МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
Факультет інформаційних технологій

М.А.ХОДУКІН

НАВЧАЛЬНО-МЕТОДИЧНИЙ ПОСІБНИК

з освітньої компоненти «Архітектура комп'ютера та вбудовані мікропроцесорні системи з використанням Arduino»

КРИВИЙ РІГ 2023

Рецензенти:
І.Н. Вдовиченко, к.т.н., доцент кафедри комп’ютерних систем та мереж Криворізького національного університету;

Д.Г. Медведєв, к.т.н., доцент кафедри інформатики і прикладного програмного забезпечення Державного університету економіки і технологій.

Рекомендовано до друку навчально-методичною радою Державного університету економіки і технологій протокол № 9 від 22.02.2023р.

М.А. Ходукін: Навчально-методичний посібник з освітньої компоненти «Архітектура комп'ютера та вбудовані мікропроцесорні системи з використанням Arduino» (частина 2) для студентів першого (бакалаврського) рівня вищої освіти спеціальності 121 «Інженерія програмного забезпечення» денної та заочної форм навчання. – Кривий Ріг: Державний університет економіки і технологій, 2023. –102 с.

Навчально-методичний посібник призначений для студентів першого (бакалаврського) рівня вищої освіти спеціальності 121 «Інженерія програмного забезпечення» денної та заочної форм навчання. НМ посібник може бути корисним для студентів першого (бакалаврського) рівня вищої освіти спеціальності 122 «Комп'ютерні науки».

У НМП подано основний теоретичний матеріал з принципами роботи у програмно-апаратному комплексі Arduino. Розглянуті основні функції, які використовуються для керування периферією, форматування даних та обміну даними. В результаті виконання лабораторних робіт здобувачі вищої освіти зможуть писати и налагоджувати програми у середовищі розробки Arduino.

Навчально-методичний посібник розроблено у відповідності до навчального плану з метою надання здобувачам вищої освіти допомоги у освоєнні теоретичного матеріалу та при виконанні ними лабораторних та індивідуальних завдань.

ЗМІСТ

1. ЗАГАЛЬНІ ВІДОМОСТІ

Дуже довгій час, для того щоб зібрати пристрій на мікроконтролері, необхідно було знати основи схемотехніки, будову та роботу конкретного процесора, вміти програмувати та виготовляти електронну техніку. Потрібні також програматори, налагоджувачі та інші допоміжні пристрої. Без величезного обсягу знань та дорогого обладнання не було важко обійтися. Зараз, з появою пристроїв, що дають можливість працювати з мікроконтролерами без серйозної матеріальної бази та знання багатьох предметів, все змінилося. Прикладом такого пристрою може бути проект Arduino італійських розробників.

1.1 Програмно-апаратний комплекс Arduino

Arduino та його клони є наборами, що складаються з готового електронного блоку та програмного забезпечення. Електронний блок тут — це друкована плата із встановленим мікроконтролером та мінімумом елементів, необхідних для його роботи. Фактично, електронний блок Arduino є аналогом материнської плати сучасного комп'ютера. На ньому є роз'єми для підключення зовнішніх пристроїв, а також роз'єм для зв'язку з комп'ютером, яким здійснюється програмування мікроконтролера. Особливості використовуваних мікроконтролерів ATmega фірми Atmel дозволяють виконувати програмування без застосування спеціальних програматорів. Все, що потрібно для створення нового електронного пристрою - це плата Arduino, кабель зв'язку і комп'ютер. Другою частиною платформи Arduino є програмне забезпечення для створення програм, що керують. Воно об'єднало в собі найпростіше середовище розробки та мову програмування, що є варіантом мови C/C++ для мікроконтролерів. До нього додані елементи, що дозволяють створювати програми без вивчення апаратної частини. Так що для роботи з Arduino практично достатньо знання лише основ програмування на C/C++. Створено в середовищі Arduino і безліч бібліотек, що містять код, що працює із різними пристроями.

Інакше кажучи, **Arduino** це – апаратно-програмна платформа, основними компонентами якої є мікроконтролерна плата введення/виведення та середовище розробки на мові «*Arduino C*» (*C ++* з фреймворком *Wiring*). Arduino, як і інші аналогічні засоби, має на меті звільнити користувача від необхідності заглиблюватися в деталі внутрішнього устрою мікроконтролерів, надавши йому простий і зручний інтерфейс для їх програмування.

Arduino на відміну від інших систем надає ряд переваг:

1. *Просте і зручне середовище програмування*. Середовище програмування Arduino зрозуміле і просте для початківців, але при цьому досить гнучке для просунутих користувачів. Завдяки цьому, студенти, які вивчають програмування, зможуть легко освоїти Arduino.
2. *Відкрите апаратне забезпечення.* Пристрої Arduino побудовані на базі мікроконтролерів Atmel ATmega328 і ATmega168. Завдяки тому, що всі схеми модулів Arduino опубліковані під ліцензією Creative Commons, досвідчені інженери і розробники можуть створювати свої версії пристроїв

на основі існуючих. І навіть звичайні користувачі можуть збирати дослідні зразки Arduino для кращого розуміння принципів їх роботи і економії коштів.

3. *Розширюване програмне забезпечення з відкритим вихідним кодом.* Програмне забезпечення Arduino має відкритий вихідний код, завдяки цьому досвідчені програмісти можуть змінювати і доповнювати його. Можливості мови Arduino можна також розширювати за допомогою C++ бібліотек. Завдяки тому, що він заснований на мові AVR-C, просунуті користувачі, що бажають розібратися в технічних деталях, можуть легко перейти з мови Arduino на C або вставляти AVR-C код безпосередньо в програми Arduino.
4. *Кросплатформеність*. Програмне забезпечення Arduino працює на операційних системах Windows, Macintosh OSX і Linux, в той час, як більшість подібних систем орієнтовані на роботу тільки в Windows.
5. *Низька вартість*. У порівнянні зі схожими апаратними платформами, плати Arduino мають відносно низьку вартість.

1.2 Плати Arduino

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

Контролери *Arduino Uno, Arduino Leonardo, Arduino Pro* - пристрої на основі 8-розрядного мікроконтролера.

Arduino Due – це пристрій на основі мікропроцесора Atmel SAM3X8E ARM Cortex-M3. Це перша плата Arduino на базі 32-розрядного мікроконтролера ARM. Завдяки використанню 32-розрядної ядра ARM, Arduino Due багато в чому перевершує типові плати на базі 8-розрядних мікроконтролерів. Найбільш суттєві відмінності полягають в наступному:

- 32-бітове ядро дозволяє обробляти 4х-байтові дані всього за один такт. Тактова частота - 84 МГц.
- Обсяг оперативної пам'яті SRAM складає 96 КБ.
- Обсяг флеш-пам'яті програм - 512 КБ.

Наявність DMA-контролера, що дозволяє розвантажити центральний процесор від виконання ресурсномістких операцій з пам'яттю.

Arduino YUN – це контролер із вбудованим Wi-Fi модулем під управлінням ОС Linux і системою команд Arduino. Arduino YUN є комбінацією класичного Arduino Leonardo (на базі мікроконтролера ATmega32U4) і Wi-Fi системи на кристалі, що працює під управлінням Linux (дистрибутив ОС GNU/Linux на основі OpenWRT для мікропроцесорів MIPS).

Arduino Robot – перша офіційна версія Arduino, в конструкції якого передбачено колеса. Робот складається з двох плат, кожна з яких містить свій мікропроцесор. Плата приводів (Motor Board) контролює роботу двигунів, в той час, як керуюча плата (Control Board) зчитує показання датчиків і приймає

рішення про подальші операції. Кожна з двох плат є повноцінним пристроєм Arduino, програмованим за допомогою середовища розробки Arduino IDE.

Arduino Esplora – це мікропроцесорний пристрій, спроектований на основі Arduino Leonardo. Esplora відрізняється від усіх попередніх плат Arduino наявністю безлічі вбудованих, готових до використання датчиків для взаємодії. Esplora має вбудовані звукові і світлові індикатори (для виведення інформації), а також кілька датчиків (для введення інформації), таких, як джойстик, слайдер, датчик температури, акселерометр, мікрофон і світловий датчик.

Arduino ADK – це пристрій на основі мікроконтролера ATmega2560. У ньому реалізований USB-хост для підключення смартфонів на базі операційної системи Android.

Arduino Nano

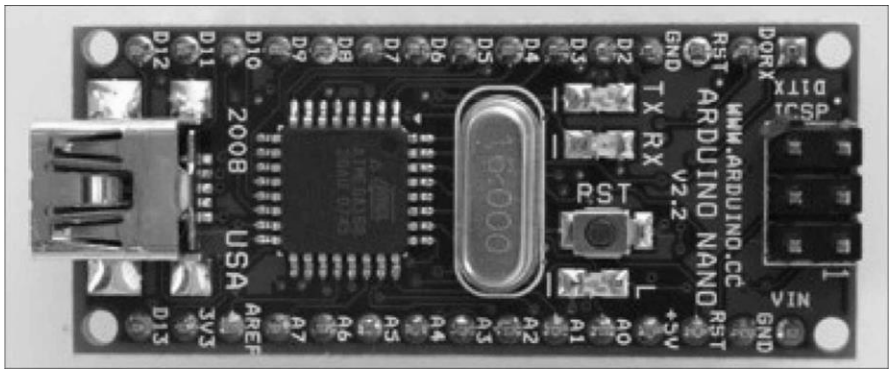


Рис. 1.1 Плата Arduino Nano

Arduino Nano – це повнофункціональний мініатюрний пристрій на базі мікроконтролера ATmega328 (Arduino Nano 3.0) або ATmega168 (Arduino Nano 2.x), адаптований для використання з макетної платі.

Характеристики

- Мікроконтролер: Atmel ATmega168 або ATmega328
- Робоча напруга (логічний рівень): 5В
- Напруга живлення (рекомендована): 7-12В
- Напруга живлення (гранична): 6-20В
- Цифрові входи/виходи: 14 (з яких 6 можуть використовуватися як ШІМ-виходи)
- Аналогові входи: 8
- Максимальний струм одного виведення: 40 мА
- Flash-пам'ять: 16 КБ (ATmega168) або 32 КБ (ATmega328) з яких 2 КБ використовуються завантажувачем
- SRAM: 1 КБ (ATmega168) або 2 КБ (ATmega328)
- EEPROM: 512 байт (ATmega168) або 1 КБ (ATmega328)
- Тактова частота: 16 МГц
- Розміри плати: 1.85 см x 4.3 см

Arduino Nano може живитися через кабель Mini-USB, від зовнішнього джерела живлення з не стабілізованою напругою 6-20В або зі стабілізованою напругою 5В. Пристрій автоматично вибирає джерело живлення з найбільшим

рівнем напруги.

Входи і виходи. З використанням функцій *pinMode (), digitalWrite ()* і *digitalRead ()* кожен з 14 цифрових виводів Arduino Nano може працювати в якості входу або виходу. Робоча напруга виводів - 5В. Максимальний струм, який може віддавати або споживати один вивід, становить 40 мА. Всі виводи пов'язані з внутрішніми підтягуючими резисторами (за замовчуванням відключеними) номіналом 20-50 кОм. 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.

Зовнішні переривання: виводи 2 і 3. Дані виводи можуть бути налаштовані в якості джерел переривань, що виникають при різних умовах: при низькому рівні сигналу, по фронту, по спаду або при зміні сигналу.

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

Інтерфейс SPI: виводи 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Дані виводи дозволяють здійснювати зв'язок по інтерфейсу SPI. У пристрої реалізована апаратна підтримка SPI.

Світлодіод: Вбудований світлодіод, приєднаний до виводу 13.

I^2C: виводи 4 (SDA) і 5 (SCL). З використанням бібліотеки Wire дані виводи можуть здійснювати зв'язок по інтерфейсу I^2C (TWI).

Крім перерахованих, на платі існує ще кілька виводів:

- *AREF*. Опорна напруга для аналогових входів. Може бути задіяна функцією analogReference ().
- *Reset*. Формування низького рівня (LOW) на цьому виводу призведе до перезавантаження мікроконтролера. Зазвичай цей вивід служить для функціонування кнопки скидання на платах розширення.

Arduino Uno

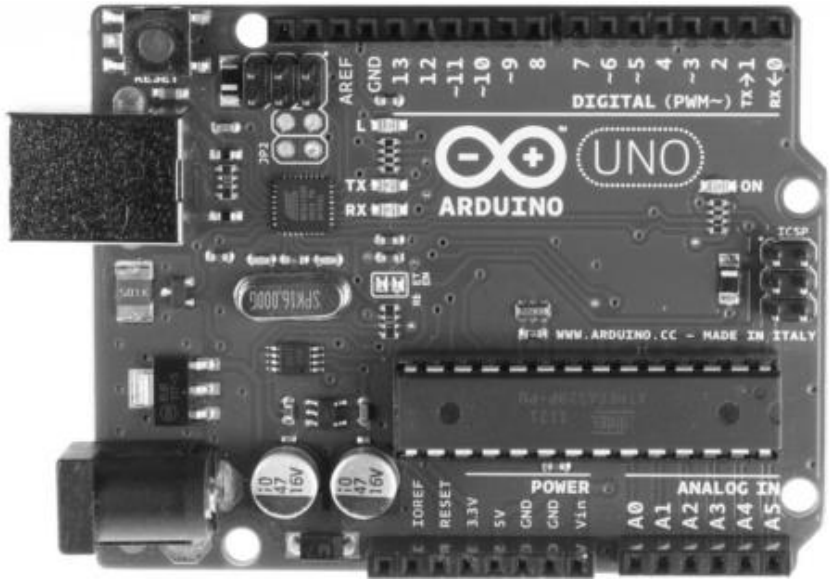


Рис. 1.2 Плата Arduino Uno

Arduino Uno - це пристрій на основі мікроконтролера ATmega328. До його складу входить все необхідне для зручної роботи з мікроконтролером.

Характеристики

- Мікроконтролер: ATmega328
- Робоча напруга: 5 В
- Напруга живлення (рекомендована): 7-12 В
- Напруга живлення (гранична): 6-20 В
- Цифрові входи/виходи: 14 (з них 6 можуть використовуватися в якості ШІМ-виходів)
- Аналогові входи: 6
- Максимальний струм одного виводу: 40 мА
- Максимальний вихідний струм виводу: 3.3V 50 мА
- Flash-пам'ять: 32 КБ (ATmega328) з яких 0.5 КБ використовується завантажувачем
- SRAM: 2 КБ (ATmega328)
- EEPROM: 1 КБ (ATmega328)
- Тактова частота: 16 МГц

Arduino Uno може живитися від USB або від зовнішнього джерела живлення - тип джерела вибирається автоматично.

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 контакт необхідно під'єднати до виводів Gnd і Vin роз'єму POWER.

Напруга зовнішнього джерела живлення може бути в межах від 6 до 20 В. Однак, зменшення напруги живлення нижче 7В призводить до зменшення напруги на виводі 5V, що може стати причиною нестабільної роботи пристрою. Використання напруги більше 12В може призводити до перегріву стабілізатора напруги і виходу плати з ладу. З огляду на це, рекомендується використовувати джерело живлення з напругою в діапазоні від 7 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 як подавати 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 – 12 В), від USB (5В) або через вивід Vin (7 – 12 В). 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. 3.3 В, 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 інформації про робочу напругу мікроконтролера Arduino. Залежно від напруги, зчитуваної з виводу IOREF, плата розширення може переключитися на відповідне джерело живлення або задіяти перетворювачі рівнів, що дозволить їй працювати як з 5В, так і з 3.3В пристроями.

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 ATmega8U2, яка виконує роль 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.[1]

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 13 піну. При подачі значення HIGH на цей пін, світлодіод включається, при відправці LOW – вимикається.

В Arduino Uno є 6 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 5 В. 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 виводу призведе до пере завантаження мікроконтролера. Зазвичай цей вивід служить для

[1] Для отримання додаткової інформації див. функцію *attachInterrupt ()*

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. Мікроконтролер ATmega16U2 на платі виконує функції такого перетворювача, і при підключенні до ПК, дозволяє Arduino включитися, як віртуальний COM- порт. Прошивка мікросхеми 16U2 використовує стандартні драйвера USB- COM, тому установка зовнішніх драйверів не потрібна. На платформі Windows необхідний тільки відповідний .inf-файл. У пакет програмного забезпечення Arduino входить спеціальна програма, що дозволяє зчитувати і відправляти на Arduino прості текстові дані. 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 Uno.

У мікроконтролері ATmega328 також реалізована підтримка послідовних інтерфейсів I^2C (TWI) і SPI. У програмне забезпечення Arduino входить бібліотека Wire, що дозволяє спростити роботу з шиною I^2C. Для роботи з інтерфейсом SPI використовують бібліотеку SPI.

Автоматичне (програмне) скидання

Щоб кожен раз перед завантаженням програми не було потрібно натискати кнопку скидання, Arduino Uno спроектований таким чином, що дозволяє здійснювати його скидання програмно, з підключеного комп'ютера. Один з виводів ATmega8U2/16U2, який бере участь в керуванні потоком даних (DTR), з'єднаний з виводом RESET мікроконтролера ATmega328 через конденсатор номіналом 100 нФ. Коли на лінії DTR з'являється нуль, вивід RESET також переходить в низький рівень на час, достатній для перезавантаження мікроконтролера. Дана особливість використовується для того, щоб можна було прошивати мікроконтролер всього одним натисненням кнопки в середовищі програмування Arduino. Така архітектура дозволяє зменшити тайм-аут завантажувача, оскільки процес прошивки завжди синхронізований зі спадом сигналу на лінії DTR.

Захист USB від перевантажень

В Arduino Uno є відновлювані запобіжники, що захищають USB-порт комп'ютера від коротких замикань і перевантажень. Незважаючи на те, що більшість комп'ютерів мають власний захист, такі запобіжники забезпечують додатковий рівень захисту. Якщо USB-портом споживається струм більше 500 мА, запобіжник автоматично розірве з'єднання до усунення причин короткого замикання або перевантаження.

Arduino Mega

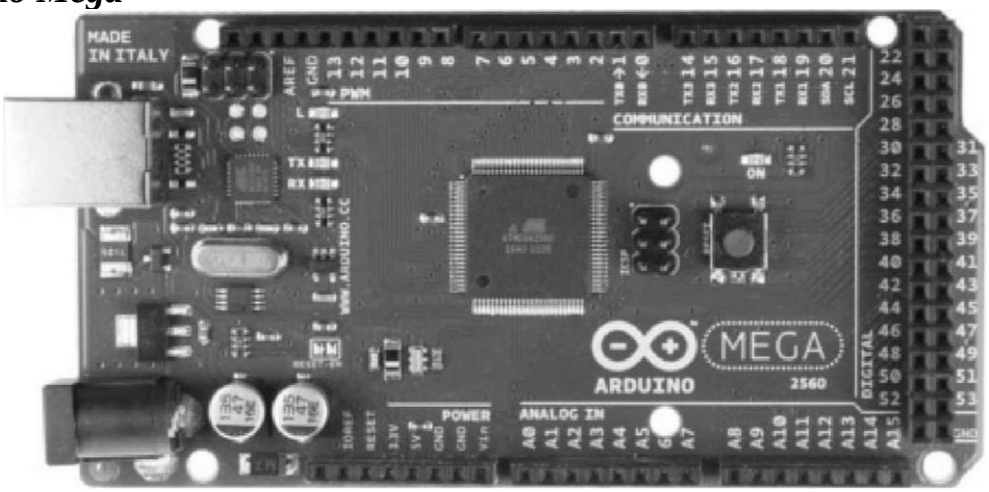


Рис. 1.3 Плата Arduino Mega

Плата Arduino Mega 2560 призначена для створення проектів, в яких не вистачає можливостей звичайних Arduino Uno. У цьому пристрої максимальна з усіх плат сімейства Arduino кількість виводів і розширений набір інтерфейсів. Також у Arduino Mega більше вбудованої пам'яті.

Плата має 54 цифрових входів/виходів (14 з яких можуть використовуватися як виходи ШІМ), 16 аналогових входів, 4 послідовних портів UART, кварцовий генератор 16 МГц, USB коннектор, роз'єм живлення, роз'єм ICSP і кнопка перезавантаження.

Характеристики

- Мікроконтролер: ATmega2560
- Робоча напруга: 5 В
- Вхідна напруга (рекомендована): 7-12 В
- Вхідна напруга (гранична): 6-20 В
- Цифрові Входи/Виходи: 54 (14 з яких можуть бути сконфігуровані як виходи ШІМ)
- Аналогові входи: 16
- Постійний струм через вхід/вихід: 40 мА
- Постійний струм для виводу 3.3 В: 50 мА
- Флеш-пам'ять: 256 КБ (з яких 8 КБ використовуються для завантажувача)
- ОЗУ: 8 КВ
- Незалежна пам'ять: 4 КБ
- Тактова частота: 16 МГц

Arduino Mega може отримувати живлення як через підключення по USB, так і від зовнішнього джерела живлення. Джерело живлення вибирається автоматично.

Зовнішнє живлення (не USB) може подаватися через перетворювач напруги AC/DC (блок живлення) або акумуляторною батареєю. Перетворювач напруги підключається за допомогою роз'єму 2.1 мм з позитивним полюсом на центральному контакті. Провід від батареї підключаються до виводів Gnd і Vin роз'єму живлення (POWER).

Платформа може працювати при зовнішньому живленні від 6 В до 20 В.

При напрузі живлення нижче 7 В, вивід 5 В може видавати менше 5 В, при цьому платформа може працювати нестабільно. При використанні напруги вище 12 В регулятор напруги може перегрітися і пошкодити плату. Рекомендований діапазон від 7 В до 12 В.

Для обміну даними по USB плата Mega2560 використовує мікроконтролер ATmega8U2, запрограмований як конвертер USB-UART.

Виводи живлення:

Vin. Вхід використовується для подачі живлення від зовнішнього джерела (за відсутності 5 В від роз'єму USB або іншого регульованого джерела живлення). Подача напруги живлення відбувається через даний вивід.

5V. Регульоване джерело напруги, що використовується для живлення мікроконтролера і компонентів на платі. Живлення може подаватися від виводу Vin через регулятор напруги, або від роз'єму USB, або іншого регульованого джерела напруги 5 В.

3V3. Напруга на виводі 3.3 В генерується мікросхемою FTDI на платформі. Максимальне споживання струму 50 мА.

GND. Виводи заземлення.

Пам'ять

Мікроконтролер ATmega2560 має: 256 КБ флеш-пам'яті для зберігання коду програми (4 КБ використовується для зберігання завантажувача), 8 КБ ОЗУ і 4 КБ EEPROM (яка читається і записується за допомогою бібліотеки EEPROM).

Входи і Виходи

Кожен з 54 цифрових пінів Mega, використовуючи функції *pinMode (), digitalWrite (), і digitalRead (),* може налаштовуватися як вхід або вихід. Виводи працюють при напрузі 5 В. Кожен вивід має підтягуючий резистор (стандартно відключений) 20-50 кОм і може пропускати струм до 40 мА. Деякі виводи мають особливі функції: ATmega2560 підтримує 4 порти послідовної передачі даних UART, а також інтерфейси I^2C (TWI) і SPI.

Послідовні порти: Послідовний порт 0 (RX) і 1 (TX); Послідовний порт 1: 19 (RX) і 18 (TX); Послідовний порт 2: 17 (RX) і 16 (TX); Послідовний порт 3: 15 (RX) і 14 (TX). Виводи використовуються для отримання (RX) і передачі (TX) даних TTL. Виводи 0 і 1 підключені до відповідних виводів мікросхеми послідовного порту ATmega8U2.

Зовнішнє переривання: 2 (переривання 0), 3 (переривання 1), 18 (переривання 5), 19 (переривання 4), 20 (переривання 3), і 21 (переривання 2). Дані виводи можуть бути налаштовані на виклик переривання або на молодшому значенні, або на передньому чи задньому фронті, або при зміні значення.

PWM: 2 до 13 і 44-46. Будь-який з виводів забезпечує ШІМ з роздільною здатністю 8 біт за допомогою функції *analogWrite ().*

SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). За допомогою даних виводів здійснюється зв'язок SPI, наприклад, використовуючи бібліотеку SPI.

LED: 13. Вбудований світлодіод, підключений до цифрового виводу 13. Якщо значення на виведення має високий потенціал, то світлодіод горить.

I²C: 20 (SDA) і 21 (SCL). За допомогою виводів здійснюється зв'язок по інтерфейсу I²C (TWI). Для створення використовується бібліотека Wire.

На платформі Mega2560 є 16 аналогових входів, кожен з роздільною здатністю 10 біт (тобто може приймати 1024 різних значення). Стандартно виводи мають діапазон вимірювання до 5 В відносно мінусу живлення, проте є можливість змінити верхню межу за допомогою використання AREF і функції *analogReference ().*

Додаткові виводи:

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

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

Велику популярність плата Arduino набула не тільки через низьку вартість, легкість розробки та програмування, але, головним чином, завдяки наявності плат розширення (шилдів), які додають Arduino додаткову функціональність. Шилди підключаються до Arduino за допомогою наявних штирьових роз'ємів. Він встановлюється зверху на контролер, утворюючи своєрідний «бутерброд». Існує безліч різних за функціональністю шилдів – від найпростіших, призначених для макетування, до складних, що являють собою окремі багатофункціональні пристрої.

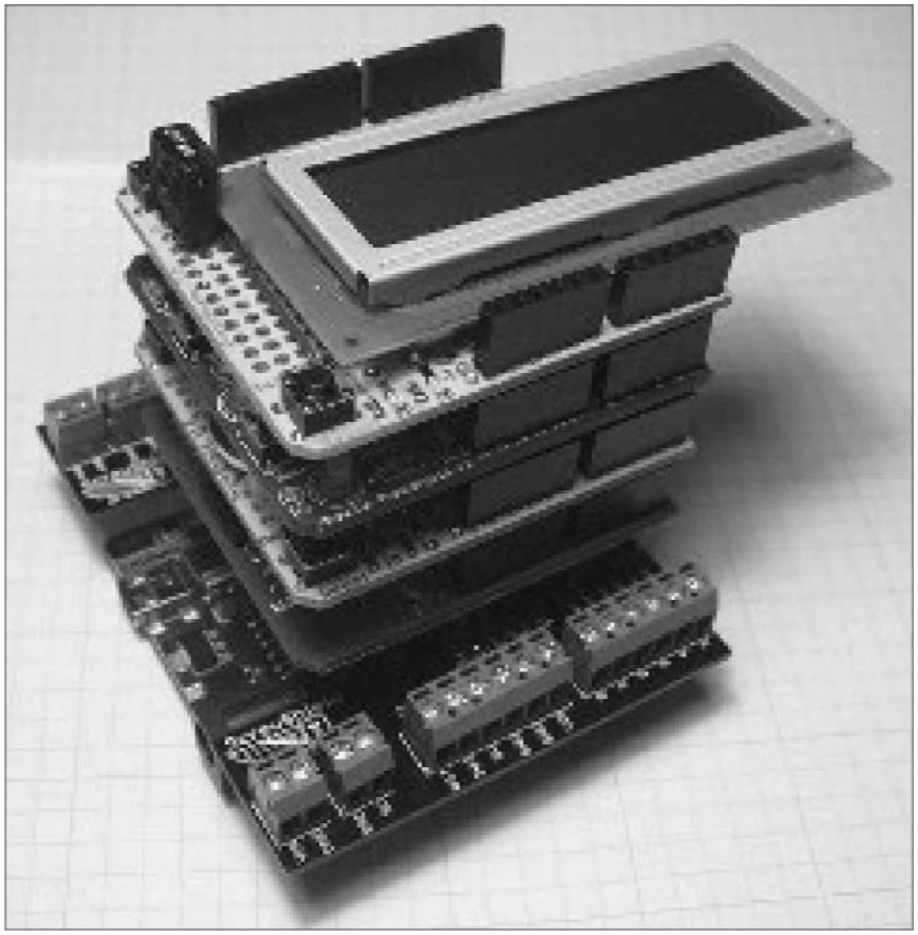


Рис. 1.4 Модульна структура встановлення плат розширення

Кілька прикладів де яких шилдів:

- ✓ **Ethernet Shield** – забезпечує підключення до Інтернету (Рис. 1.5);
- ✓ **MicroSD Shield** – забезпечує запис даних на карти MicroSD (Рис. 1.6);
- ✓ **Motor Shield** – забезпечує управління двигунами постійного струму (Рис. 1.7);

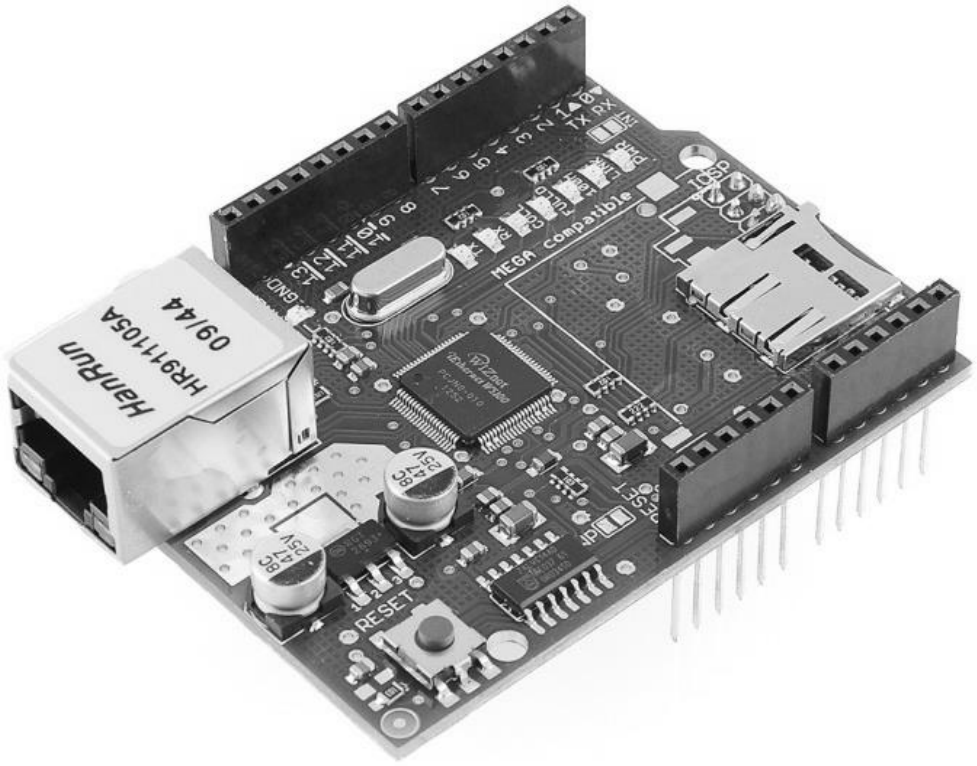


Рис. 1.5 Ethernet Shield

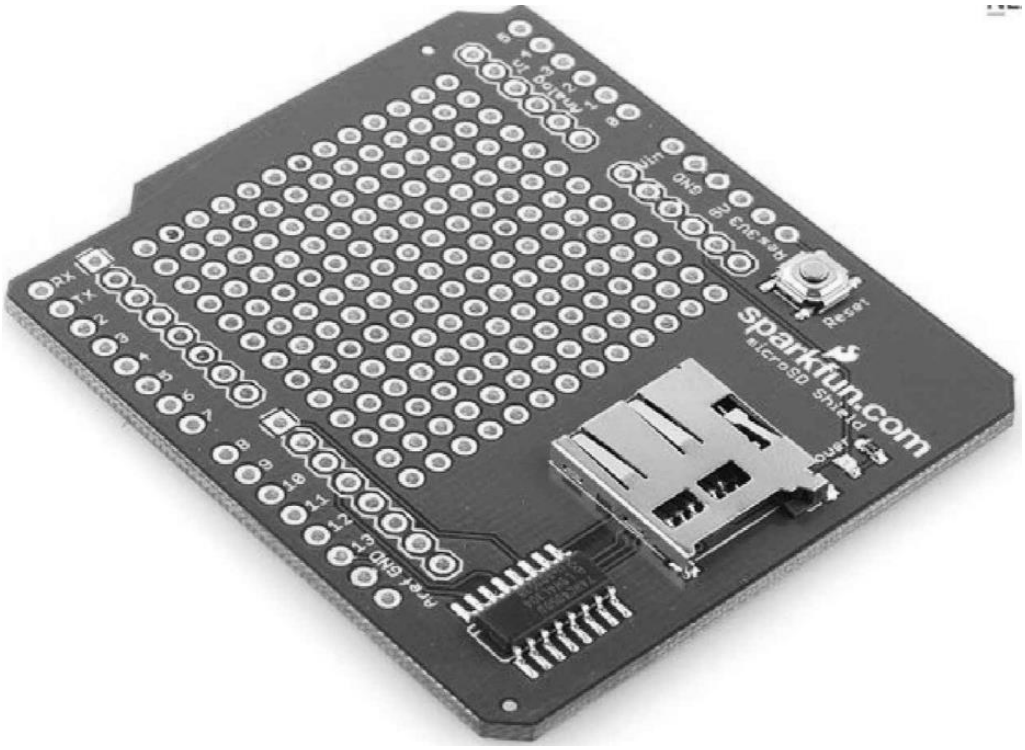


Рис. 1.6 MicroSD Shield

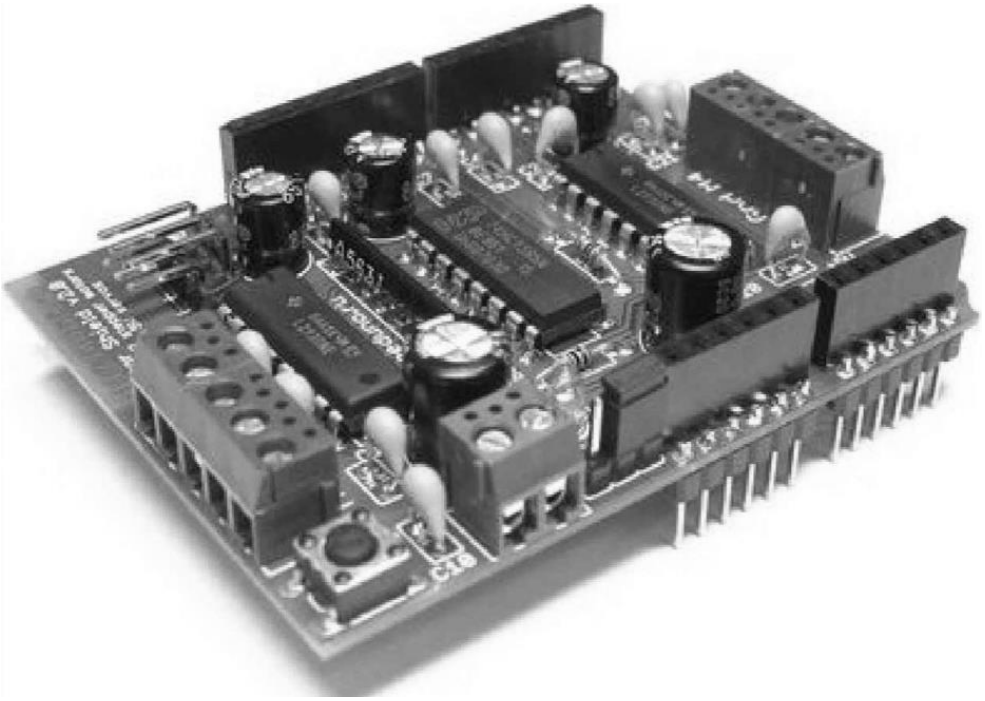


Рис. 1.7 Motor Shield

Кількість плат розширення (шилдів) постійно зростає. Ознайомитись із їх переліком можна на офіційному сайті проекту Arduino за адресою: *http://www.Arduino.cc/.*

1.3 Середовище розробки Arduino

Розробка власних програм на базі плат, сумісних з архітектурою Arduino, здійснюється в офіційному безкоштовному середовищі програмування Arduino IDE. Середовище призначене для написання, компіляції та завантаження власних програм в пам'ять мікроконтролера, що встановлений на платі Arduino-сумісного пристрою. Основою середовища розробки є мова Processing/Wiring — це, фактично, звичайний C++, доповнений простими і зрозумілими функціями для керування введенням/виведенням на контактах (пінах). Існують версії середовища для операційних систем Windows, Mac OS та Linux. Останню версію середовища Arduino можна завантажити зі сторінки завантаження офіційного сайту *http://Arduino.cc/en/Main/Software* (рис.1.8).

