Лабораторний практикум з курсу Програмування робототехнічних систем спеціальність: Середня освіта (Інформатика)

Біляй Юрій Петрович

20 січня 2019 р.

Зміст

Розділ 1

Основи елетричних схем

1.1 Основні закони електрики

1.1.1 Закон Ома

Закон Ома – головний закон електрики $I = \frac{U}{R}$

1.1.2 Потужність

Потужність – міра швидкості зміни електричної енергії в іншу форму $P = I \cdot U$

Знаючи закон Ома, потужність можна обчилити $P = I^2 \cdot R = \frac{U^2}{R}$

В процесі роботи частина енегрії трансформується в тепло. Через це гріються комп'ютери, телефони та інша техніка. $P = P_W + P_D$, де P – споживча потужність, P_W – корисна потужність, P_D – потужність, що переходить в нагрівання.

1.1.3 Коротке замикання

З'єднання плюс і мінус напряму, за законом Ома, ожначає дуже велику силу струму і як наслідок до дуже великої потужності нагріваннння, що у результаті спричиняє згорання. Такий процес називається коротким замиканням. Ніколи не допускайте його появу, ні за яких обставин!

1.1.4 Послідовне з'єднання

При послідовному з'єднанні сила струму на кожному споживачеві однакова, а напруга падає в кожному наступному споживачеві.

1.1.5 Паралельне з'єднання

При паралельному з'єднанні напруга навколого кожного споживача однакова. Сила струму залежить від опору кожного окремого споживача.

1.2 Основні компоненти

1.2.1 Конденсатор

Конденса́тор (англ. capacitor) – система з двох чи більше електродів (обкладок), які розділені діелектриком, товщина якого менша у порівнянні з розміром обкладок. Така система має взаємну електричну ємність і здатна зберігати електричний заряд.

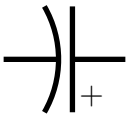
Конденсатор є пасивним електронним компонентом і широко застосовується в електронних схемах для блокування постійного струму, пропускаючи змінний струм.

Конденсатор це маленький акумулятор який швидко заряджається і швидко розряджається.

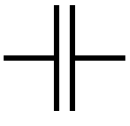
Основні характеристики

Назва	*Позначення*	*Одиниці виміру*
Ємність (номінал)	C	Фарад
Точність (допуск)	$\pm$	%
Максимальна напруга	V	Вольт

Електролітичний конденсатор має плюс і мінус, мінус відрізняється більш короткою ніжкою і/або білою смугою на корпусі.



Керамічний конденсатор

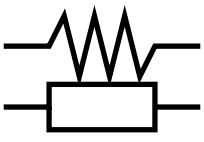


1.2.2 Резистор

Резистори належать до електричних компонентів, що застосовуються в схемах електротехніки та електроніки для обмеження сили струму та розподілу напруги. Резистори — найпоширеніші пасивні компоненти електронної апаратури, що використовуються як навантаження, споживачі та подільники в колах живлення, як елементи фільтрів, шунти, в колах формування імпульсів і т.д.

Основні характеристики.

Назва	*Позначення*	*Одиниці виміру*
Опір (номінал)	R	Ом
Точність (допуск)	$\pm$	%
Потужність	P	Ватт



Кольорове кодування резисторів. Наносити номінал резистора на корпус числами – дорого і непрактично: вони виходять дуже дрібними. Тому номінал і допуск кодують кольоровими смужками.

При з'єднанні резисторів *послідовно* їх еквівалентною схемою буде резистор з опором, рівним сумі опору окремих резисторів:

$$R = R_1 + R_2 + \cdots + R_N = \sum_{i=1}^{N} R_i$$

При *паралельному* з'єднанні резисторів обернена величина еквівалентного опору (провідність) дорівнює сумі обернених величин усіх опорів (провідностей).

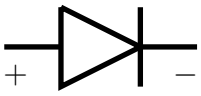
$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} + \cdots + \frac{1}{R_N} = \sum_{i=1}^{N} \frac{1}{R_i}$$

1.2.3 Діод

Діод – електронний прилад з двома електродами, що пропускає електричний струм лише в одному напрямі.

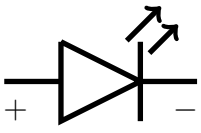
Основні характеристики.

Назва	*Позначення*	*Одиниці виміру*
Падіння прямо напруги	V_F	Вольт
Максимальна обернена напруга яку може стримати	V_{DC}	Вольт
Максимальний прямий потік	I_F	Ампер



1.2.4 Світлоіод

Світлодіод – вид діода, який світиться, коли через нього проходить струм від анода(+) до катода(-). Ніжка анода довша за ножку катоду.



Назва	*Позначення*	*Одиниці виміру*
Падіння прямої напруги	V_F	Вольт
Номінальни струм	I	Ампер
Інтенсивність (яскравість)	I_V	Кандела

Власний опір світлодіода після насичення досить малий тому без резистора, що обмежує струм, він перегорить.

Вибір потрібного резистора. Розрахуємо який резитор опору R потрібно взяти щоб отримати оптимальний результат. Для прикладу нехай характеристики світлодіода який потрібно під'єднати такі: $V_F = 2.3\text{В}, I = 20\text{мА}$, напруга джерела живлення $V_{CC} = 5\text{В}$.

Знайдемо оптимальний опір R і мінімально допустиму потужність резистора P_R. Спочатку обчислимо яку напругу повинен взяти на себе резистор: $U_R = V_{CC} - V_F = 5\text{В} - 2.3\text{В} = 2.7\text{В}$. За законом Ома знайдемо значення опору, за якого буде забезпечено таке падіння: $R = \frac{U_R}{I} = \frac{2.7\text{В}}{0.02\text{А}} = 135\text{Ом}$. Таким чином при опорі більше за 135Ом яскравість буде нижчою за заявлену у специфікації виробу, при опорі меншому за 135Ом строк роботи світлодіода буде меншим.

Тепер знайдемо потужність, яку за такого процесу, резистору доведеться розсіювати:

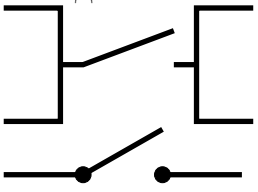
$P_R = I^2 \cdot R = 0.02\text{А}^2 \cdot 135\text{Ом} = 0.054\text{Вт}$.

Це означає, що за потужності резистора меншій за 54 мВт резистор перегорить.

1.2.5 Кнопка

Тактова кнопка – простий механізм, за допомого якого можна замкнтуи електричне коло поки кнопка натиснута.

Кнопки з 4 контактами можна розглядати як 2 пари рельс, що з'єднуються під час натиснення.



Під час замикання та розмикання контактів кнопки між пласниами кнопки виникають мікроіскри, за рахонок чого відбуваються десятки перемикань за декілька мілісекунд. Це явище називається "дрижанням"(англ. *bounce*). Цей ефект потрібно враховувати якщо потрібно фіксувати натиснення.

Поки кнопка натиснута, вихідна напруга $V_{out} = V_{cc}$, але коли вона відпущена, $V_{out} \neq 0$. Кнопка і провідники в цьому випадку працюють як антена, і V_{out} буде «шуміти», приймаючи випадкові значення «з повітря».

Поки з'єднання немає, необхідно створити резервний шлях для визначення напруги. Для цього використовують один з двох варіантів.

Якщо є натиснення: $V_{out} = V_{cc}$.

Якщо немає натиснення: $V_{out} = 0$.

Якщо є натиснення: $V_{out} = 0$.

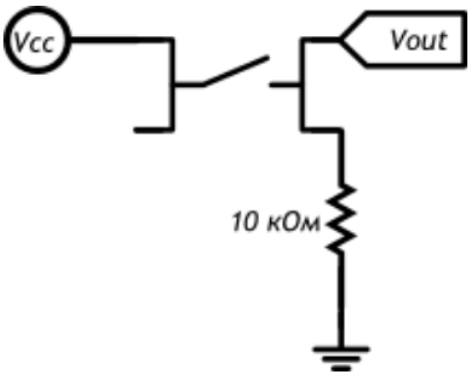


Рис. 1.1: Схема з стягуючим резистором

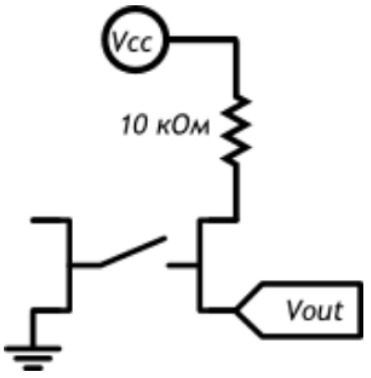


Рис. 1.2: Схема з підтягуючим резистором

Якщо немає натиснення: $V_{out} = V_{cc}$.

1.2.6 Широтно-імпульсна модуляція

Мікроконтролери зазвичай не можуть видавати довіну напругу. Вони можуть видати або напругу живлення (наприклад, 5В), або землю (тобто 5В).

Але рівнем напруги можна управляти багатьма присторями: наприклад, яскравість світлодіода або швидкість обертання мотора. Для симуляції неповної напруги використовується ШІМ (Широтно-Імпульсна Модуляція, англ. *Pulse Width Modulation* або просто PWM).

Вихід мікроконтролера перемикається між землею і V_{cc} тисячі разів в секунду. Або, як ще кажуть, має частоту в тисячі герц. Око не помічає мерехтіння більше 50Гц, тому нам здається, що світлодіод не мерехтить, а горить в півсили.

Аналогічно, розігнаний мотор не може зупинити вал за мілісекунди, тому ШІМ-сигнал змусить обертатися його в неповну силу.

1.2.7 Дільник напруги

Послідовно підключені резистори ділять напругу, що надходить на них у певній пропорції.

Сила струму, що протікає через резистори однакова, тому що вони з'єднані послідовно, і по закону Ома може бути розрахована як:

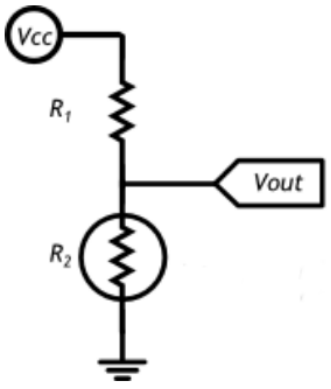
$$I = \frac{V_{CC}}{R_1 + R_2}$$

За тим же законом Ома можна обчислити напругу V_{out}, яке падає на резисторі R_2:

$$V_{out} = U_2 = I \cdot R_2 = \frac{R_2 \cdot V_{CC}}{R_1 + R_2}$$

З отриманої формули видно, що чим більше R_2 щодо R_1, тим більша напруга падає на ньому.

