МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ МОРСЬКИЙ УНІВЕРСИТЕТ

Кафедра «Технічна кібернетика й інформаційні технології

ім. проф. Р.В. Меркта»

Затверджено
НМК Інституту інформаційних
технологій та іноваційного
підприємництва
Протокол № 3 від 04.12.2025
Керівник НМК

_____ Андрій ІВАНОВ

«4» грудня 2025 р.

Методичні вказівки

до лабораторних робіт

з дисципліни «Основи концепції інтернет речей».

Частина 1. Дротові технологій інтернету речей.

для здобувачів вищої освіти спеціальності F3 «Комп'ютерні науки»

першого (бакалаврського) рівня для денної та заочної форми навчання.

Одеса 2025

Методичні вказівки до лабораторних робіт з дисципліни «Основи концепції інтернет речей» Частина 1. Дротові технології інтернет речей підготовлени Ларіним Дмитром Георгійовичем, к.т.н., доцентом, та Тузовим Олександром Валерійовичем, старшим викладачем кафедри «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Меркта» Одеського національного морського університету.

Методичні вказівки до лабораторних робіт з дисципліни «Основи концепції інтернет речей» Частина 1. Дротові технології інтернет речей схвалено кафедрою «Технічна кібернетика й інформаційні технології ім. проф. Р.В. Меркта» ОНМУ

«_06_»__11__ 2025 року, протокол №_8__

Завідувач кафедрою _____ Павло НОСОВ

Зміст

Лабораторна работа “Середовище розробки Arduino IDE”

Теоретичні положення

Інтерфейс середовища розробки Ардуіно (завантажити https://www.arduino.cc/en/main/software) містить наступні основні елементи: текстовий редактор для написання коду, область для виведення повідомлень, текстова консоль, панель інструментів з традиційними кнопками і головне меню. Даний софт дозволяє комп'ютеру взаємодіяти з Ардуіно як для передачі даних, так і для прошивки коду в контролер.

Написання програм

Програми, створювані в середовищі розробки Ардуіно, іноді ще називають скетчами. Скетчі пишуться в текстовому редакторі і зберігаються у файлах з розширенням .ino. Вбудований текстовий редактор має стандартні інструменти копіювання, вставки, пошуку і заміни тексту. Область повідомлень у вікні програми є, свого роду, зворотним зв'язком для користувача і інформує його про події (в тому числі і про помилки), що виникають в процесі запису або експорту написаного коду. Консоль відображає у вигляді тексту потік вихідних даних середовища Ардуіно, включаючи всі повідомлення про помилки та ін. генерується нею інформацію. У нижньому правому куті вікна програми показується модель поточної плати і послідовний порт, до якого вона підключена. Кнопки на панелі інструментів призначені для створення, відкриття, збереження і прошивки програм в пристрій. Окрема кнопка запускає програму **SerialMonitor**.

Verify (Перевірити)

Перевірити код на помилки.

Upload (Прошити)

Скомпілювати програму і "зашити" її в мікроконтролер

Ардуіно. Детальніше про прошивці-див. нижче.

Примітка: щоб прошити мікроконтролер через зовнішній програматор - потрібно затиснути клавішу "shift" перед натисканням на цю іконку. При цьому текст біля кнопки зміниться на"Upload using Programmer".

New (Створити)

Додаткові команди знаходяться в меню: File, Edit, Sketch, Tools і Help. У цих меню завжди активні тільки ті пункти, які можна застосувати до поточного елементу або фрагменту коду.

Меню "Edit (Правка)"

* Copy for Forum (скопіювати для форуму)

Скопіювати код програми в буфер обміну в спеціальному форматі, зручному для постингу на форум (з підсвічуванням синтаксису).

* Copy as HTML (скопіювати як HTML)

Скопіювати код програми в буфер обміну у вигляді HTML-коду, зручного для вбудовування в веб-сторінки.

Меню " Sketch (Програма)"

* Verify / Compile (Перевірити / Компілювати)

Перевірити код програми на помилки.

* Show Sketch Folder (показати папку Програми)

Відкрити папку з файлом поточного скетча.

* Add File... (Додати файл...)

Додає вихідний файл до поточної програми (вибраний файл буде скопійований з вихідної папки). Доданий файл з'явиться в новій вкладці головного вікна. Видалити файли можна за допомогою таб-меню.

* Import Library (імпортувати бібліотеку)

Додає бібліотеку до вашої програми шляхом вставки оператора #include спочатку коду. Докладніше про бібліотеки див.нижче. Крім того, у версіях IDE 1.0.5 і вище реалізована можливість імпортування бібліотек з .zip-архівів.

Меню " Tools (Інструменти)"

* Auto Format (Автоформат)

Ця команда наводить красу у вашому коді, а саме: робить однакові відступи відповідних відкривають і закривають фігурних дужок, Додаткові відступи коду всередині логічних блоків.

* Archive Sketch (заархівувати скетч)

Утворити .zip-архів поточного скетча. Результуючий архів поміщається в папку з програмою.

* Board (Плата)

Вибрати модель використовуваного Ардуіно. Опис різних плат Ардуіно див.нижче.

* Serial Port (послідовний порт)

Це меню містить список всіх послідовних пристроїв, присутніх в системі (як фізичних, так і віртуальних). Їх список повинен оновлюватися автоматично при кожному відкритті головного меню.

* Programmer (Програматор)

Дозволяє вибрати зовнішній програматор для прошивки мікроконтролера без використання USB-з'єднання. Зазвичай ця функція потрібна рідко-наприклад, для прошивки завантажувача в новий Мікроконтролер.

* Burn Bootloader (прошити завантажувач)

Це меню дозволяє прошити завантажувач в контролер Ардуіно. При нормальній роботі Ардуіно ця функція зазвичай не потрібно, але вона може бути корисна, якщо вам раптом потрібно замінити Мікроконтролер ATmega на новий (який з магазину йде без завантажувача). Перед прошивкою переконайтеся, що в меню Boards обрана саме ваша плата.

Скетчбук (робоча папка)

У середовищі розробки Ардуіно використовується принцип організації скетчбука: всі ваші програми (або скетчі) зберігаються в одному місці. Щоб

переглянути їх, необхідно вибрати меню File > Sketchbook або клацнути по кнопці Open на панелі інструментів. Директорія для зберігання ваших програм буде автоматично створена при першому запуску середовища Ардуіно. Її місце розташування завжди можна змінити у вікні налаштувань програми.

Починаючи з версії 1.0, файли скетчів мають розширення .ino. У попередніх версіях використовувалося розширення .pde. У нових версіях програми (1.0 і старше) файли .pde як і раніше можна відкрити, тільки їх розширення буде автоматично перейменовано на .ino.

Вкладки, компіляція і робота з декількома файлами

Середовище Ардуіно дозволяє працювати з програмами, що складаються з декількох файлів (кожен з яких відкривається в окремій вкладці). Наприклад, це можуть бути файли Ардуіно (без розширення), С-файли (з розширенням .c), файли c++ (з розширенням .cpp) або заголовні файли (.h).

Прошивка

Перед тим, як прошивати програму в контролер, необхідно правильно вибрати плату і послідовний порт в меню Tools > Board і Tools > Serial Port відповідно. Різновиди плат Ардуіно перераховані нижче. Послідовний порт на Mac-системах буде виглядати приблизно так: /dev/tty.usbmodem241 (для Arduino Uno, Mega2560 або Leonardo) або /dev/tty.usbserial - 1B1 (для Duemilanove або старіших версій Ардуіно з USB), або /dev/tty.USA19QW1b1P1.1 (для USB-UART перетворювачів Keyspan). На Windows-системах, необхідний порт швидше за все буде COM1 або COM2 (для пристроїв з послідовним інтерфейсом), або COM3, COM4, COM5 і вище (для Ардуіно з USB) - визначити необхідний номер порту можна в диспетчері пристроїв, відшукавши рядок "USB serial device" в розділі порти. На Linux-

системах послідовний порт буде виглядати як /dev/ttyUSB0, /dev/ttyUSB1 або щось на зразок.

Після вибору використовуваного порту і плати, необхідно натиснути кнопку Upload на панелі інструментів або вибрати пункт Upload з меню File. Після цього відбудеться скидання Ардуіно і почнеться процес завантаження програми в пам'ять контролера. У старих моделях (до Arduino Diecimila) функція авто-скидання відсутня, тому перед прошивкою таких пристроїв необхідно вручну натиснути кнопку Скидання на платі. У процесі завантаження на більшості моделей Ардуіно будуть блимати світлодіоди RX і TX. По завершенню процесу прошивки, програма видасть відповідне повідомлення або помилку.

Завантаження програми в Ардуіно здійснюється за допомогою завантажувача - невеликої програми, прошитої в пам'яті мікроконтролера, яка дозволяє завантажувати в нього код без зовнішніх апаратних засобів. Завантажувач активізується на кілька секунд після скидання пристрою, після чого він запускає на виконання останній завантажений в контролер скетч. При запуску завантажувача буде блимати вбудований світлодіод, підключений до 13 ніжці контролера.

Бібліотека

Бібліотеки розширюють функціональність програм і несуть в собі додаткові функції, наприклад, для роботи з апаратними засобами, функції по обробці даних і т.д. для підключення бібліотеки необхідно вибрати її з меню Sketch>Import Library. Після цього бібліотека буде скомпільована, а в початок програми буде додано один або кілька операторів #include. Пам'ятайте, що бібліотеки завантажуються в контролер разом зі скетчем, тому кожна підключена бібліотека треубет додаткового місця в пам'яті мікроконтролера. Відповідно, якщо та чи інша бібліотека більше не використовується у вашій програмі - просто видаліть оператор #include з програми.

Список основних бібліотек, описаних на сайті, наведено тут. Деякі з них встановлюються разом із середовищем розробки Ардуіно, інші можна завантажити з різних джерел. У версіях IDE 1.0.5 і старше реалізована можливість імпорту бібліотек прямо з Zip-архіву і підключення їх до поточного скетчу. Інструкції щодо встановлення сторонніх бібліотек.

Програма " Serial Monitor"

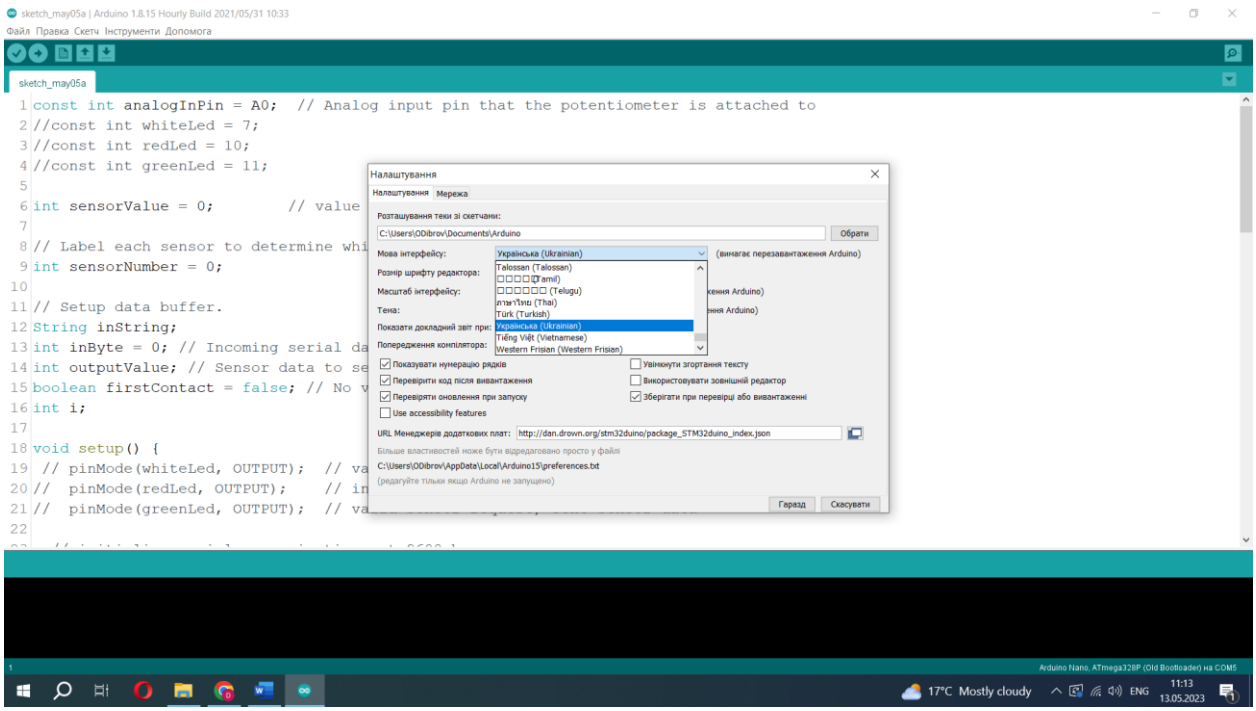
Відображає дані, що надходять від Ардуіно на комп'ютер по послідовному інтерфейсу (підтримується робота як з USB, так і зі звичайними версіями Ардуіно). Щоб відправити дані зовнішньому пристрою, досить просто ввести текст у вікні програми і клацнути по кнопці "Відправити" (або натиснути Enter). Зі списку, що випадає необхідно вибрати тільки швидкість передачі даних, відповідну тій швидкості, яку Ви вказали у функції Serial.begin() у вашому скетчі. Пам'ятайте, що на Mac і Linux системах, Ардуіно буде скидатися при кожному підключенні програми до пристрою (відповідно, після скидання скетч буде виконуватися заново).