Архівний файл, що буде завантажений, вже містить все необхідне, у тому числі і драйвери. Після закінчення завантаження розпаковуємо завантажений файл у зручному для себе місце. Далі потрібно встановити драйвери. Для цього потрібно підключити Arduino до комп'ютера. На контролері має спалахнути індикатор живлення — зелений світлодіод. Операційна система починає спробу інсталяції драйвера, яка закінчується повідомленням **Програмне знеціннення драйвера не було встановлено**. Для завершення встановлення потрібно виконати де яку послідовність дій. Відкрити *Диспетчер пристроїв*. В переліку пристроїв знайти значок *Arduino Uno* (пристрій буде відзначений знаком оклику). В контекстному меню позначки *Arduino Uno* обрати пункт **Оновити драйвери** і далі пункт **Виконати пошук драйверів на цьому комп'ютері**.

Вказати шлях до драйвера – та сама папка на комп'ютері, куди розпаковували завантажений архів. В результаті отримуємо повідомлення **Оновлення програмного забезпечення для даного пристрою завершено**. Таким чином операційна система завершить інсталяцію драйвера.

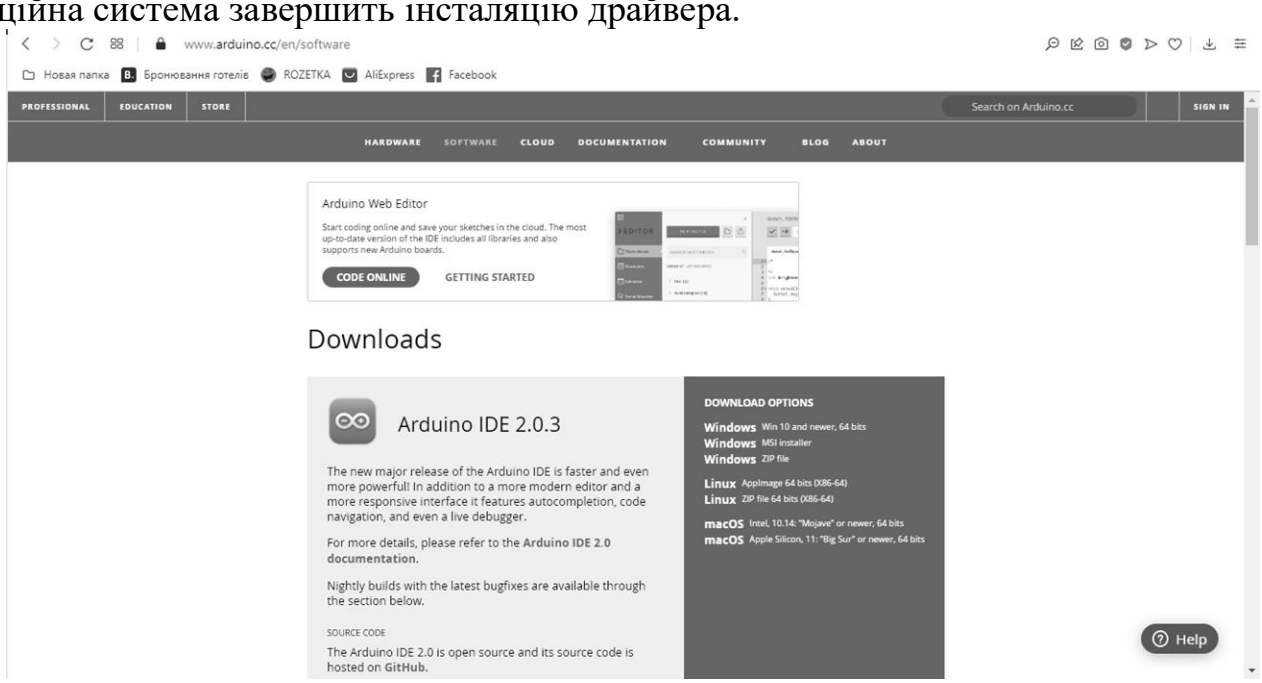


Рис.1.8 Завантаження середовища розробки

Для перевірки працездатності рекомендовано скористатися тестовим прикладом:

- Відкрийте середовище розробки Arduino та запустити тестову програму File > Examples > 0l.Basics > Blink.
- Тепер в меню Tools>Board необхідно вибрати пункт меню, що відповідає вашій моделі Arduino.
- У меню Tools>Serial Port виберіть послідовний порт, до якого підключена ваша плата. Як правило, це COM-порт з номером 3 (COM3) або вище (COM1 і COM2 зазвичай асоційовані з апаратними портами). Щоб дізнатися потрібний порт, можна тимчасово від'єднати Arduino і ще раз відкрити меню; зниклий порт і буде тим портом, з яким асоційований мікроконтролер. Назад підключіть пристрій до комп'ютера і виберіть з меню необхідний порт.
- Тепер можна працювати та завантажувати скетчі в Arduino (рис.1.9).

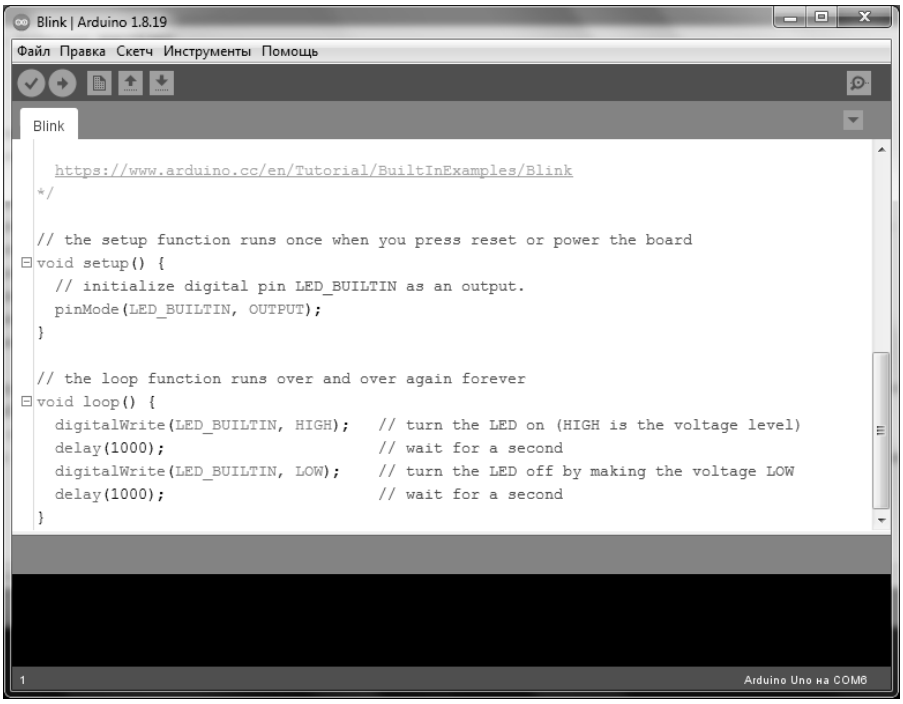


Рис.1.9 Загальний вигляд

Опис Arduino IDE

Середовище розробки Arduino (рис.1.9) складається із:

- ✓ вбудованого редактора програмного коду,
- ✓ області повідомлень,
- ✓ консолі виведення тексту,
- ✓ панелі інструментів з кнопками команд, що використовуються найчастіше,
- ✓ декількох меню.

Програма, написана в середовищі Arduino, називається *скетч.* Скетч пишеться в текстовому редакторі, що має колірне підсвічування створюваного програмного коду. Під час збереження і експорту проекту в області повідомлень з'являються пояснення та інформація про помилки. Вікно виведення тексту (консоль) показує повідомлення Arduino, що включають повні звіти про помилки та іншу інформацію. Кнопки панелі інструментів дозволяють перевірити і записати програму, створити, відкрити і зберегти скетч, відкрити моніторинг послідовної шини.

Додаткова функціональність скетчам, що розробляються, може бути додана за допомогою *бібліотек*, що являють собою спеціальним чином оформлений програмний код, що реалізує деякий функціонал, який можна підключити до створюваного проекту. Існує багато спеціалізованих бібліотек. Зазвичай бібліотеки пишуться так, щоб спростити вирішення того чи іншого завдання та приховати від розробника деталі програмно-апаратної реалізації. Середовище Arduino IDE поставляється з набором стандартних бібліотек: *Serial, EEPROM, SPI, Wire* та ін. Знайти їх можна у підкаталозі *libraries* каталогу встановлення *Arduino*. Необхідні бібліотеки також можуть бути завантажені з різних ресурсів. Папка бібліотеки розміщується в каталозі стандартних бібліотек (підкаталог *libraries* каталогу установки Arduino).

Всередині каталогу з ім'ям бібліотеки знаходяться файли *.cpp, *.h. До багатьох бібліотек надаються приклади, що розташовані в папці *Examples*. Якщо бібліотека встановлена правильно, вона з'являється у меню *Sketch\ImPort Library*. Вибір бібліотеки в меню призведе до додавання у вихідний код рядка:

#include <ім'я бібліотеки.h>

Ця директива включає заголовний файл з описом об'єктів, функцій і констант бібліотеки, які тепер можуть бути використані в проекті. Середовище Arduino компілюватиме створюваний проект разом із зазначеною бібліотекою.

Середовищем Arduino використовує принцип блокнота: стандартне місце для зберігання програм (скетчів). Скетчі з блокнота відкриваються через меню *File\Sketchbook* або кнопкою *Open* на панелі інструментів. При першому запуску програми Arduino автоматично створюється директорія для блокнота. Розташування блокнота змінюється через діалогове вікно *Preferences*.

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

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

При завантаженні скетчу використовується завантажувач (*Bootloader*) Arduino, невелика програма, що завантажується в мікроконтролер на платі. Вона дозволяє завантажувати програмний код без використання додаткових апаратних засобів. Завантажувач (Bootloader) активний протягом декількох секунд при перезавантаженні платформи і при завантаженні будь-якого з скетчів в мікроконтролер. Робота завантажувача (Bootloader) розпізнається по миготінню світлодіода на 13 піні (наприклад, при перезавантаженні плати).

2. ПРОГРАМУВАННЯ ARDUINO

Мова програмування середовища розробки аналогічна Wiring. Строго кажучи, це C/C++, доповнений деякими бібліотеками. Програми обробляються за допомогою препроцесора, а потім компілюються за допомогою AVR-GCC.

Разом з тим мова проста в освоєнні, і на даний момент Arduino – це, мабуть, найзручніший спосіб програмування пристроїв на мікроконтролерах.

Особливості мови Arduino

Мова Arduino має чотири складових:

- оператори
- дані
- функції
- бібліотеки

Оператори:

- setup ()
- loop ()
- оператори мови C

Дані: типи даних з мови C

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 одно» - ==. 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;

}

Цей цикл буде виконуватися, поки умова, що 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

& (побітове І)
| (побітове АБО)

^ (побітове XOR або виключаюче АБО)
~ (побітове НЕ)
<< (побітовий зсув вліво)
>> (побітовий зсув вправо)

Складні оператори

++ (інкремент)
-- (декремент)
+ = (складене додавання)
- = (складене віднімання)
* = (складене множення)
/ = (складене ділення)
& = (складене побітове І)
| = (складене побітове 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 скетч. Використовується вона для ініціалізації змінних, визначення режимів роботи виводів, запуску бібліотек що використовуються 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, прийом–передача 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 – тип даних цілі числа. Це основний тип даних для зберігання чисел.

В Arduino Uno (та інших платах на базі 8 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 (без знакові 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.

В Arduino Duo (32-х бітний) змінні double мають точність 8 байт (64 біта).

Створення (оголошення) *масиву.* 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').

2.1 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 та 40 мА в піковому значенні) в навантаження, що до нього підключене. INPUT_PULLUP – порт працює як вхід, але до нього підключається так 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 зчитати значення (int).
- Значення, що повертаються HIGH або LOW.

2.2 Функції часу

У ATmega328 передбачені три таймери/лічильники, на яких реалізовано функції часу, які використовують для формування і виміру часових інтервалів.

Основні функції часу:

1. 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.)

2. delayMicroseconds (ms)

- Параметри: ms – 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 ().*

3. millis ()

- Параметри: немає.
- 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.

4. Micros()

- Параметри: немає.
- 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 (Due 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, роздільна здатність функції становить вісім мікросекунд.

5. pulseIn (pin, value)
pulseIn (pin, value, timeout)

- Параметри: pin – номер виводу, на якому буде очікуватися сигнал; value – тип імпульсу: HIGH або LOW; timeout – 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.

Приклад використання функції *millis ():*

```
unsigned long time;
void setup ()  {
        Serial.begin (9600);
}
void loop ()   {
        Serial.print ( "Time:");
        time = millis (); // Виводимо час з моменту старту програми
        Serial.println (time); // Чекаємо 1 секунду, щоб не відправляти великий масив
даних
        delay (1000);
}
```

2.3 Асинхронний послідовний обмін

Найбільш розповсюджена форма в мікроконтроллерних системах – асинхронний обмін, при якому байт даних пересилається 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 випливати біт парності (контрольний біт), що використовується для перевірки правильності отриманих даних.

За бітом парності передається стоп-біт, що використовується приймачем для обробки закінчення передавання пакету.

Асинхронний пакет даних наведено на рис. 2.1. Існує набір параметрів, що повинний бути відомий при реалізації обміну. Одним з таких параметрів є число переданих бітів даних, що визначається типом приймального і передавального пристроїв. Пакет на рис. 2.1 містить тільки 5 біт даних (таке число бітів використовувалося в телетайпах), але можливі пакети довжиною до 8 біт.

Поряд з бітами парності («odd») чи непарності («even») можливі інші варіанти контрольних бітів: «no», «mark» і «SPace». «no» означає відсутність біта парності в пакеті. «mark» чи «SPace» означає, що замість біта парності завжди посилається „1” («mark») чи „0” («space»), відповідно. Ці варіанти контрольних бітів використовуються досить рідко – у тих випадках, коли необхідно дати приймачу додатковий час на оброблення пакета.

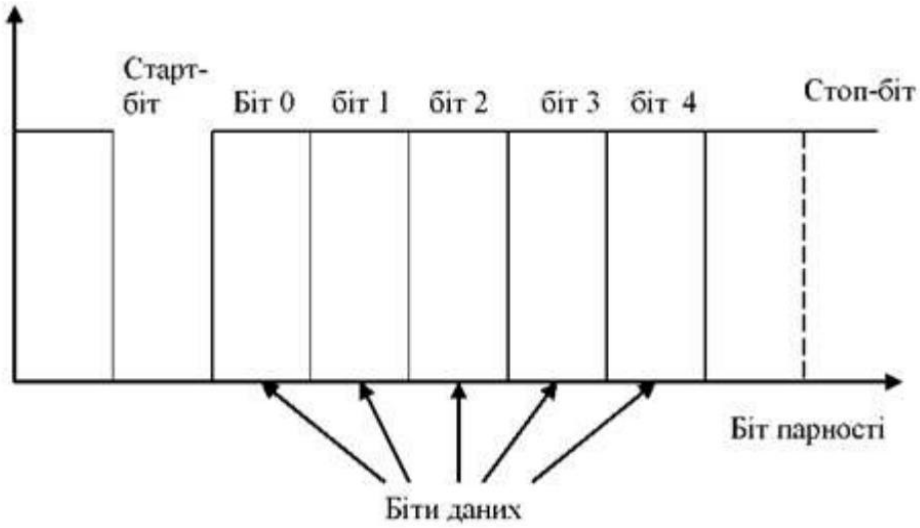


Рис. 2.1 Асинхронне послідовне передавання даних

Кількість стоп-бітів також може бути різною. Другий стоп-біт може вводитися для тієї ж мети, що і контрольні біти «mark» і «SPace» – щоб дати приймачу більше часу для обробки прийнятого пакета.

Практично всі сучасні пристрої використовують для асинхронного обміну формат даних «8-N-1», що означає передачу 8 біт даних, відсутність біта парності й один стоп-біт. Біт парності і додатковий стоп-біт, звичайно, не потрібні для послідовного зв’язку.

Найбільш популярний протокол асинхронного послідовного зв’язку називається «RS-232», що у наш час є міжнародним стандартом. Це дуже старий стандарт, який використовували для зв’язку комп’ютерів через комунікаційні (COM) порти. Але і зараз стандарт «RS-232» широко використовується у вигляді з'єднання через віртуальні COM порти при використанні протоколу USB.

Організація обміну даними між платою Arduino і комп’ютером через USB

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. Для відправки даних вибирається швидкість передачі зі списку, відповідна 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*. Клас – це абстрактний тип даних. За допомогою класу описується деяка сутність (її характеристики і можливі дії). Наприклад, клас може описувати змінні, параметри і функції послідовного

порту. Клас Serial містить близько 20 функцій. Розглянемо деякі.

Функції для роботи з послідовним портом плати Arduino:

1. if (Serial)

- Параметри: немає.
- Значення, що повертаються – boolean: повертає true, якщо вказаний послідовний порт готовий до роботи.
- Опис: дозволяє перевірити готовність певного послідовного порту.

2. Serial.begin(speed)
Serial.begin(speed, config)

- Параметри: *speed:* швидкість в бітах на секунду (бодах) – long, *config:* задає кількість біт даних, перевірку парності и стопові біти:
- *SERIAL_5N1*
- *SERIAL_6N1*
- *SERIAL_7N1*
- *SERIAL_8N1 (за замовченням)*
- *SERIAL_5N2*
- *SERIAL_8E2*
- *SERIAL_7O2*
- Опис: задає швидкість передачі даних по послідовному інтерфейсу в бітах в секунду (бодах). 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.

Другий аргумент цієї функції необов'язковий. Він дозволяє налаштувати кількість біт даних, перевірку парності і стопових біт. За замовчуванням, посилка складається з 8 біт даних, без перевірки парності, з одним стоповим бітом.

3. Serial.end ()

- Параметри: немає.
- 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 ().*

4. Serial.available()

- Параметри: немає.
- 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).

5. Serial.Read()

- Параметри: немає.
- Значення, що повертаються: перший байт прийнятих даних (або -1, якщо таких нема) – int.

- Опис: зчитує дані, що надходять по послідовному інтерфейсу.

6. 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».

  Приклад:
  Serial.print(78) - виведе "78"
  Serial.print(1.23456) - виведе "1.23"
  Serial.print('N') - виведе "N"
  Serial.print("Hello world.") - виведе "Hello world."

Другий параметр необов'язковий. Він задає формат виведення; цей параметр може набувати таких значень: BIN (двійкова система з основою 2), OCT (восьмерична система з основою 8), DEC (десяткова система з основою 10), HEX (шістнадцятирічна система з основою 16). Для числа з плаваючою точкою цей параметр визначає кількість десяткових знаків після коми.

Приклад:
Serial.print (78, BIN) - виведе "1001110"
Serial.print (78, OCT) - виведе "116"
Serial.print (78, DEC) - виведе "78"
Serial.print (78, HEX) - виведе "4E"
Serial.println (1.23456, 0) - виведе "1"
Serial.println (1.23456, 2) - виведе "1.23"
Serial.println (1.23456, 4) - виведе "1.2346"

7. Serialprintln (val)
Serial.println (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 обов'язково.
- Опис: виводить через послідовний порт ASCII-текст в зрозумілому для людини вигляді з символами повернення каретки (ASCII 13 або '\ r') і нового

рядка (ASCII 10 або 'n'). Ця команда має такі ж форми, як і *Serial.print ().*

Приклад коду для роботи з послідовним портом:

```
int inCOMingByte = 0;
void setup() {
        Serial.begin(9600);
}
void loop() {
        if (Serial.available() > 0) {
                inCOMingByte = Serial.Read();
                Serial.print("I received: ");
                Serial.println(inCOMingByte, DEC);
        }
}
```

2.4 Переривання

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 паралельних 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

і реалізовані в файлі:

\ Hardware \ cores \ Arduino \ WInterrupts.c

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.

В Arduino Due є ще одне значення: HIGH – переривання буде спрацьовувати щоразу, коли на виводі присутній високий рівень сигналу (тільки для Arduino Due).

- Значення, що повертаються: немає.

2. detachInterrupt(interrupt)
detachInterrupt(digitalPinToInterrupt(pin))
detachInterrupt(pin) *(тільки для Due, Zero)*

- Параметри: *interrupt* — номер переривання, який потрібно відключити; *pin* — контакт, на якому відключається переривання (тільки для Due)
- Значення, що повертаються: немає.
- Опис: Забороняє задане переривання.
  Забороняє переривання, для того, щоб відключити доступ до даних в процесі виконання переривання.

3. noInterrupts ()

- Параметри: немає.
- Значення, що повертаються: немає.
- Опис: забороняє переривання (повторно дозволити їх можна функцією interrupts ()).
  Переривання дозволяють деяким важливим завданням виконуватися у фоновому режимі і за замовчуванням включені. Якщо переривання відключені, деякі функції не будуть працювати, а надходять від інших пристроїв дані можуть ігноруватися. Однак, переривання можуть незначно сповільнювати виконання програми, тому в найбільш критичних до часу ділянках коду вони можуть бути відключені.

4. interrupts ()

- Параметри: немає.
- Значення, що повертаються: немає.
- Опис: повторно дозволяє переривання (після того, як вони були відключені функцією *noInterrupts ()*).

Аналогове введення/виведення. Функція ШІМ

1. analogWrite[2] (pin, value)

- Параметри: *pin* – вивід, на якому буде формуватися напруга; *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 із заданим коефіцієнтом заповнення до наступного виклику функції *analogWrite()* (або до моменту виклику *digitalRead ()* або *digitalWrite (),* взаємодіючих з цим же виводом). Частота ШІМ становить приблизно 490 Гц. На більшості плат

[2] . Функція *analogWrite ()* не має нічого спільного з аналоговими виводами і функцією *analogRead()*

Arduino (на базі мікроконтролерів ATmega168 або ATmega328) функція *analogWrite ()* працює з виводами 3, 5, 6, 9, 10 і 11. На Arduino Mega функція працює з виводами з 2 по 13.

Програмування АЦП

Дуже корисний модуль в складі мікроконтролера – аналого-цифровий перетворювач. Він дозволяє мікроконтролеру вимірювати довільне напруження.

Аналого-цифровий перетворювач зчитує величину напруги на аналогових пінах. Це дає можливість зчитувати дані з датчика освітленості, виміряти напругу живлення і т.д.

Основні команди для програмування АЦП в Arduino:

1. analogReference (type)

- Параметри: *type* – тип джерела опорної напруги (DEFAULT, INTERNAL, INTERNAL1V1, INTERNAL2V56 або EXTERNAL).
- Значення, що повертаються: немає.
- Опис: встановлює джерело опорної напруги, що використовується при зчитуванні аналогового сигналу (іншими словами, задає максимальне значення вхідного діапазону).

  Для вибору джерела опорної напруги доступні значення:
  - ✓ DEFAULT: опорна напруга за замовчуванням 5 В (на 5 В-платах Arduino) або 3.3 В (на 3.3 В-платах Arduino).
  - ✓ INTERNAL: внутрішня опорна напруга дорівнює 1.1 В в мікроконтролерах ATmega168 і ATmega328, або 2.56 В в мікроконтролері ATmega8 (не доступно в Arduino Mega).
  - ✓ INTERNAL1V1: внутрішнє опорна напруга 1.1 В (тільки для Arduino Mega).
  - ✓ INTERNAL2V56: внутрішнє опорна напруга 2.56 В (тільки для Arduino Mega).
  - ✓ EXTERNAL: за опорну напругу буде використовуватися напруга, що прикладена до виводу AREF (від 0 до 5В).

2. analogRead (pin)

- Параметри: *pin* – номер виводу, з якого буде зчитуватися напруга (0 - 5 для більшості плат, 0 - 7 для Mini і Nano, 0 - 15 для Mega).
- Значення, що повертаються: ціле число *int* (від 0 до 1023).
- Опис: зчитує величину напруги з зазначеного аналогового виводу. АЦП перетворювач перетворює вхідну напругу з діапазону 0 – 5 В в 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 і роздільна здатність можуть змінюватися за допомогою функції *analogReference ().*

Для зчитування значення з аналогового входу потрібно близько 100 мікросекунд (0.0001 с), тому максимальна частота опитування виводу приблизно дорівнює 10 000 разів в 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, наведення від руки поблизу плати і т.д.

2.5 Клас String

Клас String з'явився в ядрі, починаючи з версії 0019. Він дозволяє здійснювати більш складну обробку текстових рядків, ніж звичайні масиви символів. За допомогою класу String можливо об'єднувати і розширювати рядки, здійснювати пошук і заміну підрядків, і багато іншого. Він займає більше пам'яті, ніж простий масив символів, але надає більше можливостей.

Загально прийнято називати рядки з масивів символів з маленькою літери «s» (string), а екземпляри класу String - з великою «S». Слід мати на увазі, що рядкові константи, укладені в подвійні лапки, інтерпретуються як масиви символів, а не екземпляри класу String.

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

Опис String()

Створює екземпляр класу *String*. Існує кілька різних версій цієї функції, які створюють об'єкт String з різних типів даних (іншими словами, формують рядок з послідовності символів):

- рядкова константа в подвійних лапках (тобто масив символів);
- одиночний символ в одинарних лапках;
- другий примірник класу String;
- цілочисельна константа типу int або long;
- цілочисельна константа типу int або long із зазначенням основи системи числення;
- цілочисельна змінна типу int або long
- цілочисельна змінна типу int або long із зазначенням основи системи числення.

При створенні об'єкта String з числа, результуючий рядок буде містити ASCII-представлення цього числа. За замовчуванням вважається, що число зазначено в десятковій системі, тому:

String string = String (13)

приведе до створення рядка "13". Можна вказувати основу системи числення, наприклад:

String string = *String (13, HEX)*

приведе до створення рядка "D", що є шістнадцятковим поданням десяткового числа 13. Те ж саме в двійковій системі:

String string = String (13, BIN)

приведе до створення рядка "1101", що є двійковим поданням числа 13.

Текстові рядки

Опис: текстові рядки можуть бути оголошені 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. Далі описано другий спосіб.

Приклади:

```
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".
```

Допустимі операції при оголошенні рядків

- Оголосити масив символів без його ініціалізації (Strl).
- Оголосити масив символів з одним надлишковим елементом, компілятор сам додасть необхідний нульовий символ (Str2).
- Додати нульовий символ явно (Str3).
- Ініціалізувати масив за допомогою рядкової константи, укладеної в лапки; компілятор створить масив необхідного розміру з нульовим символом в кінці (Str4).
- Ініціалізувати масив за допомогою рядкової константи, явно вказавши його розмір (Str5).
- Ініціалізувати масив надлишкового розміру, залишивши місце для більш довгих рядків (Str6).

Нульовий завершальний символ

Як правило, всі рядки завершуються нульовим символом (ASCII код 0). По суті, це означає, що довжина рядка повинна бути на 1 символ більше, ніж текст, який необхідно у ній зберігати. Саме тому Str2 і Str5 повинні бути довжиною 8 символів, не дивлячись на те, що слово «Arduino» займає всього 7 – остання позиція автоматично заповнюється нульовим символом. Розмір Str4 автоматично стане рівним 8 – один символ потрібно для завершального нуля. У рядку Str3 було самостійно вказано нульовий символ (позначається '\ 0').

Слід мати на увазі, що в цілому можна оголосити рядок і без завершального нульового символу (наприклад, якщо задати довжину Str2 що дорівнює 7, а не 8). Однак, це призведе до непрацездатності більшості строкових функцій, тому не слід навмисно так робити.

Основні команди:

1. String (val)
String (val, base)

- Параметри: *val* – змінна, значення якої необхідно представити у вигляді об'єкта String - string, char, byte, int, long, unsigned int, unsigned long, base; (необов'язковий параметр) – основа системи числення, в якій необхідно представити ціле число.
- Значення, що повертається: немає.

2. indexOf()

- Опис: здійснює пошук символу або підрядку в рядку (String). За

замовчуванням, пошук здійснюється з початку рядка, однак можна вказати і певну позицію, з якою необхідно почати пошук. Така можливість дозволяє знаходити всі входження підрядка в рядку.

