МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ІНТЕГРОВАНІ КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ ПРОЄКТУВАННЯ

КУРС ЛЕКЦІЙ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою
«Системи керування літальними апаратами та комплексами»
спеціальності 173 Авіоніка

Укладачі: Ю.В. Бобков, П.О. Піщела

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

УДК 004.43, 004.9 (075)
Х71

Укладачі: *Бобков Юрій Володимирович*, канд. техн. наук, доц.
Піщела Павло Олександрович

Рецензент *Лук'янов П. В.,* канд. фіз.-мат. наук, ст. наук. співробітник, в.о. завідувача кафедри АРБ ННІАТ КПІ ім. Ігоря Сікорського

Відповідальний редактор *Чепілко М. М.*, докт. фіз.-мат. наук, проф.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 1 від 02.10.2025 р.)
за поданням Вченої ради Навчально-наукового інституту аерокосмічних технологій
(протокол № 7/25 від 29.09.2025 р.)

Х71

Інтегровані комп'ютерні технології проєктування [Електронний ресурс] : курс лекцій : навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Системи керування літальними апаратами та комплексами» спец. 173 Авіоніка / КПІ ім. Ігоря Сікорського; уклад.: Ю.В. Бобков, П.О. Піщела. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 125 с.

Навчальний посібник призначений для забезпечення лекційних занять з дисципліни «Інтегровані комп'ютерні технології проєктування», що належить до вибіркових освітніх компонентів циклу професійної підготовки здобувачів ступеня бакалавра спеціальності 173 «Авіоніка», за освітньою програмою «Системи керування літальними апаратами та комплексами».
На сучасному етапі розвитку підвищення ефективності розробки систем керування літальними апаратами та комплексами можливе, в першу чергу, за рахунок застосування новітніх досягнень в області комп'ютерного проєктування. Одним з потужних засобів комп'ютерного проєктування є графічне середовище розробки LabVIEW, застосуванню якого для розробки та моделювання різноманітних систем присвячений цей навчальний посібника.
Матеріали посібника можуть бути застосовані також при викладанні вибіркової дисципліни «Комп'ютерне проєктування систем авіоніки», а також можуть бути корисними здобувачам інших спеціальностей та фахівцям, що цікавляться застосуванням середовища LabVIEW в інженерному проєктуванні.

УДК 004.43, 004.9 (075)

Реєстр. № НП 25/26-051. Обсяг 5 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП

При проєктуванні та дослідженні складних об’єктів широко застосовуються різноманітні комп’ютерні технології. Вони дозволяють розробити комп’ютерну модель, що відтворює реальний об’єкт, розрахувати її основні параметри та дослідити функціонування в заданих умовах, що також моделюються. Отримані результати комп’ютерного моделювання переносяться на реальні об’єкти. Такий алгоритм дозволяє значно скоротити терміни та витрати при проєктуванні та дослідженні різноманітних систем.

Для вирішення задач, що потребують виготовлення макетних дослідних зразків, або дослідження впливу на об’єкт є досить працезатратними процесами. На даному етапі розвитку науки і техніки розроблені різноманітні комп’ютерні програми, середовища і пакети, що враховують специфіку відповідної предметній галузі та спрощують розробку виробів. При цьому основний акцент робиться на тому, щоб програмне забезпечення могли використовувати спеціалісти цієї галузі у звичній для них формі.

Зокрема, при вирішенні задач проєктування систем керування літальними апаратами та комплексами може застосовуватись графічне середовище розробки LabVIEW. Воно дозволяє створювати системи: збору, обробки та візуалізації інформації; керування різними об’єктами, технологічними процесами та пристроями. За допомогою LabView можна створювати і звичайне прикладне програмне забезпечення, наприклад, аналізувати та обробляти отримані від мікроконтролера дані, витрачаючи мінімальний час на програмування і не використовуючи низькорівневі мови програмування. Перевагами LabVIEW є зручний графічний інтерфейс, велика кількість стандартних блоків, що реалізують більшість функціональних елементів та процесів. Створені в LabVIEW віртуальні прилади можуть безпосередньо підключатись за допомогою спеціальних апаратних засобів до реальних об’єктів для отримання інформації, її обробки та подальшої видачі необхідних команд, знов таки на реальні об’єкти.

В навчальному посібнику з курсу лекцій вибіркової дисципліни «Інтегровані комп’ютерні технології проектування» розглянуті загальні питання щодо організації середовища LabVIEW, а також його застосування для розробки та моделювання технічних систем. Суттєва увага приділена типам даних, елементам графічного відображення, генерації сигналі, вводу/виводу даних тощо. Для покращення засвоєння матеріалу в посібнику наведено багато ілюстрацій та практичних прикладів, що дають змогу перейти до безпосереднього застосування теоретичного матеріалу для проєктування систем керування літальними апаратами та комплексами на комп’ютері.

Матеріали посібника можуть бути застосовані також при викладанні вибіркової дисципліни «Комп’ютерне проєктування систем авіоніки», а також можуть бути корисними здобувачам інших спеціальностей та фахівцям, що цікавляться застосуванням середовища LabVIEW в інженерному проєктуванні.

1 ІНТЕГРОВАНІ КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ ПРОЄКТУВАННЯ ТА ЇХ МІСЦЕ В ПРОФЕСІЙНІЙ ДІЯЛЬНОСТІ

В найбільш загальному розумінні комп'ютерне проєктування є наукою про розробку і використання комп'ютерних програм, призначених для моделювання, проєктування та дослідження різноманітних об'єктів.

Саме конкретний вид об'єктів визначає й особливості щодо виду та призначення комп'ютерних технологій та програмних продуктів.

В найбільш простому варіанті для проєктування різноманітних комп'ютерних систем можуть застосовуватись мови програмування, наприклад, C++, Java, Python та ін., а також алгоритми та структури даних; бази даних, операційні системи (ОС), у тому числі ОС реального часу.

Значне місце займають інтегровані системи автоматизованого проєктування систем, конструкцій та технологічних процесів різного призначення, що об'єднані в систему CAD/CAM/CAE:

1) CAD — система конструювання (Computer Aided Design), модулі якої призначені переважно для виконання графічних робіт та графічного проєктування;

2) CAM — система для вирішення задач виробництва за допомогою комп'ютерів (Computer Aided Manufacturing), яка призначена переважно для вирішення завдань технологічної підготовки виробництва;

3) CAE — аналітично-розрахункова підсистема (Computer Aided Engineering), яка призначена для виконання інженерних розрахунків, аналізу та перевірки проєктних рішень.

Для вирішення задач авіаційної та ракетно-космічної техніки, зокрема для розробки та проєктування систем керування літальними апаратами та комплексами, можуть використовуватись різні комп'ютерні середовища, що дозволяють створювати електронні схеми, різноманітні блоки, вузли, пристрої, прилади та системи в цілому, наприклад: MicroCap, Altium Designer, Multisim, LabVIEW, MatLab/Simulink та інші. Кожне з них має свої особливості та сферу застосування.

В даному курсі розглядається застосування для вирішення задач проєктування систем керування літальними апаратами та комплексами графічного середовища розробки LabVIEW.

1.1 Графічне середовище розробки LabVIEW

LabVIEW – це один з основних продуктів компанії National Instruments. Слід зазначити, що LabVIEW – це абревіатура, яка розшифровується як **Lab**oratory **V**irtual **I**nstrumentation **E**ngineering **W**orkbench.

Вже з назви зрозуміло, що продукт орієнтований на лабораторні дослідження, вимірювання та збір даних, створення пристроїв керування.

LabVIEW – це середовище розробки та платформа для виконання програм, створених на графічній мові програмування «G» фірми National Instruments.

Простішою мовою, LabVIEW – це середовище створення додатків для задач збору, обробки, візуалізації інформації від різноманітних приладів, лабораторних установок і т.п., а також для керування технологічними процесами і пристроями. За допомогою LabVIEW можна також створювати і звичайне прикладне програмне забезпечення, наприклад, аналізувати та оброблювати отримані від мікроконтролера дані, витрачаючи при цьому мінімальний час на програмування та не використовуючи низько рівневі мови програмування [1].

Дискусія з питання чи є LabVIEW мовою програмування відкрита, оскільки тут немає стандарту, як, наприклад ANSI «C». У вузьких кругах розробників кажуть, що пишуть його на мові програмування «G». Фактично такої мови програмування не існує, але в цьому і є перевага даного середовища розробки: від версії до версії в програму вводяться нові конструкції. Важко уявити, що в новій версії мови «C» з’явиться, наприклад, нова структура для *for*- циклу. А в LabVIEW це цілком можливо.

1.1.1 Історія створення LabVIEW

Компанія National Instruments була створена в 1976 році трьома засновниками – Джеффом Кодоски (Jeff Kodosky), Джеймсом Тручардом (James Truchard) та Біллом Новіним (Bill Nowlin) в американському місті Остін (Austin), штат Техас. Основною спеціалізацією компанії є інструментальні засоби для вимірювання та автоматизації виробництва.

Перша версія LabVIEW була випущена в 1986 році (версія для Apple Mac). Інженери компанії вирішили створити антипод «традиційним» мовам програмування – графічне середовище розробки. Основним ідеологом графічного підходу став Джефф. Рік за роком випускались нові версії. Першою кросплатформовою версією (включаючи Windows) була третя версія, випущена в 1993 році.

В Остіні по сьогоднішній день розташований головний офіс компанії. Сьогодні в компанії працюють близько чотирьох тисяч людей, а офіси знаходяться майже в сорока країнах світу.

Кросплатформове графічне середовище розробки додатків LabVIEW універсальна мова програмування. Не дивлячись на те, що продукт тісно зв’язаний з апаратним забезпеченням компанії National Instruments, тим не менш LabVIEW, не має прив’язі до конкретної комп’ютерної системи. Даний продукт з успіхом може застосовуватись для створення приладів і великих систем, для обробки текстів і зображень, для роботи з базами даних тощо. Вихідний код програм написаних в LabVIEW переносимий, і додатки будуть виглядати однаково в усіх системах (Windows, Linux, MacOS).

LabVIEW – досить високорівнена мова програмування. Але є можливість включати «низькорівневі» модулі на інших мовах програмування. Наприклад, можливе використання ассемблерних вставок, для чого потрібно сгенерувати DLL і додати виклики в код.

З іншого боку, високорівневість дозволяє швидко виконувати досить не тривіальні операції з даними, на які в звичайних мовах програмування може бути написано багато строчок коду. Проте деякі операції низькорівневих мов (наприклад, роботу з покажчиками), не так просто реалізувати в LabVIEW, зважаючи на його «високорівневість».

Основні конструкції мови програмування LabVIEW, наявні та аналогічні «традиційним» мовам [1]:

- змінні (локальні, глобальні);

- розгалуження;

- цикли (for, while);

- групування операцій.

1.1.2 Переваги LabVIEW

1 LabVIEW створювалась для інженерів, а не для програмістів. Тому це середовище (і мова програмування G) зроблені максимально інтуїтивно зрозумілими. Тому швидкість його опанування досить мала.

2 Відлагодження програм більш наочне, враховуючи графічний процес розробки. Елементами синтаксису є об’єкти (термінали, функції, вузли і т.п.), що спрощує пошук помилок.

3 LabVIEW створений для написання програма для автоматизації в промисловості та навчальних лабораторій, тому для роботи з периферією у своєму складі має дуже великий набір бібліотек (робота з RS-232 (COM порт) та RS-485, LPT, USB, протоколи TCP /IP, UDP і т.д.). Зараз дуже багато відомих виробників систем збору даних, камер машинного зору, промислового обладнання поставляють разом з ними готовий набір бібліотек для їх

застосування в LabVIEW, з великою кількістю прикладів. Це дозволяє в стислий термін розпочати роботу з обладнання та підлаштувати його під свої завдання.

LabVIEW використовується для програмування різних DAQ-пристроїв, систем контролю зображення та руху, апаратних засобів, що мають інтерфейси типу GPIB, VXI, PXI, RS-232 та RS-485. За допомогою програмного середовища LabVIEW можна розробляти програмно-апаратні комплекси для тестування, вимірювання, введення даних, аналізу та керування зовнішнім обладнанням.

4 LabVIEW підтримує роботу з обладнанням компанії National Instruments.

5 В LabVIEW є великий набір бібліотек для обробки та аналізу сигналу, різні перетворювачі, фільтри тощо. А при використанні додаткових модулів обробки сигналу у потрібній області цей список розширюється практично нескінченно.

6 LabVIEW модульна система. Тому встановивши потрібний модуль або тулкіт можна розширити її можливості. Так, є модуль для машинного зору, модуль роботи зі звуком, кроковими двигунами, модуль для генерації звітів у MS Office, Internet модуль для передачі та публікації даних у мережі, для якого використовуються LabVIEW Web Server та програмні стандарти TCP/IP та Active X.

7 LabVIEW дозволяє створювати закінчені додатки та компілювати повноцінний *exe* додаток.

8 LabVIEW кроссплатформове середовище. Крім написання програм під різні види операційних систем, вона дозволяє писати програмне забезпечення для Windows Mobile, вбудованих систем (CompactRIO, систем на основі Linux і Windows Embeded), підтримує ARM мікроконтролери, модуль для FPGA (в основному для обладнання National Instruments, проте можна писати програми і для деяких FPGA від Xilinx).

9 LabVIEW дозволяє створювати закінчені додатки, в дуже стислі терміни, в ній дуже проста система налагодження та тестування програм.

10 LabVIEW дуже легко працює з паралельними потоками, вони дуже легко створюються та керуються, що дуже важливо у великих проєктах.

11 LabVIEW дозволяє створювати як маленькі прості програми, так і величезні проєкти з великою кількістю датчиків та складним інтерфейсом користувача.

12 На LabVIEW є величезна кількість матеріалу, книг та ресурсів, тому проблем з її освоєнням немає.

Це був неповний перелік переваг цього продукту.

1.1.3 Недоліки LabVIEW

1 Незважаючи на величезну гнучкість даного середовища, є деякі обмеження, наприклад, утиліти в LabVIEW писати незручно, та й ігрові процеси чи текстовий процесор (типу Word) написати неможливо.

2 LabVIEW дещо поступається за продуктивністю іншим мовам програмування (це ціна за швидкість та гнучкість), але не значно. В теперішній час, коли потужність комп'ютерів зростає, це не дуже великий недолік.

3 Щоб запустити програму, створену за допомогою LabVIEW на комп'ютері, на якому саме середовище не встановлено, потрібно ставити спеціальний безкоштовний додаток LabVIEW Run-Time, або робити інсталятор для автоматичного встановлення Run-Time. Але LabVIEW Run-Time має досить великий обсяг (для простого застосування близько 200 Мбайт).

4 LabVIEW призначений в першу чергу для вирішення кола завдань пов'язаних з автоматизацією, збором та обробкою даних з датчиків і т.п., а тому у більш загальних завданнях часто програє класичним середовищам програмування.

1.1.4 Застосування LabVIEW

За допомогою LabVIEW дуже легко і швидко перевірити алгоритм роботи пристрою, змоделювати, переглянути поведінку моделі, а потім перенести на кінцевий реальний пристрій. Дуже доречним є застосування LabVIEW для збору даних з пристроїв, що виготовляються на основі мікроконтролерів, а також для їх налагодження [2].

В ньому також є широкі можливості для написання програмного забезпечення до різних засобів вимірювання (осцилографи, логічні аналізатори, гіроскопи, акселерометри і т.д.).

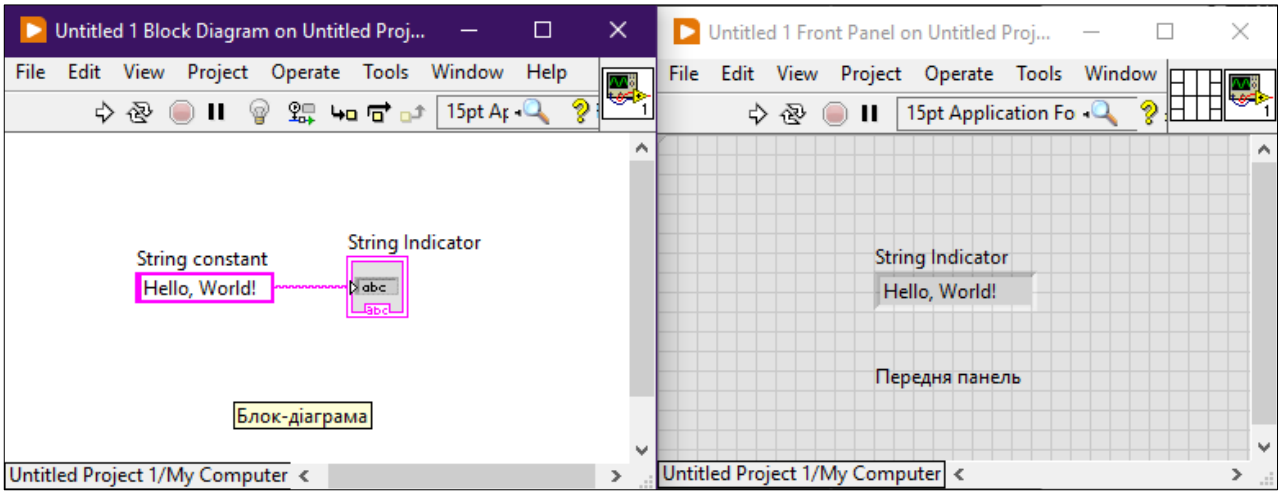
У LabVIEW програмні модулі називаються Virtual Instruments (Віртуальні Інструменти) або VI. Вони зберігаються у файлах із розширенням *.vi.

VI – це блоки, з яких складається LabVIEW – програма. Будь-яка LabVIEW програма містить щонайменше один VI. У термінах мови **C** можна досить сміливо провести аналогію з функцією, з тією різницею, що в LabVIEW одна функція міститься в одному файлі.

Можливо також створювати бібліотеки інструментів з VI. Зрозуміло, один VI може бути викликаний з іншого VI. В принципі, кожен VI складається з двох частин: Блок-Діаграма (**Block Diagram**) та Передня Панель (**Front Panel**).

Блок-діаграма – це програмний код (точніше візуальне графічне представлення коду), а Передня панель – це інтерфейс. Приклад найпростішого VI представлений на рисунку 1.1.

Достатньо широко замість **VI** для найменування створених у LabVIEW програмних елементів та пристроїв застосовують поняття «віртуальний прилад» (**ВП**).



а) б)

Рисунок 1.1 – Приклад Virtual Instruments (а – Block Diagram, б – Front Panel)

В основі LabVIEW лежить парадигма потоків даних – принцип DataFlow.

Dataflow (потік даних) – загальний термін, що відноситься до алгоритмів або архітекторів паралельних обчислень, в яких виконання кожної операції проводиться при готовності всіх операндів, при цьому послідовність виконання команд заздалегідь не задається [1].

Вперше графічну модель обчислень, керованих потоком даних, запропонував у 1968 р. у своїй докторській дисертації співробітник Стенфордського університету Дуайн Едемс (він же ввів термін dataflow).

У наведеному прикладі константа і термінал індикатора з’єднані між собою лінією. Ця лінія називається Wire. Можна назвати її «дротом», провідником. По дротах (провідниках) передаються дані від одних елементів іншим. Уся ця концепція називається Data Flow.

Блок-діаграма – це вузли (ноди), виходи одних вузлів приєднані до входів інших вузлів. Вузол розпочне виконання лише тоді, коли прибудуть усі необхідні для роботи дані.

На діаграмі вище дві ноди. Одна з них – константа. Цей вузол самодостатній – він починає виконання негайно. Другий вузол – індикатор. Він відобразить дані, які передає константа, але не відразу, а тільки коли прибудуть дані від константи.

Розглянемо, складніший приклад: додавання та множення двох чисел.

У традиційних мовах програмування треба написати приблизно таку програму:

```
int x, y, sum, mul;
//…
sum = x + y;
mul = x * y;
```

Аналогічний приклад реалізації в LabVIEW наведений на рисунку 1.2.

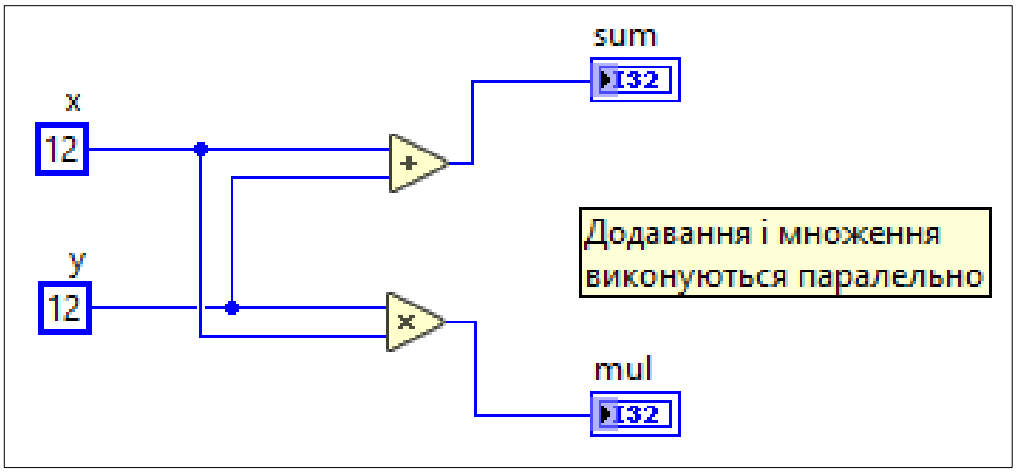


Рисунок 1.2 – Простий приклад LabVIEW з числовими операціями

Зверніть увагу на те, що додавання та множення автоматично виконуються паралельно. На двопроцесорному комп'ютері будуть автоматично задіяні обидва процесори.

Вигляд в LabVIEW циклів «**while / for**» та структур «**if / then / else**» показаний на рисунку 1.3.

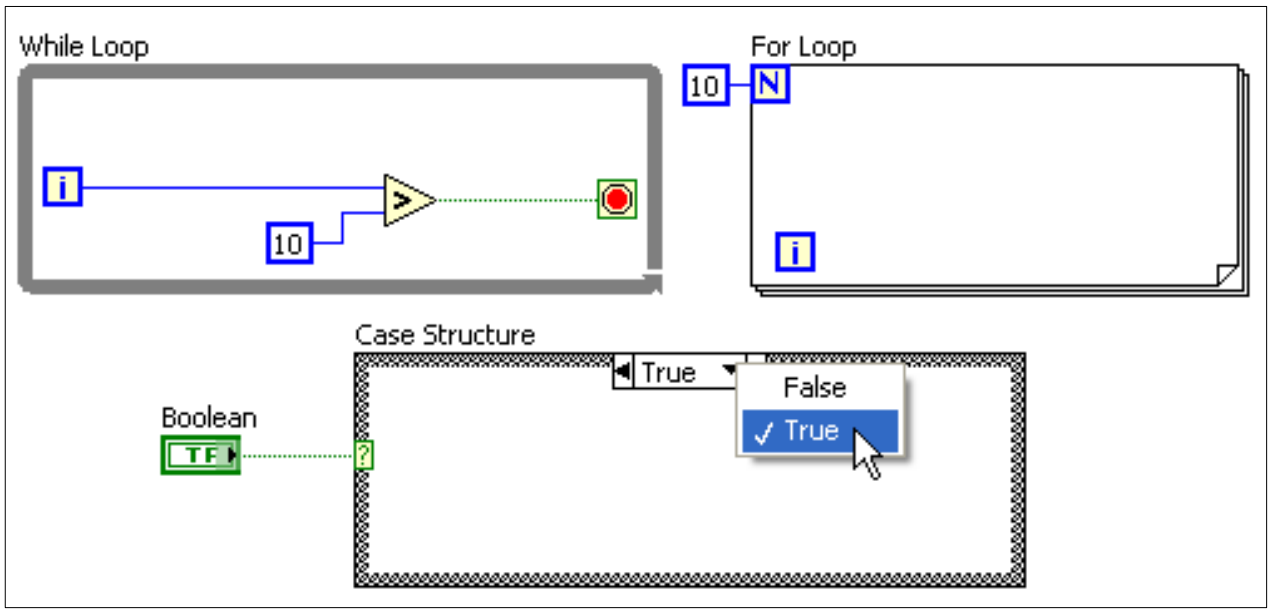


Рисунок 1.3 – Структури LabVIEW

Як згадувалося, всі елементи виконуватимуться паралельно. Інженеру-програмісту не потрібно замислюватися над тим, як розпаралелити завдання на кілька потоків, які можна виконувати паралельно на кількох процесорах.

В останніх версіях програми можна навіть явно вказати на якому з процесорів повинен виконуватися той чи інший while-цикл. Зараз існують надбудови і для текстових мов, що дозволяють просто домогтися підтримки багатопроцесорних систем, але так просто, як на LabVIEW, це ніде не реалізовано.

Якщо казати про багатопоточність, то треба також зазначити, що в розпорядженні розробника багатий вибір інструментів для синхронізації потоків – семафори, черги, рандеву і т.п.

LabVIEW включає багаті набори елементів для побудови інтерфейсів користувача. В LabVIEW процес створення інтерфейсів відбувається значно швидше ніж в Дельфі.

Приклад побудови інтерфейсу LabVIEW показаний на рисунку 1.4.

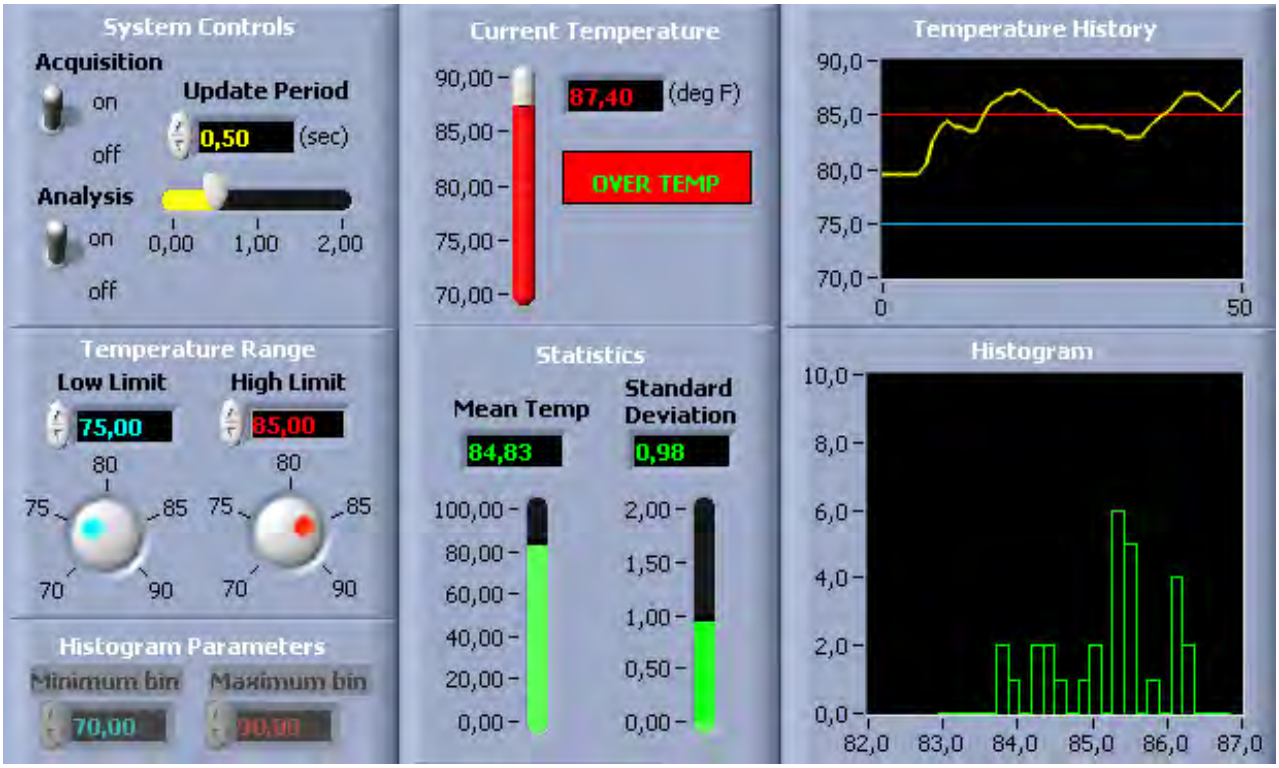


Рисунок 1.4 – Приклад побудови інтерфейсу в LabVIEW

Стандартний комплект LabVIEW включає також блоки для роботи з ***ini***- файлами, реєстром, функції для роботи з двійковими і тестовими файлами, математичні функції, потужні інструменти для побудови графіків. Також до вже згаданої можливості викликів DLL, LabVIEW дозволяє працювати з ActiveX компонентами та .NET.

Починаючи з восьмої версії в LabVIEW було додано підтримку класів — мова стала об’єктно-орієнтованою. Реалізовану підтримку не можна назвати повною, проте основні риси об’єктно-орієнтованих мов — успадкування та поліморфізм зберігаються.

Також функціональність мови можна розширити додатковими модулями, наприклад, NI Vision Toolkit – для обробки зображень та машинного зору та інші. А за допомогою модуля Applcation Builder можна згенерувати exe-файл, що виконується. За допомогою Internet Toolkit можна працювати з ftp- серверами, за допомогою Database Connectivity Toolkit – з базами даних і т.д.

Часто можна почути, що графічний код погано сприймається. Справді, з незвички велика кількість іконок та провідників дещо шокує. Також розробники-початківці створюють програми-«простирадла» і програми-«спагеті». Однак досвідчений LabVIEW-розробник ніколи не створить діаграм, що перевищують розмір екрану, навіть якщо програма складається із сотень модулів. Добре розроблена програма фактично «самодокументується», оскільки в основі лежить графічне представлення (рисунок 1.5).

Може скластись враження, що LabVIEW – інтерпретатор і блок-діаграми постійно інтерпретуються ядром. Проте це не так.

LabVIEW – це компілятор. Проте компіляція відбувається «на льоту» — у будь-який момент розробки програма завжди готова до запуску.

Також LabVIEW-код може бути скомпільований у повноцінний файл, який може бути запущений на комп’ютері без встановленої LabVIEW (але для цього вимагається встановлення утиліти LabVIEW Run-Time).

Також можна зібрати пакет-інсталятор без застосування сторонніх утиліт типу InstallShield.

Розглянемо розробку віртуального приладу VI на простому прикладі.

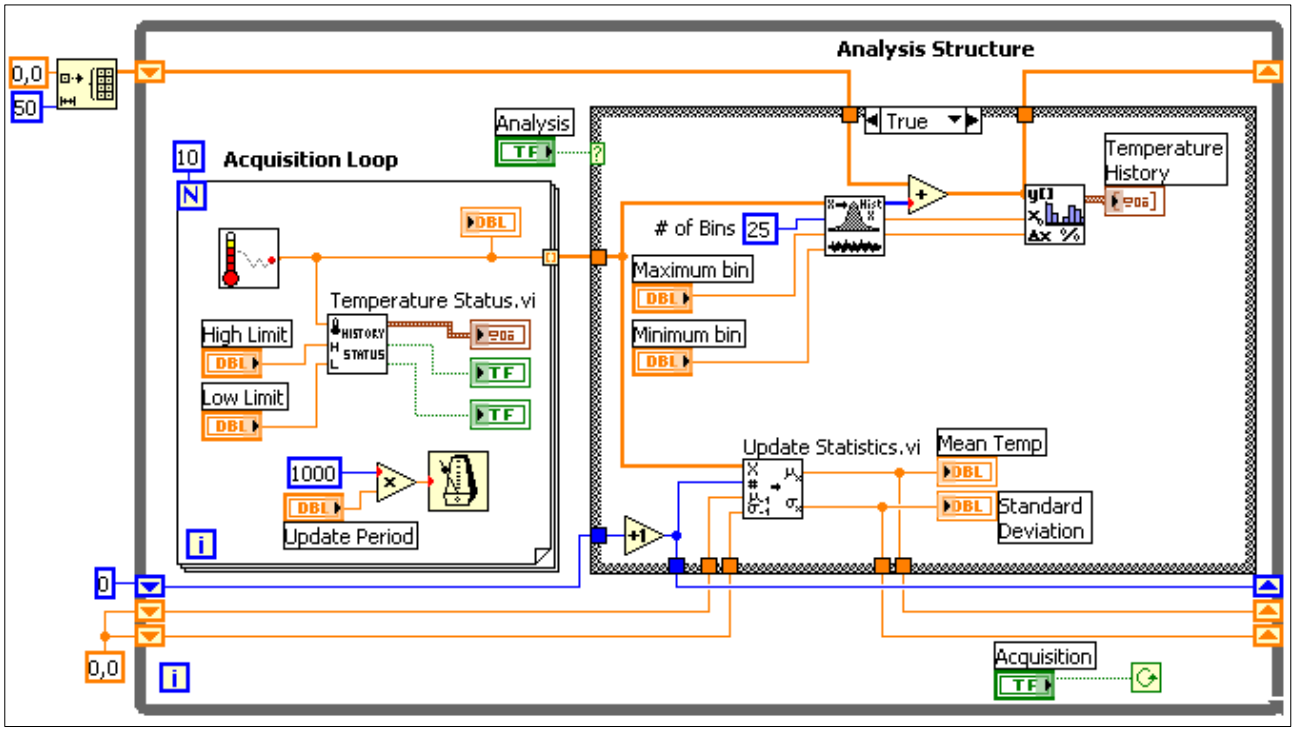


Рисунок 1.5 – Приклад компактної програми LabVIEW

Запустимо пакет LabVIEW, а потім створимо новий віртуальний прилад командою:

New → Blank VI.

Розмістімо на Front Panel два контрола (Controls), для чого треба натиснути послідовно правою кнопкою миші (далі просто ПК) наступні елементи меню:

→ Numeric → Numeric Controls,

і один індикатор натисканням ПК:

→ Numeric → Numeric Indicator.

Незважаючи на те, що імена в LabVIEW, на відміну від класичних текстових мов програмування, можна давати довільні і навіть однакові, бажано

називати їх відповідно функціоналу, щоб у великому проєкті потім не плутатися який термінал до чого відноситься.

Назвемо встановлені елементи так, як показано на рисунку 1.6.

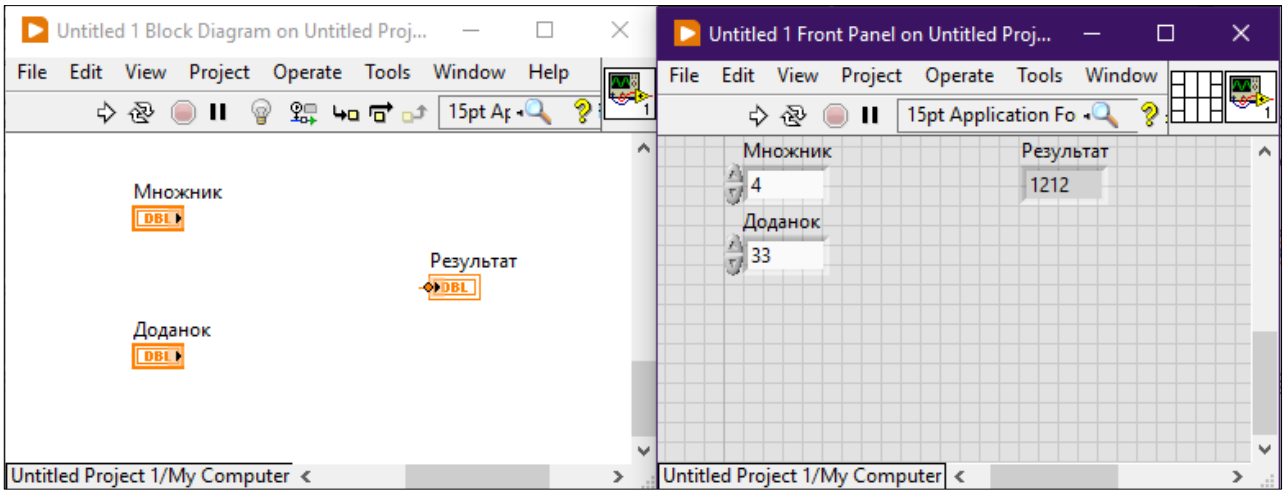


Рисунок 1.6 – Створення додатку в LabVIEW

Після створення елементів керування перейдемо на блок-діаграми (Block Diagram). Для цього треба натиснути клавішу **CTRL+E**.

