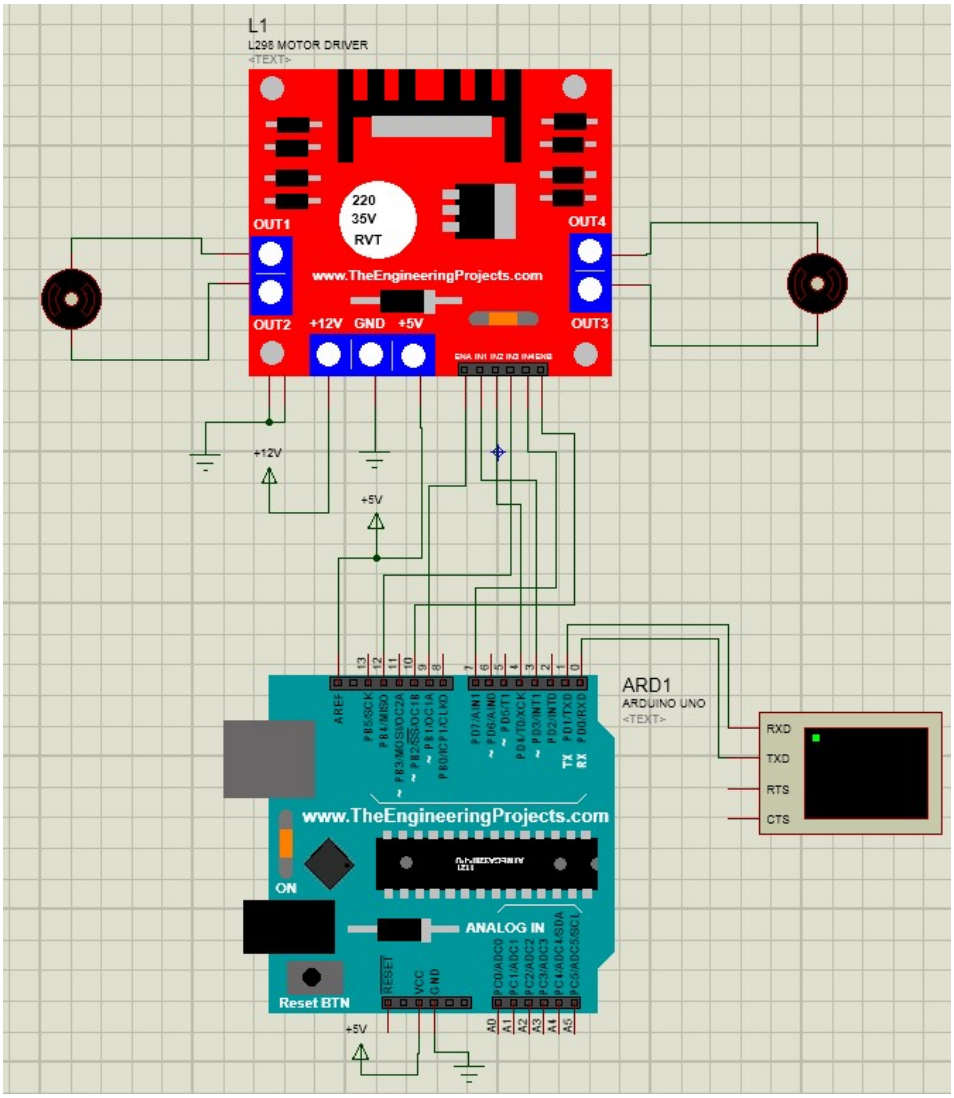


Рис. 7.26.

В даному прикладі використовується умовно «активний» режим роботи драйвера двигунів. Для моделювання роботи системи керування та налагодження програмного забезпечення можна замінити модуль Bluetooth HC-06 віртуальним терміналом (рис. 7.26). В процесі моделювання системи у вікні віртуального терміналу потрібно ввести необхідну команду (цифри від 1 до 6). Після того, як команду буде введено, анімовані компоненти моторів почнуть обертатися в ту чи іншу сторону, в залежності від введеної команди.

