Міністерство освіти і науки України
Прикарпатський національний університет імені Василя Стефаника
Кафедра комп'ютерної інженерії та електроніки

Когут І.Т., Дзундза Б.С., Грига В.М., Голота В.І.

Лабораторний практикм
з дисциплін «Технології проектування комп’ютерних систем», «Дослідження і програмування пристроїв зв’язку з об’єктом», “Компютерна схемотехніка”

навчально-методичний посібник для студентів спеціальності «Комп’ютерна інженерія», «Електроніка»

м. Івано-Франківськ
«НАІР»
2022

УДК 614.8(075.8)
Б-39

Затверджено
на засіданні кафедри комп'ютерної інженерії та електроніки
протокол № 1 від 26 серпня 2020 року
Рекомендовано до друку
Вченою радою фізико-технічного факультету
протокол № 1 від 28 серпня 2020 року

Рецензенти:
Петришин Михайло Любомирович – кандидат технічних наук, доцент кафедри комп'ютерних наук та інформаційних систем ДВНЗ "Прикарпатський національний університет імені Василя Стефаника",
Терлецький Андрій Іванович – кандидат фізико-математичних наук, доцент кафедри комп'ютерної інженерії та електроніки ДВНЗ "Прикарпатський національний університет імені Василя Стефаника".

Б-39

Лабораторний практикм з дисциплін «Технології проектування комп'ютерних систем», «Дослідження і програмування пристроїв зв'язку з об'єктом», "Компютерна схемотехніка": навчально-методичний посібник для студентів спеціальності «Комп'ютерна інженерія», «Електроніка» / [Укаладачі: Когут І.Т., Дзундза Б.С., Грига В.М., Голота В.І.] – Івано-Франківськ: НАІР, 2022. – 149 с.

У навчально-методичному посібнику наведено методичні рекомендації до лабораторних робіт з дисциплін «Програмування периферійних пристроїв», «Дослідження і програмування пристроїв зв'язку з об'єктом», викладено короткі теоретичні відомості з питань архітектури і особливостей використання мікроконтролерів типу AVR, розглянуто роботу периферійних пристроїв. В якості середовища програмування використано Arduino IDE. Матеріал посібника дає можливість студентам сформувати практичні вміння і навички для проектування і програмування електронних пристроїв на 8 розрядних мікроконтролерах.

Для студентів вищих навчальних закладів ОР "Бакалавр", "Магістр".

ЗМІСТ

ВСТУП……..…….	5
Лабораторна робота № 1. Плати Arduino, живлення, логічні рівні сигналів, дискретні входи, система тактування мікроконтролера. Середовище розробки програм…….	6
Лабораторна робота № 2. Опитування дискретних входів. Управління за допомогою кнопок. Проектування підсистеми визначення рухомих об’єктів розумного будинку на основі сенсора руху HC-SR501.	21
Лабораторна робота № 3. Підсилення дискретних виходів. Керування індуктивним навантаженням. Електрозамок на основі актюатора.	29
Лабораторна робота № 4. Таймери. Переривання. Вимірювання інтервалу часу між двома вхідними імпульсами. Послідовна передача даних. UART.	35
Лабораторна робота № 5. Широтно-імпульсна модуляція. Формування кольору за допомогою RGB-світлодіода. Керування сервоприводом.	42
Лабораторна робота № 6. Робота з I2C шиною передачі даних. Під’єднання рідкокристалічного символьного дисплея через I2C інтерфейс. Годинник реального часу.	47
Лабораторна робота № 7. Однопровідна двонапрямлена шина передачі даних 1-Wire. Побудова цифрового термометра з використанням символьного рідкокристалічного дисплея LCD1602.	57
Лабораторна робота № 8. Робота із сенсором температури та вологості DHT22 і програмування протоколу обмуну даних з ним в середовищі Arduino IDE.	67
Лабораторна робота № 9. АЦП. Введення аналогових сигналів в мікроконтролер. Проектування підсистеми пожежної сигналізації розумного будинку з використанням аналогового давача газу MQ-2.	73
Лабораторна робота № 10. Проектування підсистеми визначення рухомих об’єктів розумного будинку з використанням ультразвукового давача відстані HC-SR04.	80
Лабораторна робота № 11. Вимірювання електричного опору. Проектування підсистеми визначення вологи чи протікання води з використанням давача води FC-37.	85
Лабораторна робота № 12. Безпровідна передача даних. Проектування підсистеми ідентифікації об’єктів доступу до приміщення з використанням модуля радіочастотної ідентифікації RC522.	89

Лабораторна робота № 13-14. Безпровідна передача даних через канали зв’язку з використанням GSM модуля SIM800.	97
Лабораторна робота № 15. Отримання точного часу та координат з GPS супутників з використанням GSM модуля NEO-6M.	136
Література	148
Рекомендації до оформлення звітів виконаних лабораторних робіт	

ВСТУП

Мікроконтролер (англ. microcontroller), або цифрова однокристальна мікро-ЕОМ, виконана у вигляді мікросхеми, що включає мікропроцесор, блоки пам'яті для збереження коду програм і даних, порти введення-виведення і блоки зі спеціальними функціями (лічильники, компаратори, АЦП тощо). Використовується для керування електронними пристроями. Посуті, це комп'ютер, здатний виконувати прості завдання. Використання однієї мікросхеми значно знижує розміри, енергоспоживання і вартість пристроїв, побудованих на базі мікроконтролерів.

Одним з найпопулярніших сімейств мікроконтролерів у світі є AVR – сімейство восьмибітових мікроконтролерів фірми Atmel, що і розглядаються у даному посібнику. Мікроконтролери AVR мають Гарвардську архітектуру (програма і дані знаходяться в різних адресних просторах) і систему команд, близьку до ідеології RISC. Архітектура AVR дозволяє застосовувати операційні системи при розробленні приладів, основними з яких є FREERTOS і uOS. У навчальних цілях зручно використовувати макетні плати Arduino які сумісні з великою кількістю давачів та зовнішніх пристроїв, мають зручне об’єктно-орієнтоване середовище розробки програм та хорошу підтримку.

У лабораторному практикумі розглянуто особливості програмування та моделювання роботи електричних схем з використанням мікроконтролерів AVR і середовища розробки Arduino IDE. Сімейства таких мікроконтролерів на сьогодні широко використовуються в побутових приладах, системах автоматизації та інформаційно-вимірювальній техніці. Тому актуальним завданням є впровадження і вивчення їх в програмі вищої школи, як в теоретичному, так і в практичному аспектах, особливо при підготовці молодших спеціалістів за спеціальністю “Компютерна інженерія”, “Електроніка”.

Перед виконанням лабораторних робіт потрібно ознайомитись з матеріалом, викладеним в даному лабораторному практикумі та переліком літературних джерел, які наведені в ньому. До виконання лабораторних робіт допускаються студенти, які виявили необхідну ступінь теоретичної підготовки. Захист проводиться на основі оформлених звітів до лабораторних робіт в обсязі, зазначеному в лабораторному практикумі та згідно зразка. Звіт з лабораторної роботи готується індивідуально кожним студентом. До виконання лабораторних робіт потрібно мати документацію на мікроконтролер ATMega328, встановлене програмне середовище Arduino IDE.

Лабораторна робота №1

Плати Arduino, живлення, логічні рівні сигналів, дискретні входи, система тактування мікроконтролера. Середовище розробки програм.

Мета роботи: вивчення структури AVR мікроконтролера, а також набуття початкових навичків програмування мікроконтролерів в середовищі Arduino IDE.

Теоретичні відомості

1.1. Апаратні засоби.

Почнемо ми з представника цієї лінійки AVR мікроконтролерів, як Atmega 328.

ATMEGA328

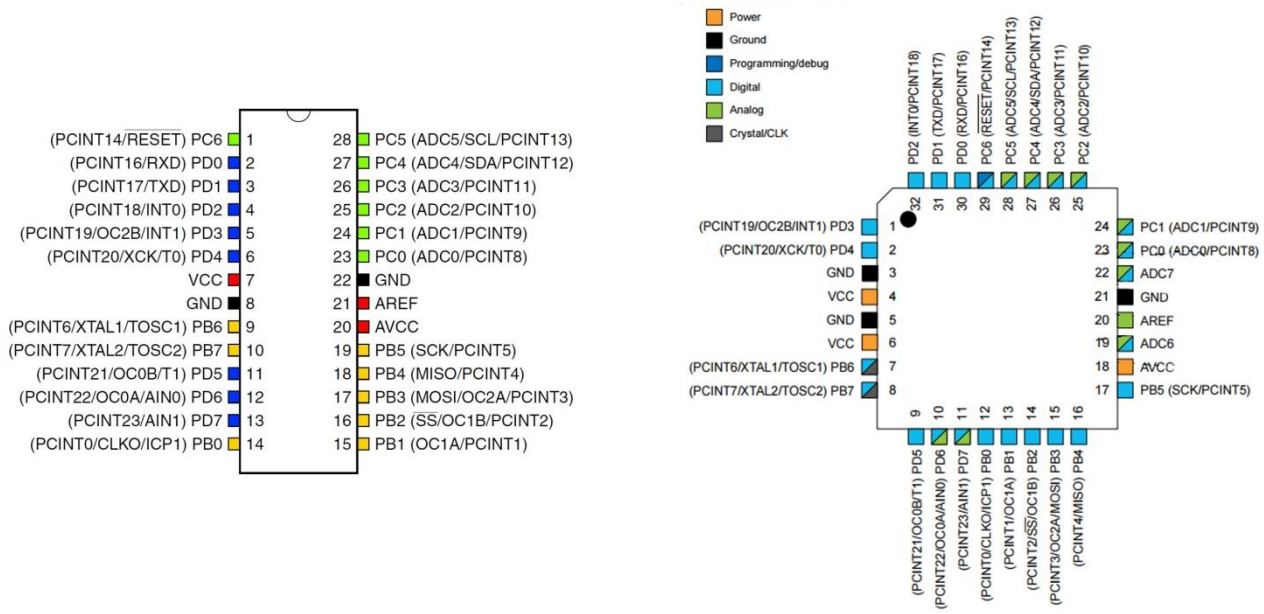


Рис. 1.1 – Функції виводів мікроконтролера ATMEGA328.

Мікроконтролери представленої моделі включають три порти введення-виведення. Порти - це такі шини (лінії) даних, які працюють в двох напрямках - і на вивід даних, і на введення даних, тобто є двонаправленими.

Порт B неповний, має 6 виводів. Порт C також є неповним, від нульової до шостої ніжки. А порт D є повноправним портом, має всі 8 виводів, тобто байт даних від 0 до 7.

Живиться мікроконтролер від 5 Вольт. Можна живити його також від 3,3 Вольта, він також буде відмінно працювати. Але максимальна частота в цьому випадку становитиме тільки 8 МГц. Тактову частоту 16 МГц можна використати тільки при живленні напругою 5 В.

Arduino, як і інші аналогічні засоби (Raspberry, Netmedia'sBX-24, Phidgets, MIT'sHandyboardi та ін.), мають на меті звільнити користувача від необхідності заглиблюватися в деталі внутрішньої будови мікроконтролерів, надавши йому простий і зручний інтерфейс для їх програмування. Плата Arduino Uno (Nano) містить мікроконтролер фірми Atmel AVR (ATmega328 або ATmega168) та елементну обв'язку для програмування і інтеграції з іншими схемами. На кожній платі обов'язково присутні: лінійний стабілізатор напруги 5В і 16МГц кварцевий генератор (в деяких версіях керамічний резонатор). У мікроконтролер попередньо прошитий завантажувач, тому зовнішній програматор не потрібен. Плати Arduino дозволяють використовувати більшу частину I/O виводів мікроконтролера в зовнішніх схемах.

Плати можна розділити на контролери, шілди і аксесуари. Контролери-це найважливіша частина. Контролер - це плата, яка містить мікроконтролер і в яку записується виконувана програма. Шілди - це плати розширення, які містять ту чи іншу периферію, керовану контролером.

Arduino Nano, Arduino UNO

Arduino UNO(Nano) - це повнофункціональний пристрій на базі мікроконтролера ATmega328, адаптований для використання з макетної платі. Arduino Nano розроблено і випускається компанією Gravitech. Характеристики:

•Мікроконтролер:Atmel ATmega328 (ATmega168)

•Робоча напруга (логічний рівень): 5В

•Напруга живлення (рекомендована):7-12В

•Цифрові входи/виходи: 14 (6 з яких можуть використовуватися як ШІМ-виходи)

•Аналогові входи: 8

•Максимальний струм одного виводу: 20 мА

•Flash-пам'ять: 32 КБ (ATmega328) з яких 2 КБ використовуються завантажувачем

•SRAM: 2 КБ (ATmega328)

•EEPROM: 1 КБ (ATmega328)•

Тактова частота: 16 МГц

Arduino може живитися через кабель USB.

Крім основних, деякі виводи Arduino можуть виконувати додаткові функції: Послідовний інтерфейс: виводи 0 (RX) і 1 (TX). Використовуються для отримання (RX) і передачі (TX) даних по послідовному інтерфейсу. Ці виводи з'єднані з відповідними виводами мікросхеми-перетворювача USB-UART від FTDI.

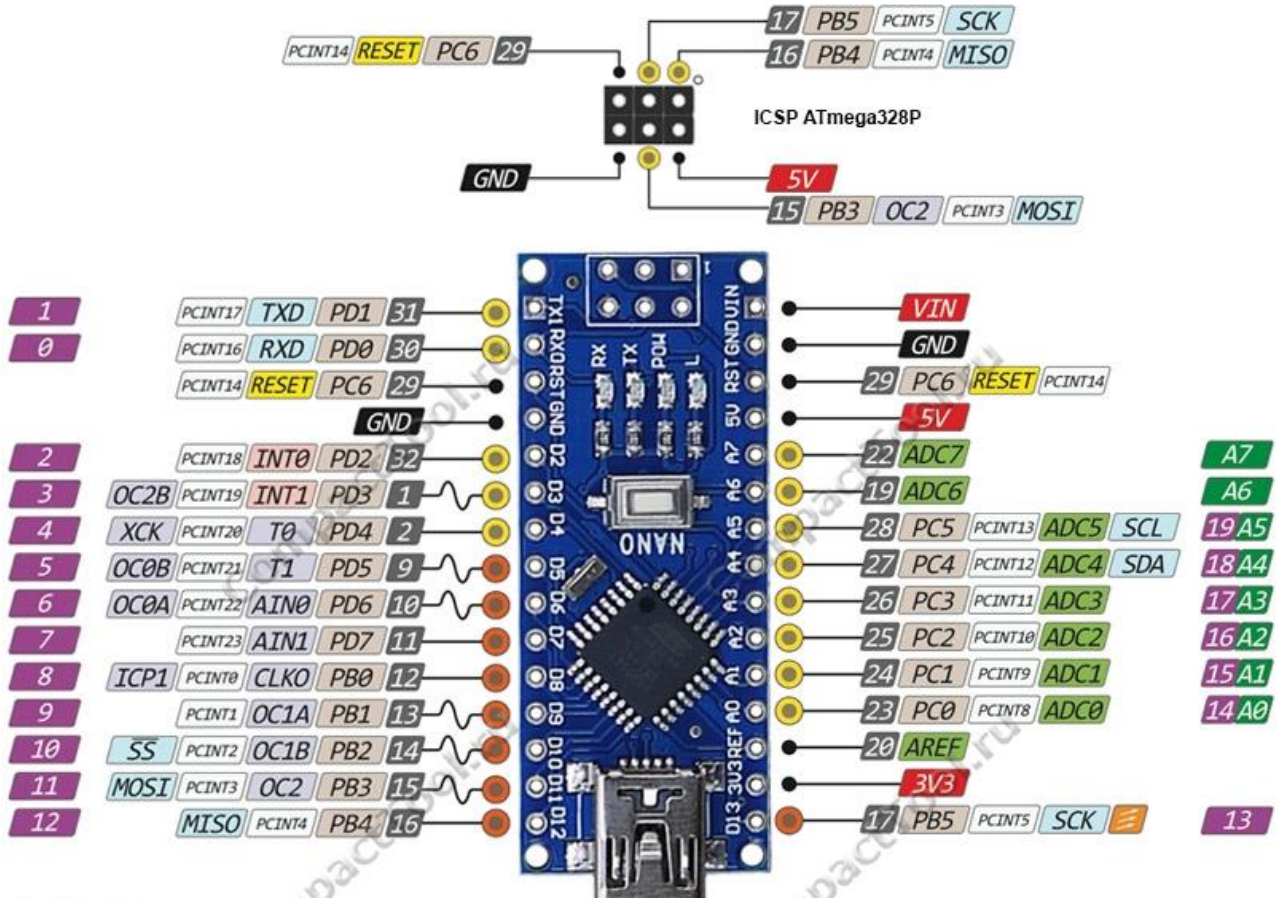


Рис. 1.2 –Плата ArduinoNano.

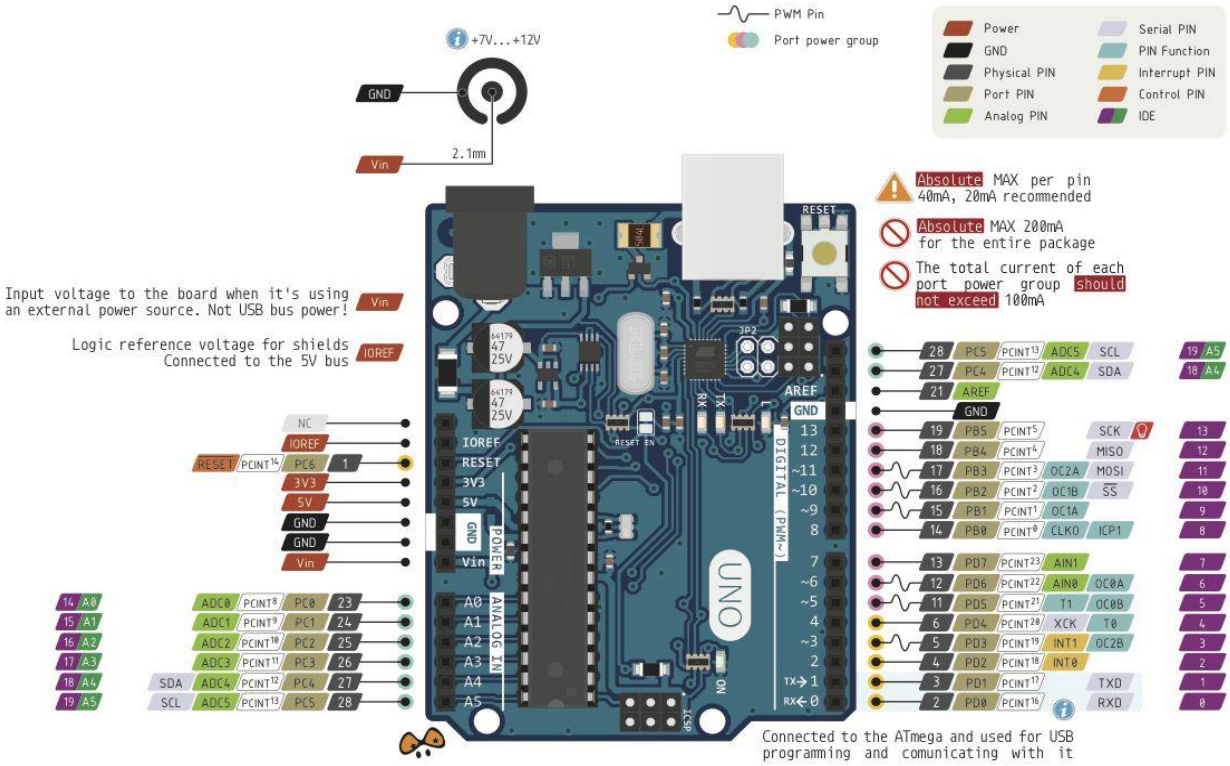


Рис. 1.3 –Плата Arduino UNO.

У якості зовнішнього джерела живлення (не USB) може використовуватися мережевий AC/DC-адаптер або акумулятор/батарея. Штекер адаптера (діаметр - 2.1мм, центральний контакт - позитивний) необхідно

вставити у відповідний роз'єм живлення на платі. У разі живлення від акумулятора/батареї, її провід необхідно під'єднати до виводів GND і Vin роз'єму POWER. Рекомендується використовувати джерело живлення з напругою в діапазоні від 7В до 12В.

Виводи живлення, що розташовані на платі:

VIN. Напруга, що надходить в Arduino безпосередньо від зовнішнього джерела живлення (не пов'язане з 5В від USB або іншою стабілізованою напругою). Через цей вивід можна подавати як зовнішнє живлення, так і споживати струм, коли пристрій живиться від зовнішнього адаптера.

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

3V3. 3.3В, що надходять від стабілізатора напруги на платі. Максимальний струм, споживаний від цього виводу, становить 50 мА.

GND. Загальний мінусовий вивід.

IOREF. Цей вивід надає платам розширення інформацію про робочу напругу мікроконтролера Arduino. Залежно від напруги, зчитуваної з виводу IOREF, плата розширення може переключитися на відповідне джерело живлення або задіювати перетворювачі рівнів, що дозволить їй працювати як з 5В, так і з 3.3В-пристроями.

Пам'ять

Обсяг флеш-пам'яті ATmega328 становить 32 КБ (з яких 0.5 КБ використовуються завантажувачем). Мікроконтролер також має 2 КБ пам'яті SRAM і 1 КБ EEPROM (з якої можна зчитувати або записувати інформацію за допомогою бібліотеки EEPROM).

Входи і виходи

З використанням функцій *pinMode()*, *digitalWrite()* і *digitalRead()* кожен з 14 цифрових виводів може працювати в якості входу або виходу. Рівень напруги на виводах обмежений 5В. Максимальний струм, який може віддавати або споживати один вивід, становить 40 мА. Усі виводи пов'язані з внутрішніми підтягуючими резисторами (за замовчуванням відключеними) номіналом 20-50 кОм. Крім цього, деякі виводи Arduino можуть виконувати додаткові функції:

Послідовний інтерфейс: виводи 0 (RX) і 1 (TX). Використовуються для отримання (RX) і передачі (TX) даних по послідовному інтерфейсу. Ці виводи з'єднані з відповідними виводами мікросхеми перетворювача USB-UART.

Зовнішні переривання: виводи 2 і 3. Можуть служити джерелами переривань, що виникають при фронті, спаді або при низькому рівні сигналу на

цих виводах. Для отримання додаткової інформації див. Функцію attachInterrupt().

ШІМ: виводи 3, 5, 6, 9, 10 і 11. За допомогою функції analogWrite() можуть виводити 8-бітові аналогові значення в вигляді ШІМ-сигналу.

Інтерфейс SPI: виводи 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Із застосуванням бібліотеки SPI дані виводи можуть здійснювати зв'язок по інтерфейсу SPI.

Світлодіод: Вбудований світлодіод, приєднаний до виводу 13. При відправці значення HIGH світлодіод включається, при відправці LOW - вимикається.

В Arduino Uno є 6 (8) аналогових входів (A0 - A5), кожен з яких може отримувати аналогову напругу у вигляді 10-бітного числа (1024 різних значень). За замовчуванням, вимір напруги здійснюється в діапазоні від 0 до 5В. Проте, верхню межу цього діапазону можна змінити, використовуючи вивід AREF і функцію *analogReference()*.

TWI: вивід A4 або SDA і вивід A5 або SCL. З використанням бібліотеки Wire дані виводи можуть здійснювати зв'язок по інтерфейсу TWI.

AREF. Опорна напруга для аналогових входів. Може бути задіяна функцією *analogReference ()*.

Reset. Формування низького рівня (LOW) на цьому виводі призведе до перезавантаження мікроконтролера. Зазвичай цей вивід служить для функціонування кнопки скидання на платах розширення.

Зв'язок з комп’ютером

Arduino Uno надає ряд можливостей для здійснення зв'язку з комп'ютером, ще одним Arduino або іншими микроконтролерами. У ATmega328 є універсальний асинхронний приймач-передавач (UART), що дозволяє здійснювати послідовну передачу даних за допомогою цифрових виводів 0 (RX) і 1 (TX) та перетворювача USB-UART, який при підключенні до ПК, дозволяє Arduino включитися, як віртуальний COM-порт. При передачі даних через мікросхему-перетворювач USB-UART під час USB-з'єднання з комп'ютером, на платі будуть блимати світлодіоди RX і TX. Бібліотека SoftwareSerial дозволяє реалізувати послідовний зв'язок на будь-яких цифрових виводах Arduino.

У мікроконтролері ATmega328 також реалізована підтримка послідовних інтерфейсів I2C (TWI) і SPI. У програмне забезпечення Arduino входить бібліотека Wire, що дозволяє спростити роботу з шиною I2C. Для роботи з інтерфейсом SPI використовують бібліотеку SPI.

1.2. Середовище розробки Arduino

Інтегроване середовище розробки Arduino IDE – це кросплатформовий додаток на Java, що включає в себе редактор коду, компілятор і модуль передачі прошивки в плату.

Опис Arduino IDE

Середовище розробки Arduino (рис.1.4) складається із вбудованого текстового редактора програмного коду, області повідомлень, вікна виведення тексту (консолі), панелі інструментів з кнопками часто використовуваних команд і декількох меню. Для завантаження програм і зв'язку середовище розробки підключається до апаратної частини Arduino.

Програма для Arduino (скетч) пишеться в текстовому редакторі. Під час збереження і експорту проекту в області повідомлень з'являються пояснення, також можуть відображатися повідомлення про помилки.

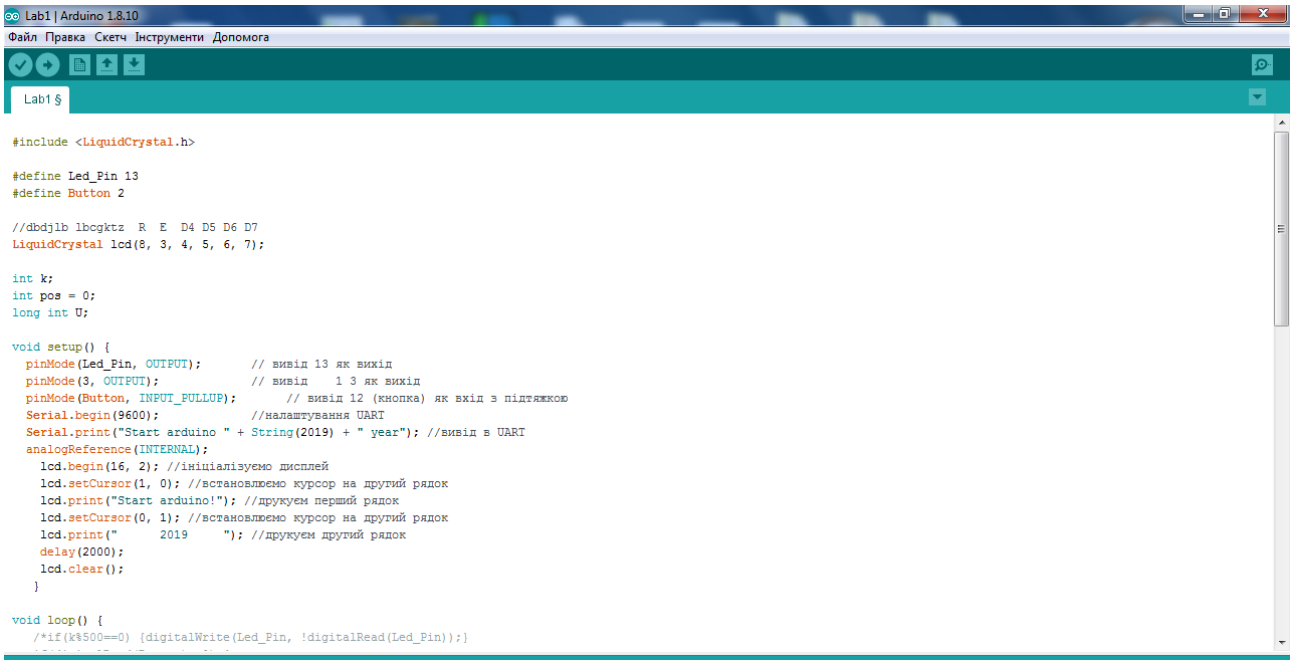


Рис.1.4. Загальний вигляд середовища Arduino.

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>Sketchbook або кнопкою Open на панелі інструментів.

Файли коду можуть бути стандартними Arduino (без розширення), файлами C (розширення *.c.), файлами C ++ (*.cpp) або файлами заголовків (.h).

Завантаження скетчу в Arduino

Після вибору порту і платформи необхідно натиснути кнопку завантаження на панелі інструментів або вибрати пункт меню File>Upload to I/O Board. Середовище розробки Arduino виведе повідомлення про завершення 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 із скетчів в мікроконтролер. Робота завантажувача (Bootloader) розпізнається по миготінню світлодіода (13 вивід, наприклад, при перезавантаженні плати).

Бібліотеки

Бібліотеки додають додаткову функціональність скетчам, наприклад, при роботі з апаратною частиною або при обробці даних. Для використання бібліотеки необхідно вибрати меню Sketch>Include Library. Можна вибрати бібліотеки зі списку або завантажити через Library Manager (рис.1.5).

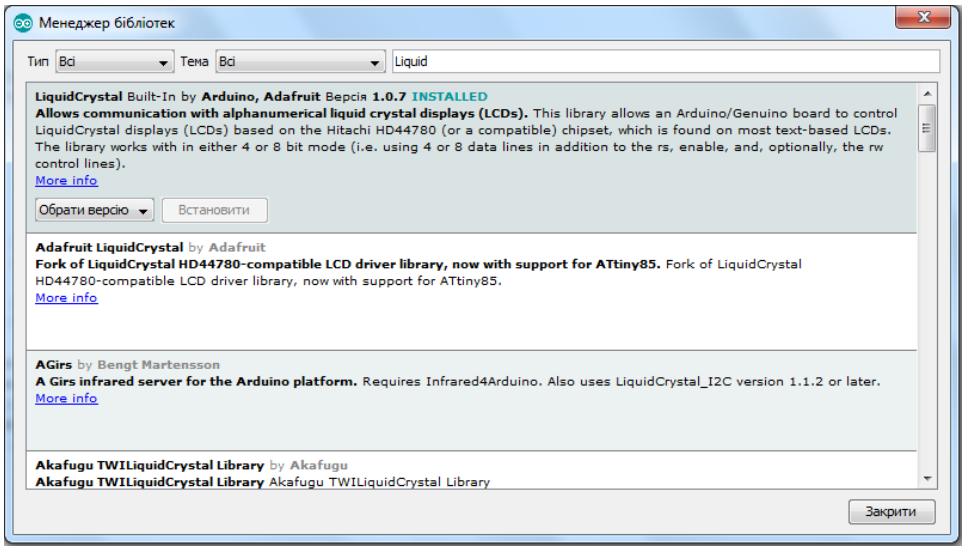


Рис. 1.5. Менеджер завантаження бібліотек.

Завантаження бібліотек вимагає додаткового місця в пам'яті Arduino. Невикористані бібліотеки можна видалити з скетчу прибравши директиву *#include*.

Деякі бібліотеки включені в середовище розробки Arduino. Інші можуть бути завантажені з різних ресурсів. Для встановлення викачаних бібліотек необхідно створити директорію «libraries» в папці проекту і розпакувати архів. Наприклад, для встановлення бібліотеки DateTime її файли повинні знаходиться в папці / libraries / DateTime папки проекту.

1.3. Мова програмування для Arduino

Середовище розробки засновано на мові програмування Processing і спроектовано для програмування новачками, які не знайомі близько з розробкою програмного забезпечення. Умовно кажучи, це C/C++, доповнений деякими бібліотеками. Програми обробляються за допомогою препроцесора, а потім компілюються за допомогою AVR-GCC.

Особливості мови Arduino

Мова Arduino має чотири складових:

- операторі,
- дані,
- функції,
- бібліотеки.

Операторі:

- setup (),
- loop (),
- оператори мови С.

Дані: типи даних з мови С.

Функції:

- цифрове введення / виведення,
- аналогове введення / виведення,
- час,
- математичні обчислення,
- тригонометрія,
- випадкові числа,
- біти і байти,
- зовнішні переривання,
- переривання.

Бібліотеки:

- EEPROM,
- SD,
- SPI,
- SoftwareSerial,
- Wire,
- допоміжні класи,
  - клас Serial,
  - клас Stream.

Оголошення змінної

Оголошення змінної відбувається таким чином: спочатку вказується тип даних для цієї змінної а потім назва цієї змінної. Оператор присвоювання (=) – не є знаком рівності і не може використовуватися для порівняння значень. Оператор рівності записується як «подвійне рівне» - ==. Присвоєння використовується для збереження певного значення в змінній. Наприклад, запис виду a = 10 задає змінній a значення числа 10.

Цикли

Якщо ми знаємо точну кількість дій (ітерацій) циклу, то можемо використовувати цикл *for*. Синтаксис його виглядає приблизно так:

for (дія до початку циклу; умова продовження циклу; дія в кінці кожної ітерації циклу)

{інструкція циклу;

інструкція циклу 2;

інструкція циклу N;

}

Ітерацією циклу називається один прохід цього циклу.

Коли ми не знаємо, скільки ітерацій повинен зробити цикл, нам знадобиться цикл *while* або *do ... while*. Синтаксис циклу while виглядає наступним чином:

while (Умова)

{Тіло циклу;}

Даний цикл буде виконуватися, поки умова, вказана в круглих дужках є істиною.

Конструкція розгалуження

Оператор if служить для того, щоб виконати будь-яку операцію в тому випадку, коли умова є вірною. Умовна конструкція завжди записується в круглих дужках після оператора if. Усередині фігурних дужок вказується тіло умови. Якщо умова виконається, то почнеться виконання всіх команд, які знаходяться між фігурними дужками.

Оператор else. Кожному оператору if відповідає тільки один оператор *else*. Сукупність цих операторів - else if означає, що якщо не виконалася попередня умова, то перевірити дану. Якщо жодна з умов не є вірною, то виконується тіло оператора else.

Оператори порівняння

== (дорівнює)

! = (Не дорівнює)

<(Менше ніж)

> (Більше ніж)

<= (Менше або дорівнює)

> = (Більше або дорівнює)

Логічні оператори

&& (І)

| (АБО)

! (НЕ)

Бітові оператори

& (Побітовое І)

| (Побітовое АБО)

^ (Побітовое XOR або виключаюче АБО)

~ (Побітовое НЕ)

<< (побітовий зсув вліво)

>> (побітовий зсув вправо)

Складні оператори

++ (інкремент)

-- (декремент)

+ = (Складене додавання)

- = (складене віднімання)

* = (Складене множення)

/ = (Складене ділення)

& = (Складене побітовое І)

| = (Складене побітовое АБО)

Функції

Будь-яка функція має тип, як і будь-яка змінна. Функція може повертати значення, тип якого аналогічний типу самої функції. Якщо функція не повертає ніякого значення, то вона повинна мати тип *void* (такі функції іноді називають процедурами). При оголошенні функції, після її типу має стояти ім'я функції і дві круглі дужки - відкриваюча і закриваюча, всередині яких можуть знаходитися один або кілька аргументів функції, яких також може не бути взагалі. Після списку аргументів функції ставиться відкриваюча фігурна дужка, після якої знаходиться саме тіло функції. В кінці тіла функції обов'язково ставиться фігурна дужка, що закриває його.

Функції setup (), loop ()

Під час виклику функції *setup()*, програма ініціалізується і встановлює початкові значення. Функція *setup*() викликається, коли стартує мікроконтролер.

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.

Функція *loop* () забезпечує нескінченний робочий цикл програми. У циклі виконується опитування стану виводів, зміна їх стану, прийомпередача даних робота з АЦП та ін.

Приклад:

```
int buttonPin = 3;
    void setup()
    { // put your setup code here, to run once:
    }
void loop()
    {
// ...
    }
```

Типи даних Arduino

void – ключове слово void використовується тільки при оголошенні функцій. Воно вказує на те, що оголошувана функція не повертає ніякого значення.

boolean – змінні типу boolean можуть приймати одне з двох значень: true або false. Кожна змінна типу boolean займає в пам'яті один байт.

char – тип даних, який займає в пам'яті 1 байт і зберігає символьне значення. Символи пишуться в одинарних лапках, наприклад: 'A' (сукупність символів - рядки - пишуться у подвійних лапках: "ABC").

unsigned char – Те ж саме, що і тип даних *byte*. Беззнаковий тип даних, що займає в пам'яті 1 байт.

int – цілочисельний тип даних. Це основний тип даних для зберігання чисел.

В Arduino Uno (і інших платах на базі 8-ти бітних мікроконтролерів) змінні типу int зберігають 16-бітові (2-байтові) значення. Така розмірність дає діапазон від -32768 до 32767.

unsigned int – в 8-ми бітних мікроконтролерів, змінні типу *unsigned int* (беззнакові цілі) містять двобайтові значення. Відмінність полягає в тому, що замість негативних чисел вони можуть зберігати лише позитивні значення в зручному діапазоні від 0 до 65535.

long – змінні типу long володіють розширеним розміром для зберігання чисел і мають розмірність 32 біта (4 байта), що дозволяє їм зберігати числа в діапазоні від -2 147 483 648 до 2 147 483 647.

unsigned long – мають розмірність 32 біта (4 байта). Змінні типу *unsigned long*, на відміну від звичайного long, зберігають тільки позитивні числа в діапазоні від 0 до 4 294 967 295.

short – це 16-бітний тип даних.

float – тип даних для чисел з плаваючою точкою. Числа з плаваючою точкою часто використовуються для подання аналогових або безперервних величин, оскільки дають можливість окреслити їх більш точно, ніж цілі числа. Числа з плаваючою точкою мають 32 біта (4 байта) інформації і можуть досягати величезних значень від -3.4028235E + 38 до 3.4028235E + 38. Точність дрібних чисел типу *float* становить 6-7 десяткових знаків. Мається на увазі загальна кількість цифр, а не кількість знаків після коми.

double – для 8-ми бітних мікроконтролерів змінні типу *double* займають 4 байта. Аналогічні змінним float.

Створення (оголошення) *массиву*.

Масив – це область пам'яті, де можуть послідовно зберігатися кілька значень.

```
int myInts[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[6] = {2, 4, -8, 3, 2};
char message[6] = "hello";
```

string – текстовий рядок. Може бути оголошений двома способами: можна використовувати тип даних String, який входить в ядро, починаючи з версії 0019; або оголосити рядок як *масив символів char* з нульовим символом в кінці.

```
char Str1 [15];
char Str2 [8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3 [8] = { 'a', 'r', 'd', 'u', 'i', 'n', 'o', '\ 0'};
char Str4 [] = "arduino";
char Str5 [8] = "arduino";
char Str6 [15] = "arduino".
```

Рядки завжди оголошуються в подвійних лапках ("Abc"), а символи завжди оголошуються в одинарних лапках ('A').

Цифрове введення / виведення

За замовчуванням всі порти Arduino визначаються як входи, і немає потреби описувати це в коді. Порти зазвичай прописуються в функції ініціалізації змінних.

Ініціалізація порту вводу-виводу *Arduino*:

***1. pinMode* (pin, mode)**

Параметри: pin: номер виводу, режим роботи якого задається, mode: приймає значення: INPUT – вхід, в цьому режимі відбувається зчитування даних з датчиків, стану кнопок, аналогового і цифрового сигналу.

Порт знаходиться в так званому високоімпедансному стані, тобто на вході високий опір. OUTPUT – вихід, залежно від команди прописаної в коді, порт

приймає значення одиниці або нуля. Вихід стає свого роду керованим джерелом живлення і видає максимальний струм (20 мА) в навантаження, що до нього підключене. INPUT_PULLUP – порт працює як вхід, але до нього підключається та званий підтягуючий резистор з номіналом 20 – 50 кОм.

Значення, що повертаються – немає.

***2.* digitalWrite (pin, value)**

Параметри: pin: номер виводу, value: значення HIGH або LOW. Значення, що повертаються – немає.

***3.* digitalRead (pin)**

Параметри**:** pin: номер цифрового виводу, з якого необхідно вважати значення (int). Значення, що повертаються HIGH або LOW.

Функція паузи delay (ms)

Параметри: ms - кількість мілісекунд, на які необхідно призупинити програму.Значення, що повертаються – немає.Опис: припиняє виконання програми на вказаний проміжок часу (в мілісекундах).

