Міністерство освіти і науки України
Карпатський національний університет імені Василя Стефаника

Н.В. Превисокова, В.А. Ровінський

Технології цифрової обробки сигналів та програмування систем з використанням C++, C# і бібліотеки Intel IPP

Посібник-довідник

Івано-Франківськ
2025

УДК 004.4
ББК 32.973.2 – 018
П-58

Рекомендовано Вченою радою факультету математики та інформатики Карпатського національного університету імені Василя Стефаника як навчальний посібник для студентів спеціальностей "Комп'ютерні науки" та "Інформаційні системи та технології" підготовки (протокол № 9 від 22 жовтня 2025р.).

Рецензенти:
Горєлов В.О., кандидат технічних наук, доцент (КНУВС ім.В.Стефаника)
Стрілецький Ю. Й., доктор технічних наук, професор (ІФНТУНГ)

П-58 Превисокова Н.В., Ровінський В.А.Технології цифрової обробки сигналів та програмування систем з використанням C++, C# і бібліотеки Intel IPP. Посібник – довідник. Івано-Франківськ, 2025. 176 с.

Посібник призначений для вивчення функцій бібліотеки оптимізації Intel IPP технологій і методів обробки сигналів з використанням бібліотеки Intel IPP на основі мов C# і C++. Основна увага приділяється аспектам застосування бібліотеки для цифрової обробки сигналів та побудови спеціалізованого програмного забезпечення максимально можливої швидкодії. Проведений огляд функцій IPP, розглянуті основні арифметичні, логічні, статистичні функції, функції дискретних перетворень Фур'є та вейвлет-перетворень, фільтрації сигналів. Матеріал представлений у формі довідника функцій з короткими прикладами програмної реалізації на Intel IPP , C++, C#.

Для студентів технічних спеціальностей, які вивчають дисципліни «Перетворення форми інформації та цифрова обробка інформації», «Цифрова обробка інформації», «Програмування систем цифрової обробки інформації», «Методи і алгоритми обробки зображень».

УДК 004.4
ББК 32.973.2 – 018.

ЗМІСТ

ПЕРЕДМОВА

Посібник-довідник написано на основі досвіду викладання курсів «Програмування систем цифрової обробки інформації» та «Цифрова обробка інформації» на факультеті математики та інформатики Карпатського національного університету імені Василя Стефаника для студентів спеціальностей «Комп'ютерні науки» та «Інформаційні системи та технології».

Посібник-довідник є першою спробою створення навчально-довідкової книги з проблем реалізації технологій обробки інформації та побудови інформаційних систем засобами бібліотеки Intel IPP. Він побудований за функціональний принципом і охоплює всі основні складові частини реалізації технології цифрової обробки сигналів. У кожному розділі після теоретичного матеріалу подано приклади програмної реалізації процедур обробки інформаційних потоків,

Посібник призначений для вивчення бібліотеки Intel IPP на основі мови C++. Основна увага приділяється спеціальним аспектам застосування функцій для обробки цифрової сигналів та побудови спеціалізованого програмного забезпечення максимально можливої швидкодії. Проведений огляд ієрархії функцій IPP, розглянуті основні арифметичні, логічні, статистичні функції, функції дискретних перетворень Фур'є, Гільберта та вейвлет-перетворення. Розглянуті також функції фільтрації сигналів. Матеріал представлений у вигляді довідника функцій з короткими прикладами практичного використання.

Короткий, довідниковий стиль викладу, орієнтований на розгляді прикладів дозволяє використовувати навчальний посібник самостійно при підготовці до занять, та при самостійному вивченні курсу.

1. БІБЛІОТЕКА INTEL IPP

1.1. Бібліотека Intel® Integrated Performance Primitives (IPP) та її застосування для цифрової обробки сигналів

Цифрова обробка сигналів (Digital Signal Processing, DSP) є фундаментальною технологією в сучасній електроніці та комп'ютерних системах. Вона полягає в маніпуляції дискретними сигналами за допомогою математичних алгоритмів для покращення якості, витягнення інформації, видалення шумів чи стиснення даних. DSP перетворює аналогові сигнали (наприклад, звук чи зображення) на цифрові, дозволяючи обробляти їх на комп'ютерах чи спеціалізованих процесорах. Ця технологія знайшла широке застосування в багатьох галузях, від розваг до медицини, завдяки своїй ефективності та гнучкості.

Одним з популярних варіантів використання сучасних ЕОМ є виконання мультимедійних додатків. Кодування і декодування відео- і аудіо, обробка зображень і сигналів часто вимагає значного обсягу обчислювальних ресурсів, що може накладати певні обмеження на використовуване програмне забезпечення. Іншим важливим фактором, що вимагає ефективного використання обчислювальних ресурсів, є час автономної роботи мобільного пристрою (ноутбука, планшетного комп'ютера, смартфона).

Необхідно відзначити, що багато високорівневі алгоритми обробки мультимедіа даних в результаті зводяться до базових операцій над простими структурами (наприклад, масивами), причому продуктивність цих операцій істотно впливає на продуктивність алгоритму в цілому. Деякі з базових операцій можуть використовуватися різними високорівневими алгоритмами. Виходячи з цих факторів, за доцільне виконувати оптимізацію саме низькорівневих функцій. Даний підхід лежить в основі розробленої компанією Intel бібліотеки Integrated Performance Primitives [1], Яка являє собою набір кроссплатформенних бібліотек, що містять велику кількість низькорівневих оптимізованих під Intel CPU функцій, які можуть бути використані при розробці різних мультимедійних додатків таких як:

- Відео-кодеки: H.264, MPEG2, MPEG4, VC-1
- Аудіо-кодеки: AAC, AC3, MP3
- Стиснення зображень (кодеки JPEG, JPEG2000, JPEGXR)
- Обробка зображень
- Обробка сигналів
- Стиснення природної мови (кодеки AMR, AMR-WBE, G.711, G.722, G.723. G.726, G.728, G.729, GSM-AMR, GSM-MFR)
- криптографічні додатки

Intel IPP - це набір крос-платформних бібліотек, що містять велику кількість високопродуктивних функцій, які можуть бути використані для аудіо-, відео-кодеків (наприклад, H.263, MPEG-4), стиснення зображень (JPEG і JPEG2000), обробки зображень (двовимірних масивів), обробки сигналів (одновимірних масивів або векторів), стиснення природному мовленні, систем комп'ютерного зору, криптографічних додатків, а також допоміжні математичні функції. Необхідність створення такого набору функцій безпосередньо випливає з існуючих реалій розробки прикладного програмного забезпечення:

- існує певний набір обчислювально складних функцій, які потрібні для додатків;
- розробники змушені виробляти ретельну оптимізацію цих функцій для отримання необхідної продуктивності;
- процес оптимізації складний і займає тривалий час.

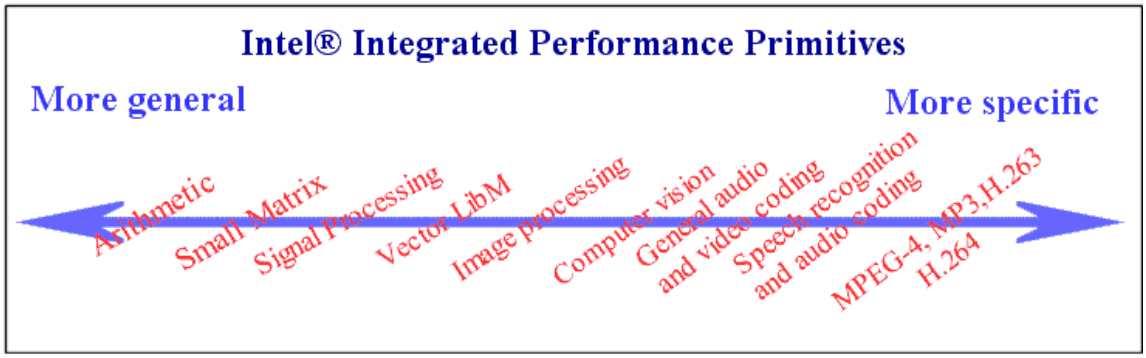
Дійсно, багатьом розробникам потрібно реалізувати одну і ту ж функціональність. А з ростом кількості апаратних платформ частка низькорівневих (ассемблерних) реалізацій зменшилася значно. Intel IPP містить набір низькорівневих процедур, які дозволяють ефективно виконуватися мультимедійним інструкціям на будь-якому процесорі.

1.2. Структура Intel IPP

Примітиви в Intel IPP можуть бути розділені на три основні групи за типом оброблюваних даних:

- "Сигнали" (термін умовний і використовується для позначення одновимірних даних) (IPPS);
- "Зображення" (двовимірні масиви колірних даних) (IPPI);
- матриці невеликий розмірності (масиви NxM) (IPPM).

Існує також логічний підрозділ, що відповідає за криптографічні операції (IPPCP) і структурно входить в перший домен. Незалежно від того, в який домен входить функція, вона може бути класифікована за ступенем специфічності:



Бібліотека Intel IPP 4.0 підтримує загальний API-інтерфейс і забезпечує можливість створення більш високопродуктивних** додатків для будь-яких

платформ на базі процесорів Intel - для настільних ПК, серверів, мобільних і кишенькових ПК.

1.3. Структура бібліотеки і модель використання

Бібліотека Intel IPP являє собою набір оптимізованих реалізацій базових (низькорівневих) операцій (дані реалізації поставляються в бінарному вигляді). Логічно бібліотеку можна розділити на 3 великих модуля: обробка сигналів (одновимірних масивів), обробка зображень і відео, операції над матрицями невеликого розміру і реалістичний рендеринг. Для того щоб полегшити процес розробки ПЗ з використанням бібліотеки, окремо у вигляді вихідних кодів поставляються високорівневі API для аудіо та відеокодеків, функцій обробки зображень і мови. Дані API можуть бути використані при розробці власних програм.

Intel® Integrated Performance Primitives (Intel® IPP) [5] - бібліотека високопродуктивних інструментів і програмних функцій обробки даних. Функції бібліотеки можна розділити на чотири основні групи:

Функції обробки звукових сигналів:

- дискретне перетворення Фур'є;
- функції фільтрації:
- деякі функції розпізнавання і кодування мови;
- функції стиснення даних і ін.

Функції обробки зображень і відео:

- функції конвертації зображень з одного колірного простору в інший;
- операції відсікання;
- морфологічні операції;
- лінійні перетворення;
- статистичні функції (гістограма, центральні моменти, сума, інтеграли, математичне очікування і середньоквадратичне відхилення набору пікселів);
- геометричні перетворення зображень з використанням різних методів інтерполяції;
- вейвлет-перетворення;
- реалізації деяких алгоритмів комп'ютерного зору (детектор ребер Канни, перетворення Хафа, обчислення оптичного потоку і т.п.);
- функції стиснення зображень;
- функції кодування відео і ін.

Функції роботи з невеликими з векторами і матрицями невеликих розмірів:

- операції над векторами (додавання, віднімання, множення і т.п.);

- операції з матрицями (транспонування, обчислення визначника, множення, визначення сліду матриці і т.п.);
- функції рішення систем лінійних алгебраїчних рівнянь і ін.

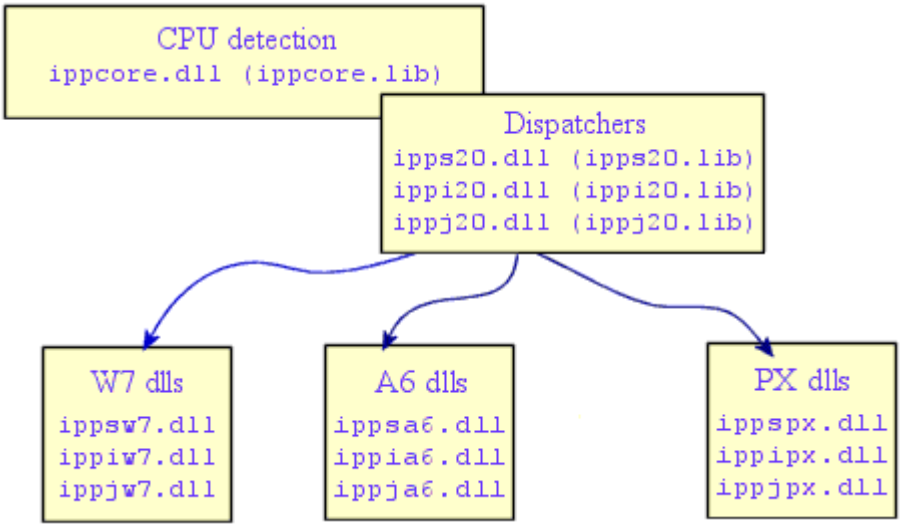
Функції шифрування.

1.4. Принципи роботи і приклади використання Intel IPP

Кожна функція з Intel IPP має кілька варіантів, оптимізованих під різні архітектури процесорів.

Існує чотири способи включення функціональності Intel IPP в додаток:

- динамічна лінковка;
- статична лінковка;
- змішана лінковка (просунута статична);
- динамічна лінковка з додатковими можливостями.



Для ефективного виконання на будь-якому апаратному забезпеченні доцільно використовувати динамічна лінковку. У цьому випадку програма після запуску визначає процесор і підставляє в місця виклику функцій бінарний код з відповідною бібліотеки:

При використанні статичної лінковки досягається менший розмір програми, але ціною жорсткої прив'язки до типу процесора. Змішана лінковка, по суті, є статичною, але при цьому дозволяє зменшити розмір файлу.

1.5. Типи даних та іменування функцій

Іменування функцій

Правила іменування функцій Intel IPP однакові для всіх охоплених доменів. Імена функцій в бібліотеці Intel IPP мають наступну структуру:

ipp <data-domain> <name> | _ <datatype> || _ <descriptor> | (<Параметри>)

Розглянемо докладніше призначення окремих частин в даній структурі.

<Data-domain>- тип оброблюваних даних:

s(Signal) - одномірний масив

i(Image, video) - двомірний масив пікселів

m(Matrice) - матриця або вектор

r - вихідні дані для функцій реалістичного рендеринга зображень і обробки 3D даних (представлення даних залежить від використовуваної функції)

Приклади: ***ipps, ippi, ippm, ippr***

<Name> = <operation> [_ modifier] - ім'я функції:

<Operation>- символічну назву виконуваної операції

modifier (Додатковий параметр) - додаткова специфікація функції

Приклади: ***Copy, Malloc, DCTFwd_CToC*** (пряме дискретне косинус-перетворення, в якому вхідні і вихідні значення є комплексними)

Типи даних

Поле ***тип даних datatype*** вказує типи даних, що використовуються функцією, у наступному форматі:

<Datatype> = <bit_depth> <bit_interpretation> - тип оброблюваних даних:

<Bit_depth>= 1 | 8 | 16 | 32 | 64 (розмір елемента даних (в бітах))

<Bit_interpretation>= <U | s | f> [c]

u - цілі числа без знака

s - цілі числа зі знаком

f - дійсні числа

c - комплексні числа

ПРИМІТКА

У списках параметрів функції, ***Ipp*** префікс додається до типу даних. Наприклад, 8-бітові підписані дані позначаються як ***Ipp8s*** типу. Ці типи даних IP-даних Intel визначені у відповідних файлах заголовків бібліотеки.

Для функцій, які працюють з одним типом даних, тип даних поле містить лише одне із перелічених вище значень.

Якщо функція працює на сигналах джерела та призначення, які мають різні типи даних, відповідні ідентифікатори типів даних перераховані в назві функції в порядку джерела та призначення наступним чином:

<datatype> = <src1Datatype> [src2Datatype] [dstDatatype]

Наприклад, функція ***ippsDotProd_16s16sc_Sfs*** обчислює точковий добуток 16-розрядних коротких та 16-розрядних складних коротких векторів джерела та зберігає результат у 16-розрядному складному короткому векторі призначення. dstDatatype модифікатор відсутній в назві, оскільки другий операнд і результат є того ж типу. Результат масштабований.

Існує кілька типів даних, а саме ***24u***, ***24s*** і ***16f*** які не підтримуються Intel IPP, але можуть бути перетворені на підтримувані типи даних для подальшої обробки функціями бібліотеки.

Intel IPP підтримує такі типи даних:

Тип	*Тип C / ++*	*Тип Intel IPP*
8u	***unsigned char***	***Ipp8u***
8s	***signed char***	***Ipp8s***
16u	***unsigned short***	***Ipp16u***
16s	***signed short***	***Ipp16s***
16sc	***complex short***	***Ipp16sc***
32u	***unsigned int***	***Ipp32u***
32s	***signed int***	***Ipp32s***
32f	***float***	***Ipp32f***
32fc	***complex float***	***Ipp32fc***
64s	***int64 (long long)***	***Ipp64s***
64f	***double***	***Ipp64f***
64fc	***complex double***	***Ipp64fc***

У разі, якщо у функції є кілька аргументів різних типів (або тип вихідних даних відрізняється від типу вхідних даних), використовується наступний формат типу оброблюваних даних***:***
<Datatype>= <Src1Datatype> [src2Datatype] [dstDatatype]

<Descriptor>- опис додаткових особливостей даних, які обробляються функцією

приклади:

M - використовується маска, яка визначає пікселі зображення, які будуть оброблені функцією

R - функція обробляє певну область зображення (ROI - region of interest)

<Параметри>- параметри функції; вказуються в наступному порядку:

вхідні аргументи, вихідні аргументи, додаткові аргументи

Розглянемо функцію ***ippiCopy_8u_C3R.*** Згідно описаної вище структури, дана функція копіює ***(<Name>=Copy***) зображення (***ippi,<Data_domain>=i***), елементами якого є 8-бітові цілі числа без знака (***<Data_type>=8u***).

