НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

Тернопільський національний технічний університет імені Івана Пулюя

Кафедра автоматизації технологічних процесів і виробництв

Лабораторна робота № 16 з курсу ”Проектування систем автоматизації”

Ознайомлення із контролерами сімейства Arduino

Тернопіль 2016

Методичні вказівки до лабораторної роботи № 16 “Ознайомлення із контролерами сімейства Arduino” з курсу "Проектування систем автоматизації". Шкодзінський О.К., Пісьціо В.П., Медвідь В.Р., Галушка А.В., Тернопіль: ТНТУ, 2016 - 27 с.

Для студентів напряму: 6.050202 "Автоматизоване управління технологічними процесами"

Автори: Шкодзінський О.К., Пісьціо В.П., Медвідь В.Р., Галушка А.В.

Методічні вказівки розглянуті, схвалені і затверджені на засіданні кафедри автоматизації технологічних процесів і виробництв (протокол № 4 від 21.11.2016 року).

Тема роботи.

Ознайомлення із контролерами сімейства Arduino

Мета роботи

Ознайомитись із контролерами сімейства Arduino, визначити їх можливості та базові принципи їх програмування для побудови систем малої автоматизації.

Загальні відомості про контролери

Мікроконтролери застосовуються, перш за все, для автоматизації в метрології, техніці управління і автоматичного регулювання. Перевага мікроконтролерів полягає в тому, що можна ефективно і з малими затратами вимірювати і інтерпретувати фізичні величини, щоб потім приймати необхідні рішення і виконувати необхідні дії.

Область можливих застосувань мікроконтролерів надзвичайно широка: від приватного домогосподарства (наприклад, для управління теплицею або освітленням) до промислового виробництва, де можуть обслуговуватися і експлуатуватися комплексні пристрої, керовані системами мікроконтролерів. Наприклад при автоматизації теплиці контролер може фіксувати дані про температуру навколишнього середовища і вологості ґрунту, отримані від давачів. Результати вимірювання далі піддаються логічній обробці в мікроконтролері і при необхідності формуються сигнали управління насосом для поливу.

Структура та принцип дії мікроконтролера

Контролер представляє собою, мікрокомп’ютер і містить усі його основні модулі центральний процесор (CPU), оперативна пам'ять даних (RAM), пам'ять програм (Flash-пам'ять) та зовнішні пристрої.

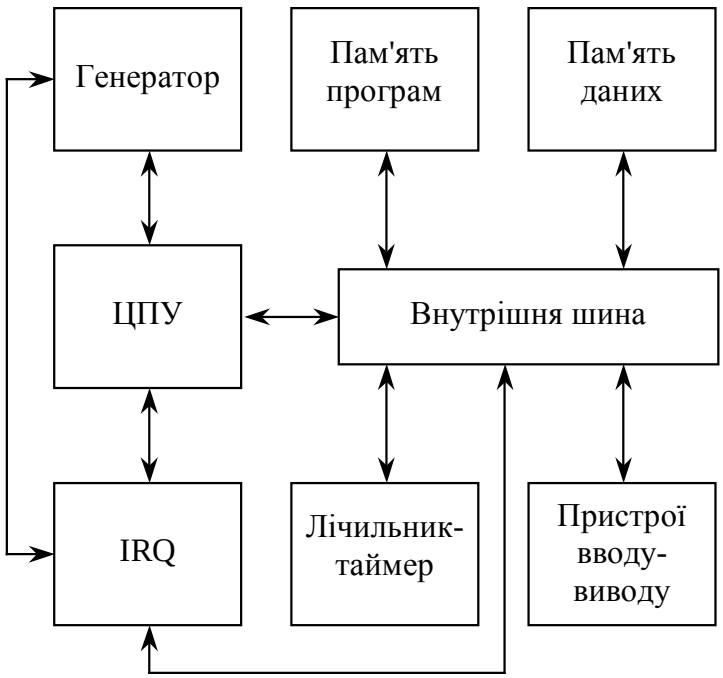


Рис. 1. Спрощена структура мікроконтролера.

Центральний процесор - основний функціональний пристрій мікроконтролера він виконує програму, що міститься у пам'яті, здійснює обмін даними із іншими пристроями системи та за допомогою тактового генератора задає загальну синхронізацію роботи мікроконтролера.

На відміну від "великих" ЕОМ мікроконтролер будується за гарвардською архітектурою, тобто має окрему пам'ять для збереження даних та програм. При цьому виконати програму із пам'яті даних не можливо, проте можна зберігати константи у пам'яті програм.

Пам'ять програм призначена для збереження програм та констант, що практично не змінюються у процесі роботи системи і побудована на основі Flash-пам'яті. Структура пам'яті програм показана на рис. 2.

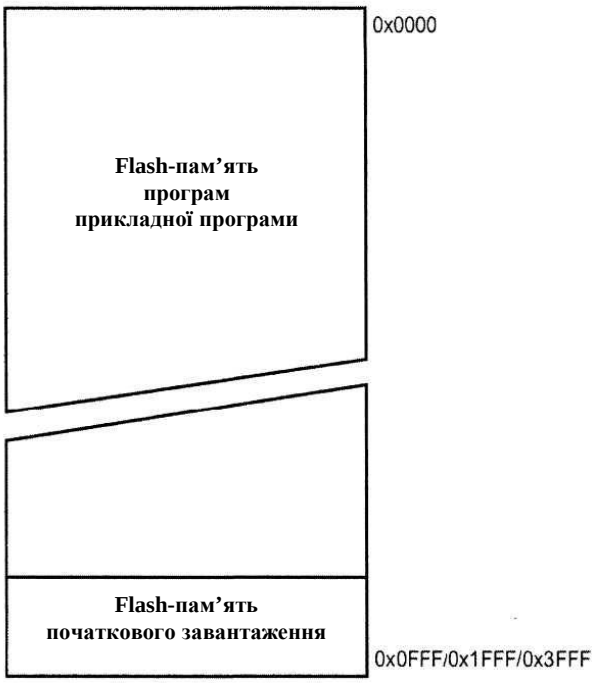


Рис. 2. Flash-пам'ять програм мікроконтролера ATmega168

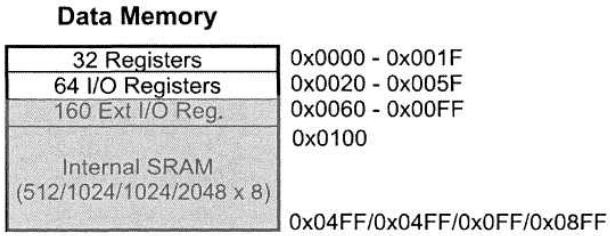


Рис. 3. Структура ОЗП мікроконтролера ATmega168

Оперативна пам'ять мікроконтролера (ОЗП) призначена виключно для збереження даних, що змінюються у процесі роботи системи. Програму виконати із оперативної пам'яті не можливо. Значення записуються та зберігаються в ОЗП лише при включеному живленні системи. На рис. 3 зображена структура ОЗП мікроконтролера ATmega168.

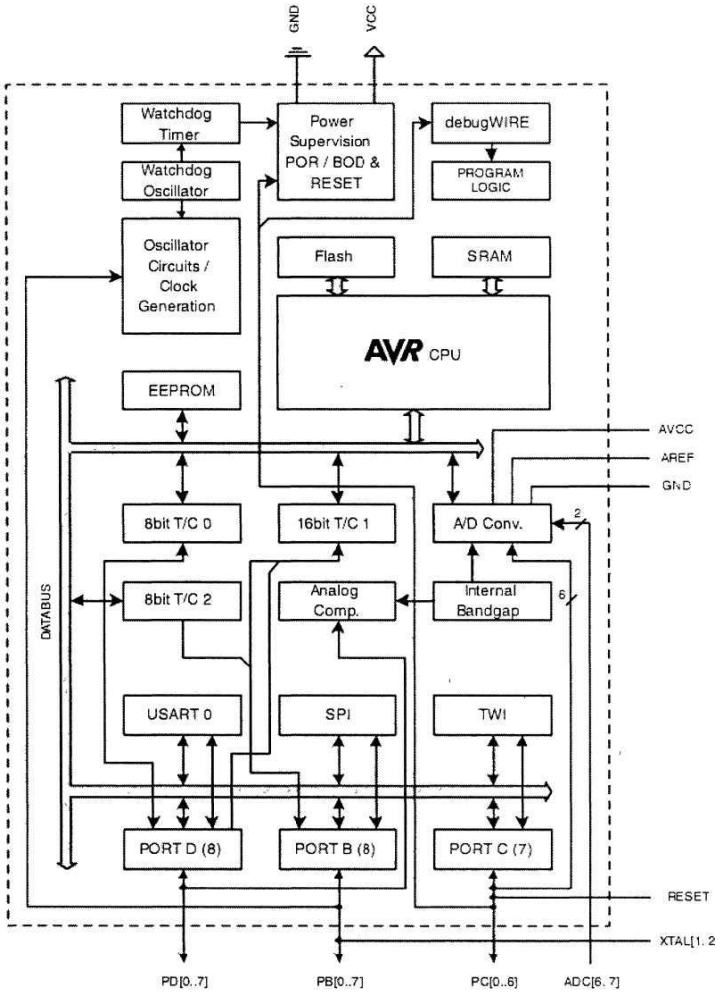


Рис. 4. Блок-схема мікроконтролера ATmega168

Зовнішні пристрої мікроконтролерів призначені для виконання операції обміну даними між мікроконтролером та "зовнішнім світом" до них відносять зовнішні інтерфейси, наприклад, цифрові входи і виходи (Input/Output), таймери, АЦП, ЦАП, тощо. Для прикладу на наступному рисунку наведена структурна схема мікроконтролера.