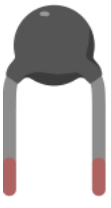
Зчитування резистивних сенсорів. Якщо замість R_2 використовувати не постійний резистор, а датчик, який змінює свой опір, V_{out} буде залежати від вимірюваного значення.



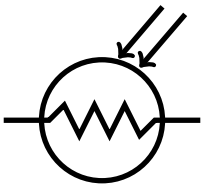
Використовуючи мікроконтролер можна вимірювати напругу. Таким чином, можна використовувати властивості дільника напруги для отримання показань від сенсора.

Приклади резистивних датчиків

Термістор змінює свій опір залежно від власної температури.



Фоторезистор змінює свій опір залежно від сили світла, що потрапляє на його керамічну "змійку".

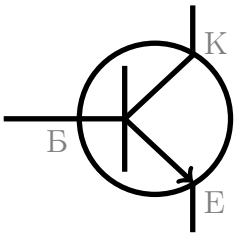


Потенціометр ще називають змінним резистором, тримерами. Це дільник з двох резисторів в одному корпусі. Тому у нього 3 ноги: харчування, вихід, земля.

Співвідношення R_1 і R_2 змінюється поворотом ручки. Від 100% на користь R_1 до 100% на користь R_2.

1.2.8 Біполярний транзистор

Транзистор – це електронна кнопка. На кнопку натискають пальцем, а на біполярний транзистор – струмом.



Транзистори використовують для управління потужними джерелами струму за допомогою слабких сигналів з мікроконтролера.

Нога, що виконує роль «кнопки» називається база (англ. *base*). Поки через базу тече невеликий струм, транзистор відкритий: Великий струм може втікати в колектор (англ. *collector*) і витікати з емітера (англ. *emitter*)

Основні характеристики

Назва	*Позначення*	*Одиниці виміру*
Максимальна напруга колектор-емітер	V_{CE}	Вольт
Максимальний струм через колектор	I_C	Ампер
Коефіцієнт посилення	h_{fe}	

Типова схема підключення

Транзистор підсилює максимально допустимий струм в h_{fe} раз:

$$I_{CE} = I_{BE} \cdot h_{fe}$$

Приклад розрахунку

Якщо керуючий сигнал на базі транзистора з h_{fe} і резистором номіналом 1кОм становить 5В:

Який максимальний струм зможе пропустити через себе транзистор (I_{BE})?

Яким за величиною буде керуючий струм (I_{CE})?

$V_B = 5\text{В}$, $R = 1\text{кОм}$, $h_{fe} = 50$.

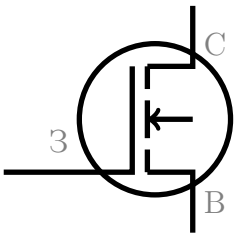
$I_{BE} = \dfrac{V_B}{R} = \dfrac{5\text{В}}{1000\text{Ом}} = 5 \cdot 10^{-3}\text{А}.$

$I_{CE} = I_{BE} \cdot h_{fe} = 250 \cdot 10^{-3}\text{А}.$

Отже, якщо на базу подається 5В через резистор в 1кОм, транзистор відкриється настільки, що зможе пропустити до 250мА. При цьому керуючий струм складе всього 5мА.

1.2.9 Польовий транзистор

Польовий MOSFET-транзистор – ключ для управління великими струмами за допомогою невеликого напруги.



"Кнопка"називається затвором (англ. *gate*). Поки на затворі є невелика напруга, транзистор відкритий: великий струм може втікати в стік (англ. *drain*) і витікати з витоку (англ. *source*).

На відміну від біполярного транзистора польовий контролюється саме напругою, а не струмом. Тобто у відкритому стані струм через затвор не проходить.

Використовуйте MOSFET для управління великими струмами, від сотень міліампер, коли біполярного транзистора вже не досить.

Основні характеристики

Назва	*Позначення*	*Одиниці виміру*
Максимальна напруга стік-витік	V_{DS}	Вольт
Максимальний струм через стік	I_D	Ампер
Опір стік-витік	R_{DSon}	Ом
Потужність, що розсіюється	P_D	Ватт

Транзистор не ідеальний і частина пропускної потужності перетворюється в тепло.

$P_H = I^2 \cdot R_{DSon}$

Якщо P_H перевищить P_D, без використання додаткового охолодження транзистор згорить.

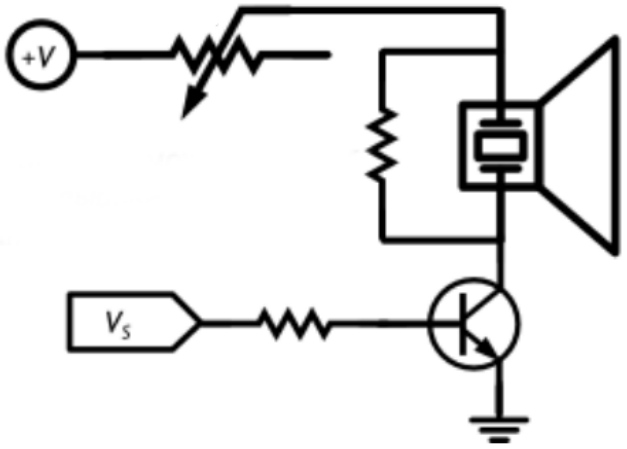
1.2.10 П'єзодинамік

П'єзогенератор звуку (англ. *buzzer*) переводить змінну напругу в коливання мембрани, яка в свою чергу створює звукову хвилю. П'єзодинамік – це конденсатор, який звучить при зарядці і розрядці.

Основні характеристики

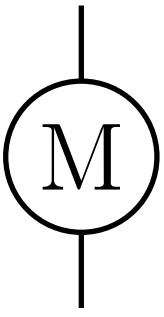
Назва	*Позначення*	*Одиниці виміру*
Номінальна напруга	V	Вольт
Гучність (на заданій відстані)	P	Децибел
Пікова частота	f_P	Герц
Ємність	C	Фарад

Підключення з регулюванням гучності. За допомогою потенціометра можна зменшити струм за рахунок чого зменшиться гучність. За допомогою ризистора можна вирівняти напругу поки транзистор закритий.



1.2.11 Мотор

Мотор перетворює електричну енергію в механічну енергію обертання.



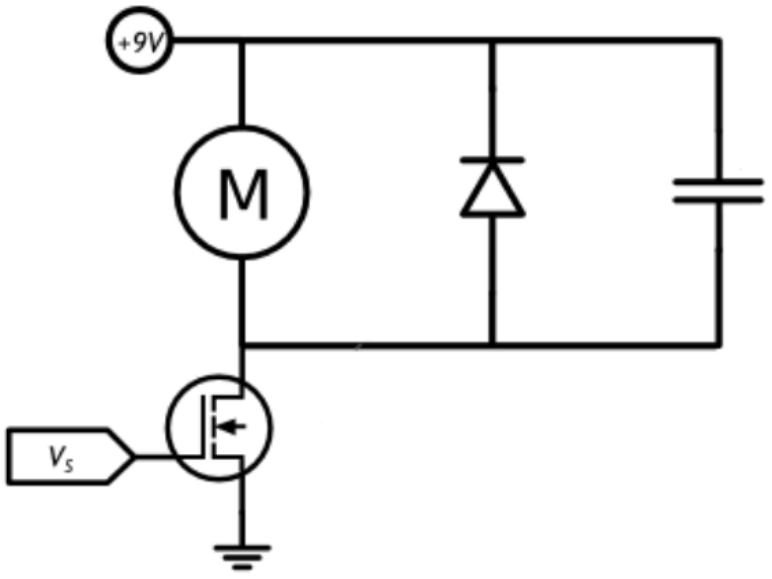
Найпростіший вид мотора – колекторний. При подачі напруги в одному напрямку вал крутиться за годинниковою стрілкою, в зворотному напрямку – проти годинникової.

Основні характеристики

Назва	*Позначення*	*Одиниці виміру*
Номінальна напруга	V	Вольт
Струм споживання без навантаження	I_F	Ампер
Струм споживання при блокуванні	I_S	Ампер
Швидкість обертання без навантаження	ω	c^{-1}
Максимальний крутний мометн	τ	Н×м

Мотори – потужні споживачі з рядом побічних ефектів. Для управління ними необхідні додаткові компоненти.

Для використання мотора доцільніше використовувати додаткове джерело живлення з рекомендованою для мотора напругою (+9V). За допомогою транзистора можна управляти потужним моторм за допомого слабкого сигналу V_S. ШІМ-сигнал використовують для регулювання швидкістю обертання вала. Під час роботи мотор створює електромагнітні шуми. Використання конденсатора дозволяє згладити пульсації. Коли відбувається гальмування мотора, він працює як генератор, створюючи напругу зворотної полярності. Діод потрібен для пропускання утвореного струму чез себе. Без діода вийде з ладу керуючий транзистор.



Розділ 2

Основи роботи з Arduino

2.1 Arduino

Що таке Arduino?

Arduino - це open-source платформа, яка складається з двох основних частин: самої плати (часто званої мікроконтролер) і програмного забезпечення (спеціальної оболонки для програмування плати) або IDE (Integrated Development Environment). Програмне забезпечення запускається на персональному комп'ютері і дозволяє записувати розроблений вами код на плату. Загальна інформація про Arduino

Arduino знайшли особливо сильну популярність серед людей, які тільки починають займатися електронікою. На те є кілька причин. На відміну від більшості попередників, Arduino не вимагає додаткового обладнання (програматора) для завантаження коду на плату - використовується простий USB-кабель. Оболонка для програмування - Arduino IDE використовує спрощену версію C ++, що полегшує процес навчання для новачків. Крім того, Arduino використовує стандартизований форм фактор для більшості своїх плат, завдяки чому з'явився цілий комплект додаткових "Шілд".