Приклад найпростішої програми – мигаємо світлодіодом.

```
void setup() {
  //ініціалізуємо 13 порт
  //до якого підключено світло діод на вихід
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite(13, HIGH);   // LED on
  delay(1000);              // павза 1 с
  digitalWrite(13, LOW);    // LED off
  delay(1000);              // павза 1 с
}
```

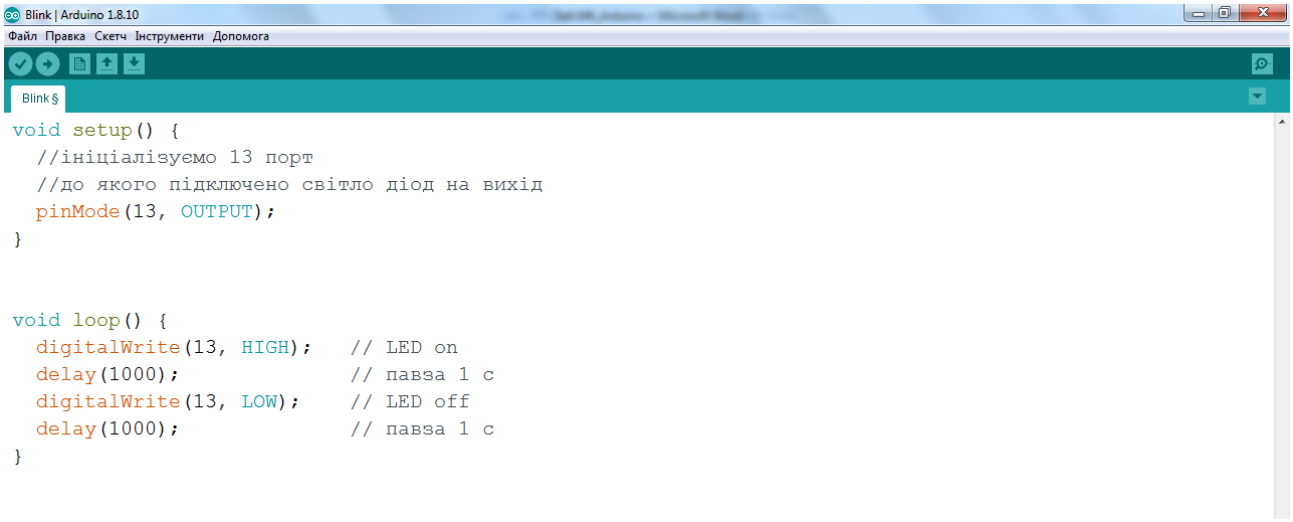
Практичне завдання до лабораторної роботи

Реалізувати алгоритм мигання світлодіодом який би не заважав мікроконтролеру виконувати інші функції.

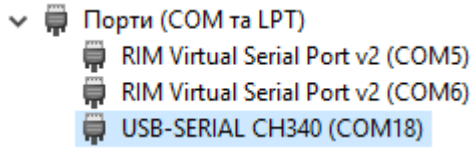
Порядок виконання роботи

1. Ознайомитися з апаратною частиною і підготувати макетну плату Arduino.
2. Запустити середовище Arduino IDE. Ознайомитися з інтерфейсом і синтаксисом мови програмування.
3. Створити новий файл, вибравши у меню Файл -> Створити.

4. Вставити наступний програмний код у створений файл та зберегти його.



5. Виконуємо перевірку (компіляцію) програмного коду натиснувши клавішу .
6. Наступним кроком під'єднуємо USB кабель до плати Arduino і під'єднуємо до USB-порту персонального комп'ютера.
7. Далі налаштовуємо COM-порт натиснувши у меню середовища Arduino IDE Інструменти ->Порт.
8. Вибираємо номер COM-порта такий який висвічується у встановленому драйвері USB-SERIAL CH430 в диспетчері пристроїв ПК.



9. Завантажуємо програму в мікроконтролер макетної плати Arduino UNO натиснувши кнопку .
10. Спостерігаємо результати роботи програми.
11. Модифікувати програму так щоб світлодіод мигав кожні 0,5 с.
12. Реалізувати мигання світлодіодом з використанням функції паузи ***delay*** лише один раз.
13. Реалізувати алгоритм мигання світлодіодом якій би не заважав мікроконтролеру виконувати інші функції.
14. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.

4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Чим відрізняються логічні операції &&, || від бітових операцій &, | ?
2. Які переваги платформи Arduino?
3. Які недоліки використання функції *delay (ms)?*
4. Для чого і в яких випадках застосовують підтягуючий резистор?
5. Які основні недоліки найпростішої програми мигання світлодіодом.
6. Що таке третій стан виводу мікроконтролера?

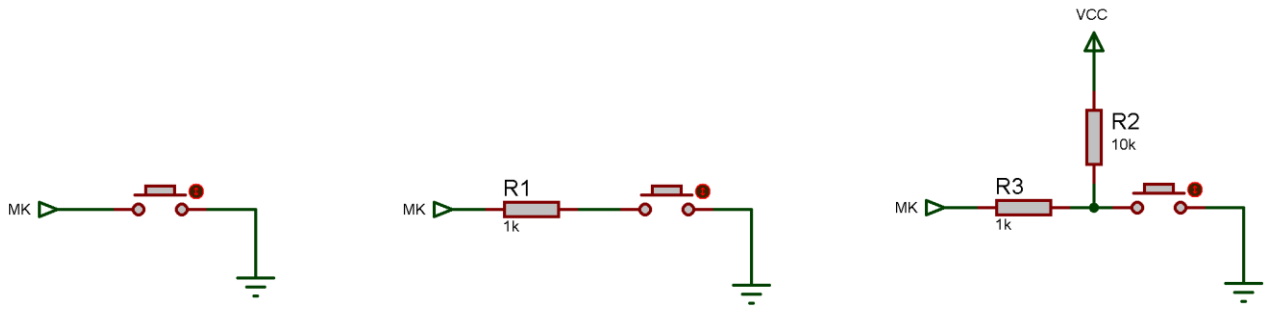
Лабораторна робота №2

Опитування дискретних входів. Управління за допомогою кнопок. Проектування підсистеми визначення рухомих об’єктів розумного будинку на основі сенсора руху HC-SR501.

Мета роботи: Опитування дискретних входів. Управління за допомогою кнопок. Проектування підсистеми визначення рухомих об’єктів “розумного будинку” на основі сенсора руху HC-SR501.

Теоретичні відомості

Для підключення кнопки до мікроконтролера можна скористатися однією з наведених на рис 2.1. схем.



а б в

Рис 2.1. Схеми підключення кнопки до мікроконтролера.

При використанні двох перших варіантів підключення внутрішнього підтягуючого резистора є обов’язкове. Варіант в є оптимальним хоча і містить 2 здавалося б зайвих резистори, але використання зовнішнього підтягуючого резистора на 10 кОм робить схему значно стійкішою до завад, а резистор R3 не дасть вигоріти порту мікроконтролера при натисканні на кнопку, якщо його в програмі випадково буде налаштовано на вихід в стані HIGH.

При замиканні і розмиканні перемикачів у колі виникають імпульсні завади, викликані брязкотом контактів.

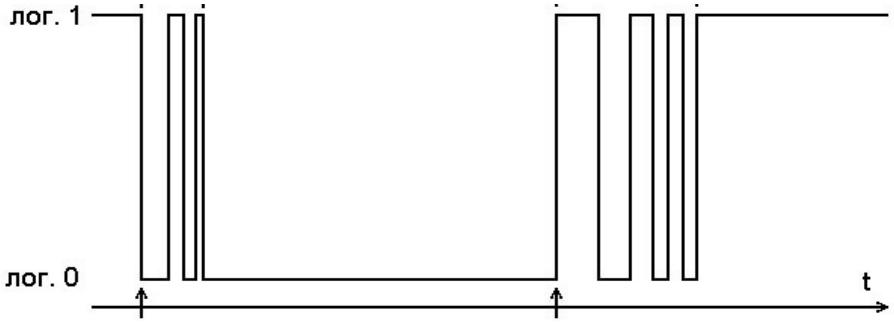


Рис 2.2. Моменти натиску та відпускання кнопки.

При натисненні на кнопку або спрацюванні перемикача напруга не відразу встановлюється на заданому рівні, а «скаче» протягом деякого часу, поки коло надійно не замкнеться. Після того, як кнопка буде відпущена чи контакти розімкнені, напруга також «скаче», поки не встановиться на відповідному рівні, що відповідає цьому стану кнопки чи іншого пристрою перемикання.

Так як контролер дуже швидко зчитує стан виводів завади, викликані брязкотом контактів, можуть інтерпретуватися як декілька розмикань і замикань ключа. Якщо клавіша використовується для керування пристроєм, то цей пристрій сприйме завади як багаторазове натискання клавіші, що викличе значні ускладнення при його використанні.

Для визначення стану кнопки використовують функцію digitalRead (pin) Параметри**:** pin: номер цифрового виводу, значення типу int. Значення, що повертаються HIGH або LOW. Для усунення брязкоту контактів потрібно використовувати паузи завідомо більші ніж тривалість перехідних процесів при замиканні та розмиканні контактів, але досить малі щоб користувач їх не помічав.

Часто для роботи систем сигналізацій використовують інфрачервоні давачі руху в результаті їх спрацювання відбудеться оповіщення та світлова індикація порушення периметру.

Для реалізації функції перебування сторонніх осіб в будинку, використано давач руху на основі піроелектричного ефекту (PIR, Passive InfraRed motion sensor). Такі давачі часто використовуються в охоронних системах і в побуті для виявлення руху в приміщенні. Також на принципі визначення руху будується автоматичне включення освітлення. Піроелектричні давачі досить прості в налаштуванні, недорогі досить прості у встановлені та обслуговуванні.

Принцип роботи піроелектричних давачів руху

Піроелектрики – це діелектрики, які створюють електричне поле при зміні їх температури. На основі піроелектриків роблять давачі вимірювання температури, наприклад, LHI778 або IRA-E700. Кожен такий давач містить два чутливих елемента розміром 1х2 мм, підключених протилежною полярністю. І відповідно наявність саме двох елементів допоможе відслідковувати рух.

Вигляд давача IRA-E700 компанії Murata наведено на рис. 2.2.

Рис. 2.3. Давач IRA-E700 компанії Murata.

В давачі руху HC-SR501 (рис. 2.3.), використовується один такий піроелектричний давач. Зверху піроелектрика розташована півсфера яка розбита на декілька сегментів. Кожен сегмент цієї сфери є лінзою, яка фокусує теплове випромінювання на різні ділянки PIR-давача. Часто в якості лінзи використовують лінзу Френеля.

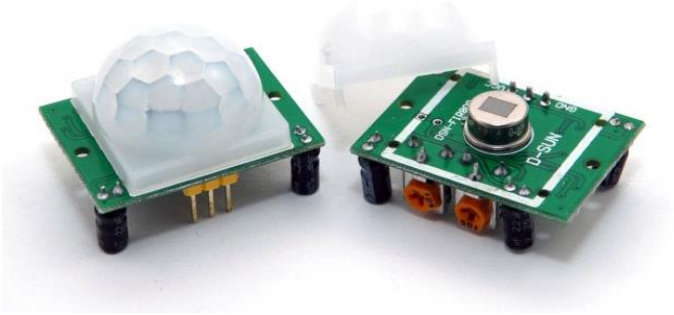


Рис. 2.4. Інфрачервоний давач руху HC-SR501.

Принцип роботи давача руху наступний. Припустимо, що давач встановлений в порожній кімнаті. Кожен чутливий елемент отримує постійну дозу випромінювання, а значить і напруга на них має постійне значення (рис. 2.4).

Як тільки в кімнату заходить людина, вона потрапляє спочатку в зону огляду першого елемента, що призводить до появи позитивного електричного імпульсу на ньому (рис. 2.4.).

Людина рухається, і його теплове випромінювання через лінзи потрапляє вже на другий PIR-елемент, який генерує негативний імпульс. Електронна схема давача руху реєструє ці різноспрямовані імпульси і робить висновки про те, що в поле зору давача потрапила людина. На виході давача генерується позитивний імпульс (рис. 2.4.).

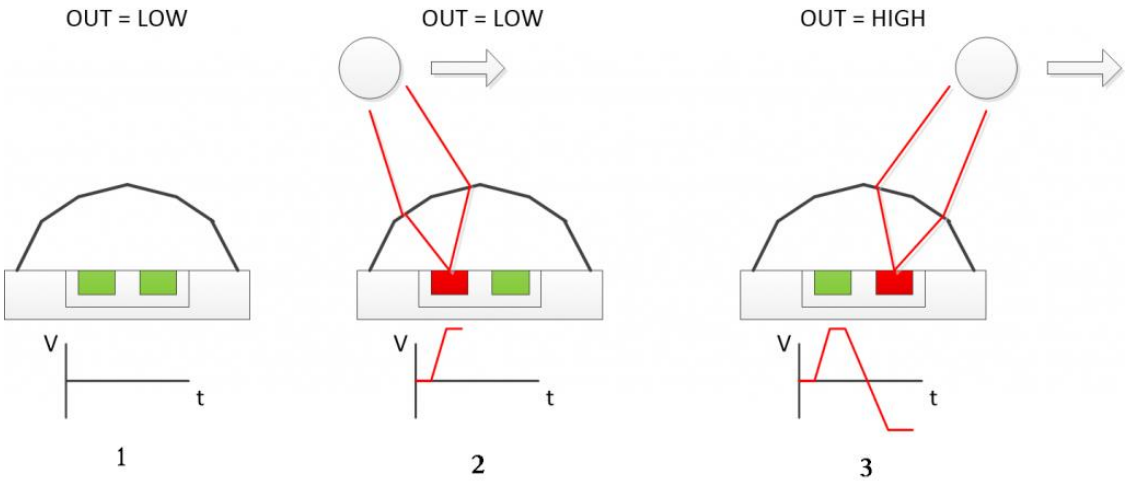


Рис. 2.5. Принцип роботи давача руху.

Характеристики давача руху HC-SR501:

- живлення: постійна напруга 4,5-20 В;
- у режимі очікування струм споживання: менше 50 мкА;
- максимальний струм під час роботи: 65 мА;
- напруга логічних рівнів: 3,3 В;
- відстань спрацювання: 3-7 м, за замовчуванням 7 м;
- максимальний кут огляду: 110°, на відстані 7 м – 120°;
- збереження високого рівня виходу при присутності: 20-300 с;
- час ігнорування подій після фіксації: 0,2 с;
- температура навколишнього повітря при роботі: -15 ... 70 ° С;
- розміри 32 х 24 х 28 мм.

Налаштування та підключення модуля HC-SR501

Цей модуль дуже поширений і застосовується в безлічі DIY проектів через невисоку вартість.

У модуля є два змінних резистора і перемичка для налаштування режиму. Один з потенціометрів регулює чутливість приладу. Чим вона більше, тим далі «бачить» давач. Також чутливість впливає на розмір детектованого об’єкта. Наприклад, можна виключити із спрацювання домашніх тварин.

Другий потенціометр регулює час спрацьовування *T*. Якщо давач виявив рух, він генерує на виході позитивний імпульс довжиною *T*.

Третій елемент управління – перемичка, яка перемикає режим давача. У положенні *L* давач веде відлік *T* від першого спрацьовування. Припустимо, ми хочемо керувати освітленням у ванній кімнаті. Зайшовши в кімнату, людина спричинить спрацьовування давача, і світло включиться рівно на час

T. Після закінчення періоду, сигнал на виході повернеться в початковий стан, і давач буде очікувати наступного спрацьовування.

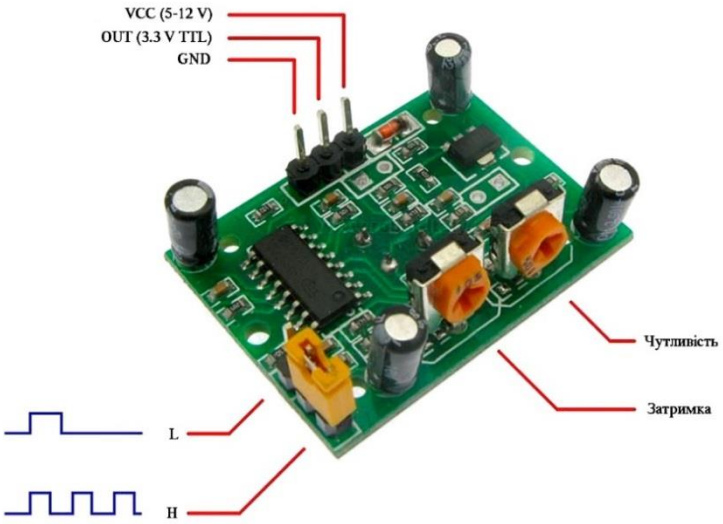


Рис. 2.6. Призначення елементів керування модуля HC-SR501.

У положенні *H* давач починає відлік часу *T* кожен раз після виявлення руху. Іншими словами, будь-який рух людини буде призводити до обнулення таймера відліку *T*. По-замовчуванню, перемичка знаходиться в стані *H*.

Для з'єднання з мікроконтролером у HC-SR501 є три виводи. Підключаємо їх до Arduino за наступною схемою:

HC-SR501	GND	VCC	OUT
Arduino Nano	GND	+5V	A2

Практичне завдання до лабораторної роботи

Реалізувати схему та програму керванння світлодіодом за допомогою кнопки.

Реалізувати схему підключення давача руху HC-SR501 до плати Arduino UNO та запрограмувати його роботу в середовищі Arduino IDE.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, інфрачервоний давач руху HC-SR501, кнопку і провідники.
2. На макетні платі скласти схему підєднання кнопки до Arduino.
3. Прошити в плату Arduino просту програму керування світлодіодом за допомогою кнопки.

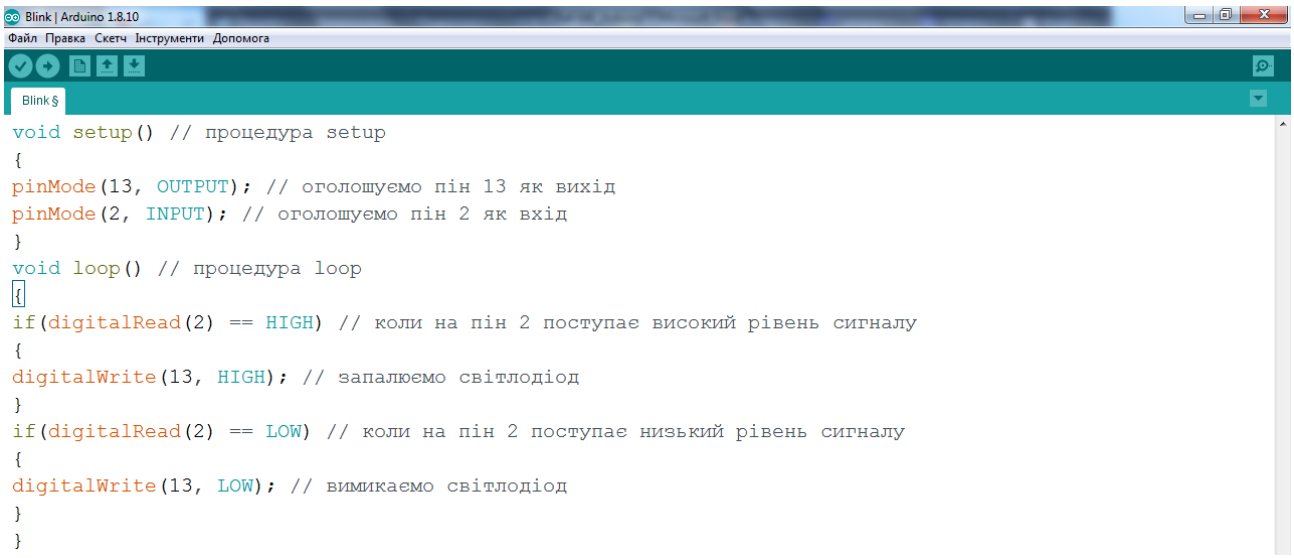


Рис. 2.7. Проста програма зчитування стану кнопки.

4. Модифікувати програму щоб використовувати тільки один оператор if.
5. Модифікувати програму щоб при кожному натиску кнопки стан світло діода мінявся на протилежний, врахувати брязкіт контактів.
6. Виконати під'єднання давача руху HC-SR501 до макетної плати згідно схеми підключення (рис. 2.7), щоб бачити розпіновку сигналу живлення (VCC), землі (GND) та вихідного цифрового сигналу (OUT) на давачі руху HC-SR501 потрібно зняти його лінзу.

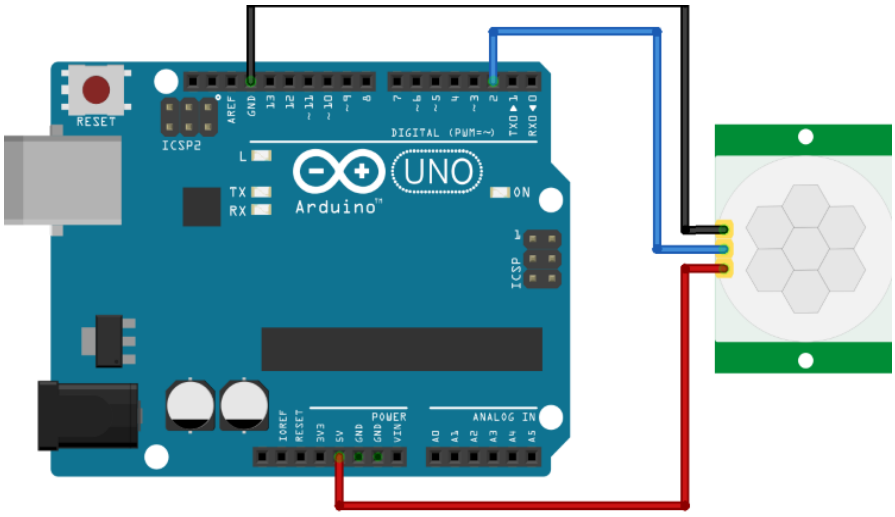


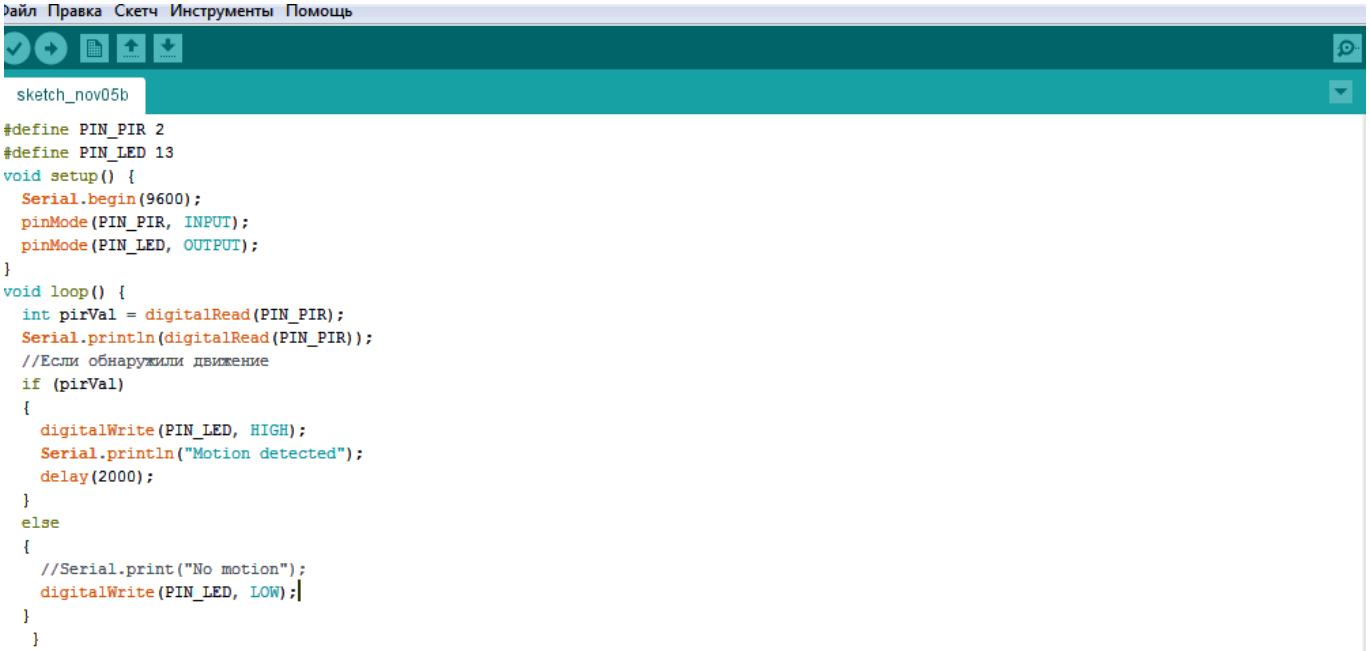
Рис. 2.8. Схема підключення давача руху HC-SR501 до плати Arduino UNO.

7. Створити новий файл. Вставити наступний програмний код у створений файл та зберегти його.

```
#define PIN_PIR 2
#define PIN_LED 13
```

```
void setup() {
  Serial.begin(9600);
  pinMode(PIN_PIR, INPUT);
  pinMode(PIN_LED, OUTPUT);
}
void loop() {
  int pirVal = digitalRead(PIN_PIR);
  Serial.println(digitalRead(PIN_PIR));
  //Якщо виявлено рух
  if (pirVal)
  {
    digitalWrite(PIN_LED, HIGH);
    Serial.println("Motion detected");
    delay(1000);
  }
  else
  {
    //Serial.print("No motion");
    digitalWrite(PIN_LED, LOW);
  }
   }
```

8. Виконуємо перевірку (компіляцію) програмного коду натиснувши клавішу .



9. Наступним кроком під’єднуємо плату Arduino UNO до персонального комп’ютера.

10. Завантажуємо програму в мікроконтролер макетної плати Arduino UNO натиснувши кнопку .
11. Відкриваємо монітор послідовного порту зайшовши у меню Інструменти -> Монітор послідовного порту.
12. Спостерігаємо результати роботи програми, коли рух об’єкта є і руху об’єкта немає.
13. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Що таке брязкіт контактів і як з ним боротися?
2. Для чого використовують давачі руху на основі піроелектричного ефекту?
3. Що таке піроелектрики?
4. Яка будова давача руху HC-SR501?
5. Опишіть принцип роботи давача руху HC-SR501.
6. Назвіть характеристки давача руху HC-SR501?
7. Який принцип підключення давача руху HC-SR50?

Лабораторна робота № 3.
Підсилення дискретних виходів. Керування індуктивним навантаженням. Електрозамок на основі актюатора.

Мета роботи: Вивчення схем підключення силових елементів до мікроконтролера, отримання напичок керування потужними навантаженнями.

Теоретичні відомості

Підсилення дискретних виходів.

Вивід мікроконтролера може видати струм до 20мА, але підключати щось потужніше ніж світлодіод до мікроконтролера напряму не прийнято. Також мікроконтролер може видати напругу не більше 5В, а навантаженням часто потрібно значно вищу напругу живлення. Для підключення таких навантажень використовують транзистори, симістори, електромеханічні та твердотільні реле.

Один із варіантів - поставити транзистор. Транзистор можна підібрати на сотні міліампер, а то і на один ампер. Якщо не вистачає одного транзистора, то їх можна включати каскадами, при цьому силовий транзистор відкривається за допомогою малопотужного транзистора.

Оскільки у нас прийнято, що лог. “1” це включено, а лог. “0” вимкнено, то “1” у нас буде подавати живлення, а “0” знімати навантаження. Виберемо біполярний транзистор. Реле потрібно 80мА, тому шукаємо транзистор з колекторним струмом більш 80мА. В імпортних даташітах цей параметр називається Ic, в наших Ік. Перше що спало на думку BC546 (а також BC547, BC548, BC549). У транзистора, в першу чергу, треба визначити призначення виводів, де у нього колектор, де база, а де емітер. Зробити це найкраще по даташиту.

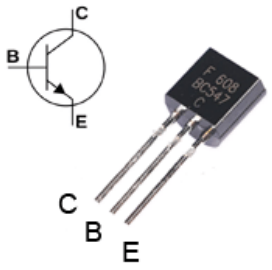


Рис. 3.1. Розміщення виводів транзистора

Транзистор це підсилювач струму, тобто якщо ми пропустимо через ланцюг база-емітер струм, то через ланцюг колектор-емітер зможе пройти струм рівний вхідному, помноженому на коефіцієнт підсилення h_{fe}. якій для цього транзистора складає близько 300.

Максимальна напруга виводу мікроконтролера при подачі в порт одиниці = 5 Вольт (падінням напруги в 0,7 Вольт на база-емітерному переході тут можна знехтувати). Опір базового ланцюга складає близька 10000 Ом. Значить струм 0.5мА - абсолютно незначний струм. А на виході в цей момент часу буде Ic = Ibe * hfe = 0.0005 * 300 = 0.150A. 150мА більше ніж 100мА, але це всього лише означає, що транзистор відкриється повністю і видасть максимум що може. А значить наше реле нормально працюватиме.

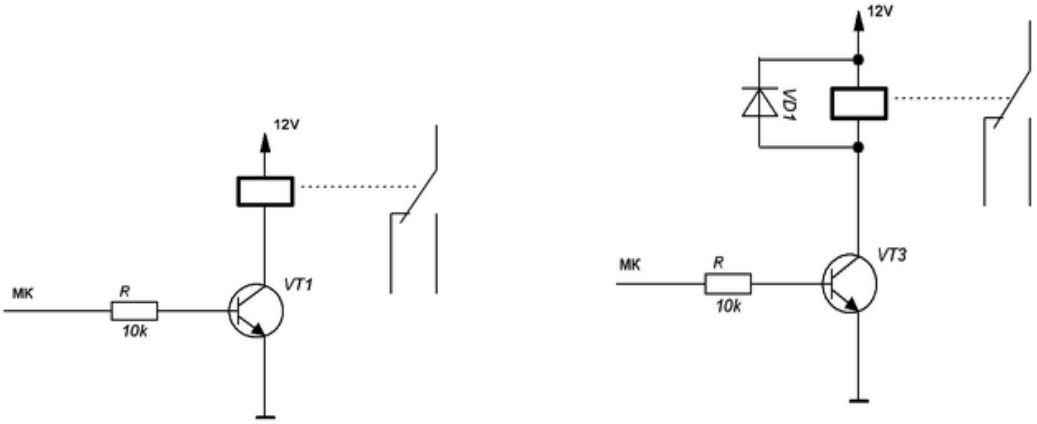


Рис. 3.2. Схеми підсилення виводів мікроконтролера за допомогою транзистора та електромеханічного реле.

Керування індуктивним навантаженням. В реле у якості виконавчого елемента використовується котушка. А котушка має неабияку індуктивність, так що різко обірвати струм в ній неможливо. Якщо це спробувати зробити, то потенційна енергія, накопичена електромагнітним полем, призведе до електричного пробою ізоляції або напівпровідникових елементів. При нульовому струмі обриву, цим місцем буде напруга - при різкому перериванні струму, на котушці буде потужний сплеск напруги, в сотні вольт. Якщо струм обривається механічним контактом, то буде повітряний пробій - іскра. А якщо обривати транзистором, то його просто напросто пробє.

Треба кудись дівати енергію котушки. Не проблема, замкнем її на себе ж, поставивши діод. При нормальній роботі діод включений зустрічно напрузі і струм через нього не йде. А при виключенні напруга на індуктивності буде вже протилежною і пройде через діод.

Автомобільний актюатор.

Дані пристрої достатньо прості. Їх основу складає маленький електродвигун, який з'єднаний з рухомим штоком за принципом рейкової або гвинтової передачі, а до нього вже кріпиться тяга від механічного замка. Коли відбувається подача напруги на один з проводів, що живлять електродвигун, вал мотора починає обертатися в потрібну сторону, при цьому обертаючи шток. Разом зі штоком зсувається і тяга, прикріплена іншим кінцем до важеля

автомобільного механічного замка. Як наслідок, відбувається один з двох варіантів: звільнення замків або їх блокування. Шток ходить на дуже короткі проміжки, тому електродвигун швидко заклинюється. Щоб в такі моменти не відбулося перегоряння моторчика, на актюатор не можна подавати напругу тривалий період часу. Тому, коли двері зачиняються або відмикаються, блок керування замком подає напругу на актюатори всього на кілька секунд, а потім автоматично відключає їх. Такі періоди досить різні за часом, але в середньому це дві секунди. Цього достатньо для спрацьовування приводів, але за цей час обмотка електродвигуна кожного з них не перегорить.

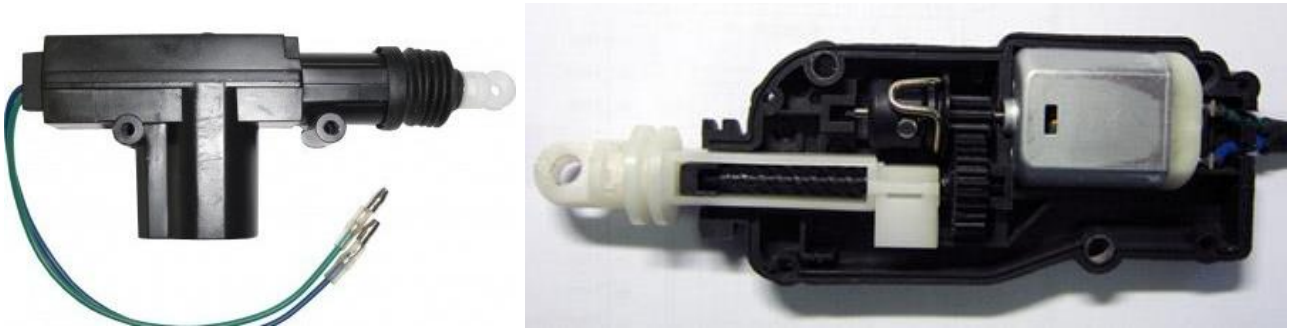


Рис. 3.3. Загальний вигляд та будова автомобільного актюатора.

Електроприводи автомобільного замка не мають чітко встановлених плюсового і масового проводів, вони змінюються в залежності від завдання, яку повинен виконати привід: відімкнути або замкнути замок. Цим розподілом також повинен займатися блок керування. Його електронні компоненти з'єднують ланцюг живлення приводів таким чином, що на один провід подається плюс, а на іншому з'являється мінус. Від того, на який дріт електроприводу прийде напруга, залежить, в який бік буде обертатися вал його двигуна. За рахунок подачі струму на різні дроти вал крутиться в різні боки, висуваючи або втягуючи шток і, відповідно, замикаючи або відмикаючи замок дверей автомобіля.

Схеми зміни полярності.

Для зміни полярності можна використовувати електромеханічне реле з двома парами перекидних контактів, як показано на схемі (рис. 3.4).

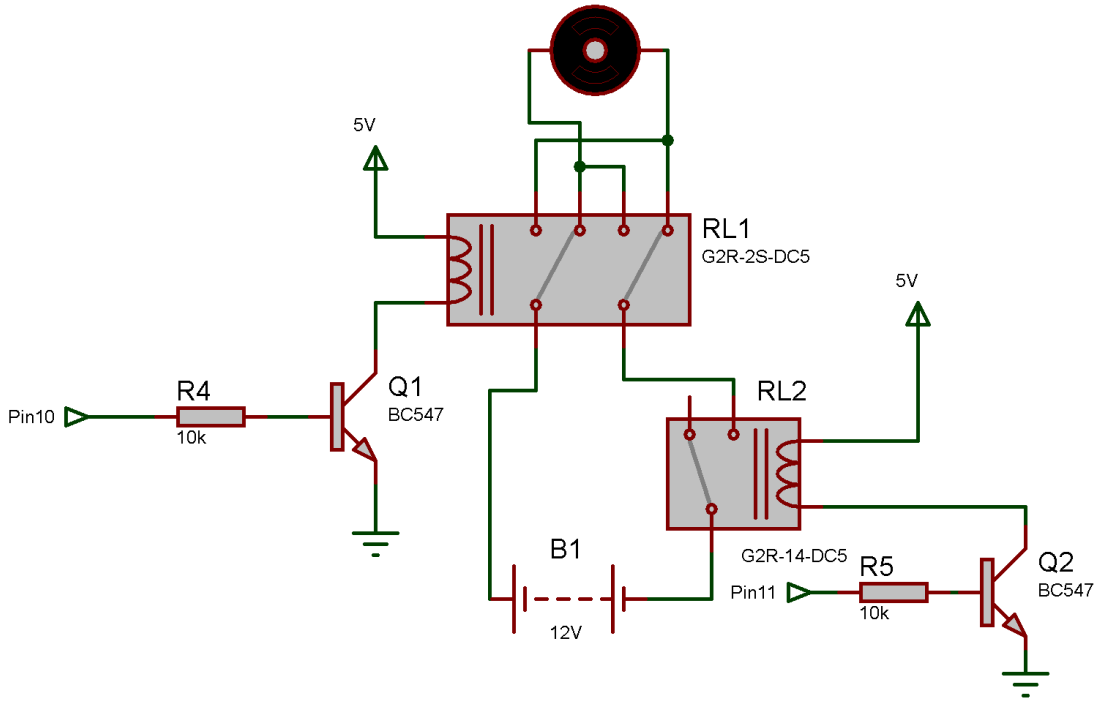


Рис. 3.4. Електрична принципова схема електрозамка.

Реле RL1 у нормальному стані вмикає навантаження в одній полярності а при спрацюванні контакти реле перемикають навантаження так щоб воно було ввімкнено в зворотній полярності. Для ввімкненні навантаження в цьому випадку використовується друге реле RL2. Одже для закриття замка нам достатньо на дві секунди увімкнути реле RL2, відкривши транзистор Q2. Щоб відкрити замок спочатку вмикаємо реле RL1, яке змінить полярність підключення актюатора і тільки після цього на дві секунди вмикаємо реле RL2.

Практичне завдання до лабораторної роботи

Реалізувати схему та програму керванння електрозамком за допомогою однієї кнопки.

Придумайте і реалізуйте алгоритм кодового замка з однією кнопкою використовуючи комбінацію коротких і довгих натисків.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, плату реле, кнопку і провідники та скласти схему електрозамка на основі автомобільного актюатора.
2. Скласти та прошити в плату Arduino просту програму керування актюатором як показано на знімку екрана.

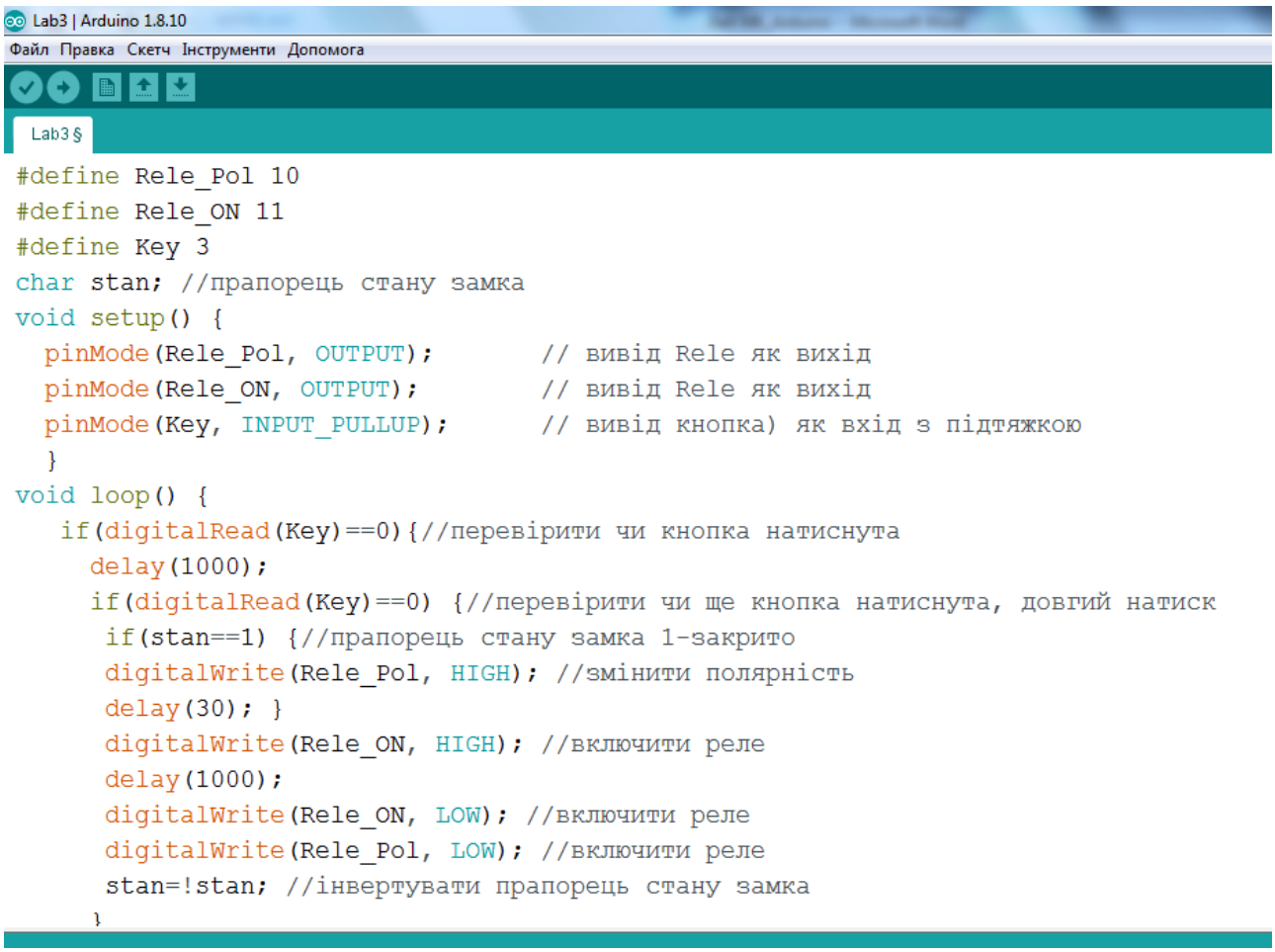


Рис. 2.6. Проста програма зчитування стану кнопки.