Програмування мікроконтролерів

Програмування мікроконтролерів здійснюється на мовах низького чи високого рівня. Мікроконтролер може виконувати команди записані лише у вигляді спеціальних машинних кодів. Сукупність всіх команд, що виконує мікроконтролер називається системою команд. Як відомо, для того, щоб виконати програму записану на будь-якій мові програмування її необхідно перетворити в машинні коди, тобто, відтранслювати.

Для мов високого рівня такий процес є однонаправленим - отримані коди команд процесора неможливо знову перевести в оператори мови програмування. Команди мови високого рівня, не залежать від особливостей та набору команд мікроконтролера, тому можливо перенесення програм між процесорами різних типів.

Для мов низького рівня, на відміну від мов високого між машинними кодами та символічними позначеннями є взаємо однозначна відповідність. Тому можливе пряме та зворотне перетворення програми та машинного коду.

Для зручності при програмуванні машинні коди команд позначають за допомогою спеціальних позначень. Сукупність позначень команд процесора називають асемблером. Асемблер відноситься до мов низького рівня.

Кожний тип мікроконтролера володіє своїм набором команд і власним асемблером. Таким чином основним недоліком асемблера є відсутність сумісності програм.

Переваги мови асемблер:

◊ Повний контроль над ресурсами МП системи.

◊ Найвища швидкодія отриманої програми.

Недоліки мови асемблер :

◊ Висока складність розробки програм.

◊ Неможливість перенесення програми на інший МП.

Мікроконтролери Arduino можуть програмуватись на будь-якій мові програмування , проте найчастіше їх програмують за допомогою мови С із використанням Arduino IDE. Компілятор С від Arduino дещо простіший, ніж професійні С-компілятори, але досить ефективний. З компілятором Arduino не потрібно піклуватися про програмуванні складних апаратних засобів, оскільки в середовищі розробки є відповідні вбудовані команди та бібліотеки для обслуговування пристроїв вводу-виводу, давачів, тощо.

Мова програмування Arduino-C підтримує принцип абстрагування від апаратних засобів (Hardware-Abstraction-Layer), що сильно полегшує програмування. Тому що безпосередню ініціалізацію апаратних засобів виконує компілятор Arduino. Наприклад, достатньо вказати, Serial.init(9600), щоб запрограмувати режим роботи послідовного порта без прямого програмування регістрів мікроконтролера.

Короткий опис сімейства мікроконтролерів Arduino

Основа Arduino - контролер ATMega компанії Atmel поширеного 8-розрядного сімейства AVR. До нього додається блок живлення і мікросхеми послідовного чи USB - інтерфейсу. Через останній відбувається завантаження програм користувача і, при необхідності, обмін даними між персональним комп'ютером і платою Arduino під час виконання програми.

Типова плата Arduino надає в розпорядження користувача 14 цифрових входів чи виходів, з них типово шість можна використовувати як аналоговий вихід (із 8-розрядний ШІМ, ще шість входів можуть приймати аналогові сигнали із 10-розрядним апаратним АЦП послідовного наближення. У якості додаткових інтерфейсів передбачені шини SPI і I^2C.

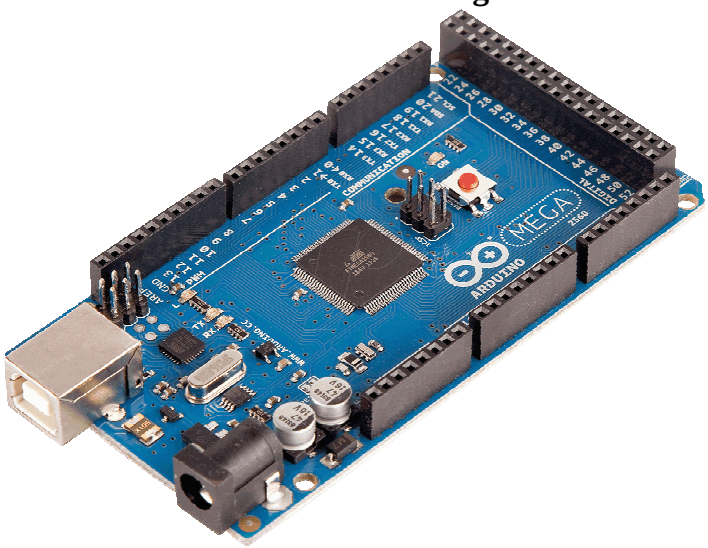
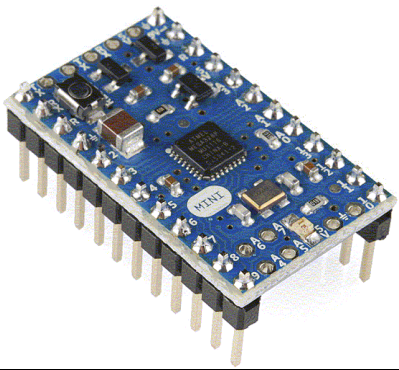
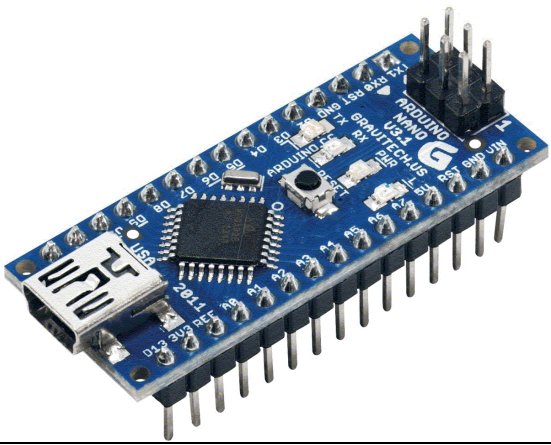
Існує багато варіантів виконання плат Arduino. Оригінальні вироби виробляє італійська фірма Smart Projects. Існують також численні клони і копії інших постачальників, нарешті, зустрічаються вільно поширювані апаратні засоби, у тому числі плати для

самостійного виготовлення. Найбільш розповсюдженими платами є ArduinoMega, ArduinoUno, Arduino Mini, Arduino Nano та Freeduino.

Технічні характеристики деяких із плат наведені нижче. Для Arduino також випущено безліч плат розширення, деякі із котрих будуть використані у наступних лабораторних роботах. Серед них є плати різних провідникових і без провідних інтерфейсів, плати контролю двигунів, давачів різної складності, адаптерів різних інтерфейсів і індикаторів, тощо.

Цікавою є плата Arduino ProtoShield, що призначена для прототипування саморобних пристроїв без пайки. Усі експерименти можна проводити на панелі із контактними гніздами.

Табл. 1

Зображення та назва	Технічні характеристики
Плата Arduino Mega. 	Мікроконтролер ATMega1280. Flash-пам'ять 128 кбайт, RAM-пам'ять 8 кбайт, EEPROM 4 кбайт. Тактова частота 16 МГц. 54 цифрових канали вводу/виводу, із них 14 можуть використовувати ШІМ із стандартними бібліотеками. 4 апаратних порти UART. Апаратні інтерфейси I^2C, SPI. 16 аналогових входів із 10 розрядним АЦП. Інтерфейс USB Габаритні розміри приблизно 101 х 53 х 12 мм.
Arduino Mini 	Мікроконтролер ATmega168 з тактовою частотою 16 МГц. Порт USB, EEPROM 512 біт, ОЗП 1 кбайт, Flash 16 кбайт (2 кбайт використовуються для початкового завантаження системи). 14 цифрових каналів вводу/виводу, 6 із них можуть використовуватись для ШІМ. Вісім аналогових входів із 10 розрядним АЦП.
Arduino Nano 	ATmega328 або 168 з тактовою частотою 16 МГц. Програмування через вбудований USB. Функція автоматичного скидання. 14 цифрових каналів вводу/виводу, 6 із них можуть використовуватись для ШІМ. Вісім аналогових входів із 10-розрядним АЦП. Flash-пам'ять об'ємом 32 кбайт або 16 кбайт. ОЗП 1 кбайт. EEPROM 512 байт або 1 кбайт. Габаритні розміри 18х43 мм.

Плата розширення Arduino Ethernet дозволяє підключати плату Arduino до локальної мережі та Інтернету. Вона побудована на контролері Wiznet W5100, що забезпечує підтримку протоколів TCP та UDP. Допустимо одночасне підключення до чотирьох

мережевих з’єднань. Програмне забезпечення Arduino включає бібліотеку та різні програми для виконання мережевих функцій.