Block Diagram це «код» програми, який описує алгоритм роботи програми, тоді як Front Panel є інтерфейсом взаємодії з користувачем або іншою підпрограмою.

Після створення елементів керування та відображення на Front Panel, на Block Diagram з’явилися 3 термінали (рисунок 1.6).

Термінал – це представлення елементів керування в блок-діаграмі. Через них ми отримуємо дані та передаємо назад у Front Panel. Є ще інші методи керування контролами та індикаторами на Front Panel, такі як Property Node та Virables, які будуть розглядатись пізніше.

Тепер додаємо на блок-діаграму (БД) три елементи:

- додавання **Add** (ПК → Numeric → Add)

- множення **Multiply** (ПК → Numeric → Multiply).

- зведення у квадрат **Square** (ПК → Numeric → Multiply).

Приєднаємо їх так, як показано на рисунку 1.7.

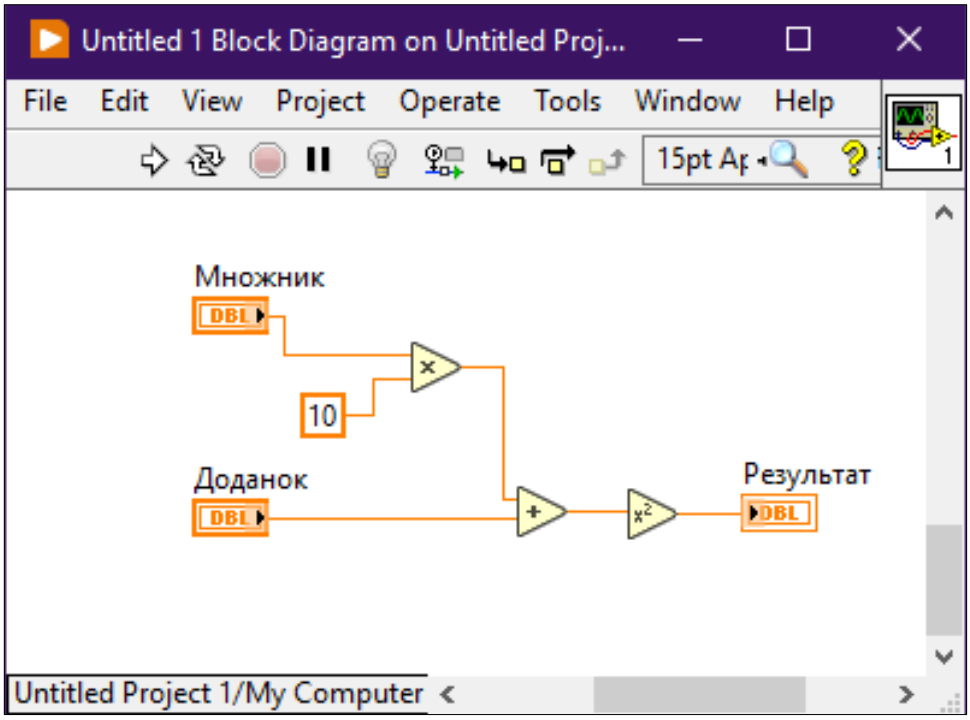


Рисунок 1.7 – Створення додатку з числовими операціями LabVIEW

Щоб створити константу, наприклад, число 10, як другий множник, потрібно натиснути ПК біля введення до вузла множення:

→ Create → Constant .

Для запуску програми треба натиснути клавіши **CTRL+R**, або стрілочку (рисунок 1.8). Оскільки створена програма не містить циклів, вона виконається один раз і завершить свою роботу.

Щоб подивитися, як вона працює, існує «Тривалий запуск» (**Run Continousy**). Для цього треба натиснути кільцеву стрілку (рисунок 1.8). Тривалий запуск дуже зручний інструмент на етапі налагодження підпрограм, тобто VI.

Рисунок 1.8 – Елементи запуску додатку LabVIEW

Для зміни даних в програмі треба змінити значення контролів на Фронт панелі (ФП) і переглянути як змінюється значення на індикаторі «Результат» (рисунок 1.9).

Запуск програми в тривалому режимі дозволяє змінювати значення контролів під час роботи додатку, без перезапуску. На рисунку 1.9 наведено роботу саме в програми в тривалому режимі запуску.

Для запуску режиму налагодження потрібно перейти на **БД** та натиснути **Highlight Execution** (Підсвітити виконання). У цьому режимі видно, як ідуть дані по «провідниках» і видно їх значення. Тут можна спостерігати принцип DataFlow: поки до вузла (множення, або додавання, або будь-якого іншого елемента **БД**) не надійдуть вхідні дані, він не починає виконувати операцію і не передасть вихідні дані далі. Так елемент **Add** очікує, коли вузол **Multiply** виконає множення контрола на константу і тільки потім здійснює додавання, і передає дані на вузол зведення у квадрат (рисунок 1.10).

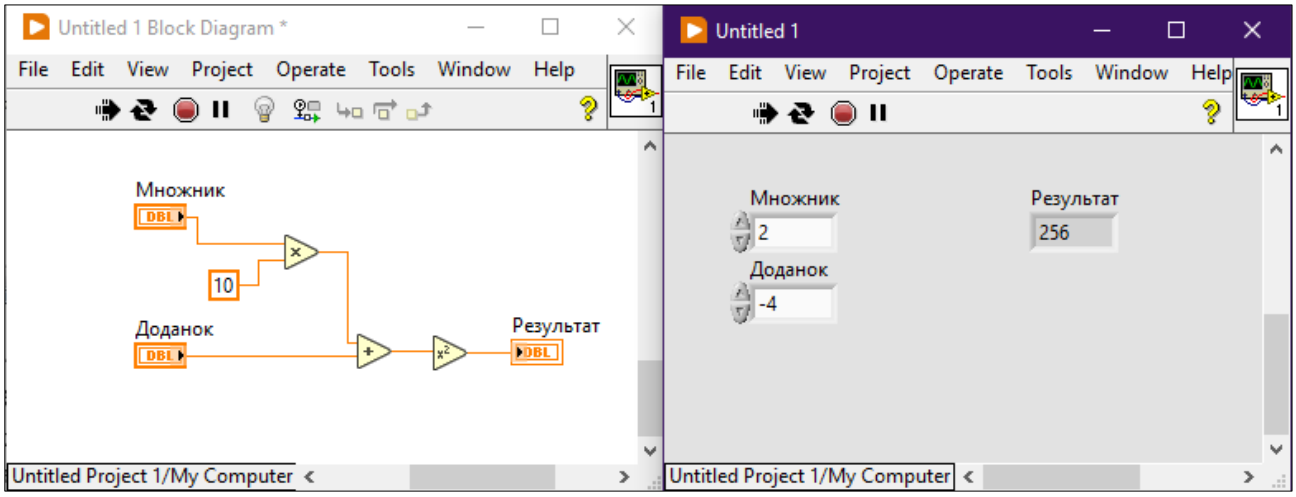


Рисунок 1.9 – Результат виконання розробленого додатку

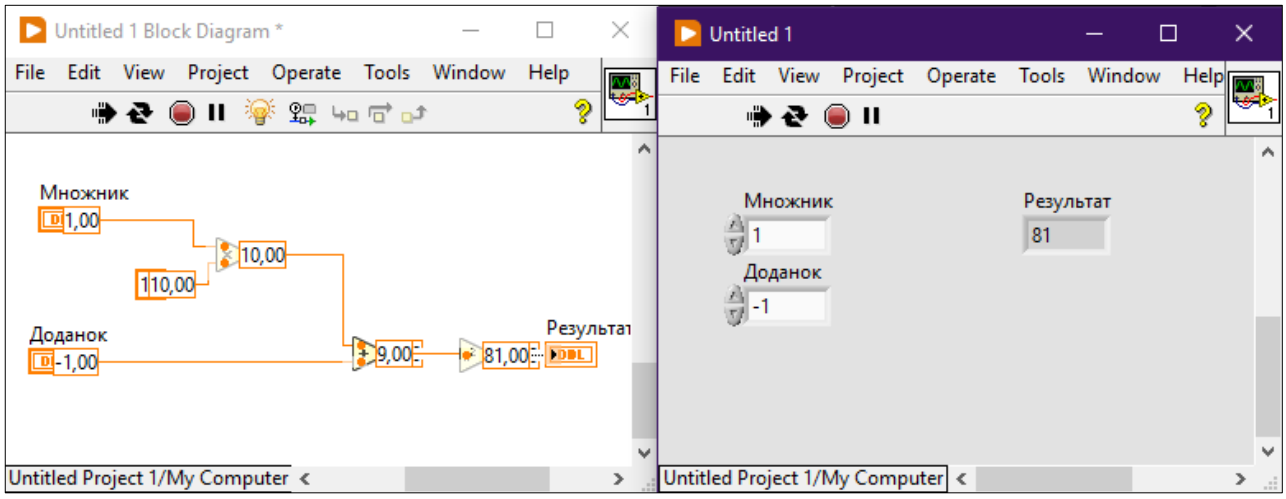


Рисунок 1.10 – Режим налагодження Highlight Execution

При розробці програм для наочності потрібно намагатися, щоб провідники були рівні, з мінімальними вигинами, а також намагатися не перетинати провідники між собою (рисунок 1.11). Також прийнято розташовувати контроли зліва, а індикатори праворуч. Так програма виглядає більш наочною та зрозумілою. В такому випадку добре видно виконання принципу потоку даних.

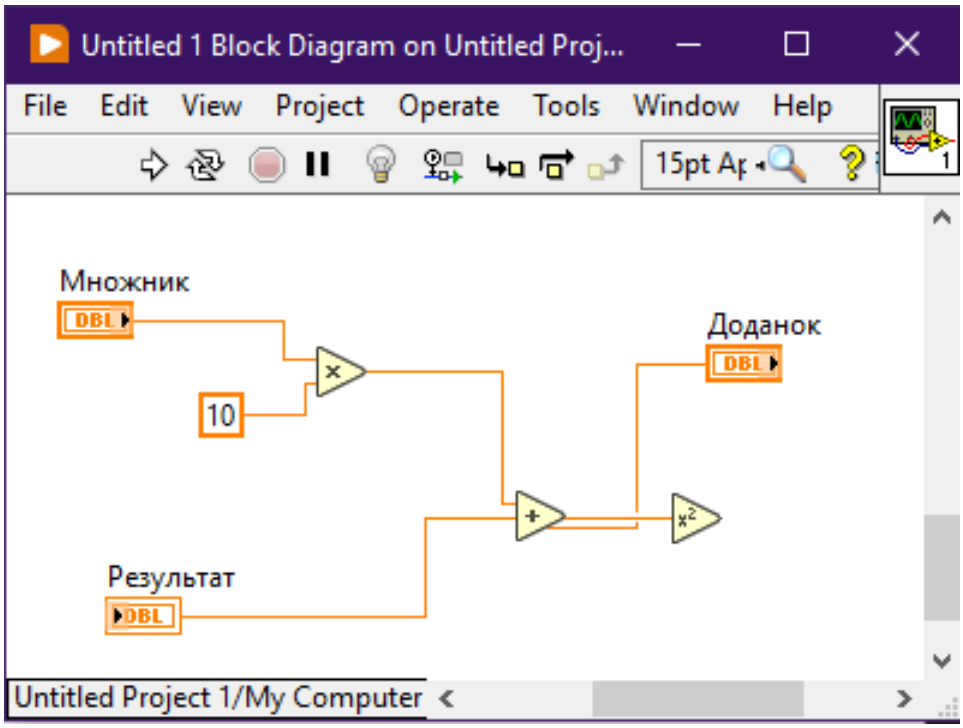


Рисунок 1.11 – Не бажаний стиль розробки, перетинання провідників

Для приведення блоку діаграми до зрозумілого прийнятного вигляду з версії 8.6 з'явився інструмент **CleanUP Diagram**. З версії 2009 з'явилася можливість «підчищати» лише виділену частину блок-діаграми. Але його доцільно використовувати тільки з маленькими блок-діаграмами, оскільки при очищенні відразу великих проєктів, код виходить менш «читабельним», хоча і компактним.

Для копіювання (дублювання) блок-діаграми її треба виділити за допомогою миші і натиснути **CTRL+C**, а потім – **CTRL+V**, як і інших програм Windows. До скопійованих контролів та індикаторів автоматично додаються цифри 2 і на **ФП** з'являються однойменні елементи (рисунок 1.11).

Для того щоб знайти який термінал якому елементу відповідає, можна використовувати подвійний клик ПК на ньому.

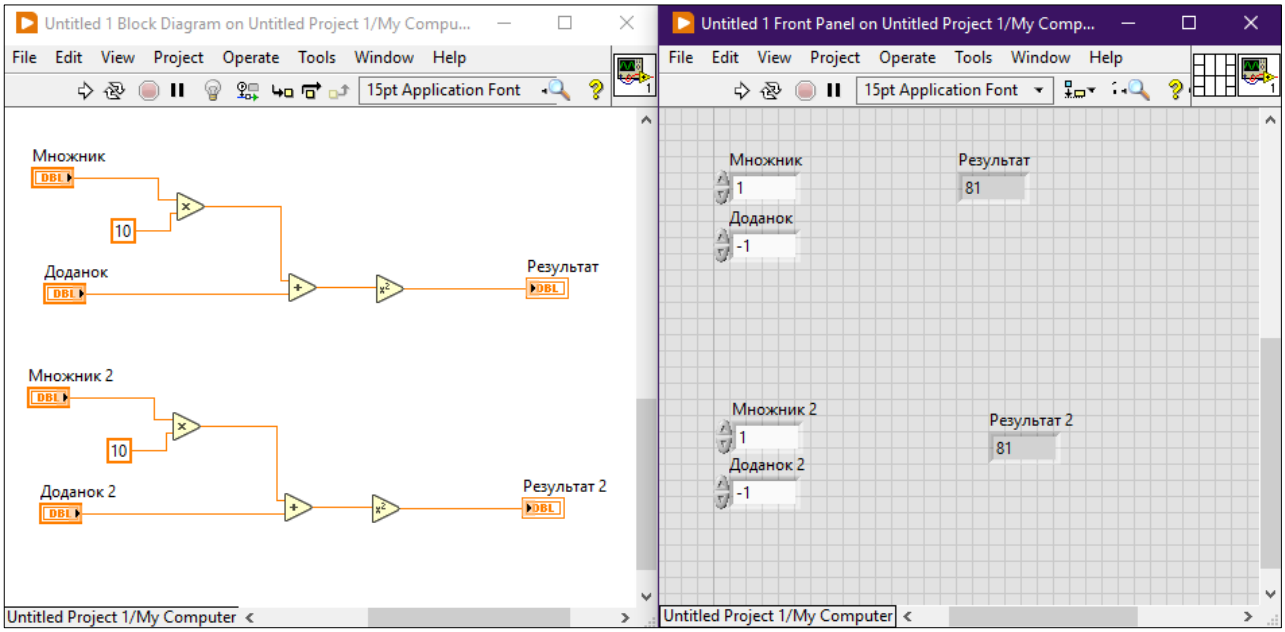


Рисунок 1.12 – Копіювання (дублювання) блок-діаграми

Тепер у нас в програмі два незалежні потоки всередині однієї програми.

Потоки незалежні якщо під час виконання жодний з вузлів одного потоку не чекає даних з вузла іншого потоку. Важко уявити скільки би коду пішло щоб зробити два незалежні потоки всередині однієї програми в текстових мовах програмування (не важливо C або Delphi або інша).

Зазвичай незалежні потоки використовуються в більш-менш серйозних програмах і, дуже часто. У випадку зв’язку з периферійними пристроями [1]. Так, наприклад, в одному потоці може відбуватися прийом даних із COM порту, в іншому – обробка повідомлень від користувача, а в третьому – обробка та запис даних у файл.

Запустіть виконання програми в режимі тривалого запуску та увімкніть підсвічування коду (**Highlight Execution**). Спочатку може здатися, що потоки виконуються послідовно і залежать один від одного, насправді це зроблено, щоб програміст встигав стежити за виконанням програма. Також можна відзначити, що якщо використовується комп’ютер з процесором з двома ядрами, тоді

LabVIEW сама розділяє потоки, так що кожному ядру дістається свій потік. Якщо ж одноядрений процесор, тоді він швидко перемикається між двома потоками, але швидкість виконання буде значно вищою, ніж якби програма виконувалася в одному потоці.

Починаючи з версії 2009 у LabVIEW, з'явилася можливість вказувати, який цикл на якому ядрі виконувати.

1.1.5 Контрольні питання

1.1.5.1 Що таке «віртуальний прилад»?

1.1.5.2 Які основні елементи містить і які функції виконує інструментальна панель лицевої панелі Front Panel **ВП**?

1.1.5.3 Які основні елементи містить і які функції виконує інструментальна панель блок-діаграми Block Diagram **ВП**?

1.1.5.4 Яке призначення палітр в LabVIEW?

1.2 Типи даних LabVIEW

1.2.1 Основні типи даних LabVIEW

В основі LabVIEW, як відзначалось вище, лежить парадигма потоків даних – принцип DataFlow.

Dataflow (потік даних) — загальний термін, що відноситься до алгоритмів або архітектур паралельних обчислень, в яких виконання кожної операції проводиться при готовності всіх її операндів, при цьому послідовність виконання команд не задається заздалегідь [1].

Вперше графічну модель обчислень, керованих потоком даних, запропонував 1968 р. у своїй докторській дисертації співробітник Стенфордського університету Дуайн Едемс (він же запровадив термін dataflow).

Розглянемо типи даних LabVIEW.

По аналогії з традиційними мовами програмування розрізняють на наступні групи:

1) **чисельний** тип даних (**Numeric**), який ділиться на:

- цілочисельні числа (**Integer**), які можуть бути зі знаком (I8, I16, I32, I64) і без знака (U8, U16, U32, U64). Цифра після літери, це число біт, який займає тип. Такі числа показуються на блок-діаграмі (**БД**) синім кольором (у вигляді синіх терміналів);

– числа з плаваючою комою (**Floating point**), які можуть бути: одинарної точності (single (32 bit)), подвійної точності (double (64-bit)) розширеної точності (extended (128-bit)), а також типу **Fixet-Point**, в якому можна задати потрібну точність для дробової та цілої частини. Число з плаваючою комою може бути комплексним. Відображаються числа з

плаваючою комою на **БД** помаранчевим кольором (у вигляді помаранчевих терміналів), а Fixet-Point – сірим кольором;

2) **логічний** тип даних (**Boolean**), який приймає два значення: **Істина** (**True**, «**1**») або **Неправда** (**False**, «**0**»). Показуються на **БД** зеленим кольором (у вигляді зелених терміналів);

3) **строковий** (**String**) тип даних, який містить текст у ASCII форматі. Відображаються на **БД** рожевим кольором (у вигляді рожевих терміналів);

4) **шлях** (**Path**) – це шлях до файлу, що відображається у вигляді терміналів. Шлях до файлу близький до рядкового типу, однак, LabVIEW форматує його, використовуючи стандартний синтаксис для використовуваної платформи;

5) **масиви** (**Array**) – тип, який представляє собою набір (групу, об'єднання) даних одного типу. Елементами масиву називають групу складових його об'єктів.

Масиви елементів можуть мати різну розмірність. Розмірність масиву це сукупність стовпців (довжина) та рядків (висота). Глибина – це загальна кількість елементів у масиві. Масив може мати одну і більше розмірностей з максимальною кількістю до 2^{31} елементів у кожному напрямку, або скільки дозволяє оперативна пам'ять.

Використання масивів зручно при роботі з групами даних одного типу і при накопиченні даних після обчислень, що повторюються. Усі елементи масиву упорядковані. Кожному елементу масиву надано індекс, що забезпечує легкий до нього доступ. Індекс першого елемента масиву завжди 0. Отже, індекси масиву перебувають у діапазоні від 0 до $N-1$, де N – кількість елементів у масиві. Наприклад, для $N = 10$ індекс знаходиться в межах від 0 до 9.

Відображаються масиви на **БД** кольором елементів, якими утворений масив, але товстішими лініями. Якщо ж розмірність масиву подвійна (масив з

масивів) або потрійна, то відображається подвійною або потрійною лінією кольору елементів, що його складають;

6) **кластери** (**Cluster**) – це тип даних, який представляє собою набір (групу, об'єднання) даних різних типів. Якщо кластери включають різні типи даних, то вони відображаються товстою рожевої лінією. Якщо кластер утворений лише чисельними елементами (тобто цілими і/або числами з плаваючою комою), то товстою темно помаранчевою (коричневою) лінією;

7) **сигнальний** тип даних (**Waveform**) – є кластером елементів, що містить дані, початкове значення часу та інтервал часу між вимірами;

8) **динамічний** тип даних (**Dynamic**) – містить крім даних сигналу додаткову інформацію, наприклад назву сигналу або дату і час його отримання. Більшість експрес-**ВП** приймають та/або повертають дані динамічного типу. Дані динамічного типу можна спрямовувати до будь-якого елемента відображення або поля введення, який приймає дані чисельного, логічного або сигнального типу. Динамічний тип відображається у вигляді темно-синіх терміналів.

8) **Variant** — особливий тип даних, до якого можна привести будь-який інший тип. Використовується для взаємодії з компонентами ActiveX, NET і т.д. Відображається на **БД** товстим сірим кольором;

9) **інші** типи. У LabVIEW, крім основних, є інші типи даних, які не зустрічаються в класичних мовах програмування. Наприклад, **refnum** – подібність вказника.

Дані між об'єктами блок-діаграми передаються по сполучним лініям – провідникам даних. Провідник даних аналогічний змінним у текстових мовах програмування.

Кожен провідник даних має єдине джерело даних, але може передавати їх до багатьох **ВП** та функцій. Провідники даних відрізняються кольором, стилем і товщиною лінії, залежно від типу даних, що передаються.

Приклади основних типів провідників даних представлені у таблиці 2.1.

Таблиця 1.1 – Приклади основних типів провідників даних

Тип провідника даних	**Одне значення**	**Одновимірний (1D) масив**	**Двохвимірний (2D) масив**	**Колір**
Чисельний				Помаранчевий (с плаваючою комою). Синій (цілочисельний)
Логічний				Зелений
Строковий				Рожевий

Приклади основних типів даних та провідників типів даних представлені на рисунку 1.13.

Для визначення типу змінної чи константи достатньо викликати Context Help (знак питання «?» на **БД** чи натиснути CTRL+H) та навести курсор миші на об’єкт. Таким же методом можна дізнатись призначення того чи іншого вузла, або підпрограми [3].

Розглянемо застосування основних типів даних на простих прикладах **ВП**.

1.2.2 Застосування типів даних LabVIEW у віртуальних приладах

Розглянемо застосування типів даних LabVIEW у ВП на простому прикладі. Для цього створимо найпростіший віртуальний пристрій, який може складати два числа.

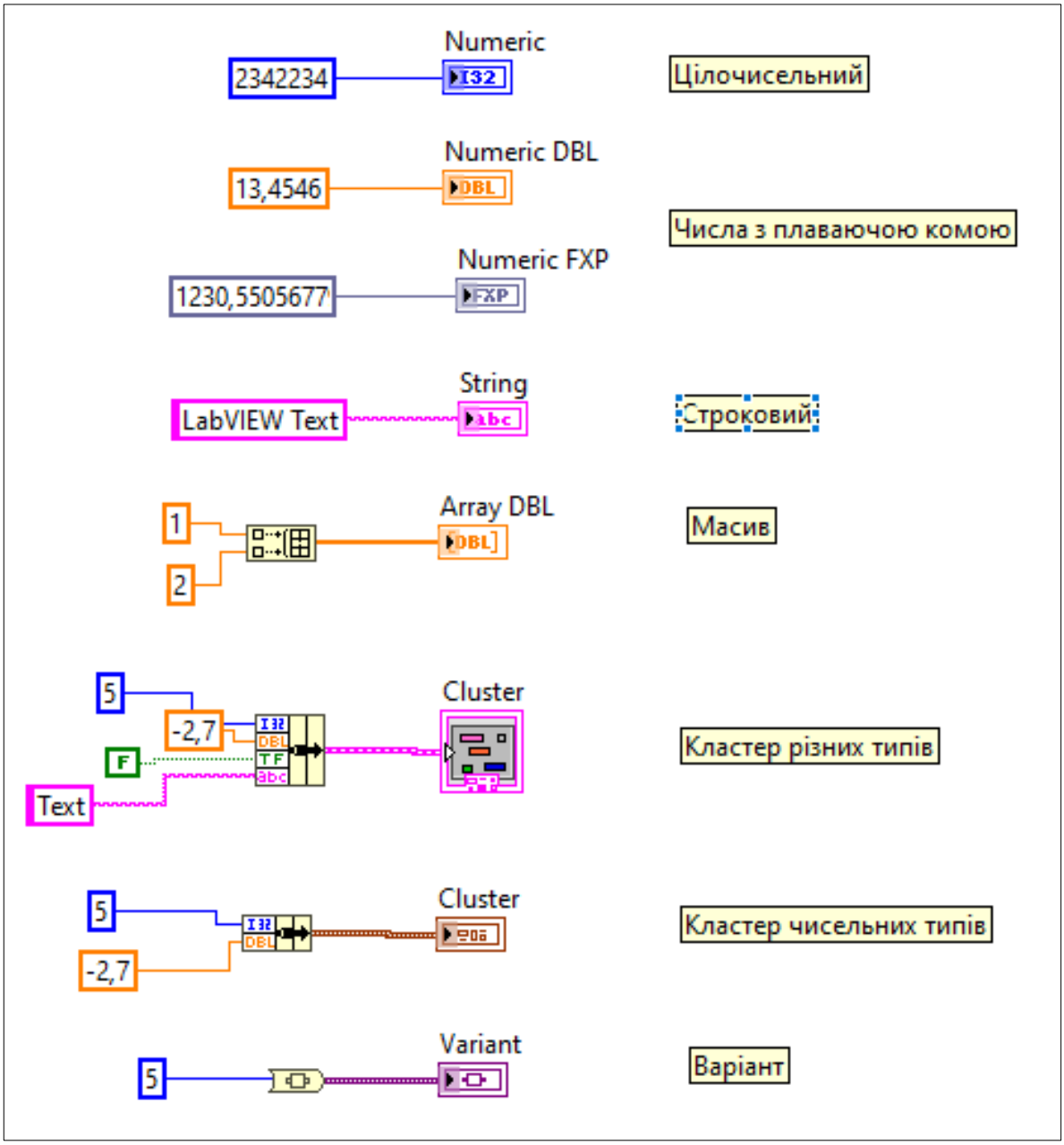


Рисунок 1.13 – Основні типи даних LabVIEW

1 Помістимо на передню панель елемент керування (**Control**), перемістивши покажчик миші на робочу поверхню передньої панелі і натиснувши праву кнопку миші. У меню елементів керування **Controls** (рисунок 1.14) з розділу **Modern** (сучасні) виберемо пункт **Numeric** (числові, цифрові).

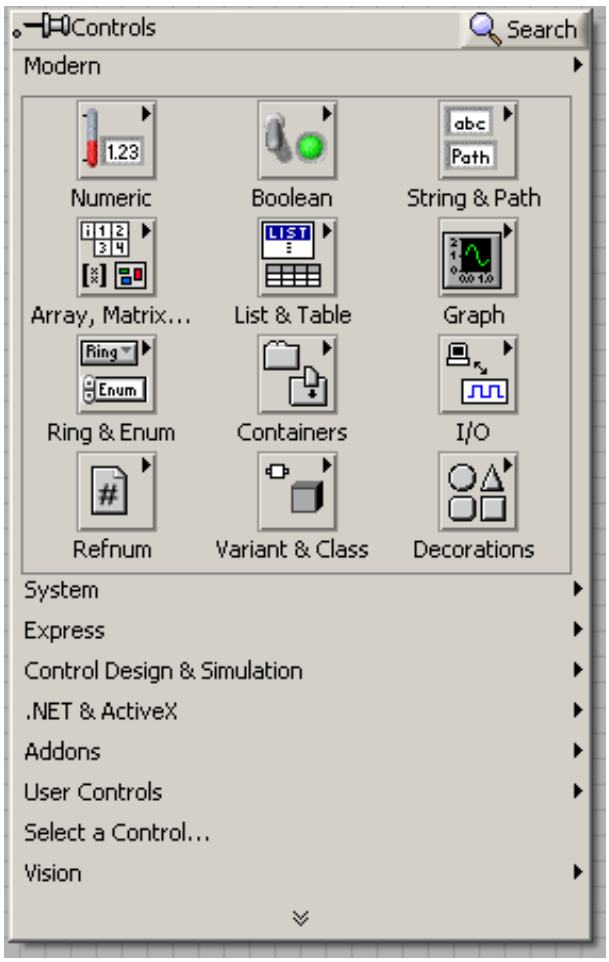


Рисунок 1.14 – Меню Controls та розділ Modern

2 У меню **Numeric** виберемо **Numeric control** (числовий елемент керування), зображений на рисунку 1.15. Вказівник миші набуде вигляду руки, що тримає пунктирні прямокутники – контур елемента керування.

3 Перемістимо вказівник на передню панель і встановимо елемент, натиснувши один раз на ліву кнопку миші. Далі треба двічі клікнути лівою кнопкою миші на напис **Numeric** над елементом керування та ввести текст, наприклад: «Число А».

4 Додаємо на передню панель ще один елемент керування та назвемо його «Число Б».

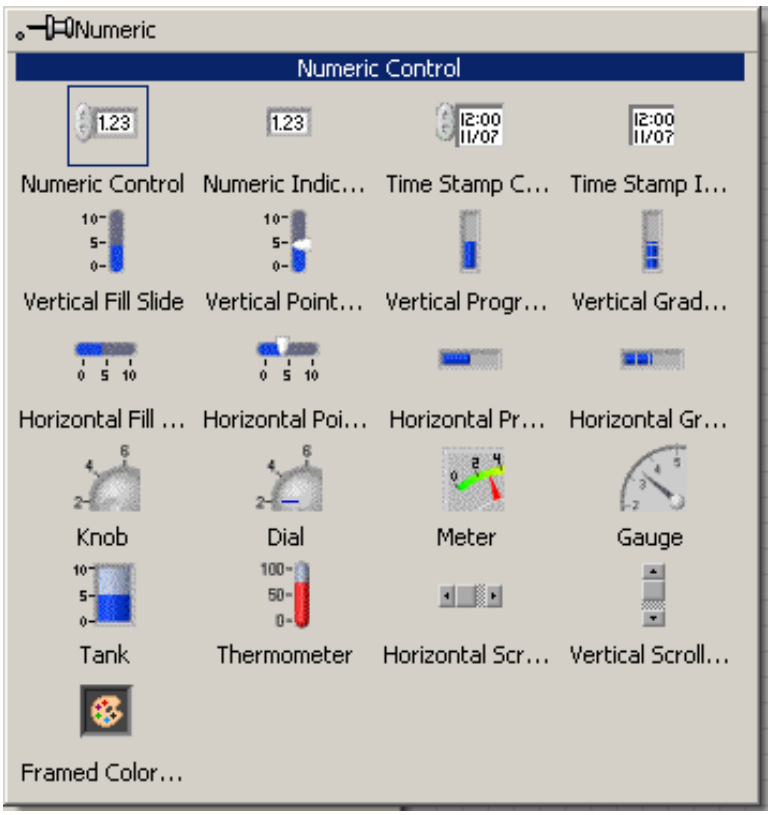


Рисунок 1.15 – Меню Numeric

5 Аналогічним чином додаємо на передню панель індикатор:

Controls → Modern → Numeric → Numeric Indicator.

Назвемо його «Сума».

Вид отриманої передньої панелі показаний рисунку 1.16, а блок-діаграма – на рисунку 1.17.

Для завдання операції додавання потрібно:

2) натиснути правою кнопкою миші на білому полі вікна блок-діаграми. У меню **Functions** (функції), що з'явилось,, виберіть розділ:

Programming (програмування) → Numeric →Add (скласти);

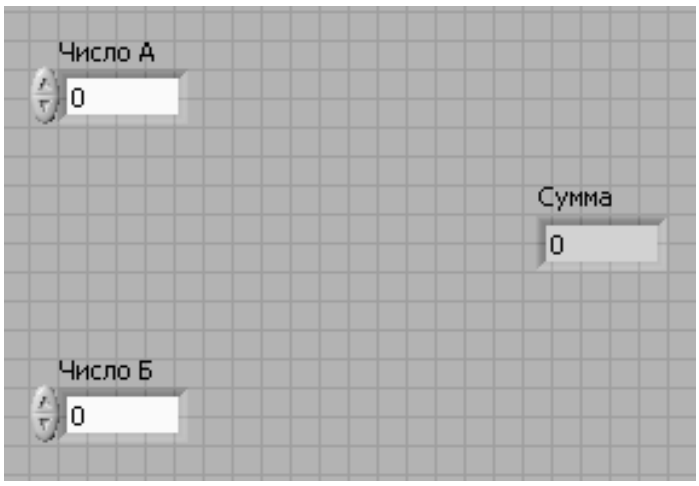


Рисунок 1.16– Елементи керування та індикатор на передній панелі ВП

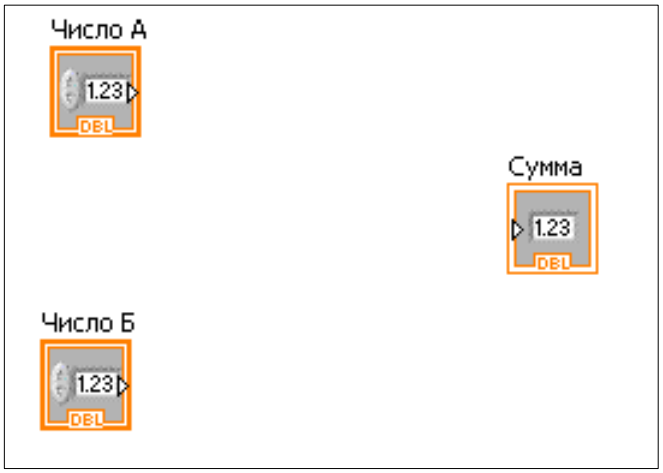


Рисунок 1.17– Термінали елементів керування та індикатора на блок-діаграмі

2) помістити на блок-діаграму піктограму оператора додавання ;

3) клікнути лівою кнопкою миші та підвести інструмент до терміналу Сума;

4) використовуючи інструмент **Wiring tool** (викликається лівою кнопкою миші) з'єднати:

- термінал «Число А» та вхід **x** оператора додавання;

- термінал «Число Б та контакт **y** оператора додавання;

- термінал Сума з контактом **x+y** оператора додавання.

Отриманий результат представлений на рисунку 1.18.

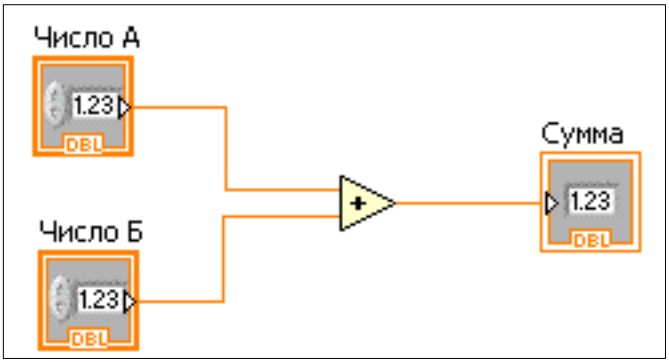


Рисунок 1.18 – Блок-діаграма підсумовуючого ВП

Дослідимо роботу підсумовуючого ВП при різних значеннях та типах вхідних даних (цілочислені, з плаваючою комою, логічні, масиви).

Щоб задати об'єкту той чи інший тип даних, потрібно клікнути правою кнопкою по ньому і потім вибрати (рисунку 1.19):

Representation → потрібний тип даних.

Найчастіше використовується строковий, цілочисельний та логічний типи даних.

Передача та прийом даних із периферії (COM, USB, Ethernet) здійснюється через строковий тип.

