Міністерство освіти і науки України
Сумський державний університет

Тищенко К. В., Ткач О. П.

ПРОГРАМУВАННЯ СИСТЕМ ЗБОРУ ТА АНАЛІЗУ ДАНИХ

Навчальний посібник

Рекомендовано вченою радою Сумського державного університету

Суми
Сумський державний університет
2023

УДК 004.4:681.5(075.8)
Т 47

Рецензенти:
С. О. Лебединський – кандидат фізико-математичних наук, науковий співробітник відділу квантової електродинаміки сильних полів Інституту прикладної фізики НАН України;
Ю. О. Шкурдода – доктор фізико-математичних наук, професор, доцент кафедри електроніки, загальної та прикладної фізики Сумського державного університету

Рекомендовано до видання
вченою радою Сумського державного університету
як навчальний посібник
(протокол № 15 від 29 червня 2023 року)

Тищенко К. В.
Т 47 Програмування систем збору та аналізу даних /
К. В. Тищенко, О. П. Ткач. – Суми : Сумський державний університет, 2023. – 189 с.

У навчальному посібнику подано навчальні матеріали з дисципліни «Програмування систем збору і аналізу даних».

Призначений для студентів закладів вищої освіти спеціальності «Електроніка».

УДК 004.4:681.5(075.8)

ЗМІСТ

С.

Вступ

Цей навчальний посібник допоможе здобувачам освіти розібратися в принципах і техніках програмування систем збирання та аналізування даних.

Перший розділ присвячений розробленню та програмуванню систем збирання даних, аналізуванню та використанню результатів для вирішення реальних проблем. Тут розглянуто мікроконтролерні платформи, їх характеристики та особливості, використовувані для розроблення технічних рішень збирання й аналізування даних. Навчальний посібник містить відомості про програмування мікроконтролерів Arduino мовою C++, що допоможуть читачам розробляти ефективні програмні рішення. Здобувачі освіти будуть навчатися розробляти системи збирання даних, що вміщують створення апаратної складової та програмного забезпечення для оброблення інформації, одержаної із зовнішнього середовища, та її коректного аналізування.

Другий розділ містить інформацію про теоретичні й практичні аспекти збирання та аналізування сигналів різної природи. Розглянуті особливості збирання аналогових та дискретних сигналів, а також робота з лічильниками. Значна увага приділена темі систем узгодження сигналів із вимірювальних перетворювачів та їх практичній реалізації в системах збирання.

Автори впевнені, що цей навчальний посібник буде дуже корисним для здобувачів освіти, які цікавляться програмуванням та аналізуванням даних. Він допоможе їм набути практичних навичок та здобути знання, потрібні для розв’язування широкого кола задач, пов’язаних з автоматизацією фізичного експерименту, створенням систем типу «Інтернет речей», та аналізування сигналів різної природи.

Розділ 1
Мікроконтролерні системи збирання даних: апаратне та програмне забезпечення

1.1 Загальні відомості про мікроконтролери

Перш ніж почати працювати з апаратною обчислювальною платформою Arduino, важливо ознайомитися із загальними відомостями про мікроконтролери. Мікроконтролери застосовують насамперед для автоматизації в метрології, техніці керування й автоматичного регулювання. Перевага мікроконтролерів полягає в тому, що з їх допомогою можна ефективно та з малими витратами вимірювати й інтерпретувати фізичні величини, щоб потім ухвалювати необхідні рішення та виконувати потрібні дії.

Галузь можливих застосувань мікроконтролерів надзвичайно широка: від домогосподарства (наприклад, для керування теплицею або освітленням) до промислового виробництва, де обслуговують та експлуатують комплексні пристрої, керовані системами мікроконтролерів. На рисунку 1.1 наведено типовий приклад оброблення даних для керування зрошувальною установкою теплиці. Контролер фіксує дані про температуру довкілля, вологості ґрунту, одержану від датчиків. Результати вимірювання у подальшому піддають логічному обробленню в мікроконтролері, після цього формують сигнали керування насосом для поливання.

1.1.1 Структура та принцип роботи контролера

Контролер являє собою, по суті, мікрокомп’ютер і містить усі властиві йому основні модулі (рис. 1.2). Стандартні блоки кожного мікроконтролера – це

центральний процесор (CPU), оперативна пам’ять (RAM), а також пам’ять програм (Flash-пам’ять) і зовнішні пристрої.

Центральний процесор. Основний функціональний пристрій мікроконтролера – центральний процесор (CPU – Central Processing Unit). Його можна порівняти з «мозком» мікроконтролера. Сигнали в ньому представлені в цифровій формі і над ними виконуються арифметичні й логічні операції.

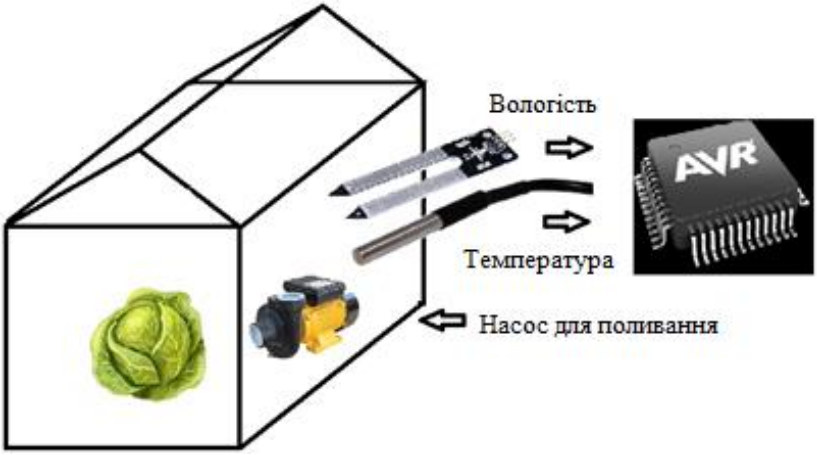


Рисунок 1.1 – Система керування мікрокліматом теплиці

Оперативна пам’ять і пам’ять програм. Оперативну пам’ять і пам’ять програм традиційно розглядають окремо. Програма користувача, яку ми самі писали, зберігається в незалежній Flash-пам’яті програм. Залежно від типу контролера пам’ять програм може займати об’єм від декількох кілобайтів до мегабайтів. Крім того, в деяких обчислювальних системах можна збільшити пам’ять програм, підключаючи зовнішні Flash-накопичувачі.

Оперативна пам’ять служить для тимчасового зберігання різних проміжних даних. Тут зберігаються й результати обчислень, одержані під час виконання

програми. Призначення оперативної пам’яті ОЗП (RAM – Random Access Memory) – можливість швидкого звернення до обмеженої кількості даних. Її об’єм зазвичай значно менший, а швидкодія набагато більша, ніж Flash-пам’яті. Значення записуються й зберігаються в ОЗП під час виконання, вона енергозалежна на відміну від Flash-пам’яті, тобто після перезавантаження контролера вміст ОЗП повністю стирається. На рисунку 1.3 зображена структура ОЗП мікроконтролера ATmega328P.

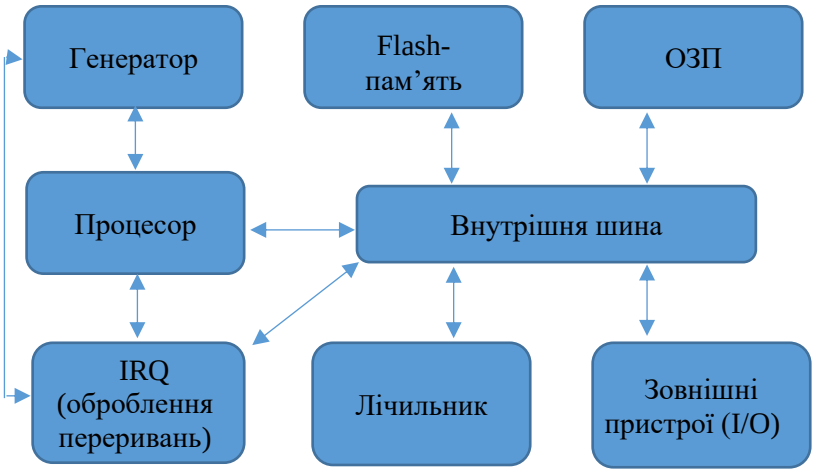


Рисунок 1.2. – Спрощена структура мікроконтролера

Зовнішні пристрої. Зовнішніми (периферійними) пристроями часто називають усі компоненти мікроконтролера, крім центрального процесора й пам’яті. Зокрема, до них відносять зовнішні інтерфейси, наприклад, цифрові входи і виходи (Input / Output – скорочено I/O). Більшість мікроконтролерів забезпечені також різними цифровими та аналоговими входами й виходами.

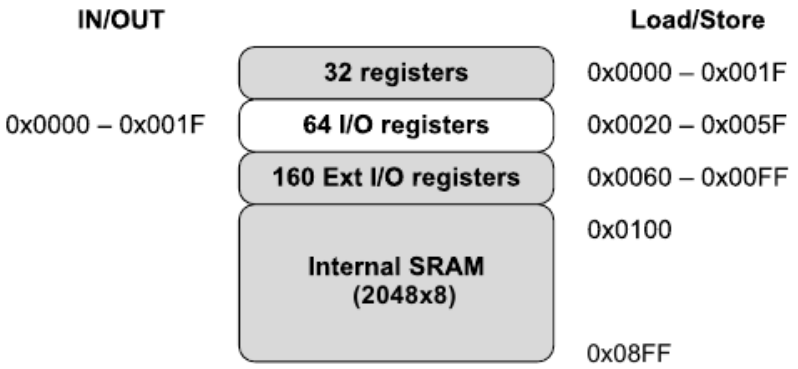


Рисунок 1.3 – RAM-пам’ять мікроконтролера ATmega328P

1.1.2 Порівняння технологій RISC і CISC

Розгляд RISC- і CISC-технологій – це вже більш глибокий погляд на цифрову й мікроконтролерну техніку. Відразу потрібно зауважити, що контролери AVR для Arduino базуються на технології RISC.

Технологія CISC. У разі технології CISC до ОЗП завантажуються й програма, й дані. Говорять також про те, що код програми та дані поділяють між собою одну й ту саму область пам’яті. Це мало сенс, зокрема, в перших обчислювальних системах, оскільки оперативна пам’ять була дорогою.

Для мікроконтролера значно важливішою відмітною ознакою є структура команд. Комп’ютер з CISC-технологією має у своєму розпорядженні великий асортимент дуже вузькоспеціалізованих команд. У цифровій техніці команда – це послідовність певних байтів. Один байт може приймати 256 різних станів. Щоб реалізувати більше ніж 256 різних команд, потрібні додаткові байти. Таким чином, спеціальна команда займає багато місця, наприклад п’ять, байтів. Завантаження цієї команди триває довше, ніж короткої однобайтової команди.

Технологія RISC. Було встановлено, що в CISC-комп'ютерах зазвичай близько 90 % вихідного коду програм складається лише з 30 різних команд. Так виникла думка реалізувати в центральному процесорі систему обмеженої кількості коротких і швидковиконуваних команд. Таким чином, у RISC-мікроконтролерах команди, зазвичай, складаються не більше ніж з 1–2 байтів. Довгу спеціальну команду доводиться складати з декількох коротких. Щоб досягти рівної продуктивності з CISC-комп'ютерами, більшість RISC-комп'ютерів має велику кількість регістрів. Регістр – це вбудована в центральний процесор надшвидкісна пам'ять. Ще одна відмінна ознака RISC-систем – чітке фізичне й логічне розділення між областями пам'яті програм і даних.

Тож можна зазначити, що CISC-комп'ютер має безліч спеціальних команд, які займають великий об'єм пам'яті й потребують зазвичай тривалого часу виконання. Команди RISC-комп'ютера використовують менше пам'яті й виконуються набагато швидше. Проте недолік RISC-технології полягає в тому, що спеціальні команди доводиться замінювати ланцюжками з декількох основних команд. Таким чином, і CISC-, і RISC-технологія мають свої переваги та недоліки. Необхідно зазначити, що не існує ні повністю RISC-, ні повністю CISC-систем.

1.1.3 Програмування мікроконтролерів

Ступінь інтеграції мікропроцесорів і мікроконтролерів усе більше зростає, і вони все ширше проникають у прикладні галузі метрології, техніки керування, автоматичного регулювання. Навіть у звичайному житті мікроконтролери стають дедалі популярнішими. Так відбувається, з одного боку, з огляду на те, що сьогодні складні аналогові схеми замінюють більш простими

цифровими мікроконтролерами. Проте вирішальна перевага мікроконтролерів – це неперевершене співвідношення ціна / продуктивність.

Поняття програми. Програма – це опис процесу оброблення інформації. Під час виконання програми розраховується сукупність вихідних значень, зважаючи на сукупність змінних або постійних вхідних значень. Мета виконання програми – збирання даних або отримання відгуку на вхідні значення. Поряд із власне обчисленнями програма може містити команди для доступу до апаратних засобів комп'ютера й для керування процесом виконання алгоритму. Програма складається з декількох рядків так званого початкового тексту. Кожний рядок містить один або кілька арифметичних або керувальних операторів. Не лише самі команди, а й послідовність їх виконання істотно впливає на результат оброблення інформації. Виконання відповідних операцій відбувається послідовно (почергово). Впорядковану певним чином послідовність інструкцій програми називають також алгоритмом.

Програмування мовою С. Мова програмування С (ANSI C) достатньо проста для вивчення. С – це мова програмування високого рівня, яку створив Денніс Річі (Dennis Ritchie) на початку 70-х років XX ст. в Bell Laboratories для операційної системи UNIX. З того часу ця мова дуже поширена. Сфери застосування мови С досить різноманітні. Її використовують, наприклад, у системному й прикладному програмуванні. Основні модулі всіх систем UNIX і ядро багатьох операційних систем запрограмовані на мові С.

Інші мови, наприклад, C++, Objective-C, C#, Java, PHP і Perl, орієнтуються на синтаксис і властивості мови С. Вивчення цієї мови програмування дуже вигідне, оскільки в подальшому легше освоїти багато систем мікроконтролерів. Майже для всіх мікроконтролерів існує безкоштовний

компілятор C, пропонований виробником мікроконтролера. Компілятор C від Arduino дещо простіший, ніж професійні C-компілятори, але досить ефективний. Із компілятором Arduino не потрібно піклуватися про програмування складних апаратних засобів, оскільки в середовищі розроблення є відповідні вбудовані команди.

1.2 Мікроконтролерна платформа Arduino

Arduino – апаратна обчислювальна платформа, основними компонентами якої є плата введення / виведення на базі мікроконтролера AVR і середовище розроблення мовою Arduino (C/C++).

Апаратна частина. Апаратні засоби Arduino включають популярні та доступні комплектувальні, які випускають серійно, тому принцип роботи системи зрозумілий, налаштування схеми згідно з вимогами розробника просте й забезпечена можливість подальшої модифікації.

Плата Arduino складається з мікроконтролера Atmel AVR (ATmega328 і ATmega168 – у нових версіях і ATmega8 – у старих) та елементного обв'язування для програмування та інтеграції з іншими схемами. На кожній платі обов'язково наявні лінійний стабілізатор напруги 5 В (у деяких моделях 3,3 В) і кварцовий резонатор 16 МГц. У мікроконтролер попередньо прошитий завантажувач, тому зовнішній програматор не потрібен. На концептуальному рівні всі плати програмують через інтерфейс RS-232 (послідовне з'єднання), але реалізація цього способу відрізняється від версії до версії. Плата Serial Arduino містить просту інвертувальну схему для конвертації рівнів сигналів RS-232 у рівні ТТЛ і навпаки. Актуальні на сьогодні плати, на зразок Diecimila, програмують через USB, це здійснюють через мікросхему конвертера

USB-to-serial, наприклад, FTDI FT232 або Prolific PL2303. У деяких варіантах, таких як Arduino Mini або неофіційні Boarduino, для програмування потрібне під'єднання окремої плати або кабеля USB-to-serial.

Плати Arduino дозволяють використовувати більшу частину I/O виводів мікроконтролера в зовнішніх схемах. Наприклад, у платі Diecimila доступно 14 цифрових вводів / виводів (рівні «LOW» – 0В і «HIGH» – 5В), 6 із яких можуть видавати ШІМ-сигнал, і 6 аналогових входів (0-5В). Ці виводи доступні на верхній частині плати через 0,1 дюймові розніми типу «мама». На ринку доступні кілька зовнішніх плат розширення, відомих як «shields».

Назва «Arduino» (і похідні від нього) є торговою маркою для офіційного продукту, її не можна використовувати для похідних робіт без дозволу. В офіційному документі, про використання назви Arduino наголошують, що проєкт відкритий для всіх бажаючих працювати над офіційним продуктом. Результатом захисту назви стало відгалуження від Arduino групи користувачів, що привело до випуску еквівалентної плати, яку назвали Freeduino, її найменування не є торговою маркою, його можна використовувати в будь-яких цілях.

1.2.1 Різновиди плат Arduino: клони, оригінали та сумісність

Періодично команда розробників Arduino випускає нові плати, також світова спільнота розробила та клонувала ще велику кількість плат. Спробуємо розібратися, що собою являють ці розробки.

Для початку потрібно зазначити, що всі «розміри» й різновиди Ardino-плат абсолютно сумісні один з одним – якщо вас зацікавив проєкт на Ardino Nano, то ви з легкістю зможете реалізувати його на Ardino Uno або Ardino Mega,

причому ні в коді, ні в схемі переробляти нічого не доведеться. Можна й навпаки, наприклад, з Ardino Mega на Ardino Mini – лише б виводів та пам'яті вистачило. Так само ніякої різниці немає у виборі конкретної плати всередині розмірного ряду – можна взяти проєкт для Arduino Diecimila і спокійно портувати його на UNO й навпаки. Тим більше немає ніякої принципової відмінності, хто зробив цю плату і яка її назва, основою буде той самий мікроконтролер ATMega.

Оригінальні плати. Розробники випускають плату в декількох основних форм-факторах:

– Ardino xxx – стандартний розмір, 20 входів / виходів, повна сумісність з усіма Шілд;

– ArdinoMega xxx – збільшений розмір, 70 входів / виходів, сумісність не з усіма Шілд;

– ArdinoNano xxx – зменшений розмір, 22 входи / виходи, не сумісна з Шілд;

– ArdinoMini xxx – ще менший розмір, 20 входів / виходів, не сумісна з Шілд, не має USB.

Ardino xxx. Стандартний і найпоширеніший розмір. Коли говорять «Ardino» («звичайна Ardino»), зазвичай мають на увазі саме такі плати.

Найперші плати були в цьому форм-факторі, відповідно саме він пережив найбільше реінкарнацій (USB-версії в хронологічному порядку виходу):

Extreme, NG, Diecimila, Duemilanove, Uno, Leonardo.

Зараз на офіційному сайті доступні для купівля лише Leonardo та Uno, проте в мережі «Інтернет» можна придбати велику кількість різноманітних плат у цьому форм-факторі, оскільки він найбільш підходить середньому користувачеві. Усі ці плати мають однакову кількість входів / виходів, зібраних на однакових рознімах (для підключення периферії й Шілд), програмуються за USB й базуються на мікроконтролері ATMega. На ранніх версіях

був установлений ATMega8, потім – ATMega168, на сьогодні – ATMega328.

На ATMega8 лише 3 ШІМ-виходи, 8 кБ під скетч 1 Кб оперативної пам'яті, але для багатьох додатків цього вистачає. У ATMega168 уже 6 ШІМ-каналів і 16 Кб під прошивання, а в 328-й 32 кБ під програми і 2 кБ ОЗП. До речі, не вся флеш-пам'ять доступна користувачеві, частину її займає бутлоадер.

На всіх платах до UNO стояв чіп-перетворювач USB-UART FT232, що дозволяє під'єднувати плату прямо до USB і програмувати без програматора. У разі підключення до системи з'являється віртуальний COM-порт, який використовує середовище розроблення Ardino для програмування.

На Arduino UNO (рис. 1.4 а) вирішили замінити перетворювач USB-UART на мікроконтролер Atmega8U2 (у більш пізніх ревізіях 16U2) – у нього залите спеціальне прошивання, що робить те саме що і FT232. Це дало деякі переваги, наприклад, підвищилася швидкість прошивання: тепер замість ~ 10 секунд потрібно чекати ~ 3 секунди, а головне, в цей МК-конвертор можна записати своє прошивання, і перетворити Ardino на мишку, клавіатуру або міді-пристрій.

Також варто зробити застереження: ця Atmega8U2 / 16U2 насправді робить не те саме, що FT232, вона не реалізує дуже зручної можливості – BitBang, тому перетворити плату на програматор уже не вийде.

В UNO з'явилися на «верхньому лівому» конекторі SDA і SCL – піни інтерфейсу I^2C, але вони дублюють SDA і SCL на аналогових входах 4 і 5, і функціонал це не розширює. На цей час старі Шілд підходять до UNO, нові Шілд (яких поки що дуже мало), повністю сумісні зі старими платами, хоча й загрожують «уткнутися» в неї

новими пінами – їх, можливо, доведеться підігнути або відкусити.

Leonardo (рис. 1.4 б). Ця плата дійсно крок уперед – усе розміщено на одному чіпі, USB незалежний ні від UART, ні взагалі від яких би то не було пінів. Плата побудована на ATmega32u4 і порівняно з попередніми моделями має більші можливості. На 0,5 кБ збільшилася ОЗП, ШІМ-виходів стало на 1 більше, аналогових входів стало 12 (6 – де в усіх Ardino, нові +6 розкидані по цифрових пінах) і, як уже зазначали, розділені USB та UART. Так само підтримуються не лише віртуальний COM-порт, а й мишка та клавіатура, набагато простіше ніж це реалізовано в UNO.

Правда «крок уперед» вийшов із деякими нюансами – бутлоадер тепер займає 4 кБ, а ще в будь-який скетч для Леонардо додається підтримка USB. Найпростіший скетч `blink` для Duemilanove / UNO займе 1 084 байти, а для Leonardo – 4 858 байт. Фізично Леонардо має ту саму карту розведення, що й UNO, тому він сумісний зі старими Шілд.

ArdinoMega xxx. Серія збільшених плат (за розміром і характеристиками) презентована такими моделями (в хронологічному порядку): Mega, Mega2560 (рис. 1.5) і Arduino ADK.

До плати під’єднуються майже всі Шілд, але через різне (із «звичайними» Ardino) розміщення виводів SPI-інтерфейсу, Шілд використовують його з цифрових пінів 11, 12, 13, що будуть не сумісними. Приклад – старий Ethernet Шілд. На новому SPI використовують зі стандартної вилки ISP і все добре працює і на «мегах», і на «звичайних» Ardino.

На платах велика кількість виводів:

- 54 цифрових, із яких 15 – із ШІМ;
- 16 – аналогових входів;

Пам’ять:

- 128 (Mega) / 256 кБ (Mega2560) – флеш;

– 8 кБ оперативної пам’яті;
– 4 кБ EEPROM;
– фізичних UART.

а б

Рисунок 1.4 – Зовнішній вигляд плат Arduino UNO (а) та Leonardo (б)

«Мега» побудована на ATmega1280, а «2560» і «ADK» – на ATmega2560, тому плати розрізняють об’ємом пам’яті, до того ж у нових – 2560 і ADK – USB-частина виконана на ATmega8U2 (на більш пізніх ревізіях 2560 – на ATmega16U2), так само, як і в УНО.

ADK також має USB-host, від якого очікують сумісність з Android-телефонами.

Arduino Nano. Маленька плата з mini-USB. Шілд до неї не підходять, але сама вона зручно розміщується в макетній платі. У ранніх версіях, як і в UNO використовують ATmega168, на сьогодні – 328. Як USB-UART міст застосовують перетворювач FT232.

Arduino Mini. Крихітна мікроконтролерна плата (тут якийсь історичний ляп – Ardino міні, чомусь значно менше, ніж Arduino нано). Пережила кілька версій – має незначні відмінності в призначенні деяких виводів. З Шілд не сумісна, але зручна для вбудовування в закінчені пристрої – нічого зайвого. На міні немає USB – програмується вона за допомогою перехідника USB-Serial (наприклад, на базі тієї

самої FT232). Є варіанти плат, які працюють на 3,3 В і 8 МГц, що зручно для використання в поєднанні з пристроями мобільного стандарту живлення (3,3 В).

Рисунок 1.5 – Зовнішній вигляд плати Arduino Mega2560

Проєкт Arduino – повністю відкритий (доступна вся технічна документація, необхідна для виробництва), плати копіює й творчо переробляє велика кількість виробників. Обмеження стосується лише назви «Arduino» – його не можна використовувати для назви неіталійських плат.

Усе, що розроблено не компанією Arduino, можна умовно поділити на три групи: «клони», «сумісні» й «Arduino-подібні» .

Клони. Клонами вважають плати Arduino, які виготовляють сторонні компанії з використанням офіційної документації. Іноді вони змінюють колір і назву готового продукту. Такі плати (клони) повністю копіюють Arduino, вони сумісні з нею, тобто відмінність між клоном та оригіналом полягає у виробнику – відповідно відмінності можуть бути щодо складання. Загалом забезпечити гідну

якість може китайський виробник, тому купівля оригінальних плат зазвичай призводить до зайвих витрат.

Сумісні. Деякі виробники пішли не шляхом копіювання, а вирішили щось додати до проєкту і розробили велику кількість своїх плат, повністю сумісних з Arduino – умовно їх можна назвати «сумісними переробками» (переосмисленнями). Наприклад: Freeduino, Freetronics, Eleven, Seeeduino, CraftDuino, Diavolino, Japanino і багато інших.

Зазвичай доопрацювання й перероблення мають досить естетичний характер (не мають принципових змін функціонала або характеристик), інакше плати втратили б сумісність. Це додаткові розніми, інше розміщення світлодіодів і кнопок, свій монтаж, застосування інших компонентів (в інших корпусах, інших розмірів), інші схеми живлення, відмінний USB-рознім. Є версії міні-, нано- та мегасумісних переробок.

Сумісність з Arduino складається з двох речей:

1. Сумісність із платами розширеннями – Шілд. Для цього розміщення і вид рознімів повинен бути, як і на оригінальній платі Arduino.

2. Програмна сумісність (програмна частина проєкту Arduino – це середовище розроблення (IDE), бібліотеки та скетчі).

На платах Arduino встановлені мікроконтролери фірми Atmel, сім'ї ATMega – ATMega8 / 168 / 328 – на всіх, окрім Мега (ATMega1280 / 2560) і Леонардо (ATMega32U4).

Зазвичай тактування МК відбувається кварцовим резонатором на 16 МГц (рідше – 8 МГц)

Живлення МК на платах відбувається від джерела 5 В (рідше – 3,3 В).

Завантаження скетчів відбувається через бутлоадер (спеціальна програма-завантажувач, заздалегідь прошита в

МК), хоча в останніх версіях середовища з'явилася можливість прошивання скетчу через програматор.

Тобто будь-яка плата, що задовольняє перелічені умови (тип контролера, частота, напруга живлення, наявність бутлоадера), зможе використовувати всі напрацювання спільноти Arduino – скетчі, бібліотеки, й записувати все можна буде в тому самому середовищі Arduino і завантажувати звідти ж.

За деяких допрацювань, можна модернізувати бібліотеки для використання не в середовищі Arduino чи середовищі для використання плат із нехарактерними МК або частотами їх роботи. Надалі будемо вважати програмно сумісними лише ті плати, які коректно запрацюють без будь-яких допрацювань.

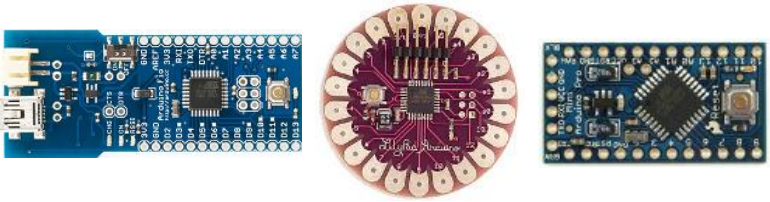
Arduino-подібні. Деякі виробники вносять більш істотні зміни, втрачаючи сумісність із Шілд. Ці плати можна назвати Arduino-подібними, з яких найбільш поширеними є такі:

Arduino Fio (рис. 1.6 а) – плата для портативних пристроїв з живленням від літієвих батарей;

Lily Pad (рис. 1.6 б) – кругла плата для «електронного одягу»;

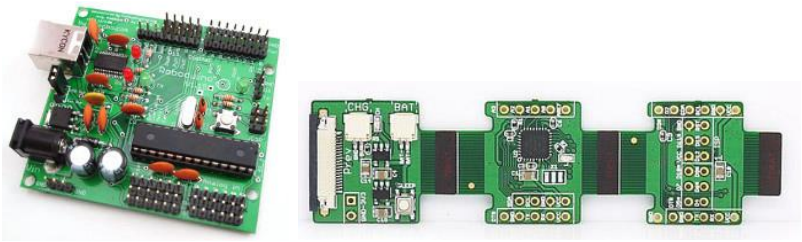
Pro Mini (рис. 1.6 в) – значною мірою перероблена версія Arduino Mini.

Менш відомі приклади – Roboduino (рис. 1.7 а) – плата для керування великою кількістю сервоприводів. Незважаючи на подібний з Arduino вигляд, Шілд із нею не сумісна. Менш подібна на родоначальника Rainbowduino – для керування світлодіодними матрицями, Ні до чого не подібна версія Seeeduino Film (рис. 1.7 б) від Seeed studio. Є ще велика кількість різноманітних плат подібних до Arduino, кожна з яких має свої особливості й спеціально заточена під використання з певними апаратними засобами.



а б в

Рисунок 1.6 – Зовнішній вигляд Arduino-подібних плат Ardino Fio (а), Lily Pad (б), Pro Mini (в)



а б

Рисунок 1.7 – Зовнішній вигляд Arduino-подібних плат Roboduino (а) і Seeeduino Film (б)

1.2.2 Підготовка до роботи з Arduino

Перш ніж почати експериментувати й програмувати, потрібно виконати деяку попередню підготовку. На персональному комп'ютері необхідно встановити драйвер для віртуального COM-порту (послідовний інтерфейс) й середовище програмування / розроблення для Arduino.

Установлення драйвера. Спочатку потрібно встановити драйвер для мікросхеми перетворювача USB – UART. Якщо ви використовуєте оригінальну плату Arduino або високоякісний клон, необхідно встановити драйвер мікросхеми FT232RL, хоча цю операцію можна й не проводити, оскільки цей драйвер уже входить до

комплекту інсталятора середовища розроблення Arduino. У разі використання сумісних плат потрібно встановити драйвер перетворювача мікросхеми в платі (здебільшого використовують мікросхеми CH340, CP2102 або PL2303). Мікроконтролер перетворювача інтерфейсів запрограмований таким чином, що в менеджері пристроїв плата позначена як віртуальний COM-порт (віртуальний послідовний інтерфейс). У разі підключення плати комп'ютер отримує додатковий COM-інтерфейс. ОС Windows не сприяє відмінності між фізичним та віртуальним COM-інтерфейсом. Новий інтерфейс отримує вільне ім'я, наприклад, COM7.

Установлення програмного забезпечення Arduino. Установимо середовище програмування Arduino, що базується на програмі Processing. Processing – це просте інтегроване середовище розроблення, призначене спеціально для користувача, який поверхнево володіє програмуванням, проте потребує написання власної програми. Останню стабільну версію середовища можна завантажити з офіційного сайту проєкту: https://www.arduino.cc/ у вигляді встановлювача або ж архіву з розпакованою версією, що не потребує встановлення (портативна версія). Важливо регулярно виконувати оновлення програмного забезпечення, оскільки до середовища постійно додають нові бібліотеки, драйвери, а також виправляють наявні помилки.

Інтегроване середовище розроблення (IDE – Itegrated Development Environment) Arduino – це кросплатформений додаток на Java, що вміщує редактор коду, компілятор і модуль передавання прошивання в плату.

Середовище розроблення спроєктоване для програмування новачками, не ознайомленими з розробленням програмного забезпечення. Мова програмування аналогічна використовуваній у проєкті

Wiring. Це C/C++, доповнена деякими бібліотеками та з певними обмеженнями (наприклад, недоступні можливості багатопоточного програмування та відсутня об'єктна модель C++). Програми обробляють за допомогою препроцесора, а потім компілюють за допомогою вільно поширюваного компілятора AVR-GCC.

Середовище розроблення Arduino. Після встановлення або копіювання з архіву програми Arduino можна запустити файл Arduino.exe. Для зручності виклику програми можна створити ярлик на робочому столі. В Arduino-IDE розміщені різні інструменти й налаштування, що полегшують роботу з програмою Arduino (рис. 1.8).

Розглянемо докладніше середовище розроблення Arduino. З меню можна викликати всі функції Arduino, далі ми ще будемо знайомитися з окремими його пунктами. Під головним меню знаходиться панель інструментів, де розміщені такі команди:

– *Перевірка* – виконується перевірка на наявність помилок у програмному коді;

– *Завантаження* – формування файлу, що буде переданий до плати мікроконтролера, та його завантаження;

– *Новий* – створення нового файлу Arduino;

– *Відкрити* – відкриття тексту програми;

– *Зберегти* – збереження тексту програми;

– *Монітор COM-порту* – відкриття вбудованого командного терміналу.

До початку роботи потрібно задати вихідні установки. Для цього необхідно вибрати потрібну плату Arduino та використовуваний інтерфейс (рис. 1.9). У цьому разі зазначена плата Arduino Nano. Якщо потрібна інша, то необхідно вибрати відповідну плату зі списку.

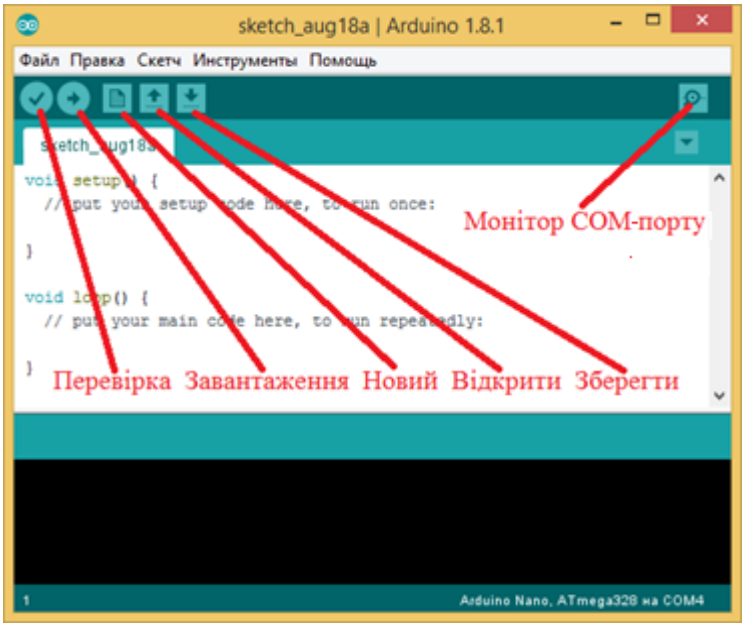


Рисунок – 1.8 – Середовище розроблення Arduino

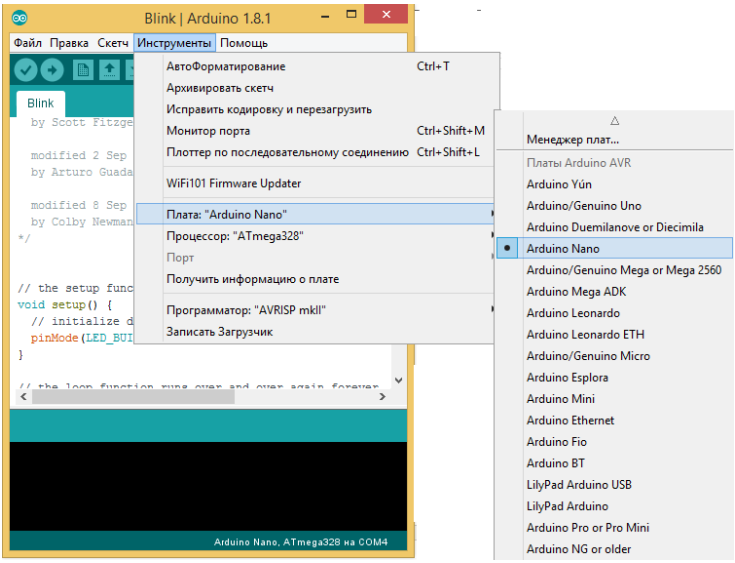


Рисунок 1.9 – Вибір плати мікроконтролера

1.3 Основи програмування Arduino

Біти і байти. На цей час найменша одиниця вимірювання інформації в обчислювальній техніці – це біт. Англійською мовою двійкові числа позначають як binary digit (двійкова цифра). Так виникло штучно утворене слово «біт» (Bit). Один біт може набувати лише двох значень: 0 або 1 (наявність або відсутність електричного струму). Інформація може уявлятися як послідовність із певного числа бітів. У нашому випадку 1 байт – це послідовність із 8 бітів. Отже, один байт може уявлятися й інтерпретуватися як 256 різних комбінацій. Половину байта, тобто 4 біти, називають тетрадою або напівбайтом:

4 біти = 1 напівбайт = 1 тетрада;

8 бітів = 2 напівбайти = 1 байт.