Текст програми для контролера Arduino UNO у складі системи керування рухом платформи за допомогою команд, отриманих зі смартфону по Bluetooth, має наступний вигляд:

```
// оголошення змінної для зберігання номеру виводу,
// до якого підключений вхід дозволу каналу A (ENA)драйвера двигуна
int Motor_A_Enable = 9;

// оголошення змінної для зберігання номеру виводу, до якого
// підключений перший вхід каналу A (позначений як IN1 на драйвері)
int Motor_A_Reverse = 3;

// оголошення змінної для зберігання номеру виводу, до якого
// підключений другий вхід каналу A (позначений як IN2 на драйвері)
int Motor_A_Forward = 4;

// оголошення змінної для зберігання номеру виводу,
// до якого підключений вхід дозволу каналу B (ENB)драйвера двигуна
int Motor_B_Enable = 10;

// оголошення змінної для зберігання номеру виводу, до якого
// підключений перший вхід каналу B (позначений як IN3 на драйвері)
int Motor_B_Reverse = 12;

// оголошення змінної для зберігання номеру виводу, до якого
// підключений другий вхід каналу B (позначений як IN4 на драйвері)
int Motor_B_Forward = 7;

// ця функція викликається один раз при вмиканні живлення контролера
// саме тут виконується налаштування режимів роботи виводів контролера
// та периферійних вузлів, таких як UART для реалізації зв'язку по Bluetooth
void setup() {
  // налаштування швидкості передавання даних по UART 9600 біт/с
  Serial.begin(9600);

  // всі виводи контролера, які виконують керування драйвером двигуна
  // налаштовуються на вихід - це виводи з номерами 9, 3, 4, 10, 12, 7
  // номери виводів зберігаються у відповідних змінних, оголошених раніше
  // це зроблено для зручності, адже при підключенні драйвера до інших виводів
контролера
  // потрібно буде внести зміни лише в одному місці програми - на початку скетчу,
  // при оголошенні та початковій ініціалізації цих змінних
```

```
  pinMode(Motor_A_Enable, OUTPUT);
  pinMode(Motor_A_Forward, OUTPUT);
  pinMode(Motor_A_Reverse, OUTPUT);

  pinMode(Motor_B_Enable, OUTPUT);
  pinMode(Motor_B_Forward, OUTPUT);
  pinMode(Motor_B_Reverse, OUTPUT);
}

// код в цій функції виконується циклічно протягом всього часу,
// поки на контролер поступає напруга живлення
void loop() {
  // перевірка наявності в буфері прийому UART даних
  if(Serial.available() > 0)  // якщо отриманий байт даних, умова виконується
  {
    // оголошення додаткової змінної для збереження прийнятого по UART байту даних
    char data;
    // зчитування прийнятого байту даних в змінну
    data = Serial.read();
    // передавання прийнятого байту даних в зворотньому напряму (exo)
    // це зроблено для інформування передавача (в даному випадку смартфон)
    // про те, що система керування отримала команду
    Serial.write(Serial.read());
    // реалізація конструкції вибору - перевіряється вміст змінної
    // на рівність одній з альтернатив
    switch (data)
    {
      // якщо прийняте значення '1' (ASCII-код 49), на виводах контролера
      // формуються сигнали, що забезпечують рух платформи вперед
      case '1':
        analogWrite(Motor_B_Enable, 100); // формування ШІМ-сигналу на виводі 10
        analogWrite(Motor_A_Enable, 100); // формування ШІМ-сигналу на виводі 9
        // встановлення на виводі 3 контролера низького рівня сигналу
        digitalWrite(Motor_A_Reverse, LOW);
        // встановлення на виводі 12 контролера низького рівня сигналу
        digitalWrite(Motor_B_Reverse, LOW);
        // встановлення на виводі 4 контролера високого рівня сигналу
        digitalWrite(Motor_A_Forward, HIGH);
        // встановлення на виводі 7 контролера високого рівня сигналу
        digitalWrite(Motor_B_Forward, HIGH);
        break;

      // якщо прийняте значення '2' (ASCII-код 50), на виводах контролера
      // формуються сигнали, що забезпечують рух платформи назад
      case '2':
        analogWrite(Motor_B_Enable, 100); // формування ШІМ-сигналу на виводі 10
        analogWrite(Motor_A_Enable, 100); // формування ШІМ-сигналу на виводі 9
        // встановлення на виводі 4 контролера низького рівня сигналу
        digitalWrite(Motor_A_Forward, LOW);
        // встановлення на виводі 7 контролера низького рівня сигналу
        digitalWrite(Motor_B_Forward, LOW);
        // встановлення на виводі 3 контролера високого рівня сигналу
```

```
      digitalWrite(Motor_A_Reverse, HIGH);
      // встановлення на виводі 12 контролера високого рівня сигналу
      digitalWrite(Motor_B_Reverse, HIGH);
      break;

    // якщо прийняте значення '3' (ASCII-код 51), на виводах контролера
    // формуються сигнали, що забезпечують рух платформи вперед ліворуч
    case '3':
      analogWrite(Motor_A_Enable, 100); // формування ШІМ-сигналу на виводі 9
      analogWrite(Motor_B_Enable, 0); // заборона роботи каналу B
      // встановлення на виводі 3 контролера низького рівня сигналу
      digitalWrite(Motor_A_Reverse, LOW);
      // встановлення на виводі 4 контролера високого рівня сигналу
      digitalWrite(Motor_A_Forward, HIGH);
      break;

    // якщо прийняте значення '4' (ASCII-код 52), на виводах контролера
    // формуються сигнали, що забезпечують рух платформи вперед праворуч
    case '4':
      analogWrite(Motor_B_Enable, 100); // формування ШІМ-сигналу на виводі 10
      analogWrite(Motor_A_Enable, 0); // заборона роботи каналу A
      // встановлення на виводі 12 контролера низького рівня сигналу
      digitalWrite(Motor_B_Reverse, LOW);
      // встановлення на виводі 7 контролера високого рівня сигналу
      digitalWrite(Motor_B_Forward, HIGH);
      break;

    // якщо прийняте значення '5' (ASCII-код 53), на виводах контролера
    // формуються сигнали, що забезпечують рух платформи назад ліворуч
    case '5':
      analogWrite(Motor_A_Enable, 100); // формування ШІМ-сигналу на виводі 9
      analogWrite(Motor_B_Enable, 0); // заборона роботи каналу B
      // встановлення на виводі 3 контролера високого рівня сигналу
      digitalWrite(Motor_A_Reverse, HIGH);
      // встановлення на виводі 4 контролера низького рівня сигналу
      digitalWrite(Motor_A_Forward, LOW);
      break;

    // якщо прийняте значення '6' (ASCII-код 54), на виводах контролера
    // формуються сигнали, що забезпечують рух платформи назад праворуч
    case '6':
      analogWrite(Motor_B_Enable, 100); // формування ШІМ-сигналу на виводі 10
      analogWrite(Motor_A_Enable, 0); // заборона роботи каналу A
      // встановлення на виводі 12 контролера високого рівня сигналу
      digitalWrite(Motor_B_Reverse, HIGH);
      // встановлення на виводі 7 контролера низького рівня сигналу
      digitalWrite(Motor_B_Forward, LOW);
      break;
      // якщо модуль Bluetooth отримує будь-яке значення, яке не співпадає з
переліченими,
      // тоді обидва мотори вимикаються шляхом формування сигналів низького рівня
      // на дозволяючі входи драйвера двигунів
```

```
    default:
      analogWrite(Motor_A_Enable, 0);
      analogWrite(Motor_B_Enable, 0);
      break;
  }
 }
}
```

Значення скважности задається функцією analogWrite (pin, число), де число змінюється від 0 до 255, і прямо пропорційно скважности сигналу.

7.6.2. Розробка програмних засобів дистанційного керування для ОС Android у середовищі Android Studio

Для розробки програми для смартфонів з операційною системою Android використовується середовище Android Studio, яке можна завантажити з офіційного сайту за посиланням:

https://developer.android.com/studio

Процедура встановлення середовища Android Studio на комп'ютер з операційною системою Windows є стандартною і не потребує додаткових коментарів. Після запуску середовища Android Studio потрібно створити новий проект. Для цього з меню File потрібно вибрати підменю New і команду New Project, як показано на рис. 7.27.

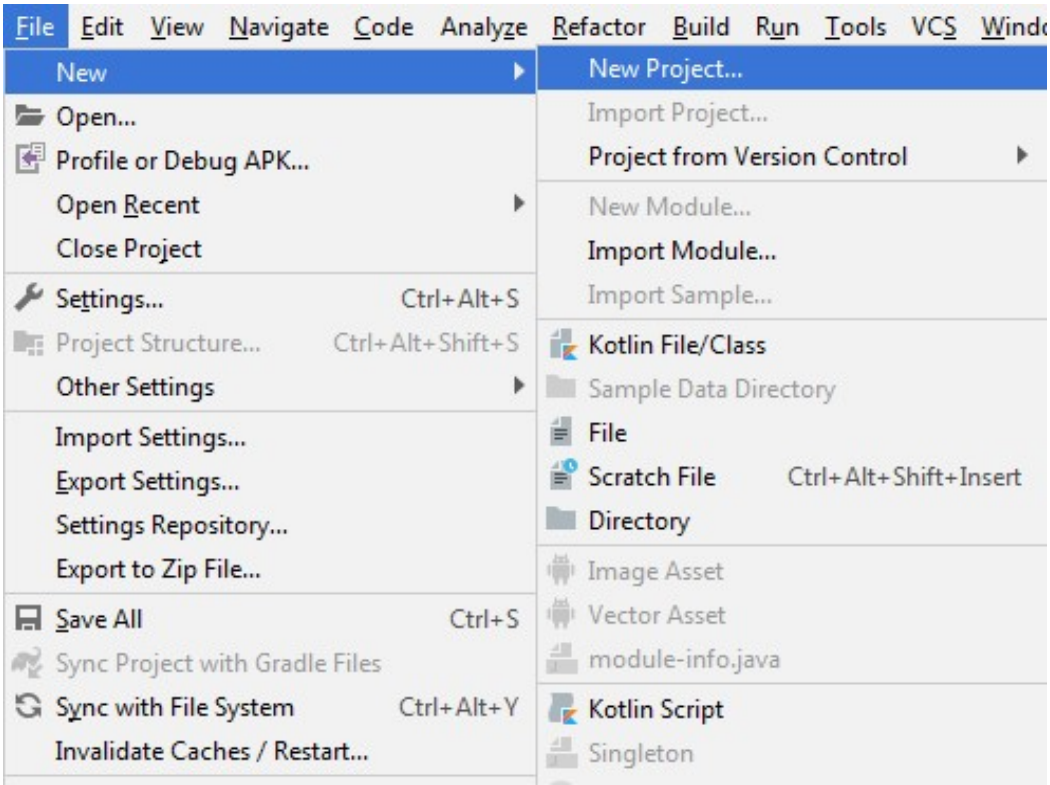


Рис. 7.27.

Після цього з’явиться діалогове вікно (рис. 7.28), в якому необхідно вказати тип додатку – для планшетів та смартфонів. Із запропонованого набору варіантів потрібно вибрати Empty Activity і натиснути на кнопку Next у нижній частині вікна праворуч.