Синтаксис:

string. indexOf(val)

string.indexOf(val, from)

- Параметри: *string* – змінна типу String; *val* – шукане значення (char або String); *from* – початкова позиція для пошуку.
- Значення, що повертаються: індекс підрядку val в рядку String, або -1, якщо підрядок не знайдено.

3. lastIndexOf ()

- Опис: здійснює пошук символу або підрядку в рядку (String). За замовчуванням, пошук здійснюється з кінця рядка, однак можна вказати і певну позицію, з якої необхідно переглядати рядок у зворотному порядку. Така можливість дозволяє знаходити всі входження підрядку в рядку.

Синтаксис:

string.lastIndexOf (val)

string.lastIndexOf (val, from)

- Параметри: *string* – змінна типу String; *val* – шукане значення (char або String); *from* – початкова позиція для пошуку в зворотному порядку.
- Значення, що повертаються: індекс підрядку val в рядку String, або -1, якщо підрядок не знайдено.

4. length ()

- Опис: повертає довжину рядка String в символах[3].

Синтаксис:

string.length ()

- Параметри: *string* – змінна типу String.
- Значення, що повертаються: довжина рядка String в символах.

5. substring ()

- Опис: повертає підрядок в рядку (String). Функція приймає два параметри: початковий індекс і кінцевий індекс в рядку (необов'язковий параметр). При цьому початковий індекс є включеним (відповідний символ буде включений в підрядок), а кінцевий індекс - НЕ включеним (відповідний символ не включається в результуючий підрядок). Якщо кінцевий індекс відсутній, то значення, що повертається функцією, буде містити всі символи від початкового індексу і до кінця рядка.

Синтаксис:

string.substring (from)

string.substring (from, To)

- Параметри: *string* – змінна типу String; *from* – початковий індекс в рядку; *To* (необов'язковий параметр) – кінцевий індекс в рядку.
- Значення, що повертаються: підрядок.

---

[3] Зверніть увагу, що при підрахунку кількості символів функція не враховує завершальний нульовий символ

6. charAt ()

- Опис: повертає вказаний символ з рядка String.

  Синтаксис:

  string.charAt (n)

- Параметри: *string* – змінна String; *n* – номера символу.
- Значення, що повертаються: n-ний символ рядка String.

7. COMpareTo()

- Опис: перевіряє два рядки типу String на рівність, а також дозволяє визначити, який з порівнюваних рядків йде раніше іншого в алфавітному порядку. Рядки порівнюються посимвольно по ASCII-коду кожного символу. Наприклад, символ 'a' йде до символу 'b', але після символу 'A'. Всі цифри йдуть до букв.

  Синтаксис:

  string.COMpareTo (string2)

- Параметри: *string* – змінна типу String; *string2*– друга змінна типу String.
- Значення, що повертаються: -1 (якщо рядок string йде до рядка string2 в алфавітному порядку); 0 (якщо рядок string еквівалентний рядку string2); 1 (якщо string йде після string2).

8. ToInt ()

- Опис: перетворює рядок (String) в ціле число. Рядок повинен починатися з символьного запису цілого числа.

  Синтаксис:

  string.ToInt ()

- Параметри: *string з*мінна типу String.
- Значення, що повертаються: long.

3. ПРОГРАМУВАННЯ МЕРЕЖЕВИХ ВЗАЄМОДІЙ

На теперішній час, область Інтернету досить широко використовується апаратними пристроями, що володіють підтримкою мережевих взаємодій. Принтери, побутові пристрої автоматики не тільки стають все більш інтелектуальними, а й підтримують можливість підключення до Інтернету. Плати Arduino займають передові позиції серед саморобних інтернет-пристроїв, підтримуючи можливість дротового або бездротового підключення до Інтернету за допомогою плат розширення *Ethernet* або *Wifi*.

3.1 Мережеве обладнання

Є декілька варіантів підключення Arduino до мережі. Можна, для підключення через локальну мережу, використати плату розширення Ethernet з Arduino Uno, або модель Arduino із вже вбудованим адаптером Ethernet. Або ж взагалі придбати плату розширення Wifi для підключення до бездротової мережі.

Плата розширення Ethernet

Плата розширення Ethernet (рис. 1.5) яка дає можливість підключення до мережі Ethernet, але також, має слот для карти пам'яті MicroSD, яку можна використовувати для зберігання даних.

В офіційних платах використовується мікросхема W5100, але можна знайти більше найдешевші плати розширення Ethernet, що побудовані на наборі мікросхем ENC28J60. Але ці більше дешеві плати не сумісні з бібліотекою Ethernet, і це треба враховувати при використанні в своїх схемах.

Альтернативою використанню окремої плати розширення може бути, наприклад, плата Arduino із вже вбудованим адаптером Ethernet. Офіційною вважається модель Arduino Ethernet, однак у продажу є і досить непогані та сумісні з Uno плати. Прикладом може бути плата EtherTen, що виробляється компанією Freetronics (www.freetronics.COM).

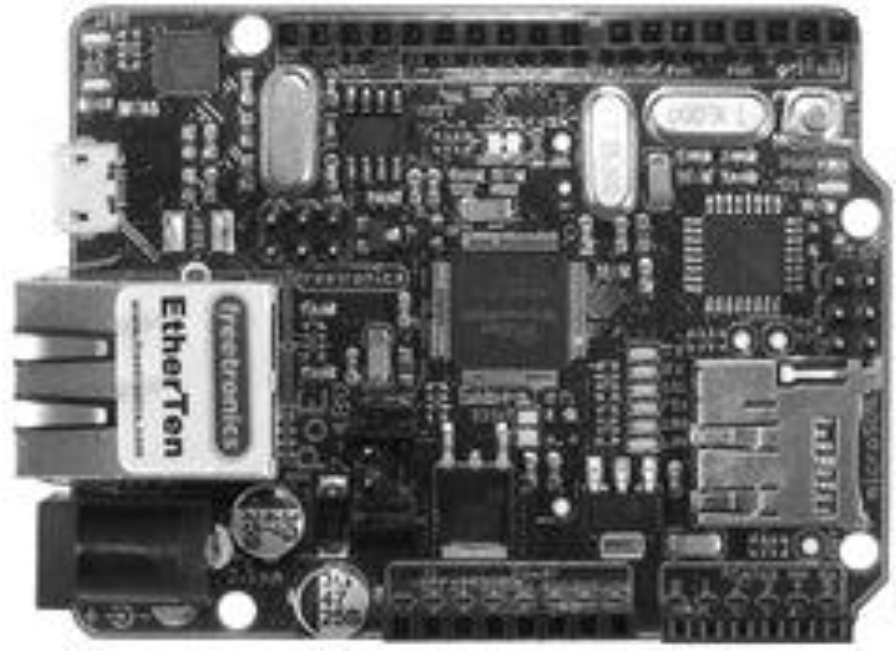


Рис. 3.1 Плата EtherTen

Для мережевих проектів на основі Arduino, більш зручнішими є комбіновані плати, що містять все необхідне. Плати Arduino Ethernet підтримують технологію живлення лініями Ethernet (POWER over Ethernet, PoE) через окремий інжектор PoE, що дозволяє зменшити кількість дротів, що йдуть до плати Arduino, до єдиного кабелю Ethernet. Плати EtherTen випускаються вже налаштованими живленням з використанням технології PoE. Більше повну інформацію про використання технології PoE у платах EtherTen можна знайти на сайті розробника.

Плата розширення Wifi.

Головна проблема підключення до Інтернету через Ethernet полягає в необхідності прокладання кабелю. Якщо є потреба у підключенні Arduino до Інтернету або мережі без використання дротів, тоді знадобиться плата розширення Wifi. Ці плати коштують досить дорого, але і тут можна знайти більш дешевші альтернативи сторонніх виробників.

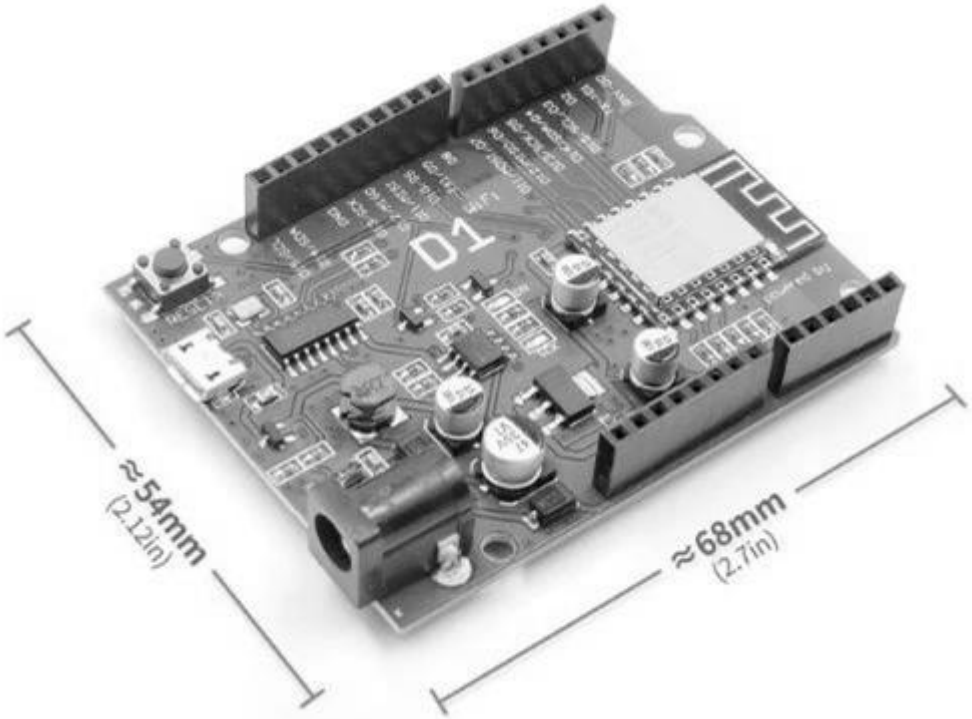


Рис. 3.2 Плата розширення Arduino UNO R3 з Wifi на основі CH340G ESP8266 ESP-12 WemOS

Наприклад, Wifi Shield ESP-12E (рис. 3.2) уявляє собою плату Arduino з вбудованим WIFI-модулем, яка має формфактор Arduino Uno. Він спроектований на основі поширеного чіпа ESP8266EX компанії ESPressif. Мікросхема володіє розширеною версією 32-бітного процесора Lx106 і вбудованої оперативної пам'яттю. Для подальшої роботи з даної платою в середовищі розробки Arduino IDE необхідно буде встановити драйвер CH340.

Варіантом більш простого (за ціною, але не за якістю) модуля для підключення різних пристроїв до мережі через Wi-Fi, може бути плата, що будується на базі чіпа ESP8266 компанії Espressif (рис. 3.3). Ця плата відрізняється енергоефективністю та високим ступенем інтеграції, що дозволяє використовувати мінімум елементів обв'язування чіпа.

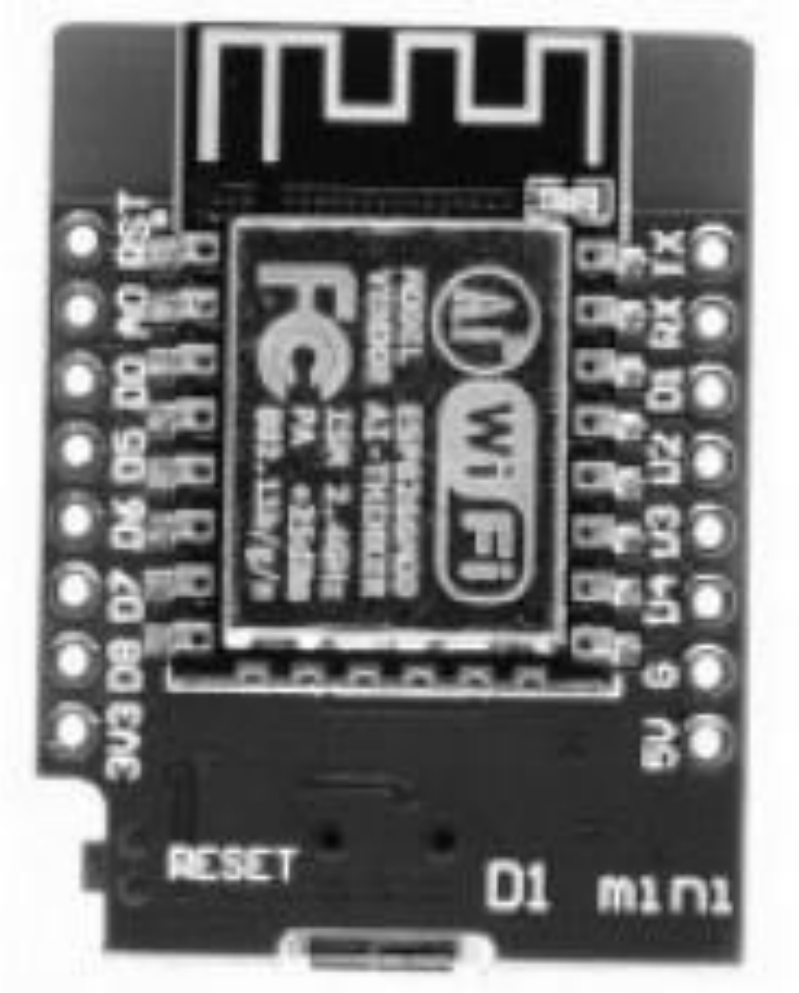


Рис. 3.3 Wifi модуля ESP8266

Чіп ESP8266 може використовуватися двома способами:

- як міст UART-Wifi, при цьому він керується зовнішнім пристроєм за допомогою АТ-команд
- як самостійний пристрій. Керування здійснюється вбудованим в кристал мікроконтроллером Tensilica's L106 Diamond series

Також модуль пропонує кілька варіантів роботи з Wi-Fi мережами, у тому числі може виступати як клієнт мережі, і сам створювати точку доступу. Також слід враховувати, що модуль ESP8266 розрахований тільки на 3,3 В.

Технічні характеристики:

- ✓ Підтримувані мережі: 802.11 b/g/n
- ✓ Швидкість UART: 9600
- ✓ Робоча напруга: 3,3 В
- ✓ Максимальний струм споживання (при передачі): 240 мА
- ✓ Розміри: 23,5 х 16 мм

Існує безліч модифікацій плат, які називаються зазвичай від ESP-01 до ESP-12. Сьогодні вже з'явилися ще інші назви плат від сторонніх розробників. Відмінності в платах полягає, в основному в портах введення-виведення, кількості флеш-пам'яті, виду конекторів і т.п. Процесор - той самий, отже з погляду програмування немає значення яку плату програмувати.

3.2 Бібліотека Ethernet

Для роботи з Ethernet Shield використовується стандартна бібліотека Arduino Ethernet Library. Вона постачається у складі дистрибутива Arduino. З моменту випуску в 2011 версії Arduino 1.0, бібліотека Ethernet зазнала суттєвих змін. Вона не тільки дозволяє платі Arduino з адаптером Ethernet діяти в ролі веб-сервера або веб-клієнта (можливість надсилати запити, подібно браузерам), але й реалізує додаткові можливості, такі як підтримка протоколу динамічної конфігурації мережного вузла (Dynamic HOSt Configuration ProTocol, DHCP), що автоматично привласнює платі IP-адресу. Повний опис бібліотеки можна знайти на офіційному сайті проекту Arduino за посиланням: https://www.Arduino.cc/reference/en/libraries/ethernet/.

Організація з'єднання

На першому етапі, перш ніж приступити до взаємодій через мережу, необхідно встановити з'єднання з мережею. Це завдання вирішує функція Ethernet.begin(). Вона дозволяє вручну вказати всі параметри з'єднання з використанням наступного синтаксису:

Ethernet.begin(Mac, ip, dns, gateway, subnet)

Розглянемо кожен із цих параметрів:

- *Mac* – MAC-адреса мережевої карти;
- *ip* – IP-адреса плати (будь-який допустимий у вашої мережі);
- *dns* – IP-адреса сервера доменних імен (Domain Name Server, DNS);
- *gateway* – IP-адреса шлюзу для виходу в Інтернет;
- *subnet* – маска підмережі.

Усі параметри, окрім *Mac*, є необов'язковими, і в 90% випадків вам доведеться вказувати лише параметри *Mac* та *ip* або, можливо, тільки *Mac*. Всі інші параметри будуть налаштовані автоматично. MAC-адреса, або адреса доступу до середовища (Media Access Control), – це унікальна ідентифікатор інтерфейсу мережі. Іншими словами, це адреса плати розширення Ethernet або чогось іншого, що надає мережевий інтерфейс у розпорядження Arduino. Ця адреса має бути унікальною лише для вашої мережі. Його зазвичай можна знайти на наклейці зі зворотного боку плати Ethernet або Wifi (рис. 3.4) або на упаковці. Якщо ви користуєтесь старою платою, яка не має MAC-адреси, то можете просто створити свою адресу. Але треба пам'ятати, що не повинно бути в одній локальній мережі двох пристроїв з однією тією ж MAC-адресою. Можна створити з'єднання з мережею із застосуванням DHCP і отримати динамічний IP-адреса, як показано далі:

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

byte Mac[]= {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};

void    setup() {
                Ethernet.begin(Mac);
}
```

Рис. 3.4 MAC-адреси на маршрутизаторі Tp-Link

Якщо потрібно присвоїти платі фіксовану IP-адресу, а це бажано, коли плата Arduino діє в ролі веб-сервера, можна використати приблизно такий код:

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

byte Mac[] = {0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
byte ip[] = {10, 0, 1, 200};

void setup() {
                Ethernet.begin(Mac, ip);
}
```

IP-адреса у параметрі *ip* повинна бути допустимою для мережі, в якій встановлено модуль. Якщо викликати функцію *Ethernet.begin* без параметра з IP-адресою, вона спробує отримати його з використанням DHCP і поверне 1, якщо з'єднання було встановлено та динамічний адреса успішно отримана, інакше поверне 0. Можна написати тестову скетч, який буде встановлювати з'єднання та викликати функцію *localIP* для отримання IP-адреси, що призначено Arduino. Наступний приклад виконує таку перевірку та виводить повідомлення з результатами в монітор послідовного порту:

```
// sketch_dhcp

#include  <SPI.h>
#include  <Ethernet.h>

byte Mac[] = {0x00, 0xAA, 0xBB, 0xCC, 0xDE, 0x02};

void setup() {
        Serial.begin(9600);
        while (!Serial){}; // для совместимости с Leonardo

        if (Ethernet.begin(Mac)) {
        Serial.println(Ethernet.localIP());
        }
        Else {
        Serial.println("Could not connect To network");
```

```
        }
}

void loop() {
}
```

Налаштування веб-сервера

Більшість функцій, що необхідні для реалізації веб-сервера, міститься в клас *EthernetServer*. Щоб запустити веб-сервер після встановлення з'єднання з мережею потрібно пройти ще два етапи.

По-перше, потрібно створити новий об'єкт сервера, вказавши номер порту, який має використовуватись для прийому вхідних запитів. Це оголошення знаходиться в скетчі перед функцією setup:

```
EthernetServer server = EthernetServer(80);
```

Зазвичай для прийому запитів веб-сервери використовують порт 80. Тобто, якщо налаштувати свій веб-сервер на обслуговування порту 80, то не доведеться додавати цей номер в URL-адреси, щоб зв'язатися з сервером.

По-друге, щоб фактично запустити сервер, у функції setup потрібно виконати наступну команду:

```
server.begin();
```

Ця функція запустить сервер, який чекатиме, доки хтось не запитає сторінку, яку обслуговує цей сервер. Фактичне обслуговування здійснюється у функції *loop* із застосуванням функції *available*. Ця функція повертає *null* (якщо немає запитів на обслуговування) або об'єкт *EthernetClient*. Цей об'єкт використовується також для надсилання вихідних запитів із Arduino до зовнішніх веб-серверів. У цьому випадку *EthernetClient* представляє з'єднання між веб-сервером та браузером клієнта. Отримавши цей об'єкт, можна прочитати вхідний запит за допомогою *Read* та повернути HTML-відповідь за допомогою функцій *Write, print* та *println*. Закінчивши відправку HTML-відповіді клієнту потрібно завершити сеанс викликом функції *sTop* об'єкта клієнта. Далі більш детально буде роглянуто як це зробити на прикладі «Фізичний веб-сервер».

Виконання запитів

Плата Arduino може діяти не тільки як веб-сервер, але і як веб-браузер, надсилаючи запити віддаленим веб-серверам, які можуть перебувати у локальній мережі або в Інтернеті. Щоб виконати запит, спочатку потрібно встановити з'єднання з мережею, як у випадку з веб-сервером, що вже було описано раніше, але замість об'єкта *EthernetServer* створити об'єкт *EthernetClient*:

```
EthernetClient client;
```

Більше з об'єктом клієнта нічого не потрібно робити, перш ніж надсилати веб-запит. Щоб надіслати веб-запит, потрібно виконати наступні дії:

```
if (client.connect ("api.Openweathermap.org", 80))
{
        client.println ("GET  /data/2.5/weather?q=Manchester,uk HTTP/1.0");
        client.println();
        while (client.connected())
        {
                while (client.available())
                {
                        Serial.Write(client.Read());
                }
        }
        client.sTop();
}
```

Функція *connect* поверне *true*, якщо з'єднання з веб-сервером було успішно встановлено. Дві команди *client.println* надсилають веб-серверу запит на отримання бажаної сторінки. Потім два вкладені цикли *while* читають дані, поки клієнт залишається підключеним до веб-сервера і продовжують надходити дані. Може здатися привабливим поєднати два цикли *while* в один з умовою *client.available() && client.connected(),* але таке об'єднання далеко не те саме, що два окремих цикли, оскільки дані можуть надходити від веб-сервера фрагментами через низьку швидкість мережі або з інших причин. Зовнішній цикл підтримує з'єднання відкритим, а внутрішній витягує дані. Це блокуюче рішення (скетч не зможе робити жодних інших дій, доки виконання запиту не завершиться).

Приклади використання Ethernet (фізичний веб-сервер)

Розглянемо приклад практичного використання бібліотеки Ethernet. Приклад ілюструє найчастіший випадок використання мережевих можливостей Arduino. Тут плата виступає у ролі веб-сервера. Підключившись до веб-серверу Arduino, відвідувачі зможуть не лише читати аналогові входи, а й змінювати рівні на цифрових виходах, через використання радіо-кнопок на веб-сторінці (рис. 3.5).

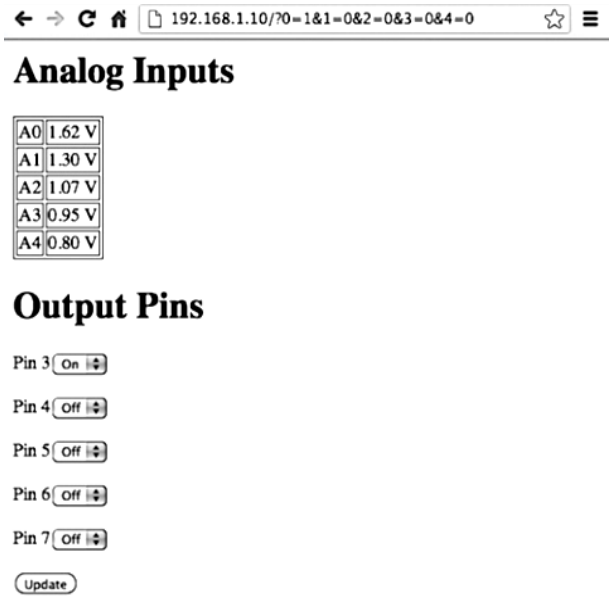


Рис. 3.5 Інтерфейс фізичного веб-сервера

Цей приклад демонструє простий спосіб з'єднання плати Arduino зі смартфоном або планшетним комп'ютером. Для надсилання запитів Arduino достатньо найпростішого браузера. В першої частині скетчу буде розміщено код, стандартний для будь-якого скетчу, що реалізує мережеві взаємодії. Тут виконується імпортування бібліотек та визначення об'єктів *EthernetServer* та *EthernetClient*. Далі визначаються змінні:

```
const int numPins = 5;
int pins[] = {3, 4, 5, 6, 7};
int pinState[] = {0, 0, 0, 0, 0};
char line1[100];
char buffer[100];
```

Константа *numPins* визначає розмір масивів *pins* і *pinState*. Масив *pinState* призначений для зберігання станів цифрових виходів, *HIGH* або *LOW*. Функція *setup* налаштовує всі контакти, перелічені у масиві *pins*, на роботу у режим цифрових виходів. Вона також встановлює з'єднання з мережею, як було показано у прикладах раніше (sketch_dhcp). Нарешті, масиви символів *line1* та *buffer* призначені для зберігання першого та наступних рядків HTTP-запиту відповідно.

Далі наводиться функція loop:

```
void loop(){
        client = server.available();
        if (client) {
                if (client.connected()) {
                        ReadHeader();
                        if (! pageNameIs("/")) {
                                client.sTop();
                                return;
                        }
                        client.println(F("HTTP/1.1 200 OK"));
                        client.println(F("Content-Type: tEXT/html"));
                        client.println();

                        sendBody();
                        client.sTop();
                }
        }
}
```

Функція перевіряє наявність будь-яких запитів від браузерів, які очікують на обробку. Якщо отримано запит і з'єднання з клієнтом ще не розірвано, викликається функція *ReadHeader*. Функція *ReadHeader* читає вміст заголовка запиту в буфер (*line1*) і пропускає інші рядки запиту. Це необхідно, щоб отримати ім'я сторінки, запитаної браузером, та будь-які додаткові параметри запиту, якщо вони є. Через великий обсяг тексту, який надсилається скетчем в монітор послідовного порту та мережу, тут використовується функція *F*, що зберігає масиви символів у флеш-пам'яті.

Після читання заголовка викликається функція *pageNameIs*, щоб перевірити збіг імені запрошеної сторінки з ім'ям кореневої сторінки (/). Якщо

була запитана не коренева сторінка, такий запит ігнорується. Це важливо, тому що багато браузерів посилають веб-серверу додаткові запити. Ці запити не потрібно плутати з іншими запитами до сервера.

Тепер потрібно згенерувати відповідь із заголовком та деякою розміткою HTML, яку міг би відобразити браузер. Функція *sendHeader* генерує відповідь "OK", щоб показати, що запит браузера визнано припустимим. Функція *sendBody*, що представлена далі, організована набагато складніше:

```
void sendBody() {
        client.println(F("<html><body>"));
        sendAnalogReadings();
        client.println(F("<h1>Output Pins</h1>"));
        client.println(F("<form method='GET'>"));
        setValuesFromParams();
        setPinStates();
        sendHTMLforPins();
        client.println(F("<input type='submit' value='Update'/>"));
        client.println(F("</form>"));
        client.println(F("</body></html>"));
}
```

Вона виводить простий макет HTML-сторінки, спираючись на безліч допоміжних функцій, які були створені, щоб розбити код на більш керовані фрагменти. Перша з них – *sendAnalogReadings*:

```
void sendAnalogReadings() {
    client.println(F("<h1>Analog Inputs</h1>"));
    client.println(F("<table border='1'>"));
    for (int i = 0; i < 5; i++) {
        int Reading = analogRead(i);
        client.print(F("<tr><td>A"));
        client.print(i);
        client.print (F("</td><td>"));
        client.print ((float) Reading / 205.0);
        client.println(F(" V</td></tr>"));
    }
    client.println ("</table>");
}
```

Вона виконує обхід усіх аналогових входів, читає їх значення та виводить HTML-таблицю з прочитаними значеннями у вольтах. *SendBody* викликає також функції *setValuesFromParams* і *setPinStates*. Перша записує в масив *pinStates* стани *HIGH* або *LOW* цифрових виходів, витягуючи їх із параметрів запиту за допомогою функції *valueOfParam*:

```
int valueOfParam(char param) {
        for (int i = 0; i < strlen(line1); i++) {
                if (line1[i] == param && line1[i+1] == '=') {
                        return (line1[i+2] — '0');
                }
        }
        return 0;
}
```

Функція *valueOfParam* очікує отримання параметра запиту у вигляді однієї єдиної цифри. Як виглядають ці параметри, можна побачити, якщо запустити Наприклад, відкрити сторінку в браузері та клацнути на кнопці *Update* (Оновити). Адреса URL-адреса в адресному рядку браузера зміниться, і в ньому з'являться параметри:

192.168.1.10/?0=1&1=0&2=0&3=0&4=0

Перелік параметрів починається після символу *?* Параметри мають вигляд *X=Y* та відокремлюються один від одного символом *&*. Зліва від знака = знаходиться ім'я параметра (у даному випадку цифри від 0 до 4), а праворуч - значення (у даному прикладі 1 означає "включено", а 0 - "вимкнено"). Для простоти параметри у цьому прикладі можуть мати лише одно символьні значення. Функція *setPinStates* встановлює стан цифрових виходів відповідно до значень елементів масиву *pinStates*.

Повертаючись до функції *sendBody*...після таблицею зі значеннями аналогових входів потрібно надіслати розмітку HTML з колекцією списків, що розкриваються, відповідно цифровим виходам. У кожному списку потрібно вибрати пункт *On* (Увімкнено) або *Off* (Вимкнено) залежно від поточного стану цифрового виходу. Для цього потрібно додати текст «*selected*» у значення, що відповідає Станом даного виходу в масиві *pinStates*. Код розмітки HTML для цифрових виходів полягає у форму, щоб відвідувач міг змінити значення у формі та, клацнувши на кнопці *Update* (Оновити), згенерувати новий запит до цієї сторінки з відповідними параметрами для встановлення цифрових виходів. А так буде виглядати розмітка HTML- сторінки:

```
<html><body>
<hq>Analog Inputs</h1>
<table border='1'>
<tr><td>A0</td><td>0.58 V</td></tr>
<tr><td>A1</td><td>0.63 V</td></tr>
<tr><td>A2</td><td>0.60 V</td></tr>
<tr><td>A3</td><td>0.65 V</td></tr>
<tr><td>A4</td><td>0.60 V</td></tr>
</table>
<h1>OUTPUT Pins</h1>

<form method='GET'>
<p>Pin 3<select name='0'>
<option value='0'>Off</option>
<option value='1' selected>On</option>
</select></p>
<p>Pin 4<select name='1'>
<option value='0' selected >Off</option>
<option value='1'>On</option>
</select></p>
<p>Pin 5<select name='2'>
<option value='0' selected >Off</option>
<option value='1'>On</option>
</select></p>
```

```
<p>Pin 6<select name='3'>
<option value='0' selected >Off</option>
<option value='1'>On</option>
</select></p>
<p>Pin 7<select name='4'>
<option value='0' selected >Off</option>
<option value='1'>On</option>
</select></p>
<Input type='submit'> value='Update'/>
</form>
</body></html>
```

3.3 Бібліотека Wifi[4]

Бібліотека Wifi дуже схожа на бібліотеку Ethernet. Якщо у скетчі замінити *Ethernet* на *Wifi*, *EthernetServer* на *WifiServer* та *EthernetClient* на *WifiClient*, решта коду залишиться майже незмінним. Головна відмінність бібліотеки Wifi від Ethernet, це – підключення до мережі.

Організація з'єднання

Насамперед потрібно імпортувати бібліотеку Wifi:

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

Щоб встановити з'єднання з мережею, слід викликати команду *Wifi.begin* та передати їй ім'я бездротової мережі та пароль:

```
Wifi.begin ("MY-NETWORK-NAME", "mypassword");
```

Особливі функції у бібліотеці Wifi

Бібліотека Wifi включає декілька спеціальних функцій. Вони перераховані у табл. 3.1.