Коефіцієнт перерахунку між двійковими одиницями вимірювання кількості інформації дорівнює 1 024 (табл. 1.1). Виняток – перетворення біта на байт, оскільки в цьому разі коефіцієнт перерахунку дорівнює 8. Ці некруглі числа виникають через те, що в інформатиці обчислення проводять за основою «2» (2^n).

Таблиця 1.1. – Співвідношення між одиницями вимірювання кількості інформації

Найменування	**Коефіцієнт перерахунку**
1 байт	8 бітів
1 кілобайт (Кбайт)	1 024 байти (10^2 байтів)
1мегабайт (Мбайт)	1 024 Кбайти (10^2 Кбайтів)
1гігабайт (Гбайт)	1 024 Мбайти (10^2 Мбайтів)
1терабайт (Тбайт)	1 024 Гбайти (10^2 Гбайтів)

1.3.1 Базова структура програми

Структура більшості програм дуже подібна, оскільки широко використовують процедурне програмування. Процедурне програмування полягає в розбитті комп'ютерних програм на невеликі окремі завдання, які позначають як процедури (підпрограми). Найменший і неподільний крок за цього методу – це команда (оператор, інструкція). Програму виконують послідовно від команди до команди. Сам термін «процедура» означає «просуватись» і походить від латинського слова «procedo». Програміст задає мікроконтролеру за допомогою програми, в якій послідовності потрібно діяти. Мета цього принципу програмування – простота розроблення і можливість повторного використання фрагментів (модулів) вихідного коду.

Під час послідовного програмування код, що складається з окремих процедур, виконується в циклі. На рисунку 1.10 показане виконання послідовності команд «уведення даних – оброблення – виведення даних».

1.3.2 Переривання виконання програми

Іноді виникає необхідність перервати послідовне виконання програми. Це виконують за допомогою переривань (`Interrupt`). Основну частину програми виконують, як і під час послідовного програмування – нескінчений цикл, у якому повторюють окремі процедури. Як тільки відбудеться зовнішнє або внутрішнє переривання, наприклад натискання кнопки, основний цикл (`Main Loop`) перерветься та відбудеться перехід до програми оброблення переривань (`Interrupt - Routine`). У ній відпрацьовують спеціальні завдання, наприклад аварійне вимкнення і т. ін. Далі робота триває знову в основному

циклі, де можуть виконуватися допоміжні операції. Виконання переривань ілюструє рисунок 1.11.

1.3.3 Структура програми Arduino

Програма Arduino складається з таких основних частин:

– коментарів та опису програми;
– заголовків файлів і підключення бібліотеки;
– оголошення глобальних змінних;
– стандартного налаштування `void setup ()` (порти й конфігурація);
– основного циклу `void loop ();`
– власних процедур.

Докдержуватися деяких основних правил програмування зовсім не важко. Наступний крок – застосування вже вивченого матеріалу для написання невеликої програми.

Тепер створимо першу справжню програму Arduino. Як вправа можна передрукувати вихідний текст лістингу 1.1.

Лістинг 1.1 – Перша програма для контролера Arduino

```
// Перша програма Arduino :-)

int ledPin = 13;  //Світлодіод під'єднано до цифрового виводу 13 (створюємо глобальну змінну ledPin і присвоюємо їй значення «13»)

void setup()   //блок налаштування
{
 Serial.begin(9600);     //ініціалізація послідовного передавання даних на швидкості 9600 бод
```

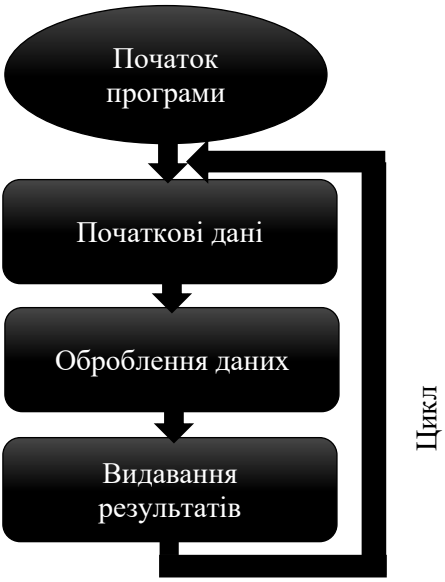


Рисунок 1.10 – Послідовне виконання програми

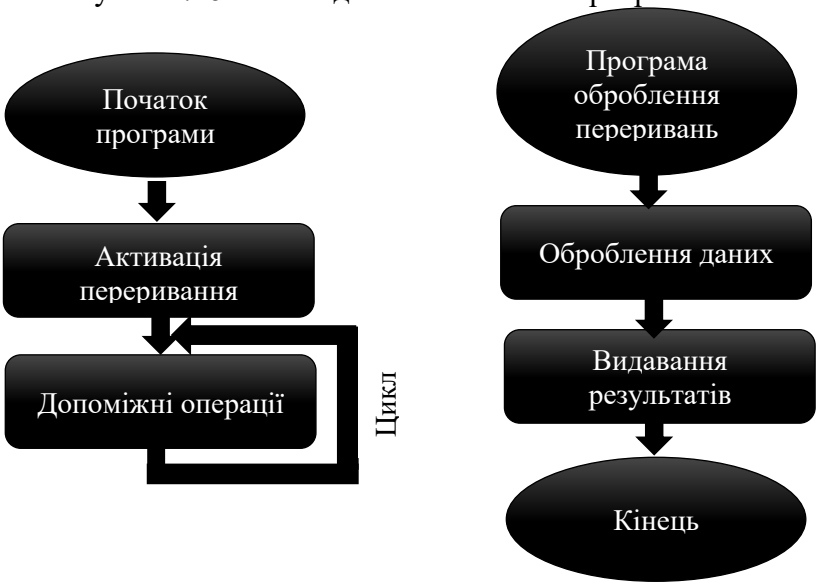


Рисунок 1.11 – Переривання виконання програми

```
   pinMode(ledPin, OUTPUT); //встановлення
цифрового порту ledPin як «вихід»

   Serial.println("First Arduino Program");
//відсилаємо повідомлення на віртуальний
COM-порт
    Serial.println();
   }
   void loop(){    //Основний цикл
     Serial.print("The sum 5 + 188 is ");
     Serial.print(5+188);
     while(true)        //Нескінченний цикл
     {
      digitalWrite(ledPin,  HIGH);       //
Вмикаємо світлодіод
      delay(500);                 //Чекаємо
500 мс
      digitalWrite(ledPin,            LOW);
//Вимикаємо світлодіод
      delay(500);                         //
Чекаємо 500 мс
      continue;
     }
   }
```

З натисканням кнопки скидання програма запускається знову.

Запустіть термінал після компіляції вихідного коду і передавання до контролера. Наша програма розраховує суму двох чисел і виводить результат через термінал. Після виведення результату починає миготіти світлодіод.

Програма ілюструє найважливіші принципи послідовного алгоритму. Спочатку контролеру Arduino повідомляється, що змінна `ledPin` повинна містити число 13. Потім виконується установлення початкових значень. Тут зазначається, до якого виводу приєднується світлодіод

і задається швидкість послідовного інтерфейсу, що дорівнює 9 600 бодів. Установлення завершується виведенням інформації через послідовний інтерфейс. Далі відбувається перехід до основного циклу з ім'ям `loop()`. У ньому перша функція `Serial.print` виводить текст, а друга – суму 5 + 188. Пам'ятайте, що функція `Serial.println` після виведення здійснює перехід до лівого краю з передаванням керування на початок наступного рядка (CR + LF), а `Serial.print` – ні.

Після виведення інформації запускається нескінченний цикл while, і виконується безперервне миготіння світлодіода.

1.3.4 Команди Arduino та їх застосування

Коментарі в початковому тексті. Щоб правильно читати програму після закінчення певного часу й розуміти її, необхідно ретельно та акуратно документувати свій вихідний код. Можна впроваджувати документацію прямо у вихідний код. Для цього є різні символи коментарів, призначені для включення звичайних текстів (лістинг 1.2). У середовищі розроблення Arduino текст коментарів виділений сірим кольором.

Лістинг 1.2 – Застосування коментарів у вихідному коді

```
//Такий вигляд має однорядковий коментар
/*
А так пишуться
Багаторядкові коментарі
*/
```

Коментарі в коді програми значно полегшують її читання через деякий час. Подумайте, чи зможете ви пригадати через 1 рік, що робили сьогодні в програмі?

Фігурні дужки. Фігурні дужки виділяють фрагмент коду для компілятора.

Блок коду завжди відкривається символом «{»і завершується «}». Між дужками розміщені команди.

```
type Main_Function ()
{
Між дужками знаходяться команди
}
```

Крапка з комою. Крапка з комою завершує команду. Якщо пропустити цей символ, компілятор Arduino видасть повідомлення про помилку.

```
int x = 42; // Тут змінна x оголошується як Integer і їй присвоюється число 42, крапка з комою завершує визначення.
```

Типи даних і змінні. Кожна програма складається з різних змінних, які або залежать від зовнішнього оточення (наприклад, аналоговий або цифровий вхід), або потрібні для розрахунку в програмі й виведення результатів. У розпорядженні програміста є різні типи змінних: `Byte` (байт), `Integer` (ціле число), `Long` (довге ціле число) і `Float` (число з плаваючою комою). Тип змінної завжди потрібно ставити перед її застосуванням.

Ім'я змінної. У середовищі розроблення Arduino є відмінність між верхнім і нижнім регістрами символів імені змінної. В Arduino допускається підкреслення _ , його часто використовують, щоб зробити довгі імена змінної розбиранняливими. Ключові слова (if, while, do і т. ін.) не можуть бути іменами змінної. Імена глобальних змінних і функцій не можуть збігатися. Крім того, не допускається одночасне задання функцій і локальних змінних з одним і тим самим ім'ям.

Локальні й глобальні змінні. Якщо змінна оголошується в межах функції, процедури або як аргумент функції, вона є локальною. Це означає, що змінна існує

лише всередині своєї функції. Змінна, оголошена поза функцією, є глобальною і визначена для всіх функцій у межах нашої програми.

```
byte Variable; // Змінна типу Byte,
вона може набувати значень від 0 до 255
float PI = 3.1415; // Константа PI
оголошена як Float (з плаваючою точкою)
int myArray[9]; //Оголошено масив
розмірністю 10 елементів типу Integer.
```

1.3.5 Типи даних

Розглянемо, які типи даних існують і скільки вони займають пам'яті.

***Boolean*.** Змінна типу Boolean може набувати двох станів: true (істина) або false (брехня). Така змінна займає 1 байт у пам'яті. Значення true відповідає логічній одиниці, а false – це не логічний нуль, як часто думають, а стан, відмінний від логічної одиниці.

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

Byte. 1 байт – це 8 біт, тому така змінна може набувати значення від 0 до 255.

```
byte myVariable = 0; // Змінна
ініціалізується нульовим значенням
```

Char. Символ має розмір 1 байт. Змінна Char – це символ в одиночних лапках (апострофах). Якщо потрібний рядок символів, то їх розміщують у подвійних лапках (наприклад, «ПРИВІТ»). Символ отримує номер із набору символів ASCII. Наприклад, літера «A» – це число 65. Символьні змінні можуть набувати значень від –127 до +127.

```
char myChar = "A"; // Число 65
```

***Unsigned Char*.** Символи без знака (unsigned Char) поводить себе як символи зі знаком, але вони можуть набувати лише позитивних значень у діапазоні від 0 до 255.

```
unsigned char myChar = "B"; // Число 66
```

Int (Integer). Ціле число (integer) складається з двох байтів і може набувати значень від –32768 до +32768.

```
int myIntegerVariable = -2973; // Ціла змінна зі значенням -2973
```

Unsigned int. Тип unsigned integer охоплює змінні в діапазоні від 0 до 65 535 (2^{16} –1).

На відміну від int тип unsigned int не має знака. Така змінна займає в пам'яті два байти.

```
unsigned int
myUnsignedIntVariable = 50000; //Ціла змінна без знака зі значенням 50 000
```

Long. Змінна типу Long складається з чотирьох байтів і може набувати значень від –2147483648 до +2147483647.

```
long bigNumber = 10000000; // змінна Long зі значенням 10 000 000
```

Unsigned Long. Змінна unsigned Long складається з чотирьох байтів і може набувати значень від 0 до +4294967295. Змінна не має знака і може зберігати лише позитивні значення.

```
unsigned long veryBigNum = 54544454544; // Дуже, дуже велика змінна
```

Float. Змінні Float можуть зберігати 32 біти. Діапазон можливих значень лежить в межах від –3.4028235E+38 до +3.4028235E+38. Для цієї змінної потрібно чотири байти в пам'яті.

```
float Var = 100.42; // змінна Float зі значенням 100,42
```

String. Рядкова змінна (string) – це масив (Array) змінних Char і нульового символа.

Для кожного символа потрібно, таким чином, один байт і в кінці ланцюжка додатково ще один байт для нульового символа. Таким чином, наприклад, для слова «Hallo» потрібно шість байтів.

```
    char myString [] = "Hallo Max"; //
Потрібно 10 байтів
```

Arrays. Масив (Arrays) – це впорядкований набір змінних одного типу (рис. 1.12). В інформатиці масив позначає структуру даних. За допомогою масиву дані однакового типу (byte, int і т. ін.) заносяться до пам'яті комп'ютера таким чином, що доступ до даних стає можливим через покажчик. У мові Arduino нумерація елементів масиву починається з нуля, в деяких інших компіляторах нумерація починається з одиниці. Крім того, важливо знати, що Arduino на сьогодні підтримує лише одновимірні масиви. Приклад роботи з масивами наведений у лістингу 1.3.

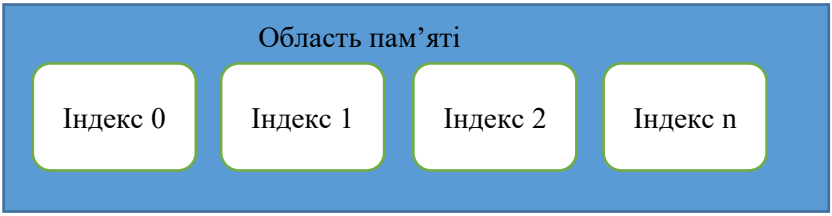


Рисунок 1.12 – Структура масиву

Лістинг 1.3 – Приклад роботи з масивами

```
//Arrays
int Array_1 [3];
int Array_2 [] = {1, 2, 3};

void setup () {
Serial.begin (9600);
Serial.println ( "Arduino Arrays");
Serial. println ();
}
void loop () {
byte x;
Array_1 [0] = 1;
```

```
Array_1 [1] = 2;
Array_1 [2] = 3;
Array_1 [3] = 4;

Serial.println ( "Array 1");
Serial.println "--------------------");
//Видавання даних першого масиву
for (x = 0; x <3; x ++) X
Serial.print (Array_1 [x]);
Serial.println ();
}
Serial.println ( "Array 2");
Serial.println "--------------------");
// Видавання даних другого масиву
Serial.print (Array_2 [0]);
Serial.println ();
Serial.print (Array_2 [1]);
Serial.println ();
Serial.print (Array_2 [2]);
while (1);
}
```

У першому масиві ми задаємо розмір «3». Таким чином, у цьому масиві можна зберігати чотири змінні типу Integer (16 бітів). Розмірність другого масиву не зазначена, і три його елементи задані у фігурних дужках. Мова йде про динамічний масив. В основному циклі ми надаємо елементам першого масиву різні значення, в нашому випадку 1, 2, 3, 4. Можете вказати інші значення, щоб краще зрозуміти роботу з масивами.

Змінна лічильника `x` циклу `for` містить значення покажчика масиву для виведення значень у терміналі. Значення елементів другого масиву й покажчика теж виводяться в терміналі.

Цикл `while (1);` являє собою нескінченний цикл основної програми для її одноразового запускання.

1.3.6 Оператори

Кожному типу даних відповідають свої оператори, які вказують на операції, що можуть застосовуватися. Розглянемо можливі оператори Arduino та їх дію.

Арифметичні оператори:

= (Присвоювання).

+ (Додавання).

– (Віднімання).

* (Множення).

/ (Ділення).

% (Ділення за модулем – обчислення залишку від ділення).

Оператори порівняння:

== (Дорівнює, наприклад, `A == в`).

!= (Не дорівнює, наприклад, `A != в`).

< (Менше, наприклад, `A < в`).

> (Більше, наприклад, `A > в`).

<= (Менше дорівнює, наприклад, `A <= в`).

> = (Більше дорівнює, наприклад, `A > = в`).

Побітова арифметика:

& (Бітове І).

| (Бітове АБО).

~ (Бітове НІ).

Булева арифметика:

&& (Логічне І, наприклад, якщо вираз `&& Unswer_b` істинний, тоді виконувати якусь дію).

|| (Логічне АБО, наприклад, якщо вираз `|| Unswer_b` істиннний, тоді виконувати якусь дію).

! (Логічне заперечення, наприклад, якщо вираз `!Unswer_b` істиний, тоді виконувати якусь дію).

Збільшення і зменшення значення:

++ (Інкремент, наприклад `i++`, змінна `i` буде збільшена на одиницю).

`--` (Декремент, наприклад `i--`, змінна `i` буде зменшена на одиницю).

`+=` (Присвоювання з інкрементом, наприклад `i+=5`, змінна `i` буде збільшена на 5, і результат присвоєний змінній `i`).

`–=` (Присвоювання з декрементом, наприклад `i–=5`, змінна `i` буде зменшена на 5, і результат присвоєний змінній `i`).

`*=` (Присвоювання з множенням, наприклад `i*=2`, змінна `i` буде помножена на 2, і результат присвоєний змінній `i`).

`/=` (Присвоювання з діленням, наприклад: `i/=2`, змінна `i` буде поділена на 2, і результат присвоєний змінній `i`).

Константи. Приклади констант:

```
HIGH / LOW (HIGH = 1, LOW = O).
INPUT / OUTPUT (INPUT = O, OUTPUT = 1).
true / false (true = 1, false! = 1).
```

Цілочислові константи. Цілочислові константи в коді – це числа, наприклад, 123. За замовчуванням ці числа інтерпретують як цілі типу `int`, однак ви можете змінити це за допомогою модифікаторів `U` і `L`.

Щоб задати цілочисловій константі інший тип, потрібно після неї записати модифікатор:

– ‘`u`’ або ‘`U`", щоб привести константу до беззнакового типу даних. Наприклад: `33u`;

– ‘`l`’ або ‘`L`’, щоб привести константу до типу даних long. Наприклад: `100000L`;

– ‘`ul`’ або ‘`UL`’, щоб привести константу до типу `unsigned long`. Наприклад: `32767ul`.

Цілочислові константи інтерпретуються як числа в десятковій системі числення, тому для задання числа в

іншій системі необхідно використовувати спеціальні префікси (табл. 1.2).

Основа десяткової системи числення – 10. Математичні операції з такими числами всім знайомі. Константи без будь-яких префіксів вважають десятковими.

Приклад:

```
101 // те ж саме, що й 101 у десятковій системі ((1 * 10 ^ 2) + (0 * 10 ^ 1) + 1)
```

Основа двійкової системи числення – 2. Для записування чисел у цій системі використовують лише 0 і 1.

Приклад:

```
B101 //те саме, що й 5 у десятковій системі ((1 * 2 ^ 2) + (0 * 2 ^ 1) + 1)
```

Таблиця 1.2 – Приведення констант до різних систем числення

Основа системи	**Приклад**	**Префікс**	**Коментар**
10 (десяткова)	618	Немає	
2 (двійкова)	B1111011	«B»	Працює лише з 8-бітними значеннями (0-255). Коректні символи 0-1
8 (восьмерична)	0173	«0»	Коректні символи 0-7
16 (шістнадцяткова)	0x7B	«0x»	Коректні символи 0-9, A-F, a-f

Префікс двійкової системи можна використовувати лише з числами розмірністю 1 байт (8 бітів) у діапазоні від 0 (B0) до 255 (B11111111). Для записування числа типу `int`

(16 бітів) у двійковому форматі можна використовувати подвійну операцію:

```
myInt = (B11001100 * 256) + B10101010;
// B11001100 - старший байт.
```

Основа восьмеричної системи числення – 8. Для записування чисел в цій системі використовують лише цифри в діапазоні від 0 до 7. Вісімкові числа характеризуються префіксом «0».

Приклад:

```
0101 // те саме, що й 65 в десятковій
системі ((1 * 8 ^ 2) + (0 * 8 ^ 1) + 1)
```

Увага! Можна допустити помилку, яку дуже складно виявити, якщо ненавмисно дописати 0 перед константою, оскільки в цьому разі остання буде інтерпретуватися компілятором як вісімкове число.

Основа шістнадцяткової системи числення – 16. Для записвання чисел у цій системі використовують цифри від 0 до 9, а також літери від `A` до `F`; у цьому разі A має значення 10, B – 11, і т. д. до F, еквівалентного 15. Шістнадцяткові значення характеризуються префіксом «0x». Зверніть увагу, що `A-F` можна вводити як у верхньому, так і в нижньому регістрі (`a-f`).

Приклад:

```
0x101 // те саме, що і 257 у десятковій
системі ((1 * 16 ^ 2) + (0 * 16 ^ 1) + 1)
```

Константи з плаваючою комою. Подібно до цілочислових констант константи з плаваючою комою використовують для того, щоб зробити програмний код більш читабельним. Під час компіляції коду дробові константи замінюють їх числовими значеннями. Наприклад, `n = 0.005;`

Константи з плаваючою комою також можуть бути виражені в експоненціальній формі. Обидва символи ‘E’ і ‘e’ інтерпретуються як показники експоненти.

Приклад:

2.34E5 = 2.34 · 10 ^ 5 = 234000;

67e-12 = 67.0 · 10 ^ -12 = 0.000000000067.

Директива `#define`. Define – це директива препроцесора, яка запускається перед компіляцією. Можна сказати, що мова йде про власний маленький компілятор, який заздалегідь перетворює команди `#define` на константи. Директива дозволяє задавати постійне значення змінної:

```
#define Variable 1 // Без крапки з комою
```

Тут змінна `Variable` набуває значення 1. Кожний раз, коли в програмі трапляється ім'я `Variable`, значення змінної замінюється на 1. Для нас ця заміна неочевидна, але компілятор діє так, як описано.

1.3.7 Керувальні конструкції

Для реагування на події (умови) в кожній програмі використовують, так звані, керувальні конструкції. Вони мають конструкцію, аналогічну тим, які використовують у мовах C і C++, і такий вигляд: if... else-if... else або switch case.

У наведених далі лістингах виведення даних здійснюється через послідовний порт Arduino і вони можуть бути зчитані через термінальну програму (монітор COM-порту, Putty Terminal або будь-яку іншу). Завантажте приклади в плату мікроконтролера і запустіть після цього термінальну програму, щоб розібратися, що відбувається.

Оператор `if`. Лістинг 1.4 ілюструє синтаксис, а лістинг 1.5 – приклад використання оператора `if`.

Лістинг 1.4 – Синтаксис конструкції `if`

```
    if (VarA == VarB) {
    //Тут знаходиться код, який
виконується, якщо VarA дорівнює VarB
    }
```

Лістинг 1.5 – Приклад використання оператора `if`

```
    // конструкція if
    int x;
    void setup(){
     Serial.begin(9600);
     Serial.println("If Construction");
     Serial.println();
    }
    void loop(){
      if(x==10) {
        Serial.println("Змінна «x» почала
дорівнювати 10");
        while(1);
      }
      x++;
    }
```

Основний цикл `void loop ()` виконується до того часу, поки змінна `x` типу `Integer` не дорівнюватиме 10. Тіло циклу складають оператори між `if` і дужками `{}`. За допомогою `if` можна реалізувати простий спосіб передавання керування (розгалуження в програмі). Ви можете самі експериментувати з іншими операторами. Тут також допустимі логічні операції: !=, <, > і т. ін.

if ... else. Лістинг 1.6 пояснює синтаксис, а лістинг 1.7 – приклад використання оператора `if ... else`.

Лістинг 1.6 – Синтаксис конструкції `if … else`

```
if (VarA> VarB){
//Код, який повинен виконуватися, якщо
VarA більше, ніж VarB
}
  else {
  // Код, який повинен виконуватися,
якщо VarA не більше, ніж VarB
  }
```

Лістинг 1.7 – Приклад використання оператора `if … else`

```
//Конструкція  if... else...
int x;
void setup() {
 Serial.begin(9600);
 Serial.println("If and Else ");
 Serial.println();
}
void loop() {
  if(x==10){
    Serial.println("Змінна «x» почала
дорівнювати 10!");
    while(1);
  }
  else{
    Serial.print("X = ");
    Serial.println(x);
  }
  x++;
}
```

Конструкцією `else` можна реалізувати різні альтернативні варіанти. Програма виводить значення змінної `x`, поки воно не досягне 10. Коли змінна `x` буде

дорівнювати 10, запускається фрагмент програми між дужками

else if {}. Можливість багаторазового чергування команд дає оператор `else if`. Тут можуть запитуватися різні стани змінних. Залежно від логічного значення (`true` або `false`) в умовному операторі `else if` виконується відповідна гілка програми. Синтаксис пояснює лістинг 1.8.

Лістинг 1.8 – Синтаксис конструкції `else if`

```
if (VarA != VarB) {
//Код, що виконується
}
else if (VarA == VarB) {
//Код, що виконується
}
else (VarA > VarB) {
//Код, що виконується
}
```

Лістинг 1.9 показує, як залежно від значення змінної x виводиться різний текст.

Лістинг 1.9 – Приклад розгалуження програм за допомогою конструкції `else.. if`

```
//Конструкція else if..
int x;
void setup() {
 Serial.begin(9600);
 Serial.println("Else If ");
 Serial.println();
}
void loop() {
  if(x==65){
    Serial.println("Змінна «x» почала
дорівнювати 65!");
```

```
        while(1);
      }
      else if(x==10){
        Serial.println("Змінна «x» почала
дорівнювати 10!");
        while(1);
      }
        else{
          Serial.print("X = ");
          Serial.println(x);
        }
      x++; }
```

Switch case. Конструкція `switch case` діє подібно оператору `else if`. Тут також залежно від логічного значення відповідного виразу буде виконано відповідний розділ коду. За замовчуванням можна задати альтернативний варіант за відсутності відповідності умов усередині оператора `case`. Вихід з оператора `case` здійснюється за допомогою оператора `break` (лістинг 1.10). Приклад наведений у лістингу 1.11.

Лістинг 1.10 – Синтаксис конструкції `switch case`

```
Switch (Variable) {
case 1:
//Код, який повинен виконуватися
case 1:
break;
//Код, який повинен виконуватися
break;
default:
//Альтернативний код, який виконується,
якщо всі інші умови не виконуються
}
```

Лістинг 1.11 – Приклад використання конструкції switch case

```
   //Switch Case
   int x;
   void setup() {
    Serial.begin(9600);
    Serial.println("Конструкція      Switch
Case");
    Serial.println();
   }
   void loop(){
     switch(x)
     {
        case 10:
        Serial.println("значення        «x»
досягло 10");
        break;
        case 20:
        Serial.println("значення        «x»
досягло 20");
        break;
        case 30:
        Serial.println("значення        «x»
досягло 30");
        while(1);
        break;
       default:
        Serial.print("X = ");
        Serial.println(x);
     }
     x++;  }
```

В основному циклі значення змінної x збільшується на одиницю. За певної умови в команді switch у термінал буде виведено повідомлення.

1.3.8 Цикли

Під час програмування часто потрібні цикли, наприклад, щоб реалізувати десятковий, двійковий лічильники або змусити програму нескінченну кількість разів повторювати виконання деякого фрагмента. Цикл підходить також для організації зчитування даних із послідовного інтерфейсу. Існує кілька типів циклів, у кожного з яких є свої особливості, з ними ми й ознайомимося.

for. Цикл `for` рахує змінну вгору або вниз у межах одного зазначеного діапазону значень. У цьому разі може здаватися певна величина наростання (інкремент). Лістинг 1.12 ілюструє синтаксис циклу for, а лістинг 1.13 – приклад його використання.

Лістинг 1.12 – Синтаксис циклу `for`

```
    for (Умова початку; Умова закінчення;
Лічильник) {
      //блок програми
    }
```

Лістинг 1.13 – Приклад використання циклу `for`

```
    // Цикл For
    int x;
    void setup(){
      Serial.begin(9600);
      Serial.println("For Construction");
      Serial.println();
    }
    void loop(){
      Serial.println("Збільшення на 1");
      for(x=0;x<11;x++){
         Serial.print("X = ");
```

```
      Serial.println(x);
   }

   Serial.println("Збільшення на 2");
   for(x=0;x<11;x=x+2) {
      Serial.print("X = ");
      Serial.println(x);
   }
   Serial.println("Зменшення змінної від
10 до 1 з кроком 1");
   for(x=10;x!=0;x--) {
      Serial.print("X = ");
      Serial.println (x);
   }
   // Закінчення програми
   while(1);
}
```

Програма містить три цикли `for` і пояснює принцип роботи за допомогою виведення показань лічильника. Щоб рахувати до 10, потрібно вказувати максимальне значення лічильника циклу, що дорівнює 11. Таким чином, цикл працює, поки значення лічильника менше від 11. Якщо задати максимальне значення, яке дорівнює 10, то цикл не відпрацює останнього кроку.

while* і *do while. Ще два варіанти циклу – це `while` і `do while`. Оператори в тілі циклу `do while` виконуються один раз, потім перевіряється умова закінчення циклу, тобто цей цикл виконається, щонайменше, один раз. Оператор `while` часто застосовується для організації нескінченних циклів, проте цикл можна перервати за допомогою оператора `break`. Якщо необхідно перевірити умову перед ви виконанням операторів циклу, то потрібно використовувати оператор `while`. Спочатку проводиться перевірка, а потім виконуються оператори тіла циклу.

Лістинг 1.14 ілюструє синтаксис циклу `while`, а лістинг 1.15 – приклад його використання.

Лістинг 1.14 – Синтаксис циклів `while` і `do while`

```
//Нескінченні цикли
while (1) {
//Працювати за будь-яких умов
}
//Нескінченні цикли з умовою скасування
while (1) {
  Var ++;
  if (Var> 10) break;
}
// While
While (Var <10) {
  Var ++;
}
//Do While
do {
  Var ++;
} while (Var <10);
```

Лістинг 1.15 – Приклад використання циклів `while` і `do while`

```
// While & Do While
int X=0;
void setup(){
  Serial.begin(9600);
  Serial.println("Конструкції While та
Do While ");
  Serial.println();
}
void loop() {
  while(1) {
    X++;
```

```
    Serial.println (X);
    if(X>9) break;
  }

  X=0;
  Serial.println();
  while(X<10) {
    X++;
    Serial.println(X);
  }

  X=0;
  Serial.println();
// Do While
  do {
    X++;
    Serial.println(X);
  }while( X < 10 );
  while(1); //Кінець програми
}
```

1.3.9 Функції та підпрограми

У процесі програмування дуже часто потрібні функції. Вони роблять програму наочнішою, а також дозволяють реалізувати власні команди. Ви можете створити свої функції й процедури (підпрограми, які не повертають значень), що будуть потрібні знову й знову, і використовувати їх у майбутніх проєктах (принцип модульної побудови). Відмінність між функцією й процедурою (універсальна назва `Sub Routine` (підпрограма)) полягає в тому, що функція повертає значення, а процедура – ні. У функції може, наприклад, виконуватися математичний вираз, який повертає результат переданої змінної.

Підпрограма. Лістинг 1.16 ілюструє приклад підпрограми.

Лістинг 1.16 – Реалізація підпрограм

```
// Arduino Sub-Routinen
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino Sub-
Routinen");
   Serial.println();
}
void loop(){
   Issuance1();
   Issuance2();
   while(1);
}
void Issuance1(){
   Serial.println("Issuance 1");
}
void Issuance2(){
   Serial.println("Issuance 2");
}
```

Функції. Функція – це блок кодів програми, в яких є ім'я і ряд команд, які запускаються під час виконання функції. Приклади функцій `void setup {}` і `void loop {}` уже траплялися раніше. Існують також убудовані функції, що будуть розглянуті далі.

Писати власні функції має сенс, щоб спростити повторювані задачі й покращити структуру програми. У разі оголошення зазначають тип функції, він ідентичний типу даних, наприклад, `int` для типів `integer`. Якщо функція не повертає значення, її тип буде `void`. Крім типу, в разі оголошення функції задають ім'я і в дужках

перелічують усі параметри, які повинні їй передаватися. Лістинг 1.17 ілюструє приклад функції.

Лістинг 1.17 – Приклад реалізації функції

```
// Arduino Function
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino Function");
   Serial.println();
}
void loop(){
   int Sum;
   Sum = Addition(12,55);
   Serial.print("12 + 55 = ");
   Serial.println(Sum);

   while(1);
}
int Addition(int x, int y) {
   int sum;
   sum=x+y;
   return sum;
}
```

Continue. За командою continue решта поточного циклу (`do`, `for` або `while`) пропускається й запускається код після блоку `{}`. Лістинг 1.18 ілюструє приклад.

Лістинг 1.18 – Приклад використання команди `Continue`

```
// Arduino Continue
int i=0;
void setup() {
   Serial.begin(9600);
   Serial.println("Arduino Continue");
   Serial.println();
```

```
}
void loop(){
  for(i=0;i<10;i++) {
    if(i%2==0) {
      continue;
    }
    Serial.print(i);
    Serial.println(" Не ділиться на 2");
  }
   while(1);
}
```

Тут оператор continue завжди перериває цикл for, якщо змінна `i` не ділиться на 2 без залишку.

Функції перетворення типу. Функції `char (),byte (), int (), long ()` і `float ()` перетворять тип змінної. Таким чином, ми можемо, наприклад, з однобайтової змінної зробити змінну типу `Long`. Суть подібних операцій – наведення типу даних для подальших обчислень.

Ось перелік функцій перетворення типу:

- `Char ()` – перетворює значення на символ (`Charakter`).
- `Byte ()` – перетворює значення на `Byte`.
- `Int ()` – перетворює значення на `integer`.
- `Long ()` – перетворює значення на `Long`.
- `Float ()` – перетворює значення на `Float`.

Математичні функції

min (x, y). Розраховує мінімум із двох значень і повертає менше значення. Лістинг 1.19 ілюструє приклад використання цієї функції.

Лістинг 1.19 – Приклад використання функції `min(x,y)`

```
   // Arduino min(x,y) Function
   int x,y,Erg=0;
   void setup(){
      Serial.begin(9600);
      Serial.println("Arduino min(x,y)
Function");
      Serial.println();
   }
   void loop(){
       Erg=min(10,55);
       Serial.print(Erg);
       Serial.println();
      while(1);
   }
```

max {x, y). Розраховує максимум із двох значень і повертає число з більшим значенням. Лістинг 1.20 ілюструє приклад використання `max(x,y)`.

Лістинг 1.20 – Приклад використання функції `max(x,y)`

```
   // Arduino max(x,y) Function
   int x,y,Erg=0;
   void setup(){
      Serial.begin(9600);
      Serial.println("Arduino max(x,y)
Function");
      Serial.println();
   }
   void loop(){
       Erg=max(10,55);
       Serial.print(Erg);
       Serial.println();
      while(1);
   }
```

abs (x). Розраховує абсолютну величину числа. Лістинг 1.21 ілюструє приклад використання `abs(x)`.