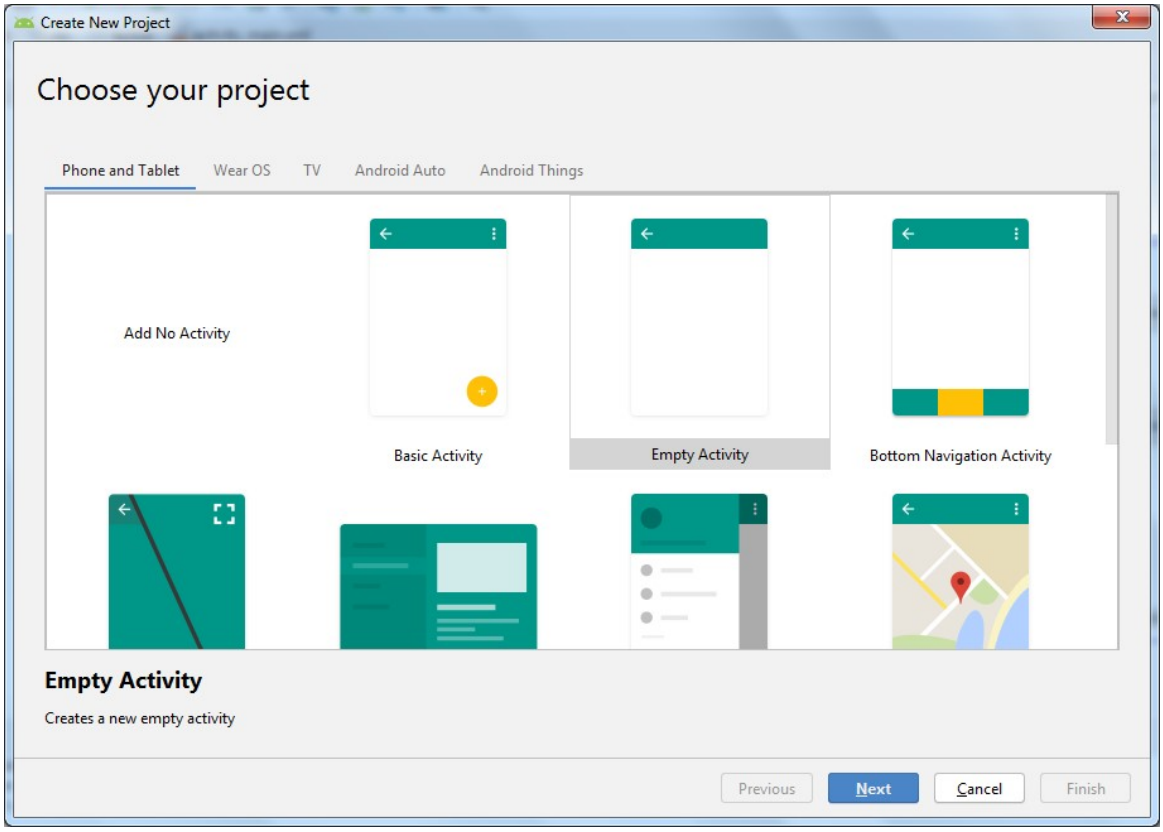


Рис. 7.28.

В наступному діалоговому вікні, що представлене на рис. 7.29, потрібно ввести назву додатку (в даному прикладі TestApp), вибрати мову програмування Java та вказати API, що буде використовуватися. Цей приклад демонструє розробку додатку для версії операційної системи Android 6.0 (Marshmallow), і, відповідно, використовується API 23. Після того, як всі налаштування проекту зроблені, слід натиснути на кнопку Finish. Створення проекту займе деякий час, що може складати декілька хвилин, в залежності від характеристик комп’ютера. Слід дочекатися, коли всі файли проекту будуть створені, і приступити до програмування.

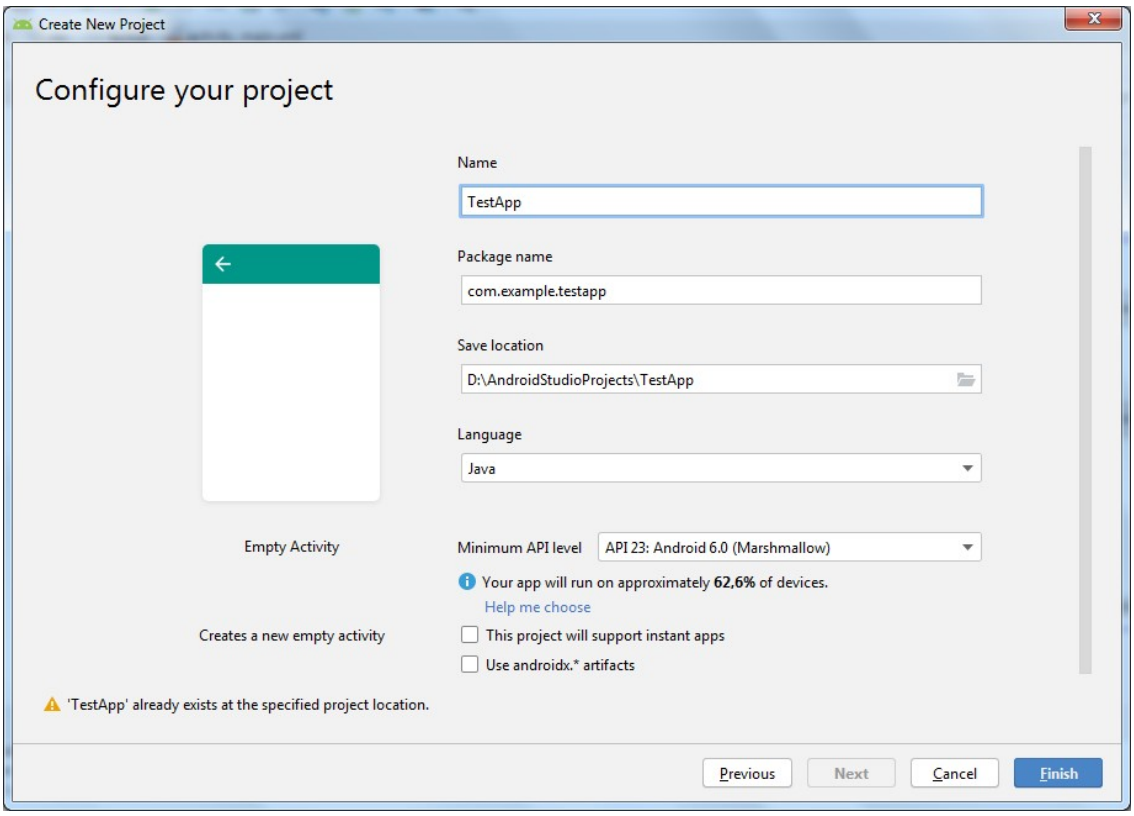


Рис. 7.29.

Перш за все потрібно створити інтерфейс користувача, на якому будуть розташовані кнопки керування рухом платформи. Для цього потрібно у дереві файлів проекту, що відображається в лівій частині головного вікна середовища розробки Android Studio, вибрати файл activity_main.xml і відкрити його подвійним натисканням лівої кнопки миші. В середовищі відкриється текстовий редактор, в якому на мові XML виконується опис інтерфейсу програми та властивості компонентів, що використовуються. На рис. 7.30 показане головне вікно середовища Android Studio. Ліворуч можна бачити структуру файлів проекту у вигляді дерева. По центру розташоване вікно текстового редактору. Праворуч міститься вікно, де в графічному вигляді зображений інтерфейс програми. Зовнішній вигляд елементів та їх розташування можна змінювати як за допомогою миші та редагуванням властивостей окремих компонентів, так і шляхом безпосереднього редагування XML-файлу, що є більш зручним.