Таблиця 3.1. Спеціальні функції у бібліотеці Wifi

Функція	Опис
Wifi.config	Встановлює статичні IP-адреси плати, сервера імен (**DNS**) та шлюзу
Wifi.SSID	Повертає рядок ідентифікатора бездротової мережі **SSID**
Wifi.RSSI	Повертає значення потужності сигналу типу **long**
Wifi.encriptionType	Повертає числовий код, який відповідає методу шифрування
Wifi.scanNetworks	Повертає кількість знайдених мереж, але жодної додаткової інформації про них не повертається
Wifi.MacAddress	Розміщує MAC-адресу адаптера Wifi у шестибайтний масив, що передається як параметр

Приклад використання Wifi

За приклад візьмемо попередній скетч, адаптувавши його для роботи з платою розширення Wifi. Відзначимо тільки відмінності від оригінальної версії. Щоб підключитися до бездротової точки доступу, потрібно вказати ім'я бездротової мережі та пароль:

---

[4] Повний опис бібліотеки Wifi можна знайти за адресою: https://www.Arduino.cc/reference/en/libraries/Wifi/

char SSID[] = "My network name"; // имя сети (SSID)
char pass[] = "mypassword"; // пароль для доступу к сети

Також потрібно змінити імена класів сервера та клієнта, *EthernetServer* та *EthernetClient*, на *WifiServer* та *WifiClient*:

WifiServer server(80);
WifiClient client;

При визначенні об'єкта сервера так само потрібно вказати номер порту – 80.

Наступна розбіжність між двома платами полягає в ініціалізації підключення. В даному випадку має використовуватися команда:

Wifi.begin(SSID, pass);

Інший код залишився майже без змін. У функції *loop* потрібно додати команду *delay(1)* перед зупинкою клієнта, вона дасть клієнту час завершити читання до того, як з'єднання буде закрито. У версії на основі бібліотеки Ethernet така затримка не потрібна. Також рекомендовано подекуди об'єднувати кілька викликів *client.print* в один, щоб кожен виклик виводив більш довгі рядки. Цей прийом збільшує швидкість взаємодій, тому що плата Wifi дуже неефективно обробляє короткі рядки. Але не треба забувайте, що кожен окремий виклик *client.print* чи *client.println* не може обробляти рядки довші за 90 байт – вони просто не будуть відправлені. Версія програми на основі бібліотеки Wifi працює помітно повільніше за версію на основі бібліотеки Ethernet та витрачає на завантаження близько 45 секунд. Плата розширення Wifi підтримує можливість зміни прошивки, і якщо у майбутньому розробники Arduino підвищать ефективність роботи плати Wifi, безперечно варто подумати про оновлення прошивки у своїй платі розширення.

4. ЛАБОРАТОРНІ РОБОТИ

Лабораторна робота №1 «Середовище розробки Arduino IDE»

Тема: Знайомство із середовищем розробки Arduino IDE

Мета: Знакомство з базовими функціями середовища розробки Arduino IDE

Інструменти для виконання роботи: Комп'ютер з підключенням до Internet; плата Arduino з підключенням через USB порт (Arduino Uno, Arduino Mega та ін.).

Теоретичні відомості.

Платформа **Arduino** – це сімейство мікроконтролерів на базі процесорів Atmel, STM и ARM. Усі мікроконтролери програмуються на мові C/C++ в середовище розробки **Arduino IDE**. Більшість плат Arduino (окрім деяких маленьких, таких як Micro, Pro Mini) мають ідентичне розміщення виводів (**піни, pins**) та дозволяють підключення уніфікованих сторонніх модулів, що звуться **шилдамі** (**Shield**). На всіх платах є набір цифрових та аналогових пінів, а також **інтерфейси SPI**[5] и **I²C**[6]. Для роботи зі сторонніми модулями середовище розробки має менеджер бібліотек, де зібрано найчастіше використовувані для Arduino бібліотеки.

Більшість мікроконтролерів мають підтримку **USB** (вбудовану або зовнішню як окремий перетворювач) та підключаються до операційної системи як **послідовний порт** (**COM** порт). COM порт - спеціальний порт для послідовної передачі даних між пристроями. Може бути апаратним (спеціальний COM роз'єм на системної платі ПК), або емулюється поверх іншого апаратного протоколу (наприклад, поверх USB, як у випадку з Arduino). Послідовний порт використовується для завантаження програми на мікроконтролер, а також може використовуватися для взаємодії ПК и програми для мікроконтролера (інша назва програми в Arduino IDE – "**скетч**").

Скетч для мікроконтролера Arduino уявляє собою файл з розширенням .ino, який містить код на мові C/C++. Код включає декілька основних блоків:

- Підключення файлів бібліотек.
- Оголошення глобальних змінних, констант, макроозначень.
- Оголошення функцій, структур та класів.

Функція **setup()** – викликається один раз на початку роботи програми, саме тут потрібна пройти ініціалізація контролера – налаштування пінів, запуск послідовного порту, SPI и I²C інтерфейсів.

Функція **loop()** – викликається мікроконтролером в нескінченному циклі, тут відбувається основна робота.

Після завантаження на мікроконтролер програма зберігається навіть після від'єднання Arduino, до тих пір, поки вона не буде перезаписана новою програмою.

---

[5] https://ru.wikipedia.org/wiki/Serial_Peripheral_Interface
[6] https://ru.wikipedia.org/wiki/I%C2%B2C

Виконання роботи.

1) Встановлення середовища розробки:

Завантажити потрібну версію середовища розробки для встановлення на ПК (місце знаходження файлів та версію середовища вказує викладач, в прикладах розглядається версія IDE 1.8.2). Запустити на виконання програму встановлення середовища розробки.

2) Підключення плати до середовища розробки:

Підключити плату до ПК USB кабелем. В середовище розробки обрати потрібну версію плати (зазвичай на платі вказано її найменування):

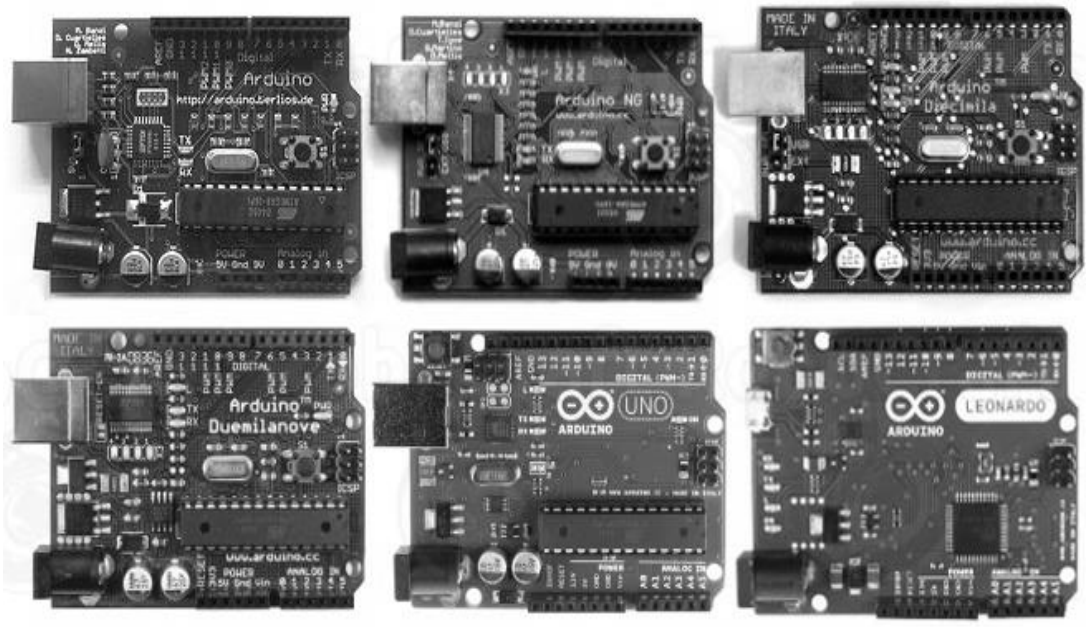


Рис. 4.1 Плати Arduino

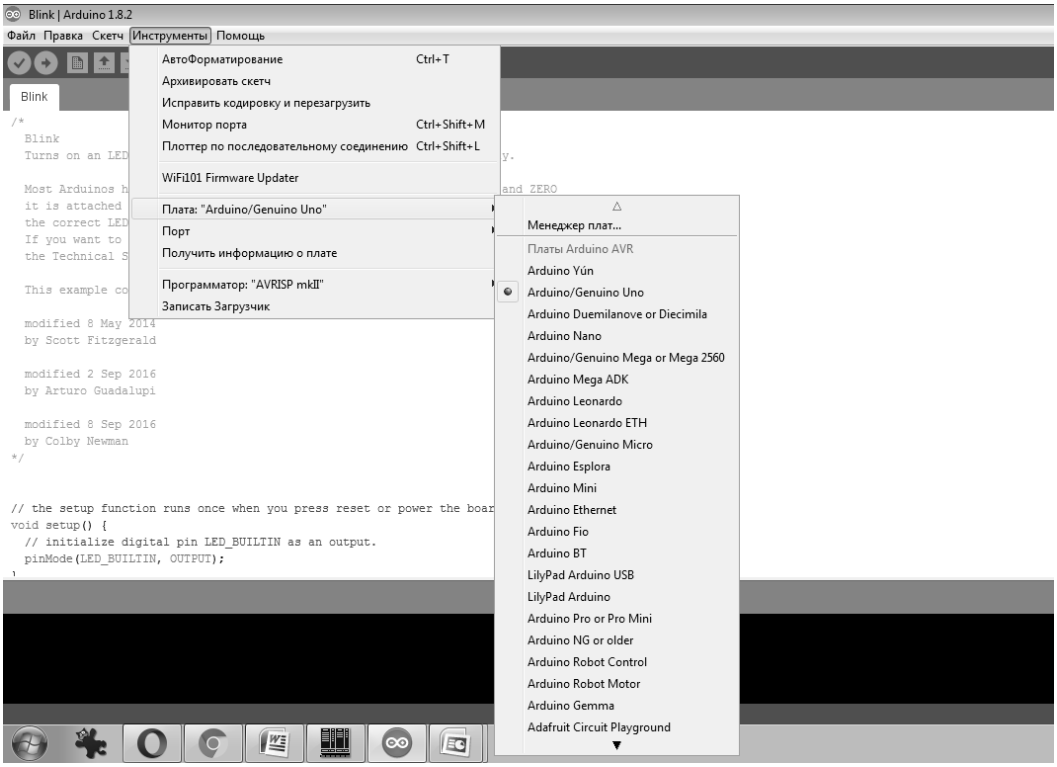


Рис. 4.2 Обрання версії плати Arduino в середовище IDE

Обрати потрібний порт:

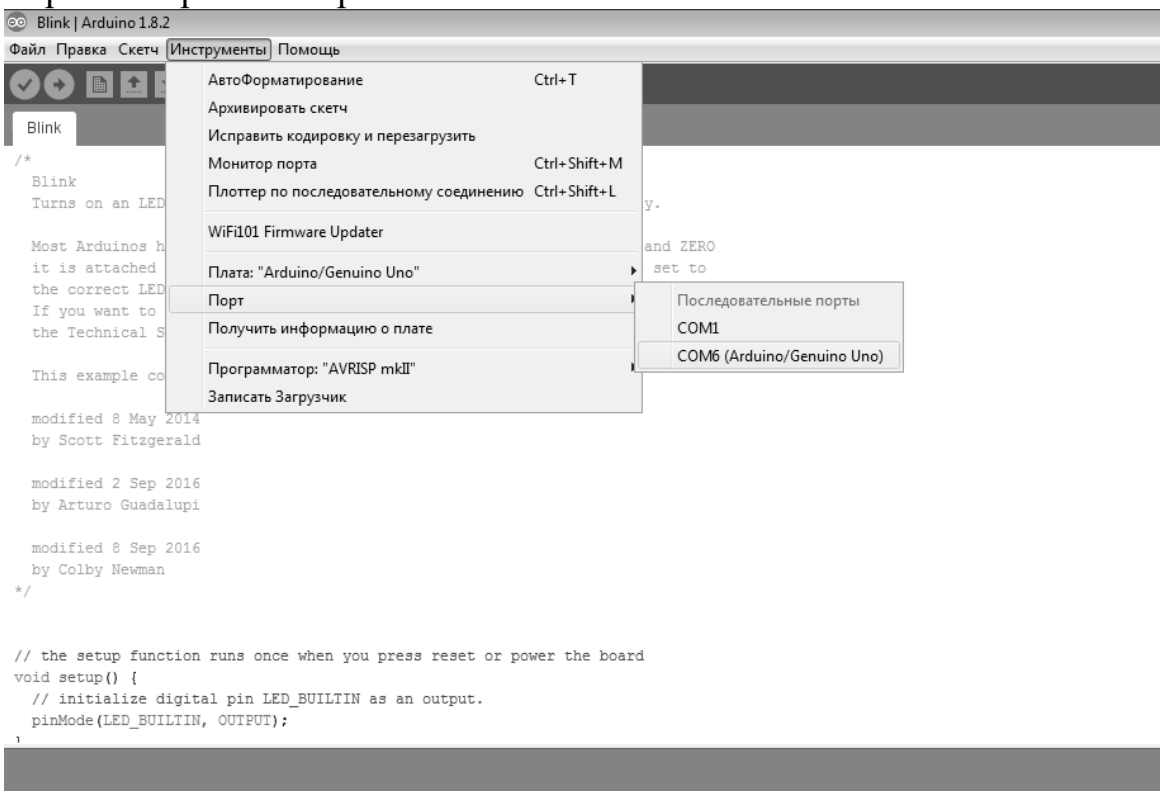


Рис. 4.3 Обрання версії порту для роботи з платою Arduino в середовище IDE

В ОС Windows порти відображаються з назвами виду COM1, COM2, и т.д.). Якщо викликати пункт меню "*Инструменты -> Получить информацию о плате (Get Board info)*", можна перевірити: чи потрібна плата обрана.

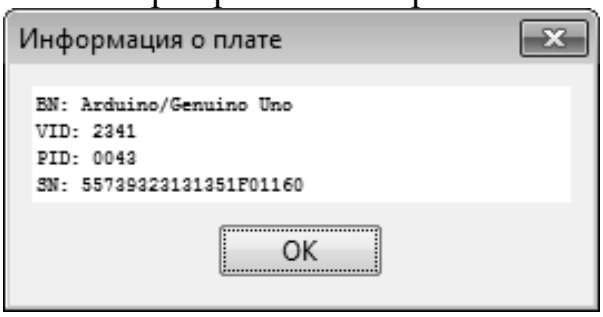


Рис. 4.4 Отримання інформації о платі Arduino

3) Завантаження тестового приклада:

Завантажте скетч Blink зі стандартних бібліотек прикладів (*Файл -> Примеры -> 0.1Basics -> Blink*). Перевірте обраний порт та плату. Натисніть кнопку *Загрузить* () або оберіть пункт меню "*Скетч -> Загрузка*".

Після успішного завантаження в ніжній частині вікна середовища з’явиться напис: «*Загрузка завершена*» а на платі почне мерехтить світлодіод.

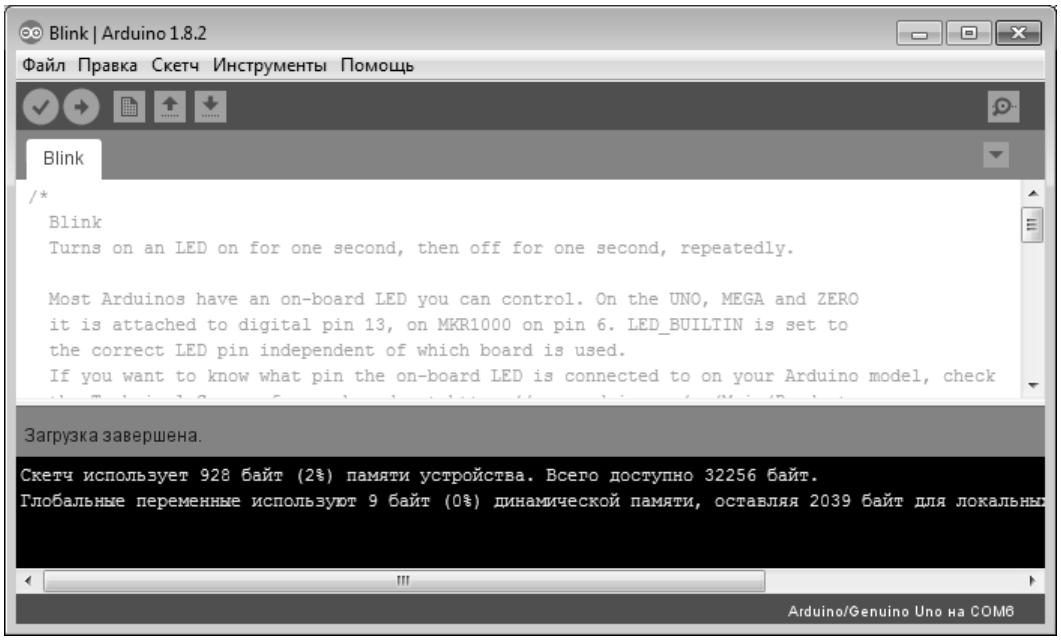


Рис. 4.5 Стандартний скетч Blink

4) Керування бібліотеками:

Іноді, в процесі роботи, виникає потреба в підключенні сторонніх бібліотек, наприклад, для роботи з різноманітними датчиками. Бібліотека уявляє собою набір заголовкових файлів (з розширенням *.h*) та файлів з кодом (розширення *.c* або *.cpp*). У разі підключення бібліотеки в скетч додаються підключення потрібних заголовкових файлів (рядок виду *#include <...>*). Наприклад, підключімо стандартну бібліотеку *Mouse*. Обираємо "*Скетч -> Подключить библиотеку -> Mouse*". При цьому IDE автоматично добавить зміни в скетч, що е необхідними для підключення бібліотеки. У верхньої частині скетча з'явиться рядок: *#include <Mouse.h>*.

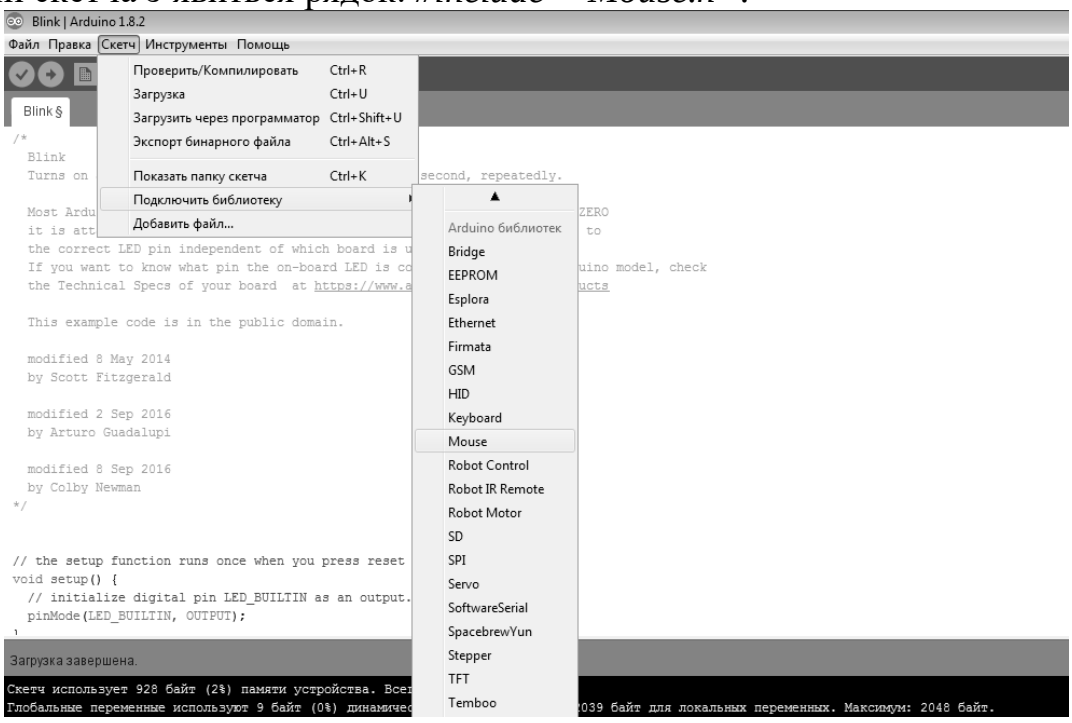


Рис. 4.6 Підключення стандартних бібліотек

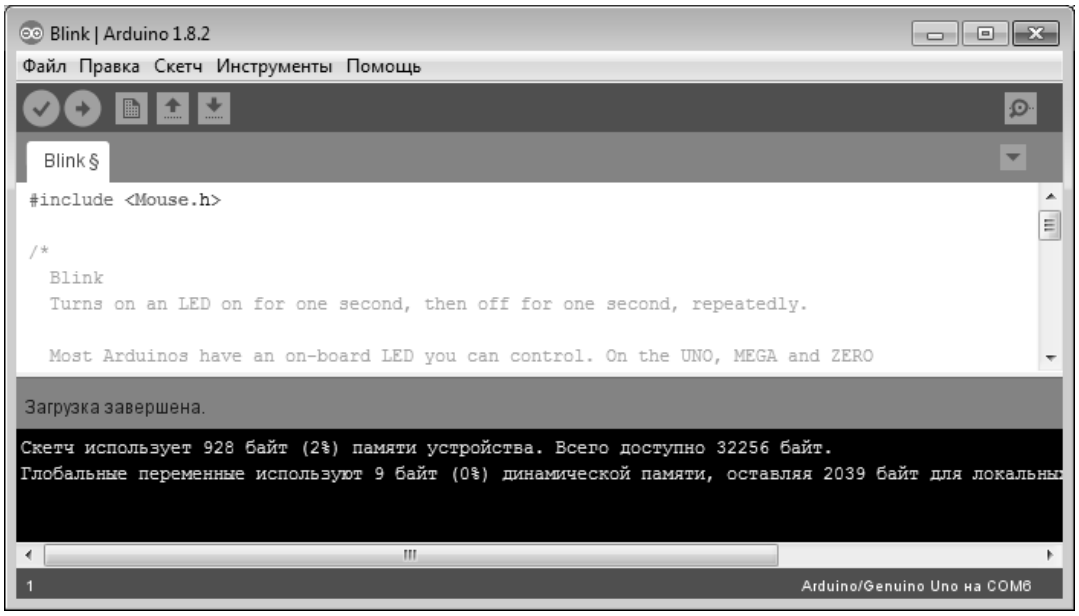


Рис. 4.7 Підключення стандартних бібліотек в скетчі

Одразу після встановлення в середовище розробки вже доступний базовий набір бібліотек. Інші бібліотеки спочатку потрібно завантажити в IDE, для цього використовується менеджер бібліотек: *"Скетч -> Подключить библиотеку -> Управление библиотеками"*. В менеджері присутні бібліотеки для роботи с різноманітними модулями/протоколами і т.п. Після завантаження бібліотеки вона становиться доступної для підключення к скетчу.

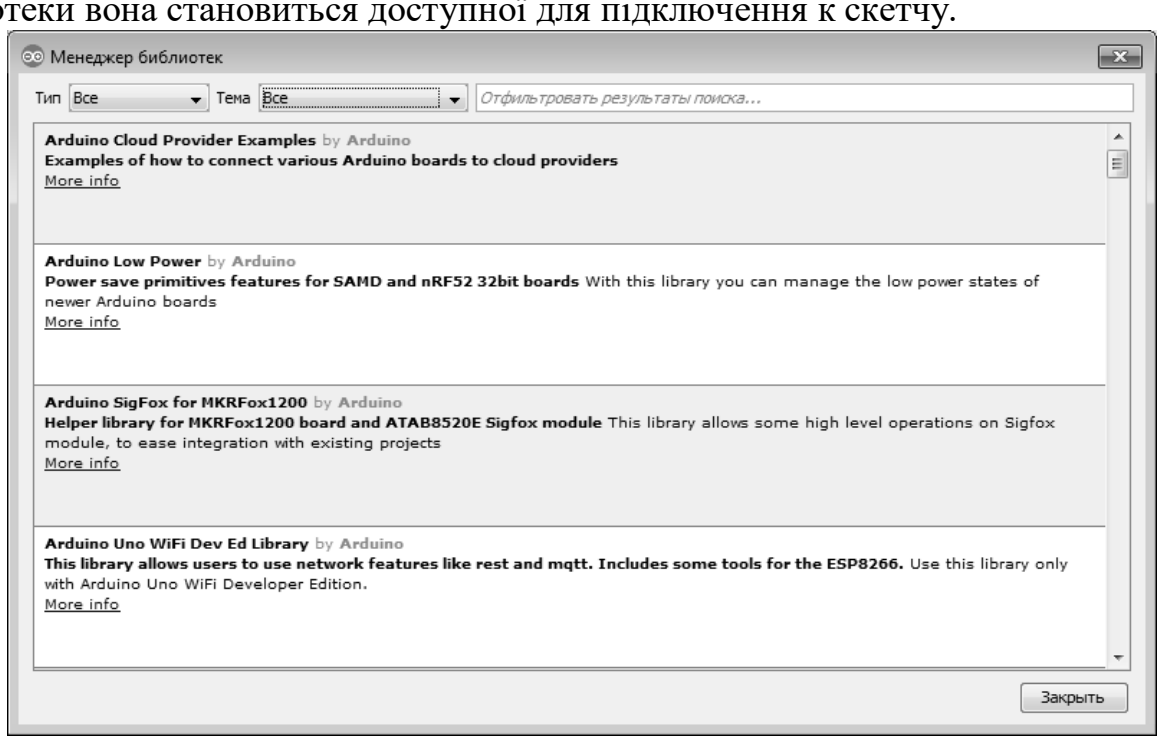


Рис. 4.8 Менеджер бібліотек

В верхній частині менеджера знаходяться два поля фільтрації по типу і темі, а також поле текстового пошуку за назвою та описом бібліотеки. Якщо обрати бібліотеку з переліку з’явиться кнопка "*Установка*" (для деяких бібліотек також можна обрати версію, якщо е декілька версій).

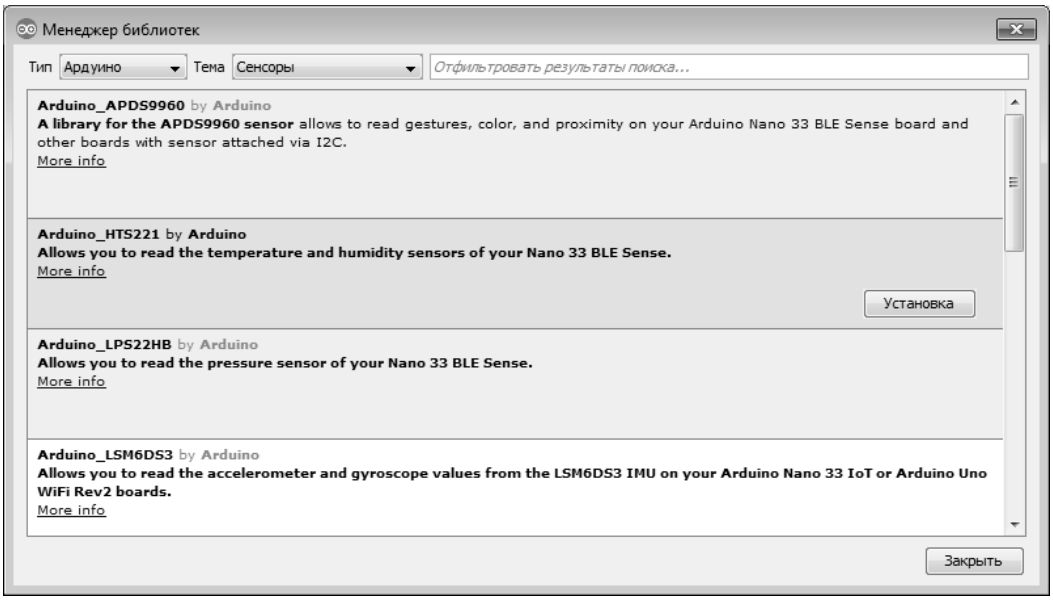


Рис. 4.9 Встановлення бібліотек через менеджер бібліотек

Завдання: знайдіть в менеджері бібліотек бібліотеки для роботи з датчиками температури (встановлювати не потрібно, сформуйте лише перелік)

Огляд ресурсів по ARDUINO:

https://www.Arduino.cc/ – офіційний сайт
https://www.Arduino.cc/en/Main/Software – середовище розробки Arduino IDE
https://www.Arduino.cc/en/Reference/HomePage – довідник по мові розробки
http://Arduino-diy.COM/ – багато інформації з підключення до Arduino різноманітних модулів, приклади проектів на Arduino

Лабораторна робота №2 «Світлодіоди»

Тема: Керування світлодіодом.

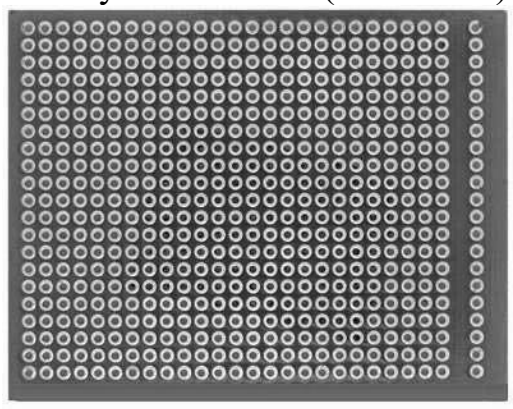
Мета: Знайомство з принципами розробки найпростіших електронних схем на базі монтажної плати (підключенням напівпровідникових пристроїв к мікроконтролеру і написання коду для керування світлодіодами).

Інструменти для виконання роботи: Комп'ютер з встановленим середовищем розробки Arduino IDE; плата Arduino з підключенням через USB порт (Arduino Uno, Arduino Mega та ін.); монтажна плата BreadBoard; дроти (перемички типу "П-П"); кнопка (перемикач); світлодіоди; резистори 10 кОм і 220 Ом; потенціометр (змінний резистор 10 кОм).

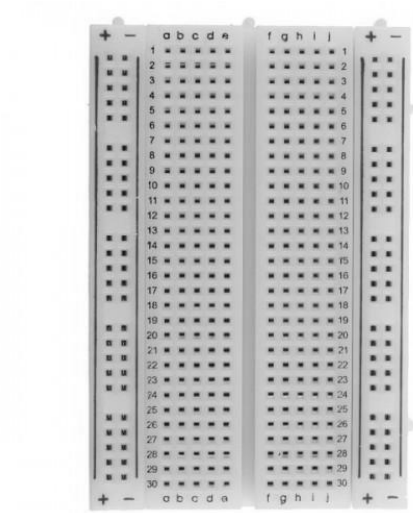
Теоретичні відомості.

Цифрові піни на мікроконтролерах Arduino можуть працювати в трьох режимах: INPUT, INPUT_PULLUP і OUTPUT. В режимі OUTPUT можна програмно встановлювати на піні цифровий сигнал за допомогою функції **digitalWrite**. Для встановлення рівня використовуються зумовлені константи HIGH і LOW. Прийнято, що HIGH позначає високий рівень напруги (3,3 або 5 вольт в залежності від робочої напруги плати), а LOW - низький (близько 0 вольт). При цьому можливе живлення пристроїв зі споживанням струму до 40 мА. В режимі INPUT можна подавати цифровий сигнал на пін і програмно зчитувати його в мікроконтролері за допомогою функції **digitalRead**. При зчитуванні логічного значення буде отримано значення HIGH, якщо рівень поданого на пін напруги вище 2 вольт для 3,3 вольтової плати і 3 вольт для 5 вольтової плати. Для встановлення режиму роботи цифрового піна використовується функція **pinMode**.

Для розробки простих електронних схем часто використовуються **макетні** (**монтажні**) плати (**BreadBoard**). Вони бувають двох основних видів: плати з металізованими отворами для пайки (Рис. 4.10а) і макетні плати без застосування пайки (Рис. 4.10б).