Лістинг 1.21 – Приклад використання функції `abs(x)`

```
// Arduino abs(x) Function
int Erg;
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino abs(x,y)
Function");
   Serial.println();
}
void loop(){
    Erg=abs(3.1415);
    Serial.print(Erg);
   while(1);
}
```

constrain (x, a, b). Функція перевіряє, і якщо потрібно задає нове значення так, щоб воно було в області допустимих значень, заданій параметрами.

Параметри:

– `x` – значення, що перевіряється (будь-який тип);

– `a` – нижня межа області допустимих значень, будь-який тип;

– `b` – верхня межа області допустимих значень, будь-який тип.

Значення, що повертається:

– `x` – якщо x входить до області допустимих значень [a..b]

– `a` – якщо x менше, ніж a;

– `b` – якщо x більше, ніж b.

Лістинг 1.22 ілюструє приклад перевірки входження числа до заданого діапазону.

Лістинг 1.22 – Приклад використання функції constrain(x, a, b)

```
// Arduino constrain(x, a, b) Function
int x,Erg;
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino constrain(x,
a, b) Function");
}
void loop(){
    for(x=0;x<60;x++){
      Erg=constrain(x, 10, 50);
      Serial.println(Erg);
    }
   while(1);
}
```

map (x, fromLow, fromHigh, toLow, toHigh). Функція map – це корисна функція для перерахунку значень з одного діапазону в інший. Вона ідеально підходить, наприклад, для перетворення великої вхідної змінної на маленьку вихідну змінну. Лістинг 1.23 ілюструє приклад перерахунку з використанням функції map.

Лістинг 1.23 – Приклад використання функції map

```
// Arduino map(x, fromLow, fromHigh,
toLow, toHigh) Function
int x, Erg;
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino Map
Function");
   Serial.println();
}
void loop(){
```

```
        for(x=0;x<20;x++){
          Erg=map(x,0,20,5,15);
          Serial.println(Erg);
        }
       while(1);
   }
```

pow (base, exponent). Функція `pow` видає результат піднесення до степеня першого значення аргументу з другим значенням аргументу (перше значення аргументу – число, друге – степінь). У лістингу 1.24 обидва аргументи й результат мають тип `float`.

Лістинг 1.24 – Приклад піднесення числа до степеня функцією `pow(base,exponent)`

```
   // Arduino pow(base,exponent) Function
   int Erg;
   void setup() {
      Serial.begin(9600);
      Serial.println("Arduino
pow(base,exponent) Function");
      Serial.println();
   }
   void loop(){
       Erg=pow(10,5);
       Serial.println(Erg);
      while(1);
   }
```

sq (x). Функція обчислює квадрат аргументу. Лістинг 1.25 ілюструє приклад роботи функції `sq(x)`.

Лістинг 1.25 – Приклад піднесення числа до степеня 2 функцією `sq(x)`

```
// Arduino sq(x) Function
int Erg;
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino sq(x)
Function");
   Serial.println();
}
void loop(){
    Erg=sq(9);
    Serial.print(Erg);
    Serial.println();
   while(1);
}
```

sqrt (x). Ця функція, обернена до `sq(x)`, вона обчислює квадратний корінь числа. Лістинг 1.26 ілюструє приклад її роботи.

Лістинг 1.26 – Приклад обчислення квадратного кореня числа функцією `sqrt(x)`

```
// Arduino sqrt(x) Function
int Erg;
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino sqrt(x)
Function");
   Serial.println();
}
void loop(){
    Erg=sqrt(9);
    Serial.print(Erg);
    Serial.println();
```

```
        while(1);
    }
```

sin (rad), ***cos (rad), tan (rad)***. Функції розраховують синус, косинус і тангенс числа відповідно. Аргумент зазначають у радіанах. Значення, що повертається, – це синус, косинус або тангенс аргументу. Лістинг 1.27 ілюструє приклад використання цих функцій.

Лістинг 1.27 – Приклад визначення `sin(x), cos(x), tan(x)`

```
    // Arduino sin(x), cos(x), tan(x)
Function
    float ErgS, ErdC, ErgT, x=1.0;
    void setup(){
        Serial.begin(9600);
        Serial.println("Arduino sin(x)
Function");
        Serial.println();
    }
    void loop(){
         ErgS=sin(x);
         ErgC=cos(x);
         ErgT=tan(x);

         Serial.print("Sin ");
         Serial.print(x);
         Serial.print(" = ");
         Serial.println(ErgS);
         Serial.print("Cos ");
         Serial.print(x);
         Serial.print(" = ");
         Serial.println(ErgC);
         Serial.print("Tan ");
         Serial.print(x);
```

```
    Serial.print(" = ");
    Serial.println(ErgT);
   while(1);
}
```

1.4 Прикладне прорамування в середовищі Arduino

1.4.1 Послідовний інтерфейс уведення / виведення

Зв'язок через інтерфейс UART застосовують досить часто. Мікроконтролер може посилати, а також приймати дані від комп'ютера або інших мікроконтролерів.

В Arduino для цього існує кілька команд. Деякі вже траплялися у попередніх прикладах, наприклад, `Serial.print ()` і `Serial.println ()`. Мікроконтролер має вбудований апаратний інтерфейс UART. Універсальний асинхроннний приймач-передавач (UART – Universal Asynchronous Receiver Transmitter) можна також імітувати за допомогою програмного забезпечення. Емулятор є не настільки швидкісним, як апаратний інтерфейс, але все-таки дозволяє встановлювати одночасне підключення до декількох станцій. На рисунку 1.13 зображений апаратний інтерфейс UART, реалізований в Arduino.

В Arduino команда `Serial.println` посилає послідовність символів (так званий рядок) через інтерфейс UART. «Невидимі» символи CR (Carriage Return) і LF (LineFeed) додаються автоматично, вони позначають кінець рядка. Якщо перехід на інший рядок не потрібний, то необхідно вказати команду `Serial.print`. Якщо в `Serial.print` передається число, воно автоматично перетворюється на текст. Передається не число, а ASCII-код для цього числа. Наприклад, число 42 складається з двох символів 4 і 2. Будуть передані два ASCII-коди для

символів «4» і «2» і ASCII-коди обох службових символів «CR» і «LF». Важливо, щоб приймачі й передавачі були завжди встановлені на однакову швидкість у бодах.

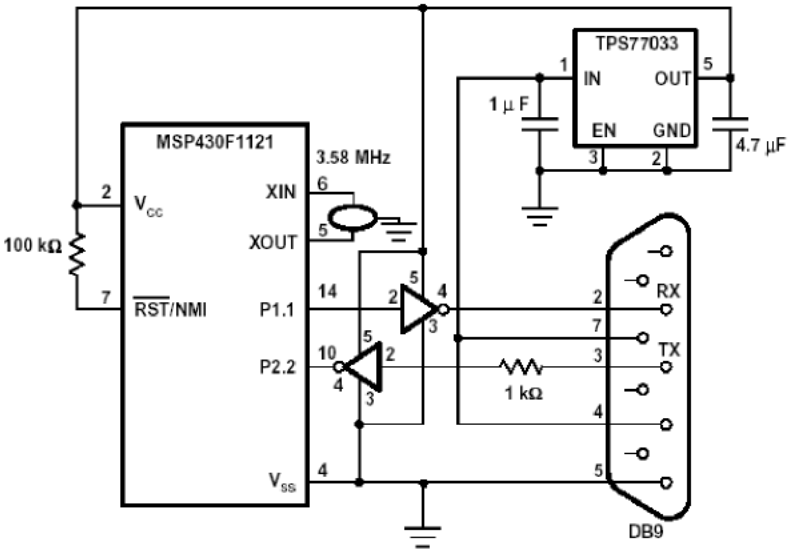


Рисунок 1.13 – Апаратний інтерфейс UART мікроконтролера

Функції роботи з послідовним інтерфейсом. *Serial.begin (Baudrate)*

Функція `Serial.begin()` задає швидкість послідовного інтерфейсу в бодах. Бод позначає число бітів за 1 секунду. Якщо ми вибираємо 9 600 бодів, один біт передається за 1/9600 = 0,000104 с = 104 мкс.

Функція `Serial.begin (Baudrate)` відкриває послідовний порт і встановлює швидкість у бодах (швидкість передавання даних) для послідовного передавання даних. Можливі такі стандартні швидкості в бодах:

– 300;

– 1 200;
– 4 800;
– 9 600;
– 14 400;
– 19 200;
– 28 800;
– 38 400;
– 57 600;
– 115 200.

У разі використання плати Arduino Mega або іншої з декількома апаратними інтерфейсами UART конфігурацію можна задати таким чином:

```
Serial.begin (9600);
Seriall.begin (38400);
Serial2.begin (19200);
Serial3.begin (4800).
```

Усі інтерфейси мають номер. Перший інтерфейс – UART0. Якщо потрібно вивести символ, то в цьому разі застосовують функцію `SerialX.print()`, де `X` відповідає номеру інтерфейсу UART.

```
Serial.println ("Hallo, this is UART 0");
Seriall.println ("Hallo, this is UART 1");
Serial2.println ("Hallo, this is UART 2");
Serial3.println ("Hallo, this is UART 3");
```

У разі послідовної комунікації цифрові виводи 0 (RX) і 1 (TX) не можуть використовуватися одночасно.

Serial.end (). Для завершення роботи з послідовним інтерфейсом і використання виводів за іншим призначенням можна застосовувати функцію `Serial.end()`.

Serial.read (). Функція `Serial.read()` зчитує один байт із послідовного інтерфейсу:

```
int x;
x = Serial.read();
```

Serial.available (). Функція `Serial.available()` вказує, чи є дані в буфері послідовного порту. Ця функція дозволяє, наприклад, пропустити блок програми, якщо в буфері немає ніяких даних. Ця функція дуже важлива для роботи з послідовним інтерфейсом. У лістингу 1.28 показано, як ця функція працює на практиці.

Лістинг 1.28 – Приклад використання функції `Serial.available()`

```
// Arduino Serial.available Function
byte eingabe, ausgabe;
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino
Serial.available Function");
   Serial.println();
}
void loop(){
   if (Serial.available() > 0){
      eingabe=Serial.read();
      Serial.print("були отримані такі
дані: ");
      ausgabe=char(eingabe);
      Serial.println(eingabe);
   }
}
```

Serial.flush (). Ця функція видаляє вміст послідовного буфера. Застосовують її, щоб очистити буфер після розподілу послідовних даних. Якщо виникає помилка комунікації й дані пошкоджуються, то вони видаляються з буфера.

Serial.print (). Функція `Serial.print()` виводить дані з послідовного буфера передавання. Допустимі типи: `Integer`, `Byte`, `Char` і `Float`.

Функція serial .print (x) без зазначення формату виводить десяткове число з буфера UART:

```
int b = 79;
Serial.print (b);
```

Виводить число 79 з буфера UART.

Функція serial.print (b, DEC) із зазначенням формату DEC виводить число як ASCII-рядок із буфера UART:

```
int b = 79;
Serial.print (b, DEC);
```

Виводить ASCII-рядок «79» з буфера UART.

Функція Serial .print (b, Hex) із зазначенням формату Hex виводить із буфера

UART число як ASCII-рядок у шістнадцятковому форматі:

```
int b = 79;
Serial.print (b, Hex);
```

Виводить ASCП-рядок «4F» із буфера UART.

Функція Serial.print (b, OCT) із зазначенням формату OCT виводить із буфера

UART число як ASCII-рядок у вісімковому форматі:

```
int b = 79;
Serial.print (b, OCT);
```

Виводить ASCП-рядок «117» із буфера UART.

Функція Serial.print (b, BIN) із зазначенням формату Bin виводить з буфера

UART число як ASCII-рядок у двійковому форматі:

```
int b = 79;
Serial.print (b, BIN);
```

Виводить ASCII-рядок «1001111» із буфера UART.

Функція serial.print (b, Byte) із зазначенням формату Byte виводить із буфера

UART число у вигляді окремого байта:

```
int b = 79;
```

```
Serial.print (b, Byte);
```

Виводить ASCII-символ «O» з буфера UART.

Лістинг 1.29 ілюструє приклади виведення в різних форматах.

Лістинг 1.29 – Виведення чисел у різних форматах

```
// Arduino Serial.print Function
int x;
void setup(){
   Serial.begin(9600);
   Serial.println("Arduino  Serial.print
Function");
   Serial.println();
}
void loop(){
  Serial.print("NO FORMAT");
  Serial.print("\t");  // Друк табуляції
  Serial.print("DEC");      // Десяткове
  Serial.print("\t");
  Serial.print("HEX");
// шістнадцятирічне
  Serial.print("\t");
  Serial.print("OCT");    // Восьмерічне
  Serial.print("\t");
  Serial.print("BIN");      // Двійкове
  Serial.print("\t");
  Serial.println("BYTE");   // Типу Byte
  // Розглядається лише частина таблиці
ASCII
  for(x=0; x< 64; x++){
    // Виведення даних у різних форматах
    Serial.print(x);
    Serial.print("\t");
    Serial.print(x, DEC);
    Serial.print("\t");
```

```
      Serial.print(x, HEX);
      Serial.print("\t");
      Serial.print(x, OCT);
      Serial.print("\t");
      Serial.print(x, BIN);
      Serial.print("\t");
      Serial.println(x, BYTE);
      delay(200);        //Затримка 200 мс
    }
    Serial.println("");
  }
```

Serial.println (). Функція виводить дані з послідовного порту і здійснює автоматитичний перехід на новий рядок.

Serial. write (). Функція `Serial.write()` здійснює побайтове виведення даних із буфера послідовного порту:

```
Serial.write (val);
Serial.write (str);
Serial.write (buf, len);
```

Значення параметрів:

- `val` – посилає окремий байт;
- `str` – посилає рядок побайтово;
- `buf` – посилає масив побайтово;
- `len` – довжина масиву (розмір буфера).

Лістинг 1.30 ілюструє приклад використання функції `Serial.write()`.

Лістинг 1.30 – Виведення даних у послідовний інтерфейс за допомогою функції `Serial.write()`

```
// Arduino Serial.write Function
byte val = 65;
char str[] = "Test";
byte buf[] = {'H','a','l','l','o'};
byte len = 3;
void setup(){
```

```
      Serial.begin(9600);
      Serial.println("Arduino Serial.write
Function");
      Serial.println();
   }
   void loop(){
     Serial.println("Символ ASCII");
     Serial.write(val);
     Serial.println();
     Serial.println("String 1");
     Serial.write(str);
     Serial.println();

     Serial.println("String 2");
     Serial.write(buf, len);
     Serial.println();
     while(1);
   }
```

У функції `Serial.write (buf, len)` була зазначена змінна len така, що дорівнює «3». Отже, будуть виведені лише перші три символи.

1.4.2 Особливості роботи послідовного інтерфейсу

Передавання пакета починається зі стартового біта (низький рівень), далі йдуть 8 біт даних. Безпосередньо після бітів даних передається біт перевірки і, нарешті, стоповий біт (рис. 1.14).

Зчитування рядка символів через послідовний інтерфейс. Вам уже відомо, як можуть передаватися і братися окремі символи і ланцюжки символів (рядки). Для взаємодії та справжнього введення даних слів цього ще не достатньо. Потрібно написати програму, що дозволяє

використовувати будь-які рядки або блоки даних (лістинг 1.31).

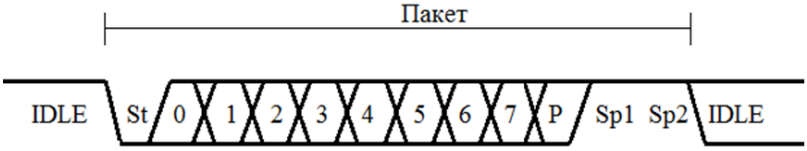


Рисунок 1.14 – Вигляд передавання даних через інтерфейс UART:
St –стартовий біт, завжди низький рівень;
n – біти даних (від 0 до 8);
P – біт перевірки;
Sp – стоповий біт, завжди високий рівень;
IDLE – немає передачі по комунікаційній лінії (RxDn або TxDn). За стану IDLE лінія може мати високий рівень

Лістинг 1.31 – Зчитування рядка символів з послідовного інтерфейсу

```
   // Arduino Serial Read
   #define INLENGTH 20
   #define INTERMINATOR 13
   char inString[INLENGTH+1];
   int inCount;
   void setup(){
      Serial.begin(9600);
      Serial.println("Arduino        Serial
read");
      Serial.println();
      Serial.println("Введіть   текст   із
максимум 20 символів: ");
   }
   void loop(){
         inCount = 0;
         do {
           while (Serial.available()==0);
```

```
            inString[inCount]              =
Serial.read();

if(inString[inCount]==INTERMINATOR) break;
          }while(++inCount < INLENGTH);
          inString[inCount] = 0;
          Serial.print(inString);
   }
```

Програма чекає появи символу в буфері та зчитує його в масив inString. Якщо було взято більше 20 символів або символ CR, масив виводиться в термінал.

Послідовне виведення даних з обчисленням. Практичний приклад роботи з послідовним інтерфейсом ілюструє лістинг 1.32 – невелика програма, яка перераховує градуси за Цельсієм у градуси за Фаренгейтом і навпаки.

Лістинг 1.32 – Виведення даних з обчисленням

```
   // Arduino Gradus
   float Grad = 25.5;
   float Fahrenheit = 88.2;
   void setup(){
      Serial.begin(9600);
      Serial.println("Arduino   Gradus   to
Fahrenheit converter ");
      Serial.println();
   }
   void loop(){
      Serial.print(Grad);
      Serial.print(" Gradus ");

Serial.print(Grad_to_Fahrenheit(Grad));
      Serial.println(" Fahrenheit");
      Serial.println();
```

```
      Serial.print(Fahrenheit);
      Serial.print(" Fahrenheit");

Serial.print(Fahrenheit_to_Grad(Fahrenheit
));
      Serial.println(" Grad");
      Serial.println();
      while(1);
   }
   float Grad_to_Fahrenheit(float grad){
      float erg;
      erg = grad * 9;
      erg = erg / 5;
      erg = erg + 32;
      return erg;
   }
   float           Fahrenheit_to_Grad(float
fahrenheit){
      float erg;
      erg = fahrenheit - 32 ;
      erg = erg * 5 ;
      erg = erg / 9;
      return erg;
   }
```

1.4.3 Програмна емуляція UART

Коли виникає необхідність використання декількох послідовних пристроїв, а в мікроконтролері є лише один апаратний інтерфейс UART, середовище розроблення Arduino пропонує можливість програмної емуляції інтерфейсу UART. Під час програмної реалізації UART використовуються виводи 2 і 3. Дані зчитуються в стек розміром 64 байти. Недоліком програмного інтерфейсу

UART є те, що для цього потрібні додаткові системні ресурси.

Обмеження під час застосування програмного інтерфейсу UART:

– максимальна швидкість передавання – 9 600 бодів;

– відсутня функція `serial.available();`

– функція `serial.read ()` чекає того часу, поки не з'являться дані в буфері;

– функція `serial.read ()` повинна запускатися в циклі, якщо функція не викликається, а дані в цей час надходять, вони зникають.

У бібліотеці UART від Arduino є такі функції: `softwareserial(), begin(), read(), print(), println()`.

Лістинг 1.33 ілюструє приклад конфігурації послідовного інтерфейсу UART.

Лістинг 1.33 – Приклад програмної реалізації інтерфейсу UART

```
   // Arduino Software
UART
   #include <SoftwareSerial.h>
     //Підключення бібліотеки Software UART
   #define rxPin 2
   #define txPin 3
   #define ledPin 13

   // Налаштування програмного UART
   SoftwareSerial mySerial =
SoftwareSerial(rxPin, txPin);
   byte pinState = 0;
   void setup() {
   // Конфігурація виходів
     pinMode(rxPin, INPUT);
```

```
    pinMode(txPin, OUTPUT);
    pinMode(ledPin, OUTPUT);
    mySerial.begin(9600);
  }
  void loop(){
    char someChar = mySerial.read();   //
Прослуховування порту
    mySerial.print(someChar);     //
Виведення символу на екран
    toggle(13);    //Зміна          стану
світлодіода
  }
  void toggle(int pinNum){
    digitalWrite(pinNum, pinState);
    pinState = !pinState;
  }
```

1.4.4 Конфігурація входу / виходу та налаштування порту

Щоб чип Arduino знав, який вивід ми хотіли б використовувати як вхід або вихід, потрібно зазначити це в підпрограмі `void setup()`. Далі ми розглянемо, як це зробити. Якщо така конфігурація не виконується, то всі виводи мікроконтролера після ввімкнення встановлюються у високоімпедансний стан (Z-стан).

pinMode (pin, mode). Функція використовується в програмі `void setup()`, щоб конфігурувати контакт плати як вхід або як вихід:

```
pipMode (pin, OUTPUT); // контакт
встановлюється як вихід
```

У чипі ATmega також є вбудований підтягувальний резистор (20 кОм), яким керують за допомогою програмного забезпечення. Вбудований підтягувальний резистор можна використовувати так:

```
   pinMode    (pin,    INPUT);    //Контакт
встановлюється як вхід
   digitalWrite        (pin,        HIGH);
//Під'єднується підтягувальний резистор
```

Підтягувальний резистор зазвичай під'єднують, щоб приєднувати входи як комутатор. Бачимо, що контакт конфігурується як вхід, але на ньому встановлюється високий рівень. Це лише метод активізувати внутрішній підтягувальний резистор.

Виводи, що конфігуруються як вихід, мають малий повний опір і можуть навантажуватися приєднаними елементами та схемами з струмом максимум 40 мА. Цього вистачить для свічення світлодіода (з урахуванням послідовно ввімкненого опору), але недостатньо, щоб експлуатувати більшість типів реле, соленоїдів або електродвигунів. Короткі замикання виводів плати Arduino, а також занадто великі струми можуть пошкодити вихідний контакт і навіть вивести з ладу мікроконтролер. Тому краще підключати зовнішні компоненти до виходу через резистор 470 Ом або 1 кОм.

digitalRead (pin). Функція `digitalRead ()` зчитує значення заданого цифрового виводу з перезультатом HIGH або LOW, що відповідає 1 або О. Номер виводу установлюють або як змінна, або як константа (від 0 до 13).

```
   value = digitalRead (Pin); //value
присвоюється значення таке, що дорівнює
значенню на вхідному виводі Pin
```

digitalWrite (pin, value). Встановлює рівень HIGH або LOW на заданому виводі. Номер виводу може задаватися або як змінна, або як константа (від 0 до 13).

```
   digitalWrite (pin, HIGH); // встановлює
на контакті високий рівень (+5 В)
```

1.4.5 Зчитування стану кнопки

Кнопка з підтягувальним резистором. У наступному прикладі (лістинг 1.34) зчитується стан кнопки, під'єднаної до цифрового входу (контакт 12), і результат відображається за допомогою світлодіода. Під час натискання на кнопку світлодіод гасне, а під час відпускання – загоряється. Тут справа в тому, що ми активізували внутрішній підтягувальний резистор і, таким чином, наш вхідний вивід 12 має високий рівень. Якщо ми натискаємо на кнопку, яка під'єднує вивід до GND (загальний провід), світлодіод не горить, оскільки на вході тепер низький рівень. Такого самого результату можна досягти і з використанням зовнішнього підтягувального резистору номіналом близько 10 кОм між виводом і джерелом напруги VCC (+5В), водночас не потрібно примусово встановлювати високий рівень сигналу.

Лістинг 1.34 – Зчитування стану кнопки з підтягувальним резистором

```
//IO-Pin Tester
int led=13;
int pin=12;
int value=0;
void setup() {
  pinMode(led,OUTPUT);
  pinMode(pin,INPUT);
  digitalWrite(pin, HIGH);
}

void loop() {
  value=digitalRead(pin);
  digitalWrite(led,value);
}
```

Вивід контролера ATmega витримує струм до 40 мА. Загальне навантаження на мікроконтролер (залежно від типу корпусу) не може бути більше ніж 200 мА. Виводи можуть відрізнятися один від одного. Якщо потрібні точні дані, завжди дивіться паспортні дані.

Кнопка з узгоджувальним резистором. У попередньому прикладі кнопка була під'єднана до контакту з внутрішнім підтягувальним резистором. Під час натискання кнопки контакт з'єднувався з GND (загальною шиною). У наступному прикладі (лістинг 1.35) показано, як можна використовувати кнопку через напругу живлення VCC (+5 В). Так як тут не ввімкнений підтягувальний резистор, то потрібно додати зовнішній узгоджувальний резистор між виводом, до якого під'єднана кнопка і GND, щоб вивід не опинився в невизначеному стані. Номінал резистора повинен бути близько 10 кОм.

Лістинг 1.35 – Зчитування стану кнопки з узгоджувальним резистором

```
//Кнопка з узгоджуючим резистором
int led=13;
int pin=12;
int value=0;
void setup(){
  pinMode(led,OUTPUT);
  pinMode(pin,INPUT);
}
void loop(){
  value=digitalRead(pin);
  digitalWrite(led,value);
}
```

1.4.6 Введення аналогових даних та АЦП

Для вимірювання аналогової напруги плата мікроконтролера Arduino має внутрішній аналого-цифровий перетворювач (ADC – Analog Digital Converter). Вбудований АЦП 10-розрядний, тобто крок зміни дорівнює значенню аналогової напруги 0,0048 В за опорної напруги U_{ref} = 5 В. Плата мікроконтролера має шість аналогових входів, але лише один внутрішній АЦП. У такому разі канали будуть перемикатися, це називають мультиплексна схема. З цієї простої формули можна легко розрахувати роздільну здатність:

$$U_{step} = U_{ref} / \text{розрядність}.$$
$$U_{step} = 5 \text{ В} / 1024 = 0{,}0048 \text{ В}.$$

Якщо потрібно дізнатися цифрове значення, то можна розрахувати його за формулою

Значення = 1024 · (напруга на вході АЦП) / U_{ref} .

Точність вимірювання становить ±2 кроки. За опорної напруги 5 В точність перетворення становить ±0,0097 В. Це означає, що АЦП вимірює прикладену напругу з точністю до двох цифр після коми. Зрозуміло, що опорна напруга повинна бути стабільною, оскільки від неї напряму залежить точність вимірювання.

analogRead (pin)

Функція зчитує значення встановленого аналогового входу з роздільною здатністю 10 біт. Функція допустима лише для виводів 0–5 (маємо на увазі Arduino Uno):

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

Аналогові виводи, на відміну від цифрових, не потрібно зазначати як вхід або вихід на початку програми.

Розглянемо невелику програму (лістинг 1.36) для

вимірювання напруги на вході 0 АЦП (ANALOG IN O). Для регулювання напруги використовуємо змінний резистор між виводами A0 і VCC.

Лістинг 1.36 – Зчитування аналогових даних

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

Під час обертання ручки потенціометра буде видно, що значення на виході АЦП змінюється від 0 до 1 023, залежно від положення ручки. Максимуму в 1 023 може і не бути досягненим, оскільки деякі потенціометри мають невеликий залишковий опір. Для зручного подання даних у вольтах, а не в бітах, можна виконати перетворення, додавши до коду вище такі команди:

```
float voltage = value * (5.0 / 1023.0);
Serial.println(voltage);
```

1.4.7 Аналоговий вихід. ШІМ

На платі Arduino розміщені шість виходів із широтно-імпульсною модуляцією (ШІМ): контакти 3, 5, 6, 9, 10 і 11. Вони можуть використовуватися для цифро-аналогового перетворення, управління сервоприводами або для формування звукових сигналів. У процесі ШІМ (PWM – Pulse Width Modulation) змінюється прогальність

імпульсної послідовності. Прогальність свідчить про співвідношення тривалості ввімкненого стану до періоду повторювання імпульсів. Водночас частота й рівень сигналу залишаються завжди однаковими. Змінюється лише тривалість переходу від високого до низького рівня (рис. 1.15).

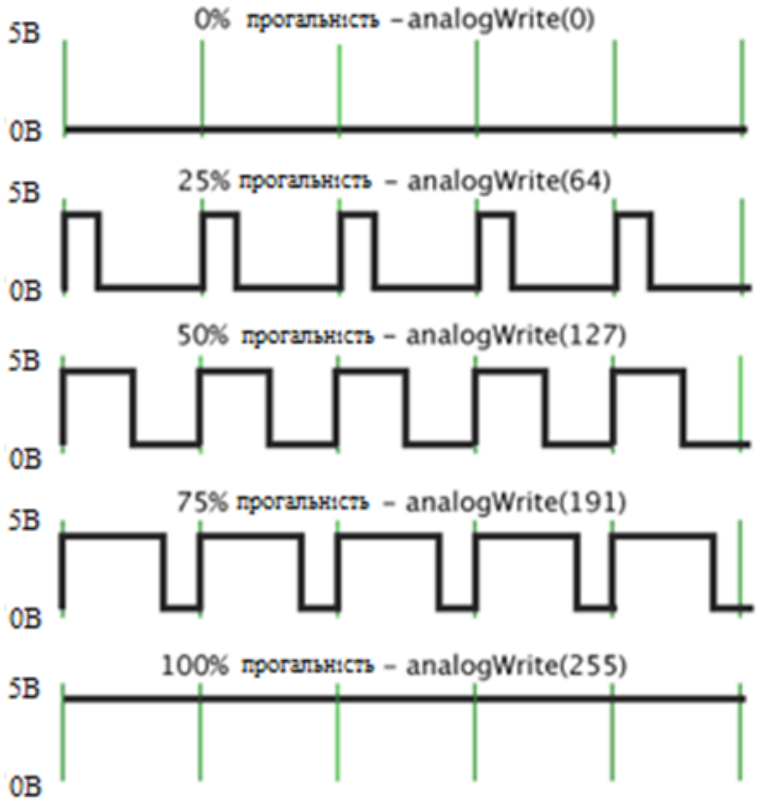


Рисунок 1.15 – Ілюстрація реалізації функції прогальності

analogWrite (pin, value). Ця команда формує на виході псевдоаналогові значення за допомогою модуляції ширини імпульсної послідовності (ШІМ). Значення може

встановлюватись у вигляді змінної або константи в діапазоні 0–255:

```
   analogWrite (pin, value); // Запис
значення до аналогового виводу
```

За нульового значення напруга на заданому аналоговому виході теж буде нульовою. Значення 255 відповідає напрузі 5 В. За значень між 0 і 255 вивід перемикається між 0 В і 5 В, чим більше значення, тим довше вивід має високий рівень (5 В). За значення 64 на виводі три чверті періоду буде 0 В і одну чверть – 5 В. Значення 127 призводить до того, що вихідна напруги одну половину періоду часу має високий рівень, а другу – низький рівень. Під час значення 191 напруга на виводі впродовж однієї чверті періоду становить О В а три чверті – повні 5 В. Оскільки ця функція базується на апаратних засобах, то постійний сигнал запускається незалежно від програми, і рівень сигналу на виводі буде зберігатися до наступного виклику функції `analogWrite()` (чи до виклику `digitalRead()` або `digitalWrite()` для того самого виводу).

Аналогові виводи, на відміну від цифрових, не потрібно попередньо налаштовувати як вхід або вихід.

У прикладі (лістинг 1.37) два світлодіоди перемикаються з високою частотою. Внаслідок цього значення ШІМ визначає яскравість світіння.

Лістинг 1.37 – Приклад використання ШІМ

```
// Analog Write
int value;
int LEDgr=10;
int LEDrd=11;
void setup() {
```

```
   //У цій програмі нічого не потребує
попередньої ініціалізації
   }
   void loop(){
    for(value=0;value<255;value++) {
      analogWrite(LEDgr, value);
      analogWrite(LEDrd, 255-value);
      delay(5);
    }
    delay(1000);
    for(value=255;value!=0;value--) {
      analogWrite(LEDgr, value);
      analogWrite(LEDrd, 255-value);
      delay(5);
    }
    delay(1000);
   }
```

Якщо приєднати невеликий п'єзоакустичний перетворювач до аналогового виходу, то можна «почути» ШІМ-сигнал. Хоча звук буде не дуже гучним.

1.4.8 Деякі спеціальні функції

Введення паузи за допомогою *delay*. У попередніх прикладах ми вже неодноразово задавали паузу за допомогою функції `delay()`.

delay (ms). Зупиняє програму на час, зазначений у мілісекундах, причому значення 1 000 відповідає 1 секунді:

```
   delay (1000); // затримка на одну
секунду
```

delayMicroseconds(). Зупиняє програму на час, зазначений у мікросекундах, причому значення 1 000 відповідає 1 мілісекунді:

```
   delayMicroseconds (1000); // затримка
на одну мілісекунду
```

Функції випадкових чисел. Під час написання вимірювальних, керувальних, регулювальних або ігрових програм часто потрібні випадкові числа, наприклад, якщо в будинку в довільний час повинно запалюватися та гаснути освітлення. Для цієї мети передбачена функція `random()`.

randomSeed (seed). Функція встановлює значення або початкове число для `random()`:

```
randomSeed (value); // встановлює значення value як випадкове початкове число
```

Застосування `randomSeed ()` гарантує кращий результат. Випадкові числа можна передати як аргумент, наприклад, функцій `millis()` (див. далі) або `analogRead()`, щоб згенерувати електричні перешкоди на аналоговому виводі.

random (min, max). Функція `random` здійснює генерацію псевдовипадкових значень у межах певного діапазону мінімального та максимального значень:

```
value = random (100, 200); // встановлює value випадковим числом у діапазоні між 100 і 200
```

Використовуйте `random (min, max)` після функції `randomSeed ()`.

Лістинг 1.38 ілюструє приклад генерації випадкових чисел.

Лістинг 1.38 – Приклад використання функції `random`

```
// random numbers
int x,y=0;
void setup(){
  randomSeed(100);
  Serial.begin(9600);
```

```
    Serial.println("Arduino random
numbers ");
    Serial.println();
  }
  void loop(){
    for(x=0;x<20;x++)
    {
      y=random(0, 10);
      Serial.print(y);
      Serial.print(",");
    }
    Serial.println();
    for(x=0;x<20;x++){
      y=random(10,100);
      Serial.print(y);
      Serial.print(",");
    }
    Serial.println();
    for(x=0;x<20;x++){
      y=random(0,x+1);
      Serial.print(y);
      Serial.print(",");
    }
    Serial.println();
    while(1);
  }
```

Ось можливі результати виконання лістингу 1.38:

0, 9, 5, 5, 9, 3, 2, l, l, 9, 4, 3, 9, 9, 5, 6, l, 0, 4, 8

83, 24, 24, 99, 92, 36, 97, 35, 13, 10, 43, 98, 88, 52, 89, 86, 29, 35, 37, 58

0, 0, 1, 3, 0, 1, 1, 4, 6, 9, 7,3, 1, 3, 5, 8, 9, 9, 17, 18

Запустіть програму кілька разів і переконайтеся, що завжди видаються різні числа. Але результати завжди будуть розміщуватися в заданому діапазоні від

мінімального до максимального значення.

1.4.9 Вимірювання часових інтервалів

Щоб визначити, скільки минуло часу після запуску програми або виконання якоїсь операції, в Arduino є спеціальні функції. Зрозуміло, що можна самому створювати цикли з зупиненням усієї програми та очікуванням,, до того часу, поки не закінчиться необхідний інтервал часу. Час може визначатися в мілісекундах або мікросекундах.

millis (). Функція повертає значення часу (в мілісекундах), що минув після запуску програми:

```
value = millis (); // повертає інтервал часу в мілісекундах
```

Змінна типу `unsigned long` переповнюється приблизно через 50 днів і обнуляється. Лістинг 1.39 ілюструє приклад використання цієї функції.

Лістинг 1.39 – Вимірювання часу функцією `millis ();`

```
// timekeeping
unsigned long value;
void setup(){
  Serial.begin(9600);
  Serial.println("Arduino timekeeping
");
  Serial.println();
}
void loop(){
  Serial.print("Time: ");
  value=millis();
  Serial.println(value);
  delay(1000);
}
```

micros (). Функція повертає інтервал часу (в мікросекундах) із моменту запуску програми:

```
value = micros (); // повертає інтервал часу в мікросекундах
```

Переповнення настає приблизно через 70 хвилин (за тактової частоти контролера 16 МГц), і підрахунок починається з нуля. Функція використовується аналогічно `millis()`.

1 000 мілісекунд відповідають 1 000 000 мікросекундам.

1.5 Протоколи зв'язку

1.5.1 Використання протоколу I²C

Послідовний протокол обміну даними IIC (частіше використовується запис I^2C – Inter-Integrated Circuits, міжмікросхемне з'єднання) використовує для передавання даних дві двонаправлені лінії зв'язку, які називаються шина послідовних даних SDA (Serial Data) і шина тактування SCL (Serial Clock). Також є дві лінії для живлення. Шини SDA і SCL підтягуються до шини живлення через резистори (рис. 1.16).