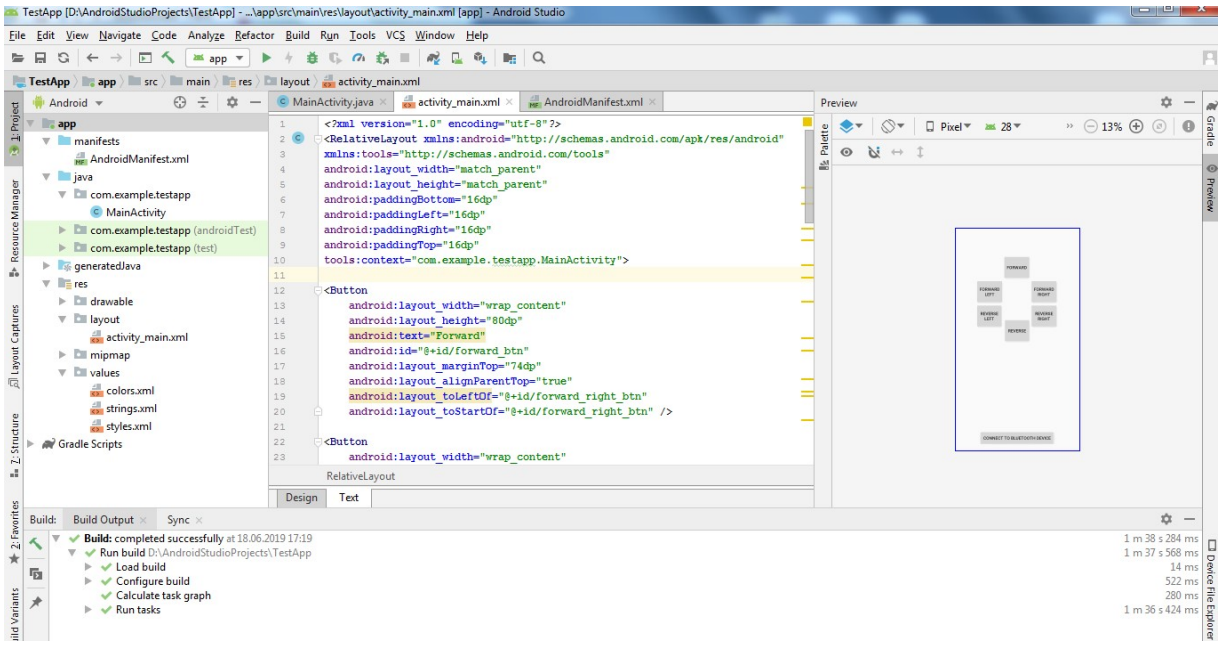


Рис. 7.30.

Далі наведено вміст XML-файлу activity_main.xml, що описує графічний інтерфейс користувача.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:layout_height="match_parent"
android:paddingBottom="16dp"
android:paddingLeft="16dp"
android:paddingRight="16dp"
android:paddingTop="16dp"
tools:context="com.example.testapp.MainActivity">

<Button
    android:layout_width="wrap_content"
    android:layout_height="80dp"
    android:text="Forward"
    android:id="@+id/forward_btn"
    android:layout_marginTop="74dp"
    android:layout_alignParentTop="true"
    android:layout_toLeftOf="@+id/forward_right_btn"
    android:layout_toStartOf="@+id/forward_right_btn" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="80dp"
    android:text="forward&#10;LEFT"
    android:id="@+id/forward_left_btn"
    android:layout_below="@+id/forward_btn"
    android:layout_toLeftOf="@+id/forward_btn"
    android:layout_toStartOf="@+id/forward_btn" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="80dp"
    android:text="forward&#10;RIGHT"
    android:id="@+id/forward_right_btn"
    android:layout_above="@+id/reverse_btn"
```

```
    android:layout_toRightOf="@+id/reverse_btn"
    android:layout_toEndOf="@+id/reverse_btn" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="80dp"
    android:text="REVERSE"
    android:id="@+id/reverse_btn"
    android:layout_below="@+id/forward_left_btn"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="48dp" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="80dp"
    android:text="Reverse&#10;Left"
    android:id="@+id/reverse_left_btn"
    android:layout_below="@+id/forward_left_btn"
    android:layout_alignLeft="@+id/forward_left_btn"
    android:layout_alignStart="@+id/forward_left_btn"
    android:layout_centerHorizontal="true" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="80dp"
    android:text="Reverse&#10;Right"
    android:id="@+id/reverse_right_btn"
    android:layout_below="@+id/forward_right_btn"
    android:layout_toRightOf="@+id/reverse_btn"
    android:layout_alignRight="@+id/forward_right_btn"
    android:layout_alignEnd="@+id/forward_right_btn" />

<Button
    android:layout_width="wrap_content"
    android:layout_height="50dp"
    android:text="Connect to Bluetooth Device"
    android:id="@+id/bluetooth_connect_btn"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true" />
</RelativeLayout>
```

Всі текстові надписи на кнопках зберізаються в окремому файлі strings.xml, і при редагуванні надписів на кнопках за допомогою вікна властивостей вони автоматично додаються в цей файл з відповідним ідентифікатором. При XML-опису замість жорстко прописаних текстових строк використовуються посилання на відповідні ресурси. Це зручно, адже вся інформація зберігається в одному місці, і, наприклад, при необхідності зміни мови інтерфейсу програми, зміни потрібно буде внести лише в один файл – string.xml.

Тепер потрібно відкрити для редагування файл MainActivity, в якому знаходиться лістинг програми, що наведено нижче.

```
package com.example.testapp; // назва пакету, в який об'єднані класи

// підключення класів для роботи з інтерфейсом Bluetooth
import android.bluetooth.BluetoothAdapter;
```

```
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothSocket;
// підключення класу для роботи з повідомленнями
import android.content.Intent;
// підключення класу для забезпечення сумісності зі старими версіями
import android.support.v7.app.AppCompatActivity;
// підключення класу для тимчасового зберігання даних під час виконання програми
import android.os.Bundle;
// підключення класу для обробки дотиків до екрану
import android.view.MotionEvent;
// підключення базового класу для всіх елементів інтерфейсу
import android.view.View;
// підключення класу для роботи з кнопками
import android.widget.Button;
// підключення класу для відображеня повідомлень на екрані смартфону
import android.widget.Toast;
// підключення класу для обробки виключних ситуацій
import java.io.IOException;
// підключення класу для роботи з потоком виведення
import java.io.OutputStream;
// підключення класу для роботи з колекцією (набором)
import java.util.Set;
// підключення класу для генерування унікальних ідентифікаторів
import java.util.UUID;

// головний клас програми наслідується від класу AppCompatActivity
public class MainActivity extends AppCompatActivity {

// оголошення констант з інформацією про MAC адресу модуля Bluetooth
   private final String DEVICE_ADDRESS = "34:A3:25:78:43:4C";
   private final UUID PORT_UUID = UUID.fromString("00001101-0000-1000-8000-
00805f9b34fb");

// оголошення змінних для зберігання посилань на об’єкти для роботи з Bluetooth
   private BluetoothDevice device;
   private BluetoothSocket socket;
// оголошення змінної для зберігання посилань на об’єкт для роботи зпотоком
виведення    private OutputStream outputStream;

// оголошення змінних для зберігання посилань на об’єкти кнопок інтерфейсу
   Button forward_btn, forward_left_btn, forward_right_btn, reverse_btn,
reverse_left_btn, reverse_right_btn, bluetooth_connect_btn;

// оголошення строкової змінної для зберігання команди, яка буде відправлена через
// Bluetooth модуль
   String command;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
```