Зображення є трьохканальним (***C3,<Descriptor>=C3R***), копіювання здійснюється з заданої області зображення (***R,<Descriptor>=C3R***).

1.6. Ініціалізація бібліотеки. Виділення і звільнення пам'яті.

Функції бібліотеки Intel IPP оптимізовані під широкий набір архітектур. Для вибору оптимальної реалізації для середовища, в якій виконується програма, необхідно на її початку викликати такі функції (в залежності від типу використовуваної лінковки)

IppStatus ippStaticInit (void)

Функція визначає тип використовуваного процесора і вибирає найбільш підходящий для даного процесора статичний код. Ця функція може бути застосована лише при статичній лінковці.

Синтаксис:
Випадок 1: Статичне підключення бібліотеки

IppStatus ippStaticInit(void);

Випадок 2: Динамічне підключення бібліотеки

IppStatus ippInit(void);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Опис:
Функція ippStaticInit використовується при статичному підключенні бібліотеки Intel IPP. Вона аналізує поточну архітектуру процесора (SSE, AVX, AVX2, AVX-512) та активує відповідну оптимізовану гілку коду, щоб забезпечити максимальну продуктивність.

Функція ***ippInit*** застосовується у випадку динамічної лінковки. Вона виконує аналогічну ініціалізацію, але для бібліотек, підключених під час виконання програми (DLL або SO). Завдяки цьому бібліотека IPP автоматично підбирає найефективніший набір інструкцій для конкретного процесора, що виконує програму.

Приклад використання ***ippInit***:

```
IppStatus status;
status = ippInit();
if (status != ippStsNoErr)
// помилка ініціалізації IPP
```

Повернені значення:

Функції ***ippStaticInit*** та ***ippInit*** повертають код типу ***IppStatus***. Якщо ініціалізація пройшла успішно, повертається ***ippStsNoErr***. У разі виникнення проблем повертається відповідний код помилки.

Malloc

Виділяє пам’ять, вирівняну до 64-байт

Синтаксис: Випадок 1: Виділення пам'яті для блоків 32-бітової довжини

```
Ipp8u* ippsMalloc_8u (int len);
Ipp16u* ippsMalloc_16u (int len);
Ipp32u* ippsMalloc_32u (int len);
Ipp8s* ippsMalloc_8s (int len);
Ipp16s* ippsMalloc_16s (int len);
Ipp32s* ippsMalloc_32s (int len);
Ipp64s* ippsMalloc_64s (int len);
Ipp32f* ippsMalloc_32f (int len);
Ipp64f* ippsMalloc_64f (int len);
Ipp8sc* ippsMalloc_8sc (int len);
Ipp16sc* ippsMalloc_16sc (int len);
Ipp32sc* ippsMalloc_32sc (int len);
Ipp64sc* ippsMalloc_64sc (int len);
Ipp32fc* ippsMalloc_32fc (int len);
Ipp64fc* ippsMalloc_64fc (int len);
```

Випадок 2: Виділення пам'яті для функцій платформи

```
Ipp8u* ippsMalloc_8u_L (IppSizeL об);
Ipp16u* ippsMalloc_16u_L (IppSizeL обр);
Ipp32u* ippsMalloc_32u_L (IppSizeL обр);
Ipp8s* ippsMalloc_8s_L (IppSizeL об);
Ipp16s* ippsMalloc_16s_L (IppSizeL об);
Ipp32s* ippsMalloc_32s_L (IppSizeL об);
Ipp64s* ippsMalloc_64s_L (IppSizeL len);
Ipp32f* ippsMalloc_32f_L (IppSizeL об);
Ipp64f* ippsMalloc_64f_L (IppSizeL об);
Ipp8sc* ippsMalloc_8sc_L (IppSizeL об'єкт);
Ipp16sc* ippsMalloc_16sc_L (IppSizeL об'єкт);
Ipp32sc* ippsMalloc_32sc_L (IppSizeL об'єкт);
```

Ipp64sc* ippsMalloc_64sc_L (IppSizeL об'єкт);
Ipp32fc* ippsMalloc_32fc_L (IppSizeL об);
Ipp64fc* ippsMalloc_64fc_L (IppSizeL об'єкт);

Включити файли: ***ipps.h*** , з ***_L*** суфіксом: ***ipps_l.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
len Кількість елементів для виділення.

Опис:
Ця функція виділяє блок пам'яті, вирівняний до 64-байтової межі для елементів різних типів даних.

Приклад використання ***ippsMalloc_8u:***

```
void func_malloc(void)
{
Ipp8u* pBuf = ippsMalloc_8u(8*sizeof(Ipp8u));
if(NULL == pBuf) not enough memory ippsFree(pBuf);
}
```

Повернені значення:
Повернене значення ***ippsMalloc*** є вказівником на вирівняний блок пам'яті. Якщо в системі немає пам'яті, то повертається значення ***NULL***. Щоб звільнити цей блок, використовуйте ***ippsFree***.

Free

Звільняє пам’ять, виділену функцією ***ippsMalloc***.

Синтаксис: ***void ippsFree (void* ptr);***

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ippcore.lib, ippvm.lib

Параметри: Вказівник на блок пам'яті, який потрібно звільнити. Блок пам'яті вказаний на ***ptr*** виділяється функцією ***ippsMalloc***.

Опис:

Ця функція звільняє вирівняний блок пам'яті, виділений функцією ***ippsMalloc***.

SetNumThreads

Встановлює кількість потоків у багатопотоковості

Синтаксис:

Випадок 1: Встановлення кількості потоків для операцій над об'єктами 32-бітового розміру: ***IppStatus ippSetNumThreads (int numThr);***

Випадок 2: Встановлення кількості потоків для операцій над об'єктами 64-бітового розміру: ***IppStatus ippSetNumThreads_L (int numThr);***

Включити файли: ***ippcore.h, ippcore64x.h***

Параметри: ***numThr*** - кількість потоків, повинна бути більше нуля.

Опис: Ця функція встановлює кількість потоків OpenMP. Кількість встановлених потоків може бути менше вказано ***numThr***

Значення: ***ippStsNoErr, ippStsSizeErr, ippStsNoOperation***

Режим округлення

Загальні функції обробки сигналів використовують округлення. Режим округлення за замовчуванням є найближчим парним, тобто номером фіксованої точки x=N + α, 0 ≤α <1, де N є цілим числом, округлюється так:

$$\lfloor x \rceil = \begin{cases} N, & 0 \le \alpha < 0.5, \\ N+1, & 0.5 < \alpha < 1, \\ N, & \alpha = 0.5,\ N \text{ — парне}, \\ N+1, & \alpha = 0.5,\ N \text{ — непарне}. \end{cases}$$

Цей вираз описує **правило округлення числа** x=N+αx, де N — ціла частина, а α — дробова частина.

Формула означає:

якщо дробова частина менше 0.5 — округлюємо донизу;

якщо більша за 0.5 — округлюємо догори;

якщо рівно 0.5 — тоді результат залежить від парності NNN: парні округлюються донизу, непарні догори (це т.зв. **round-to-even**, або **banker's rounding**).

Деякі функції мають додаткові режими округлення, які задаються параметром ***roundMode***.

Важливо

Функції стиснення даних, цілісності даних, обробки рядків та арифметики з фіксованою точністю не виконують округлення.

Масштабування (***ScaleFactor***)

Деякі функції обробки сигналів, що працюють на цілочисельних даних, використовують масштабування внутрішньо обчислених вихідних результатів цілим числом ***ScaleFactor***, який вказаний як один із параметрів функції. Ці функції мають ***Sfs*** дескриптор в їх іменах.

Коефіцієнт масштабування може бути від'ємним, додатним або нульовим. Масштабування застосовується, оскільки внутрішні обчислення зазвичай виконуються з більшою точністю, ніж типи даних, що використовуються для вхідних та вихідних сигналів.

Масштабування цілочисельного результату здійснюється множенням вихідних векторних значень на ***2-scaleFactor*** перед поверненням функції. Це допомагає зберегти або діапазон вихідних даних, або їх точність. Зазвичай масштабування з додатним коефіцієнтом виконується операцією зміни. Результат округлюється до найближчого парного цілого числа (див. "Режим округлення").

Наприклад, ціле число ***Ipp16s*** результат операції піднесення до квадрату ***ippsSqr*** для вхідного значення 200 дорівнює 32767 замість 40000, тобто результат насичений, і точне значення не може бути відновлено.

Масштабування вихідного значення з коефіцієнтом ***ScaleFactor= 1*** дає результат 20000, який не є насиченим, і точне значення можна відновити як 20000* 2. Таким чином, зберігається діапазон вихідних даних.

Наступний приклад показує, як можна частково зберегти точність за допомогою масштабування.

Операція цілого квадратного кореня ***ippsSqrt***(без масштабування) для вхідного значення 2 дає результат, що дорівнює 1 замість 1,414. Масштабування внутрішньо обчисленого вихідного значення з коефіцієнтом ***ScaleFactor = -3*** дає результат 11 і дозволяє відновити більш точне значення як 11* 2 -3 = 1,375.

2. ФУНКЦІЇ ВЕКТОРНОЇ ІНІЦІАЛІЗАЦІЇ

У цьому розділі описані функції, які ініціалізують значення векторних елементів. Всі векторні елементи можуть бути ініціалізовані загальним нулем або іншим заданим значенням. Вони також можуть бути ініціалізовані відповідними значеннями других векторних елементів.

2.1. Копіювання даних

Copy

Копіює вміст одного вектора в інший.

Синтаксис:

IppStatus ippsCopy_8u (const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsCopy_16s (const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsCopy_32s (const Ipp32s* pSrc, Ipp32s* pDst, int len);
IppStatus ippsCopy_32f (const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsCopy_64s (const Ipp64s* pSrc, Ipp64s* pDst, int len);
IppStatus ippsCopy_64f (const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsCopy_16sc (const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsCopy_32sc (const Ipp32sc* pSrc, Ipp32sc* pDst, int len);
IppStatus ippsCopy_32fc (const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsCopy_64sc (const Ipp64sc* pSrc, Ipp64sc* pDst, int len);
IppStatus ippsCopy_64fc (const Ipp64fc* pSrc, Ipp64fc* pDst, int len);

Включити файли: ipps.h

Доменні залежності
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc Вказівник на вектор-джерело.

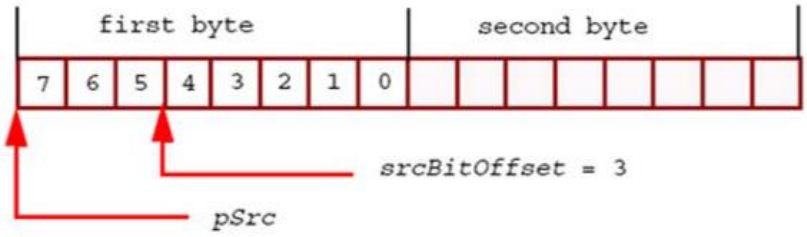
pDst Вказівник на вектор призначення.
len Кількість елементів для копіювання.

Опис: Ця функція копіює першs ***len*** елементів з вихідного вектора ***pSrc*** у вектор призначення ***pDst***.

ippsCopy_1u. Ця функція копіює елементи вектора, який має а 8uтип даних. Це означає, що кожен байт складається з восьми послідовних елементів вектора (1 біт на елемент). Вам потрібно вказати початкову позицію вихідного та кінцевого векторів ***ysrcBitOffset*** і ***dstBitOffset***

параметрів відповідно. Порядок бітів кожного байта є оберненим до порядку елементів. Це означає, що перший елемент у векторі представляє останній (сьомий) біт першого байта у векторі, як показано на малюнку нижче.

Розмітка бітів для функції ippsCopy_1u [1].



Примітка:

Ці функції виконують лише операції копіювання, описані вище, і не призначені для переміщення даних. Їх поведінка непередбачувана, якщо буфери джерела та призначення перекриваються. Для переміщення даних використовуйте ***ippsMove***.

Повернені значення:

ippStsNoErr Вказує на відсутність помилки

ippStsNullPtrErr Вказує на помилку, коли ***pSrc*** або ***pDst*** вказівник є ***NULL***.

ippStsSizeErr Позначає помилку, коли *лен* менше або дорівнює нулю.

Приклад:

У наведеному нижче прикладі показано, як використовувати ***ippsCopy*** функцію.

```
IppStatus copy(void)
{
char src[] = “to be copied\0”;
char dst[256];
return ippsCopy_8u(src, dst, strlen(src)+1);
}
```

CopyLE, CopyBE

Копіює вміст одного бітового вектора в інший.

Синтаксис:

IppStatus ippsCopyLE_1u (const Ipp8u* pSrc, int srcBitOffset, Ipp8u* pDst, int dstBitOffset, int len);

IppStatus ippsCopyBE_1u (const Ipp8u* pSrc, int srcBitOffset, Ipp8u* pDst, int dstBitOffset, int len);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc Вказівник на вектор-джерело.

pDst Вказівник на вектор призначення.

len Кількість елементів для копіювання.

srcBitOffset Зміщення, в бітах, від першого байта вихідного вектора.

dstBitOffset Зміщення, в бітах, від першого байта вектора призначення.

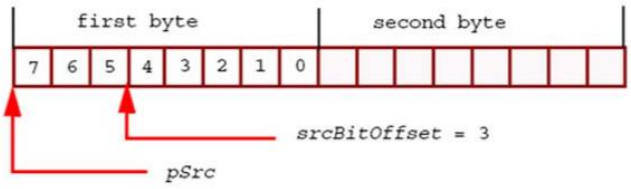
Опис:

Ці функції копіюють перші ***len*** елементів з вихідного вектора ***pSrc*** у вектор призначення ***pDst***.

Ці функції копіюють елементи вектора, який має ***8u*** тип даних. Це означає, що кожен байт складається з восьми послідовних елементів вектора (1 біт на елемент). Вам потрібно вказати початкову позицію вихідного та кінцевого векторів у ***srcBitOffset*** і ***dstBitOffset*** параметрів відповідно.

Для ***ippsCopyLE_1u*** функції, порядок розрядів кожного байта є оберненим до порядку елементів. Це означає, що перший елемент у векторі представляє останній (сьомий) біт першого байта у векторі, як показано на малюнку нижче.

Розмітка бітів для функції ***ippsCopyLE_1u*** [1]



Для ***ippsCopyBE_1u*** функції, порядок бітів кожного байту є звичайним. Це означає, що перший елемент у векторі представляє останній (нульовий) біт першого байта у векторі, як показано на малюнку нижче.

Повернені значення:

ippStsNoErr Вказує на відсутність помилки.

ippStsNullPtrErr Вказує на помилку, коли ***pSrc*** або ***pDst*** вказівник є ***NULL***

ippStsSizeErr Позначає помилку, коли: ***len*** менше або дорівнює нулю або ***srcBitOffset*** чи ***dstBitOffset*** менші нуля

Move

Переміщує вміст одного вектора до іншого вектора.

Синтаксис:

```
IppStatus ippsMove_8u (const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsMove_16s (const Ipp16s* pSrc, Ipp16s* pDst, int len);
IppStatus ippsMove_32s (const Ipp32s* pSrc, Ipp32s* pDst, int len);
IppStatus ippsMove_32f (const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsMove_64f (const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsMove_64s (const Ipp64s* pSrc, Ipp64s* pDst, int len);
IppStatus ippsMove_16sc (const Ipp16sc* pSrc, Ipp16sc* pDst, int len);
IppStatus ippsMove_32sc (const Ipp32sc* pSrc, Ipp32sc* pDst, int len);
IppStatus ippsMove_32fc (const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsMove_64sc (const Ipp64sc* pSrc, Ipp64sc* pDst, int len);
IppStatus ippsMove_64fc (const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
```

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc Вказівник на вихідний вектор, який використовується для ініціалізації ***pDst***.

pDst Вказівник на вектор призначення, який потрібно ініціалізувати.

len Кількість елементів для переміщення.

Опис:

Ця функція рухається першою ***len*** елементи з вихідного вектора ***pSrc*** у вектор призначення ***pDst***. Якщо деякі частини вихідного та кінцевого векторів перекриваються, тоді функція забезпечує переміщення вихідних байтів джерела в частинах, що перекриваються, (це означає, що вони скопійовані перед перезаписом) у відповідні частини вектору призначення.

Повернені значення:
ippStsNoErr Вказує на відсутність помилки.
ippStsNullPtrErr Вказує на помилку, коли ***pSrc*** або ***pDst*** вказівник є ***NULL***.
ippStsSizeErr Позначає помилку, коли ***len*** менше або дорівнює нулю.