У мережі є хоча б один ведучий пристрій (Master), який ініціалізує передавання даних і генерує сигнали синхронізації. У мережі також є ведені пристрої (Slave), які передають дані за запитом ведучого. У кожного веденого пристрою є унікальна адреса, за якою ведучий і звертається до нього. Адреса пристрою зазначається в паспорті (datasheet). До однієї шині I^2C може бути під'єднано до 127 пристроїв, зокрема кілька ведучих. До шини можна під'єднувати пристрої в процесі роботи, тобто вона підтримує «гаряче під'єднання».

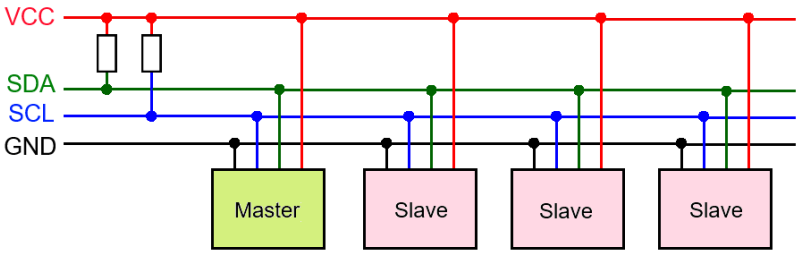


Рисунок 1.16 – Структурна схема мережі I²C

Реалізація I²C в Arduino. Arduino використовує для роботи за інтерфейсом I²C два порти. Наприклад, в Arduino UNO і Arduino NANO аналоговий порт A4 відповідає SDA, аналоговий порт A5 відповідає SCL. Для інших моделей плат відповідність виводів така показана в таблиці 1.3

Таблиця 1.3 – Реалізація I²C у різних платах Arduino

Плата	Пін SDA	Пін SCL
Arduino Uno, Nano, Pro і Pro Mini	A4	A5
Arduino Mega	20	21
Arduino Leonardo	2	3
Arduino Due	20, SDA1	21, SCL1

Бібліотека "Wire" для роботи з I²C. Для полегшення обміну даними з пристроями за шиною I²C для Arduino написана стандартна бібліотека `Wire.h`, функції для роботи з якою подані в таблиці 1.4. Більшість пристроїв, які під'єднуються за протоколом I²C для зручності використання мають власні бібліотеки, які наслідують функції бібліотеки `Wire.h`

1.5.2 Використання протоколу SPI

Інтерфейс SPI, розроблений компанією «Моторола», являє собою повнодуплексний послідовний стандарт

Таблиця 1.4 – Функції бібліотеки Wire

Функція	**Призначення**
`begin(address)`	Ініціалізація бібліотеки та під'єднання до шини I2C. Якщо не зазначена адреса, то пристрій вважається ведучим
`requestFrom()`	Використовується ведучим пристроєм для запиту визначеної кількості байт від веденого
`beginTransmission (address)`	Початок передавання даних до веденого пристрою за заданою адресою
`endTransmission()`	Припинення передавання даних веденому
`write()`	Запис даних від веденого у відповідь на запит
`available()`	Повертає кількість байт даних, доступних для приймання від веденого
`read()`	Зчитування байта, переданого між пристроями
`onReceive()`	Зазначає функцію, яка буде викликана під час приймання даних веденим пристроєм від ведучого
`onRequest()`	Зазначає функцію, яка буде викликана під час приймання даних ведучим пристроєм від веденого

зв’язку, який підтримує одночасний двонаправлений обмін даними між ведучим пристроєм і одним або декількома веденими. Оскільки протокол SPI не має формального стандарту, робота різних пристроїв SPI може дещо відрізнятися (наприклад, різним може бути число переданих у пакеті біт або може бути відсутнім лінія вибору веденого пристрою). Далі розглянемо загальноприйняті команди SPI, які підтримуються в Arduino IDE.

Залежно від вимог конкретного пристрою існують чотири основних способи реалізації протоколу SPI. SPI-пристрої є під час обміну як ведені синхронні пристрої, дані синхронізуються з тактовим сигналом (SCLK). Ведений пристрій може набувати даних або за позитивним, або за негативним фронтом тактового сигналу (так звана фаза синхронізації), а активний стан SCLK за замовчуванням може бути високим або низьким рівнем (так звана полярність синхронізації). У підсумку виходить, що обмін SPI в цілому можна налаштувати чотирма способами (табл. 1.5).

Таблиця 1.5 – Режими SPI в Arduino IDE.

Пор. №	**Режим SPI**	**Полярність синхронізації**	**Фаза синхронізації**
1	Mode 0	LOW	За фронтом синхронізатора
2	Mode 1	LOW	За спадом синхронізатора
3	Mode 2	HIGH	За спадом синхронізатора
4	Mode 3	HIGH	За фронтом синхронізатора

Під'єднання пристроїв SPI. Систему обміну даними через SPI нескладно налаштувати. Для зв'язку між ведучим і всіма веденими пристроями використовується три виводи:

– послідовний сигнал синхронізації (SCLK);
– вихід ведучого, вхід веденого (MOSI);
– вхід ведучого, вихід веденого (MISO).

У кожного веденого пристрою також є контакт вибору даного пристрою (контакт SS). Отже, загальна кількість портів введення-виведення, необхідних на ведучому пристрої, завжди буде $3 + n$, де n – число ведених пристроїв. Приклад SPI-системи з двома веденими пристроями зображений на рисунку 1.17.

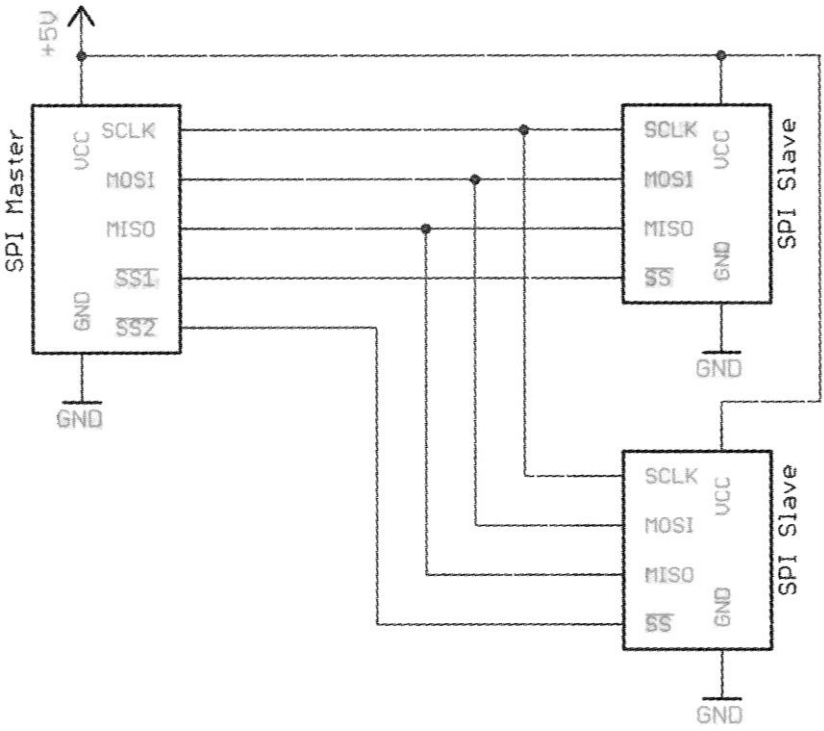


Рисунок 1.17 – Приклад конфігурації SPI-пристроїв

Конфігурація інтерфейсу SPI. Будь-який інтерфейс SPI містить, як мінімум, чотири лінії передавання даних. Для кожного веденого пристрою додаються додаткові лінії SS. Перш ніж відправляти або одержувати дані через SPI, потрібно з'ясувати, що роблять ці лінії введення-виведення і як вони повинні бути під'єднані (табл. 1.6).

Таблиця 1.6 – Лінії введення-виведення інтерфейсу SPI

Лінії SPI	**Призначення**
MOSI	Лінія для відправлення послідовних даних від ведучого пристрою до веденого
MISO	Лінія для відправлення послідовних даних від веденого пристрою до ведучого
SCLK	Лінія синхронізації послідовних даних
SS	Лінія вибору веденого пристрою, активний рівень – низький

На відміну від інтерфейсу I^2C, підтягувальні резистори тут не потрібні, і протокол повністю двонаправлений. Отже, щоб під'єднати пристрій SPI до плати Arduino, необхідно з'єднати його з контактами MOSI, MISO, SCLK і SS. Після цього все готово до використання Arduino бібліотеки SPI. Так як SPI не є універсальним стандартом, деякі виробники пристроїв SPI можуть по-різному називати лінії зв'язку SPI. Лінію вибору веденого іноді називають CS, лінію синхронізації – CLK; контакти MOSI і MISO ведених пристроїв називають входом послідовних даних (SDI) і виходом послідовних даних (SDO) відповідно.

Протокол передавання даних SPI. Передавання даних за SPI синхронізується тактовим сигналом і залежить від стану ліній SS. Усі команди, що відправляються ведучим, з'являються на входах MOSI, MISO, SCLK всіх ведених пристроїв. Стан контакту SS повідомляє пристрою

ігнорувати ці дані або набувати їх. Під час написання програми потрібно враховувати, що під час передавання даних лише один контакт SS повинен мати низький рівень.

Послідовність дій для зв'язку з пристроєм SPI виглядає так:

1. Встановити низький рівень на лінії SS пристрою, з яким хочете встановити зв'язок.

2. Перемикати на тактовій лінії рівень сигналу вгору і вниз зі швидкістю, меншою або такою, що дорівнює швидкості передавання, що підтримується веденим пристроєм.

3. На кожному такті відправляти по лінії MOSI або одержувати по лінії MISO 1 біт даних.

4. Продовжувати доти, поки передавання (або приймання) не закінчиться, і зупинити перемикання тактової лінії.

5. Установити на SS високий рівень.

Порівняння SPI і I2C. На відміну від I^2C, шина SPI має окремі лінії для відправлення та одержання даних, а також додаткову лінію для вибору веденого пристрою. Це потребує наявності додаткових виводів, але усуває проблему адресації веденого пристрою. SPI-інтерфейс порівняно з I^2C, простіший в реалізації й працює за більш високої швидкості. Багато видів пристроїв, зокрема акселерометри, цифрові потенціометри, дисплеї тощо, доступні й в SPI- і в I^2C-версіях. Зрештою, вибір пристрою залежить від конкретної ситуації. Більшість початківців вважають, що працювати з пристроями SPI легше, ніж із пристроями I^2C.

1.5.3 Енергонезалежна пам'ять EEPROM

EEPROM – (Electrically Erasable Programmable Read-Only Memory) постійний запам'ятовувальний пристрій, що

програмується та очищується за допомогою електрики, один із видів енергонезалежної пам'яті. Пам'ять такого типу може стиратися і заповнюватися даними кілька десятків тисяч разів. Використовується в твердотільних накопичувачах. Однією з різновидів EEPROM є флеш-пам'ять (Flash Memory).

Мікроконтролери Atmega8 і Atmega168, які працюють в Arduino мають 512 байт EEPROM, Atmega328 – 1кб незалежної пам'яті, в якій можна зберігати будь-які дані, які будуть доступні після вимкнення живлення. Це може стати в нагоді для зберігання будь-яких даних або значень.

Для роботи з цією пам'яттю в складі Arduino IDE є бібліотека `EEPROM.h` (\hardware\libraries\EEPROM\). Бібліотека містить усього дві функції – читання та записування даних.

byte EEPROM.read (address). Функція зчитує байт із незалежної пам'яті EEPROM. Якщо байт до цього ніколи не перезаписували – повернене значення буде 255.

Параметри:

`address` – порядковий номер комірки пам'яті для читання – від 0 до 1 023 (int) (для Atmega328).

Значення, що повертається: байт, що зберігається в комірці пам'яті.

Приклад зчитування даних з енергонезалежної пам'яті показаний у лістингу 1.40.

Лістинг 1.40 – Зчитування даних з EEPROM

```
   //Зчитування EEPROM
   //Зчитує значення всіх байтів незалежної
пам'яті і виведення їх в COM-порт
   #include <EEPROM.h>
   int address = 0; // початкова адреса
пам'яті EEPROM
   byte value;
```

```
   void setup () {
   Serial.begin (9600);
   }
   void loop () {
   value = EEPROM.read (address); //
зчитуємо значення з поточною адресою
EEPROM
   Serial.print (address);
   Serial.print ( "\ t");
   Serial.print (value, DEC);
   Serial.println ();
   address = address + 1; // встановлюємо
наступний елемент пам'яті
   if (address == 1024)
   address = 0; // якщо адреса досягла
1024, то знову переходимо на 0
   delay (500);
   }
```

Void EEPROM.write (address, value). Функція записує байт в енергонезалежну пам'ять.

Параметри:

`address`: порядковий номер комірки пам'яті для запису – від 0 до 1023 (int);

`value`: байт для записування – від 0 до 255 (byte)

Значення, що повертається: нічого.

Документація (*datasheet*) на мікроконтролери Atmega8 / 168 / 328 свідчить, що можлива кількість циклів перезапису даних у пам'яті обмежена 100 000 разів (Write / Erase Cycles). Це слід враховувати під час використання цієї пам'яті.

Так само документація свідчить, що час, необхідний для завершення циклу запису становить 3.3 ms . Якщо в цей час спробувати що-небудь зчитати / записати в EEPROM, то така спроба закінчиться невдачею, проте ця затримка вже

враховується бібліотекою EEPROM.h, тому в додатковому виклику delay () немає необхідності.

Приклад запису інформації в енергонезалежну пам'ять показаний у лістингу 1.41.

Лістинг 1.41 – Запис даних в енергонезалежну пам'ять

```
   // Зберігає в незалежній пам'яті EEPROM
значення, зчитаного з аналогового входу
analog input 0.
   #include <EEPROM.h>
   int addr = 0; // поточне значення
адреси EEPROM
   void setup () {
   }
   void loop () {
   int val = analogRead (0) / 4; //
ділення на 4 необхідно, щоб перевести
значення від 0-1023 у діапазон від 0 до
255.
   EEPROM.write (addr, val); // записуємо
значення в незалежну пам'ять
   addr = addr + 1; // встановлюємо
наступний елемент пам'яті.
   if (addr == 1024)
   addr = 0; //під час досягнення кінця
пам'яті - повертаємося на початок
   delay (100);
   }
```

1.5.4 Використання переривань в Arduino

Часто під час роботи з проєктами на мікроконтролерах потрібно запускати фонову функцію через однакові проміжки часу. Це часто реалізується установкою апаратного таймера для генерації переривання. Це

переривання запускає програму оброблення переривань (Interrupt Service Routine, ISR) для управління періодичним перериванням.

Переривання (Interrupt) – це сигнали, що переривають нормальний перебіг програми. Переривання зазвичай використовуються для апаратних пристроїв, що потребують негайної реакції на появу подій. Наприклад, система послідовного порту або UART мікроконтролера повинна бути обслужена під час одержання нового символу. Якщо цього не зробити швидко, новий символ може бути втрачений.

Переривання може бути викликане у разі настання якої-небудь події, водночас виконання поточної послідовності команд призупиняється й керування передається обробнику переривання, який реагує на подію та обслуговує її, після чого повертає управління в перерваний код.

Залежно від джерела виникнення сигналу переривання поділяють на:

– асинхронні або зовнішні (апаратні) – події, які створені зовнішніми джерелами (наприклад, периферійними пристроями) та можуть відбутися в довільний момент: сигнал від таймера, мережевої карти або дискового накопичувача, натискання клавіш клавіатури, рух мишки;

– синхронні або внутрішні – події в самому процесорі як результат порушення якихось умов під час виконання машинного коду: поділ на нуль або переповнення, звернення до неприпустимих адрес або неприпустимий код операції;

– програмні (частковий випадок внутрішнього переривання) – ініціюються виконанням спеціальної інструкції в коді програми. Програмні переривання, зазвичай використовуються для звернення до функцій вбудованого програмного забезпечення (firmware),

драйверів і операційної системи.

Під час надходження нового символу UART генерує переривання. Мікроконтролер зупиняє виконання основної програми (вашого додатка) і перескакує на програму оброблення переривань (ISR), призначену для цього переривання. У цьому разі це переривання за одержанням символу. Ця ISR захоплює новий символ із UART, поміщає в буфер, потім очищає переривання і виконує повернення. Коли ISR виконує повернення, мікроконтролер повертається в основну програму та продовжує її з точки викликання. Все це відбувається в фоновому режимі та не впливає безпосередньо на основний код вашої програми.

Якщо запускається багато переривань або переривання генерує швидкодіючий таймер, ваша основна програма буде виконуватися повільніше, тому що мікроконтролер розподіляє свій машинний час між основною програмою та всіма функціями оброблення переривань.

Виникає думка, чому б просто не перевіряти новий символ час від часу, замість використання такого складного процесу переривання. Обчислимо приклад, щоб побачити, наскільки важливі процеси переривання. У вас є послідовний порт зі швидкістю передавання даних 9 600 бод. Це означає, що кожен біт символу посилається з частотою 9 600 Гц або близько 10 кГц. На кожен біт йде 100 мкс. Близько 10 біт потрібно, щоб послати один символ, так що ми одержуємо один повний символ кожну мілісекунду. Якщо наш UART буферизується, ми повинні витягти останній символ до завершення приймання наступного, це дає нам на всю роботу 1 мс. Якщо наш UART не буферизується, ми повинні позбутися від символу за 1 біт або 1 мкс. Розглянемо для початку буферизований приклад.

Ми повинні перевіряти одержання байта швидше, ніж кожну мілісекунду, щоб запобігти втраті даних. Щодо Arduino – це означає, що наша функція циклу повинна

звертатися для читання статусу UART і можливо, байта даних 1 000 разів за 1 секунду. Це легко здійснити, але сильно ускладнить код, який вам потрібно написати. Уявіть, що вам потрібно обслуговувати кілька пристроїв введення-виведення, або, що необхідно працювати на набагато більшій швидкості передавання. Це може призвести до багатьох неприємностей.

Із перериваннями вам не потрібно відстежувати надходження символу. Апаратура подає сигнал за допомогою переривання, і процесор швидко викличе ISR, щоб вчасно захопити символ. Замість виділення величезної частки процесорного часу на перевірку статусу UART, ви ніколи не повинні перевіряти статус, просто встановлюєте апаратне переривання й виконуєте необхідні дії в ISR. Ваша головна програма безпосередньо не зачіпається, і від апаратного пристрою не потрібно особливих можливостей.

Переривання за таймером. Розглянемо використання програмного таймера 2 для періодичних переривань. Вихідна ідея полягала у використанні цього таймера для генерації частоти биття в звукових проєктах Arduino. Щоб виводити тон або частоту, нам потрібно перемикати порт введення-виведення на узгодженій частоті. Це можна робити з використанням циклів затримки. Це просто, але означає, що наш процесор буде зайнятий, нічого не виконуючи, а чекаючи точного часу перемикання. З використанням переривання за таймером ми можемо зайнятися чимось іншим, а вивід нехай перемикає ISR, коли таймер подасть сигнал, що час настав.

Нам потрібно лише встановити таймер, щоб подавав сигнал з перериванням у потрібний час. Замість прокручуванні циклу для затримки за часом, наша основна програма може робити щось інше, наприклад, контролювати датчик руху або керувати електроприводом. Опишемо ISR лише в аспектах, що стосуються переривань

таймера 2.

Таймери на Arduino. Arduino використовує три таймери контролера ATMega:

– Таймер 0 (системний час, ШІМ 5 і 6). Використовують для зберігання лічильника часу роботи програми. Функція `millis ()` повертає число мілісекунд із моменту запускання програми, використовуючи ISR глобального збільшення таймера 0. Таймер 0 також використовують для реалізації ШІМ на виводах 5 і 6;

– Таймер 1 (ШІМ 9 і 10). Використовують для реалізації ШІМ для цифрових виводів 9 і 10;

– Таймер 2 (ШІМ 3 і 11). Використовують для управління виходами ШІМ для цифрових виводів 3 і 11.

Хоча всі таймери використовують, лише Таймер 0 має призначену таймером ISR. Це означає, що ми можемо використати Таймер 1 і/або Таймер 2 під свої потреби. Проте після цього ви не зможете використовувати ШІМ на деяких портах введення-виведення. Якщо ви плануєте використовувати ШІМ, майте це на увазі. У прикладі використано Таймер 2, що вплине на роботу виводів 3 і 11. Тестова програма повністю вимикає ШІМ на цифрових виводах, керованих таймером 2.

Установлення Таймера 2. Наведена нижче програма (лістинг 1.42) показує створену функцію установлення Таймера 2. Ця функція під'єднує переривання з переповнення Таймера 2, встановлює попередньо заданий масштаб для таймера, підраховує завантажуване значення таймера, що дає бажану частоту переривань за часом.

Лістинг 1.42 – Установлення таймера 2

```
   #define TIMER_CLOCK_FREQ 2000000.0 //
2MHz for / 8 prescale from 16MHz
   /* Установлення Таймера2.
Конфігурується 8-бітний Таймер2 ATMega для
```

```
генерації переривання з заданою частотою.
Повертає початкове значення таймера, яке
повинно бути завантажено в TCNT2 всередині
вашої процедури ISR.
   */
   unsigned char SetupTimer2 (float
timeoutFrequency) {
   unsigned char result; // Початкове
значення таймера.
   // Підрахунок початкового значення
таймера
   result = (int) ((256.0-
(TIMER_CLOCK_FREQ / timeoutFrequency)) +
0.5);
   // Установки Таймер2: Дільник частоти /
8, режим 0
   // Частота = 16MHz / 8 = 2Mhz або 0.5
мкс
   // Дільник / 8 задає робочий діапазон
   // Жорстко запрограмуємо це.
   TCCR2A = 0;
   TCCR2B = 0 << CS22 | 1 << CS21 | 0 <<
CS20;
   // Під'єднання переривання з
переповнення Timer2
   TIMSK2 = 1 << TOIE2;
   // завантажує таймер для першого циклу
   TCNT2 = result;
   return (result);
   }
```

Спочатку визначається тактова частота таймера. Показано, що тактова частота встановлена 2 МГц, оскільки ми використовуємо ділення на 8 (дільник частоти) опорної частоти 16 МГц. Це жорстко запрограмовано в функції. Функція має один аргумент – бажану частоту переривань, і

повертає значення, яке необхідно перезавантажувати в таймер у процедурі ISR. Функція не обмежує необхідну частоту, але не потрібно занадто її завищувати.

Далі підраховується значення, перезавантажене в таймер. Це дуже простий підрахунок, але вимагає операцій із плаваючою крапкою. Нам потрібно зробити це тільки один раз, оскільки операції з плаваючою крапкою дуже дорогі в перерахунку на машинний час. Візьмемо, що таймер буде встановлений на 2 МГц за кожного рахунку. Завантажуване значення – це кількість відліків, яку ми хочемо зробити за 2 МГц між перериваннями.

Наступна частина коду встановлює таймер у режим 0 і вибирає дільник частоти / 8. Режим 0 – це базовий режим таймера, а дільник / 8 показує, як ми отримуємо лічильник, який рахує за частотою 2 МГц або 0,5 мкс на відлік. Далі під'єднується переривання з переповнення. Після виконання цього коду мікроконтролер буде викликати ISR кожен раз, коли лічильник прокрутиться від 0xFF до 0x00. Це трапиться, коли лічильник прорахує від нашого завантаженого значення через FF і назад до 00. Ми завантажуємо значення лічильника в таймер і повертаємо це завантажене значення, щоб ISR могла використовувати його пізніше.

Завантаження мікроконтролера перериваннями.

Щоб дати вам уявлення про ефект, припустимо, що таймер ISR запускався б кожні 20 мкс. Процесор, що працює на 16 МГц, може виконати близько 1 машинної команди кожні 63 нс або близько 320 машинних команд для кожного циклу переривання (20 мкс). Припустимо також, що виконання кожного рядка програми може зайняти багато машинних команд. Кожна інструкція, яка використовується в ISR, забирає час, доступний для виконання будь-якої іншої програми. Якби наша ISR використовувала близько 150 машинних циклів, ми

використали б половину доступного процесорного часу. У разі активних переривань головна програма відкладалася б на половину часу, займаного нею в інших випадках.

Якщо у вас буде занадто довга ISR, ваша головна програма буде виконуватися дуже повільно, якщо ж ISR буде довшим, ніж тривалість циклу таймера, то практично ніколи не виконається ваша головна програма, і, крім того, зрештою відбудеться збій системного стека.

Вимірювання завантаження переривань. Напишемо тестовий код, який виконує оцінювання завантаження і дозволяє вивести вимірювання на послідовний порт. Таймер не встановлено в режим, коли він перезавантажується автоматично. Це означає, що ISR повинна перезавантажити таймер для наступного інтервалу рахунку. Було б правильніше автоматично перезавантажувати таймер, але, використовуючи цей режим, ми можемо виміряти час, що проводиться в ISR, і відповідно виправити час, що завантажується у таймер. За допомогою цієї корекції ми також отримуємо і число, що показує, скільки часу ми проводимо в ISR.

Це полягає в тому, що таймер зберігає час, навіть якщо він переповнений і перерваний. Наприкінці нашої ISR ми можемо захопити поточне значення лічильника таймера. Це значення – той час, який він відібрав у нас до наступної точки програми. Це сумарний час, витрачений на перехід в процедуру переривання та виконання програми в ISR. Невелика похибка буде в тому, що не підраховується час, витрачений на команду перезавантаження таймера, але ми можемо виправити її емпірично. Саме тому в формулі підрахунку завантажується значення 257 замість 256, і зайвий такт компенсує команду перезавантаження таймера.

ISR Таймера 2. ISR для переривання з переповнення Таймера 2 показана в лістингу 1.43.

Лістинг 1.43 – Оброблення переривань з переповнення Таймера 2

```
   #define TOGGLE_IO 9 // вивод Arduino
для перемикання за таймером ISR
   // Timer2 покажчик вектора переривання
з переповнення
   ISR (TIMER2_OVF_vect) {
   // Перемикання IO-виведення в інший
стан.
   digitalWrite (TOGGLE_IO,! digitalRead
(TOGGLE_IO));
   // Захоплення поточного значення
таймера. Це величина помилки через
затримку оброблення переривання та
виконання цієї функції
   latency = TCNT2;
   // Перезавантаження таймера та корекція
з затримки
   TCNT2 = latency + timerLoadValue;
   }
```

Ця функція коротка і її основне завдання – перемикати порт введення-виведення. Після перемикання вона захоплює поточне значення таймера й використовує його для корекції затримки на перезавантаження таймера. Значення затримки глобальне, його може відстежувати головна програма для вищезазначених вимірювань завантаження. Це кількість тактів на частоті 2 МГц, яка займає ця ISR для виконання своїх функцій.

Важливо щоб ISR була короткою, так як вона викликається кожні 20 мкс за таймера, налаштованого на 50 кГц. Ви можете робити більше операцій в ISR, але вам потрібно знайти баланс між інтервалом переривань і

кількістю операцій, які виконуються в них. Значення затримки допоможе в цьому, як описано нижче.

ISR була б набагато швидше, якби ми безпосередньо зверталися до виводу за допомогою регістрів порту. Але краще використовувати загальні бібліотечні функції.

Головна програма. Функція Setup (). Функція `Setup ()` (лістинг 1.44) викликається за допомогою системної програми Arduino одноразово під час запуску програми. Вона ініціалізує порти введення-виведення й таймер. Вона також виконує виведення в послідовний порт, тобто показує, що програма запущена.

Лістинг 1.44 – Реалізація функції `setup () ` для ініціалізації Таймера 2

```
   void setup (void) {
   pinMode (TOGGLE_IO, OUTPUT);
   // Встановлює порт, який нам потрібно
перемикати в ISR, вихідним.
   Serial.begin (9600); // Запускаємо
послідовний порт
   Serial.println ( "Timer2 Test");
   // Повідомлення про запуск програми
   timerLoadValue = SetupTimer2 (44100);
// Запускає таймер і отримує завантажуване
значення таймера.
   // Виводимо завантажуване значення
таймера
   Serial.print ( "Timer2 Load:");
   Serial.println (timerLoadValue, HEX);
   }
```

`Setup ()` починається з установлення порту в режим виходу, так щоб ми могли перемикати його в ISR. Потім

вона активує послідовний порт і виводить текстове повідомлення, щоб показати, що програма працює.

Далі викликаємо функцію `SetupTimer2` з частотою, встановленою в 44 100 Гц, загальною частотою дискретизації звуку. Значення, що повертається зберігається в глобальній змінній `timerLoadValue` для подальшого використання в ISR.

Нарешті, `Setup ()` виводить `timerLoadValue` так, що ми можемо переконатися, що воно розміщене в розумних межах.

На цій стадії таймер запущений і наша процедура ISR викликається із заданою частотою. Якщо під'єднати осцилограф, ви побачите перемикання виводу, що генерує частоту, таку, що дорівнює половині інтервалу таймера. Половина виходить тому, що ми встановлюємо порт у низький рівень в одній ISR, і записуємо в нього високий рівень в інший.

Головна програма. Функція Loop (). Розглянемо програмний код, поданий у лістингу 1.45. Функцію циклу викликаємо циклічно, поки програма запущена. Кожен раз під час повернення з циклу він викликається знову. Текст програми виглядає складним, але насправді все, що він робить – усереднює величину затримки ISR Таймера 2 і виводить результати вимірювань після одержання 100 замірів.

Необхідно зазначити, що функція циклу нічого не робить щодо перемикання лінії введення-виведення. Все це керується ISR, що дозволяє функції циклу зайнятися іншими підрахунками, не звертаючи уваги на процеси, що відбуваються в ISR. У разі з послідовним портом, ви не повинні турбуватися про завантаження наступного символу в UART, коли він готовий. Ви займаєтеся своїми справами, а послідовний порт керується в фоновому режимі. Це одна з переваг програм, керованих перериваннями. Функції

викликаються у разі появи події, і відокремлені від вашої прикладної програми.

Лістинг 1.45 – Реалізація циклу `loop ()` для підрахунку часу виконання обробника переривань

```
   void loop (void) {
   // Збирає затримку ISR кожні 10 мс.
   delay (10);
   // Збирає поточне значення затримки з
ISR і збільшує лічильник на 1
   latencySum + = latency;
   sampleCount ++;
   // Як тільки набереться 10 замірів,
обчислює й виводить результат вимірювань
   if (sampleCount> 99) {
   float latencyAverage;
   float loadPercent;
   // Обчислює середню затримку
   latencyAverage = latencySum / 100.0;
   // обнуляє значення лічильника
   sampleCount = 0;
   latencySum = 0;
   // Обчислює очікуваний відсоток
завантаження процесора
   loadPercent = latencyAverage / (float)
timerLoadValue;
   loadPercent * = 100; // Перераховує
частки у відсотки;
   // Виводить середню затримку
   Serial.print ( "Latency Average:");
   Serial.print ((int) latencyAverage);
   Serial.print ( ".");
   latencyAverage - = (int)
latencyAverage;
```

```
   Serial.print ((int) (latencyAverage *
100));
   // Виводить очікуваний відсоток
завантаження
   Serial.print ( "Load:");
   Serial.print ((int) loadPercent);
   Serial.println ( "%");
     }
   }
```

Функція циклу починається з затримки 10 мс (зауважте, що ми не можемо використовувати подібні затримки, якщо потрібно перемикати порт введення-виведення з високою частотою без переривань). Затримка 10 мс регулює те, як часто ми робимо заміри затримки і виведення вимірювань. Після того як ми одержимо 100 вимірювань, кожне з яких займає 10 мс, результат виводиться кожну секунду.

Далі значення часу затримки з ISR накопичується в глобальній змінній `latencySum`. Ми просто захоплюємо поточне значення затримки і додаємо його до того, що вже накопичено. Крім того, ми збільшуємо на одиницю лічильник накопиченої кількості вимірювань.

Тепер ми перевіряємо, чи було вже накопичено 100 замірів. Якщо ні, то пропускаємо решту коду і повертаємося. Якщо накопичилося 100 замірів, то розраховуємо середнє значення й зберігаємо результат у `latencyAverage`. Після цього очищаємо акумулятор і лічильник вимірювань, щоб можна було почати все заново.

Тепер, маючи вимірювання затримки, ми можемо підрахувати очікуване завантаження процесора. Нам відомо, що таймер переповнюється кожного разу, коли він дораховує від нашого перезавантажуваного значення до 0xFF і потім назад до 0x00, це показник переповнення. Відсоток завантаження буде визначатися як затримка /

число тактів в ISR. Це значення підраховується й виводиться як результат вимірювання, так що ми можемо оцінити вплив нашої програми ISR.

Час, який займає головна програма – це залишок часу від виконання всіх фонових ISR, ISR цього таймера та інших, постійно активних, наприклад, Таймера 0 і послідовного порту.

У прикладі програми таймер завантажений шістнадцятковим значенням D4, або десятковим 212. Це означає, що він буде перерваний кожного разу, як тільки відрахує 44 такти. Ми знаємо, що поки процесор виконує код ISR, таймер відраховує близько 20 тактів, так що залишається близько 24 до того, як він знову повернеться в ISR. Ці 24 такти – весь час, який наша головна програма одержує на виконання. Так що із загального часу 44 такти між перериваннями ми витрачаємо 20 тактів на ISR, залишаючи близько 24 тактів прикладній програмі. Це становить близько 45 %, витрачених процесором на ISR.

Сама ISR, без обслуговування будь-яких корисних завдань, використовує близько 6 % доступних ресурсів. Час, що залишився, використовують функції цифрового читання й запису.

Розділ 2
Основи збирання та оброблення сигналів

2.1 Огляд систем збирання даних і вимірювальних перетворювачів

Призначенням системи збирання даних (Data Acquisition – DAQ) є вимірювання параметрів фізичного явища, такого як світло, температура, тиск або звук. Система збирання даних складається з таких елементів:

- вимірювальний перетворювач (датчик);
- сигнал;
- система узгодження сигналу;
- пристрій збирання даних;
- драйвер і програмний додаток.

Використовуючи ці п’ять складових елементів, ви можете ввести досліджуваний сигнал у комп’ютер (або інший пристрій оброблення даних) для подальшого аналізу й подання результатів.

2.1.1 Вимірювальні перетворювачі (датчики)

Збирання сигналу – це процес перетворення фізичного явища в дані, які потім може використовувати комп’ютер. Будь-яке вимірювання починається з використання перетворювача або датчика, що перетворює фізичне явище в електричний сигнал. Датчики можуть створювати електричний сигнал під дією таких величин, як температура, тиск, звукові коливання або інтенсивність світла. У таблиці 2.1 наведено деякі з найбільш поширених перетворювачів.

Різні датчики потребують особливих умов для перетворення фізичного явища у вимірний сигнал. Наприклад, для вимірювання температури за допомогою

терморезистора необхідний струм збудження. Термопара не потребує струму збудження, проте їй потрібна компенсація холодного спаю. Під час вимірювання деформації тензодатчиками використовують складання резисторів, так званий міст Уітстона (або тензометричний міст). Перед установленням системи ви повинні знати, чи має використовуваний перетворювач спеціальні вимоги щодо під'єднання.

Таблиця 2.1 – Перетворювачі фізичних величин

Явище	**Перетворювач**
Температура	Термопари; термочутливі резистори (терморезистори); термістори; датчики на інтегральних схемах
Світло	Вакуумні фотодатчики; фоторезистори; фотодіоди; прилади з зарядовим зв’язком
Звук	Мікрофони
Сила і тиск	Тензодатчики; п’єзоелектричні вимірювальні перетворювачі; динамометричні елементи
Положення (зміщення)	Потенціометри; лінійні індуктивні датчики на основі диференціального трансформатора; оптичні датчики положення
Витрата рідини	Манометри-витратоміри; вертушечні витратоміри; ультразвукові витратоміри
Кислотність	pH-електроди

2.1.2 Сигнали

За допомогою датчиків ви перетворюєте фізичне явище в сигнал. Не всі сигнали вимірюються однаково, тому насамперед вам необхідно розділити сигнали на два типи: аналогові й дискретні (цифрові). Після того, як це зроблено, вирішіть, яку інформацію ви хочете одержати від сигналу. Можливі типи інформації, які ви можете одержати з сигналу: стан, швидкість зміни, рівень, форма й частота (рис. 2.1).