```
        // присвоювання змінним посилань на обєкти кнопок, що розташовані на інтерфейсі

        // кнопка керування рухом вперед
        forward_btn = (Button) findViewById(R.id.forward_btn);
        // кнопка керування рухом вперед ліворуч
        forward_left_btn = (Button) findViewById(R.id.forward_left_btn);
        // кнопка керування рухом вперед праворуч
        forward_right_btn = (Button) findViewById(R.id.forward_right_btn);
        // кнопка керування рухом назад
        reverse_btn = (Button) findViewById(R.id.reverse_btn);
        // кнопка керування рухом назад праворуч
        reverse_right_btn = (Button) findViewById(R.id.reverse_right_btn);
        // кнопка керування рухом назад ліворуч
        reverse_left_btn = (Button) findViewById(R.id.reverse_left_btn);
        // кнопка підключення по Bluetooth
        bluetooth_connect_btn = (Button) findViewById(R.id.bluetooth_connect_btn);

        // OnTouchListener обробник події натискання на кнопку руху вперед
        // (тривале натискання на кнопку)
        forward_btn.setOnTouchListener(new View.OnTouchListener() {
            @Override
            public boolean onTouch(View v, MotionEvent event) {
                if (event.getAction() == MotionEvent.ACTION_DOWN)
                // MotionEvent.ACTION_DOWN спрацьовує коли кнопка утримується
натисненою
                {
                    command = "1";          // запис в змінну значення 1 – команда руху вперед
                    try // початок блоку обробки виключних ситуацій
                    {
                        // передавання значення команди через Bluetooth модуль
                        outputStream.write(command.getBytes());
                    }
                    catch (IOException e)     // якщо виникає помилка при виконанні операції
                    // введення-виведення, викликається цей обробник виключної ситуації
                    {
                        e.printStackTrace(); // виведення повідомлення про помилку у консоль
                    }
                }
                // MotionEvent.ACTION_UP викликається якщо відбувається відпускання
кнопки
                else if(event.getAction() == MotionEvent.ACTION_UP)
                {
                    command = "10"; // запис в змінну значення 10 – зупинка платформи
                    try     // початок блоку обробки виключних ситуацій
                    {
                        // передавання значення команди через Bluetooth модуль
                        outputStream.write(command.getBytes());
                    }
                    // якщо виникає помилка при виконанні операції
                    // введення-виведення, викликається цей обробник виключної ситуації
                    catch(IOException e)
                    {
```

```
            e.printStackTrace();     // виведення повідомлення про помилку у консоль
          }

      }
      return false;        // завершення виконання функції, повертається false
    }
});

// OnTouchListener обробник натискання на кнопку руху назад
// структура цього обробника події відрізняється від попереднього (натискання на
// кнопку вперед) лише значенням, що записується в змінну command
// все інше залишається незмінним, тому коментарі не потрібні
reverse_btn.setOnTouchListener(new View.OnTouchListener(){
    @Override
    public boolean onTouch(View v, MotionEvent event)
    {
      if(event.getAction() == MotionEvent.ACTION_DOWN)
      {
        command = "2";

        try
        {
          outputStream.write(command.getBytes());
        }
        catch (IOException e)
        {
          e.printStackTrace();
        }
      }
      else if(event.getAction() == MotionEvent.ACTION_UP)
      {
        command = "10";
        try
        {
          outputStream.write(command.getBytes());
        }
        catch(IOException e)
        {
        }
      }
      return false;
    }
});

// OnTouchListener обробник натискання на кнопку руху вперед ліворуч
// структура цього обробника події відрізняється від попереднього (натискання на
// кнопку вперед) лише значенням, що записується в змінну command
// все інше залишається незмінним, тому коментарі не потрібні
forward_left_btn.setOnTouchListener(new View.OnTouchListener(){
    @Override
    public boolean onTouch(View v, MotionEvent event)
    {
```

```
        if(event.getAction() == MotionEvent.ACTION_DOWN)
        {
            command = "3";
            try
            {
                outputStream.write(command.getBytes());
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
        else if(event.getAction() == MotionEvent.ACTION_UP)
        {
            command = "10";
            try
            {
                outputStream.write(command.getBytes());
            }
            catch(IOException e)
            {
            }
        }
        return false;
    }
});

// OnTouchListener обробник натискання на кнопку руху вперед праворуч
// структура цього обробника події відрізняється від попереднього (натискання на
// кнопку вперед) лише значенням, що записується в змінну command
// все інше залишається незмінним, тому коментарі не потрібні
forward_right_btn.setOnTouchListener(new View.OnTouchListener(){
    @Override
    public boolean onTouch(View v, MotionEvent event)
    {
        if(event.getAction() == MotionEvent.ACTION_DOWN)
        {
            command = "4";
            try
            {
                outputStream.write(command.getBytes());
            }
            catch (IOException e)
            {
                e.printStackTrace();
            }
        }
        else if(event.getAction() == MotionEvent.ACTION_UP)
        {
            command = "10";
            try
            {
```

```
            outputStream.write(command.getBytes());
        }
        catch(IOException e)
        {
            e.printStackTrace();
        }
    }
    return false;
  }
});

// OnTouchListener обробник натискання на кнопку руху назад ліворуч
// структура цього обробника події відрізняється від попереднього (натискання на
// кнопку вперед) лише значенням, що записується в змінну command
// все інше залишається незмінним, тому коментарі не потрібні
reverse_left_btn.setOnTouchListener(new View.OnTouchListener(){
  @Override
  public boolean onTouch(View v, MotionEvent event)
  {
    if(event.getAction() == MotionEvent.ACTION_DOWN)
    {
      command = "5";
      try
      {
        outputStream.write(command.getBytes());
      }
      catch (IOException e)
      {
        e.printStackTrace();
      }
    }
    else if(event.getAction() == MotionEvent.ACTION_UP)
    {
      command = "10";
      try
      {
        outputStream.write(command.getBytes());
      }
      catch(IOException e)
      {
        e.printStackTrace();
      }
    }
    return false;
  }
});

// OnTouchListener обробник натискання на кнопку руху назад праворуч
// структура цього обробника події відрізняється від попереднього (натискання на
// кнопку вперед) лише значенням, що записується в змінну command
// все інше залишається незмінним, тому коментарі не потрібні
reverse_right_btn.setOnTouchListener(new View.OnTouchListener(){
```

```
        @Override
        public boolean onTouch(View v, MotionEvent event)
        {
            if(event.getAction() == MotionEvent.ACTION_DOWN)
            {
                command = "6";
                try
                {
                    outputStream.write(command.getBytes());
                }
                catch (IOException e)
                {
                    e.printStackTrace();
                }
            }
            else if(event.getAction() == MotionEvent.ACTION_UP)
            {
                command = "10";
                try
                {
                    outputStream.write(command.getBytes());
                }
                catch(IOException e)
                {
                    e.printStackTrace();
                }
            }
            return false;
        }
    });

    // обробник натискання на кнопку підключення до платформи по інтерфейсу
Bluetooth
    // структура цього обробника події відрізняється від попереднього (натискання на
    // кнопку вперед) лише значенням, що записується в змінну command
    // все інше залишається незмінним, тому коментарі не потрібні
    bluetooth_connect_btn.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            if(BTinit())
            {
                BTconnect();
            }
        }
    });
}

// функція ініціалізації Bluetooth модуля
public boolean BTinit()
{
    boolean found = false;
    BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
```