Приклад:
У наведеному нижче прикладі показано, як користуватися функцією ***ippsMove***:

```
Ipp8u pSrc[10] = { "123456789" };
Ipp8u pDst[6];
int len = 6;
IppStatus status;
status = ippsMove_8u ( pSrc, pDst, len );
if(ippStsNoErr != status)
printf("IPP Error: %s",ippGetStatusString(status));

Result:
pSrc = 123456789
pDst = 123456
```

Set

Ініціалізує вектор з елементами, заданими загальним значенням.

Синтаксис:
IppStatus ippsSet_8u (Ipp8u val, Ipp8u* pDst, int len);
IppStatus ippsSet_16s (Ipp16s val, Ipp16s* pDst, int len);
IppStatus ippsSet_16sc (Ipp16sc val, Ipp16sc* pDst, int len);
IppStatus ippsSet_32s (Ipp32s val, Ipp32s* pDst, int len);
IppStatus ippsSet_32f (Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsSet_32sc (Ipp32sc val, Ipp32sc* pDst, int len);
IppStatus ippsSet_32fc (Ipp32fc val, Ipp32fc* pDst, int len);
IppStatus ippsSet_64s (Ipp64s val, Ipp64s* pDst, int len);
IppStatus ippsSet_64f (Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsSet_64sc (Ipp64sc val, Ipp64sc* pDst, int len);
IppStatus ippsSet_64fc (Ipp64fc val, Ipp64fc* pDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pDst Вказівник на вектор, який потрібно ініціалізувати.
len Кількість елементів для ініціалізації.
val Значення, що використовується для ініціалізації вектора ***pDst***.

Опис:
Ця функція ініціалізує першу ***len*** елементи дійсного або складного вектора ***pDst*** містити одне і те ж значення ***val***.

Повернені значення:
ippStsNoErr Вказує на відсутність помилки.
ippStsNullPtrErr Вказує на помилку, коли ***pDst*** вказівник є ***NULL***.
ippStsSizeErr Позначає помилку, коли ***len*** менше або дорівнює нулю.

Приклад:
У наведеному нижче прикладі коду показано, як користуватися функцією ***ippsSet***.

```
#include <stdio.h>
#include <string.h>
#include "ipps.h"

/* Збирайте й лінкуйте з ippcore.lib, ippvm.lib */

IppStatus set_example_ascii(void)
{
   Ipp8u buf[16];          /* місце для 15 символів + '\0' */
   int len = 15; /* скільки байтів встановити (не рахуючи нуль-термінатора) */

   /* Ініціалізуємо перші len байтів символом 'A' */
   IppStatus st = ippsSet_8u((Ipp8u)'A', buf, len);
   if (st != ippStsNoErr) {
      fprintf(stderr, "ippsSet_8u failed: %d\n", st);
      return st;
   }

   /* Поставимо термінатор - тепер можемо друкувати як C-string */
   buf[len] = 0;

   printf("Result: \"%s\"\n", (char*)buf); /* виведе 15 'A' */
   return ippStsNoErr;
}
```

Zero

Ініціалізує вектор з нульовими елементами.

Синтаксис

IppStatus ippsZero_8u (Ipp8u* pDst, int len);
IppStatus ippsZero_16s (Ipp16s* pDst, int len);
IppStatus ippsZero_32s (Ipp32s* pDst, int len);
IppStatus ippsZero_32f (Ipp32f* pDst, int len);
IppStatus ippsZero_64s (Ipp64s* pDst, int len);
IppStatus ippsZero_64f (Ipp64f* pDst, int len);
IppStatus ippsZero_16sc (Ipp16sc* pDst, int len);
IppStatus ippsZero_32sc (Ipp32sc* pDst, int len);
IppStatus ippsZero_32fc (Ipp32fc* pDst, int len);
IppStatus ippsZero_64sc (Ipp64sc* pDst, int len);
IppStatus ippsZero_64fc (Ipp64fc* pDst, int len);

Включити файли: ***ipps.h***

Доменні залежності
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pDst Вказівник на вектор, який потрібно ініціалізувати
len. Кількість елементів для ініціалізації

Опис:

Ця функція ініціалізує першу ***len*** елементи вектора ***pDst*** до нуля. Якщо ***pDst*** є складним вектором, як дійсну, так і уявну частини обнуляють.

Повернені значення:
ippStsNoErr Вказує на відсутність помилки
ippStsNullPtrErr Вказує на помилку, коли ***pDst*** вказівник є ***NULL***.
ippStsSizeErr Позначає помилку, коли ***len*** менше або дорівнює нулю.

Приклад:

У наведеному нижче прикладі коду показано, як використовувати ***ippsZero***.

```
IppStatus zero(void) {
char src[] = “zero”;
return ippsZero_8u(src, strlen(src));
}
```

3. ОСНОВНІ ФУНКЦІЇ БІБЛІОТЕКИ INTEL IPP

3.1. Логічні функції

And

Обчислює побітове «І» двох векторів.

Синтаксис:

IppStatus ippsAnd_8u (const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len);
IppStatus ippsAnd_16u (const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsAnd_32u (const Ipp32u* pSrc1, const Ipp32u* pSrc2, Ipp32u* pDst, int len);
IppStatus ippsAnd_8u_I (const Ipp8u* pSrc, Ipp8u* pSrcDst, int len);
IppStatus ippsAnd_16u_I (const Ipp16u* pSrc, Ipp16u* pSrcDst, int len); IppStatus ippsAnd_32u_I (const Ipp32u* pSrc, Ipp32u* pSrcDst, int len);

Включити файли: ***ipps.h***
Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc1, pSrc2 Вказівники на два вихідні вектори.
pDst Вказівник на вектор призначення.
pSrc Вказівник на вектор джерела для операції на місці.
Len Кількість елементів у векторі.

Опис:
Ця функція обчислює побітове І відповідних елементів векторів ***pSrc1*** і ***pSrc2***, і зберігає результат у векторі ***pDst***.

OrC

Обчислює побітове АБО скалярного значення та кожного елементу вектора.

Синтаксис:

IppStatus ippsOrC_8u (const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len);
IppStatus ippsOrC_16u (const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len);
IppStatus ippsOrC_32u (const Ipp32u* pSrc, Ipp32u val, Ipp32u* pDst, int len);
IppStatus ippsOrC_8u_I (Ipp8u val, Ipp8u* pSrcDst, int len);
IppStatus ippsOrC_16u_I (Ipp16u val, Ipp16u* pSrcDst, int len); IppStatus ippsOrC_32u_I (Ipp32u val, Ipp32u* pSrcDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
val Вхідне скалярне значення.
pSrc Вказівник на вектор-джерело.
pDst Вказівник на вектор призначення.
pSrcDst Вказівник на джерело та вектор призначення для місця операції.
len Кількість елементів у векторі.

Опис:
Ця функція обчислює побітове АБО скалярного значення ***val*** і кожен елемент вектора ***pSrc***, і зберігає результат у ***pDst***.

OR

Обчислює побітове АБО двох векторів.

Синтаксис:
IppStatus ippsOr_8u (const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len);
IppStatus ippsOr_16u (const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsOr_32u (const Ipp32u* pSrc1, const Ipp32u* pSrc2, Ipp32u* pDst, int len);
IppStatus ippsOr_8u_I (const Ipp8u* pSrc, Ipp8u* pSrcDst, int len);
IppStatus ippsOr_16u_I (const Ipp16u* pSrc, Ipp16u* pSrcDst, int len); IppStatus ippsOr_32u_I (const Ipp32u* pSrc, Ipp32u* pSrcDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc1, pSrc2 Вказівники на два вихідні вектори.
pDst Вказівник на вектор призначення.
pSrc Вказівник на вектор джерела для операції на місці.
pSrcDst Вказівник на джерело та вектор призначення для місця операції.
Len Кількість елементів у векторі.

Опис:

Ця функція обчислює побітове АБО відповідних елементів векторів ***pSrc1*** і ***pSrc2***, і зберігає результат у векторі ***pDst***.

XorC

Обчислює побітове XOR скалярного значення та кожного елементу вектора.

Синтаксис:
IppStatus ippsXorC_8u (const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len);
IppStatus ippsXorC_16u (const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len);
IppStatus ippsXorC_32u (const Ipp32u* pSrc, Ipp32u val, Ipp32u* pDst, int len);
IppStatus ippsXorC_8u_I (Ipp8u val, Ipp8u* pSrcDst, int len);
IppStatus ippsXorC_16u_I (Ipp16u val, Ipp16u* pSrcDst, int len); IppStatus ippsXorC_32u_I (Ipp32u val, Ipp32u* pSrcDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
val Вхідне скалярне значення.
pSrc Вказівник на вектор-джерело.
pDst Вказівник на вектор призначення.
pSrcDst Вказівник на джерело та вектор призначення для місця операції.
len Кількість елементів у векторі.

Опис:
Ця функція обчислює побітове XOR скалярного значення ***val*** і кожен елемент вектора ***pSrc***, і зберігає результат у ***pDst***.

Xor

Обчислює побітове XOR двох векторів.

Синтаксис:
IppStatus ippsXor_8u (const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len);
IppStatus ippsXor_16u (const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsXor_32u (const Ipp32u* pSrc1, const Ipp32u* pSrc2, Ipp32u* pDst, int len);
IppStatus ippsXor_8u_I (const Ipp8u* pSrc, Ipp8u* pSrcDst, int len);

IppStatus ippsXor_16u_I (const Ipp16u* pSrc, Ipp16u* pSrcDst, int len);
IppStatus ippsXor_32u_I (const Ipp32u* pSrc, Ipp32u* pSrcDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc1, pSrc2 Вказівники на два вихідні вектори.
pDst Вказівник на вектор призначення.
pSrc Вказівник на вектор джерела для операції на місці
pSrcDst Вказівник на джерело та вектор призначення для місця операції
len Кількість елементів у векторі.

Опис:
Ця функція обчислює побітове XOR відповідних елементів векторів ***pSrc1*** і ***pSrc2***, і зберігає результат у векторі ***pDst***.

Not

Обчислює побітове НЕ векторних елементів.

Синтаксис:
IppStatus ippsNot_8u (const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsNot_16u (const Ipp16u* pSrc, Ipp16u* pDst, int len);
IppStatus ippsNot_32u (const Ipp32u* pSrc, Ipp32u* pDst, int len);
IppStatus ippsNot_8u_I (Ipp8u* pSrcDst, int len);
IppStatus ippsNot_16u_I (Ipp16u* pSrcDst, int len);
IppStatus ippsNot_32u_I (Ipp32u* pSrcDst, int len);

Включити файли: ***ipps.h***
Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc Вказівник на вектор-джерело.
pDst Вказівник на вектор призначення.
pSrcDst Вказівник на джерело та вектор призначення для місця операції.
len Кількість елементів у векторі.

Опис:

Ця функція обчислює побітове НЕ відповідних елементів векторів ***pSrc***, і зберігає результат у векторі ***pDst***.

LShiftC

Зсуває біти в векторних елементах вліво.

Синтаксис:

IppStatus ippsLShiftC_8u (const Ipp8u* pSrc, int val, Ipp8u* pDst, int len);
IppStatus ippsLShiftC_16s (const Ipp16s* pSrc, int val, Ipp16s* pDst, int len);
IppStatus ippsLShiftC_16u (const Ipp16u* pSrc, int val, Ipp16u* pDst, int len);
IppStatus ippsLShiftC_32s (const Ipp32s* pSrc, int val, Ipp32s* pDst, int len);
IppStatus ippsLShiftC_8u_I (int val, Ipp8u* pSrcDst, int len);
IppStatus ippsLShiftC_16u_I (int val, Ipp16u* pSrcDst, int len);
IppStatus ippsLShiftC_16s_I (int val, Ipp16s* pSrcDst, int len);
IppStatus ippsLShiftC_32s_I (int val, Ipp32s* pSrcDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

val Кількість бітів, на яку функція зміщує кожен елемент вектор ***pSrc*** або ***pSrcDst***.

pSrc Вказівник на вектор-джерело.

pDst Вказівник на вектор призначення.

pSrcDst Вказівник на джерело та вектор призначення для місця операції.

len Кількість елементів у векторі.

Опис:

Ця функція зміщує кожен елемент вектора ***pSrc*** від ***val*** біти ліворуч, а результат зберігає в ***pDst***.

RShiftC

Зсуває біти в векторних елементах вправо.

Синтаксис:
IppStatus ippsRShiftC_8u (const Ipp8u* pSrc, int val, Ipp8u* pDst, int len);
IppStatus ippsRShiftC_16s (const Ipp16s* pSrc, int val, Ipp16s* pDst, int len);
IppStatus ippsRShiftC_16u (const Ipp16u* pSrc, int val, Ipp16u* pDst, int len);
IppStatus ippsRShiftC_32s (const Ipp32s* pSrc, int val, Ipp32s* pDst, int len);
IppStatus ippsRShiftC_8u_I (int val, Ipp8u* pSrcDst, int len);
IppStatus ippsRShiftC_16u_I (int val, Ipp16u* pSrcDst, int len);
IppStatus ippsRShiftC_16s_I (int val, Ipp16s* pSrcDst, int len);
IppStatus ippsRShiftC_32s_I (int val, Ipp32s* pSrcDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри: ***val*** Кількість бітів, на яку функція зміщує кожен елемент вектор ***pSrc*** або ***pSrcDst***.

pSrc Вказівник на вектор-джерело.

pDst Вказівник на вектор призначення.

pSrcDst Вказівник на джерело та вектор призначення для місця операції.

Len Кількість елементів у векторі.

Опис:

Ця функція зміщує кожен елемент вектора ***pSrc*** від ***val*** біти праворуч і зберігає результат у ***pDst***.

Зверніть увагу: арифметичний правий зсув застосовується до знакових (signed) типів і при зсуві заповнює старші біти копією знакового біту (тобто зберігає знак числа), тоді як логічний правий зсув застосовується до беззнакових (unsigned) типів і заповнює старші біти нулями.

3.2. Арифметичні функції

Add

Додавання елементів двох векторів.

Синтаксис:

Випадок 1. Операції, що виконуються не на місці (out-of-place), для даних з плаваючою комою та для цілих типів без масштабування.

IppStatus ippsAdd_16s(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len);
IppStatus ippsAdd_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsAdd_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);
IppStatus ippsAdd_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int len);
IppStatus ippsAdd_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst, int len);
IppStatus ippsAdd_8u16u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsAdd_16u(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsAdd_32u(const Ipp32u* pSrc1, const Ipp32u* pSrc2, Ipp32u* pDst, int len);
IppStatus ippsAdd_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32f* pDst, int len);

Випадок 2. Операції, що виконуються не на місці (out-of-place), для цілих типів із масштабуванням:

IppStatus ippsAdd_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsAdd_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsAdd_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsAdd_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsAdd_16sc_Sfs(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, Ipp16sc* pDst, int len, int scaleFactor);
IppStatus ippsAdd_32sc_Sfs(const Ipp32sc* pSrc1, const Ipp32sc* pSrc2, Ipp32sc* pDst, int len, int scaleFactor);
IppStatus ippsAdd_64s_Sfs(const Ipp64s* pSrc1, const Ipp64s* pSrc2, Ipp64s* pDst, int len, int scaleFactor);

Випадок 3. Операції, що виконуються на місці (in-place), для даних з плаваючою крапкою та для цілих типів без застосування масштабування.

IppStatus ippsAdd_16s_I(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len);
IppStatus ippsAdd_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsAdd_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);
IppStatus ippsAdd_32fc_I(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, int len);
IppStatus ippsAdd_64fc_I(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, int len);
IppStatus ippsAdd_16s32s_I(const Ipp16s* pSrc, Ipp32s* pSrcDst, int len);

Випадок 4. Операції, що виконуються на місці (in-place), для цілих типів із застосуванням масштабування.

IppStatus ippsAdd_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsAdd_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsAdd_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsAdd_32s_ISfs(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsAdd_16sc_ISfs(const Ipp16sc* pSrc, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsAdd_32sc_ISfs(const Ipp32sc* pSrc, Ipp32sc* pSrcDst, int len, int scaleFactor);

Випадок 5. Операції, що виконуються не на місці (out-of-place), для 64-бітних чисел з плаваючою крапкою (Ipp64f).

IppStatus ippsAdd_32f_L(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, Ipp64u len);

Включені файли: ***ipps.h , ipps64x.h*** (for 64x flavors)

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***, ***ipps.lib, ippcore_tl.lib, ipps_tl.lib***

Параметри:
pSrc1, pSrc2 — вказівники на вхідні (джерельні) вектори.
pDst — вказівник на вектор призначення (вихідний вектор).
pSrc — вказівник на вектор-джерело для операцій **на місці** (in-place).
pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операцій **на місці** (результат записується в той самий буфер).
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування (див. розділ «Масштабування цілих» / *Integer Scaling*).

Опис:
Ця функція додає елементи вектора ***pSrc1*** до відповідних елементів вектора ***pSrc2*** і зберігає результат у ***pDst***. Варіанти ***ippsAdd***, що виконуються на місці (in-place), додають елементи вектора ***pSrc*** до відповідних елементів вектора ***pSrcDst*** і зберігають результат у ***pSrcDst***. Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***; якщо обчислене значення виходить за межі допустимого діапазону для цільового типу, воно піддається сатурації.