Рисунок 2.1 – Інформація, що може бути одержана з сигналу

Аналогові сигнали. Аналоговий сигнал може мати будь-який рівень напруги залежно від часу. Оскільки аналоговий сигнал може бути в будь-якому стані в будь-який момент часу, фізичні властивості, які ви будете вимірювати, відрізняються від тих, які вимірюють у цифровому сигналі.

Інформація в аналоговому сигналі. Ви можете вимірювати рівень, форму та частоту аналогового сигналу, як це показано на рисунку 2.2.

Рівень – вимірювання рівня напруги аналогового сигналу схоже на вимірювання стану цифрового сигналу.

Відмінність у тому, що аналоговий сигнал може мати будь-який рівень напруги, а цифровий може мати два фіксованих рівні: 0 або 5 вольт.

Форма – вимірювання форми сигналу часто є дуже важливим, оскільки аналогові сигнали можуть мати будь-яке значення в кожен момент часу. Наприклад, синусоїдальний сигнал відрізняється від пилкоподібного. Вимірювання форми сигналу надає інформацію для аналізу інших параметрів, таких як пікові (амплітудні) значення, параметри фронту або середнє значення.

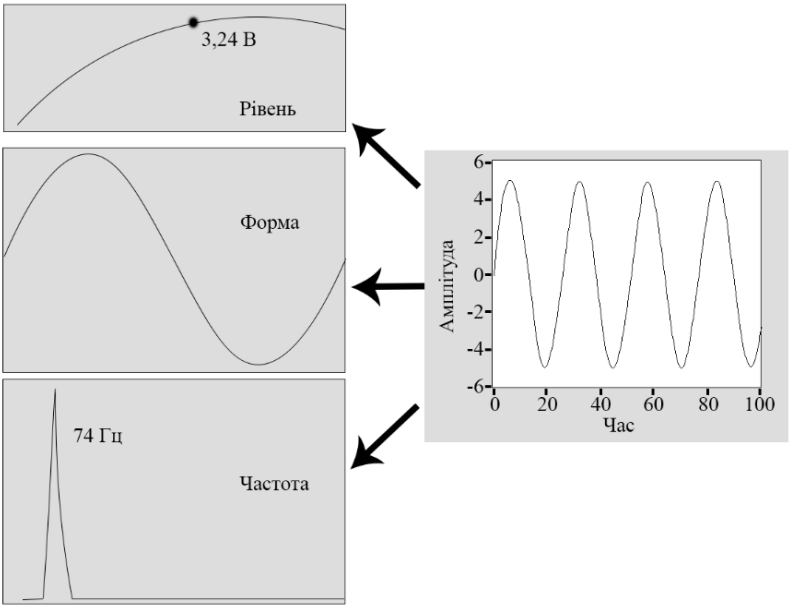


Рисунок 2.2 – Інформація, одержана з аналогового сигналу

Частота – вимірювання частоти аналогового сигналу схоже на вимірювання частоти цифрового сигналу. Проте безпосередньо поміряти частоту аналогового сигналу ви не

зможете, для цього необхідний програмний аналіз (зазвичай це перетворення Фур'є), щоб виділити частотну інформацію.

Рівень більшості сигналів не сильно змінюється з часом. Водночас сигнал зазвичай потрібно вимірювати з високою точністю. Вам для цього необхідний пристрій збирання даних з великою роздільною здатністю, але невеликою частотою вибірки. Використовуючи різні перетворювачі (датчики), ви можете вимірювати напруги джерел живлення, температуру середовищ й об'єктів, тиск усередині ємностей або навантаження на частину механізму, як показано на рисунку 2.3.

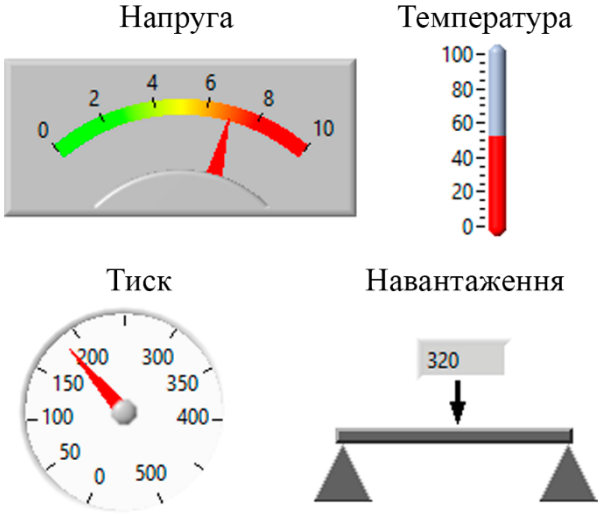


Рисунок 2.3 – Вимірюваний аналоговий сигнал

Вимірюючи форму сигналу, ви одержуєте залежність рівня сигналу від часу. Деякі сигнали дуже швидко змінюються з часом, а вам знову потрібно з високою точністю вимірювати їх рівень. Для таких сигналів необхідно використовувати пристрій збирання даних із

великою роздільною здатністю та високою швидкістю вибірки.

Велику кількість прикладів вимірювання форми сигналів можна знайти в медичній, електронній та автомобільній промисловості – від вимірювання параметрів серцебиття й відеосигналів до вимірювання вібрації пружини. Після введення сигналу в комп'ютер ви зможете проаналізувати його форму для виділення необхідної інформації.

Наприклад, під час вимірювання кров'яного тиску найбільш інформативні пікові значення. Проте через наявність постійної часу ланцюга опір-конденсатор (RC-ланцюжок) вам швидше за все доведеться працювати з залежністю амплітуди від часу, як це показано на рисунку 2.4.

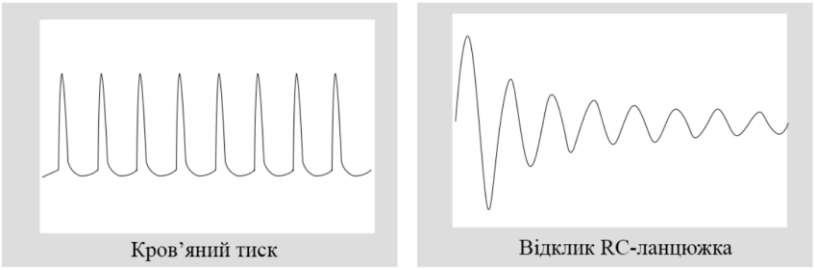


Рисунок 2.4 – Інформація відклику RC-ланцюжка

Під час вимірювання частоти сигналу необхідна залежність сигналу від часу. Багато сигналів дуже швидко змінюються в часі, і водночас вам потрібно з високою точністю вимірювати частоту сигналу. Тому необхідно використовувати пристрій збирання даних із великою роздільною здатністю й високою швидкістю дискретизації. Одержавши сигнал залежно від часу, ви можете одержати і

його залежність від частоти за допомогою програмного забезпечення.

Дискретні сигнали. Дискретні (цифрові) сигнали мають два можливих стани: «ввімкнено» (високий рівень напруги, стан «істина») і «вимкнено» (низький рівень напруги, стан «брехня»). У сучасній техніці дискретні сигнали часто подані сигналами транзисторно-транзисторної логіки (ТТЛ-сигнали). Технічні умови ТТЛ сигналів визначають рівень напруги від 0 вольт до 0,8 вольт як «низький», а рівень напруги від 2 вольт до 5 вольт як «високий». Більшість цифрових пристроїв мають ТТЛ-сумісний сигнал.

Інформація в дискретному сигналі. У дискретному сигналі можна виміряти лише два параметри: стан (рівень) і швидкість зміни.

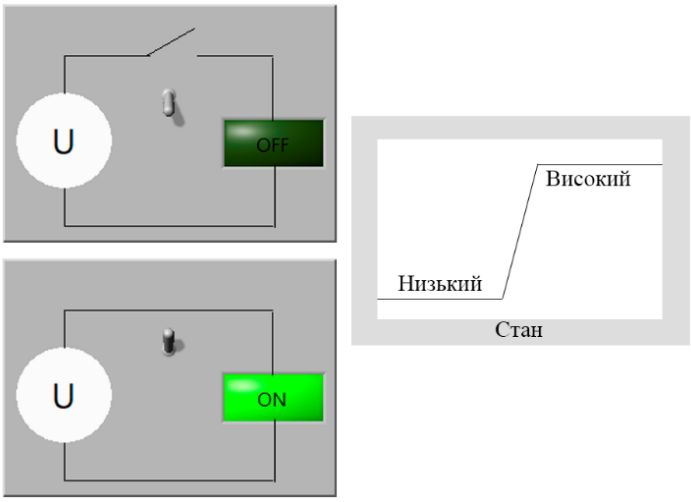
Стан – цифровий сигнал має два можливих стани (рівні): «включено» і «вимкнено». Тому один із можливих для вимірювання параметрів – це перевірка стану: «ввімкнено» або «вимкнено».

Частота (швидкість зміни) – Стан цифрового сигналу також може змінюватися з часом. Тому інший вимірюваний параметр – це швидкість зміни станів сигналу з часом.

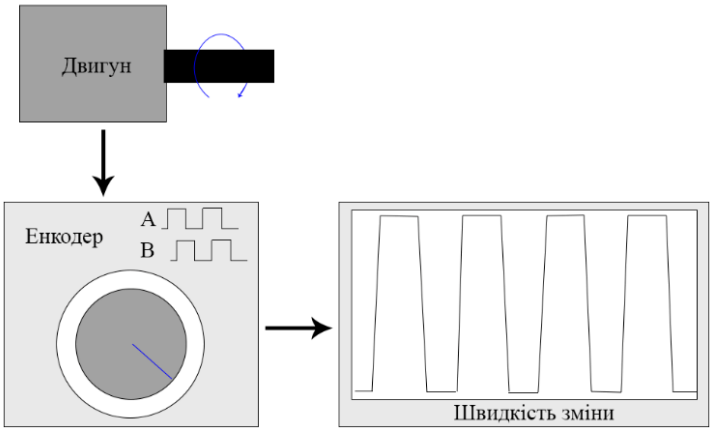
Розглянемо приклад вимірювання стану дискретного сигналу. Припустимо, у вас є перемикач, який потрібно контролювати. Цей перемикач керує ввімкненням і вимиканням світла (рис. 2.5 а). Коли перемикач розімкнений, ви виміряєте 0 вольт («вимкнено»). Коли перемикач замкнений, ви виміряєте 5 вольт («ввімкнено»).

Тепер розглянемо наступний приклад вимірювання частоти дискретного сигналу. Припустимо, що у вас є двигун, і ви хочете визначити, наскільки швидко обертається його вал (рис. 2.5 б). Пристрій кодування (енкодер) за кутом повороту – це датчик, що перетворює інформацію про обертальний рух вала двигуна в дискретний сигнал. Коли вал обертається, енкодер генерує

два цифрових сигнали. Кожен сигнал це серія високих і низьких станів, яку називають



а



б

Рисунок 2.5 – Приклад вимірювання дискретного сигналу

послідовністю імпульсів (pulse train). За певного повороту вала генерується імпульс. Величина кута повороту вала, яка припадає на один імпульс, залежить від пристрою енкодера. Вимірюючи частоту імпульсів в одній із послідовностей, ви можете визначити наскільки швидко обертається вал. Вимірюючи обидві послідовності імпульсів, можна визначити не лише швидкість обертання вала, а також і напрямок.

2.1.3 Загальні уявлення про узгодження сигналів

Ви не завжди зможете під'єднати сигнал безпосередньо до пристрою збирання даних. Можливо, для точного вимірювання потрібно додатково змінити сигнал, щоб узгодити його параметри до вимог вхідного сигналу пристрою збирання даних.

Узгодження сигналів – одна з найбільш важливих технологій у системі вимірювань і автоматизації. Вона забезпечує інтерфейс між сигналами, датчиками та вимірювальною системою.

Більшості вимірювальних перетворювачів для виконання роботи потрібен певний клас додаткового зовнішнього обладнання. Наприклад, терморезисторам необхідний струм збудження, тензодатчикам – схема, що складається з резисторів (міст Уітстона). Узгодження сигналів – це процес вимірювання та керування сигналами з метою поліпшення точності, створення розв’язки, фільтрації тощо.

Для вимірювання сигналів, що надходять від датчиків, вам необхідно перетворити їх у форму, що найбільш підходить для пристроїв збирання даних. Наприклад, вихідна напруга більшості термопар дуже мала й чутлива до шумів. Тому необхідне попереднє підсилення сигналу перед його оцифруванням. Найпоширенішими типами

узгодження сигналів є підсилення, лінеаризація, збудження («запитка») перетворювача й гальванічна розв'язка («ізоляція»).

У таблиці 2.2 показані найбільш поширені типи вимірювальних перетворювачів і сигналів, а також типи узгодження сигналів, які кожен із них потребує.

Таблиця 2.2 – Поширені типи перетворювачів сигналів і типи узгодження

Перетворювачі / сигнали	**Метод узгодження**
Термопари	Підсилення, лінеаризація, компенсація холодного спаю
Резистивні температурні детектори	Збудження струмом, лінеаризація
Тензодатчики	Збудження напругою, конфігурація моста, лінеаризація
Синфазна та висока напруга	Оптична розв'язка
Навантаження, що потребують перемикання змінних напруг або великих струмів	Електромеханічні та твердотільні реле
Сигнали частотними перешкодами	Фільтри частот

Підсилення – найбільш поширений тип узгодження сигналів. Підсилення електричних сигналів збільшує точність результувальному оцифрованному сигналі й зменшує вплив шумів. Типовим прикладом є сигнал з термопари, напруга якої міститься в діапазоні декількох мілівольт.

Якщо ви подасте сигнал від термопари безпосередньо на

пристрій збирання даних (ПЗД), то зміна температури на один-два градуси така система може не зафіксувати. Використовуючи підсилення, ви одержите сигнал із кращою відповідністю робочому діапазону ПЗД. Підсилення сигналу може здійснюватися безпосередньо самим пристроєм збирання даних або зовнішнім підсилювачем у безпосередній близькості від його джерела.

Для досягнення найбільшої можливої точності та збільшення відношення сигнал / шум (signal to noise ratio – SNR) сигнал необхідно підсилити так, щоб його максимальна амплітуда дорівнювала максимальному діапазону вхідного сигналу аналого-цифрового перетворювача ПЗД.

У разі підсилення сигналу самим ПЗД вимірюватися і оцифровуватися буде не лише сам сигнал, а і перешкоди, які можуть наводитися на з'єднувальні дроти, що погіршить співвідношення сигнал / шум. Підсилення сигналу в безпосередній близькості від його джерела з використанням зовнішнього підсилювача зменшує шкідливий вплив шуму на сигнал, і в цьому разі оцифрований результат точніше відповідає слабкому вихідного сигналу. На рисунку 2.6 показано систему даних із двоступеневим підсиленням сигналу.

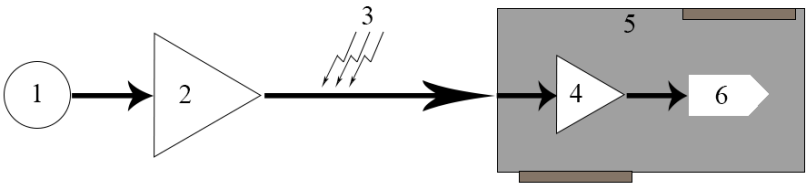


Рисунок 2.6 – Система збирання даних із двоступеневим підсиленням сигналу: 1 – слабкий сигнал; 2 – зовнішній підсилювач; 3 – перешкоди, що діють на лінію зв'язку; 4 – інструментальний підсилювач; 5 – пристрій збирання даних; 6 – АЦП

Існує кілька способів зменшення шуму:

– використовувати екрановані кабелі або виту пару проводів;

– мінімізувати довжину проводів для зменшення перешкод, які можуть на них наводитися;

– розміщувати сигнальні дроти далеко від силових кабелів змінного струму та комп'ютерних моніторів для зменшення наведень на частоті 50 Гц або 60 Гц.

Відношення сигнал / шум показує рівень шуму щодо корисного сигналу і визначається як напруга сигналу, поділена на напругу шуму (перешкоди). Чим більше це відношення, тим краще. Як показано в таблиці 2.3, максимальне відношення сигнал / шум досягається для сигналу, підсиленого лише зовнішнім підсилювачем, і мінімальне – лише в пристрої збирання даних.

Таблиця 2.3 – Розрахунок відношення сигнал / шум для різних способів підсилення сигналу

	Напруга сигналу	**Коефіцієнт підсилення**	**Перешкоди в підвідних дротах**	**Підсилення DAQ пристрою**	**Оцифрована напруга**	**Відношення сигнал / шум**
Підсилення лише в DAQ-пристрої	0,01 В	Немає	0,001 В	× 100	1,1 В	10
Підсилення в SCXI і DAQ-пристрої	0,01 В	× 10	0,001 В	× 10	1,01 В	100
Підсилення лише в SCXI	0,01 В	× 100	0,001 В	Немає	1,001 В	1 000

Інші типи узгодження сигналів

Лінеаризація. Багато датчиків мають нелінійну залежність вихідного сигналу від зміни вимірюваного фізичного явища. Наприклад, зміна напруги термопари на 10 мВ зазвичай не означає зміни температури на 10 градусів. У більшості датчиків є таблиці перерахунку, які зазначають, як масштабувати їх показання для отримання точних результатів. Лінеаризацію датчика можна здійснити як за допомогою апаратної частини ПЗД, так і програмно.

Збудження («запитка») перетворювача. Системи узгодження можуть генерувати сигнали збудження, необхідні для роботи деяких датчиків. Для тензодатчиків, терморезисторів і акселерометрів необхідні зовнішні напруга та струм відповідно для збудження їх ланцюгів під час вимірювання фізичних явищ.

Гальванічна розв'язка («ізоляція»). Ще один поширений тип узгодження – гальванічне ізолювання потенціалів датчиків від ланцюгів вимірювальної системи з метою електробезпеки. Досліджуваний сигнал може мати високий потенціал або пікові викиди напруги щодо потенціалу ПЗД, які можуть пошкодити його схему або завдати шкоди оператору, тому не можна приєднувати датчики безпосередньо до пристрою збирання даних без будь-якої гальванічної розв'язки (ізоляції). Розв'язка також часто використовується для усунення впливу відмінностей потенціалів заземлення на процес вимірювання ПЗД. Якщо ПЗД і джерело сигналу мають різні потенціали землі, може виникнути паразитний електричний ланцюг з замиканням через землю. Такі ланцюги можуть бути причиною помилок у цифровому поданні виміряного сигналу. Більш того, якщо різниця потенціалів землі джерела сигналу і ПЗД велика, можна пошкодити вимірювальну систему.

Фільтрація. Використовувати фільтрацію можна для видалення небажаних компонент (перешкод) із прийнятого

сигналу. Велика частина перешкод створюється ланцюгами змінного струму, наприклад, блоком живлення комп'ютера або електроосвітлювальним обладнанням. Перешкоди від промислових кіл змінного струму з'являються на частоті 50 Гц. Для видалення такої перешкоди з сигналу застосовують фільтр нижніх частот із частотою зрізу менше ніж 50 Гц. Фільтрацію можна здійснювати як за допомогою апаратних засобів, так і програмно.

2.2 Введення та виведення аналогових даних

Оброблення цифрових сигналів має багато переваг, порівняно з аналоговими, тому перед обробленням у комп'ютері, аналогові сигнали перетворяться в дискретну форму. Цифровим називається такий сигнал, який може мати обмежену низку значень залежних і незалежних змінних. Незалежними змінними зазвичай є час або координати, залежними – амплітуда.

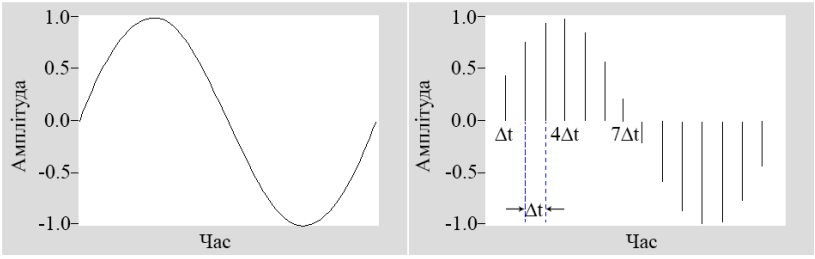
Цифрові (дискретні) сигнали оточують нас скрізь. Телефонні компанії використовують цифрові сигнали для передавання голосу. Радіо, телевізійні та звукові системи використовують поетапне перетворення сигналу в цифрову форму через її точність відтворення, можливість «придушення» шумів і широких можливостей оброблення сигналів. Знімки NASA віддалених планет і космічного простору часто піддаються цифровому обробленню для видалення шумів і виділення корисної інформації. Дані в економічній галузі, результати перепису населення і ціни акцій на фондових біржах – все це доступно в цифровому вигляді.

2.2.1 Дискретизація сигналів

Для оброблення аналогового сигналу насамперед необхідно перетворити його в дискретне подання. На практиці під цим розуміємо використання апаратного пристрою – аналого-цифрового перетворювача (АЦП). Розглянемо аналоговий сигнал *x(t)*, який дискретизується кожні *t* секунд. Часовий інтервал *t* називають інтервалом дискретизації або періодом дискретизації. Зворотня величина, 1/*t*, називається частотою дискретизації або частотою вибірки з одиницею вимірювання кількість вибірок за 1 секунду. Кожне з дискретних значень *x(t)* у моменти часу *t* = 0, *t*, 2 *t*, 3 *t*, тощо називаються вибірками. Отже, x (0), x (t), x (2t) утворюють повний набір вибірок. Сигнал *x(t)* може бути поданий дискретним набором вибірок:

$$\{x(0), x(t), x(2t), x(3t), \ldots, x(kt), \ldots\} \qquad (2.1)$$

На рисуноку 2.7 а показано аналоговий сигнал і відповідну йому оцифрованну версію (рис. 2.7 б), де вибірки задані в дискретні моменти часу.



а б

Рисунок 2.7 – Приклад оцифрування аналогового сигналу

Оцифрований сигнає це – дискретизована версія сигналу x (t). Звернемо увагу, що елементи послідовності *x = {x [i]}*

нумерують цілочисельними змінними, вони не містять будь-якої інформації про частоту вибірки. Знаючи лише значення вибірок, що містяться в *x*, ви не дізнаєтеся, з якою швидкістю проводилася вибірка (оцифрування).

Частота вибірки. Один із найбільш важливих параметрів вимірювальної системи з аналоговим введенням або виведенням є частота, з якою вимірювальний пристрій виконує вибірку вхідного сигналу або генерацію вихідного. Частота сканування або частота вибірки визначає, наскільки часто відбувається аналого-цифрове або цифро-аналогове перетворення. За більшої частоти вибірки збирається більше точок за одиницю часу, що дозволяє побудувати краще подання вихідного сигналу, ніж за низької частоти вибірки. Генерація одногерцевого сигналу з використанням 1000 точок на період за частоти 1000 вибірок в секунду створює набагато більш точне представлення, ніж використання 10 точок на період за частоти 10 виб/с.

Накладення частот. Занадто низька частота вибірки призводить до такого явища, як накладення частот (aliasing), що викликає спотворення в поданні аналогового сигналу. Недостатня швидкість оцифрування є причиною того, що сигнал виглядає так, ніби його частота відмінна від дійсної. Щоб уникнути накладення частот оцифрування виконують із частотою, більшою частоти самого сигналу.

На рисунку 2.8 показано задовільно оцифрований сигнал (1) і ефект накладення частот через недостатню частоту вибірки (2).

Для точного подання частоти сигналу під час вимірювання необхідно робити вибірки з частотою, більшою подвоєної максимальної частотної компоненти сигналу, відповідно до теореми Котельникова. Частота Найквіста – це максимальна частота сигналу, за якої його можна точно подати без ефекту накладення частот із цією частотою вибірки. Частота Найквіста дорівнює половині

частоти вибірки. У сигналах, що мають частотні компоненти, які перевищують частоту Найквіста, з'являться хибні низькочастотні складові. Частота цієї складової така, що дорівнює за модулем різниці між частотою вхідного сигналу і найбільш близькою частотою, такою, що дорівнює цілому числу, помноженому на частоту вибірки.

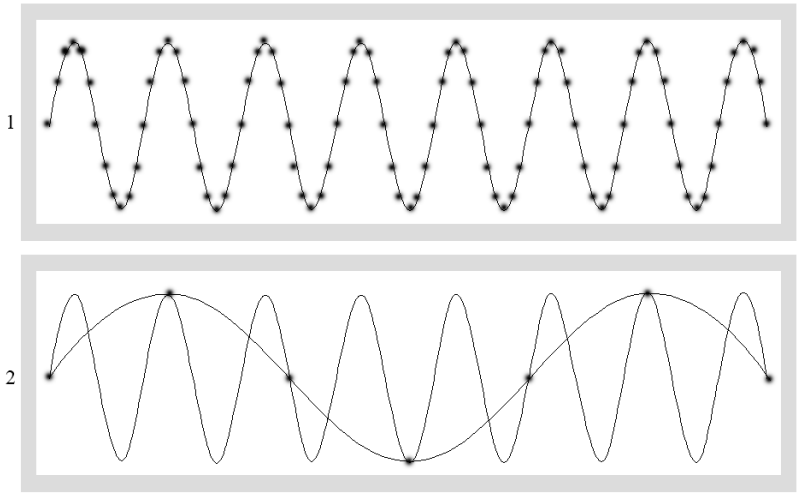


Рисунок 2.8 – Ілюстрація ефекту накладання частот: 1 – задовільно оцифрований сигнал; 2 – згладжування (накладання спектрів) унаслідок недостатньої частоти вибірки

Наприклад, припустимо, що частота вибірки *fs* дорівнює 100 Гц. Припустимо також, що вхідний сигнал містить компоненти з частотами: 25 Гц, 70 Гц, 160 Гц і 510 Гц. Частотні компоненти нижче частоти Найквіста (*fs*/2 = 50 Гц) оцифровуються правильно. Частотні компоненти вище частоти Найквіста з'являються як побічні. Наприклад, F1 (25 Гц) з'являється на правильній частоті, а F2 (70 Гц), F3

(160 Гц) і F4 (510 Гц) мають низькочастотні хибні компоненти на частотах 30 Гц, 40 Гц і 10 Гц відповідно, як показано на рисунку 2.9.

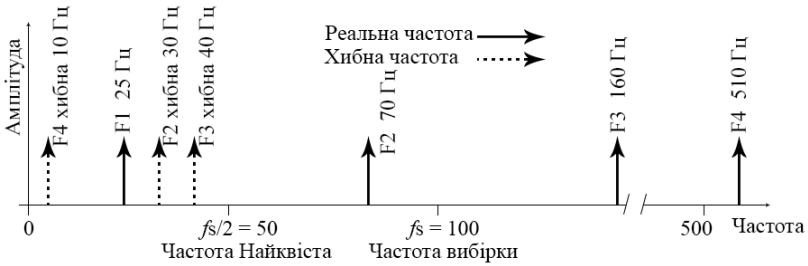


Рисунок 2.9 – Приклад появи хибних частот в оцифрованому сигналі

Для обчислення хибних частот використовується вираз

Хибна частота = Абсолютне значення (найбільш близька частота, така, що дорівнює цілому числу, помноженому на частоту вибірки, – частота вхідного сигналу)

Наприклад:
Хибна F2 = | 100 – 70 | = 30 Гц;
Хибна F3 = | (2) 100 – 160 | = 40 Гц;
Хибна F4 = | (5) 100 – 510 | = 10 Гц.

Визначення частоти вибірки. Можливо ви захочете здійснювати вибірку з максимально можливою частотою для цього вимірювального пристрою. Проте, якщо ви будете оцифровувати дуже швидко впродовж тривалих проміжків часу, то вам може не вистачити оперативної пам’яті або дискового простору для накопичення даних.

На рисунку 2.10 показано вплив різних частот оцифровування на сигнал.

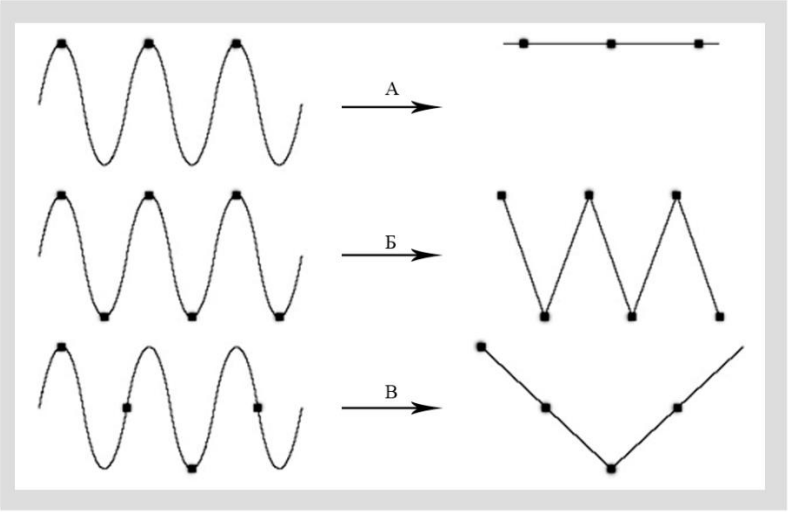


Рисунок 2.10 – Приклад впливу частоти дискретизації на подання сигналу

Позиція А: оцифрування синусоїдального сигналу частотою *f* і з такою самою частотою дискретизації *fs*. Унаслідок цього, одержані вибірки утворюють постійний сигнал. Проте, якщо ви збільшите частоту оцифрування до 2*fs*, то оцифрована осцилограма матиме ту саму частоту (або таку саму кількість циклів), що і вихідна осцилограма, але буде виглядати як трикутний сигнал, що показано на **позиції Б**. Значно збільшуючи частоту дискретизації порівняно з *fs*, можна більш точно відтворити осцилограму. На **позиції В** частота вибірки дорівнює 4*fs*/3. Оскільки в цьому разі частота Найквіста менше *fs*, то буде відтворюватися осцилограма з неправильною частотою та формою.

Теорема Котельникова є відправною точкою під час вибору достатньої частоти вибірки – вона повинна в два рази перевищувати максимальну частотну компоненту в

сигналі. На жаль, цієї частоти зазвичай не вистачає для практичних цілей. Сигнали, що трапляються в реальних системах, часто містять складові, що лежать вище частоти Найквіста. Це призводить до додавання помилкових компонент у точно оцифрований сигнал, що створює спотворені цифрові дані. Тому на практиці вибірку проводять із частотою, що багаторазово перевищує частоту вхідного сигналу. Для промислових вимірювань є звичайним перевищення в 5–10 разів.

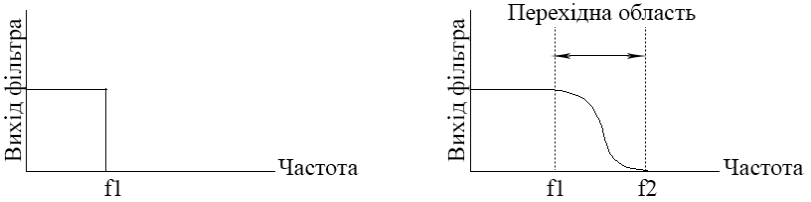
2.2.2 Фільтр захисту від накладення частот

У п. 2.2.1 ви переконалися, що частота дискретизації повинна, принаймні, в два рази перевищувати максимальну частоту сигналу. Іншими словами, максимальна частота вхідного сигналу повинна бути в два рази меншою або дорівнювати половині частоти вибірки. Проте як на практиці дізнатися, що ця умова точно виконана? Навіть якщо ви впевнені, що вимірюваний сигнал обмежений зверху за частотою, наведення паразитних сигналів (таких як силова мережа або інше електромагнітне випромінювання) можуть містити частоти, що перевищують частоту Найквіста. Ці частотні складові можуть внести помилкові компоненти в необхідний частотний діапазон, і тому призведуть до неправильних результатів оцифрування.

Щоб бути повністю впевненим, що частотні складові вхідного сигналу обмежені, перед АЦП застосовують фільтр низьких частот (ФНЧ-фільтр, що пропускає низькі частоти і послабляє високі). Цей фільтр називають фільтром захисту від накладення частот, оскільки він запобігає появі хибних компонент під час дискретизації, пригнічуючи високі частоти (вище частоти Найквіста). Фільтри захисту від накладення спектрів – аналогові

фільтри. На рисунку 2.11 показано перехідну характеристику фільтра захисту від накладення спектрів.

Ідеальний фільтр захисту від накладення спектрів пропускає всі необхідні частоти (нижче f1) і відсікає небажані (рис. 2.11 а). Проте ідеальних фільтрів не існує. На практиці робота фільтра виглядає, як показано на рис. 2.11 б. Реальні фільтри захисту від накладення спектрів пропускають всі частоти < f1 і відсікають > f2. Область між f1 і f2 називають перехідною смугою фільтра, в якій відбувається поступове ослаблення частотних компонент. Звичайно хотілося б, щоб проходили лише сигнали з частотами < f1, адже сигнали в перехідній смузі можуть викликати появу помилкових компонент. Тому на практиці необхідно оцифровувати з частотою, що перевищує подвоєну максимальну частотну компоненту з перехідною смугою. Отже, частота вибірки більше ніж у два рази перевищує максимальну частоту вхідного сигналу (f1).



а б

Рисунок 2.11. – Перехідна характеристика фільтра захисту від накладення спектрів

2.2.3 Архітектури пристроїв збирання даних

Кількість і розміщення компонентів використовуваного пристрою збирання даних залежать від його типу. Архітектура пристрою впливає на спосіб дискретизації сигналу. DAQ-пристрої аналогового введення можуть мати одну з двох найбільш поширених архітектур, показаних на

рисунку 2.12.

Архітектури інтервальної та циклічної вибірки складаються з одного мультиплексора (комутатора), інструментального підсилювача і АЦП (рис. 2.12 а). У такій схемі всі вхідні канали повинні спільно використовувати один АЦП. Використання лише одного АЦП робить подібну архітектуру ефективною з точки зору ціни, тому вона застосовується в пристроях масового сегмента. Архітектура одночасної вибірки складається з інструментального підсилювача і АЦП для кожного з каналів (рис. 2.12 б). Хоча ця архітектура дорожча порівняно з тими, що використовують один АЦП на всі канали, вона дозволяє виконувати одночасні вибірки з різних каналів, що дозволяє синхронізувати вхідні сигнали з різних каналів.

Термінологія, яку застосовують під час дискретизації сигналів:

– вибірка на канал за 1 секунду – кількість вибірок, одержаних за одну секунду з одного каналу;

– тактовий генератор вибірки (ТГВ, Sample Clock) – послідовність імпульсів, що використовуються для запуску збирання даних. Кожного разу, коли тактовий генератор вибірки видає імпульс, береться одна вибірка з одного каналу;

– тактовий генератор перетворювача аналогового введення (ТГП, AI Convert Clock) – послідовність імпульсів, що використовуються для запуску АЦП;

– тривалість вибірки (Sample Duration) – час, який займає одержання одного набору вибірок із каналів. Використовується така формула для обчислення тривалості вибірки:

Тривалість вибірки = (кількість каналів – 1) · період імпульсів ТГП.

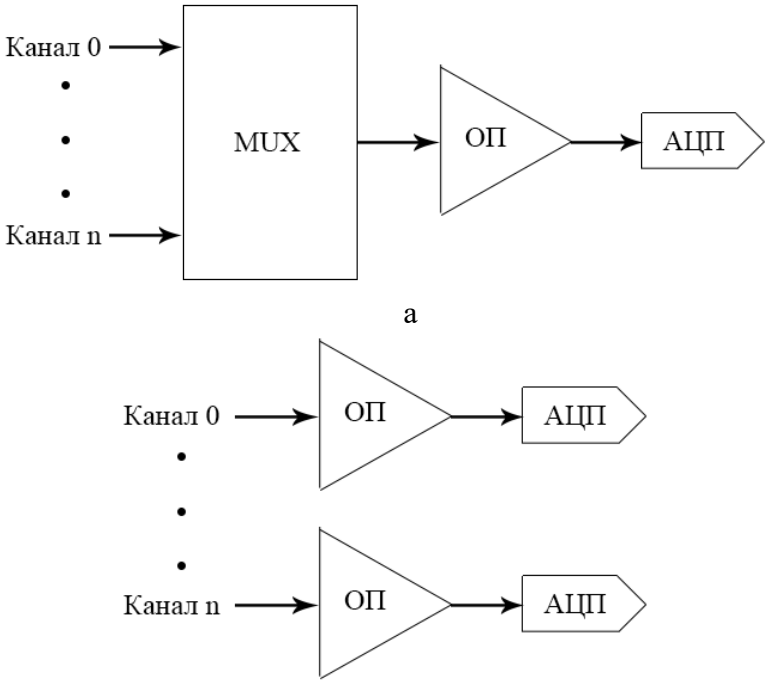


Рисунок 2.12 – Архітектура найбільш поширених DAQ-пристроїв аналогового введення

Інтервальна вибірка. Під час оцифрування сигналу ви можете використовувати різні способи дискретизації: інтервальний, циклічний і одночасний. На рисунку 2.13 показано приклад інтервальної вибірки.