```
      if(bluetoothAdapter == null) // перевірка, чи підтримує смартфон Bluetooth
      {
         Toast.makeText(getApplicationContext(), "Device doesn't support bluetooth",
Toast.LENGTH_SHORT).show();
      }
      if(!bluetoothAdapter.isEnabled()) // перевірка, чи увімкнено Bluetooth; якщо ні,
      // програма запитує дозвіл на його вмикання
      {
         Intent enableAdapter = new
Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
         startActivityForResult(enableAdapter,0);
         try
         {
            Thread.sleep(1000);
         }
         catch(InterruptedException e)
         {
            e.printStackTrace();
         }
      }
      Set<BluetoothDevice> bondedDevices = bluetoothAdapter.getBondedDevices();
      if(bondedDevices.isEmpty()) // перевірка наявності підключення між пристроями
      {
         Toast.makeText(getApplicationContext(), "Please pair the device first",
Toast.LENGTH_SHORT).show();
      }
      else
      {
         for(BluetoothDevice iterator : bondedDevices)
         {
            if(iterator.getAddress().equals(DEVICE_ADDRESS))
            {
               device = iterator;
               found = true;
               break;
            }
         }
      }
      return found;
   }

   public boolean BTconnect()
   {
      boolean connected = true;
      try
      {
           // створення сокету для утримання з'єднання
         socket = device.createRfcommSocketToServiceRecord(PORT_UUID);
socket.connect();
           // виведення на екран смартфону повідомлення про вдале підключення по
Bluetooth
         Toast.makeText(getApplicationContext(),
```