3. Модифікувати програму щоб короткій натиск кнопки закривав замок а довгий відкривав.
4. Поекспереметувати з часом необхідним на закривання та відкривання замка, визначити мінімальний час необхідний для стабільного відкривання та закривання замка.
5. Придумати алгоритм, щоб замок відкривався тільки після введення правильного коду, модифікувати програму для реалізації даного алгоритму та перевірити її роботу.
6. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.

7. Висновки.

Контрольні запитання

1. Вкажіть основні переваги використання дерективи препроцесора #define.
2. Які основні параметри транзистора враховуються при виборі його для схеми керування потужним навантаженням за допомогою мікроконтролера?
3. Навіщо потрібен резистор між базою транзистора та виводом мікроконтролера, які критерії його вибору?
4. Яка роль діода у схемі підключенні реле до мікроконтролера?
5. Який основний недолік двопровідного автомобільного актюатора.
6. Які переваги замка на основі актюатора перед електромагнітним?
7. Наведіть інші варіанти схем зміни полярності на актюаторі?

Лабораторна робота № 4.

Таймери. Переривання. Вимірювання інтервалу часу між двома вхідними імпульсами. Послідовна передача даних. UART.

Мета роботи: Вивчення периферійних пристроїв мікроконтролера. Отримати навики практичного використання таймерів, переривань та послідовної передачі даних.

Теоретичні відомості

У ATMega328 наявні три таймери/лічильники, на яких реалізовано функції часу, які використовують для формування і виміру часових інтервалів.

Основні функції часу:

delay (ms)

- Параметри: ms - кількість мілісекунд, на які необхідно призупинити програму.
- Значення, що повертаються – немає
- Опис: припиняє виконання програми на вказаний проміжок часу (в мілісекундах). (В 1 секунді - 1000 мілісекунд.)

delayMicroseconds (us)

- Параметри: us - кількість мікросекунд, на які необхідно призупинити програму.
- Значення, що повертаються – немає.
- Опис: припиняє виконання програми на вказаний проміжок часу (в мікросекундах). На даний момент найбільша кількість, що дозволяє сформувати точну затримку - 16383.

В майбутніх версіях Arduino цей показник може бути змінений. Для створення затримок тривалістю більше, ніж кілька тисяч мікросекунд, використовуйте функцію delay ().

millis ()

- Параметри – Ні.
- Значення, що повертаються – кількість мілісекунд, що пройшли з моменту старту програми.
- Опис: повертає кількість мілісекунд, що пройшли з моменту старту програми Arduino. Число, що повертається, переповниться (скинеться в 0) через приблизно 50 днів.

micros()

- Параметри – Ні.

• Значення, що повертаються – кількість мікросекунд, що минули з моменту старту програми.

• Опис: повертає кількість мікросекунд, що минули з моменту початку виконання програми Arduino. Число, що повертається, переповниться (скинеться в 0) через приблизно 70 хвилин.

На платах Arduino з тактовою частотою 16 МГц (Duemilanove і Nano) роздільна здатність цієї функції становить чотири мікросекунди (тобто значення, що повертається, буде завжди кратне чотирьом). На платах Arduino з тактовою частотою 8 МГц (LilyPad), роздільна здатність функції становить вісім мікросекунд.

pulseIn (pin, value)

pulseIn (pin, value, timeout)

• Параметри: pin - номер виводу, на якому буде очікуватися сигнал; value: тип зчитує імпульс: HIGH або LOW; timeout (опціонально): час очікування імпульсу в мікросекундах (значення за замовчуванням – одна секунда).

• Значення, що повертаються – тривалість імпульсу (в мікросекундах) або 0 в разі відсутності імпульсу протягом таймаута.

• Опис: зчитує тривалість імпульсу (будь-якого – HIGH або LOW) на виведення. Наприклад, якщо задане значення (value) – HIGH, то функція pulseIn () очікує появи на виведення сигналу HIGH, потім вимірює час і чекає перемикання в стан LOW, після чого зупиняє відлік часу. Функція повертає тривалість імпульсу в мікросекундах, або 0 в разі відсутності імпульсу протягом певного часу очікування. Емпіричним шляхом встановлено, що при використанні функції для вимірювання широких імпульсів можливе виникнення помилок. Функція працює з імпульсами тривалістю від 10 мікросекунд до 3 хвилин.

Переривання – це сигнали, що переривають нормальний перебіг програми. Вони зазвичай використовуються для апаратних пристроїв, що вимагають негайної реакції на появу подій.

Обробка переривань у мікроконтролері відбувається за допомогою модуля переривань, який приймає запити переривання й організовує перехід до виконання визначеної програми. Запити переривання можуть надходити як від зовнішніх джерел, так і від джерел, розташованих у різних внутрішніх модулях мікроконтролера. Як входи для прийому запитів від зовнішніх джерел, найчастіше використовуються виводи портів вводу/виводу, для яких ця функція є альтернативною. Джерелами запитів зовнішніх переривань також можуть бути будь-які зміни зовнішніх сигналів на деяких спеціально виділених лініях портів вводу/виводу.

Arduino надає свої функції для роботи з зовнішніми перериваннями. Ці функції оголошені у файлі:

\ Hardware \ cores \ arduino \ wiring.h

Функції переривання Arduino:

1. attachInterrupt (interrupt, function, mode)

•□Параметри: *interrupt*: номер переривання (0 – pin D2, 1 – pin D3);

function: функція, яку необхідно викликати при виникненні переривання; ця функція повинна бути без параметрів і не повертати ніяких значень (таку функцію іноді називають оброблювачем переривання); *mode*: визначає умову, за якої має спрацьовувати переривання.

Mode може приймати одне з чотирьох визначених значень: LOW – переривання буде спрацьовувати щоразу, коли на виводі присутній низький рівень сигналу, CHANGE – переривання буде спрацьовувати щоразу, коли змінюється стан виводу**,** RISING – переривання спрацює, коли стан виводу зміниться з низького рівня на високий**,** FALLING – переривання спрацює, коли стан виводу зміниться з високого рівня на низький.

•□Значення, що повертаються – немає.

2. detachInterrupt (pin)

•□Опис: Забороняє задане переривання.

•□Управління перериваннями *noInterrupts ().*

•□Параметри – ні.

•□Значення, що повертаються – немає.

Забороняє переривання, для того, щоб відключити доступ до даних в процесі виконання переривання.

3. interrupts ()

•□Параметри – ні.

•□Значення, що повертаються – немає.

•□Опис: повторно дозволяє переривання (після того, як вони були відключені функцією *noInterrupts ()*).

Асинхронний послідовний обмін

Найбільш розповсюджена форма в мікроконтроллерних системах — асинхронний обмін, при якому байт даних посилається як пакет, що включає інформацію про початок і кінець передавання даних, а також інформацію для контролю помилок.

Першим передається не біт даних, а старт-біт, що вказує на початок передавання даних (початок пакета). Цей біт використовується приймачем для синхронізації процесу читання даних, що передаються за старт-бітом (молодший біт даних йде першим). Після бітів даних може передаватися біт

парності (контрольний біт), що використовується для перевірки правильності отриманих даних. За бітом парності передається стоп-біт, що використовується приймачем для обробки кінця передавання пакета.

Для зв'язку з платою Arduino можна використовувати спеціальну програму моніторингу послідовного порту (Serial Monitor), вбудовану в програмне забезпечення Arduino. Монітор послідовної шини відображає дані, що посилаються в плату Arduino через віртуальний послідовний порт за допомогою USB.

Також доступний інструмент «Послідовний плотер», який дозволяє будувати часові залежності числових даних що виводяться в порт. Вісь Y показує дані, передані по послідовному порту. Вісь X складається з 500 пунктів, і кожен пункт еквівалентний одному виконанню команди Serial.println (). Це означає, що графік оновлюється кожного разу, коли ви використовуєте Serial.println () з новим значенням.

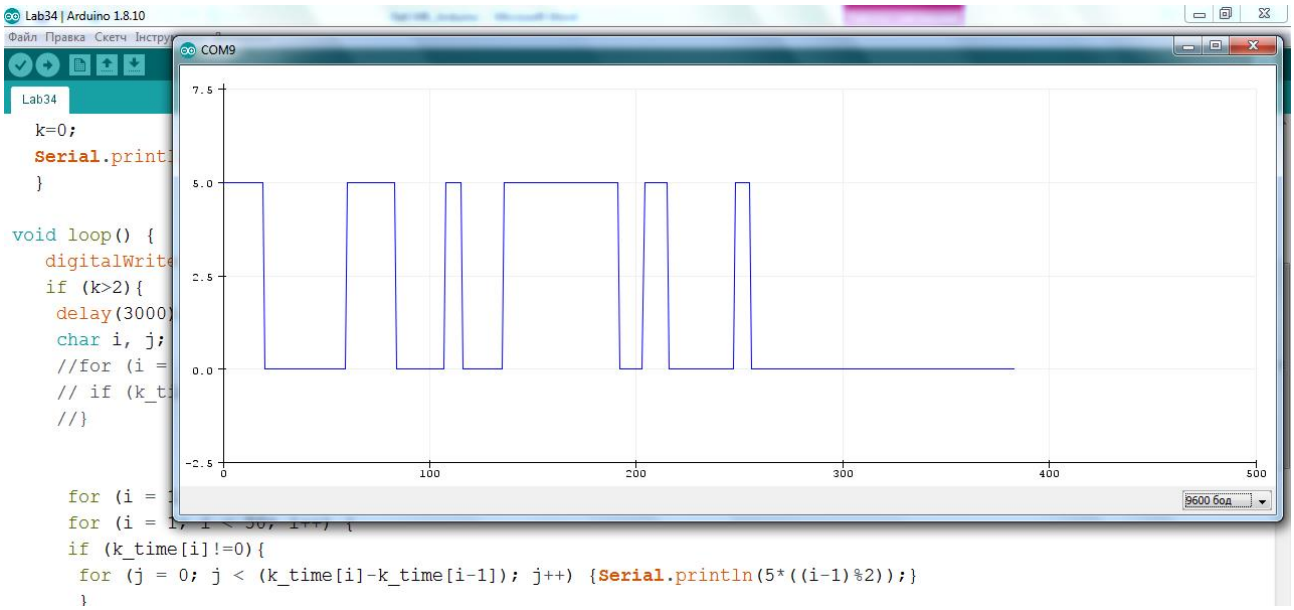


Рис. 4.1. «Осцилограма» брязкоту контактів побудована за допомогою інструменту «Послідовний плотер». Вісь X – час в мкс.

Для відправки даних вибирається швидкість передачі зі списку, відповідно значенню Serial.begin в скетчі. Потім необхідно ввести текст і натиснути кнопку Send або Enter. Плата Arduino Nano має один послідовний порт (також відомий як UART або USART): Serial, що використовується для зв'язку з комп'ютером через USB. Він пов'язаний з цифровими виводами 0 (RX) і 1 (TX). Таким чином, під час роботи послідовного порту, виводи 0 і 1 не можуть використовуватися в якості цифрових входів або виходів. Для забезпечення зв'язку плати Arduino з комп'ютером або іншими пристроями використовується клас Serial.

Serial.begin(speed)

- Параметри: *speed*: швидкість в бітах на секунду (бодах) – long. Для взаємодії з комп'ютером слід використовувати одну з попередньо встановлених швидкостей обміну: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600 або 115200. По замовчуванню швидкість 9600.

Serial.end ()

- Параметри – ні.
- Значення, що повертаються – немає.
- Опис: функція розриває послідовний зв'язок, після чого виводи RX і TX знову можна використовувати як виводи загального призначення. Для відновлення послідовного з'єднання необхідно використовувати функцію *Serial.begin ()*

Serial.available()

- Параметри – ні.
- Значення, що повертаються: кількість байт, доступних для зчитування.
- Опис: повертає кількість байт (символів) доступних для зчитування з буфера послідовного порту. Під символами розуміються дані, які вже прийняті і зберігаються в послідовному приймальному буфері (який може зберігати максимум 64 байта).

Serial.read()

- Параметри – ні.
- Значення, що повертаються: перший байт прийнятих даних (або -1, якщо таких нема) – int.
- Опис: зчитує дані, що надходять по послідовному інтерфейсу.

Serial.print(val)

Serial.print(val, format)

- Параметри: *val***:** значення, яке необхідно вивести - будь-який тип даних; *format***:** визначає систему числення (для цілочисельних типів), а також кількість десяткових знаків після коми (для чисел з плаваючою точкою).
- Значення, що повертаються: кількість виведених байт. Зчитування цього значення не є обов'язковим.
- Опис: функція виводить через послідовний порт заданий ASCII-текст у вигляді, зрозумілому для людини. Ця команда може мати кілька різних форм. При виведенні числа кожній його цифрі відповідає один ASCII-символ. Дробові числа теж виводяться у вигляді ASCII-цифр, при цьому після коми за замовчуванням залишається два десяткових знака. Байти виводяться у вигляді окремих символів, а символи і рядки виводяться без змін – «як є».

Практичне завдання до лабораторної роботи

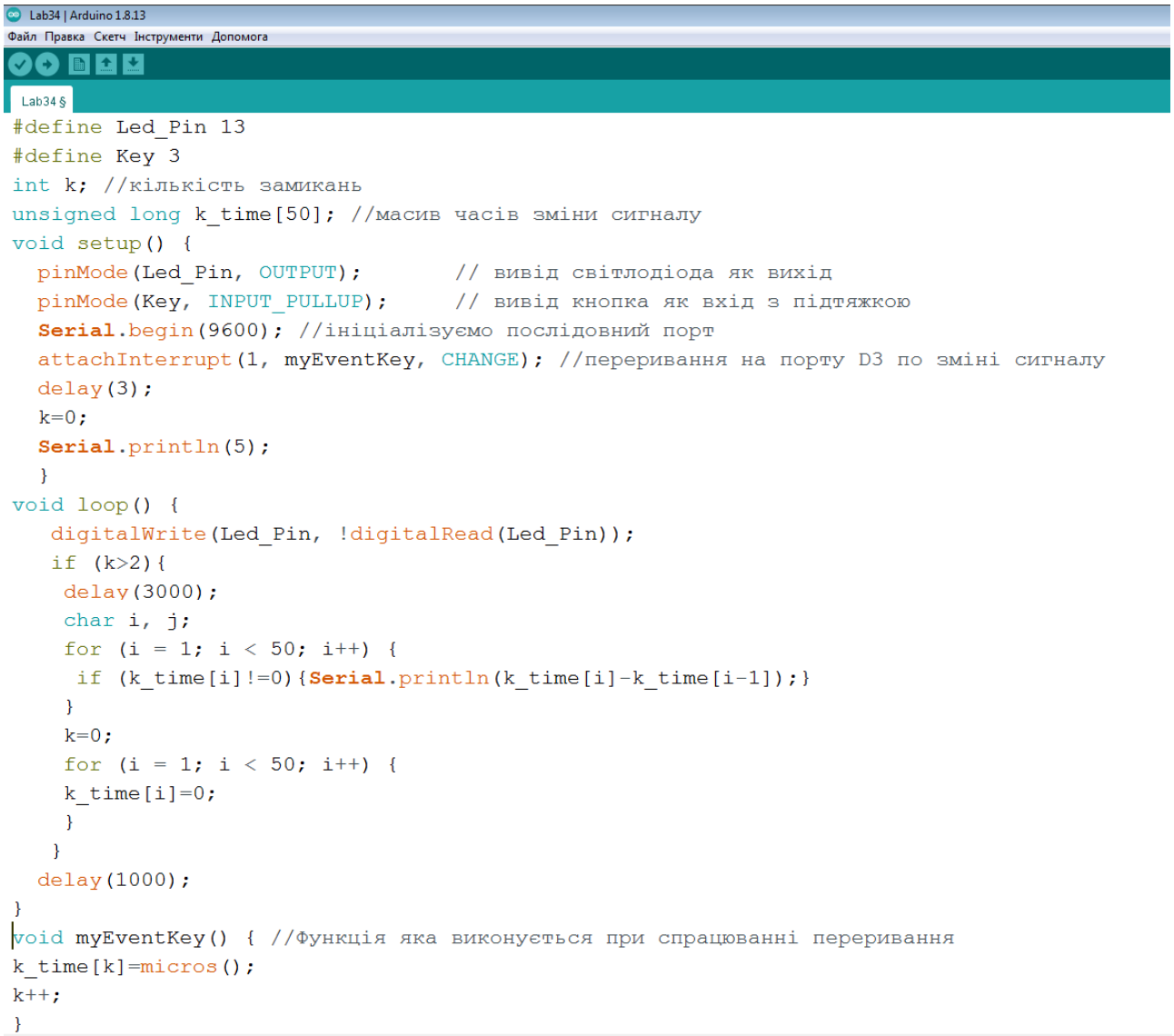
Виміряйте та виведіть на комп'ютер час напротязі якого кнопка була натиснута.

Дослідіть бряскіт контакту кнопки при натискання і відпусканні, виведіть кількість спрацювань кнопки при однократному натисканні і відпусканні.

Виведіть часові інтервали між змінами логічного рівня сигналу які описують брязкіт контактів кнопки. За допомогою інструменту "Послідовний плотер" побудуйте «осцилограму» брязкоту контактів.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, кнопку і провідники.
2. Під'єднайте кнопку одним контактом до виводу D3 Arduino, а іншим через резистор 1 кОм на спільний провід GND.
3. Скласти та завантажити в плату Arduino програму яка вимірює часові інтервали між змінами логічного рівня сигналу які описують брязкіт контактів кнопки.




```
#define Led_Pin 13
#define Key 3
int k; //кількість замикань
unsigned long k_time[50]; //масив часів зміни сигналу
void setup() {
  pinMode(Led_Pin, OUTPUT);       // вивід світлодіода як вихід
  pinMode(Key, INPUT_PULLUP);     // вивід кнопка як вхід з підтяжкою
  Serial.begin(9600); //ініціалізуємо послідовний порт
  attachInterrupt(1, myEventKey, CHANGE); //переривання на порту D3 по зміні сигналу
  delay(3);
  k=0;
  Serial.println(5);
  }
void loop() {
   digitalWrite(Led_Pin, !digitalRead(Led_Pin));
   if (k>2){
    delay(3000);
    char i, j;
    for (i = 1; i < 50; i++) {
     if (k_time[i]!=0){Serial.println(k_time[i]-k_time[i-1]);}
    }
    k=0;
    for (i = 1; i < 50; i++) {
    k_time[i]=0;
    }
   }
  delay(1000);
}
void myEventKey() { //Функція яка виконується при спрацюванні переривання
k_time[k]=micros();
k++;
}
```

Рис. 4.2. Проста програма дослідження брязкоту контактів.

4. Модифікувати програму щоб виводився на комп’ютер тільки максимальний час напротязі якого кнопка була натиснута та кількість спрацювань кнопки при однократному натисканні і відпусканні.
5. Дослідіть мінімальний час за який Ви можете натиснути кнопку.
6. Придумайте алгоритм і за допомогою інструменту ”Послідовний плотер” побудуйте «осцилограму» брязкоту контактів.
7. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Який тип даних використовує функція ***micros()*** для зберігання часу.
2. Що таке брязкіт контактів і як з ним боротися.
3. Що таке таймер мікроконтролера, яке його можливе застосування.
4. Для чого потрібні переривання, які особливості їх використання.

Лабораторна робота № 5.

Широтно-імпульсна модуляція. Формування кольору за допомогою RGB-світлодіода. Керування сервоприводом.

Мета роботи: Вивчення таймерів мікроконтролерів, генерування ШІМ сигналів.

Теоретичні відомості

Оскільки виводи AVR мікроконтролера цифрові то отримати проміжні значення між “0” і “1” ми не можемо. Багато навантажень (нагрівники, двигуни) досить інертні і швидко вмикаючи\вимикаючи навантаження можна керувати потужністю яка передається в навантаження.

Широтно-імпульсна модуляція (ШІМ — англ. pulse-width modulation, PWM), або модуляція за тривалістю імпульсів — процес керування шириною (тривалістю) високочастотних імпульсів за законом, який задає низькочастотний сигнал. В електроніці це може бути керування середнім значенням вихідної напруги шляхом зміни тривалості замкнутого стану електронного ключа, наприклад, у схемі ключового стабілізатора напруги.

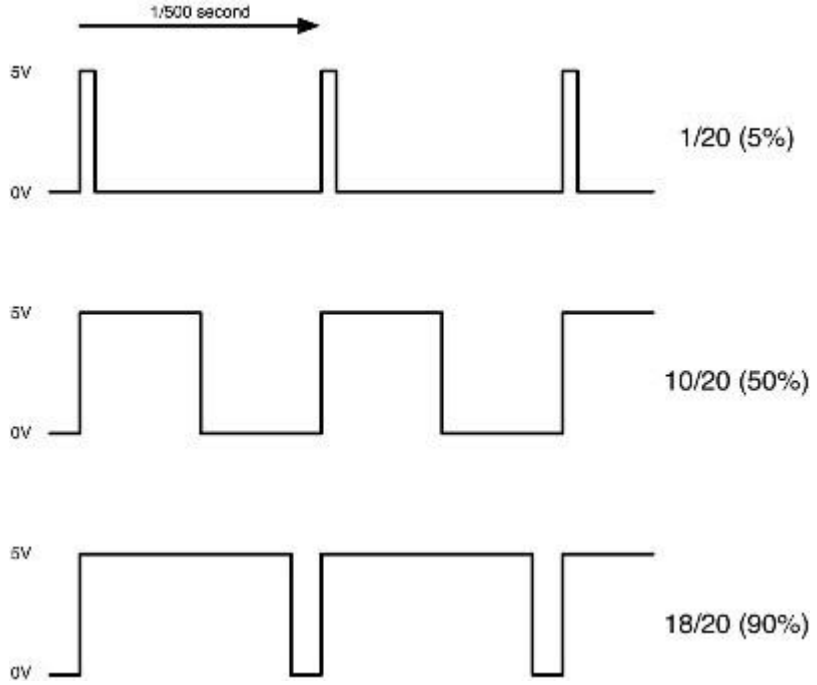


Рис. 5.1. Широтно імпульсна модуляція.

Якщо спробувати управляти яскравістю світлодіоду використовуючи широтно-імпульсну модуляцію то він швидко мигатиме з частотою ШІМ, але за рахунок інерції зору людина сприйматиме це як світіння з меншою яскравістю.

RGB (скорочено від англ. Red, Green, Blue — червоний, зелений, синій) — адитивна колірна модель, що описує спосіб синтезу кольору, за якою червоне, зелене та синє світло накладаються разом, змішуючись у різноманітні кольори. Широко застосовується в техніці, що відтворює зображення за допомогою випромінення світла.

У даній моделі колір кодується градаціями складових каналів (Red, Green, Blue). Тому за збільшення величини градації певного каналу — зростає його інтенсивність під час синтезу. Кількість градацій кожного каналу залежить від розрядності бітового значення RGB. Зазвичай використовують 24-бітну модель, у котрій визначається по 8 біт на кожен канал, і тому кількість градацій дорівнює 256, що дозволяє закодувати 256^3 = 16 777 216 кольорів.

Колірна модель RGB призначена сприймати, представляти та відображати зображення в електронних системах, таких як телебачення та комп'ютери, хоча її також застосовували у традиційній фотографії.

Червоний, зелений і синій світлодіоди (RGB LED) можуть бути одним світлодіодом, який фактично містить три світлодіоди. Три світлодіоди: червоний, зелений і синій. При використанні ШІМ для керування яскравістю кожного зі світлодіодних кольорів, ви можете зробити, щоб світлодіод світився будь-яким кольором.

Хоча RGB LED містить три нормальні двох вивідні світлодіоди, це не означає, що світлодіодний корпус повинен мати шість виводів, тому що один вивід кожного світлодіоду може бути єдиним спільним:

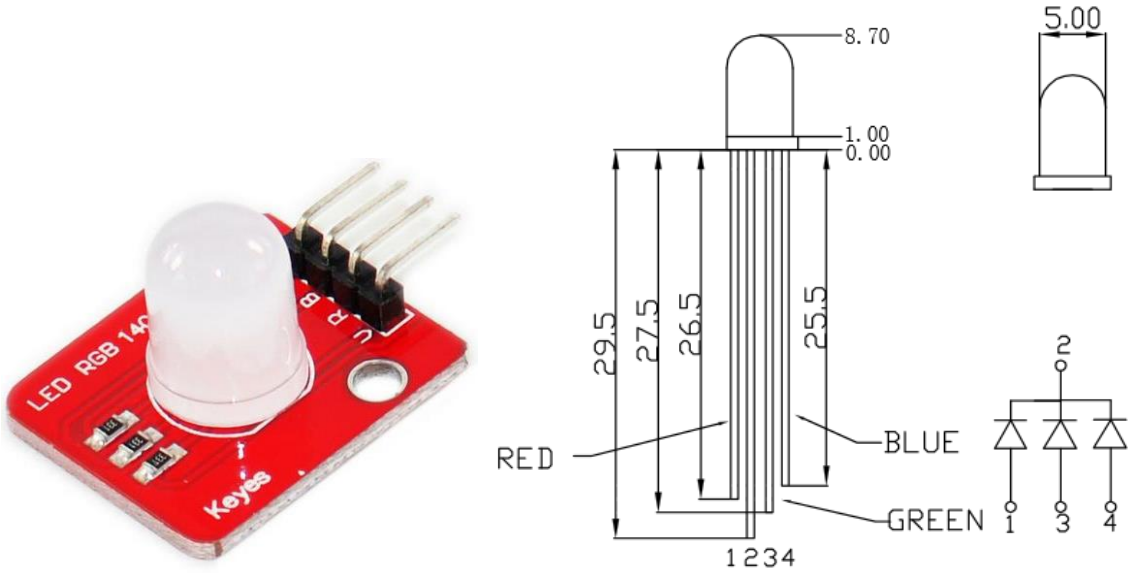


Рис. 5.2. RGB світлодіод та його схема підключення.

Якщо негативні підключення кожного світлодіоду зв'язані один з одним, то отриманий спільний вивід називається загальним катодом, а якщо позитивні зв'язки є загальними, то спільний вивід називається загальним анодом. Корпус RGB LED може бути прозорим або дифузним. Якщо він прозорий, то ви зможете побачити червоні, зелені та сині світлодіоди всередині корпусу, і колір не буде так добре змішуватися разом. Дифузні корпуси змішують світло від трьох світлодіодів набагато краще.

У Arduino ШІМ сигнал виводиться функцією

analogWrite (pin, value)

- Параметри:

pin – вивід, на якому буде формуватися напруга (ШІМ виводи 3, 5, 6, 9, 10 і 11); *value* – коефіцієнт заповнення – лежить в межах від 0 (завжди вимкнений) до 255 (завжди включений).

- Значення, що повертаються: немає
- Опис: формує задану аналогову напругу на виводі у вигляді ШІМ сигналу. Може використовуватися для варіювання яскравості світіння світлодіода або управління швидкістю обертання двигуна. Після виклику *analogWrite* (), на виводі буде безперервно генеруватися ШІМ-сигнал з заданим коефіцієнтом заповнення до наступного виклику функції *analogWrite()* (або до моменту виклику *digitalRead ()* або *digitalWrite ()*, взаємодіючих з цим же виводом).

Практичне завдання до лабораторної роботи

Реалізуйте плавну зміну кольору, дослідіть як формуються різні кольори за допомогою RGB компонент.

Дослідіть рівномірність формування і сприйняття кольору у всьому діапазоні інтенсивностей.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, кнопки і провідники.
2. Під’єднайте кнопки одним контактом до виводу D8 ,D7, D6 Arduino, а іншим через резистор 1 кОм на спільний провід GND.
3. Підєднайте червоний канал RGB світлодіода до вивода 11, зелений до вивода 10, синій до вивода 9. Під’єднайте живлення +5В.
4. Скласти та завантажити в плату Arduino програму яка плавно змінює колір.

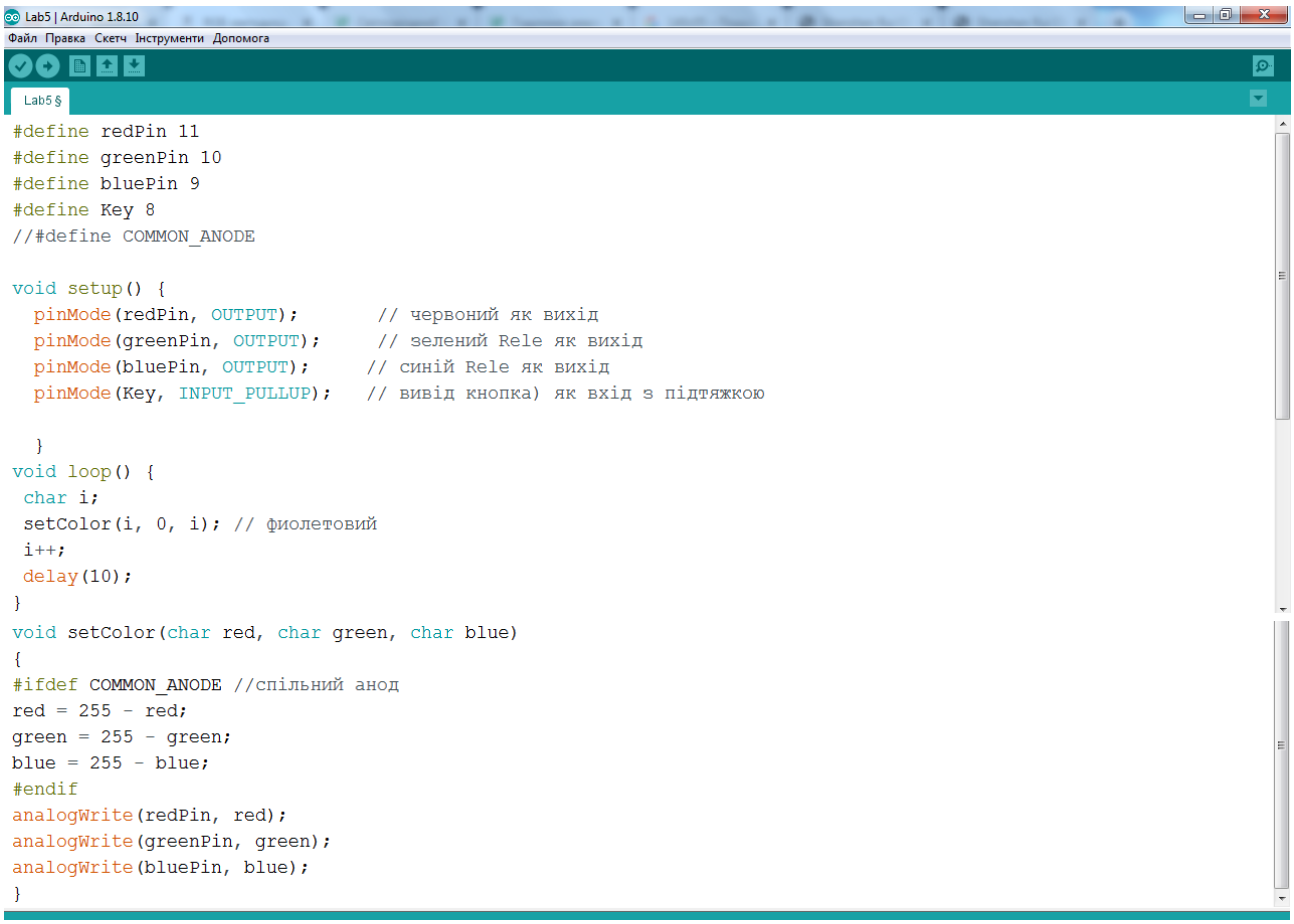


Рис. 4.2. Проста програма з використанням ШІМ.

5. Модифікувати програму щоб кольори плавно переливалися, червоний, синій, зелений.
6. Створіть програму щоб інтенсивність вибраного кольору можна було регулювати кнопками, також реалізуйте кнопку вибору каналу кольору.
7. Доповніть програму виводом інтенсивності кожного каналу в послідовний порт .
8. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Які переваги і недоліки RGB моделі кодування кольору.
2. Чому при однаковій інтенсивності змішування кольорів проходить з видимою похибкою.
3. Які недоліки ШІМ для формування аналогових сигналів.
4. Чому при використанні ШІМ можна добитися плавної зміни інтенсивності свічення світлодіода.
5. Які переваги використання ШІМ каналів таймера перед програмнов реалізацією ШІМ.

Лабораторна робота № 6.

Робота з I2C шиною передачі даних. Під’єднання рідкокристалічного символьного дисплея через I2C інтерфейс. Годинник реального часу.

Мета роботи: Вивчення апаратних засобів обміну даними МК, навчитися програмувати I2C інтерфецйс передачі даних.

Теоретичні відомості

Двопровідний послідовний інтерфейс TWI ідеально підходить для типових додатків на мікроконтролерах і вимагає тільки дві лінії зв'язку. Протокол TWI дозволяє проектувальнику системи зовні зв'язати до 128 різних пристроїв через одну двопровідну двосторонню шину, де одна лінія – лінія синхронізації SCL і друга – лінія даних SDA. В якості зовнішніх апаратних компонентів, які потрібні для реалізації шини, потрібен лише підтягуючий до плюса живлення резистор на кожній лінії шини. Всі пристрої, які підключені до шини, мають свої індивідуальні адреси, а механізм визначення вмісту шини підтримується протоколом.

Відмінні особливості:

- Гнучкий, простий, ефективний послідовний комунікаційний інтерфейс, що вимагає тільки дві лінії зв'язку;
- підтримка як Master (ведучого або майстра), так і Slave (веденого або підлеглого) режимів роботи;
- можливість роботи, як приймача, так і як передавача;
- 7-розрядний адресний простір дозволяє підключити до шини до 128 підлеглих пристроїв;
- швидкість передачі даних до 400 кГц;

схема шумозниження, яка підвищує стійкість до завад на лініях шини;
програмована адреса для Slave режиму з підтримкою загального виклику;

- пробудження мікроконтролера з режиму сну при визначенні заданої адреси на шині.

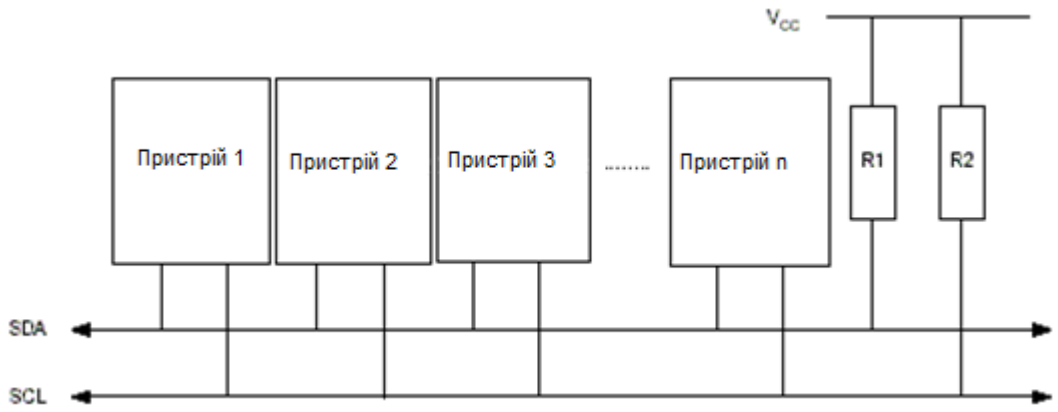


Рис. 6.1.Система протоколу TWI (I2C)

Як показано на рис. 6.1. [1], обидві лінії шини підключені до шини живлення через підтягуючі резистори. У всіх сумісних з TWI пристроях, як драйвер шини, використовуються транзистор або з відкритим стоком, або з відкритим колектором. Так реалізована функція, яка дуже важлива для двобічної роботи інтерфейсу. Низький логічний рівень на лінії шини TWI генерується, якщо один або більше з TWI-пристроїв виводить логічний 0.

Високий рівень на лінії присутній, якщо всі TWI-пристрої перейшли у третій високоімпедансний стан, дозволяючи підтягуючим резисторам задати рівень логічної 1. Зверніть увагу, що при підключенні до шини TWI декількох AVR мікроконтролерів, для роботи шини, усі ці мікроконтролери повинні бути підключені до живлення.

Кількість пристроїв, яка може бути підключено до однієї шини обмежується гранично допустимої ємністю шини (400 пФ) і 7-розрядним адресним полем. Підтримуються два різних набори технічних вимог, де один набір - для шин зі швидкістю передачі даних нижче 100 кГц, а інший дійсний для швидкостей понад 400 кГц.

Уся передача даних складається із стартової посилки, бітів і стопової посилки. Початок передачі визначається Start послідовністю – низький рівень SDA при високому рівні SCL.

При передачі інформації від Master до Slave, Master генерує такти на SCL і видає біти на SDA, які Slave зчитує коли SCL знаходиться в стані лог. “1”.

При передачі інформації від Slave до Master, Master генерує такти на SCL і відстежує лінією SDA – зчитує дані. А Slave, коли SCL прийме значення лог. “0”, виставляє на SDA біт, який Master зчитує коли SCL назад переходить з низького рівня на високий. Закінчується все STOP послідовністю. Коли при високому рівні на SCL лінія SDA переходить з низького на високий рівень.

Перший пакет від Master до Slave – це фізична адреса пристрою і біт напрямку. Сама адреса складається з семи біт (ось чому до 127 пристроїв на

шині), а восьмий біт означає що буде робити Slave на наступному байті - приймати або передавати дані. Дев'ятим бітом йде біт підтвердження ACK.

Якщо Slave розпізнав свою адресу повністю, то на дев'ятому такті він переведе лінію SDA в 0, згенерувавши ACK - тобто зрозумів. Тоді Master продовжить передачу даних. Якщо Slave не відповів, тобто SDA на дев'ятому такті не буде переведено в 0 (не буде ACK), то майстер припинить свої спроби підєднатися.

Після адресного пакета йдуть пакети з даними в ту або іншу сторону, в залежності від біта RW в заголовному пакеті

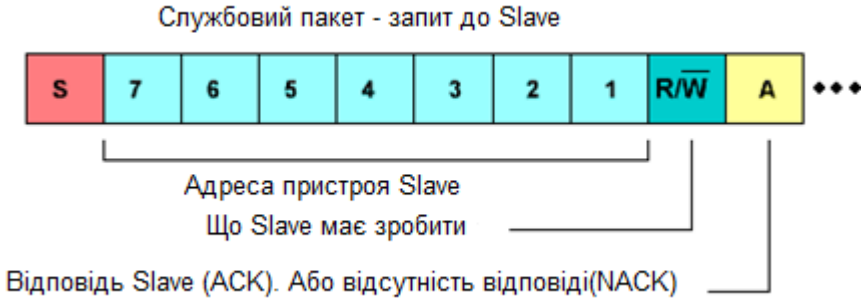


Рис. 6.2. Схема передачі пакетів

Робота з I2C/TWI на платах Arduino

На платах Arduino UNO, Nano, Pro Mini контакт SDA – A4, контакт SCL – A5.

Бібліотека Wire.h дозволяє Arduino взаємодіяти з різними пристроями по інтерфейсу I2C / TWI. Згідно з протоколом I2C, адреса пристрою може складатися як з 7, так і з 8 біт. Як правило, 7 біт ідентифікують пристрій, в той час, як восьмий біт задає напрям передачі даних: від пристрою (читання) або до нього (запис). Всі функції бібліотеки Wire.h використовують 7-бітну адресацію, тим самим обмежуючи діапазон можливих адрес в межах 0 – 127.

Функції I2C / TWI на платах Arduino:

Wire.begin()

Wire.begin(address)

- Параметри: *address* – 7-бітова адреса Slave-пристрою (необов'язковий параметр); якщо адреса не вказана, то Arduino виступає в ролі Master пристрою.
- Значення, що повертаються – немає.
- Опис: ініціалізує бібліотеку Wire і підключає Arduino до шини I2C в ролі ведучого (master) або веденого (slave) пристрою. Як правило, ця функція викликається тільки один раз.