У найбільш поширеному методі інтервальної вибірки (interval sampling) всі канали пристрою використовує один АЦП. Водночас для управління мультиплексором (Multiplexer – MUX) використовуються тактові генератори вибірки й перетворювача аналогового введення. Для того щоб зрозуміти, як взаємодіють ці генератори, розглянемо приклад одержання сигналів із двох каналів. Якщо

генератор вибірки дає сигнал про початок збирання даних, мультиплексор під'єднує перший канал до АЦП, і одноразово спрацьовує тактовий генератор перетворювача. Як тільки ТГП посилає імпульс, АЦП одержує одну точку даних із першого каналу. Перед тим як ТГП знову пошле імпульс, мультиплексор під'єднає другий канал до АЦП. Як тільки це відбудеться, з наступним імпульсом ТГП АЦП візьме одну точку даних уже з другого каналу. Під час завершення часу (тривалості) вибірки, ТГВ знову генерує імпульс, і цикл повторюється. ТГВ задає частоту, з якою пристрій виконує вибірку даних з усіх каналів. А тактовий генератор перетворювача фактично керує одержанням вибірок. Оскільки під час інтервальної вибірки використовуються ТГВ і ТГП, пристрій може робити вибірку з каналів за короткий проміжок часу.

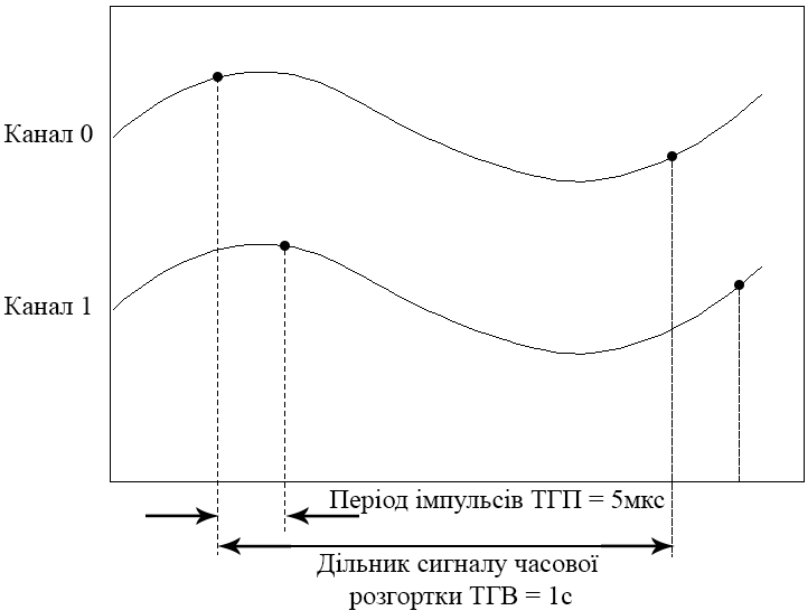


Рисунок 2.13 – Приклад інтервальної вибірки

На рисунку 2.13 пристрій виконує вибірку з кожного каналу кожну секунду, а затримка між вибірками з різних каналів дорівнює всього лише 5 мкс (це визначається періодом тактового генератора перетворювача аналогового вводу). Отже, ви можете досягти майже одночасної вибірки з каналів, маючи лише один АЦП, що вигідно з точки зору вартості DAQ-пристрою.

Циклічна вибірка. За циклічної вибірки (round-robin sampling) також використовується один АЦП для всіх каналів. Різниця між цим методом та інтервальною вибіркою полягає в тому, що за циклічної вибірки не використовується генератор розгортки (Scan clock) – традиційна назва ТГВ. Тактовий генератор каналів (channel clock) запускає розгортку й визначає час між вибірками. Рисунок 2.14 ілюструє приклад циклічної вибірки.

У прикладі з інтервального вибіркою пристрій запускав оцифрування кожного каналу з інтервалом за 1 секунду. Каналів збирання даних було два. В цьому прикладі умови ті самі, проте тут лише один генератор, так що точки даних повинні бути розподілені рівномірно. Єдиний спосіб розподілити дані рівномірно з частотою одна вибірка за 1 секунду на один канал і дві вибірки на два канали в секунду – це використовувати ТГП із частотою дві вибірки за 1 секунду. Різниця між цим прикладом та інтервальною вибіркою в тому, що тривалість вибірки тепер дорівнює 0,5 секунди замість 5 мкс.

Під час використання інтервальної вибірки точки даних із різних каналів беруться в близькі моменти часу. Під час використання циклічного оцифрування вибірки розділені великим проміжком часу. Хоча циклічна вибірка простіша, так як використовує всього лише один генератор, її можна використовувати, лише, якщо часові відношення між сигналами не мають великого значення. Таку архітектуру оцифрування даних можна знайти в застарілих і пристроях

бюджетного сегмента.

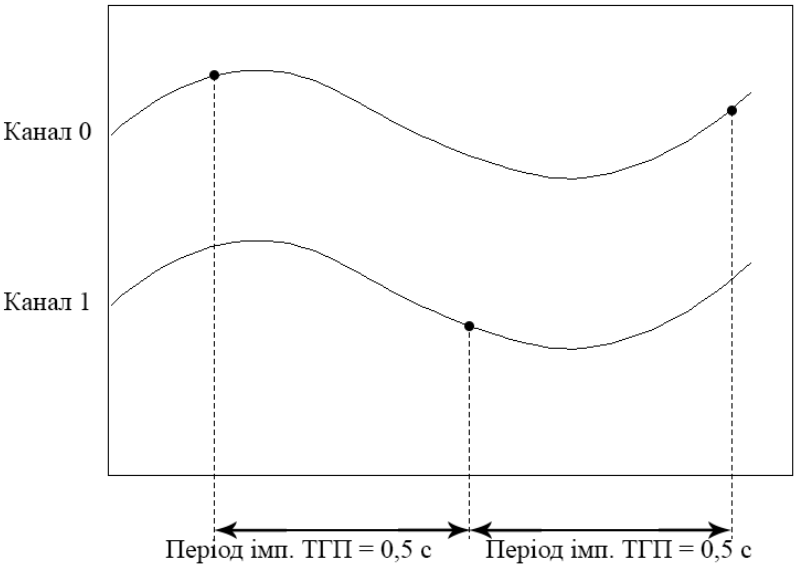
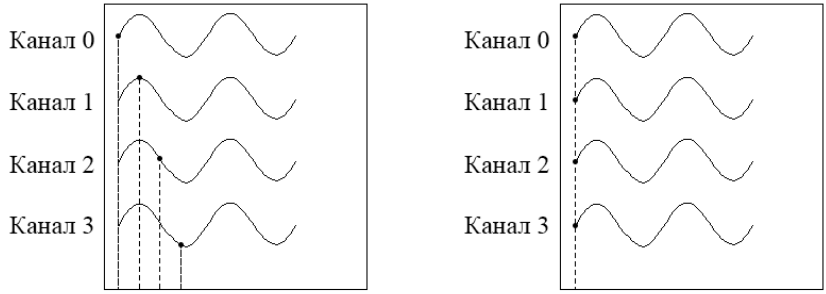


Рисунок 2.14 – Приклад циклічної вибірки

Одночасна вибірка. Якщо часові співвідношення між сигналами відіграють важливу роль, можна використовувати інтервальну вибірку, але в ряді випадків інтервальна розгортка не може з достатньою точністю забезпечити часових співвідношень між сигналами. У цьому разі необхідно використовувати одночасну вибірку, як показано на рисунку 2.15.

У разі одночасної вибірки (simultaneous sampling) використовують по одному АЦП для кожного з каналів, так що ви можете оцифровувати сигнали в усіх каналах одночасно. Хоча це вимагає більш дорогої архітектури, ніж під час інтервальної вибірки, зате виключається запізнювання між каналами, обумовлене спільним використанням АЦП всіма каналами. Оскільки за такої дискретизації вибірки з каналів відбуваються одночасно,

для завдання частоти вибірки досить лише ТГВ.



а б

Рисунок 2.15 – Порівняння інтервальної (а) та одночасної (б) вибірки

Порівняємо всі три типи дискретизації, визначаючи фазовий зсув, що виникає під час оцифрування чотирьох 50 кГц сигналів із частотою 200 кГц. У разі циклічної дискретизації всі вибірки повинні бути рівномірно розподілені в часі, що є причиною 15 мкс затримки між часом вибірки в каналі 0 і часом вибірки в каналі 3. Це відповідає фазовому зсуву 270 градусів. За інтервальної дискретизації будемо вважати, що міжканальна затримка становить 5 мкс. Знову одержуємо 15 мкс затримку між нульовим і третім каналами. У разі одночасної дискретизації затримка між крайніми каналами становитиме три наносекунди, що призведе до фазового зсуву 0,054 градуса. Отже, одночасна вибірка має величезну перевагу, зберігаючи часові співвідношення між сигналами, хоча й досягається це більш високою вартістю.

2.2.4 Архітектура систем виведення аналогових сигналів

Більшість пристроїв мають цифро-аналоговий перетворювач (ЦАП) для кожного з каналів аналогового виведення (рис. 2.16). Цифро-аналогові перетворювачі оновлюються одночасно, так що для операцій виведення необхідний лише один тактовий генератор. Тактовий генератор аналогового виведення створює сигнал регенерації (update clock). Вихідні сигнали каналів аналогового виведення синхронізовані так само, як і канали аналогового введення під час одночасної вибірки сигналів.

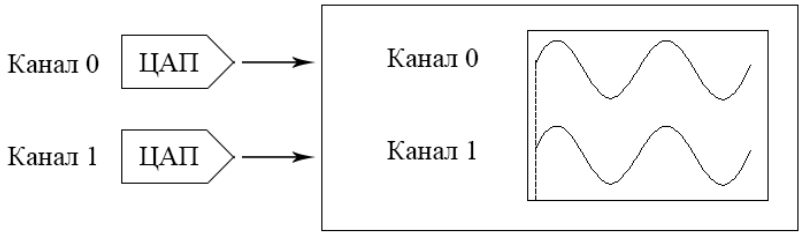


Рисунок 2.16 – Приклад цифро-аналогового перетворювача DAQ-пристроїв

ЦАП має робочий діапазон, що визначається опорною напругою. За опорну напругу може бути внутрішній або зовнішній сигнал. Робочий діапазон ЦАП можна встановити однополярним або біполярним.

Біполярний робочий діапазон. Біполярний сигнал має діапазон, що містить додатні та від'ємні значення. Якщо ви встановите пристрій у режим біполярного сигналу, то робочий діапазон ЦАП визначати потрібно так:

Максимальна напруга = + *Vref*;

Мінімальна напруга = -*Vref*.

Наприклад, якщо ви використовуєте внутрішню опорну напругу +10 В, то робочий діапазон ЦАП буде встановлено

від –10 до +10 В. Однак якщо сигнал буде змінюватися від –5 до +5 В, то ви не зможете використовувати максимальну роздільну здатність ЦАП. Щоб її довести до максимуму, можна встановити внутрішню опорну напругу величиною +5В. Тепер робочий діапазон ЦАП буде від –5 до +5 В, збігаючись із діапазоном зміни сигналу, і ви будете повністю використовувати роздільну здатність ЦАП для генерації сигналу.

Однополярний робочий діапазон. Однополярний сигнал має діапазон змін, що містить лише позитивні значення. Якщо ви встановите пристрій у режим однополярного сигналу, то робочий діапазон ЦАП визначати потрібно так:

Максимальна напруга = + Vref;
Мінімальна напруга = 0 В.

2.3 Введення та виведення дискретних (цифрових) сигналів

Цифрові лінії DAQ пристрої приймають і генерують ТТЛ сумісні сигнали. ТТЛ сигнал має два стани: високий і низький логічні рівні, як показано на рисунку 2.17. Низький логічний рівень відповідає сигналам із напругою від 0 В до +0,8 В. Сигнали з високим логічним рівнем повинні мати напругу від +2 В до +5 В. Сигнали в діапазоні від +0,8 В до +2,0 В є невизначеними.

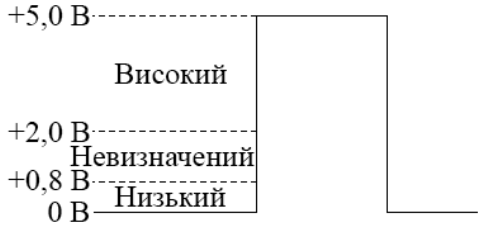


Рисунок 2.17 – ТТЛ-сумісний сигнал

Щоб бути впевненим, що цифрові лінії правильно вимірюють сигнал, переконайтеся, що рівень напруги сигналу ніколи не потрапляє в діапазон від +0,8 В до +2,0 В.

Термінологія цифрового введення / виведення. Нижче наведені кілька часто вживаних термінів під час опису операцій цифрового введення / виведення:

– **біт** (Bit) – найменша одиниця даних, яку використовують під час дискретних операцій. Біти – виконавчі одиниці, тобто вони можуть набувати значень 1 або 0;

– **лінія** (Line) – окремий сигнал у цифровому порті. Різниця між бітом і лінією полягає в тому, що біт пов'язують із передаванням реальних даних, а коли говорять про лінію, то мають на увазі пристрій, яким передається біт. Однак терміни лінія та біт дещо взаємозамінні. Наприклад, 8 бітний порт це те саме, що порт, який складається з восьми ліній;

– **порт** (Port) – набір цифрових ліній. Зазвичай лінії згруповані в 4-бітний або 8-бітний порт. Старі DAQ пристрої мають два 4-бітних порти, а більш сучасні мають один 8-бітний порт;

– **розрядність порту** (Port Width) – кількість ліній у порті;

– **маска** (Шаблон) (Mask) – визначає, які із цифрових ліній ігноруються. Наприклад, якщо ви записуєте в порт, але не хочете записувати в усі лінії, то можете встановити маску для пропуску відповідних ліній.

2.3.1 Рахункові сигнали

Лічильники працюють із ТТЛ-сумісними сигналами (рис. 2.18). Максимальний час наростання / спадання сигналу – 50 нс.

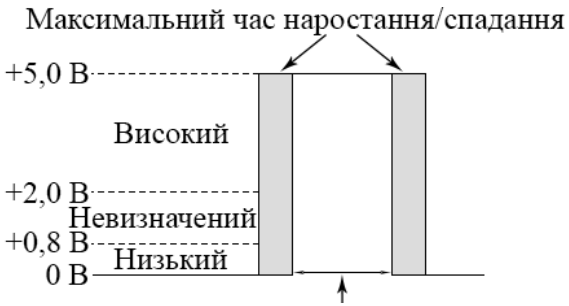


Рисунок 2.18 – Сигнал лічильника

Цифрові пристрої введення / виведення можуть установлювати або відстежувати стан у цифровий лінії. На відміну від них, лічильники мають відношення не лише до стану сигналу, а й переходу з одного стану в інший. Лічильник може детектувати наростаючі фронти (перехід зі стану з низьким логічним рівнем у стан із високим логічним рівнем) і спадні фронти (перехід зі стану з високим логічним рівнем у стан із низьким логічним рівнем). Існує два важливих параметри, що стосуються до детектування наростаючих і спадних фронтів, а саме, час наростання / спадання й мінімальна тривалість імпульсу. Час наростання / спадання характеризує, наскільки швидко сигнал переходить з одного стану в інший. Існує деяка мінімальна затримка часу, через який лічильник знову зможе зафіксувати фронт після детектування наростаючого або спадного фронту сигналу. Цю затримку називають мінімальної тривалістю імпульсу. Мінімальна тривалість залежить від використовуваної мікросхеми лічильника. Для визначення мінімальної тривалості імпульсу, необхідної лічильником будь-якого DAQ пристрою, подивіться документацію до цього пристрою.

Існує п’ять різних типів операцій із використанням лічильника: підрахунок фронтів, генерація імпульсів,

вимірювання параметрів імпульсу, вимірювання частоти й вимірювання положення.

Лічильник має такі основні компоненти:

– **регістр відліків** (Count Register) – зберігає поточний відлік лічильника. Цей регістр можна опитувати програмно;

– **інформаційний вхід** (Source) – сигнал на цьому вході може змінити поточний відлік, записаний у регістрі. Лічильник чекає наростаючих або спадного фронтів саме на цьому вході. Вибір типу фронту, що викликає зміна відліку, здійснюється програмно. Тип вибраного фронту називають активним фронтом сигналу. Коли на інформаційному вході приймається активний фронт, відлік змінюється. Чи буде активний фронт збільшувати або зменшувати поточний відлік, також налаштовується програмно. Інформаційний сигнал повинен бути ТТЛ-сумісним;

– **вхід дозволу** (Gate) – сигнал на цьому вході визначає, чи буде активний фронт на інформаційному вході (Source) змінювати відлік. Підрахунок може відбутися, коли рівень напруги на цьому вході високий, низький або знаходиться в проміжку між різними комбінаціями наростаючих і спадного фронтів. Налаштування цього входу здійснюють програмно. Сигнал дозволу нагадує маску в цифровий лінії під час операцій дискретного введення / виведення, оскільки дозволяє фіксувати або ігнорувати активні фронти на інформаційному вході;

– **вихід** (Out) – на ньому генеруються імпульси, серія імпульсів, або, по-іншому, послідовність імпульсів. Вихідний сигнал ТТЛ-сумісний.

Контакти лічильника. Для операцій аналогового введення, аналогового виведення й цифрового введення / виведення виділені спеціальні контакти. Однак лічильники для своїх операцій використовують комбінацію контактів PFI (Programmable Function Input – програмовані

функціональні входи) та спеціально виділених контактів. Вихідні контакти лічильників використовують винятково для генерації імпульсів на виході лічильника. Контакти інформаційного входу та входу дозволу є PFI-контактами. Крім роботи як входів лічильника, вони можуть бути використані й для інших цілей. Можливість використання одного контакту для різних цілей забезпечує більшу гнучкість у розробленні.

Терміни, важливі для розуміння під час роботи з лічильниками:

– **максимальний відлік** (Terminal Count) – останній відлік перед тим, як лічильник досягне 0. Наприклад, коли лічильник, що збільшує відліки, досягне свого максимального значення, то говорять, що він досяг кінцевого відліку. Подальше збільшення відліку змусить лічильник відкотитися назад і почати підрахунок з 0;

– **роздільна здатність** (Resolution) – максимальна кількість відліків лічильника, виражене в бітах. Ця формула дозволяє обчислити максимальне значення лічильника, беручи до увагийого роздільну здатність:

$$максимальне\ значення = 2^{роздільна\ здатність} - 1.$$

Лічильники найчастіше мають такі роздільні здатності 16, 24 або 32 біти;

– **часова розгортка** (Timebase) – сигнал заданої частоти, що генерується DAQ пристроєм. Часові розгортки зазвичай мають частоти в діапазоні від 100 Гц до 80 МГц. Усередині DAQ пристрою є лінія, по якій розгортку можна подати на інформаційний вхід лічильника для використання сигналу відомої частоти.

2.3.2 Підрахунок фронтів імпульсів

Підрахунок фронтів імпульсів є найбільш часто використовуваної операцією лічильника. У цьому режимі основну увагу приділяють вимірюванню сигналу на інформаційному вході. Операція найпростішого підрахунку фронтів ґрунтується на основному визначенні лічильника (рис. 2.19). Активні фронти сигналу на інформаційному вході збільшують значення регістра відліків. Активним фронтом можна програмно призначати наростаючий або спадаючий фронти. Вхід дозволу та вихід лічильника в цьому режимі не використовують.

Вимірювання часу – це один з різновидів найпростішого підрахунку фронтів. Під час виконання найпростішого підрахунку фронтів параметри джерела сигналу зазвичай невідомі. Для вимірювання параметрів якраз і використовують лічильник (рис. 2.20). Під час вимірювання часу джерелом є часова розгортка відомої частоти. Знання частоти часової розгортки можна використовувати для вимірювання проміжків часу. У цьому режимі на інформаційний вхід (Source) лічильника замість вимірюваного сигналу подають часову розгортку.

Наступна формула дозволяє обчислити час, що пройшов:

Час = (значення регістра відліків) × (період часової розгортки),

де *період часової розгортки = 1 / частота часової розгортки*.

Єдина відмінність між вимірюванням часу та найпростішим підрахунком фронтів полягає в сигналі, що подається на інформаційний вхід лічильника.

Вимірювання часу. Якщо лічильник налаштований для найпростішого підрахунку фронтів або вимірювання часу, то значення відліків зростає, коли на вхід надходить

активний фронт. У наступному прикладі як активний фронт обрано наростаючий. Значення відліків збільшується кожного разу, коли надходить наростаючий фронт (рис. 2.21).

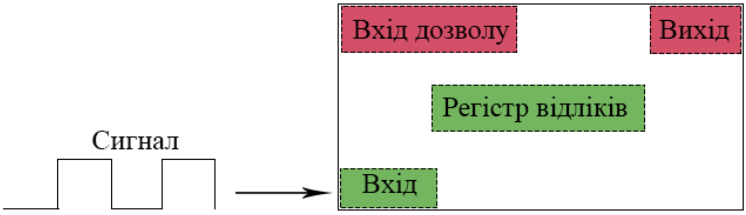


Рисунок 2.19 – Підрахунок фронтів імпульсів

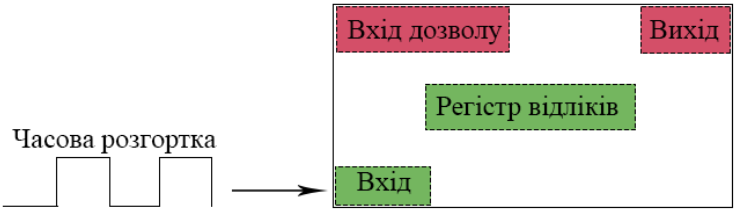


Рисунок 2.20 – Підрахунок часових інтервалів

Рисунок 2.21 – Підрахунок часу

Зверніть увагу, що відлік не збільшиться до тих пір, поки лічильник не буде в стані готовності. Лічильник має фіксоване число, до якого він може робити відлік і яке визначається його роздільною здатністю. Наприклад, 24-бітний лічильник може рахувати до 16777215. Коли такий лічильник досягне величини 16777215, тоді буде досягнутий кінцевий відлік. Наступний активний фронт змусить лічильник відкотитися назад і почне відлік із 0.

Генерація імпульсів. Лічильник не лише вимірює ТТЛ-сигнали, він також і генерує їх. Використання лічильника для генерації ТТЛ сигналу приводить до генерації імпульсів. Сигнал, показаний на рисунку 2.22, одержаний на виході лічильника. Згенерований сигнал може бути одиночним імпульсом або мати послідовність імпульсів. Для допомоги в генерації імпульсів лічильник використовує сигнал часової розгортки на своєму інформаційному вході.

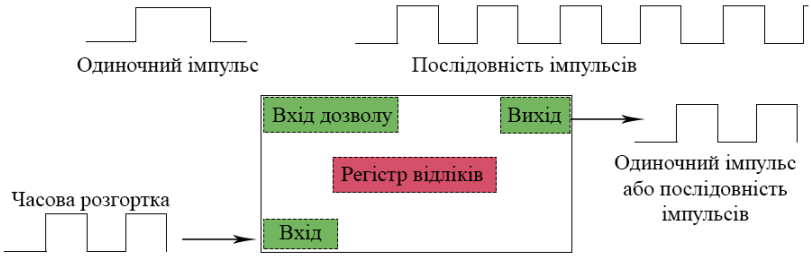


Рисунок 2.22 – Генерація сигналів за допомогою лічильника

2.3.3 Параметри імпульсів

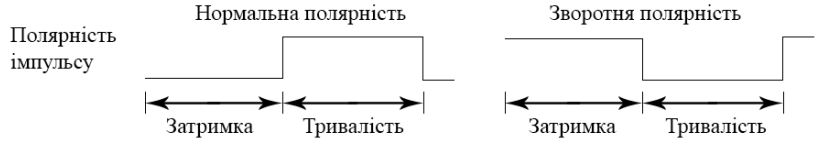
Для генерації імпульсу ви повинні розуміти, які параметри має імпульс. Імпульс складається з двох частин: затримки (delay) та тривалості (width). Затримка – перша стадія генерації імпульсу, а тривалість – друга. Затримка і тривалість завжди мають протилежні логічні рівні.

Наприклад, якщо затримка має низький логічний рівень, тоді тривалість повинна мати високий логічний рівень. Імпульс може мати нормальну та зворотну полярність. У імпульсу нормальної полярності затримка має низький логічний рівень, а тривалість – високий. У імпульсу зворотної полярності навпаки, затримка має високий логічний рівень, а тривалість – низький.

Періодом імпульсів називають час, необхідний для завершення одного циклу, так що, складаючи час затримки та тривалість, ми одержимо період імпульсів. Знаючи період імпульсів, легко одержати частоту проходження імпульсів, що дорівнює величині, зворотній періоду імпульсів.

Затримка та тривалість не завжди рівні між собою, тому необхідна властивість імпульсу, який допомагає визначити, що більше, затримка чи тривалість. Параметр, що використовують для цього, називають прогальністю (duty cycle). На рисунку 2.23 наведено відповідну формулу. Прогальність – це число, яке змінюється від 0 до 1. Його часто перетворюють на відсотки. Імпульс, у якого затримка дорівнює тривалості, матиме прогальність, рівну 0,5 % або 50 %.

Коефіцієнт заповнення більше 50 % означає, що тривалість більше затримки, якщо ж коефіцієнт заповнення менше 50 %, то затримка більша ніж тривалість.



Період імпульсів = Затримка + Тривалість
Частота = 1 / Період імпульсів
Прогальність = Тривалість / Період імпульсів

Рисунок 2.23 – Параметри імпульсів

Вимірювання параметрів імпульсів. Під час вимірювання імпульсу досліджуваний сигнал використовують як сигнал дозволу (Gate), а на інформаційний вхід (Source) подається часова розгортка відомої частоти. Ця ситуація показана на рисунку 2.24. Знаючи частоту розгортки та значення в регістрі відліків

можна визначити такі характеристики імпульсів на вході дозволу, як період і тривалість.

Вимірювання періоду імпульсу. Вимірюючи період, ви повинні підраховувати кількість активних фронтів на вході лічильника. Однак, на відміну від простого підрахунку фронтів, збільшення відліків буде відбуватися лише тоді, коли один із періодів досліджуваного сигналу надійде на вхід дозволу. Рисунок 2.25 ілюструє процес вимірювання періоду, що запускається й зупиняється наростаючим фронтом сигналу на вході дозволу (Gate).

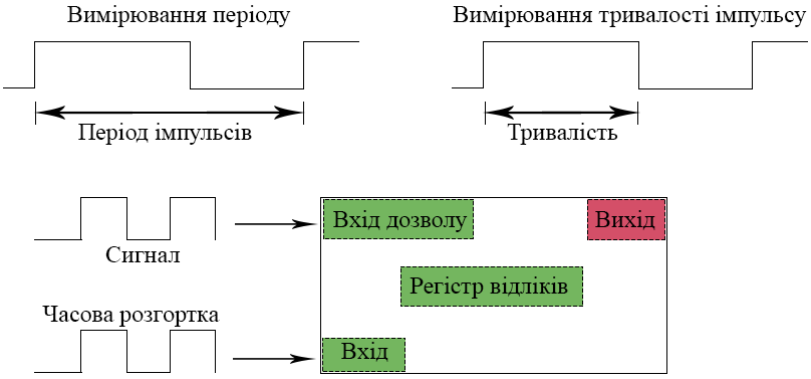


Рисунок 2.24 – Вимірювання параметрів імпульсів

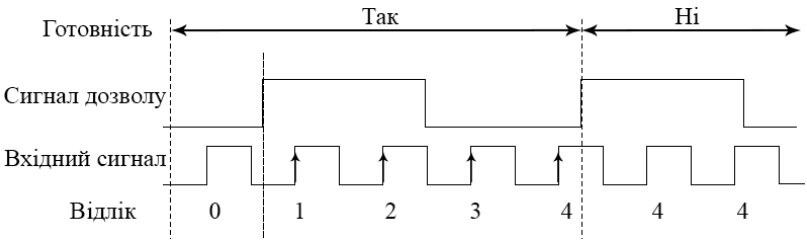


Рисунок 2.25 – Вимірювання періоду імпульсів

Ви можете також запускати і зупиняти підрахунок спадними фронтами сигналу. Відлік відображають

кількість наростаючих фронтів сигналу на інформаційному вході за період часу між двома наростаючими фронтами сигналу на вході дозволу. Отже, для вимірювання періоду необхідний сигнал із двома наростаючими або спадними фронтами. Окремий імпульс має лише один наростаючий та один спадний фронт, тому ви не зможете виміряти період за одним імпульсом.

У попередньому прикладі на один період сигналу дозволу сталося чотири відліки. Згадайте, що сигналом на вході лічильника є часова розгортка відомої частоти. Припустимо, що ми використовуємо розгортку з частотою 100 кГц. Формулу для обчислення періоду записують у такий спосіб:

Період імпульсів = кількістьть відліків × (1 / частота вхідного сигналу).

У попередньому прикладі з 100 кГц розгорткою формула дасть такий результат:

Період імпульсів = 4 × (1/100 000) = 0,04 мілісекунди.

Вимірювання півперіоду дуже схоже на вимірювання періоду, але тепер ми вимірюємо час між двома послідовними фронтами. Формула для обчислення півперіоду виглядає так:

Напівперіод імпульсів = к-ть відліків × (1 / (2 × частота вхідного сигналу)).

У попередньому прикладі з 100 кГц розгорткою формула дасть такий результат:

Період імпульсів = 4 × (1/200000) = 0,02 мілісекунди.

Вимірювання тривалості імпульсу. Вимірювання тривалості імпульсу дуже схоже на вимір періоду імпульсів. Різниця полягає в тому, де ви зупиняєте підрахунок. Під час вимірювання періоду ви запускали й зупиняли підрахунок двома наростаючими фронтами сигналу на вході дозволу. У разі вимірювання тривалості відлік триває лише під час тривалості імпульсу, так що ви запускаєте підрахунок

одним фронтом, а зупиняєте фронтом іншого типу. Значення числа відліків наростає лише за час між двома різнотипними фронтами, як це проілюстровано на рисунку 2.26.

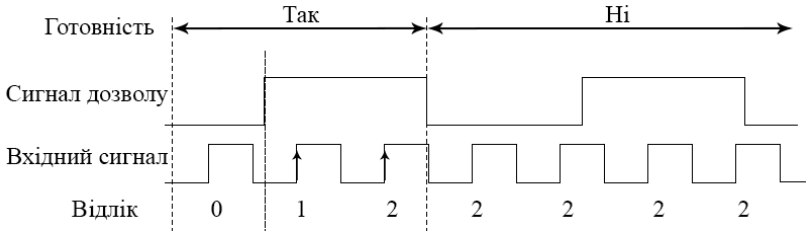


Рисунок 2.26 – Вимірювання тривалості імпульсів

Формула для обчислення тривалості імпульсу збігається з формулою для періоду:

Тривалість імпульсу = к-ть відліків × (1 / частота вхідного сигналу).

У попередньому прикладі сигнал із частотою 100 кГц приводить до такої тривалості імпульсів:

Тривалість імпульсу = 2 ·· (1 / 100,000) = 0.02 мілісекунди.

Значення 0,02 мілісекунди – це половина періоду, так що сигнал на вході дозволу має прогальність 50 %.

Вимірювання частоти. Розглянемо три способи вимірювання частоти ТТЛ імпульсів із використанням одного або декількох лічильників. Частота осцилограми дорівнює її зворотному періоду в будь-який даний момент часу. Отже, найпростіший спосіб вимірювання частоти полягає у взятті зворотної величини від виміряного періоду. Два інші способи вимірювання частоти є необхідними в разі, коли під час наближення частоти сигналу на вході дозволу до частоти часової розгортки лічильника, помилка вимірювання частоти першим способом сильно наростає.

Період. Перший спосіб вимірювання частоти – це

вимірювання періоду. Як тільки період знайдений, візьміть від нього зворотне значення для обчислення частоти. Перевагами зазначеного методу є використання лише одного лічильника й простота виконання, однак цей метод упевнено спрацьовує тільки для порівняно повільних сигналів, оскільки точність вимірювання періоду залежить від кількості фронтів вхідного сигналу, які прийшли за один період сигналу на вході дозволу.

Помилка синхронізації. Під час наближення частоти сигналу на вході дозволу до частоти вхідного сигналу, на вимірювання періоду сильно впливає помилка синхронізації. Наприклад, розглянемо вимірювання періоду з використанням внутрішньої часової розгортки з частотою 20 МГц, поданої на інформаційний вхід лічильника. Припустимо, що сигнал на вході дозволу має частоту 5 МГц, що становить 25 % частоти вхідного сигналу. Рисунок 2.27 показує три можливих сценарії розвитку подій, коли перший та останній фронти вхідного сигналу можуть бути включені, а можуть бути й не внесені до вимірювань періоду.

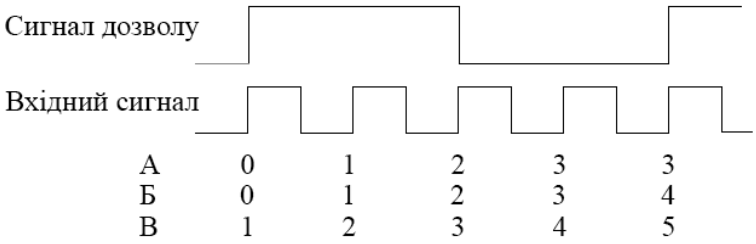


Рисунок 2.27 – Помилка синхронізації під час вимірювання параметрів імпульсів

У першому сценарії (А) під час вимірювання губляться перший та останній вхідні фронти, приводячи до підрахунку лише трьох фронтів. У другому сценарії (Б) захоплюються чотири перших фронти, і втрачається

останній. У третьому сценарії (В) усі п’ять фронтів вхідного сигналу будуть підраховані. Другий сценарій, очевидно, призводить до самого точному результату. Але, оскільки фронти вхідного сигналу практично збігаються в часі з фронтами сигналу дозволу, лічильник рівноймовірно може виконати вимірювання за одним із трьох вищевказаних сценаріїв.

Вимірювання імпульсів завжди має похибку на ± 1 період вхідного сигналу, якою можна знехтувати, коли один період вхідного сигналу становить максимум 1 % виміряного періоду імпульсів. Однак один період вхідного сигналу в нашому прикладі становить від 33 % до 20 % виміряного, що призводить до помилки синхронізації, якої можна уникнути, вибираючи іншу схему вимірювання. Таблиця показує, як відбувається вимірювання періодів сигналів із частотами 50 кГц і 5 МГц.

Таблиця 2.4 – Помилка синхронізації під час вимірювання сигналів різної частоти

Дійсна частота	**Кількість 50 нс циклів**	**Помилка вимірювання «+1» цикл**	**Помилка Вимірювання «-1» цикл**	**Частота з помилкою «+1» цикл**	**Частота з помилкою «-1» цикл**
50 кГц	400	401	399	49,88 кГц	50,13 кГц
5 МГц	4	5	3	4 МГц	6,67 МГц

Усереднення. У другому методі вимірювання частоти використовують два лічильники: один – для генерації послідовності імпульсів із відомою частотою, а інший – для вимірювання періоду. Лічильник 1 вимірює період, використовуючи як вхідний зовнішній сигнал, а не внутрішню часову розгортку. Сигнал дозволу для лічильника 1 надходить із виходу лічильника 0, генерує послідовність імпульсів. Оскільки ви знаєте частоту

вихідного сигналу з лічильника 0, ви точно знаєте тривалість циклів сигналу дозволу для лічильника 1. Знаючи кількість фронтів сигналу, що надійшов на вхід лічильника 1, ви можете знайти частоту, ділячи період, виміряний лічильником 1, на період сигналу дозволу. Наприклад, вихідний сигнал лічильника 0 має частоту 10 Гц, період сигналу дозволу дорівнює 0,1 с. Якщо впродовж цього часу ви нарахували 100 фронтів на вході лічильника, то знаєте частоту вхідного сигналу лічильника 1, а саме, вона дорівнює (100 ± 1) /0,1 або 1 000 ± 10 Гц. Часто цей метод називають High Frequency with 2 Counters (Висока частота з двома лічильниками).