а) для пайки



б) без застосування пайки

Рис. 4.10 Макетні плати

В лабораторних роботах передбачається використання макетних плат без застосування пайки. Контакти цієї плати з'єднані всередині особливим чином, як показано на малюнку:

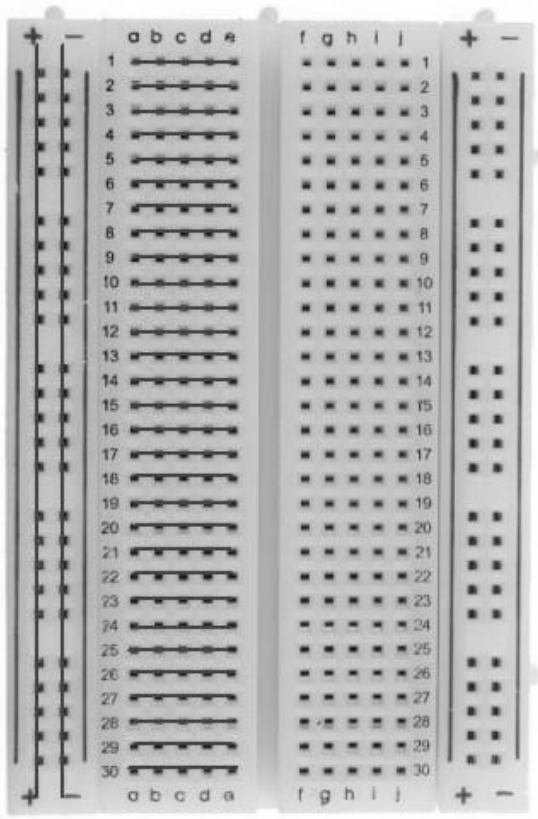


Рис. 4.11 Контакти макетної плати

Потенціометр – це змінний резистор з регульованим опором. При обертанні ручки потенціометра його опір змінюється від 0 до деякого максимального значення.

АЦП (аналогово-цифровий перетворювач) - мікросхема, що перетворить аналогове значення напруги в цифрове. На платах Arduino присутні кілька вбудованих АЦП, доступних програмно за допомогою функції analogRead (), що приймає номер порту АЦП (A0 - A5). Так як в Arduino використовуються 8-розрядні АЦП, на виході виходять значення від 0 до 1024 для напруги на вході, відповідно, від 0 до 5 В.

ШІМ (PWM, широтно-імпульсна модуляція) - процес генерації прямокутних імпульсів з мінливою шириною імпульсу.

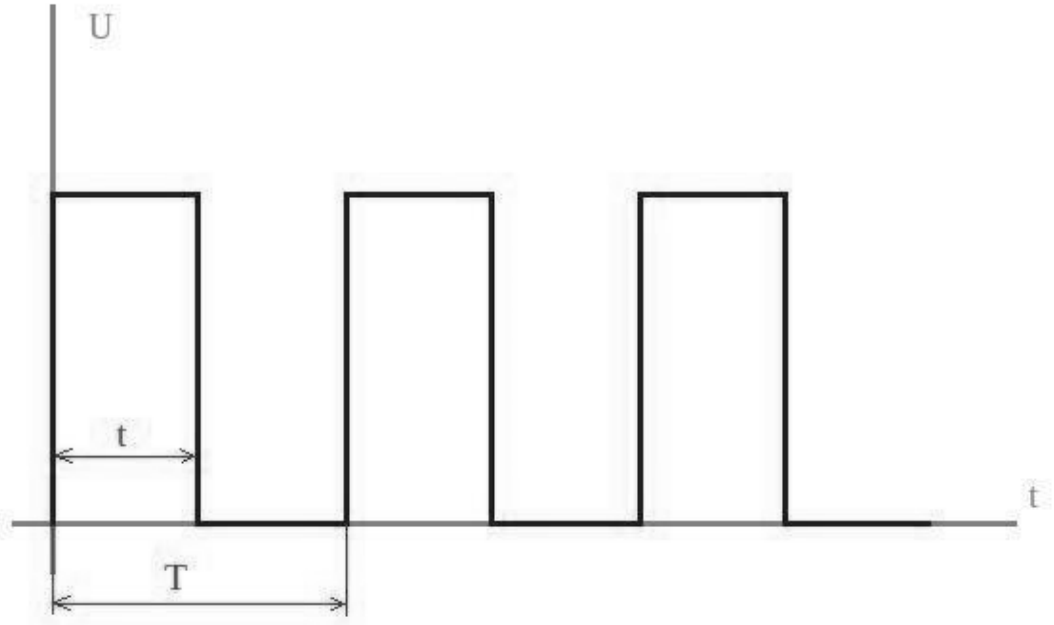


Рис. 4.12 Генерація прямокутних імпульсів

T – період ШІМ (в Arduino використовується 490 Гц, іноді буває 980 Гц)

t - тривалість імпульсу, змінюється в межах від 0 до T

Ставлення t / T називається **прогальність імпульсу** (англ. *Duty cycle*, рос. *Скважность*). На мікроконтролерах Arduino ШІМ доступна на деяких цифрових пінах, при цьому запускається вона за допомогою функції **analogWrite ()**. Функції передається параметр, який приймає значення від 0 до 255, яких регулює тривалість імпульсу від 0 до T, і відповідно прогальність імпульсу від 0 до 1.

Світлодіод – напівпровідниковій пристрій, який трансформує електрику в світло (ефект, відомий як електролюмінесценція). Довга ніжка на світлодіоді позначає "+", а коротка – "-".

Рис. 4.13 Зображення червоного світлодіоду.

Кнопка виглядає наступним чином:

Рис. 4.14 Зображення кнопки.

У кнопки є особливість – прямий контакт працює як перемикач запирання, тобто від початку вона буде передавати сигнал. А ось перехресне під'єднання працює як перемикач відпирання, тобто натиснута кнопка передає сигнал далі.

Виконання роботи.

1) Керування світлодіодом за допомогою кнопки.

Підключити світлодіод та кнопку так, як вказано на схемі (Рис. 4.15).

Послідовно зі світлодіодом в схему додається резистор 220 Ом, а кнопку через резистор 10 кОм потрібно «посадити на землю».

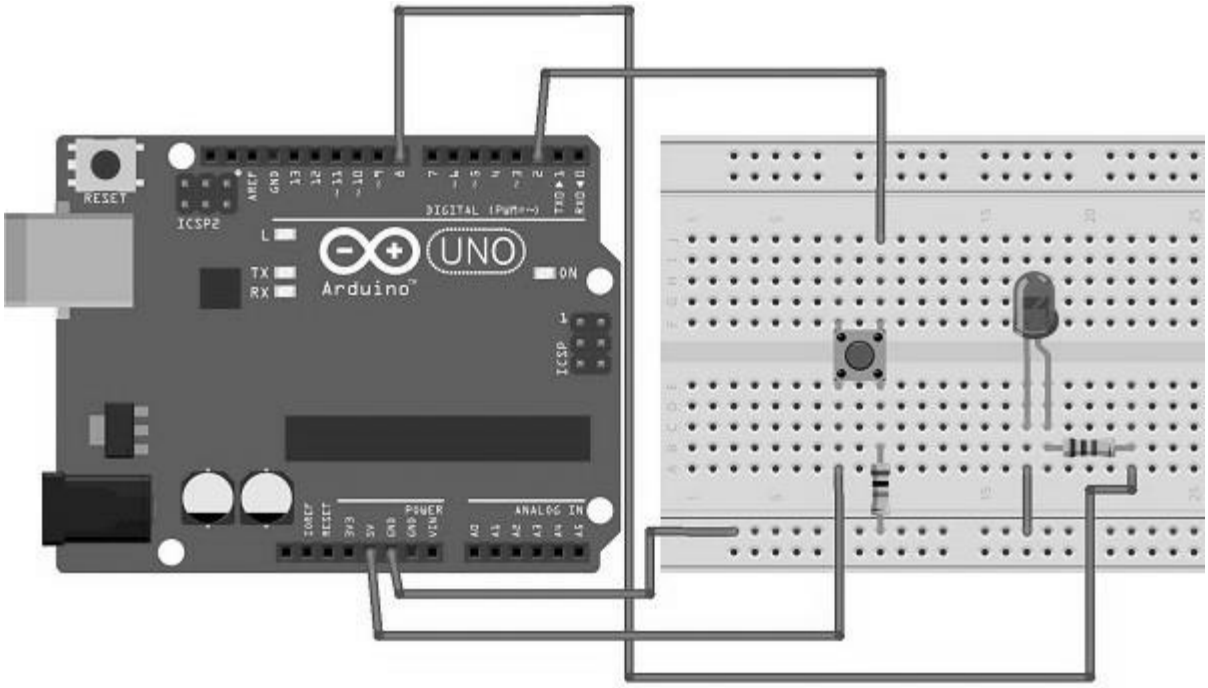


Рис. 4.15 Схема керування світлодіодом за допомогою кнопки.

Завантажте код:

```
const int button = 2; //на пин 2 вешаем кнопку
int led = 8; //на пин 8 вешаем диод
void setup() {
pinMode(led, OUTPUT); //настраиваем работу кнопки: пин "на выход"
pinMode(button, INPUT); //настраиваем работу диода: пин "на вход"
}
void loop() {
 if (digitalRead(button) == HIGH) //проверка нажатия кнопки
        {
         digitalWrite(led, HIGH); //если нажата - включаем диод
        }
else
        {
         digitalWrite(led, LOW); //если нет - диод выключен
        }
//digitalWrite(led, digitalRead(button)); // а так можно обойтись одной строчкой
}
```

2) Розробка скетчу для мерехтіння світлодіода.

Створити новий скетч і зберегти його з назвою Led. Перевірити, що обрана правильна плата і порт.

Для управління світлодіодом через цифровий пін потрібно виконати 3 дії:

1. Підключити діод на потрібний цифровий пін (в цьому прикладі буде задіяний 4 пін плати Arduino - інше в схемі керування можна залишити без змін)

2. Налаштувати цифровий пін в функції setup ():

pinMode (4, OUTPUT);

3. В функції loop змінити стан піна з затримкою таким чином, щоб отримати ефект мерехтіння:

digitalWrite (4, HIGH);
delay (1000);
digitalWrite (4, LOW);
delay (1000);

Ефект мерехтіння отримано за допомогою використання функції **delay (ms),** де ms – кількість мілісекунд[7], на які необхідно призупинити програму.

3) Розробка коду для керування яскравістю світлодіода за допомогою потенціометра.

Зібрати схему з світлодіодом і потенціометром на макетної платі так, як вказано на схемі:

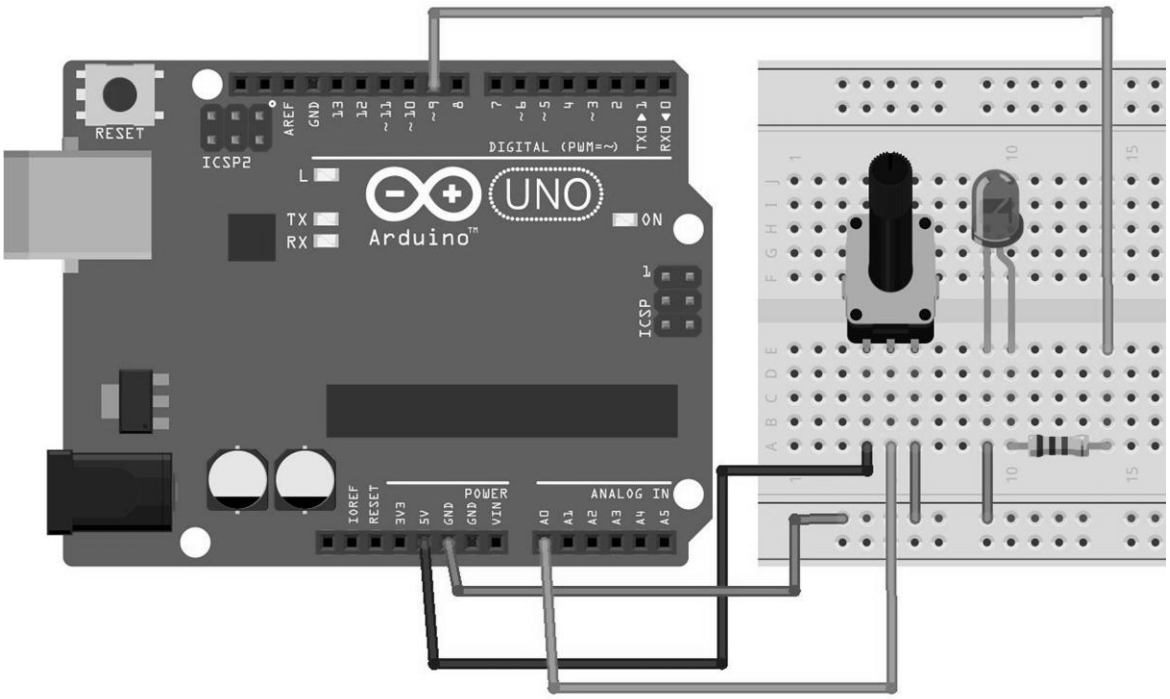


Рис. 4.16 Схема керування світлодіодом за допомогою потенціометра.

Світлодіод потрібно підключити до порту плати з підтримкою ШІМ (позначені символом "~", в нашому прикладі це пін 9). Потенціометр підключає крайніми контактами до Vcc (живлення 5 В) та Gnd на платі, середнім контактом до аналогового входу, наприклад, до A0.

Створити новий скетч і зберегти його з назвою LedBrightness (Для

[7] В 1 секунді - 1000 мілісекунд.

використання ШІМ і АЦП на Arduino не потрібна додаткова налаштування пінів, єдине, що потрібно – в функції loop зчитувати значення з аналогового входу і встановлювати відповідну прогальність імпульсу).

```
int led=9; // даём имена пинов со светодиодом
int pot= A0 // и потенциометром
void setup() {
        pinMode(led, OUTPUT); // пин со светодиодом — выход
        pinMode(pot, INPUT); // пин с потенциометром - вход
}
void loop() {
        int x; // объявляем переменную x
        // считываем напряжение с потенциометра:
        // будет получено число от 0 до 1023
        // делим его на 4, получится число в диапозоне
        // 0-255 (дробная часть будет отброшена)
        x = analogRead(pot) / 4;
        analogWrite(led, x); // выдаём результат на светодиод
}
```

Або також можна одним рядком:

```
analogWrite(9,analogRead(A0)/4);
```

Завантажте скетч і перевірте, що при обертанні ручки потенціометра яскравість світлодіода змінюється від 0 до деякого максимального значення.

Завдання: зібрати схему та створити програму керування трьома діодами за певним алгоритмом роботи (наприклад світлофор, або ялинкова гірлянда).

Контрольні питання

1) Які режими роботи у цифрових пінів на мікроконтролерах Arduino?
2) Опишіть роботу функцій digitalWrite( ) та digitalRead( ).
3) Яке призначення функцій analogWrite ( ) та analogRead ( )?
4) Опишіть роботу функції delay ( ).
5) Що виконує рядок *pinMode (5, OUTPUT);*?

Лабораторна робота №3 «Клас Serial»

Тема: Взаємодія з контролером Arduino через послідовний порт плати з використанням класу Serial.

Мета: Навчитися приймати та передавати дані з комп’ютера в плату Arduino через послідовний порт використовуючи середовище розробки Arduino IDE.

Інструменти для виконання роботи: Комп’ютер з встановленим середовищем розробки Arduino IDE; плата Arduino з підключенням через USB порт (Arduino Uno, Arduino Mega та ін.); монтажна плата BreadBoard; дроти (перемички типу “П-П"); світлодіоди; резистори 10 кОм і 220 Ом.

Теоретичні відомості.

Послідовний порт (Serial pORt, COM-порт) — двонаправлений послідовний інтерфейс, призначений для обміну байтовою інформацією. Послідовний тому, що інформація через нього передається по одному біту, біт за бітом (на відміну від паралельного порту, що передає відразу всі біти даних одночасно кількома паралельними каналами). Найчастіше для послідовного порту персональних комп'ютерів використовується стандарт *RS-232*[8]. Раніше послідовний порт використовувався для підключення терміналу, пізніше для модема або миші. Зараз він використовується для з'єднання з джерелами безперебійного живлення, для зв'язку з апаратними засобами обчислювальних систем.

Хоча деякі інші інтерфейси комп'ютера (такі як Ethernet, FireWire і USB) також використовують послідовний спосіб обміну, назва «Послідовний порт» закріпилася за портом, що має стандарт *RS-232*.

Послідовний інтерфейс UART (universal asynchronous receiver/transmitter) – тип асинхронного приймача-передавача, компонентів комп'ютерів та периферійних пристроїв, що передає дані між паралельною та послідовною формами. Уявляє собою логічну схему, з одного боку підключену до шини обчислювального пристрою, а з іншого має два або більше виводи для зовнішнього з'єднання. UART звичайно використовується спільно з іншими комунікаційними стандартами, такими як EIA RS-232.

UART це зазвичай окрема мікросхема чи частина мікросхеми, що використовується для з'єднання через комп'ютерний чи периферійний послідовний порт. UART нині загалом включені в мікроконтролери. Багато сучасних мікросхем сьогодні випускаються з можливістю комунікації в синхронному режимі, такі прилади називають USART.

[8] https://ru.wikipedia.org/wiki/RS-232

Рис. 4.17 Формат послідовного коду даних UART

Кожен біт передається за рівні проміжки часу. Час передачі одного біта визначається швидкістю передачі. Швидкість передачі вказується в **бодах** (біт в секунду). Крім бітів даних інтерфейс UART вставляє в потік біти синхронізації: стартовий і стоповий. Таким чином, для передачі байта інформації потрібно 10 бітів. Похибка тимчасових інтервалів передачі бітів повинна бути не більше 5% (рекомендується не більше 1,5%).

Існують варіанти з різною кількістю бітів даних, бітів синхронізації, може бути доданий біт контролю парності і т.п. Але ці формати використовуються рідко. Головне:

- в неактивному режимі вихід UART знаходиться в високому стані;
- передача байта починається зі стартового біта (низького рівня);
- передача байта закінчується стоповим бітом (високого рівня);
- дані передаються молодшим бітом вперед;
- для передачі байта потрібно 10 бітів;
- час передачі одного байта розраховується виходячи з швидкості передачі і кількості бітів (10).

Часто використовуються такі стандартні швидкості передачі інтерфейсу UART наведено в табл. 4.1.

Табл. 4.1. Швидкості передачі інтерфейсу UART

швидкість передачі, бод	час передачі одного бита, мкс	час передачі байта, мкс
4800	208	2083
9600	104	1042
19200	52	521
38400	26	260
57600	17	174
115200	8,7	87

Обмін інформацією через UART відбувається в дуплексному режимі, тобто передача даних може відбуватися одночасно з прийомом. Для цього в інтерфейсі UART є два сигнали:

- TX - вихід для передачі даних;
- RX - вхід для прийому даних.

При з'єднанні двох UART пристроїв вихід TX одного пристрою

з'єднується з входом RX іншого. А сигнал TX другого UART підключається до входу RX першого.

Послідовний інтерфейс UART в Arduino.

Будь-яка плата Arduino має, як мінімум, один апаратний послідовний інтерфейс UART. Тобто в контролері існує електронний вузол, в регістр якого програма тільки завантажує байт для передачі, а формування сигналів обміну і всі інші операції робить цей вузол. Може бути реалізована і програмна передача даних по протоколу UART. В цьому випадку всі сигнали формуються програмою (і це займає ресурси процесора).

Плата Arduino UNO має один порт UART, сигнали якого підключені до пінів 0 (сигнал **RX**) і 1 (сигнал **TX**). Сигнали мають логічні рівні TTL (0 ... 5 В). Через ці контакти (0 і 1) можна підключити до плати інший пристрій, що має інтерфейс UART.

Крім функції зв'язку з іншими контролерами порт UART плати Arduino UNO використовується для завантаження в контролер програми з комп'ютера. Для цього до цих же сигналів (RX і TX) підключені відповідні виводи мікросхеми ATmega16U2 - перетворювача інтерфейсу USB/UART (мікросхема перетворювача підключена через резистори опором 1 кОм).

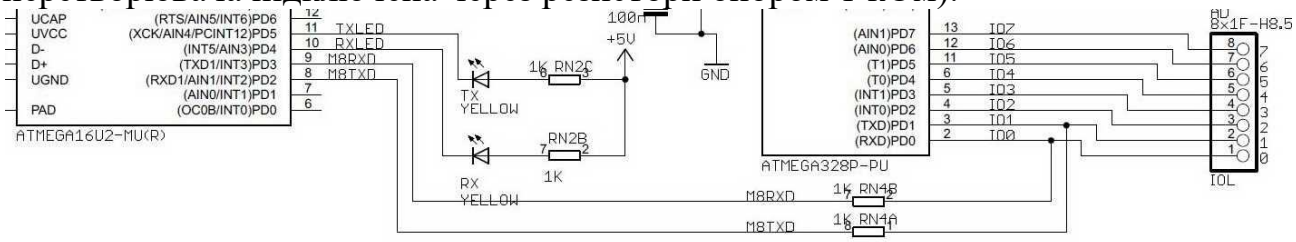


Рис. 4.18 Фрагмент схеми плати Arduino UNO R3

Таким чином, при вільних пінах 0 і 1 плати Arduino сигнали з мікросхеми ATmega16U2 надходять на контролер ATmega328. А якщо до плати підключити зовнішнє UART пристрій, то його сигнали будуть мати пріоритет, тому що ATmega16U2 підключена через резистори.

Перетворювач інтерфейсу ATmega16U2 дозволяє підключати плату Arduino до комп'ютера через USB порт. На комп'ютер встановлюється драйвер. Він створює на комп'ютері віртуальний COM порт. Через нього і відбувається обмін.

Бібліотека Serial для роботи з UART Arduino.

Для роботи з апаратними UART контролерами в Arduino існує вбудований клас **Serial**. Він призначений для керування обміном даними через UART.

Через послідовний інтерфейс дані завжди передаються в двійковому коді. В класі Serial дані можуть передаватися в двох форматах:

• як бінарний код;

• як ASCII символи.

Наприклад, монітор послідовного порту в програмі Arduino IDE приймає дані як ASCII текст. Для того, щоб він вивів на екран комп'ютера число "65" треба передати коди символів "6" і "5". А код "65" монітор відобразить як

символ "A".

Основні функції класу Serial[9]

1. Serial.begin(speed, config)

Параметри:

- **speed:** швидкість в бітах на секунду (бодах) **– long;**
- **config:** задає кількість біт даних, перевірку парності та стопові біти**.**

Задає швидкість передачі даних по послідовному інтерфейсу в бітах в секунду (бодах). 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).

Другий аргумент цієї функції необов'язковий. Він дозволяє налаштувати кількість біт даних, перевірку парності і степових біти. За замовчуванням, посилка складається з 8 біт даних, без перевірки парності, з одним стоповим бітом.

Serial.begin(9600); //встановлює послідовно з'єднання на швидкості 9600 бод.

2. Serial.available()

Параметри: нема.

Значення, що повертаються: кількість байт, доступних для зчитування.

Повертає кількість байт (символів) доступних для зчитування з буфера послідовного порту. Під символами розуміються дані, які вже прийняті і зберігаються в послідовному приймальному буфері (максимум 64 байта)

3. Serial.Read()

Параметри: нема.

Значення, що повертаються: перший байт прийнятих даних (або -1, якщо таких нема) – **int**.

Опис: зчитує дані, що надходять по послідовному інтерфейсу.

4. 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 обов'язково.

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».

Якщо потрібно передати в монітор порту не ASCII код числа, а саме число, то є можливість це зробити за допомогою функції **Write ().**

5. Serial.Write(uint8_t c)

[9] Більш детально http://Arduino.ru/Reference/Serial

Виклик:
Serial.Write(val);
Serial.Write(str);
Serial.Write(buf, len);

Функція записує дані в послідовний порт. Дані надсилаються як байт або послідовність байт.

Параметри:

- **val:** змінна для передачі, як єдиний байт**;**
- **str**: рядок для передачі, як послідовність байт;
- **buf**: масив для передачі, як послідовність байт
- **len:** довжина масиву.

Виконання роботи.

1) Зібрати схему керування трьома діодами відповідно до завдання в лабораторної роботі №2.
2) Підключити схему до живлення.
3) Завантажити скетч *Monitor_pORta-pruklad.ino* в мікроконтролер Arduino:

```
//char data = 0;   // переменная для хранения полученного байта
int data = 0;   // переменная для хранения полученного байта

void setup() {
        Serial.begin(9600); // устанавливаем последовательное соединение
}
void loop() {
        if (Serial.available() > 0) {  //если есть доступные данные
        // считываем байт
        data = Serial.Read();
        // отсылаем то, что получили
        Serial.print("я получил: ");
        Serial.println(data,DEC);
        Serial.print("А я получил: ");
        Serial.Write(data);
        }
}
```

4) Перевірити правильність роботи програми.

Завдання: змінити програму керування трьома діодами по команді комп'ютера (початковий стан роботи світлодіодів контролера згідно завдання до лабораторної роботи №2 або згідно табл. 4.2).

Контролер Arduino повинен виконувати три команди:

1. Команду перевірки з'єднання між комп'ютером і контролером (на певний символ контролер повинен відповісти підтвердженням).

2. Команду управління світлодіодами. При отриманні якої встановлюється певний режим (приклади режимів роботи наведено в табл. 4.2). Отримання команди підтвердити.

3. Команду виключення світлодіодів, при отриманні якої світлодіоди вимикаються.

Команди складаються з одного символу.

Табл. 4.2. Режими роботи світлодіодів

№ варіанта	**Режим до отримання команди**		**Режим після отримання команди**	
	Червоний світлодіод	**Зелений світлодіод**	**Червоний світлодіод**	**Зелений світлодіод**
1	ввімкнений постійно	мигтить	мигтить	ввімкнений постійно
2	вмикаються-вимикаються протифазно		ввімкнений постійно	мигтить
3	ввімкнений постійно	ввімкнений постійно	вмикаються-вимикаються протифазно	
4	мигтить	ввімкнений постійно	вмикаються-вимикаються одночасно	
5	вмикаються-вимикаються одночасно		ввімкнений постійно	ввімкнений постійно
6	мигтить	ввімкнений постійно	ввімкнений постійно	мигтить

Контрольні питання

1) Який клас та функції застосовують для роботи з послідовним портом плати Arduino?
2) Опишіть основні особливості функції Serial.available().
3) Зазначте параметри функції Serial.print( ).
4) Які дві характеристики має функція Serial.begin( )?

Лабораторна робота №4 «Датчики»

Тема: Отримання даних з датчиків.

Мета: Дослідити роботу різноманітних датчиків.

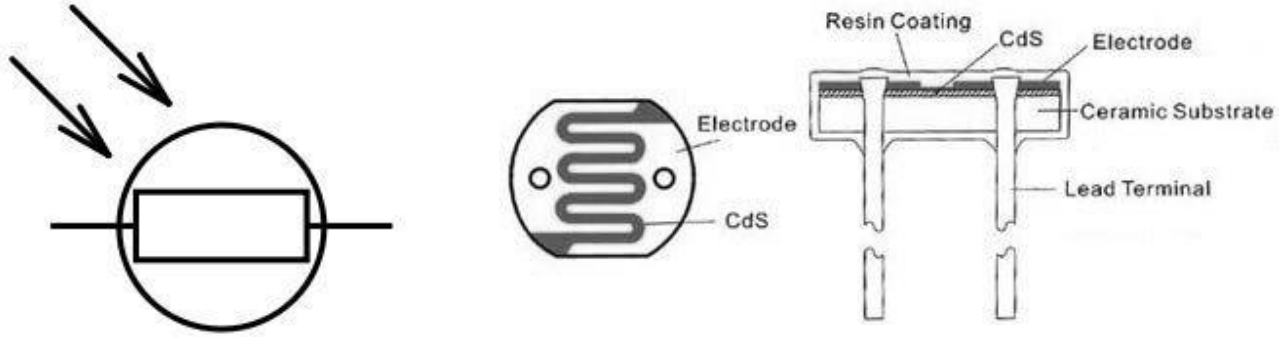
Інструменти для виконання роботи: комп'ютер з встановленим середовищем розробки Arduino IDE; плата Arduino з підключенням через USB порт (Arduino Uno, Arduino Mega та ін.); монтажна плата BreadBoard; дроти (перемички типу "П-П"); світлодіоди; резистори 220 Ом; фоторезистор, датчики вимірювання відстані, температури і вологості.

Теоретичні відомості.

Датчик (давач, сенсор) – засіб вимірювання у вигляді конструктивної сукупності одного або декількох вимірювальних перетворювачів величини, що вимірюється і контролюється. Він виробляє вихідний сигнал, зручний для дистанційного передавання, подальшого перетворення, обробки і (або) зберігання, але не піддається безпосередньому сприйняттю спостерігачем. Інакше кажучи, датчики дозволяють отримувати інформацію про навколишнє середовище і передавати її в зручному вигляді.

Так як датчиків існує дуже багато, то вибір конкретного датчика залежить від задачі, що розв'язується на даний час в певних умовах.

Фоторезистор – це напівпровідниковий прилад, опір якого змінюються в залежності від освітлена його чутливої поверхні (зміна опору під впливом світлового потоку називається фоторезистивним ефектом).



а) умовне графічне позначення б) принцип дії

Рис. 4.19 Фоторезистор

Принцип дії полягає в наступному: між двома провідними електродами знаходиться напівпровідник, коли напівпровідник не освітлений – його опір великий (до одиниць МОм). Коли ця область освітлена її провідність різко зростає, а опір відповідно падає. Як напівпровідника можуть використовуватися такі матеріали як: сульфід Кадмію, Сульфид свинцю, Гіпс Кадмію та інші. Від вибору матеріалу при виготовленні фоторезистора залежить його спектральна характеристика.

У фоторезисторів немає p-n переходу, тому не має значення, в якому напрямку протікає струм. Перевірити фоторезистор можна за допомогою мультиметра в режимі вимірювання опору, вимірявши опір освітленого і затемненого елемента.

Приблизну залежність опору від освітленості можна бачити на графіку (рис. 4.20 а). Тут показано, як змінюється струм при певній напрузі в залежності від кількості світла, де Ф = 0 – темрява, а Ф3 – яскраве світло. На наступній діаграмі (рис. 4.20 б) наведено зміна струму при постійній напрузі, але змінюється освітленості.

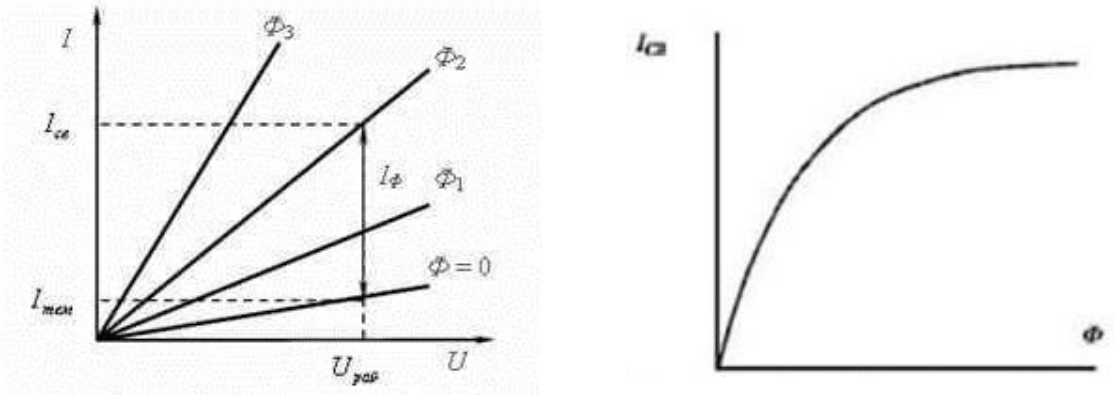


Рис. 4.20 Залежність опору та струму фоторезистора від освітленості

Фоторезистори мають певну інерційність, тобто його опір змінюється не моментально після опромінення світловим потоком, а з невеликою затримкою. Цей параметр називається гранична частота. Це частота синусоїдального сигналу, що модулює світловий потік через елемент, при якій чутливість елемента знижується в $\sqrt{2}$. Швидкодія компонентів зазвичай лежить в межах десятків мікросекунд. Таким чином, використання фоторезистора в схемах, де потрібна швидка реакція обмежена.

У промисловості і побутовій електроніці фоторезистори використовуються для вимірювання освітленості, підрахунку кількості чого-небудь, визначення перешкод та іншого. Основне його призначення – переводити кількість світла, що потрапляє на чутливу площу, в корисний електричний сигнал. Сигнал надалі може оброблятися аналогової, цифрової логічною схемою або схемою на базі мікроконтролера.

Реалізація пристрою на базі ARDUINO UNO з фоторезистором, за допомогою якого можна відстежити слабкий світловий потік представлена на рис. 4.21 (фоторезистор комутується з платою ARDUINO через резистор 10 кОм, а світлодіод – 220 Ом).