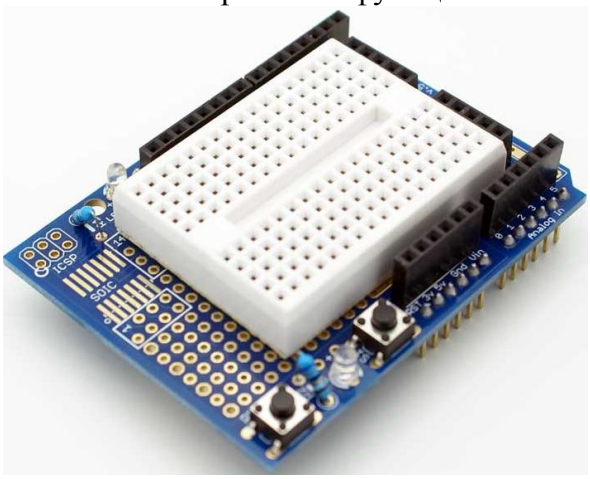


Рис. 5. Arduino ProtoShield

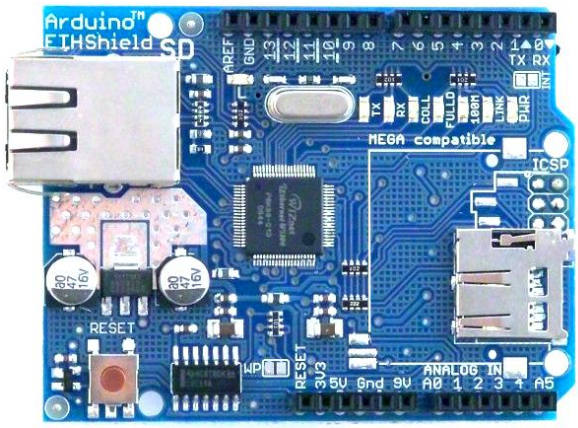


Рис. 6. Плата розширення Ethernet

Експериментальна плата мікроконтролера

Всі підключення експериментальної плати (рис. 7) можна здійснювати через роз'єми. Світлодіод "PWR" включення живлення сигналізує, що на плату мікроконтролера подається живлення. Світлодіод на платі, що постійно підключений до цифрового виводу 13 плати Arduino, позначений символом "L". Світлодіод з позначенням "TX" та "RX" показують роботу послідовного порту і мигають у процесі обміну даними. Детальна принципова схема плати Freeduino наведена у додатку 1.

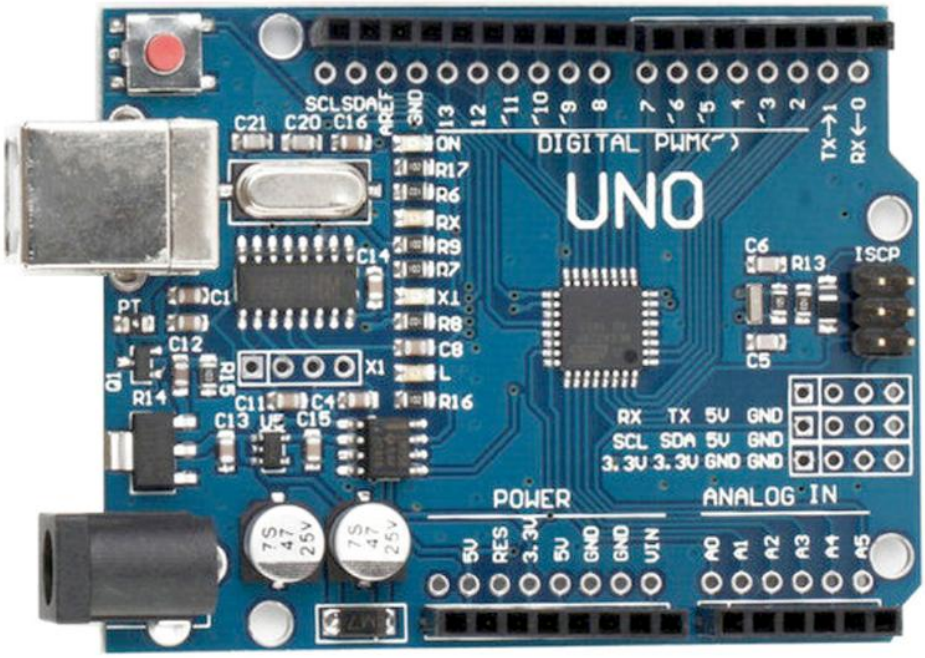


Рис. 7. Експериментальна плата

Живлення на плату може поступати через USB-роз'єм чи штекер блока живлення. Вибір джерела живлення здійснюється установкою перемички "PWR SEL" у положення "USB" чи "EXT". При великих навантаженнях варто використовувати зовнішнє живлення (9 - 15 В). Кнопка Reset (Скид) виконує пере завантаження системи. Те ж саме відбувається при виключенні та повторному включенні живлення. Порт ISP служить для програмування мікроконтролера через ISP-програматор для початкового програмування і як додатковий вивід послідовного інтерфейсу.

Середовище розробки Arduino

Програмування апаратних засобів Arduino зручно здійснювати у середовищі Arduino-IDE (Integrated Development Environment), проте, можливе програмування Arduino із використанням, наприклад, AVRStudio. Загальний вигляд вікна даної програми показаний на наступному рисунку.

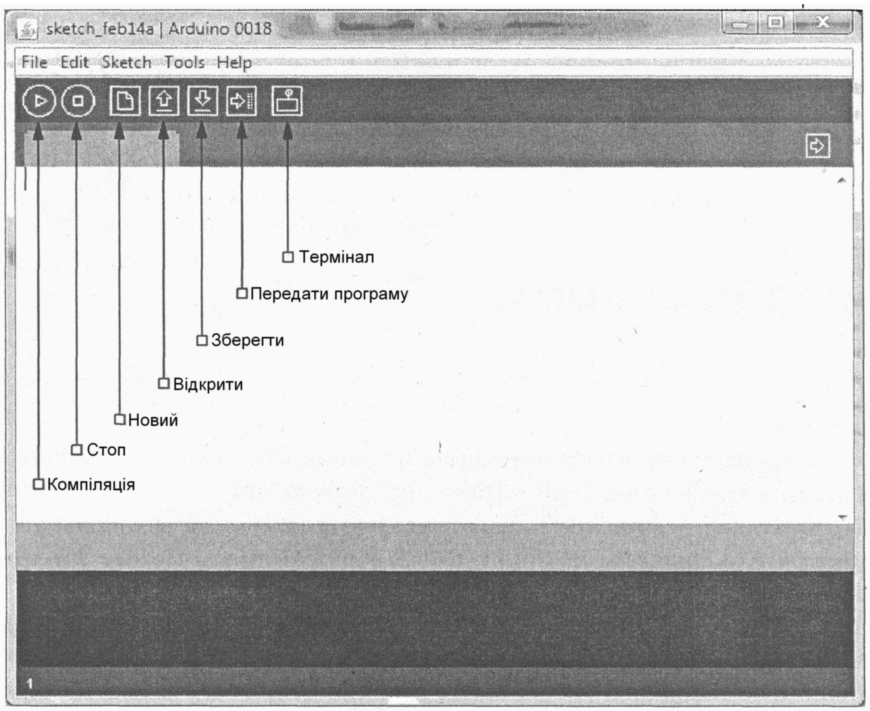


Рис. 8. Середовище розробки Arduino

Далі будуть описані окремі можливості програми. Під головним меню знаходиться панель інструментів, де розташовані такі команди:

- ◊ Компіляція - формування файлу, який буде переданий в плату мікроконтролера.
- ◊ Стоп - скасування компіляції.
- ◊ Новий - створення нового файлу Arduino.
- ◊ Відкрити - відкриття тексту програми.
- ◊ Зберегти - збереження тексту програми.
- ◊ Передати програму - передача програми в плату мікроконтролера.
- ◊ Термінал - відкрити вбудованого ASCII-терміналу.

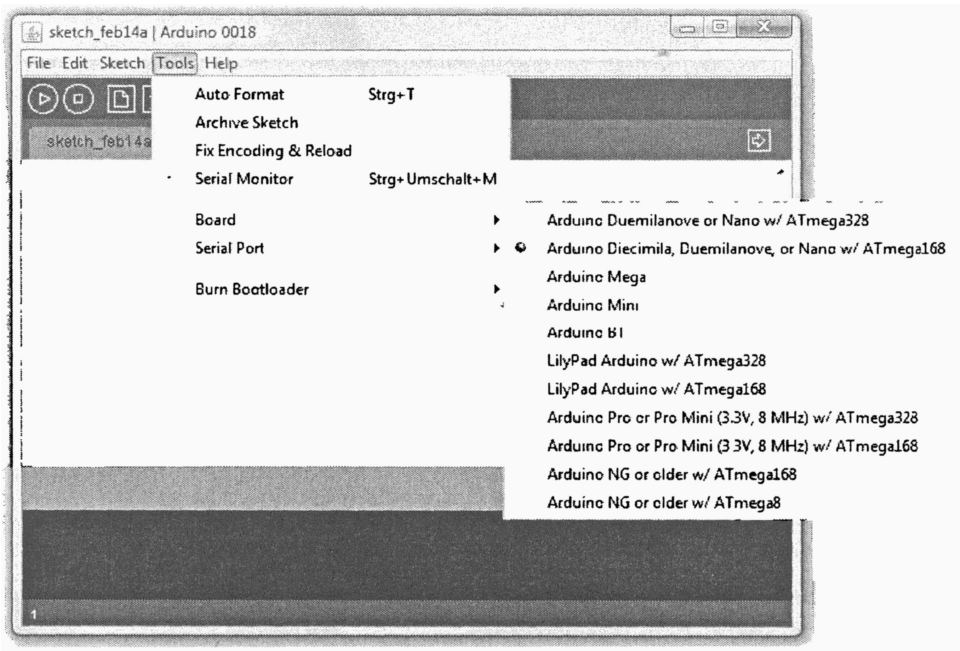


Рис. 9. Вибір плати мікроконтролера

До початку роботи необхідно задати вихідні установки. Для цього потрібно вибрати потрібну плату Arduino і використовуваний інтерфейс. В даному випадку вказана плата Arduino Diecimila. Якщо потрібна інша, то слід вибрати відповідну плату зі списку.