2. Wire.requestFrom (address, quantity)

Wire.requestFrom (address, quantity, stop)

•Параметри: *address* – 7-бітова адреса відомого пристрою, у якого запитуються дані; *quantity*: кількість запитуваних байт; *stop*: boolean. При значенні true буде відправлений запит з стоповим бітом, що дозволить звільнити шину. При значенні false – з'єднання буде підтримуватися в активному стані.

•Значення, що повертаються – *byte*: кількість байт, повернутих веденим пристроєм.

•Опис: функція запрошує дані у веденого пристрою (slave); як правило, використовується тільки ведучим пристроєм (Master). Після виклику *requestFrom ()* запитувані дані повинні бути зчитані за допомогою функцій *available ()* і *read ()*.

Починаючи з версії Arduino 1.0.1, функція *requestFrom ()* може приймати третій параметр – логічне значення, що забезпечує кращу сумісність з деякими I2C-пристроями. Якщо цей параметр дорівнює true, то функція *requestFrom ()* відправить запит із стоповим бітом, що дозволить звільнити шину I2C. Якщо цей параметр дорівнює false, то після відправлення запиту шина як і раніше буде зайнята, що запобігає надсиланню сторонніх повідомлень іншими провідними пристроями. Цей режим дозволяє Master'у відправляти по декілька запитів за один сеанс. Значення за замовчуванням – true.

3. Wire.beginTransmission(address)

•Параметри: *address:* 7-бітова адреса пристрою

•Значення, що повертаються немає.

•Опис: починає процедуру передачі даних по інтерфейсу I2C веденому пристрою з вказаною адресою. Для подальшої відправки даних, необхідно спершу поставити їх в чергу за допомогою функції *write ()*, після чого здійснити, безпосередньо, передачу функцією *endTransmission ()*.

4. Wire.endTransmission()

Wire.endTransmission(stop)

•Параметри *stop*: boolean. При значенні true буде відправлений запит зі стоповим бітом, що дозволить звільнити шину. При значенні false – з'єднання буде підтримуватися в активному стані, що запобігає надсиланню сторонніх повідомлень іншими ведучими пристроями. Цей режим дозволяє ведучому відправляти по декілька запитів за один сеанс.

•Значення, що повертаються: byte, байт даних, що характеризує статус передачі:

0: передача успішна

1: обсяг даних занадто великий для буфера передачі

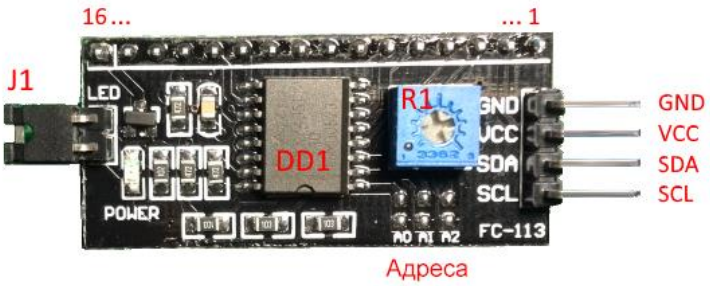
2: отриманий NACK при передачі адреси

3: отриманий NACK при передачі даних

Під’єднання LCD1602 через I2C до Arduino Uno

Дисплей LCD1602A може містити плату конвертер для перетворення паралельного 8-бітного інтерфейсу дисплея в шину I2C, по якій він і підключається до плати Arduino за адресою яка виставляється перемичками. Наявність послідовного інтерфейсу дозволяє спілкуватися з контролером Arduino за допомогою 2-х провідного зв’язку, це допоможе зменшити кількість використаних цифрових пінів контролера для підключення додаткової периферії.

Контактні площадки A1 ... A3 потрібні для зміни адреси I2C пристрою. Запаюючи відповідні перемички, можна змінювати адресу пристрою. На рис. 6.3 наведено відповідність адрес і перемичок: «0» відповідає відсутністю контакту, «1» - встановленій перемичці. За замовчуванням всі 3 перемички розімкнені і адреса пристрою 0x27.



A2	A1	A0	Адреса
1	1	1	**0x20**
1	1	0	**0x21**
1	0	1	**0x22**
1	0	0	**0x23**
0	1	1	**0x24**
0	1	0	**0x25**
0	0	1	**0x26**
0	0	0	**0x27**

Рис. 6.3. Відповідність адрес і перемичок модуля PCF8574.

Дисплей LCD 1602A I2C може одночасно відображати до 32 символів (16 символів, 2 рядки). Дисплей оснащений світлодіодним підсвічуванням, колір підсвітки може бути різним в залежності від модифікації.

Контролер дисплея HD44780 має ПЗП в якій зберігаються цифри, символи латиниці, для їх відображення на дисплеї. Відсутні символи, в тому числі і символи кирилиці, можна завантажувати в пам’ять ОЗП контролера.

Мікросхема може використовуватися для управління дисплеєм під управлінням контролера HD44780, в 4-х бітному режимі. Для цієї мети на платі встановлена мікросхема PCF8574, яка є перетворювачем шини I2C в паралельний 8 бітний порт.

Плата модуля розведена таким чином, щоб її можна було відразу підключити до дисплею. На платі одразу встановлені підтягуючі резистори на лініях SCL і SDA, потенціометр для регулювання контрастності дисплея та джампер який вмикає/вимикає підсвічування дисплею.

Нам потрібно мати додатковий модуль LCD1602 PK-конвертора в I2C. Він припаюється до дисплею як видно на рис 6.4, краще з нижньої сторони.

Схема підключення дисплея LCD1602 до Arduino Uno через I2C перехідник наведена на рис. 6.4.

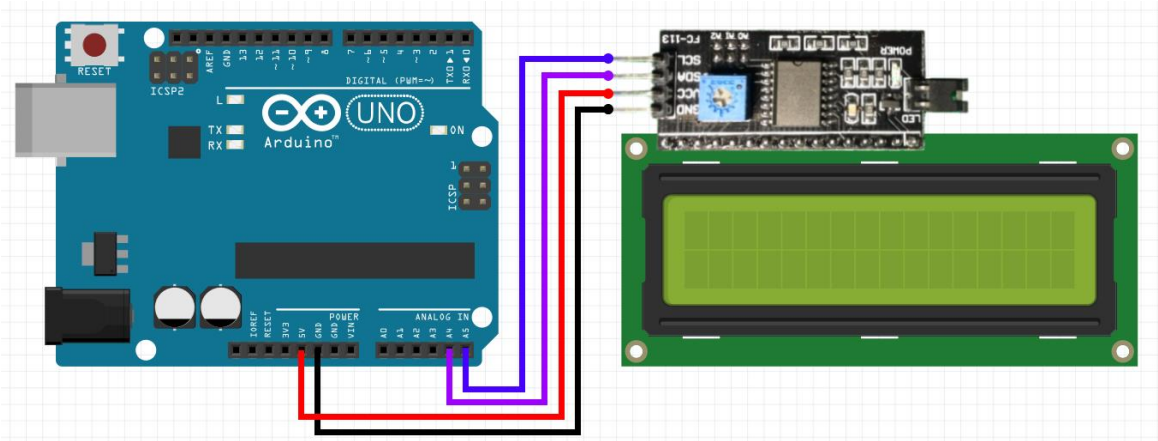


Рис. 6.4. Схема підключення дисплея LCD1602 через I2C перехідник.

Характеристики LCD 1602 I2C:

- Тип дисплея: рідкокристалічний символьний.
- Кількість символів в рядку: 16.
- Рядків: 2.
- Контролер HD44780.
- Світлодіодна підсвітка.
- Символьний тип відображення, є можливість загрузки символів.
- Кут огляду: 180 °.
- Напруга живлення: 5В.
- Робоча температура: від -20 °С до +70 °С.
- Розміри: 98 х 60 х 12 мм.
- Вага: 80 гр.

Використаний спосіб підключення в даній лабораторній роботі працює зі спеціально бібліотекою LiquidCrystal_I2C.h. Встановіть дану бібліотеку. Наведено нижче простий приклад програмного коду роботи з дисплеєм.

```
#include <Wire.h> // підключення бібліотеки роботи з I2C
#include <LiquidCrystal_I2C.h> //бібліотека роботи з LCD
LiquidCrystal_I2C      lcd(0x27,16,2);      //Налаштовуємо
параметри дисплея
void setup() {
  lcd.init(); //ініціалізуємо дисплей
  lcd.backlight();// Включаємо підсветку дисплея
```

```
  lcd.setCursor(0, 0); //поміщаємо курсор в 0 рядок 0 символ
  lcd.print("Laboratorna N6"); // виводимо надпис
  lcd.setCursor(3, 1); //поміщаємо курсор на 3 символ 1 рядок
  lcd.print("KI-31, 2020"); //виводимо надпис
}
void loop()
{
  // Встановлюємо курсор на 1 рядок 0 символ.
  lcd.setCursor(0, 1);
  // Виводимо на екран кількість секунд з моменту включення
  lcd.print(millis()/1000);
  delay(100)
}
```

Годинник реального часу DS1307

Мікроконтролер має апаратні таймери і з їх допомогою можна вимірювати інтервали часу, але всі ці таймери скидаються при знеструмленні мікроконтролера, тому є незручними і енергозатратними для вимірювання тривалих проміжків часу та реалізації годинника і календаря. Для цих задач випускається велика кількість спеціалізованих мікросхем, які називають годинниками реального часу. Такі мікросхеми реалізують годинник, календар та будильник, і можуть працювати від однієї батарейки CR2032 10 років. Зручно використовувати готові модулі уже з батарейкою, кварцовим резонатором, та підтягуючими резисторами на шині I2C. Одна з найпопулярніших моделей модуля годинника реального часу - DS1307 (рис. 6.4.). Модуль відмінно поєднується з мікроконтролерами Arduino, напруга живлення яких дорівнює 5 В.

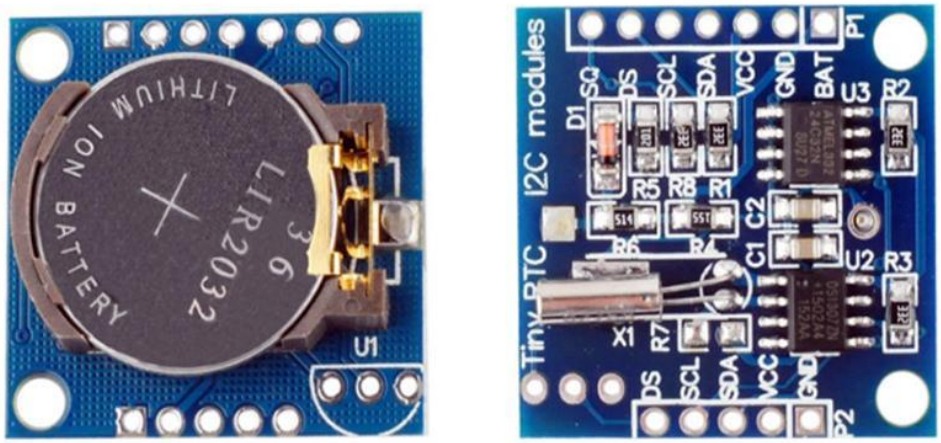


Рис. 6.4. Модуль годинника реального часу.

Модуль працює від автономного живлення - батарейки і продовжує вести відлік часу, навіть якщо на Вашому проекті на Arduino немає живлення. Використовуючи модуль реального часу, Ви можете відстежувати час, навіть якщо Ви захочете внести зміни у Ваш скетч і перепрограмувати мікроконтролер.

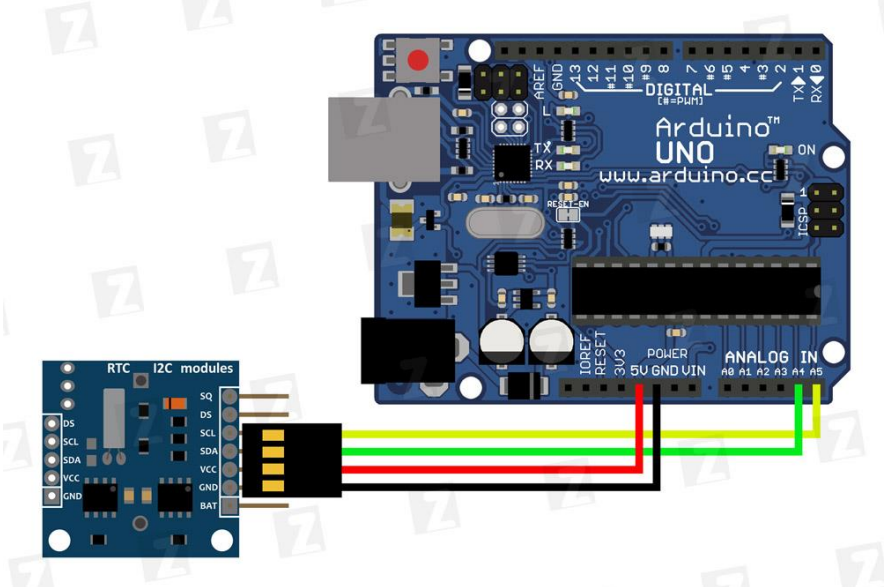


Рис. 6.4. Під’єднання модуля реального часу.

Необхідні бібліотеки:

для роботи з DS1307: http://www.pjrc.com/teensy/td_libs_DS1307RTC.html
для роботи з часом: http://www.pjrc.com/teensy/td_libs_Time.html

Приклад простої програми встановлення часу

Час задається вручну в програмному коді і завантажується в плату Arduino. Даний спосіб не дуже точний тому час на компіляцію і завантаження може займати різний часовий проміжок.

```
#include <Time.h> // бібліотека роботи з часом
#include <DS1307RTC.h> // бібліотека роботи з часом
void setup()  {
  Serial.begin(9600);
  setSyncProvider(RTC.get);   // отримуємо час з RTC
  if (timeStatus() != timeSet)
    Serial.println("Unable  to  sync  with  the  RTC");
//синхронізация не вдалася
  else
    Serial.println("RTC has set the system time");
  //встановити час вручну 16.02.2020 12:53
  TimeElements te; //об’єкт для збереження часу
  te.Second = 0; //секунди
  te.Minute = 53; //хвилини
  te.Hour = 12; //години
  te.Day = 16; //день
  te.Month = 2; // місяць
te.Year = 2020 - 1970; //рік в бібліотеці відраховується
з 1970
  time_t timeVal = makeTime(te);
  RTC.set(timeVal);
  setTime(timeVal);
}
void loop() {
//вивід часу
Serial.print(hour());
Serial.print(":")
Serial.print(minute());
Serial.print(":")
Serial.print(second());
  Serial.print(" ");
  Serial.print(day());
  Serial.print(" ");
  Serial.print(month());
  Serial.print(" ");
  Serial.print(year());
  Serial.println();
  delay(1000);
}
```

Практичне завдання до лабораторної роботи

Реалізуйте виведення дати і часу на рідкокристалічний дисплей LCD1602.

Випробуйте різні формати виведення, виберіть оптимальне на вашу думку розміщення дати і часу на дисплеї.

Запропонуйте методи підвищення точності встановлення часу.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, модуль дисплея та годинника реального часу і провідники.
2. Підєднайте дисплей згідно рис 6.3. та перевірте роботу прикладу наведеного в теоретичних відомостях.
3. Під'єднайте модуль реального часу згідно рис 6.5. та перевірте роботу прикладу наведеного в теоретичних відомостях.
4. Використовуючи макетну плату підєднайте паралельно модулі дисплея і реального часу та напишіть програму виведення часу та дати на дисплей.
5. Оптимізуйте програму і вивід даних.
6. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Які переваги та недоліки використання перехідника для роботи рідкокристалічного дисплея LCD1602 через протокол I2C?
2. Які особливості використання шини I2C?
3. Чому можливе паралельне підключення дисплея і годинника реального часу і як при цьому вирішуються конфлікти використання шини обома пристроями?
4. Які функції ведучого присторю на шині I2C?

Лабораторна робота № 7.

Однопровідна двонаправлена шина передачі даних 1-Wire. Побудова цифрового термометра з використанням рідкокристалічного символьного дисплея LCD1602.

Мета роботи: Вивчення апаратних засобів обміну даними МК, навчитися програмувати 1-Wire інтерфецйс передачі даних.

Теоретичні відомості

Інтерфейс 1-Wire був запропонований фірмою Dallas Semiconductor в кінці 90-х років минулого століття. Системи 1-Wire привабливі завдяки легкості монтажу, низькій вартості пристроїв, можливості вибирати користувача при підключенні до функціонуючої мережі, великому числу пристроїв в мережі і т.д. Типова система 1-Wire складається з керуючого контролера (майстра або ведучого) і одного або декількох пристроїв (ведених), під'єднаних до загальної шини (рис.7.1) [1].

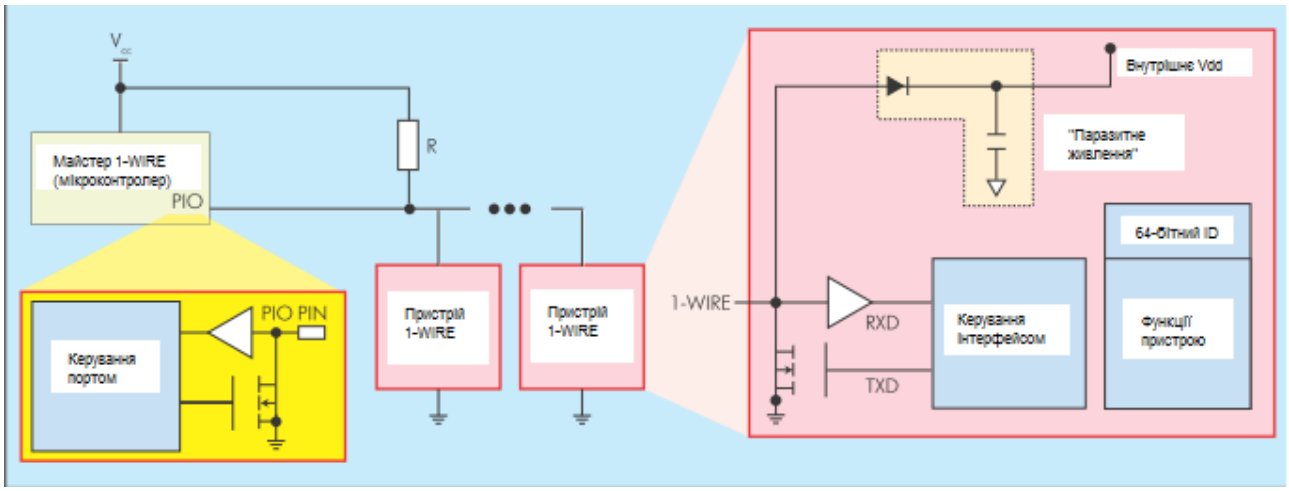


Рис.7.1. Конфігурація системи 1-Wire.

Пристрої підключаються до шини за схемою з відкритим колектором і підтягуючим резистором. Рівень сигналів в шині - від 3 до 5 В. У пасивному стані в лінії підтримується високий рівень напруги. Всі сигнали формуються за допомогою замикання сигнальної шини на землю (низький рівень напруги).

Головна особливість шини 1-Wire в тому, що вона використовує лише два дроти, один – сигнальний, інший – для заземлення пристроїв. По сигнальному проводу можливе і електроживлення пристроїв 1-Wire – так зване паразитне живлення. Джерелом живлення служить конденсатор, який заряджається від сигнальної лінії, що входить до складу відомих пристроїв кола. Більшість

пристроїв 1-Wire підтримують дві швидкості передачі даних: стандартну – близько 15 кбіт / с і підвищену (overdrive) – близько 111 кбіт / с. Зрозуміло, що чим вище швидкість, тим більше обмежень на довжину шини і число під'єднаних до неї пристроїв.

Режим передачі даних по шині 1-Wire – напівдуплексний: майстер і ведені пристрої передають дані по черзі. Кожна транзакція через інтерфейс 1-Wire починається з того, що майстер передає імпульс Reset. Для цього він переводить напругу в шині на низький рівень і утримує його в цьому стані протягом 480 мкс (рис.7.2) [1]. Потім майстер відпускає шину, і підтягуючий резистор повертає напругу до високого логічного рівня. Всі ведені пристрої, виявивши сигнал Reset і дочекавшись його закінчення, передають свій сигнал - Presence. Він являє собою сигнал низького рівня тривалістю 100-200 мс. Пристрій може генерувати сигнал Presence і без імпульсу Reset – наприклад, у такий спосіб він повідомляє про себе при підключенні до шини.

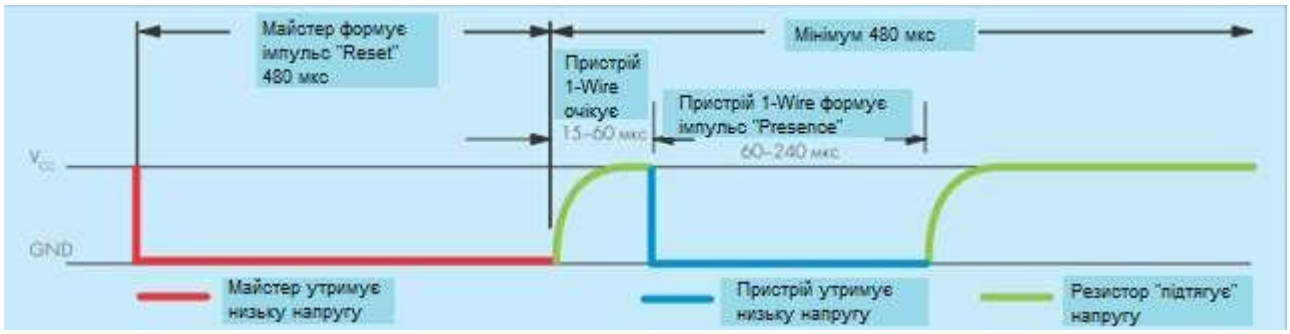


Рис.7.2. Послідовність ініціалізації шини 1-Wire.

Після передачі імпульсу Presence пристрій 1-Wire готовий до прийому команд. Весь інформаційний обмін в шині відбувається під керуванням майстра. Для передачі кожного біта виділяється спеціальний часовий проміжок (таймслот) тривалістю близько 80 мкс. На початку кожного таймслота майстер переводить лінію на нульовий рівень. Якщо далі майстер хоче передати лог. "0", він утримує напругу на низькому рівні як мінімум 60 мкс (рис.7.3) [1]. При передачі одиниці майстер утримує нульову напругу 5-6 мкс, а потім відпускає лінію і вичікує приблизно 60 мкс до початку формування наступного таймслота.

Якщо майстер очікує дані від ведених пристроїв, він також позначає початок таймслота, обнулити лінію на 5-6 мкс, після чого перестає утримувати низьку напругу і протягом короткого часу відстежує стан лінії (рис.7.3, б). Якщо пристрій хоче передати нуль, то обнуляє лінію відразу після реєстрації імпульсу початку тайм-слота. Якщо пристрою потрібно передати одиницю, він ніяких дій не робить. Відзначимо, що наведені значення тимчасових інтервалів

відповідають стандартній швидкості передачі даних через інтерфейс 1-Wire. У режимі overdrive ці інтервали відповідно зменшуються.

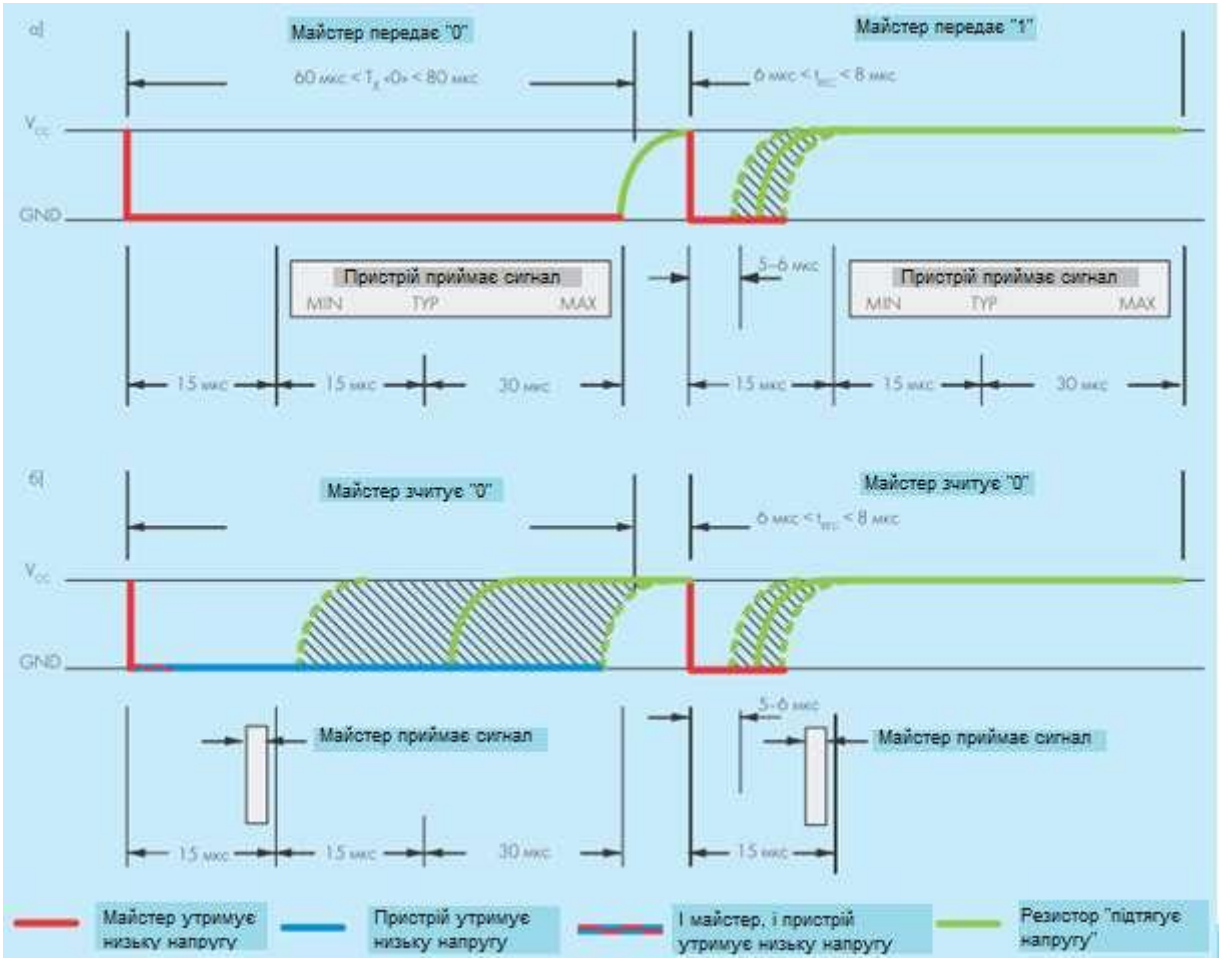


Рис.7.3. Передача інформаційних бітів по шині 1-Wire.

Весь обмін на шині 1-Wire відбувається за допомогою спеціальних команд. Їх число для кожного типу пристроїв може бути різне. Але є і мінімальний набір стандартних команд, які підтримують всі 1-Wire-пристрої – так звані ROM-команди.

Формат команд простий – ідентифікатор команди (1 байт), за яким можуть слідувати дані (ідентифікатор пристрою, корисні дані і т.п.). Всі пристрої в мережі знають довжину кожної команди. У кожного пристрою 1-Wire є 64-розрядний ідентифікатор (ID). Він складається з 8-розрядного коду сімейства, який ідентифікує тип пристрою та функції, які він підтримує, 48-розрядної серійного номера і 8-бітного поля коду циклічного надлишкового контролю (CRC-8). ID вводиться при виготовленні пристрою і зберігається в ПЗП.

Увесь обмін командами ініціює майстер. Початок нового циклу транзакцій він зазначає командою Reset, і, отримавши підтвердження, вибирає пристрій спеціальною командою MATCH ROM, передаючи її ідентифікатор

(5516) і 64 біта ID адресованому пристрою. Отримавши таку команду, ведений пристрій з даним ID очікує нових команд від майстра, а всі інші залишаються в пасивному стані до наступної команди Reset. В системі з одним пристроєм можна не передавати ID, використовуючи команду SKIP ROM. В результаті ведений пристрій вважає себе обраним без отримання адреси. Після того, як Майстер вибрав пристрій для взаємодії, можна починати процес управління цим пристроєм і обміну даними з ним. Для цього використовуються команди, які специфічні для кожного типу пристроїв. Але щоб почати роботу з певним пристроєм, майстер повинен знати його ID. Якщо в системі тільки один Ведений пристрій, його адресу можна визначити за допомогою команди READ ROM. У відповідь на команду READ ROM пристрій передає свою 64-бітову адресу (рис.7.4) [1].

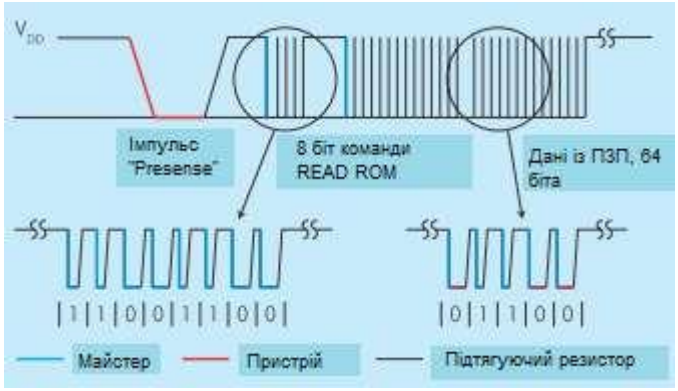


Рис.7.4. Читання адреси пристрою.

Якщо ж в системі декілька пристроїв з невідомими ID, спроба використати команду READ ROM призводить до колізії. У цьому випадку для визначення адреси використовується спеціальний алгоритм пошуку, в основі якого лежить команда SEARCH ROM. Майстер передає команду SEARCH ROM. У відповідь всі пристрої, підключені до шини, висилають молодший біт своєї адреси. Властивості шини 1-Wire такі, що при одночасній передачі сигналів усіма пристроями результат буде дорівнювати логічній 1 значень всіх надісланих бітів. Отже, сумарний відгук дорівнює “1”, тільки коли сигнали від всіх пристроїв рівні 1. Після прийому першого біта адреси, Майстер ініціює наступний таймслот, в якому пристрій передає інвертований перший біт. Порівнюючи значення результатів запиту істинного і інверсного бітів, можна отримати певну інформацію про значеннях перших бітів адрес пристроїв (див. табл. 7.1) [1].

Табл. 7.1 – Значення бітів адрес пристроїв

Істинний біт	Інверсний біт	Інформація
0	**0**	У поточному біті адреси є як 0, так і 1. Це так звана «розбіжність» (discrepancy)

0	**1**	У біті адреси присутні тільки нулі.
1	**0**	У біті адреси присутні тільки одиниці
1	**1**	У пошуку не бере участі жоден пристрій

Таким чином, при комбінаціях 0 1 і 1 0 майстер знає значення першого біта адреси, фіксує його і по тій же схемі може переходити до визначення наступного. Після отримання інверсного біта майстер передає певний біт веденим пристроям. Якщо його значення збігається із значенням поточного біта з адреси пристрою, то пристрій продовжує брати участь в пошуку і видає у відповідь наступний біт своєї адреси. Якщо не було «розбіжності», то значення, що виставляється майстром біта визначено. У разі розбіжності майстер посилає нульовий біт. Така послідовність – читання біта адреси та інверсного біта, передача біта майстром – повторюється для наступних 63 бітів адреси. Таким чином, алгоритм пошуку послідовно виключає всі пристрої, поки не залишається один останній – його адреса і визначається в першому циклі пошуку.

Після того, як адресу першого пристрою визначено, пошук триває для наступного пристрою. Алгоритм запам'ятовує місце останньої розбіжності івибирає іншу гілку дерева пошуку (майстер посилає в цьому місці біт з іншим значенням). Процес триває до тих пір, поки не буде пройдена гілка, відповідна останньому пристрою. В результаті пошуку стають відомі адреси всіх пристроїв, під'єднаних до шини та їх число. Можливість ідентифікації і швидкого включення в мережу тільки-но підключеного пристрою, робить 1-Wire ефективним рішенням для багатьох додатків. На практиці це означає, що прилад досить просто підключити до мережі, і всі подальші транзакції відбудуться автоматично. Наприклад, так можна прочитати дані з пам'яті датчика, прочитати код електронної мітки або електронного ключа, прийняти масив значень від приладової мережі і т.п.

Не менш важливо, що мережа 1-Wire не вимагає окремої лінії для передачі тактових сигналів. І, звичайно, величезне число ID пристроїв, що підключаються вигідно виділяє її на тлі інших послідовних мереж.

Наявність унікальних 64-бітних адрес дозволяє широко використовувати пристрої 1-Wire в системах аутентифікації. Такі пристрої функціонують, наприклад, в домофонних ключах. Мікросхеми з підтримкою 1-Wire (наприклад, DS2401, DS2431, DS28E01-100) використовуються також для ідентифікації картриджів принтерів, медичних сенсорів, ємностей з реагентами та ін. Перевага мікросхем 1-Wire в тому, що для контролю ідентифікованого пристрою потрібен всього один контакт. Ще одне поширене застосування 1-Wire - системи автоматизації. В першу чергу це системи багатоточкового

вимірювання температури різних середовищ і моніторингу теплового режиму приміщень.

Програмування 1-Wire

Для початку роботи, необхідно імпортувати бібліотеку OneWire.h.

OneWire myWire (pin)

За допомогою зазначеного виводу створюється об'єкт OneWire. Хоча до одного виводу можна підключити відразу декілька пристроїв, найчастіше це важко реалізувати. Тому, якщо використовується кілька пристроїв, треба підключати їх групами не до одного, а до кількох виводів. Ви можете створити кілька об'єктів OneWire - по одному на кожен вивід.

- ***myWire.search (addrArray)***

Шукає наступний пристрій. Масив *addrArray* - це масив з 8 байтів. Якщо пристрій виявлено, *addrArray* заповнюється адресою цього пристрою, після чого функція повертає true. Якщо більше пристроїв не знайдено, функція повертає false.

- ***myWire.reset_search ()***

Починає новий пошук. Наступне використання пошуку почнеться на першому пристрої.

- ***myWire.reset ()***

Робить скидання шини 1-Wire. Як правило, це потрібно перед комунікацією з будь-яким пристроєм.

- ***myWire.select (addrArray)***

Вибирає пристрій, чия адреса вказана в дужках. Потрібно після скидання - щоб вибрати пристрій, який потрібно використовувати. Подальша комунікація буде саме з цим пристроєм, але до нового скидання.

- ***myWire.skip ()***

Пропускає вибір пристрою. Це працює лише тоді, коли підключено лише один пристрій, тобто ви можете не використовувати пошук, а відразу отримати доступ до вашого пристрою.

- ***myWire.write (num)***

Записує байт.

- ***myWire.write (num, 1)***

Записує байт і залишає включеним паразитне живлення, що підключене до шини 1-Wire.

- ***myWire.read ()***

Зчитує байт.

- ***myWire.crc8 (dataArray, length)***

Розраховує перевірочний CRC для масиву даних.

DS18B20 це цифровий вимірювач температури, з дозволом перетворення 9 - 12 розрядів і функцією тривожного сигналу контролю за температурою. Параметри контролю можуть бути задані користувачем і збережені в енергонезалежній пам'яті датчика.

Датчик температури DS18B20

DS18B20 обмінюється даними з мікроконтролером за однопровідною лінією зв'язку, використовуючи протокол інтерфейсу 1-Wire.

Живлення датчик може отримувати безпосередньо від лінії даних, без використання зовнішнього джерела. В цьому режимі живлення датчика відбувається від енергії, яка запасається при передачі даних на «паразитій» ємності.

Діапазон вимірювання температури становить від -55 до +125 ° С. Для діапазону від -10 до +85 ° С похибка не перевищує 0,5 ° С.

У кожної мікросхеми DS18B20 є унікальний серійний код довжиною 64 розряди, який дозволяє декільком датчикам підключатися на одну загальну лінію зв'язку. Тобто через один порт мікроконтролера можна обмінюватися даними з декількома датчиками, розподіленими на значній відстані. Режим вкрай зручний для використання в системах екологічного контролю, моніторингу температури в будівлях, вузлах устаткування.

Основна функція DS18B20 - перетворення температури датчика в цифровий код. Роздільна здатність перетворення задається 9, 10, 11 або 12 біт. Це відповідає роздільній здатності - 0,5 (1/2) °С, 0,25 (1/4) °С, 0,125 (1/8) °С і 0,0625 (1/16) °С. При включенні живлення, стан регістра конфігурації встановлюється на роздільну здатність 12 біт.

Після включення живлення DS18B20 знаходиться в стані пониженого енергопоживання. Щоб ініціювати вимірювання температури мікроконтролер повинен виконати команду ПЕРЕТВОРЕННЯ ТЕМПЕРАТУРИ [44h]. Після завершення перетворення, результат вимірювання температури буде знаходитися в 2 байтах регістра температури, і датчик знову перейде в стан спокою.

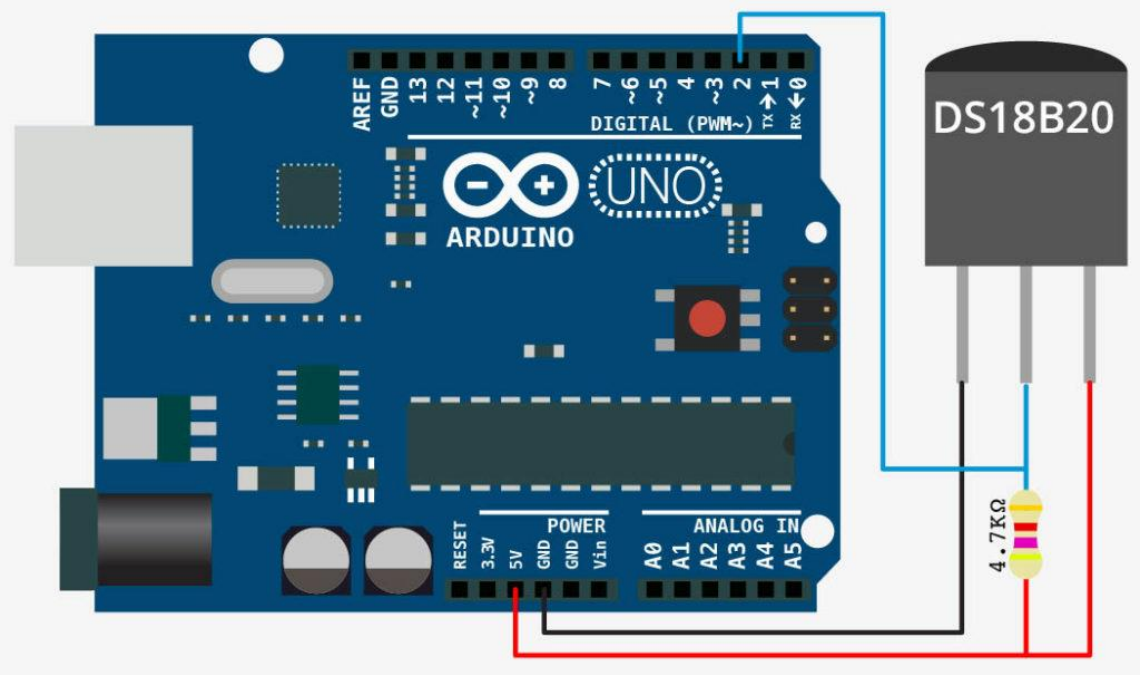


Рис 7.5. Під’єднання датчика температури.

Приклад програми для зчитування даних з датчика:

```
#include <OneWire.h>
OneWire ds(8); // Створюємо об'єкт OneWire для шини 1-Wire, робота з датчиком через вивід 8
void setup(){
Serial.begin(9600);
}
void loop(){
byte data[2]; // Місце для значення температури
ds.reset(); // Починаємо взаємодію зі скидання всіх попередніх команд і параметрів
ds.write(0xCC); // Даємо датчику DS18b20 команду пропустити пошук за адресою. У нашому випадку тільки однин пристрій
ds.write(0x44); // Даємо датчику DS18b20 команду виміряти температуру. Саме значення температури ми ще не отримуємо - датчик його збереже у внутрішню пам'ять
delay(1000); // Мікросхема вимірює температуру, а ми чекаємо.
ds.reset(); // Тепер готуємося отримати значення виміряної температури
ds.write(0xCC);
```

```
ds.write(0xBE);   // Просимо передати нам значення регістрів та температура
// Отримуємо і зчитуємо відповідь
data[0] = ds.read(); // Читаємо молодший байт значення температури
data[1] = ds.read(); // А теперь старший
// Формуємо підсумкове значення:
// - спершу "склеюємо" значення,
//  - потім множимо його на коефіцієнт, відповідний роздільної здатності (для 12 біт за замовчуванням - це 0,0625)
float temperature = ((data[1] << 8) | data[0]) * 0.0625;
// Виводимо отримане значення температури в монітор порту
Serial.println(temperature);
}
```

Практичне завдання до лабораторної роботи

Реалізуйте виведення температури на рідкокристалічний дисплей LCD1602.