Arduino Uno показана на малюнку нижче:

Arduin Uno

Arduino Uno - одна з найпопулярніших плат в лінійці і є відмінним вибором для початківців. Технічні характеристики цієї моделі будуть розглянуті нижче.

Оболонка Arduino IDE:

Arduino IDE

Чи повірите чи ні, але показані на малюнку вище 10 рядків коду достатньо, щоб змусити блимати вбудований на плату світлодіод. Можливо, сам код для вас зараз не дуже зрозумілий, але повірте, він гранично логічний і лаконічний. Після цієї статті і декількох туторіали, вам не складе

труднощів його реалізувати самостійно.

У цій статті ми зупинимося на таких основних моментах:

Які проекти можна реалізувати з Arduino Основні вузли плат Arduino Номенклатура найвдаліших моделей Arduino Додаткові (периферійні) пристрої для Arduino

Рекомендуємо також додатково почитати

Arduino призначена не тільки для вузькоспеціалізованих фахівців. При цьому процес їх освоєння буде набагато легше і приємніше, якщо у вас за плечима базові знання схемотехніки і електротехніки. Рекомендуємо отримати хоча б загальне розуміння перерахованих нижче речей перш ніж заглиблюватися в дивовижний світ Arduino:

Що таке електрика? Закон Ома Електричний ланцюг Інтегральна схема (мікросхема) аналоговий сигнал цифровий сигнал

Навіщо вам Arduino?

Arduino розроблена для ... Всіх. Так, у всякому разі, заявлено на офіційному сайті компанії. Список приблизно такий: артисти, дизайнери, хакери, програмісти, інженери, для всіх, хто цікавиться розробкою і втіленням інтерактивних проектів. Arduino може взаємодіяти з кнопками, світлодіодами, двигунами, динаміками, GPS-модулями, температуру, камерами, інтернетом і навіть вашим смартфоном або телевізором! Подібна гнучкість в поєднанні з тим, що софт від Arduino - абсолютно безкоштовний, самі плати досить дешеві і легкі в освоєнні привела до появи величезної спільноти шанувальників даної платформи, які викладають власні шматки коду, бібліотеки та інструкції для величезної кількості проектів з використанням Arduino.

Arduino використовуються в якості "мізків"для роботів, 3D принтерів, в системах автоматизованого поливу, світлодіодних кубах, грілках, в системах "розумних будинків"і т.д. Список постійно зростає. Всі проекти і не перерахуєш. Скажімо так: Arduino знаходять застосування практично в будь-якому проекті, де необхідна автоматизація.

Arduino_Game_Pad_Project

І це тільки вершина айсберга. Якщо вам цікаво поглянути на Arduino проекти в дії, ось кілька посилань на хороші ресурси (англійською мовою):

Instructables Bildr Arduino Playground The ITP Physical Computing Wiki LadyAda Make: Projects

З чого складається плата Arduino?

Випускаються різні моделі Arduino. Кожна з них "заточена"для різних завдань. Деякі плати принципово відрізняються від наведеної на малюнку нижче. Але більшість з них мають такі однакові вузли:

Arduino Uno - опис Роз'єм живлення (USB / роз'єм для адаптера)

Кожна плата Arduino повинна приєднуватися до джерела живлення.

Arduino Uno може живитися від USB кабелю від вашого персонального комп'ютера Або від окремого адаптера, який підключається до передбаченого на платі роз'єму. На малюнку з'єднання через USB відзначено (1), а роз'єм для зовнішнього джерела живлення - (2).

USB також використовується для завантаження вашої програми (скетчу) на плату.

Примітка! Не використовуйте джерело живлення з напругою на виході більше 20 вольт. Це може привести до того, що ваша плата перегорить. Рекомендоване напруга живлення для Arduino - від 6 до 12 вольт. Роз'єми (Піни) (5V, 3.3V, GND, Analog, Digital, PWM, AREF)

Піни на вашій платі Arduino - це передбачені роз'єми, до яких ви будете підключати дроти від периферійних пристроїв (дуже часто для прототипів використовують монтажні плати (макетна плата, макетке) і дроти з коннекторами на кінцях). На Arduino кілька типів пинов, кожен з яких підписаний відповідно до виконуваної функцією.

GND (3): скорочення від 'Ground' - 'Земля'. На платах кілька пинов GND, кожен з яких може використовуватися для заземлення вашої електричного кола. 5V (4) і 3.3V (5): як ви могли вже здогадатися - піти, які на виході забезпечують харчування 5 вольт і 3.3 вольт відповідно. Більшість компонентів, які підключаються до Arduino, благополучно харчуються саме від 5 або 3.3 вольт. Analog (6): на ділянці, який підписаний 'Analog In' (від A0 до A5 на Arduino Uno) розташовані аналогові входи. Ці Піни дозволяють зчитувати сигнали від аналогових датчиків (наприклад, датчик температури) і перетворювати їх в цифрові значення, якими ми надалі оперуємо. Digital (7): навпроти аналогових пинов знаходяться цифрові Піни (від 0 до 13 на Arduino Uno). Ці Піни використовуються для цифрових вхідних (input) сигналів (наприклад, натискання кнопки) і для генерації цифрових вихідних (output) сигналів (наприклад, харчування світлодіода). PWM (8): ви напевно помітили знак ( ) поряд з деякими цифровими пинами (3, 5, 6, 9, 10, і 11 на UNO). Ці Піни працюю як в звичайному цифровому режимі, так і в режимі ШІМ-модуляції (PWM). Якщо пояснити коротко - ці Піни можуть імітувати аналоговий вихідний сигнал (наприклад, для поступового згасання світлодіода). AREF (9): Цей пін використовується досить рідко. У деяких випадках це підключають в схему для установки максимального значення напруги на аналогових входах (від 0 до 5 вольт).

Кнопка скидання (Reset Button)

Як і на оригінальних Nintendo, на Arduino є кнопка скидання (reset) (10). При натисканні на неї контакт скидання замикається з землею і код, завантажений на Arduino починає відпрацьовувати заново. Корисна опція, якщо ваш код відпрацьовує без повторів, але ви хочете протестить його

роботу. Індикатор живлення (Power LED)

Трохи праворуч і нижче написи "UNO"встановлений світлодіод, підписаний "on"(11). Цей світлодіод повинен загорітися, коли ви підключили Arduino до джерела живлення. Якщо світлодіод не зайнявся - поганий знак;). Світлодіоди TX і RX

TX - скорочення від transmit (передача), RX - від receive (прийом). Ці умовні позначення часто зустрічаються в електроніці для позначення контактів, які відповідають за серійний обмін даними. На Arduino Uno ці контакти зустрічаються два рази на цифрових пінах 0 і 1 і в якості світлодіодів TX і RX (12). Ці світлодіоди дозволяють візуально відслідковувати, передає або приймає дані Arduino (наприклад, при завантаженні програми на плату). Головна інтегральна мікросхема (IC)

Чорна деталь з металевими коннекторами з двох сторін це інтегральна мікросхема, мікропроцесор (IC або Integrated Circuit) (13). Можете сміливо вважати, що це "мізки"нашої Arduino. Цей чіп різний в різних моделях Arduino, але зазвичай він відноситься до лінійки мікропроцесорів ATmega від компанії ATMEL. Це може виявитися важливою інформацією для завантаження скетчу на плату. Модель інтегральної мікросхеми зазвичай вказана на її верхній корпусних частини. Для додаткової інформації про вашу мікросхемі варто звернутися до її даташіту. Регулятор напруги

Регулятор напруги (14) is виконує функцію, вказану в назві - контролює напругу, яка надходить на плату Arduino. Можете його собі уявити як охоронця, який не пропускає занадто велика напруга на плату, щоб уникнути її ушкоджень. Звичайно ж, у регулятора є своя межа. Так що живити Arduino напругою більше 20 вольт не можна. Номенклатура плат Arduino

Arduino виробляє різні плати, кожна з який має власні особливості. Крім того, Arduino дотримуються моделі open source, завдяки чому інші можуть модифікувати і виробляти клони Arduino, розширювати і змінювати їх функціонал і форм-фактор. Нижче наведені короткі відомості про різні моделі Arduino. Arduino Uno (R3)

Arduino Uno - відмінний вибір для початківців. Дуже збалансована плата, на якій є, все, що вам може знадобитися і мінімум зайвого. На платі 14 цифрових пинов, які працюю на вхід і на вихід (6 з них підтримують ШІМ-модуляцію), 6 аналогових входів. Підключається плата за допомогою USB. Є джек для окремого джерела живлення, кнопка скидання і т.п. Для початку роботи з мікро контролером достатньо підключити плату до комп'ютера за допомогою USB кабелю.

Arduino Uno R3 LilyPad Arduino

Основна плата в лінійки LilyPad Arduino! LilyPad розроблена в першу чергу для використання на одязі. Піни з'єднуються з периферійними при-

строями за допомогою струмопровідної нитки. Є купа додаткових плат розширень для LilyPad. Більшість з них спроектовані таким чином, що не бояться вологи.

LilyPad Arduino

Основна плата в лінійки LilyPad Arduino! LilyPad розроблена в першу чергу для використання на одязі. Піни з'єднуються з периферійними пристроями за допомогою струмопровідної нитки. Є купа додаткових плат розширень для LilyPad. Більшість з них спроектовані таким чином, що не бояться вологи.

LilyPad Arduino RedBoard

Розробка SparkFun, яка програмується за допомогою USB Mini-B кабелю в оболонці Arduino IDE. Основними перевагами виробник називає: стабільність роботи під ОС Windows 8 завдяки драйверам з необхідною цифровим підписом. На платі використовується чіп USB / FTDI, який менше за габаритами в порівнянні з чіпом на Arduino UNO. Для заливки скетчу на плату в IDE вибирається модель Arduino UNO. Регулятор напруги розрахований на харчування в діапазоні від 7 до 15 вольт.

RedBoard Arduino Arduino Mega (R3)

Arduino Mega - немов старший брат Uno. На платі багато (54!) Цифрових входів / виходів (14 з них підтримують ШІМ-модуляцію). Завдяки великій кількості пінів, плата використовується для комплексних проектів, в яких підключається велика кількість периферії (наприклад, сведодіодов або кнопок). Підключення до комп'ютера реалізується таким же кабелем як і на Arduino Uno. Природно, передбачений джек для адаптера.