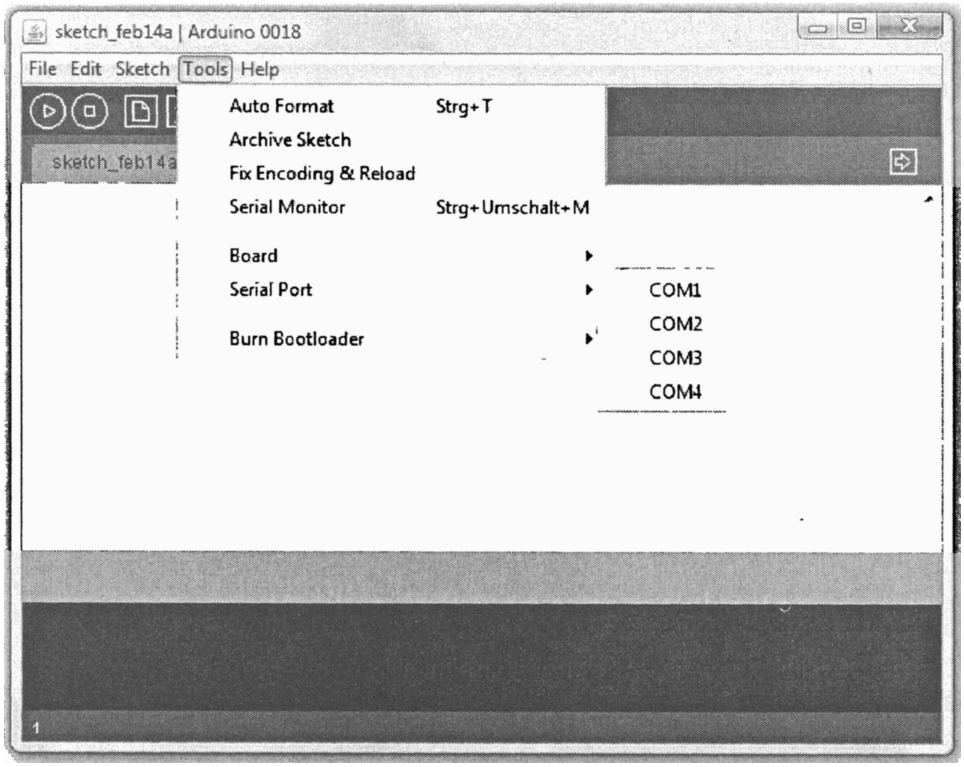


Рис. 10. Вибір послідовного інтерфейсу

Перед тим як відкрити середовище розробки (IDE) необхідно приєднати плату мікроконтролера до персонального комп'ютера і встановити драйвери належним чином. Інакше в списку не з'явиться COM-порт. Потім слід встановити швидкість в бодах в терміналі. Для цього потрібно натиснути піктограму Terminal і задати швидкість 9600 бод. Ця швидкість є типовою для більшості програм.

Для перевірки програмних і апаратних засобів, варто використати просту програму. При цьому спочатку у меню File знайдемо команду New, з’явиться нове вікно редагування.

Далі слід набрати код , що приведений нижче та зберегти його. Краще за все створити спеціальну папку, наприклад, Program_Arduino та вказати в налаштуваннях цю папку як каталог для збереження програм. При вказані назви, "Blink" у якості імені програми, ArduinoIDE створює підкаталог у обраному каталозі із іменем Blink і зберігає у ньому файл під іменем Blink.pde.

```
/*******************************
Світлова сигналізація
Використовується світлодіод "L" на платі Arduino
*******************************/
int ledPin =13;
// Світлодіод приєднується до цифрового виходу 13
// Стандартна підпрограма конфігурує наш цифровий порт
// Ця підпрограма виконується тільки один раз при запуску
void setup() {
// Порт конфігурується як вихід
pinMode(ledPin, OUTPUT);
// Основна програма - це нескінченний цикл
void loop()
{
digitalWrite(ledPin, HIGH);   // Включення світлодіода
delay(1000);                  // Очікування 1 секунду
digitalWritededPin, HIGH);    // Виключення світлодіода
delay(1000);                  // Очікування 1 секунду
}
```

Для перевірки програми слід натиснути в панелі інструментів круглу кнопку із стрілкою вправо або за допомогою гарячої комбінації <Ctrl>+<R> запустити трансляцію програми.

При відсутності повідомлень про помилку, можна передати програму в плату мікроконтролера, натиснувши на панелі інструментів квадратну кнопку із стрілкою вправо з крапками (чи натиснути комбінацію <Ctrl>+<U>). Після завантаження програми повинен почати моргати світлодіод "L" на платі мікроконтролера.

Основи програмування Arduino

Основний цикл програми та переривання.

При автоматизації обладнання код програми звичайно виконується у нескінченому циклі. На рис. 11 зліва показано виконання послідовності команд "ввід даних — обробка — вивід".

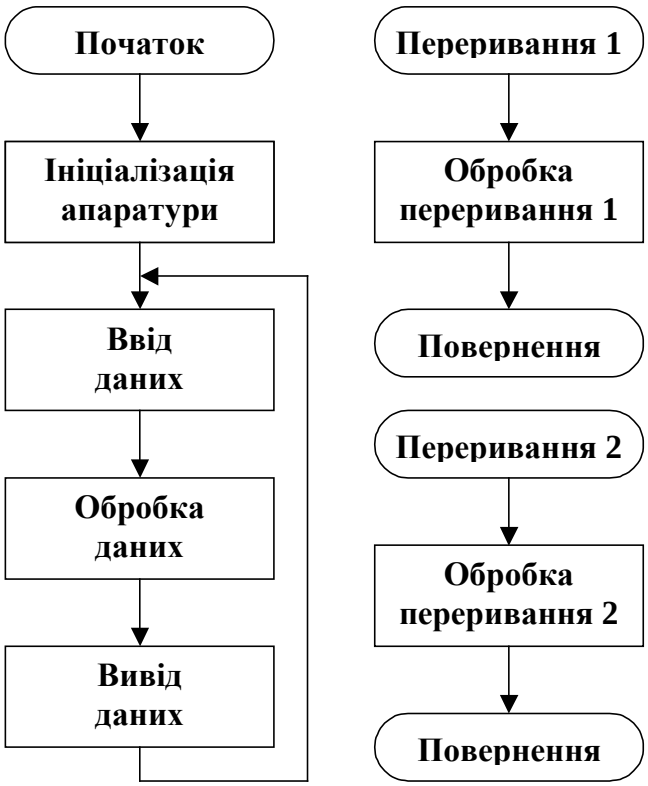


Рис. 11. Основний цикл та обробка переривань

Іноді виникає необхідність провести швидке обслуговування того чи іншого обладнання. Це здійснюється за допомогою переривань (Interrupt). Основна частина програми виконується як і при послідовному програмуванні, але коли відбудеться зовнішнє чи внутрішнє переривання, наприклад при натисканні кнопки, процесор перерве виконання основного циклу і відбудеться перехід на підпрограму обробки переривань (Interrupt-Routine). Після завершення оброки переривання контролер знову перейде до виконання основного циклу програми. Для успішного повернення у основну програму слід на початку підпрограми обробки переривання зберегти всі регістри процесора, що можуть бути змінені підпрограмою обслуговування, а у кінці підпрограми збережені значення мають бути відновлені.

Базова структура програми

Програма Arduino складається з наступних основних частин:

- ◊ Коментарі та опис програми.
- ◊ Заголовки файлів і підключення бібліотеки.
- ◊ Оголошення глобальних змінних.
- ◊ Стандартна настройка void setup () (порти і конфігурація).
- ◊ Основний цикл void loop ().
- ◊ Власні процедури.

Коментарі

Коментарі призначені для документування програми вони значно полегшують її читання. У середовищі Arduino текст, що заданий коментарем, виділений сірим кольором.

Коментар може бути

◊ одно стрічковим і починатись із символів //

```
x = 2 // Це простий коментар
```

◊ чи багато стрічковим і заключатись у дужки

```
/* Це довгий коментар              */
```

◊ або такі

```
/*************************************************************
*    Це коментар у три строки                                *
*************************************************************/
```

Операторні дужки

Операторні дужки виділяють фрагмент коду для компілятора. Блок коду завжди відкривається символом { і завершується символом }. Між ними можуть бути розміщені команди, опис змінних, тощо. Всі змінні, описи процедур, та інші об’єкти задані у дужках після закриття дужок стають недоступними для компілятора. Вважають, що операторні дужки задають область видимості для описаних параметрів. Область видимості простягається від моменту опису до закриття відповідної операторні дужки.