Спілкуватися з Ардуіно можна також через Processing, Flash, MaxMSP та ін.

Налагодження

Деякі параметри можна задати безпосередньо у вікні налаштувань програми (на Mac-системах це вікно викликається з меню Arduino, на Windows і Linux-системах - з меню File). Інші параметри знаходяться в конфіг-файлі, місцезнаходження якого також зазначено у вікні налаштувань.

Підтримувані мови



Середовище розробки Ардуіно переведена більш ніж на 30 різних мов. За замовчуванням, мова IDE вибирається виходячи з мовних налаштувань вашої операційної системи. (Зверніть увагу: на Windows і Linux-системах мова IDE визначається за регіональними настройками, що відповідають за формат дати і валюти, а не за мовою самої операційної системи).

Якщо ви хочете вручну змінити поточну мову програми, запустіть середовище Ардуіно і відкрийте вікно налаштувань. У полі Editor language з'явиться список підтримуваних мов. Виберіть зі списку бажану мову і перезапустіть програму, щоб зміни вступили в силу. Якщо вибрана вами мова не підтримується, то за замовчуванням IDE довантажить англійську локалізацію.

Щоб скинути мовні настройки середовища і повернути автоматичний вибір її мови по регіональних налаштувань операційної системи, в випадаючому списку необхідно вибрати пункт System Default. Зміни набудуть чинності після перезапуску IDE Arduino. І навпаки, щоб зміна мовних налаштувань операційної системи вплинули на поточний мову програми, необхідно просто перезапустити середу Ардуіно.

Різновиди плат

Вибирати модель використовуваної плати в середовищі Ардуіно необхідно з двох причин:

1. Щоб задати параметри, використовувані під час компіляції і прошивки скетчів (такі, як тактова частота, бодрейт і ін.);
2. Щоб задати налаштування фьюз-бітів, використовувані під час прошивки завантажувача в контролер плати.

Виконання

У цьому прикладі показано як за допомогою контролера Arduino змусити блимати світлодіод.

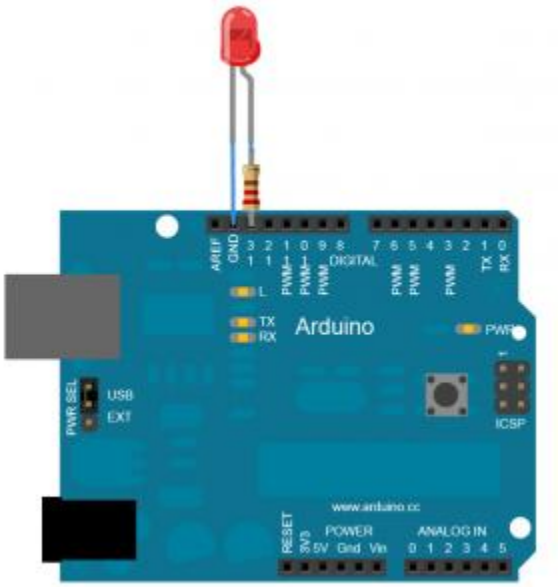
Необхідні компоненти

* контролер Arduino
* світлодіод
* резистор 220 Ом

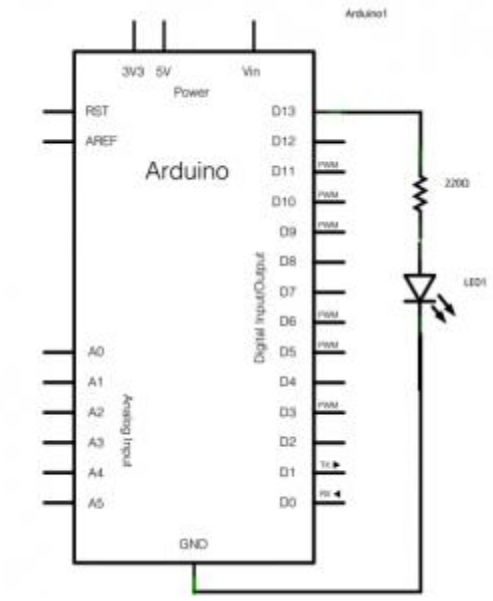
Підключення

Ми підключаємо резистор опором 220 Ом до виходу номер 13 (pin 13), до резистора в свою чергу підключаємо анод (зазвичай довга ніжка) світлодіода. Катод приєднуємо до землі (Gnd). Потім підключаємо контролер через USB кабель до комп'ютера і завантажуємо наведений нижче код на котроллер Arduino.

Більшість плат Arduino мають вбудований SMT (Surface-mount techology) світлодіод, підключений до виходу 13. Якщо ви запустите код на таких платах без підключення внешніего світлодіода, то ви повинні побачити миготіння вбудованого світлодіода на платі.



Схема



Код

У коді першим рядком задаємо режим виходу для вхід/виходу (pin) 13:

pinMode(13, OUTPUT);

В основному циклі (loop) програми запалюємо світлодіод:

digitalWrite(13, HIGH);

На виході 13 з'являється напруга 5 в.Світлодіод запалюється. Потім ми вимикаємо світлодіод:

digitalWrite(13, LOW);

Змінивши напругу на виході на 0 вольт, ми вимкнули світлодіод. Для того щоб людське око встигав помічати перемикання світлодіода введемо затримку за допомогою функції delay().

Код

```
/*
  Запалюємо світлодіод на одну секунду, потім вимикаємо його
  на одну секунду в циклі.
 */

void setup() {
  // Ініціалізуємо цифровий вхід / вихід в режимі виходу.
  // Вихід 13 на більшості плат Arduino підключений до
  //світлодіода на платі.
  pinMode(13, OUTPUT);
}

void loop() {
  digitalWrite (13, HIGH); // запалюємо світлодіод
  delay (1000); // чекаємо секунду
  digitalWrite (13, LOW); // вимикаємо світлодіод
  delay (1000); // чекаємо секунду
}
```

Послідовність виконання

1. Встановити середовище ArduinoIDE
2. Ознайомиться з інтерфейсом ArduinoIDE.
3. Зібрати схему.
4. Завантажити скетч в плату контролера.
5. Продемонструвати роботу викладачеві.
6. Змінити частоту миготіння світлодіода. (Повторити п. 4, 5)

Додатково. Робота з функцією analogWrite ()

Опис

Видає аналогову величину (ШІМ хвилю) на порт вхід/виходу. Функція може бути корисна для управління яскравістю підключеного світлодіода або швидкістю електродвигуна. Після виклику analogWrite () на виході буде генеруватися постійна прямокутна хвиля із заданою шириною імпульсу до

наступного виклику analogWrite (або виклику digitalWrite або digitalRead на тому ж порту вхід / виходу). Частота ШІМ сигналу приблизно 490 Hz.

На більшості плат Arduino (на базі мікроконтролера ATmega168 або ATmega328) ШІМ підтримують порти 3, 5, 6, 9, 10 і 11, на платі Arduino Mega порти з 2 по 13.

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

Синтаксис

analogWrite (pin, value)

Параметри

- pin: порт вхід / виходу на який подаємо ШІМ сигнал.
- value: період робочого циклу значення між 0 (повністю вимкнено) та 255 (сигнал поданий постійно).

Приклад: підключаємо до 11 входу світло діод та керуємо його яскравістю.

```
void setup() {
}

void loop() {
     for (int i=0; i<255; i++){
     analogWrite(11, i);
          delay (10);
         }
}
```

Дивимось на яскравість світлодіода.

Лабораторна робота «Побудова системи передавання інформації за допомогою UART»

Послідовний інтерфейс UART

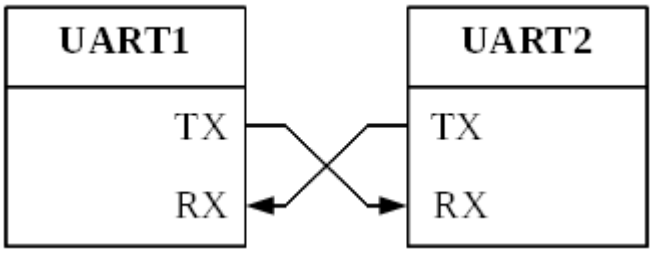
UART в перекладі це універсальний асинхронний приймач.

UART може використовуватися як для взаємодії компонентів всередині одного пристрою, так і для підключення пристроїв між собою. Для зовнішніх підключень сигнали з Рівнями логіки ТТЛ або КМОП підходять мало через низьку завадостійкості. Поширеним стандартом фізичного рівня для UART, який підходить для підключення зовнішніх пристроїв є RS-232. Цьому стандарту, зокрема, відповідає послідовний порт (COM-порт) комп'ютера. Так що, Мікроконтролер за допомогою схеми перетворення рівнів може обмінюватися інформацією з COM-портом комп'ютера, але про це трохи пізніше.

Варіанти підключення UART

В UART передача даних відбувається в послідовній формі, тобто по одному біту. Тому для передачі в одному напрямку потрібен один провідник; для повнодуплексного двонаправленого зв'язку потрібно два провідника.

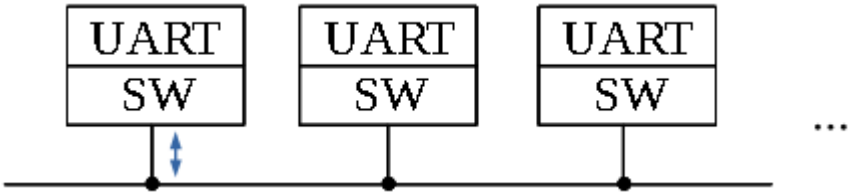
Вихід позначають TD або TX (transmitted data), вхід - RD або RX (received data). Для підключення двох пристроїв вихід одного підключають до входу іншого і вхід першого - до виходу другого.



Можливий варіант використання UART для напівдуплексного двонаправленого зв'язку по одному проводу. У цьому випадку висновки TX і RX кожного пристрою з'єднують разом. Весь час, поки пристрої не передає даних, воно тримає вихід у відключеному стані (переводить в Z-стан, стан з

високим опором; можна використовувати режим роботи з відкритим стоком - під час паузи в передачі на виході UART формується лог. 1, що рівносильно переходу в Z-стан). Пристрій може мати апаратну підтримку напівдуплексного обміну даними, тоді потрібно лише вибрати потрібний режим роботи. Якщо апаратної підтримки немає, напівдуплексний режим легко реалізується програмно. Для цього потрібно відключати передавач, коли пристрій не передає даних, щоб звільнити лінію для здійснення передачі іншими пристроями і відключати приймач під час роботи свого передавача, щоб не приймати власну передачу (або програмно відкидати дані, що передаються своїм передавачем).

До однопровідної лінії можна підключити кілька пристроїв, які будуть утворювати мережу для передачі даних. Арбітраж в цій мережі повинен бути реалізований програмно.



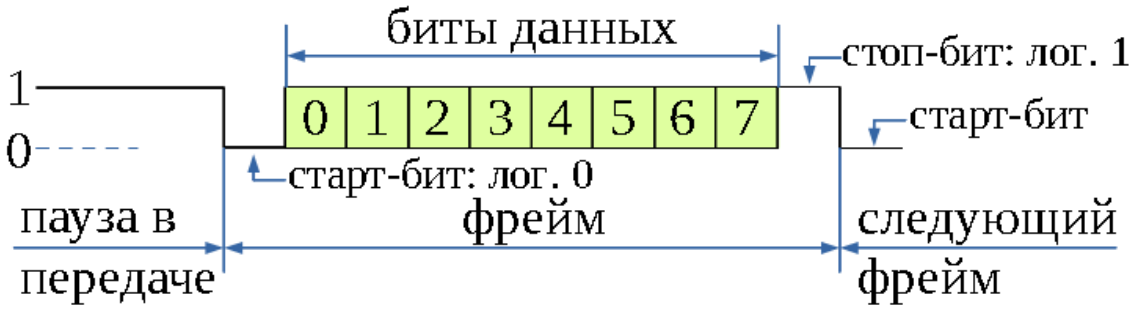
Як бачимо, об'єднувати пристрої за допомогою UART дуже просто. Для двонаправленого підключення потрібні тільки три провідника (з урахуванням загального проводу), а для односпрямованого або двонаправленого напівдуплексного - всього два.

Формат передачі даних UART

У відсутності передачі на виході UART присутній рівень лог. 1.

Дані передаються у вигляді посилок (фреймів), кожна з яких складається з стартового біта, бітів даних і одного або декількох стоп-бітів. Тривалість всіх бітів однакова, пов'язана зі швидкістю передачі співвідношенням T=1 / S. існує ряд стандартних швидкостей передачі: 110, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600 бод. Якщо всередині одного пристрою зв'язок можна здійснювати на довільній

швидкості, то для зв'язку із зовнішніми пристроями слід дотримуватися стандартних величин.