Arduino Mega Arduino Leonardo

Leonardo - перша розробка Arduino, в якій використовується один мікроконтролер із вбудованим USB. Це означає, що плата стає простіше і дешевше. Так як плата підключається безпосередньо до USB без конвертера, є бібліотеки, які дозволяють емулювати комп'ютерну мишу, клавіатуру і багато іншого!

Arduino Leonardo Arduino Pro Mini

Arduino Pro Mini - найкращий варіант для ваших проектів, в яких необхідна висока мобільність або кріплення контролера безпосередньо на рухомих вузлах вашого механізму. Повний гайд по використанню плат Arduino Pro Mini можна знайти в цій статті.

Arduino Pro Mini Додаткові пристрої для Arduino

Безумовно, Arduino сама по собі вже прекрасна. Але як окремий вузол вона на багато що не здатна. Щось треба до неї підключати. На просторах інтернету величезна кількість туторіали і проектів, з яких ви можете черпати ідеї для своїх проектів. У цій частині ми зробимо невеликий огляд датчиків і Шілд (плат розширень) для Arduino. Датчики (сенсори)

За допомогою коротенького коду і Arduino ви можете управляти найширшим спектром датчиків - сенсорів, які дозволяють вимірювати рівень освітленості, температуру, тиск, відстань, силу, вологість, радіоактивність, прискорення і багато іншого. На малюнку нижче наведено кілька з величезної кількості датчиків, сумісних з Arduino:

датчики Arduino Шілд (Shields) для Arduino

Крім усього іншого, є така чудова річ як Шілд - по суті це окрема схема живлення, яка має коннектори і сідає на вашу плату Arduino і забезпечує спрощене управління двигунами (Motor Шилд), підключення до інтернету (Ethernet Шилд), радіозв'язок, управління рідкокристалічними і сенсорними екранами і т.д.

2.2 Arduino IDE

2.3 Fritzing

Розділ 3

Програмування в середовищі Arduino IDE

3.1 Змінні

3.1.1 Типи даних

Комп'ютери та Arduino в тому числі, працюють з різними типами даних. В їх основі лежить арифметично-логічний пристрій (АЛП), що виконує арифметичні і логічні операції з клітинками пам'яті: $R1 + R2$, $R3 * R7$, $R4\&R5$ і т.д. Для АЛП немає різниці, який тип даних відображати користувачеві: текст, цілі числа, числа з плаваючою комою або навіть частина програмного коду.

Команди для цих операцій надходять від компілятора, а команди компілятору передаються від користувача. Саме програміст, визначає для компілятора, що це значення – ціле, а інше значення - число з плаваючою комою.

Характеристика основних типів даних в Arduino IDE Оболонка для програмування Arduino по суті являє з себе мову `C++` з підтримкою великої кількості бібліотек для полегшення процесу написання програм. `C++` пропонує широкий вибір різних типів даних.

Нижче представлений список основних типів даних, які використовуються в скетчах Arduino. Поруч з кожним типом даних вказано його розмір. Зверніть увагу, сто змінні типу `signed` дають можливість оперувати позитивними і від'ємними числами, а змінні типу `unsigned` допускають тільки роботу з додатними значеннями.

`boolean` (8 біт) – просте логічне `true / false`

`byte` (8 біт) – `unsigned` число в діапазоні від $0 - 255$

`char` (8 біт) – `signed` число в діапазоні від -128 до 127. У деяких випадках компілятор буде інтерпретувати цей тип даних як символ, що

може призвести до несподіваних результатів.

`unsigned char` (8 біт) – то ж що і `byte`; для ясності коду рекомендується замість цього типу даних використовувати `byte`.

`word` (16 біт) – `unsigned` число в діапазоні від 0 до 65535

`unsigned int` (16 біт) – те саме, що і `word`. Рекомендується замінювати типом даних `word` для скорочення коду і ясності

`int` (16 біт) – `signed` число в діапазоні від -32768 до 32767. Один з найпоширеніших типів даних, який дуже часто використовується для оголошення змінних в скетчах-прикладах для Arduino, вбудованих в Arduino IDE

`unsigned long` (32 біта) – `unsigned` число в діапазоні від $0-4,294,967,295$. Найчастіше цей тип даних використовується для зберігання результатів функції `millis()`, яка повертає кількість мілісекунд, протягом якого виконувалась дастина коду.

`long` (32 біта) – `signed` число в діапазоні від $-2,147,483,648$ до $2,147,483,647$

`float` (32 біта) – `signed` число в діапазоні від $-3.4028235 \cdot 10^{38}$ до $3.4028235 \cdot 10^{38}$. Числа з плаваючою комою не характерні для Arduino і компілятору доведеться довше опрацьовувати дані. Так що рекомендується за можливості їх уникати.

3.2 Структури

3.2.1 Базовий код для програмування план Arduino

Функція `setup()` викликається на початку скетчу. Вона використовується для ініціалізації змінних, настройки режимів роботи пінів (на введення або на виведення). Функція `setup()` виконується один раз після подачі живлення або перезавантаження плати Arduino.

Після виконання функції `setup()`, циклічно виконується функція `loop()`, яка безпосередньо є основою програми для управління платою Arduino.

Код, наведений нижче, не виконує ніяких завдань, але його структура корисна як база для всіх програм. Крім того, зверніть увагу на те, як залишаються коментарі в скетчах.

Кожен рядок, який починається з (`//`) не читатися компілятором, так що у ньому можна записувати будь-які дані.

Програмний код:

Скетч 3.1: Структура програми

```
int void() {
   for (int i=0; i<5; i++) {
      while (true) {
         docool();
      }
   }
// коментарі українською
}
```

3.2.2 Умовний оператор

Вираз `if()` є основним для всіх керуючих структур в програмуванні. Цей вираз дозволяє вам здійснювати чи ні певним чином впливати залежно від умови, що є `true` (виконується) або `false` (не виконується). Синтаксис умови `if` виглядає наступним чином:

Скетч 3.2: Умовний оператор (неповна форма)

```
if (someCondition) {
   // дії, які потрібно виконати, якщо умова виконується
}
```

Так само є подібні варіації структури з використанням if-else. Виглядає це наступним чином:

Скетч 3.3: Умовний оператор (повна форма)

```
if (someCondition) {
   // дії, які потрібно виконати, якщо умова виконується
}
else {
   // дії, які потрібно виконати, якщо умова не виконується
}
```

Також є структура else-if, за допомогою якої ви можете перевірити друга умова, якщо перше не виконується:

Скетч 3.4: Умовний оператор (декільки умов)

```
if (someCondition) {
   // дії, які потрібно виконати, якщо умова виконується
}
else if (anotherCondition) {
   // дії, які потрібно виконати тільки якщо не виконується пер
      ша умова
   // а друга умова виконується
```

```
}
```

Кількість розгалужень може бути необмеженою.

3.2.3 Оператор вибору

`switch`

3.2.4 Циклічні оператори

Циклічні оператори використовуються для повтоерння однакових або однотипних дій.

for Доцільно використовувати коли відома кількість поврень, які потрібно виконати.

Скетч 3.5: Оператор повторення for

```
for(int i=0; i<10; i++) {
    // всі оператори всередині фігурних дужок називаються тілом
        циклу
    Serial.println(i);
}
```

Перший аргумент – початкове значення лічильника. В даному випадку створюється внутрішня змінна `i`, якій надається початкового значення 0.

Другий аргумент – умова за якої тіло циклу виконується. Зазвичайумова вказується як заежність від лічильника. У прикладі тіло циклу виконуватиметься поки значення лічильника буде меншим за 10.

Третій аргумент – спосіб зміни лічильника. У прикладі лічильник з кожною ітерацією збільшується на 1. Таким чином, враховуючи умову, тіло циклу викорається 10 разів.

Кожен аргумент не є обов'язковим і може бути пропущений під час оголошення циклу. Але для коректної роботи потрібно конторювати всі складові окремо. Навдемо приклад тієї ж програми з порожніми агрументами.

Скетч 3.6: Оператор повторення for

```
int i = 0; // оголошення та ініціалізація лічильника за межами
    циклу
for(;;) {
    Serial.println(i);
    i++; // зміна значення лічильника
    if (i >= 10) break;
}
```

Для аналогічної роботи такої циклічної структури використовується оператор `break`, який потрібен для дострокового переривання виконання циклу.

Цикл while

Цикл do...while

3.2.5 Масиви

Масив – це набір однотипних змінних, доступ до яких здійснюється через їх індекс. У мові програмування C, на якому заснований Arduino, масиви можуть бути досить складними.

Створення (оголошення) масиву. Всі методики, представлені нижче, підходять для створення (оголошення) масиву.

Скетч 3.7: Оголошення масиву

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

Ви можете оголосити масив без його ініціалізації як `myInts`.

У `myPins` оголошується масив без безпосередньої вказівки розміру. Компілятор вважає кількість елементів і створює масив відповідного розміру.

Можна одночасно форматувати і вказати розмір вашого масиву, як, наприклад, це зроблено в `mySensVals`. Зверніть увагу, що при оголошенні масиву типу `char`, необхідний ще один елемент для зберігання обов'язкового `null`-символу.

Доступ до елементів масиву. Індексація в масивах починається з нуля. Тобто, перший елемент масиву буде мати порядковий номер `0`. Таким чином:

`mySensVals [0] == 2, mySensVals [1] == 4`, і так далі

Тобто, наприклад, в масиві з десяти елементів, останній елемент буде мати індекс дев'ять. наприклад:

`int myArray [10] = 9,3,2,4,3,2,7,8,9,11; myArray[9]` містить 11, а вираз `myArray[10]` некоректний і містить випадкові дані (іншу адресу пам'яті).

З цієї причини треба бути акуратним при доступі до масивів. Звертаючись до елементу масиву, значення якого більше ніж оголошений розмір

ви счітвиете з пам'яті значення, яке призначене для інших завдань. Зчитування таких даних просто призведе до некоректних результатів роботи програми в подальшому. Також не дуже хорошою ідеєю є запис даних в довільні місця. Крім того, відстежити цю помилку при перевірці коду теж складно.

На відміну від BASIC або JAVA, компілятор C не перевіряє, чи отримуємо ми доступ до елементу масиву в попередньо оголошених межах.