Типи даних і змінні

У середовищі розробки Arduino C є відмінність між великими та малими символами у імені змінної. Також не на першій позиції в Arduino у іменах змінних допускається вживання символу _ і цифр. Ключові слова (if, while, do і т.д.) не можуть бути іменами змінних. Імена глобальних змінних і функцій не можуть збігатися. Крім того, не допускається одночасне завдання функцій і локальних змінних з одним і тим же ім'ям. Для опису у розпорядженні програміста є різні типи змінних. Тип змінної завжди потрібно ставити перед її описом.

Якщо змінна оголошується в межах функції, процедури або як аргумент функції, вона є локальною. Це означає, що змінна існує тільки всередині своєї функції. Змінна, оголошена поза функцією, є глобальною. Вона визначена для всіх функцій в межах програми.

Приклад

```
byte Variable;
// Змінна типу byte, вона може набувати значень
// Від 0 до 255
float PI = 3.1415; // Константа PI оголошена як Float
int myArray[10]
//*********************************************************
// Масив Byte, оголошений складається
// З 10 елементів типу Int.
// До відповідного елементу звертаються через
// Покажчик: Var (x)
// Нумерація елементів масивів починається з 0
//*********************************************************
```

Стандартні типи даних

Void

Змінна типу Void не займає місце у пам'яті і служить лише для того, щоб вказати що процедура не повертає значення або для того щоб взяти адресу якогось об’єкта іншого типу у пам'яті.

Boolean

Змінна типу Boolean може мати два стани: true (істина) або false (неправда). Така змінна займає 1 байт в пам'яті. Значення true відповідає одиниці, а false - це стан, відмінний від логічної одиниці.

```
boolean Load = true; // Змінна істинна
```

Byte

Змінна типу байт має 8 біт, тому вона може приймати значення від 0 до 255 чи від -128 до 127. Для визначення діапазону використовують специфікатор signed для чисел із знаком та unsigned для чисел без знаку. За замовчування byte - число без знака. Тобто при описі

```
signed byte Var1 = 0;
unsigned byte Var2 = 0;
```

Змінна Var1 буде змінюватись у межах -128...127, а Var2 у межах 0...255.

Char

Char теж має розмір 1 байт. Змінна char — це символ в одиночних лапках. Якщо потрібна стрічка із декількох символів, то їх заключають у подвійні лапки (наприклад "ПРИВІТ"). За замовчуванням код символу char є число із знаком і представляє код у межах -128...127.

Якщо необхідно використати символи як числа без знака необхідно вказати специфікатор unsigned для чисел без знаку і специфікатор signed для чисел із знаком.

Int

Ціле число має довжину 2 байти. Число unsigned int t е числом без знака від 0 до 216 - 1, а signed int t е числом із знаком від -2^{15} до $2^{15} - 1$. За замовчуванням вважається, що int має знак, тобто за замовчуванням описи

```
int Var1 = 50000;
signed int Var1 = 50000;
```

еквіваленті.

Long

Змінна Long займає 4 байти, правила опису змінної аналогічні правилу опису змінної типу int.

Float

Змінна типу float призначена для збереження чисел із плаваючою комою і займає 32 бита.

Стрічка

Строкова змінна string — це масив змінних типу char, що закінчується символом із нульовим кодом. Таким чином, для слова "Hello" потрібно 6 байтів.

Масив

Масив — впорядкований набір змінних одного типу, кожна із яких має свій номер - індекс. Нумерація елементів масиву починається з нуля.

Опис масиву проходить у формі

```
тип_елемента назва_масиву[число_елементів];
```

Кваліфікатор volatile

Ключове слово (кваліфікатор) **volatile**, використовується перед типом змінної, щоб вказати компілятору про можливість зміни значення змінної за межами основної програми та підпрограм, що викликаються з неї: наприклад, якщо змінна відображає стан апаратного регістра, що зв'язаний х портом вводу даних і може змінювати стан залежно від стану обладнання, котре до неї під'єднано, або у відповідному апаратному перериванні.

Кваліфікатор static

Ключове слово **static** використовується для створення локальної змінної у функціях, що не знищується між викликами функції. Звичайні локальні змінні створюються і знищуються при кожному виклику функції, на відміну від них статичні змінні залишаються після виклику функції, зберігаючи свої значення між викликами. Наприклад таким чином можна з легкістю організувати лічильник разів виклику функції.

Функції та підпрограми

Призначення функцій та підпрограм - виділення окремих частин коду, що повторюється у програмі та розбиття програми на окремі незалежні фрагменти.

Синтаксис опису функцій та підпрограм єдиний

```
Тип_результату Ім'я(тип_параметра1 параметр1, тип_параметра 2 параметр2)
{
//Опис функції чи процедури
   return zz;  // значення змінної, що повертається
}
```

Якщо функція не повертає значення оператор return не потрібний. Вказати, що функція не повертає значення можна записом void у полі тип_результату. Можливий опис функцій із однаковим іменем і різним числом аргументів. Компілятор вважає такі функції різними. У випадку необхідності функції можуть мати аргумент за замовчуванням: у тамому випадку якщо при виклику функції аргумент не заданий замість нього підставляється значення задане при описі функції.

Приклад опису функції із аргументом за замовчуванням

```
int sum(int a, int, int c  = 0) //функція повертає суму 3 аргументів. Якщо третій
// аргументне заданий  - суму двух
{ return a + b + c;}
```

При використанні системи можна також використовуватиі вбудовані функції, деякі із них будуть описані нижче.

Функції перетворення типу

Функції char (), byte (), int (), long () и float () перетворюють тип змінної, тобто здійснюють явне приведення типів для подальший обчислень.

char() - перетворює значення в символ (char);

byte() - перетворює значення в byte.

int() - перетворює значення в integer.

long() - перетворює значення в Long.

Функції арифметичних операцій

Табл. 2. Функції арифметичних операцій

Функція	Опис
min(x, y)	Обчислює і повертає мінімум із двох значень x та y
max(x, y)	Обчислює і повертає максимум із двох значень x та y
abs(x)	Обчислює модуль числа x
constrain(x,a,b)	Приводить значення x у діапазон між a та b.
map(x, fromLow, fromHigh, toLow, toHigh)	Функція лінійної інтерполяції проміжного значення x. Межі зміни x задані величинами fromLow, fromHigh, межі зміни результату toLow, toHigh
pow(base, exp)	Піднесення числа base у степінь exp
sq(x)	Квадрат аргументу
sqrt(x)	Квадратний корінь числа x
sin(x)	Синус x, заданого у радіанах
cos(x) **tan(x)**	Косинус x, заданого у радіанах Тангенс x, заданого у радіанах.

Деякі спеціальні функції наведені у наступній таблиці.

Табл. 3. Деякі спеціальні функції

Функція	Опис
delay(ms)	Зупиняє програму на час, вказаний в мілісекундах
random(min, max)	Функція random здійснює генерацію псевдо випадкових значень в межах вказаного діапазону
randomSeed(seed)	Функція встановлює значення або початкове число для random
millis()	Функція повертає значення часу (в мілісекундах), що минув після останнього виклику функції. Переповнення виникає через кожні 70 хвилин.

Вирази та оператори

Кожному типу даних відповідають свої оператори, які вказують, які операції можуть застосовуватися. Значення виразу обчислюється із використанням пріорітету операцій та дужок, зазвичай зліва на право. Якщо значення виразу має бути збережено використовують оператори присвоювання, вказані у наступній таблиці.

Табл. 4

Оператор	Опис.
=	Присвоювання, наприклад A = B + C, змінній A буде присвоена сума значень змінних B та C.
+ =	Присвоювання з інкрементом, наприклад i + = 5, змінна i буде збільшена на 5 і результат присвоєно змінній i.
- =	Присвоювання з декрементом, наприклад i - = 5, змінна i буде зменшена на 5 і результат присвоєно змінній i.
* =	Присвоювання з множенням, наприклад i * = 2, змінна i буде помножена на 2 і результат присвоєно змінній i.
/ =	Присвоювання з діленням, наприклад i / = 2, змінна i буде розділена на 2 і результат присвоєно змінній i.

При виконанні оператора присвоювання спочатку обчислюється вираз у правій частині оператора, а потім здійснюється запис обчисленого значення (можливо із виконанням додаткових дій) у змінну, що знаходиться у правій частині оператора. Якщо необхідно, здійснюється перетворення типу результату операцій до типу змінної, у котру здійснюється присвоювання. Таке перетворення називається приведенням типів.

При обчисленні виразів можуть бути використані арифметичні, логічні, побітові оператори та оператори порівняння. При виконані складних виразів оператори ранжуються і виконуються згідно їх пріоритету. Можливі оператори у виразах вказані у наступній таблиці, операції, що мають більший пріоритет ідуть першими. Якщо операції мають однаковий пріоритет вони виконуються у порядку слідування - зліва на право.