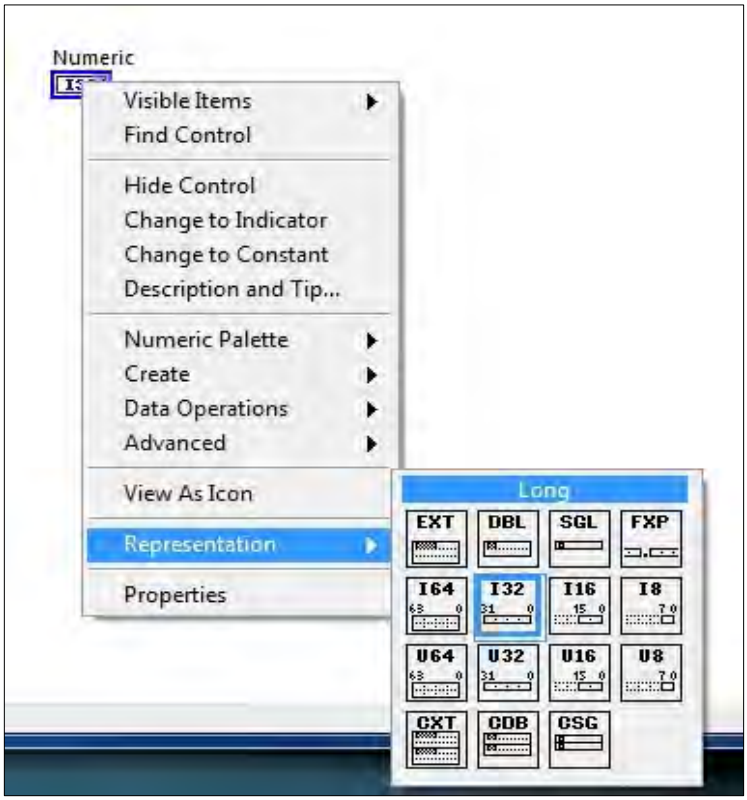


Рисунок 1.19 – Задання типу даних об’єкта

Всі кнопки та індикатори стану описуються типом **Boolean**.

З цілих часто використовується U8 (приймає значення від 0..255). Масив з елементів цього типу можна перетворити на рядок. Також часто використовується I32 як індекс всіх циклів.

1.2.3 Масиви LabVIEW

Для створення масиву існує кілька методів [3].

1 Шляхом об’єднання за допомогою вузла **Build Array**:

Права кнопка миші (ПК) → Programming → Array → Build Array.

Цей елемент може розтягуватись мишкою за верхній або нижній край з додаванням потрібної кількості вхідних елементів. Потім на вхід цього вузла потрібно подати елементи, які додаються до масиву. Якщо залишити входи порожніми, виникає помилка – зруйнована стрілка запуску (рисунок 1.20).

Теж відбудеться при додаванні різнотипних елементів (при додаванні числа та рядка, наприклад). Якщо об'єднуються подібні типи даних і можливе їх перетворення на загальний тип — це відбуватиметься автоматично до ширшому виду. Це перетворення буде показано червоною кулькою (рисунок 1.20).

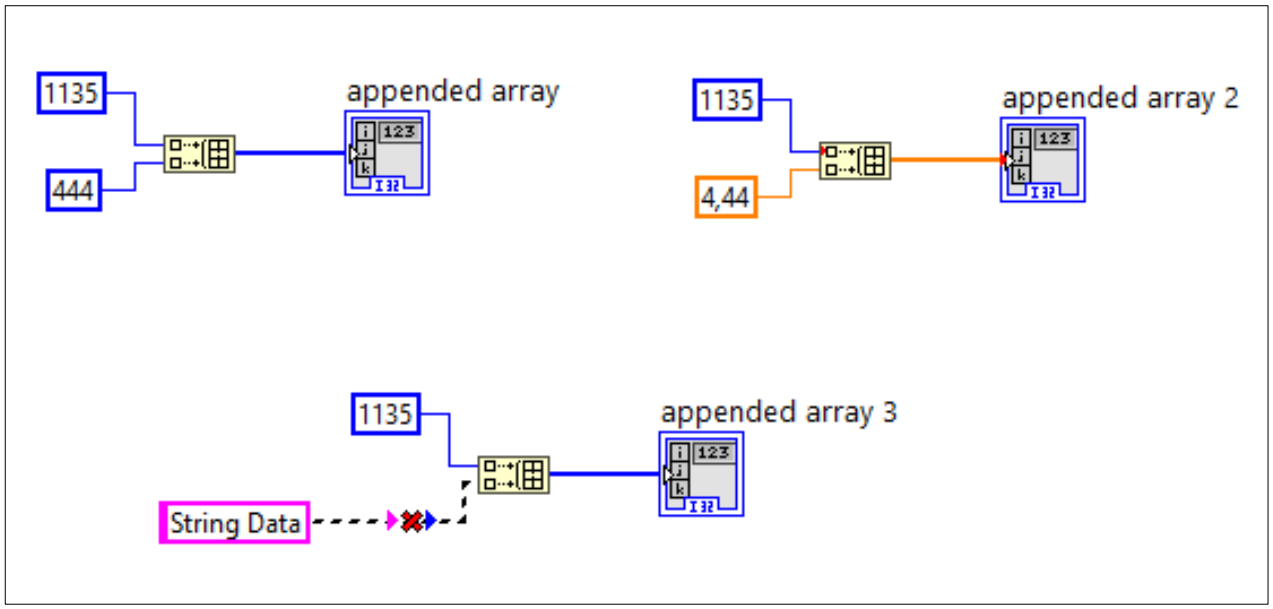


Рисунок 1.20 – Задання масиву

2 Якщо потрібно створити масив з N елементів з одним однаковим значенням, тоді можна використовувати **Initialize Array**:

ПК → Programming → Array → Initialize Array.

Увімкніть контекстну довідку (**Context Help, CTRL+H**). Розмістить вузол **Initialize Array** на **БД**. Наведіть на нього мишку. У довідці зазначено, що за

замовчуванням цей вузол має 3 входи та значок збільшення числа входів за допомогою миші: до входу **Element** потрібно під'єднати елемент, масив з якого ви хочете створити.

Наприклад задаємо число 10,2 типу **Double** (рисунок 1.21). Для цього клацніть **ПК** на вході блоку:

ПК → Create → Constant.

За замовчанням створюється константа типу Double. Якщо потрібен інший тип, то він обирається таким чином:

ПК → Representation → потрібний тип.

Вводимо потрібне нам значення (у прикладі 10,2). Потім розмірність та кількість елементів (**Dimension Size**). Розмірність визначається протягуванням вниз нижньої границі.

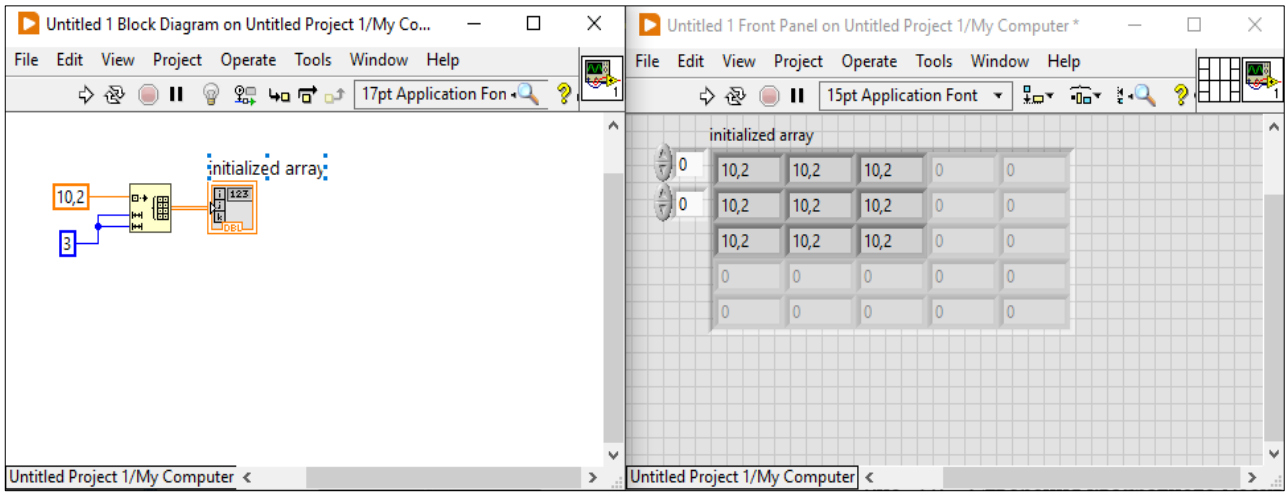


Рисунок 1.21 – Створення квадратного масиву 3x3

Так можна створювати масиви 2D, 3D тощо.

Зробимо квадратний масив 3 на 3 елементи. Для цього протягнемо на одну позицію нижню границю та отримаємо ще один вхід. Далі виконуємо:

ПК → Create → Constant,

вводимо значення «3» та з'єднуємо з другим входом (рисунок 1.21).

На виході блоку клацаємо мишею:

ПК → Create → Indicator.

Переходимо на **ФП** і за нижній край розтягуємо наш індикатор масиву. Запускаємо програму та отримуємо масив заповнених значень 10,2.

3 Часто потрібен елемент керування у вигляді масиву. Для його створення на **ФП** виконуємо (рисунок 1.22):

ПК→ Modern→ Array, Matrix & Cluster→ Array,

та розміщуємо порожній контейнер масиву на формі **ФП**. Потім додаємо потрібний елемент:

ПК → Modern→ Numeric → Control,

і розміщуємо його всередині контейнера (рисунок 1.22).

Контейнер можна розтягнути за нижній кут. Щоб додати розмірність (зробити 2D масив), необхідно клікнути на блоці і виконати:

ПК → Add Demension.

Далі можна заповнити вручну значення. Аналогічно можна створити масив констант і індикатор (рисунок 1.22).

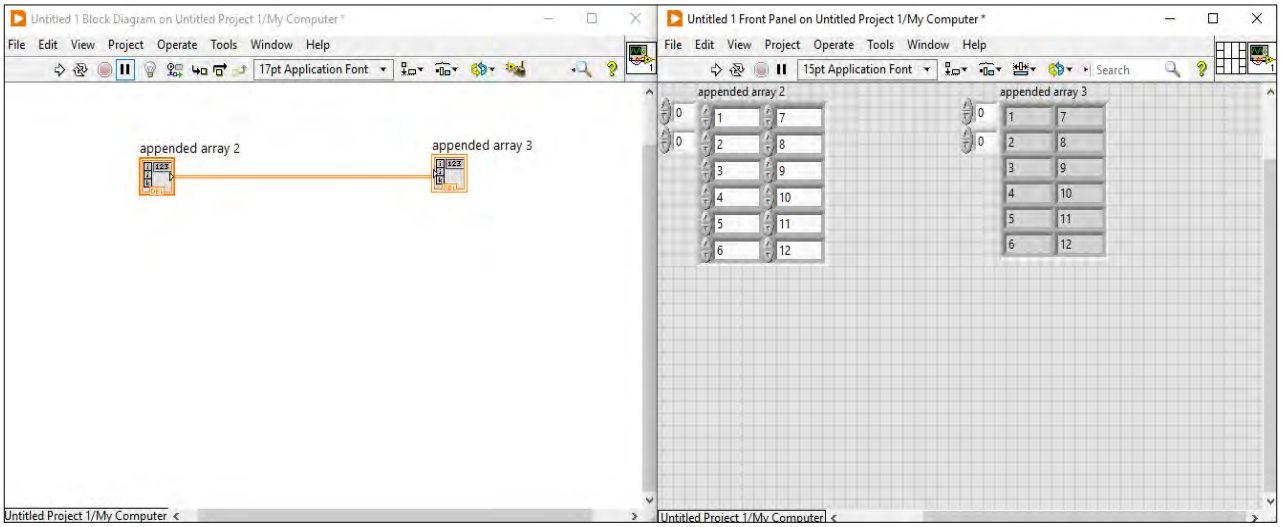


Рисунок 1.22 – Створення масиву через елемент керування

4 Найчастіший спосіб для отримання масиву - це виведення даних за допомогою циклів (рисунок 1.23). Цикли будуть розглянуті нижче.

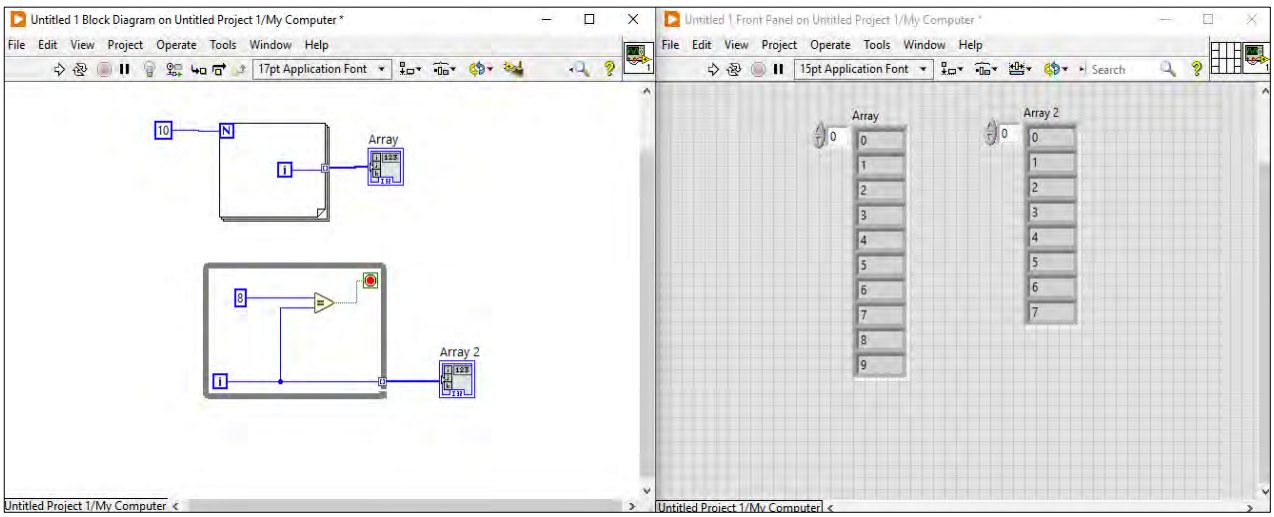


Рисунок 1.23 – Створення масиву за допомогою циклів

Якщо будь-який об'єкт розмістити всередині циклу та вивести дані з цього циклу з індексацією, то отримаємо масив.

Індикація для циклу **For Loop** включається автоматично, а для циклу **While Loop** включається за допомогою натискання ПК на терміналі виведення даних із циклу і потім командою **Enable Indexing**. Якщо її вимкнути, тоді на виході буде значення останньої ітерації циклу (рисунок 1.23).

1.2.4 Контрольні питання

1.2.4.1 Назвіть чисельні типи даних в LabVIEW.

1.2.4.2 Які особливості даних типу масиви (**Array**)?

1.2.4.3 Які особливості даних типу кластери (**Cluster**)?

1.2.4.4 Які особливості сигнального типу даних?

1.2.4.5 Які особливості динамічного типу даних?

2 ПРОЄКТУВАННЯ ЗА ДОПОМОГОЮ СТРУКТУР LabVIEW

2.1 Структури LabVIEW

Мова LabVIEW включає основні конструкції керування - структури, що мають аналоги в «традиційних» мовах, зокрема цикли та розгалуження, які найчастіше реалізуються за допомогою наступних інструментів:

- **While Loop** – цикли;

- **For Loop** – цикли з перевіркою завершення та без;

- **Case Structure** – розгалуження.

Приклади циклів **While Loop** і **For Loop**, а також розгалуження **Case Structure** (if / then / else – структура) показані на рисунку 2.1.

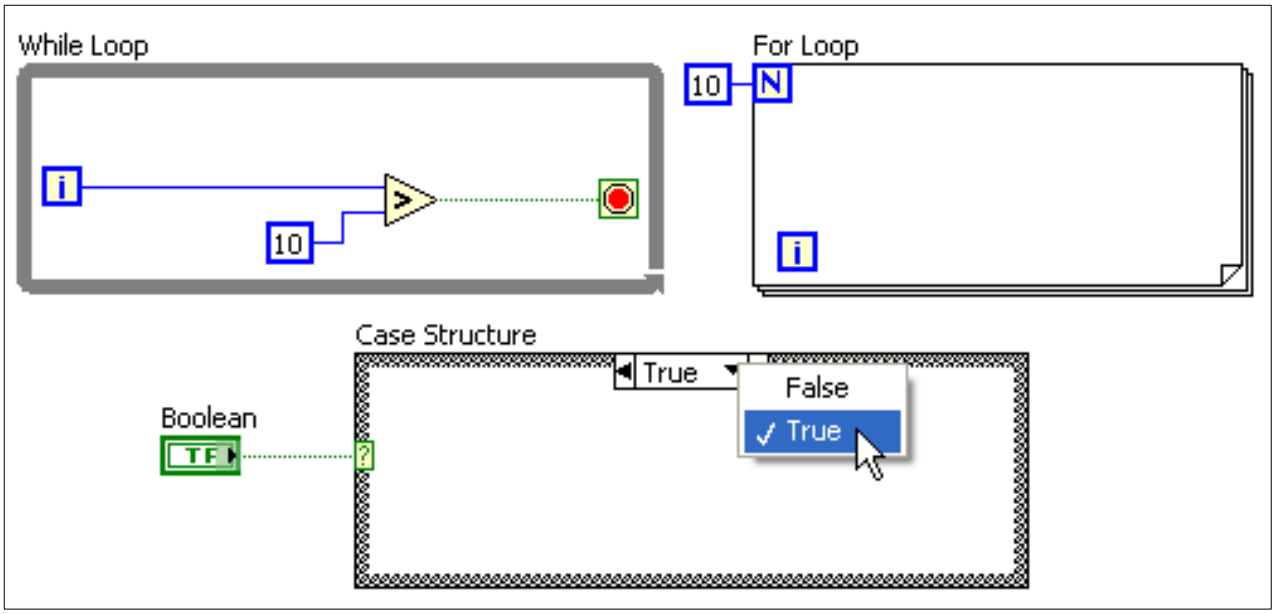


Рисунок 2.1 – Цикли While/For та розгалуження Case Structure

Як згадувалося, всі елементи виконуватимуться паралельно. В останніх версіях можна навіть явно вказати на якому з процесорів повинен виконуватися той чи інший while-цикл.

Структури є графічним представленням операторів циклу та операторів **Case** (Варіанту), що використовуються у текстових мовах програмування.

Структури на блок-діаграмі використовуються для виконання повторюваних операцій над потоком даних, операцій у визначеному порядку та накладання умов виконання операцій.

Середовище LabVIEW містить такі основні структури:

- цикл **While Loop** (за умовою);

- цикл **For Loop** (з фіксованою кількістю ітерацій);

- структура **Case** (Варіант);

- структура **Sequence** (Послідовність);

- структура **Event** (Подія);

- вузол **Formula Node** (вузол Формули);

- вузол **MathScript Nod**e (Математичний вузол) та ін.

За допомогою різних структур визначається алгоритм роботи програми. Тепер розглянемо лише основні структури, зокрема цикли **For Loop** і **While Loop**, структуру розгалуження **Case, Sequence** (Flat і Stacked), **Diagram Disable** і трохи торкнемося **Formula Node**.

2.1.1 Цикл For Loop

Цикл **For Loop** (з фіксованим числом ітерацій) виконує повторювані операції над потоком даних певну кількість разів.

На рисунку 2.2 наведено представлення роботи циклу **For Loop**:

1. Цикл **For Loop** (далі скорочено **For**) у середовищі LabVIEW.

2. Еквівалентна блок-схему роботи циклу **For**.

3. Приклад текстового аналога коду роботи циклу **For** у звичайній мові програмування.

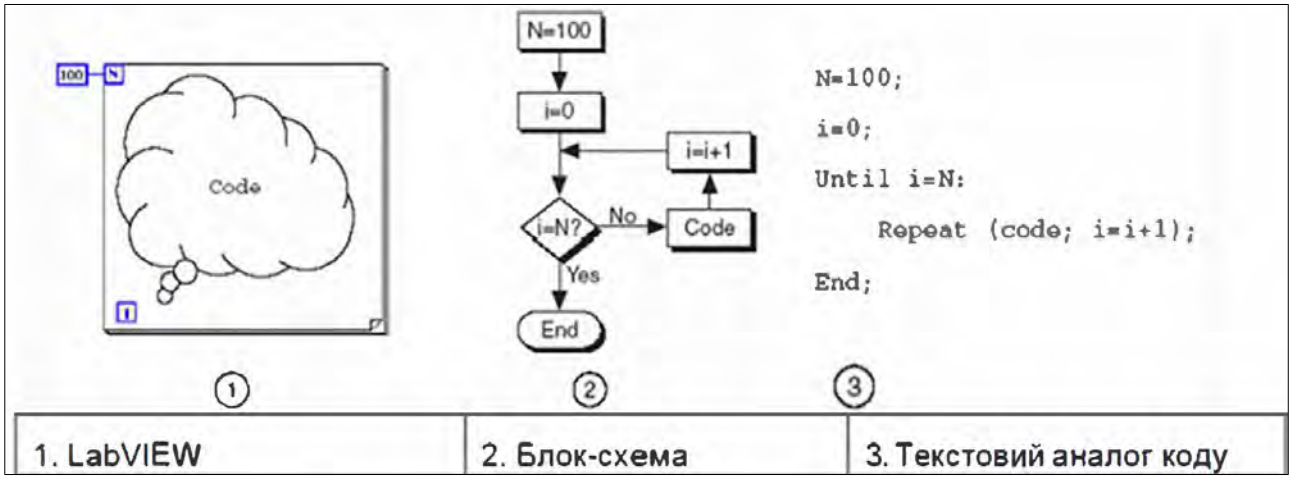


Рисунок 2.2 – Цикл **For Loop** і представлення його роботи

Цикл **For**, розташований в палітрі (на панелі) функцій у розділі:

Functions → Programming → Structures.

Після того, як цикл вибраний на панелі **Functions** (Функцій), слід за допомогою курсору виділити частину блок-діаграми, яку необхідно помістити в цикл [3]. Після відпускання кнопки миші виділена область блок-діаграми поміщається в тіло циклу. Додавання об'єктів блок-діаграми до тіла циклу здійснюється переміщенням або перетягуванням об'єктів за допомогою миші.

Значення, присвоєне терміналу максимальної кількості ітерацій **N** циклу, показаного на рисунку 2.2 зліва зверху, визначає максимальну кількість повторень операцій над потоком даних і для наведеного прикладу **N** = 100.

Термінал лічильника ітерацій "**i**", показаний зліва внизу, містить значення кількості виконаних ітерацій. Початкове значення лічильника ітерацій завжди дорівнює 0.

Цикл **For** завершує роботу, виконавши задану максимальну кількість ітерацій **N**.

У циклі **For** може використовуватися функція очікування:

Wait Until Next ms Multiple -  .

Ця функція використовується для синхронізації дій. Вона забезпечує інтервал між ітераціями, що дорівнює інтервалу часу, необхідному для того, щоб мілісекундний внутрішній таймер комп'ютера досяг значення, кратного введеному користувачем. Існує ймовірність, що перший період циклу буде коротким.

Використовується також інша функція очікування:

Wait (ms) -  ,

яка додає заданий час очікування до часу виконання програми. Це може спричинити труднощі, якщо час виконання програми є змінним.

Дані можуть надходити до циклу **For** (або виходити з нього) через термінали вхідних/вихідних даних циклу. Термінали вхідних/вихідних даних циклу передають дані зі структур та до структури.

Термінали вхідних/вихідних даних циклу відображаються у вигляді суцільних прямокутників на межі області циклу **For**. Прямокутник приймає

колір типу даних, що передаються терміналом. Дані виходять із циклу після його завершення. Якщо дані надходять у цикл **For** через термінал вхідних/вихідних даних циклу, виконання циклу починається при надходженні даних до терміналу.

Розглянемо приклад застосування циклу **For (**рисунок 2.3).

Розмістить на **БД** структуру **For Loop**:

ПК → Programming → Structures → For Loop,

і розтягніть її на невелику область. Тепер усі об'єкти, що будуть розміщені всередині цієї структури, будуть виконуватись циклічно ту кількість разів, яка буде задана.

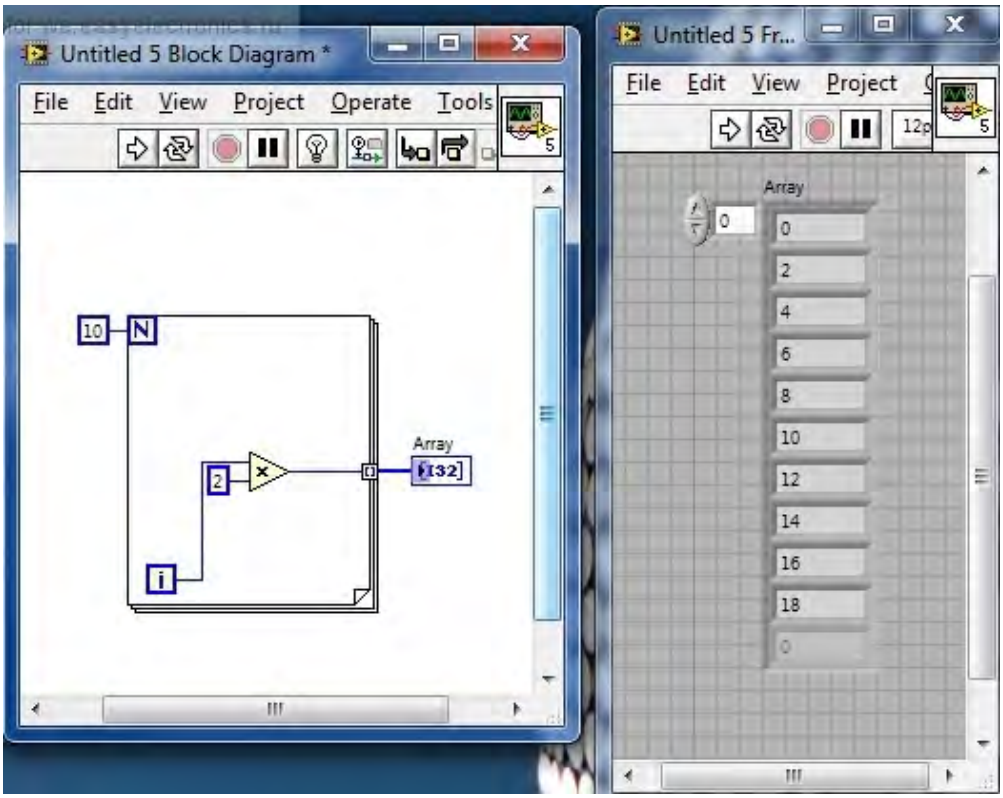


Рисунок 2.3 – Приклад застосування структури **For Loop**

Щоб задати скільки разів потрібно повторити цикл, створіть константу звичним чином біля символу **N** і встановіть число ітерацій, наприклад, 10, як показано на рисунку 2.3. Усередині циклу є лічильник ітерацій **i**, за яким можна дізнатися скільки разів вже виконався цикл.

Додайте всередину циклу вузол множення **Multiply** і до його другого входу підключіть лічильник ітерацій **i**. Потім створіть константу «2» на першому вході **Multiply**. Така послідовність потрібна для того, щоб створювана константа була типом **I32**, а не **double**, що встановлюється за замовчуванням для числових констант.

Підключіть вихід блоку множення до границі циклу **For Loop** і потім за допомогою правої кнопки миші клацніть на вихідному тунелі, що утворився, та виконайте:

ПК → Create→Indicator.

Цим ми створили одновимірний масив. Тепер можна запустити на виконання програму (**CTL+R**) і подивитись на **ФП** значення масиву (рисунок 2.3). Щоб швидко знайти індикатор на фронт-панелі можна двічі клікнути по терміналу індикатора.

2.1.2 Цикл While Loop

Цикл **While Loop** (за умовою) працює доти, доки не виконається логічна умова виходу з циклу. Цикл **While Loop** аналогічний циклам **Do** та **Repeat Until**, які використовуються в текстових мовах програмування.

На рисунку 2.4 наведено представлення роботи циклу **While Loop**:

1. Цикл **While Loop** (далі скорочено **While**) у середовищі LabVIEW.

2. Еквівалентна блок-схему роботи циклу **While**.

3. Приклад текстового аналога коду роботи циклу **While** у звичайній мові програмування.

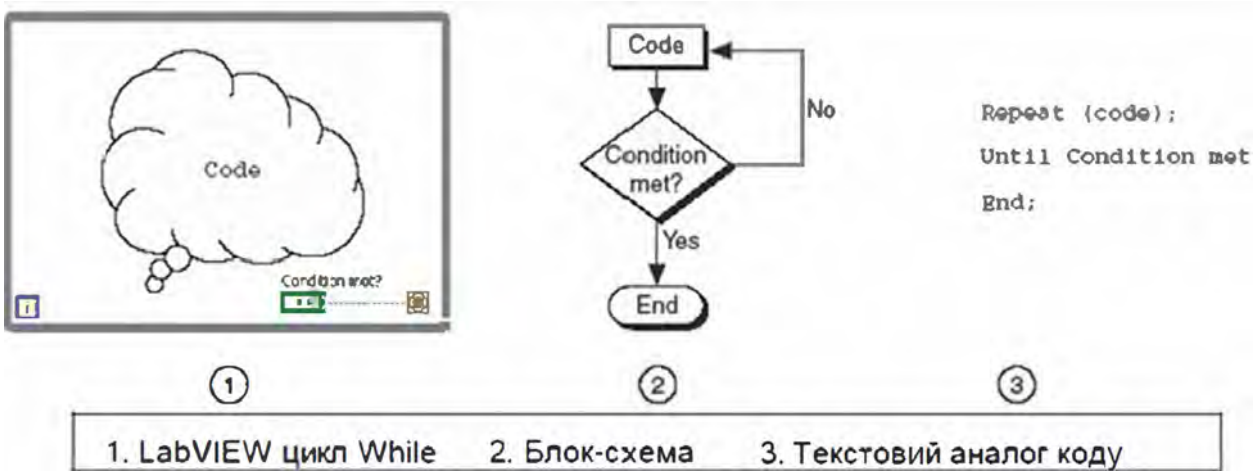


Рисунок 2.4 – Цикл **While Loop** та представлення його роботи

Цикл **While** знаходиться на палітрі:

Functions → Programming → Structures.

Після того, як цикл вибраний на панелі **Functions** (Функцій), слід за допомогою курсору виділити частину блок-діаграми, яку необхідно помістити в цикл [3]. Після відпускання кнопки миші виділена область блок-діаграми поміщається в тіло циклу. Додавання об'єктів блок-діаграми до тіла циклу здійснюється переміщенням або перетягуванням об'єктів.

Блок-діаграма циклу **While** виконується доти, доки виконається умова виходу з циклу **Loop Condition**. За замовчуванням термінал умови виходу має такий вигляд –  . Це означає, що цикл виконуватиметься до надходження терміналу умови виходу значення **TRUE**. У цьому випадку термінал умови виходу називається терміналом **Stop If True** (Зупинка, якщо Істина).

Термінал лічильника ітерацій «**i**» ( ) містить значення кількості виконаних ітерацій. Початкове значення терміналу завжди дорівнює нулю.

Дані можуть надходити у цикл **While** (або виходити з нього) через термінали вхідних/вихідних даних циклу. Термінали вхідних/вихідних даних циклу передають дані зі структур та до структури. Термінали вхідних/вихідних даних циклу відображаються у вигляді суцільних прямокутників на границі області циклу **While**. Прямокутник приймає колір типу даних, що передаються терміналом.

Дані виходять із циклу після його завершення. Якщо дані надходять у цикл **While** через термінал вхідних/вихідних даних циклу, виконання циклу починається при надходженні даних до терміналу.

Для застосування циклу розмістить на **БД** структуру **While Loop** за допомогою виконання мишею:

ПК → Programming → Structures → While Loop

і далі і розтягніть її на невелику область (рисунок 2.5).

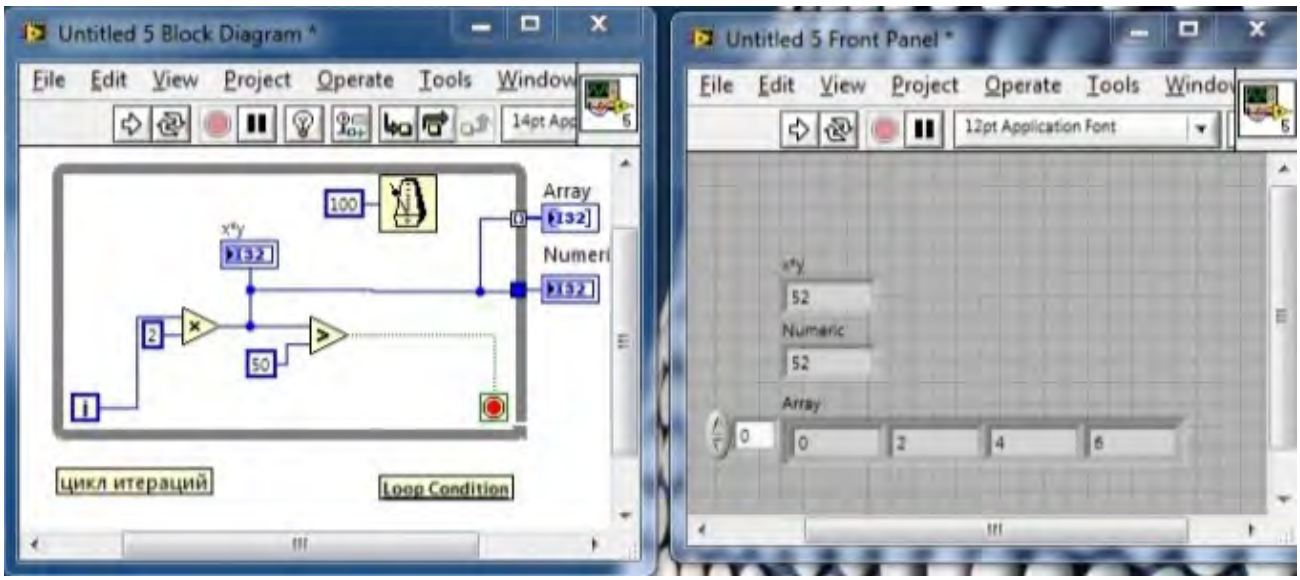


Рисунок 2.5 – Приклад застосування структури **While Loop**

Це цикл за умовою, який виконується до того часу, поки на вхід **Loop Condition** не поступає потрібне значення. В іншому випадку він аналогічний структурі **For Loop** і в нього також є цикл ітерацій «**i**».

Далі до циклу можна додати вузол множення та константу, і потім з'єднати, як показано на рисунку 2.5. Крім того додайте вузол порівняння **Greater** шляхом виконання:

ПК → Programming → Comparison → Greater

і створіть на його вході константу. Вихід вузла порівняння приєднайте до **Loop Condition**. Цикл буде виконуватися до тих пір, поки результат помножений на два не буде більше 50 (тобто цикл виконається 26 разів). До виходу множення додайте індикатор:

ПК → Вихід→ Create→ Indicator.

Щоб при виконанні програми бачити як змінюється цей індикатор, додайте затримку, бо інакше цикл дуже швидко виконається і ви побачите тільки кінцевий результат. Для цього виконайте:

ПК → Programming → Timing → Wait Unitil Next ms.

Створіть біля входу константу, що дорівнює 100 мс.

Потім приєднайте вихід вузла множення до границі структури. На відміну від структури **For Loop**, в циклі **While** за замовчуванням зроблений не індексований тунель. Для відображення даних створіть на його виході індикатор. Потім протягніть ще одну лінію зв’язку з виходу вузла множення до границі на тунелі для створення масиву:

ПК на тунелі → Enable Indexing.

Після цього можна запустити виконання програми (рисунок 2.5).

2.2. Застосування структур Case, Sequence (Flat і Stacked) і Diagram Disable

2.2.1 Структура Case

Структура **Case** (**Case Structure**) призначена для розгалуження алгоритму програми. Вона еквівалентна операторам **if…else** та **switch** мови **C**.

Структура **Case** (рисунок 2.6) має дві або більше варіантів піддіаграми.

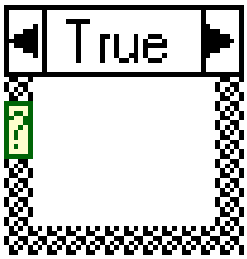
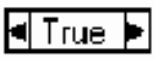


Рисунок 2.6 – Структура **Case**

На рисунку 2.6 представлена тільки одна піддіаграма варіанта, яка і буде працювати при виконанні цієї структури в такому випадку (на наведений момент часу). Вхідне значення терміналу селектора структури визначає, яка саме піддіаграма виконуватиметься на даний (наведений) момент часу.