Для надання значення елементу масива під номером `0`:

```
mySensVals [0] = 10;
```

Для отримання значення елемента масиву за номером `4`:

```
x = mySensVals [4];
```

3.3 Функції

3.3.1 Цифрове введення/виведення

digitalRead()

Опис: Читає значення `HIGH` або `LOW` з визначеного цифорового піна.

Синтакс: `digitalRead(pin)`

Параметри: `pin` – номер цифрового піна (ціле число) з якого потрібно зчитати дані.

Результат виконання: значення `HIGH` або `LOW`, якщо є чи немає сигнал на вказаному піні відповідно.

Встановлює прочитане значення з піна 13 і зипис цього значення у пін 7.

Скетч 3.8: Використання функції digitalRead()

```
1  int ledPin = 13;    // Світлодіод приєднано до піна 13
2  int inPin = 7;      // Кнопка приєнана до цифорового піна 7
3  int val = 0;        // Змінна для збереження отриманого значення
4
5  void setup()
6  {
7    pinMode(ledPin, OUTPUT);       // Встановлення 13 піна для вив
       едення
8    pinMode(inPin, INPUT);         // Встановлення 7 піна для введ
       ення
9  }
10
11 void loop()
```

```
12 {
13   val = digitalRead(inPin);      // зчитуємо значення на 7 піні
14   digitalWrite(ledPin, val);     // встановлюємо значення для св
       ітлодіода рівним прочитаному занченню
15 }
```

Примітки Якщо пін не фізично приєднано то значення функції `digitalRead()` може набувати випадкових значеннь `HIGH` або `LOW`.

Аналогові вхідні контакти можна використовувати як цифрові, які називаються `A0`, `A1` та ін.

digitalWrite()

Опис: Записує значення `HIGH` або `LOW` до визначеного цифорового піна.

Якщо пін-код був налаштований як `OUTPUT` за допомогою функції `pinMode()`, його напруга буде встановленf на відповідне значення: 5В (або 3.3В на платах 3.3В) для значення `HIGH`, 0В (земля) для `LOW`.

Якщо пін налаштований як `INPUT`, digitalWrite() активуватиме (`HIGH`) або вимкне (`LOW`) внутрішню підтяжку на вхідному піні. Рекомендується встановити `pinMode()` у значення `(INPUT_PULLUP)` щоб увімкнути внутрішній підтягуючий резистор.

Якщо не встановилювати `pinMode()` в `OUTPUT` і підключили світлодіод до шпильки, під час виклику `digitalWrite(HIGH)` світлодіодний індикатор може виглядати затемненим. Без явного встановлення `pinMode()`, `digitalWrite()` буде активувати внутрішній підтягуючий резистор, який діє як обмежуючий резистор.

Синтакс: `digitalWrite(pin, value)`.

Параметри: `pin` – номер цифрового піна (ціле число) в який потрібно здійснити запис даних.

`value` – значення `HIGH` або `LOW`.

Результат виконання: `viod` (нічого).

Приклад використання Встановлення цифрового піна 13 в значення `OUTPUT` і перемикання значення з `HIGH` на `LOW` з інтервалом 1с = 1000мс.

Скетч 3.9: Використання функції digitalWrite()

```
1 void setup()
2 {
```

```
3   pinMode(13, OUTPUT);           // sets the digital pin 13 as
        output
4 }
5
6 void loop()
7 {
8   digitalWrite(13, HIGH);        // sets the digital pin 13 on
9   delay(1000);                   // waits for a second
10  digitalWrite(13, LOW);         // sets the digital pin 13 off
11  delay(1000);                   // waits for a second
12 }
```

Примітки Аналогові вхідні контакти можна використовувати як цифрові, які називаються `A0`, `A1` та ін.

pinMode()

Опис: Налаштовує вказаний пін для введення або введення.

Як і в Arduino 1.0.1, можна ввімкнути внутрішні підсилювачі резисторів за допомогою режиму `INPUT_PULLUP`. Крім того, режим `INPUT` явно вимикає внутрішні підказки.

Синтакс: `pinMode(pin, mode)`

Параметри: `pin` – номер піна, режим якого потрібно встановити. `mode`: `INPUT, OUTPUT` або `INPUT_PULLUP`.

Результат виконання: `viod` (нічого).

Приклад використання див. скетч `digitalwrite()`.

Примітки Аналогові вхідні контакти можна використовувати як цифрові, які називаються `A0`, `A1` та ін.

3.3.2 Аналогове введення/виведення

analogRead()

Опис: Читає значення з вказаного аналогового піна. Плата Arduino містить 6 каналів (8 каналів на Mini і Nano, 16 на Mega), 10-бітний аналого-цифровий перетворювач. Це означає, що він буде перетворювати вхідну напругу від 0 до 5 вольт до цілих значень між 0 і 1023. Це дає роздільну здатність між показниками: 5 вольт / 1024 одиниці або 0.0049 вольт (4,9

мВ) на одиницю. Діапазон введення та роздільну здатність можна змінити за допомогою `analogReference()`.

Для читання аналогового входу потрібно близько 100 мікросекунд (0,0001с), тому максимальна швидкість читання становить близько 10000 разів у секунду.

Синтакс: `analogRead(pin)`

Параметри: `pin` – номер аналогового вхідного піна для читання. (від 0 до 5 на більшості плат, від 0 до 7 на Mini і Nano, від 0 до 15 на Mega)

Результат виконання: ціле число `int` (від 0 до 1023).

Приклад використання Зчитуванння і відображення напруги на обраному піні.

Скетч 3.10: Використання функції digitalWrite()

```
int analogPin = 3;      // середня ножка потенціометра приєдана до піна 3
                        // інші до +5В і землі 0.
int val = 0;            // змінна для збереження прочитаного значення

void setup()
{
  Serial.begin(9600);               //  встановлення серійного порта
}

void loop()
{
  val = analogRead(analogPin);      // зчитування даних з піна
  Serial.println(val);              // виведення значення
}
```

Примітки. Якщо аналоговий вхідний пін не під'єднано, значення, яке повертає analogRead(), коливається залежно від ряду факторів (наприклад, значень інших аналогових входів, наближення вашої руки до плати тощо).

analogReference()

Опис: Налаштовує опорну напругу, що використовується для аналогового вводу (тобто значення, яке використовується як верхня частина діапазону вхідних даних). Варіанти:

Плати Arduino AVR (Uno, Mega та ін.) `DEFAULT`: аналоговий номер за замовчуванням 5 В (на платах Arduino 5V) або 3,3 В (на платах Arduino на 3,3 В)

`INTERNAL`: вбудований довідник, що дорівнює 1.1 вольтам ATmega168 або ATmega328P та 2.56 вольтам ATmega8 (недоступний на Arduino Mega)

`INTERNAL1V1`: вбудований довідник 1.1V (тільки для Arduino Mega)

`INTERNAL2V56`: вбудований довідник 2.56V (тільки для Arduino Mega)

`EXTERNAL`: напруга, накладена на штифт AREF (лише від 0 до 5В), використовується як еталон.

Arduino SAMD Boards (Zero та ін.)

`AR_DEFAULT`: стандартна аналогова довідка 3,3 В

`AR_INTERNAL`: вбудований довідник 2,23 В

`AR_INTERNAL1V0`: вбудований довідник 1.0V

`AR_INTERNAL1V65`: вбудований довідник 1,65 Вт

`AR_INTERNAL2V23`: вбудований довідник 2,23 В

`AR_EXTERNAL`: напруга, застосована до PIN-коду AREF, використовується як еталонна

Arduino SAM Boards (Due)

`AR_DEFAULT`: стандартна аналогова довідка 3.3В. Це єдиний підтримуваний варіант для Due.

Синтакс: `analogReference(type)`

Параметри: `type` – тип посилання на використання.

Результат виконання: `viod` (нічого).

Примітки. Після зміни аналогового посилання, перші декілька показань з `analogRead()` можуть бути неточними.

Не використовуйте нічого менше 0В або більше 5В для зовнішньої опорної напруги на піні AREF! Якщо ви використовуєте зовнішнє посилання на пін AREF, перед виконанням `analogRead()` слід встановити аналогове посилання на `EXTERNAL`. В іншому випадку, ви закоротите активну опорну напругу (внутрішньо сформовану) та пін AREF, що може пошкодити мікроконтроллер на платі Arduino.

Крім того, ви можете підключити зовнішню опорну напругу до піна AREF через резистор 5K, що дозволяє перемикати між зовнішніми та внутрішніми еталонними напругами. Зверніть увагу, що резистор змінить напругу, яка використовується в якості еталона, оскільки на вході AREF є внутрішній 32K резистор. Два виступають в якості дільника напруги, так

що, наприклад, 2.5В, застосованого через резистор, дасть $\frac{2.5 \cdot 32}{32+5} \approx 2.2$В на піні AREF.

analogWrite()

Опис: Записує аналогове значення (ШІМ) у обраний пін. Може бути використана для зміни яскравості світіння світлодіодів або для керування двигуном з різною швидкістю. Після виклику `analogWrite()`, контакт буде генерувати стійку квадратну хвилю заданого робочого циклу до наступного виклику `analogWrite()` (або виклику `digitalRead()` або `digitalWrite()`) на одному піні. Частота сигналу ШІМ на більшості пінів становить приблизно 490 Гц. На Uno та аналогічних платах контакти 5 та 6 мають частоту приблизно 980 Гц.

На більшості плат Arduino (на базі мікропроцесорів ATmega168 або ATmega328P) ця функція працює на контактах 3, 5, 6, 9, 10 і 11. На Arduino Mega він працює на контактах 2-13 і 44-46. Старіші версії Arduino з ATmega8 підтримують `analogWrite()` тільки на пінах 9, 10 і 11. Arduino DUE підтримує `analogWrite()` на контактах від 2 до 13, а також контактів `XDAC0` та `DAC1`. На відміну від шпильки PWM, `DAC0` та `DAC1` є цифровими аналоговими перетворювачами і служать справжніми аналоговими виходами. Не потрібно викликати `pinMode()`, щоб встановити PIN-код як вихідний сигнал перед назвою `analogWrite()`. Функція `analogWrite` не має нічого спільного з аналоговими пінами або функцією аналогового читання.