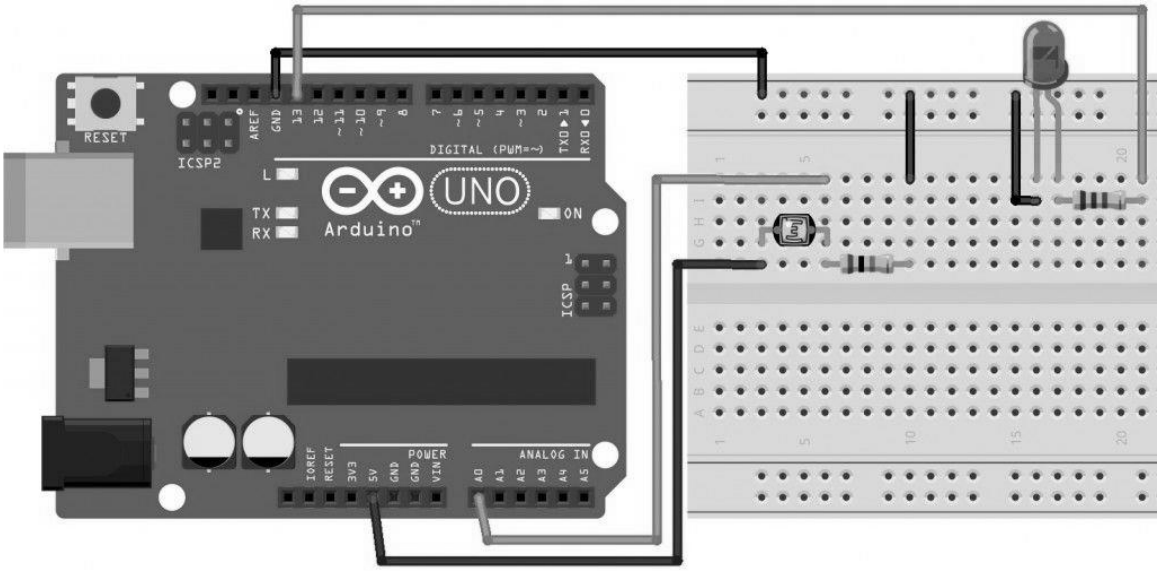


Рис. 4.21 Макет схеми «день»/«ніч» з фоторезистором та світлодіодом

Приклад коду, що демонструє роботу макета:

```
#define led 13          //змінна led – зберігає номер піна, через який керується світлодіод
#define ldr A0          //змінна ldr – пін, через який знімають показники фоторезистора
void setup() {
    /*вказуємо режим роботи піна зі світлодіодом НА ВИХІД, фоторезистор підключено
    до порту analog in, що заздалегідь говорить о том, що він в режимі INPUT (ВХІД),
    тому не прописується */
      pinMode(led, OUTPUT);
}
void loop() {
    /*якщо показник освітленості менш ніж 800, то світлодіод вмикається (на пін з
    номером led буде виставлено 5В – значення HIGH), інакше світлодіод вимкнено (на пін
    з номером led буде виставлено 0В – значення LOW) */
      if (analogRead(ldr) < 800) {
            digitalWrite(led, HIGH);
      }
      else {
            digitalWrite(led, LOW);
      }
}
```

Датчик температури LM35 – це інтегральний датчик температури з широким діапазоном температур, доволі високою точністю вимірювання та каліброваним виходом по напрузі. LM35 забезпечує вимір температури з точністю ± 0,25° C в кімнатних умовах і з точністю ± 0,75° C в повному діапазоні робочих температур -55 ... +150° C, без зовнішньої калібрування або підгонки вихідної напруги. Вихід напруги LM35 збільшується на 10мВ на градус Цельсія підвищення температури. LM35 може працювати від джерела живлення 5В, а напруга струму менше 60 мкА. У зв'язку з тим, що датчик споживає струм лише 60 мкА, у нього дуже низький рівень власного розігріву, менше ніж 0,1° C при нерухомому повітрі.

З недоліків можна виділити погіршення параметрів при видаленні на значну відстань. В цьому випадку джерелами перешкод можуть стати

радіопередавачі, реле, перемикачі та інші пристрої. Також існує проблема, коли температура вимірюваної поверхні і температура навколишнього середовища сильно розрізняються. В цьому випадку датчик показує середнє значення між двома температурами.

а) б)

Рис. 4.22 Датчик LM35

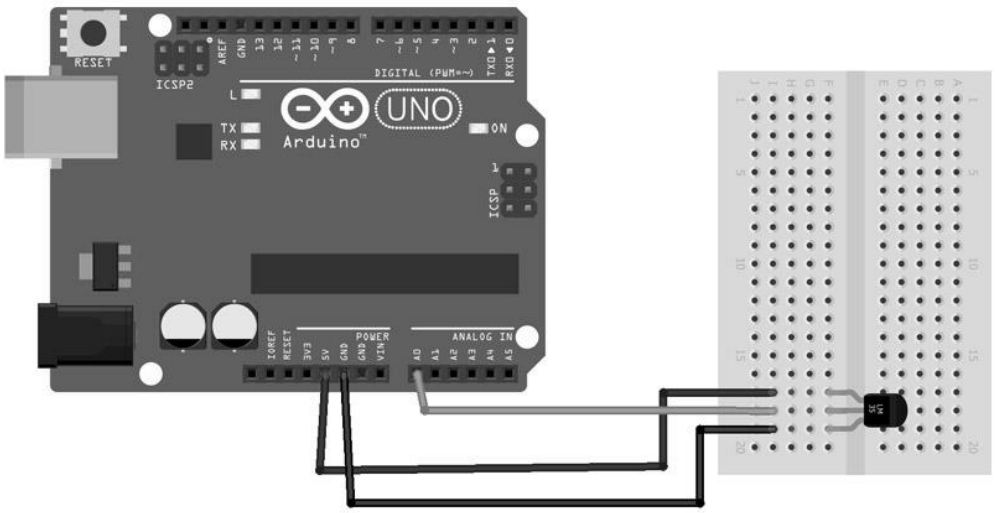


Рис. 4.23 Схема підключення датчика LM35 до плати ARDUINO UNO

Приклад коду, що демонструє роботу датчика:

```
float tempC;                    //змінна для зберігання значення температури
int Reading;                    //змінна для значень датчика
#define tempPin  0              //пін, з якого отримуємо значення датчика
void setup() {
        Serial.begin(9600);     //встановлює швидкість обміну по послідовному порту 9600
}
void loop() {
        Reading = analogRead(tempPin);   //отримуємо покази датчика
        tempC = Reading/1024*5*100;      //перетворюємо покази в градуси
        Serial.print("Температура: ");
        Serial.print (tempC);
        Serial.println ("*C");
        delay(1000);            //інтервал запиту на покази сенсора
}
```

Датчик DHT – це цифровий датчик температури і вологості, що дозволяє калібрувати цифровий сигнал на виході. Складається з ємнісного датчика

вологості і термістора. Також, датчик містить в собі АЦП для перетворення аналогових значень вологості і температури. Алгоритм спілкування сенсора з контролером наступний:

1. Мікроконтролер посилає запит на показники та змінює сигнал з «0» на «1»;
2. Датчик отримує запит та відповідає зміною бітового сигналу з «0» на «1»;
3. У випадку, коли на запит буде отримане підтвердження, датчик видає мікроконтролеру пакет даних в розмірі 5 байт (в двох перших байтах температура, в третьому і четвертому вологість, п'ятий байт - контрольна сума для виключення помилок вимірювання).
4. Після передачі пакета даних датчик переходить в сплячий режим до наступного запиту з боку мікроконтролера.

Завдяки тому, що сенсор робить вимірювання тільки за запитом, досягається достатня енергоефективність (датчик споживає струм лише 100 мкА в режимі очікування).

Існують дві версії сенсорів DHT. Виглядають вони майже однаково. Основні відмінності - в технічних характеристиках.

Табл. 4.3. Технічні характеристики сенсорів **DHT**

	DHT11	**DHT22**
Визначення вологості:	20 – 80% RH ± 5% (макс.)	0 – 100% RH ± 5% (макс.)
Визначення температури:	0 – 50ºC ± 2% (макс.)	-40 – 125ºC ± 0,5% (макс.)
Частота опитування:	не більше 1 Гц	до 0,5 Гц
Розміри:	15,5 x 12 x 5,5 мм	15,1 x 25 x 7,7 мм
Живлення:	3,5 – 5,5 В	
Струм споживання (макс.):	2,5 мА (режим передавання), 100мкА (режим очікування)	
4 виводи з відстанню між контактами 0,1"		

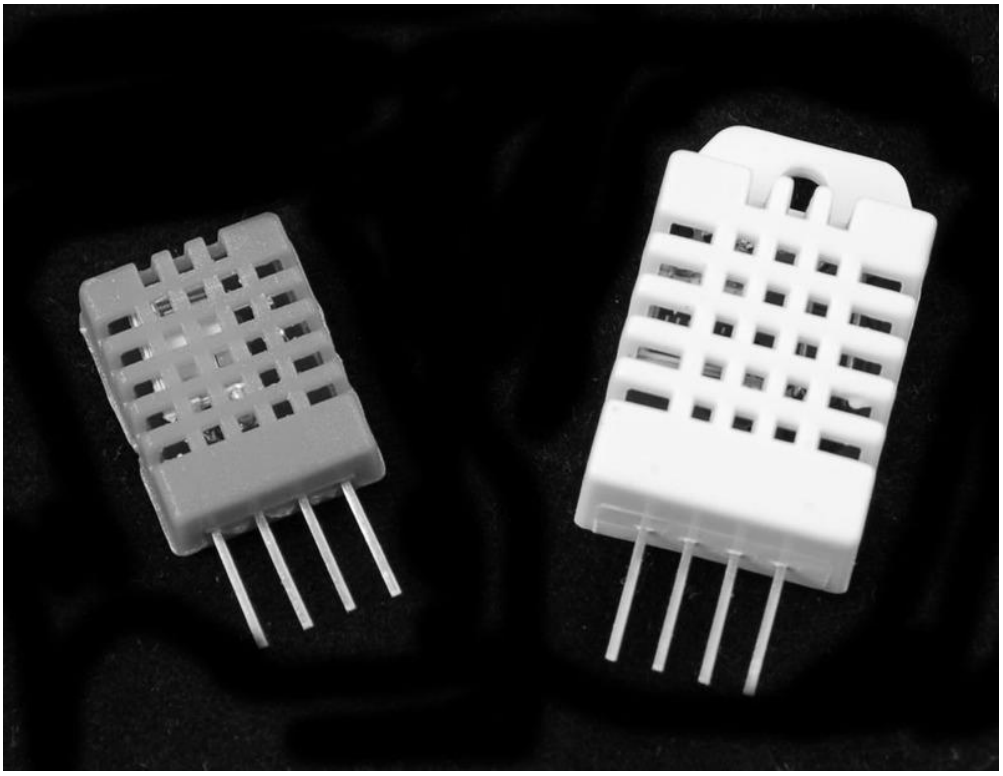


Рис. 4.24 Зовнішній вигляд сенсорів DHT11 та DHT22

Виводи:

1. VCC (3 – 5 В живлення)
2. Data Out - Вивід даних
3. NC - не використовується
4. Загальний

Для роботи з датчиком використовують бібліотеки, наприклад **dht.h**, що містить декілька функцій:

1. **dht DHT;** ініціалізується робота сенсора**;**
2. **DHT.Read**NN**(dht_pin);** зчитуваються показники датчика, де NN – тип датчика (11 – DHT11; 22 – DHT22); dht_pin – номер піну, до якого підключений контакт Data Out;
3. **DHT.humidity** повертає значення температури повітря по Цельсію;
4. **DHT.temperature** повертає значення вологості повітря.

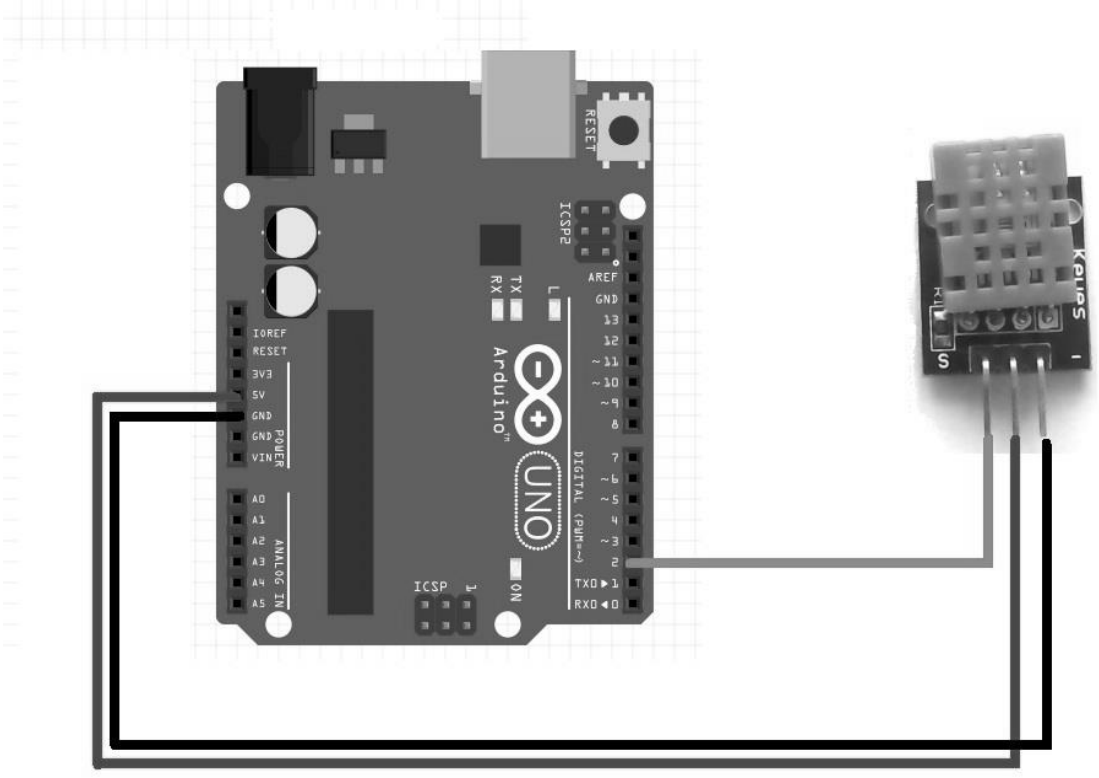


Рис. 4.25 Схема підключення датчика DHT11 до плати ARDUINO UNO

Приклад коду:

```
// Оголошення змінних:
#include  <dht.h>
#define dht_pin A0                    //номер аналогового піна, до якого підключений датчик
dht DHT;                              //Ініціюємо датчик
void setup(){
        Serial.begin(9600);
        delay(1000);
}
void loop(){
        DHT.Read11(dht_pin);          //считываем значення датчика (тип DHT11)
        Serial.print("влажность воздуха: ");
        Serial.print(DHT.humidity);
```

```
        Serial.print("%  ");
        Serial.print("температура воздуха: ");
        Serial.print(DHT.temperature);
        Serial.println("C  ");
        delay(2000);                    //затримка 2 секунди мім вимірюванням
}
```

Ультразвуковий далекомір HC-SR04 – прилад безконтактного типу, що забезпечує високоточне вимірювання і стабільність. На його роботу істотно не впливають електромагнітні випромінювання і сонячна енергія. Але, так як в основу принципу дії покладено ультразвук, то такий датчик не підходить для визначення відстані до об'єктів, що поглинають звук. Оптимальними для вимірювання є предмети з рівною гладкою поверхнею.

Рис. 4.26 Далекомір HC-SR04

Дія датчика відстані HC-SR04 заснована на принципі ехолокації. Він випромінює звукові імпульси в простір і приймає відбитий від перешкоди сигнал. По часу відбивання звукової хвилі до перепони і назад визначається відстань до об’єкта. Відбиття звукової хвилі починається з подачі імпульсу довжиною не менше 10 мікросекунд на вивід **TRIG** далекоміра. Як тільки імпульс закінчується, далекомір випромінює в простір перед собою пачку звукових імпульсів частотою 40 кГц. В цей же час на виводі **ECHO** встановлюється логічна одиниця. Як тільки датчик приймає відбитий сигнал, на виводі **ECHO** встановлюється логічний нуль. По довжині логічної одиниці на виводі **ECHO** («Затримка відлуння» на рис. 4.27) визначається відстань до перешкоди.

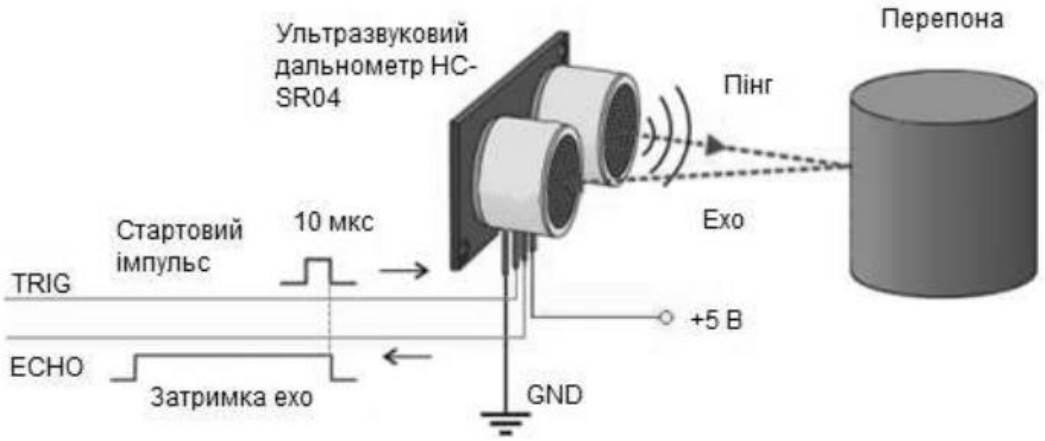


Рис. 4.27 Принцип роботи ультразвукового далекоміра

Діапазон вимірів відстані далекоміра HC-SR04 – до 4 метрів з мінімальною градацією шкали в 0,3 см. Кут спостереження – 30°, ефективний кут – 15°. Струм живлення в режимі очікування 2 мА, при роботі – 15 мА.

Виводи:

1. VCC – живлення;
2. TRIG – вхід для подачі строба запуску;
3. ECHO – вивід для зняття імпульсу у відповідь;
4. GND – земля.

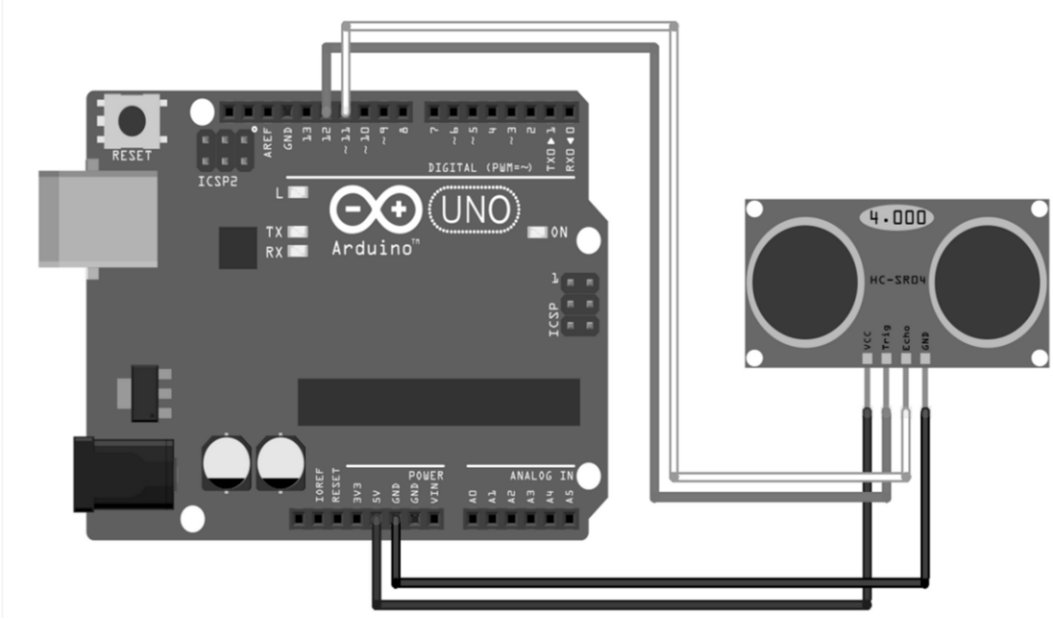


Рис. 4.28 – Схема підключення далекоміра до плати ARDUINO UNO

Виводи **TRIG** и **ECHO** – підключені до виводів 12 и 11 плати Arduino. В кожному повторенні циклу loop() обраховується дистанція і її значення виводиться в порт.

Спочатку генеруємо імпульс запуску[10], потім створюємо 10-мікросекундний імпульс, який є тригером для початку випромінювання

[10] Якщо датчик відстані не виконує зчитування сигналу, то перетворення вихідного сигналу ніколи не прийме значення короткого імпульсу - LOW. Тому цикл починається з «примусового» рядка digitalWrite(trigPin, LOW);

далекоміром звукового пакета в простір. Коли імпульси надійдуть то перепони, вони віддзеркаляться і будуть прийняти на виході **ECHO.** Далі потрібно запам’ятати час від початку передачі звукової хвилі до її повернення та користаючись цім значенням, розрахувати дистанцію до об’єкта.

Відстань дорівнює швидкість помножити на час: S = V×t. Швидкість звуку в повітрі 340 м/сек, час в мікросекундах ми знаємо (змінна **duratuion**). Щоб отримати час duratuion в секундах, необхідно розділити його на 1 000 000. Так як звук проходить подвійну відстань – до об’єкта і назад – треба ще розділити результат на два, тобто відстань до об’єкта складає: S = 34000 см/сек × duration / 1 000 000 сек / 2 = 1,7 см/сек / 100.

Для роботи буде використана функція pulseIn(), вона 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 вхід/вихід буде LOW. Функція pulseIn () повертає довжину сигналу в мікросекундах. Функція повертає 0, якщо протягом заданого часу (таймаута) не було зафіксовано сигналу. Функція може вимірювати сигнали довжиною від 10 мікросекунд до 3 хвилин.

Виклик:

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

Параметри:

- **pin**: номер порту вхід/вихід, на якому очікується сигнал. (int);
- **value**: тип імпульсу, що очікується (HIGH або LOW**);**
- **timeout** (не обов’язковий): час очікування сигналу (таймаут) в мікросекундах; за замовченням – одна секунда. (unsigned long)

Скетч для визначення та виведення відстані і в послідовний порт:

```
const int TRIGPin = 12;
const int ECHOPin = 11;
long distance, duration; //змінні для зберігання дистанції до об’єкта та часу затримки

void setup() {
// ініціалізація послідовного порту
        Serial.begin(9600);
//визначаємо вхід/вихід
        pinMode(TRIGPin, OUTPUT);       // тригер - вихідний пін
        pinMode(ECHOPin, INPUT);        // ехо – вхідний пін
}
void loop() {
//Визначення часу затримки
        digitalWrite(TRIGPin, LOW);
        delayMicroseconds(2);       //спочатку генеруємо короткий імпульс 2 мксек
        digitalWrite(TRIGPin, HIGH);        // генеруємо імпульс запуску
/* Виставивши високий рівень сигналу, чекаємо близько 10 мікросекунд. У цей момент
датчик буде посилати сигнали з частотою 40 КГц */
        delayMicroseconds(10);
        digitalWrite(TRIGPin, LOW);
/*визначаємо на піні ECHOPin довжину рівня HIGH (мксек), це час затримки
```

```
акустичного сигналу на ехолокаторі */
        duration = pulseIn(ECHOPin, HIGH);
// Визначення дистанції до об’єкта
        distance = duration* 1.7 * 0.01; //розраховуємо дистанцію з датчика до об’єкта в см
        Serial.println(distance); // виводимо значення дистанції в послідовний порт
        delay(200); //затримка між вимірюваннями за для коректної роботи
}
```

Для полегшення роботи з датчиком відстані HC SR04 на Arduino можна використовувати бібліотеки. Наприклад скетч з використанням бібліотеки <Ultrasonic.h> виглядає так:

```
#include  <Ultrasonic.h>
#define CM 1
#define INC 0
Ultrasonic ultrasonic (12, 11);     //підключаємо вхідний та вихідний сигнали

void setup() {
        Serial.begin(9600);    // ініціалізація послідовного порту
}
void loop() {
        float dist_cm = ultrasonic.Ranging (CM);
        Serial.println (dist_cm); //виводимо значення дистанції в послідовний порт
        delay(1000); //затримка між вимірюваннями за для коректної роботи
}
```

Конструктор Ultrasonic приймає два параметри: номера пінів, до яких підключені контакти TRIG та ECHO відповідно. Бібліотека має один єдиний метод **Ranging**, як параметр якому задається, у що перераховувати відстань до об'єкта: в сантиметри або в дюйми.

Для використання бібліотеки в проектах потрібно розташувати її в папці бібліотеки каталогу інсталяції Arduino.

Виконання роботи.

1) Зібрати макети відповідно схеми (по черзі перевіряються схеми згідно рис. 4.1, рис. 4.23, рис. 4.25 та рис. 4.28).
2) Підключити схему до живлення.
3) Завантажити програму в мікроконтролер Arduino (до кожного макета відповідний скетч).
4) Перевірити правильність роботи програми (по черзі перевірити роботу скетчів до кожного макета).

Завдання: створити скетч програми, що буде використовувати функції контролю параметрів температури, вологості, освітлення та відстані (наприклад «теплиця» або «музей»).

Контрольні питання

1) Що таке датчик (сенсор)?
2) Яка залежність опору на фоторезисторі від освітленості?

3) Який алгоритм роботи має програма для зчитування показників датчика температури LM35?
4) Призначення та підключення датчика DHT.
5) Опишіть основні функції бібліотеки **dht.h.**
6) Призначення та основні характеристики далекоміра HC-SR04.
7) Які методи має бібліотека Ultrasonic?

Лабораторна робота №5 «Переривання**»**

Тема: Переривання в Arduino.

Мета: Дослідити роботу переривань в контролерах Arduino.

Інструменти для виконання роботи: комп'ютер з встановленим середовищем розробки Arduino IDE; плата Arduino з підключенням через USB порт (Arduino Uno, Arduino Mega та ін.); монтажна плата BreadBoard; дроти (перемички типу "П-П"); світлодіоди; резистори 220 Ом; фоторезистор, датчики вимірювання відстані, температури і вологості.

Теоретичні відомості.

У реальній програмі необхідно одночасно здійснювати багато різних дій. Крім того часто необхідно проводити реакцію на зовнішні події, вимірювати інтервали часу, тощо. Всі такі операції виконуються циклічно і паралельно, із різними періодами циклів, жодну із них не можна призупинити.

Зручним методом роботи із такими подіями буде режим переривань. У такому режимі за сигналом запиту переривання робота основної програми призупиняється, а управління передається підпрограмі обслуговування переривання. Після обслуговування переривання управління передається основній програмі і вона виконується із місця зупинки. Для основної програми виконання підпрограми обслуговування переривань невидиме. Короткий час роботи такої підпрограми не буде впливати на виконання основної програми.

Контролери ATmega мають багато різних джерел переривань частина котрих використана у середовищі Arduino. Наприклад можна так настроїти систему, щоб кожні 2 мс викликалась підпрограма обслуговування переривання таймера 2, що запускає підпрограму користувача. Встановлення режиму і часу спрацювання таймера Arduino проводиться зазвичай через апаратні регістри мікроконтролера. Для полегшення програмування існує безліч бібліотек, що полегшують використання переривань.

Наприклад бібліотека **MsTimer2** призначена для конфігурування апаратного переривання від таймера 2 мікроконтролера. Вона має всього три функції:

MsTimer2 :: set (unsigned long ms, void (*f) ( ))

Функція встановлює час переривання у мс. З таким періодом буде викликатись функція обробник переривання **f( )**. Вона повинна бути об'явлена як void (не повертає нічого) та не мати аргументів.

MsTimer2 :: start() – функція дозволяє переривання від таймера.

MsTimer2 :: sTop() – функція забороняє переривання від таймера.

Простий приклад з паралельною обробкою сигналу можна розглянути на вже знайомої нам кнопки (див. лабораторну роботу №2). В скетчі, що виконує аналогічні функції, але використовує переривання, функція **setup( )** задає час циклу переривання за таймером 2 мс і вказує ім'я обробника переривання **timerInterrupt**. Функція обробки сигналу кнопки **button1.scanState( )** викликається в обробнику переривання таймера кожні 2 мс.

Таким чином, стан кнопки оброблюється паралельним процесом. А в

основному циклі програми перевіряється ознака кліка кнопки та змінюється стан світлодіода. Сам скетч, за умови використання бібліотек, виглядає досить легким та зрозумілим.

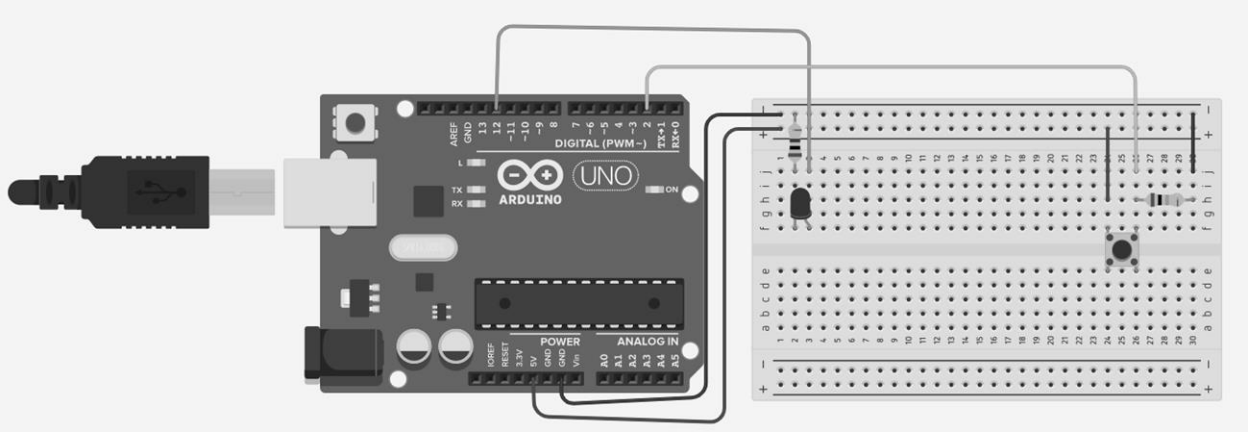


Рис. 4.29 Схема керування світлодіодом за допомогою кнопки

Приклад коду_1:

```
//  Нажатие на кнопку меняет состояние светодиода
#include  <MsTimer2.h> // подключаем библиотеку обработеки прерывания от таймера
#include  <Button2.h> // подключаем библиотеку работы с кнопкой (см. лекцію)
#define LED 6     // светодиод подключен к выводу 6
#define BUTTON 2  // кнопка подключена к выводу 2
Button button(BUTTON, 15);  // создание объекта - кнопка

void setup() {
        pinMode(LED, OUTPUT);                   // определяем вывод светодиода как выход
        MsTimer2::set(2, timerInterupt);        // задаем период прерывания по таймеру 2 мс
        MsTimer2::start();                      // разрешаем прерывание по таймеру
}
void loop() {
// управление светодиодом
        if ( button.flagClick == true ) {       // был клик кнопки
                button.flagClick= false;        // сброс признака
                digitalWrite(LED, ! digitalRead(LED));  // инверсия состояния светодиода
        }
}
// обработчик прерывания
void  timerInterrupt() {
        button.scanState();       // вызов метода ожидания стабильного состояния для кнопки
}
```

Інший приклад… Бувають випадки, коли для визначення моменту деякої вхідної події (наприклад, натискання кнопки) використовується такий код:

```
void loop() {
              if (digitalRead(InputPin) ==LOW) {
                // Выполнить какие-то действия
              }
}
```

Цей код постійно перевіряє рівень напруги на контакті **InputPin**, і коли **digitalRead** повертає **LOW**, виконуються якісь дії, позначені коментарем

// Выполнить какие-то действия. Це цілком робоче рішення, але якщо всередині функції **loop** потрібно виконати велику кількість інших операцій, то всі ці операції займуть потрібен час, тому є можливість проґавити[11] коротке натискання на кнопку, поки процесор буде зайнятий чимось іншим.