AddProductC

Додавання добутків елементів вектора на константу в акумулюючий вектор.

Синтаксис:
IppStatus ippsAddProductC_32f(const Ipp32f* pSrc, const Ipp32f val, Ipp32f* pSrcDst, int len);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ippcore.lib, ippvm.lib

Параметри:
pSrc — вказівник на вхідний вектор.
val — значення, на яке множиться кожний елемент вхідного вектора перед додаванням.
pSrcDst — вказівник на вектор, що одночасно є джерелом і вектором призначення для операції на місці.
len — кількість елементів у векторі.

Опис:
Ця функція множить кожний елемент вхідного вектора ***pSrc*** на значення ***val*** і додає отриманий результат до відповідного елемента акумуляторного вектора ***pSrcDst*** за формулою:

$$pSrcDst[n] = pSrcDst[n] + pSrc[n] \cdot val, \quad 0 \le n < \text{len}$$

AddProduct

Додає добутки двоїх векторів в акумулятор

Синтаксис:

Випадок 1. Операції над даними з плаваючою крапкою.

IppStatus ippsAddProduct_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pSrcDst, int len);
IppStatus ippsAddProduct_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pSrcDst, int len);
IppStatus ippsAddProduct_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pSrcDst, int len);

IppStatus ippsAddProduct_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pSrcDst, int len);

Випадок 2. Operations on integer data with scaling.

IppStatus ippsAddProduct_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsAddProduct_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsAddProduct_16s32s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32s* pSrcDst, int len, int scaleFactor);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc1, pSrc2 — вказівники на вхідні вектори.
pSrcDst — вказівник на вектор-акумулятор призначення.
len — кількість елементів у векторах.
scaleFactor — коефіцієнт масштабування.

Опис:
Ця функція множить кожний елемент вхідного вектора ***pSrc1*** на відповідний елемент вектора ***pSrc2*** і додає результат до відповідного елемента акумуляторного вектора pSrcDst за формулою:

$$pSrcDst[n] = pSrcDst[n] + pSrc1[n] \cdot pSrc2[n], \quad 0 \le n < \text{len}.$$

Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***. Якщо результат виходить за межі діапазону допустимих значень для типу даних, він насичується (saturation).

Mul

Множення елементів двох векторів.

Синтаксис:

Випадок 1. Операції, що виконуються не на місці (out-of-place), для даних з плаваючою крапкою та для цілих типів без масштабування.
IppStatus ippsMul_16s(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len);
IppStatus ippsMul_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsMul_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);

IppStatus ippsMul_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int len);
IppStatus ippsMul_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst, int len);
IppStatus ippsMul_8u16u(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp16u* pDst, int len);
IppStatus ippsMul_32f32fc(const Ipp32f* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int len);
IppStatus ippsMul_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32f* pDst, int len);

Випадок 2. Операції, що виконуються не на місці (out-of-place), для цілих типів із застосуванням масштабування.

IppStatus ippsMul_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsMul_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsMul_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsMul_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsMul_16sc_Sfs(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, Ipp16sc* pDst, int len, int scaleFactor);
IppStatus ippsMul_32sc_Sfs(const Ipp32sc* pSrc1, const Ipp32sc* pSrc2, Ipp32sc* pDst, int len, int scaleFactor);
IppStatus ippsMul_16s32s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsMul_16u16s_Sfs(const Ipp16u* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len, int scaleFactor);

Випадок 3. Операції, що виконуються на місці (in-place), для даних з плаваючою крапкою та для цілих типів без масштабування.

IppStatus ippsMul_16s_I(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len);
IppStatus ippsMul_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsMul_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);
IppStatus ippsMul_32fc_I(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, int len);
IppStatus ippsMul_64fc_I(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, int len);
IppStatus ippsMul_32f32fc_I(const Ipp32f* pSrc, Ipp32fc* pSrcDst, int len);

Випадок 4. Операції, що виконуються на місці (in-place), для цілих типів із застосуванням масштабування.

IppStatus ippsMul_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsMul_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsMul_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsMul_32s_ISfs(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsMul_16sc_ISfs(const Ipp16sc* pSrc, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsMul_32sc_ISfs(const Ipp32sc* pSrc, Ipp32sc* pSrcDst, int len, int scaleFactor);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc1, pSrc2 — вказівники на вхідні (джерельні) вектори.

pDst — вказівник на вектор призначення.

pSrc — вказівник на вектор-джерело для операцій на місці (in-place).

pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операцій на місці.

len — кількість елементів у векторі.

scaleFactor — коефіцієнт масштабування (див. розділ «Integer Scaling»).

Опис:

Ця функція перемножує відповідні елементи векторів ***pSrc1*** і ***pSrc2*** та зберігає результат у ***pDst***. Варіанти ***ippsMul***, що виконуються на місці (in-place), множать елементи вектора ***pSrc*** на елементи вектора ***pSrcDst*** і зберігають результат у ***pSrcDst***. Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***. Якщо обчислене значення виходить за межі допустимого діапазону для цільового типу даних, результат насичується (***saturation***). Варіант функції з суфіксом ***Low*** вимагає, щоб добуток кожної пари елементів не перевищував діапазон типу ***Ipp32s***.

SubC

Віднімає сталу (константну) величину від кожного елемента вектора.

Синтаксис:

Випадок 1. Операції, що виконуються не на місці (out-of-place), над даними з плаваючою крапкою.

```
IppStatus ippsSubC_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsSubC_32fc(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, int len);
IppStatus ippsSubC_64f(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsSubC_64fc(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, int len);
```

Випадок 2. Операції, що виконуються не на місці (out-of-place), над цілими даними.

```
IppStatus ippsSubC_8u_Sfs(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsSubC_16u_Sfs(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsSubC_16s_Sfs(const Ipp16s* pSrc, Ipp16s val, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSubC_32s_Sfs(const Ipp32s* pSrc, Ipp32s val, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsSubC_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc val, Ipp16sc* pDst, int len, int scaleFactor);
IppStatus ippsSubC_32sc_Sfs(const Ipp32sc* pSrc, Ipp32sc val, Ipp32sc* pDst, int len, int scaleFactor);
```

Випадок 3. Операції на місці (in-place) над даними з плаваючою крапкою та цілими типами без масштабування.

IppStatus ippsSubC_16s_I(Ipp16s val, Ipp16s* pSrcDst, int len);
IppStatus ippsSubC_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsSubC_64f_I(Ipp64f val, Ipp64f* pSrcDst, int len);
IppStatus ippsSubC_32fc_I(Ipp32fc val, Ipp32fc* pSrcDst, int len);
IppStatus ippsSubC_64fc_I(Ipp64fc val, Ipp64fc* pSrcDst, int len);

Випадок 4. Операції на місці (in-place) над цілими даними з масштабуванням.

IppStatus ippsSubC_8u_ISfs(Ipp8u val, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_16u_ISfs(Ipp16u val, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_16s_ISfs(Ipp16s val, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_32s_ISfs(Ipp32s val, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_16sc_ISfs(Ipp16sc val, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubC_32sc_ISfs(Ipp32sc val, Ipp32sc* pSrcDst, int len, int scaleFactor);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc — вказівник на вхідний вектор.
val — скалярне значення, яке віднімається від кожного елемента вектора.
pDst — вказівник на вектор призначення.
pSrcDst — вказівник на вектор, що одночасно є джерелом і вектором призначення для операції на місці.
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування (див. розділ «Integer Scaling»).

Опис:
Ця функція віднімає значення ***val*** від кожного елемента вектора ***pSrc*** і зберігає результат у ***pDst***.
Варіанти ***ippsSubC***, що виконуються на місці (in-place), віднімають значення val від кожного елемента вектора ***pSrcDst*** і зберігають результат у ***pSrcDst***.
Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення scaleFactor; якщо вихідне значення перевищує допустимий діапазон для типу даних, результат насичується.

SubCRev

Віднімає кожний елемент вектора від константи

Синтаксис:

Випадок 1. Операції, що виконуються не на місці (out-of-place), над даними з плаваючою крапкою.

IppStatus ippsSubCRev_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsSubCRev_64f(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsSubCRev_32fc(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, int len);
IppStatus ippsSubCRev_64fc(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, int len);

Випадок 2. Операції, що виконуються не на місці (out-of-place), над цілими даними.

IppStatus ippsSubCRev_8u_Sfs(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16u_Sfs(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16s_Sfs(const Ipp16s* pSrc, Ipp16s val, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSubCRev_32s_Sfs(const Ipp32s* pSrc, Ipp32s val, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc val, Ipp16sc* pDst, int len, int scaleFactor);
IppStatus ippsSubCRev_32sc_Sfs(const Ipp32sc* pSrc, Ipp32sc val, Ipp32sc* pDst, int len, int scaleFactor);

Випадок 3. Операції на місці (in-place) над даними з плаваючою крапкою.

IppStatus ippsSubCRev_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsSubCRev_64f_I(Ipp64f val, Ipp64f* pSrcDst, int len);
IppStatus ippsSubCRev_32fc_I(Ipp32fc val, Ipp32fc* pSrcDst, int len);
IppStatus ippsSubCRev_64fc_I(Ipp64fc val, Ipp64fc* pSrcDst, int len);

Випадок 4. Операції на місці (in-place) над цілими даними.

IppStatus ippsSubCRev_8u_ISfs(Ipp8u val, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16u_ISfs(Ipp16u val, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16s_ISfs(Ipp16s val, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_32s_ISfs(Ipp32s val, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_16sc_ISfs(Ipp16sc val, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsSubCRev_32sc_ISfs(Ipp32sc val, Ipp32sc* pSrcDst, int len, int scaleFactor);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
val — скалярне значення, від якого віднімаються елементи вектора.
pSrc — вказівник на вхідний вектор.
pDst — вказівник на вектор призначення.
pSrcDst — вказівник на вектор, елементи якого віднімаються від значення ***val*** у разі операції на місці; цей же вектор зберігає результат операції ***val*** - ***pSrcDst[n].***
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування (див. розділ «Integer Scaling»).

Опис:
Ця функція віднімає кожен елемент вектора ***pSrc*** від значення ***val*** і зберігає результат у ***pDst***. Варіанти ***ippsSubCRev***, що виконуються на місці (in-place), віднімають кожен елемент вектора ***pSrcDst*** від значення ***val*** і зберігають результат у ***pSrcDst***. Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***. Якщо результат виходить за межі діапазону допустимих значень для типу даних, він насичується (saturation).

Sub

Віднімання елементів двох векторів.

Синтаксис:

Випадок 1. Операції, що виконуються не на місці (out-of-place), над даними з плаваючою крапкою та цілими типами без масштабування.

IppStatus ippsSub_16s(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len);
IppStatus ippsSub_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsSub_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);
IppStatus ippsSub_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int len);
IppStatus ippsSub_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst, int len);
IppStatus ippsSub_16s32f(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp32f* pDst, int len);

Випадок 2. Операції, що виконуються не на місці (out-of-place), над цілими даними з масштабуванням.

IppStatus ippsSub_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len, int scaleFactor);

IppStatus ippsSub_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len, int scaleFactor);

IppStatus ippsSub_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len, int scaleFactor);

IppStatus ippsSub_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDst, int len, int scaleFactor);

IppStatus ippsSub_16sc_Sfs(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, Ipp16sc* pDst, int len, int scaleFactor);

IppStatus ippsSub_32sc_Sfs(const Ipp32sc* pSrc1, const Ipp32sc* pSrc2, Ipp32sc* pDst, int len, int scaleFactor);

Випадок 3. Операції на місці (in-place) над даними з плаваючою крапкою та цілими типами без масштабування.

IppStatus ippsSub_16s_I(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len);
IppStatus ippsSub_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsSub_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);
IppStatus ippsSub_32fc_I(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, int len);
IppStatus ippsSub_64fc_I(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, int len);

Випадок 4. Операції на місці (in-place) над цілими даними з масштабуванням.

IppStatus ippsSub_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSub_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSub_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSub_32s_ISfs(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSub_16sc_ISfs(const Ipp16sc* pSrc, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsSub_32sc_ISfs(const Ipp32sc* pSrc, Ipp32sc* pSrcDst, int len, int scaleFactor);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc1 — вказівник на вхідний вектор-від'ємник (subtrahend), елементи якого віднімаються.
pSrc2 — вказівник на вхідний вектор-зменшуване (minuend), від елементів якого віднімаються елементи pSrc1.
pDst — вказівник на вектор призначення.
pSrc — вказівник на вхідний вектор-від'ємник для операції на місці.
pSrcDst — вказівник на вектор-зменшуване, який одночасно є вектором призначення для операції на місці.
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування (див. розділ «Integer Scaling»).

Опис:

Ця функція віднімає елементи вектора ***pSrc1*** від елементів вектора ***pSrc2*** і зберігає результат у ***pDst***. Варіанти ***ippsSub***, що виконуються на місці (in-place), віднімають елементи вектора ***pSrc*** від елементів вектора ***pSrcDst*** і зберігають результат у ***pSrcDst***. Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***; якщо вихідне значення перевищує допустимий діапазон для типу даних, результат насичується.

DivC

Ділить кожний елемент вектора на сталу (константну) величину.

Синтаксис:

Випадок 1. Операції, що виконуються не на місці (out-of-place), над даними з плаваючою крапкою.

IppStatus ippsDivC_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsDivC_64f(const Ipp64f* pSrc, Ipp64f val, Ipp64f* pDst, int len);
IppStatus ippsDivC_32fc(const Ipp32fc* pSrc, Ipp32fc val, Ipp32fc* pDst, int len);
IppStatus ippsDivC_64fc(const Ipp64fc* pSrc, Ipp64fc val, Ipp64fc* pDst, int len);

Випадок 2. Операції, що виконуються не на місці (out-of-place), над цілими даними з масштабуванням.

IppStatus ippsDivC_8u_Sfs(const Ipp8u* pSrc, Ipp8u val, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsDivC_16u_Sfs(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsDivC_16s_Sfs(const Ipp16s* pSrc, Ipp16s val, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsDivC_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc val, Ipp16sc* pDst, int len, int scaleFactor);

Випадок 3. Операції на місці (in-place) над даними з плаваючою крапкою.

IppStatus ippsDivC_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);
IppStatus ippsDivC_64f_I(Ipp64f val, Ipp64f* pSrcDst, int len);
IppStatus ippsDivC_32fc_I(Ipp32fc val, Ipp32fc* pSrcDst, int len);
IppStatus ippsDivC_64fc_I(Ipp64fc val, Ipp64fc* pSrcDst, int len);

Випадок 4. Операції на місці (in-place) над цілими даними з масштабуванням.

IppStatus ippsDivC_8u_ISfs(Ipp8u val, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsDivC_16u_ISfs(Ipp16u val, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsDivC_16s_ISfs(Ipp16s val, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsDivC_64s_ISfs(Ipp64s val, Ipp64s* pSrcDst, Ipp32u len, int scaleFactor);
IppStatus ippsDivC_16sc_ISfs(Ipp16sc val, Ipp16sc* pSrcDst, int len, int scaleFactor);

Включені файли:***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
val — скалярне значення, що використовується як дільник.
pSrc — вказівник на вхідний вектор.
pDst — вказівник на вектор призначення.
pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операції на місці.
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування (див. розділ «Integer Scaling»).

Опис:
Ця функція ділить кожен елемент вектора ***pSrc*** на значення ***val*** і зберігає результат у ***pDst***. Варіанти ***ippsDivC***, що виконуються на місці (in-place), ділять кожен елемент вектора ***pSrcDst*** на значення val і зберігають результат у тому ж векторі ***pSrcDst***. Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***; якщо обчислене значення виходить за межі діапазону допустимих значень для типу даних, результат насичується (saturation).

DivCRev

Ділить сталу (константну) величину на кожен елемент вектора.

Синтаксис:
IppStatus ippsDivCRev_16u(const Ipp16u* pSrc, Ipp16u val, Ipp16u* pDst, int len);
IppStatus ippsDivCRev_32f(const Ipp32f* pSrc, Ipp32f val, Ipp32f* pDst, int len);
IppStatus ippsDivCRev_16u_I(Ipp16u val, Ipp16u* pSrcDst, int len);
IppStatus ippsDivCRev_32f_I(Ipp32f val, Ipp32f* pSrcDst, int len);

Включені файли:***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

val — стала (константна) величина, яка використовується як ділене в операції.

pSrc — вказівник на вхідний вектор, елементи якого використовуються як дільники.

pDst — вказівник на вектор призначення.

pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операції на місці.

len — кількість елементів у векторі.

Опис:

Ця функція ділить сталу величину val на кожен елемент вектора pSrc і зберігає результати у векторі pDst. Варіанти функції ippsDivCRev, що виконуються на місці (in-place), ділять сталу величину val на кожен елемент вектора pSrcDst і зберігають результати у тому ж векторі pSrcDst.

Div

Ділення елементів двох векторів.

Синтаксис:

Випадок 1. Операції, що виконуються не на місці (out-of-place), над цілими даними.