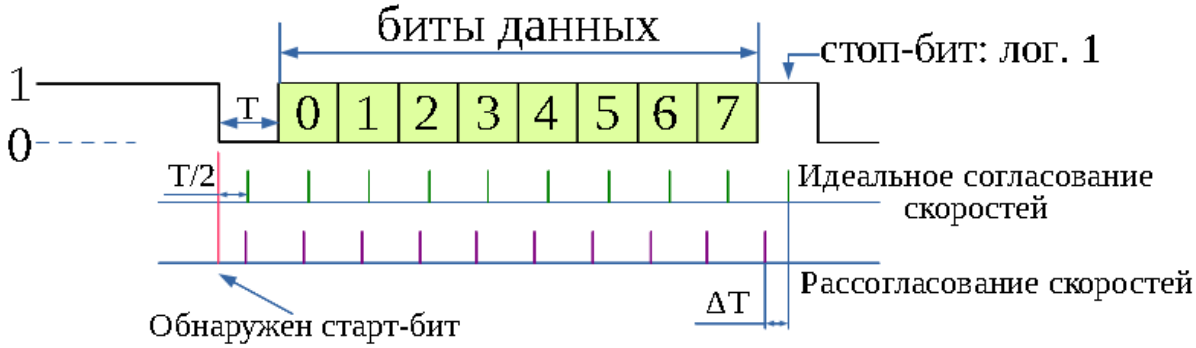
Посилка починається зі стартового біта, він завжди має значення лог. 0. Після стартового біта передаються біти даних. Кількість бітів даних може становити 5-9 в залежності від налаштувань UART. Зазвичай передається 8 біт даних або 9 біт (8 біт власне даних і один біт парності). Завершується посилка стоп-бітами, їх значення-завжди лог. 1, Кількість зазвичай становить 1, 1.5 або 2. Під кількістю стоп-бітів розуміється тривалість відповідного їм одиничного імпульсу по відношенню до тривалості бітів даних і старт-біта. Цим пояснюється можливість висловлювати кількість бітів дробовим числом. Відразу ж після стоп-бітів може починатися передача наступної посилки або може бути пауза довільної тривалості, під час якої на виході також формується рівень лог. 1.

Так як під час передачі стоп-біта і поки Лина Вільна, на виході присутній одиничне значення, а старт-біт має значення лог. 0, старт-біт дозволяє виявити момент початку передачі даних, розділити дві послідовні посилки і здійснити синхронізацію передавача і приймача.

Якщо передавач і приймач працюють на одній швидкості, налаштовані на роботу з однаковою кількістю бітів даних, стоп бітів, однаково налаштовані щодо біта парності, то для обміну даними не потрібно самостійно.

Виявивши початок старт-біта, приймач чекає протягом половини тривалості передачі біта, після чого починає зчитувати сигнал на вході з частотою, рівній швидкості передачі даних. В ідеальному випадку момент кожного зчитування припадає на середину прийнятого біта. В реальності

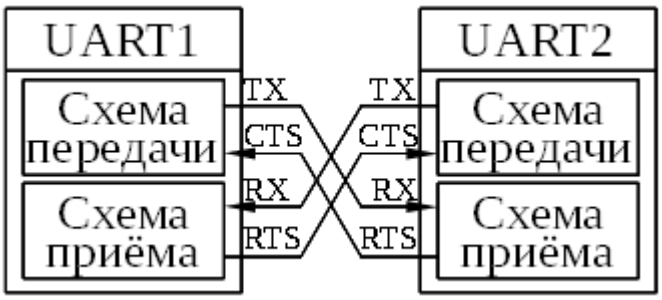
генератори тактових імпульсів передавача і приймача мають неузгодженість по частоті, в результаті кожне нове зчитування все більше зміщується щодо середини чергового біта. Важливо, щоб за час передачі однієї посилки, зміщення не перевищило половини тривалості біта, а з урахуванням перехідних процесів - зміщення краще не перевищувати четвертої частини тривалості біта. Інакше замість зчитування біта відбудеться зчитування сусіднього біта (або зчитування лінії під час перехідного процесу) і посилка буде прийнята невірно.



Управління потоком даних

Для управління потоком даних UART використовується програмний або апаратний метод. У разі програмного методу, інформація про готовність пристрою приймати дані або про необхідність зупинити передачу передається по тих же каналах, що і дані. Приймаюча сторона програмно розділяє дані і керуючі сигнали відповідно до прийнятого протоколу.

Інтерфейс UART передбачає можливість використання додаткових сигналів (CTS, RTS) для апаратного управління потоком даних. Апаратне управління може використовуватися деякими повільними пристроями або пристроями з простою схемною реалізацією. Однак воно зажадає двох додаткових ліній для підключення пристрою.



Якщо в UART включений контроль стану CTS, передавач перед відправкою чергового фрейма перевіряє вхід CTS. Якщо на CTS низький рівень, передача відбувається, інакше-ні. Якщо сигнал CTS буде встановлений під час передачі посилки (фрейма), поточна передача все одно буде завершена перед зупинкою.

Приймач, в свою чергу, встановлює на виході RTS значення лог. 0, якщо він готовий приймати Дані і встановлює лог. 1, вимагаючи від передавача зупинити передачу.

Комунікація по послідовному порту (UART), використовується для програмування та відлагодження контролера Arduino за допомогою порту USB. Існують різні пристрої, які використовують UART як вид основного зв'язку, та іноді нам треба об'єднувати два і більше контролера між собою для обміну інформацією.

Проте, у більшості Arduino є тільки один послідовний порт, який використовується при зв'язку по USB. Але як же зв'язати такий контроллер з іншим? Кінцеве використання Arduino типу Mega або подібного вирішує цю задачу, адже у нього до чотирьох послідовних портів, але якщо потрібний зв'язок з простими платами з лінійки Ардуино, тут треба шукати інший вихід. Існує особлива програмна бібліотека, яка імітує UART порт на інших цифрових контактах. У неї є декілька недоліків, але загалом вона працює.

Для демонстрації подібної комунікації знадобляться:

2 Arduino контролера

Дроти для з’єднання

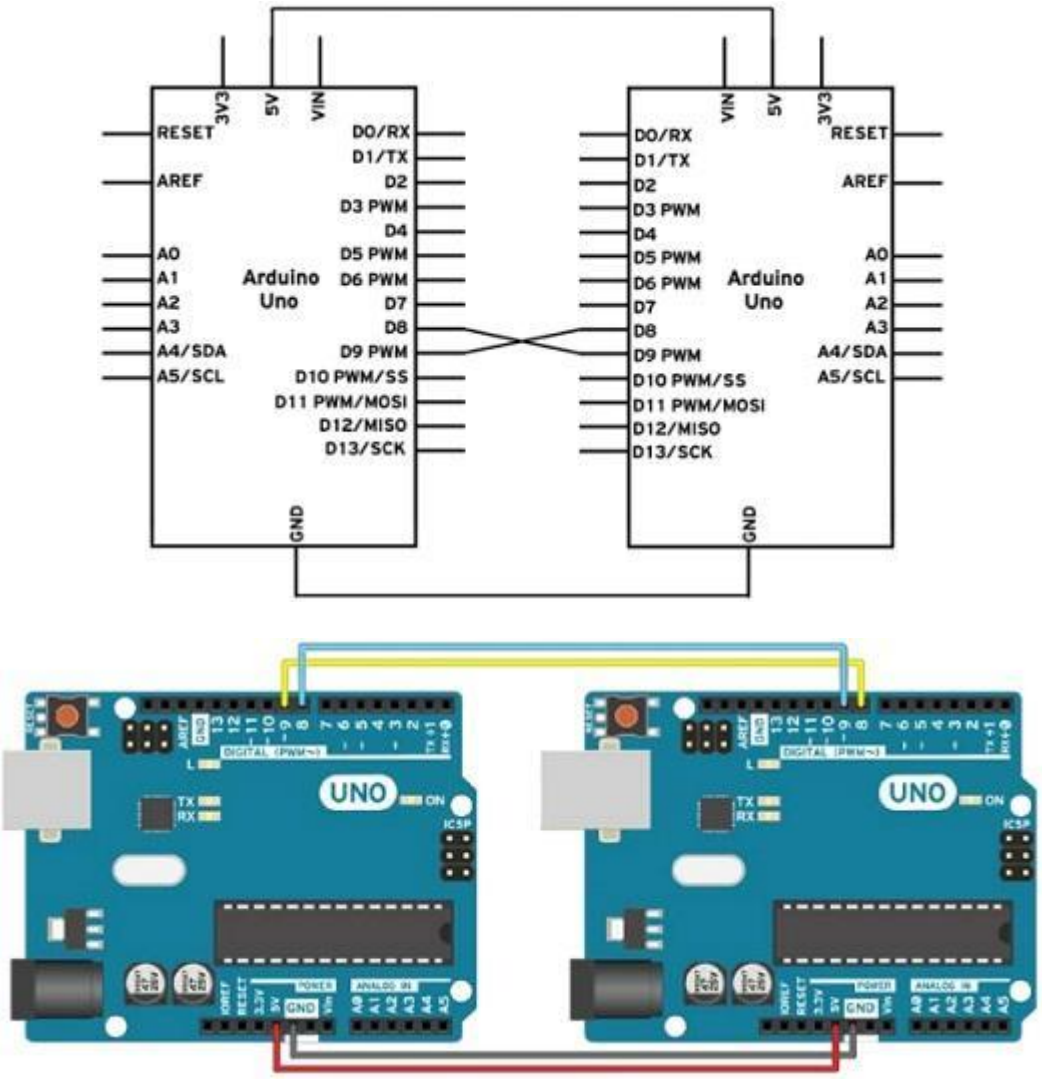
Виконаєте наступні кроки, для підключення двох Arduino UNO, за допомогою програмного послідовного порту:

1. Наприклад, скористаємося контактами 8 і 9 для RX і TX на обох Arduino, з'єднаєте контакт 8 на одному Arduino з контактом 9 на іншому, і контакт 9 на першому Arduino підключити з контактом 8 на другому.

2. З'єднайте загальний дріт GND обох Arduino разом.

3. Підключіть один Arduino до USB комп'ютера, і з'єднаєте контакт 5V цього контроллера з таким же контактом іншого або подайте на другій окреме живлення.

Ось реалізація з використанням контактів 8 і 9 для RX та TX:



Наступний код розділений на дві частини. Arduino-майстер отримуватиме команди від комп'ютера і передаватиме їх по програмному послідовному порту. Ось перша частина коду :

```
//Додаємо бібліотеку Software Serial
#include <SoftwareSerial.h>
// Вказуємо дискретні канали контролеру для зв’язку.
SoftwareSerial softSerial(8, 9); // RX, TX
void setup(){
Serial.begin(9600); // Встановлюємо швидкість передавання,
//для контролю роботи за допомогою Монітору послідовного
// порта
softSerial.begin(9600);    //    Ініціалізація    програмного
//послідовного порта
}
void loop(){
// Перевірка отримання команд з комп’ютера
if (Serial.available()){
//Відправка отриманою команди на програмний UART
softSerial.write(Serial.read());
}
}
```

А ось і код підлеглого (слейва), який інтерпретує символи, відправлені від майстра. Якщо пришол символ "a", він включить вбудований світлодіод. Якщо отримано символ "x", то світлодіод буде погашений:

```
// Підключення бібліотеки Software Serial
#include <SoftwareSerial.h>
// Призначення задіяних дискретних каналів
SoftwareSerial softSerial(8, 9); // RX, TX
// Дискретний канал, на якому висить вбудований світлодіод
int LED = 13;
void setup(){
softSerial.begin (9600); // Ініціалізація програмного
                          //послідовного порту
pinMode (LED, OUTPUT); // визначення світлодіодного виводу
                        // як вихід
}
void loop(){
// Перевіряємо, чи є що-небудь в буфері програмного
// послідовного порту
if (softSerial.available()){
// Читаємо один символ з буфера програмного послідовного
// порту і зберігаємо його змінну com
int com = softSerial.read();
// Діємо відповідно до отриманого символу
if (com = = 'x'){
```

```
// Вимкнення світлодіода
digitalWrite(LED, LOW);
}
else if (com = = 'a'){
// Включення світлодіода
digitalWrite(LED, HIGH);
}
}
}
```

Виконання лабораторної роботи

1. Зберіть схему.
2. Завантажте скетчі в контролер-Майстер і контролер-слейв.
3. Покажіть робочу систему викладачеві.
4. Модифікуйте програму так, щоб управляти можна було будь-яким мікроконтролером з будь-якого. (Повторіть п. 2, 3)

Лабораторна робота «Побудова системи передавання інформації за допомогою RS485»

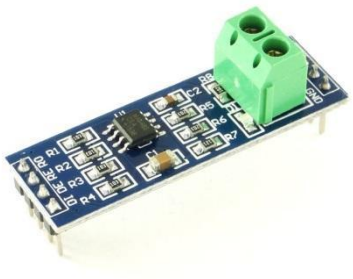
Загальні відомості

RS485-це послідовний інтерфейс, предком якого є RS232. Останній здобув популярність через COM-портів старих комп'ютерів, які якраз працювали по інтерфейсу RS232. Максимальна довжина лінії при з'єднанні по RS485 становить 1200 метрів! А якщо на лінії будуть спеціальні підсилювачі, то ще більше. Звичайно, швидкість передачі по такому довгому проводу буде всього близько 60 кб / с, але для передачі показань датчиків більше і не потрібно. В якості кабелю для RS485 використовується кручена пара. Це два дроти, сплетені один з одним. Такий кабель ще використовується в Ethernet лініях, так що його легко можна дістати. Щоб передавати дані на дистанції більше 500 метрів, буде потрібно екранована