Структура **Case** аналогічна операторам **case** чи логічним операторам (**if...then...else**) у текстових мовах програмування.

Селектор 

структури **Case**, який розташований зверху графічного зображення структури, складається з вказівника значення варіанта в центрі та стрілок прокручування вправо-вліво. Ці стрілки використовуються для перегляду можливих варіантів структури - піддіаграм.

Значення, що подається на термінал селектора варіанта  , визначає, яка піддіаграма структури, або варіант, буде виконуватися. Для селектора варіанта допустимо використовувати такі типи даних: цілий, логічний, рядковий, а також тип перерахування як значення, що подається на термінал варіанта. Не можна подавати числа з плаваючою точкою на термінал селектора варіанта, оскільки можливі помилки округлення та виникнення ситуації невизначеності.

Термінал селектора варіанта (далі – термінал варіанта) може розташовуватися будь-де на лівій границі структури **Case**. Якщо термінал варіанта логічного типу, то структура складається із двох логічних варіантів: **True** та **False**. Якщо термінал варіанта має один з наступних типів: цілочисельний, рядковий або перерахування, - то кількість варіантів може досягати (2^{31}-1) варіантів.

Для використання структури **Case** слід зазначити варіант за замовчуванням [3]. Варіант за замовчуванням або стандартна піддіаграма, виконується, якщо значення терміналу варіанта виходить за межі діапазону або не існує варіантів для можливих значень терміналу варіанта.

Клацанням правою кнопкою миші на межі структури **Case** можна додавати, дублювати, переміщати та видаляти варіанти (піддіаграми), а також відзначати варіант за замовчуванням.

Структура **Case** допускає використання вхідних та вихідних терміналів даних. Термінали вхідних даних доступні у всіх піддіаграмах, але їх використання піддіаграмою структури не є обов'язковим. Створення вихідного терміналу на одній піддіаграмі призводить до його появи на інших піддіаграмах в тому ж самому місці границі структури. Якщо хоча б у одній піддіаграмі вихідний термінал не визначено, то поле цього терміналу забарвлюється в білий колір, що говорить про помилку створення структури. Необхідно визначати значення вихідних терміналів у всіх варіантах (піддіаграми). Крім того, вихідні термінали повинні мати значення сумісних типів.

Для визначення значення вихідного терміналу слід клацанням правою кнопкою миші по терміналу викликати контекстне меню і вибрати пункти:

Create → Constant або Create → Control.

Розглянемо застосування структури **Case**. Для цього розмістить на **БД** структуру **Case Structure**:

ПК → Programming → Structures → Case Structure

і розтягніть її на невелику область (рисунок 2.7).

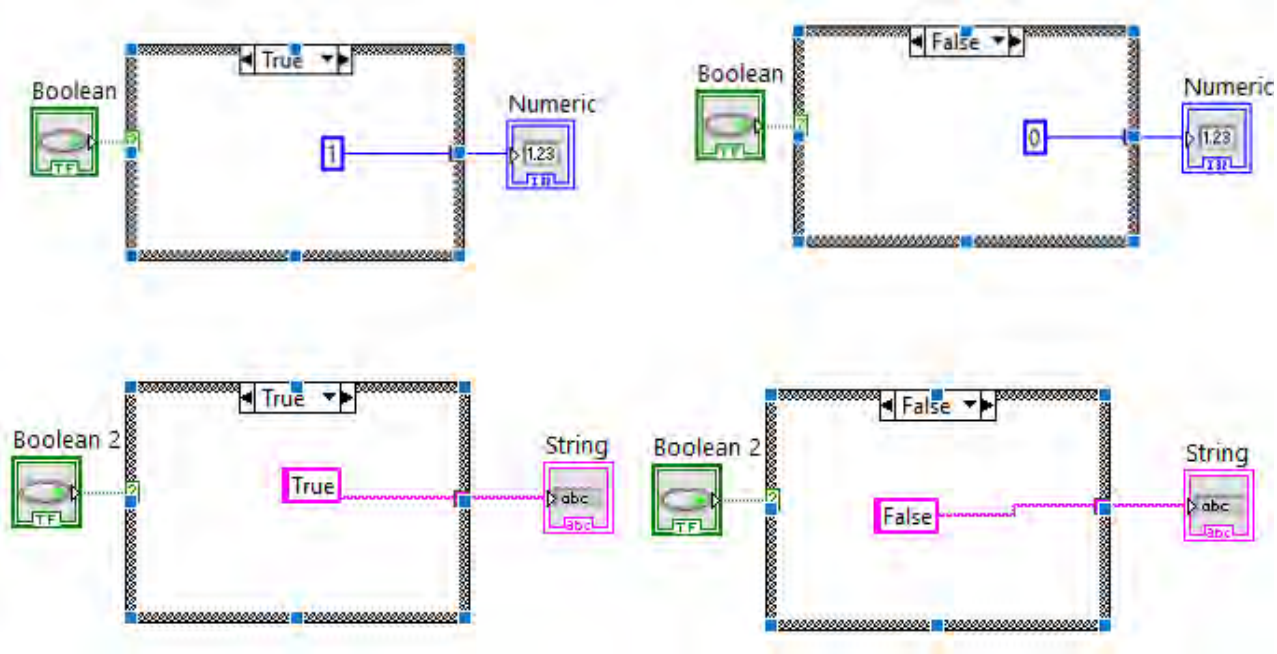


Рисунок 2.7 – Приклад застосування **Case Structure** з терміналом варіанта **Boolean**

На вході **Case Selector** (знак питання) створіть елемент **Control**, наприклад булевого типу - **Boolean** (рисунок 2.7). Потім створіть константу всередині структури на вкладці **True** зі значенням, наприклад «1», і виведіть лінію до границі, створивши вихідний термінал. Після цього створіть константу на вкладі **False** зі значенням, наприклад «0», і виведіть лінію до того ж вихідного терміналу. Створіть індикатор **Numeric** на виході структури.

Запустіть виконання програми неперервно (**Run continuously**). Натискаючи на створену кнопку **Boolean**, можна бачити, як залежно від її значення виконується та чи інша вкладка структури.

Замість константи всередині структури **Case** може бути створена логічна змінна **True-False** (рисунок 2.7).

Якщо подати на вхід **Case Selector** рядок, чи числову змінну, то можна зробити кілька вкладок на різні значення вхідної змінної (рисунок 2.8). У такий спосіб можна утворити розгалуження більше, ніж за двома вхідними значеннями.

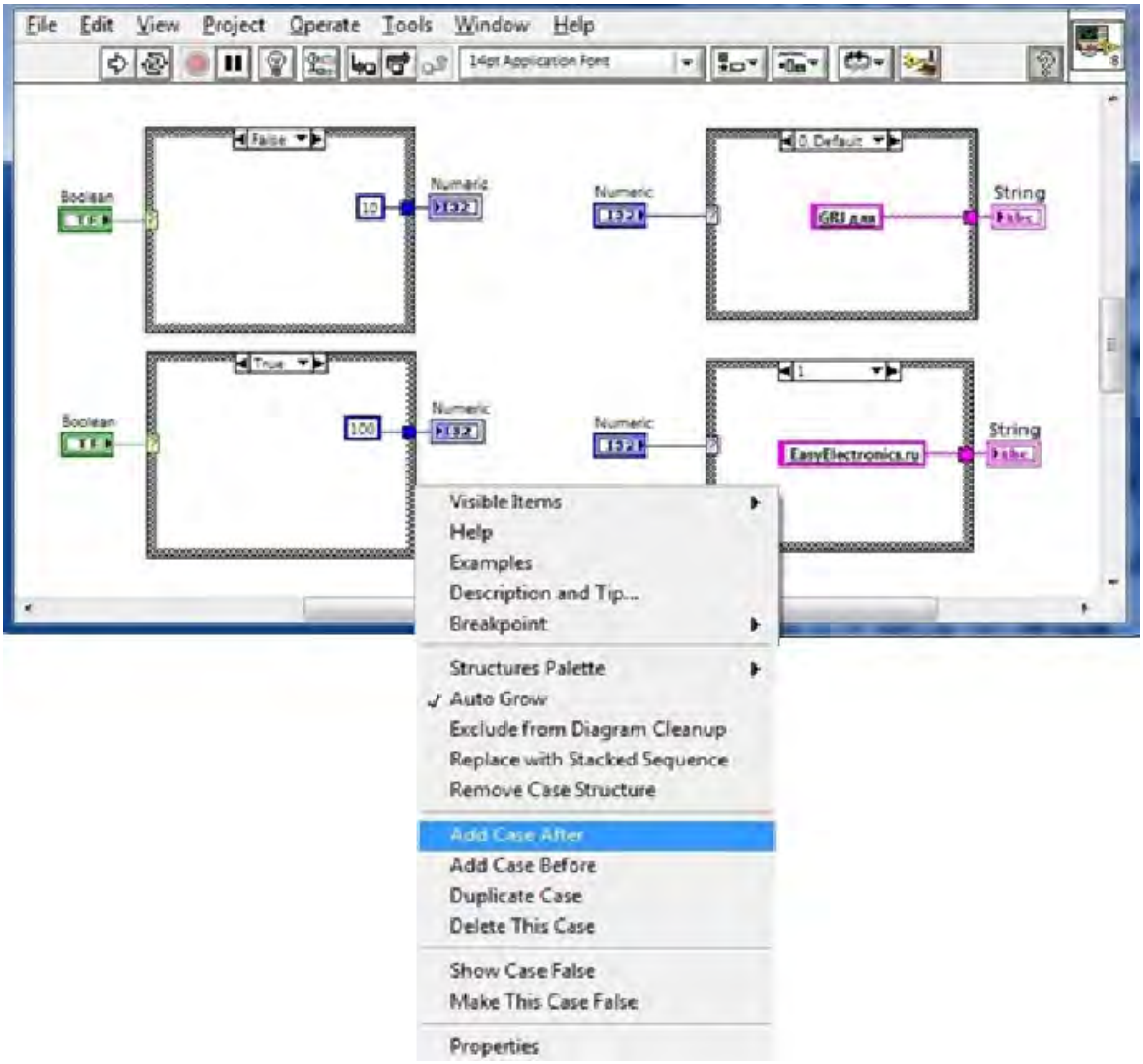


Рисунок 2.8 – Приклад застосування **Case Structure** з терміналом варіанта у вигляді числової змінної

Щоб додати ще одну вкладку, необхідно виконати (рисунок 2.8):

ПК на структурі → Add Case After або Add Case before.

Якщо значення на вході термінала **Case Selector** не відповідає значенням вкладок, тоді виконається вкладка, позначена як **Default**.

2.2.2 Структури Sequence (Flat і Stacked)

Структури **Sequence** використовуються у випадках, коли необхідно явно вказати порядок виконання програми, що розміщується в цих структурах.

Для розміщення на **БД** структури **Stacked Sequence** або **Flat Sequence** необхідно виконати:

ПК → Programming → Structures → Flat Sequence (Stacked Sequence).

Ці структури практично не відрізняються (рисунок 2.9).

Коли потрібно кілька кадрів (до 6), тоді краще використовувати **Flat Sequence**, тому що відразу видно всю блок-діаграму, і вона не прихована за кадрами. При цьому передавати значення між кадрами простіше.

Коли вкладок більше або код усередині них дуже об'ємний і не поміщається на екрані, тоді краще використовувати **Stacked Sequence**.

Щоб додати новий кадр, потрібно клікнути правою кнопкою на відповідній структурі і вибрати (рисунок 2.9):

- **Add Frame After**, якщо потрібно створити кадр після структури;

- **Add Frame Before**, якщо потрібно створити кадр після структури.

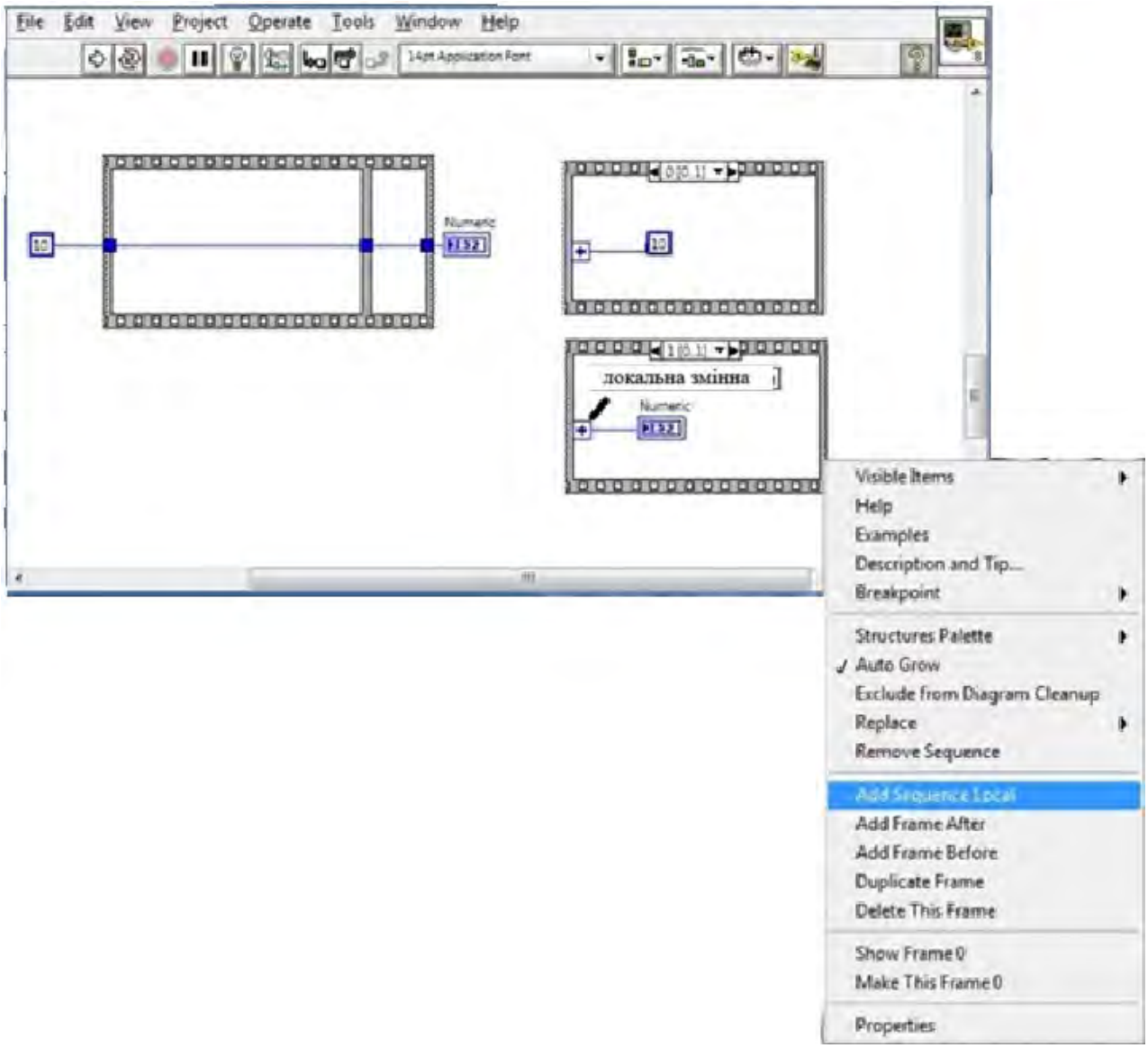


Рисунок 2.9 – Структури **Flat i Stacked Sequence**

Дані від кадру до кадру в **Stacked Sequence** передаються через локальні змінні послідовності. Для її додавання потрібно клікнути правою кнопкою на відповідній структурі і вибрати **Add Sequence Local** (рисунок 2.9), а потім її можна перенести у зручне місце та передати/зчитати з неї значення.

2.2.3 Структура Diagram Disable

Для коментування частини коду на етапі налагодження використовується структура **Diagram Disable** (рисунок 2.10).

Все, що розміщено всередині цієї структури, не обробляється компілятором LabVIEW.

Для її створення натисніть:

ПК → Programming → Structures → Diagram Disable Structure.

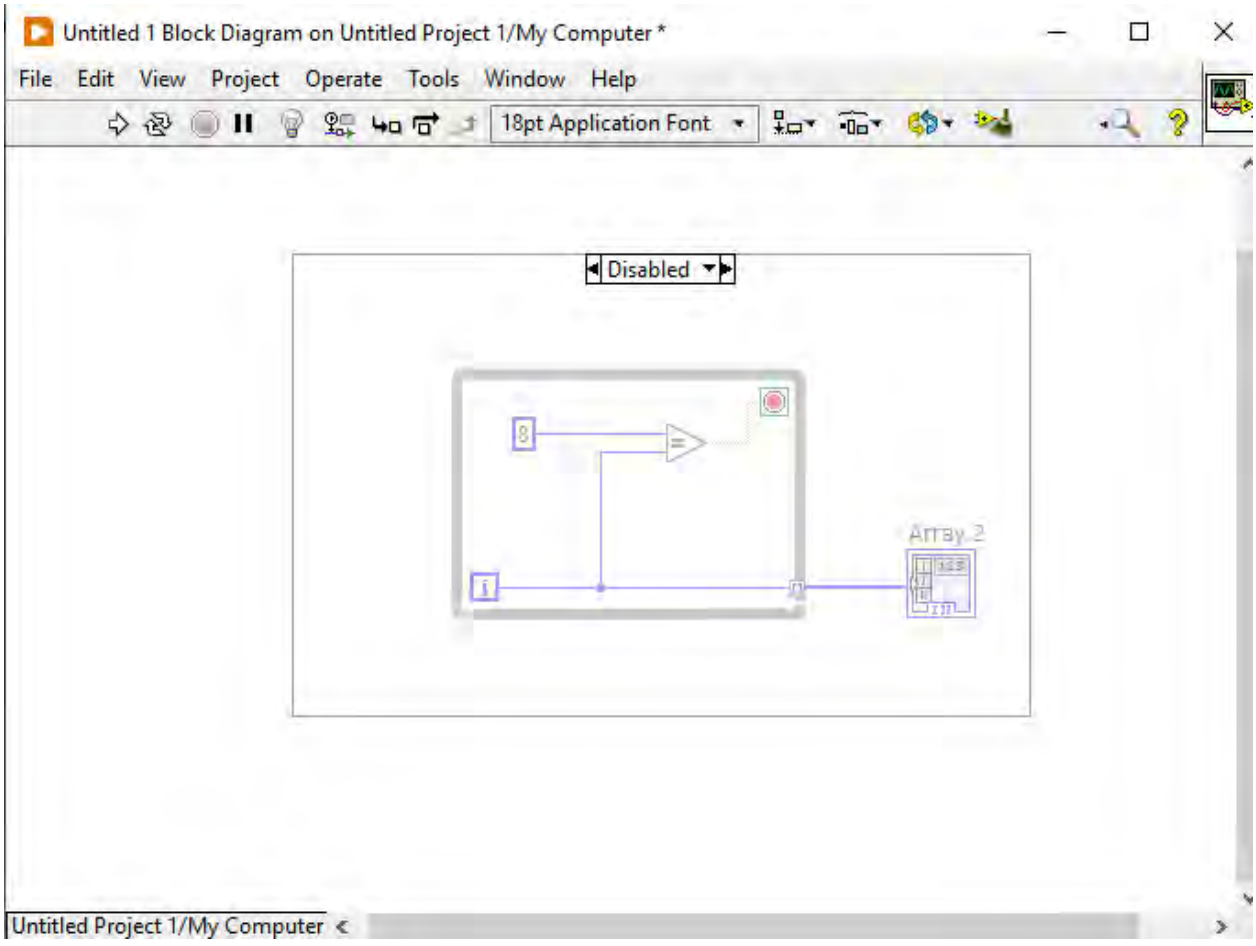


Рисунок 2.10 – Структура **Diagram Disable**

Ця структура має дві вкладки **Disabled** (вимкнено) та **Enabled** (включено). Увімкнення та вимкнення здійснюється через клік правою кнопкою миші.

2.3 Контрольні питання

2.3.1 Назвіть основні структури LabVIEW.

2.3.2 Які особливості використання структури **For Loop**?

2.3.3 Чим відрізняється структура **While Loop** від структури **For Loop**?

2.3.4 Які особливості використання структури **Case Structure**?

2.3.5 Які типи даних можуть використовуватися на терміналі селектора варіанта в **Case Structure**?

2.3.6 Яке призначення структури **Diagram Disable**?

3 ПРОЄКТУВАННЯ ЗАСОБІВ ГРАФІЧНОГО ВІДОБРАЖЕННЯ ДАНИХ В LabVIEW

Для графічного відображення даних у LabVIEW використовуються спеціальні засоби візуалізації:

- **Waveform Chart** – графік діаграм;

- **Waveform Graph** – графік осцилограм;

- **XY graph** – двохкоординатний графік осцилограм;

- **Intensity Graph** – графік інтенсивності.

3.1 Графічне відображення даних за допомогою Waveform Chart, Waveform Graph і XY Graph

3.1.1 Відображення даних у Waveform Chart

Графік діаграм (**Waveform Chart**) – це спеціальний елемент відображення даних у вигляді одного та більше графіків [3].

Графік діаграм розташований на панелі:

Controls → Modern → Graph.

На рисунку 3.1 показаний приклад **Waveform Chart** з двома графіками, що відображають: експериментальні дані (зелені квадрати) та їх поточне середнє значення (жовта лінія).

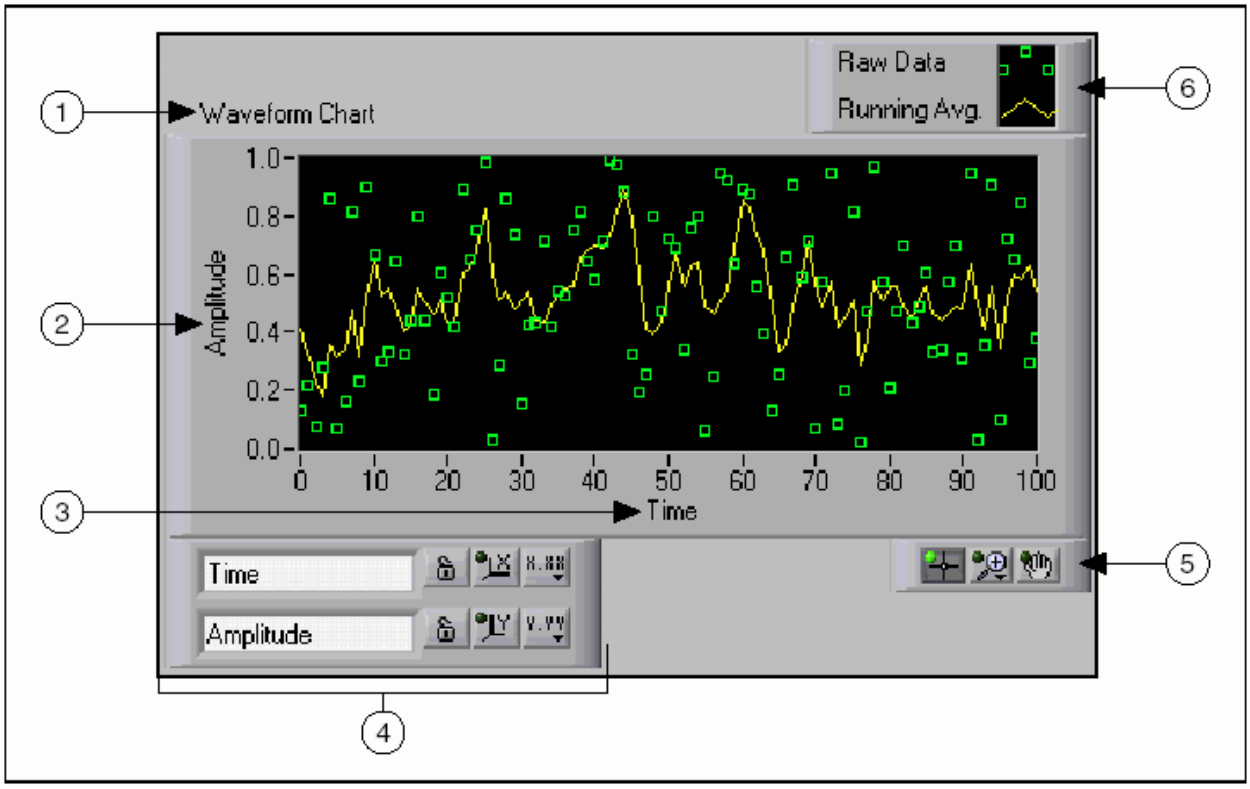


Рисунок 3.1 – Графік Діаграм **Waveform chart** та приклад його роботи

На рисунку 3.1 позначено:

1 - назва (**Label**);

2 - шкала Y (**Y-scale**);

3 - шкала X (**X-scale**):

4 - панель керування шкалами (**Scale legend**)

5 - палітра інструментів для роботи із графіком (**Graph palette**)

6 - панель керування графіком (**Plot legend**)

Графік Діаграм використовує три різні режими відображення даних (рисунок 3.2): **strip chart**, **scope chart** та **sweep chart**.

Стандартним (за замовчуванням) є режим **strip chart**.

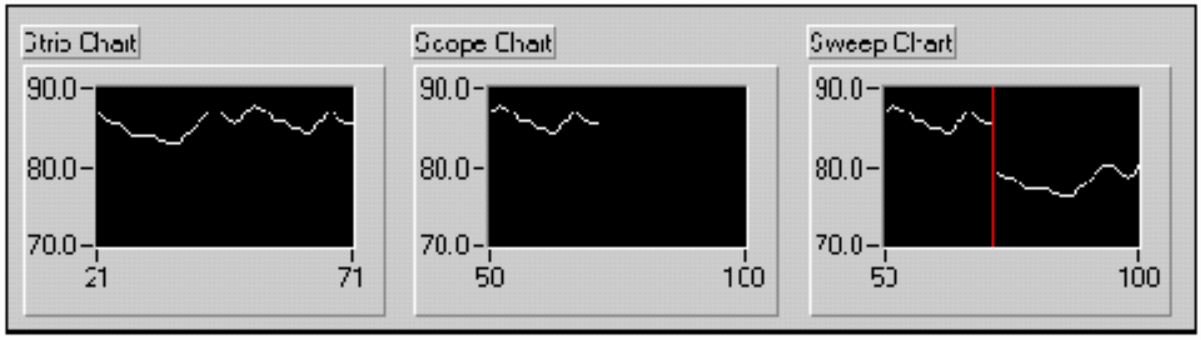


Рисунок 3.2 – Режими відображення даних у **Waveform Chart**

Завдання потрібного режиму здійснюється кліком правою кнопкою миші на діаграмі та вибором з контекстного меню пункту:

Advanced → Update Mode.

Режим **Strip Chart** представляє собою екран, що прокручується зліва направо, подібно до паперової стрічки.

Режими **Scope Chart** і **Sweep Chart** подібні до екрану осцилографа і відрізняються більшою швидкістю відображення даних у порівнянні зі **Strip Chart**.

У режимі **Scope Chart** після досягнення правої границі поле графіка очищується, і заповнення діаграми починається з лівої границі.

Режим **Sweep Chart**, на відміну режиму **Scope Chart**, не очищує поле графіка, а відокремлює нові дані від старих вертикальної лінією – маркером.

Для створення діаграм достатньо з'єднати поле виведення скалярної величини із терміналом даних **Waveform Chart**. У прикладі, наведеному на рисунку 3.3, тип даних на терміналі графіка діаграм у режимі **Strip Chart**, відповідає вхідному типу даних.

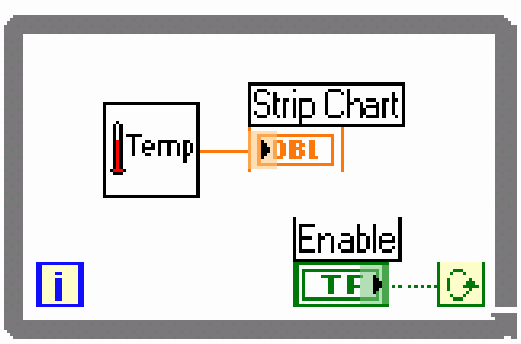


Рисунок 3.3 – Приклад використання графіка діаграм у режимі **Strip Chart**

У **Waveform Chart** може відображатися кілька графіків. Для об'єднання даних, що відображаються, використовується функція **Bundle** (згорток, пучок), розташована в палітрі:

Functions → Programming → Cluster & Variant.

Наприклад, блок-діаграма на рисунку 3.4 за допомогою функції **Bundle** поєднує вихідні дані трьох підпрограм **ВП**, що вимірюють температуру, для подальшого відображення їх даних на графіках у **Waveform Chart**.

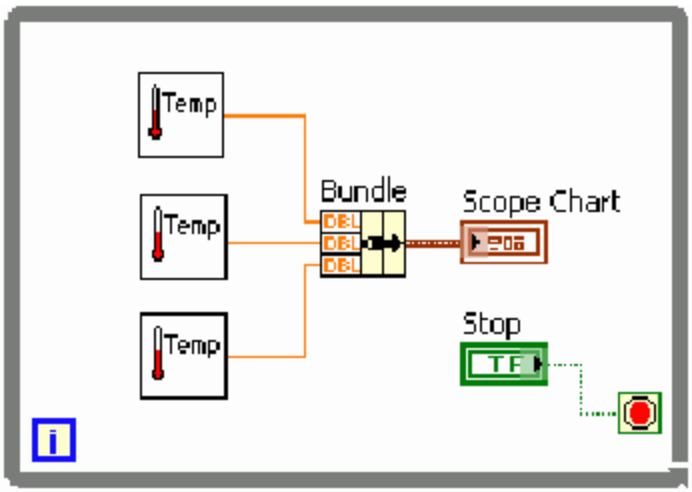


Рисунок 3.4 – Приклад об’єднання даних трьох **ВП** на одному графіку за допомогою функції **Bundle**

Термінал даних графіка діаграм має кластерний тип даних відповідно до поля виведення функції **Bundle**. Для збільшення кількості полів введення даних функції **Bundle** необхідно за допомогою інструмента ПЕРЕМІЩЕННЯ змінити термінал [3].

Розглянемо послідовність дій для налаштування графіка діаграм, показаного рисунку 3.5.

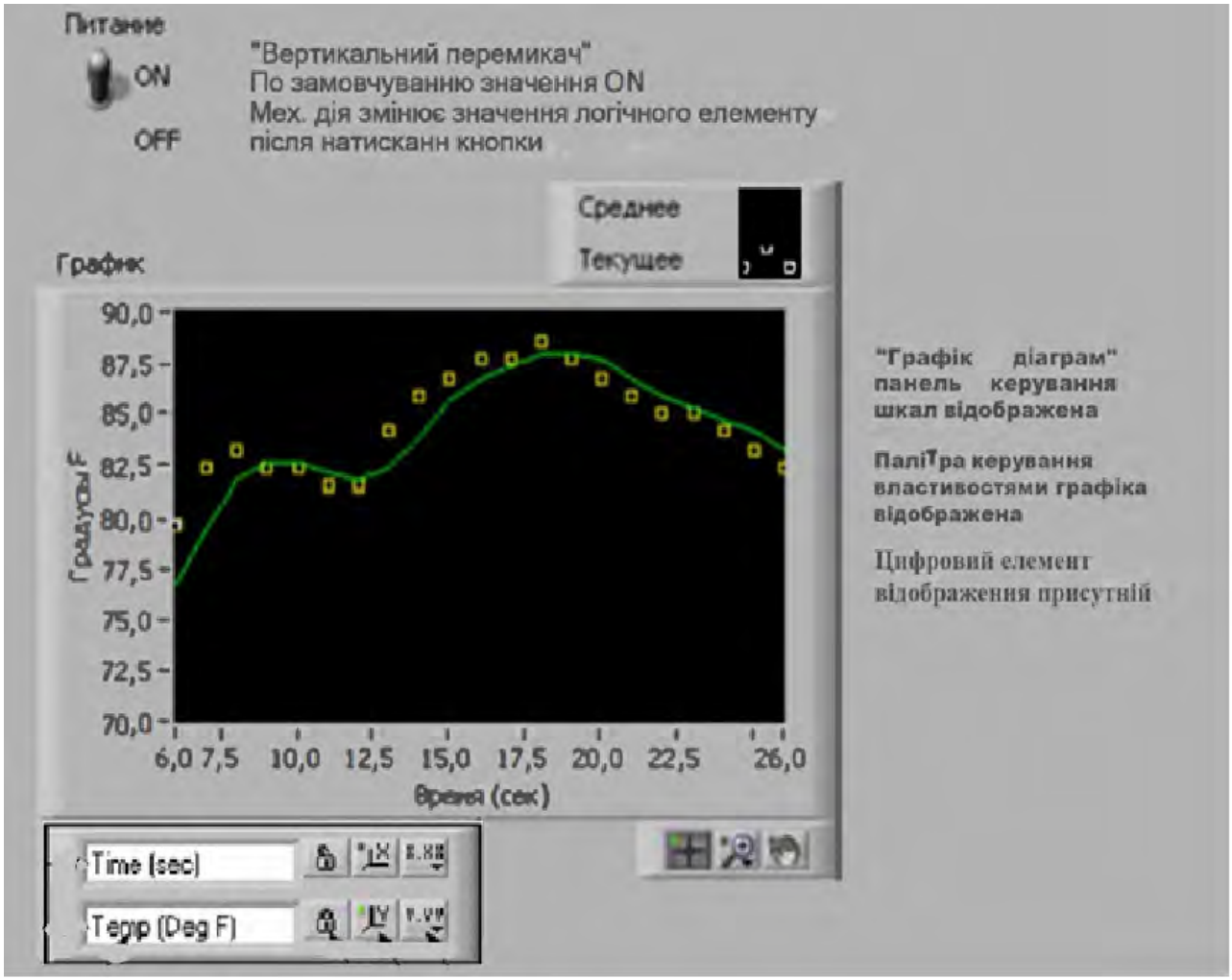


Рисунок 3.5 – Приклад налаштування графіка діаграм

На лицьову панель можна вивести панель керування графіком діаграм, панель керування масштабом шкали, палітру елементів керування діаграмою,

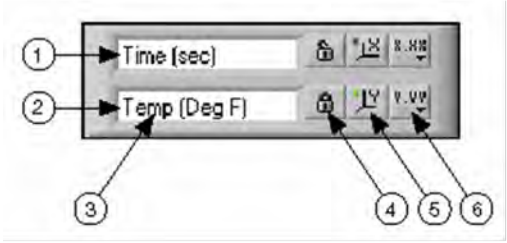
цифровий елемент відображення, смугу прокручування та вміст буфера. За замовчуванням, разом з елементом графік діаграм відображається панель керування властивостями графіка.

Налаштування осі Y здійснюється за допомогою інструмента ВВЕДЕННЯ ТЕКСТА шляхом зміни числових значень по осі Y.

Щоб відобразити панель **Scale Legend**, клікніть правою кнопкою миші на полі графіка діаграм і виберіть пункт контекстного меню:

Visible Items → Scale Legend,

як показано на рисунку 3.6.



1. Вісь X
2. Вісь Y
3. Мітки курсорів
4. Кнопка блокування (Scale Lock)
5. Кнопка автомасштабування (Autoscale)
6. Кнопка формату вісі (Scale Format)

Рисунок 3.6 – Панель **Scale Legend**

Панель керування масштабом шкали може бути переміщена до будь-якого місця лицьової панелі.

Для автоматичної зміни масштабу даних по осі Y необхідно натиснути кнопку **Autoscale** (забарвлюється в зелений колір), а кнопку **Scale Lock** вимкнути.

Для зміни формату, точності, режиму відображення, масштабу та виду розмірної сітки для кожної осі потрібно натиснути кнопку **Scale Format**.