IppStatus ippsDiv_8u_Sfs(const Ipp8u* pSrc1, const Ipp8u* pSrc2, Ipp8u* pDst, int len, int scaleFactor);

IppStatus ippsDiv_16u_Sfs(const Ipp16u* pSrc1, const Ipp16u* pSrc2, Ipp16u* pDst, int len, int scaleFactor);

IppStatus ippsDiv_16s_Sfs(const Ipp16s* pSrc1, const Ipp16s* pSrc2, Ipp16s* pDst, int len, int scaleFactor);

IppStatus ippsDiv_32s_Sfs(const Ipp32s* pSrc1, const Ipp32s* pSrc2, Ipp32s* pDst, int len, int scaleFactor);

IppStatus ippsDiv_16sc_Sfs(const Ipp16sc* pSrc1, const Ipp16sc* pSrc2, Ipp16sc* pDst, int len, int scaleFactor);

IppStatus ippsDiv_32s16s_Sfs(const Ipp16s* pSrc1, const Ipp32s* pSrc2, Ipp16s* pDst, int len, int scaleFactor);

Випадок 2. Операції, що виконуються не на місці (out-of-place), над даними з плаваючою крапкою.

IppStatus ippsDiv_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsDiv_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);
IppStatus ippsDiv_32fc(const Ipp32fc* pSrc1, const Ipp32fc* pSrc2, Ipp32fc* pDst, int len);
IppStatus ippsDiv_64fc(const Ipp64fc* pSrc1, const Ipp64fc* pSrc2, Ipp64fc* pDst, int len);

Випадок 3. Операції на місці (in-place) над цілими даними.

IppStatus ippsDiv_8u_ISfs(const Ipp8u* pSrc, Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsDiv_16u_ISfs(const Ipp16u* pSrc, Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsDiv_16s_ISfs(const Ipp16s* pSrc, Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsDiv_16sc_ISfs(const Ipp16sc* pSrc, Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsDiv_32s_ISfs(const Ipp32s* pSrc, Ipp32s* pSrcDst, int len, int scaleFactor);

Випадок 4. Операції на місці (in-place) над даними з плаваючою крапкою.

IppStatus ippsDiv_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsDiv_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);
IppStatus ippsDiv_32fc_I(const Ipp32fc* pSrc, Ipp32fc* pSrcDst, int len);
IppStatus ippsDiv_64fc_I(const Ipp64fc* pSrc, Ipp64fc* pSrcDst, int len);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc1 — вказівник на вектор-дільник (divisor).
pSrc2 — вказівник на вектор-ділене (dividend).
pDst — вказівник на вектор призначення.
pSrc — вказівник на вектор-дільник для операцій на місці.
pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операцій на місці.
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування (див. розділ «Integer Scaling»).

Опис:
Ця функція ділить елементи вектора ***pSrc2*** на відповідні елементи вектора ***pSrc1*** і зберігає результат у ***pDst***. Варіанти ***ippsDiv***, що виконуються на місці (in-place), ділять елементи вектора ***pSrcDst*** на відповідні елементи вектора ***pSrc*** і зберігають результат у ***pSrcDst***. Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***. Якщо вихідне значення перевищує діапазон даних цільового типу, результат насичується (saturation). Якщо будь-який елемент вектора-дільника дорівнює нулю, функція повертає попередження, але продовжує виконання, призначаючи відповідному результату визначене значення.

Sqr

Обчислює квадрат кожного елемента векторв

Синтаксис:

IppStatus ippsSqr_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsSqr_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsSqr_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsSqr_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsSqr_8u_Sfs(const Ipp8u* pSrc, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsSqr_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSqr_16u_Sfs(const Ipp16u* pSrc, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsSqr_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, int scaleFactor);
IppStatus ippsSqr_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsSqr_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsSqr_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsSqr_64fc_I(Ipp64fc* pSrcDst, int len);
IppStatus ippsSqr_8u_ISfs(Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqr_16s_ISfs(Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqr_16u_ISfs(Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqr_16sc_ISfs(Ipp16sc* pSrcDst, int len, int scaleFactor);

Включені файли:***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc — вказівник на вхідний вектор.
pDst — вказівник на вектор призначення.
pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операцій на місці.
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування, див. «Integer Scaling».

Опис:
Ця функція обчислює квадрат кожного елемента вектора ***pSrc*** і зберігає результат у ***pDst***. Обчислення виконується так:

$$pDst[n] = \left(pSrc[n]\right)^2$$

Варіанти ***ippsSqr***, що виконуються на місці (in-place), обчислюють квадрат кожного елемента вектора ***pSrcDst*** і зберігають результат у ***pSrcDst***. Обчислення виконується так:

$$pSrcDst[n] = \left(pSrcDst[n]\right)^2$$

Під час піднесення до квадрата цілих чисел вихід може перевищити діапазон цільового типу даних і бути насиченим. Щоб отримати точний результат, використовуйте коефіцієнт масштабування (scaleFactor).

Exp

Обчислює експоненту кожного елемента вектора.

Синтаксис:

IppStatus ippsExp_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsExp_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsExp_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsExp_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsExp_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsExp_32s_Sfs(const Ipp32s* pSrc, Ipp32s* pDst, int len, int scaleFactor);
IppStatus ippsExp_16s_ISfs(Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsExp_32s_ISfs(Ipp32s* pSrcDst, int len, int scaleFactor);

Включені файли:***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc — вказівник на вхідний вектор.
pDst — вказівник на вектор призначення.
pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операцій на місці.
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування (див. «Integer Scaling»).

Опис:
Ця функція обчислює експоненту для кожного елемента вектора ***pSrc*** і зберігає результат у ***pDst***. Обчислення виконується за формулою:

$$pDst[n] = \exp\left(pSrc[n]\right)$$

Варіанти ***ippsExp***, що виконуються на місці (in-place), обчислюють експоненту для кожного елемента вектора ***pSrcDst*** і зберігають результат у ***pSrcDst***:

$$pSrcDst[n] = \exp\left(pSrcDst[n]\right)$$

Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***; якщо вихідне значення виходить за межі діапазону цільового типу даних, результат насичується. Для цілих типів використання коефіцієнта масштабування дозволяє уникнути переповнення та зберегти потрібну точність фіксованої крапки.

Ln

Обчислює натуральний логарифм кожного елемента вектора.

Синтаксис:

IppStatus ippsLn_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsLn_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsLn_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsLn_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsLn_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsLn_32s_Sfs(const Ipp32s* pSrc, Ipp32s* pDst, int len, int scaleFactor);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc — вказівник на вхідний вектор.
pDst — вказівник на вектор призначення.
pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операцій на місці.
len — кількість елементів у векторі.
scaleFactor — коефіцієнт масштабування (див. «Integer Scaling»).

Опис:
Ця функція обчислює натуральний логарифм для кожного елемента вектора ***pSrc*** і зберігає результат у pDst:

$$pDst[n] = \ln\big(pSrc[n]\big)$$

Варіанти ***ippsLn***, що виконуються на місці (in-place), обчислюють натуральний логарифм для кожного елемента вектора ***pSrcDst*** і зберігають результат у ***pSrcDst***:

$$pSrcDst[n] = \ln\big(pSrcDst[n]\big)$$

Функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***; якщо обчислене значення виходить за межі діапазону цільового типу даних, результат насичується. Логарифм визначений лише для додатних вхідних значень; для нульових або від'ємних елементів повертається попередження, а обробка продовжується відповідно до політики IPP для спеціальних випадків.

Sqrt (Cubrt)

Обчислює квадратний (Sqrt) або кубічний (Cubrt) корінь кожного елемента вектора.

Синтаксис:

IppStatus ippsSqrt_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsSqrt_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsSqrt_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsSqrt_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsSqrt_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsSqrt_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsSqrt_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsSqrt_64fc_I(Ipp64fc* pSrcDst, int len);
IppStatus ippsSqrt_8u_Sfs(const Ipp8u* pSrc, Ipp8u* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_16u_Sfs(const Ipp16u* pSrc, Ipp16u* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_64s_Sfs(const Ipp64s* pSrc, Ipp64s* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_32s16s_Sfs(const Ipp32s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_64s16s_Sfs(const Ipp64s* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsSqrt_8u_ISfs(Ipp8u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqrt_16s_ISfs(Ipp16s* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqrt_16u_ISfs(Ipp16u* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqrt_16sc_ISfs(Ipp16sc* pSrcDst, int len, int scaleFactor);
IppStatus ippsSqrt_64s_ISfs(Ipp64s* pSrcDst, int len, int scaleFactor).

IppStatus ippsCubrt_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsCubrt_32s16s_Sfs(const Ipp32s* pSrc, Ipp16s* pDst, int len, int scaleFactor).

Включені файли: ***ipps.h.***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib.***

Параметри:
pSrc — вказівник на вхідний вектор.
pDst — вказівник на вектор призначення.

pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операцій на місці.

len — кількість елементів у векторі.

scaleFactor — коефіцієнт масштабування (див. «Integer Scaling»).

Опис:

Функція ***ippsSqrt*** обчислює квадратний корінь для кожного елемента вектора ***pSrc*** і зберігає результат у ***pDst***:

$$pDst[n] = \sqrt{pSrc[n]}$$

Версії, що виконуються на місці (in-place), записують результат у pSrcDst:

$$pSrcDst[n] = \sqrt{pSrcDst[n]}$$

Для комплексних даних корінь визначено в комплексній площині; для від'ємних дійсних аргументів повертається попередження, і обробка триває згідно з правилами «Handling of Special Cases». Для цілих типів точність можна підвищити за допомогою ***scaleFactor***.

Функція ***ippsCubrt*** обчислює кубічний корінь для кожного елемента вектора pSrc і зберігає результат у ***pDst***:

$$pDst[n] = \left(pSrc[n]\right)^{1/3}$$

Для фіксованої крапки доступна версія з масштабуванням ***ippsCubrt_32s16s_Sfs***, у якій результат масштабується відповідно до ***scaleFactor***.

3.3. Операції з комплексними числами

Real

Повертає дійсну частину складного вектора у окремий вектор.

Синтаксис:

IppStatus ippsReal_16sc(const Ipp16sc* pSrc, Ipp16s* pDstRe, int len);
IppStatus ippsReal_32fc(const Ipp32fc* pSrc, Ipp32f* pDstRe, int len);
IppStatus ippsReal_64fc(const Ipp64fc* pSrc, Ipp64f* pDstRe, int len);

Включені файли: ***ipps.h***

Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc Вказівник на складний вектор-джерело.
pDstRe Вказівник на вектор призначення для дійсних частин.
len Кількість елементів у векторі.

Опис

Ця функція копіює дійсну частину кожного елемента комплексного вектора ***pSrc*** у відповідний елемент вектора pDstRe за правилом:

$$pDstRe[n] = \mathrm{Re}\big(pSrc[n]\big), \quad 0 \le n < \mathrm{len}.$$

У версіях із цілими типами (***16sc→16s***) дійсна частина береться без масштабування; для дійсних типів з плаваючою крапкою (***32fc, 64fc***) повертається відповідне значення з подвійною/одинарною точністю.

Imag

Повертає уявну частину складного вектора у окремий вектор.

Синтаксис:

IppStatus ippsImag_16sc(const Ipp16sc* pSrc, Ipp16s* pDstIm, int len);
IppStatus ippsImag_32fc(const Ipp32fc* pSrc, Ipp32f* pDstIm, int len);
IppStatus ippsImag_64fc(const Ipp64fc* pSrc, Ipp64f* pDstIm, int len);

Включені файли: ***ipps.h***

Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc	вказівник на складний вектор-джерело.
pDstIm	вказівник на вектор призначення для уявних частин.
len	кількість елементів у векторі.

Опис

Ця функція копіює уявну частину кожного елемента комплексного вектора ***pSrc*** у відповідний елемент вектора ***pDstIm*** за правилом:

$$pDstIm[n] = \mathrm{Im}\big(pSrc[n]\big), \quad 0 \le n < \mathrm{len}.$$

Для цілих типів (***16sc→16s***) уявна частина копіюється без масштабування, тоді як для типів з плаваючою крапкою (***32fc, 64fc***) значення зберігаються у відповідному форматі одинарної або подвійної точності.

Conj

Зберігає комплексно спряжені значення елементів вектора у інший вектор або виконує операцію на місці.

Синтаксис:

IppStatus ippsConj_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int len);

IppStatus ippsConj_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsConj_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsConj_16sc_I(Ipp16sc* pSrcDst, int len);
IppStatus ippsConj_32fc_I(Ipp32fc* pSrcDst, int len);
IppStatus ippsConj_64fc_I(Ipp64fc* pSrcDst, int len);

Включені файли: ***ipps.h***

Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc — вказівник на вхідний (вихідний) вектор.
pDst — вказівник на вектор призначення.
pSrcDst — вказівник на вектор, який одночасно є джерелом і вектором призначення для операції на місці.
len — кількість елементів у векторі.

Опис:
Ця функція обчислює комплексне спряження кожного елемента вектора ***pSrc*** та зберігає результат у векторі ***pDst***. Комплексне спряження для кожного елемента визначається так:

$$\begin{cases} pDst[n].re = pSrc[n].re, \\ pDst[n].im = -pSrc[n].im, \end{cases} \quad 0 \le n < len$$

У варіантах функції, що виконуються на місці (in-place), комплексне спряження застосовується безпосередньо до елементів вектора ***pSrcDst*** за формулою:

$$\begin{cases} pSrcDst[n].re = pSrcDst[n].re, \\ pSrcDst[n].im = -pSrcDst[n].im, \end{cases} \quad 0 \le n < len$$

Функція підтримує обчислення для 16-бітових, 32-бітових і 64-бітових комплексних типів даних.

4. ФУНКЦІЇ ПЕРЕТВОРЕНЬ БІБЛІОТЕКИ INTEL IPP

4.1. Функції Сортування і операції з векторами

SortAscend, SortDescend

Сортування елементів вектора за зростанням і за спаданням.

Синтаксис

IppStatus ippsSortAscend_8u_I(Ipp8u* pSrcDst, int len);
IppStatus ippsSortAscend_16u_I(Ipp16u* pSrcDst, int len);
IppStatus ippsSortAscend_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsSortAscend_32s_I(Ipp32s* pSrcDst, int len);
IppStatus ippsSortAscend_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsSortAscend_64f_I(Ipp64f* pSrcDst, int len);
IppStatus ippsSortDescend_8u_I(Ipp8u* pSrcDst, int len);
IppStatus ippsSortDescend_16u_I(Ipp16u* pSrcDst, int len);
IppStatus ippsSortDescend_16s_I(Ipp16s* pSrcDst, int len);
IppStatus ippsSortDescend_32s_I(Ipp32s* pSrcDst, int len);
IppStatus ippsSortDescend_32f_I(Ipp32f* pSrcDst, int len);
IppStatus ippsSortDescend_64f_I(Ipp64f* pSrcDst, int len);

Включені файли: ipps.h

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc Вказівник на вихідний вектор.
pDst Вказівник на вектор призначення.
pSrcDst Вказівник на джерело та вектор призначення для місця операції.
len Кількість елементів у векторі.

Опис:
Сортування елементів вектора за зростанням і за спаданням.

Приклад:

```
int main()
{
  Ipp16u pSrc[] = { 5, 7, 2, 1, 6, 8, 9, 0, 4, 3 };
  IppStatus status;
  int i;
```

```
  printf("\nSource vector\n");
  for (i = 0; i < 10; i++) printf("%d ", pSrc[i]);

  check_sts(status = ippsSortAscend_16u_I(pSrc, 10));

  printf("\nResult values vector\n");
  for (i = 0; i < 10; i++) printf("%d ", pSrc[i]);

  EXIT_MAIN
        printf("\nExit status %d (%s)\n", (int)status, ippGetStatusString(status));
  return (int)status;
}
```

FLIP

Змінює порядок елементів у векторі на обернений

Синтаксис:

IppStatus ippsFlip_8u (const Ipp8u* pSrc, Ipp8u* pDst, int len);
IppStatus ippsFlip_16u (const Ipp16u* pSrc, Ipp16u* pDst, int len);
IppStatus ippsFlip_32f (const Ipp32f* pSrc, Ipp32f* pDst, int len);
IppStatus ippsFlip_64f (const Ipp64f* pSrc, Ipp64f* pDst, int len);
IppStatus ippsFlip_32fc (const Ipp32fc* pSrc, Ipp32fc* pDst, int len);
IppStatus ippsFlip_64fc (const Ipp64fc* pSrc, Ipp64fc* pDst, int len);
IppStatus ippsFlip_16u_I (Ipp16u* pSrcDst, int len);
IppStatus ippsFlip_8u_I (Ipp8u* pSrcDst, int len);
IppStatus ippsFlip_32f_I (Ipp32f* pSrcDst, int len);
IppStatus ippsFlip_64f_I (Ipp64f* pSrcDst, int len);
IppStatus ippsFlip_32fc_I (Ipp32fc* pSrcDst, int len);
IppStatus ippsFlip_64fc_I (Ipp64fc* pSrcDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc	Вказівник на вихідний вектор.
pDst	Вказівник на вектор призначення.
pSrcDst	Вказівник на джерело та вектор призначення для місця операції
len	Кількість елементів у векторі.