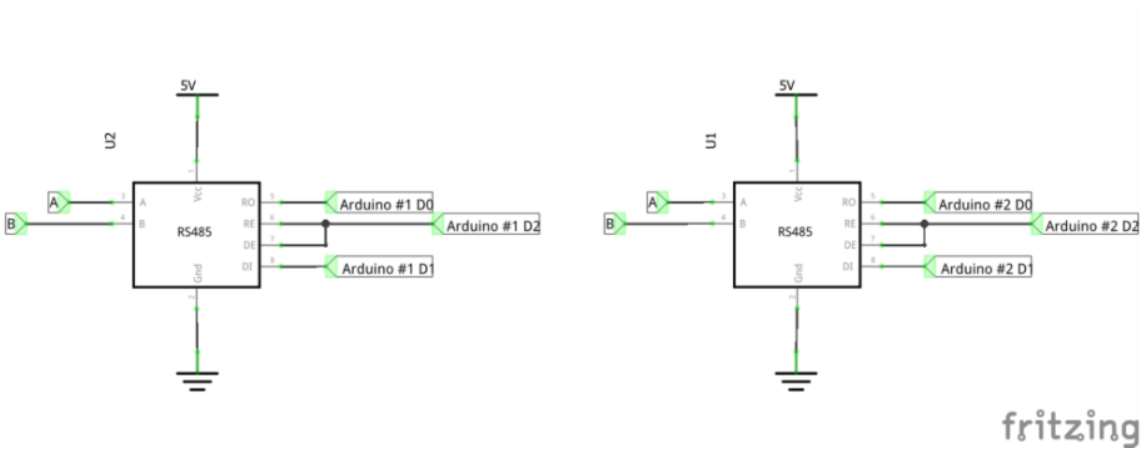
кручена пара. До одного кабелю може бути підключено 32 пристрої. Але в один момент часу тільки один пристрій може передавати дані.

Підключення двох Ардуіно за допомогою RS485

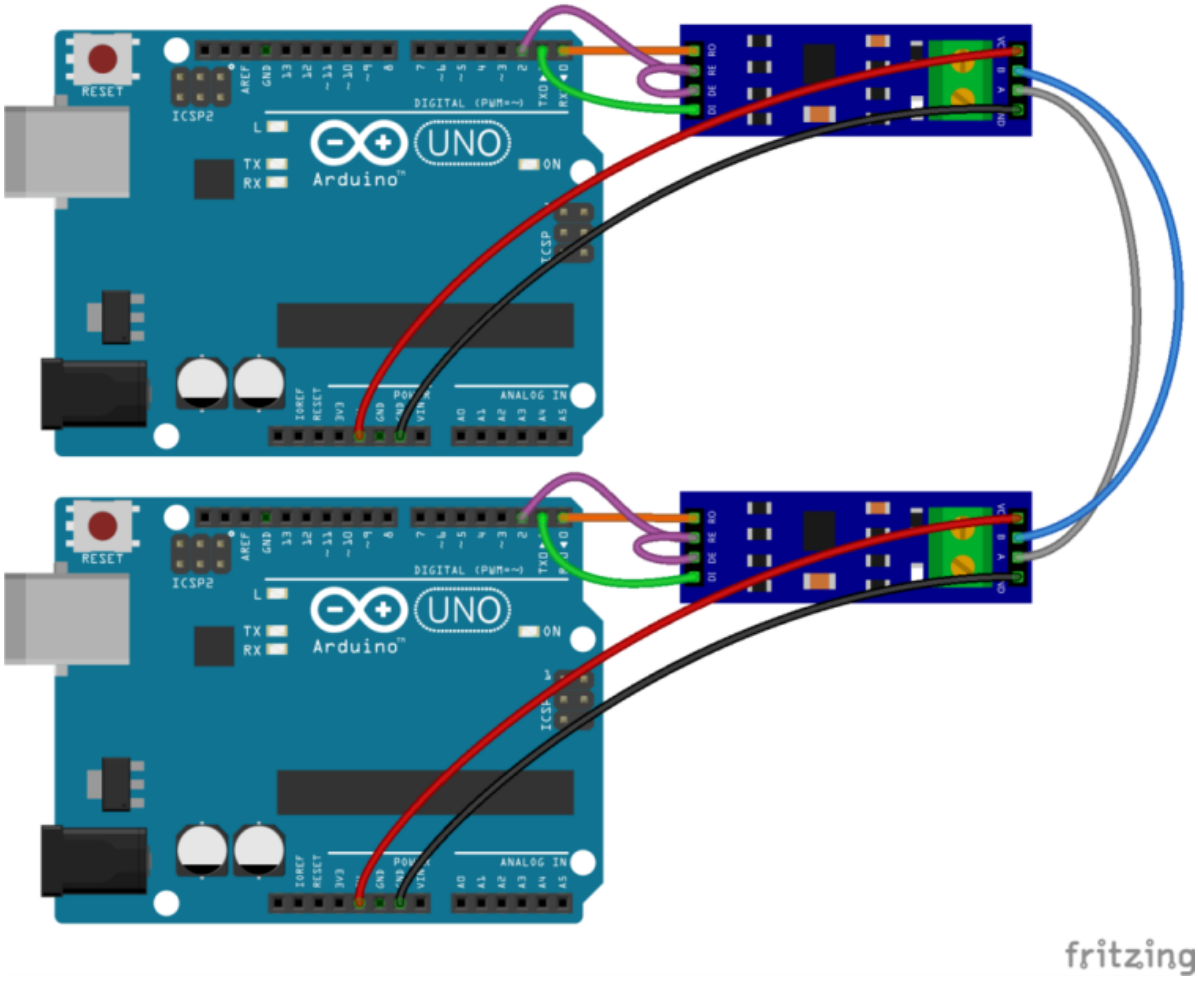
Для того, щоб підключити дві плати Ардуіно по інтерфейсу RS485 нам буде потрібно спеціальний модуль. Зазвичай такі модулі використовують поширену мікросхему MAX485.



З'єднаємо дві Ардуіно за наступною схемою:



Зовнішній вигляд макета



Примітка. На малюнку дроти A і B — прямі, і цього буде достатньо, якщо відстань буде невеликим. У разі підключення по кабелю довжиною в кілька метрів потрібно обов'язково використовувати виту пару!

Програма приймача і передавача для RS485.

Як ми вже відзначали, якщо до лінії підключені кілька пристроїв, то в один момент часу тільки одне з них може передавати дані. Виходить, нам потрібно якимось чином повідомити всім сусідам, що ми готові до передачі, а вони при цьому повинні мовчати і слухати. Робиться це за допомогою контактів DE і RE. Якщо ми подаємо високий рівень на ці контакти, то пристрій переходить в режим передавача. Низький рівень-приймача. Напишемо дві програми. Одна буде передавати в ефір текст "ping", кожні 500 мс. Інша буде слухати ефір і при отриманні текст "ping" - блимати світлодіодом №13.

Програма передавача

```
# define SerialTxControl 2 // контакт №2 буде перемикати
                           // режим приймач/передавач
# define RS485Transmit HIGH
# define RS485Receive LOW

void setup(void) {
    Serial.begin (9600); // налаштовуємо послідовний порт на
                         // швидкість 9600бод
    pinMode(SerialTxControl, OUTPUT);
    digitalWrite (SerialTxControl, RS485Transmit);
     // переводимо пристрій в режим передавача
}

void loop(void) {
    Serial.print("ping"); // відправляємо текст
    delay (500);
}
```

І програма приймача

```
# define SerialTxControl 2
# define RS485Transmit HIGH
# define RS485Receive LOW

char buffer[100];
byte state = 0;

void setup(void) {
    Serial.begin (9600);
    pinMode(13, OUTPUT);
    pinMode(SerialTxControl, OUTPUT);
    digitalWrite (SerialTxControl, RS485Receive);
    // переводимо пристрій в режим приймача
}

void loop(void) {
    int i=0;
    if( Serial.available ()) {
//якщо в порт прийшли якісь дані
        delay (5);
// трохи чекаємо, щоб вся пачка даних була
//прийнята портом
        while (Serial.available() ){
            buffer[i++] = Serial.read ();
// зчитуємо дані і записуємо їх в буфер
        }
    }
    if (i>0) {//якщо в буфері є щось
        buffer[i++]='\0'; // перетворюємо вміст буфера
```

```
                    // в рядок, додаючи нульовий символ
            if(strcmp (buffer, "ping")) {
//якщо прийнятий рядок дорівнює тексту ping
                digitalWrite (13, state);
// блимаємо світлодіодом
                state = !state;
            }
        }
}
```

Завантажуємо програми на обидві плати і спостерігаємо за тим, що відбувається. Якщо все зібрано вірно, то після подачі живлення, на другій платі почне блимати світлодіод, підключений до висновку №13. У цьому прикладі, приймач і передавач не змінюють своєї ролі на всьому протязі роботи. Удосконалимо програми і зробимо так, щоб приймач і передавач змінювалися ролями в процесі роботи.

Програма для передачі даних по RS485 в обидві сторони.

Перша Ардуіно вже не буде постійно в режимі передавача, вона буде міняти режим на приймач в момент відправки повідомлення. Друга плата буде вести себе аналогічним чином.

Програма для першої плати Ардуіно

```
# define SerialTxControl 2
# define RS485Transmit HIGH
# define RS485Receive LOW

char buffer[100];
byte state = 0;

unsigned long ping_next, t;
unsigned int ping_to = 500;

void setup(void) {
    Serial.begin(115200);
    pinMode(13, OUTPUT);
    pinMode(SerialTxControl, OUTPUT);
    digitalWrite (SerialTxControl, RS485Receive);
}
```

```
void loop(void) {
    int i=0;
    if( Serial.available() ){
        delay (5);
        while (Serial.available() ){
            buffer[i++] = Serial.read();
        }
        if (i>0){
            buffer[i++]=' \ 0';
            if (!strcmp(buffer, " pong") ){
                digitalWrite(13, state);
                state = !state;
            }
        }
    }
    // кожні 500 мс відправляємо текст " ping"
    t = millis();
    if (t>ping_next ){
        ping_next = t + ping_to;
        digitalWrite (SerialTxControl, RS485Transmit);
        Serial.print ("ping");
        delay (5);
        digitalWrite (SerialTxControl, RS485Receive);
    }
}
```

Перша плата надсилає текст «ping» кожні 500 мс. Робиться це не за допомогою delay, а за допомогою таймаутів

Програма для другої плати Ардуіно

```
# define SerialTxControl 2
# define RS485Transmit HIGH
# define RS485Receive LOW

char buffer[100];
byte state = 0;

void setup(void) {
    Serial.begin(115200);
    pinMode(13, OUTPUT);
    pinMode(SerialTxControl, OUTPUT);
    digitalWrite (SerialTxControl, RS485Receive);
}

void loop(void) {
    int i=0;
    if( Serial.available() ){
        delay (5);
        while (Serial.available() ){
            buffer[i++] = Serial.read();
```

```
            }
            if (i>0){
                buffer[i++]=' \ 0';
                if (!strcmp(buffer, " ping") ){
                    digitalWrite(13, state);
                    state = !state;
                    digitalWrite (SerialTxControl,
                                            RS485Transmit);
                    Serial.print ("pong");
                    delay(10);
                    digitalWrite (SerialTxControl,
                                                RS485Receive);
                }
            }
        }
    }
```

Завантажуємо програми. В результаті, друга плата Ардуіно буде кожні 500 мілісекунд запалювати вбудований світлодіод на виводі №13. Відразу слідом за цим світлодіод буде запалюватися і на першій Ардуіно. Ще через 500 мс, світлодіоди згаснуть в тій же послідовності. І так далі.

Виконання лабораторної роботи.

1. Зберіть схему.
2. Завантажте скетчі в контролер-майстер і контролер-слейв.
3. Покажіть робочу систему викладачеві.
4. Модифікуйте схему та програму так, щоб майстер був спроможен керувати двома мікроконтролерами. (Повторіть п. 2, 3)

Лабораторна робота «Робота з RFID reader»

Загальні положення.

Радіочастотна ідентифікація (RFID) - це технологія автоматичної безконтактної ідентифікації об'єктів за допомогою радіочастотного каналу зв'язку. Базова система RFID складається з:

* радіочастотної мітки;

* зчитування інформації (ридера);

* комп'ютера для обробки інформації.