Оптимізуйте програмний код, реалізуйте зчитування температури без використання функції паузи.

Виведіть графік зміни температури від часу за допомогою вбудованого інструмента «Послідовний плотер».

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, модуль дисплея, датчник температури і провідники.
2. Під’єднайте дисплей згідно рис 6.3. і датчик температури згідно рис. 7.5.
3. Напишіть програму виведення часу та дати на дисплей.
4. Оптимізуйте програму і вивід даних.
5. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.

5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Які переваги і недоліки використання шини OneWire?
2. Які особливості використання шини OneWire?
3. Чому можливе паралельне під’єднання датчиків та як у цьому випадку мінімізуються конфлікти при обміні даними?
4. Які функції ведучого пристрою на шині OneWire?
5. Які основні переваги і недоліки використання датчика температури DS18b20.

Лабораторна робота № 8.

Робота із сенсором температури та вологості DHT22 і програмування протоколу обміну даних з ним в середовищі Arduino IDE.

Мета роботи: засвоїти основні принципи роботи давачів визначення температури та вологості DHT11 і DHT22, навчитись під'єднувати їх до макетної плати Arduino UNO та засвоїти основні принципи програмування їх роботи в середовищі Arduino IDE.

Теоретичні відомості

Для контролю за температурою і вологістю повітря можна використовувати давачі DHT. Існують дві версії сенсорів DHT: DHT11 та DHT22 (рис. 8.1). Виглядають вони майже однаково. Терморегулятори теж однакові. Основні відмінності – в технічних характеристиках:

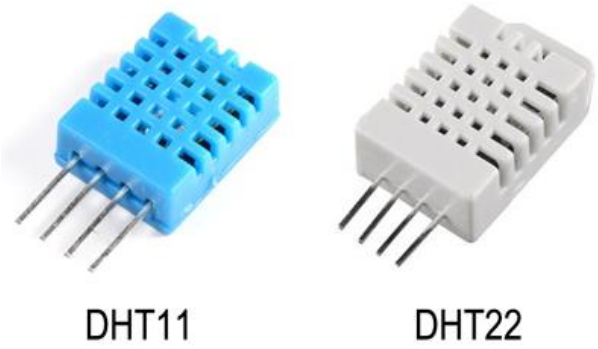


Рис. 8.1. Давачі температури та вологості DHT11 та DHT22.

DHT11:

- дуже дешевий;
- живлення від 3 до 5 В;
- максимально споживаний струм – 2.5мА при запиті даних;
- розрахований на вимірювання рівня вологості в діапазоні від 20% до 80%. При цьому точність вимірювань знаходиться в діапазоні 5%.
- вимірює температуру в діапазоні від 0 до 50 градусів з точністю +/- 2%;
- частота вимірів не більше 1 Гц (один вимір в секунду).
- розмір корпусу: 15.5 мм х 12 мм х 5.5 мм.

DHT22:

- дешевий;
- живлення від 3 до 5 В;
- максимально споживаний струм – 2,5 мА при запиті даних;

- розрахований на вимірювання рівня вологості в діапазоні від 0% до 100%. При цьому точність вимірювань знаходиться в діапазоні 2% - 5%;
- вимірює температуру в діапазоні від -40 до 125 градусів з точністю плюс-мінус 0,5 градусів за Цельсієм;
- частота вимірів до 0.5 Гц (один вимір за 2 секунди).

Згідно характеристик, давач DHT22 більш точний і має більший діапазон вимірюваних значень. Обидва давачі мають по одному цифровому виходу. Запити до них можна відправляти не частіше ніж один або два рази в секунду. Дані давачі не виділяються особливою швидкодією і точністю, але вони прості у використанні та вирізняються досить невисокою вартістю.

Давач складається з двох частин – ємнісного давача температури і гігрометра. Перший використовується для вимірювання температури, другий – для вологості повітря. Чіп, який знаходиться всередині, може виконувати аналого-цифрові перетворення і видавати цифровий сигнал, який зчитується за допомогою мікроконтролера.

Це цифрові датчики засновані на протоколі, який для зв'язку використовує один провід з відкритим колектором, тому обов'язкова підтяжка резистором 5-10кОм до плюса живлення.

Мікроконтролер виступає в ролі ведучого пристрою шини і, отже, відповідає за ініціювання зв'язку. Датчик вологості і температури DHT11 завжди залишається в якості підлеглого пристрою і відповідає даними, коли MCU виконує запит. Протокол, який використовується для зв'язку, простий і може бути узагальнено наступним чином:

Рис. 8.2. Обмін даними.

Процес обміну

Мікроконтролер подає стартовий сигнал – «притискає» шину до землі на 18-20 мс;

Після цього контролер відпускає лінію і стежить за рівнем на ній, приблизно 20-40 мкс;

Датчик, виявивши сигнал і почекавши поки рівень знову стане високим, сам «притискає» шину до землі на 80 мкс, за цей час робляться вимірювання і перетворення результатів;

Потім DHT11 відпускає лінію на 80 мкс, що вказує на те, що він готовий відправити дані;

Потім він відправляє 40 біт даних. Перед відправкою кожного біта датчик «притискає» шину до землі на 50 мкс, за яким слід 26-28 мкс для «0» або 70 мкс для «1»;

По завершенні зв'язку лінія витягується підтягуючим резистором і переходить в стан очікування.

Формат даних DHT11 / DHT22

Коли датчик вологості і температури відправляє дані, він спочатку відправляє MSb (Most Significant Bit) - старший значущий біт. Дані від датчика передаються у вигляді посилки, що складаються з 40 біт даних - це 5 байт з яких перших два вологість, наступні 2 температура і байт парності. Байт парності дорівнює сумі попередніх байт. 1 і 2 байт містять відповідно цілу і дробову частину інформації про вологість, 3 і 4 байт містять цілу і дробову частину інформації про температуру. Для датчика DHT11 2-й і 4-й байти завжди нуль. Значення цих байтів полягає в наступному:

1-й байт: відносна вологість - ціла частина в%;

2-й байт: десята частина відносної вологості в% (нуль для DHT11);

3-й байт: ціла частина температури в ° С;

4-й байт: десята частина температури в ° С (нуль для DHT11);

5-й байт: контрольна сума (останні 8 біт {1-й байт + 2-й байт + 3-й байт + 4-й байт})

Практичне завдання до лабораторної роботи

Реалізуйте виведення вологості і температури на рідкокристалічний дисплей LCD1602.

Виведіть графік зміни температури і вологості від часу за допомогою вбудованого інструменту «Послідовний плотер».

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, давач температури та вологості DHT22 і 3 провідники.
2. Виконати під'єднання давача температури і вологості DHT22 до макетної плати згідно схеми підключення (рис. 8.3) та принципової схеми (рис. 8.4)

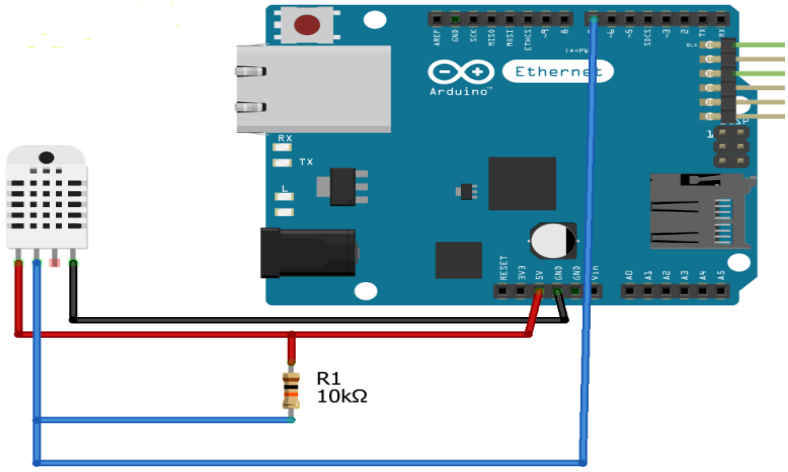


Рис. 8.3. Схема підключення давача DHT11 (DHT22)

Принципова схема підключення давача DHT11 (DHT22) до плати Arduino UNO наведена на рис. 8.4.

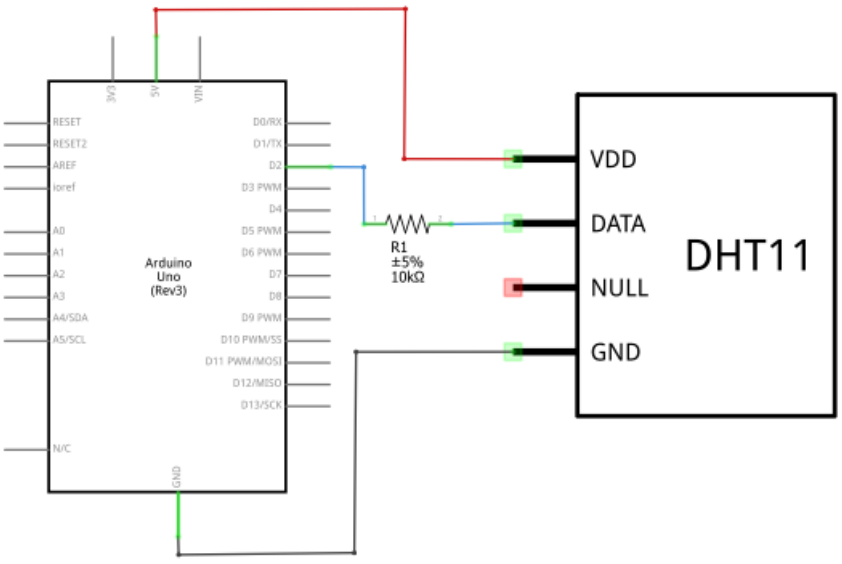


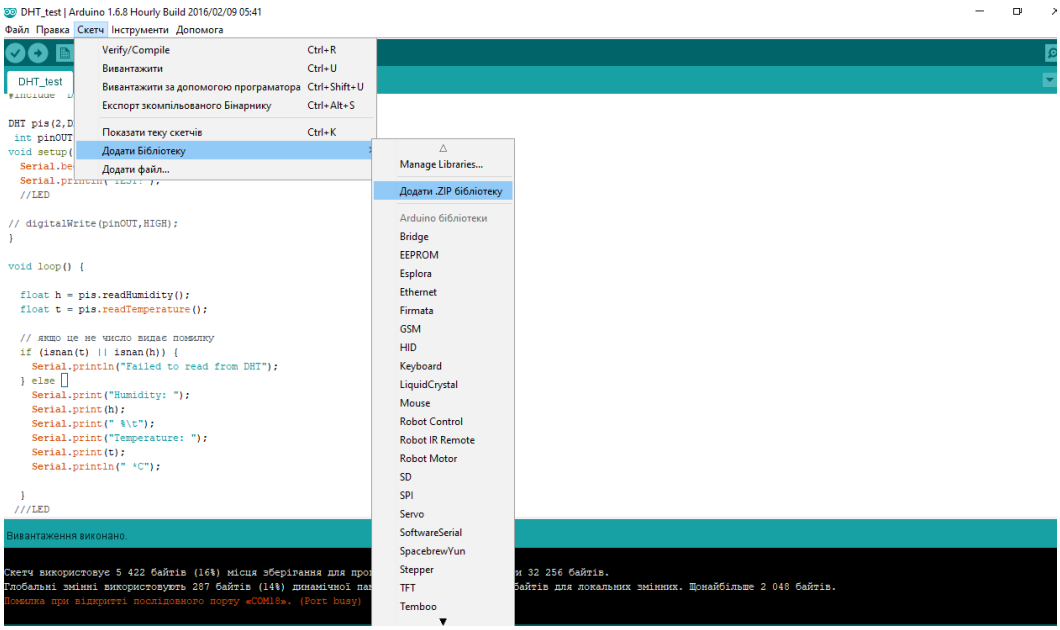
Рис. 8.4. Принципова схема підключення давача DHT11 (DHT22) до Arduino UNO

3. Запустити середовище Arduino IDE. Створити новий файл зайшовши у меню Файл -> Створити.

4. Вставити наступний програмний код у створений файл та зберегти його.

```
#include "DHT.h"
DHT pis(2,DHT22); //який номер піна використати, тип давача
 int pinOUT = 7 ;
void setup() {
  Serial.begin (57600); //швидкість передачі даних(біт/с)
  Serial.println("TEST!");
  //LED
// digitalWrite (pinOUT, HIGH);
}
void loop() {
  float h = pis.readHumidity();
  float t = pis.readTemperature();
  // якщо це не число видає помилку
  if (isnan(t) || isnan(h)) {
    Serial.println("Failed to read from DHT");
  } else {
    Serial.print("Humidity: ");
    Serial.print(h);
    Serial.print(" %\t");
    Serial.print("Temperature: ");
    Serial.print(t);
    Serial.println(" *C");
   }
 ///LED
  if (h>=50)
  {
          digitalWrite(pinOUT,LOW);
    delay(1000);
    pinMode(pinOUT,OUTPUT);
    delay(1000);
    digitalWrite(pinOUT,HIGH);
      }
}
```

5. Зайти у меню Скетч ->Додати Бібліотеку ->Додати ZIP бібліотеку та вибрати zip-файл бібліотеки для DHT-сенсора.



6. Виконуємо перевірку (компіляцію) програмного коду натиснувши клавішу .

14. Наступним кроком під’єднуємо плату Arduino UNO і завантажуємо програму в мікроконтролер макетної плати Arduino UNO натиснувши кнопку .
15. Перевіряємо роботу програми
16. Напишіть програму для виведення вологості та температури на LCD десплей.
17. Оптимізуйте програму для виведення графіка вологості та температури за допомогою інструменту «Послідовний плотер».

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Які переваги і недоліки використання нестандартної однопровідної шини передачі даних датчика DHT11 у порівнянні з шиною OneWire?
2. Який принцип роботи датчика DHT11?
3. Чи можливе паралельне підключення декількох датчиків DHT11?

Лабораторна робота № 9.

АЦП. Робота з аналоговими сигналами мікроконтролера. Проектування підсистеми пожежної сигналізації розумного будинку з використанням аналогового давача газу.

Мета роботи: засвоїти основні принципи роботи з аналоговими входами та АЦП мікроконтролера. Навчитися опрацьовувати сигнали аналогового давача газу на прикладі MQ-2.

Теоретичні відомості

На платі Arduino є 8-канальний 10-бітний АЦП (аналогово-цифровий перетворювач), який перетворює вхідну напругу з діапазону 0 – 5В у цілочисельні значення в межах від 0 до 1023 відповідно. Роздільна здатність АЦП становить: 5 В/1024 значення або 0.0049 В (4.9 мВ) на одне значення. Вхідний діапазон задається опорною напругою АЦП і може змінюватися за допомогою функції analogReference ().

Для зчитування значення з аналогового входу потрібно близько 100 мікросекунд (0.0001 с), тому максимальна частота опитування виведення приблизно дорівнює 10 000 разів на секунду. Якщо аналоговий вхід ні до чого не підключений, значення, що повертається функцією analogRead (), буде випадковим. Змінюватиметься під впливом декількох факторів: величина напруги на інших аналогових входах, електромагнітних завад і т.д.

Давач газу MQ-2.

Давач газу, побудований на базі газоаналізатора MQ-2 (рис. 9.1) та дозволяє виявляти наявність в навколишньому повітрі вуглеводневих газів (пропан, метан, бутан), диму, водню.

Давач можна використовувати для виявлення витоків промислового газу і задимлення. В результаті спрацювання давача на його виході утворюється аналоговий сигнал, пропорційний вмісту газів, до яких чутливий газоаналізатор. Чутливість може бути налаштована за допомогою змінного резистора на платі давача.

Рис. 9.1. Модуль давача диму MQ-2

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

Давач під'єднується до контролера через 3 виводи. Схема під'єднвння давача до Arduino UNO наведена на рис. 9.2. Принципова схема наведена на рис. 3.

Характеристики модуля MQ-2:

- напруга живлення: 5 В;
- струм: 160 мА.

Діапазон вимірювань:

- Пропан: 0,2-5 проміле;
- Бутан: 0,3-5 проміле;
- Метан: 5-20 проміле;
- Водень: 0,3-5 проміле;
- Пари спиртів: 0,1-2 проміле.

На базі такого давача можна реалізувати простий сигналізатор загазованості, задимлення.

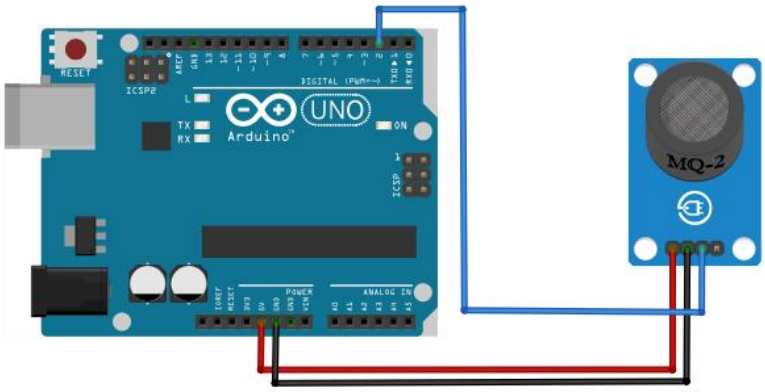


Рис. 9.2. Схема підключення модуля MQ-2 до Arduino UNO.

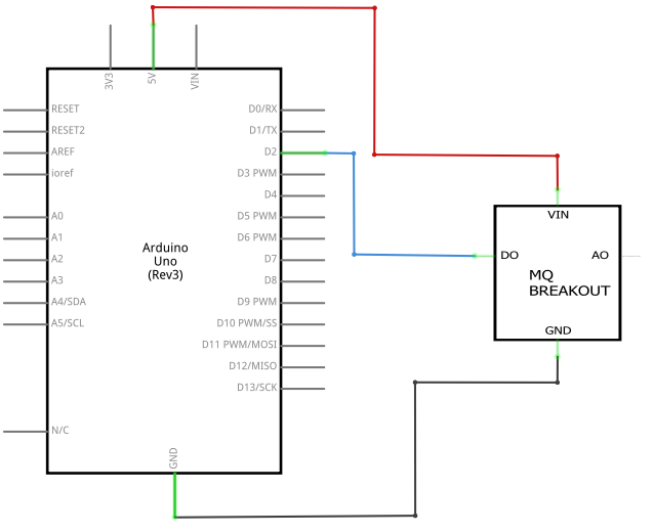


Рис. 9.3. Принципова схема підключення модуля MQ-2 до Arduino UNO

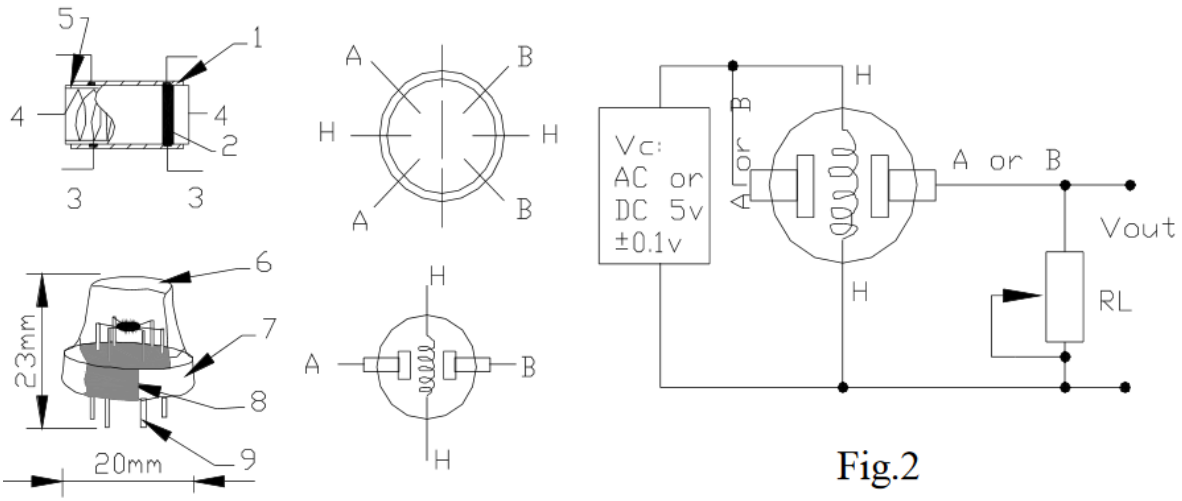


Рис. 9.3. Будова давача газу.

1. Зондування газу шар (SnO_2), 2. Електрод (Au), 3. Електроди (Pt), 4. Нагрівальна котушка сплаву Ni-Cr, 5. Трубчаста керамічна (Al_2O_3), 6. Антивибухова сітка (Сітка з нержавіючої сталі), 7. Затискне кільце (мідь покрита Ni), 8. Основа (Бакеліт), 9. Мідний вивід покритий Ni

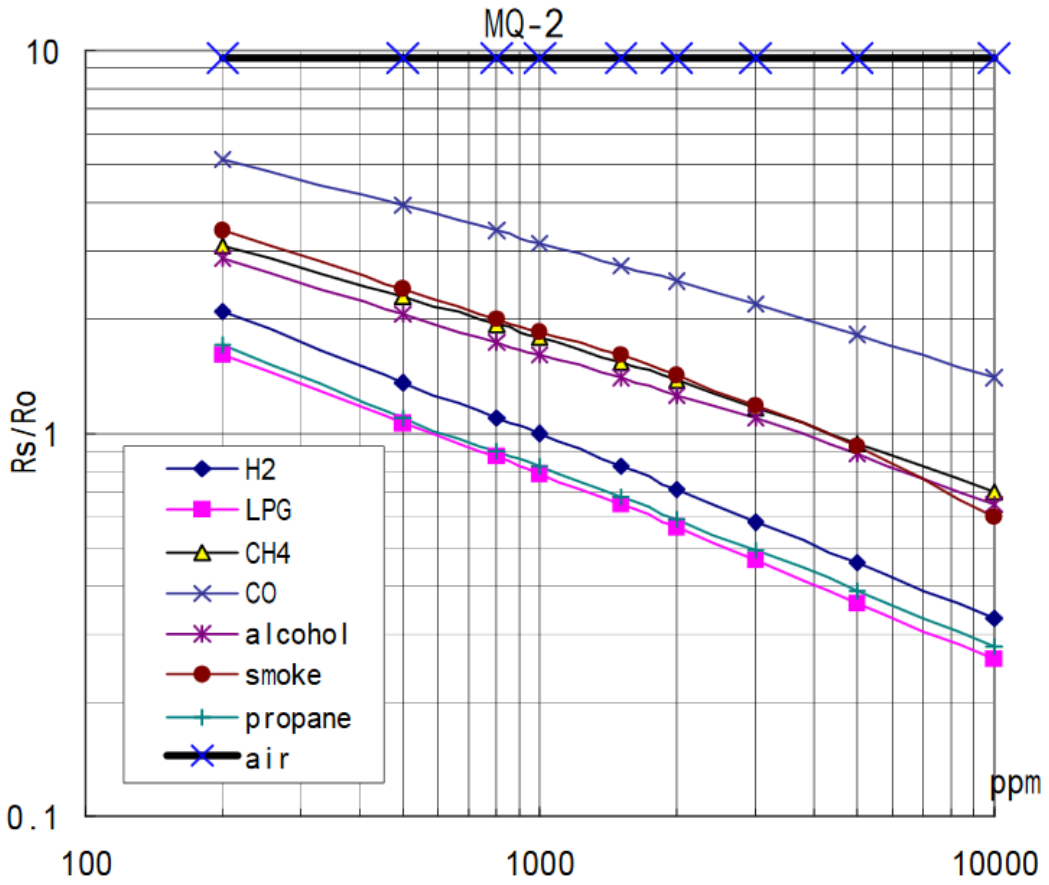


Рис. 9.4. Типові характеристики чутливості MQ-2 для кількох газів. Температура: 20 ° C .、Вологість: 65% Концентрація O2 21% RL = 5 kОм, Ro: опір датчика при 1000ppm H2 у чистому повітрі. Rs: опір датчика при різних концентраціях газів.

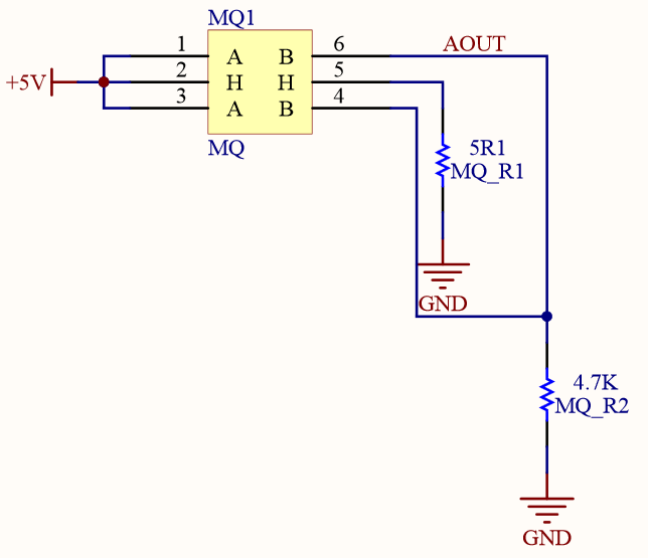


Рис. 9.5. Схема включення аналогово виходу в модулі MQ-2.

В даній схемі резистор MQ_R1 задає струм нагрівника, а резистор MQ_R2 утворює з опором чутливого елемента датчика (який під’єднаний до напруги живлення) дільник напруги, його номінал вибирається так щоб було зручно зчитувати дані вбудованим АЦП мікроконтролера.

Завдання

1. Реалізуйте програму апаратного генератора випадкових чисел на основі шумів та наводок на вільний не під'єднаний вхід АЦП.
2. Побудуйте графік зміни шумів та наводок вільного не під'єднаного входу АЦП.
3. Реалізувати схему підключення давача газу до модуля Arduino UNO згідно схеми рис. 9.2, та запрограмувати їх роботу в середовищі Arduino IDE. Виведіть графік зміни концентрації пропану з часом при піднесенні заральнички.
4. Реалізуйте простий сигналізатор при небезпечних концентраціях газів.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, давача газу MQ-2 і 3 провідники.
2. Виконати під'єднання давача газу MQ-2 до макетної плати згідно схеми підключення (рис. 9.6).

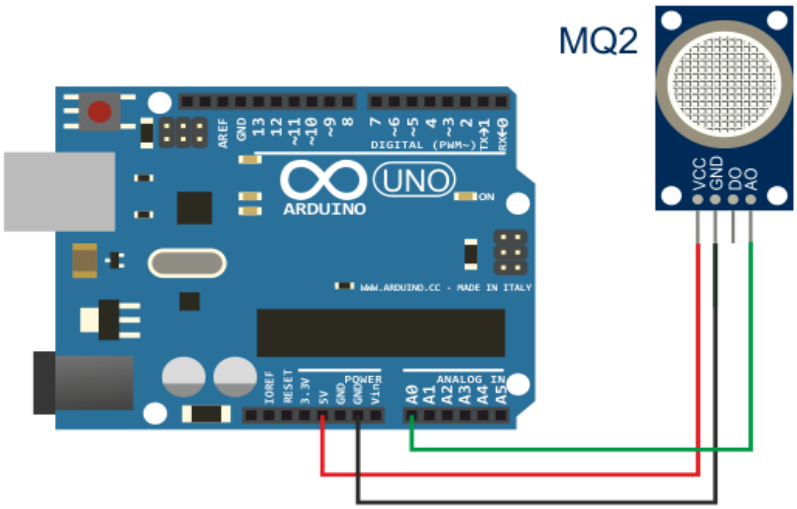


Рис. 9.6. Схема підключення давача газу MQ-2 до плати Arduino UNO.

3. Запустити середовище Arduino IDE. Створити новий файл. Вставити наступний програмний код у створений файл та зберегти його.

```
#define smokePin A0     // визначаємо аналоговий вихід до якого під'єднаний давач газу MQ-2
int sensorThres = 400; // порогове значення АЦП, при якому враховується наявність газу
void setup() {
  Serial.begin(9600);     // Встановлюємо швидкість порту 9600 бод
}
void loop() {
```

```
  int analogSensor = analogRead(smokePin); // зчитуємо значення АЦП з аналогового входу до якого під'єднано давач газу
  Serial.print (analogSensor); // виводим в порт значення АЦП сигналу з давача
  // Перевіряємо чи досягнуто порогове значення
  if (analogSensor > sensorThres) {   // якщо значення більше допустимого...
    Serial.println(" Gaz!");           // виводим в порт запис, що газ є
  }
  else {
   Serial.println(" normal");          // виводим в порт запис, що газу немає
  }
  delay(500);  // затримка в 500 мілісекунд.
}
```

4. Виконуємо перевірку (компіляцію) програмного коду натиснувши клавішу .
5. Наступним кроком під'єднуємо плату до комп'ютера і завантажуємо програму в мікроконтролер макетної плати Arduino UNO натиснувши кнопку .
6. Відкриваємо монітор послідовного порту і спостерігаємо результати роботи, коли газ не подається на давач MQ-2, встановивши швидкість у нижньому лівому куті монітора порта 9600 бод.
7. Доповнити програму для можливості виведення графіка на послідовному плотері. Спостерігаємо результати роботи, коли подати газ із запальнички на давач MQ-2.
8. Використовуючи вхід АЦП створіть генератор випадкових чисел, відобразіть їх на графіку.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Яке основне призначення давача газу MQ-2?
2. Назвіть основні характеристки давача газу MQ-2.
3. Обгрунтуйте схему підключення давача газу до Arduino UNO.
4. Опишіть основні функції програмного коду, який моделює роботу давача газу MQ-2.
5. В чомі основні проблеми використання і калібрування давача MQ-2

Лабораторна робота № 10.

Проектування підсистеми визначення рухомих об’єктів розумного будинку з використанням ультразвукового давача відстані HC-SR04.

Мета роботи: засвоїти основні принципи роботи ультразвукового давача відстані HC-SR04, навчитись під’єднувати його до макетної плати Arduino UNO та засвоїти основні принципи програмування його роботи в середовищі Arduino IDE.

Теоретичні відомості

Ультразвукові давачі відстані є альтернативою для інфрачервоних давачів руху. Ультразвукові давачі відстані зручно використовувати навпроти дверей невеликих приміщень і як тільки вони відкриватимуться, або в радіусі дії датчика буде знаходитися людина, давач буде спрацьовувати і вмикати освітлення.

Ультразвуковий давач відстані визначає відстань до об’єкта, вимірюючи час відбивання звукової хвилі від об’єкта. Частота звукової хвилі знаходиться в межах частоти ультразвуку, що забезпечує концентрований напрямок звукової хвилі, так як звук з високою частотою розсіюється в навколишньому середовищі менше. Типовий ультразвуковий давач відстані складається з двох мембран, одна з яких генерує звук, а інша реєструє відображене відлуння (рис. 10.1.). Образно кажучи, ми маємо справу зі звуковою колонкою і мікрофоном. Звуковий генератор створює маленький, з деяким періодом ультразвуковий імпульс і запускає таймер. Друга мембрана реєструє прибуття відображеного імпульсу і зупиняє таймер. Від часу таймера та швидкістю звуку можливо обчислити пройдену відстань звукової хвилі. Відстань об’єкта приблизно половина пройденого шляху звукової хвилі.

Рис. 10.1. Ультразвуковий давач відстані HC-SR04

Діапазон вимірювань становить: від 2 см до 400 см. На показання давача практично не впливають сонячне випромінювання і електромагнітні шуми.

Технічні характеристики HC-SR04:

- напруга живлення: +5 В – постійний струм;
- сила струму спокою: < 2 мА;
- робоча сила струму: 15 мА;
- ефективний робочий кут: < 15 °;
- відстань вимірювань: від 2 см до 400 см;
- роздільна здатність: 0,3 см;
- кут вимірювань: 30 градусів;
- ширина імпульсу тригера: 10 мікросекунд.

Підключення давача HC-SR04 до плати Arduino Uno наведена на рис. 10.2.

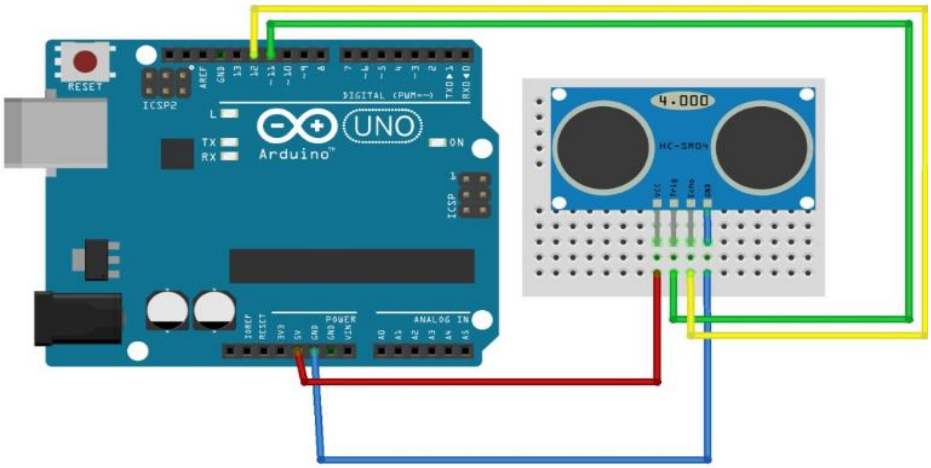


Рис. 10.2. Схема підключення давача до Arduino Uno.

Давач має 4 виводи:

1. VCC: «+» живлення;
2. TRIG (T): вивід вхідного сигналу;
3. ECHO (R): вивід вихідного сигналу (довжина сигналу рівна часу проходження хвилі і залежить від відстані об’єкта до давача);
4. GND: «-» живлення (земля).

Принцип роботи давача можна умовно розділити на 4 етапи:

1. Подаємо імпульс тривалістю 10 мкс, на вивід Trig.
2. Усередині дальноміра вхідний імпульс перетворюється у 8 імпульсів частотою 40 КГц і надсилається вперед;
3. Дійшовши до перешкоди, послані імпульси відбиваються і приймаються в результаті чого отримуємо вихідний сигнал на виводі Echo.

Функція **pulseIn ()**

Зчитує довжину сигналу на заданому порту (HIGH або LOW). Наприклад, якщо задано зчитування HIGH функцією pulseIn (), функція чекає поки на заданому порту не з'явилося HIGH. Коли HIGH отримано, включається таймер, який буде зупинений коли на порту входу/виходу буде LOW. Функція pulseIn () повертає довжину сигналу в мікросекундах. Функція повертає 0, якщо протягом заданого часу (таймаута) не зафіксований сигнал на порту.

Можливі деякі похибки в вимірювання довгих сигналів. Функція може вимірювати сигнали довжиною від 10 мікросекунд до 3 хвилин.

Синтаксис

pulseIn (pin, value)

pulseIn (pin, value, timeout)

Завдання

Реалізувати схему підключення ультразвукового давача відстані HC-SR04 до модуля Arduino UNO, та запрограмувати їх роботу.

Вивести графік зміни відстані з часом при наближенні і віддаленні предмета і давача.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, ультразвуковий давач відстані HC-SR04 і 4 провідники.
2. Виконати під’єднання ультразвукового давача відстані HC-SR04 до макетної плати згідно схеми підключення (рис. 10.2.).
3. Запустити середовище Arduino IDE. Створити новий файл. Вставити наступний програмний код у створений файл та зберегти його.

```
#define PIN_TRIG 12
#define PIN_ECHO 11
long duration, cm;
void setup() {
  // Ініціалізуємо взаємодію по послідовному порту
  Serial.begin (9600);
  //Определяем вводы и выводы
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
}
void loop() {
  // Генеруємо короткий імпульс довжиною 2-5 мікросекунд
  digitalWrite(PIN_TRIG, LOW);
```

```
  delayMicroseconds(5);
  digitalWrite(PIN_TRIG, HIGH);
  // Виставивши високий рівень, чекаємо приблизно 10 мікросекунд. В цей момент давач буде посилати сигнали з частотою 10 КГц.
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);
  //  Час затримки акустичного сигналу на ехолокаторі
  duration = pulseIn(PIN_ECHO, HIGH);
  // Перетворення часу у відстань
  cm = (duration / 2) / 29.1;
  Serial.print("Vidstan do obecta: ");
  Serial.print(cm);
  Serial.println(" sm.");
  // Затримка між вимірюваннями для коректної роботи програми
  delay(250);
}
```

4. Виконуємо перевірку (компіляцію) програмного коду натиснувши клавішу .
5. Наступним кроком під’єднуємо плату до комп’ютера. і завантажуємо програму в мікроконтролер макетної плати Arduino UNO натиснувши кнопку .
6. Спостерігаємо результати роботи ультразвукового давача відстані HC-SR04 при наближенні рухомого об’єкта до нього.
7. Виводимо результати в послідовний плотер.
8. Наступним кроком потрібно зайти у меню Скетч ->Додати Бібліотеку ->Додати ZIP бібліотеку та вибрати zip-файл бібліотеки для ультразвукового давача відстані HC-SR04.
9. Вставити наступний програмний код у створений файл та зберегти його і виконати усі вище перелічені дії.

```
#include <NewPing.h>
#define PIN_TRIG 12
#define PIN_ECHO 11
#define MAX_DISTANCE 200 //Константа для визначення максимальної відстані, яку можна рахувати коректною
NewPing sonar(PIN_TRIG, PIN_ECHO, MAX_DISTANCE);
void setup() {
```

```
 Serial.begin(9600);
}
void loop() {
  // Стартова затримка, необхідна для коректної роботи
  delay(300);
  // Отримуємо значення від давача і зберігаємо його в змінну.
  unsigned int distance = sonar.ping_cm();
  // Вивід відстані в монітор порта
  Serial.print(distance);
  Serial.println("sm");
}
```

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Яке основне призначення ультразвукового давача відстані HC-SR04?
2. Назвіть основні характеристки ультразвукового давача відстані HC-SR04.
3. Обгрунтуйте схему підключення ультразвукового давача відстані до Arduino UNO.
4. Опишіть основні функції програмного коду, який моделює роботу ультразвукового давача відстані HC-SR04

Лабораторна робота № 11.

Вимірювання електричного опору. Проектування підсистеми визначення вологи чи протікання води з використанням давача води FC-37.

Мета роботи: Навчитися опрацьовувати сигнали аналогового давача який базується на зміні електричного опору.

Теоретичні відомості

Датчик протікання води і дощу дозволяє визначити появу крапель вологи і вчасно відреагувати на це, наприклад, включивши оповіщення. Такі системи активно використовуються в аграрній галузі, в автомобілебудуванні та в інших повсякденних сферах нашого життя.

Модуль датчика складається з двох частин:

«Сенсорна» плата виявлення крапель. Вона відстежує кількість потрапляння на неї вологи. По суті, сенсор являє собою простий змінний резистор, замикає водою в різних місцях, що викликає зміну опору.

Друга частина датчика - здвоєний компаратор (як правило, LM393, але можливі варіанти LM293 і LM193). Його головне завдання - перетворення значення з сенсора в аналоговий сигнал від 0 до 5 Вольт.

Датчик живиться від напруги 5 В, який можна легко завести з будь-якої плати Arduino. Як правило, у модуля датчика є два виходи:

Аналоговий. Значення, що отримується контролером, буде варіюватися від 0 до 1023. Де 0 - все затопило або йде злива, сенсор дуже вологий, 1023 - суха погода, сенсор сухий (в деяких датчиках зустрічаються протилежні значення, 1023 - максимальна вологість, 0 - максимальна сухість) .

Цифровий. Видає високу (5В) або низьку напругу в разі перевищення деякого порогу. Рівень порогу спрацьовування регулюється за допомогою настройок потенціометром.

Підключення давача FC-37 до плати Arduino Uno наведено на рис. 11.1.
Схема підключення модуля датчика протікання до Arduino:

- VCC (вхід живлення) - повинен збігатися для з'єднаної схеми Arduino по напрузі і струму. Тобто в даному випадку 5В;
- GND – спільний провід;
- AO - аналоговий вихід (A1);
- DO - цифровий вихід (2).