Опис

Ця функція перетворює елементи вихідного вектора ***pSrc*** до вектора призначення ***pDst*** у зворотному порядку за наступною формулою:

$$pDst[n] = pSrc[\text{len} - n - 1], \quad n = 0, \ldots, \text{len} - 1$$

FindNearestOne

Знаходить елемент таблиці, який є найближчим до вказаного значення.

Синтаксис:

IppStatus ippsFindNeestsOne_16u (Ipp16u inpVal, Ipp16u* pOutVal, int* pOutIndex, const Ipp16u* pTable, int tblLen);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

inpVal	Довідкове значення.
pOutVal	Вказівник на вихідне значення.
pOutIndex	Вказівник на вихідний індекс.
pTable	Вказівник на таблицю для пошуку.
tblLen	Кількість елементів у таблиці.

Опис:

Ця функція здійснює пошук у таблиці ***pTable*** для елемента, який є найближчим до вказаного опорного значення ***inpVal***. Отриманий елемент та його індекс зберігаються в ***pOutVal*** і ***pOutIndex*** відповідно. Елементи таблиці повинні відповідати умові ***pTable[n] ≤ pTable[n+1]***. Функція використовує такий критерій відстані для визначення найближчого елемента таблиці: ***min(|inpVal -pTable[n]|).***

Threshold

Виконує порогову операцію з елементами вектора, обмежуючи значення елементів заданими значеннями.

Синтаксис:

IppStatus ippsThreshold_16s (const Ipp16s* pSrc, Ipp16s* pDst, int len, level Ipp16s, IppCmpOp relOp);
IppStatus ippsThreshold_32f (const Ipp32f* pSrc, Ipp32f* pDst, int len, Ipp32f рівень, IppCmpOp relOp);
IppStatus ippsThreshold_64f (const Ipp64f* pSrc, Ipp64f* pDst, int len, level Ipp64f, IppCmpOp relOp);
IppStatus ippsThreshold_32fc (const Ipp32fc* pSrc, Ipp32fc* pDst, int len, level Ipp32f, IppCmpOp relOp);
IppStatus ippsThreshold_64fc (const Ipp64fc* pSrc, Ipp64fc* pDst, int len, level Ipp64f, IppCmpOp relOp);
IppStatus ippsThreshold_16sc (const Ipp16sc* pSrc, Ipp16sc* pDst, int len, level Ipp16s, IppCmpOp relOp);
IppStatus ippsThreshold_16s_I (Ipp16s* pSrcDst, int len, level Ipp16s, IppCmpOp relOp);
IppStatus ippsThreshold_32f_I (Ipp32f* pSrcDst, int len, level Ipp32f, IppCmpOp relOp);
IppStatus ippsThreshold_64f_I (Ipp64f* pSrcDst, int len, level Ipp64f, IppCmpOp relOp);
IppStatus ippsThreshold_32fc_I (Ipp32fc* pSrcDst, int len, level Ipp32f, IppCmpOp relOp);
IppStatus ippsThreshold_64fc_I (Ipp64fc* pSrcDst, int len, level Ipp64f, IppCmpOp relOp);
IppStatus ippsThreshold_16sc_I (Ipp16sc* pSrcDst, int len, level Ipp16s, IppCmpOp relOp);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc Вказівник на вихідний вектор.
pDst Вказівник на вектор призначення.
pSrcDst Вказівник на джерело та вектор призначення для місця операції.
Level рівень

Значення, що використовується для обмеження кожного елемента ***pSrc*** або ***pSrcDst***. Цей параметр завжди повинен бути дійсним. Для складних версій він повинен бути позитивним і відображати величину.

Опис:

Значення цього аргументу визначають, який реляційний оператор використовувати та чи ***level*** є верхньою або нижньою межею для вхідних даних. ***relOp*** має мати одне з наступних значень:

ippCmpLess Вказує оператор "менше" і ***level*** є нижчим зв'язаний.

ippCmpGreater Вказує оператор “більше ніж” та ***level*** є верхня межа.

Ця функція виконує порогову операцію над вектором ***pSrc*** шляхом обмеження кожного елемента пороговим значенням ***level***. Функціонування функції подібне до функцій ***ippsThreshold_LT***, ***ippsThreshold_GT*** але його інтерфейс містить ***relOp*** параметр, який визначає тип операції порівняння, яку потрібно виконати.

Функція на місці ***ippsThreshold*** виконує порогову операцію над вектором ***pSrcDst*** шляхом обмеження кожного елемента пороговим значенням ***level***.

relOp аргумент вказує, який реляційний оператор використовувати: коли його значення ***ippCmpGreater*** – “більше ніж”, коли ***ippCmpLess*** - “менше ніж” і визначає, чи ***level*** є верхньою або нижньою межею для введення відповідно.

Формула для ***ippsThreshold*** із ***relOp = ippCmpLess*** це:

$$pDst[n] = \begin{cases} \text{level}, & pSrc[n] < \text{level}, \\ pSrc[n], & \text{otherwise}. \end{cases}$$

Формула для ***ippsThreshold*** із ***relOp = ippCmpGreater*** це:

$$pDst[n] = \begin{cases} \text{level}, & pSrc[n] > \text{level} \\ pSrc[n], & \text{otherwise} \end{cases}$$

Для складних версій функції ***ippsThreshold***, ***level*** аргумент завжди реальний. Формула комплексу ***ippsThreshold*** із ***relOp = ippCmpLess*** це:

$$pDst[n] = \begin{cases} \dfrac{pSrc[n] \cdot level}{|pSrc[n]|}, & \text{if } |pSrc[n]| < level, \\ pSrc[n], & \text{otherwise}. \end{cases}$$

Формула комплексу ***ippsThreshold*** із ***relOp = ippCmpGreater*** це:

$$pDst[n] = \begin{cases} \dfrac{pSrc[n] \cdot level}{|pSrc[n]|}, & |pSrc[n]| > level \\ pSrc[n], & \text{otherwise} \end{cases}$$

Примітки щодо застосування:

Для всіх складних версій ***level*** має бути позитивним і представляти величину. Величина вводу обмежена, але фаза залишається незмінною. Припускається, що нульовий вхід має нульову фазу.

Спеціальне правило застосовується до цілочисельних складних версій функції ***ippsThreshold***. Загалом, отримані координати точок на комплексній площині не є цілими числами. Функція округляє їх до цілих чисел таким чином, що порогова операція не виконується. Таким чином, для операції "менше" (з ***ippCmpLess*** прапорцем) координати округлені до нескінченності (***+Inf*** для додатних координат, і ***–Inf*** для відємних), а для операції „***більше ніж***" (з ***ippCmpGreater*** флагом) координати округлені до нуля.

Magnitude

Обчислює модуль комплексного числа

Синтаксис:

IppStatus ippsMagnitude_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDst, int len);
IppStatus ippsMagnitude_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDst, int len);
IppStatus ippsMagnitude_32fc(const Ipp32fc* pSrc, Ipp32f* pDst, int len); IppStatus ippsMagnitude_64fc(const Ipp64fc* pSrc, Ipp64f* pDst, int len);
IppStatus ippsMagnitude_16s32f(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp32f* pDst, int len);
IppStatus ippsMagnitude_16sc32f(const Ipp16sc* pSrc, Ipp32f* pDst, int len);
IppStatus ippsMagnitude_16s_Sfs(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsMagnitude_16sc_Sfs(const Ipp16sc* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsMagnitude_32sc_Sfs(const Ipp32sc* pSrc, Ipp32s* pDst, int len, int scaleFactor);

Параметри:

pSrc вказівник на вхідний комплексний вектор.

pSrcRe вказівник на вектор із дійсними частинами комплексних елементів.

pSrcIm вказівник на вектор із уявними частинами комплексних елементів.

pDst вказівник на вектор призначення, у якому зберігаються обчислені модулі елементів.

len кількість елементів у векторі.

scaleFactor коефіцієнт масштабування (див. «Integer Scaling»).

Опис:

Комплексний варіант цієї функції обчислює покомпонентну (елементну) величину модуля кожного елемента комплексного вектора ***pSrc*** і зберігає

результат у векторі ***pDst***. Модуль комплексного елемента визначається формулою:

$$\mathrm{magn}[n] = \sqrt{pSrc[n].re^2 + pSrc[n].im^2}$$

Дійсний варіант функції ***ippsMagnitude*** обчислює модуль комплексного вектора, дійсні та уявні частини якого подані окремо у векторах ***pSrcRe*** та ***pSrcIm*** відповідно, і зберігає результат у ***pDst***:

$$magn[n] = \sqrt{(pSrcRe[n])^2 + (pSrcIm[n])^2}$$

Для цілих типів функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***, щоб уникнути переповнення та зберегти точність при фіксованій крапці.

Phase

Обчислює аргумент (фазу) комплексних елементів вектора.

Синтаксис:

IppStatus ippsPhase_64fc(const Ipp64fc* pSrc, Ipp64f* pDst, int len);
IppStatus ippsPhase_32fc(const Ipp32fc* pSrc, Ipp32f* pDst, int len);
IppStatus ippsPhase_16sc32f(const Ipp16sc* pSrc, Ipp32f* pDst, int len);
IppStatus ippsPhase_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDst, int len);
IppStatus ippsPhase_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDst, int len);
IppStatus ippsPhase_16s32f(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp32f* pDst, int len);
IppStatus ippsPhase_16sc_Sfs(const Ipp16sc* pSrc, Ipp16s* pDst, int len, int scaleFactor);
IppStatus ippsPhase_16s_Sfs(const Ipp16s* pSrcRe, const Ipp16s* pSrcIm, Ipp16s* pDst, int len, int scaleFactor);

Включені файли: ***ipps.h***

Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc вказівник на вхідний комплексний вектор.
pSrcRe вказівник на вектор, що містить дійсні частини комплексних елементів.
pSrcIm вказівник на вектор, що містить уявні частини комплексних елементів.

pDst вказівник на вектор призначення, у якому зберігаються фазові (кутові) значення елементів у радіанах. Значення фази знаходяться в діапазоні [-π, π].

len кількість елементів у векторі.

scaleFactor коефіцієнт масштабування (див. «Integer Scaling»).

Опис:

Ця функція обчислює фазу (аргумент) кожного елемента комплексного вхідного вектора ***pSrc*** або комплексного вектора, дійсні та уявні компоненти якого задані у векторах ***pSrcRe*** та ***pSrcIm*** відповідно. Результат зберігається у векторі ***pDst***.

Фаза обчислюється як:

$$pDst[n] = \operatorname{atan2}\big(pSrcIm[n],\, pSrcRe[n]\big), \quad 0 \le n < \text{len}.$$

Значення фази повертаються в радіанах і лежать у діапазоні [-π, π].

Для цілих типів даних функції з суфіксом ***Sfs*** виконують масштабування результату відповідно до значення ***scaleFactor***. Якщо під час обчислення виникає переповнення або результат виходить за межі діапазону типу, він насичується (saturated).

4.2. Статистичні функції

Тут описані функції IPP Intel, які обчислюють векторні значення вимірювань: максимальне, мінімальне, середнє та стандартне відхилення.

Sum

Обчислює суму елементів вектора.

Синтаксис:

IppStatus ippsSum_32f(const Ipp32f* pSrc, int len, Ipp32f* pSum, IppHintAlgorithm hint);
IppStatus ippsSum_32fc(const Ipp32fc* pSrc, int len, Ipp32fc* pSum, IppHintAlgorithm hint);
IppStatus ippsSum_64f(const Ipp64f* pSrc, int len, Ipp64f* pSum);
IppStatus ippsSum_64fc(const Ipp64fc* pSrc, int len, Ipp64fc* pSum);
IppStatus ippsSum_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16s* pSum, int scaleFactor);
IppStatus ippsSum_32s_Sfs(const Ipp32s* pSrc, int len, Ipp32s* pSum, int scaleFactor);
IppStatus ippsSum_16s32s_Sfs(const Ipp16s* pSrc, int len, Ipp32s* pSum, int scaleFactor);
IppStatus ippsSum_16sc_Sfs(const Ipp16sc* pSrc, int len, Ipp16sc* pSum, int scaleFactor);

IppStatus ippsSum_16sc32sc_Sfs(const Ipp16sc* pSrc, int len, Ipp32sc* pSum, int scaleFactor);

Включені файли***: ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc вказівник на вхідний вектор.
pSum вказівник на змінну, куди зберігається підсумок.
len кількість елементів у векторі.
Hint підказка реалізації для варіантів із плаваючою крапкою (наприклад, швидший або точніший алгоритм; значення визначені переліком IppHintAlgorithm).
scaleFactor коефіцієнт масштабування для цілих варіантів (див. «Integer Scaling»).

Опис:

Функція обчислює суму елементів вектора pSrc і записує результат у pSum. Сума визначається як:

$$\text{sum} = \sum_{n=0}^{\text{len}-1} pSrc[n]$$

Аргумент ***hint*** для варіантів із плаваючою крапкою дозволяє вибрати реалізацію: акцент на швидкість обчислення або на підвищену точність. Для цілих типів результат може виходити за межі діапазону цільового типу і насичуватися; щоб уникнути переповнення та керувати точністю фіксованої крапки, застосовується масштабування відповідно до значення ***scaleFactor***.

Повернені значення:
У разі успіху повертається ***ippStsNoErr***. Якщо будь-який із вказівників ***pSrc*** або ***pSum*** дорівнює ***NULL***, повертається ***ippStsNullPtrErr***. Якщо ***len ≤ 0***, повертається ***ippStsSizeErr***.

Приклад:
У наведеному прикладі обчислюється сума 16-бітних цілих зі масштабуванням на 1 біт (ділення на 2):

```
#include <stdio.h>
#include "ipps.h"
void sum_example(void) {
   Ipp16s x[4] = { -32768, 32767, 32767, 32767 };
   Ipp16s sm = 0;
   IppStatus st = ippsSum_16s_Sfs(x, 4, &sm, 1);
/* (−32768 + 32767 + 32767 + 32767) >> 1 = 65533 >> 1 = 32766 */
   if (st == ippStsNoErr)
      printf("sum = %d\n", (int)sm);
   else
      printf("ippsSum failed: %d\n", st);
}
```

Max

Повертає максимальне значення вектора.

Синтаксис:

IppStatus ippsMax_16s (const Ipp16s* pSrc, int len, Ipp16s* pMax);
IppStatus ippsMax_32s (const Ipp32s* pSrc, int len, Ipp32s* pMax);
IppStatus ippsMax_32f (const Ipp32f* pSrc, int len, Ipp32f* pMax);
IppStatus ippsMax_64f (const Ipp64f* pSrc, int len, Ipp64f* pMax);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc	Вказівник на вихідний вектор.
pMax	Вказівник на вихідний результат.
len	Кількість елементів у векторі

Опис:
Ця функція повертає максимальне значення вхідного вектора ***pSrc***, і зберігає результат у ***pMax***.

MaxIndx

Повертає максимальне значення вектора та індекс максимального елемента.

Синтаксис:
IppStatus ippsMaxIndx_16s (const Ipp16s* pSrc, int len, Ipp16s* pMax, int* pIndx);
IppStatus ippsMaxIndx_32s (const Ipp32s* pSrc, int len, Ipp32s* pMax, int* pIndx);
IppStatus ippsMaxIndx_32f (const Ipp32f* pSrc, int len, Ipp32f* pMax, int* pIndx);
IppStatus ippsMaxIndx_64f (const Ipp64f* pSrc, int len, Ipp64f* pMax, int* pIndx);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc	Вказівник на вихідний вектор.
pMax	Вказівник на вихідний результат.
len	Кількість елементів у векторі.
pIndx	Вказівник на значення індексу максимального елемента.

Опис:

Ця функція повертає максимальне значення вхідного вектора ***pSrc***, і зберігає результат у ***pMax***. Якщо ***pIndx*** не є ***NULL*** покажчик, функція повертає індекс максимального елемента і зберігає його в ***pIndx***. Якщо є кілька рівних максимальних елементів, повертається перший індекс з початку.

Приклад:

Наведений нижче приклад коду демонструє, як користуватися функцією ***ippsMaxIndx***.

```
Ipp16s src[] = { 1, -2, 3, 8, -6 };
Ipp16s max;
int len = 5;
int indx;
ippsMaxIndx_16s ( src, len, &max, &indx );
```

```
Result:

max = 8  indx = 3
```

MaxAbs

Повертає максимальне абсолютне значення вектора.

Синтаксис:

IppStatus ippsMaxAbs_16s (const Ipp16s* pSrc, int len, Ipp16s* pMaxAbs);
IppStatus ippsMaxAbs_32s (const Ipp32s* pSrc, int len, Ipp32s* pMaxAbs);
IppStatus ippsMaxAbs_32f (const Ipp32f* pSrc, int len, Ipp32f* pMaxAbs);
IppStatus ippsMaxAbs_64f (const Ipp64f* pSrc, int len, Ipp64f* pMaxAbs);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc Вказівник на вихідний вектор.

pMaxAbs Вказівник на вихідний результат.

len Кількість елементів у векторі.