Метод поділу вниз. У третьому методі вимірювання частоти також використовують два лічильники, лише лічильник, що генерує послідовність імпульсів (лічильник 0), використовує зовнішній сигнал як вхідний, а лічильник, що вимірює період (лічильник 1), використовує як вхідний внутрішню часову розгортку. Як і в методі усереднення, тут використовують послідовність імпульсів із виходу лічильника 0 як сигнал дозволу для лічильника 1.

Перевагою методу розподілення вниз є менша похибка, порівняно з двома попередніми методами. Наприклад, припустимо, що запрограмували лічильник 0 на генерацію послідовності імпульсів із технічними параметрами 5 і 5. Це означає, що затримка та тривалість дорівнюють п’яти періодам вхідного сигналу, тобто період результуючої послідовності імпульсів складається з 10 періодів вхідного сигналу (частоту вхідного сигналу розділили на 10). У цьому прикладі лічильник 1 налаштований на вимірювання періоду з використанням внутрішньої часової розгортки з частотою 20 МГц як вхідний сигнал. Якщо лічильник 1 зафіксував 100 фронтів вхідного сигналу за час, що дорівнює періоду сигналу дозволу, то можна зробити висновок, що період сигналу дозволу дорівнює 5 мкс

(50 нс · 100 фронтів). Отже, можна прийти до висновку, що зовнішній сигнал, поданий на вхід лічильника 0 має період 0,5 мкс або частоту 2 МГц.

Запишемо ці міркування у вигляді рівняння:

F = (технічний параметр 1 + технічний параметр 2)· часова розгортка / (# фронтів вхідного сигналу ± 1).

У цьому прикладі:

F = (5 + 5)· 20000000 / (100 ± 1);
F = від 200000000/101 до 200000000/99;
F = від 1,980,198 до 2,020,202 Гц.

Також цей метод називають Large Range with 2 Counters (Широкий діапазон частот з двома лічильниками).

2.4 Узгодження сигналів

Типова автоматизована вимірювальна система містить вимірюване фізичне явище, датчик (вимірювальний перетворювач), систему узгодження сигналів, пристрій збирання даних і комп'ютер. Джерелом сигналу найчастіше є датчик, що вимірює фізичне явище. Різні типи й моделі датчиків видають сигнали, рівні яких змінюються в широких межах. Сигнал із датчика найчастіше подається на пристрій узгодження. Підключення здійснюють за допомогою двох або більше сполучних провідників, довжина яких сягає від одиниць сантиметрів до кілометрів, і які в реальних системах часто схильні до наведення перешкод від високовольтних джерел та інших електромагнітних завад. У більшості прикладів якість сигналу визначається якістю підключення поблизу джерела.

Практично в усіх реальних датчиків і вимірювальних

перетворювачів вихідний сигнал потребує попереднього оброблення для того, щоб пристрій збирання даних (ПЗД) міг точно й надійно його виміряти та ввести в комп'ютер. Таке попереднє оброблення частіше називають узгодженням сигналу, воно може мати функції підсилення, фільтрації, електричну ізоляцію, мультиплексування та ін. Системи узгодження сигналів або вбудовують безпосередньо в пристрій збирання даних, або розміщують в окремих пристроях, таких як Signal Conditioning eXtensions for Instrumentation (SCXI) або Signal Conditioning Components (SCC). Завданням усіх таких пристроїв є приведення вхідного сигналу до рівня та виду, оптимального для вимірювання за допомогою аналого-цифрового перетворювача (АЦП) ПЗД.

Пристрій збирання даних найчастіше являє собою модуль (плату), убудований у комп'ютер, або ж самостійний блок, із забезпеченням комунікаційних функцій за поширеними протоколами передавання даних. ПЗД володіє всіма необхідними функціями для точного й надійного введення попередньо узгоджених сигналів у комп'ютер, який потім дозволяє аналізувати, зберігати й відображати результати.

2.4.1 Конфігурація системи узгодження сигналів

SCXI – завершена архітектура узгодження сигналів, що забезпечує багатофункціональне високопродуктивне оброблення сигналу різноманітних датчиків.

Датчики безпосередньо приєднуються до термінальних вузлів (terminal block). Інструментальне шасі (SCXI Chassis) служить для розміщення модулів SCXI, забезпечення їх електроживлення та керування через спеціальну службову шину SCXIbus. Шасі SCXI, зі свого боку, підключають до вбудованого в комп'ютер пристрою збирання даних, що

керує роботою всього шасі.

Рисунок 2.28 ілюструє архітектуру системи узгодження сигналів SCXI. Датчики приєднуються до термінальних вузлів, підключених до особових панелей модулів SCXI, розміщених в інструментальному шасі. Кожен модуль SCXI використовує мультиплексор для передавання узгодженого сигналу на шину SCXIbus. ПЗД задає алгоритм управління всього шасі.

Аналогові SCXI-модулі можуть працювати в паралельному або мультиплексному режимі. У паралельному режимі модулі не використовують мультиплексування сигналів, а просто передають узгоджені сигнали безпосередньо на відповідні вхідні канали ПЗД. Отже, кожен модуль з'єднаний прямим кабелем зі своїм індивідуальним ПЗД і кожне ПЗД в комп'ютері може приймати сигнали лише від одного SCXI модуля. Під час роботи в паралельному режимі продуктивність (швидкість) збирання даних визначають винятково можливостями ПЗД, а не SCXI-системи.

На практиці більшість SCXI систем працює в мультиплексному режимі. Мультиплексування (або сигнальна комутація) дозволяє спрямовувати сотні узгоджених сигналів на одне ПЗД. У мультиплексному режимі сигнальний кабель від одного ПЗД підключається до одного з модулів у шасі SCXI. Частина цифрових ліній введення / виведення ПЗД буде задіяна для управління модулями й шасі в цілому, а саме – три або чотири лінії цифрового виведення та лінія EXTSTROBE для контролю SCXI системи. Додатково, лінії цифрового введення будуть використані для зчитування інформації з модулів SCXI.

Під час роботи з SCXI модулями аналогового введення ви можете проводити вимірювання сигналів, виконуючи одноканальное зчитування або багатоканальний опитування з апаратним тактуванням.

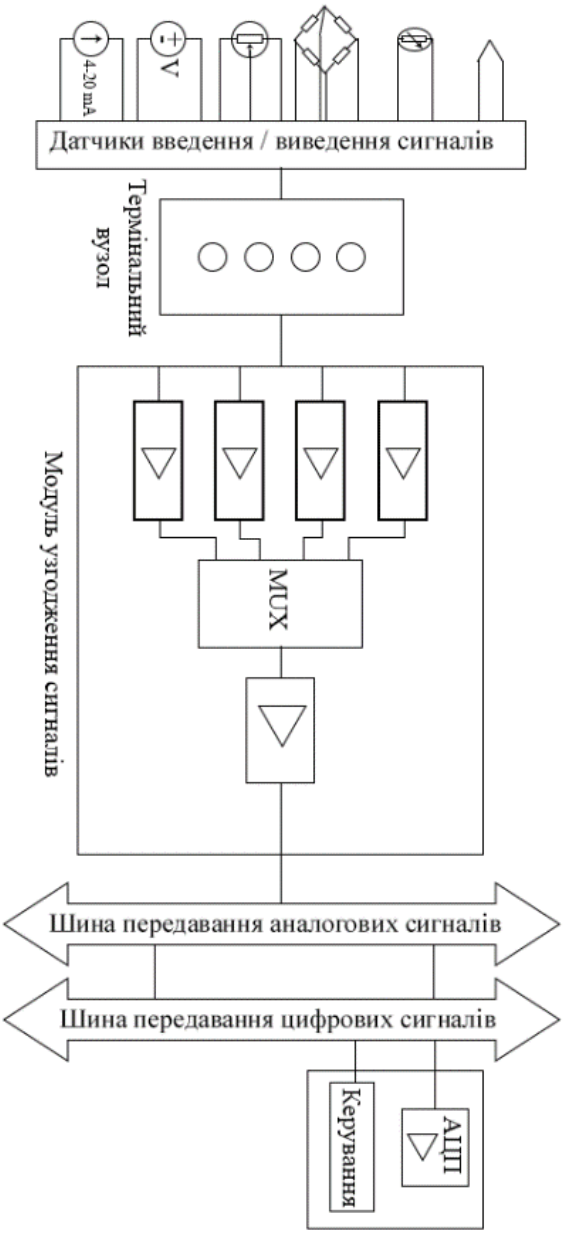


Рисунок 2.28 – Архітектура системи узгодження сигналів

Під час зчитування з одного каналу ПЗД періодично записує набір цифрових сигналів (digital pattern) у нульовий слот (SCXI Slot 0), розташований у шасі SCXI, указуючи, до якого SCXI модуля відбувається звернення. Потім ПЗД записує набір цифрових сигналів у модуль, указуючи, з якого вхідного каналу відбувається зчитування, і який налаштовує модуль на передачу корисного сигналу через аналогову шину (analog bus) шасі SCXI. У результаті, сигнал перенаправляється в канал аналогового введення ПЗД. Потім ПЗД виконує зчитування з каналу 0. Коли ви викликаєте будь-які функції одноканального аналогового введення, то всі вищевказані низькорівневі операції цифрового взаємодії та перенаправлення сигналу виконує інструментальний драйвер DAQ.

Під час багатоканального опитування ПЗД програмує нульовий слот SCXI, задаючи список модулів і кількість опитуваних каналів для кожного модуля. Для кожного з них також зазначають канал, із якого починається опитування. Після цього ПЗД або модуль починають багатоканальне опитування. Сигнал SCANCLK, що генерується ПЗД, синхронізує мультиплексування SCXI із внутрішнім тактовим генератором, що запускає аналого-цифрові перетворення в ПЗД. SCXI Slot 0 включає або відключає модулі відповідно до запрограмованого списком. Отже, система мультиплексує канали декількох модулів, підключаючи їх до каналу аналогового введення ПЗД з дуже великою частотою.

2.4.2 Основні функції узгодження

Крім маніпулювання зі специфічними вимірювальними перетворювачами, пристрої узгодження сигналів можуть виконувати широкий набір універсальних функцій узгодження для поліпшення якості, гнучкості й надійності

вимірювальної системи.

Підсилення сигналу. Оскільки реальні сигнали часто мають надзвичайно маленьку амплітуду, узгодження сигналу може збільшити точність одержаних даних. Підсилювачі збільшують рівень вхідного сигналу для кращої відповідності робочому діапазону АЦП, що дозволяє збільшити роздільну здатність і чутливість вимірювання. Хоча більшість ПЗД містять підсилювачі, багато вимірювальних перетворювачів, такі як термопари, вимагають додаткового підсилення.

Багато вимірювальних перетворювачів генерують напругу в діапазоні кількох мілівольт або навіть мікровольт. Підсилення цих слабких сигналів безпосередньо ПЗД призводить до посилення й шумів (перешкод), наведених на сигнальні дроти. Коли сигнал дуже слабкий, то навіть невеликого шуму досить для повного заглушення, що призведе до неправильних даних. Найпростіший спосіб збільшити відношення сигнал–шум полягає в посиленні сигналу якомога ближче до його джерела. Це підніме рівень сигналу вище рівня шуму до того, як наведений шум зможе безнадійно зіпсувати сигнал. Наприклад, на рисунку 2.29 показано сигнал термопари *J* типу, який змінюється з кроком 50 мкв / °С.

Припустимо, що дроти термопари, що з’єднують її з ПЗД, мають довжину 10 м і прокладені в середовищі зі складною шумовою обстановкою. Якщо джерело шуму в навколишньому середовищі наводить 200 мкв на дроти термопари, то ми одержимо під час вимірювання температури додаткові 4 °С через шум.

Підсилення сигналу в безпосередній близькості від термопари перед впливом шуму зменшить вплив останнього на кінцевий результат вимірювання. Якщо ми підсилимо сигнал у 500 разів поблизу термопари, це приведе до кроку сигналу приблизно 25 мВ / °С. Якщо

такий сигнал буде передаватися по 10-метрових дротах, то наведений шум амплітудою 200 мкВ значно менше вплине на результат вимірювання, додаючи лише 0,03 °С.

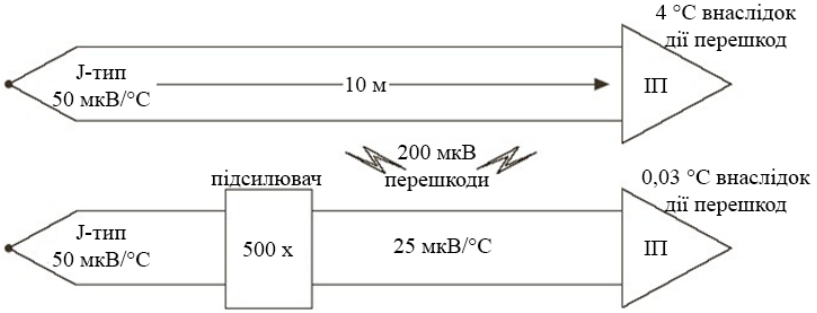


Рисунок 2.29 – Вплив перешкод на сигнал термопари

Фільтрація. Системи узгодження сигналів можуть включати фільтри для видалення небажаних перешкод у певному частотному діапазоні. Практично всі програми зі збирання даних схильні до впливу перешкод на частоті 50 Гц, наведених лініями живлення та обладнанням. Тому більшість систем узгодження сигналів містять фільтри нижніх частот (ФНЧ), спроєктовані спеціально для видалення перешкод на частоті 50 Гц із пропускною здатністю 4 Гц, так що пригнічення шуму на частоті 50 Гц здійснюється з найбільшою ефективністю (– 90 дБ).

Здебільшого фільтри поділяють на п’ять категорій: фільтри нижніх, верхніх частот, смуговий, загороджувальний і фазовий фільтри. Ця класифікація виникла по частотному діапазону (або смузі) сигналів, які фільтр пропускає без ослаблення.

Ідеальний ФНЧ не послаблює компоненти сигналу з частотами в смузі пропускання (СП), яку визначено як частотний діапазон нижчий від частоти зрізу. Зате повністю пригнічує все частотні компоненти в смузі режекції, що має частотний діапазон вищий від частоти зрізу. Фазовий зсув

ідеального ФНЧ має лінійну залежність від частоти. Лінійність фази означає, що всі частотні компоненти сигналу будуть мати однакову постійну часову затримку, що не призведе до спотворення форми сигналу. Реальні фільтри діють на сигнал подібно до математичного множення його на передавальну функцію, наближено збігаються з характеристиками ідеального фільтра. На рисунку 2.30 наведено порівняння послаблювальних властивостей передавальних функцій реального й ідеального фільтрів. Як можна бачити, для реального фільтра характерними є нерівномірність ослаблення залежно від частоти в смузі пропускання, перехід в область між смугами пропускання й режекції та смугу режекції з кінцевим ослабленням і хвилястістю.

Крім того, реальні фільтри мають деяку нелінійність фазово-частотної характеристики (ФЧХ), що означає велику затримку більш високочастотних компонент у порівняно з низькочастотними компонентами. У результаті з'являться спотворення форми сигналу. Це явище можна спостерігати, якщо пропустити прямокутний сигнал або сигнал у вигляді сходинки через ФНЧ. Ідеальний фільтр згладить фронти сигналу. Реальний фільтр викличе в ньому перехідні процеси через більші затримки високочастотних компонент.

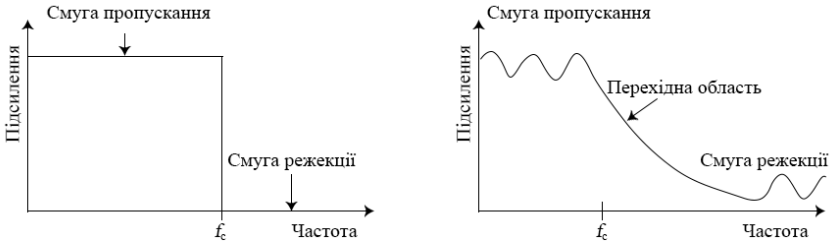
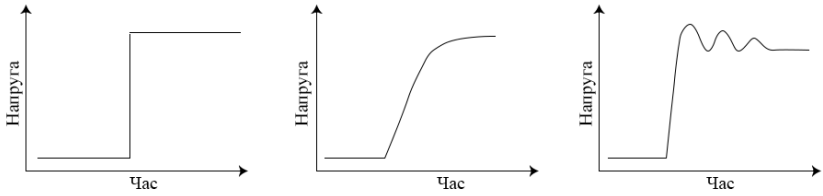


Рисунок 2.30 – Послаблювальні властивості передавальної функції ідеального (а) та реального (б) фільтрів

На рисунку 2.31 показано приклади відгуків ФНЧ на ступінчастий сигнал.



а б в

Рисунок 2.31 – Відгук на вхідний ступінчастий сигнал (а) ідеального (б) та реального (в) ФНЧ

Фільтри захисту від накладення спектрів. Іншим поширеним прикладом використання фільтрів є захист від накладення спектрів – явища, яке виникає за умови недостатньої частоти вибірки. Теорема Котельникова стверджує, що, коли ви оцифровує аналоговий сигнал, то компоненти сигналу з частотами, які перевищують половину частоти вибірки, з'являться в оцифрованих даних у вигляді низькочастотних компонент. Цього спотворення сигналу можна уникнути, якщо перед оцифруванням прибрати з нього за допомогою ФНЧ ті компоненти, частота яких перевищує половину частоти вибірки.

Раніше, на рисунку 2.8, було показано синусоїдальний сигнал, оцифрований у виділених точках. Коли ці точки використовують для відновлення осцилограми, сигнал виглядає так, як ніби має частоту меншу від вихідної. Можна збільшити частоту дискретизації або пропустити сигнал через ФНЧ для видалення високочастотних компонент. Збільшення частоти вибірки може привести до додаткових витрат і часто недоцільно, особливо коли верхня межа смуги високочастотного шуму сильно перевищує смугу корисного сигналу. Тому зазвичай на практиці використовують ФВЧ для видалення всіх

частотних компонент, що перевищують частоту Найквіста.

Запобігти накладенню спектрів можуть лише аналогові фільтри. Цифрові фільтри не підходять для цього, оскільки неможливо уникнути накладення спектрів, коли сигнал уже був оцифрований.

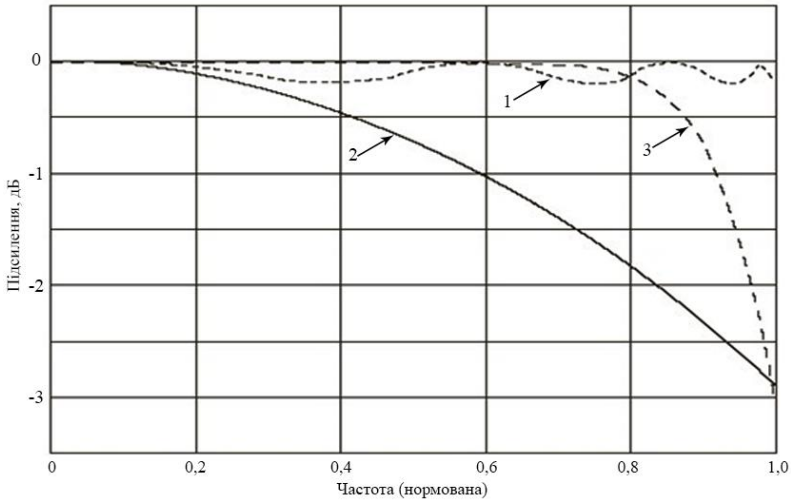
Зазвичай програмований аналоговий фільтр використовують як попередній формувач сигналів для оцифровувальних пристроїв, таких як ПЗД. Аналогові фільтри мають стандартні передавальні функції, що забезпечують компроміс характеристик реальних фільтрів, таких як крутизна спаду, хвилястість у СП і лінійність фази. Наступні фільтри мають стандартні передавальні функції: Баттерворта, Чебишева, Бесселя та еліптичний. Наприклад, фільтри Баттерворта мають дуже рівну амплітудно-частотну характеристику (АЧХ) в СП, зате фільтр Чебишева забезпечує дуже сильне ослаблення, але має недолік у надмірній хвилястості АЧХ в СП. Фільтр Бесселя має лінійну ФЧХ в усій СП, що мінімізує спотворення форми сигналу, але демонструє менш крутий спад характеристики і як наслідок слабше пригнічення сигналу в смузі режекції. Еліптичний фільтр Кауера з надзвичайно різким спадом у перехідній області вельми корисний як фільтр захисту від накладення спектрів для багатоканальних DAQ-систем. Однак сильна нелінійність ФЧХ робить його більш відповідним для додатків, що виконують аналіз частотного вмісту сигналу, а не його фазового складу або форми сигналу.

На рисунку 2.32 показано передавальні функції трьох модулів узгодження сигналів із використанням різних типів фільтрів. Еліптичний фільтр із дуже різкою перехідною областю особливо корисний як фільтр захисту від накладення спектрів у багатоканальних системах збирання даних.

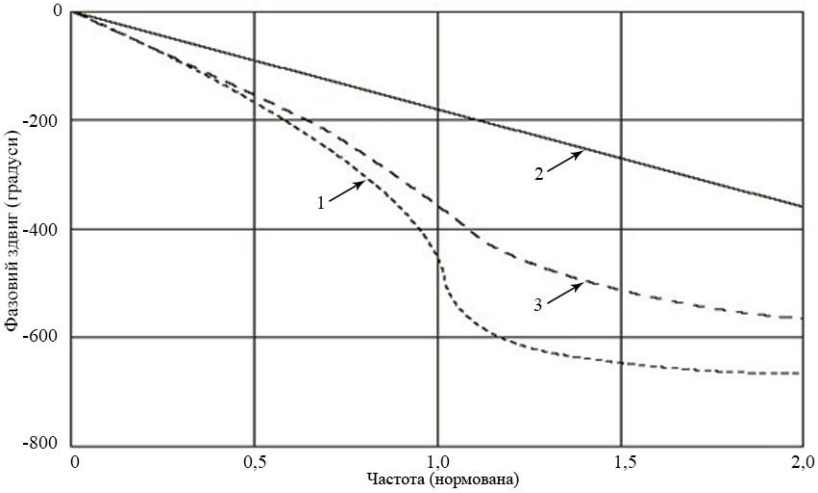
Хороше пригнічення фільтром сигналів із частотою, що

перевищує частоту зрізу, показує доцільність його використання як фільтра захисту від накладення спектрів. Через різкий спад АЧХ фільтра ви можете оцифровувати сигнал із більш низькою частотою вибірки. Наприклад, розглянемо 16-канальний додаток для сигналів зі смугою частот 10 кГц. Тут потрібні фільтри захисту від накладення спектрів, щоб уникнути спотворення сигналів небажаними перешкодами з частотами більше 10 кГц. Використовуючи, наприклад, модуль із набором еліптичних фільтрів восьмого порядку, можна запрограмувати частоту зрізу для 16 каналів, що дорівнює 10 кГц. У цьому разі ослаблення фільтрами на частоті 15 кГц досягне 80 дБ. Отже, частоту вибірки ПЗД можна безпечно встановити таку, що дорівнює подвоєній частоті 15 кГц, тобто 30 000 вибірок за секунду. Оскільки всі 16 каналів оцифровуються одним ПЗД, його необхідно запрограмувати на сумарну частоту вибірки, що дорівнює 480 000 вибірок за секунду.

Під час використання інших типів фільтрів із менш різким спадом АЧХ у перехідній області DAQ пристрій потребує оцифровувати сигнал із більшою частотою вибірки. Наприклад, типовий фільтр Баттерворта восьмого порядку, запрограмований на частоту зрізу 10 кГц, досягає придушення в 80 дБ на частоті 30 кГц. Частоту вибірки ПЗД, що використовує такий фільтр, необхідно встановити рівний 60 000 вибірок за секунду на канал, тобто 960 000 виб / с. у сумі. Більш висока частота вибірки вимагає більш дорогого ПЗД (або меншого числа каналів ПЗД) і більшої місткості пристроїв зберігання даних. Отже, більш різка перехідна область еліптичного фільтра знижує вимоги до АЦП, дозволяючи зменшити частоти вибірки та обсяг збережених даних. Однак еліптичний фільтр має сильну нелінійність ФЧХ, що показано на рисунку 2.32 б. Ця нелінійність ФЧХ фільтра робить його більш корисним у додатках, що передбачають аналіз частотного вмісту, але не



а



б

Рисунок 2.32 – Передавальні функції модулів узгодження сигналів з використанням фільтрів (а) та фазо-частотні характеристики (б) еліптичного фільтра (1), фільтра Бесселя (2) та фільтра Баттерворта (3)

фазових складових або форми сигналу. Під час вибору фільтра для збирання даних потрібно розглядати обидві його характеристики – АЧХ і ФЧХ, особливо в тих додатках, де не можна допустити втрати інформації в сигналі внаслідок його фільтрації.

Традиційні аналогові фільтри зазвичай побудовані на основі операційних підсилювачів, резисторів і конденсаторів. Однак для створення фільтрів захисту від накладення спектрів зазвичай використовують технологію перемикання конденсаторів. Більш того, конденсатор, що перемикається, замінив резистор у багатьох традиційних схемах аналогових фільтрів. Оскільки імпеданс такого конденсатора є функцією частоти сигналу перемикання, частоту зрізу такого фільтра можна змінювати, змінюючи частоту тактового сигналу, що керує перемиканням. Крім того, фільтри, що складаються з операційних підсилювачів і конденсаторів що перемикаються, більш легкі у виробництві.

Однак такі фільтри мають потенційні недоліки. Використання фільтрів лише такого типу призводить до зайвого пригнічення сигналу в модулях узгодження. Тому на практиці зазвичай використовують змішану схему, що складається з фільтрів на основі конденсаторів, що перемикаються, і фільтрів, що виконують операції безперервно, (фільтри безперервного часу), що дозволяє об’єднати технологічні та експлуатаційні переваги обох із них. Основні компоненти на етапі фільтрації процесу узгодження сигналів складаються з фільтра, що виконує операції в дискретні моменти часу (фільтр дискретного часу) на основі конденсаторів, що перемикаються, програмованого аналогового вхідного фільтра безперервного часу й програмованого аналогового вихідного фільтра також безперервного часу.

Розглянемо особливості кожного із цих фільтрів:

– *Фільтр дискретного часу на основі перемикальних конденсаторів* Частотою зрізу легко керувати, змінюючи частоту вхідного тактового сигналу.

– *Програмований вхідний аналоговий фільтр безперервного часу*. Фільтри на основі перекальних конденсаторів – це цифрові пристрої та схильні до впливу ефекту накладення спектрів. Тому аналоговий вхідний фільтр послаблює всі частотні компоненти, що лежать за межею частоти Найквіста сигналу перемикання.

– *Програмований вихідний аналоговий фільтр безперервного часу*. Вихідний сигнал фільтра на основі перемикальних конденсаторів – поетапне представлення аналогового сигналу, що реконструюється за допомогою вихідного аналогового фільтра. Цей фільтр також видаляє шум високочастотного генератора, що керує фільтром на основі перемикальних конденсаторів.

Ізоляція. Неправильне заземлення системи – один із найбільш частих випадків виникнення проблем із вимірюваннями, появою перешкод і пошкодження ПЗД. Системи узгодження сигналів з електричною ізоляцією можуть запобігти більшості цих проблем. Пристрої такого типу безконтактно передають сигнал від джерела до вимірювальної системи, використовуючи трансформатори, оптичні та ємнісні розв’язки. Крім розриву паразитного контура із замиканням через землю, ізоляція блокує великі викиди напруги та відфільтровує синфазну напругу, захищаючи операторів і дороговартісне вимірювальне обладнання.

Припустимо, що вам необхідно відстежувати температуру за допомогою термопари, припаяної до високовольтної установки, яка випромінює сильні електромагнітні поля. Хоча вихідна диференціальна напруга термопари не перевищує 50 мВ, вона може мати високий потенціал щодо заземлення через ємнісну зв’язку

установки й термопари. Потенціал між кожним із провідників диференціального сигналу й заземленням називають синфазною напругою. В ідеалі ця напруга повинна повністю ігноруватися вимірювальною системою. Однак приєднання проводів від термопари безпосередньо до неізольованих ПЗД, який зазвичай розрахований на синфазну напругу 12 вольт здебільшого призведе до пошкодження останнього. Замість цього ви можете приєднати дроти від термопари до ізольованого формувача сигналів.

Технічні вимоги до ізоляції системи. Виробники пристроїв висувають різні вимоги до електричної ізоляції. Деякі вводять лише індекс (number) ізоляції без опису, чи позначають рівень сигналу або рівень несталої напруги. Без цієї інформації та ясного розуміння процесу введення / виведення сигналів ви можете пошкодити вимірювальну систему і, можливо, піддати оператора небезпеці ураження струмом. Ряд організацій, таких як Лабораторія з техніки безпеки (Underwriters Laboratories) і Міжнародна електротехнічна комісія – IEC (International Electrotechnical Commission) розробили технічні вимоги безпеки під час проєктування високовольтних установок. Продукція, що має відповідні знаки, перевірена (в деяких ситуаціях, самими організаціями) на відповідність технічним вимогам.

Крім пошуку на продукції знаків відповідності одній із цих організацій, існує інший спосіб визначити клас ізоляції системи узгодження сигналів. Він полягає в з'ясуванні двох ключових характеристик – максимального допустимого напруги й категорії установки.

Максимальна допустима напруга – це технічна вимога, що описує максимальну постійну напругу, яку можна подавати на входи системи за нормальних умов експлуатації. Цю вимогу висувають для сигналу щодо

опорного потенціалу заземлення, тобто під нього потрапляє й рівень сигналу, і будь-яка синфазна напруга, пов'язана із сигналом.

Категорія установки – описує місце розташування, де ви можете використовувати певний вимірювальний пристрій, беручи до уваги на величину можливих перехідних сигналів (включно з перешкодами) в цьому місці. У більш загальному сенсі ця вимога описує можливі перехідні процеси, які зможе витримати пристрій.

Існують чотири категорії установки, як показано на наступному рисунку 2.33. Залежно від місцезнаходження електричної розподільчої системи існує певна кількість коливань до встановлення напруги в системі. Чим ближче система до електростанції, тим коливання напруги трапляються рідше. І навпаки, чим більше розподілена система передачі електроенергії, тим коливання напруги частішають. Коливання зменшують перенапруги, присутні в системі. Чим ближче ваша система до джерела напруги, тим більше амплітуда перехідних процесів. Залежно від місця розташування вимірювальної системи необхідно застосувати певні запобіжні заходи для захисту системи від потенційно небезпечних перенапруг в електричній розподільній системі. Загальноприйнятими є такі чотири категорії для окремих рівнів електроспоживання з різними величинами перенапруг:

– категорія установки IV – рівень розподілення. Складається з такого обладнання, як генератори, підстанції та трансформатори;

– категорія установки III – стаціонарні пристрої. Складається з обладнання, постійно приєднаного до розподільної мережі, як, наприклад, кондиціонери повітря й каміни;

– категорія установки II – обладнання, яке споживає електричну енергію від стаціонарних систем. Наприклад,

дрилі, телевізори, радіоприймачі та комп'ютери;

– категорія установки I – обладнання для приєднання до електричних мереж, у яких перенапруги під час перехідних процесів конструктивно обмежені досить невеликим рівнем. Устаткування I категорії складається, наприклад, із низьковольтних джерел струму.

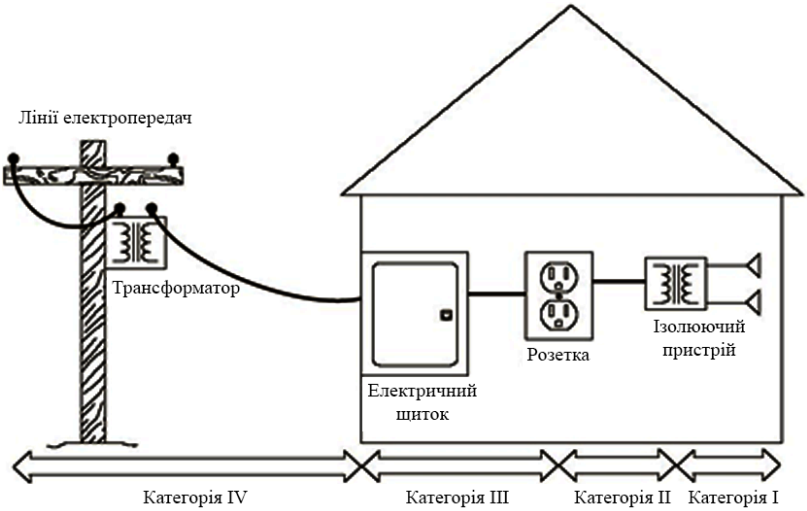


Рисунок 2.33 – Категорії установки вимірювального обладнання

Електрична ізоляція забезпечує безпечний бар'єр між користувачем і його обладнанням з одного боку та високими напругами й перенапругами під час перехідних процесів з іншого. Пристрій, що убезпечує електричну ізоляцію – це дуже корисний елемент у вашій системі.

Усі ізолювальні вимірювальні модулі мають подвійну ізоляцію для безперервної роботи з напругою 250 В. Крім того, вони протестовані й витримують напругу на вході 2 300 В упродовж однієї хвилини, що дозволяє не боятися перенапруг під час перехідних процесів. Ізолювальні

модулі повинні бути виконані відповідно до вимог IEC-1010 (вимоги безпеки електротехнічного обладнання для систем вимірювань, управління й лабораторних установок) для другої категорії установки.

2.4.3 Узгодження сигналів з датчиків

Вимірювальні перетворювачі (датчики) – це пристрої, що перетворюють фізичні явища, такі як температура, деформація, тиск або світло, на електричні величини, такі як напруга або опір. Характеристики перетворювачів визначають більшість вимог системи збирання даних до узгодження сигналу.

Термопари. Одним із найбільш часто використовуваних перетворювачів температури є термопара (thermocouple). Термопари дуже невибагливі, дешеві й можуть працювати в широкому діапазоні температур. Термопара утворюється, коли з'єднують разом два різнорідних метали, водночас точка контакту генерує невелика напруга, що є функцією температури. Ця термоелектрична напруга відома як напруга Зеєбека, названа на честь Томаса Зеєбека (Thomas Seebeck), який відкрив це явище в 1821 році. Напруга нелінійно залежить від температури. Однак, у разі малих змін температури напруга майже лінійна

$$\Delta \approx \Delta VST,$$

де ΔV – зміна напруги; S – коефіцієнт Зеєбека; а ΔT – зміна температури.

Коефіцієнт S змінюється з температурою, що обумовлює нелінійність вихідного напруги термопар у межах їх робочих діапазонів. Типи термопар позначаються великими англійськими літерами, що характеризують їх

склад відповідно до угод Національного Інституту Стандартизації США (ANSI). Наприклад, термопара J-типу складається із залізного й константанових (сплав міді та нікелю) провідників. Напругу термопар можна відстежувати за допомогою багатофункціональних систем збирання даних. Термопари мають ряд спеціальних вимог під час узгодження їх сигналів.

Схеми включення термопар. У разі вимірювання напруги термопари ви не зможете просто приєднати її до вольтметру або іншої вимірювальної системи, оскільки дроти, що йдуть від термопари до вимірювальної системи створюють додаткові термоелектричні ланцюги.

Розглянемо схему, показану на рисунку 2.34, у якій термопара J-типу поміщена в полум'я, температуру якого ви хочете виміряти. Два дроти термопари приєднані до міднх провідників ПЗД. Зверніть увагу, що схема містить три контакти різних металів – J1, J2 і J3. J1 – контакт термопари, що генерує напругу, пропорційну температурі полум'я. Контакти J2 і J3 мають свій власний коефіцієнт Зеєбека й генерують додатковий термоелектричний потенціал, пропорційний температурі терміналів ПЗД. Для визначення внеску в напругу контакту J1 необхідно знати температури контактів J2 і J3 та співвідношення напруга – температура для них. Потім можна відняти внески паразитних термопар у контактах J2 і J3 з виміряної напруги.