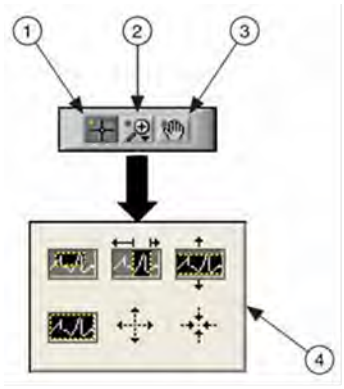
За допомогою панелі **Plot Legend** можна налаштувати вид діаграм, що відображаються на елементі графік діаграм:

- за допомогою інструмента ПЕРЕМІЩЕННЯ змінюється розмір **Plot Legend** для відображення двох графіків;

- за допомогою інструмента ВВЕДЕННЯ ТЕКСТА змінюється мітка (текст);

- за допомогою контекстного меню, що викликається кліком правою кнопкою миші по **Plot Legend**, редагується колір, фон, представлення ліній та точок.

Для відображення палітри **Graph Palette** елементів керування графіком клікніть правою кнопкою миші по елементу **Waveform Chart** і виберіть пункт контекстного меню **Visible Items Graph Palette**. Призначення елементів палітри **Graph Palette** для керування графіком показано на рисунку 3.7.



1. Інструмент переміщення (**Cursor Movement)**.
2. Кнопка масштабування (**Zoom).**
3. Інструмент панорамування курсору (**Panning tool**).
4. Панель керування маштабом шкали (**Zoom pull-down menu**).

Рисунок 3.7 – Панель **Graph Palette**

Палітра **Graph Palette** може бути перенесена до будь-якої частини лицьової панелі.

3.1.2 Відображення даних у Waveform Graph і XY Graph

Накопичені в масив дані ВП можуть бути відображені у вигляді осцилограм за допомогою інструментів **Waveform Graph** . На рисунку 3.8 показано елементи графіка осцилограм **Waveform Graph**.

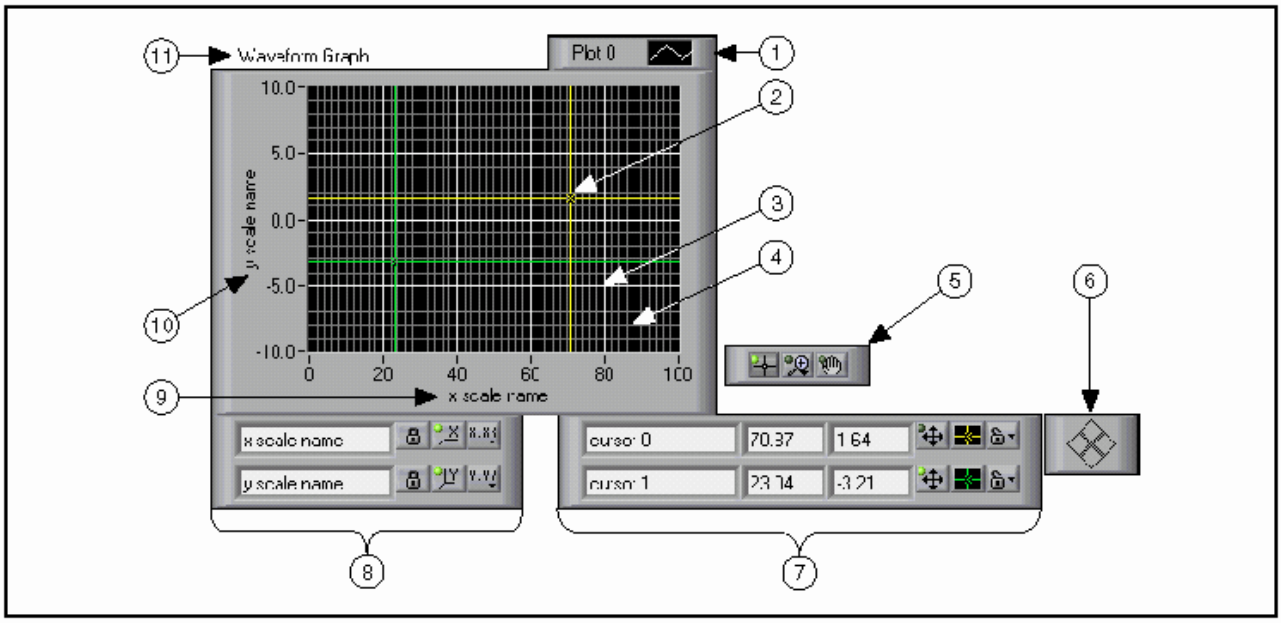


Рисунок 3.8 – Графік осцилограм **Waveform Graph**

Графік осцилограм **Waveform Graph** містить такі елементи (рисунок 3.8):

1 - панель керування властивостями осцилограм (**Plot legend**);

2 - курсор (**Cursor**);

3 - основна розмірна сітка (**Grid mark**);

4 - додаткова розмірна сітка (**Mini-grid mark**);

5 - палітра елементів керування графіком (**Graph palette**);

6 - панель переміщення курсору (**Cursor mover**);

7 - панель керування властивостями курсору (**Cursor legend**);

8 - панель керування шкалою (**Scale legend**);

9 - шкала X (**X-scale**);

10 - шкала Y (**Y-scale**).

11 - власну мітка графіка (**Label**).

Графік осцилограм (**Waveform Graph**) та двохкоординатний графік осцилограм (**XY Graph**) розташовані на панелі:

Controls → Modern → Graph.

Графік осцилограм **Waveform Graph** відображає лише однозначні функції, такі як

$$y = f(x),$$

з точками, рівномірно розподіленими по осі X.

Двохкоординатний графік осцилограм **XY Graph** відображає будь-який набір точок, чи це є рівномірно розподілена вибірка у часі чи ні [3].

Графік осцилограм та двокоординатний графік осцилограм автоматично підтримують режим відображення множини осцилограм Графік множини осцилограм використовується з метою економії простору на лицьовій панелі та для порівняння осцилограм даних між собою Для відображення множини осцилограм необхідно змінити розмір панелі **Plot legend**.

Одиночний графік осцилограм працює з одновимірними масивами і представляє дані масиву у вигляді точок на графіці з кроком збільшення по осі X

рівним **1** і початком у точці **x = 0**. Графіки також відображають кластери з встановленими початковим значенням **x, Δx** і масивом даних за шкалою Y.

Графік множини осцилограм працює з двовимірними масивами даних, де кожен рядок масиву є одиночна осцилограма даних і представляє дані масиву у вигляді точок на графіці, з кроком збільшення по осі X рівним **1** і початком у точці **x = 0**. Для представлення кожного стовпця двовимірного масиву даних у вигляді осцилограми на графіку необхідно з'єднати термінал даних масиву з вхідним терміналом даних графіка, потім клікнути правою кнопкою миші полем графіка і вибрати пункт контекстного меню **Transpose Array** (транспонування масиву).

Графіки множини осцилограм відображають кластери, що складаються з початкового значення **x, Δx** і двовимірного масиву даних за шкалою Y. Графік представляє дані за шкалою Y у вигляді точок із збільшенням **Δx** по осі X і початком у точці **x = 0**.

Графіки множини осцилограм відображають також і кластери з встановленим початковим значенням **x, Δx** і масивом даних, що містить кластери. Кожен кластер містить масив точок, що відображають дані за шкалою Y. Для створення масиву кластерів слід використовувати функцію **Bundle**, яка об'єднує масиви у кластери. Далі за допомогою функції **Build Array** створюється масив кластерів. Також можна використовувати функцію **Build Cluster Array**, яка створює масив кластерів з певними полями введення даних.

Одиночний двохкоординатний графік осцилограм працює з кластерами, що містять масиви **X** та **Y**. Двохкоординатний графік осцилограм також сприймає масиви точок, де кожна точка є кластером, що містить значення за шкалами X і Y.

Двохкоординатні графіки множини осцилограм працюють з масивами осцилограм, в яких осцилограма даних є кластером, що містить масиви значень **X** та **Y**. Двохкоординатні графіки множини осцилограм сприймають також

масиви множини осцилограм, де кожна осцилограма є масивом точок. Кожна точка – це група даних, що містить значення **x** і **y**.

3.2 Побудова графіків за допомогою Intensity Graph

Intensity Graph призначений для відображення двовимірних даних у вигляді графіків та таблиць інтенсивності (Intensity graphs and charts). Наприклад, **Intensity Graph** можна застосувати для представлення топографії місцевості, де амплітудою є висота над рівнем моря.

Як і у випадку з графіками діаграм та осцилограм, графік інтенсивності має постійний розмір дисплея, а дисплей таблиці інтенсивності має можливість прокручування. Графіки та таблиці інтенсивності приймають на вхід двовимірний масив даних, де кожне число відповідає певному кольору. Положення даного кольору на графіку визначається індексами елемента в масиві. Графіки та таблиці інтенсивності мають можливість відображати до 256 різних кольорів.

На рисунку 3.9 зображено масив розміру 4x3 візуалізований на графіку інтенсивності. Графік відображає транспонований масив.

Розглянемо налаштування графіків та таблиць інтенсивності.

Графіки та таблиці інтенсивності мають багато спільних властивостей із графіками діаграм та осцилограм, які можна відобразити або сховати, вибравши пункт контекстного меню **Visible Items**. Так як у графіках і таблицях інтенсивності з'являється третій вимір, то необхідний додатковий елемент - елемент керування колірною шкалою, який визначає діапазон та спосіб відображення кольорів даних. На рисунку 3.10 показано елементи графіка інтенсивності **Intensity Graph**.

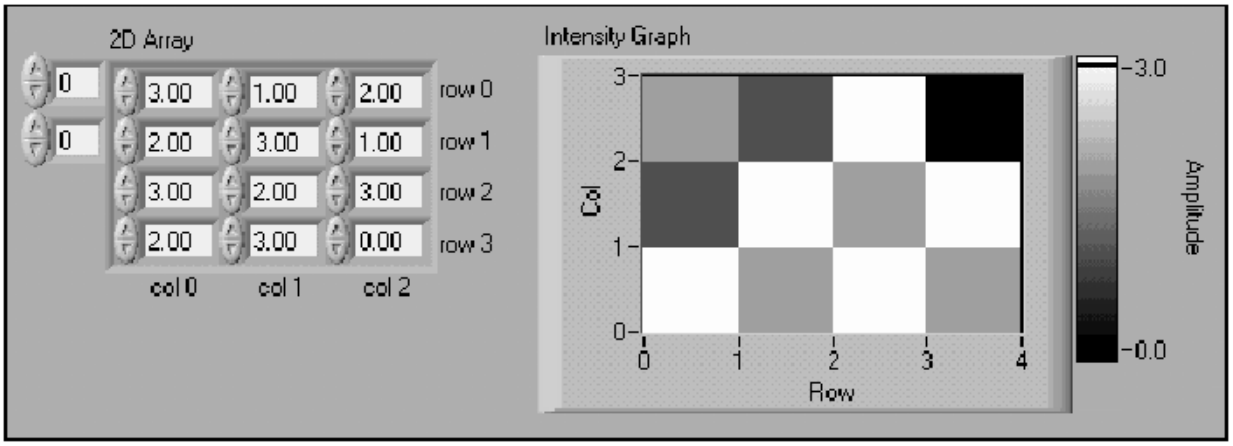


Рисунок 3.9 – Приклад подання даних на графіку інтенсивності **Intensity Graph**

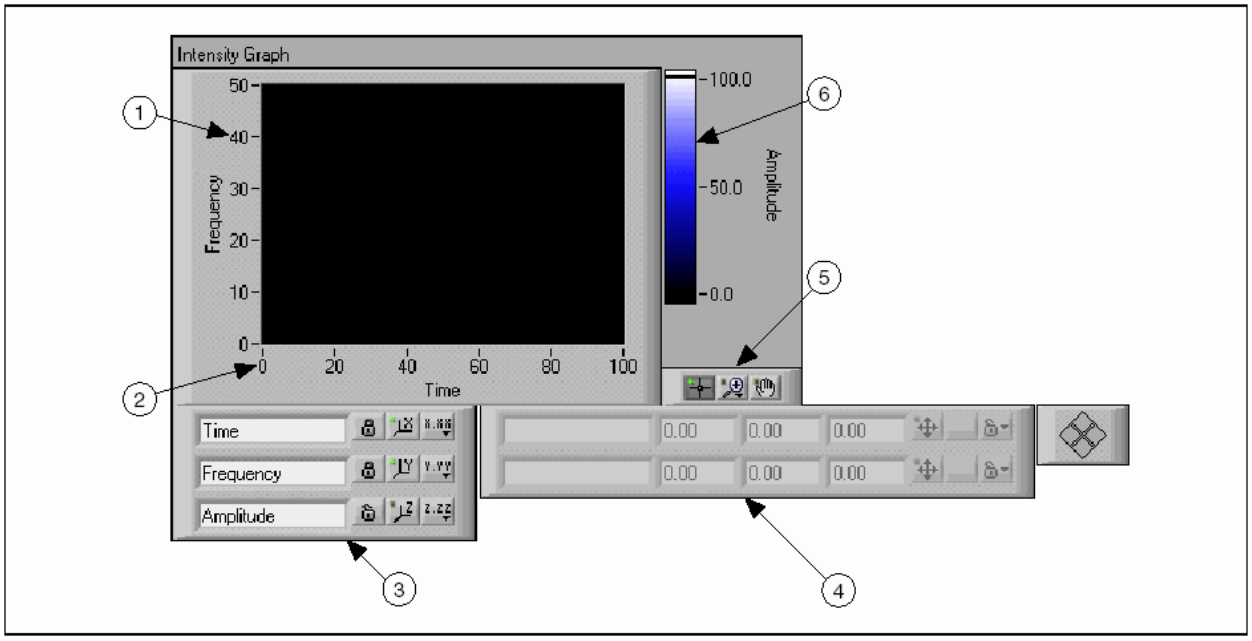


Рисунок 3.10 – Графік інтенсивності **Intensity Graph**

На рисунку 3.10 позначено:

1 - шкала Y (**Y scale**);

2 - шкала X (**X scale**);

3 - панель керування шкалами (**Scale legend**);

4 - панель керування курсорами (**Cursor legend**);

5 - палітра інструментів для роботи із графіком (**Graph Palette**);

6 - Шкала Z (шкала кольорів) (**Z scale (color ramp)**).

Щоб змінити колір, асоційований з маркером, потрібно вибрати пункт **Marker Color** у контекстному меню і вибрати колір у вікні вибору кольору. Контекстне меню викликається інструментами КЕРУВАННЯ або ПЕРЕМІЩЕННЯ натисканням правої кнопки миші по маркеру, розташованому біля колірної шкали.

Для додавання маркера до колірної шкали необхідно натиснути правою кнопкою миші на палітру кольорів і вибрати пункт **Add Marker** з контекстного меню.

Щоб змінити значення будь-якого маркера на колірній шкалі, потрібно перемістити маркер до необхідного значення інструментом КЕРУВАННЯ або використовувати інструмент ВВЕДЕННЯ ТЕКСТА для введення нового значення в текстове поле маркера.

3.3 Контрольні питання

3.3.1 Які засоби візуалізації використовуються для графічного відображення даних?

3.3.2 Які режими відображення даних використовуються в графіку діаграм **Waveform Chart**?

3.3.3 Яка функція використовується для об'єднання даних, що відображаються при виведенні кількох графіків на графік діаграм **Waveform Chart**?

3.3.4 У чому відмінність одиночного графіку осцилограм і графіку множини осцилограм?

3.3.5 В чому відмінність графіку осцилограм (**Waveform Graph**) і двохкоординатного графіку осцилограм (**XY Graph**)?

3.3.6 Як змінити колір в графікн інтенсивності **Intensity Graph**?

4 ПРОЄКТУВАННЯ ГЕНЕРАТОРІВ СИГНАЛІВ В LabVIEW

У середовищі LabVIEW розрізняють цифрові та «аналогові» сигнали.

Цифрові сигнали можуть приймати лише два значення 0 або 1. Такі сигнали характеризують стан вхідної чи вихідної цифрової лінії, або кількох ліній (регістру). До цифрових сигналів належать також послідовності імпульсів. Цифровим сигналам відповідають логічні (Boolean) джерела/приймачі даних.

«Аналогові» сигнали приймають безліч значень, що допустимі числовим поданням (Numeric), і можуть бути функціями часу, частоти, просторових координат і т.д. Лапки в даному випадку означають, що реальний аналоговий сигнал, представлений у доступній комп'ютеру цифровій формі, тобто – кінцевим числом відліків, квантованих за рівнем. Без прив'язки до часу (або іншої незалежної змінної) такий сигнал є числовим масивом. У літературі з LabVIEW для таких сигналів використовується також термін Pattern, що перекладається як послідовність (відліків сигналу).

Оцифрований аналоговий сигнал може бути отриманий з АЦП під час вимірювань і є масивом. Декілька сигналів, що надійшли через АЦП з однотипних датчиків (каналів), утворюють 2-мірний масив. Отже, можна говорити про двовимірний сигнал.

У LabVIEW існує спеціальний формат (кластер) даних – Waveform (неточний «переклад» – осцилограма), в якому зберігається інформація про абсолютну часову прив'язку сигналу: час першого відліку та крок між наступними.

ВП з бібліотеки **DAQmx** допускають читання та запис як масивів, так і осцилограм. В останньому випадку піклуватися про вказівку частоти подачі даних на ЦАП немає необхідності: якщо це допускається його технічними

характеристиками, на виході буде отримано аналоговий сигнал із правильним часовим масштабом.

Таким чином, "згенерувати сигнал" в LabVIEW означає "створити масив". Його часове масштабування можна виконати пізніше або взагалі обійтися без цього.

Для генерації сигналів - масивів (або послідовностей) у LabVIEW є широкий вибір можливостей [3]:

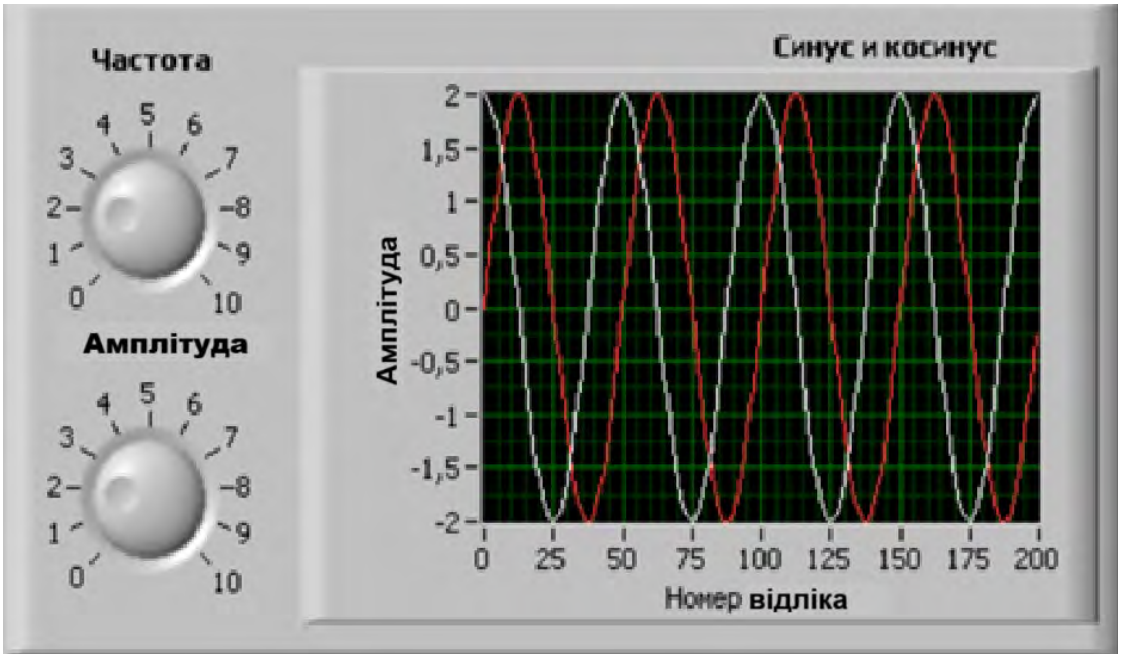
1) можна застосувати вбудовані функції із палітри **Numeric**, викликаючи їх циклічно. Наявність генератора випадкових чисел вже на цьому етапі дозволяє додавати до них шум. Використовуючи вузли **Expression Node** і **Formul Node**, можна визначити кожен відлік сигналу у вигляді формули як функцію його номера;

2) за допомогою набору функцій **Signal Generation** з палітри **Signal Processing** можна створювати ряд стандартних сигналів. Ці функції генерують послідовності відліків;

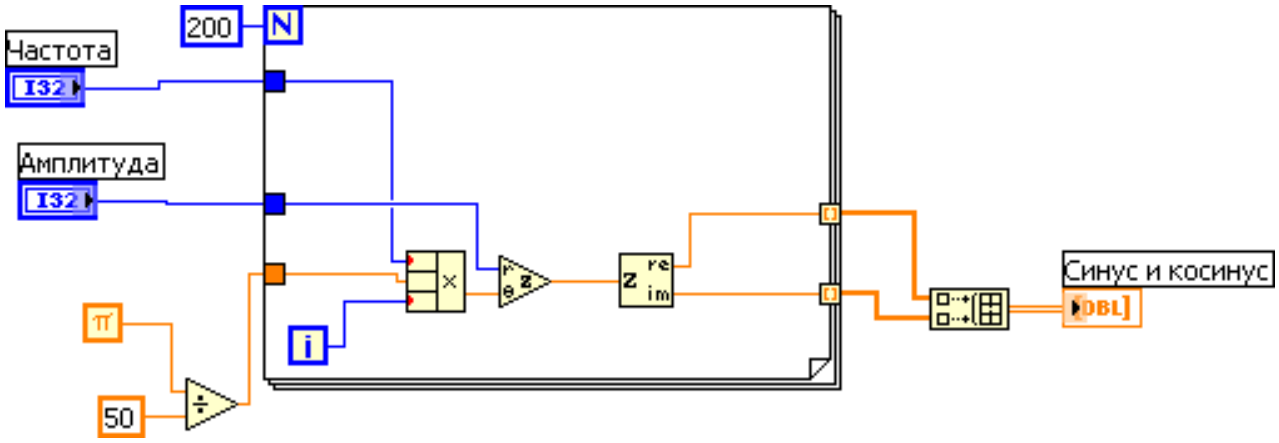
3) за допомогою бібліотеки **ВП Waveform Generation** з палітри **Signal Processing** сигнали можна одразу створювати масштабованими у часі. Зазначимо, що дані **ВП** написані з використанням функцій **Signal Generation** як вбудовані **ВП**.

4.1 Генерація сигналів за допомогою вбудованих функцій

Приклад генерації синусоїди та косинусоїди з використанням формули Ейлера, що реалізується за допомогою функцій з палітри **Numeric** та циклу **For Loop**, наведено на рисунку 4.1. При цьому фронтальна панель **ВП** представлена на рисунку 4.1, а, а блок діаграма - на рисунку 4.1, б.



а)



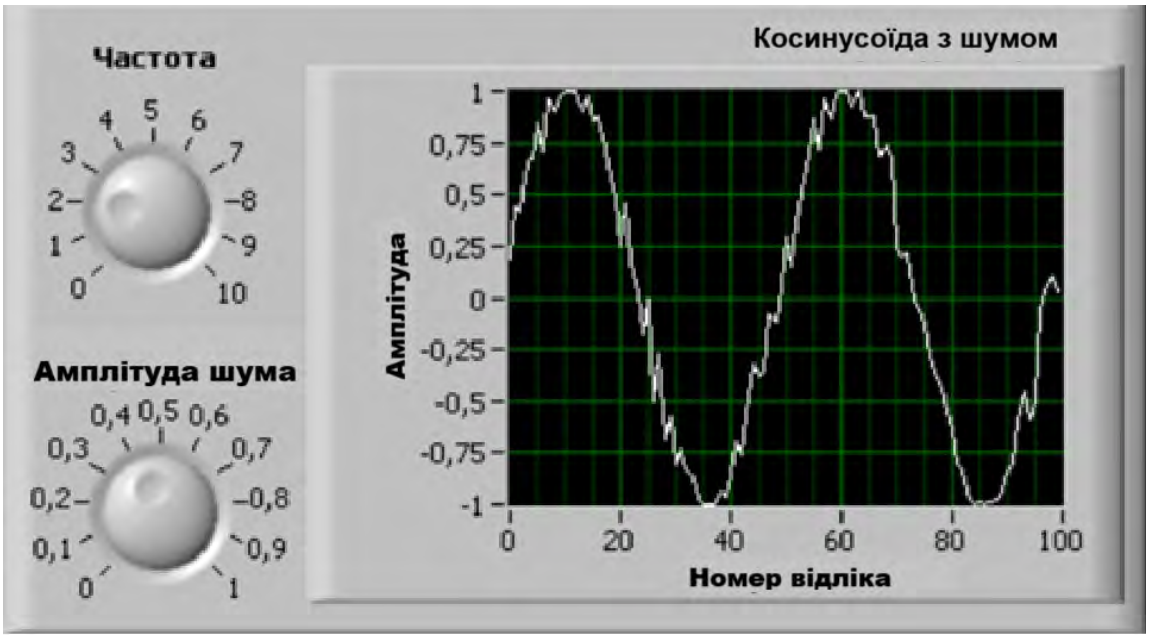
б)

Рисунок 4.1 - Генерація синусоїди та косинусоїди за допомогою формули Ейлера та функцій з палітри **Numeric**

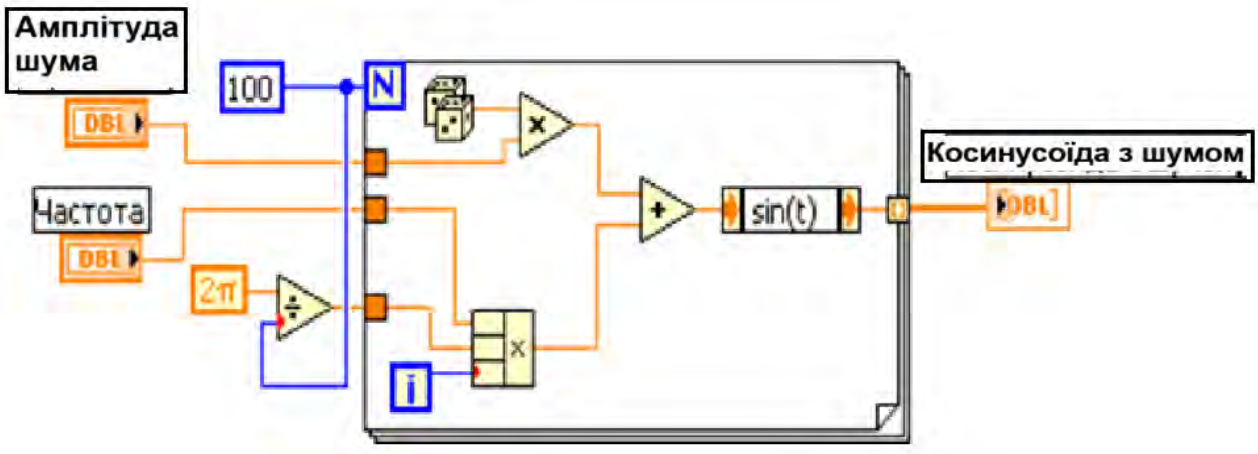
Для завдання фази комплексної експоненти у цьому прикладі використана компаундна арифметика.

У цій же панелі Numeric є генератор випадкових чисел, за допомогою якого і з використанням вузла Expression Node можна задати "зашумлену" косінусоїду.

Приклад генерації «зашумленої» косинусоїди наведено на рисунку 4.2: фронтальна панель **ВП** представлена на рисунку 4.2, а, а блок діаграма - на рисунку 4.2, б.



а)



б)

Рисунок 4.2 – Генерація «зашумленої» косинусоїди

4.2 Генерація сигналів із застосуванням палітри Signal Processing

4.2.1 Генерація сигналів за допомогою набору функцій Signal Generation

Для створення різних стандартних сигналів можна використовувати набір функцій **Signal Generation** із палітри **Signal Processing**. Ці функції генерують послідовності відліків відповідних стандартних сигналів. Для застосування функцій **Signal Generation** необхідно виконати:

Functions → Signal Processing → Signal Generation.

Приклад вікна **Signal Generation** наведено на рисунку 4.3.

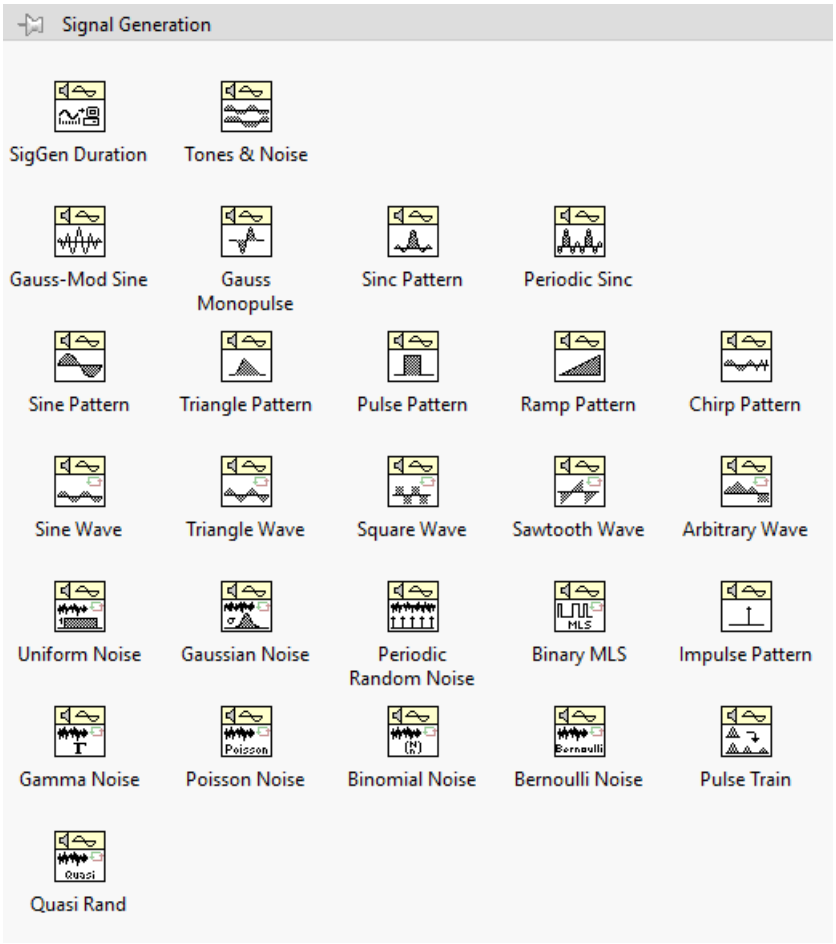


Рисунок 4.3 – Вікно **Signal Generation**

Як приклад розглянемо функціональний генератор сигналів заданої тривалості **Signal Generator by Duration**, що дозволяє відтворювати сигнали різної форми. Цей генератор має велику кількість вхідних параметрів (рисунок 4.4).

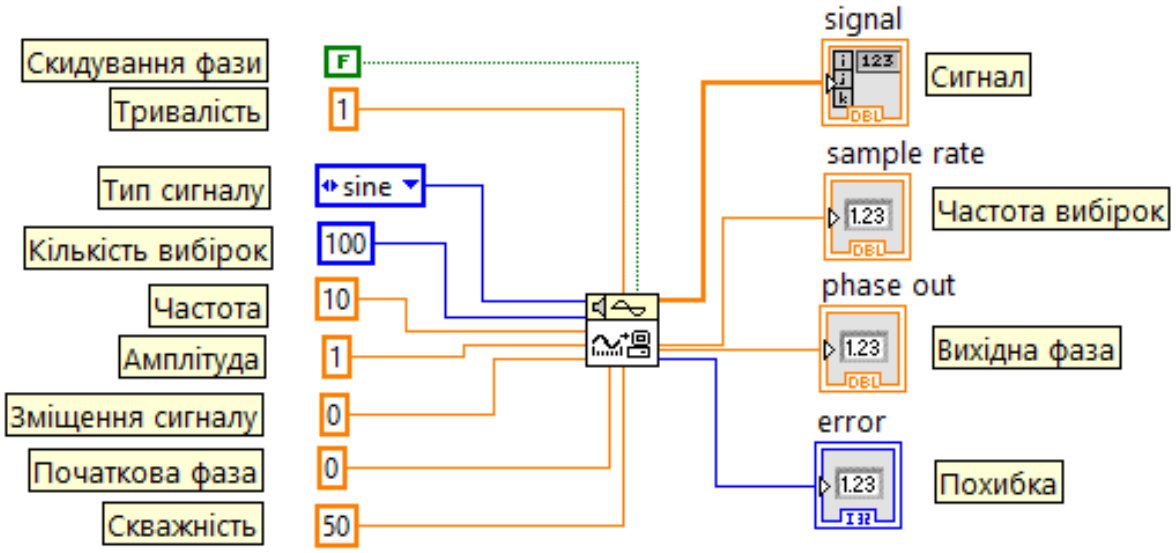


Рисунок 4.4 – Генератор сигналів **Signal Generator by Duration**

Вхід **Скидання фази** впливає на встановлення початкової фази сигналу, що генерується. Якщо на ньому встановлено значення TRUE, то фаза сигналу встановлюється відповідно до значення, вказаного на вході **Початкова фаза** (за замовчуванням встановлено значення 0). Якщо ж встановити FALSE, то вона дорівнюватиме значенню виходу **Вихідна фаза**, яке було при останньому запуску даного **ВП**. За замовчуванням на цей вхід подається значення TRUE.

На вході **Тривалість** задається час у секундах, що дорівнює тривалості вихідного сигналу. За замовчуванням значення цього входу дорівнює одиниці, а точніше – 1с.

За допомогою регулятора **Тип сигналу** можна вибрати форму сигналу, що генерується, а саме:

- синусоїдальний;

- косинусоїдальний;

- трикутний;

- прямокутний;

- пилкоподібний;

- лінійно наростаючий;

- лінійно спадаючий.

Вхід **Кількість вибірок** вказує на кількість точок сигналу у вихідному масиві. За замовчуванням встановлено 100 вибірок.

Вхід **Частота** визначає частоту вихідного сигналу. Цей параметр обов'язково потрібно узгодити зі значеннями входів **Кількість вибірок** та **Тривалість**, тому що в результаті порушення критерію Найквіста на виході можна отримати щось схоже на шум. За замовчуванням встановлено значення частоти 10 Гц.

Вхід **Амплітуда** дозволяє задавати потрібне значення амплітуди сигналу, що генерується. За замовчуванням значення входу **Амплітуда** дорівнює 1.

Вхід **Зміщення сигналу** задає постійну складову вихідного сигналу, тобто вказує, наскільки рівень сигналу буде вищим або нижчим за 0. За замовчуванням зсув відсутній, тобто, дорівнює 0.

Значення на вході **Скважність** застосовується при генерації прямокутного сигналу та визначається у відсотках. Цей параметр визначає час, протягом якого прямокутний сигнал матиме високий рівень. Параметр **Скважність** застосовується тільки для прямокутного сигналу, тому, якщо вибрати інший тип сигналу, цей вхід просто ігнорується. За замовчуванням на вході **Скважність** встановлено значення 50. При цьому на виході **Сигнал** генерується меандр – імпульсній сигнал з однаковою тривалістю високого та низького рівня.

Вихід **Частота вибірок** вказує частоту дискретизації вихідного сигналу, що визначається відношенням кількості вибірок до заданої тривалості.

Велика частина **ВП** з групи **Signal Generation** використовується для формування детермінованих сигналів. Ці спеціалізовані **ВП** можуть генерувати лише один тип сигналу. Розглянемо їх на прикладі генератора відрізка синусоїди **Sine Pattern** (рисунок 4.5).

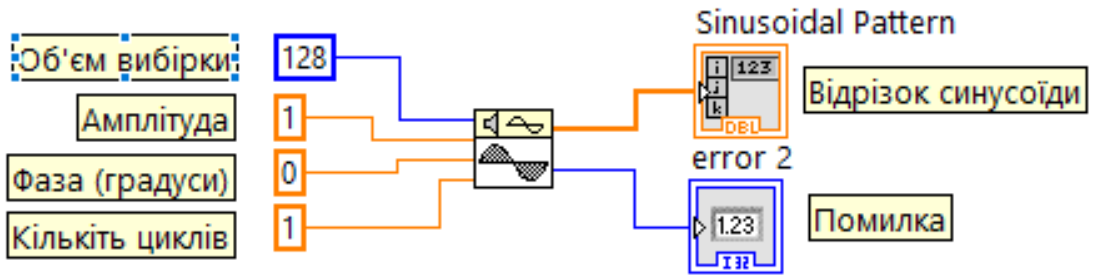


Рисунок 4.5 – Генератор синусоїди **Sine Pattern**

Входи **Амплітуда, Фаза, Об’єм вибірки** аналогічні до попереднього прикладу ВП **Signal Generator by Duration**.