Опис:

Ця функція повертає максимальне абсолютне значення елементів вхідного вектора ***pSrc*** і зберігає результат у ***pMaxAbs***.

Повернені значення:

ippStsNoErr вказує на відсутність помилки.

ippStsNullPtrErr вказує на помилку, коли один із вказівників ***pMaxAbs*** або ***pSrc*** дорівнює ***NULL***.

ippStsSizeErr позначає помилку, коли len менше або дорівнює 0.

Приклад:

Нижче наведено приклад використання функції ***ippsMaxAbs_16s***:

```
#include <stdio.h>
#include "ipps.h"

void maxabs_example(void) {
   Ipp16s src[5] = { 2, -8, -3, -1, 7 };
   Ipp16s maxAbs = 0;

   IppStatus st = ippsMaxAbs_16s(src, 5, &maxAbs);
   if (st == ippStsNoErr) {
      printf("maxAbs = %d\n", (int)maxAbs);
   } else {
      printf("ippsMaxAbs_16s failed: %d\n", st);
   }
}
```

MaxAbsIndx

Повертає максимальне абсолютне значення вектора таіндекс відповідного елемента.

Синтаксис:

IppStatus ippsMaxAbsIndx_16s (const Ipp16s* pSrc, int len, Ipp16s* pMaxAbs, int* pIndx); IppStatus ippsMaxAbsIndx_32s (const Ipp32s* pSrc, int len, Ipp32s* pMaxAbs, int* pIndx);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc вказівник на вхідний вектор.
pMaxAbs вказівник на змінну, у яку записується максимальне абсолютне значення.
Len кількість елементів у векторі.
pIndx вказівник на змінну для індексу елемента з максимальним абсолютним значенням.

Опис:
Функція обчислює максимальне абсолютне значення елементів вхідного вектора ***pSrc***, записує його у ***pMaxAbs*** та повертає через ***pIndx*** індекс відповідного елемента. Якщо існує кілька елементів з однаковим максимальним значенням за модулем, повертається індекс першої появи з початку вектора.

Min

Повертає мінімальне значення елементів вектора.

Синтаксис:

IppStatus ippsMin_16s(const Ipp16s* pSrc, int len, Ipp16s* pMin);
IppStatus ippsMin_32s(const Ipp32s* pSrc, int len, Ipp32s* pMin);
IppStatus ippsMin_32f(const Ipp32f* pSrc, int len, Ipp32f* pMin);
IppStatus ippsMin_64f(const Ipp64f* pSrc, int len, Ipp64f* pMin);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc — вказівник на вхідний вектор.
pMin — вказівник на змінну, у яку записується мінімальне значення.
len — кількість елементів у векторі.

Опис:

Ця функція знаходить мінімальне значення серед елементів вхідного вектора ***pSrc*** і записує результат у змінну ***pMin***.

Повернені значення:
ippStsNoErr — вказує на відсутність помилки.
ippStsNullPtrErr — вказує на помилку, коли ***pSrc*** або ***pMin*** дорівнює ***NULL***.
ippStsSizeErr — позначає помилку, коли ***len*** менше або дорівнює 0.

Приклад:
Нижче наведено приклад використання функції ***ippsMin***:

```
#include <stdio.h>
#include "ipps.h"

void min_example(void) {
   Ipp16s src[5] = { 1, -2, 3, 8, -6 };
   Ipp16s min = 0;
   int len = 5;

   IppStatus st = ippsMin_16s(src, len, &min);
   if (st == ippStsNoErr)
      printf("min = %d\n", (int)min);
   else
      printf("ippsMin_16s failed: %d\n", st);
}

min = -6
```

MinIndx

Повертає мінімальне значення вектора та індекс мінімального елемента.

Синтаксис:
IppStatus ippsMinIndx_16s(const Ipp16s* pSrc, int len, Ipp16s* pMin, int* pIndx);
IppStatus ippsMinIndx_32s(const Ipp32s* pSrc, int len, Ipp32s* pMin, int* pIndx);
IppStatus ippsMinIndx_32f(const Ipp32f* pSrc, int len, Ipp32f* pMin, int* pIndx);
IppStatus ippsMinIndx_64f(const Ipp64f* pSrc, int len, Ipp64f* pMin, int* pIndx);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc вказівник на вхідний вектор.
pMin вказівник на змінну, у яку записується мінімальне значення.
len кількість елементів у векторі.
pIndx вказівник на змінну для індексу мінімального елемента.

Опис:

Функція знаходить мінімальне значення серед елементів вхідного вектора ***pSrc*** і записує його у ***pMin***. Якщо ***pIndx*** не дорівнює ***NULL***, функція також повертає через pIndx індекс мінімального елемента. Якщо існує кілька елементів з однаковим мінімальним значенням, повертається індекс першої появи з початку вектора.

Повернені значення:

ippStsNoErr успішне завершення без помилок.
ippStsNullPtrErr один із вказівників (pSrc, pMin або pIndx при його використанні) дорівнює NULL.
ippStsSizeErr значення len менше або дорівнює 0.

MinAbs

Повертає мінімальне абсолютне значення елементів вектора.

Синтаксис:

IppStatus ippsMinAbs_16s(const Ipp16s* pSrc, int len, Ipp16s* pMinAbs);
IppStatus ippsMinAbs_32s(const Ipp32s* pSrc, int len, Ipp32s* pMinAbs);
IppStatus ippsMinAbs_16f(const Ipp16f* pSrc, int len, Ipp16f* pMinAbs);
IppStatus ippsMinAbs_32f(const Ipp32f* pSrc, int len, Ipp32f* pMinAbs);
IppStatus ippsMinAbs_64f(const Ipp64f* pSrc, int len, Ipp64f* pMinAbs);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc вказівник на вхідний вектор.
pMinAbs вказівник на змінну, у яку записується мінімальне абсолютне значення.
len кількість елементів у векторі.

Опис:

Ця функція знаходить мінімальне абсолютне значення серед елементів вхідного вектора ***pSrc*** та зберігає результат у ***pMinAbs***. Для від'ємних чисел використовується абсолютне значення, тобто модуль.

Повернені значення:

ippStsNoErr функція виконана успішно, помилки відсутні.

ippStsNullPtrErr вказує на помилку, коли ***pSrc*** або ***pMinAbs*** дорівнює ***NULL***.

ippStsSizeErr позначає помилку, коли ***len*** менше або дорівнює 0.

MinAbsIndx

Повертає мінімальне абсолютне значення вектора та індекс відповідного елемента.

Синтаксис:

IppStatus ippsMinAbsIndx_16s(const Ipp16s* pSrc, int len, Ipp16s* pMinAbs, int* pIndx);
IppStatus ippsMinAbsIndx_32s(const Ipp32s* pSrc, int len, Ipp32s* pMinAbs, int* pIndx);

Включені файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc вказівник на вхідний вектор.

pMinAbs вказівник на змінну, у яку записується мінімальне абсолютне значення.

len кількість елементів у векторі.

pIndx вказівник на змінну для індексу елемента з мінімальним абсолютним значенням.

Опис:

Функція обчислює мінімальне абсолютне значення елементів вхідного вектора ***pSrc***, записує його у ***pMinAbs*** та повертає через ***pIndx*** індекс відповідного елемента. Якщо існує кілька елементів з однаковим мінімальним абсолютним значенням, повертається індекс першої появи з початку вектора.

MinMax

Повертає мінімальне та максимальне значення вектора.

Синтаксис:

IppStatus ippsMinMax_8u(const Ipp8u* pSrc, int len, Ipp8u* pMin, Ipp8u* pMax);
IppStatus ippsMinMax_16u(const Ipp16u* pSrc, int len, Ipp16u* pMin, Ipp16u* pMax);
IppStatus ippsMinMax_16s(const Ipp16s* pSrc, int len, Ipp16s* pMin, Ipp16s* pMax);
IppStatus ippsMinMax_32u(const Ipp32u* pSrc, int len, Ipp32u* pMin, Ipp32u* pMax);
IppStatus ippsMinMax_32s(const Ipp32s* pSrc, int len, Ipp32s* pMin, Ipp32s* pMax);
IppStatus ippsMinMax_32f(const Ipp32f* pSrc, int len, Ipp32f* pMin, Ipp32f* pMax);
IppStatus ippsMinMax_64f(const Ipp64f* pSrc, int len, Ipp64f* pMin, Ipp64f* pMax);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc	вказівник на вхідний вектор.
pMin	вказівник на змінну, у яку записується мінімальне значення.
pMax	вказівник на змінну, у яку записується максимальне значення.
Len	кількість елементів у векторі.

Опис:
Ця функція обчислює мінімальне та максимальне значення серед елементів вхідного вектора ***pSrc*** і записує результати у змінні ***pMin*** і ***pMax*** відповідно.

Повернені значення:

ippStsNoErr	функція виконана успішно, помилки відсутні.
ippStsNullPtrErr	вказівник pSrc, pMin або pMax дорівнює NULL.
ippStsSizeErr	значення len менше або дорівнює 0.

Mean

Обчислює середнє значення елементів вектора.

Синтаксис:

IppStatus ippsMean_32f(const Ipp32f* pSrc, int len, Ipp32f* pMean, IppHintAlgorithm hint);
IppStatus ippsMean_32fc(const Ipp32fc* pSrc, int len, Ipp32fc* pMean, IppHintAlgorithm hint);

IppStatus ippsMean_64f(const Ipp64f* pSrc, int len, Ipp64f* pMean);
IppStatus ippsMean_64fc(const Ipp64fc* pSrc, int len, Ipp64fc* pMean);
IppStatus ippsMean_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16s* pMean, int scaleFactor);
IppStatus ippsMean_32s_Sfs(const Ipp32s* pSrc, int len, Ipp32s* pMean, int scaleFactor);
IppStatus ippsMean_16sc_Sfs(const Ipp16sc* pSrc, int len, Ipp16sc* pMean, int scaleFactor);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc вказівник на вхідний вектор.
pMean вказівник на змінну, у якій повертається середнє значення.
len кількість елементів у векторі.
hint підказка реалізації для варіантів із плаваючою крапкою; дозволяє обрати швидший (менш точний) або точніший (повільніший) алгоритм, значення описані в IppHintAlgorithm.
scaleFactor коефіцієнт масштабування для цілих варіантів (див. «Integer Scaling»).

Опис:
Функція обчислює середнє значення вектора ***pSrc*** і записує результат у ***pMean***. Для дійсних типів середнє визначається як

$$pMean = \frac{1}{len} \sum_{n=0}^{len-1} pSrc[n]$$

Для комплексних типів повертається комплексне середнє: окремо усереднюються дійсні та уявні частини. Аргумент ***hint*** впливає на вибір реалізації (швидкість проти точності) у варіантах із плаваючою крапкою. Для цілих типів результат може виходити за межі діапазону, тому застосовується масштабування відповідно до значення ***scaleFactor***.

StdDev

Обчислює стандартне відхилення елементів вектора.

Синтаксис:

IppStatus ippsStdDev_32f(const Ipp32f* pSrc, int len, Ipp32f* pStdDev, IppHintAlgorithm hint);
IppStatus ippsStdDev_64f(const Ipp64f* pSrc, int len, Ipp64f* pStdDev);
IppStatus ippsStdDev_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16s* pStdDev, int scaleFactor);
IppStatus ippsStdDev_16s32s_Sfs(const Ipp16s* pSrc, int len, Ipp32s* pStdDev, int scaleFactor);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc вказівник на вхідний вектор.
pStdDev вказівник на змінну, у яку записується стандартне відхилення.
Len кількість елементів у векторі.
Hint підказка реалізації для варіантів із плаваючою крапкою; дозволяє обрати швидший (менш точний) або точніший (повільніший) алгоритм, значення визначені в IppHintAlgorithm.
scaleFactor коефіцієнт масштабування для цілих варіантів.

Опис:
Функція обчислює стандартне відхилення значень вектора ***pSrc*** і записує результат у ***pStdDev***. Довжина вектора має бути щонайменше 2. Для цілих типів можливе переповнення проміжних обчислень; щоб керувати діапазоном і точністю фіксованої крапки, застосовується масштабування відповідно до значення scaleFactor. Аргумент ***hint*** у варіантах із плаваючою крапкою дозволяє обрати між пріоритетом швидкодії та точності.

Приклад:
```
#include <stdio.h>
#include <stdlib.h>
#include "ipps.h"

void stdev_example(void) {
   int len = 1000;
   Ipp32f* x = ippsMalloc_32f(len);
   if (!x) { printf("alloc failed\n"); return; }

   for (int i = 0; i < len; ++i)
      x[i] = (Ipp32f)rand() / (Ipp32f)RAND_MAX;

   Ipp32f stdev = 0.0f;
   IppStatus st = ippsStdDev_32f(x, len, &stdev, ippAlgHintFast);
   if (st == ippStsNoErr)
      printf("stdev = %f\n", (double)stdev);
   else
      printf("ippsStdDev_32f failed: %d\n", st);
   ippsFree(x);
}
```

5. РЕАЛІЗАЦІЯ ДИСКРЕТНИХ ПЕРЕТВОРЕНЬ

У бібліотеці Intel® IPP реалізовані функції, які виконують дискретні перетворення: Фур'є, дискретні косинусні перетворення (DCT), перетворення Хартлі, Гільберта, Уолша-Адамара та вейвлет-перетворення. Для наведених перетворень вхідні сигнали можуть бути дійснозначними та комплекснозначними.

5.1. Функції перетворення Фур'є

До функції перетворень Фур’є належать функці, які виконують швидке перетворення Фур'є (ШПФ, Fast Fourier Transform (FFT)) та дискретне перетворення Фур'є (ДПФ, Discrete Fourier transform (DFT)) відліків сигналу. Основні функції мають варіанти для підтримки різних вимог програми.

Спеціальні аргументи:

У функціях перетворення Фур'є необхідно вказати аргументи ***flag*** і ***hint***.

flag аргумент задає метод нормалізації результату. У наступній таблиці перелічені можливі значення для ***flag*** аргумента, із яких вказується тільки одне. Коефіцієнти A та B є множниками, що використовуються при розрахунку DFT.

flag аргументи для функцій перетворення Фур'є:

Значення	A	B	Опис
IPP_FFT_DIV_FWD_BY_N	$\frac{1}{N}$	1	Виконується пряме перетворення з нормалізацією $\frac{1}{N}$.
IPP_FFT_DIV_INV_BY_N	1	$\frac{1}{N}$	Виконується обернене перетворення з нормалізацією $\frac{1}{N}$.
IPP_FFT_DIV_BY_SQRT_N	$\frac{1}{\sqrt{N}}$	$\frac{1}{\sqrt{N}}$	Пряме і обернене перетворення здійснюються з нормалізацією $\frac{1}{\sqrt{N}}$.
IPP_FFT_NODIV_BY_ANY	1	1	Пряме або обернене перетворення здійснюється без нормалізації $\frac{1}{N}$ або $\frac{1}{\sqrt{N}}$.

5.2. Упаковані формати

Цей розділ описує основні упаковані формати ***Perm***, ***Pack***, і ***CCS***, які використовуються у функціях перетворення Фур'є.

Perm формат зберігає значення в тій послідовності, в якій їх використовують алгоритми перетворення Фур'є. Це найбільш природний

спосіб зберігання значень для алгоритмів перетворення Фур'є. ***Perm*** формат є довільною перестановкою ***Pack*** формату. Важливою характеристикою ***Perm*** формату є те, що дійсна та уявна частини даного зразка не обов'язково повинні бути суміжними.Для вхідного сигналу непарної довжини ***perm*** і ***pack format*** однакові.

CCS формат зберігає значення першої половини вихідного комплексного сигналу, отриманого в результаті прямого перетворення Фур'є.Розташування результатів прямого перетворення Фур’є в упакованих форматах (парна довжина N):

Index	Зміст	Примітки
0	R_0	Реальна частина нульової частоти
1	R_1	Реальна частина першого спектру
2	I_1	Уявна частина першого спектру
3	R_2	Реальна частина другої спектру
…	…	…
N/2 – 1	I_{N/2 – 1}	Уявна частина передсередньої частоти
N/2	R_{N/2}	Реальна частина Nyquist (середня)
N/2 + 1	I_{N/2 – 1} (зворотна)	Уявна частина симетричного спектра
…	…	…
N – 1	I_1 (зворотна)	Уявна частина першого спектру (знак мінус)

Розташування результатів прямого перетворення Фур’є в упакованих форматах (непарна довжина N):

Index	Pack / Perm / CCS	Пояснення
0	R_0	Дійсна частина нульової (DC) частоти
1	R_1	Дійсна частина першого спектру
2	I_1	Уявна частина першого спектру
3	R_2	Дійсна частина другої спектру
4	I_2	Уявна частина другої спектру
…	…	…
(N–1)/2	R_{(N–1)/2}	Дійсна частина середньої частоти
(N+1)/2 (індекс N)	I_{(N–1)/2}	Уявна частина середньої частоти як останній елемент

4.2 Функції перетворення формату

Наступні функції ***ippsConjPack***, ***ippsConjPerm***, і ***ippsConjCcs*** перетворюють дані із упакованих форматів у звичайний комплексний формат даних, використовуючи властивість FFT symmetry для перетворення дійсних даних. Вихідні дані є комплексними, довжина вихідного масиву визначається кількістю комплексних елементів у вихідному векторі. Зверніть увагу, що розмір вихідного масиву вдвічі більший за розмір вхідного масиву. Дані, що зберігаються в ***CCS*** format вимагають більший масив, ніж інші формати. Масиви з парною та непарною довжиною мають деякі специфічні особливості, що обговорюються для кожної функції окремо.