А ще є короткі імпульси від датчика, які можуть тривати мільйонні частки секунди. Ось для прийому таких подій і слід використовувати **апаратні переривання** (**Hardware interrupts**). Саме вони визначають функції, які будуть викликатись за цими подіями, незалежно від того, чим зайнятий мікроконтролер.

В ARDUINO UNO лише два контакти пов'язані з апаратними перериваннями, через що вони використовуються дуже економно. У Leonardo таких контактів п'ять, на великих платах, таких як Mega2560, їх набагато більше, а в Due всі контакти підтримують можливість переривання.

Табл. 4.4. Контакти апаратних переривань плат Arduino

	INT 0	**INT 1**	**INT 2**	**INT 3**	**INT 4**	**INT 5**
ATmega 328/168 (Nano, UNO, Mini)	D2	D3	–	–	–	–
ATmega 32U4 (Leonardo, Micro)	D3	D2	D0	D1	D7	–
ATmega 2560 (Mega)	D2	D3	D21	D20	D19	D18
Due	–	–	–	–	–	–

Команди роботи з перериваннями:

attachInterrupt(pin, function, state); – підключить переривання
detachInterrupt(pin); – вимкнути переривання

Ще раз звернемося до схеми на Рис. 4.29. Доки кнопка не натиснута на контакті D2 "висить" LOW, а в той момент, коли контакти змикаються рівень напруги на ньому підніметься до HIGH.

Приклад коду_2:

```
int led = 12;
void setup() {
        pinMode(led, OUTPUT);
        attachInterrupt(0, button, HIGH);
}
void loop() {
                // Выполнить какие-то действия
}
void button() {
        digitalWrite(led,  HIGH);
}
```

[11] Насправді пропустити факт натискання на кнопку майже неможливо, тому що за мірками мікроконтролера вона залишається дуже довго натиснутою.

Окрім налаштування контакту LED – *pinMode(led, OUTPUT);* на роботу в режимі цифрового виходу, функція *setup* за допомогою ще одного рядка – *attachInterrupt(0, button, HIGH);* пов'язує функцію *button()* з перериванням "*0"*. Тепер у відповідь на кожне переривання автоматично буде викликатись ця функція.

Аргументи функції, що тут викликається:

attachInterrupt(0, button, HIGH);

Перший аргумент – *0*. Це номер переривання. В Arduino UNO переривання 0 пов'язане з контактом D2, а переривання 1 – з контактом D3. В інших моделях Arduino ці переривання можуть бути пов'язані з іншими контактами (див. табл. 4.4).

Другий аргумент – *button.* Це ім'я функції, що повинна викликатись для обробки переривання. Вона визначається далі у скетчі. До таких функцій (підпрограм обробки переривань – **Interrupt Service Routine, ISR**), діють особливі вимоги. Вони не можуть мати параметри і нічого не повинні повертати. У цьому є певний зміст: навіть при тому, що вони викликаються в різних місцях у скетчі, немає жодного рядка коду, що здійснює прямий виклик **ISR**, тому немає жодної можливості передати їм параметри або отримати значення, що повертається.

Останній параметр функції *attachInterrupt* — це константа (в нашому випадку HIGH). Вона означає, що підпрограма обробки переривання буде викликатись лише за рівень напруги на контакті D2 HIGH (що і відбудеться в момент натискання кнопки). Взагалі існує кілька режимів (див. табл. 4.5).

Найчастіше використовуються режими переривань RISING (за позитивним перепадом) та FALLING (за негативним перепадом).

Табл. 4.5. Режими обробки переривань плат Arduino

Режим	Дія	Опис
LOW	Переривання генерується при рівні напруги LOW	У цьому режимі підпрограма обробки переривань буде викликатись постійно, поки на контакті зберігається низький рівень напруги
RISING	Переривання генерується при перепаді напруги з рівня LOW до рівня HIGH	—
FALLING	Переривання генерується при перепаді напруги з рівня HIGH до рівня LOW	—
HIGH	Переривання генерується при рівні напруги HIGH	Цей режим підтримується тільки в моделі Arduino Due, він так як і режим LOW, рідко використовується на практиці
CHANGE	Переривання генерується за будь-якої зміни рівня сигналу	

Виконання роботи.

5) Зібрати макет, що відображено на рис. 4.29 .
6) Підключити схему до живлення.
7) Завантажити скетч (спочатку *Приклад коду_1*, потім *Приклад коду_2*) в мікроконтролер Arduino.
8) Перевірити правильність роботи програм в обох випадках.

Завдання: створити скетч програми (можна використати схему[12] з одній із попередніх лабораторних робіт: «світлофор», «гірлянда», «теплиця» або «музей»), що буде реалізовувати реакцію на зміну параметрів сенсорів за допомогою переривань.

Контрольні питання

1) Що таке переривання?
2) Які значення може видавати функція attachInterrupt ( )?
3) Опишіть, коротко, алгоритм роботи програми.

[12] Якщо в схемі задіяна кнопка, то бажано описати із використанням власної бібліотеки.

Лабораторна робота №6 «Регістр зсуву 74HC595»

Тема: Використання побітових операцій у роботі з регістром зсуву 74HC595.

Мета: Ознайомитись з можливостями бітових операцій в середовищі Arduino та дослідити роботи регістру зсуву 74HC595; закріпити навички програмування режимів роботи з використанням переривань; закріпити навички роботи з тактовими кнопками.

Інструменти для виконання роботи: комп'ютер з встановленим середовищем розробки Arduino IDE; плата Arduino з підключенням через USB порт (Arduino Uno, Arduino Mega та ін.); монтажна плата BreadBoard; дроти (перемички типу "П-П"); світлодіоди; резистори 220 Ом.

Теоретичні відомості.

В попередніх лабораторних роботах було використано не багата кількість елементів, що керуються мікропроцесором з плати Arduino. Якщо ж буде стояти ускладнено завдання з більшою кількістю елементів, наприклад: зробити гірлянду на 20-30 світлодіодів, або задіяти десять кнопок, або датчики, дисплеї і т. д., то Arduino Uno може і не задовольнити потреби в пінах. Трохи більш можливостей має плата Arduino Mega, але і вона обмежена певною кількістю портів. Тобто, рано чи пізно при ускладненні своїх проектів, трапиться ситуація, коли пінів під усі потреби може не вистачити.

Для простого і надійного вирішення проблеми з дефіцитом портів введення-виведення існують так звані зсувні регістри – мікросхеми, що перетворюють всього три піна Arduino на десятки і сотні, перетворюючи послідовний інтерфейс (перелік того, що включити, а що вимкнути) на паралельний (умикнуто-вимкнено на кожній ніжці) або навпаки. Зсувний регістр (рис. 6.2)– це набір послідовно з'єднаних тригерів[13] (рис. 6.1). На відміну від стандартних регістрів, зсувні підтримують функцію зсуву вправо та вліво. Тобто здійснюється переписування даних з кожного попереднього тригера на наступний.

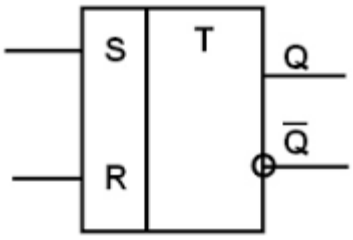


Рис. 6.1 Зображення тригера

[13] найпростіший послідовний пристрій, який може перебувати в одному з двох можливих станів і змінювати його під впливом вхідних сигналів. Тригер є базовим елементом послідовних логічних пристроїв.

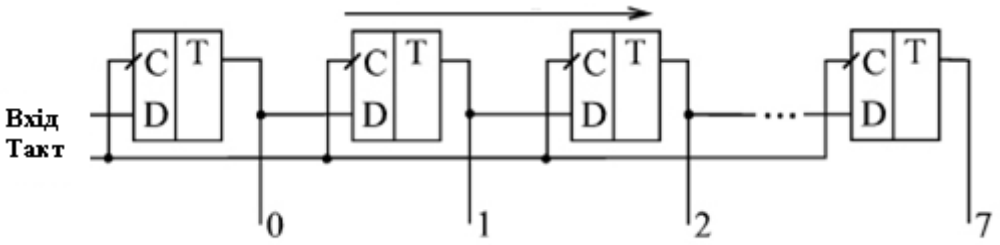


Рис. 6.2 Зображення зсувного регістру.

Існує багато технічних виконань зсувних регістрів, тут пропонується розглянути один з найпоширеніших – вихідний регістр 74HC595.

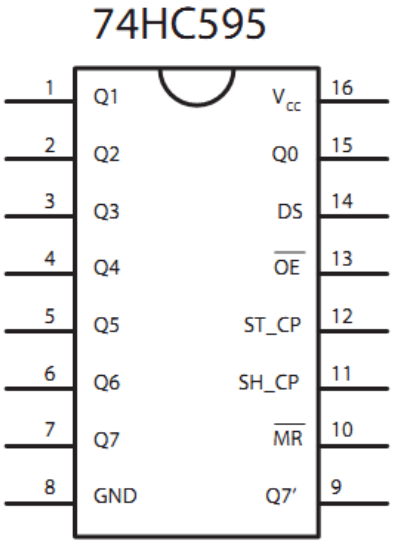


Рис. 6.3 Зображення регістру 74HC595 на принципових схемах

Основні характеристики зсувного регістру 74HC595

- Розрядність паралельного входу (виходу) – 8 біт
- Фактор та кількість пінів - DIP-16 та SO-16
- Кількість сигнальних ліній для передачі даних - 3
- Час встановлення - 20 нс
- Максимальна частота - 100 МГц
- Напруга живлення - 5 В
- Струм споживання - 40 мА

Мікросхема зсувного регістру має 16 контактів:

- Порти 15 і з 1-го по 7-й – це вісім цифрових виходів, вони позначені як Q0-Q7 відповідно. Пін Q7 відповідає першому біту, що посилається в зсувний регістр, а пін Q0 – останньому.
- Порт 8 (GND) підключається до “землі”.
- Порт 9 (Q7’) — «вихідні дані», використовується для передачі даних іншому регістру зсуву, якщо такий є.
- Порт 10 ($\overline{MR}$) – завжди повинен бути підключений до шини живлення 5 В (наприклад, до контакту 5 V на платі Arduino).
- Порти 11 (SH_SP) та 12 (ST_SP), це – вхід для тактових імпульсів

та «застібка».

- Порт 13 ($\overline{OE}$) використовується для перемикання виходів між високоомним та робочим станом та зазвичай підключається до «землі» (GND).
- Порт 14 (DS) — вхід для бітів даних, що послідовно надсилаються платою Arduino.
- Порт 16 (Vcc) – живлення 5 В (підключається до контакту 5 V на платі Arduino).

Орієнтація контактів визначається по напівкруглій мітці на корпусі регістру зсуву, вона знаходиться між пінами 1 і 16. Піни нумеруються у напрямку проти годинникової стрілки, як показано на рис. 6.3 із зображенням принципової схеми.

Мікросхема 74HC595 перетворює послідовний сигнал, що входить, через порт (DS) у вихідний паралельний сигнал на 8 виводах (Q0-Q7). Послідовна передача синхронна: для тактових сигналів використовується порт (SH_SP). Також окремим виводом (ST_SP) управляється регістр даних, що дозволяє змінювати сигнал на 8 виходах одночасно, коли усі дані передані. Таким чином, трьома портами мікроконтролера можна керувати вісьма цифровими виходами. Можна навіть робити каскади з регістрів 74HC595, підключаючи один до одного (через пін Q7’). Це дозволяє отримати 16, 24, 32 цифрові виходи.

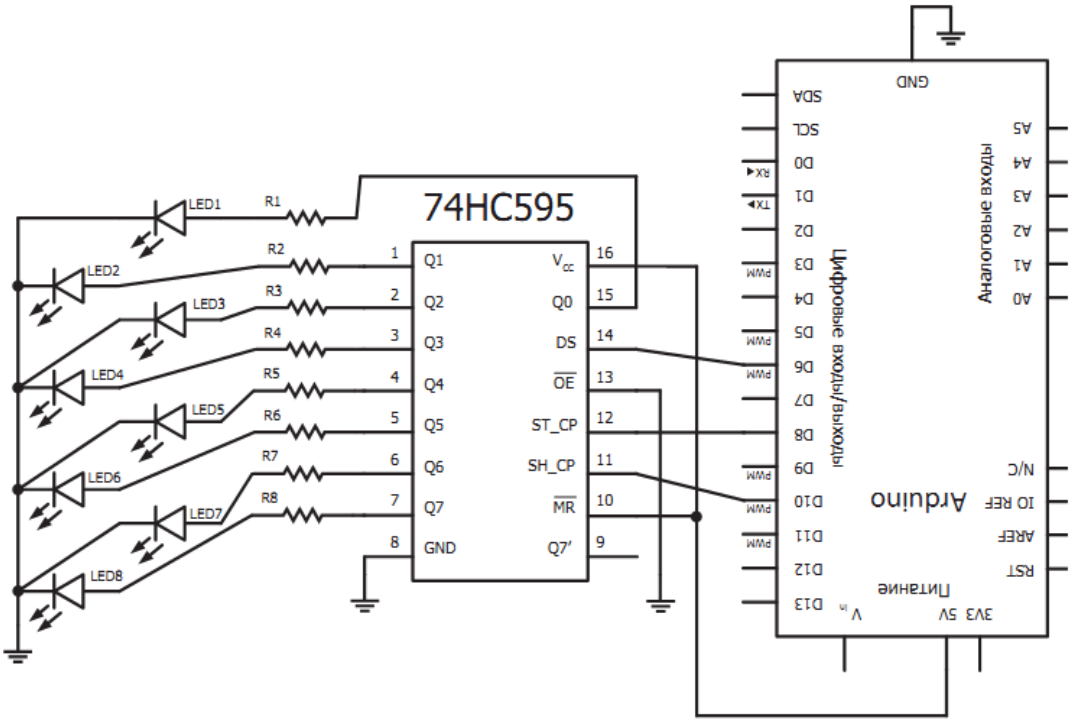


Рис. 6.4 Схема підключення регістру 74HC595 до плати ARDUINO UNO

Основні побітові операції

В основі обробки цифрових сигналів лежать бітові операції. За допомогою їх з одного або кількох сигналів на вході виходить новий сигнал, який у свою чергу може бути поданий на вхід однієї або кількох операцій. Саме бітові операції в поєднанні із елементами, що запам'ятовують (наприклад, ті

самі тригери) реалізують все розмаїття можливостей сучасної цифрової техніки.

До бітових операцій[14] над ланцюжками бітів, як правило, включають логічні побітові операції і бітові зсуви.

У середовищі Arduino значення байта починається з літери B і далі записуються значення байта побітово:

byte mybyte = B11001100;

Також при розборі байта використовуються поняття старшого (HIGH) та молодшого (LOW) біта. У прикладі старший біт 1 (перший ліворуч), молодший — 0 (перший праворуч, останній). Так, наприклад, під час виконання команди для регістра мікроконтролера:

DDRD = DDRD | B11111100;

два нулі в кінці байти (молодші біти) відповідають саме за 0 (RX) і 1 (TX) піни плати Arduino.

Побітове І (AND)

Оператор побітового І (AND) – амперсанд &. Результат дорівнює 1, тільки якщо обидві частини виразу дорівнюють 1, в інших випадках результат - 0:

0 & 0 == 0
0 & 1 == 0
1 & 0 == 0
1 & 1 == 1

Приклад:

```
byte b1 = B10110010;  //B10110010
byte b2 = B11100111;  //B11100111
byte result = b1 & b2; //B10100010
```

Побітове АБО (OR)

Оператор побітового АБО (OR) – вертикальна риска |. Результат дорівнює 0, тільки якщо обидві частини виразу дорівнюють 0, в інших випадках результат - 1:

0 | 0 == 0
0 | 1 == 1
1 | 0 == 1
1 | 1 == 1

Приклад:

```
byte b1 = B10110010;  //B10110010
byte b2 = B11100111;  //B11100111
byte result = b1 | b2; //B11110111
```

Побітове Виключення АБО (XOR)

Оператор побітового виключення АБО (XOR) – символ каретки ^. Результат дорівнює 0 тільки якщо обидві частини виразу рівні, в інших випадках результат - 1:

---

[14] Вивчаються в дискретній математиці

0 ^ 0 == 0
0 ^ 1 == 1
1 ^ 0 == 1
1 ^ 1 == 0

Приклад:

```
byte b1 = B10110010;  //B10110010
byte b2 = B11100111;  //B11100111
byte result = b1 ^ b2; //B01010101
```

Побітове НЕ (NOT)

Оператор побітового НЕ (NOT) – символ тильда ~. На відміну від & та |, використовується лише з одним операндом. Він просто інвертує біти.

Приклад:

```
byte b1 = B10110010;  //B10110010
byte result = ~ b1;   //B01001101
```

Зсув бітів (Bit Shift Operations)

Існують лише 2 зсувні операції над бітами – зміщення біта вліво <<, і зміщення біта вправо >>. Слідом за знаком усунення слідує число, що вказує на скільки позицій змістити біти.

При зміщенні біта вліво старші біти губляться, а молодші заповнюються нулями:

```
byte b1 = B10000011;  //B10000011
byte result = b1 << 3; //B00011000
```

При зміщенні біта праворуч молодші біти губляться, а старші заповнюються нулями:

```
byte b2 = B01100001;  //B01100001
result = b2 >> 3;     //B00001100
```

Операції із присвоєнням

Усі згадані вище операції (крім побітового НЕ (NOT)) можна записувати у короткій формі. Цей прийом застосовується у випадках, коли в розрахунку використовується та ж змінна, якій буде надано результат:

```
byte b1 = B10000011;

b1 = b1 >> 3;
// теж саме, що і
b1 >>= 3;

b1 = b1 << 2;
// теж саме, що і
b1 <<= 2;

b1 = b1 | B10011101;
// теж саме, що і
b1 |= B10011101;

b1 = b1 ^ B10011101;
```

```
// теж саме, що і
b1 ^= B10011101;

// и т.д.
```

Маніпуляції з одиничними бітами

Іноді виникає необхідність точкового впливу на конкретний біт одного байта. Далі наведено приклади основних операцій над одиничними бітами з використанням операцій зсуву одиничного байта (1=B00000001).

Для запису одиниці в біт під номером n (де n=[0, 1, .., 7]):

```
b |= (1 << n);
```

// Приклад:

```
byte b = B00000000;
b |= (1 << 4);  //B00010000
```

Для запису нуля в біт під номером n (де n=[0, 1, .., 7]):

```
b &= ~(1 << n);
```

// Приклад:

```
byte b = B11111111;
b &= ~(1 << 4);  //B11101111
```

Інвертувати біт під номером n (де n=[0, 1, .., 7]) можна так:

```
b ^= (1 << n);
```

// Приклад:

```
byte b = B11110000;
b ^= (1 << 4); //B11100000
b ^= (1 << 1); //B11100010
```

До стандартних функцій Arduino по роботі з бітами відносять функції: *bit(), bitWrite(), bitSet(), bitClear(), bitRead(),* при потребі за допомогою стандартних операцій можна запрограмувати індивідуальні маніпуляції з бітами. Таким чином можна реалізувати будь-яку бітову логіку.

Для зручної же роботи з регістром 74HC595 пропонується використовувати функція *shiftOut()*[15], що вбудована в Arduino. Вона здійснює побітовий зсув та вивід байту даних, починаючи з найстаршого (лівого) або молодшого (правого) значущого біта. Функція по черзі відправляє кожен біт на вказаний вивід даних, після чого формує імпульс (HIGH, потім LOW) на тактовому виводі, повідомляючи зовнішньому пристрою про надходження нового біта. Для взаємодії з пристроями, що тактуються по фронту імпульсів, перед викликом *shiftOut()* необхідно переконатися, що тактовий вивід переключений на низький рівень, наприклад, за допомогою функції *digitalWrite(clockPin, LOW).*

shiftOut(dataPin, bitorder, value).

Параметри:

- *dataPin*: пін, якому відправлятиметься кожен біт із зрушуваного байта даних (int)

[15] Функція є програмною реалізацією SPI, тому вона є швидшою, але працює лише зі спеціальними портами

- *clockPin*: тактовий вивід, який перемикатиметься кожного разу, коли на порту dataPin встановлюється коректне значення (int)
- *bitorder*: характеризує порядок, у якому зрушуватимуться і виводитися біти; може приймати значення MSBFIRST або LSBFIRST. (MOSt Significant Bit FIRSt – старший значний біт першим, або Least Significant Bit – молодший значний біт першим)
- *value*: байт даних, що зсувається (byte)

dataPin та *clockPin* мають бути вже налаштовані як виходи за допомогою функції *pinMode().*

Приклад передавання даних з використанням функції shiftOut():

```
//Вывод соединен с выводом ST_CP микросхемы 74HC595
int latchPin = 8;
//Вывод соединен с выводом SH_CP микросхемы 74HC595
int clockPin = 12;
////Вывод соединен с DS микросхемы 74HC595
int dataPin = 11;

void setup() {
//переключение выводов в режим работы "вывод", т.к. к ним идет обращение в главном цикле
pinMode(latchPin, OUTPUT);
pinMode(clockPin, OUTPUT);
pinMode(dataPin, OUTPUT);
}

void loop() {
//процедура последовательного счета
for (int j = 0; j < 256; j++) {
//формируем ноль на latchPin и удерживаем его до конца передачи
digitalWrite(latchPin, LOW);
shiftOut(dataPin, clockPin, LSBFIRST, j);
//возвращаем высокий уровень на latchin, т.е. сообщаем микросхеме о том, что
//больше не требуется воспринимать информацию
digitalWrite(latchPin, HIGH);
delay(1000);
}
}
```

Приклад практичного використання можливостей регістр 74HC595 продемонструємо на створенні гірлянди зі світло діодів (схема підключення див. рис. 6.5). Згідно скетчу *Приклад коду_1*, схема реалізує роботу шістьох режимів роботи світло діодів, які переключаються за допомогою кнопки (кожне натискання кнопки вмикає наступний режим і так по колу).

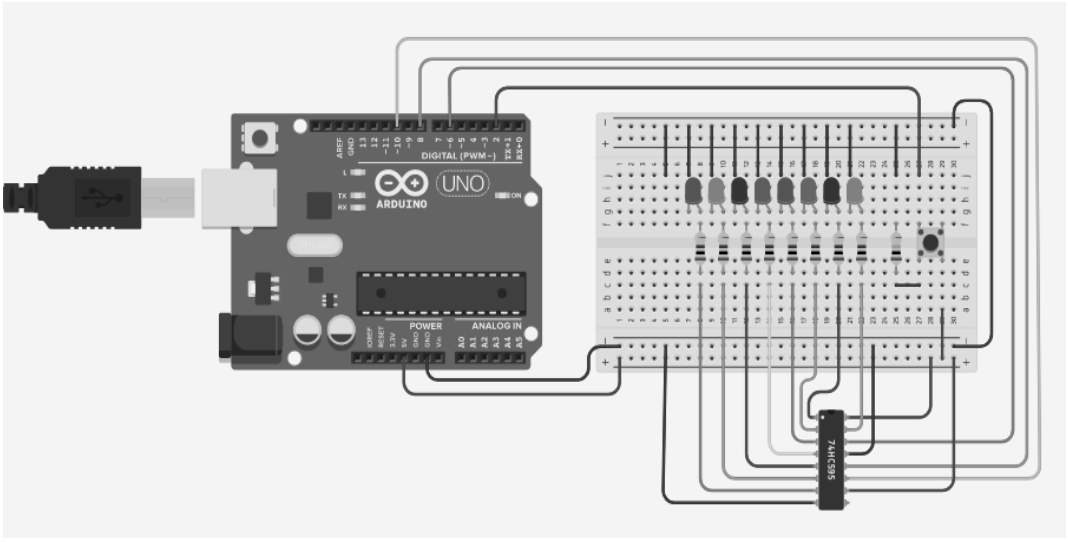


Рис. 6.5 Схема керування світлодіодами через регістр 74HC595

```
//Приклад коду_1: Керування світлодіодами через регістр 74HC595
#define  DATA 6  // цифровой выход 6 - к выводу 14 микросхемы 74HC595
#define  LATCH 8 // цифровой выход 8 - к выводу 12 микросхемы 74HC595
#define  CLOCK 10 // цифровой выход 10 - к выводу 11 микросхемы 74HC595
#define button 2  // кнопку вешаем на 2 пин
byte byteTOSend = 0; // создаем пустой байт B00000000
int seq[7] = {1,2,4,8,16,32,64}; // последовательность включения светодиодов
int flag=0;

void setup() {
// устанавливаем режим работы пинов
        pinMode(LATCH, OUTPUT);
        pinMode(CLOCK, OUTPUT);
        pinMode(DATA, OUTPUT);

        pinMode(button, INPUT_PULLUP);//настраиваем работу кнопки: пин "на выход"
        attachInterrupt(0,  button_press,  FALLING);
// ставим HIGH на "защёлку", чтобы регистр не принимал сигнал
        digitalWrite(LATCH, HIGH);
}

void loop() {
        byteTOSend = 0;
/* испорльзуем оператор switch для выборов режимов работы - проверяем какой режим сейчас задан
(перенменная flag)и выбираем, соответствующий режим (case)*/
        switch (flag) {

        case 0: //первый режим
        for (byte bitPOS = 0; bitPOS < 8; bitPOS++) {
                byteTOSend = 0; // обнуляем байт, чтобы выключить светодиод
                bitWrite(byteTOSend, bitPOS, HIGH);
                digitalWrite(LATCH, LOW);     // ставим LOW на "защёлку"
                shiftOut(DATA, CLOCK, MSBFIRST, byteTOSend);
                digitalWrite(LATCH, HIGH);    // ставим HIGH на "защёлку"
                delay(300);
```

```
        }
        break;

        case 1: //второй режим
        for (byte bitPOS = 0; bitPOS < 8; bitPOS++) { // В переменной хранится позиция изменяемого бита
            bitWrite(byteTOSend, bitPOS, HIGH); // При bitPOS=0 получим B00000001, при bitPOS=1 - B00000011, при bitPOS=2 - B00000100 и т.д.
            digitalWrite(LATCH, LOW); / устанавливаем синхронизацию "защелки" на LOW
            shiftOut(DATA, CLOCK, LSBFIRST, byteTOSend); // Инвертируем сигнал при помощи MSBFIRST, грузим с первого бита
            digitalWrite(LATCH, HIGH); //"защелкиваем" регистр, тем самым устанавливая значения на выходах
            delay(100);
        }
        delay(50);

        for (int bitPOS = 0; bitPOS < 8; bitPOS++) {
            bitWrite(byteTOSend, bitPOS, LOW);
            digitalWrite(LATCH, LOW);
            shiftOut(DATA, CLOCK, LSBFIRST, byteTOSend);
            digitalWrite(LATCH, HIGH);
            delay(100);
        }
        break;

        case 2: //третий режим
        for (byte bitPOS = 0; bitPOS < 8; bitPOS++) { // В переменной хранится позиция изменяемого бита
            bitWrite(byteTOSend, bitPOS, HIGH); // При bitPOS=0 получим B00000001, при bitPOS=1 - B00000011, при bitPOS=2 - B00000100 и т.д.
            digitalWrite(LATCH, LOW); //устанавливаем синхронизацию "защелки" на LOW
            shiftOut(DATA, CLOCK, MSBFIRST, byteTOSend); // Инвертируем сигнал при помощи MSBFIRST
            digitalWrite(LATCH, HIGH); //"защелкиваем" регистр, тем самым устанавливая значения на выходах
            delay(100);
        }
        delay(50);

        for (int bitPOS = 0; bitPOS < 8; bitPOS++) {
            bitWrite(byteTOSend, bitPOS, LOW);
            digitalWrite(LATCH, LOW);
            shiftOut(DATA, CLOCK, MSBFIRST, byteTOSend);
            digitalWrite(LATCH, HIGH);
            delay(100);
        }
        break;

        case 3: //четвертый режим
            for (byte bitPOS = 0; bitPOS < 8; bitPOS++) { // выбираем по одному диоду
            byteTOSend = 0; // обнуляем байт, чтобы выключить светодиод
            bitWrite(byteTOSend, bitPOS, HIGH);
            digitalWrite(LATCH, LOW);       // ставим LOW на "защёлку"
```

```
            shiftOut(DATA, CLOCK, LSBFIRST, byteTOSend);
            digitalWrite(LATCH, HIGH);        // ставим HIGH на "защёлку"
            delay(100);
        }
        for (byte bitPOS = 0; bitPOS < 8; bitPOS++) {
            byteTOSend = 0; // обнуляем байт, чтобы выключить светодиод
            bitWrite(byteTOSend, bitPOS, HIGH);
            digitalWrite(LATCH, LOW);     // ставим LOW на "защёлку"
            shiftOut(DATA, CLOCK, MSBFIRST, byteTOSend);
            digitalWrite(LATCH, HIGH);    // ставим HIGH на "защёлку"
            delay(100);
        }

        break;

        case 4: //пятый режим
            byteTOSend = 170; // байт, включающий нечетные биты
            digitalWrite(LATCH, LOW);     // ставим LOW на "защёлку"
            shiftOut(DATA, CLOCK, LSBFIRST, byteTOSend);
            digitalWrite(LATCH, HIGH);    // ставим HIGH на "защёлку"
            delay(300);

            byteTOSend = ~byteTOSend; // инвертированием задаем байт, включающий
четные биты
            digitalWrite(LATCH, LOW);     // ставим LOW на "защёлку"
            shiftOut(DATA, CLOCK, LSBFIRST, byteTOSend);
            digitalWrite(LATCH, HIGH);    // ставим HIGH на "защёлку"
            delay(300);
            break;

        case 5: //шестой режим
            for (int i = 0; i < 14; i++) {
          digitalWrite(LATCH, LOW);  // устанавливаем LATCH в LOW, чтобы значения на
        параллельных выходах сдвигового регистра не изменялись во время ввода последовательных
        данных
            shiftOut(DATA, CLOCK, MSBFIRST, seq[i]); // ввод последовательных данных в
        сдвиговый регистр
            digitalWrite(LATCH, HIGH);  // устанавливаем LATCH в HIGH для вывода
        значений на параллельные выводы сдвигового регистра
            delay(100);
        }

        break;

        }
}

void  button_press(){
        flag+=1;
        if (flag>=6) // если переменная "переполнена" возвращаем ей начальное значение
            {flag=0;}
}
```

Виконання роботи.

9) Зібрати макет, що відображено на рис. 6.5.
10) Підключити схему до живлення.
11) Завантажити скетч *Приклад коду_1*.
12) Перевірити усі режими роботи схеми.
13) Змінити 1 або 2 режими керування світло діодами за своїм алгоритмом.
14) Перевірити відсутність помилок в їх роботи.

Завдання: деталізація завдання у викладача, що проводе лабораторні заняття.

Контрольні питання

6) Що таке тригера?
7) Які параметри живлення має регістр зсуву 74HC595?
8) Поясніть призначення контактів Q0-Q7 та SH_SP регістру 74HC595.
9) Назвіть та пояснити основні побітові операції.
10) Назвіть параметри функції *shiftOut( ).*

Лабораторна робота №7 «Ethernet-Shield HR911105A та бібліотека EtherCard»

Тема: Підключення мікроконтролера Arduino до локальної мережі

Мета: Ознайомитись з можливостями роботи Arduino в локальної мережі на прикладі LAN модуля HR911105A.

Інструменти для виконання роботи: комп'ютер з підключенням до мережі Інтернет та з встановленим середовищем розробки Arduino IDE; плата Arduino (Arduino Uno, Arduino Mega та ін.); роутер; Ethernet-Shield HR911105A з пачкордом для під'єднання до роутера; монтажна плата BreadBoard; датчики, дроти (перемички типу "П-П", «М-П»); світлодіоди; резистори 220 Ом.

Теоретичні відомості.

Ethernet-Shield

Ethernet-адаптер ENC28J60 (мережева картка) на одному чіпі, розроблений компанією MicrochIP. Мікросхема не вимагає багато обв'язки із зовнішніх компонентів, до мікроконтролера підключається за допомогою SPI. Повністю відповідає специфікації IEEE 802.3.