Ключовим параметром є **Кількість циклів**. За замовчуванням встановлено один цикл, тобто буде згенеровано один період синусоїди [1, 3]. Але це число не обов'язково має бути цілим, його можна задавати з точністю до кількох знаків після коми. Іноді ця властивість може виявитися дуже корисною.

На рисунку 4.6 наведено два випадки, коли кількість циклів було задано дробовим: у першому випадку 0,3, а у другому – 0,5. У першому випадку вийшов сигнал складної форми, що не дуже нагадує синусоїду, а в другому - позитивний напівперіод синусоїди («випрямлена» двохнапівперіодним випрямлячем синусоїда), що застосовується досить часто.

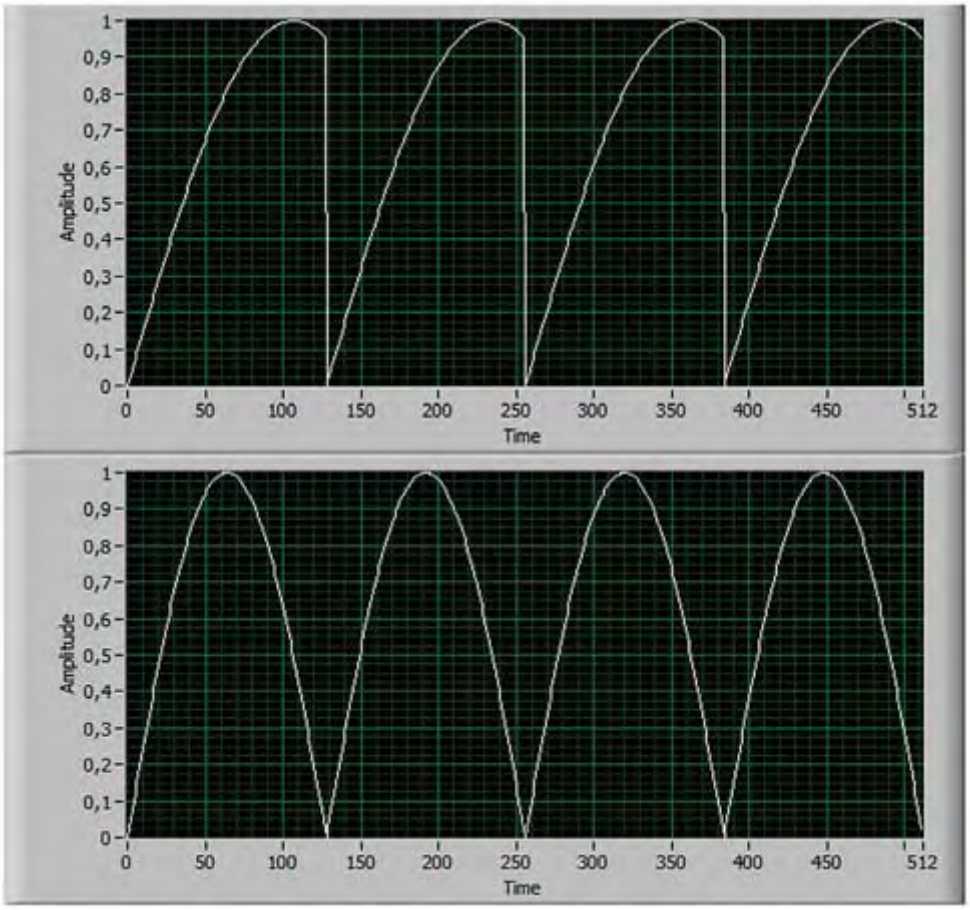


Рисунок 4.6 – Приклад дробової кількості циклів у блоці **Sine Pattern**

Крім розглянутого у списку генераторів з групи **Signal Generation** можна вказати такі найбільш розповсюджені для застосування:

- **Sine Wave** – генератор синусоїдального сигналу;
- **Triangle Wave** - генератор трикутного сигналу;
- **Square Wave** - генератор прямокутного сигналу;
- **Sawtooth Wave** генератор пилкоподібного сигналу;
- та інші, робота яких аналогічна роботі попередніх **ВП**.

У LabVIEW передбачено функцію генерації сигналу довільної форми **Arbitrary Wave** (рисунок 4.7).

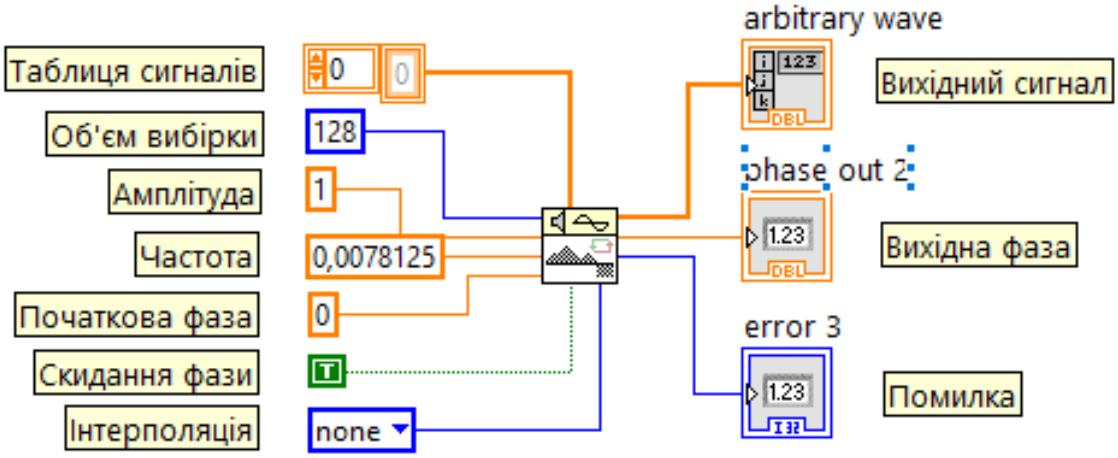


Рисунок 4.7 – Генератор сигналу довільної форми **Arbitrary Wave**

Розглянемо основні вхідні та вихідні параметри генератора (рисунок 4.7).

Вхід **Таблиця сигналів** є масивом, що задає форму коливання в межах одного періоду , причому весь сигнал вказується по точках.

На вхід **Частота** потрібно подавати значення в нормованих одиницях: відношення частоти до частоти дискретизації. Іншими словами, вони визначають, скільки точок припадатиме на один період сигналу. Наприклад: якщо встановити значення 0.1, за період буде отримано 10 точок. Тому, якщо задати на цей вхід значення більше одиниці, то на виході отримаємо пряму лінію з нульовим рівнем.

Інтерполяція визначає вид інтерполяції, що застосовується при формуванні сигналу. За замовчуванням встановлено значення "none", при якому жодних перетворень над даними зі входу **Таблиця сигналів** не виконується. Якщо ж поставити значення "linear", то **ВП** виконуватиме лінійну інтерполяцію при формуванні вихідного масиву.

У LabVIEW можна генерувати різні шумові сигнали [1, 3].

В якості прикладу розглянемо генератор рівномірного білого шуму **Uniform White Noise** (рисунок 4.8).

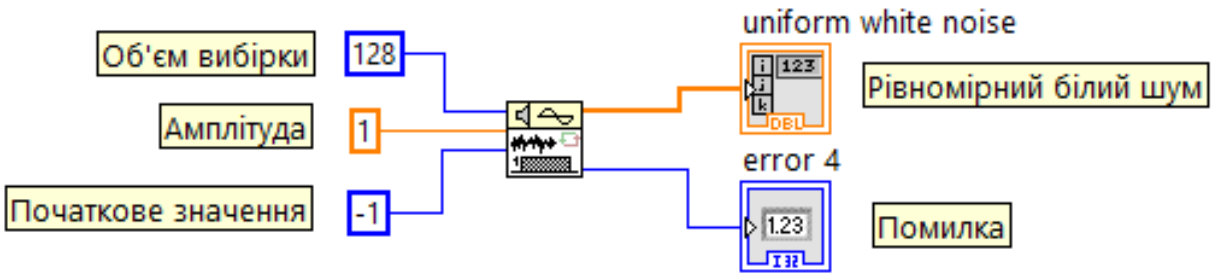


Рисунок 4.8 – Генератор рівномірного білого шуму **Uniform White Noise**

Цей **ВП** генерує псевдовипадковий білий шум із рівномірним законом розподілу.

Значення на вході **Амплітуда** вказує граничні як верхній, так і нижній рівні шуму. За замовчуванням амплітуда дорівнює 0.

Параметри інших входів встановлюються аналогічно до попередніх **ВП**.

Для формування зашумлених сигналів може використовуватися підсумовування вихідних сигналів відповідних генераторів, тобто: об’єднання виходу генератору потрібного корисного сигналу з виходом необхідного генератора шуму. При цьому необхідно узгодити їх параметри.

Щоб уникнути узгодження виходів двох генераторів можна використовувати спеціалізовані **ВП**.

Розглянемо генератор коливання та шуму, що дозволяє генерувати зашумлений гармонічний сигнал за допомогою блоку **Tones and Noise** (рисунок 4.9).

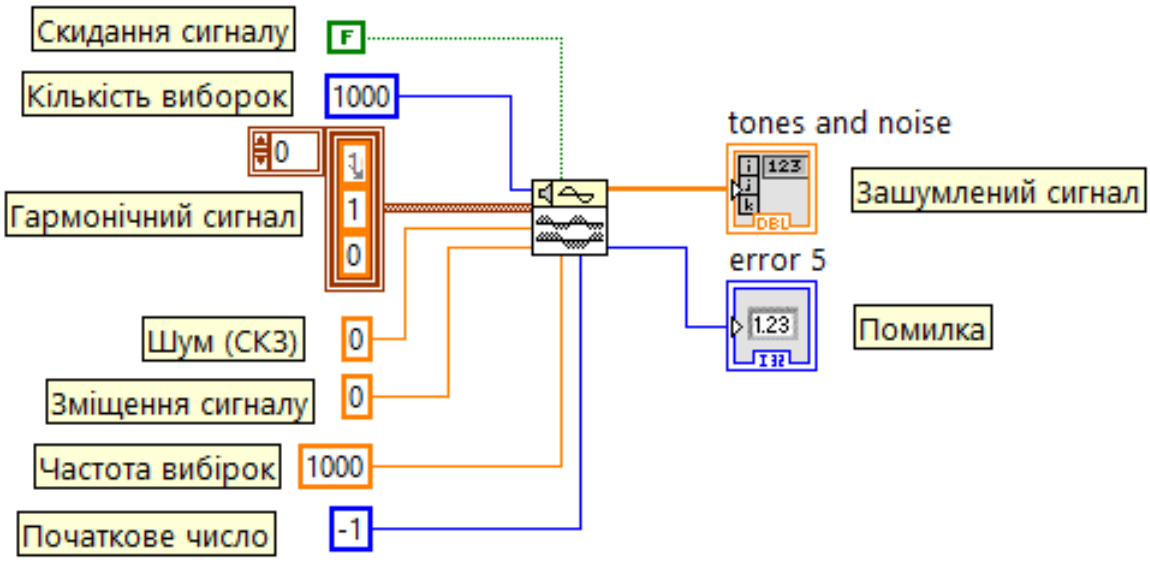


Рисунок 4.9 – Генератор коливання та шуму **Tones and Noise**

При подачі на вхід **Скидання сигналу** значення TRUE для кожного нового масиву даних, що генерується, відбувається скидання параметрів гармонічного коливання (фаза) і шуму (початкове число). За замовчуванням встановлено значення FALSE, тому скидання параметрів сигналу та шуму не відбувається.

Вхід **Гармонічний сигнал** представляє собою кластер і містить в собі основні параметри гармонічного сигналу:

- частота (за замовчуванням 10 Гц);

- амплітуда (за замовчуванням 1);

- початкова фаза (за замовчуванням 0).

За допомогою входу **Шум (СКЗ)** встановлюється середньоквадратичне значення Gauss*шуму [1, 3]. За замовчуванням на цьому вході встановлено значення 0, тому якщо його не змінити, на виході буде чиста синусоїда, без шуму.

Вхід **Частота вибірок** визначає кількість вибірок сигналу за секунду (частота дискретизації). За замовчуванням встановлено 1000 вибірок/с.

Вхід **Початкове число** зчитується лише тоді, коли на вході **Скидання сигналу** встановлено значення TRUE. У цьому випадку, якщо це число менше або дорівнює нулю, **ВП** для кожного масиву використовує заново згенеровану шумову складову. Якщо ж поставити одиницю або більше значення, то весь час повторюватиметься раніше отримана шумова складова.

4.2.2 Генерація сигналів за допомогою спеціальних ВП із бібліотеки Waveform Generation

Для генерації сигналів одразу масштабованих у часі можуть застосовуватись готові **ВП** з бібліотеки **Waveform Generation**, для чого необхідно виконати:

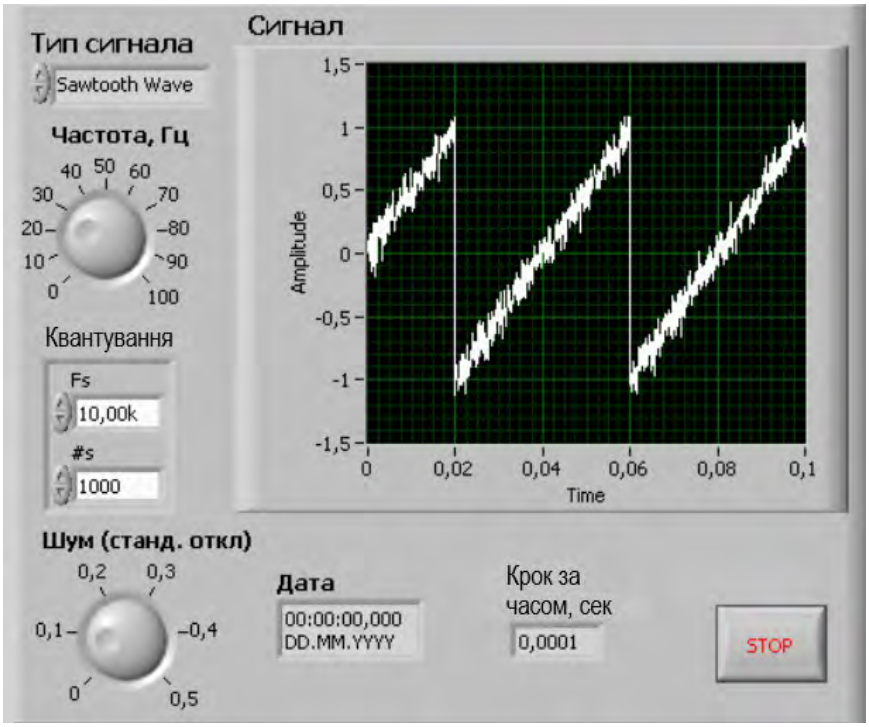
Functions → Signal Processing → Waveform Generation.

Приклад генерації пилкоподібного «зашумленого» сигналу за допомогою **ВП Basic Function Generator.vi** та **Gaussian White Noise Waveform.vi** (**Wave_form_gen.vi**) наведено на рисунку 4.10. При цьому фронтальна панель **ВП** представлена на рисунку 4.10, а, а блок діаграма - на рисунку 4.10, б.

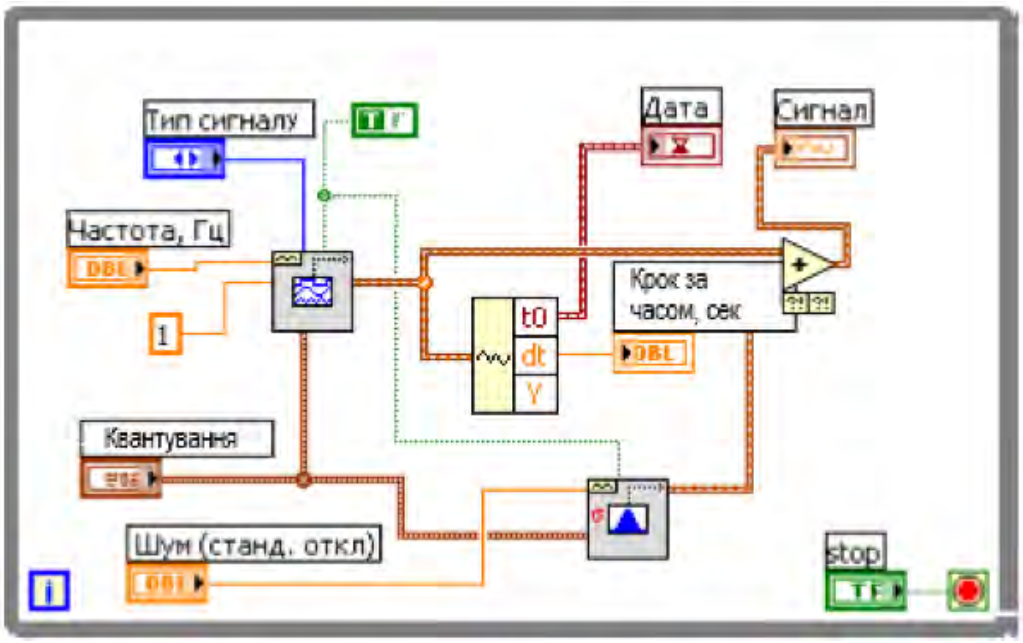
Параметри квантування на обидва ВП надходять з одного керуючого елемента квантування. Для підсумовування сигналу та шуму використовується поліморфна функція додавання. Якщо параметри квантування будуть відрізнятися, то під час підсумовування виникне помилка.

Доступ до часових параметрів згенерованої осцилограми дає функція **Get Waveform Components** із палітри **Waveform**. Індикатор **Дата** порожній рисунок 4/10, а). Щоб записати в нього системний час комп'ютера, можна скористатися функцією **Get Date Time in Second** з палітри **Time&Dialog** та

Build Waveform з палітри **Waveform**, включивши її в розрив вихідного провідника **ВП** – генератора стандартних функцій.



а)



б)

Рисунок 4.10 – Генерація пилкоподібного «зашумленого» сигналу

4.3 Контрольні питання

4.3.1 Які види сигналів розрізняють в LabVIEW?

4.3.2 Які особливості «аналогових» («квазі аналогових») сигналів в LabVIEW?

4.3.3 Якими способами можна генерувати сигнали в LabVIEW ?

4.3.4 Як впливає вхід **Скидання фази** на встановлення початкової фази генерованого сигналу в генераторі сигналів заданої тривалості **Signal Generator by Duration**?

4.3.5 Як задається форма коливання в генераторі сигналу довільної форми **Arbitrary Wave**?

5 СТРОКИ ТА ФУНКЦІЇ ФАЙЛОВОГО ВВОДУ/ВИВОДУ В LabVIEW

Значна увага у середовищі LabVIEW приділяється питанням обміну інформацією з іншими програмними комп'ютерними компонентами та зовнішніми приладами. Для цього застосовуються строки, що поєднують послідовності даних та є масивами ASCII символів, і спеціальні функції для роботи з файлами, що забезпечують введення/виведення даних в/з файл(а).

5.1 Строки

Строки - це послідовність символів ASCII, що відображаються і не відображаються. Строки забезпечують незалежний від платформи формат обміну даними.

Деякі з найбільш поширених строкових додатків забезпечують [1]:

- створення простих текстових повідомлень;

- передачу числових даних у прилади у вигляді рядків символів та перетворення рядків у числові дані;

- збереження цифрових даних на диску. Щоб зберігати числові дані як файл ASCII, необхідно перед записом перетворити їх у строки;

- діалогові вікна інструкцій та підказок.

На передній (фронтальній) панелі **ВП** строки даних з'являються у вигляді таблиць, полів введення тексту та міток.

5.1.1 Створення строкових елементів керування та відображення даних

Для роботи з текстом та мітками використовуються строкові елементи керування та відображення даних, розташовані на панелі:

Controls → Modern → String & Path.

Створення та редагування тексту строки здійснюється за допомогою інструментів КЕРУВАННЯ та ВВЕДЕННЯ ТЕКСТУ. Для зміни розміру строкового об'єкта на лицьовій панелі використовується інструмент ПЕРЕМІЩЕННЯ.

Для економії місця на лицьовій панелі можна використовувати смугу прокручування. Для цього необхідно клікнути правою кнопкою миші на строковому об'єкті і вибрати в контекстному меню пункт:

Visible Items → Scrollbar.

Тип відображення строкового об'єкта вибирається у контекстному меню. Типи відображення строки та приклади заповнення поля введення тексту показані у таблиці 5.1.

Таблиця 6.1 – Типи відображення строки

Тип відображення	Опис	Приклад тексту
1	**2**	**3**
Режим стандартного відображення (**Normal Display**)	Відображає стандартні ASCII коди, використовуючи шрифт елементу керування. Керуючі коди для друку виводяться на екран у вигляді квадратів.	There are four display types. \ is a backslash

1	2	3
Режим відображення зі зворотнім слешем недрукованих керуючих кодів (‘\’ **Codes Display**)	Виводить «\» для всіх недрукованих керуючих кодів.	There\sare\sfour\ sdisplay\.\n\\ \sis\sa\sbackslash
Режим скритого відображення тексту (**Password Display**)	Виводить «*» для всіх кодів текстового простору	*********
Режим відображення 16-тирічних ASCII кодів (**Hex Display**)	Виводить значення ASCII коду для кожного символу	5468 6573 6520 6172 6520 666F 6C61 7368 2E

5.1.2 Створення елементу керування «Таблиця»

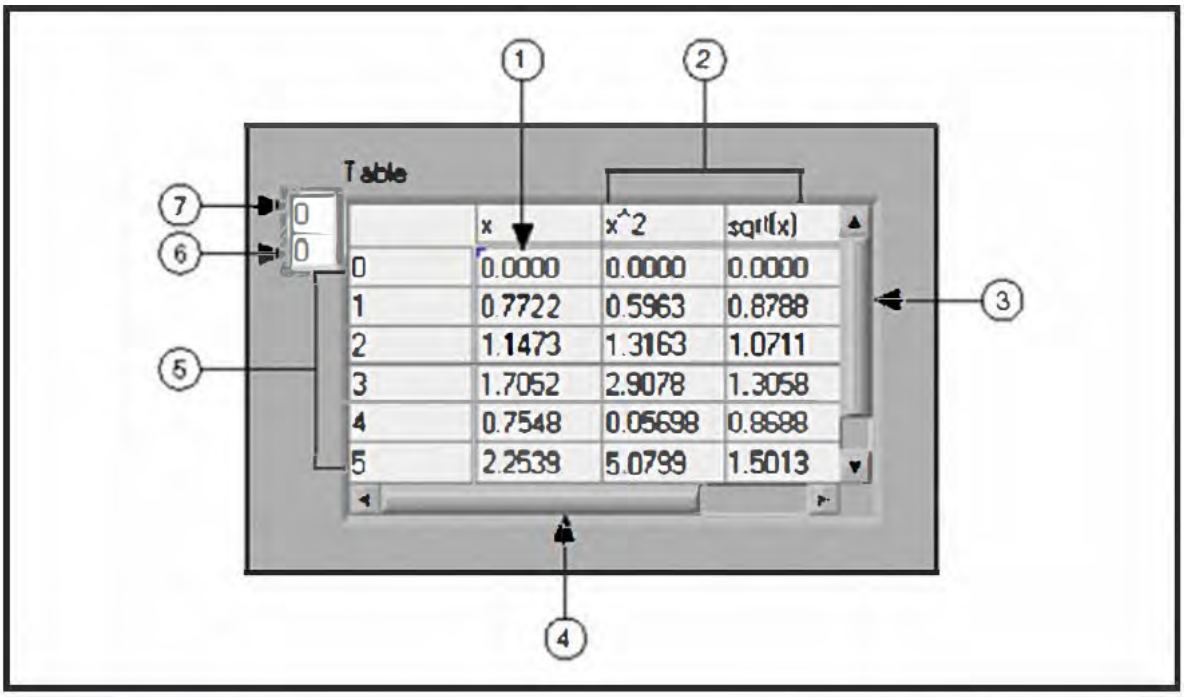
Елемент керування **Таблиця** розташований на панелі

Controls → Modern → List & Table

і призначений для створення таблиць на передній панелі.

Таблиця відображає двовимірний масив рядків і кожна комірка масиву знаходиться в рядку та стовпці таблиці. На рисунку 5.1 показано таблицю та її складові частини.

Таблиця – це двовимірний масив даних. Для ініціалізації значень комірок таблиці використовується інструмент КЕРУВАННЯ або ВВЕДЕННЯ ТЕКСТУ, за допомогою яких достатньо ввести текст у виділену комірку.



1. Комірка таблиці
2. Заголовок стовпця
3. Вертикальна полоса прокрутки
4. Горизонтальна полоса прокрутки
5. Заголовок строки
6. Індекс по горизонталі
7. Індекс по вертикалі

Рисунок 5.1 – Елемент керування **Таблиця**

Таким чином, для використання таблиці як елемента відображення даних, необхідно двовимірний масив чисел перетворити на двовимірний масив строк. Заголовки рядків та стовпців таблиці, як у таблиці символів, автоматично не відображаються. Необхідно створити одновимірний масив строк, що містить заголовки рядків та стовпців таблиці.

5.1.3 Функції роботи зі строками

Для редагування та керування строками на блок-діаграмі слід користуватися функціями обробки строк [1], що розташовані на панелі:

Functions → Programming → String.

Розглянемо основні функції роботи зі строками:

1) **String Length** – видає кількість символів строки, включаючи пробіли. Наприклад, функція **String Length** для тексту «*The quick brown fox*» видає значення «19», тобто 19 символів;

2) **Concatenate Strings** – поєднує строки та одновимірні масиви строк у окрему строку. Для збільшення полів введення даних функції слід змінити її розмір.

Наприклад, об'єднавши попередню строку «The quick brown fox» з наступним масивом строк:

	jumped	over	the	lazy	dog

,

- функція **Concatenate Strings** на виході виведе наступну строку:

«The quick brown fox jumped over the lazy dog»;

3) **String Subset** – видає підстроку певної довжини **length**, починаючи зі значення **offset** (зміщення). Зміщення першого елемента в рядку дорівнює **0**. Наприклад, якщо в полі введення даних функції подати попередню строку «The quick brown fox», то функція **String Subset** при значеннях:

offset = 4, length = 5

видасть значення:

quick.

4) **Match Pattern** – шукає для строки повторювану послідовність, подану на поле введення даних **regular expression**, починаючи зі значення зміщення **offset**, і, якщо знаходить відповідність, розбиває строку на три підстроки. Якщо відповідність не знайдено, поле виведення даних **match substring** є порожнім, а значення поля виведення даних **offset past match** (зміщення послідовності, що повторюється для строки) дорівнює -1.

Наприклад, на поле **regular expression** (шаблон підстроки) подається значення «**:**», а рядок на вході:

«VOLTS DC: +1.22863E+1».

Функція **Match Pattern** видасть величини:

- **before substring** (перед підстрокою) - «VOLTS DC»;

- **match substring** (шаблон підстроки) – «:»;

- **after substring** (після підстроки) – «+1.22863E+1»;

- **offset past match** – «9».

5.1.4 Перетворення чисельних даних в строкові

Для перетворення числових даних в строкові використовуються віртуальний прилад **Build Text Express** та функція **Format Into String** (конвертування в строку) [1]. Обидві ці функції мають вхідні та вихідні кластери помилок. При нестачі місця на блок-діаграмі краще використовувати функцію **Format Into String**.

Експрес-**ВП Build Text Express**, розташований на панелі:

Functions → Express → Output,

і здійснює об'єднання вхідних строк. Якщо вхідні величини мають не строковий тип даних, то вони перетворюються на строку відповідно до налаштувань цього Експрес-**ВП**.

При розміщенні Експрес-**ВП Build Text** на блок-діаграмі з'являється діалогове вікно параметрів **Configure Build Text**. У прикладі, наведеному на рисунку 5.2, значення напруги подається на вхід Експрес-**ВП** і перетворюється до формату даних з плаваючою комою з 4 числами після коми. Потім це значення додається до кінця строки **Voltage is** (Напруга дорівнює).

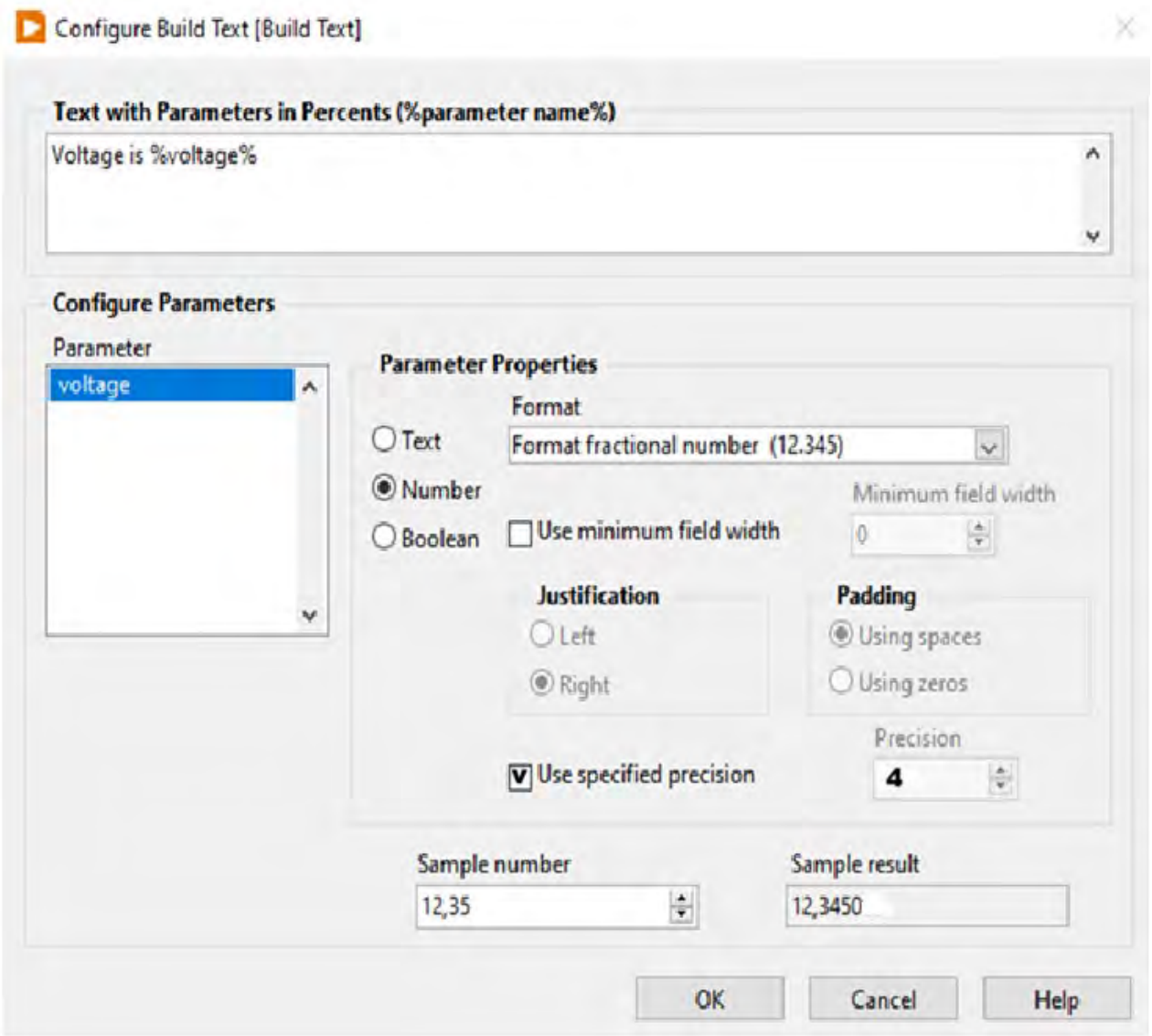


Рисунок 5.2 – Віно налаштувань **Configure Build Text**

При такому налаштуванні Експрес-**ВП** виглядає так, як представлено на рисунку 5.3. Для спостереження за вихідною строкою використовується налагоджувальний індикатор. Будь-які значення, що подаються на поле введення даних **Beginning Text**, будуть приєднуватися до початку тексту діалогового вікна налаштувань.

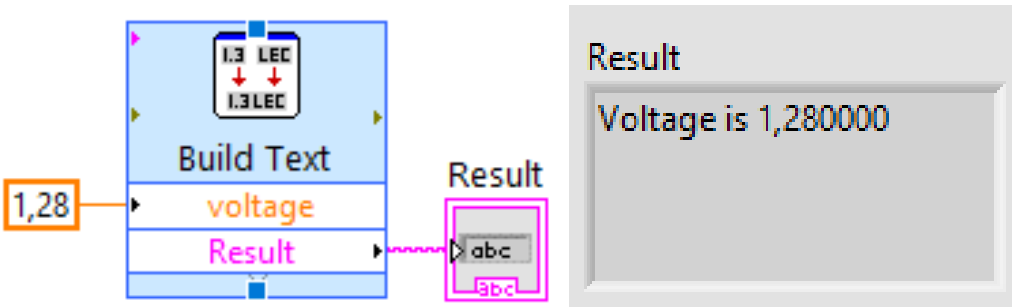


Рисунок 5.3 – Диалогове вікно налаштувань у **Build Text**

Функція **Format Into String** перетворює параметри будь-якого формату, такі як числові дані, на строку. Для збільшення кількості параметрів слід змінити розмір функції.

У прикладі на рисунку 5.4 функція **Format Into String** видає вказану на індикаторі налагодження строку при значеннях полів:

- **format string** (формат строки) - «%.4f»;

- **input string** (вхідна строка) – «Voltage is » (враховуючи пробіл в кінці);

- параметр – «1,28».

У форматі строки **format string**:

- **%** – вказує початок формату строки;

- число після точки визначає точність подання числа, тобто в прикладі «4» показує кількість знаків після коми»

- **f** – вказує тип даних із плаваючою комою.

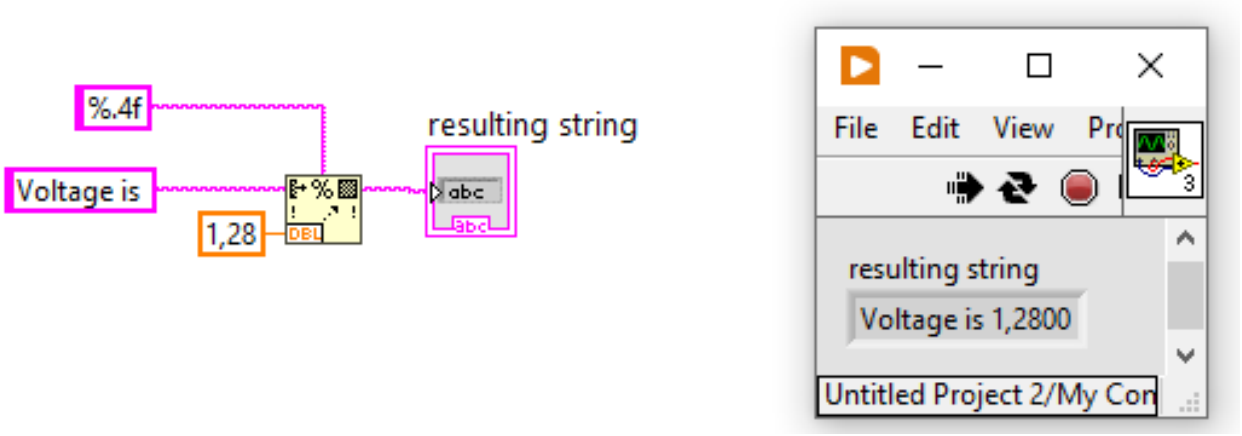


Рисунок 5.4 – Функція **Format Into String**

Для створення та редагування формату строки слід клікнути правою кнопкою миші по функції та вибрати пункт контекстного меню **Edit Format String**. На рисунку 5.5 показано вигляд діалогового вікна **Edit Format String** попереднього прикладу.