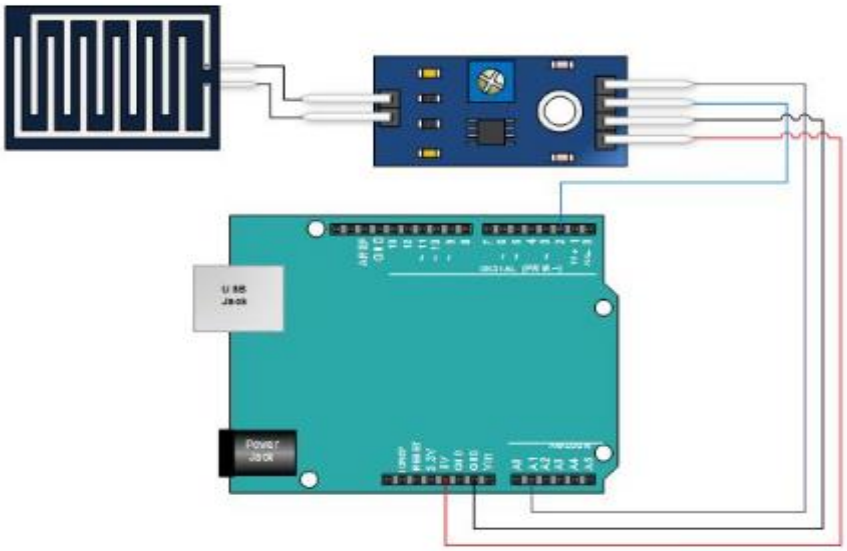


Рис. 11.1. Схема підключення давача води FC-37 до Arduino Uno.

Аналоговий вихід приєднуємо до аналогового піну мікроконтролера, наприклад, A1. Цифровий вихід, відповідно підключається до одного з цифрових пінів. Напругу можна подати з виводу макетної плати Arduino 5B, земля з'єднується з виводом GND.

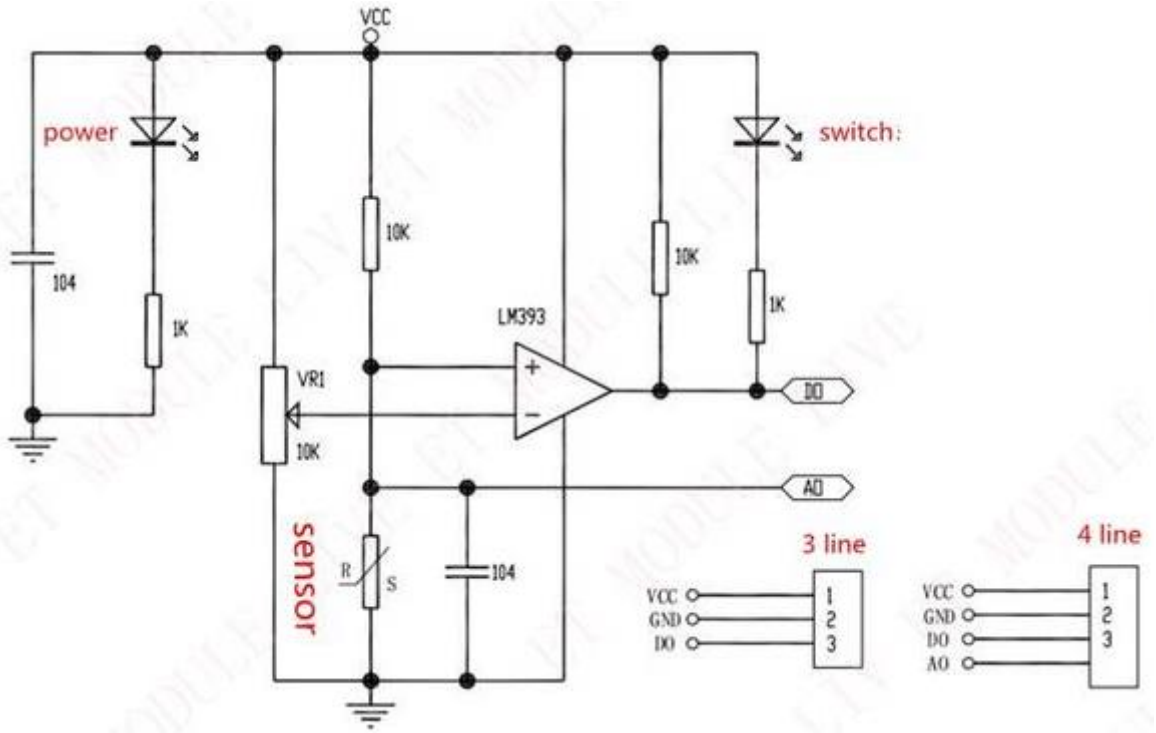


Рис. 11.2. Схема давача вологи.

Завдання

Реалізувати схему підключення давача води FC-37 до модуля Arduino UNO, та запрограмувати їх роботу.

Реалізувати сигналізатор протікання води та монітор рівня вологи.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, давач води FC-37 і 6 провідників.
2. Виконати під'єднання давача води FC-37 до макетної плати згідно схеми підключення (рис. 11.1).
3. Запустити середовище Arduino IDE. Створити новий файл і вставити наступний програмний код у створений файл та зберегти його.

```
#define PIN_ANALOG_RAIN_SENSOR A1 // Аналоговий вхід для сигналу  давача протікання води
#define PIN_DIGITAL_RAIN_SENSOR 2 // Цифровий вхід для сигналу давача протікання води

void setup(){
Serial.begin(9600);
}
void loop(){
int sensorValue = analogRead(PIN_ANALOG_RAIN_SENSOR); // Зчитуємо значення з аналогового порта
Serial.print("Analog value: ");
Serial.println(sensorValue);   //   Виводимо   аналогові значення в монітор порта

sensorValue  =  digitalRead(PIN_DIGITAL_RAIN_SENSOR);  // Зчитуємо дані з цифрового порта
 Serial.print("Digital value: ");
Serial.println(sensorValue); // Виводимо цифрові значення в монітор порта
delay(1000); // Затримка між вимірюваннями
}
```

4. Виконуємо перевірку (компіляцію) програмного коду натиснувши клавішу .
5. Наступним кроком під'єднуємо плату до комп'ютера і завантажуємо програму в мікроконтролер макетної плати Arduino UNO натиснувши кнопку .
6. Відкриваємо монітор послідовного порту зайшовши у меню Інструменти -> Монітор послідовного порту і спостерігаємо результати роботи давача води FC-37 подавши на нього вологу (протерти вологою салфеткою).
7. Розробити прогаму сигналізатора протікання води та монітор рівня вологи.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Яке основне призначення давача води FC-37?
2. Назвіть основні характеристки резистивного давача води FC-37.
3. Обгрунтуйте схему підключення давача води до Arduino UNO.
4. Опишіть основні функції програмного коду сигналізатора протікання води.

Лабораторна робота № 12.

Безпровідна передача даних. Проектування підсистеми ідентифікації об’єктів доступу до приміщення з використанням модуля радіочастотної ідентифікації RC522.

Мета роботи: Вивчення радіочастотного модуля безпровідної передачі даних та набуття практичних навичок програмування безпровідних інтерфейсів. Ознайомлення з інтерфейсом SPI.

Теоретичні відомості

Радіочастотна ідентифікація (RFID) – це технологія безконтактної ідентифікації об’єктів за допомогою радіочастотного каналу зв’язку.

Для входу до приміщення використовують RFID-модуль, який надає доступ до будинку за допомогою технології RFID-міток у вигляді безконтактної картки або брилка, а також використовується як ключ для ввімкнення/вимкнення сигналізації.

Ідентифікація об’єктів проводиться за унікальним ідентифікатором, який має кожна електронна мітка. Зчитувач випромінює електромагнітні хвилі певної частоти. Мітки відправляють у відповідь інформацію – ідентифікаційний номер, дані пам’яті та інші дані.

Рис. 12.1. RFID-модуль RC522.

Переваги технології RFID:

- безконтактна;
- можливість прихованого встановлення міток;
- висока швидкість зчитування даних;
- можливість установки в шкідливих середовищах;

Існує велика різноманітність RFID-міток. Мітки бувають активні і пасивні (без вбудованого джерела енергії, живляться від струму, індукованого в

антені сигналом від рідера). Мітки працюють на різній частоті: LF (125-134 кГц), HF (13,56 МГц), UHF (860-960 МГц).

Прилади, які читають інформацію з міток і записують в них дані, називаються рідерами (зчитувачами). У проектах Arduino як зчитувач дуже часто використовують модуль RFID-RC522 (рис. 12.1). Модуль виконаний на мікросхемі MFRC522 фірми NXP, яка забезпечує роботу з мітками HF (на частоті 13,56 МГц). У комплекті з модулем RFID-RC522 йдуть дві мітки, одна у вигляді карти, інша у вигляді брилка.

Технічні характеристики RFID-модуля RC522:

- напруга живлення: 3,3V;
- струм споживання: 13-26 mA;
- робоча частота: 13,56MHz;
- дальність зчитування: 0-60 мм;
- інтерфейс: SPI, UART та I2C;
- швидкість передачі: максимальна 10 Мбіт/с;
- розмір: 40 мм х 60 мм.

Інтерфейси і призначення виводів модуля RC522

Мікросхема MFRC522 підтримує інтерфейси SPI, UART та I2C (рис. 12.2). Вибір інтерфейсу здійснюється встановленням логічних рівнів на певних входах мікросхеми. На даному модулі обраний інтерфейс SPI.

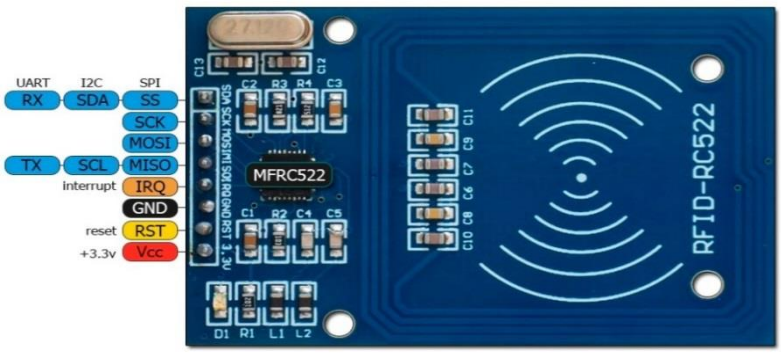


Рис. 12.2. Призначення виводів RFID-модуля.

Призначення виводів інтерфейсу SPI:

SDA – шина передачі даних;

SCK – сигнал синхронізації;

MOSI – передача від master до slave;

MISO – передача від slave до master;

RST – вивід для скидання;

IRQ – вивід переривання;

GND – земля;

VCC – живлення 3.3 В.

Сигнал скидання RST – це сигнал, що надходить від цифрового виходу контролера. При надходженні сигналу LOW відбувається перезавантаження зчитувача. Також рідер встановленням на RST низького рівня повідомляє, що знаходиться в режимі сну, для виведення модуля з режиму сну необхідно подати на даний вивід сигнал HIGH.

У ролі міток використовуються брилки і карти, що йдуть в комплекті з модулем (рис. 12.3).

Рис. 3. RFID-мітки для модуля RFID RC522.

Завдання

Реалізувати схему підключення модуля RFID-RC522 до плати Arduino UNO згідно схем на рис. 12.5 та запрограмувати їх роботу в середовищі Arduino IDE.

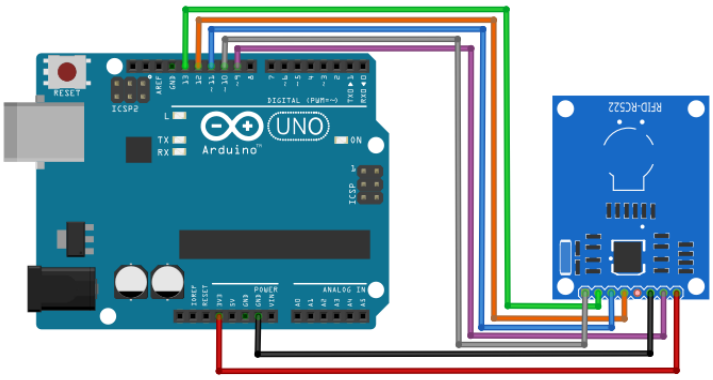


Рис. 12.4. Схема підключення модуля RFID-RC522.

На рис. 12.5 подано принципову схему підключення модуля до плати Arduino UNO.

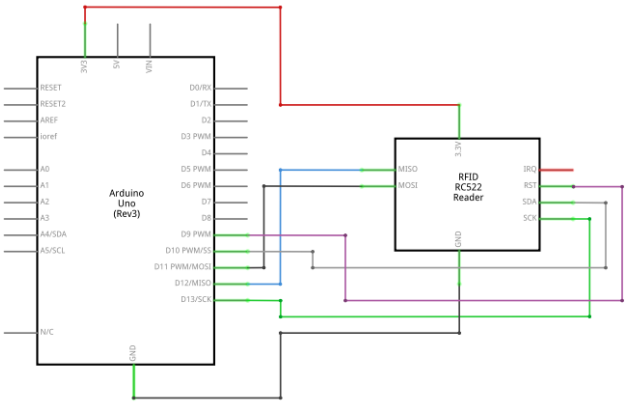
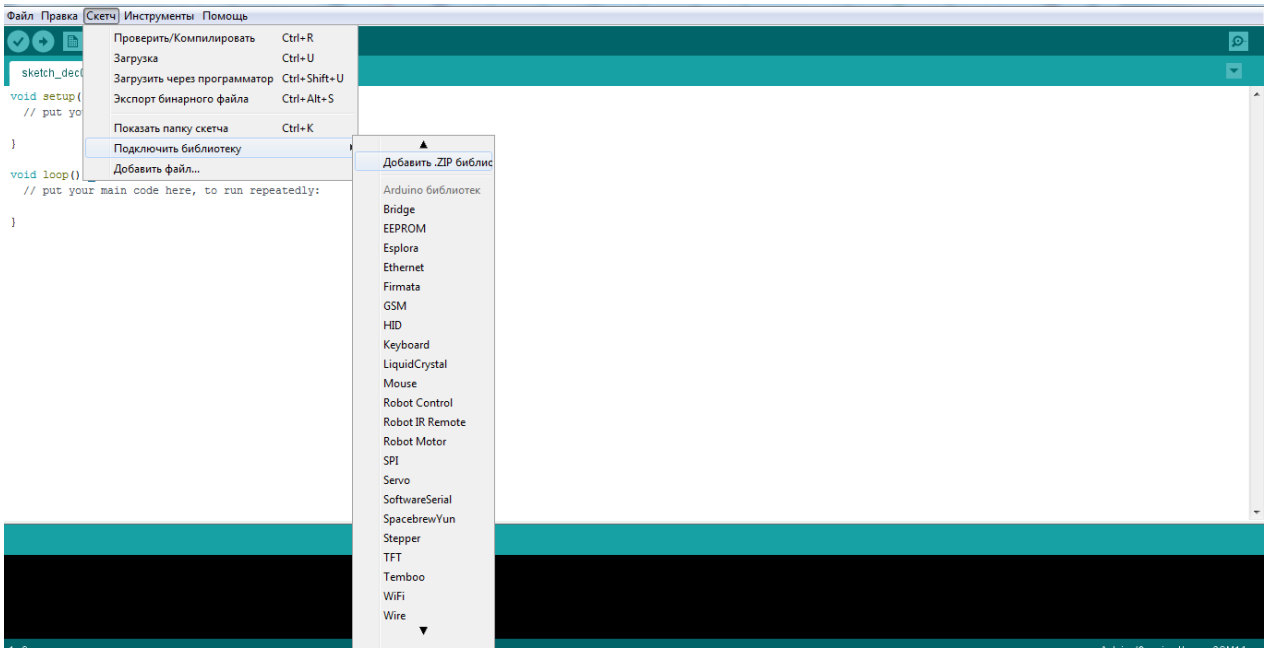


Рис. 12.5. Принципова схема підключення модуля до плати Arduino UNO.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, модуль радіочастотної ідентифікації RC522 і 7 провідників.
2. Виконати під'єднання модуля радіочастотної ідентифікації RC522 до макетної плати Arduino UNO згідно схеми підключення (рис. 12.5).
3. Запустити середовище Arduino IDE. Створити новий файл.
4. Далі потрібно зайти у меню Скетч ->Додати Бібліотеку ->Додати ZIP бібліотеку та вибрати zip-файл бібліотеки для модуля радіочастотної ідентифікації RC522.



5. Наступним кроком вставляємо та запускаємо наступний код для зчитування шістнадцяткового номера RFID мітки карти та брилка.

```
#include <SPI.h>
#include <MFRC522.h>
```

```
// константи під’єднання контактів SS и RST
#define RST_PIN 9
#define SS_PIN 10
// Ініціалізація MFRC522
MFRC522 mfrc522(SS_PIN, RST_PIN); //Створення екземпляру
MFRC522.
void setup()
{
Serial.begin(9600); // Ініціалізація послідовного порту
SPI.begin(); // Ініціалізація SPI
mfrc522.PCD_Init(); // Ініціалізація MFRC522
}
void loop()
{
if ( ! mfrc522.PICC_IsNewCardPresent())
return;
// читання карти
if ( ! mfrc522.PICC_ReadCardSerial())
return;
// показати результати читання UID і тип мітки
Serial.print(F("Card UID HEX:"));
dump_byte_array(mfrc522.uid.uidByte, mfrc522.uid.size);
Serial.println();
Serial.print(F("PICC type: "));
byte piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
Serial.println(mfrc522.PICC_GetTypeName(piccType));
delay(2000);
}
// Вивід результату читання даних в HEX-виді
void dump_byte_array(byte *buffer, byte bufferSize)
{
for (byte i = 0; i < bufferSize; i++)
{
Serial.print(buffer[i] < 0x10 ? " 0" : " ");
Serial.print(buffer[i], HEX);
}
}
```

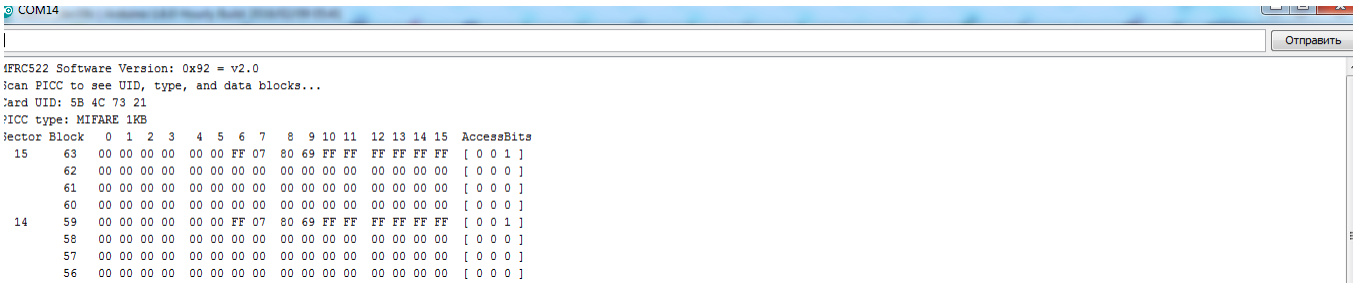
6. Виконуємо перевірку (компіляцію) програмного коду натиснувши клавішу .

7. Наступним кроком під’єднуємо плату до комп’ютера і завантажуємо програму в мікроконтролер макетної плати Arduino UNO натиснувши кнопку .
8. Відкриваємо монітор послідовного порту зайшовши у меню Інструменти -> Монітор послідовного порту.
9. Спостерігаємо результати зчитування шістнадцяткового номера RFID мітки карти чи брелка.
10. Наступним кроком створимо новий файл та завантажемо прогамний код, який дає можливість зчитати, крім номеру карти чи брилка, її тип та доступні для зчитування дані. Слід зазначити, що карти стандарту (типу) MIFARE 1K складаються із 16 секторів, кожний із яких містить 4 блоки, а кожний блок містить 16 байт даних.

```
#include <MFRC522.h>
#include <SPI.h>
#define RST_PIN  9  //
#define SS_PIN  10  //
MFRC522 mfrc522(SS_PIN, RST_PIN); // Створюємо екземпляр
MFRC522
void setup() {
 Serial.begin(9600);        //    Ініціалізуємо    монітор
послідовного порту
 SPI.begin();   // Ініціалізуємо SPI шину
 mfrc522.PCD_Init();  // Ініціалізуємо RFID модуль
 ShowReaderDetails(); // Виводимо дані про модуль MFRC522
 Serial.println(F("Scan PICC to see UID, type, and data
blocks..."));
}
void loop() {
 // Шукаємо нову карту
 if ( ! mfrc522.PICC_IsNewCardPresent()) {
  return;
 }
 // Вибираємо одну із карт
 if ( ! mfrc522.PICC_ReadCardSerial()) {
  return;
 }
 // Виводим дані з карти
 mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
}
```

```
void ShowReaderDetails() {
 // Отримуємо номер версії модуля
 byte v = mfrc522.PCD_ReadRegister(mfrc522.VersionReg);
 Serial.print(F("MFRC522 Software Version: 0x"));
 Serial.print(v, HEX);
 if (v == 0x91)
  Serial.print(F(" = v1.0"));
 else if (v == 0x92)
  Serial.print(F(" = v2.0"));
 else
  Serial.print(F(" (unknown)"));
 Serial.println("");
 // Коли отримуємо 0x00 або 0xFF, передача даних порушена
 if ((v == 0x00) || (v == 0xFF)) {
  Serial.println(F("WARNING: Communication failure, is the MFRC522 properly connected?"));
 }
}
```

11. Наступим кроком компілюємо скетч. Якщо компіляція пройшла без помилок, завантажуємо програму у плату.
12. Відкриваємо монітор порта та підносимо карту а потім брелок до зчитувача.
13. Спостерігаємо як зчитуються ідентифікатор, тип і дані з карти.



Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.

7. Висновки.

Контрольні запитання

1. Яке призначення RFID-модуля MFRC522 та його основні характеристики.
2. Яка суть технології на основі RFID-міток.
3. Назвіть інтерфейси, які підтримує мікросхема MFRC522.
4. Обгрунтуйте схему підключення модуля радіочастотної ідентифікації RC522 до Arduino UNO.
5. Опишіть основні функції програмного коду, який моделює роботу модуля радіочастотної ідентифікації RC522.

Лабораторна робота № 13-14.

Безпровідна передача даних через канали зв'язку з використанням GSM модуля SIM800.

Мета роботи: Вивчення безпровідної передачі даних через GSM канали мобільного зв'язку та набуття практичних навичок програмування GSM модулів передачі даних.

Теоретичні відомості

Для зв'язку мікроконтролерних пристроїв через мережі мобільних операторів зручно використовувати GSM модулі, наприклад SIM800

Особливості SIM800:

• Чотирьохдіапазонний GSM / GPRS модуль, 850/900/1800/1900 Мгц
• В залежності від версії модему, інтерфейс UART, FM-radio, Bluetooth.
• Управління AT-командами.
• Вбудований стек TCP / IP, UDP / IP, HTTP, FTP, Email, PING, MMC.
• Позиціонування по базовим станціям.
• Декодування і формування DTMF-тонів.
• Відтворення аудіофайлів локально і в сторону віддаленого абонента.

Рис. 13.1. Загальний вигляд модуля.

Але насправді - модуль SIM800L ховається під металевою кришкою з наклейкою:

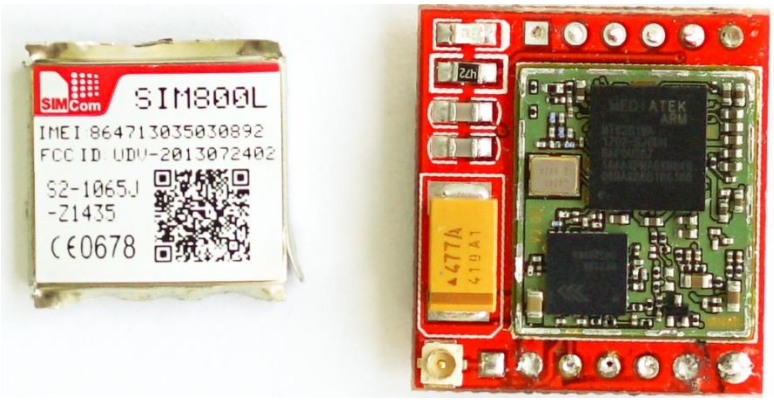


Рис. 13.2. Вигляд модуля в середині плати.

Модуль SIM800L побудований на базі чіпа Mediatek ARM MT6261. За GSM / GPRS-зв'язок відповідає 4-діапазонний (GSM850 / GSM900 / DCS1800 / PCS1900) приймач RF7198.

У даного модуля виведений необхідний мінімум, для доступу до переважної більшості функцій, включаючи голосові (виходи для підключення мікрофону і динаміка):

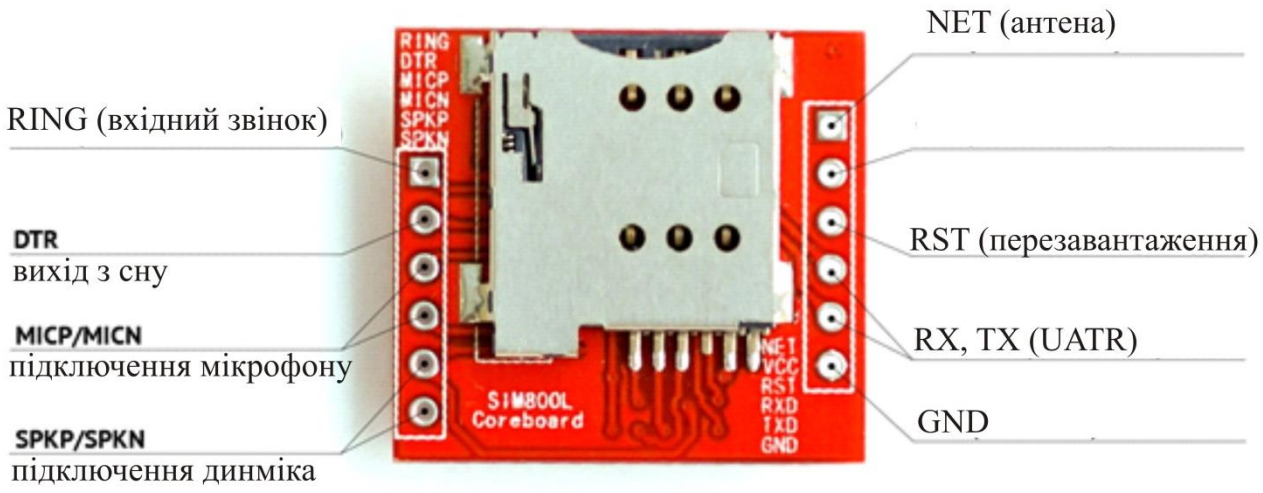


Рис. 13.3. Виводи модуля.

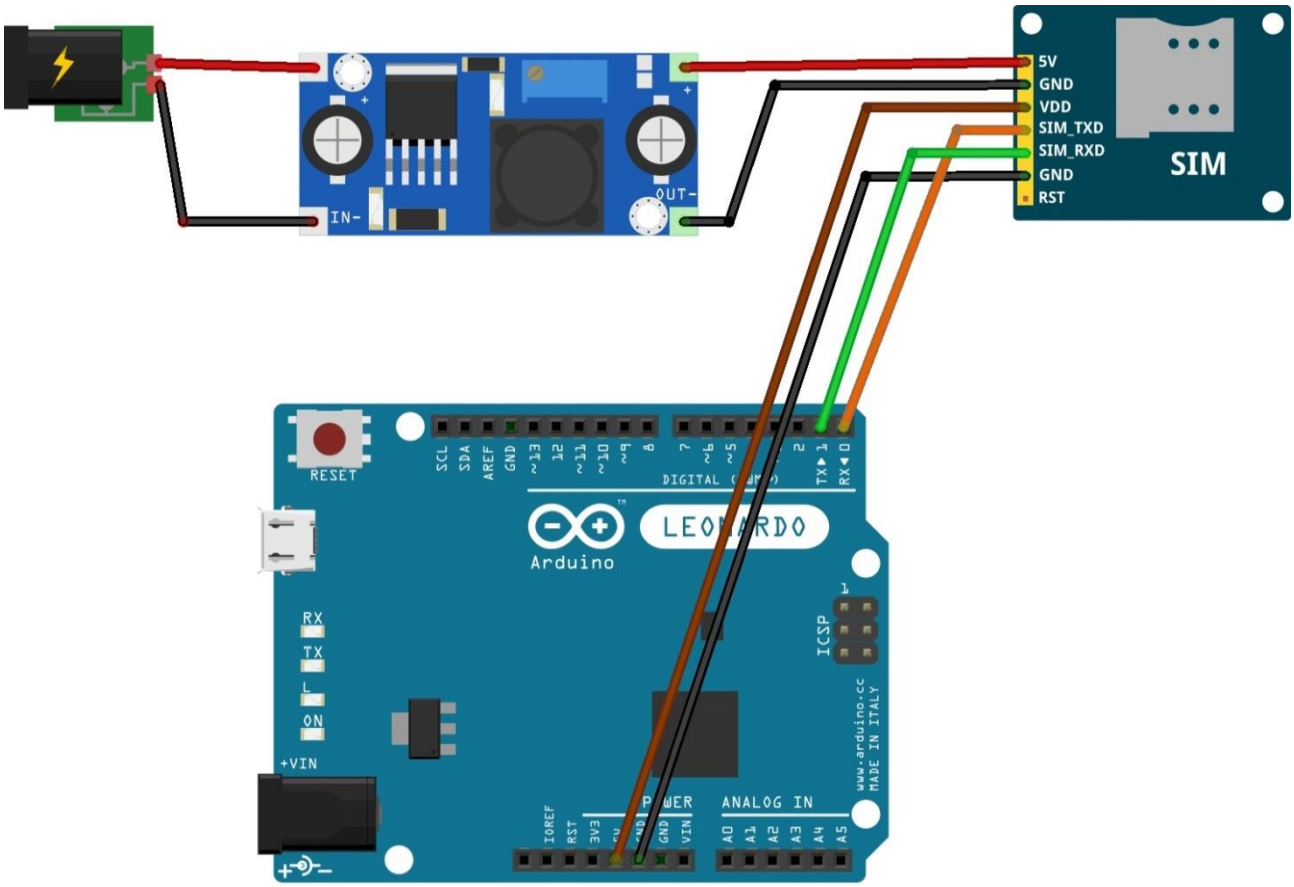


Рис. 13.4. Під’єднання модуля SIM800L до плати Arduino.

Синтаксис AT-команд

Взаємодія з модулем відбувається за допомогою AT-команд. Кожна AT-команда повинна починатися з двох букв AT, набраних в будь-якому регістрі: AT, aT, At, at. Команда повинна починатися з нового рядка (параметр Newline у вікні Serial).

Всі AT-команди синтаксично діляться на 3 основні групи: базові, з параметром S і розширені.

Базові команди мають наступний синтаксис: AT <x> <n> або AT & <x> <n>, де <x> є командою, а <n> - переданим параметром (параметрами). Параметр <n> є необов'язковим і в разі його відсутності буде використано значення за замовчуванням. Прикладами можуть служити, команда ATI, яка повертає ідентифікаційні дані модуля, або команда AT&V повертає поточну конфігурацію модуля.

Команди з параметром S виглядають наступним чином: ATS <i> = <m>, де <i> - індекс S-регістра, а <m> - значення, яке йому необхідно привласнити. У разі відсутності значення <m>, буде присвоєно значення за замовчуванням. Наприклад, регістр ATS0 відповідає за кількість гудків перед автоматичною відповіддю на вхідний дзвінок, і, відповідно, команда ATS0 встановить значення за замовчуванням - 0 (не відповідати на виклик), а команда ATS0 = 2 змусить відповідати модуль автоматично після 2 гудка.

Розширені команди можуть викликатися в декількох режимах:

AT-команди (від англ. ATtention - «увага») - набір команд, що складається з серій коротких текстових рядків, які об'єднуються разом, щоб сформувати повні команди операцій, таких як набір номера, початок з'єднання або змінити налаштування.

Детально з повним набором AT-команд можна ознайомитися в документі - SIM800 Series_AT Command Manual_V1.09.pdf (3,07 MB)

Прикладом використання розширених команд можуть бути команди для управління FM-радіо - AT + FMOPEN, AT + FMCLOSE і т.д.

Як приклад для даної групи команд, також можна привести команду AT + IPR. Командою AT + IPR =? можна дізнатися значення, які може приймати параметр. Командою AT + IPR? можна дізнатися поточну швидкість обміну даними з модулем, а командою AT + IPR = <...> - задати іншу швидкість.

В одному рядку можна вказувати декілька команд. Всі наступні команди (виключаючи першу) записуються без префікса AT. Після кожної розширеної команди необхідно ставити крапку з комою;. Після базових команд або команд з параметром S, крапка з комою; не потрібна. приклад:

ATV0E1+DDET=1,0,1; Q0S0=1+CLIP=1; +CMGF=1; &W

Довжина всього рядка з декількома командами не повинна перевищувати 556 символів (не рахуючи початкового префікса AT)

За дублювання в терміналі відправленої команди відповідає параметр Echo Mode. Для зміни значення цього параметра існує команда ATE<value>, де <value> - значення параметра, 0 - вимкнено, 1 - ввімкнений (за замовчуванням). Наприклад, команда ATE0 відключить цей режим, а команда ATE1 включить.

У модулі SIM800L існує параметр, який встановлює ступінь інформативності випадаючих помилок.

AT+CMEE?	**+CMEE: <*n*>** **OK** <N> - інформування про помилки 0 - повідомлення про власне наявності помилки ERROR 1 - відповідь містить цифровий код помилки (розшифровка кодів в розділі 19 Довідника по AT-командам) 2 - відповідь містить опис помилки	+CMEE: 0 OK
Команда записи нового значения	AT+CMEE=<*n*>	**OK**

Незапитувані повідомлення

Є ще одна категорія повідомлень одержуваних від модуля - незапитувані повідомлення. Це повідомлення, які, як видно з назви, можуть приходити без здійснення користувачем будь-яких дій.

Найпоширенішими прикладами незапитуваних повідомлень, є:

Повідомлення	Опис	Приклад
RING	Повідомлення вхідного дзвінка	RING
+CMTI	Повідомлення приходу нового SMS-повідомлення	+CMTI: "SM",2
+CLIP	Автовизначник номера під час вхідного дзвінка	+CLIP: "+380501234567",145,"",0,"",0
+CUSD	Отримання відповіді на відправлений USSD-запит	+CUSD: 0, " Vash balans 198.02 r.: *111#", 15
UNDER-VOLTAGE POWER DOWN	Повідомлення про некоректну напругу модуля	UNDER-VOLTAGE WARNNING

UNDER-VOLTAGE WARNNING OVER-VOLTAGE POWER DOWN OVER-VOLTAGE WARNNING		
+CMTE	Повідомлення про некоректну температуру модуля	+CMTE: 1

У готових проектах, з метою економії пам'яті МК рекомендується використовувати комбінацію параметрів: відключений Echo Mode (ATE0), цифровий формат відповідей модуля (ATV0) і цифровий код помилок (AT + CMEE = 1). В освітніх цілях і при тестуванні ПЗ: включений (за замовчуванням) Echo Mode (ATE1), текстовий (за замовчуванням) формат відповідей модуля (ATV1) і текстовий код помилок (AT + CMEE = 2).

На виконання кожної команди модулю потрібен час. Відправивши модулю послідовно кілька команд, він, прекрасно впорається і відзвітує про успішне виконання.

```
#include <SoftwareSerial.h>
SoftwareSerial SIM800(8, 9);     // 8-RX Arduino (TX SIM800L), 9-TX Arduino (RX SIM800L)

void setup() {
  Serial.begin(9600);   //Швидкість обміну даними з комп'ютером
  Serial.println("Start!");
  SIM800.begin(9600);    // Швидкість обміну даними з модемом
  SIM800.println("AT");  //Автонастройка швидкості
  SIM800.println("AT+CLVL?");     //Запитуємо гучність динаміка
  SIM800.println("AT+CMGF=1");  //Включити TextMode для SMS
  SIM800.println("AT+DDET=1,0,1");  //Включити DTMF
  SIM800.println("AT+CLIP=1");       //Включити АОН
}

void loop() {
```

```
  if (SIM800.available()) // Очікуємо приходу даних (відповіді) від модему
    Serial.write(SIM800.read()); //... і виводимо їх в Serial
  if (Serial.available())//Очікуємо команди по Serial...
    SIM800.write(Serial.read());//... і відправляємо отриману команду модему
}
```

Але при уважному розгляді видно, що порядок відповідей не збігається з порядком відправлених команд, а це значить, що неможливо однозначно ідентифікувати відповідність приходу відповіді відправленої команди.

Більш коректним є підхід, при якому МК зупинить виконання програми до тих пір, поки не буде отримана відповідь на відправлену команду. Для цього, можливо використовувати наступні 2 функції - sendATComand () і waitResponse ():

```
String sendATCommand(String cmd, bool waiting) {
  String _resp = ""; //Змінна для зберігання результату
  Serial.println(cmd); //Дублюємо команду в монітор порту
  SIM800.println(cmd); //Відправляємо команду модулю
  if (waiting) { //Якщо необхідно дочекатися відповіді...
    _resp = waitResponse();//...чекаємо, коли буде передана відповідь
    //Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки можна закоментувати
    if (_resp.startsWith(cmd)) { // Якщо Echo включено то видаляємо з відповіді повтор команди
      _resp = _resp.substring(_resp.indexOf("\r", cmd.length()) + 2);
    }
    Serial.println(_resp); // Дублюємо відповідь в монітор порту
  }
  return _resp; //Повертаємо результат. Пусто, якщо проблема
}

String waitResponse() { //Функція очікування відповіді і повернення отриманого результату
  String _resp = ""; //Змінна для зберігання результату
```

```
  long _timeout = millis() + 10000; // Змінна для відстеження таймаута (10 секунд)
  while (!SIM800.available() && millis() < _timeout) {}; //Чекаємо відповіді 10 секунд, якщо прийшла відповідь або настав таймаут, то...
  if (SIM800.available()) { //Якщо є, що зчитувати...
    _resp = SIM800.readString(); //... зчитуємо і запам'ятовуємо
  }
  else { //Якщо прийшов таймаут, то...
    Serial.println("Timeout..."); //...сповіщаємо про це і...
  }
  return _resp; //... повертаємо результат. Пусто, якщо проблема
}
```

В функції sendATCommand(cmd, waiting) перший параметр містить відправлену команду, а другий параметр waiting задає, чи потрібно програмі дочекатися відповіді від GSM-модуля або ні.

Якщо режим Echo Mode увімкнений, то рядки 8-10 в попередньому програмному коді (функція sendATCommand()) можно закоментувати.