```
            "Connection to bluetooth device successful", Toast.LENGTH_LONG).show();
        }
        catch(IOException e)
        {
            e.printStackTrace();
            connected = false; // запис в змінну значення false, що означає втрату зв'язку
        }
        if(connected) // якщо підключення активне
        {
            try
            {
                outputStream = socket.getOutputStream(); // отримання вихідного потоку
сокета
            }
            catch(IOException e)
            {
                e.printStackTrace();
            }
        }
        return connected;
    }
    @Override
    protected void onStart()
    {
        super.onStart();
    }
}
```

І останнє, що потрібно зробити перед компіляцією програми – це відредагувати файл AndroidManifest.xml, вміст якого наведено нижче.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.testapp">

    <uses-permission android:name = "android.permission.BLUETOOTH" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Після того, все зроблено, можна запустити програму на виконання. Для цього слід натиснути на кнопку Run ‘app’, на яку вказує стрілка на рис. 7.31.

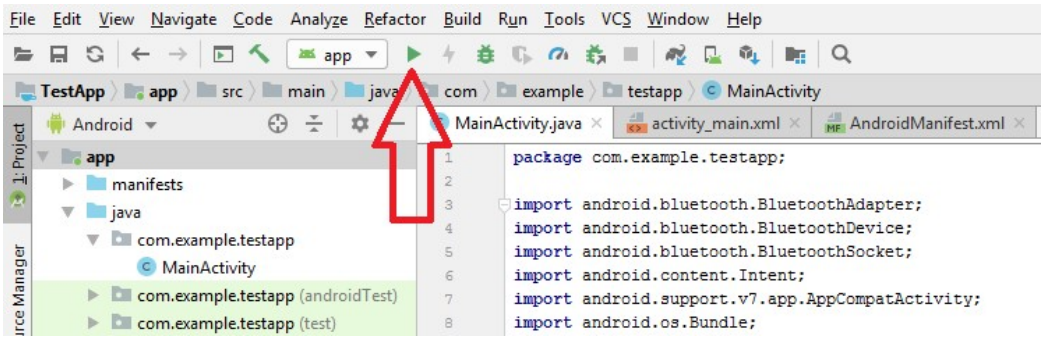


Рис. 7.31.

Після натискання на кнопку Run ‘app’ з’явиться діалогове вікно, представлене на рис. 7.32. В цьому діалоговому вікні потрібно вибрати, на якому пристрої (віртуальному чи реальному) відбудеться запуск додатку. Якщо до комп’ютера підключений смартфон, то у списку Connected Devices з’явиться його назва (за умови, що на смартфоні включений режим відлагодження). Як саме активувати режим відлагодження детально описано на сайті за посиланням:

https://remontka.pro/usb-debug-enable-android/

Отже, якщо запуск програми відбувається на реальному пристрої, то зі списку слід вибрати назву смартфона та натиснути на кнопку ОК. Через декілька секунд на екрані смартфона відкриється вікно програми. Якщо запуск додатку відбувається на віртуальному приладі, тоді спочатку потрібно створити новий віртуальний пристрій, для чого натиснути на кнопку Create New Virtual Device у лівій нижній частині діалогового вікна.

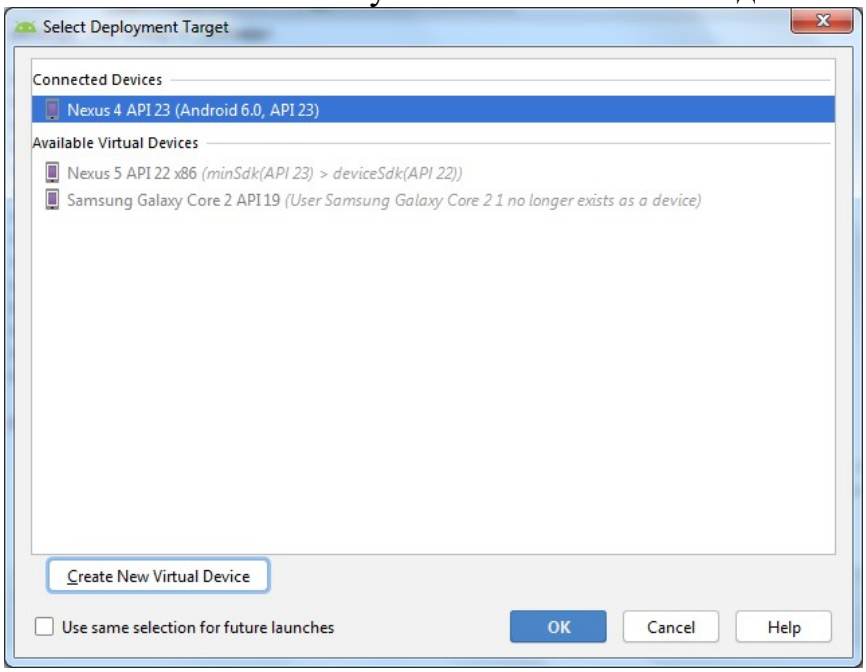


Рис. 7.32.

Далі, у діалоговому вікні, що при цьому відкриється (рис. 7.33), потрібно обрати назву смартфону або планшету, емулятор якого буде створено. В даному прикладі використовується Nexus 4. Слід звернути увагу на те, що на пристрої повинна бути встановлена операційна система Android версії, не нижче ніж 6.0 (API 23).

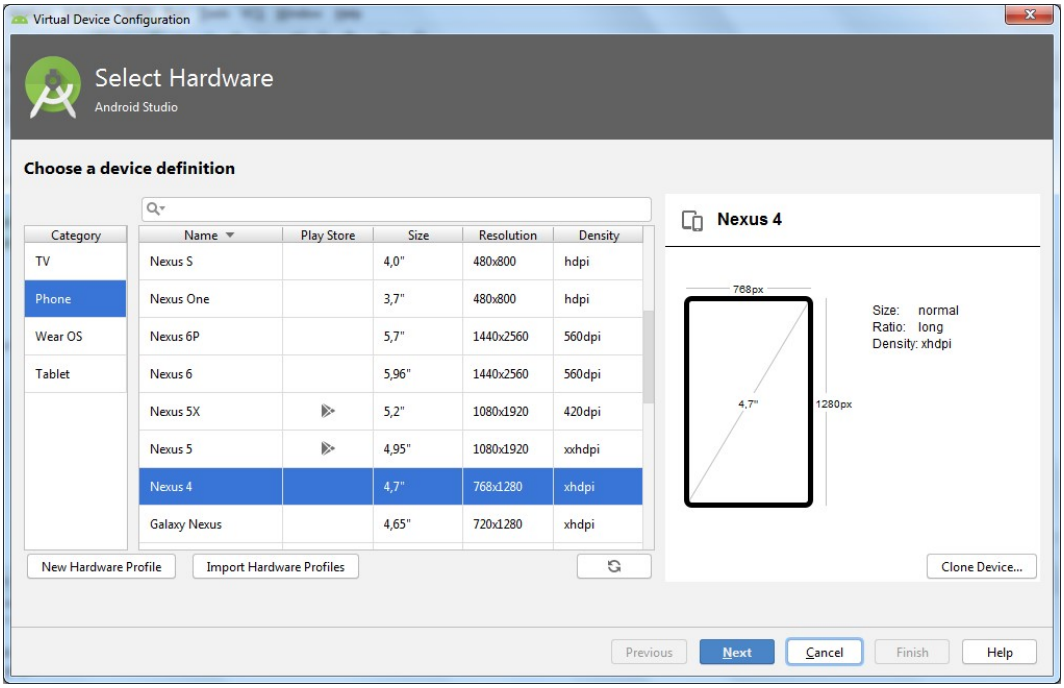


Рис. 7.33.

Створивши віртуальний пристрій, через певний час, який може складати декілька хвилин, відкриється вікно емулятора із відкритим вікном створеної програми, як показано на рис. 7.34.

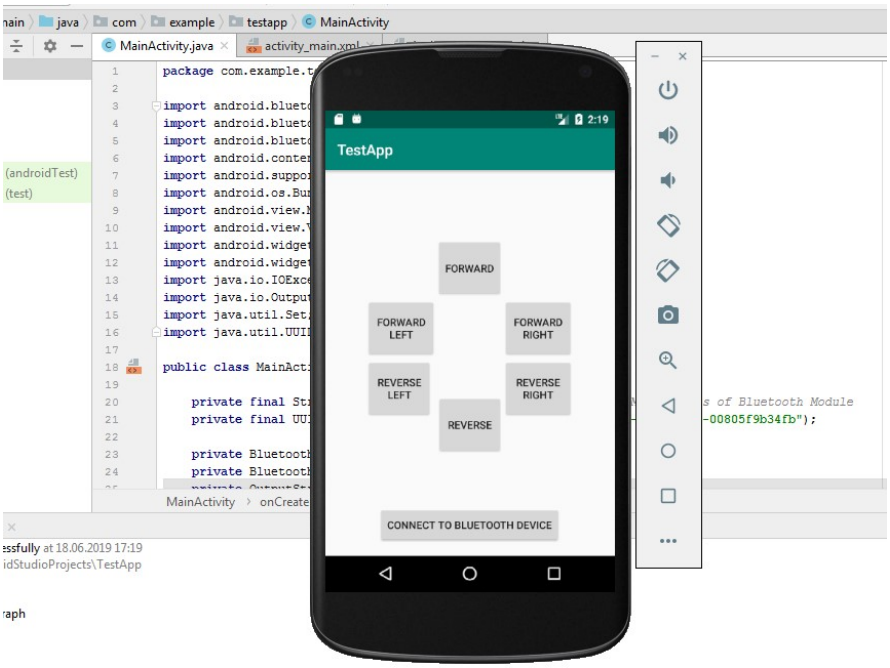


Рис. 7.34.

На цьому створення додатку для операційної системи Android для керування рухомою платформою по радіоканалу з використанням інтерфейсу Bluetooth завершено. Поглибити знання зі створення програм для смартфонів з операційною системою Android можна ознайомившись з відеоуроками за посиланням:

https://www.youtube.com/watch?v=3sHm_-nWPEI

Ці відеоуроки орієнтовані на початківців, і в них детально описується процедура створення проекту в Android Studio. Також текстові уроки з програмування для смартфонів можна знайти за посиланнями:

https://www.fandroid.info/videouroki-po-osnovam-razrabotki-android-prilozhenij-ves-spisok/

та

https://coursehunters.net/course/android-razrabotka-s-nulya-do-professionala-java-kotlin

Отже, послідовність дій для реалізації цього проекту наступна:

1. Купити контролер Arduino Uno, Bluetooth-радіомодем HC-06, драйвер двигунів L298, 4 акумулятори розміру AA.
2. Підключити Bluetooth-радіомодем до контролера Arduino у відповідності з наведеною схемою.
3. Підключити драйвер двигунів L298N Motor Driver до контролера Arduino у відповідності до наведеної схеми.
4. Розташувати всі елементи на двохколісній рухомій платформі та закріпити їх.
5. Підключити контролер Arduino до комп'ютера. При цьому важливо перевірити, щоб вивід RX контролера не був підключений, інакше неможливо буде завантажити програму в контролер.
6. Завантажити програму в контролер Arduino code.
7. Підключити вивід RX контролера Arduino після завантаження програми до модуля HC-06.
8. Завантажити додаток для Android на смартфон якщо використовується Android Studio.
9. Вставити акумулятори та увімкнути живлення. При цьому на контролері, радіомодемі та драйвері загоряться світлодіоди.
10. Відкрити додаток на смартфоні.
11. натиснути на кнопку «Connect to Bluetooth Device Device». Якщо на смартфоні Bluetooth був вимкнений, з'явиться діалогове вікно із запитом дозволу на його вмикання.
12. Після того, як Bluetooth увімкнено, потрібно знову натиснути на цю кнопку

13. Через декілька секунд світлодіоди на Bluetooth-радіомодемі перестануть блимати, що свідчитиме про успішне підключення.

14. Натискати на кнопки керування рухомою платформою та спостерігати за результатом.

Завдання до розділу 7

1. Розробити мікропроцесорну систему керування кроковим двигуном.
2. Розробити систему передавання даних між комп'ютерами з використанням радіомодемів.
3. Розробити систему передавання даних по оптичному інфрачервоному каналу.
4. Розробити багатоканальний генератор сигналів складної форми.
5. Розробити мікропроцесорну систему керування рухом гусеничного робота.
6. Розробити мікропроцесорну систему радіокерування двоколісним роботом.
7. Розробити систему дистанційного керування RGB-світлодіодною стрічкою по інфрачервоному каналу або Bluetooth.
8. Розробити систему керування освітленням приміщення за допомогою акустичних сигналів.
9. Розробити систему безконтактної радіочастотної ідентифікації (RFID).
10. Розробити систему охоронної сигналізації приміщення з передаванням інформації на мобільний телефон.

Список літератури

1. Баранов В. Н. Применение микроконтроллеров AVR. Схемы, алгоритмы, программы / В. Н. Баранов. — М. : Издательский дом «Додэка-XXI», 2004. — 288 с.
2. Голубцов М. С. Микроконтроллеры AVR: от простого к сложному / М. С. Голубцов. — М. : Солон-пресс, 2003. — 288 с.
3. Голышко А. Основа будущей сотовой связи / А. Голышко // Радио. — 2001. — №3. — С. 68-70.
4. Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel / В. В. Гребнев. — М. : ИП РадиоСофт, 2002. — 174 с.