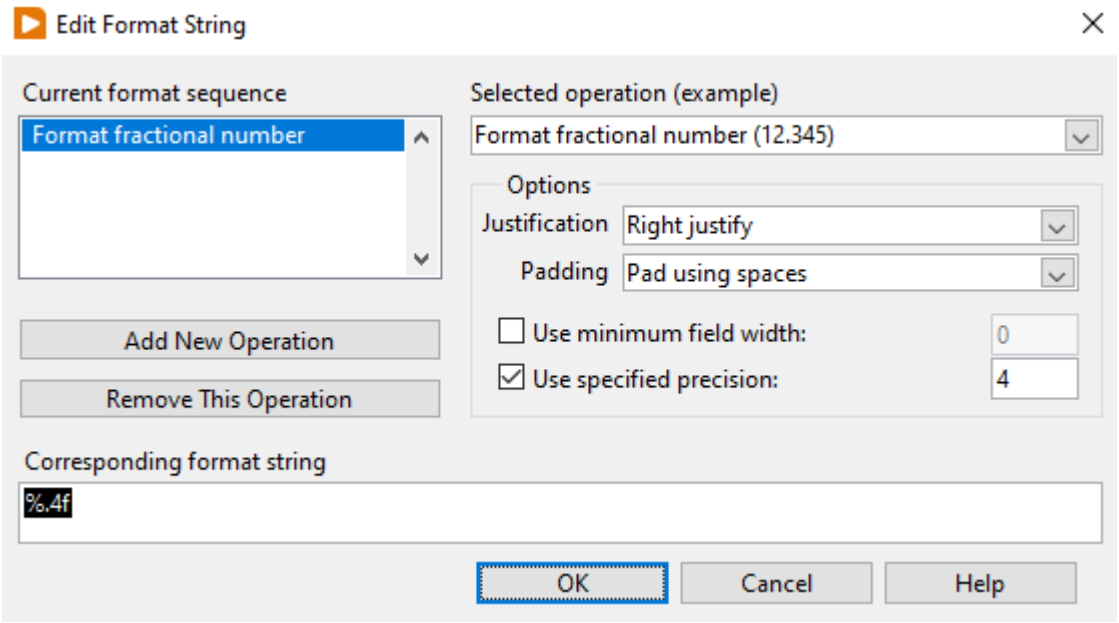


Рисунок 5.5 – Діалогове вікно **Edit Format String**

Для перетворення строки на числові дані слід використовувати функцію **Scan From String.**

Функція **Scan From String** перетворює строку, що містить допустимі числові символи, такі як «**0-9**, +, -, **e**, **E**» і роздільник «**.**», у дані числового формату. Функція починає переглядати строку, що подається на поле введення даних **input string**, починаючи із номера символу, що задається на полі **initial search location**. Функція може переглядати вхідну строку різних типів даних, таких як числові або логічні дані, ґрунтуючись на форматі строки. Для збільшення кількості полів виведення даних слід змінити розмір функції.

Наприклад, при значеннях на полях введення даних (рисунок 5.6):

- **format string** – %f;

- **initial search location** – 8:

- **input string** – VOLTS DC+1.28E+2,

- функція видає результат 128.00.

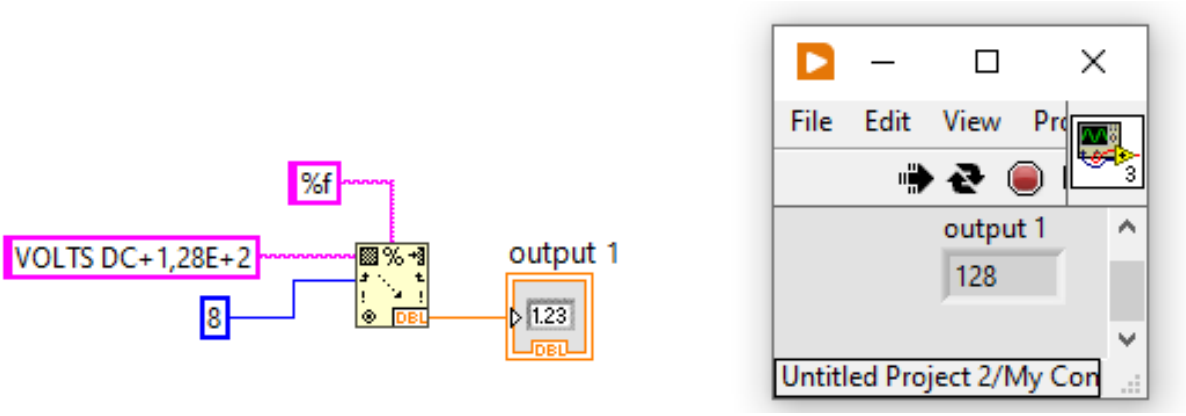


Рисунок 5.6 – Функція **Scan From String**

У форматі рядка **%** – вказує початок формату строки, **f** – вказує тип даних із плаваючою комою [1].

Для створення та редагування формату строки слід клікнути правою кнопкою миші за функцією та вибрати пункт контекстного меню **Edit Scan String**.

5.2 Функції файлового вводу/виводу

Функції файлового вводу/виводу відіграють значну роль у обміні даними і виконують файлові операції зчитування та запису.

Типова операція файлового вводу/виводу включає такі процеси (рисунок 5.7):

1) створення або відкриття файлу. Після відкриття файлу його представляє унікальний ідентифікатор, званий **refnum** (посилання);

2) **VI (ВП)** або функція файлового введення-виведення читає дані з файлу або записує їх у файл;

3) закриття файлу;

4) перевірку виникнення помилок.

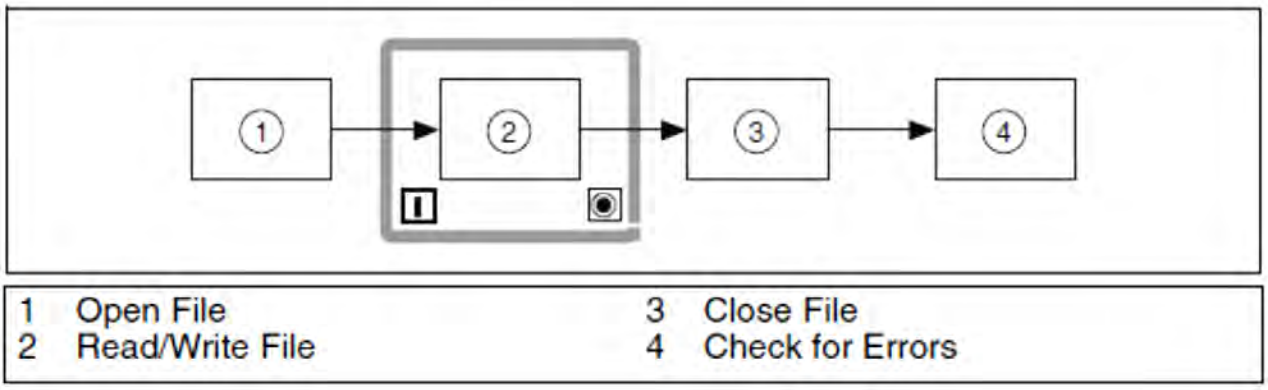


Рисунок 5.7 – Типова операція файлового вводу/виводу

У LabVIEW можна використовувати або створювати файли в таких форматах: Binary, ASCII, LVM і TDMS.

Binary. Файл має двійковий формат, що є базовим для всіх інших форматів.

Двійкові файли є найбільш ефективними, оскільки вони займають найменший дисковий простір і не потрібують конвертувати дані в текстовий формат і назад під час кожної операції збереження або вилучення даних.

ASCII. Файл ASCII представляє собою особливий тип двійкового файлу, який є стандартом для більшості програм. Він складається з набору ASCII-кодів. Файли ASCII також називаються текстовими файлами.

Текстові файли є найлегшим форматом для використання та поширення. Майже всі комп'ютери та програми можуть читати і записувати текстові файли. Більшість додатків для керування приладами також використовують текстові рядки.

Однак текстові файли зазвичай займають більше місця, ніж двійкові файли. Крім того, є труднощі довільного доступу до числових даних у текстових файлах. Хоча кожен символ у рядку займає точно 1 байт, кількість знакомісць для подання числа в текстовому вигляді зазвичай не є фіксованою. Щоб знайти дев'яте число в текстовому файлі, потрібно спочатку прочитати і перетворити попередні вісім чисел.

LVM. Файл даних вимірювань у LabVIEW (*.lvm) - текстовий файл із роздільниками у вигляді табуляції, який можна відкрити в табличному або текстовому редакторі. Файл *.lvm містить інформацію про дані, наприклад, дату і час їхньої генерації. Цей формат - спеціальний тип ASCII-файлу, створений для LabVIEW.

TDMS. Спеціальний тип двійкового файлу, створений для продуктів National Instruments. Він складається з двох окремих файлів:

- двійкового файлу, де зберігаються дані та записані властивості даних;

- двійкового файлу індексів, що забезпечує зведену інформацію про всі атрибути та покажчики в двійковому файлі.

Функції файлового вводу/виводу розташовані на панелі:

Functions → Programming → File I/O.

Вони призначені для:

- відкриття та закриття файлу даних;

- зчитування та запису даних з/в файл(а);

- зчитування та запису даних з/в файл(а) у вигляді таблиці символів;

- переміщення та перейменування файлів та каталогів;

- зміни параметрів файлу;

- створення, зміни та зчитування конфігураційних файлів.

Палітра функцій файлового вводу/виводу (рисунок 5.8) може бути розділена на три частини:

- функції високого рівня (high level File I/O);

- функції низького рівня (low level File I/O);

- підпалітра функцій розширених можливостей (advanced File I/O).

Функції файлового вводу/виводу високого рівня розташовані у верхньому рядку палітри **Functions → Programming → File I/O**. Вони призначені для виконання основних операцій із вводу/виводу даних.

Використання функцій файлового вводу/виводу високого рівня дозволяє скоротити час і зусилля програмістів під час запису та зчитування даних в/з файл(а).

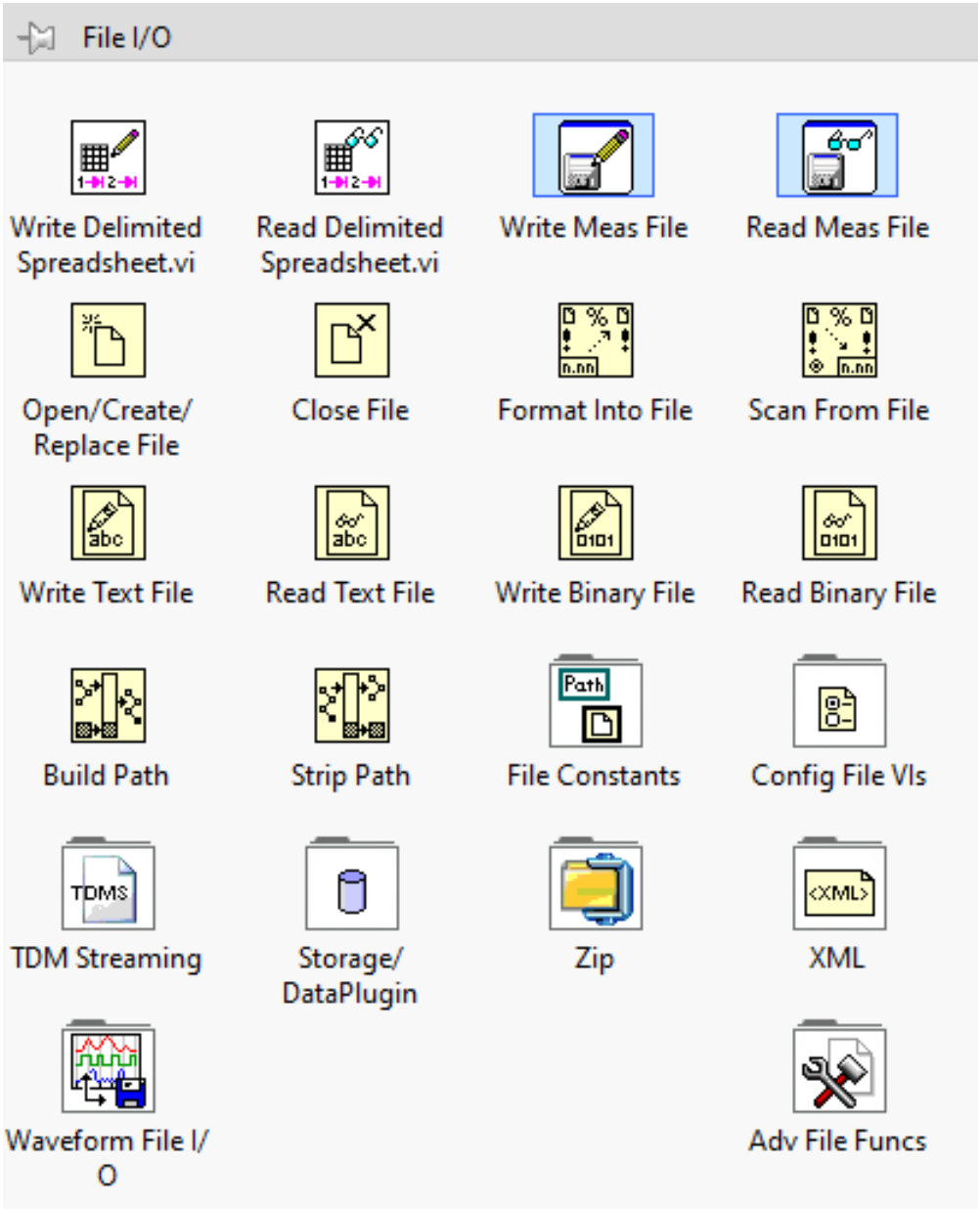


Рисунок 5.7 – Палітра функцій файлового вводу/виводу

Функції файлового вводу/виводу високого рівня виконують запис та зчитування даних та операції закриття та відкриття файлу. За наявності помилок функції файлового вводу/виводу високого рівня відображають діалогове вікно з описом помилок та пропонують на вибір: продовжити виконання програми або зупинити її. Однак через те, що функції даного класу поєднують весь процес роботи з файлами в один **ВП**, переробити їх під певне завдання важко.

У LabVIEW є такі **ВП** високорівневого файлового вводу-виводу:

- **Write Delimited to Spreadsheet.vi** - здійснює запис до файлу електронних таблиць, перетворює двовимірний чи одновимірний масив чисел подвійної точності на текстовий рядок і записує рядок до нового ASCII-файлу або додає рядок до наявного. При цьому дані можна транспонувати.

ВП відкриває або створює файл до запису в нього, а після запису закриває. Цей **ВП** можна використовувати для створення текстового файлу, що читається більшістю табличних додатків;

- **Read Delimited to Spreadsheet.vi** - здійснює зчитування з файлу електронних таблиць. При цьому він читає задану кількість ліній чи рядків із числового текстового файлу, починаючи із заданого в символах зміщення, і перетворює дані у двовимірний масив чисел із подвійною точністю.

ВП відкриває файл до читання з нього, а після читання закриває. Цей **ВП** можна використовувати для читання табличного файлу, збереженого в текстовому форматі;

- Write to Measurement File - запис у файл вимірювань. Експрес-**ВП**, який записує дані в текстовий (*.lvm) або двійковий (*.tdms) файл вимірювань. Можна задати метод збереження, формат файлу (*.lvm або *.tdms), тип заголовку і роздільник;

- **Readfrom Measurement File** - зчитування з файлу вимірювань за допомогою експрес - **ВП**, який читає дані з текстового (*.lvm) або двійкового (*.tdms) файлу вимірювань. Можна задати ім'я файлу, формат файлу і розмір сегмента.

Для специфічних завдань слід використовувати функції файлового вводу/виводу низького рівня.

Функції файлового вводу/виводу низького рівня розташовані в середньому рядку палітри **Functions → Programming → File I/O**.

Низькорівневі **ВП** файлового вводу/виводу виконують по одній операції файлового вводу/виводу. Наприклад, існує одна функція для відкриття ASCII - файлу, одна функція - для його читання, і одна функція - для закриття. Використовуйте Функції файлового вводу/виводу низького рівня доцільно використовувати, коли файловий ввод/вивід відбувається всередині циклу.

Функції файлового вводу/виводу низького рівня використовуються для створення нового або звернення до раніше створеного файлу, запису та зчитування даних та закриття файлу. Функції низького рівня роботи з файлами підтримують усі операції, необхідні під час роботи з файлами.

Додаткові функції роботи з файлами (**Advanced File I/O**) розташовані на панелі:

Functions → Programming → File I/O → Advanced File Functions

і призначені для керування окремими операціями над файлами.

Функції файлового вводу/виводу можна також використовувати для потокового запису на диск, що зберігає ресурси пам'яті шляхом зменшення кількості разів, коли функція взаємодіє з операційною системою для відкриття та закриття файлу. Потоковий запис на диск - методика збереження файлів відкритими, поки, наприклад, повторювані операції запису виконуються в циклі.

Щоб уникнути відкриття і закриття одного й того самого файлу, потрібно передати посилання (**refnum**) на файл у цикл. Під час відкриття файлу, пристрою або мережевого з'єднання, LabVIEW створює посилання, пов'язане з цим файлом, пристроєм або мережевим з'єднанням. Для всіх операцій з відкритими файлами, пристроями або мережевими з'єднаннями використовується це посилання для ідентифікації об'єкта.

На рисунку 5.8 наведено приклад звичайного, не потокового запису на диск, коли **ВП** повинен відкривати та закривати файл у кожній ітерації циклу.

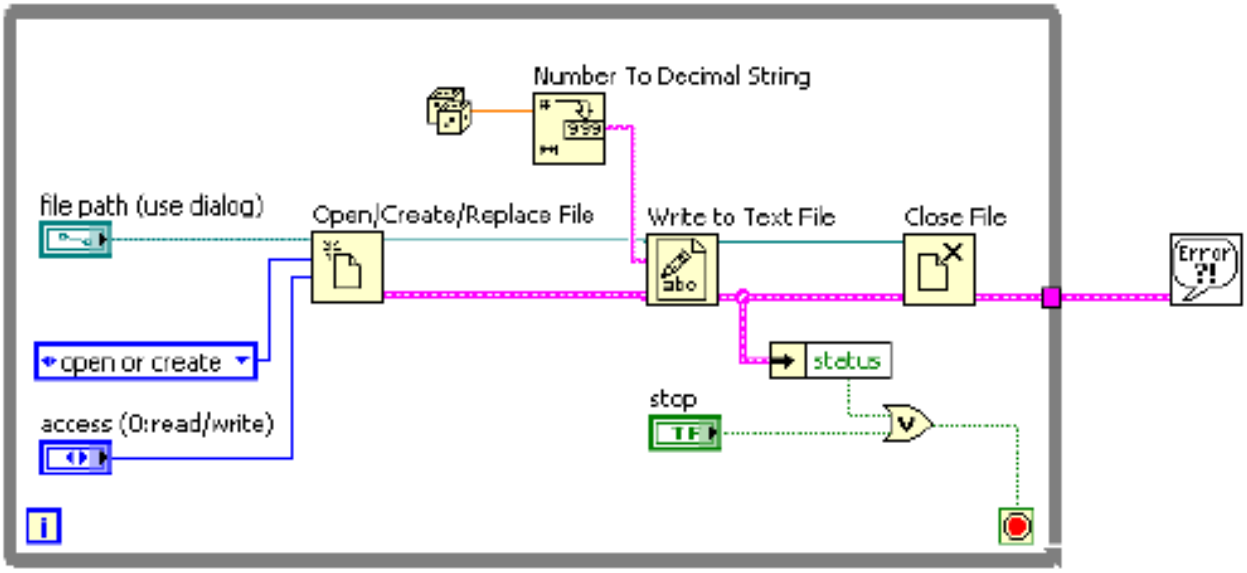


Рисунок 5.9 – Приклад не потокового запису на диск

На рисунку 5.10 представлений приклад **ВП, що** використовує потоковий запис на диск для зменшення кількості разів, коли він повинен взаємодіяти з операційною системою для відкриття і закриття файлу. Відкривши файл один раз до початку циклу і закривши його після закінчення циклу, можна значно підвищити швидкодію **ВП**, оскільки прибираються дві операції з файлами в кожній ітерації циклу.

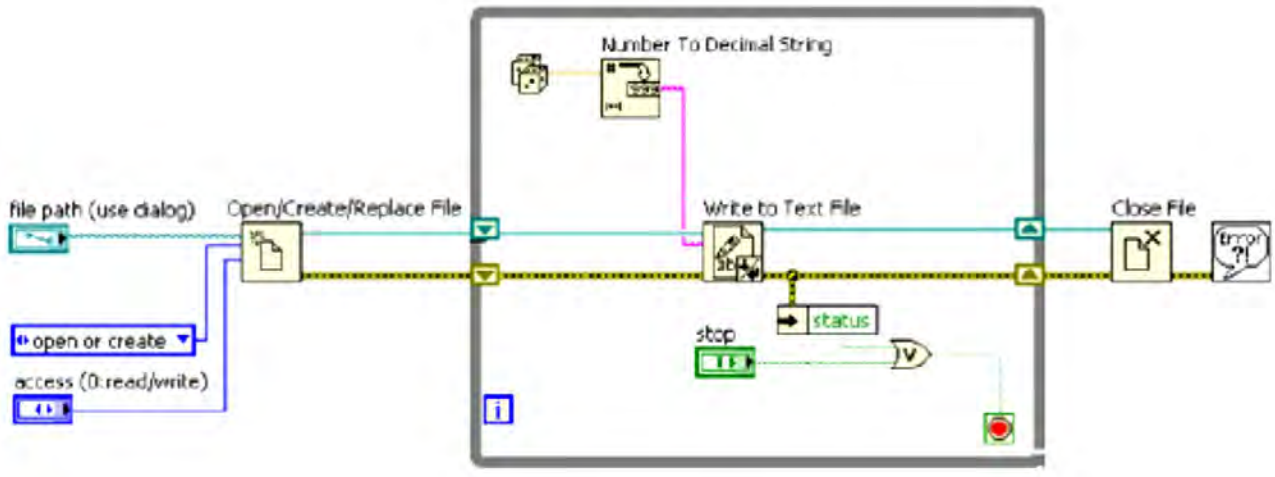


Рисунок 5.10 – Приклад потокового запису на диск

5.2.1 Основи файлового вводу/виводу

Стандартні операції вводу/виводу даних з файлу, як було вказано, складаються з наступної послідовності дій:

1 Створення чи відкриття файлу. Вказується місце розташування існуючого файлу або шляхи для створення нового файлу за допомогою діалогового вікна LabVIEW. Після відкриття файл LabVIEW створює посилання на нього.

2 Здійснення операцій зчитування або запису даних в/з файл(а).

3 Закриття файлу.

4 Обробка помилок.

Для здійснення основних операцій низькорівневого файлового вводу/виводу використовуються такі **ВП** та функції (рисунок 5.7):

1) **Open/Create/Replace File** - відкриває, перезаписує існуючий файл або створює новий. Якщо **file path** (шлях розміщення файлу) не вказаний, **ВП** виводить на екран діалогове вікно, в якому можна створити новий або вибрати вже існуючий файл;

2) **Read File (Binary, Text)** – зчитує відповідні дані з файлу, що визначається за посиланням **refnum**, і видає дані на поле виведення **data**, на поле **count** подається значення кількості даних, що зчитуються. Зчитування даних починається з місця, яке визначається елементами **pos mode** і **pos offset**, і залежить від формату файлу;

3) **Write File (Binary, Text)** – записує дані у файл, який визначається за посиланням **refnum**. Запис починається з місця, що визначається полями введення даних **pos mode** і **pos offset** для потокового файлу байтових даних, і покажчиком кінця файлу для файлу протокольованих даних;

4) **Close File** – закриває файл, вказаний у посиланні **refnum**;

5) **обробка помилок.** Підпрограми ВП та функції низького рівня містять інформацію про помилки. Для їх обробки використовуються підпрограми обробки помилок, такі як **Simple Error Handler VI** (**ВП** «Простий обробник помилок»), розташований на панелі:

Functions → Programming → Dialog & User Interface.

Поля введення **error in** та виведення **error out** інформації про помилки використовуються у кожному **ВП** для обміну інформацією про помилки між різними **ВП**.

Під час роботи **ВП** LabVIEW перевіряє наявність помилок у кожному вузлі. Якщо LabVIEW не знаходить помилок, вузол виконується нормально. Якщо LabVIEW виявляє помилку в одному вузлі, його виконання переривається, а інформація про помилку передається наступному вузлу. Наступний вузол чинить так само, і в кінці виконання LabVIEW повідомляє про помилки.

5.2.2 Збереження даних у новому або вже існуючому файлі

У файл, створений або відкритий за допомогою функцій вводу/виводу, можна записати дані будь-якого типу. При необхідності доступу до файлу з боку інших програм або користувачів слід записувати дані у вигляді рядка ASCII символів [1, 2].

Доступ до файлу можна здійснити програмним шляхом або за допомогою діалогового вікна.

Для доступу до файлу за допомогою діалогового вікна на полі введення **file path** підпрограми **ВП Open/Create/Replace File VI** не слід подавати дані. Програмний доступ до файлу заощаджує час.

Таблиця 5.2 показує організацію шляхів до файлів.

Таблиця 5.2 – Організація шляхів до файлів

Платформа	**Шлях до файлу**
Windows	Складається з ім’я кореневого шляху, двокрапки, зворотнього слешу, розділяючого директорію, та ім’я файлу. Наприклад: *C:\testdata\test1.dat* – шлях до файлу *test1.dat* в папці *testdata*.
UNIX	Складається з прямого слеша, розділяючого директорії та ім’я файла. Наприклад: */home/testdata/test1.dat* – шлях до файлу *test1.dat* в папці *testdata* в каталозі *home*. Ім’я файла та директорії чутливі до регистрів.
MacOS	Складається з ім’я кореневого шляху, двокрапки, ім’я папок, розділениз двокрапками, та ім’ям файла. Наприклад: *Hard Disk:testdata:test1.dat* – шлях до файлу *test1.dat* в папці *testdata* на диску *Hard Disk*.

У наведеному на рисунку 5.11 прикладі показано, як записати строку даних у файл при програмному вказуванні шляху та імені файла. Якщо файл вже існує, він перезаписується, якщо ні - то створюється новий файл.

Підпрограма **ВП Open/Create/Replace File VI** відкриває файл **test1.dat**. **ВП** також створює посилання на файл та кластер помилок.

При відкриванні файлу, пристрою або мережевого з'єднання, LabVIEW створює посилання на об'єкт. Усі операції з відкритими об'єктами виконуються за допомогою посилань.

Кластер помилок та посилання на файл послідовно передаються від вузла до вузла. Оскільки вузол не може виконуватись, доки не визначені всі його вхідні поля даних, то ці два параметри змушують вузли працювати у певному порядку.

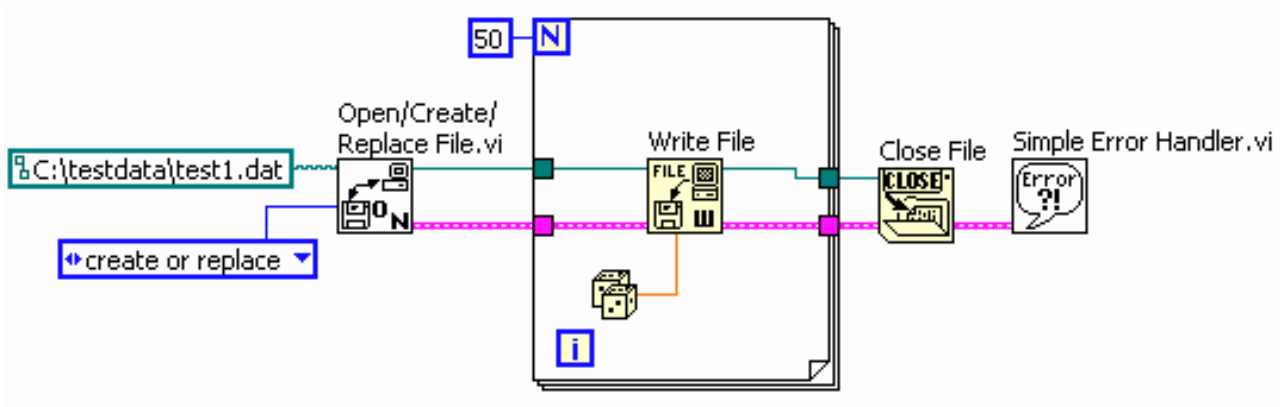


Рисунок 5.11 – Приклад запису строки даних у файл

Підпрограма **ВП Open/Create/Replace File VI** передає посилання на файл і кластер помилок функції **Write File**, яка здійснює запис файла на диск. Функція **Close File** закриває файл після отримання кластера помилок та посилання на файл із функції **Write File**.

Підпрограма **ВП Simple Error Handler VI** перевіряє наявність помилок та виводить інформацію про них у діалоговому вікні. Якщо в одному з вузлів допущено помилку, наступні вузли не виконуються, і кластер помилок передається до підпрограми **ВП Simple Error Handler VI**.

В якості прикладу запису даних у файл розглянемо запис осцилограми LabVIEW.

Для запису осцилограми даних у файл використовуються підпрограми **ВП Waveform File I/O Vis**, розміщені на палітрі **Functions** у розділі:

Functions → Waveform → Waveform File I/O.

Для запису даних у двійковий файл використовуються **ВП** запису та читання осцилограми:

- **Write Waveforms to File;**

- **Read Waveforms from File Vis**.

Під час використання **ВП Export Waveforms to Spreadsheet File VI** об'єкт осцилограми зберігається у файл у вигляді таблиці символів.

Цей **ВП** подібний до високорівневого **ВП Write to Spreadsheet File**. Він відкриває файл даних під іменем, визначеним у полі **file path**, або відкриває діалогове вікно, якщо це поле не визначено. Об'єкт осцилограми даних під'єднується безпосередньо до входу цього **ВП** і конвертує осцилограми даних у таблицю символів, використовуючи заданий у полі **Tab** роздільник.

Можна відкрити наявний файл або створити новий, також можна додати заголовок до файлу даних або записати колонку часових міток. Після запису даних у файл він закривається. Поле **error** виводить інформацію про помилки.

Якщо передбачається використовувати осцилограму тільки всередині ВП, то доцільно зберегти її у вигляді файлу протоколу (*.log).

На рисунку 5.12 приведений приклад побудови **ВП**, що вводить кілька осцилограм, відображає їх на графіку і потім записує у файл протоколу даних.

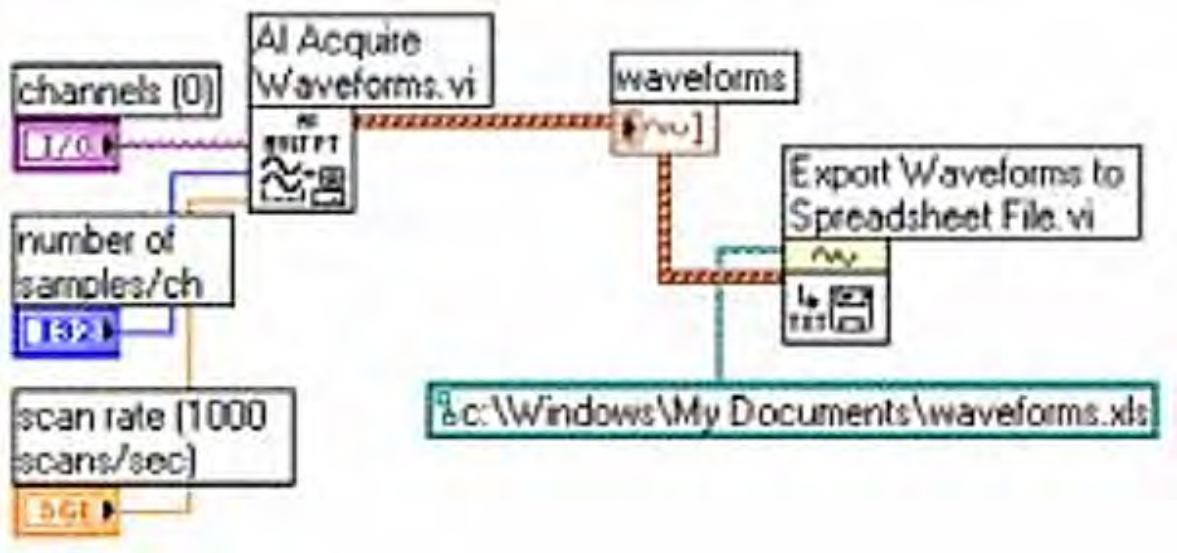


Рисунок 5.12 – Приклад запису декількох осцилограм у файл протоколу даних

Для зчитування осцилограм із файлу даних використовується **ВП Read Waveform from File**. Після того, як одну осцилограму буде зчитано, можна додавати або змінювати компоненти даних типу **waveform** (осцилограма) за допомогою функції **Build Waveform**, або вилучати компоненти осцилограми за допомогою функції **Get Waveform Attribute**.

ВП, наведений на рисунку 5.13, зчитує осцилограму з файлу, змінює компоненту осцилограми **dt** і відображає змінену осцилограму на графіку.

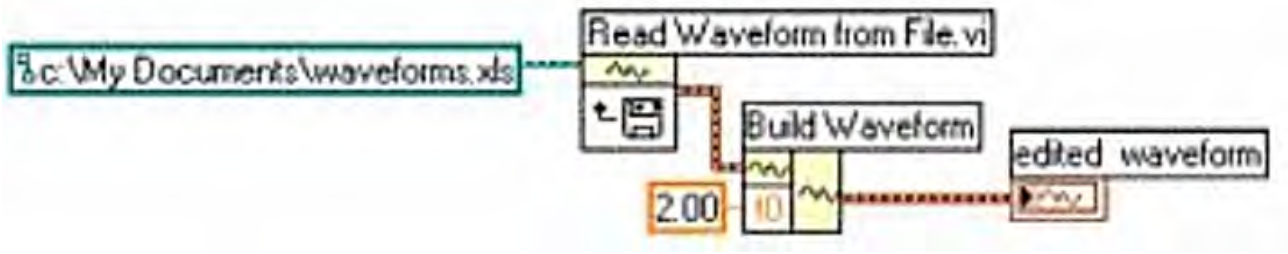


Рисунок 5.13 – Приклад зчитування осцилограми з файлу

ВП Read Waveform from File може також зчитувати з файлу кілька осцилограм. У такому разі ВП повертає масив осцилограм, який може відображатися у вигляді кількох кривих на одному графіку.

На рисунку 5.14 наведений **ВП**, у якому організований доступ до файлу, що містить кілька осцилограм. Функція **Index Array** читає першу і третю осцилограму з файлу і будує їхні криві на окремих індикаторах Waveform Graph.

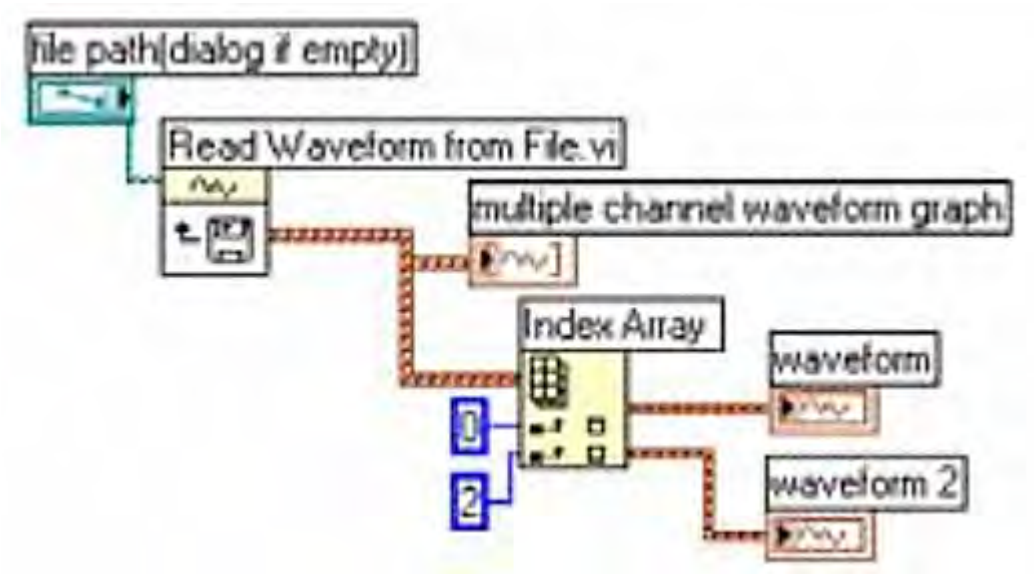


Рисунок 5.14 – Приклад зчитування кількох осцилограм з файлу

5.3 Контрольні питання

5.3.1 Які операції зі строками можна виконувати за допомогою?