Ідентифікація об'єктів проводиться за унікальним цифровим кодом, який зчитується з пам'яті Електронної мітки, що прикріплюється до об'єкта ідентифікації. Зчитувач містить в своєму складі передавач і антену, за допомогою яких випромінюється електромагнітне поле певної частоти. Потрапили в зону дії зчитувального поля радіочастотні мітки "відповідають" власним сигналом, що містить інформацію (ідентифікаційний номер товару, призначені для Користувача дані і т.д.). Сигнал вловлюється антеною зчитувача, інформація розшифровується і передається в комп'ютер для обробки. Переважна більшість сучасних систем контролю доступу (СКД) використовує в якості засобів доступу ідентифікатори, що працюють на частоті 125 кГц. Це проксіміті-карти доступу (тільки читання), найпоширенішими є карти EM-Marin, а також HID, Indala. Карти цього стандарту є зручним засобом відкривання дверей і турнікетів. Але не більше. Ці карти не володіють ніякої захищеністю, легко копіюються і підробляються і, відповідно, нічого не дають для захисту об'єкта від несанкціонованого проникнення.

Справжній захист від копіювання і підробки забезпечують такі ідентифікатори, в чіпах яких реалізована криптографічний захист. Це безконтактні смарт-карти, що працюють на частоті 13,56 МГц, найбільш поширеними з них є карти Mifare. У картах цих стандартів криптозахист організована на високому рівні, і підробка таких карт практично неможлива.

Необхідні компоненти:

* контролер Arduino UNO R3;

* плата для прототипування;

* RFID-зчитувач RC522;

• брелок;

• карта;

* дроти тато-тато.

Модуль RC522-RFID-модуль 13,56 МГц з SPI-інтерфейсом. У комплекті до модуля йдуть 2 RFID-мітки – у вигляді карти і брелока.

Основні характеристики:

* заснований на мікросхемі MFRC522;

* напруга живлення: 3,3 В;

* споживаний струм: 13-26 мА;

* робоча частота: 13,56 МГц;

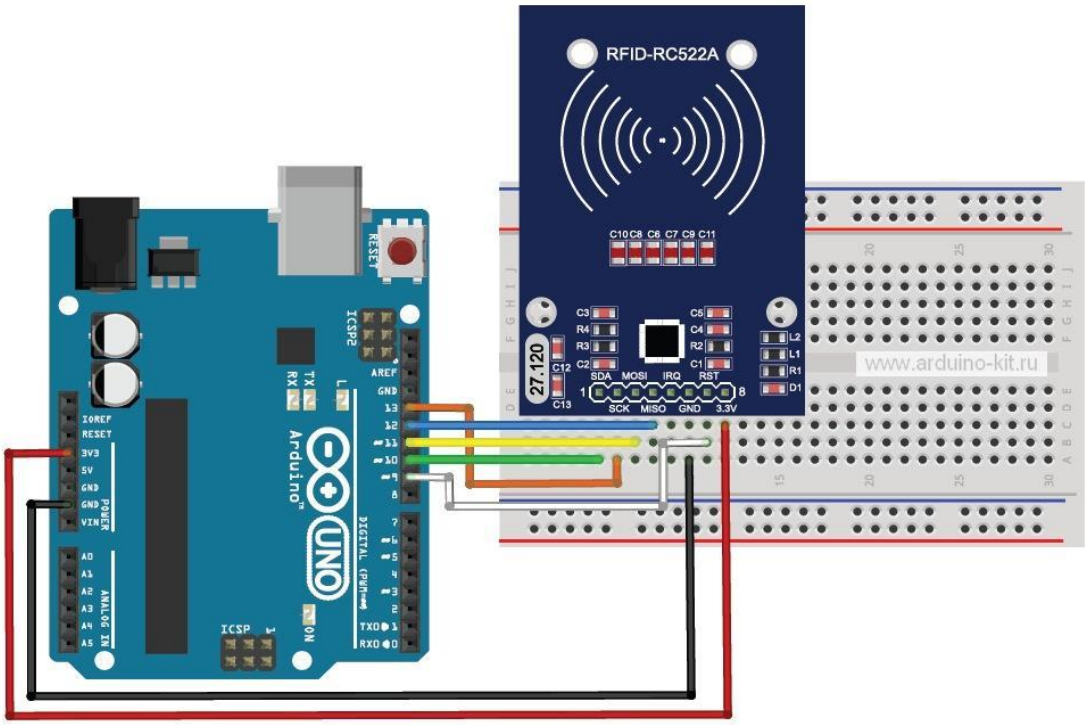
* дальність зчитування: 0~60 мм;

* інтерфейс: SPI, максимальна швидкість передачі 10 Мбіт / с;

* розмір: 40 х 60 мм;

* читання та запис RFID-міток.

Схема підключення модуля до плати Arduino



Arduino arduino uid (унікальний ідентифікаційний номер) RFID-мітки (карти або брелока). При написанні скетча будемо використовувати бібліотеку MFRC522 (https://github.com/miguelbalboa/rfid).

Програма.

```
// Підключення библиотек
#include <SPI.h>
#include <MFRC522.h>
// константи підключення контактів SS и RST
#define RST_PIN 9
#define SS_PIN 10
MFRC522 mfrc522(SS_PIN, RST_PIN); // Створення MFRC522
void setup()
{
Serial.begin(9600); // Ініціалізація послідовного порта
SPI.begin(); // Ініціалізація SPI
mfrc522.PCD_Init(); // Ініціалізація MFRC522
}
void loop()
{
if ( ! mfrc522.PICC_IsNewCardPresent())
return;
// читання карти
if ( ! mfrc522.PICC_ReadCardSerial())
return;
// результат читання UID та клас мітки
Serial.print(F("Card UID:"));
dump_byte_array(mfrc522.uid.uidByte, mfrc522.uid.size);
Serial.println();
Serial.print(F("PICC type: "));
byte piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
Serial.println(mfrc522.PICC_GetTypeName(piccType));
delay(2000);
}
// Виведення результату читання даних у виглялі HEX
void dump_byte_array(byte *buffer, byte bufferSize)
{
for (byte i = 0; i < bufferSize; i++)
{
Serial.print(buffer[i] < 0x10 ? " 0" : " ");
Serial.print(buffer[i], HEX);
}
}
```

Порядок підключення:

1. Будуємо наведену схему

2. Завантежуємо програму. Відкриваємо монітор послідовного порту.

3. Підносимо мітку (карту або брелок) до зчитувача і бачимо висновок в послідовний порт даних мітки теплоносія і тип.

Мітки Mirafe дозволяють записувати на них інформацію. У наступному скетчі ми організуємо на карті лічильник, який буде інкрементуватися при піднесенні карти до зчитувача. У послідовний порт будемо виводити показання лічильника.

Програма

```
// Підключення библиотек
#include <SPI.h>
#include <MFRC522.h>
// константи підключення контактів SS и RST
#define RST_PIN 9
#define SS_PIN 10
MFRC522 mfrc522(SS_PIN, RST_PIN); // Створення MFRC522
MFRC522:MIFARE_Key key;
byte sector = 1;
byte blockAddr = 4;
byte dataBlock[] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
byte trailerBlock = 7;
byte status;
byte buffer[18];
byte size = sizeof(buffer);
void setup()
{
Serial.begin(9600); // Ініціалізація послідовного порта
SPI.begin(); // Ініціалізація SPI
mfrc522.PCD_Init(); // Ініціалізація MFRC522
// Значення ключа (A или B) – FFFFFFFFFFFFh значення
// виробника
for (byte i = 0; i < 6; i++)
key.keyByte[i] = 0xFF;
}
void loop()
{
if ( ! mfrc522.PICC_IsNewCardPresent())
return;
// читання картки
if ( ! mfrc522.PICC_ReadCardSerial())
return;
// результат читання UID та клас мітки
Serial.print(F("Card UID:"));
dump_byte_array(mfrc522.uid.uidByte, mfrc522.uid.size);
Serial.println();
Serial.print(F("PICC type: "));
byte piccType = mfrc522.PICC_GetType(mfrc522.uid.sak);
Serial.println(mfrc522.PICC_GetTypeName(piccType));
// Читання даних з блоку
Serial.print(F("Reading data from block "));
Serial.print(blockAddr);
Serial.println(F(" ..."));
```

```
Serial.print(F("Data for count ")); Serial.print(blockAddr);
Serial.println(F(":"));
dump_byte_array(buffer, 2); Serial.println();
Serial.println();
for (byte i = 0; i < 16; i++) // запис до buffer[]
dataBlock[i]=buffer[i];
// отримання лічильника (0 и 1)
int count1=(buffer[0]<<8)+buffer[1];
Serial.print("count1=");
Serial.println(count1);
count1=count1+1; // інкремент лічильника
dataBlock[0]=highByte(count1);
dataBlock[1]=lowByte(count1);
// Аутентифікація key B
Serial.println(F("Authenticating again using key B..."));
// Запис даних до блоку
Serial.print(F("Writing data into block "));
Serial.print(blockAddr);
Serial.println(F(" ..."));
dump_byte_array(dataBlock, 2); Serial.println();
}
// Виведення результату читання даних у виглялі HEX
void dump_byte_array(byte *buffer, byte bufferSize)
{
for (byte i = 0; i < bufferSize; i++)
{
Serial.print(buffer[i] < 0x10 ? " 0" : " ");
Serial.print(buffer[i], HEX);
}
}
```

Виконання лабораторної роботи № 3.

1. Зберіть схему.
2. Завантажте скетчі в контролер.
3. Покажіть робочу систему викладачеві.
4. Модифікуйте програму так, щоб за допомогою картки управляти вбудованим світлодіодом мікроконтролера. (Повторіть п. 2, 3)

Лабораторна робота «Побудова системи передавання інформації за допомогою SPI»

Загальні положення

Послідовний периферійний інтерфейс (SPI) - це Синхронний послідовний протокол передачі даних, який використовується мікроконтролерами для швидкого обміну даними з одним або декількома периферійними пристроями на коротких відстанях. Він також може використовуватися для зв'язку між двома мікроконтролерами. При SPI-з'єднанні завжди є один головний пристрій (зазвичай мікроконтролер), який управляє периферійними пристроями. Це повнодуплексне з'єднання, яке означає, що дані передаються і приймаються одночасно. Максимальна швидкість передачі даних вище, ніж в системі зв'язку I2C.

Як правило, є три лінії, загальні для всіх пристроїв:

MISO (Master In Slave Out) - підпорядкована лінія для відправки даних майстру,

MOSI (Master Out Slave In) - головна лінія для відправки даних на периферію,

SCK (Serial Clock) - тактові імпульси, що синхронізують передачу даних, що генеруються провідним пристроєм.

і одна лінія специфічна для кожного приладу:

SS (Slave Select)-pin-код на кожному пристрої, який ведучий може використовувати для включення і відключення певних пристроїв.

Коли на слейві рівень вибору пристрою низький, він зв'язується з ведучим пристроєм. Коли він високий, він ігнорує майстра. Це дозволяє мати кілька пристроїв SPI, які спільно використовують одні і ті ж лінії MISO, MOSI і CLK.

Щоб написати код для нового пристрою SPI, необхідно відзначити кілька речей:

- Яка максимальна швидкість SPI, яку може використовувати Ваш пристрій? Це контролюється першим параметром в SPISettings. Якщо ви використовуєте мікросхему з частотою 15 МГц, використовуйте 15000000. Arduino автоматично використовує найкращу швидкість, яка дорівнює або менше числа, використовуваного за допомогою SPISettings.

- Дані зсуваються в найбільш значущий біт (MSB) або найменш значущий біт (LSB) в першу чергу? Це контролюється другим параметром SPISettings, або MSBFIRST, або LSBFIRST. Більшість мікросхем SPI використовують MSB першого порядку даних.

- Чи є час очікування даних високим або низьким? Чи є зразки на висхідному або низхідному фронті тактових імпульсів? Ці режими керуються третім параметром в SPISettings.

Стандарт SPI вільний, і кожен пристрій реалізує його трохи по-різному. Це означає, що при написанні коду ви повинні звертати особливу увагу на дані пристрою.

Примітка про виведення Slave Select (SS) на платах на основі AVR

Всі плати на базі AVR мають пін SS, який корисний, коли вони діють в якості веденого пристрою, керованого зовнішнім ведучим пристроєм. Оскільки ця бібліотека підтримує тільки головний режим, цей пін повинен бути встановлений завжди в якості вихідного, інакше інтерфейс SPI може бути автоматично переведений в ведений режим апаратним забезпеченням, що призведе до непрацездатності бібліотеки.

Однак можна використовувати будь-який пін як Slave Select (SS) для пристроїв. Наприклад, Arduino Ethernet shield використовує pin 4 для управління підключенням SPI до вбудованої SD-карти і pin 10 для управління підключенням до контролера Ethernet.

Комунікації

Наступна таблиця показує, на яких пінах лінії SPI розташовані на різних платах Arduino:

Arduino / Genuino Board	**MOSI**	**MISO**	**SCK**	**SS (slave)**	**SS (master)**	**Level**
Uno or Duemilanove	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	–	5V
Mega1280 or Mega2560	51 or ICSP-4	50 or ICSP-1	52 or ICSP-3	53	–	5V
Leonardo	ICSP-4	ICSP-1	ICSP-3	–	–	5V

Due	ICSP-4	ICSP-1	ICSP-3	–	4, 10, 52	3,3V
Zero	ICSP-4	ICSP-1	ICSP-3	–	–	3,3V
101	11 or ICSP-4	12 or ICSP-1	13 or ICSP-3	10	10	3,3V
MKR1000	8	10	9	–	–	3,3V

Зверніть увагу, що MISO, MOSI і SCK доступні в узгодженому фізичному розташуванні на заголовку ICSP; це корисно, наприклад, при розробці екрану, який працює на кожній платі.