Синтаксис: `analogWrite(pin, value)`.

Параметри: `pin` – пін для запису. Дозволений тип даних: `int`.

`value` – значення від 0 (завжди вимкнено) і 255 (завжди увімкнуто). Дозволені типи даних: `int`.

Результат виконання: `viod` (нічого).

Приклад використання Встановлює вихідний сигнал на світлодіод, пропорційний значенню, прочитаному з потенціометра.

Скетч 3.11: analogWrite

```
int ledPin = 9;       // Світлодіод приєднаний до піна 9
int analogPin = 3;    // потенціометри приєднано до піра 9
int val = 0;          // змінна для зберігання прочитаного значе
    ння

void setup()
{
```

```
7   pinMode(ledPin, OUTPUT);    // встановити пін у режим виведенн
        я
8 }
9
10 void loop()
11 {
12   val = analogRead(analogPin);    // читаємо значення з потенціо
        метра
13   analogWrite(ledPin, val / 4);  // значення прочитане
        analogRead знаходиться в межах від 0 до 1023, analogWrite
        дозволяє записувати значення від 0 до 255
14 }
```

Примітки. Виходи ШІМ, що генеруються на пінах 5 та 6, матимуть більше робочих циклів. Це пов'язано з взаємодією з функціями `millis()` і `delay()`, які мають такий же внутрішній таймер, який використовується для генерації цих результатів ШІМ. Це буде помічено в основному в налаштуваннях низького циклу робочого циклу (наприклад, 0-10), і це може призвести до значення 0, яке не повністю відключає вихід на пінах 5 і 6.

3.3.3 Додаткове введення/виведення

noTone()

Опис: Зупинка генерації квадратної хвилі. Не працює, якщо не попередньо не генерувався тональний сигнал.

Синтаксис: `noTone(pin)`.

Параметри: `pin`: – пін, на якому зупиняється генерування тону

Результат виконання: `viod` (нічого).

Примітки. Якщо ви хочете генерувати різні звуки на декількох пінах, вам потрібно зупинити на одному піні перед тим, як викликати `tone()` на наступному піні.

pulseIn()

Опис: Читає імпульс (`HIGH` або LOW) на піні. Наприклад, якщо значення `HIGH`, то `pulseIn()` чекає, що пін буде переходити від `LOW` до `HIGH`, потім чекає, що пін буде мати значення `LOW` і зупиниться синхронізація. Повертає довжину імпульсу в мікросекундах або скасовується і повертає 0, якщо за певний час не було отримано повного імпульсу.

Терміни цієї функції були визначені емпірично і, ймовірно, відображатимуть помилки у більш довгих імпульсах. Працює з імпульсами від 10 мкс до 3 хвилин.

Синтакс: `pulseIn(pin, value)`
`pulseIn(pin, value, timeout)`

Параметри: `pin` – номер піна, на якому потрібно прочитати імпульс. Тип даних `int`

`value` – тип імпульсу для читання: або `HIGH`, або `LOW`.

`timeout` (необов'язково): кількість мікросекунд, що очікує початку пульсу; за замовчуванням - одна секунда. Тип даних `unsigned long`.

Результат виконання: довжина імпульсу (в мікросекундах) або 0, якщо пульс не стартував до тайм-ауту. Тип даних `unsigned long`.

Приклад використання У прикладі розраховано тривалість часу імпульсу на піні 7.

Скетч 3.12: pulseIn

```
int pin = 7;
unsigned long duration;
void setup()
{
   pinMode(pin, INPUT);
}
void loop()
{
   duration = pulseIn(pin, HIGH);
}
```

pulseInLong()

Опис: pulseInLong () є альтернативою pulseIn (), що краще при обробці довгих імпульсів і переривань, що зачіпають сценарії.

Читає імпульс (HIGH або LOW) на шпильці. Наприклад, якщо значення HIGH, pulseInLong () чекає, що штифт буде переходити від LOW до HIGH, починає терміни, а потім чекає, що штифт буде LOW і зупиниться. Повертає довжину імпульсу в мікросекундах або відмовляється і повертає 0, якщо в тайм-аут не було отримано повного імпульсу.

Час виконання цієї функції був визначений емпіричним способом і, ймовірно, відображатиме помилки в коротших імпульсах. Працює з імпульсами від 10 мкс до 3 хвилин у довжину. Ця процедура може вико-

ристовуватися лише у тому випадку, якщо активовані переривання. Крім того, найвищу роздільну здатність отримують з великими інтервалами.

Синтакс: `pulseInLong(pin, value)`
`pulseInLong(pin, value, timeout)`

Параметри: `pin` – номер піна, на якому ви хочете прочитати імпульс. Тип даних `int`.

`value` – тип імпульсу для читання: або `HIGH`, або `LOW`. Тип даних `int`.

`timeout` (необов'язковий праметр) – кількість мікросекунд, що очікує початку пульсу; за замовчуванням одна секунда. Тип даних `unsigned long`.

Результат виконання: довжина імпульсу (в мікросекундах) або 0, якщо пульс не стартував до тайм-ауту. Тип даних `unsigned long`.

Приклад використання У прикладі розраховано тривалість часу імпульсу на піні 7.

Скетч 3.13: pulseInLong

```
int pin = 7;
unsigned long duration;

void setup() {
  pinMode(pin, INPUT);
}

void loop() {
  duration = pulseInLong(pin, HIGH);
}
```

Примітки. Ця функція спирається на `micros()`, тому її не можна використовувати в контексті `noInterrupts()`.

shiftIn()

Опис: Зміщення в байті даних один біт одночасно. Починається з найбільшого (тобто найменшого) або найменшого (найменшого) значного біта. Для кожного біта годинник штифт витягується високо, наступний біт читається з лінії передачі даних, а потім годинник шпилька знімається низьким.

Якщо ви взаємодієте з пристроєм, який синхронізується за зростаючими краями, перед першим викликом `shiftIn()`, наприклад, зателефонувавши до цифрового запису `(clockPin, LOW)`.

Примітка: це програмне забезпечення; Arduino також забезпечує бібліотеку `SPI`, яка використовує апаратну реалізацію, яка швидше працює, але працює лише на конкретних пінах.

Синтаксис: `byte incoming = shiftIn(dataPin, clockPin, bitOrder)`

Параметри: `dataPin` – пін, на який вводиться кожен біт. Тип даних `int`.

`clockPin` – пін для перемикання, щоб сигналізувати читання від `dataPin`

`bitOrder` – який порядок переміщення в біти: `MSBFIRST` або `LSBFIRST`.

Результат виконання: Прочитане значення. Тип даних `byte`.

shiftOut()

Description

Shifts out a byte of data one bit at a time. Starts from either the most (i.e. the leftmost) or least (rightmost) significant bit. Each bit is written in turn to a data pin, after which a clock pin is pulsed (taken high, then low) to indicate that the bit is available.

Note- if you're interfacing with a device that's clocked by rising edges, you'll need to make sure that the clock pin is low before the call to shiftOut(), e.g. with a call to digitalWrite(clockPin, LOW).

This is a software implementation; see also the SPI library, which provides a hardware implementation that is faster but works only on specific pins. Syntax

shiftOut(dataPin, clockPin, bitOrder, value) Parameters

dataPin: the pin on which to output each bit (int)

clockPin: the pin to toggle once the dataPin has been set to the correct value (int)

bitOrder: which order to shift out the bits; either MSBFIRST or LSBFIRST. (Most Significant Bit First, or, Least Significant Bit First)

value: the data to shift out. (byte) Returns

Nothing Example Code

For accompanying circuit, see the tutorial on controlling a 74HC595 shift register.

Скетч 3.14: shiftOut

```
1 //Pin connected to ST_CP of 74HC595
2 int latchPin = 8;
```

```
3  //Pin connected to SH_CP of 74HC595
4  int clockPin = 12;
5  ////Pin connected to DS of 74HC595
6  int dataPin = 11;
7
8  void setup() {
9    //set pins to output because they are addressed in the main
       loop
10   pinMode(latchPin, OUTPUT);
11   pinMode(clockPin, OUTPUT);
12   pinMode(dataPin, OUTPUT);
13 }
14
15 void loop() {
16   //count up routine
17   for (int j = 0; j < 256; j++) {
18     //ground latchPin and hold low for as long as you are
         transmitting
19     digitalWrite(latchPin, LOW);
20     shiftOut(dataPin, clockPin, LSBFIRST, j);
21     //return the latch pin high to signal chip that it
22     //no longer needs to listen for information
23     digitalWrite(latchPin, HIGH);
24     delay(1000);
25   }
26 }
```

Notes and Warnings

The dataPin and clockPin must already be configured as outputs by a call to pinMode().

shiftOut is currently written to output 1 byte (8 bits) so it requires a two step operation to output values larger than 255.

// Do this for MSBFIRST serial int data = 500; // shift out highbyte shiftOut(dataPin, clock, MSBFIRST, (data » 8)); // shift out lowbyte shiftOut(dataPin, clock, MSBFIRST, data);

// Or do this for LSBFIRST serial data = 500; // shift out lowbyte shiftOut(dataPin, clock, LSBFIRST, data); // shift out highbyte shiftOut(dataPin, clock, LSBFIRST, (data » 8));

tone()

Description

Generates a square wave of the specified frequency (and 50

Only one tone can be generated at a time. If a tone is already playing on a different pin, the call to tone() will have no effect. If the tone is playing on the same pin, the call will set its frequency.

Use of the tone() function will interfere with PWM output on pins 3 and 11 (on boards other than the Mega).

It is not possible to generate tones lower than 31Hz. For technical details, see Brett Hagman's notes. Syntax

tone(pin, frequency)

tone(pin, frequency, duration) Parameters

pin: the pin on which to generate the tone

frequency: the frequency of the tone in hertz - unsigned int

duration: the duration of the tone in milliseconds (optional) - unsigned long Returns

Nothing Notes and Warnings

If you want to play different pitches on multiple pins, you need to call noTone() on one pin before calling tone() on the next pin.

3.3.4 Робота з часом

delay()

Description

Pauses the program for the amount of time (in milliseconds) specified as parameter. (There are 1000 milliseconds in a second.) Syntax

delay(ms) Parameters

ms: the number of milliseconds to pause (unsigned long) Returns

Nothing Example Code

The code pauses the program for one second before toggling the output pin.