5. Давыдов В. Visual C++. Разработка Windows-приложений с помощью MFC и API-функций / В. Давыдов. — СПб. : БХВ-Петербург, 2014. — 576 с.
6. Довбуш Г. Visual C++ на примерах / Г. Довбуш, А. Хомоненко. — СПб. : БХВ-Петербург, 2012. — 528 с.
7. Евстифеев А. В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL / А. В. Евстифеев. — М. : Издательскй дом «Додэка-XXI», 2004. — 558 с.
8. Зиборов В. MS Visual C++ 2010 в среде .NET / В. Зиборов. — Москва : Питер, 2012. — 320 с.
9. Карвинен Т. Делаем сенсоры. Проекты сенсорных устройств на базе Arduino и Raspberry Pi / Т. Карвинен, К. Карвинен, В. Валтокари. — М. : Вильямс, 2015. — 80 с.
10. Мешков А. Visual C++ и MFC / А. Мешков, Ю. Тихомиров. — СПб. : БХВ-Петербург, 2013. — 546 с.
11. Мортон Д. Микроконтроллеры AVR. Вводный курс / Д. Мортон. — М. : Издательскй дом «Додэка-XXI», 2006. — 272 с.
12. Панюкова Т. А. Языки и методы программирования. Создание простых GUI-приложений с помощью Visual C++ / Т. А. Панюкова, А. В. Панюков. — М. : Мир, 2015. — 144 с.
13. Пахомов Б. C/C++ и MS Visual C++ 2012 для начинающих / Б. Пахомов. — М. : СИНТЕГ, 2015. — 518 с.
14. Петин В. Проекты с использованием контроллера Arduino, 2-е издание / В. Петин. — СПб. : БХВ-Петербург, 2015. — 448 с.
15. Полубенцева М. C/C++. Процедурное программирование / М. Полубенцева. — СПб. : БХВ-Петербург, 2014. — 448 с.
16. Ратынский М. В. Основы сотовой связи / М. В. Ратынский, Д. Б. Зимин. Под. ред. Д. Б. Зимина. — М. : Радио и связь, 2000. — 248 с.
17. Рябенький В.М. Verilog. Практика проектування цифрових пристроїв на ПЛІС / В. М. Рябенький, О. О. Ушкаренко. — Миколаїв : Іліон, 2007. — 324 с.
18. Рябенький В. М. Компьютерное управление внешними устройствами через стандартные интерфейсы / В. М. Рябенький, В. Е. Ходаков. А. О. Ушкаренко. — Херсон : Олди-плюс, 2008. — 380 с.

19. Рябенький В. М. Інформаційне забезпечення розподілених систем керування автономними електроенергетичними установками : монографія / В. М. Рябенький, О. М. Юрченко, О. О. Ушкаренко, М. Ф. Сопель. — Київ : Інститут електродинаміки, 2012. — 208 с.
20. Рябенький В. М. Оптимізація систем керування автономними електроенергетичними установками з газодизель-генераторними агрегатами : монографія / В. М. Рябенький, О. М. Юрченко, О. О. Ушкаренко, М. Ф. Сопель. — Київ : Інститут електродинаміки, 2013. — 226 с.
21. Рябенький В. М. Схемотехніка електронних пристроїв та систем. Том 2. Прикладна теорія цифрових автоматів / В. М. Рябенький, М. Т. Фісун, О. О. Ушкаренко. — Миколаїв : Вид-во ЧДУ ім. Петра Могили, 2010. — 380 с.
22. Рябенький В. М. Схемотехніка електронних пристроїв та систем. Том 3. Мікропроцесорна техніка / В. М. Рябенький, О. О. Ушкаренко, В. С. Буряк. — Миколаїв : Іліон, 2012. — 445 с.
23. Рябенький В. М. Схемотехніка електронних пристроїв та систем. Том 6. Апаратно-програмні засоби відображення інформації / В. М. Рябенький, О. О. Ушкаренко. — Миколаїв : Іліон, 2013. — 464 с.
24. Рябенький В. М. Схемотехніка: пристрої цифрової електроніки: підручник у 2 т. [Текст] / В. М. Рябенький, В. Я. Жуйков, Ю. С. Ямненко, О. В. Борисов. — Київ : НТУУ «КПІ» Вид-во «Політехніка», 2015. — Т. 1. — 400 с.
25. Рябенький В. М. Схемотехніка: пристрої цифрової електроніки: підручник у 2 т. / В. М. Рябенький, В. Я. Жуйков, Ю. С. Ямненко, О. В. Борисов. — Київ : НТУУ «КПІ» Вид-во «Політехніка», 2015. — Т. 2. — 360 с.
26. Рябенький В. М. Программная реализация алгоритмов цифровой обработки сигналов : монография / В. М. Рябенький, А. О. Ушкаренко. — Николаев : НУК, 2016. — 244 с.
27. Рябенький В. М. Автоматизація процесів моніторингу параметрів та керування автономними електроенергетичними системами з дизель-генераторними агрегатами : монографія / В. М. Рябенький, О. І. Дорогань, О. О. Ушкаренко. — Миколаїв : НУК, 2017. — 240 с.
28. Трамперт В. AVR-RISC микроконтроллеры / В. Трамперт. — М. : МК-Пресс, 2006. — 484 с.
29. Угрюмов Е. П. Цифровая схемотехника / Е. П. Угрюмов. — СПб. : БХВ-Петербург, 2000. — 528 с.
30. Улли Соммер. Программирование микроконтроллерных плат Arduino/Freeduino. — СПб. : БХВ-Петербург, 2012. — 256 с.
31. Якименко Ю. І. Мікропроцесорна техніка / Ю. І. Якименко, Т. О. Терещенко, Є. І. Сокол, В. Я. Жуйков, Ю. С. Петергеря. — Київ : Політехніка, 2004. — 440 с.

УДК 681.5, 004.4
ББК 32.84
Р 98

Рябенький В. М.

Програмовані електронні системи керування, збору та обробки інформації : підручник / В. М. Рябенький, О. О. Ушкаренко. – Миколаїв : НУК, 2019. – 490 с.

У підручнику описуються основи проектування та програмування мікропроцесорних систем керування, збору та обробки інформації на базі 8-розрядних мікроконтролерів AVR. Розглянуті питання створення систем збору, обробки інформаці на базі апаратно-програмних засобів Arduino.

Велика увага приділяється особливостям використання вбудованих в мікроконтролер периферійних пристроїв. Детально описано роботу різних інтерфейсів та протоколів обміну даними (RS-232, I^2C, 1-Wire, SPI, USB, Ethernet) з датчиками фізичних величин та мікросхемами пам'яті. Описуються принципи отримання та обробки даних з датчиків, які найбільш часто використовуються в мікропроцесорних системах керування, збору та обробки інформації. Приводиться велика кількість прикладів програм для мікроконтролерів на мові Сі, а також прикладів програм з розвиненим графічним інтерфейсом для комп'ютера, розроблених у середовищі Visual Studio. Набір конкретних схемних та програмних рішень може бути використаний при розробці широкого спектра мікропроцесорних систем керування, збору та обробки інформації.

Підручник призначений для студентів спеціальностей 171 – «Електроніка», 172 – «Телекомунікації та радіотехніка» і суміжних спеціальностей, а також всіх тих, хто цікавиться комп'ютерною електронікою і програмуванням.

Навчальне видання

РЯБЕНЬКИЙ Володимир Михайлович,
УШКАРЕНКО Олександр Олегович

ПРОГРАМОВАНІ ЕЛЕКТРОННІ СИСТЕМИ КЕРУВАННЯ, ЗБОРУ ТА ОБРОБКИ ІНФОРМАЦІЇ

Підручник

(*українською мовою*)

Комп'ютерне складання та верстання *В. В. Москаленко*

Підписано до друку 28.12.2019 р. Формат 60×48/16. Ум. друк. Арк. 17,3. Тираж 300 прим. Зам. № 152.
Видавець і виготовник Національний університет кораблебудування
імені адмірала Макарова
просп. Героїв України, 9, м. Миколаїв, 54025
E-mail: publishing@nuos.edu.ua
Свідоцтво суб'єкта видавничої справи ДК № 2506 від 25.05.2006 р.