5.3.2 Які типи відображення присутні в LabVIEW?

5.3.3 Які функції роботи зі строковими даними досліджувались? Що ці функції можуть роботи зі строковими даними?

5.3.4 Яка послідовність дій при роботі з файлами в LabVIEW? Перелічить функції палітри File I/O, які дозволять виконати цю послідновність?

5.3.5 Назвіть особливості організації шляху до файлів.

5.3.6 Які особливості запису/зчитування осцилограм до/з файлу?

6 ЗАСТОСУВАННЯ ПОСЛІДОВНОГО ПОРТУ В LabVIEW ДЛЯ ОБМІНУ ДАНИМИ

6.1 Загальні відомості про послідовний порт

Програмний комплекс LabVIEW дозволяє отримувати та передавати оброблені дані до пристроїв з'єднаних за допомогою послідновного порта (COM-порта).

Послідовний порт – двонаправлений послідовний інтерфейс для обміну байтовою інформацією. Перадача інформації через даний порт відбувається побітово, тому він називається послідовним. Прикладами послідовного порта є більш відомі інтерфейси RS-232, RS-422, RS-485, UART, Ethernet, Firewire, USB та інші.

Важливою особливістю інтерфейсної системи послідовного порту є те, що вона може передавати дані в послідовному режимі від пристрою, що видає дані по паралельній шині. Взаємне перетворення послідовних та паралельних форматів даних виконується за допомогою регістрів зсуву, що мають функцію паралельного доступу.

Даний стандарт обміну даними широко застосовується в різних інформаційно-вимірювальних системах, різноманітних системах авіаційної та ракетно-космічної техніки, зокрема в системах керування. Застосування COM-порту в системах керування літальними апаратами та комплексами можна пояснити потребою в обміні інформацією між окремими елементами систем, наприклад на основі даних, що надходять з датчиків, необхідно формувати команди керування на виконавчі пристрої.

6.2 Бібліотеки LabVIEW для роботи з послідовним портом

Для розробки віртуальних приладів з послідовним портом в програмному комплексі LabVIEW передбачено бібліотеки **VISA** та **Serial Port**.

VISA – це API (Application Programming Interface), який надає інтерфейс програмування для керування приладами Ethernet/LXI, GPIB, послідовними портами, USB, PXI та VXI у середовищах розробки додатків NI, таких як LabVIEW, LabWindows/CVI та Measurement Studio. API інсталюється через драйвер NI-VISA.

NI-VISA використовує багато однакових операцій для зв'язку з інструментами, незалежно від типу інтерфейсу. Це спрощує перемикання інтерфейсів і забезпечує застосування одних функцій для багатьох різних інструментів. Наприклад, команда VISA для запису рядка ASCII в інструмент на основі повідомлень є однаковою незалежно від того, чи є інтерфейс приладу послідовним, GPIB або USB.

Загальний вигляд бібліотеки VISA показаний на рисунку 6.1.

Serial Port – бібліотека зі стандартним функціями, які потрібні для розробки віртуальних приладів при налаштування обміну по послідовному порту. Такими функціями є налаштування порта (**Configure Port**), запис данних (**Write**), зчитування даних (**Read**), закриття порту (**Close**), зчитування буфера даних (**Bytes at Port**), зупинка (**Break**), встановлння розміру буфера (**Set Buffer Size**), очищення буфера (**Flush Buffer**).

Загальний вигляд бібліотеки Serial Port (Serial) показаний на рисунку 6.2.

Функціонал бібліотек VISA та Serial Port достатній для обміну даними між LabVIEW і пристроями з послідовним портом.

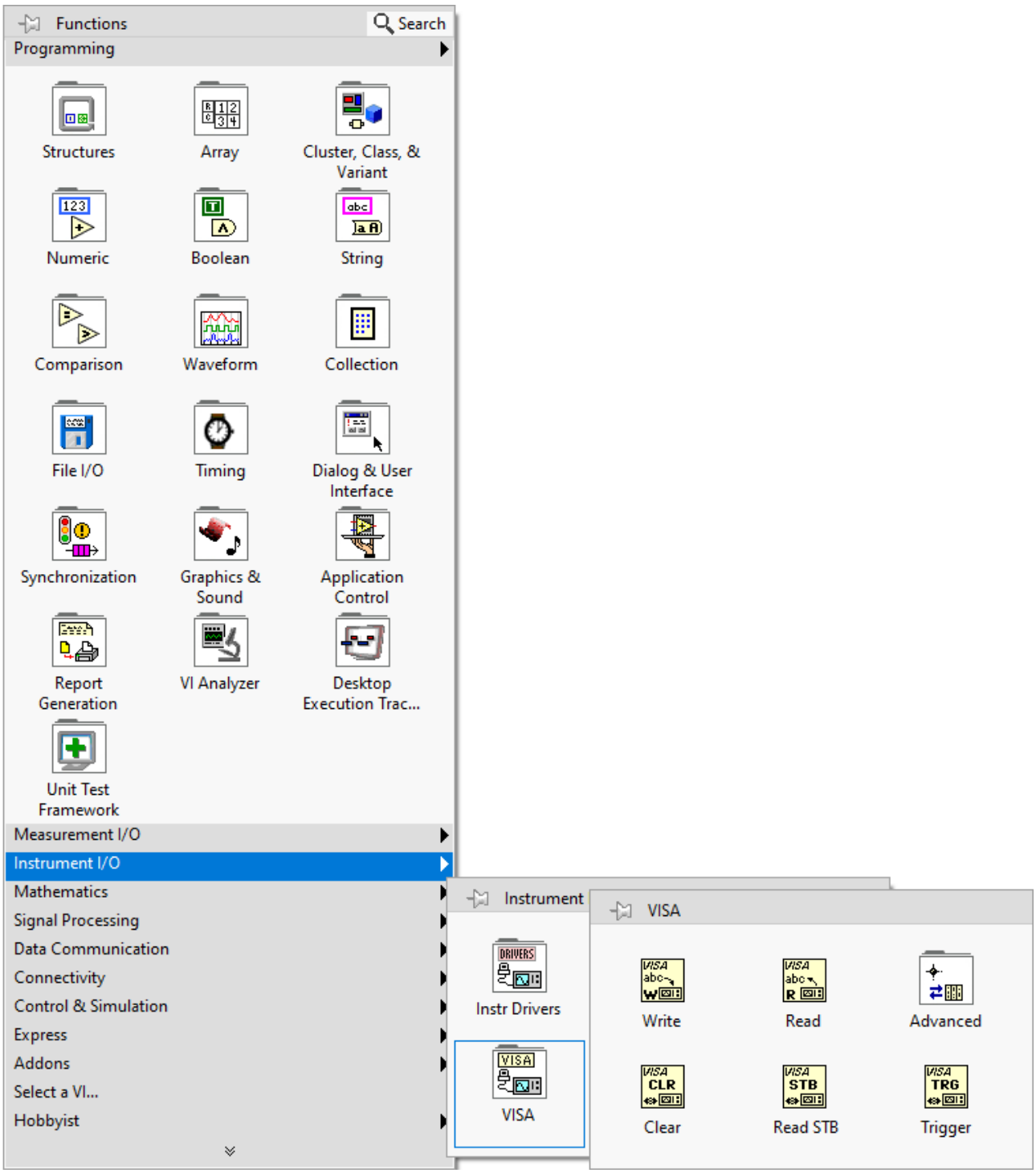


Рисунок 6.1 – Бібліотека VISA

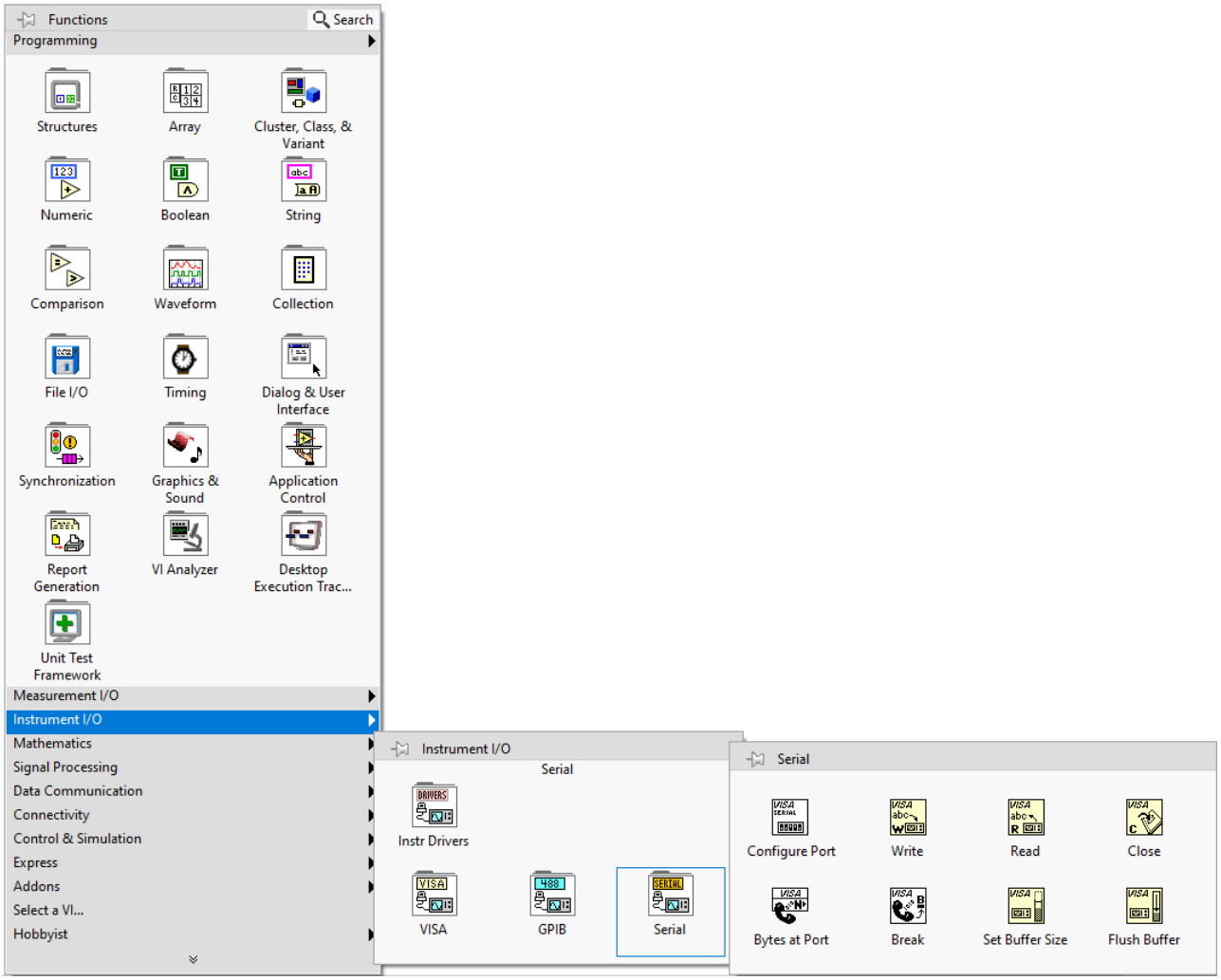


Рисунок 6.2 – Бібліотека Serial Port

Для використання послідовного зв'язку потрібне завдання чотирьох параметрів:

- швидкість передавання;

- кількість бітів (**data bits**) для шифрування переданого символу;

- наявність біт-парності (**parity bit**);

- кількість стоп-бітів (**stop bits**).

Кожен переданий символ упаковується в кадр символу, який складається з одиночного стартового біта (**start bit**), за ним ідуть біти символу (**data bits**), біт

парності (**parity bit**), якщо встановлений, і задана кількість стопових бітів (**stop bit or bits**).

Розглянемо більш детально функції бібліотеки VISA для підключення по послідовному порту:

- **VISA Configure Serial Port VI** ініціалізує порт, ідентифікований ім'ям ресурсу VISA (**VISA resource name**), відповідно до заданих налаштувань;

- **Timeout** встановлює значення часу очікування під час обміну даними через послідовний порт;

- **Baud rate** - встановлює швидкість передачі даних;

- **data bits** (біти даних), **parity** (контроль парності) і **flow control** (керування потоком даних) задають відповідні параметри для послідовного порту;

- **error in** і **error out** - відповідно вхідний і вихідний кластери помилок, що містять інформацію про умови появи помилок у **ВП**.

На рисунку 6.3 показано, як надіслати команду запиту ідентифікації «***IDN?**» приладу, під'єднаному до послідовного порту COM2.

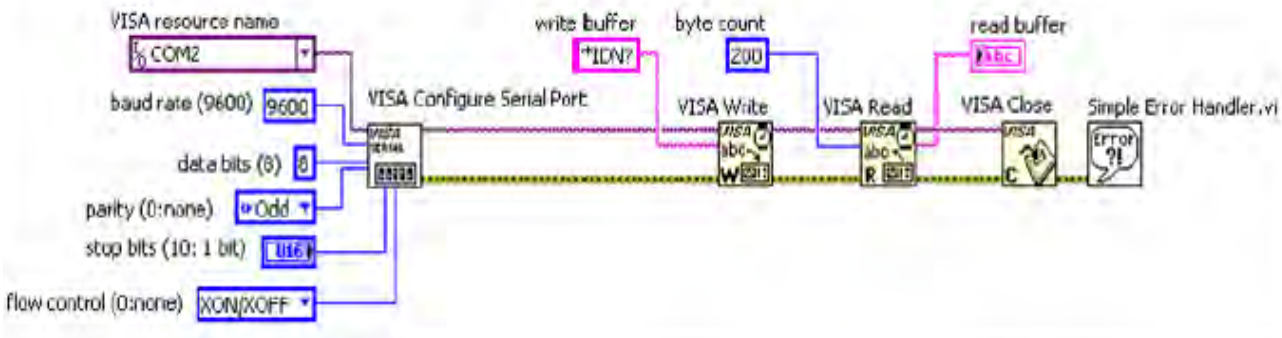


Рисунок 6.3 – Приклад налаштування послідовного порту для VISA

VISA Configure Serial Port VI відкриває зв'язок із портом COM2 і встановлює такі налаштування: 9600 бод, 8 біт даних, контроль непарності, 1 стоп біт і програмне квітування XON/XOFF. Потім функція **VISA Write** надсилає

команду. Функція **VISA Read** зчитує до 200 байт у буфер читання, а **Simple Error Handler VI** перевіряє умови виникнення помилки.

6.3 Застосування послідовного порту LabVIEW для прийому даних з гіроскопа та акселерометра

Розглянемо приклад побудови віртуального приладу LabVIEW для прийому даних з зовнішніх датчиків по послідовному порту.

В система керування літальними апаратами найбільш важливими є датчики інерціальної навігаційної системи.

Тому в якості зовнішніх датчиків будемо використовувати датчики компанії STMicroelectronics, а саме: трьохосьовий гіроскоп L3GD20 та трьохосьовий акселерометр LSM303DLHC.

Функціонально ці датчики розміщені на платі STM32F3 Discovery (рисунок 6.4).

З'єднання плати STM32F3 Discovery з комп'ютером здійснюється через порт USB. Для програмного обміну з LabVIEW використовується віртуальний COM-порт (VCP – Virtual COM Port). Для обміну даними через послідовний порт VCP додаємо на блок-діаграму **ВП** LabVIEW наступні елементи:

- **VISA Configure Port** (бібліотека **Serial** палітри **Instrument I/O**);
- **Bytes at Port** (бібліотека **Serial** палітри **Instrument I/O**);
- **Read** (бібліотека **Serial** палітри **Instrument I/O**);
- **Close** (бібліотека **Serial** палітри **Instrument I/O**);
- **String Indicator** (бібліотека **String** на фронтальній панелі);
- **VISA Resource name** (бібліотека **I/O** на фронтальній панелі);

- **Simple Error Handler** (бібліотека **Dialog and User Interface**).

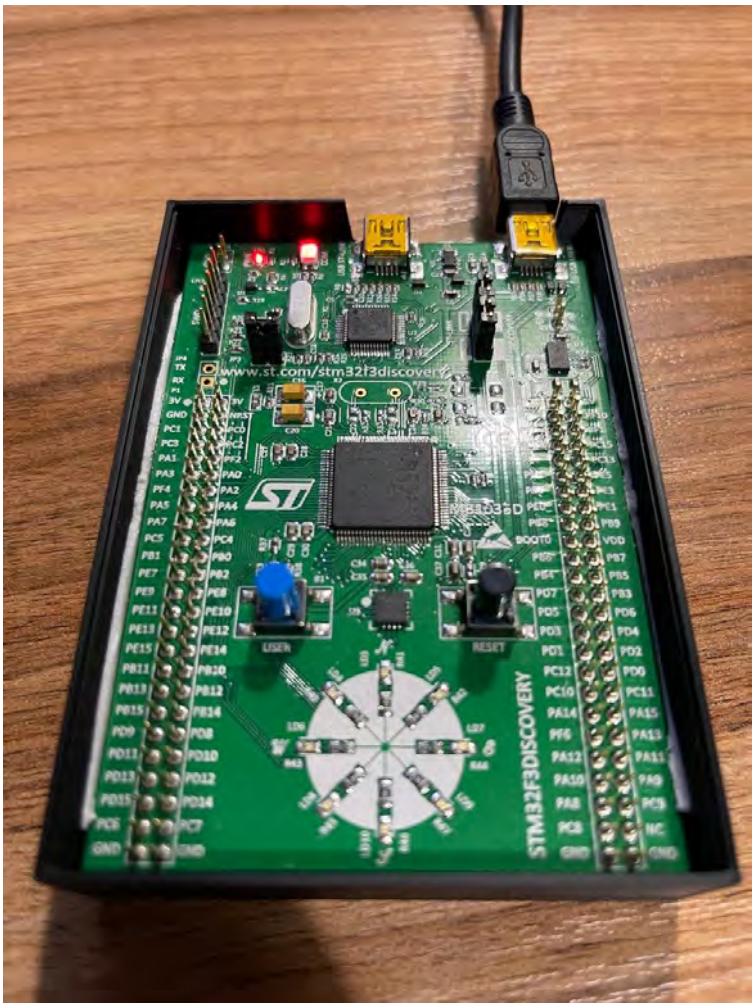


Рисунок 6.4 – Плата STM32F3 Discovery з датчиками

Для побудови схеми необхідно провести з’єднання елементів блок-схеми, а саме:

1) **VISA Resource Name** приєднується до відповідного входу **VISA Configure Port**;

2) **VISA Resource Name Out** приєднується до входу **Reference** блоку **Bytes at Port**;

3) вихід **Reference Out** блоку **Bytes at Port** приєднується до **VISA Resource Name** входу на блоці **Read**;

4) вихід **error out** блоку **Bytes at Port** приєднується до **error in** блоку **Read**;

5) вихід **Number of bytes at Serial Port** блоку **Bytes at Port** приєднується до **bytes count** блоку **Read**;

6) вихід **VISA Resource Name Out** блоку **Read** приєднується до **VISA Resource Name** блоку **Close**;

7) вихід **error out** блоку **Read** приєднується до **error in** блоку **Close**;

8) вихід **read buffer** з'єднуємо з **String Indicator**.

З'єднання елементів блок-схеми показано на блок-діаграмі програми на рисунку 6.5.

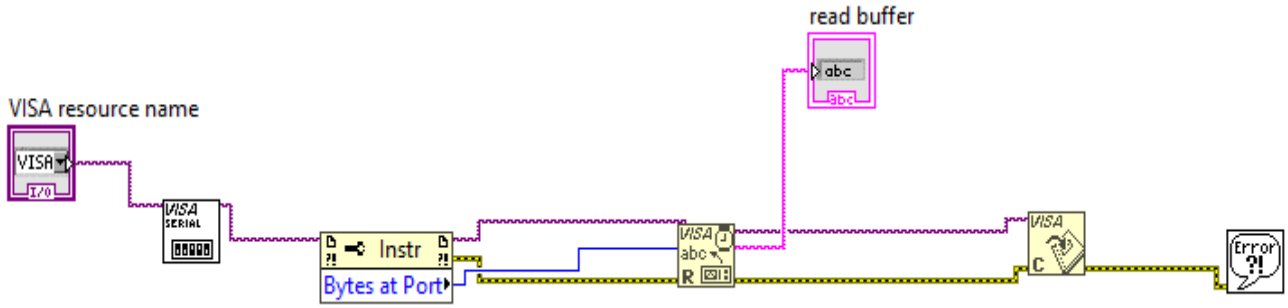


Рисунок 6.5 – Блок-діаграма отримання даних з послідовного порту

Перед запуском **ВП** на фронтальній панелі у випадаючому списку **VISA Resource Name** потрібно обрати відповідний COM-порт. Зазвичай у списку відображаються тільки активні порти. В даному випадку у списку наявний лише один порт, до якого приєднана плата з датчиками. Результат запуску програми наведено на рисунку 6.6.

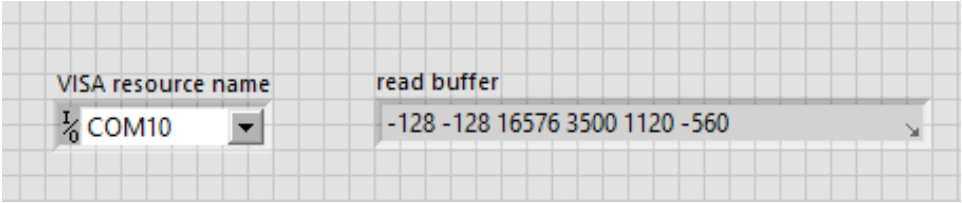


Рисунок 6.6 – Результат отримання даних з COM порта

Для безперервного роботи схеми і отримання даних з датчиків треба додати цикли до розробленої схеми. Для цього блоки **Bytes at Port**, **Read** і **String Indicator** додаємо в блок циклу **While Loop**. Умовою виходу з циклу є або помилка при прийомі даних, або натискання кнопки **Stop**, яку додаємо окремо на фронтальній панелі. Виходи **error out** та кнопки **Stop** подаємо на вхід блоку порівняння **Or** (бібліотека **Boolean**). Вихід **Or** з'єднується з умовою виходу з циклу.

Блоки **Read**, та **String Indicator** вносяться в структуру **Case Structure**, для перевірки умови наявності даних в послідовному порті.

Для перевірки наявності даних в послідовному порті на логічну умову входу **Case Structure**, подається перевірка кількості байтів блоком **Greater Than 0** (бібліотека **Comparison**).

Блок-діаграма безперервного отримання даних від датчиків по послідовному порту показана на рисунку 6.7.

Результат роботи програми виводяться на передню панель **ВП**, як показано на рисунку 6.8.

Для візуалізації масиву отриманих даних потрібно попередньо його розбити на частини та перетворити зі строкового типу даних (**String**) в чисельний формат (**Double**).

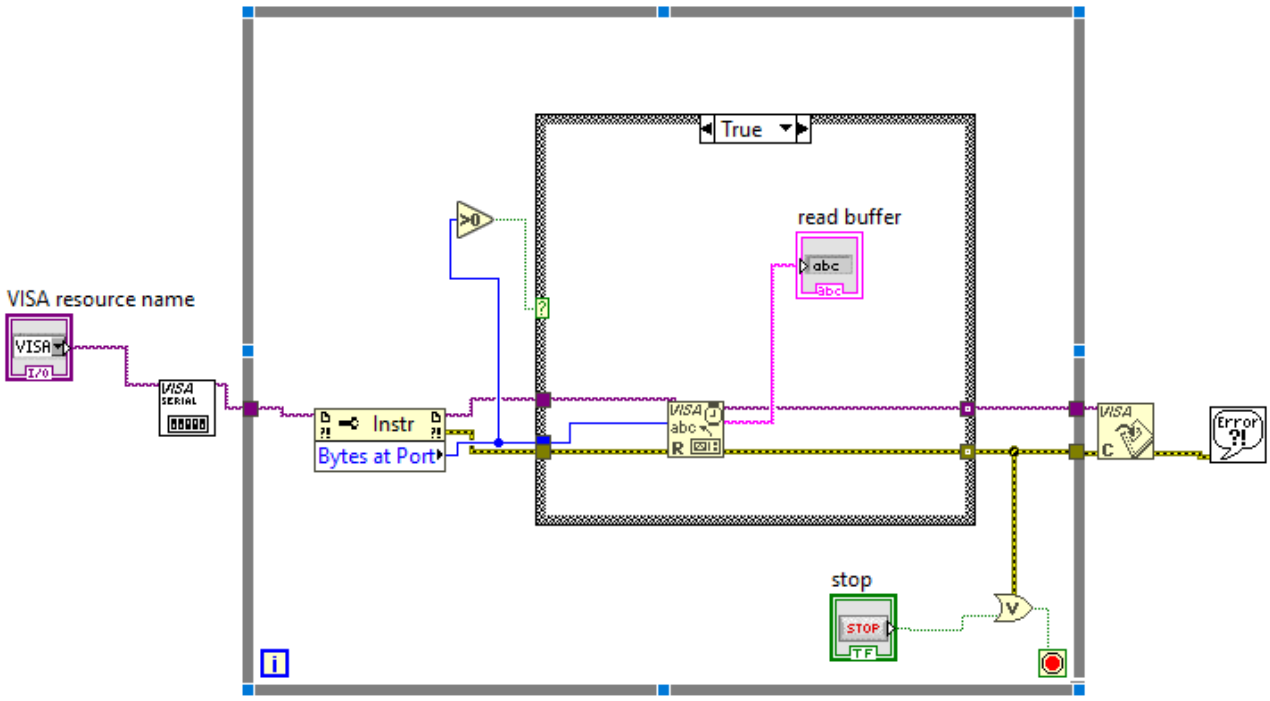


Рисунок 6.7 - Блок діаграма безперервного отримання даних по COM-порту

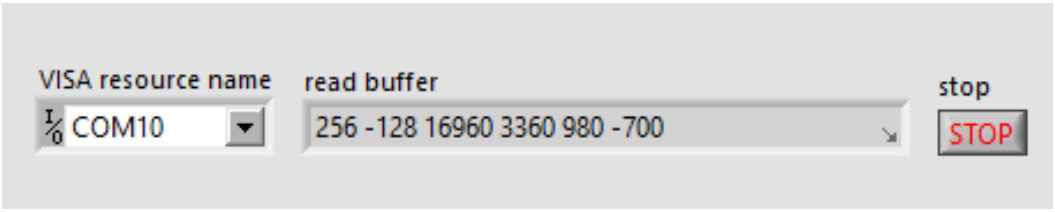


Рисунок 6.8 – Вивід даних з COM-порту у безперервному режимі роботи

Для цього спочатку відокремимо всі числа. Найпростіше це роботи послідовно за допомогою блока **Search/Split String**. На вхід **string** подаємо отриманий масив даних з COM порту. На вхід **search string/char**, подаємо символ відокремлення, в даному випадку таким символом є пробіл (**space**), який можна додати як константу **Space Constant** (бібліотека **String**).

Вихід **substring before match** віддає відокремлену частину масиву до пробілу. У даному випадку це буде перше число, що є даними по одній з осей

акселерометра. Ці строкові дані подаємо на перетворювач строкових даних в числові **Fract/Exp String To Number**.

Вихід **match+rest of string** блоку **Search/Split String**, подаємо знову на інший блок **Search/Split String**. Блоків **Search/Split String** і **Fract/Exp String To Number** повинно бути **6**, оскільки дані отримуємо по трьох осях акселерометра та трьох осях гіроскопа. Остаточний вигляд блок діаграми **ВП** показаний на рисунку 6.9.

Після перетворення даних в числовий формат, для зручного відображення сформуємо кластер за допомогою блоку **Bundle** (бібліотека **Cluster, Class & Variant**). Сигнали акселерометра (перших три цифри в рядку буфера) подаємо на один **Bundle**, останніх три - на другий. Для відображення даних на виході блоків **Bundle** додаємо графік діаграм **Waveform Chart** (рисунок 6.8).

Результати роботи розробленого віртуального приладу показано на рисунку 6.10 у вигляді графіків вихідних сигналів трьохосьового акселерометра та трьохосьового гіроскопа.

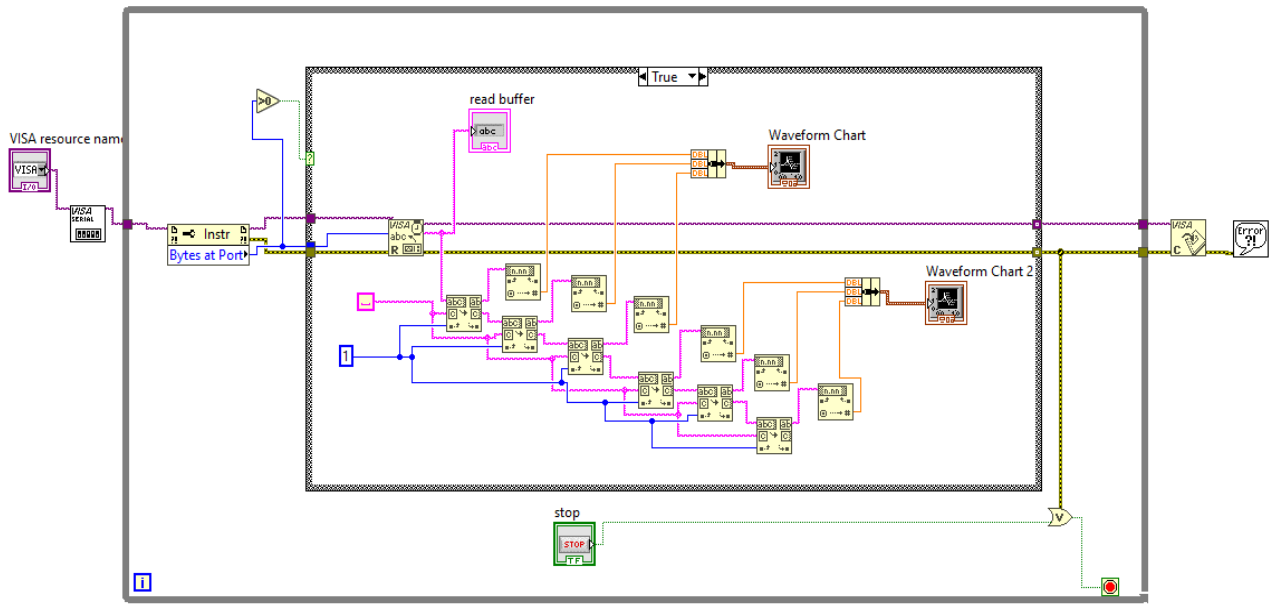


Рисунок 6.9 – Блок діаграма віртуального приладу прийому даних з COM-порту

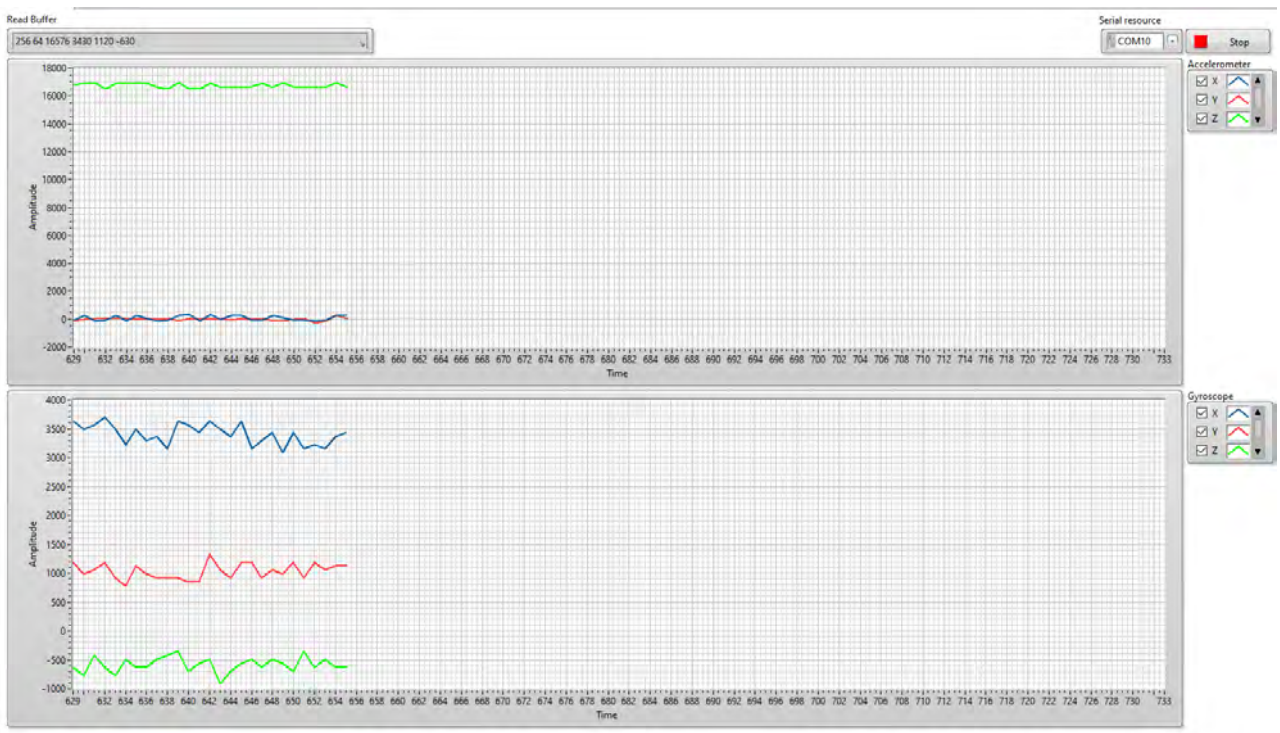


Рисунок 6.10 – Результат роботи **ВП** при прийомі по COM-порту сигналів акселерометрів і гіроскопів

Розроблений віртуальний прилад дозволяє отримувати дані з датчиків (гіроскопів та акселерометрів) по послідовного порту та відображати їх як у числовому, так і графічному вигляді. Отримані дані можуть піддаватись наступній обробці за потрібними алгоритмами, зокрема для вирішення задач орієнтації та навігації. Розроблене програмне забезпечення може бути легко адаптовано до іншого обладнання, що має інтерфейс для передачі даних по COM-порту.

6.4 Контрольні питання

6.3.1 З якими послідовними портами може працювати LabVIEW?

6.3.2 Які типи бібліотеки присутні в LabVIEW для роботи з послідовними портами?

6.3.3 Які функції потрібні для реалізації прийому даних по COM порту?

6.3.4 Яким чином можна відображувати отримані дані по послідовному порту?

6.3.5 Як організувати безперервний прийом даних по COM порту?

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

Базова

1. Івашко В.В. Конспект лекцій з навчальної дисципліни «Програмне забезпечення інформаційно-вимірювальних систем»: Основи LabVIEW. Керування приладами в LabVIEW. - Чернівці: Чернівецький нац. ун-т ім. Юрія Федьковича, 2021. – 80 с.

2 Власенко В.М., Савченко, В.І. Вивчення перехідних процесів в електричних колах з використанням комп'ютерної технології LabVIEW / В.М. Власенко, В.І. Савченко // Вісн. Черкаського нац. ун-ту ім. Богдана Хмельницького. – 2016 - Серія: Педагогічні науки 20 (353). - С. 97-101.

3. Комп'ютерне проєктування систем авіоніки: Лабораторний практикум [Електронний ресурс]: навч. посіб. для здобувачів ступеня бакалавра спеціальності 173 «Авіоніка», за освітньою програмою «Системи керування літальними апаратами та комплексами» / КПІ ім. Ігоря Сікорського; уклад.: Ю. В. Бобков, А. А. Сердюк. – Електронні текстові дані (1 файл: 3,63 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2022. – 97 с.

Допоміжна

1 Peter A. Blume. The LabVIEW Style Book – Boston: PRENTICE HALL, 2007 – 396 pages.

2 Jeffry Travis, Jim Kring. LabVIEW for Everyone: Graphical Programming Made Easy and Fun. – Boston: PRENTICE HALL, 2016 – 1032 pages.

3 LabVIEW osnove vol.1.- Austin, Texas: National Instruments Corporate, 2009 - 273 pages.