Скетч 3.15: delay

```
int ledPin = 13;                  // LED connected to digital
    pin 13

void setup()
{
    pinMode(ledPin, OUTPUT);      // sets the digital pin as
        output
}

void loop()
{
    digitalWrite(ledPin, HIGH);   // sets the LED on
    delay(1000);                  // waits for a second
    digitalWrite(ledPin, LOW);    // sets the LED off
    delay(1000);                  // waits for a second
}
```

Notes and Warnings

While it is easy to create a blinking LED with the delay() function, and many sketches use short delays for such tasks as switch debouncing, the use of delay() in a sketch has significant drawbacks. No other reading of sensors, mathematical calculations, or pin manipulation can go on during the delay function, so in effect, it brings most other activity to a halt. For alternative approaches to controlling timing see the millis() function and the sketch sited below. More knowledgeable programmers usually avoid the use of delay() for timing of events longer than 10's of milliseconds unless the Arduino sketch is very simple.

Certain things do go on while the delay() function is controlling the Atmega chip however, because the delay function does not disable interrupts. Serial communication that appears at the RX pin is recorded, PWM (analogWrite) values and pin states are maintained, and interrupts will work as they should.

delayMicroseconds()

Description

Pauses the program for the amount of time (in microseconds) specified as parameter. There are a thousand microseconds in a millisecond, and a million microseconds in a second.

Currently, the largest value that will produce an accurate delay is 16383. This could change in future Arduino releases. For delays longer than a few thousand microseconds, you should use delay() instead. Syntax

delayMicroseconds(us) Parameters

us: the number of microseconds to pause (unsigned int) Returns

Nothing Example Code

The code configures pin number 8 to work as an output pin. It sends a train of pulses of approximately 100 microseconds period. The approximation is due to execution of the other instructions in the code.

Скетч 3.16: delayMicroseconds

```
1  int outPin = 8;                 // digital pin 8
2
3  void setup()
4  {
5     pinMode(outPin, OUTPUT);      // sets the digital pin as
          output
6  }
7
8  void loop()
9  {
10    digitalWrite(outPin, HIGH);   // sets the pin on
11    delayMicroseconds(50);        // pauses for 50 microseconds
```

```
12    digitalWrite(outPin, LOW);     // sets the pin off
13    delayMicroseconds(50);         // pauses for 50 microseconds
14 }
```

Notes and Warnings

This function works very accurately in the range 3 microseconds and up. We cannot assure that delayMicroseconds will perform precisely for smaller delay-times.

As of Arduino 0018, delayMicroseconds() no longer disables interrupts.

micros()

Description

Returns the number of microseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 70 minutes. On 16 MHz Arduino boards (e.g. Duemilanove and Nano), this function has a resolution of four microseconds (i.e. the value returned is always a multiple of four). On 8 MHz Arduino boards (e.g. the LilyPad), this function has a resolution of eight microseconds. Syntax

time = micros() Parameters

Nothing Returns

Returns the number of microseconds since the Arduino board began running the current program.(unsigned long) Example Code

The code returns the number of microseconds since the Arduino board began.

Скетч 3.17: micros

```
1  unsigned long time;
2
3  void setup(){
4     Serial.begin(9600);
5  }
6  void loop(){
7     Serial.print("Time: ");
8     time = micros();
9
10    Serial.println(time);  //prints time since program started
11    delay(1000);           // wait a second so as not to send
          massive amounts of data
12 }
```

Notes and Warnings

There are 1000 microseconds in a millisecond and 1,000,000 microseconds in a second.

millis()

Description

Returns the number of milliseconds since the Arduino board began running the current program. This number will overflow (go back to zero), after approximately 50 days. Syntax

time = millis() Parameters

Nothing Returns

Number of milliseconds since the program started (unsigned long) Example Code

The code reads the milllisecond since the Arduino board began.

Скетч 3.18: milis

```
unsigned long time;

void setup(){
    Serial.begin(9600);
}
void loop(){
    Serial.print("Time: ");
    time = millis();

    Serial.println(time);     //prints time since program started
    delay(1000);              // wait a second so as not to send
        massive amounts of data
}
```

Notes and Warnings

Please note that the return value for millis() is an unsigned long, logic errors may occur if a programmer tries to do arithmetic with smaller data types such as int's. Even signed long may encounter errors as its maximum value is half that of its unsigned counterpart.

3.3.5 Математичні функції

abs()

Description

Calculates the absolute value of a number. Syntax

abs(x) Parameters

x: the number Returns

x: if x is greater than or equal to 0.

-x: if x is less than 0. Notes and Warnings

Because of the way the abs() function is implemented, avoid using other functions inside the brackets, it may lead to incorrect results.

abs(a++); // avoid this - yields incorrect results

abs(a); // use this instead - a++; // keep other math outside the function

constrain()

Description

Constrains a number to be within a range. Syntax

constrain(x, a, b) Parameters

x: the number to constrain, all data types a: the lower end of the range, all data types b: the upper end of the range, all data types Returns

x: if x is between a and b

a: if x is less than a

b: if x is greater than b Example Code

The code limits the sensor values to between 10 to 150.

sensVal = constrain(sensVal, 10, 150); // limits range of sensor values to between 10 and 150

map()

Description

Re-maps a number from one range to another. That is, a value of fromLow would get mapped to toLow, a value of fromHigh to toHigh, values in-between to values in-between, etc.

Does not constrain values to within the range, because out-of-range values are sometimes intended and useful. The constrain() function may be used either before or after this function, if limits to the ranges are desired.

Note that the "lower bounds"of either range may be larger or smaller than the "upper bounds"so the map() function may be used to reverse a range of numbers, for example

y = map(x, 1, 50, 50, 1);

The function also handles negative numbers well, so that this example

y = map(x, 1, 50, 50, -100);

is also valid and works well.

The map() function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated, and are not rounded or averaged. Syntax

map(value, fromLow, fromHigh, toLow, toHigh) Parameters

value: the number to map

fromLow: the lower bound of the value's current range

fromHigh: the upper bound of the value's current range

toLow: the lower bound of the value's target range

toHigh: the upper bound of the value's target range Returns

The mapped value. Example Code

/* Map an analog value to 8 bits (0 to 255) */ void setup()

void loop() int val = analogRead(0); val = map(val, 0, 1023, 0, 255); analogWrite(9, val);

Appendix

For the mathematically inclined, here's the whole function

Скетч 3.19: ”

```
long map(long x, long in_min, long in_max, long out_min, long
    out_max)
{
  return (x - in_min) * (out_max - out_min) / (in_max - in_min)
      + out_min;
}
```

max()

Description

Calculates the maximum of two numbers. Syntax

max(x, y) Parameters

x: the first number, any data type y: the second number, any data type Returns

The larger of the two parameter values. Example Code

The code ensures that sensVal is at least 20.

sensVal = max(sensVal, 20); // assigns sensVal to the larger of sensVal or 20 // (effectively ensuring that it is at least 20)

Notes and Warnings

Perhaps counter-intuitively, max() is often used to constrain the lower end of a variable's range, while min() is used to constrain the upper end of the range.

Because of the way the max() function is implemented, avoid using other functions inside the brackets, it may lead to incorrect results

max(a–, 0); // avoid this - yields incorrect results

max(a, 0); // use this instead - a–; // keep other math outside the function

min()

Description

Calculates the minimum of two numbers. Syntax

min(x, y) Parameters

x: the first number, any data type

y: the second number, any data type Returns

The smaller of the two numbers. Example Code

The code ensures that it never gets above 100.

sensVal = min(sensVal, 100); // assigns sensVal to the smaller of sensVal or 100 // ensuring that it never gets above 100.

Notes and Warnings

Perhaps counter-intuitively, max() is often used to constrain the lower end of a variable's range, while min() is used to constrain the upper end of the range.

Because of the way the min() function is implemented, avoid using other functions inside the brackets, it may lead to incorrect results

min(a++, 100); // avoid this - yields incorrect results

min(a, 100); a++; // use this instead - keep other math outside the function

pow()

Description

Calculates the value of a number raised to a power. Pow() can be used to raise a number to a fractional power. This is useful for generating exponential mapping of values or curves. Syntax

pow(base, exponent) Parameters

base: the number (float)

exponent: the power to which the base is raised (float) Returns

The result of the exponentiation. (double) Example Code

See the (fscale) function in the code library.

sq()

Description

Calculates the square of a number: the number multiplied by itself. Syntax

sq(x) Parameters

x: the number, any data type Returns

The square of the number. (double)

sqrt()

Calculates the square root of a number. Description Syntax

sqrt(x) Parameters

x: the number, any data type Returns

The number's square root. (double)

trigonometry

cos, sin, tan

Description

Calculates the cosine of an angle (in radians). The result will be between -1 and 1. Syntax

cos(rad) Parameters

rad: The angle in Radians (float). Returns

The cos of the angle (double).

3.3.6 Псевдовипадкові числа

random()

Description

The random function generates pseudo-random numbers. Syntax

random(max) random(min, max) Parameters

min - lower bound of the random value, inclusive (optional)

max - upper bound of the random value, exclusive Returns

A random number between min and max-1 (long) . Example Code

The code generates random numbers and displays them.

Скетч 3.20: random

```
long randNumber;

void setup(){
   Serial.begin(9600);

   // if analog input pin 0 is unconnected, random analog
   // noise will cause the call to randomSeed() to generate
   // different seed numbers each time the sketch runs.
   // randomSeed() will then shuffle the random function.
   randomSeed(analogRead(0));
}

void loop() {
   // print a random number from 0 to 299
   randNumber = random(300);
   Serial.println(randNumber);

   // print a random number from 10 to 19
   randNumber = random(10, 20);
   Serial.println(randNumber);

   delay(50);
}
```

Notes and Warnings

If it is important for a sequence of values generated by random() to differ, on subsequent executions of a sketch, use randomSeed() to initialize the random

number generator with a fairly random input, such as analogRead() on an unconnected pin.

Conversely, it can occasionally be useful to use pseudo-random sequences that repeat exactly. This can be accomplished by calling randomSeed() with a fixed number, before starting the random sequence.