Arduino SPI

З'єднання між двома пристроями SPI завжди відбувається між провідним пристроєм і веденим пристроєм. Майстер є активною частиною в цій системі і повинен забезпечити тактовий сигнал, на якому заснована послідовна передача даних. Ведений не здатний генерувати тактовий сигнал і, отже, не може активуватися самостійно. Ведений просто відправляє і отримує дані, якщо ведучий генерує необхідний тактовий сигнал. Майстер, однак, генерує тактовий сигнал тільки при відправці даних. Це означає, що ведучий повинен відправити дані веденому, щоб прочитати дані від веденого.

Використовуються наступні функції. Ви повинні включити SPI.x.

- SPI.begin () - ініціалізує шину SPI, встановлюючи SCK, MOSI і SS на виходи, витягаючи SCK і MOSI low, а SS high.

- SPI.setClockDivider (роздільник) - установка роздільника годин SPI щодо системного годинника. На платах на основі AVR доступні наступні роздільники: 2, 4, 8, 16, 32, 64 або 128. Значення за замовчуванням-SPI_CLOCK_DIV4, яке встановлює SPI-годинник на одну чверть частоти системних годин (5 МГц для плат з частотою 20 МГц).

Роздільник може бути (SPI_CLOCK_DIV2, SPI_CLOCK_DIV4, SPI_CLOCK_DIV8, SPI_CLOCK_DIV16, SPI_CLOCK_DIV32, SPI_CLOCK_DIV64, SPI_CLOCK_DIV128).

- SPI. transfer (val) − передача SPI заснована на одночасній відправці і отриманні: отримані дані повертаються в Val.

- SPI.beginTransaction(SPISettings(speedMaximum, dataOrder, dataMode)) - speedMaximum-це годинник, dataOrder(MSBFIRST або LSBFIRST), dataMode (SPI_MODE0, SPI_MODE1, SPI_MODE2 або SPI_MODE3).

Взагалі кажучи, існує чотири способи передачі. Ці режими керують тим, чи зсуваються дані на висхідному або низхідному фронті сигналу синхронізації даних (званому фазою синхронізації), а також тим, чи знаходяться годинник в режимі очікування при високій або низькій полярності (званої полярністю синхронізації). Чотири режими поєднують полярність і фазу відповідно до цієї таблиці:

Mode	**Clock Polarity (CPOL)**	**Clock Phase (CPHA)**	**Output Edge**	**Data Capture**
SPI_MODE0	0	0	Falling	Rising
SPI_MODE1	0	1	Rising	Falling
SPI_MODE2	1	0	Rising	Falling
SPI_MODE3	1	1	Falling	Rising

Схема макету

З'єднаємо дві плати Arduino UNO разом; одну як майстер, а іншу як слейв.

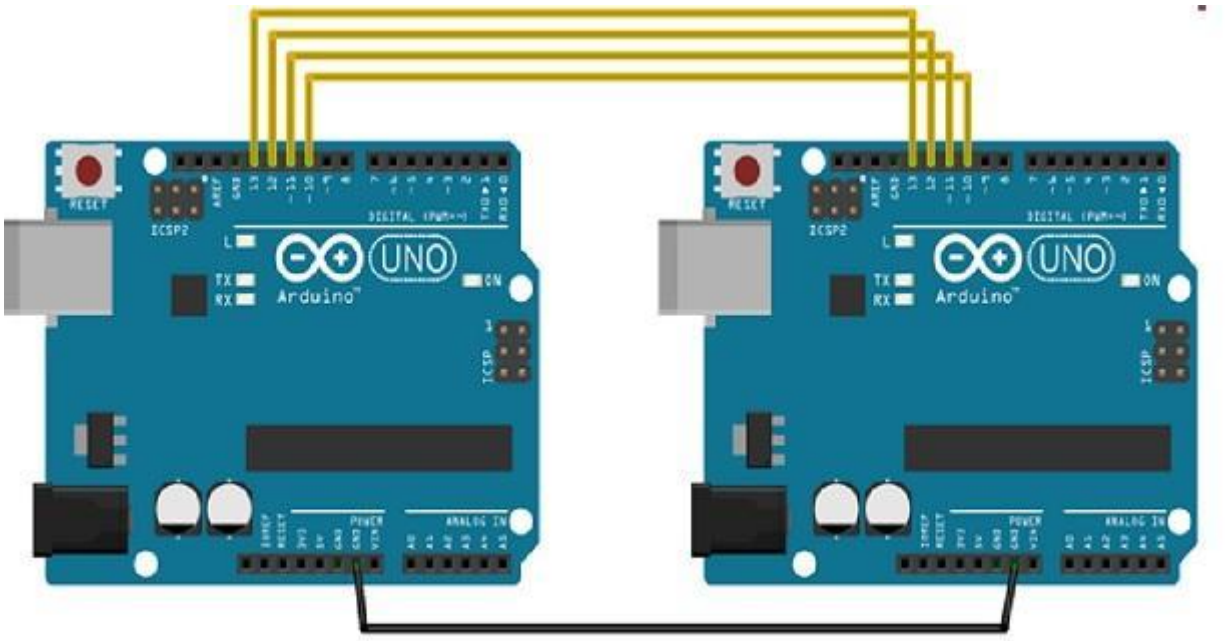
(SS) : pin 10

(MOSI) : pin 11

(MISO) : pin 12

(SCK) : pin 13

Земля тут спільна. Нижче наводиться схема зв'язку між обома контролерами



Приклад Arduino SPI як майстер

Майстер-пристрій посилає Hello, world! для веденого пристрою.

```
#include <SPI.h>

void setup (void) {
Serial.begin(115200); //швидкість передавання 115200 бод
digitalWrite(SS, HIGH); // вимикаємо Slave Select
SPI.begin ();
SPI.setClockDivider(SPI_CLOCK_DIV8);//роздільник DIV8
}

void loop (void) {
char c;
digitalWrite(SS, LOW); // вмикаємо Slave Select
// send test string
for (const char * p = "Hello ONMU!\r" ; c = *p; p++)
{
SPI.transfer (c);
Serial.print(c);
}
digitalWrite(SS, HIGH); // вимикаємо Slave Select
delay(2000);
}
```

Приклад Arduino SPI як слейв

Ведений блок очікує даних, як тільки дані надходять - змінна процесу стає true, вказуючи, що є дані в буфері. В основному циклі ми читаємо цей буфер і відправляємо в послідовний термінал.

```
#include <SPI.h>
char buff [50];
volatile byte indx;
volatile boolean process;

void setup (void) {
Serial.begin (115200);
pinMode(MISO, OUTPUT); // встановлюємо мастер на output
SPCR |= _BV(SPE); // увімкнути  SPI в режим slave
indx = 0; // буфер порожній
process = false;
SPI.attachInterrupt(); // увімкнути переривання
}

ISR (SPI_STC_vect) // SPI перервання
{
byte c = SPDR; // читати байт з SPI Data Register
if (indx < sizeof buff) {
buff [indx++] = c; // зберегти дані в наступному індексі в
                     //буфері масиву
if (c == '\r') // перевірити закінчення слова
process = true;
}
}

void loop (void) {
if (process) {
process = false; // скинути процес
Serial.println (buff); //друк масиву в serial monitor
indx= 0; //знулення індексу масиву
}
}
```

Відкрийте послідовний монітор веденого пристрою, ви побачите “ Hello ONMU!”.

Виконання лабораторної роботи.

1. Зберіть схему.
2. Завантажте скетчі в контролер-майстер і контролери-слейви.
3. Покажіть робочу систему викладачеві.
4. Модифікуйте програму так, щоб майстер був спроможен по запиту отримувати дані зі слейва. (Повторіть п. 2, 3)

Лабораторна робота «Побудова системи передавання інформації за допомогою I2C»

Загальні відомості

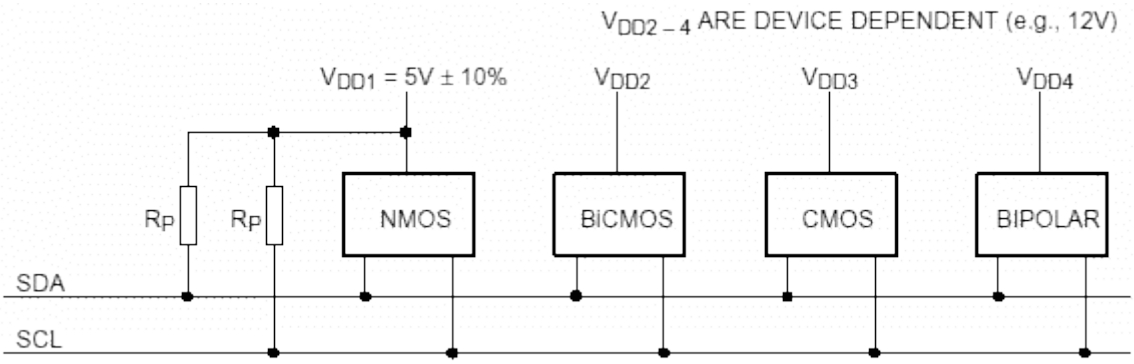
I2C шина є однією з модифікацій послідовних протоколів обміну даних. У стандартному режимі забезпечується передача послідовних 8-бітних даних зі швидкістю до 100 Кбіт/с, і до 400 Кбіт/с в "швидкому" режимі. Для здійснення процесу обміну інформацією по I2C шині, використовується всього два сигнали лінія даних SDA лінія синхронізації SCL для забезпечення реалізації двонаправленості шини без застосування складних арбітрів шини вихідні каскади пристроїв, підключених до шини, мають відкритий стік або відкритий колектор для забезпечення функції монтажного "і".

Проста двопровідна послідовна шина I2C мінімізує кількість з'єднання між ІС, ІС мають менше контактів і потрібно менше доріжок. Як результат - друковані плати стають більш простими і технологічними при виготовленні. Інтегрований I2C-протокол усуває необхідність в дешифраторах адреси та іншої зовнішньої логіці узгодження.

Максимальна допустима кількість мікросхем, приєднаних до однієї шини, обмежується максимальною ємністю шини 400 пФ.

Вбудований в мікросхеми апаратний алгоритм помехоподавления забезпечує цілісність даних в умовах перешкод значної величини.

Всі I2C-сумісні пристрої мають інтерфейс, який дозволяє їм зв'язуватися один з одним по шині навіть в тому випадку, якщо їх напруга живлення істотно відрізняється. На наступному малюнку представлений принцип підключення декількох ІМС з різними напругами живлення до однієї шини обміну.



Кожен пристрій розпізнається за унікальною адресою і може працювати як передавач або приймач, в залежності від призначення пристрою.

Крім того, пристрої можуть бути класифіковані як провідні і ведені при передачі даних. Ведучий-це пристрій, який ініціює передачу даних і виробляє сигнали синхронізації. При цьому будь-який адресується пристрій вважається веденим по відношенню до ведучого.

Виходячи з специфікації роботи шини, в кожен окремий момент в шині може бути тільки один ведучий, а саме той пристрій, який забезпечує формування сигналу SCL шини. Ведучий може виступати як в ролі ведучого-передавача, так і ведучого-приймача. Проте-шина дозволяє мати кілька провідних, накладаючи певні особливості їх поведінки у формуванні сигналів управління і контролю стану шини. Можливість підключення більше одного ведучого до шини означає, що більш ніж один ведучий може спробувати почати пересилання в один і той же момент часу. Для усунення "зіткнень", який може виникнути в даному випадку, розроблена процедура арбітражу - поведінки ведучого при виявленні" захоплення " шини іншим ведучим.

Процедура синхронізації двох пристроїв

Ця процедура заснована на тому, що всі I2C-пристрої підключаються до шини за правилом монтажного і.у вихідному стані обидва сигнали SDA і SCL знаходяться у високому стані.

Адресація в шині I2C

Кожен пристрій, підключений до шини, може бути програмно адресовано за унікальною адресою.

Для вибору приймача повідомлення ведучий використовує унікальний адресну компоненту в форматі посилки. При використанні однотипних пристроїв, ІС часто мають додатковий селектор адреси, який може бути реалізований як у вигляді додаткових цифрових входів селектора адреси, так і у вигляді аналогового входу. При цьому адреси таких однотипних пристроїв виявляються рознесені в адресному просторі пристроїв, підключених до шини.

У звичайному режимі використовується 7-бітна адресація.

Процедура адресації на шині I2C полягає в тому, що перший байт після сигналу СТАРТ визначає, який ведений адресується ведучим для проведення циклу обміну. Виняток становить адреса "загального виклику", який адресує всі пристрої на шині. Коли використовується ця адреса, всі пристрої в теорії повинні послати сигнал підтвердження. Однак, пристрої можуть обробляти "загальний виклик" на практиці зустрічаються рідко.

Перші сім бітів першого байта утворюють адресу веденого. Восьмий, молодший біт, визначає напрямок пересилання даних. "Нуль" означає, що ведучий буде записувати інформацію в обраного веденого. "Одиниця" означає, що ведучий буде зчитувати інформацію з веденого.