Табл. 5. Оператори мови

Пріоритет	Позначення	Опис	Приклад
1	2	3	4
1	++	Суфіксний / постфіксний інкремент.	A = P++ A присвоюється значення i, а потім P збільшується на 1
	- -	Суфіксний / постфіксний декремент	A = P-- A присвоюється значення i, а потім P зменшується на 1
	()	Виклик функції	A = B(2) Викликається функція B() із аргументом 2, результат поміщається у змінну A, функція B повертає результат, котрий приводиться до типу A
	[]	Звертання до масиву по індексу	A = C[2] Змінній A присвоюється значення змінної із індексом 2 із масиву C
	.	Вибір елемента за посиланням	A = B.C Змінній A присвоюється значення поля C із структури B
	−>	Вибір елемента за вказівником	A = B->C Змінній A присвоюється значення поля C із структури на котру посилається B

Табл. 5. Оператори мови

Пріоритет	Позначення	Опис	Приклад
1	2	3	4
2	++	Префіксний інкремент	A = ++P P спочатку збільшується на 1, а потім значення присвоюється значення змінній A
	--	Префіксний декремент	A = --P P спочатку зменшується на 1, а потім значення присвоюється значення змінній A
	-	Унарний мінус (чи плюс)	A = -P A присвоюється значення -P
	!	Логічне НІ	A = !P Булева змінна P інвертується і поміщаються у змінну A. Значення P не змінюється
	~	Побітове НІ	A = !P Біти числа P інвертуються, а потім поміщаються у змінну A
	(type)	Приведення до типу type	Змінна явно приводиться до вказаного типу
3	*	Множення	A * B Обчислюється вираз A * B
	/	Ділення	A / B Обчислюється вираз A / B
	%	Остача від ділення	A % B Обчислюється остача від ділення A на B
4	+	Додавання	A + B Обчислюється вираз A + B
	−	Віднімання	A - B Обчислюється вираз A - B
5	<<	Побітовий зсув вліво	A << B Значення A зсуваєтьтся на B розрядів вліво
	>>	Побітовий зсув вправо	A >> B Значення A зсуваєтьтся на B розрядів вправо
6	<	Оператор порівняння <	A < B Значення A порівнюється із B якщо умова виконується результат рівний true, інакше false
	<=	Оператор порівняння ≤	A <= B Значення A порівнюється із B якщо умова виконується результат рівний true, інакше false
	>	Оператор порівняння >	A > B Значення A порівнюється із B якщо умова виконується результат рівний true, інакше false

Табл. 5. Оператори мови

Пріоритет	Позначення	Опис	Приклад
1	2	3	4
6	>=	Оператор порівняння ≥	A **>=** B Значення A порівнюється із B якщо умова виконується результат рівний true, інакше false
7	==	Оператор порівняння =	A == B Значення A порівнюється із B якщо вони рівні результат рівний true, інакше false
	!=	Оператор порівняння ≠	A != B Значення A порівнюється із B якщо вони не рівні результат рівний true, інакше false
8	&	Побітове І	A & B Результат рівний логічному побітовому І між A та B. Якщо A = 110, а B = 011, то результат буде рівний 010
9	^	Побітове виключаюче АБО (XOR)	A ^ B Результат рівний виключаю чому АБО між A та B - тобто біт результату рівний 1, якщо у відповідних бітах операндів різні двійкові числа Якщо A = 0110, а B = 1010, то результат буде рівний 1100
10	|	Побітове АБО (OR)	A | B Результат рівний АБО між A та B - тобто біт результату рівний 1, якщо хоча б у одного із операндів у відповідному розряді 1 Якщо A = 0110, а B = 1010, то результат буде рівний 1110
11	&&	Логічне І	A && B Результат рівний логічному І між A та B. Якщо A = true, а B = false, то результат буде рівний false
12	||	Логічне АБО	A || B Результат рівний логічному АБО між A та B. Якщо A = true, а B = false, то результат буде рівний true
13	?:	Тернарний оператор	(умова)?(вираз 1):(вираз 2) Якщо умова виконується обчислюється вираз 1, якщо ні вираз 2. Результат рівний обчисленому виразу
	=	Пряме присвоювання	A = B + C, змінній A буде присвоєна сума значень змінних B та C.

Табл. 5. Оператори мови

Пріоритет	Позначення	Опис	Приклад
1	2	3	4
13	+=	Присвоювання з сумуванням	P += 5, змінна P буде збільшена на 5 і результат присвоєно змінній P.
	−=	Присвоювання з відніманням	P -= 5, змінна і буде зменшена на 5 і результат присвоєно змінній P.
	*=	Присвоювання із множенням	P *= 2, змінна P буде помножена на 2 і результат присвоєно змінній P.
	/=	Присвоювання із діленням	P /= 2, змінна P буде розділена на 2 і результат присвоєно змінній P.
	%=	Присвоювання із остачею від ділення	P %= 2, змінна P буде розділена на 2 і остача від ділення буде присвоєна змінній P.
	<<=	Присвоювання із побітовим зсувом вліво	P <<= 2, значення змінної P буде зсунуто на 2 розряди вліво і записано у P.
	>>=	Присвоювання із побітовим зсувом вправо	P >>= 2, значення змінної P буде зсунуто на 2 розряди вправо і записано у P
	&=	Присвоювання із побітовою логічною операцією I	P &= B Над змінною P та B проведеться операція побітового I, результат запишеться у P.
	^=	Присвоювання із побітовою логічною операцією Виключаюче АБО	P ^= B Над змінними P та B проведеться операція побітового виключаючого АБО, результат запишеться у P.
	|=	Присвоювання із побітовою логічною операцією АБО	P |= B Над змінними P та B проведеться операція побітового АБО, результат запишеться у P.
14	,	Операція слідування.	P = (A,B) Спочатку виконується вираз A, потім B. Результат обчислення виразу A відкидається.

Директиви предпроцесора

Перед процесом компіляції зазвичай запускається спеціальна коротка програма, що носить назву предпроцесора. Вона призначена для здійснення макропідстановок, які визначаються так званими директивами. Кожна директива має стояти на початку рядка і починатись із символу #. Найбільш часто використовують директиви define та include.

Директива define створює макровизначення. Зазвичай її використовують для завдання констант:

```
#define variable 1
// Без крапки з комою! Змінна variable отримує значення 1.
// Всякий раз, коли в програмі зустрічається ім'я variable,
// воно замінюється на 1.
```

Проте директива define може бути використана для створення макросів, і навіть написання свого діалекту мови програмування, але докладний опис всіх можливостей директиви define виходить за межі даних методичних вказівок.

Директива include призначена для включення у програму макросів, визначень та кодів програм описаних у інших файлах. Формат директиви наступний:

```
#include "Filename.ext"
```

За директивою у процес компіляцій включається також файл Filename.ext, який поміщається у позицію де знаходилась директива. Ім'я файлу Filename.ext має бути охоплено дужками <> або подвійними лапками " ". У першому випадку файл, що включається шукається у системних каталогах бібліотек, у другому - у каталозі програми, що компілюється.

Крім того існують декілька інших директив. Зауважимо, що предпроцесор "нічого не знає" про роботу компілятора та вихідної програми, тому слід бути обережним при неочевидному використанні директив.

Керуючі конструкції

Для можливості реагувати на події та умови кожній програмі потрібні, керуючі конструкції. Вони вказуються за допомогою керуючих операторів if. . . else-if. . . else або switch case.

Оператор if

Оператор if - оператор умови. Призначений для перевірки умов і зміни поведінки програми залежно від умови. Якщо умова виконується код № 1, якщо ні код № 2.

```
if(умова)
{ /* код № 1, виконується якщо умова справджується   */}
else
{ /* код № 2, виконується якщо умова не справджується */}
```

Умова має бути виразом, що повертає (чи може повертати) результат типу boolean. Зауважимо, що слово else і код № 2 у дужках за ним можуть бути відсутні. У разі коли код складається із одного оператора операторні дужки можуть бути пропущені.

```
if(x!=10){ y = 5;}
else y = 2;
```

Часто потрібно при невиконанні одної умови відразу перейти до аналізу наступної. Наприклад буває потрібно визначити чи змінні рівні або одна змінна більша за другу, а потім залежно від того яка змінна більше виконати ту чи іншу дію. Така конструкція може бути легко реалізована за допомогою спеціальної модифікації оператора else if. Умови A == B та A < B можуть бути замінені на інші.

```
if(A == B)
{ /* Код № 1 запускається якщо A = B */ }
else if (A < B)
{ /* Код № 2 запускається якщо A < B */ }
else
{ /* Код № 3 запускається якщо A > B */ }
```

Приклад

```
int x = 22;
int y = 33;
void setup()
{ Serial.begin(9600);
   Serial.println();
}
void loop()
{  if(x == y)
   {Serial println("X = Y");}
   else if (x < y)
   { Serial println("X < Y");}
```

```
   else
   { Serial println("X > Y");};
}
```

Оператор switch - case

Конструкція switch - case веде себе подібно оператору else if, проте керуюче оператором значення обчислюється один раз і значення умови перевіряється на рівність константам.. У операторі можна задати альтернативний варіант, при відсутності відповідності умов у середині оператора case. Вихід із оператора case здійснюється при допомозі оператора break. При відсутності оператора break буде виконана наступна порція коду.

Синтаксис оператора switch case

```
switch(Вираз)
{
case 1:
/* Код № 1, що запускається якщо вираз рівний 1 */
break;
case 2:
/* Код № 2, що запускається якщо вираз рівний 2 */
break;
case 4:
/* Код № 3, що запускається якщо вираз рівний 4 */
case 5:
/* Код № 4, що запускається якщо вираз рівний 4 або 5 */
break;
default:
//Код  запускається, якщо всі попередні умови не виконались
//Тобто якщо вираз < 1, або вираз = 3 або вираз > 5
}
```

Цикли

При програмуванні часто потрібні цикли, щоб забезпечити обробку масивів, стрічок даних, тощо. Існує декілька типів циклів.