The max parameter should be chosen according to the data type of the variable in which the value is stored. In any case, the absolute maximum is bound to the long nature of the value generated (32 bit - 2,147,483,647). Setting max to a higher value won't generate an error during compilation, but during sketch execution the numbers generated will not be as expected.

randomSeed()

Description

randomSeed() initializes the pseudo-random number generator, causing it to start at an arbitrary point in its random sequence. This sequence, while very long, and random, is always the same.

If it is important for a sequence of values generated by random() to differ, on subsequent executions of a sketch, use randomSeed() to initialize the random number generator with a fairly random input, such as analogRead() on an unconnected pin.

Conversely, it can occasionally be useful to use pseudo-random sequences that repeat exactly. This can be accomplished by calling randomSeed() with a fixed number, before starting the random sequence. Parameters

seed - number to initialize the pseudo-random sequence (unsigned long). Returns

Nothing Example Code

The code explanation required.

Скетч 3.21: ”

```
long randNumber;

void setup(){
   Serial.begin(9600);
   randomSeed(analogRead(0));
}

void loop(){
   randNumber = random(300);
   Serial.println(randNumber);

   delay(50);
}
```

3.3.7 Функції комунікації

Serial

Опис

Використовується для зв'язку між платою Arduino та комп'ютером або іншими пристроями. Всі плати Arduino мають щонайменше один послідовний порт (також відомий як UART або USART): серійний. Він підтримує зв'язок на цифрових контактах 0 (RX) і 1 (TX), а також з комп'ютером через USB. Таким чином, якщо ви користуєтеся цими функціями, ви також не можете використовувати штифти 0 і 1 для цифрового введення або виведення. Ви можете використовувати вбудований серійний монітор середовища Arduino для спілкування з платою Arduino. Натисніть кнопку серійного монітора на панелі інструментів та оберіть таку ж швидкість передачі, який використовується для дзвінка, щоб розпочати ().

Послідовний зв'язок на штирях TX / RX використовує рівні логіки TTL (5 В або 3.3 В залежно від плати). Не підключайте ці контакти безпосередньо до послідовного порту RS232; вони працюють на +/- 12 В і можуть пошкодити плату Arduino.

Arduino Mega має три додаткових послідовних порту: Serial1 на штифтах 19 (RX) і 18 (TX), Serial2 на штифтах 17 (RX) і 16 (TX), Serial3 на штифтах 15 (RX) і 14 (TX). Щоб використовувати ці штифти для спілкування з вашим персональним комп'ютером, вам знадобиться додатковий USB-послідовний адаптер, так як він не підключений до USB-послідовного адаптера Mega. Щоб використовувати їх для зв'язку з зовнішнім TTL-послідовним пристроєм, підключіть TX-штифт до шпильки RX вашого пристрою, RX до шпильки TX-пристрою вашого пристрою та підставу мега на землю вашого пристрою.

Arduino DUE має три додаткових 3.3V TTL послідовних портів: Serial1 на штифтах 19 (RX) і 18 (TX); Serial2 на штифтах 17 (RX) і 16 (TX), Serial3 на штифтах 15 (RX) і 14 (TX). Шпильки 0 і 1 також підключені до відповідних штифтів послідовного чіпа USB-to-TTL ATmega16U2, підключеного до порту налагодження USB. Крім того, на своєму чіпі SAM3X є власний USB-послідовний порт, SerialUSB '.

Плата Arduino Leonardo використовує Serial1 для зв'язку через TTL (5V) серійний на контакти 0 (RX) і 1 (TX). Серійний номер зарезервований для зв'язку USB CDC. Для отримання додаткової інформації зверніться до сторінки "Леонардо"про початок роботи та сторінку обладнання.

stream

Description

Stream is the base class for character and binary based streams. It is not called directly, but invoked whenever you use a function that relies on it.

Stream defines the reading functions in Arduino. When using any core functionality that uses a read() or similar method, you can safely assume it calls on the Stream class. For functions like print(), Stream inherits from the Print class.

Some of the libraries that rely on Stream include :

Serial

Wire

Ethernet

SD

Keyboard

Description

The keyboard functions enable 32u4 or SAMD micro based boards to send keystrokes to an attached computer through their micro's native USB port.

Note: Not every possible ASCII character, particularly the non-printing ones, can be sent with the Keyboard library. The library supports the use of modifier keys. Modifier keys change the behavior of another key when pressed simultaneously. See here for additional information on supported keys and their use. Notes and Warnings

These core libraries allow the 32u4 and SAMD based boards (Leonardo, Esplora, Zero, Due and MKR Family) to appear as a native Mouse and/or Keyboard to a connected computer.

A word of caution on using the Mouse and Keyboard libraries: if the Mouse or Keyboard library is constantly running, it will be difficult to program your board. Functions such as Mouse.move() and Keyboard.print() will move your cursor or send keystrokes to a connected computer and should only be called when you are ready to handle them. It is recommended to use a control system to turn this functionality on, like a physical switch or only responding to specific input you can control.

When using the Mouse or Keyboard library, it may be best to test your output first using Serial.print(). This way, you can be sure you know what values are being reported. Refer to the Mouse and Keyboard examples for some ways to handle this. Functions

Keyboard.begin() Keyboard.end() Keyboard.press() Keyboard.print() Keyboard.println() Keyboard.release() Keyboard.releaseAll() Keyboard.write()

Mouse

Description

The mouse functions enable 32u4 or SAMD micro based boards to control cursor movement on a connected computer through their micro's native USB port. When updating the cursor position, it is always relative to the cursor's previous location. Notes and Warnings

These core libraries allow the 32u4 and SAMD based boards (Leonardo, Esplora, Zero, Due and MKR Family) to appear as a native Mouse and/or Keyboard to a connected computer.

A word of caution on using the Mouse and Keyboard libraries: if the Mouse or Keyboard library is constantly running, it will be difficult to program your board. Functions such as Mouse.move() and Keyboard.print() will move your cursor or send keystrokes to a connected computer and should only be called when you are ready to handle them. It is recommended to use a control system to turn this functionality on, like a physical switch or only responding to specific input you can control.

When using the Mouse or Keyboard library, it may be best to test your output first using Serial.print(). This way, you can be sure you know what values are being reported. Refer to the Mouse and Keyboard examples for some ways to handle this. Functions

Mouse.begin() Mouse.click() Mouse.end() Mouse.move() Mouse.press() Mouse.release() Mouse.isPressed()

Опис:

Синтакс:

Параметри:

Результат виконання:

Приклад використання

Примітки.

Розділ 4

Лабораторно-практичні роботи

4.1 ЛПР№1 [Світлодіод]

Необхідні компоненти:

- плата Arduino Uno;
- світлодіод;
- резистор на 220 Ом;
- з'єднувальні дроти.

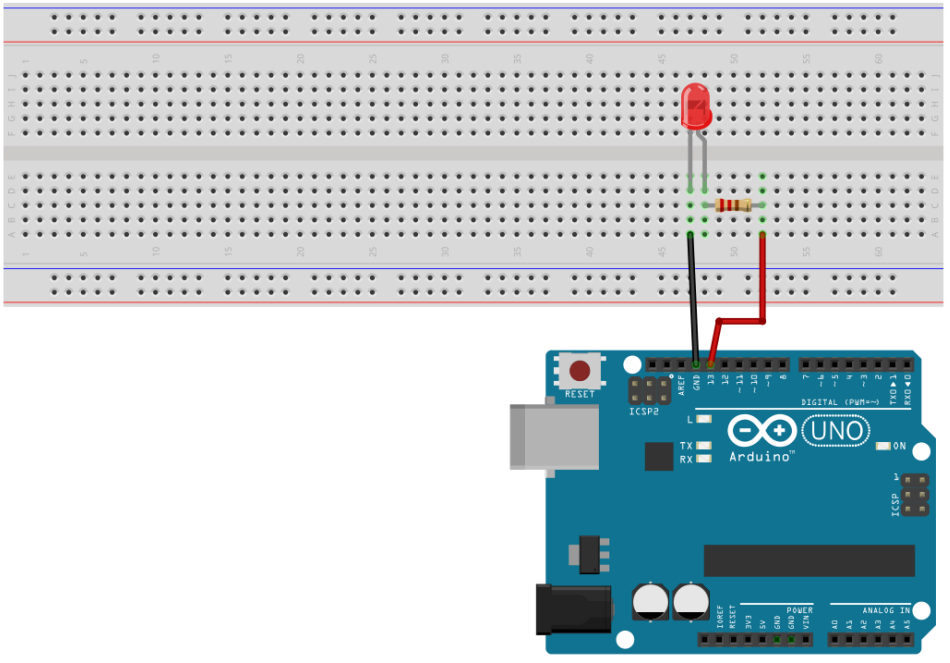


Рис. 4.1: Розташування елементів

Скетч 4.1: Програмний код

```
int LED = 13;

void setup() {
   pinMode(LED, OUTPUT);
}
```

```
6
7  void loop() {
8      digitalWrite(LED, HIGH);
9      delay(1000);
10     digitalWrite(LED, LOW);
11     delay(1000);
12 }
```

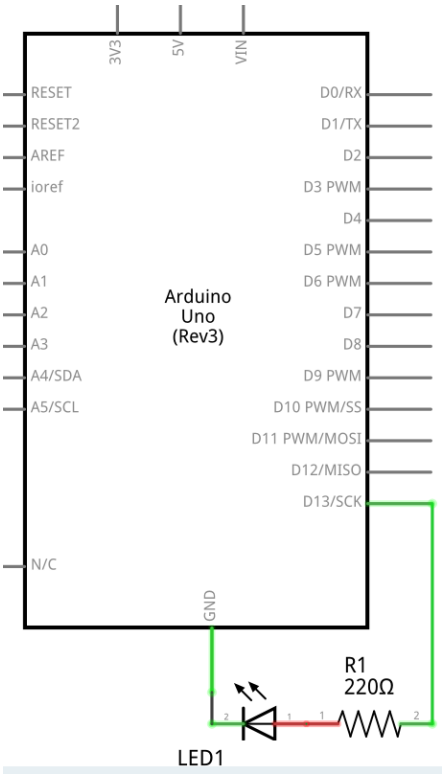


Рис. 4.2: Схема підключення

4.2 ЛПР№2 [Сигналізація]