Мікросхема випускається в 28-пінових DIP корпусах, але існують і готові модулі з усією обв'язкою та роз'ємом для мережного кабелю. Саме таким є LAN модуль **HR911105A**.

Рис. 7.1 LAN Ethernet шилд HR911105A на мікросхемі ENC28J60

Ethernet-шилд з маркуванням HR911105A підходить до багатьох плат контролера. Мікросхема ENC28J60[16] живиться від 5В, а споживає досить багато – до 250 мА. Стільки їй потрібно для живлення драйверів передавача, але є і режим зниженого енергоспоживання, коли вся силова частина відключається. Розшифрування позначень контактів на платі Ethernet-шилд HR911105A наведено на рiс.7.2.

[16] Повний опис мікросхеми див. даташит: www.Microchip.COM/doclisting/TechDoc.aspx?type=datasheet

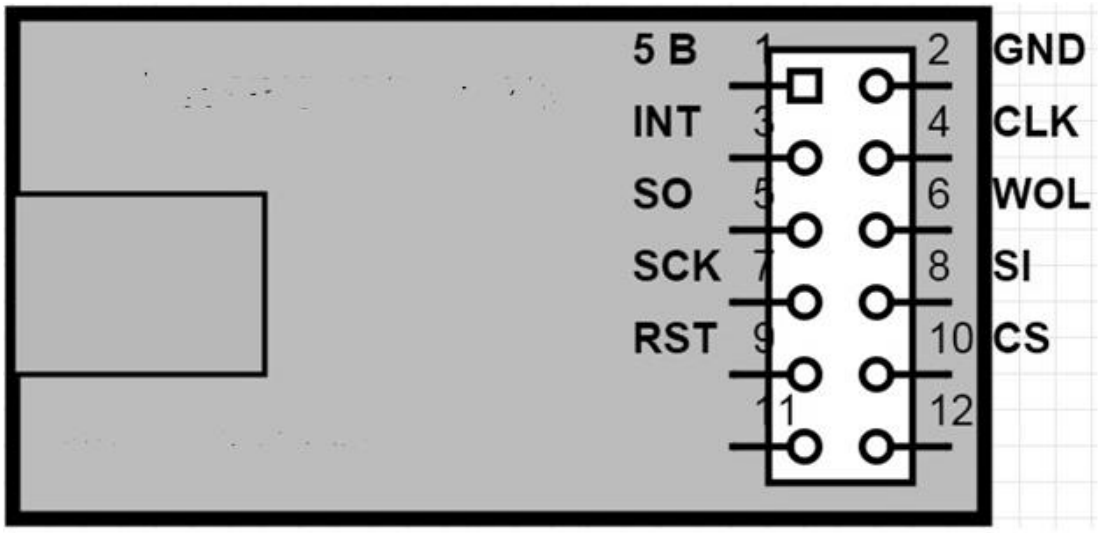


Рис. 7.2 Вид зверху на плату

Підключати Ethernet шилд до Arduino Uno рекомендовано за такою схемою:

VCC – 5V
GND – GND
SCK – Pin 13
SO – Pin 12
SI – Pin 11
CS – Pin 8

А до Arduino Mega так:

VCC – 5V
GND – GND
SCK – Pin 52
SO – Pin 50
SI – Pin 51
CS – Pin 53

Для роботи з цим модулем також, рекомендовано використовувати бібліотеку **EtherCard**.

Основні відомості по інтерфейсу SPI (Serial Peripheral Interface).

Послідовний периферійний інтерфейс (SPI) – це синхронний протокол послідовної передачі даних, що використовується для зв'язку мікроконтролера з одним або декількома периферійними пристроями. Інтерфейс SPI відрізняється відносно високою швидкістю та призначений для зв'язку близько розташованих пристроїв. Він також може використовуватися для взаємодії двох мікроконтролерів. Відповідно до протоколу SPI, один із взаємодіючих пристроїв (зазвичай мікроконтролер) завжди є «ведучий» і контролює «ведені» периферійні пристрої. Як правило, всі пристрої, що взаємодіють, об'єднані трьома загальними лініями:

MISO (Master In Slave Out) – лінія передачі даних від «веденого» пристрою (Slave) до «ведучого» (Master),

MOSI (Master Out Slave In) – лінія для передачі даних від «ведучого» пристрою (Master) до «веденого» (Slave),

SCK (Serial Clock) – тактові імпульси, що генеруються «ведучим» пристроєм (Master) для синхронізації процесу передачі даних.

Окрім перерахованих, на кожен пристрій відводиться окрема лінія:

SS (Slave Select) – вивід, що є на кожному «веденому» пристрої. Він призначений для активізації Майстром того чи іншого периферійного пристрою.

Периферійні пристрої (Slave) взаємодіють з «ведучим» (Master) тоді, коли на виводах SS є низький рівень сигналу. В іншому випадку дані від Master-пристрою ігноруватимуться. Така архітектура дозволяє взаємодіяти з кількома SPI-пристроями, підключеними до однієї і тієї ж шини: MISO, MOSI та SCK.

Особливості роботи виведення SS в Ардуіно на базі AVR

Усі моделі Arduino на основі мікроконтролерів AVR мають роз'єм виведення SS, який використовується в режимі роботи Slave (наприклад, при керуванні Ардуіно зовнішнім «ведучим» пристроєм). Однак, у бібліотеці реалізовано лише режим роботи Master, тому в цьому режимі виведення SS має бути налаштовано як вихід. В іншому випадку SPI може переключитися апаратно в режим Slave, що призведе до непрацездатності функцій бібліотеки. Для керування виведенням SS периферійних пристроїв можна використовувати будь-який із доступних виводів. Наприклад, на платі розширення Arduino Ethernet для взаємодії з вбудованою карткою SD і контролером Ethernet по SPI використовуються піни 4 і 10 відповідно.

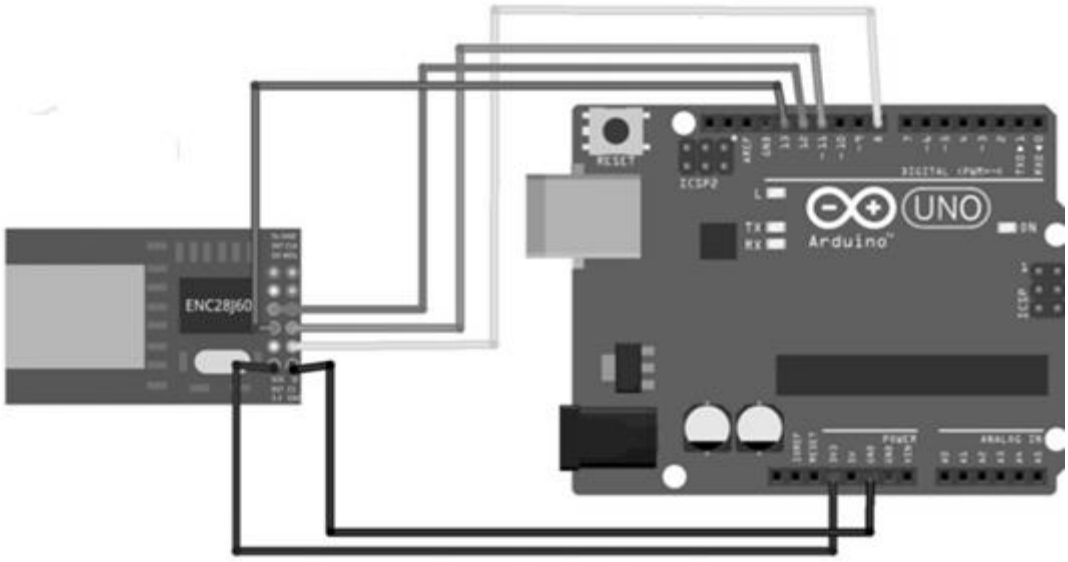


Рис. 7.3 Підключення HR911105A до плати Arduino Uno

Якщо схемі зібрана та перевірено згідно рис. 7.3, бібліотеку EtherCard розміщено у відповідному місці (C:\Program Files\Arduino\libraries\ EtherCard-Master), то можна приступати до перевірки роботи схеми. Для цього пропонується скористатися стандартним прикладом Pings з папки Приклади \ EtherCards (пінгування віддаленого сервера)

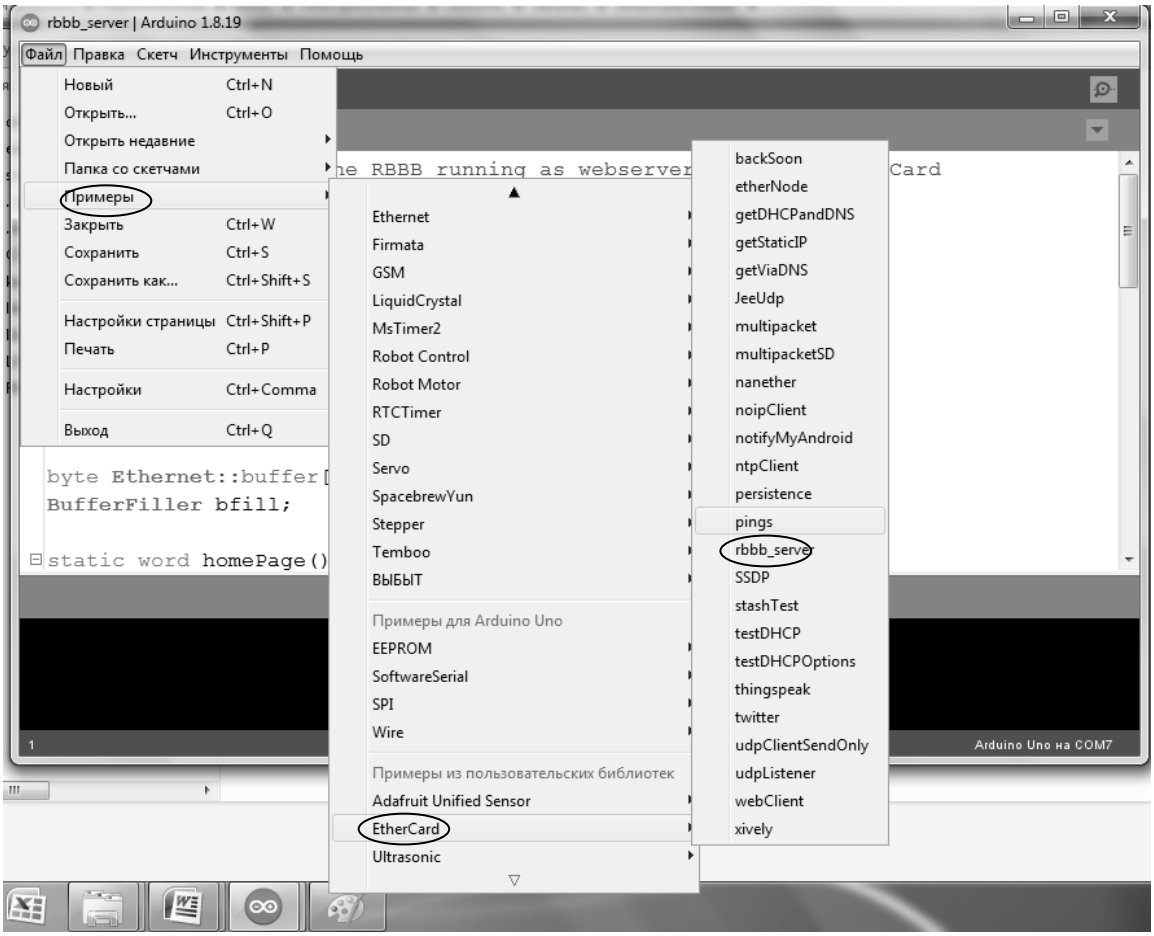


Рис. 7.4 Приклад Pings – пінгування віддаленого сервера

Скетч пінгує ресурс www.google.com, та виводить в монітор порта його ip-адресу та час пінгу. І якщо в моніторі порта відображається ці відомості (рис. 7.5), то мережа налагоджена та працює відповідним чином.

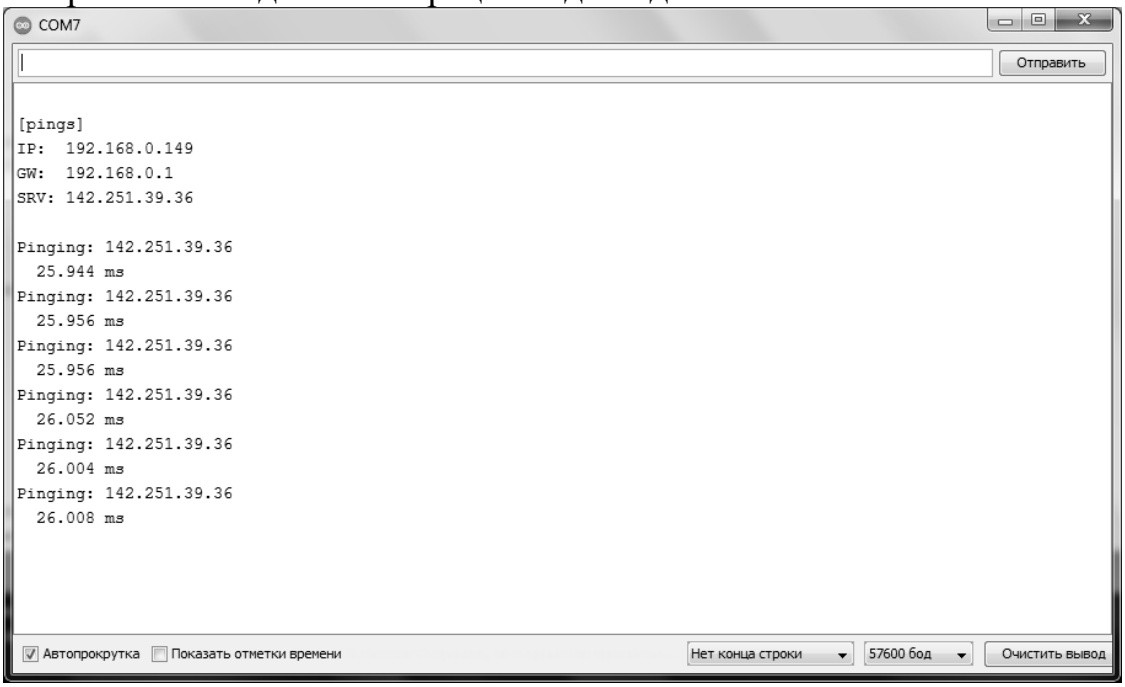


Рис. 7.5 Перегляд результату роботи скетчу Pings через монітор порту.

Другий стандартний приклад rbbb_server (див. рис. 7.4) демонструє роботу в якості найпростішого веб-сервера:

```
// This is a demo Of the RBBB running as webserver with the EtherCard
// 2010-05-28 <jc@wippler.nl>
//
// License: GPLv2

#include  <EtherCard.h>

// ethernet interface Mac address, must be unique on the LAN
static byte myMac[] = { 0x74,0x69,0x69,0x2D,0x30,0x31 };
static byte myip[] = { 192,168,0,149 };

byte Ethernet::buffer[500];
BufferFiller bfill;

static word homePage() {
  long t = millis() / 1000;
  word h = t / 3600;
  byte m = (t / 60) % 60;
  byte s = t % 60;
  bfill = ether.tcpOffset();
  bfill.emit_p(PSTR(
    "HTTP/1.0 200 OK\r\n"
    "Content-Type: tEXT/html\r\n"
    "Pragma: no-cache\r\n"
    "\r\n"
    "<meta http-equiv='refresh' Content='1'/>"
    "<title>RBBB server</title>"
    "<h1>$D$D:$D$D:$D$D</h1>"),
      h/10, h%10, m/10, m%10, s/10, s%10);
  return bfill.POSition();
}

void setup () {
  Serial.begin(57600);
  Serial.println(F("\n[RBBB Server]"));
  // CHANGE 'SS' To your Slave Select pin, if you arn't using the DEFAULT pin
  if (ether.begin(sizeOf Ethernet::buffer, myMac, SS) == 0)
    Serial.println(F("Failed To access Ethernet Controller"));
  ether.staticSetup(myip);
}

void loop () {
  word len = ether.packetReceive();
  word POS = ether.packetLoop(len);

  if (POS)  // check if valid tcp data is received
    ether.httpServerReply(homePage()); // send web page data
}
```

У випадку, коли спроба отримати доступ до контролера Ethernet буде не вдалою в моніторі порта можна буде побачити відповідне повідомлення: *Failed To access Ethernet Controller* (див. рис. 7.6).

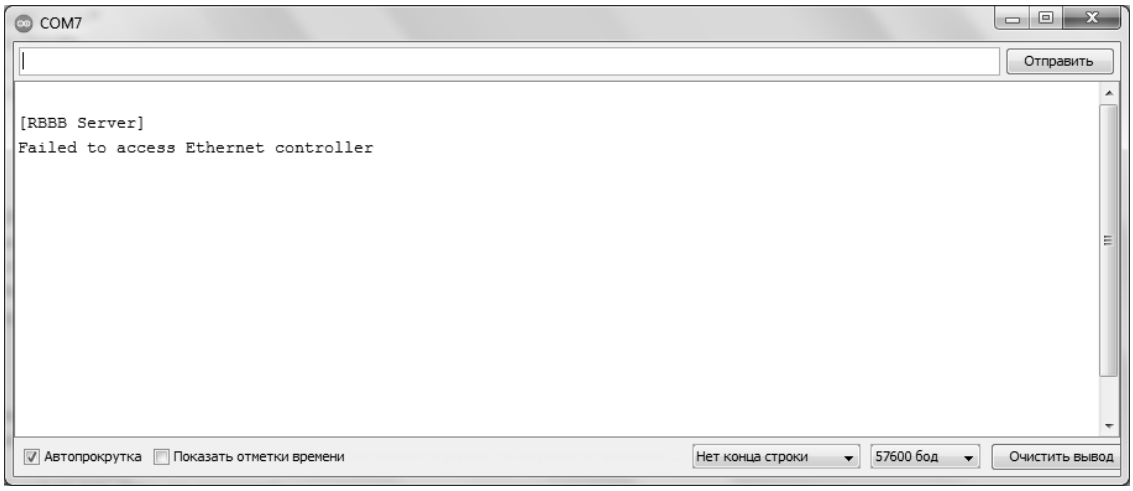


Рис. 7.6 Демонстрація роботи RBBB як веб-сервера з EtherCard через монітор порта (немає доступу до контролера Ethernet).

Якщо е зв'язок із сервером, то роботу сервера можна спостерігати і через будь якій браузер будь якого пристрою локальній мережі (ПК, планшет, смартфон), як це відображено на рис. 7.7.

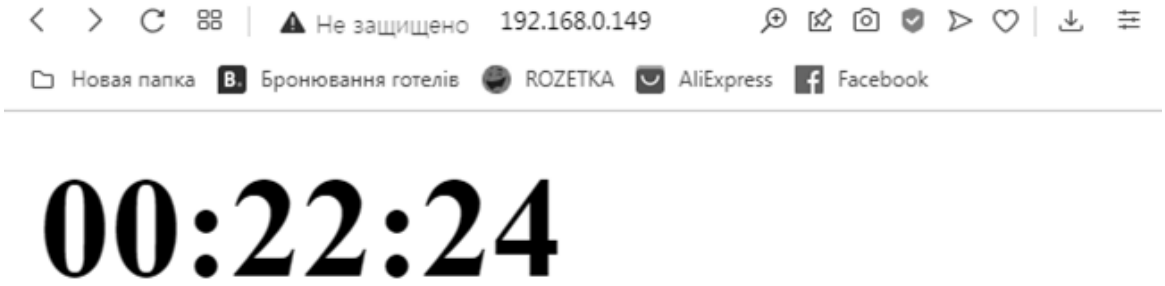


Рис. 7.7 Демонстрація роботи веб-сервера з EtherCard.

Для подальшого просування знадобляться знання та навички по роботі з мовою HTML та досвід використання датчиків. Наприклад простий веб-сервер, що здійснює клімат-контроль приміщення в реальному часі: відображає параметри температури та вологості та через який дистанційно можна керувати приладами обігріву та зволожування (в лабораторних умовах їх імітуємо світлодіодами) можна описати так:

```
// Приклад коду_1: Управляем Arduino с Web страницы: Webserver Ethernet ENC28J60.

#include  <EtherCard.h> // Подключаем  библиотеку для работы с Ethernet-модулем
#include  <dht.h> /// Подключаем библиотеку датчика температуры и влажности
dht DHT; //Инициируем датчик
#define dht_pin A0 //показания датчика с пина A0 храним в переменной dht_pin
int Pins[] = {2,3};
// на пинах 2 и 3 висят реле приборов управления климатом
// дальше для обращения к пину будут ипользоваться ссылки на пин: PIN1,PIN2
boolean PinStatus[3]; //состояние переключ-лей приборов опис. переменой PinStatus
float t; //вещественная переменная t будет хранить значение температуры
float h; //вещественная переменная h будет хранить значение влажности
char t_str[10]; //символьный массив для хранения значения температуры
char h_str[10]; //символьный массив для сохранения значения влажности
```

```
//*****static****используется для создания переменной, которая видна только одной функции. В отличие от локальных переменных, которые создаются и уничтожаются при каждом вызове функции, статические переменные остаются после вызова функции, сохраняя свои значения между её вызовами
    static byte myMac[] = {0x5A,0x5A,0x5A,0x5A,0x5A,0x5A }; //MAC Address д. б. уникальным.
    static byte myip[] = {192,168,0,202 }; //ip Address нашей Web страницы.
    byte Ethernet::buffer[900]; // Буфер отправки и получения, чем больше данных на Web странице, тем больше понадобится значения буфера (настроить размер буфера до 900 октетов)
    BufferFiller bfill;

//Неизменяемые строки и данные во время работы скетча можно хранить во флеш-памяти. Для этого используется ключ PROGMEM
    const char http_OK[] PROGMEM =
    "HTTP/1.0 200 OK\r\n" //стартовая строка ответа (код 200 - "успех")
    "Content-Type: tEXT/html; charset=utf-8\r\n" //тело передается в коде HTML, кодировка UTF-8 (для кириллических символов)
    "Pragma: no-cache\r\n\r\n";
    const char http_Found[] PROGMEM =
    "HTTP/1.0 302 Found\r\n" //код 302 сообщает клиенту о необходимости сделать запрос по другому URI.(?)
    "Location: /\r\n\r\n";
    const char http_UnauthORized[] PROGMEM =
    "HTTP/1.0 401 UnauthORized\r\n" //код 401 Информирует об ошибках со стороны клиента.(?)
    "Content-Type: tEXT/html\r\n\r\n"
    "<h1>401 UnauthORized</h1>";
    // Делаем функцию для оформления нашей Web страницы.

    void homePage(){
        bfill.emit_p(PSTR("$F"
        "<meta http-equiv='refresh' Content='10'/>"
        "<title>Server_TEMPERATURE</title>"
        "<head> <h2>теплица-1</h2> </head> <br />"
        "<body bgcolOR=""#99CCFF"">"
        "<br />"
        "<font size=""4"">"
        "Температура: <b>"
        "$S &deg;C<br />"      //"$S"-????, "&deg" - вывод символа процентов
        "Влажность: <b>"
        "$S %<br />"
        "<br />"
        "</b>"
        "включить обогрев: <a href=\"?PIN1=$F\">$F</a><br />"
        "включить полив: <a href=\"?PIN2=$F\">$F</a><br />"),
        http_OK,
        t_str,
        h_str,
        PinStatus[0]?PSTR("Off"):PSTR("on"),
        PinStatus[0]?PSTR("<font   colOR=\"green\"><b>ON</b></font>"):PSTR("<font
        colOR=\"red\">OFF</font>"),
        PinStatus[1]?PSTR("Off"):PSTR("on"),
        PinStatus[1]?PSTR("<font   colOR=\"green\"><b>ON</b></font>"):PSTR("<font
    colOR=\"red\">OFF</font>")
        );
    }
```

```
void setup(){
    Serial.begin(9600);
    if (ether.begin(sizeOf Ethernet::buffer, myMac) == 0); //Инициализация библиотеки EtherCard сомандой ether.begin() - запуск сервера, обслуживающего соединение (  ,мак адрес, порт)
    if (!ether.dhcpSetup());
// Динамический IP адрес, это не удобно, периодический наш IP адрес будет меняться.
// Нам придётся каждый раз узнавать кой адрес у нашей страницы.
    ether.printIp("My Router IP: ", ether.myip); // Выводим в Serial монитор IP адрес который нам присвоил DHCP-сервера (Router).

// Здесь мы подменяем наш динамический IP на статический / постоянный IP Address нашей Web страницы.
// Теперь не важно какой IP адрес присвоит нам Router, автоматический будем менять его, например на "192.168.0.202".
    ether.staticSetup(myip); //Для настройки статического IP-адреса используйте ether.staticSetup()
    ether.printIp("My SET IP: ", ether.myip); // Выводим в Serial монитор статический IP адрес.
    ether.printIp("DHCP-сервер:", ether.dhcpip); // выводим IP-адрес DHCP-сервера

//-----при старте все кнопки в состояние "ВЫКЛЮЧЕНО"
    for(int i = 0; i <= 2; i++) {
        pinMode(Pins[i],OUTPUT);
        PinStatus[i]=false;
    }
}

void loop(){
    digitalWrite(dht_pin, OUTPUT);
    delay(1);
    DHT.Read11(dht_pin); //считываем датчик
    t=DHT.temperature;      // переменной t присвоили текущее значение температуры
    h=DHT.humidity;         // переменной h присвоили текущее значение влажности
    dtOStrf(t, 6, 2, t_str); //функция dtOStrf() преобразует данные с плавающей точкой в массив символов (для удобства вывода на печать)
    dtOStrf(h, 6, 2, h_str);

//dtOStrf(floatvar, StringLengthIncDecimalPoint, numVarsAfterDecimal, charbuf);
//где: floatvar - преобразуемая переменная типа float;
//StringLengthIncDecimalPoint - длина получаемого символьного значения;
//numVarsAfterDecimal - количество символов после запятой;
//charbuf - символьный массив для сохранения результата преобразования

// получаем размер пакета
    word len = ether.packetReceive(); // check for ethernet packet / проверить ethernet пакеты (как описано в библиотеке копирует принятые данные в буфер и возвращает длину пакета данных)
    word POS = ether.packetLoop(len); // check for tcp packet / проверить TCP пакеты. (как описано в библиотеке анализирует полученные данные определенной длины, возвращает адрес смещения полезных данных в пакете или ноль, если ничего нет или служебные данные, как выяснилось также возвращает ноль если длина данных меньши восьми байт.)

    if (POS) {//heck if valid tcp data is received -если получены действительные данные tcp
        bfill = ether.tcpOffset();  //адрес начала буфера
        char *data = (char *) Ethernet::buffer + POS;
        if (strncmp("GET /", data, 5) != 0) {
            bfill.emit_p(http_UnauthORized);
        }
```

```
        else {
                data += 5;
                if (data[0] == ' ') {
                homePage(); // Return home page Если обнаружено изменения на станице, запускаем функцию.
                for (int i = 0; i <= 2; i++)
                digitalWrite(Pins[i],PinStatus[i]);
                }
                else if (strncmp("?PIN1=on ", data, 9) == 0) {
// Set PinStatus true AND redirect To home page/Установите PinStatus в true и перенаправить на домашнюю страницу
                        PinStatus[0] = true;
                        bfill.emit_p(http_Found);
                }
                else if (strncmp("?PIN1=Off ", data, 10) == 0) {
// Set PinStatus false AND redirect To home page/ Установить PinStatus в false и перенаправить на домашнюю страницу
                        PinStatus[0] = false;
                        bfill.emit_p(http_Found);
                }
                else if (strncmp("?PIN2=on ", data, 9) == 0) {
                        PinStatus[1] = true;
                        bfill.emit_p(http_Found);
                }
                else if (strncmp("?PIN2=Off ", data, 10) == 0) {
                        PinStatus[1] = false;
                        bfill.emit_p(http_Found);
                }
        }
                else {
        // Page not Found
                        bfill.emit_p(http_UnauthORized);
                }
        }
        ether.httpServerReply(bfill.POSition()); // send http rESPonse Отправка ответа на запрос HTTP.
        }
}
```

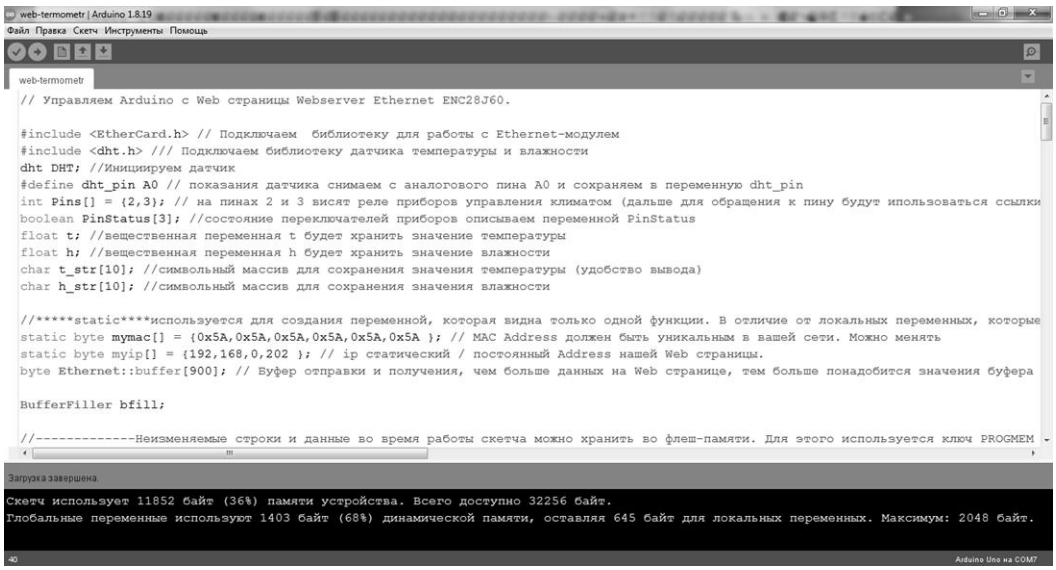


Рис. 7.8 Завантаження скетча *Приклад коду_1* в Arduino Uno.

В моніторі порта будуть виводитись IP адреси веб-сервера: та, що отримана від роутера і та, що надана в скетчі (рис. 7.9), а в браузері сторінка системи «Клімат-контроль» (рис. 7.10).

Рис. 7.9 Демонстрація роботи скетчу *Приклад коду_1* в моніторі порта.

Рис. 7.10 Демонстрація роботи веб-сервера «Клімат-контроль».

Виконання роботи.

1) Зібрати макет, що відображено на рис. 7.3.
2) Підключити схему до живлення.
3) Перевірити працездатність схеми на стандартних прикладах: **pings** та **rbbb_server**.
4) Додайте до схеми датчик DHT та 2 світлодіода.
5) Завантажити скетч *Приклад коду_1* та перевірити працездатність схеми.

Завдання: створити власний веб-сервер на Arduino з елементами керування (деталізація завдання у викладача, що проводе лабораторні заняття).

Контрольні питання

1) Що таке ENC28J60 та HR911105A?
2) Чим MISO відрізняється від MOSI?
3) Для чого призначена лінія SS**?**
4) Як дізнатися IP-адресу будь якого ресурсу в глобальній мережі?
5) Для чого і як використовують флеш-пам’ять при збереженні даних в скетчі *Приклад коду_1?*

5. СПИСОК ДЖЕРЕЛ

1. Офіційний сайт Arduino: https://www.arduino.cc/
2. Офіційний сайт компанії Atmel: https://www.microchip.com/
3. Плати Arduino: https://doc.arduino.ua/ru/hardware/
4. Середовище розробки Arduino: https://doc.arduino.ua/ru/guide/Environment
5. Програмування Arduino: https://doc.arduino.ua/ru/prog/
6. Інтернет-журнал з електроніки "РадиоЛис": https://radiolis.pp.ua/arduino
7. Інформаційний ресурс: https://arduino-diy.com/
8. "Мікропроцесори та мікроконтролери": https://microchipinf.com/ua/
9. Домашня сторінка бібліотеки AVR: https://www.nongnu.org/avr-libc/