Цикл for

Цикл for є найбільш універсальним із існуючих у програмному середовищі. Синтаксис циклу for наступний

```
for(іціалізація; умова; дії у кінці кожної ітерації)
{ /* Тіло циклу */; }
```

Ініціалізація - блок ініціалізації змінних виконується один раз на початку циклу.

Умова - вираз, що дає результат типу boolean. Якщо умова виконалась, тобто результат true - цикл виконується ще один раз. Якщо умова не виконалась (false) необхідно завершити цикл.

Дії у кінці кожної ітерації виконуються наприкінці кожного проходження циклу і зазвичай підготовлюють змінні до наступної ітерації.

Приклад: цикл виконується 10 разів від 0 до 10 з кроком 1

```
for( x = 0 ; x < 11 ; x++)
{/* Код що виконується 10 разів */; }
```

Зауважимо, що будь-яка частина циклу у круглих дужках може бути відсутня. Якщо відсутня умова вважається, що вона дає результат true.

Вийти із циклу можна і у його середини за допомогою оператора break.

```
int i = 0;
for(;;)
{/* Код виконується 10 разів */;
  //Щось виконуємо
  if(i > 9) {break;};
```

```
  i++;
  //Ще щось виконуємо
}
```

Цикли while та do while

Цикл while є більш простим варіантом циклу for. Формат циклу показаний на нижче.

```
while(умова)
{
  //тіло циклу
}
```

Цикл виконується лише якщо умова справджується. Після виконання один раз циклу знову перевіряється умова. Якщо вона і далі справджується цикл виконується знову.

Ще один варіант циклу - цикл do while.

Формат циклу наступний:

```
do {
  //тіло циклу
}
while(умова)
```

На відміну від попереднього разу тут спочатку виконується тіло циклу, а потім перевіряється умова повторення. Якщо умова справджується цикл виконується ще раз.

Приклад

```
var = 0;
while(var < 200){
  // виконати щось 200 разів
  var++;}
```

Ввід-вивід

У середовищі програмування Arduino відсутні операції вводу-виводу - ситуація аналогічна мові програмування С. Для вводу та виводу даних необхідно скористатись спеціальними стандартними (чи нестандартними) бібліотеками.

При використанні найбільш поширених бібліотек можливий ввід із таких джерел даних:

- ◊ ввід двійкового коду через входи портів вводу-виводу
- ◊ ввід значення аналогових сигналів
- ◊ ввід значень через послідовний порт

Також можливий вивід даних із таких джерел:

- ◊ вивід двійкового коду через входи портів вводу-виводу
- ◊ вивід значення аналогових сигналів за допомогою широтно-імпульної модуляції
- ◊ вивід значень у послідовний порт.

Ввід та вивід двійкового коду через порти вводу-виводу

Найпростішим типом вводу-виводу є ввід та вивід цифрових двійкових даних через порти вводу. Для побітового вводу та виводу даних можна використовувати наступні функції pinMode, digitalRead, digitalWrite. Якщо необхідно вивести чи зчитати відразу цілий байт можна використовувати команди які напряму звертаються до регістрів мікроконтролера DDRx, PINx, та PORTx.

Настроювання ліній на ввід чи вивід даних

Перед вводом чи виводом даних необхідно вказати мікроконтролеру як настроїти порт, звичайно це здійснюється у підпрограмі void setup() за допомогою функції

```
pinMode(pin, mode).
```

pin задає номер контакту, що конфігурується,

mode вказує спосіб використання контакту і може приймати значення:

OUTPUT - вивід настроєний на вивід;

INPUT - вивід настроєний на ввід.

Виводи, що конфігурується як вихід, можуть приймати і віддавати струм до 20 мА. У ATMega також кожний контакт має вбудований підтягуючий резистор (20 кОм), що може бути включений програмно за допомогою функції digitalWrite:

```
pinMode(pin, INPUT);     // Вивід встановлюється як вхід
digitalWrite(pin, HIGH);// Підключається підтягуючий резистор
```

Підтягуючий резистор підключають, замість зовнішнього резистора при використанні кнопок та інших пристроїв, що комутують вхід на землю.

Зчитування стану лінії

Для зчитування стану даних вхідної лінії використовується функція

digitalRead(pin)

вона зчитує значення виводу із результатом high чи low, що відповідає 1 та 0. Номер виводу може встановлюватись як змінна чи константа.

Приклад:

```
value = digitalRead(Pin);
```

Встановлення стана лінії

Встановлення стана лінії здійснюється функцією digitalWrite.

Формат функції такий

digitalWrite(pin, val);

Номер виводу вказується як змінна чи константа у pin, стан лінії - у змінній чи константі val.

У наступному прикладі зчитується стан кнопки, підключеної до цифрового входу (контакт 12), і результат відображається за допомогою світлодіода "L". При натисканні кнопки світлодіод гасне.

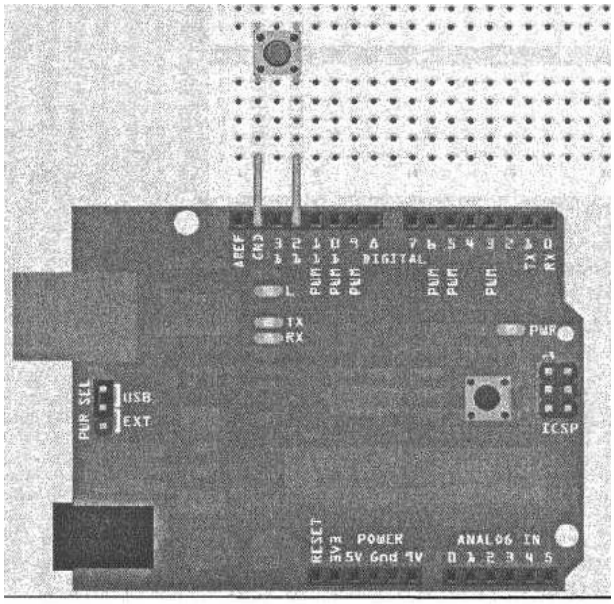


Рис. 12. Макет для роботи програми

Приклад програми

```
int led=13; int pm=12; int value=0;
void setup()
{
  pinMode(led, OUTPUT);
  pinMode (pin, INPUT) ;
  digitalWrite(pin, HIGH) ; //#####
}
void loop() {
  value = digitalRead(pm) ;
 digitalWrite (led, value) ;
}
```

Можна використати схему із зовнішнім підтягуючим резистором. Тоді команда позначена коментарем ##### не потрібна.

Послідовний ввід/вивід

Мікроконтролер може посилати, та приймати дані від комп'ютера чи інших пристроїв через послідовний порт UART (Universal Asynchronous Receiver Transmitter).

Для роботи із портом використовується декілька функцій.

`Serial.begin(Baudrate)`

Функція Serial.begin() дозволяє роботу послідовного інтерфейсу та його швидкість роботи. Константа чи змінна Baudrate задає швидкість передачі у бодах (фактично у бітах за секунду). Можливі наступні швидкості передачі: 300, 1200, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115 200. При послідовному зв'язку цифрові виводи, що пов'язані із лініями RX (0) та TX (1) не можуть використовуватись для простого вводу-виводу даних

`Serial.end()`

Завершує роботу послідовного порта. Лінії порта переводяться у режим простого цифрового вводу-виводу.

`long Serial.print() та Serial.println()`

Функції призначені для виводу даних у ASCII коді. Можуть виводити строки даних, цілі, булеві, символьні змінні та змінні із плаваючою комою. Функція Serial.println() після виводу вказаних даних добавляє символи переводу строки. Функція повертає число байтів, котрі необхідно вивести у результаті операції. Формат функції показаний нижче:

`Serial.print(val, format) або Serial.print(val)`

val - змінна, що виводиться

format - для цілих чисел двійковий (BIN), десятковий (DEC - за замовчуванням) чи шістнадцятьковий (HEX) формат виводу, для чисел із плаваючою комою - число десяткових знаків у числі.

```
Serial.print(" Dec ");   // Вивід стрічки даних
Serial.print(x);         // Вивести X як десяткове число
Serial.print(" is BIN ");// Вивід стрічки
Serial.println(x, BIN);  // Вивести X як двійкове число і
                         // перевести строку
```

`long Serial.write(val)`

Видає у послідовний порт дані у двійковому форматі. Тобто внутрішнє представлення даних. Функція повертає число байтів, котрі необхідно вивести у результаті операції. Формат функції показаний нижче:

`Serial.write(val, len) або Serial.write(val)`

val - змінна, що виводиться

len - число байтів, що необхідно вивести із змінної.

`int Serial.available()`

Вказує скільки символів у буфері послідовного порту.

`int Serial.read()`

Зчитує перший байт із прийнятих даних послідовного порту. Якщо даних немає - функція повертає -1.

`string Serial.readString()`

Функція зчитує строку, що прийшла у буфер та повертає її як результат. Стрічка може бути проаналізована та із неї можуть бути виділені потрібні змінні.

`int Serial.parseInt() та float parseFloat()`

Функція виділяє із послідовного буфера ціле чи дійне число і повертає їх. Функція працює без очікування надходження чисел. Якщо числа не має видається 0.