Після того, як адреса посланий, кожен пристрій в системі порівнює перші сім біт після сигналу СТАРТ зі своєю адресою. При збігу пристрій вважає себе обраним як ведений-приймач або як ведений-передавач, в залежності від біта напрямку.

Адреса веденого може складатися з фіксованої і програмованої частини.

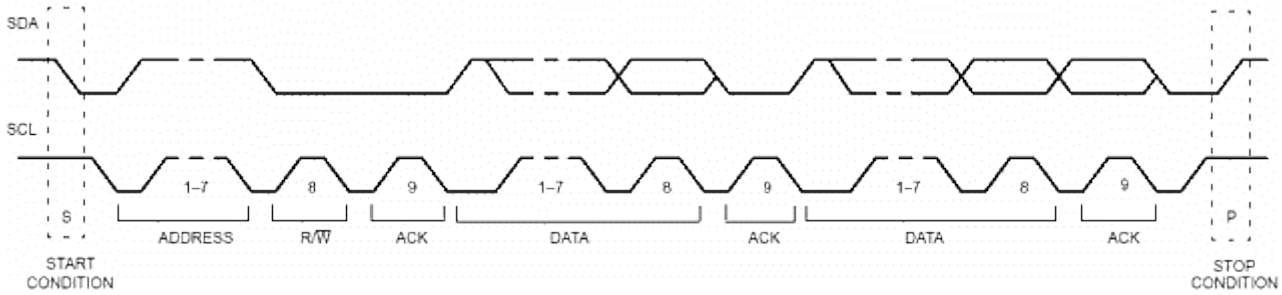
Часто трапляється, що в системі буде кілька однотипних пристроїв (наприклад ІМС пам'яті, або драйверів LED-індикаторів), тому за допомогою програмованої частини адреси стає можливим підключити до шини максимально можливу кількість таких пристроїв. Кількість програмованих

біт в адресі залежить від кількості вільних висновків мікросхеми. Іноді використовується один висновок з аналоговою установкою програмованого діапазону адрес, як це, наприклад, реалізовано в IMC SAA1064. При цьому в залежності від потенціалу на цьому адресному виведенні IMC, можливо зміщення адресного простору драйвера так, щоб однотипні IMC не конфліктували між собою на загальній шині.

Всі IMC, що підтримують роботу в стандарті шини I2C, мають набір фіксованих адрес, перелік яких вказаний виробником в описах контролерів.

Комбінація біт 11110xx адреси зарезервована для 10-бітної адресації.

У загальному вигляді процес обміну по шині від моменту формування стану СТАРТ до стану СТОП можна проілюструвати наступним малюнком :



Як випливає з специфікації шини, допускаються як прості формати обміну, так і Комбіновані, коли в проміжку від стану СТАРТ до стану СТОП ведучий і ведений можуть виступати і як приймач і як передавач даних. Комбіновані формати можуть бути використані, наприклад, для управління послідовною пам'яттю.

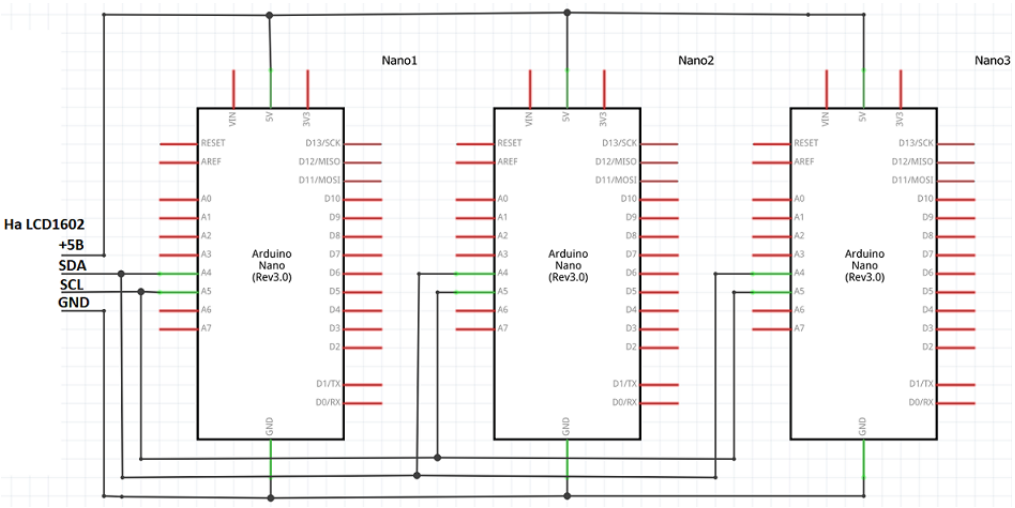
Підключення контролерів по шині I2C

Необхідно взяти три контролера Arduino Nano, об'єднати їх шиною I2C, і організувати обмін даними. Перший контролер буде виконувати роль ведучого, а інші два - роль веденого.

Для відображення даних буду використовувати LCD-індикатор 1602 з модулем I2C, який буде підключений на ту ж комунікаційну шину.

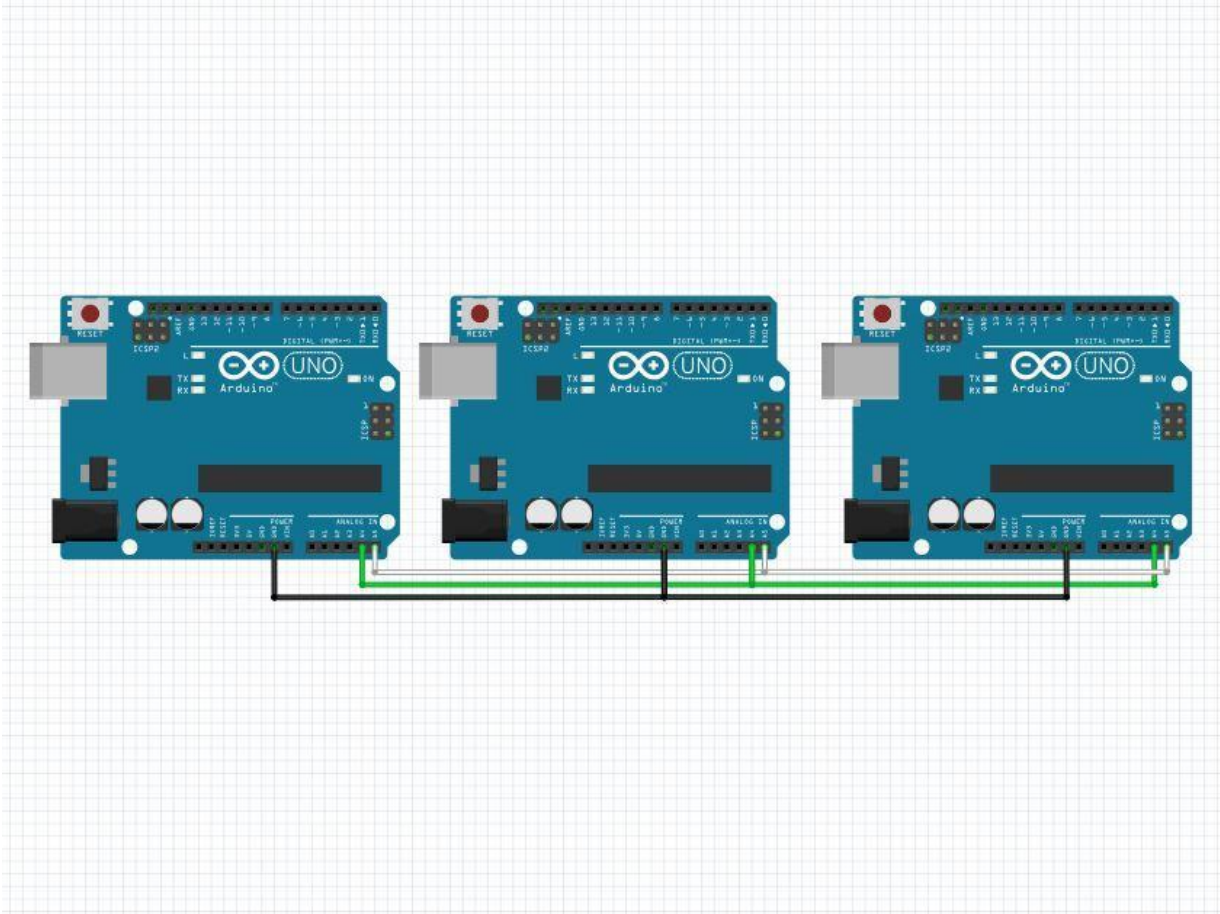
Перший контролер буде послідовно опитувати другого і третього контролера. Прийняті дані перший контролер повинен виводити на індикатор. Опитування ведених Arduino Nano буде проводитися з частотою 1 раз/сек.

Схема підключення



Чотири дроти від кожного з 4-х пристроїв потрібно з'єднати паралельно. Висновок A4 плати Arduino Nano - це шина SDA протоколу I2C, а A5 - це SCL.

Живлення буде подаватися просто на один з контролерів через mini USB вхід.



У LCD адреса в мережі I2C за замовчуванням 27. У другого контролера встановимо адресу 2 і у третього 3. У першого контролера адреса не потрібена.

Програма контролера-майстра.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Set the LCD address to 0x27 for a 16 chars
// and 2 line display
LiquidCrystal_I2C lcd(0x27, 16, 2);
int nano1=0;
int nano2;
int nano3;

void setup()
{
  Serial.begin(9600);
// initialize the LCD
lcd.begin();

// Turn on the blacklight and print a message.
lcd.backlight();
}
```

```
void loop()
{
  lcd.setCursor(0, 0);
  lcd.print(nano1);

  Wire.requestFrom(2, 2); // request 6 bytes from
                          //slave device #8
  int i=0;nano2=0;
  while (Wire.available()) { // slave may send less
                             //than requested
    byte c = Wire.read(); // receive a byte as character
    Serial.print(c);
    if (i==0) nano2 = ((c & 0xff) << 8); else nano2=nano2|c;
    i++;
  }
  Serial.println("");
  Serial.println(nano2);
  lcd.setCursor(9, 0);
  lcd.print(nano2);
  delay(100);

  Wire.requestFrom(3, 2);     // request 6 bytes from slave
  //device #8
  i=0;nano3=0;
  while (Wire.available()) { // slave may send less
                             //than requested
    byte c = Wire.read(); // receive a byte as character
    Serial.print(c);
    if (i==0) nano3 = ((c & 0xff) << 8); else nano3=nano3|c;
    i++;
  }
  lcd.setCursor(0, 1);
  lcd.print(nano3);
  delay(100);

  nano1++;
  delay(800);
}
```

Перший контролер змінює свою змінну типу integer і виводить її значення на індикатор. Так само він по черзі опитує слейв з 2-м і 3-м адресою. Запитує у них два байти інформації, перетворює їх в змінну integer. В результаті в першому контролері крутяться три змінні з трьох Nano і він може вивести їх на індикатор.

Програма другого контролера

```
#include <Wire.h>

int nano2=0;
```

```
byte high[2];
void setup() {
  Wire.begin(2);       // join i2c bus with address #8
  Wire.onRequest(requestEvent); // register event
}

void loop() {
  delay(1000);
  nano2--;
}

// function that executes whenever data is requested
// by master
// this function is registered as an event, see setup()
void requestEvent() {
  high[0] = (nano2 >> 8);
  high[1] = (nano2 & 0xff);
  Wire.write(high[0]); // respond with message of 2 bytes
  Wire.write(high[1]);
}
```

Програма третього контролера