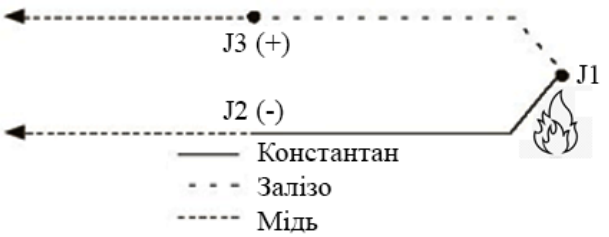


Рисунок 2.34 – Підключення термопари J-типу до ПЗД

Компенсація холодного спаю. Термопари вимагають певної опорної температури для компенсації паразитних термопар. Термін «холодний спай» (cold junction) прийшов із поширеної практики витримування опорного контакту за температури 0 °С у воді з льодом. Довідкові таблиці Національного інституту стандартів і технологій (NIST) були створені за допомогою установки, показаної на рисунку 2.35. Виміряна напруга залежить від різниці температур T_1 і T_{ref}. У цьому разі випадку T_{ref} дорівнює 0 °С. Зверніть увагу, що оскільки контакти підвідних дротів вольтметра знаходяться за тієї самої температури, тобто ізотермічності, напруги, що генеруються в цих двох точках рівні за величиною та протилежні за знаком. Тому результуюча напруга, унесена цими двома контактами, дорівнює нулю.

У цих умовах під час вимірювання температури більше 0 °С напруга термопари буде додатною. Якщо вимірювана температура менше 0 °С, то вихідна напруга термопари буде від'ємною. Коли опорний спай і вимірювальний контакт мають однакову температуру, результуюча напруга дорівнює нулю.

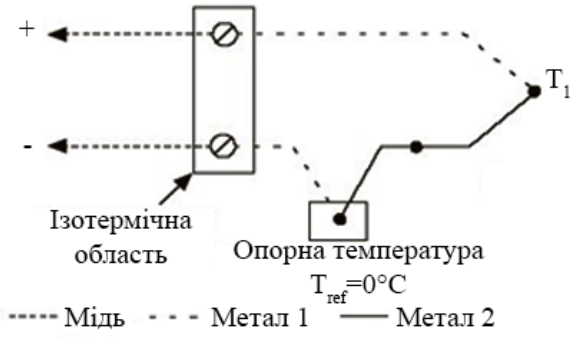


Рисунок 2.35 – Вимірювання температури з опорним контактом, що витримується за температури 0 °С

Хоча застосування ємності з льодом призводить до точного результату вимірювання, її не завжди зручно використовувати. Більш практичний прийом полягає у вимірюванні температури холодного спаю і безпосередньому зчитуванні напруги з термопари, а потім у відніманні вкладів термоелектричних напруг паразитних термопар. Цей процес називають компенсацією холодного спаю (КХС). Можна легко обчислити КХС, використовуючи деякі властивості термопар.

Використовуючи рівняння для термопари з урахуванням проміжних металевих провідників і роблячи деякі прості припущення, можна бачити, що напруга, яку вимірюють ПЗД в схемі, показаній на рисунку 2.34, залежить лише від типу термопари, напруги термопари й температури холодного спаю. Виміряна напруга фактично не залежить від матеріалу проводів і холодних спаїв J2 і J3.

Відповідно до рівняння термопари з урахуванням проміжних металевих провідників, показаних на рисунку 2.36, використання будь-якого типу дротів у схемі з термопарою не чинитиме жодного впливу на вихідну напругу, якщо обидва кінці цього проводу матимуть однакову температуру.

Рисунок 2.36 – Підключення термопари з урахуванням проміжних металевих провідників

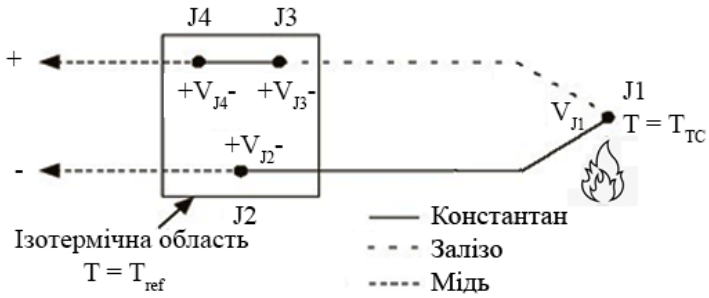


Рисунок 2.37 – Вставка додаткового провідника в ізотермічну область

Розглянемо схему, показану на рисунку 2.37. Вона схожа на схему рисунку 2.34 за винятком того, що невеликий відрізок константанового дроту був уставлений безпосередньо перед контактом J3, і передбачається, що його контакти мають однакову температуру. Припускаючи, що контакти J3 і J4 мають також однакову температуру, то, використовуючи рівняння термопари з урахуванням проміжних металевих провідників, можна зробити висновок, що схема на рисунку 2.37 – електрично еквівалентна схемі на рисунку 2.34. Отже, будь-який результат вимірювання, одержаний у схемі рисунку 2.37, застосуємо також до схеми на рисунку 2.34.

На рисунку 2.37 контакти J2 і J4 одного типу (мідь-константан). Оскільки обидва вони знаходяться в ізотермічній області, то J2 і J4 мають одну температуру. Однак ці контакти включені в протилежних напрямках, а отже, їх сумарний вклад у виміряну напругу дорівнює нулю. Контакти J1 і J3 теж одного типу (залізо-константан) і також включені в протилежних напрямках, але можуть мати різні температури. Тому на повну виміряну напругу можуть впливати лише напруги контактів J1 і J3. Використовуючи позначення VJx(Ty) для напруги, згенерованого контактом Jx за температури Ty, все

завдання дослідження термопари зводиться до такого рівняння:

$$VMEAS = VJ1(TTC) + VJ3(Tref),$$

де V*MEAS* – напруга, що вимірюється ПЗД, T*TC* – температура термопари J1, а T*ref* – температура опорного контакту.

Зверніть увагу, що в цьому рівнянні VJx(Ty) – це напруга, згенерована за температури Ty стосовно деякої опорної температури. Поки напруги V*J*1 і V*J*3 – функції температури щодо однакової опорної температури, дане рівняння буде правильним. Як говорилося раніше, наприклад, довідкові таблиці термопар NIST одержані під час використання опорного контакту з температурою 0 °C.

Оскільки контакт J3 того самого типу, що і J1, але включений у протилежному напрямку, то VJ3(Tref) = -VJ1(Tref). Оскільки VJ1 – це напруга, що генерується досліджуваною термопарою, то її можна перейменувати у VTC. Отже, рівняння може бути записано в такому вигляді:

$$VMEAS=VTC(TTC)+VTC(Tref).$$

Тому, вимірюючи V*MEAS* і T*ref* і знаючи співвідношення напруга – температура для термопари, можна визначити температуру термопари.

Методи, що використовують компенсацію холодного спаю, вимагають, щоб температуру опорного контакту можна було виміряти безпосередньо будь-яким датчиком. Датчик із безпосереднім зчитуванням генерує вихідну напругу, залежну лише від температури в точці вимірювання. Для цього зазвичай використовують

напівпровідникові датчики, термістори або резистивні температурні детектори. Наприклад, ряд термінальних ПЗД містить термістори, розташовані поблизу клем із гвинтовим кріпленням, до яких приєднують провідники від термопар.

Лінеаризація даних. Вихідна напруга термопар сильно нелінійна. Коефіцієнт Зеєбека для деяких термопар може змінюватися в три та більше разів під час зміни температури в усьому робочому діапазоні. З цієї причини ви повинні або апроксимувати криву залежності напруги від температури поліномами, або використовувати довідкову таблицю 2.5. Апроксимація поліномами виглядає так:

$$T = +a_0 a v_1 + a v_{2+2} + + \ldots a v_n \qquad n$$

де v – напруга термопари у вольтах; T – температура в градусах Цельсія; a_0 – a_n – коефіцієнти, індивідуальні для кожного типу термопар.

Деформація. Деформація (ε) описує зміну розмірів або форми тіла під дією прикладеної сили.

Більш точно деформацію визначають як відношення зміни довжини $\varepsilon = \frac{\Delta L}{L}$, як показано на рисунку 2.38.

Деформація може бути позитивною (розтягнення) й негативною (стиснення). Незважаючи на безрозмірність цієї величини, іноді деформацію висловлюють у таких одиницях, як мм / мм.

У практичних додатках виявляється, що амплітуда вимірюваної деформації надзвичайно мала. Тому деформацію часто висловлюють через мікродеформацію (με), що дорівнює $\varepsilon \cdot 10^{-6}$.

Коли брусок деформує одновісна сила, як на рисунку 2.38, то виникає явище, відоме як деформація Пуассона, що викликає зміну периметра D поперечного перерізу бруска.

Таблиця2.5. – Граничні значення вихідних напруг термопар (мВ)

Тип термопари	Провідник		Діапазон температур (°С)	Діапазон напруг (мВ)	Коефіцієнт Зеєбека (МкВ/°С)
	+	–			
E	хромель	конст-антан	Від –270 до 1 000	Від –9,835 до 76,358	58,70 при 0 °С
J	залізо	конст-антан	Від –210 до 1 200	Від –8,096 до 69,536	50,37 при 0 °С
K	хромель	алю-мель	Від –270 до 1 372	Від –6,548 до 54,874	39,48 при 0 °С
T	мідь	конст-антан	Від –270 до 400	Від –6,258 до 20,869	38,74 при 0 °С
S	платина – 10 % родій	платина	Від –50 до 1 768	Від –0,236 до 18,698	10,19 при 600 °С
R	платина – 13 % родій	платина	Від –50 до 1 768	Від –0,226 до 21,108	10,35 при 600 °С

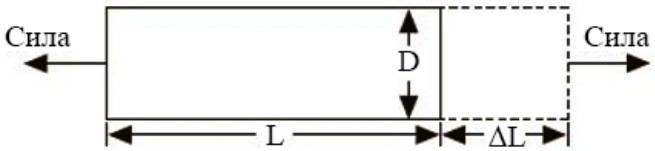


Рисунок 2.38 – Поздовжня одновісна деформація

Величина зміни визначається властивістю самого матеріалу й характеризується відношенням (коефіцієнтом) Пуассона. Коефіцієнт Пуассона ν матеріалу визначають як негативне відношення деформації в поперечному напрямку (перпендикулярно до сили) до деформації в поздовжньому напрямку (паралельно до сили), або $\nu = -\varepsilon_т / \varepsilon$. Коефіцієнт Пуассона для сталі, наприклад, може змінюватися від 0,25 до 0,3.

Тензодатчик. Хоча існує кілька методів вимірювання деформації, найбільш поширеним є використання тензодатчика, пристрою, чий електричний опір змінюється пропорційно величині його деформації. Наприклад, п'єзорезистивний тензодатчик – це напівпровідниковий пристрій, опір якого нелінійно залежить від деформації. Найчастіше використовуваним тензодатчиком є зіставний металізований тензодатчик.

Металізований тензодатчик складається з дуже тонкого дроту або здебільшого металевої фольги, розташованої у вигляді сітки. Сітковий шаблон доводить до максимуму кількість металевих дротів або фольги, що піддаються деформації в паралельному напрямку, як показано на рисунку 2.39. Площу поперечного перерізу сітки роблять мінімальною для зменшення впливу поперечної деформації та деформації зсуву. Сітка прикріплюється до тонкої підкладки, що називається тримачем і яка приєднана безпосередньо до досліджуваного зразка. Тому деформація, яку відчуває зразок, передається безпосередньо тензодатчиками, що відповідає на неї зміною електричного опору. Тензодатчики виробляють серійно, а величина їх номінального опору може змінюватися від 30 до 3 000 Ом. Дуже важливо правильно прикріпити тензодатчик до досліджуваного зразка, так щоб деформація правильно передавалася від зразка через сполучну речовину та підкладку тензодатчика до найчутливішого елемента. Щоб

правильно змонтувати датчик, варто користуватись інформацією, наданою його виробником.

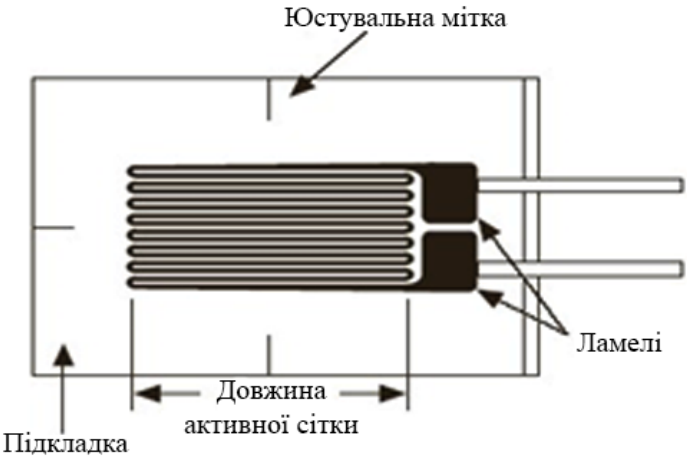


Рисунок 2.39 – Дротовий тензодатчик

Головним параметром тензодатчика є його чутливість до деформації, кількісно вираженої коефіцієнтом тензочутливості (gauge factor). Коефіцієнт тензочутливості (КТ) визначають як відношення відносної зміни опору й відносної зміни довжини (деформації):

$$\text{КТ} = \frac{\frac{\Delta R}{R}}{\frac{\Delta L}{L}} = \frac{\frac{\Delta R}{R}}{\varepsilon}$$

Коефіцієнт тензочутливості серійних металевих тензодатчиків зазвичай дорівнює 2. В ідеалі опір тензодатчика повинен змінюватися лише у відповідь на деформацію. Однак матеріали й тензодатчика, і зразка, до якого він прикріплений, реагують також і на температуру. Виробники тензодатчиків докладають усіх зусиль, щоб

мінімізувати чутливість до температури, спеціально обробляючи матеріал датчика для компенсації температурного розширення матеріалу зразка, для якого передбачено використовувати датчик. Але хоча компенсовані датчики зменшують температурну чутливість, зовсім прибрати її не можливо. Наприклад, розглянемо датчик, скомпенсований для алюмінію, що має температурний коефіцієнт 23 ppm /°C. За умови номінального опору 1 000 Ом еквівалентна помилка вимірювання деформації все ж дорівнює 11,5με/°C. Отже, дуже важлива додаткова температурна компенсація.

Вимірювання з використанням тензодатчиків. Вимірювання деформацій рідко призводять до величин, що перевищує кілька тисячних. Тому цей процес вимагає акуратного вимірювання дуже невеликих змін опору. Наприклад, припустимо, що випробуваний зразок піддається деформації в 500με. Відносна зміна електричного опору тензодатчика з коефіцієнтом тензочутливості 2, дорівнюватиме всього лише $2 \cdot (500 \times 10^{-6}) = 0{,}1$ %. Для датчика з номінальним опором 120 Ом ця зміна становитиме лише 0,12 Ом.

Для вимірювання таких малих змін опору й для компенсації температурної чутливості, спільно з тензодатчиками практично завжди використовують мостову схему з джерелом збудження струмом або напругою. Поширений міст Уітстона (Wheatstone bridge), показаний на рисунку 2.40, складається з чотирьох резистивних плечей, до яких прикладена напруга збудження V_{EX}.

Вихідну напругу моста V_0 визначають за формулою

$$V_0 = \left[\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2}\right] V_{EX}$$

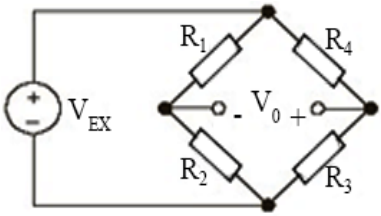


Рисунок 2.40 – Класична схема мосту Уітстона

Із цього рівняння випливає, що при $R_1/ R_2 = R_3/ R_4$, вихідна напруга V_O дорівнює нулю. У цьому разі про міст кажуть, що він збалансований. Будь-яка зміна опору одного з плечей мосту приведе до зміни вихідної напруги, тому, якщо ви замініте опір R_4 на активний тензодатчик, як показано на рисунку 2.41, то будь-яка зміна опору тензодатчика призведе до розбалансування мосту й генерації ненульовогї вихідної напруги. Якщо номінальний опір тензодатчика позначити за R_G, то зміну опору ΔR через деформацію можна записати у вигляді $\Delta R = R_G \times КТ \times \varepsilon$. Припускаючи, що $R_1 = R_2$ і $R_3 = R_G$, рівняння мосту можна переписати так, щоб виразити відношення V_O/V_{EX} у вигляді функції від деформації. Зверніть увагу на присутність множника $1 / (1 + КТ \times \varepsilon/2)$, який свідчить про нелінійну залежність вихідної напруги чверть мостової схеми від деформації.

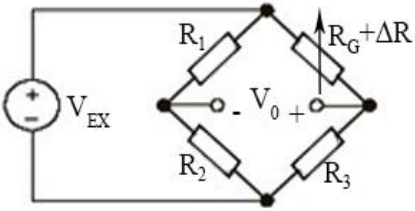


Рисунок 2.41 – Конфігурація четверть мостової схеми мосту Уітстона з тензодатчиком

$$\frac{V_0}{V_{EX}} = -\frac{\text{КТ} \times \varepsilon}{4}\left(\frac{1}{1 + \text{КТ}\frac{\varepsilon}{2}}\right)$$

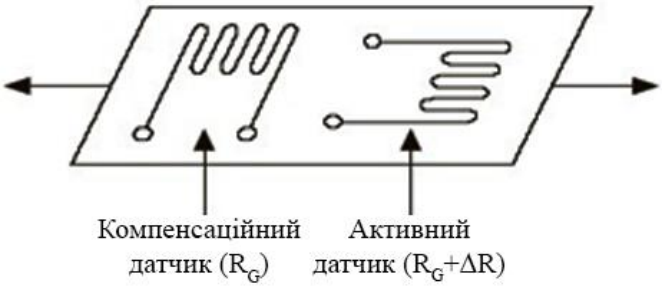


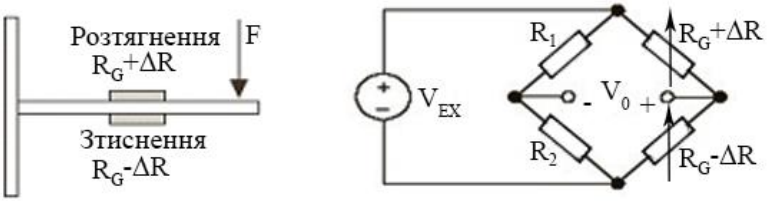
Рисунок 2.42 – Конфігурація тензодатчика з компенсацією температури

Використовуючи два тензодатчики в мосту, можна виключити вплив температури. Наприклад, на наступному рисунку 2.42 показано конфігурацію тензодатчиків, коли один із них активний (R_G+ ΔR), а другий розташований поперечно до прикладеної деформації. Тому остання має невеликий вплив на другий тензодатчик, що називають у цьому разі компенсаційним тензодатчиком (dummy gauge), однак будь-яка зміна температури буде впливати на обидва датчики однаково. Унаслідок цього відношення їх опорів не зміниться, тоді й напруга V_O не зміниться, тобто вплив температури буде мінімальним.

Альтернативно можна подвоїти чутливість моста до деформації, роблячи обидва датчики активними, але включеними в протилежних напрямках. Наприклад, на рисунку 2.43 а показано балку, що згинається, де один датчик установлений на розтягнення (R_G + ΔR), а другий – на стиснення (R_G – ΔR). Така півмостова конфігурація, принципову схему якої показано на рисунку 2.43 б, приводить до того, що вихідна напруга лінійна та

приблизно дорівнює подвоєній напрузі четвертьмостової схеми.

Ми можемо ще сильніше збільшити чутливість схеми, замінюючи опори всіх плечей моста активними тензодатчиками, і встановлюючи два з них на розтягнення та два на стиснення. На рисунку 2.44 показано повномостову схему.



а б

Рисунок 2.43 – Конфігурація півмостової схеми мосту Уітстона з двома тензодатчиками

$$\frac{V_0}{V_{EX}} = -\frac{\text{КТ} \times \varepsilon}{2}$$

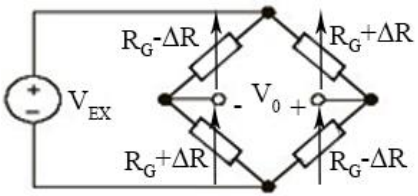


Рисунок 2.44 – Конфігурація повномостової схеми мосту Уітстона з чотирма тензодатчиками

Рівняння, наведені тут для схем мосту Уітстона, припускають, що спочатку міст збалансований, тобто без деформації вихідна напруга дорівнює нулю. На практиці, однак, допустимі відхилення величин опорів і деформація під час установлення датчика призводять до того, що існує

деяке початкове значення напруги зсуву. Зазвичай із початковим напруженням зсуву вчиняють двояко. По-перше, ви можете використовувати спеціальну схему занулення початкового зсуву (балансування) для підстроювання опорів у мосту, щоб його вихідна напруга дорівнювала нулю. Або ж можна виміряти початкову напругу без деформації та компенсувати його програмно. Рівняння деформації для чверть-, пів- і повномостових схем, що враховують початкові вихідні напруги, можна знайти в розділі Тензометричні рівняння цього підрозділу.

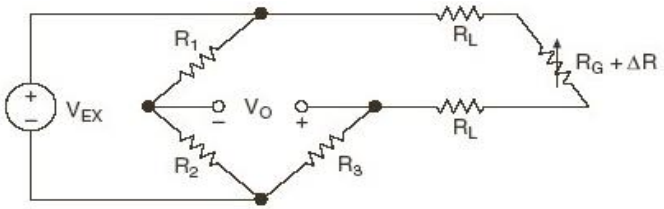
Опір підвідних дротів. Рисунки й рівняння, наведені вище, ігнорували опір у підвідних дротах (lead wires) тензодатчика. Це може бути корисним на етапі вивчення основ вимірювань із використанням тензодатчиків, однак на практиці це може бути дуже небезпечним. Наприклад, розглянемо двопровідне приєднання тензодатчика (рис. 2.45 а). Припустимо, що кожен провідник, що з'єднує тензодатчик, має довжину 15 метрів і має опір 1 Ом. Тоді обидва провідники додають 2 Оми до опору кожного плеча мосту. Крім додаткової помилки зсуву, опір провідників призводить до зменшення вихідної напруги. Це зменшення кількісно характеризується множником $(1 + R_L/ R_G)$. Можна компенсувати цю помилку, вимірюючи опір підвідних дротів R_{Li} використовуючи це значення в тензометричних рівняннях. Однак є більш серйозна проблема, що полягає в зміні їх опорів під дією температури. Знаючи типові температурні коефіцієнти мідного дроту, можна підрахувати, що невелика зміна температури може призвести до помилки вимірювання порядку декількох με. Тому кращою схемою включення тензодатчиків у четвертьмостову схему є схема з використанням трьох дротів, показана на рисунку 2.45 б. У цій конфігурації R_{L1} і R_{L3} розташовані в сусідніх плечах мосту. Будь-які зміни опору через температуру компенсують один одного. Третій

провідник з опором R_{L2} приєднаний до входу вимірювальної системи. Струм, що тече по провіднику, дуже малий, тому впливом цього опору можна знехтувати.

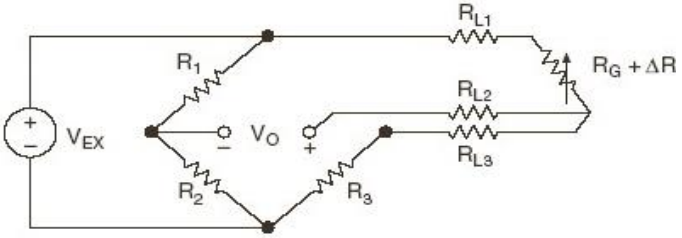
Узгодження сигналів із тензодатчиків. Вимірювання з використання тензодатчиків передбачає виявлення надзвичайно малі зміни опору. Отже, для достовірних вимірювань необхідні правильний вибір і використання мосту, узгодження сигналів, схеми з'єднань і компонентів для збирання даних.

Вибір мосту. Якщо ви не використовуєте повномостову схему з чотирма активними тензодатчиками, то в частині плечей мосту необхідно використовувати опорні резистори, тому зазвичай пристрої узгодження сигналів із тензодатчиків являють собою напівмостову схему, що складається з двох високоточних опорних резисторів. Саме значення номінального опору додаткових резисторів менш важливо, ніж те, наскільки вони збігаються за величиною. В ідеалі резистори однакові та створюють стабільну опорну напругу у $V_{EX} / 2$ на негативному вході каналу вимірювання.

Збудження мосту. Пристрої узгодження сигналів тензодатчиків зазвичай забезпечують джерело постійної напруги для живлення мосту. Оскільки досі не існує стандарту для рівня напруги збудження, вона може змінюватися від 3 В до 10 В. Хоча більша напруга збудження приводить до пропорційного збільшення вихідної напруги, вона може створити велику помилку через явище саморазігрівання. Важливо, щоб напруга збудження була точною та стабільною, однак можна використовувати й менш точну або стабільну напругу й проводити (тим не менше) точні вимірювання, використовуючи управління / стеження напругою збудження.



а



б

Рисунок 2.45 – Приєднання тензодатчиків із використанням підвідних дротів: а – двопровідна схема, б – трипровідна схема

Управління напругою збудження. Якщо схема з тензодатчиком розташована на деякій відстані від пристрою узгодження сигналів і джерела збудження, то виникає можливе джерело помилок – падіння напруги, обумовлене опором дротів, що підводять напругу збудження до мосту. деякі пристрої узгодження сигналів забезпечують додаткову можливість – дистанційне керування / стеження (remote sensing) для компенсації цієї помилки.

Існує два основних способи дистанційного керування / стеження. Перший із них – дистанційне керування зі зворотним зв'язком, коли ви приєднуєте додаткові дроти управління до точки, у якій дроти напруги збудження з'єднані зі схемою мосту. Додаткові дроти служать для регулювання джерела напруги збудження з метою компенсації втрат у провідниках і забезпечення мосту необхідною напругою.

В іншому методі використовують окремий вимірювальний канал для безпосереднього вимірювання напруги збудження, прикладеного паралельно мосту. Оскільки дроти вимірювального каналу проводять дуже слабкий струм, то їх опір майже не матиме впливу на вимірювання й ним можна знехтувати. Потім значення виміряної напруги збудження буде використовуватися в перетвореній напруги на деформацію для компенсації втрат у провідниках.

Підсилення сигналу. Вихідна напруга тензодатчиків і мостів порівняно мала. На практиці, більшість тензомостов і перетворювачів деформацій видають напругу менше 10 мкВ / В (10 мкв вихідної напруги на 1 вольт напруги збудження). Тому пристрої узгодження сигналу зазвичай містять підсилювачі для збільшення рівня сигналу, що приведе до більшої роздільної здатності вимірювання та кращого співвідношення сигнал – шум.

Балансування мосту, занулення напруги зсуву. Після установки мосту малоймовірно, що його вихідна напруга буде точно дорівнювати нулю за відсутності деформації. Здебільшого невелике розходження опорів у плечах мосту та опір підвідних проводів викликають генерацію деякої початкової напруги зсуву. Існує кілька способів, що дозволяють системі справлятися із цим явищем.

Програмна компенсація. У цьому разі перед деформацією ви повинні зробити вимірювання початкової напруги, яка потім буде використана в тензометричних рівняннях. Цей метод простий, швидкий у використанні й не вимагає ручних налаштувань. Єдиний його недолік – це неможливість видалення напруги зсуву мосту. Якщо ця напруга досить велика, вона буде обмежувати КП підсилювача у вихідному ланцюгу моста, обмежуючи тим самим динамічний діапазон вимірювання.

Схема занулення напруги зсуву. У другому методі

балансування використовують змінний резистор для фізичного підлаштування вихідної напруги до нуля (рис. 2.46). Обертаючи ручку потенціометра (R_{POT}), ви можете керувати рівнем вихідної напруги мосту й установити початкову напругу в нуль. Значення R_{NULL} установлює діапазон, який схема може збалансувати.

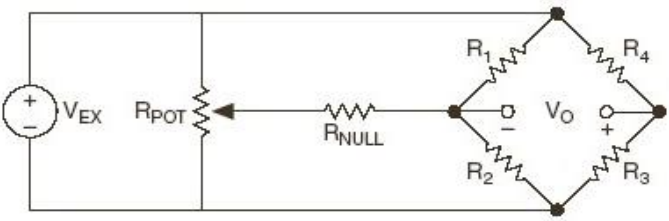


Рисунок 2.46 – Схема занулення напруги зсуву

Буферизоване занулення напруги зсуву. Третій метод, як і перший, не має безпосереднього впливу на міст. У ньому схема занулення додає перебудовуване постійнезначення напруги до вихідної напруги інструментального підсилювача.

Калібрування з використанням шунта. Такий метод – звичайна процедура перевірки вихідної напруги вимірювальної системи на основі тензодатчиків щодо деякої наперед заданої деформації. У разі калібрування симулюється деформація методом зміни опору одного з плечей мосту на деяку відому величину. Це відбувається за рахунок шунтування високоомним резистором із заданим значенням одного з плечей моста, що створює відоме ΔR. Ви можете виміряти вихідну напругу мосту й порівняти її з очікуваним значенням, а результат використовувати для корекції систематичної помилки вимірювання або просто, щоб переконатися, що установку зібрано правильно.

Тензометричні рівняння. Для спрощення запису рівнянь і врахування розбалансованих мостів в

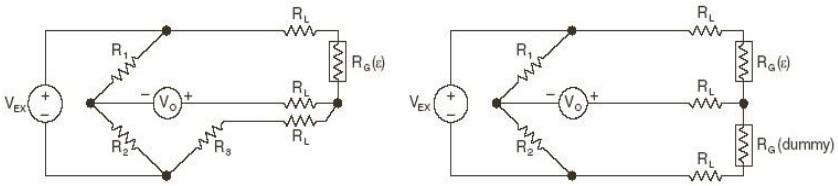
недеформованому стані введено відношення Vr

$$V_r = \frac{V_{0(\text{деф})} - V_{0(\text{ без деф})}}{V_{EX}}$$

де $V_{O\ (деф)}$ – виміряна вихідна напруга під час деформації, а $V_{O\ (без\ деф)}$ – початкова напруга, тобто вихідна напруга без деформації. V_{EX} – напруга збудження.

Позначимо (+ε) і (-ε) активні тензодатчики, що працюють на розтягнення й на стиснення, відповідно. Позначення (-νε) введемо для тензодатчика, включеного в поперечному напрямку, так що його опір змінюється здебільшого внаслідок Пуасонівської деформації, амплітуду якого позначають як -νε. На рисунку 2.47 показані мостові схеми включення тензодатчиків, та рівняння до них.

Інші позначення, що використовують у рівняннях:
- R_G = Номінальне значення опору тензодатчика;
- GF = коефіцієнт тензочутливості тензодатчика;
- R_L = Опір підвідних дротів.

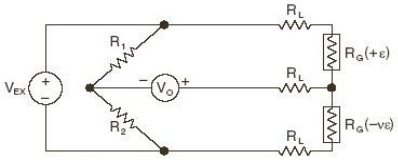


$$\varepsilon = \frac{-4V_r}{GF(1+2V_r)} \bullet \left(1+\frac{R_L}{R_G}\right)$$

Четвертьмостова схема 1

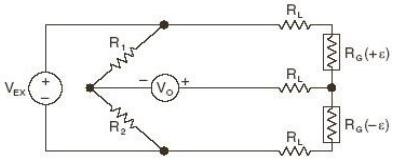
$$\varepsilon = \frac{-4V_r}{GF(1+2V_r)} \bullet \left(1+\frac{R_L}{R_G}\right)$$

Четвертьмостова схема 2



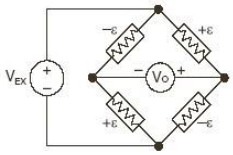
$$\varepsilon = \frac{-4V_r}{GF[(1+\nu)-2V_r(\nu-1)]} \bullet \left(1+\frac{R_L}{R_G}\right)$$

Півмостова схема 1



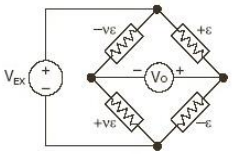
$$\varepsilon = \frac{-2V_r}{GF} \bullet \left(1+\frac{R_L}{R_G}\right)$$

Півмостова схема 2



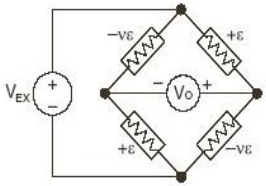
$$\varepsilon = \frac{-V_r}{GF}$$

Повномостова схема 1



$$\varepsilon = \frac{-2V_r}{GF(\nu+1)}$$

Повномостова схема 2



$$\varepsilon = \frac{-2V_r}{GF[(\nu+1)-V_r(\nu-1)]}$$

Повномостова схема 3

Рисунок 2.47 – Мостові схеми та тензометричні рівняння

Список літератури

1. Мікроконтролери: архітектура, програмування та застосування в електромеханіці : навч. посіб. / Ю. С. Грищук. – Харків : НТУ «ХПІ», 2019. – 384.
2. Smythe R. J. Advanced Arduino Techniques in Science – Wainfleet, ON, Canada, 2021. – 279 p.
3. Margolis M, Jepson B., Weldin N. Arduino Cookbook, 3rd Edition - – O'Reilly Media, Inc, 2020. – 800 p.
4. Banzi M. Getting Started with Arduino, 3rd Edition / CA : O'Reilly Media, 2015. – 262 c.
5. Monk C. Programming Arduino: Getting Started with Sketches, 2nd edition. – New York : McGraw-Hill Education, 2016. – 192 c.
6. Blum J. Exploring Arduino. – Hoboken, NJ : Wiley, 2013. – 384 c.
7. Pardum J. Beginning C for Arduino, Second Edition. – New York : Apress, 2015. – 288 c.
8. Geddes M. Arduino Project Handbook: 25 Practical Projects to Get You Started. – San Francisco, CA : No Starch Press, 2016. – 272 c.
9. Boxall J. Arduino Workshop: A Hands-On Introduction with 65 Projects. – San Francisco, CA : No Starch Press, 2013. – 392 c.
10. I. H. Witten, E. Frank, and M. A. Hall. Data Mining: Practical Machine Learning Tools and Techniques. – San Francisco, CA : Morgan Kaufmann, 2016. – 654 p.
11. J. Han, M. Kamber, and J. Pei. Data Mining: Concepts and Techniques. – Amsterdam, Netherlands : Elsevier, 2011. – 744 p.
12. Мікропроцесорна техніка» : конспект лекцій : навч. посіб. / КПІ ім. Ігоря Сікорського ; уклад.:

Т. О. Терещенко, О. В. Хоменко. Київ : КПІ ім. Ігоря Сікорського, 2017. – 165 с.

13. Схемотехніка електронних систем : підручник : у 3 кн. Кн. 3. Мікропроцесори та мікроконтролери / В. І. Бойко та ін. – 2-ге вид., допов. і переробл. – Київ : Вища шк., 2004. – 399 с.

14. Kirinaki N. Data Acquisition and Signal Processing for Smart Sensors 1st Edition. – Wiley, 2002. – 320 p.

15. Pullela S., Desai P. Handbook of Data Acquisition and Signal Processing. – Measurement Computing Corporation, 2012. – 145 p.

16. Maurizio P. Data Acquisition Systems: From Fundamentals to Applied Design. Springer, 2015.

Електронне навчальне видання

Тищенко Костянтин Володимирович,
Ткач Олена Петрівна

ПРОГРАМУВАННЯ СИСТЕМ ЗБОРУ ТА АНАЛІЗУ ДАНИХ

Навчальний посібник

Дизайн обкладинки К. В. Тищенка
Редактори: С. М. Симоненко, Н. М. Мажуга, О. Ф. Дубровіна
Комп'ютерне верстання К. В. Тищенка

Формат 60x84/16. Ум. друк. арк. 11,04. Обл. вид. арк. 10,86.

Видавець і виготовлювач
Сумський державний університет,
вул. Римського-Корсакова, 2, м. Суми, 40007
Свідоцтво суб'єкта видавничої справи ДК № 3062 від 17.12.2007.