Приклад зчитування цілого числа з послідовного порту

```
val = Serial.parseInt();
```

Аналоговий ввід даних та АЦП

Для вимірювання аналогової напруги плата Arduino має внутрішній 10-розрядний АЦП тобто роздільна здатність АЦП (крок квантування) рівний напрузі 0,0048 В при

опорній напрузі $V_{ref} = 5$ В. Плата мікроконтролера має 6 аналогових входів, що підключаються до одного внутрішнього АЦП.

Значення напруги на вході АЦП визначається як

$$U_{in} = \frac{N \cdot V_{ref}}{1024}$$

Для роботи із аналоговим входом використовується функція

```
analogRead(pin)
```

Функція зчитує значення установленого аналогового входу з роздільною здатністю 10 біт, вона доступна для виводів 0-5:

```
value = analogRead(pin); // встановлює value рівній значенню
                         //напруги на вході
```

Аналогові виводи, на відміну від цифрових не потрібно програмувати на вхід чи вихід. Проте можна змінювати джерело напруги V_{ref} за допомогою функції

```
analogReference(ref)
```

Змінна чи константа ref може приймати значення

DEFAULT - опорна напруга рівна напрузі живлення 5 чи 3.3 В

INTERNAL - опорна напруга рівна напрузі 1.1 В для ATmega168 чи ATmega328 та 2.56 В для ATmega8

INTERNAL1V1 - опорна напруга рівна напрузі 1.1 В (для Arduino Mega)

INTERNAL2V56 - опорна напруга рівна напрузі 2.56 В (для Arduino Mega)

EXTERNAL - опорна напруга подається на вхід AREF.

Приклад програми для читання даних із входу 0 АЦП і видачі їх на послідовний порт

```
int ADC0 = 0;
int value;
int LEDpin = 13;
void setup() {
  Serial.begin(9600);}
void loop() {
  value = analogRead(ADCO);
  Serial.print("ADCO = "),
  Serial.println(value);
  delay(1000); }
```

Значення відліку передається кожну секунду на послідовний порт. Якщо помножити значення на масштабуючий множник отримаємо значення у вольтах

Аналоговий вивід даних

На платі Arduino знаходяться шість сигналів з широтно-імпульсною модуляцією (ШІМ) - контакти 3, 5, 6, 9, 10 та 11. Вони можуть використовуватись для цифро-аналогового перетворення. У процесі ШІМ змінюється відношення включеного стану лінії до періоду їх повторення - скважність імпульсів. Зауважимо, що на більш старих Arduino на основі ATmega8 ШІМ наявний лише на контактах 9, 10 та 11.

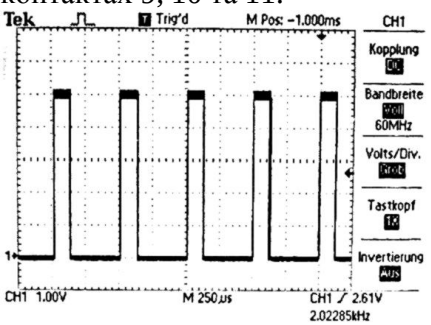


Рис. 13. Скважність 25%

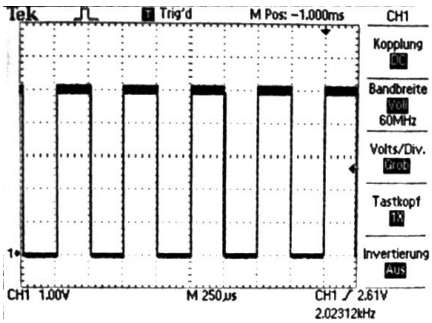


Рис. 14. Скважність 50%

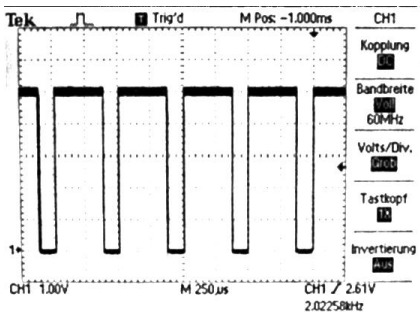


Рис. 15. Скважність 75%

Фільтр потім виділяє із сигналу низькочастотну складову, що пропорційна значенню, котре записано у відповідний порт. Для запису аналогових значень використовується функція

```
analogWrite(pin, value)
```

Змінна чи константа pin вказує вивід через котрий здійснюється вивід значення, value вказує значення величини, що виводиться. Вона має бути у діапазоні 0-255. Нульовому значенню відповідає нульова напруга на виході, 255 - максимальна напруга.

Завдання

Написати програму для автоматизації вимірювання характеристики деякого приладу. На вхід приладу поступає аналогова напруга у діапазоні 0-5 В. За допомогою Arduino необхідно видавати цю напругу та контролювати значення, що знімається із вихідних затискачів приладу. Прилад може давати вихідну напругу у діапазоні 0-5 В. Дані, що вимірюються направляються за допомогою послідовного інтерфейсом у ПК. Процес вимірювання починається при натиснутій кнопці S1.

Виводи підключення приладу та закон зміни вхідної напруги задається викладачем (у програмі необхідно передбачити легкий метод зміни закону та виводів)

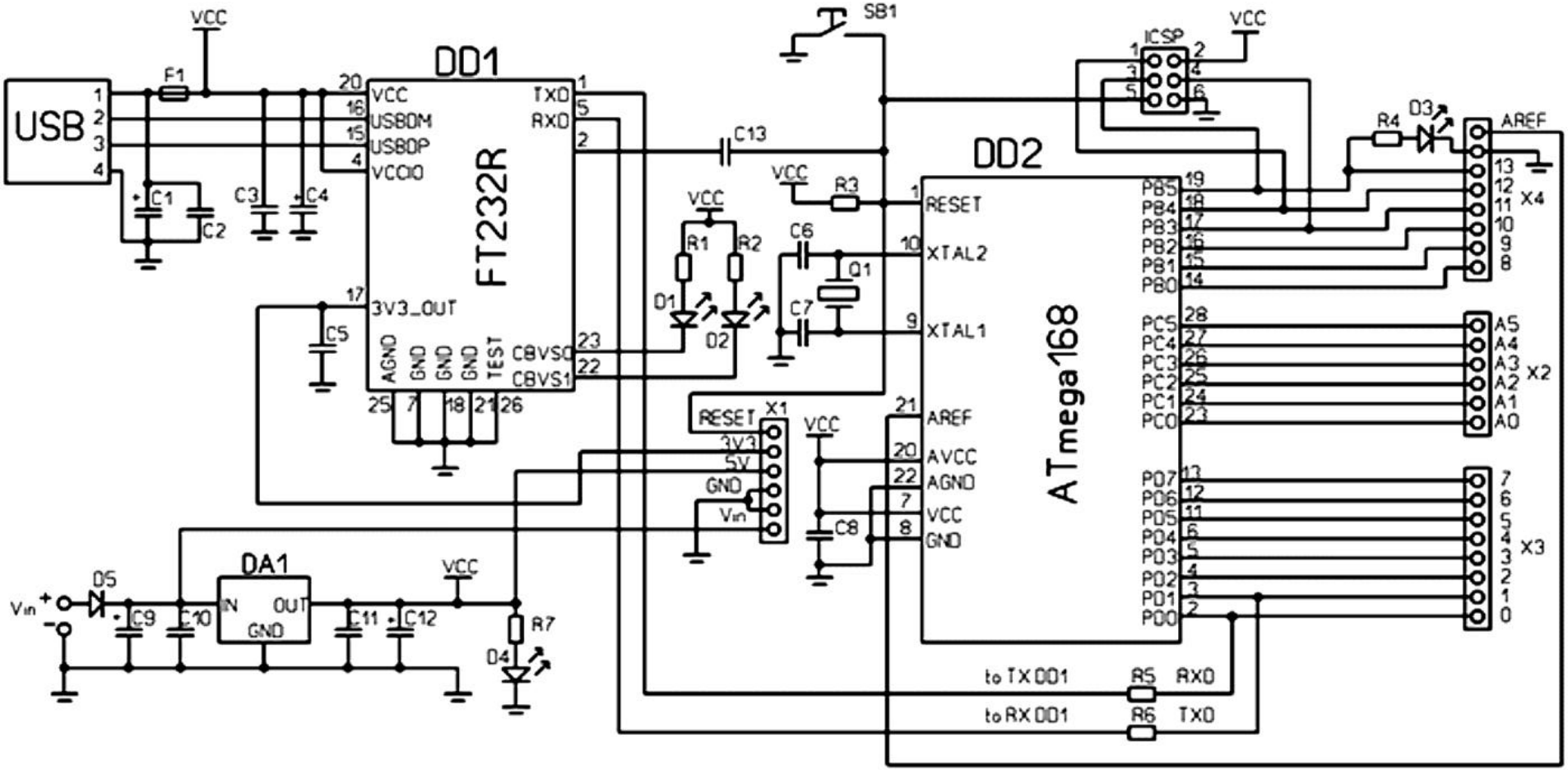
Роботу написаної програми продемонструвати викладачу.

Література

1. Соммер У. Программирование микроконтроллерных плат Arduino/Freeduino. — СПб.: БХВ-Петербург, 2012. — 256 с. ил — (Электроника)

2. Юревич Е.И. Проектирование технических систем. СПб.Питер 2001. 96 с.

Додаток 1. Принципова схема експериментальної плати Arduino Uno



Зміст