```
#include <Wire.h>

int nano2=0;
byte high[2];
void setup() {
  Wire.begin(3);          // join i2c bus with address #8
  Wire.onRequest(requestEvent); // register event
}

void loop() {
  delay(1500);
  nano2--;
}

// function that executes whenever data is requested
//by master
// this function is registered as an event, see setup()
void requestEvent() {
  high[0] = (nano2 >> 8);
  high[1] = (nano2 & 0xff);
  Wire.write(high[0]); // respond with message of 2 bytes
  Wire.write(high[1]);
}
```

Відрізняються останні дві програми просто адресою в функції Wire.begin (3); і частотою зміни змінної.

Ці програми постійно змінюють змінну integer і очікують запиту від майстра. При запиті ця змінна розкладається на два байти і відправляється як відповідь на запит ведучому контролеру.

Таким чином роботу зв'язку по I2C можна контролювати за змінними значеннями трьох змінних на рідкокристалічному індикаторі.

Виконання лабораторної роботи.

1. Зберіть схему.
2. Завантажте скетчі в контролер-майстер і контролери-слейви.
3. Покажіть робочу систему викладачеві.
4. Модифікуйте програму так, щоб майстер був спроможен по запиту отримувати дані зі відповідного слейва. (Повторіть п. 2, 3)

Бібліотека Serial для роботи з UART Ардуіно.

Для роботи з апаратними UART контролерами в Ардуіно існує вбудований клас Serial. Він призначений для управління обміном даними через UART. Перед тим, як розповісти про функції класу Serial я хочу пояснити різницю в форматі даних обміну.

Через послідовний інтерфейс дані завжди передаються в двійковому коді. Питання як ці дані інтерпретувати, як сприймати. Наприклад, переданий двійковий код " 01000001 "(десятковий 65). Як його відобразити на екрані? Може бути передано число 65 і на екрані треба вивести "65". А може це код літери "A", тоді на екрані треба написати"A". Просто необхідно знати в якому форматі передаються дані.

У 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.

void begin(long speed)

Дозволяє роботу порту UART і задає швидкість обміну в бод (біт в сек). Для завдання швидкості передачі даних рекомендується використовувати стандартні значення (таблиця в розділі "послідовний інтерфейс UART").

Serial.begin (38400); / / ініціалізація порту, швидкість 38400 бод

void end(void)

Відключає порт UART, звільняє висновки RX і TX.

Serial.end (); / / закрити порт UART

int available(void)

Повертає кількість байт, прийнятих послідовним портом і записаних в буфер. Буфер послідовного порту може зберігати до 64 байт. У разі порожнього буфера повертає 0.

int n;

n= Serial. available (); / / в n число прийнятих байтів

int read(void)

Повертає черговий байт з буфера послідовного порту. Якщо буфер порожній-повертає число-1 (0xffff).

receiveByte= Serial.read (); / / читання байта з буфера

void flush(void)

Очікує закінчення передачі даних з буфера послідовного порту.

Serial.flush (); / / чекаємо закінчення передачі

print()

Виводить дані через послідовний порт UART у вигляді ASCII символів. Функція має різні форми виклику для різних форматів і типів даних.

print (char d) якщо аргумент типу char виводить в порт код символу

char d= 83;

Serial.print (d); / / виводить код 83 (символ S)

Serial.print ('S'); / / виводить код 83 (символ S)

print(byte d)

Дані типу Byte виводяться кодом числа

byte d= 83;

Serial.print (d); / / виводить код 83 (символ S)

Serial.print (byte (83)); / / виводить код 83 (символ S)

print (int d) якщо аргумент-цілий тип, то виводить рядок з десятковим поданням числа

int d= 83;

Serial.print (d); / / виводить рядок " 83”

Serial.print (83); / / виводить рядок " 83”

print (float) речові типи виводяться символами ASCII, два знаки після коми

float d= 7.65432;

Serial.print (d); / / виводить рядок " 7.65”

Serial.print (7.65432); / / виводить рядок " 7.65”

print (*str) якщо аргумент покажчик на масив або рядок, то масив або рядок побайтно передається в порт.

char letters[3]= {65, 66, 67};

Serial.print ("букви"); / / виводить рядок " букви”

Serial.print (letters); / / виводить рядок з 3 символів з кодами 65, 66, 67

print (int d, DEC) виводить рядок ASCII-десяткове представлення числа

int d= 83;

Serial.print (d, DEC); / / висновок рядка " 83”

print (int d, HEX) виводить рядок ASCII-шістнадцятиричне представлення числа

int d= 83;

Serial.print (d, HEX); / / висновок рядка " 53”

print (int d, OCT) виводить рядок ASCII-вісімкове представлення числа

int d= 83;

Serial.print(d, OCT); // висновок рядка “123”

print (int d, BIN) виводить рядок ASCII-двійкове представлення числа

int d= 83;

Serial.print (d, BIN); / / висновок рядка " 01010011”

print (int d, BYTE) виводить код молодшого байта числа

int d= 0x0283;

Serial.print (d, BYTE); / / висновок числа 83 (код символу S)

print (float d, N) для дійсних чисел параметр N задає кількість цифр після коми.

Serial.print (7.65432, 0); / / виводить рядок " 7”

Serial.print (7.65432, 2); / / виводить рядок " 7.65”

Serial.print (7.65432, 4); / / виводить рядок " 7.6543”

println()

Виводить дані через послідовний порт UART у вигляді ASCII символів з додаванням символів перенесення рядка (\r, код 13) і (\n, код 10). Тобто наступне повідомлення буде відображатися з нового рядка. В іншому аналогічна функції print ().

int d= 83;

Serial.print (d, DEC); / / висновок рядка " 83”

Serial.println (d, DEC); / / вивід рядка " 83 \ r \ n”

int write()

Виводить двійкові дані через послідовний порт UART. Повертає кількість переданих байтів.

int write (val) передає байт

Serial.write (83); / / передає байт 83

int write (str) передає рядок, як послідовність байтів

int bytesNumber; / / число байтів

bytesNumber= Serial.write ("рядок"); / / передає рядок "рядок", повертає довжину рядка

int write (*buf, len) передає байти з масиву, число байтів – len.

char buf= " рядок”;

Serial.write (buf, 3); / / виводить рядок " стор”

int peek(void)

Повертає наступний байт з буфера послідовного порту, не видаляючи його з буфера. Якщо буфер порожній, то повертає значення -1. Функція повертає те ж значення, що і функція read ().

int readBytes(* buf, len)

Зчитує байти, що надходять на послідовний порт, і записує їх в буфер. Припиняє роботу після прийому заданої кількості байтів або в разі тайм-ауту. Повертає кількість прийнятих байтів. Тайм-аут задається функцією setTimeout ().

setTimeout(long time)

Задає час тайм-ауту для функції readBytes (). Час time вказується в мс, за замовчуванням воно дорівнює 1000 мс.

void serialEvent()

Викликається при надходженні даних в послідовний порт. По суті-переривання по прийому даних послідовним портом.

serialEvent() {

// код для обробки даних порту

}

Застосування класу Serial.

Клас Serial вбудований. Для нього не треба шукати бібліотеку і підключати її. Щоб використовувати UART достатньо в setup () дозволити роботу порту і задати швидкість:

void setup() {

Serial.begin (9600); / / ініціалізуємо порт, швидкість 9600

}

Тепер можна передавати дані за допомогою функцій print () або write ().

Serial.println ("Message to monitor"); / / повідомлення в монітор послідовного порту

Якщо через порт виводиться кілька байтів, то клас Serial записує їх в програмний буфер і послідовно по байту передає дані в контролер UART в міру передачі.

Опис методів бібліотеки SPI

Для написання коду для нового SPI пристрою вам необхідно звернути увагу на кілька моментів:

* Яка максимальна швидкість SPI, яку може використовувати Ваш пристрій? Вона управляється першим параметром в SPISettings. Якщо ви використовуєте чіп з тактовою частотою 15 МГц, використовуйте значення 15000000. Arduino буде автоматично використовувати кращу швидкість, рівну або меншу, ніж значення, яке ви використовували в SPISettings.

• В якому порядку передаються дані: спочатку йде старший значущий біт (MSB, Most Significant Bit) або молодший значимий біт (LSB, Least Significant Bit)? Він управляється другим параметром в SPISettings: або MSBFIRST, або LSBFIRST. Більшість SPI чіпів використовують порядок з йде першим MSB.

* На лінії даних під час очікування повинен бути встановлений високий або низький логічний рівень? Вибірка даних відбувається по наростаючому або по спадаючому фронту тактових імпульсів? Ці режими контролюються третім параметром в SPISettings.

Стандарт SPI надає свободу, і кожен пристрій реалізує з невеликими відмінностями. Це означає, що при написанні коду ви повинні приділити особливу увагу технічному опису пристрою.

SPISettings

Опис

Об'єкт SPISettings використовується для налаштування SPI порту вашого пристрою. Всі три параметри об'єднуються в один об'єкт SPISettings, який передається методу SPI.beginTransaction().

Коли всі ваші параметри є константами, SPISettings слід використовувати безпосередньо в SPI.beginTransaction(). Синтаксис дивіться

нижче. При константах цей синтаксис призводить до меншого і швидшого коду.

Якщо будь-які з ваших параметрів є змінними, ми можете створити Об'єкт SPISettings для зберігання трьох параметрів. Потім ви можете передати методу SPI.beginTransaction () ім'я цього об'єкта. Створення іменованого Об'єкта SPISettings може бути більш ефективним, якщо ваші параметри не є константами, особливо якщо максимальна швидкість – це змінна, обчислена або налаштована, а не число, яке ви керуєте безпосередньо в коді скетча.

Синтаксис

SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0));

Примітка: найкраще підходить, якщо всі 3 параметри – константи.

SPISettings mySettting(speedMaximum, dataOrder, dataMode);

Примітка: найкраще підходить, якщо всі 3 параметри – змінні.

Параметр

* speedMaximum: максимальна швидкість зв'язку. Для SPI чіпа з тактовою частотою 20 МГц використовуйте 20000000.

* dataOrder: MSBFIRST або LSBFIRST.

* dataMode: SPI_MODE0, SPI_MODE1, SPI_MODE2 або SPI_MODE3.

Значення, що повертається

Ні.

begin()

Опис

Ініціалізує шину SPI установкою SCK, MOSI і SS в режим виходу, установкою на SCK і MOSI низького логічного рівня, а на SS – високого логічного рівня.

Синтаксис

SPI.begin();

Параметр

Ні.

Значення, що повертається

Ні.

end()

Опис

Відключає шину SPI (режими роботи висновків не змінюються).

Синтаксис

SPI.end();

Параметр

Ні.

Значення, що повертається

Ні.

beginTransaction()

Опис

Ініціалізує шину SPI, використовуючи переданий об'єкт SPISettings.

Синтаксис

SPI.beginTransaction(mySettings);

Параметр

mySettings: вибрані налаштування відповідно до SPISettings (дивіться вище).

Значення, що повертається

Ні.

endTransaction()

Опис

Зупиняє використання шини SPI. Зазвичай викликається після скасування вибору веденого чіпа, щоб дозволити використовувати шину SPI іншим бібліотекам.

Синтаксис

SPI.endTransaction();

Параметр

Ні.

Значення, що повертається

Ні.

setBitOrder()

Опис

Ця функція не повинна використовуватися в нових проектах. Для налаштування параметрів використовуйте SPISettings з SPI.beginTransaction().

Встановлює порядок передачі бітів на шину SPI: або MSBFIRST (спочатку старший значущий біт), або LSBFIRST (спочатку молодший значущий біт).

Синтаксис

SPI.setBitOrder(order);

Параметр

order: LSBFIRST або MSBFIRST.

Значення, що повертається

Ні.

setClockDivider()

Опис

Ця функція не повинна використовуватися в нових проектах. Для налаштування параметрів використовуйте SPISettings з SPI.beginTransaction().

Встановлює дільник частоти тактового сигналу SPI относительно тактової частоти системи. На платах на базі AVR доступні наступні значення дільника: 2, 4, 8, 16, 32, 64 або 128. Параметр за замовчуванням – це

SPI_CLOCK_DIV4, який встановлює частоту тактового сигналу SPI, що дорівнює чверті від частоти опорного генератора мікроконтролера (4 МГц для плат 16 МГц).

Arduino Due

На платі Due частота опорного генератора може бути поділена на значення від 1 до 255. Значення за замовчуванням-21, що встановлює частоту тактового сигналу на 4 МГц, як і на інших платах Arduino.

Синтаксис

SPI.setClockDivider(divider);

Параметр

* divider: SPI_CLOCK_DIV2, SPI_CLOCK_DIV4, SPI_CLOCK_DIV8, SPI_CLOCK_DIV16, SPI_CLOCK_DIV32, SPI_CLOCK_DIV64 або SPI_CLOCK_DIV128(тільки на платах AVR);

* slaveSelectPin: виведення веденого пристрою SS (тільки на Arduino Due);

* divider: число від 1 до 255 (тільки на Arduino Due).

Значення, що повертається

Hi.

setDataMode()

Опис

Ця функція не повинна використовуватися в нових проектах. Для налаштування параметрів використовуйте SPISettings з SPI.beginTransaction().

Встановлює режим даних SPI: полярність і фазу тактового сигналу. Для більш детальної інформації про інтерфейс SPI дивіться статтю " Назад до основ: SPI (послідовний периферійний інтерфейс)».

Синтаксис

SPI.setDataMode(mode);

Параметр

* mode: SPI_MODE0, SPI_MODE1, SPI_MODE2 або SPI_MODE3;

* slaveSelectPin: виведення веденого пристрою SS (тільки на Arduino Due).

Значення, що повертається

Ні.

transfer(), transfer16()

Опис

Передача SPI заснована на одночасних відправці і прийомі: прийняті дані повертаються в receivedVal (або receivedVal16). У разі передачі буфера, отримані дані зберігаються в цьому ж місці (старі дані замінюються прийнятими даними).

Синтаксис

receivedVal = SPI.transfer(val);

receivedVal16 = SPI.transfer16(val16);

SPI.transfer(buffer, size);

Параметр

* val: байт, який необхідно передати по шині;

* val16: змінна з двох байтів, яку необхідно передати по шині;

* buffer: масив даних для передачі.

Значення, що повертається

Прийняті дані.

usingInterrupt()

Опис

Якщо ваша програма здійснює передачі SPI всередині обробника переривання, вивовите дану функцію, щоб зареєструвати номер або ім'я переривання в бібліотеці SPI. Це дозволить SPI.beginTransaction () запобігти конфліктам використання. Зверніть увагу, що переривання, вказане при

виклику usingInterrupt () буде відключено при виклику beginTransaction () і знову включено в endTransaction ().

Синтаксис

SPI.usingInterrupt(interruptNumber);

Параметр

interruptNumber: відповідний номер переривання.

Значення, що повертається

Прийняті дані.