При такому підході, появиться можливість, по-перше, дочекатися відповіді від модуля, коли це необхідно, а по-друге, появиться можливість проаналізувати отриману відповідь:

```
#include <SoftwareSerial.h> // Бібліотека програмної реалізації обміну по UART-протоколу
SoftwareSerial SIM800(8, 9); //RX, TX
String _response = ""; //Змінна для зберігання відповіді модуля
void setup() {
  Serial.begin(9600); //Швидкість обміну даними з комп'ютером
  SIM800.begin(9600); //Швидкість обміну даними з модемом
  Serial.println("Start!");
  sendATCommand("AT", true); //Автонастройка швидкості
  sendATCommand("AT+CLVL?", true); //Запитуємо гучність динаміка
  sendATCommand("AT+CMGF=1", true); //Включити TextMode для SMS
```

```
  sendATCommand("AT+DDET=1,0,3", true); //Включити DTMF - в цьому рядку навмисне допущена помилка - неприпустимий параметр
  do {
    _response = sendATCommand("AT+CLIP=1", true); //Включємо АОН
    _response.trim(); //Забираємо пробільні символи на початку і кінці
  } while (_response != "OK"); //Чи не пускати далі, поки модем не поверне ОК
  Serial.println("CLI enabled"); //Інформуємо, що АОН включений
  // Перевірка пройдена, модем повідомив про Готовність, можна запускати основний цикл...
}
String sendATCommand(String cmd, bool waiting) {
  String _resp = ""; //Змінна для зберігання результату
  Serial.println(cmd); //Дублюємо команду в монітор порту
  SIM800.println(cmd); //Відправляємо команду модулю
  if (waiting) { //Якщо необхідно дочекатися відповіді...
    _resp = waitResponse();//...чекаємо, коли буде передана відповідь
    //Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки можна закоментувати
    if (_resp.startsWith(cmd)) { //Прибираємо з відповіді дублює команду
      _resp = _resp.substring(_resp.indexOf("\r", cmd.length()) + 2);
    }
    Serial.println(_resp); //Дублюємо відповідь в монітор порту
  }
  return _resp; // Повертаємо результат. Пусто, якщо проблема
}
String waitResponse() { //Функція очікування відповіді і повернення отриманого результату
  String _resp = ""; //Змінна для зберігання результату
  long _timeout = millis() + 10000; //Змінна для відстеження таймаута (10 секунд)
```

```
  while (!SIM800.available() && millis() < _timeout)  {};
//Чекаємо відповіді 10 секунд, якщо прийшла відповідь або
настав таймаут, то...
  if (SIM800.available()) { //Якщо є, що зчитувати...
    _resp = SIM800.readString(); //... зчитуємо і
запам'ятовуємо
  }
  else { //Якщо прийшов таймаут, то...
    Serial.println("Timeout..."); //...сповіщаємо про це
і...
  }
  return _resp; //... Повертаємо результат. Пусто, если
проблема
}

void loop() {
  if (SIM800.available()) // Очікуємо приходу даних
(відповіді) від модему...
    Serial.write(SIM800.read()); //... і виводимо їх в
Serial
  if (Serial.available()) //Очікуємо команди по Serial...
    SIM800.write(Serial.read());  //...і відправляємо
отриману команду модему
}
```

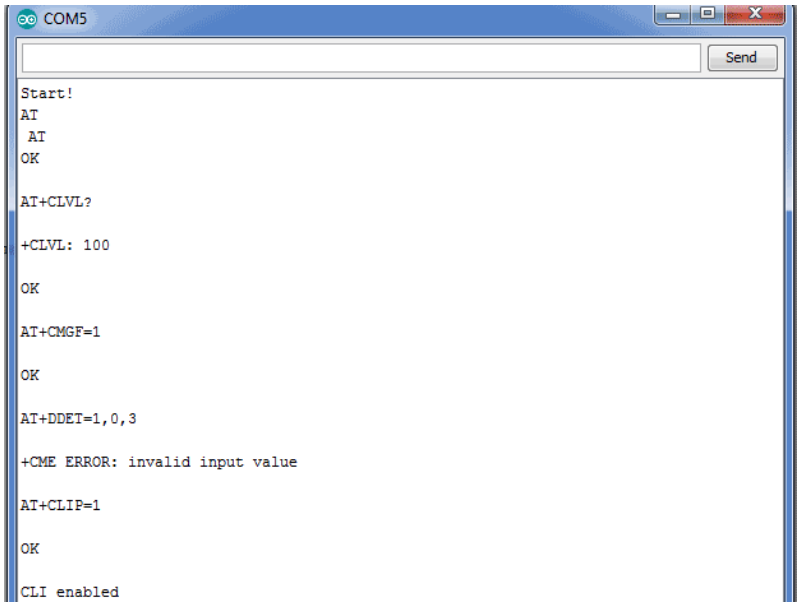


Рис. 13.5. Результати виконання AT команд.

Тепер залишається трішки модифікувати процедуру loop() таким чином, щоб можна було аналізувати інші незепрошені відповіді і повідомлення в основному тілі прогами:

```
  void loop () {
      if (SIM800.available ()) {// Якщо модем, щось відправив ...
        _response = waitResponse (); // Отримуємо відповідь від модему для аналізу
        Serial.println (_response); // Якщо потрібно виводимо в монітор порту
       // ... тут можна аналізувати дані отримані від GSM-модуля
     }
      if (Serial.available ()) {// Очікуємо команди по Serial ...
        SIM800.write (Serial.read ()); // ... і відправляємо отриману команду модему
   };
}
```

Тут і дальше, пропонується використати даний шаблон в якості основи для написання програми:

```
  #include <SoftwareSerial.h> // Бібліотека програмної реалізації обміну по UART-протоколу
  SoftwareSerial SIM800 (8, 9); // RX, TX
  String _response = ""; // Змінна для зберігання відповіді модуля
  void setup () {
    Serial.begin (9600); // Швидкість обміну даними з комп'ютером
    SIM800.begin (9600); // Швидкість обміну даними з модемом
    Serial.println ( "Start!");

    sendATCommand ( «AT«, true); // Автонастройка швидкості
  // sendATCommand ( "AT + CLVL?", True); // запитувана гучність динаміка
  // sendATCommand ( "AT + CMGF = 1", true); // Включити TextMode для SMS
  // sendATCommand ( "AT + DDET = 1,0,0", true); // Включити DTMF
  }

  String sendATCommand (String cmd, bool waiting) {
```

```
    String _resp = ""; // Змінна для зберігання результату
    Serial.println (cmd); // дублюємо команду в монітор порту
    SIM800.println (cmd); // Відправляємо команду модулю
    if (waiting) {// Якщо необхідно дочекатися відповіді ...
      _resp = waitResponse (); // ... чекаємо, коли передадуть відповідь
      // Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки можна закоментувати
      if (_resp.startsWith (cmd)) {// Прибираємо з відповіді дублюємо команду
        _resp = _resp.substring (_resp.indexOf ( "\ r", cmd.length ()) + 2);
      }
      Serial.println (_resp); // Дублюючи відповідь в монітор порту
    }
    return _resp; // Повертаємо результат. Пусто, якщо проблема
  }

  String waitResponse () {// Функція очікування відповіді і повернення отриманого результату
    String _resp = ""; // Змінна для зберігання результату
    long _timeout = millis () + 10000; // Змінна для відстеження таймаута (10 секунд)
    while (! SIM800.available () && millis () <_timeout) {}; // Чекаємо відповіді 10 секунд, якщо прийшла відповідь або настав таймаут, то ...
    if (SIM800.available ()) {// Якщо є, що зчитувати ...
      _resp = SIM800.readString (); // ... зчитуємо і запам'ятовуємо
    }
    else {// Якщо прийшов таймаут, то ...
      Serial.println ( "Timeout ..."); // ... сповіщаємо про це і ...
    }
    return _resp; //...повертаємо результат. Пусто, якщо проблема
  }

  void loop () {
```

```
    if (SIM800.available ()) {// Якщо модем, щось відправив ...
      _response = waitResponse (); // Отримуємо відповідь від модему для аналізу
      Serial.println (_response); // Якщо потрібно виводимо в монітор порту
      // ... тут можна аналізувати дані отримані від GSM-модуля
    }
    if (Serial.available ()) {// Очікуємо команди по Serial ...
      SIM800.write (Serial.read ()); //...і відправляємо отриману команду модему
    };
}
```

Вхідні/вихідні виклики

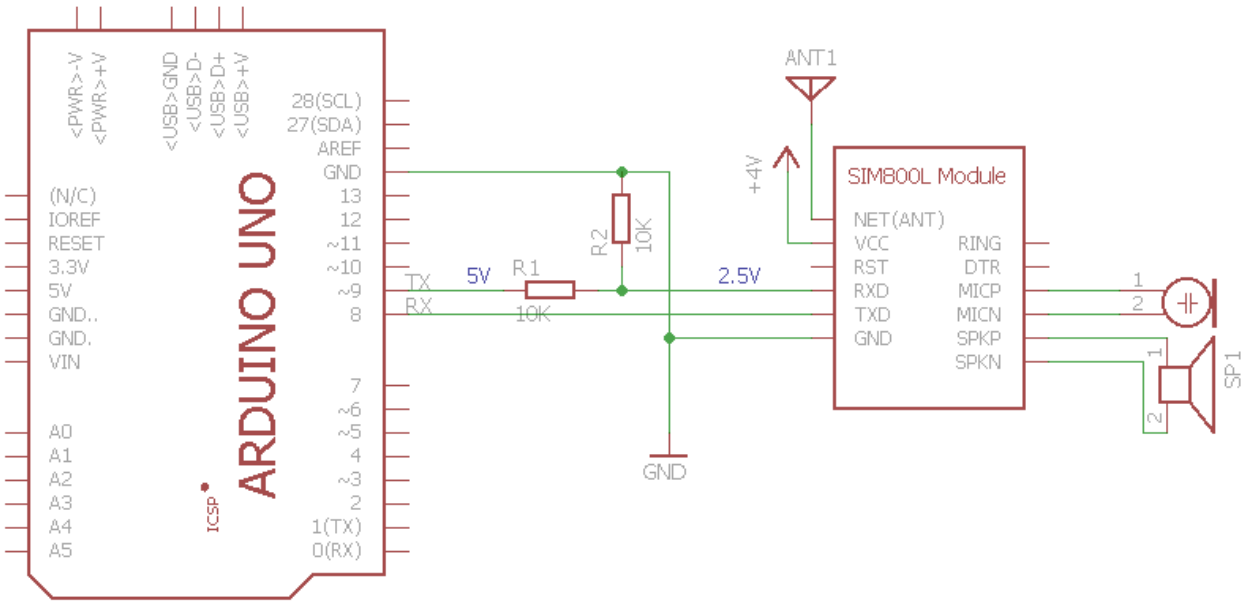


Рис. 13.6. Схема підключення.

Можливі ситуації, коли необхідно скористатися голосовими функціями модуля SIM800L, а саме здійснювати вихідні дзвінки в якості оперативних повідомлень, або приймати вхідні дзвінки, для здійснення управління по DTMF. Для використання голосових функцій за прямим призначенням, до аудіоінтерфейсу мікрофона (MICP і MICN) і колонок (SPKP і SPKN) можна безпосередньо підключити роздільні динамік і мікрофон або дротову гарнітуру, через роз'єм Jack 3.5 мм PJ-342:

Основні команди для управління голосовими функціями представлені в таблиці:

Опис	Команда, опис параметрів	Відповіді	Приклад
Здійснити вихідний	**ATD<*phonenumber*>;** <***phonenumber***> — номер абонента, що	**OK**, **BUSY**,	ATD+38004522441;

виклик	викликається в повному форматі (+38004522441)	**NO DIALTONE** та ін.	OK
Повторити набір останнього набраного номера	**ATDL**	**OK**, **BUSY**, **NO DIALTONE** та ін.	ATDL OK
Установка гучності динаміка під час розмови	**AT+CLVL=<*level*>** **<*level*>** — необхідний рівень гучності (ціле число в діапазоні 0 ... 100) *значення автоматично зберігається*	**OK**, **ERROR**	AT+CLVL=100 OK
Установка гучності дзвінка	**AT+CRSL=<*level*>** **<*level*>** — необхідний рівень гучності (ціле число в діапазоні 0 ... 100) *значення автоматично зберігається*	**OK**, **ERROR**	AT+CRSL=100 OK
Відповісти на вхідний дзвінок	**ATA** *всі команди записані в цьому ж рядку після команди ATA ігноруються*	**OK**	ATA OK
Скинути вхідний дзвінок	**ATH** *всі команди записані в цьому ж рядку після команди ATH ігноруються*	**OK**	ATA OK
Автоматична відповідь при вхідному виклику	**ATS0=<*n*>** **<*n*>** — кількість гудків до автоматичної відповіді під час отримання виклику: 0 - автовідповідь відключений 1 ... 255 - кількість дзвінків, після яких модуль автоматично відповість на вхідний дзвінок. *при установці занадто великої кількості гудків, виклик може бути перерваний оператором*	**OK**, **ERROR**	ATS0=2 OK
Включення Автовизначник номера (АВН)	**AT+CLIP=<*n*>** **<*n*>** — параметр включення функції: 0 - АОН вимкнений 1 - АОН включений *параметр не зберігається, необхідна установка при кожному запуску, за замовчуванням значення 0*	**OK**	AT+CLIP=1 OK
Незапрошуваних повідомлення + CLIP	**+CLIP:** <*number*>,<*type*>[,<*subaddr*>,<*satype*>,<*alphaId*>,<*CLI validity*>] <*Number*> - *телефонний номер абонента* <*Type*> - *формат зазначеного телефонного номера* *129 - тип невідомий* *161 - національний номер*	**OK**	RING +CLIP: "+38004522441",145,"",0,"",0

	145 - номер в міжнародному форматі *177 - внутрішній номер оператора* *<Subaddr> - додатковий номер* *<Satype> - формат додаткового номера* *<AlphaId> - порядковий номер абонента в телефонній книзі (якщо записаний)* *<CLI validity> - статус АОН* *0 - номер визначено* *1 - абонент заборонив визначати свій номер* *2 - АОН недоступний або обмежений мережею*		
Незапрошувані повідомлення RING (дзвінок)	**RING**		RING

Для здійснення вихідних дзвінків використовується команда ATD <phonenumber>; де <phonenumber> - номер телефону в будь-якому повному форматі - +380 *******, наприклад, ATD + 380500005500;

Для того, щоб відстежити момент відповіді на вихідний дзвінок, необхідно виконати команду AT + COLP = 1. Справа в тому, що за замовчуванням (AT + COLP = 0), модуль відповість ОК, відразу після відправки команди ATD. Команда AT + COLP = 1 встановлює такий режим, при якому, після відправки команди модулю, відповідь ОК буде отримано, тільки коли виклик буде прийнятий, інакше - BUSY, NO DIAL TONE, NO CARRIER.

Відповісти на вхідний дзвінок в ручному режимі можна командою ATA. Або командою ATS0 = <n> можна встановити можливість автоматичної відповіді після <n> -гудків.

Для того, щоб реагувати на вхідні дзвінки, потрібно відстежувати незапрошувані повідомлення RING, і далі вживати заходи - або відповідати на вхідний дзвінок командою ATA, або скидати його командою ATH.

Для того, щоб автоматично відповідати на всі вхідні дзвінки досить один раз встановити автовідповідь командою ATS0 = 1.

```
  #include <SoftwareSerial.h> //Бібліотека програмної реалізації обміну по UART-протоколу
  SoftwareSerial SIM800 (8, 9); //RX, TX
  String _response = ""; //Змінна для зберігання відповіді модуля
  void setup () {
    Serial.begin (9600); //Швидкість обміну даними з комп'ютером
    SIM800.begin (9600); //Швидкість обміну даними з модемом
    Serial.println ( "Start!");
```

```
    sendATCommand ( «AT«, true); //Відправили AT для
настройки швидкості обміну даними
    // Команди налаштування модему при кожному запуску
    //  _ response = sendATCommand ( "AT + CLIP = 1",
true); //Включаємо АОН
    //  _ response = sendATCommand ( "AT + DDET = 1",
true); //Включаємо DTMF
  }
  String sendATCommand (String cmd, bool waiting) {
    String _resp = ""; //Змінна для зберігання результату
    Serial.println (cmd); //дублюємо команду в монітор
порту
    SIM800.println (cmd); //Відправляємо команду модулю
    if (waiting) {// Якщо необхідно дочекатися відповіді
...
      _resp = waitResponse (); //... чекаємо, коли буде
передана відповідь
      // Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки
можна закоментувати
      if  (_resp.startsWith  (cmd))  {//Прибираємо  з
відповіді дублюємо команду
        _resp = _resp.substring (_resp.indexOf ( "\ r",
cmd.length ()) + 2);
      }
      Serial.println (_resp); // дублюємо відповідь в
монітор порту
    }
    return _resp; //Повертаємо результат. Пусто, якщо
проблема
  }
  String waitResponse () {//Функція очікування відповіді
і повернення отриманого результату
    String _resp = ""; //Змінна для зберігання результату
    long _timeout = millis () + 10000; // Змінна для
відстеження таймаута (10 секунд)
    while (! SIM800.available () && millis () <_timeout)
{}; //Чекаємо відповіді 10 секунд, якщо прийшла відповідь
або настав таймаут, то ...
    if (SIM800.available ()) {// Якщо є, що зчитувати ...
      _resp = SIM800.readString (); //... зчитуємо і
запам'ятовуємо
    }
    else {//Якщо прийшов таймаут, то ...
      Serial.println ( "Timeout ..."); //... сповіщаємо
про це і ...
```

```
    }
    return _resp; //... повертаємо результат. Пусто, якщо
проблема
  }
  void loop () {
    if (SIM800.available ()) {//Якщо модем, щось
відправив ...
      _response = waitResponse (); //Отримуємо відповідь
від модему для аналізу
      _response.trim (); //Прибираємо зайві пробіли на
початку і кінці
      Serial.println (_response); //Якщо потрібно
виводимо в монітор порту
      if (_response.startsWith ( "RING")) {//Є вхідний
дзвінок
        sendATCommand ( "ATA", true); //Відповідаємо на
виклик
        // або
        sendATCommand ( "ATH", true); //Відхиляємо виклик
      }
    }
    if (Serial.available ()) {//Очікуємо команди по
Serial ...
      SIM800.write (Serial.read ()); //...і відправляємо
отриману команду модему
    };
}
```

Щоб забезпечити індивідуальний підхід до кожного виклику необхідно включити Автовизначник номера (АВН) - команда AT + CLIP = 1 (за замовчуванням відключений). Після того як АВН включений, кожне незапрошуване повідомлення RING буде додатково містити інформацію про номер телефону:

Тепер неважко зробити білий список телефонних номерів, на які модуль буде автоматично відповідати. Усі вхідні дзвінки із сторонніх номерів, які не включені в білий список, будуть скидатися:

```
#include <SoftwareSerial.h> //Бібліотека програмної
реалізації обміну по UART-протоколу
SoftwareSerial SIM800 (8, 9); //RX, TX
String _response = ""; //Змінна для зберігання відповіді
модуля
void setup () {
  Serial.begin (9600); //Швидкість обміну даними з
комп'ютером
  SIM800.begin (9600); //Швидкість обміну даними з
модемом
```

```
  Serial.println ( "Start!");
  sendATCommand ( «AT«, true); //Відправили AT для
настройки швидкості обміну даними
  //Команди налаштування модему при кожному запуску
  _response = sendATCommand ( "AT + CLIP = 1", true);
//Включаємо АОН
  // _ response = sendATCommand ( "AT + DDET = 1", true);
//Включаємо DTMF
}
String sendATCommand (String cmd, bool waiting) {
  String _resp = ""; //Змінна для зберігання результату
  Serial.println (cmd); //дублюємо команду в монітор
порту
  SIM800.println (cmd); //Відправляємо команду модулю
  if (waiting) {//Якщо необхідно дочекатися відповіді ...
    _resp = waitResponse (); // ... чекаємо, коли
передадуть відповідь
    //Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки
можна закоментувати
    if (_resp.startsWith (cmd)) {//Прибираємо з відповіді
дублюємо команду
      _resp = _resp.substring (_resp.indexOf ( "\ r",
cmd.length ()) + 2);
    }
    Serial.println (_resp); //Дублюючи відповідь в
монітор порту
  }
  return _resp; //Повертаємо результат. Пусто, якщо
проблема
}
String waitResponse () {//Функція очікування відповіді і
повернення отриманого результату
  String _resp = ""; //Змінна для зберігання результату
  long _timeout = millis () + 10000; //Змінна для
відстеження таймаута (10 секунд)
  while (! SIM800.available () && millis () <_timeout)
{}; //Чекаємо відповіді 10 секунд, якщо прийшла відповідь
або настав таймаут, то ...
  if (SIM800.available ()) {//Якщо є, що зчитувати ...
    _resp = SIM800.readString (); //... зчитуємо і
запам'ятовуємо
  }
  else {//Якщо прийшов таймаут, то ...
    Serial.println ( "Timeout ..."); //... сповіщаємо про
це і ...
```

```
  }
  return _resp; //... повертаємо результат. Пусто, якщо
проблема
}

void loop () {
  if (SIM800.available ()) {//Якщо модем, щось відправив
...
    _response = waitResponse (); //Отримуємо відповідь
від модему для аналізу
    _response.trim (); //Прибираємо зайві пробіли на
початку і в кінці
    Serial.println (_response); //Якщо потрібно виводимо
в монітор порту
    String whiteListPhones = "+ 7928xxxxxxx, +
7928xxxxxxx, + 7928xxxxxxx"; //Білий список телефонів
    if (_response.startsWith ( "RING")) {//Є вхідний
дзвінок
      int phoneindex = _response.indexOf ( "+ CLIP: \"
"); //Чи є інформація про визначення номера, якщо так, то
phoneindex> -1
      String innerPhone = ""; //Змінна для зберігання
певного номера
      if (phoneindex> = 0) {//Якщо інформація була
знайдена
        phoneindex + = 8; //Парс рядок і ...
        innerPhone = _response.substring (phoneindex,
_response.indexOf ( "\" ", phoneindex)); //... отримуємо
номер
        Serial.println ( "Number:" + innerPhone);
//Виводимо номер в монітор порту
      }
      //Перевіряємо, щоб довжина номера була більше 6
цифр, і номер повинен бути в списку
      if (innerPhone.length ()> = 7 &&
whiteListPhones.indexOf (innerPhone)> = 0) {
        sendATCommand ( "ATA", true); //Якщо так, то
відповідаємо на виклик
      }
      else {

        sendATCommand ( "ATH", true); //Якщо немає, то
відхиляємо виклик
      }
    }
```

```
  }
  if (Serial.available ()) {//Очікуємо команди по Serial
    SIM800.write (Serial.read ()); //... і відправляємо
отриману команду модему
  };
}
```

Прийом/відправка/видалення SMS

Тема, яка акутальна при взаємодії з GSM-модулем в 95% випадків - це, звичайно ж, прийом і відправка SMS, за допомогою яких відбувається віддалене управління чим-небудь, наприклад, мікроконтролером.

В даному розділі буде розглянута робота з SMS в текстовому форматі (Text Mode), всі приклади будуть також наведені для роботи з SMS в текстовому форматі. Текстовий формат набагато простіший, але він не дозволяє обмінюватися SMS-повідомленнями на мовах, відмінних від англійської. PDU-формат має більш великі можливості, але досить складний в освоєнні і використанні.

Для взаємодії з SMS існує значний список команд. Найбільш часто застосовані із них наведені в таблиці:

Описание	Команда	Параметр	Відповідь	Приклади
Вибір формату SMS	AT+CMGF=<*mode*>	<Mode> - формат повідомлень, значення: 0 - PDU-формат (за замовчуванням) 1 - текстовий формат	OK	**AT+CMGF=1** формат SMS в текстовий формат (Text Mode)
Видалити SMS	**AT+CMGD=<*index*>[,<*delflag*>]**	<Index> - порядковий номер видаляється повідомлення в заданій групі (> 0) <Delflag> - фільтр видаляються повідомлень (необов'язково), Значення: 0 - видалити повідомлення із заданим порядковим номером <index> 1 ... 3 - видалення повідомлень, залишаючи повідомлення за різними критеріями (див. Datasheet) 4 - видалити всі повідомлення, включаючи	OK	**AT+CMGD=0,4** видалення всіх повідомлень

		непрочитані		
Видалити всі SMS	**AT+CMGDA=<*type*>**	<Type> - тип видаляються повідомлень, значення (для TextMode): "DEL READ" - видалити всі прочитані SMS "DEL UNREAD" - видалити всі непрочитані SMS "DEL SENT" - видалити всі відправлені SMS "DEL UNSENT" - видалити всі невідправлені SMS "DEL INBOX" - видалити всі отримані SMS "DEL ALL" - видалити всі SMS	OK	**AT+CMGDA="DEL ALL"** видалення всіх повідомлень
Отримати список SMS	**AT+CMGL=<*stat*>[,<*mode*>]**	<Stat> - фільтр, значення "REC UNREAD" - отримані непрочитані SMS "REC READ" - отримані переглянуті SMS "STO UNSENT" - збережені непрочитані SMS "STO SENT" - збережені непрочитані SMS "ALL" - все SMS <Mode> - фільтр видаляються повідомлень (необов'язково, Використовується при <index> = 0), значення: 0 - змінити статус повідомлень на «прочитано» 1 - залишити статус без зміни	OK	AT+CMGL="REC UNREAD", 1 отримати прийняті непрочитані повідомлення, статус не змінювати відповідь: +CMGL: 1,"REC UNREAD","w4348435","","17/05/14,15:10:02+12" 041C04270421002....7043E0432 +CMGL: 2,"REC UNREAD","+3828xxxxxxx","","17/05/15,23:34:03+12" Test +CMGL: 3,"REC UNREAD","+3828xxxxxxx","","17/05/22,18:13:25+12" Balance OK
Прочитати SMS	**AT+CMGR=<*index*>[,<*mode*>]**	<Index> - порядковий номер повідомлення <Mode> - дія після	OK	**AT+CMGR=7,1** прочитати 7-е повідомлення, статус не змінювати

		прочитання, значення: 0 - змінити статус повідомлень на «прочитано» (по-замовчуванню) 1 - залишити статус без зміни		відповідь: + CMGR: "REC READ", "+3828xxxxxxx", "", "17/05 / 22,18: 13: 25 + 12" Balance OK
Відправити SMS	**AT+CMGS=<*da*>[,<*toda*>]<*CR*>текст сообщения<*ctrl-Z/ESC*>**	<Da> - телефон адресата в міжнародному форматі, в лапках <Toda> - додатковий номер (не використовується) <CR> - початок рядка, після відправки модуль переходить в режим прийому тексту повідомлення Після того як текст повідомлення переданий, необхідно відправити або <ctrl-Z> для відправки повідомлення, або <ESC> для скасування.	+CMGS: <n> OK	**AT+CMGS="+3828xxxxxxx"** > test message > +CMGS: 81 OK
Незапрошувані повідомлення про прихід нового SMS	**+CMTI: <*mem*>,<*index*>**	<Mem> - сховище повідомлень "SM" - SIM-карта "ME" - пам'ять модему / телефону <Index> - індекс отриманого повідомлення (показує загальне число отриманих непрочитаних SMS)		**+CMTI: "SM",3**
Отримати інформацію по сховищах SMS	**AT+CPMS?** Ответ: **+CPMS: <*mem1*>,<*used1*>,<*total1*>, <*mem2*>,<*used2*>,<*total2*>,**	<Mem1> - читання і видалення повідомлень відбувається їх цього сховища <Mem2> - повідомлення буде записано в це сховище і відправлено <Mem3> - одержувані повідомлення будуть записуватися в це сховище	OK	**AT+CPMS?**

	<mem3>,<used3>,<total3>	<Mem1>, <mem2>, <mem3> можуть мати наступні значення: "SM" - пам'ять SIM-карти "ME" - пам'ять модему / телефону "SM_P" - краща пам'ять SIM-карти "ME_P" - краща пам'ять модему / телефону "MT" - пам'ять SIM-карти і модему / телефону (SM + ME) <Usedx> - кількість повідомлень в сховищі <memx> <Totalx> - максимальна кількість повідомлень зберігається в сховищі <memx>		

Перед роботою з SMS, для використання текстового формату, його потрібно включити командою AT+CMGF=1. Оскільки цей параметр зберігається по команді AT&W, в скетчі він буде встановлюватися на початку в форматі «кілька команд в одному рядку»: AT + CMGF = 1; &W

Відправлення SMS. Формат відправки SMS в текстовому форматі виглядає наступним чином:

AT+CMGS = <phonenumber>, де <phonenumber> - повний номер телефону адресата, в будь-якому форматі в лапках, наприклад AT + CMGS = "+ 380500008100". Після виконання цієї команди з'явиться запрошення ввести текст SMS ->. Після введення повідомлення, про його закінчення потрібно повідомити - передати Ctrl + Z (або ESC для виходу без відправки). Програмно, поєднання клавіш Ctrl + Z можна замінити як (char) 26 (так як ASCII код символу - 26).

Якщо відправка SMS здійснюється через вікно Serial середовища Arduino IDE, то після введення основного тексту повідомлення необхідно відправити модулю команду завершення SMS - Ctrl + Z. Зробити це можна копіюванням символу між лапками "" і подальшої його вставкою в поле відправки даних.

Приклад програмної відправки SMS (для зручності, функціонал відправки SMS винесено в окрему функцію sendSMS()). В даному прикладі також використовується контроль статусу відправки. У разі невдалої спроби відправки, повідомлення про це з'явиться в терміналі:

```
#include <SoftwareSerial.h> //Бібліотека програмної реалізації обміну по UART-протоколу
```

```
SoftwareSerial SIM800 (8, 9); //RX, TX
String _response = ""; //Змінна для зберігання відповіді
модуля
void setup () {
  Serial.begin (9600); //Швидкість обміну даними з
комп'ютером
  SIM800.begin (9600); //Швидкість обміну даними з
модемом
  Serial.println ( "Start!");
  sendATCommand ( «AT«, true); //Відправили AT для
настройки швидкості обміну даними
  // Команди налаштування модему при кожному запуску
  // _ response = sendATCommand ( "AT + CLIP = 1", true);
// Включаємо АОН
  // _ response = sendATCommand ( "AT + DDET = 1", true);
// Включаємо DTMF
  _response = sendATCommand ( "AT + CMGF = 1; & W",
true); // Включаємо текстовий режиму SMS (Text mode) і
відразу зберігаємо значення (AT & W)!
  sendSMS ( "+ 3828xxxxxxx", "test message");
}
String sendATCommand (String cmd, bool waiting) {
  String _resp = ""; //Змінна для зберігання результату
  Serial.println (cmd); //дублюємо команду в монітор
порту
  SIM800.println (cmd); //Відправляємо команду модулю
  if (waiting) {// Якщо необхідно дочекатися відповіді
...
    _resp = waitResponse (); //... чекаємо, коли буде
передана відповідь
    // Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки
можна закоментувати
    if (_resp.startsWith (cmd)) {//Прибираємо з відповіді
дублює команду
      _resp = _resp.substring (_resp.indexOf ( "\ r",
cmd.length ()) + 2);
    }
    Serial.println (_resp); //дублюємо відповідь в
монітор порту
  }
  return _resp; //Повертаємо результат. Пусто, якщо
проблема
}
String waitResponse () {//Функція очікування відповіді і
повернення отриманого результату
```

```
  String _resp = ""; //Змінна для зберігання результату
  long _timeout = millis () + 10000; //Змінна для
відстеження таймаута (10 секунд)
  while (! SIM800.available () && millis () <_timeout)
{}; //Чекаємо відповіді 10 секунд, якщо прийшла відповідь
або настав таймаут, то ...
  if (SIM800.available ()) {//Якщо є, що зчитувати ...
    _resp = SIM800.readString (); //... зчитуємо і
запам'ятовуємо
  }
  else {//Якщо прийшов таймаут, то ...
    Serial.println ( "Timeout ..."); //... сповіщаємо про
це і ...
  }
  return _resp; //... повертаємо результат. Пусто, якщо
проблема
}
void loop () {
  if (SIM800.available ()) {//Якщо модем, щось відправив
...
    _response = waitResponse (); //Отримуємо відповідь
від модему для аналізу
    _response.trim (); //Прибираємо зайві пробіли на
початку і в кінці
    Serial.println (_response); //Якщо потрібно виводимо
в монітор порту
    // ....
    if (_response.startsWith ( "+ CMGS:")) {//Прийшло
повідомлення про відправку SMS
      int index = _response.lastIndexOf ( "\ r \ n");
//Знаходимо останнє перенесення рядка, перед статусом
      String result = _response.substring (index + 2,
_response.length ()); //Отримуємо статус
      result.trim (); //Прибираємо пробільні символи на
початку / кінці
      if (result == "OK") {//Якщо результат OK - все
нормально
        Serial.println ( "Message was sent. OK");
      }
      else {//Якщо немає, потрібно повторити відправку
        Serial.println ("Message was not sent. Error");
      }
    }
  }
```

```
  if (Serial.available ()) {//Очікуємо команди по Serial
...
    SIM800.write (Serial.read ()); //...і відправляємо
отриману команду модему
  };
}
void sendSMS (String phone, String message)
{
  sendATCommand ( "AT + CMGS = \" "+ phone +" \ "",
true); //Переходимо в режим введення тексту повідомлення
  sendATCommand (message + "\ r \ n" + (String) ((char)
26), true); //Після тексту відправляємо перенесення рядка
і Ctrl + Z
}
```

Як отримати та прочитати SMS. Тут не так все просто, як при відправці повідомлень. Справа в тому, що отримані повідомлення зберігаються в пам'яті SIM-карти. І пам'ять ця обмежена. Обсяг пам'яті можна дізнатися командою AT + CPMS ?. Якщо допустити переповнення пам'яті, повідомлення більше не зможуть приходити. Тому, щоб зберегти працездатність програми, після отримання кожного повідомлення, його потрібно обробляти і видаляти. Мануал надає великий набір команд, за допомогою яких можна здійснювати будь-які маніпуляції з прийнятими повідомленнями.

Під час приходу SMS, SIM800L генерує незапрошувані повідомлення виду +CMTI: "SM", 4. Після приходу такого повідомлення, можна програмно ініціювати процедуру читання командою AT + CMGR = <index>, <mode>, де в якості параметра <index> необхідно вказати індекс, отриманий в повідомленні - + CMTI: "SM", 4. У наведеному нижче прикладі, після отримання та обробки вхідних повідомлень, всі повідомлення видаляються командою AT + CMGDA = "DEL ALL" для економії пам'яті модуля:

```
#include <SoftwareSerial.h> //Бібліотека програмної
реалізації обміну по UART-протоколу
SoftwareSerial SIM800 (8, 9); // RX, TX
String _response = ""; //Змінна для зберігання відповіді
модуля
void setup () {
  Serial.begin (9600); //Швидкість обміну даними з
комп'ютером
  SIM800.begin (9600); //Швидкість обміну даними з
модемом
  Serial.println ( "Start!");
  sendATCommand ( «AT«, true); //Відправили AT для
настройки швидкості обміну даними
  //Команди налаштування модему при кожному запуску
```

```
  //_ response = sendATCommand ( "AT + CLIP = 1", true);
//Включаємо АОН
  //_ response = sendATCommand ( "AT + DDET = 1", true);
//Включаємо DTMF
  _response = sendATCommand ( "AT + CMGF = 1; & W",
true); //Включаємо текстовий режиму SMS (Text mode) і
відразу зберігаємо значення (AT & W)!
}
String sendATCommand (String cmd, bool waiting) {
  String _resp = ""; //Змінна для зберігання результату
  Serial.println (cmd); //дублюємо команду в монітор
порту
  SIM800.println (cmd); //Відправляємо команду модулю
  if (waiting) {//Якщо необхідно дочекатися відповіді ...
    _resp = waitResponse (); //... чекаємо, коли буде
передана відповідь
    //Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки
можна закоментувати
    if (_resp.startsWith (cmd)) {//Прибираємо з відповіді
дублює команду
      _resp = _resp.substring (_resp.indexOf ( "\ r",
cmd.length ()) + 2);
    }
    Serial.println (_resp); //дублюємо відповідь в
монітор порту
  }
  return _resp; //Повертаємо результат. Пусто, якщо
проблема
}
String waitResponse () {//Функція очікування відповіді і
повернення отриманого результату
  String _resp = ""; //Змінна для зберігання результату
  long _timeout = millis () + 10000; //Змінна для
відстеження таймаута (10 секунд)
  while (! SIM800.available () && millis () <_timeout)
{}; //Чекаємо відповіді 10 секунд, якщо прийшла відповідь
або настав таймаут, то ...
  if (SIM800.available ()) {//Якщо є, що зчитувати ...
    _resp = SIM800.readString (); //... зчитуємо і
запам'ятовуємо
```

```
  }
  else {//Якщо прийшов таймаут, то ...
    Serial.println ( "Timeout ..."); /... сповіщаємо про
це і ...
  }
  return _resp; /... повертаємо результат. Пусто, якщо
проблема
}
void loop () {
  if (SIM800.available ()) {//Якщо модем, щось відправив
...
    _response = waitResponse (); //Отримуємо відповідь
від модему для аналізу
    _response.trim (); //Прибираємо зайві пробіли на
початку і в кінці
    Serial.println (_response); //Якщо потрібно виводимо
в монітор порту
    if (_response.startsWith ( "+ CMTI:")) {//Прийшло
повідомлення про відправку SMS
      int index = _response.lastIndexOf ( ","); //
Знаходимо останню кому, перед індексом
      String result = _response.substring (index + 1,
_response.length ()); //Одержуємо індекс
      result.trim (); //Прибираємо пробільні символи на
початку / кінці
      _response = sendATCommand ( "AT + CMGR =" + result,
true); //Отримати вміст SMS
      parseSMS (_response); // розпарсити SMS на елементи
      sendATCommand ( "AT + CMGDA = \" DEL ALL \ "",
true); //Видалити всі повідомлення, щоб не забивали
пам'ять модуля
    }
  }
  if (Serial.available ()) {// Очікуємо команди по Serial
...
    SIM800.write (Serial.read ()); //... і відправляємо
отриману команду модему
  };
```

```
void parseSMS (String msg) {
  String msgheader = "";
  String msgbody = "";
  String msgphone = "";
  msg = msg.substring (msg.indexOf ( "+ CMGR:"));
  msgheader = msg.substring (0, msg.indexOf ( "\ r"));
  msgbody = msg.substring (msgheader.length () + 2);
  msgbody = msgbody.substring (0, msgbody.lastIndexOf (
"OK"));
  msgbody.trim ();
  int firstIndex = msgheader.indexOf ( "\", \ "«) + 3;
  int secondIndex = msgheader.indexOf ( "\", \ "",
firstIndex);
  msgphone = msgheader.substring (firstIndex,
secondIndex);
  Serial.println ( "Phone:" + msgphone);
  Serial.println ( "Message:" + msgbody);
  //Далі пишемо логіку обробки SMS-команд.
  //Тут також можна реалізовувати перевірку за номером
телефону
  //І якщо номер некоректний, то просто видалити
повідомлення.
}
void sendSMS (String phone, String message)
{
  sendATCommand ( "AT + CMGS = \" "+ phone +" \ "",
true); // Переходимо в режим введення тексту повідомлення
  sendATCommand (message + "\ r \ n" + (String) ((char)
26), true); //Після тексту відправляємо перенесення рядка
і Ctrl + Z
}
```

Ну а що станеться, якщо МК пропустить незапрошувані повідомлення про прихід SMS? Йому доведеться чекати приходу наступного повідомлення, що є неприйнятним сценарієм. І якщо буде прийнято відразу кілька повідомлень - буде оброблено лише останнє. Для підстраховки від таких ситуацій необхідно ввести періодичну (наприклад, раз на хвилину) перевірку наявності непрочитаних повідомлень. У разі наявності таких, кожне повідомлення зі списку буде оброблено і видалено. Приклад скетчу наведено нижче в заголовку про управління по SMS.

Управління по SMS

Для демонстрації можливостей управління зі зворотним зв'язком буде використовуватися наступна схема з 3 світлодіодами:

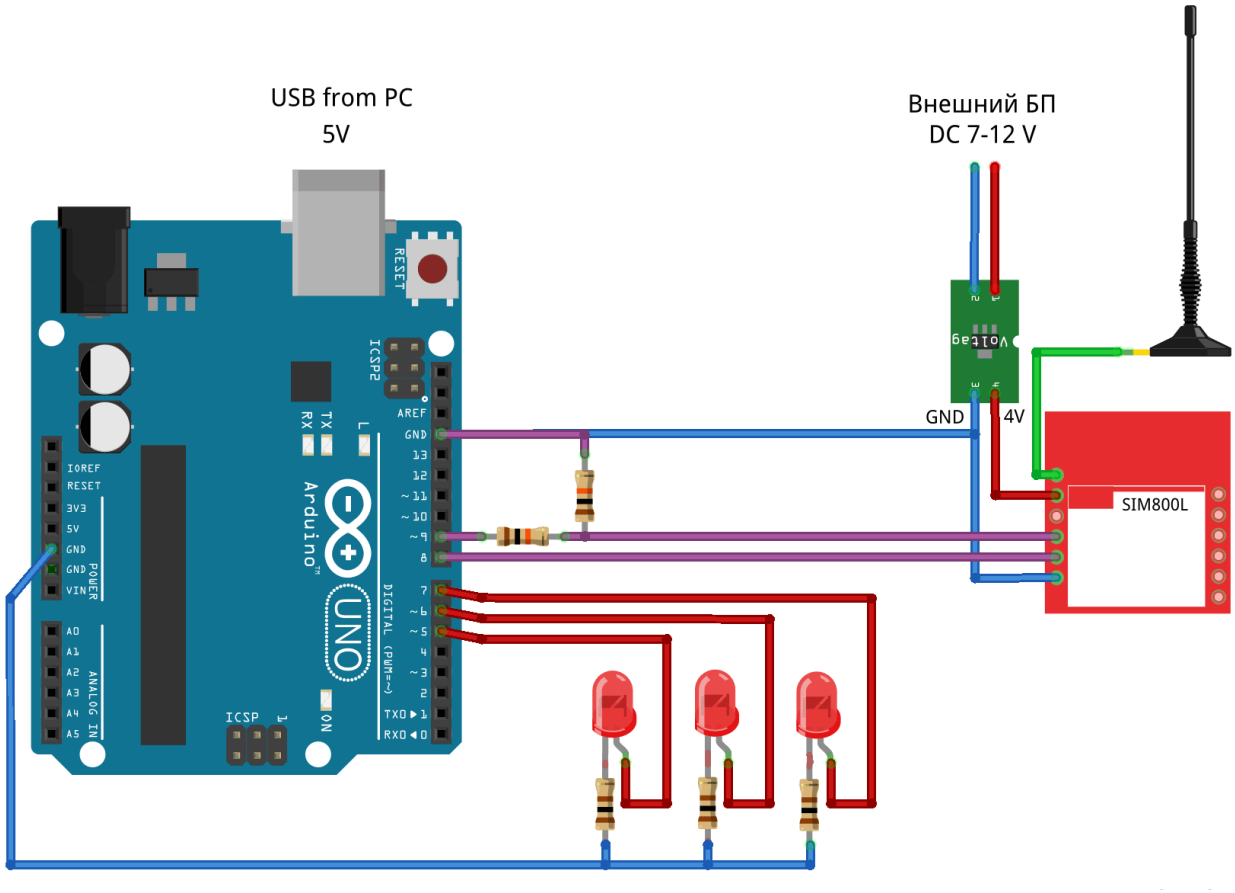


Рис. 13.6. Схема простої GSM сигналізації.

Команди будуть складатися з двох цифр - перша цифра означатиме номер світлодіода (1-3), друга - його стан (1 - ввімкнено, 0 - вимкнено). Виходить, що допустимих команд (поєднань цифр) всього 6: 11, 10, 21, 20, 31, 30. Інші команди будуть ігноруватися. Інформація про статус виконання або некоректності команди, буде виводитися в термінал скетчу:

```
#include <SoftwareSerial.h>
SoftwareSerial SIM800 (8, 9); //RX, TX
int pins [3] = {5, 6, 7}; //Виводи з підключеними
світлодіодами
String _response = ""; //Змінна для зберігання відповіді
модуля
long lastUpdate = millis (); //Час останнього оновлення
long updatePeriod = 60000; //Перевіряти кожну хвилину
String phones = "+ 3828xxxxxxx, + 3820xxxxxxx, +
3818xxxxxxx"; //Білий список телефонів
```

```
void setup () {
  for (int i = 0; i <3; i ++) {
  pinMode (pins [i], OUTPUT);//Налаштовуємо виводи
  }
  Serial.begin (9600); //Швидкість обміну даними з
комп'ютером
  SIM800.begin (9600); //Швидкість обміну даними з
модемом
  Serial.println ( "Start!");
  sendATCommand ( «AT«, true); //Відправили AT для
настройки швидкості обміну даними
  // sendATCommand ( "AT + CMGDA = \" DEL ALL \ "",
true); //Видаляємо всі SMS, щоб не завантажувати пам'ять

  //Команди налаштування модему при кожному запуску
  //_ response = sendATCommand ( "AT + CLIP = 1", true);
// Включаємо АОН
  //_ response = sendATCommand ( "AT + DDET = 1", true);
//Включаємо DTMF
  sendATCommand ( "AT + CMGF = 1; & W", true);
//Включаємо текстовий режим SMS (Text mode) і відразу
зберігаємо значення (AT & W)!
  lastUpdate = millis (); //Обнуляємо таймер
}

String sendATCommand (String cmd, bool waiting) {
  String _resp = ""; //Змінна для зберігання результату
  Serial.println (cmd); //дублюємо команду в монітор
порту
  SIM800.println (cmd); //Відправляємо команду модулю
  if (waiting) {//Якщо необхідно дочекатися відповіді ...
    _resp = waitResponse (); //... чекаємо, коли
передадуть відповідь
    //Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки
можна закоментувати
    if (_resp.startsWith (cmd)) {//Прибираємо з відповіді
дублюємо команду
      _resp = _resp.substring (_resp.indexOf ( "\ r",
cmd.length ()) + 2);
```

```
    }
    Serial.println (_resp); //Дублюємо відповідь в
монітор порту
  }
  return _resp; //Повертаємо результат. Пусто, якщо
проблема
}
String waitResponse () {//Функція очікування відповіді і
повернення отриманого результату
  String _resp = ""; //Змінна для зберігання результату
  long _timeout = millis () + 10000; //Змінна для
відстеження таймаута (10 секунд)
  while (! SIM800.available () && millis () <_timeout)
{}; //Чекаємо відповіді 10 секунд, якщо прийшла відповідь
або настав таймаут, то ...
  if (SIM800.available ()) {//Якщо є, що зчитувати ...
    _resp = SIM800.readString (); //... зчитуємо і
запам'ятовуємо
  }
  else {//Якщо прийшов таймаут, то ...
    Serial.println ( "Timeout ..."); //... сповіщаємо про
це і ...
  }
  return _resp; //... повертаємо результат. Пусто, якщо
проблема
}

bool hasmsg = false; //Прапор наявності повідомлень до
видалення
void loop () {
  if (lastUpdate + updatePeriod <millis ()) {//Пора
перевірити наявність нових повідомлень
    do {
      _response = sendATCommand ( "AT + CMGL = \" REC
UNREAD \ ", 1", true); //Відправляємо запит читання
непрочитаних повідомлень
      if (_response.indexOf ( "+ CMGL:")> -1) {//Якщо є
хоч одне, отримуємо його індекс
        int msgIndex = _response.substring
(_response.indexOf ( "+ CMGL:") + 7, _response.indexOf (
```

```
"\" REC UNREAD \ "", _response.indexOf ( "+ CMGL:")) - 1)
.toInt ( );
        char i = 0; //Оголошуємо лічильник спроб
        do {
          i ++; //Збільшуємо лічильник
          _response = sendATCommand ( "AT + CMGR =" +
(String) msgIndex + ", 1", true); //Пробуємо отримати
текст SMS за індексом
          _response.trim (); //Прибираємо пропуски на
початку / кінці
          if (_response.endsWith ( "OK")) {//Якщо
відповідь закінчується на "OK"
            if (! hasmsg) hasmsg = true; //Ставимо прапор
наявності повідомлень для видалення
            sendATCommand ( "AT + CMGR =" + (String)
msgIndex, true); //Робимо повідомлення прочитаним
```

USSD-запити

Часто буває корисно, щоб GSM-модуль сам відстежував стан балансу SIM-карти і вчасно інформував власника про наближення закінчення грошей. Як відомо найпростіший спосіб дізнатися про залишок - відправити USSD-запит.

Якщо в коді не передбачена обробка кодування UCS2 і для роботи використовується текстовий режим (Text Mode), то необхідно використовувати USSD-команди, які повертають повідомлення не на кирилиці.

Наприклад, стандартний USSD-запит перевірки балансу для оператора Водафон - *101# поверне відповідь в кодуванні UCS2 виду:1

+CUSD: 0, "003100390038002E.....02A0033003100390023", 72

Для повернення відповіді в текстовому режимі повинен використовуватися інший USSD-запит - #101#. Він поверне вже зрозумілу відповідь:

+ CUSD: 0, "Vash balans 198.02 r. Dlya Vas - nedelya besplatnogo SMS-obsh'eniya s druz'yami", 15

Робота з тональними кодами кнопок - DTMF

У ряді завдань може виникнути необхідність здійснювати управління модулем під час прийнятого виклику, за допомогою натискання цифрових клавіш телефону. Аналогічне управління організовано по меню допомоги стільникових операторів.

DTMF (англ. Dual-Tone Multi-Frequency, Тональний набір, тональний сигнал) - Двотональний багаточастотний аналоговий сигнал, який використовується для набору телефонного номера. Для кодування символу в DTMF складаються два синусоїдальних сигнали, частоти яких визначені заздалегідь.

У модулі SIM800L також є можливість декодування сигналу DTMF в конкретні коди. Для використання цієї можливості необхідно включити режим розпізнавання DTMF командою AT+DDET=1. Команда і її параметри докладно описані в таблиці:

Опис	Команда, опис параметрів	Відповіді	Незапрошувані відповіді
Функція переходу в режим DTMF	AT + DDET = <mode> [, <interval>] [, <reportMode>] [, <ssdet>] <Mode> - управління режимом: 0 - вимкнено 1 - ввімкнений <Interval> - мінімальний інтервал в мілісекундах між двома натисканнями однієї і тієї ж клавіші (діапазон допустимих значень 0-10000). За замовчуванням - 0. <ReportMode> - режим надання інформації: 0 - тільки код натиснутої кнопки 1 - код натиснутої кнопки і час утримання натискання, в мс <Ssdet> - управління функцією визначення одночастотне звуку (не використовується): 0 - виключена 1 - включена	**OK**, **Error**	Якщо <***reportMode***>=0, то: **+DTMF: <*key*>** Якщо <***reportMode***>=1, то: **+DTMF: <*key*>,<*last time*>** <***key***> — ідентифікатор натиснутою кнопки (0-9, *, #, A, B, C, D) <***last time***> — тривалість утримуючи кнопки, в мс

```
#include <SoftwareSerial.h> //Бібліотека програмної реалізації обміну по UART-протоколу
SoftwareSerial SIM800 (8, 9); //RX, TX
String _response = ""; //Змінна для зберігання відповіді модуля
void setup () {
  Serial.begin (9600); //Швидкість обміну даними з комп'ютером
  SIM800.begin (9600); //Швидкість обміну даними з модемом
  Serial.println ( "Start!");
```

```
  sendATCommand ( «AT«, true); //Відправили AT для
настройки швидкості обміну даними
  _response = sendATCommand ( "AT + DDET = 1", true); //
Включаємо DTMF
}
String sendATCommand (String cmd, bool waiting) {
  String _resp = ""; //Змінна для зберігання результату
  Serial.println (cmd); //дублюємо команду в монітор
порту
  SIM800.println (cmd); //Відправляємо команду модулю
  if (waiting) {//Якщо необхідно дочекатися відповіді ...
    _resp = waitResponse (); //... чекаємо, коли
передадуть відповідь
    // Якщо Echo Mode вимкнений (ATE0), то ці 3 рядки
можна закоментувати
    if (_resp.startsWith (cmd)) {// Прибираємо з
відповіді дублюємо команду
      _resp = _resp.substring (_resp.indexOf ( "\ r \ n",
cmd.length ()) + 2);
    }
    Serial.println (_resp); //Дублюючи відповідь в
монітор порту
  }
  return _resp; //Повертаємо результат. Пусто, якщо
проблема
}
String waitResponse () {//Функція очікування відповіді і
повернення отриманого результату
  String _resp = ""; //Змінна для зберігання результату
  long _timeout = millis () + 10000; //Змінна для
відстеження таймаута (10 секунд)
  while (! SIM800.available () && millis () <_timeout)
{}; //Чекаємо відповіді 10 секунд, якщо прийшла відповідь
або настав таймаут, то ...
  if (SIM800.available ()) {//Якщо є, що зчитувати ...
    _resp = SIM800.readString (); //... зчитуємо і
запам'ятовуємо
  }
  else {//Якщо прийшов таймаут, то ...
```

```
    Serial.println ( "Timeout ..."); //... сповіщаємо про
це і ...
  }
  return _resp; //... повертаємо результат. Пусто, якщо
проблема
}
void loop () {
  if (SIM800.available ()) // Очікуємо приходу даних
(відповіді) від модему ...
    Serial.write (SIM800.read ()); //... і виводимо їх в
Serial
  if (Serial.available ()) // Очікуємо команди по Serial
...
    SIM800.write (Serial.read ()); //... і відправляємо
отриману команду модему
}
```

Даний модуль також дозволяє вести обмін з сервером по протоколу GPRS, наприклад відправляти дані на сайт. Наприклад, відправляти зміни стану магнітного датчика (геркона), коли змінився стан датчика відправляємо на сайт час і стан датчика. Вийде прототип сигналізації із веденням журналу на відкривання/ закриваення дверей або вікон. Для роботи нам додатково знадобиться наявність власного сайту з підтримкою PHP, який можна створити на безкоштовному хостингу.

Приклад реализации на Arduino

```
#include <SoftwareSerial.h>
SoftwareSerial GSMport(2, 3); // RX, TX
int char_;
int SensorPin = 10;
int SensorState;
int SensorLastState = HIGH;

void setup() {
  delay(3000); //дамо час на ініціалізацію GSM модулю
  pinMode(SensorPin, INPUT);
  digitalWrite(SensorPin, HIGH);  //вкл. підтягаючий
резистор 20ом
  Serial.begin(9600);  //швидкість порта
  Serial.println("GPRS test");
```

```
  GSMport.begin(9600);
  gprs_init();
}

void loop() {
  SensorState = digitalRead(SensorPin);
  if (SensorState != SensorLastState) {  //якщо зміна
стану датчика
    Serial.print("sensor changed to: ");
    Serial.println(SensorState);
    SensorLastState = SensorState;
    gprs_send(String(SensorState));
    delay(100);
  }
  if (GSMport.available()) {  //якщо GSM модуль щось
послав нам, то
    Serial.println(ReadGSM());  //друкуємо в монітор
порту отриманий рядок
  }
  delay(100);
}
void gprs_init() {  //Процедура початковій ініціалізації
GSM модуля
  int d = 500;
  int ATsCount = 7;
  String ATs[] = {  //масив АТ команд
    "AT+SAPBR=3,1,\"CONTYPE\",\"GPRS\"",  //Установка
налаштувань підключення
    "AT+SAPBR=3,1,\"APN\",\"internet.tele2.ru\"",
    "AT+SAPBR=3,1,\"USER\",\"tele2\"",
    "AT+SAPBR=3,1,\"PWD\",\"tele2\"",
    "AT+SAPBR=1,1",  //Встановлюємо GPRS з'єднання
    "AT+HTTPINIT",  //Ініціалізація http сервісу
    "AT+HTTPPARA=\"CID\",1"  //Установка CID параметра
для http сесії
  };
  int ATsDelays[] = {6, 1, 1, 1, 3, 3, 1}; //массив
затримок
  Serial.println("GPRG init start");
  for (int i = 0; i < ATsCount; i++) {
```

```
    Serial.println(ATs[i]);  //посилаємо в монітор порту
    GSMport.println(ATs[i]);  //посилаємо в GSM модуль
    delay(d * ATsDelays[i]);
    Serial.println(ReadGSM());  //показуємо відповідь від
GSM модуля
    delay(d);
  }
  Serial.println("GPRG init complete");
}

void gprs_send(String data) { //Процедура відправки даних
на сервер
  //Передавання даних на сайт
  int d = 400;
  Serial.println("Send start");
  Serial.println("setup url");
  GSMport.println("AT+HTTPPARA=\"URL\",\"http://mysite.ru
/?a=" + data + "\"");
  delay(d * 2);
  Serial.println(ReadGSM());
  delay(d);
  Serial.println("GET url");
  GSMport.println("AT+HTTPACTION=0");
  delay(d * 2);
  Serial.println(ReadGSM());
  delay(d);
  Serial.println("Send done");
}
String ReadGSM() { //функція читання даних від GSM модуля
  int c;
  String v;
  while (GSMport.available()) { //зберігаємо вхідні рядок
в змінну v
    c = GSMport.read();
    v += char(c);
    delay(10);
  }
  return v;
}
```

PHP скрипт (для сервера)

Краще використовувати SQL базу даних, але для прикладу будемо записувати в файл. Зберігаємо на сервері в файл index.php

```
<? Php

   $ Ip = $ _SERVER ['REMOTE_ADDR']; // отримуємо IP адресу клієнта

   $ Client = $ _SERVER ['HTTP_USER_AGENT']; // отримуємо ідентифікатор HTTP клієнта

   $ Today = date ("Y.m.d H: i: s"); // отримуємо поточні дату і час

   $ F = fopen ("log.csv", "a"); // відкриваємо файл для додавання даних

   $ Param = $ _REQUEST ['a']; // отримуємо значення відправленої змінної "a"

   fwrite ($ f, "$ today; $ ip; $ client; sensor = $ param \ r \ n ---------- \ r \ n"); // запис даних в файл

   fclose ($ f); // закриваємо файл

?>

<P> GPRS data read page </ p>
```

В результаті на сервері повинен з'явитися файл з наступним змістом:
2015.09.02 13:26:51; 176.59.197.176; SIMCOM_MODULE; sensor=0

2015.09.02 13:27:14; 176.59.199.8; SIMCOM_MODULE; sensor=1

Практичне завдання до лабораторної роботи

Реалізувати найпростішу GSM сигналізацію з використанням датчика руху, датчика температури і модуля SIM800.

Добавте відправку SMS з даними температури на віддаленому об’єкті при виході її за встановлені межі.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, плату модуля SIM800, датчики руху і температури і провідники та складіть схему згідно рисунка.
2. Скласти та прошити в плату Arduino просту програму яка здійснює вихідний дзвінок на Ваш телефон.
3. Скласти та прошити в плату Arduino просту програму яка приймає вхідний дзвінок на Ваш телефон.
4. Скласти та прошити в плату Arduino програму яка реалізує найпростішу GSM сигналізацію з використанням датчика руху і модуля SIM800. При спрацюванні датчика, якщо стан на сигналізації виявив тривожний дзвінок на заданий номер телефону. Постановка зняття сигналізації вхідним дзвінком з визначених номерів. Стан на сигналізації індикується світлодіодом.
5. Скласти та прошити в плату Arduino програму яка реалізує найпростішу GSM сигналізацію з використанням датчика тамператури. При виході температури за межі встановленого діапазону відправляється SMS на заданий номер телефону. При вхідному дзвінку на даний номер відправляється SMS з інформацією про поточну температуру на віддаленому обєкті.
6. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.
3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Які параметри живлення потрібно забезпечити GSM модулю SIM800?
2. Які варіанти сжеми узгодження рівнів сигналів UART між мікроконтролером і GSM модулем SIM800 ви можете запропонувати ?
3. Що таке DTMF команди?
4. Що таке AT-команди?
5. Які недоліки GSM сингналізацій?

Лабораторна робота № 15.

Отримання точного часу та координат з GPS сопутників з використанням GSM модуля NEO-6M.

Мета роботи: Навчитися опрацьовувати сигнали GPS приймача, отримати практичні навички роботи з протоколом NMEA.

Теоретичні відомості

Для обчислення відстані від супутника до антени приймача вимірюється час проходження сигналу, а потім він помножується на швидкість розповсюдження сигналу.

Для визначення часу всі GPS супутники оснащені надточними атомними годинниками, чиї показання кодуються у сигнали, які передаються зі супутника. GPS-приймачі також оснащені високоточними годинниками, які у процесі ініціалізації на початку вимірювань синхронізуються з годинниками GPS супутників.

Для визначення координат антени приймача використовуються вимірювання відстаней мінімум до 4 супутників (четвертий супутник потрібен для виключення похибки, яка може бути викликана несинхронністю роботи годинників GPS супутників). Кожен GPS супутник передає радіосигнали на двох частотах: L1=1575.42 МГц і L2=1227.60 МГц. Сигнали, які використовуються для цивільних цілей, модулюються двома кодами: C/A і P. «Грубий» C/A-код (цивільний) не закодований і може прийматися всіма GPS-приймачами. «Точний» P-код приймається геодезичними і ГІС приймачами і може бути закодований під час військових дій.

Як правило, GPS не працює в будинку. Щоб зловити сигнал від супутників, з модулем потрібно вийти на вулицю або на балкон, або хоча б покласти його поруч з вікном. Крім того, GPS може погано ловити в хмарну погоду. Врахуйте також, що навіть на відкритому повітрі і в гарну погоду при першому включенні модуля може знадобитися кілька хвилин на ініціалізацію.

Існує чимало GPS-модулів, але в основному вони досить недешеві. Якщо Вам у вашому проекті з якихось причин потрібен GPS, найдешевшим в більшості випадків достатньо буде простого і недорогого модуля на базі чіпа NEO-6M. Спробуємо навчитися працювати з даним модулем.

GPS-модуль GPS NEO-6M пропонує фірма UBLOX. Даний модуль оснащений власною активною антеною з високим коефіцієнтом підсилення та пам’яттю EEPROM для зберігання інформації про конфігурацію, що суттєво

спрощує поставлену задачу. Також присутня перезаряджаєма резервна батарея, яка зберігає дані після вимкнення живлення і забезпечує гарячий старт.

Ця система виготовляється швейцарською компанією u-blox, заснованою в 1997 році. Лінійка модулів Neo-6 представлена різновидами G, Q, M, P, V і T, кожен з яких володіє своїми характерними можливостями.

Рис. 15.1. Вигляд GPS-модуля на базі чіпа NEO-6M.

Для використання GPS-модуля спочатку потрібно підключити його до живлення і контролера. Після того, як модуль виявить супутник, повинен загорітися, а через деякий час почати мерехтіти червоний світлодіод. Після цього в контролер потрібно записати бібліотеку TinyGPS і програму для роботи з модулем. Модуль видає результати у форматі NMEA. Для роботи з даними, які передає модуль і для настройки можна використовувати програму від виробника U-Center.

Для закріплення модуля на плоскій поверхні на корпусі є чотири отвори.

Призначення контактів:

- VCC (напруга живлення);
- RX (вхідні дані);
- TX (вихідні дані);
- GND (загальний контакт).

Приймач Neo-6M має такі властивості:

- час холодного або теплого старту - 27 с;

- час гарячого старту - 21 с;
- максимальна частота видачі інформації - 1 Гц;
- діапазон частот імпульсів на пін PPS - 0,25 Гц - 1 кГц;
- максимальна точність визначення положення - 2,5 м;
- максимальна точність визначення швидкості - 0,1 м / с;
- максимальна точність визначення курсу - 0,5 градуса.

Специфікація:

- TTL рівні;
- швидкість за замовчуванням 9600, конфігурація через u-center.

Neo-6M вміє використовувати SBAS (Satellite Based Augmentation System) - супутникові системи диференціальної корекції, що збільшує точність визначення положення до 2 м, а також AGPS (Assisted GPS) для зниження часу холодного старту. Отримання даних AGPS відбувається з сайту u-blox за допомогою сервісів AssistNow Online і AssistNow Offline (довгостроковий альманах). Модуль має підтримку протоколів NMEA, UBX і RTCM. UBX - пропріетарний протокол від u-blox, а RTCM - протокол для передачі модулю даних про диференціальну корекцію DGPS. Також для зв'язку доступні інтерфейси UART, I2C, SPI і USB.

У плані підключення все достатньо просто: лінії VCC, GND, RX і TX на приймачі підключаємо до + 3.3V, GND, PA9 і PA10 на STM32F378 відповідно.

Протокол зв’язку NMEA-0183

NMEA 0183 — комбінований стандарт, який визначає як електричні так і функціональні характеристики комунікаційного протоколу для обміну інформацією між навігаційними пристроями, GPS-приймачами та багатьма іншими типами обладнання.

На канальному рівні використовується конфігурація асинхронного послідовного порту «4800, 8N1», тобто: типова бітова швидкість - 4800, бітів даних – 8, біт парності – немає, стопових бітів – 1, узгодження обміну – немє.

Різновид стандарту під назвою NMEA-0183HS визначає бітову швидкість 38400. Цей варіант зазвичай використовується в автоматичних ідентифікаційних системах.

Прикладний рівень. Всі дані представляються текстовими символами ASCII між 0x20 (знак пропуску) до 0x7e (~).

Максимальна довжина повідомлення 82 символи, що включає початковий символ $ або ! і закінчення <LF>

Почаковим символом для кожного повідомлення може бути або $ (для традиційних повідомлень, що мають розділювачі полів) або ! (для повідомлень що мають спеціальну енкапсуляцію в середині)

Наступні п'ять символів ідентифікують джерело інформації (два символи) та тип повідомлення (три символи).

Всі наступні поля даних розділено символами коми ,.

Якщо дані недоступні, відповідне поле залишається порожнім — відсутні символи між двома розділовими знаками (див. приклад повідомлень).

Наступним символом, що слідує одразу після останнього поля є зірочка, але воно додається лише якщо присутня чексума.

Безпосередньо за зірочкою йде контрольна сума, представлення двозначним шістнадцятковим числом. Контрольна сума обчислюється як побітове виключне АБО ASCII-кодів всіх символів повідомлення між $ та * (не включаючи ці два символи). Згідно офіційної специфікації, контрольна сума не обов'язкова для більшості повідомлень, але вона вимагається, зокрема, для RMA, RMB та RMC.

Закінчують повідомлення символи <CR><LF>.

Приклад повідомлення, що інформує про прибуття на точку маршруту має наступний вигляд:

$GPAAM,A,A,0.10,N,WPTNME*32

де:

GP	Ідентифікатор джерела (Talker ID — GP для GPS-приймача, GL для GLONASS)
AAM	Сигнал прибуття
A	Входження у коло прибуття
A	Перпендикуляр пройдено
0.10	Радіус кола
N	Морських миль
WPTNME	Назва точки прибуття
*32	Контрольна сума

NMEA протокол описує великий список різних повідомлень, з яких можна виділити

GGA - інформація про фіксований розв’язок.

Найпопулярніше і найбільш використовуваний NMEA повідомлення з інформацією про поточний фіксований розвязок - горизонтальні координати, значення висоти, кількість використовуваних супутників і тип рішення.

$ GPGGA, 123519,4807.038, N, 01131.000, E, 1,08,0.9,545.4, M, 46.9, M ,, * 47

де:

GGA - NMEA Заголовок

123519 -UTC час 12:35:19

4807.038, N - Широта, 48 градусів 7.038 хвилини північної широти

01131.000, E - Довгота, 11 градусів 31.000 хвилини східної довготи

1 - тип рішення, StandAlone рішення

0 - немає рішення,

1 - StandAlone,

2 - DGPS,

3 - PPS,

4 - фіксований RTK,

5 - не фіксований RTK,

6 - використання даних інерційних систем,

7 - ручний режим,

8 - режим симуляції

08 - кількість використовуваних супутників

0.9 - геометричний фактор, HDOP

545.4, M - висота над рівнем моря в метрах

46.9, M - висота геоїда над еліпсоїдом WGS 84

[Порожнє поле] - час що минув з моменту отримання останньої DGPS поправки. Заповнюється при активізації DGPS режиму

[Порожнє поле] - ідентифікаційний номер базової станції. Заповнюється при активізації DGPS режиму.

GSA - загальна інформація про супутники.

Це NMEA повідомлення містить список супутників, які використовуються в підрахунку позиції і значення геометричних факторів DOPs, що визначають точність підрахунку позиції. Параметри DOP визначаються геометричним розташуванням супутників на небі. Чим краще «розподілені» на небі супутники, тим менше DOP і тим краще точність позиції. Мінімальне значення PDOP (= 1) відповідає ситуації, коли один супутник знаходиться строго над користувачем, а інші 3 рівномірно розподілені навколо на рівні горизонту. Значення PDOP обчислюється, як квадратний корінь з суми квадратів HDOP і VDOP.

$ GPGSA, A, 3,04,05,, 09,12 ,,, 24 ,,,,, 2.5,1.3,2.1 * 39

де:

GSA - NMEA заголовок

A - тип вибору між 2D і 3D рішеннями, Автоматичний (A-auto, M-manual)

3 - тип рішення, 3D рішення (1 - немає рішення, 2 - 2D рішення, 3 - 3D рішення)

04,05 ... - PRN коди використовуються в підрахунку позиції супутників (12 полів)

2.5 - просторовий геометричний фактор, PDOP

1.3 - горизонтальний геометричний фактор, HDOP

2.1 - вертикальний геометричний фактор, VDOP

GSV - Детальна інформація про супутники

Це NMEA повідомлення містить детальну інформацію для всіх відслідковуються навігатором GPS супутників. Виходячи з обмеження в 80 символів в складі одного NMEA повідомлення можуть передавати дані тільки для 4-х супутників. Відповідно для 12 супутників потрібно 3 повідомлення GSV. Поле SNR (Signal to Noise Ration) містить значення рівнів, прийнятих з супутників, навігаційних сигналів. Теоретично його значення може варіюватися від 0 до 99 і вимірюється в dB. Фактично рівень сигналу лежить в діапазоні 25 ... 35 dB. Тут варто зазначити, що даний параметр не є абсолютним і не підходить для порівняння чутливості приймачів різних моделей і виробників. В GPS навігаторах можуть використовуватися різні алгоритми обчислення рівня сигналу, що призводить до різних результатів при рівній мірі чутливості приймачів. Для кожного видимого GPS супутника передається набір інформації, що включає рівень сигналу, кут піднесення та азимут супутника. Кількість цих «наборів» визначається загальному кількістю видимих супутників, значення якого передається в окремому полі.

$ GPGSV, 2,1,08,01,40,083,46,02,17,308,41,12,07,344,39,14,22,228,45 * 75

де:

GSV - NMEA заголовок

2 - кількість повідомлень GSV в пакеті

1 - номер повідомлення в пакеті (від 1 до 3)

08 - кількість видимих супутників

01 - номер супутника

40 - кут піднесення, в градусах

083 - азимут в градусах

46 - SNR, рівень сигналу

RMC - рекомендований мінімальний набір GPS даних

Це NMEA повідомлення містить всі набори, так званих «PVT» даних. «PVT» - загальноприйняте скорочення від «position, velocity, time» (позиція, швидкість, час).

$ GPRMC, 123519, A, 4807.038, N, 01131.000, E, 022.4,084.4,230394,003.1, W * 6A

де:

RMC - NMEA заголовок

123419 - UTC час, 12:34:59

A - статус (A- активний, V- ігнорувати)

4807.038, N - Широта, 48 градусів 07.038 хвилин північної широти

01131.000, E - Довгота, 11 градусів 31.000 хвилина східної довготи

022.4 - Швидкість, в вузлах

084.4 - Напрямок руху, в градусах

230394 - Дата, 23 березня 1994 року

003.1, W - Магнітні варіації

GLL - дані широти і довготи

NMEA повідомлення зі значенням координат широти і довготи, і часу коли було обчислено це рішення.

$ GPGLL, 4916.45, N, 12311.12, W, 225444, A, * 31

де:

GLL - NMEA заголовок

4916.46, N - широта, 49 градусів 16.45 хвилин північної широти

12311.12, W-довгота, 123 градуси 11.12 хвилин західної довготи

225444 - Час фіксації в шкалі часу UTC, 22:54:44

A - Тип даних, (A - активні, V - ігнорувати)

BOD - Азимут на пункт призначення

Це NMEA повідомлення вказує азимут на точку призначення в режимі навігації.

$ GPBOD, 045., T, 023., M, DEST, ST

Новіший стандарт, NMEA 2000, передбачає вищу бітову швидкість та декілька «промовців» (talkers) без використання центрального хаба чи кільцевої буферизації пакетів.

До Arduino модуль підключається не складно, так як його напруга живлення складає 5 В, а дані передає по UART на швидкості 9600 бод. Коли модуль знаходить достатню кількість супутників для визначення свого місця розташування, він починає блимати синім світлодіодом і передавати по UART раз в секунду щось таке:

$GPRMC,160709.00,A,5546.7229,N,03749.47557,E,2.578,228.26,170617,,,A*69

$GPVTG,228.26,T,,M,2.578,N,4.774,K,A*39

$GPGGA,160709.00,5546.792,N,03749.475,E,1,07,4.00,172.3,M,13.3,M,,*5C

$GPGSA,A,3,13,20,18,30,21,15,,,,,,6.45,3.95,5.11*02

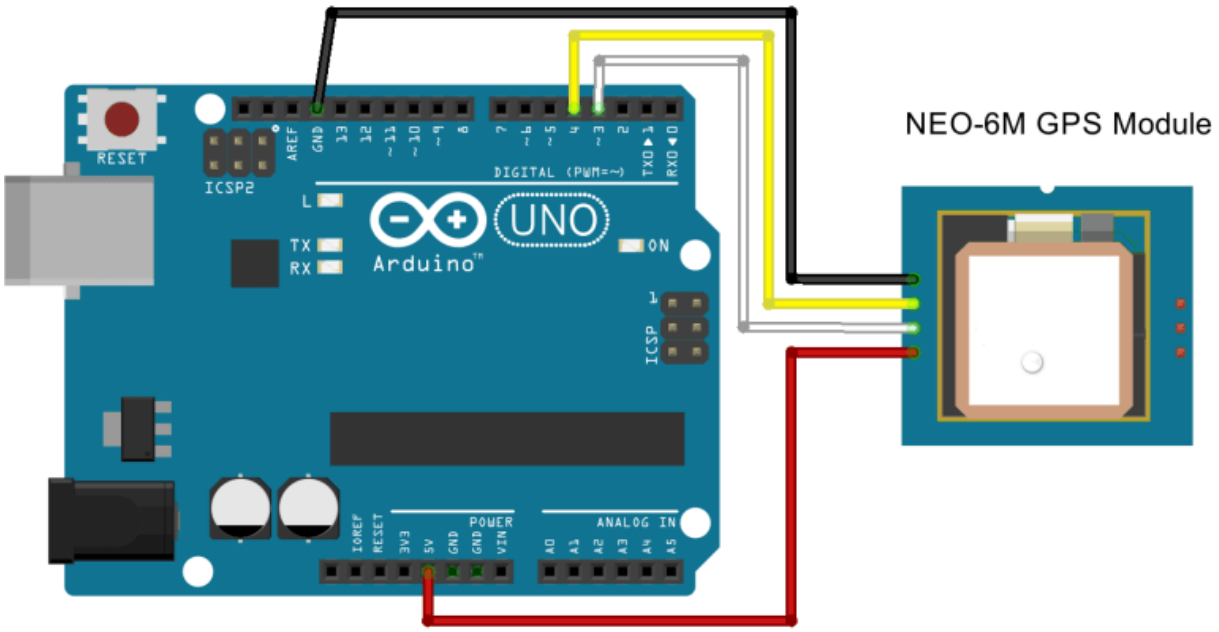


Рис. 15.1. Схема під'єднання модуля GPS NEO-6M до Arduino Uno.

Тут нескладно знайти поточні координати (5546.79229, N і 03749.47557, E), а також дату (170617) та час в UTC (160709). Цифри в кінці рядків після зірочки - це контрольні суми. Варто відзначити, що якщо модуль ще не визначив свої GPS-координати, він передає по UART трохи інші дані. Детальний опис того, як декодувати всі ці дані можна знайти в документі під назвою U-blox 6 Receiver Description Including Protocol Specification [PDF].

Для роботи з GPS модулем скористаємся бібліотекою TinyGPSPlus. Код прошивки:

```
#include <SoftwareSerial.h>
    #include <TinyGPS.h> // підключення необхідних для роботи бібліотек
    TinyGPS gps;
    SoftwareSerial gpsSerial (8, 9); // номера виводів, до яких підключений модуль (RX, TX)
    bool newdata = false;
    unsigned long start;
    long lat, lon;
    unsigned long time, date;
    void setup () {
    gpsSerial.begin (9600); // установка швидкості обміну з приймачем
    Serial.begin (9600);
    Serial.println ( "Waiting data of GPS ...");
    }
    void loop () {
```

```
    if (millis () - start> тисячі) // установка затримки
в одну секунду між оновленням даних
    {
    newdata = readgps ();
    if (newdata)
    {
    start = millis ();
    gps.get_position (& lat, & lon);
    gps.get_datetime (& date, & time);
    Serial.print ( "Lat:"); Serial.print (lat);
    Serial.print ( "Long:"); Serial.print (lon);
    Serial.print ( "Date:"); Serial.print (date);
    Serial.print ( "Time:"); Serial.println (time);
    }}
    }
    // перевірка наявності даних
    bool readgps ()
    {
    while (gpsSerial.available ())
    {
    int b = gpsSerial.read ();
    // в бібліотеці TinyGPS є помилка: не обробляються
дані з \ r і \ n
    if ( '\ r'! = b)
    {
    if (gps.encode (b))
     return true;
    }
    }
    return false;
    }
```

Варіант коду з виводом на LCD дисплкй

```
#include <Arduino.h>
#include <SoftwareSerial.h>
#include "LiquidCrystal_I2C.h"
#include "TinyGPS++.h"

const int DELAY = 100;
const int SWITCH_TIME = 5000;

LiquidCrystal_I2C lcd(0x3F, 16, 2);
SoftwareSerial gps_serial(A3, A2); /* RX, TX */
TinyGPSPlus gps_parser;

void setup()
```

```
{
    Serial.begin(9600);
    gps_serial.begin(9600);
    lcd.begin();
}

String twoDigits(int x) {
  if(x < 10) return "0" + String(x);
  else return String(x);
}

int currentModeTime = 0;
bool showLocation = false;

void loop() {
    while(gps_serial.available() > 0) {
        char temp = gps_serial.read();
        Serial.write(temp);
        gps_parser.encode(temp);
    }

    String lat  = "Unknown         ";
    String lng  = "location        ";
    if(gps_parser.location.isValid()) {
        lat = "Lat: " + String(gps_parser.location.lat(),
6);
        lng = "Lng: " + String(gps_parser.location.lng(),
6);
    }

    String date = "Unknown date    ";
    if(gps_parser.date.isValid()) {
        date = twoDigits(gps_parser.date.day()) + "/" +
               twoDigits(gps_parser.date.month()) + "/" +
               String(gps_parser.date.year()) + "      ";
    }

    String time = "Unknown time    ";
    if(gps_parser.time.isValid()) {
        time = twoDigits(gps_parser.time.hour()) + ":" +
               twoDigits(gps_parser.time.minute())  +  ":"
+
               twoDigits(gps_parser.time.second())   +   "
UTC    ";
    }
```

```
    if(showLocation) {
        lcd.setCursor(0, 0);
        lcd.print(lat);
        lcd.setCursor(0, 1);
        lcd.print(lng);
    } else { // show date and time
        lcd.setCursor(0, 0);
        lcd.print(date);
        lcd.setCursor(0, 1);
        lcd.print(time);
    }

    delay(DELAY);
    currentModeTime += DELAY;
    if(currentModeTime >= SWITCH_TIME) {
        lcd.clear();
        showLocation = !showLocation;
        currentModeTime = 0;
    }
}
```

Практичне завдання до лабораторної роботи

Реалізувати найпростіший GPS логер, який буде передавати дані про координати, час, висоту та швидкість руху в комп'ютер через UART.

Додаткове завдання. Добавте відправку аналогічних даних через GSM на віддалений сервер.

Порядок виконання роботи

1. Підготувати макетну плату Arduino UNO, плату модуля NEO-6M, провідники та складіть схему згідно рисунка 15.1.
2. Скласти та прошити в плату Arduino просту програму виводу в порт поточного часу та GPS координат.
3. Модифікуйте програму так щоб вона виводила дані про поточну швидкість, напрям руху та висоту.
4. Скласти та прошити в плату Arduino програму яка виводитиме перелік супутників та якість GPS сигналу.
5. Оформити звіт.

Зміст звіту

1. Титульна сторінка.
2. Назва та мета лабораторної роботи.

3. Завдання до лабораторної роботи.
4. Навести електричну схему реалізації системи згідно поставленого завдання.
5. Навести програмний код реалізованої системи.
6. Навести результати роботи системи.
7. Висновки.

Контрольні запитання

1. Які параметри живлення потрібно забезпечити GPS модулю NEO-6M
2. Що таке формат даних NMEA?
3. Що таке довгота, широта, азимут.
4. Які дані можна отримати з GPS супутників?
5. Які є види GPS навігації.

Цитована література:

1. Основи мікропроцесорної техніки: лабораторний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 171 «Електроніка» / В.С.Баран, Г.Г.Власюк, Ю.О.Оникієнко, О.І.Смоленська ; КПІ ім. Ігоря Сікорського. — Київ : КПІ ім. Ігоря Сікорського, 2019. –140 с.
2. RFID-модуль RC522 [Електронний ресурс] – Режим доступу: http://makerplus.ru/wiki/rfid-modul-rc522.
3. NEO-6 u-blox 6 GPS Modules Data Sheet [Електронний ресурс] – Режим доступу: https://www.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf
4. Модуль LCD1602 [Електронний ресурс] – Режим доступу: https://arduinomaster.ru/datchiki-arduino/lcd-i2c-arduino-displey-ekran/
5. Модулі DHT11 та DHT22 [Електронний ресурс] – Режим доступу: http://arduino-diy.com/arduino-datchiki-temperatury-i-vlazhnosti-DHT11-i-DHT22
6. Модуль HC-SR04 [Електронний ресурс] – Режим доступу: http://wiki.amperka.ru/%D0%BF%D1%80%D0%BE%D0%B4%D1%83%D0%BA%D1%82%D1%8B:hc-sr04-ultrasonic-sensor-distance-module
7. Модуль FC-37 [Електронний ресурс] – Режим доступу: http://www.arduino-ic.ru/products/arduino/319/27796/

Навчальне видання

Лабораторний практикм

з дисциплін «Технології проектування комп'ютерних систем», «Дослідження і програмування пристроїв зв'язку з об'єктом», "Компютерна схемотехніка"

навчально-методичний посібник для студентів спеціальності «Комп'ютерна інженерія», «Електроніка»

В авторській редакції
Формат 60x84/16. Ум. друк. арк. 7.24
Папір офсетний. Друк цифровий.
Гарнітура Times New Roman
Наклад 100 прим.

Івано-Франківськ,
Височана, 18, Видавництво «НАІР»
(0342) 50-57-82, 0504336793

Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців, виробників і розповсюджувачів видавничої продукції №4191 від 12.11.2011р.