ConjPack

Перетворює дані в Pack-форматі у комплексні числа.

Синтаксис:
IppStatus ippsConjPack_32fc (const Ipp32f* pSrc, Ipp32fc* pDst, int lenDst);
IppStatus ippsConjPack_64fc (const Ipp64f* pSrc, Ipp64fc* pDst, int lenDst);
IppStatus ippsConjPack_32fc_I (Ipp32fc* pSrcDst, int lenDst);
IppStatus ippsConjPack_64fc_I (Ipp64fc* pSrcDst, int lenDst);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc вказівник на вихідний вектор у Pack-форматі.
pDst вказівник на вектор призначення у комплексному форматі.
pSrcDst вказівник на джерело та вектор призначення (для операції in-place).
lenDst кількість елементів у векторі призначення.

Опис:
Ця функція перетворює дані у Pack-форматі вектору pSrc у комплексний формат і зберігає результат у pDst. Варіанти з суфіксом _I (наприклад, ippsConjPack_32fc_I) виконують операцію на місці — дані у pSrcDst перетворюються без створення окремого вектора призначення.

У таблиці нижче наведено приклад розпакування Pack-даних: стовпець Data містить дійсні вхідні дані, які підлягають прямому перетворенню БПФ;

стовпець Packed — результат після перетворення у Pack-формат; стовпець Extended — відповідний комплексний вектор після застосування ippsConjPack. Стовпець Length вказує номер елементів у векторі. Приклади розпакування з формату Pack:

Data	Packed	Extended	Length
FFT([1])	1	{1, 0}	1
FFT([1 2])	3, −1	{3, 0}, {−1, 0}	2
FFT([1 2 3])	6, −1.5, 0.86	{6, 0}, {−1.5, 0.86}, {−1.5, −0.86}	3
FFT([1 2 3 9])	15, −7, −2, 7	{15, 0}, {−2, 7}, {−7, 0}, {−2, −7}	4

Data — початковий вектор дійсних даних.

Packed — упакований результат прямого перетворення Фур'є (Pack format).

Extended — відновлений комплексний вектор після застосування ippsConjPack.

Length — кількість елементів у вхідному векторі.

ConjPerm

Перетворює дані у Perm-форматі до комплексного формату даних.

Синтаксис:

IppStatus ippsConjPerm_32fc (const Ipp32f* pSrc, Ipp32fc* pDst, int lenDst);
IppStatus ippsConjPerm_64fc (const Ipp64f* pSrc, Ipp64fc* pDst, int lenDst);
IppStatus ippsConjPerm_32fc_I (Ipp32fc* pSrcDst, int lenDst);
IppStatus ippsConjPerm_64fc_I (Ipp64fc* pSrcDst, int lenDst);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc вказівник на вхідний вектор у Perm-форматі.

pDst вказівник на вектор призначення у комплексному форматі.

pSrcDst вказівник на вектор, який одночасно є джерелом і призначенням для операції in-place.

lenDst кількість елементів у векторі призначення.

Опис:

Функція перетворює дані у ***Perm***-форматі з вектора ***pSrc*** у комплексний формат і зберігає результат у ***pDst***. Варіанти з суфіксом ***_I*** виконують перетворення на місці (in-place): дані у ***pSrcDst*** інтерпретуються як ***Perm*** і перезаписуються комплексними значеннями у тому ж буфері. Для ілюстрації зазвичай подається таблиця: стовпець ***Data*** містить вихідні дійсні дані перед прямим ***FFT***, стовпець ***Packed/Perm*** — результат упакування у ***Perm***-формат, стовпець ***Extended*** — відповідний комплексний вектор після застосування ippsConjPerm, стовпець ***Length*** — кількість елементів.

Приклади упакованих даних, отриманих ШПФ

Data	Packed	Extended	Length
FFT([1])	1, 0	{1, 0}	1
FFT([1 2])	3, 0, −1, 0	{3, 0}, {−1, 0}	2
FFT([1 2 3])	6, 0, −1.5, 0.86	{6, 0}, {−1.5, 0.86}, {−1.5, −0.86}	3
FFT([1 2 3 9])	15, 0, −2, 7, −7, 0	{15, 0}, {−2, 7}, {−7, 0}, {−2, −7}	4

ConjCcs

Перетворює дані із CCS формату до комплексного формату даних.

Синтаксис:

IppStatus ippsConjCcs_32fc (const Ipp32f* pSrc, Ipp32fc* pDst, int lenDst);
IppStatus ippsConjCcs_64fc (const Ipp64f* pSrc, Ipp64fc* pDst, int lenDst);
IppStatus ippsConjCcs_32fc_I (Ipp32fc* pSrcDst, int lenDst);
IppStatus ippsConjCcs_64fc_I (Ipp64fc* pSrcDst, int lenDst);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc Вказівник на вихідний вектор.
pDst Вказівник на вектор призначення.
pSrcDst Вказівник на джерело та вектор призначення
lenDst Кількість елементів у векторі.

Опис:

Ця функція перетворює дані у форматі ***CCS*** у векторі ***pSrc*** до комплексного формату даних та зберігає результати у ***pDst***.

Функція “на місці” ***ippsConjCcs*** перетворює дані у форматі ***CCS*** в ***pSrcDst*** до комплексного формату та зберігає результати в ***pSrcDst***.

У наступній таблиці наведено приклади розпакування з формату ***CCS***. Стовпець ***Data*** містить вихідні дійсні дані, які перетворюються за допомогою прямого БПФ у упаковані дані. Стовпець ***Packed*** містить упаковані дійсні дані. Результатом є комплексний вектор даних у стовпці ***Extended***. Кількість елементів у векторі вказано у стовпці ***Length***. Дані, що зберігаються в ***CCS*** форматі, містять два дійсних елементи на один комплексний.

Приклади розпакування з формату **CCS**

Data	Packed	Extended	Length
FFT([1])	1, 0	{1, 0}	1
FFT([1 2])	3, 0, -1, 0	{3, 0}, {-1, 0}	2
FFT([1 2 3])	6, 0, -1.5, 0.86	{6, 0}, {-1.5, 0.86}, {-1.5, -0.86}	3
FFT([1 2 3 9])	15, 0, -2, 7, -7, 0, 0	{15, 0}, {-2, 7}, {-7, 0}, {-2, -7}	4

5.3. Функції множення упакованих даних

Функції, описані в цьому розділі, виконують комплексне множення векторів, що зберігаються в ***Pack*** або ***Perm*** форматах. Ці функції використовуються з функцією ***ippsFFTFwd*** і ***ippsFFTInv*** для виконання швидкої згортки на дійсних сигналах.

Стандартна функція множення вектора ***ippsMul*** не може використовуватися для множення ***Pack*** або ***Perm*** форматувати вектори, оскільки: Два дійсні зразки зберігаються в ***Pack*** форматі.

Perm формат може не поєднувати дійсні частини сигналу з відповідними уявними частинами.

MulPack

Множить відповідні елементи двох векторів, що зберігаються у форматі Pack, і зберігає результати у векторі призначення.

Синтаксис:

IppStatus ippsMulPack_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsMulPack_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc1 Вказівник на перший вхідний вектор.
pSrc2 Вказівник на другий вхідний вектор.
pDst Вказівник на вихідний вектор, у якому зберігаються результати множення.
len Кількість елементів у векторах.

Опис:
Ця функція виконує поелементне множення двох векторів, представлених у форматі Pack, і зберігає результати у векторі призначення pDst.
Для кожного елемента виконується операція:

$$pDst[n] = pSrc1[n] \times pSrc2[n], \quad 0 \le n < len$$

MulPerm

Множить відповідні елементи двох векторів, що зберігаються у форматі Perm, і зберігає результати у векторі призначення.

Синтаксис:
IppStatus ippsMulPerm_32f(const Ipp32f* pSrc1, const Ipp32f* pSrc2, Ipp32f* pDst, int len);
IppStatus ippsMulPerm_64f(const Ipp64f* pSrc1, const Ipp64f* pSrc2, Ipp64f* pDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc1 Вказівник на перший вхідний вектор.
pSrc2 Вказівник на другий вхідний вектор.
pDst Вказівник на вектор призначення.
len Кількість елементів у векторах.

Опис:

Функція виконує поелементне множення двох векторів, представлених у форматі Perm, зберігаючи результати у вихідному векторі pDst.

Для кожного елемента виконується операція:

$$pDst[n] = pSrc1[n] \times pSrc2[n], \quad 0 \leq n < len$$

MulPackConj

Множить елементи вектора на елементи комплексного спряженого вектора, що зберігається у форматі Pack, виконуючи операцію без створення копій (in-place).

Синтаксис:

IppStatus ippsMulPackConj_32f_I(const Ipp32f* pSrc, Ipp32f* pSrcDst, int len);
IppStatus ippsMulPackConj_64f_I(const Ipp64f* pSrc, Ipp64f* pSrcDst, int len);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc Вказівник на вхідний вектор.

pSrcDst Вказівник на джерело та вектор призначення для операції на місці.

len Кількість елементів у векторі.

Опис:

Ця функція виконує поелементне множення елементів вектора ***pSrcDst*** на комплексно спряжені значення вектора ***pSrc***, перетвореного з ***Pack*** формату.

Результат зберігається у тому ж векторі ***pSrcDst***.

Математично операція визначається як:

$$pSrcDst[n] = pSrcDst[n] \times \overline{pSrc[n]}, \quad 0 \leq n < len$$

де $\overline{pSrc[n]}$ — це комплексно спряжене значення елемента $pSrc[n]$.

5.4. Функції швидкого перетворення Фур'є

Функції, описані в цьому розділі, обчислюють пряме і обернене швидке перетворення Фур'є дійсних і комплексних сигналів. ШПФ схожий на дискретне перетворення Фур'є (ДПФ), але значно швидший. Довжина вектора, перетвореного ШПФ, повинна бути рівною 2.

Щоб використовувати функції ШПФ, ініціалізуйте структуру специфікації, яка містить такі дані, як таблиці коефіцієнтів двійника. Функції ініціалізації створюють специфікації як для прямого, так і для оберненого перетворення. Таким чином, кількість попередніх розрахунків зменшується, а загальна ефективність збільшується.

hint аргумент, переданий функціям ініціалізації, пропонує використовувати спеціальний алгоритм, швидший або точніший.

flag аргумент задає метод нормалізації результату.

Щоб ініціалізувати структуру специфікації ШПФ, використовуйте ***ippsFFTInit_***R і ***ippsFFTInit_C*** функції. Перед використанням цих функцій вам потрібно обчислити розмір структури специфікації за допомогою ***ippsFFTGetSize_R*** і ***ippsFFTGetSize_C*** відповідно.

Комплексний сигнал може бути представлений у вигляді єдиного масиву, що містить комплексні елементи, або двох окремих масивів, що містять дійсну та уявну частини. Результат ШПФ можна запакувати ***Perm***, ***Pack***, або ***CCS*** формат.

Ви можете пришвидшити ШПФ за допомогою зовнішнього буфера. Використання зовнішнього буфера може покращити продуктивність, уникаючи розподілу та вивільнення внутрішніх буферів та зберігання даних у кеші. Розмір зовнішнього буфера повертається ***ippsFFTInit_R*** і ***ippsFFTInit_C*** функції.

Якщо зовнішній буфер не вказаний (для параметра кореспондента встановлено значення ***NULL***), тоді сама функція ШПФ виділяє пам’ять, необхідну для роботи.

FFTInit_R, FFTInit_C

Ініціалізує структуру специфікації ШПФ (FFT) для дійсних і комплексних сигналів.

Синтаксис:

Випадок 1. Операція над дійсним сигналом:

IppStatus ippsFFTInit_R_32f(IppsFFTSpec_R_32f** ppFFTSpec, int order, int flag, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);

IppStatus ippsFFTInit_R_64f(IppsFFTSpec_R_64f** ppFFTSpec, int order, int flag, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);

Випадок 2. Операція над комплексним сигналом:

IppStatus ippsFFTInit_C_32f(IppsFFTSpec_C_32f** ppFFTSpec, int order, int flag, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
IppStatus ippsFFTInit_C_64f(IppsFFTSpec_C_64f** ppFFTSpec, int order, int flag, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
IppStatus ippsFFTInit_C_32fc(IppsFFTSpec_C_32fc** ppFFTSpec, int order, int flag, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);
IppStatus ippsFFTInit_C_64fc(IppsFFTSpec_C_64fc** ppFFTSpec, int order, int flag, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
order Порядок ШПФ. Довжина вхідного сигналу визначається як N = 2^order.
flag Вказує метод нормалізації результату. Можливі значення параметра flag наведено у розділі Аргументи flag та hint.
hint Параметр застарілий. Для сучасних реалізацій встановіть значення ippAlgHintNone.
ppFFTSpec Подвійний вказівник на створювану структуру специфікації ШПФ.
pSpec Вказівник на область пам’яті для зберігання специфікації ШПФ.
pSpecBuffer Вказівник на робочий буфер.

Опис:
Ці функції ініціалізують структуру специфікації ШПФ ppFFTSpec із заданими параметрами:
order — визначає довжину перетворення (2^order);
flag — задає тип нормалізації (наприклад, пряме чи зворотне ділення на N, або відсутність нормалізації);
hint — алгоритмічна підказка (для сумісності).

Перед викликом цих функцій необхідно визначити розміри структури специфікації та робочого буфера, використовуючи функції ***ippsFFTGetSize_R*** або ***ippsFFTGetSize_C***.
Якщо функція ***ippsFFTGetSize*** повертає ***pSpecBufferSize = 0***, тоді параметр ***pSpecBuffer*** може бути ***NULL***.

Суфікс у назві функції визначає тип сигналу:
ippsFFTInit_R — для дійсних сигналів (real);
ippsFFTInit_C — для комплексних сигналів (complex).

Повернені значення:
ippStsNoErr Вказує на відсутність помилки.
ippStsNullPtrErr Вказує на помилку, коли один або кілька вказівників мають значення NULL.
ippStsFftOrderErr Вказує на помилку, коли параметр order має недопустиме значення.
ippStsFftFlagErr Вказує на помилку, коли параметр flag має недопустиме значення.

FFTGetSize_R, FFTGetSize_C

Обчислює розміри структури специфікації ШПФ (FFT) та необхідні робочі буфери.

Синтаксис:
Випадок 1. Операція над дійсним сигналом:

IppStatus ippsFFTGetSize_R_32f(int order, int flag, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);
IppStatus ippsFFTGetSize_R_64f(int order, int flag, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);

Випадок 2. Операція над комплексним сигналом:

IppStatus ippsFFTGetSize_C_32f(int order, int flag, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);
IppStatus ippsFFTGetSize_C_64f(int order, int flag, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);
IppStatus ippsFFTGetSize_C_32fc(int order, int flag, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);
IppStatus ippsFFTGetSize_C_64fc(int order, int flag, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

order Порядок ШПФ. Довжина вхідного сигналу визначається як $N = 2^{\text{order}}$

flag Вказує метод нормалізації результату.

hint Параметр застарілий. Для сучасних реалізацій встановіть ***ippAlgHintNone***.

pSpecSize Вказівник на змінну, куди буде записано розмір структури специфікації ШПФ.

pSpecBufferSize Вказівник на змінну, куди буде записано розмір робочого буфера, необхідного для ініціалізації (функцій ***ippsFFTInit_R*** або ***ippsFFTInit_C***).

pBufferSize Вказівник на змінну, куди буде записано розмір зовнішнього буфера, необхідного для виконання прямого або зворотного ШПФ.

Опис:

Ці функції обчислюють такі параметри:

pSpecSize — розмір структури специфікації ШПФ;

pSpecBufferSize — розмір робочого буфера, необхідного для функцій ініціалізації (***ippsFFTInit_R*** або ***ippsFFTInit_C***);

pBufferSize — розмір зовнішнього робочого буфера, необхідного для виконання операцій ***ippsFFTFwd*** і ***ippsFFTInv***.

Суфікси у назві функцій визначають тип даних:

ippsFFTGetSize_R — для дійсних сигналів (***real***);

ippsFFTGetSize_C — для комплексних сигналів (***complex***).

Повернені значення:

ippStsNoErr Вказує на відсутність помилки.

ippStsNullPtrErr Вказує на помилку, коли будь-який з вказівників має значення ***NULL***.

ippStsFftOrderErr Вказує на помилку, коли параметр ***order*** має недопустиме значення.

ippStsFftFlagErr Вказує на помилку, коли параметр ***flag*** має недопустиме значення.

FFTFwd_CtoC

Обчислює пряме швидке перетворення Фур’є (ШПФ) комплексного сигналу.

Синтаксис:

Випадок 1. Операція над дійсними масивами, що містять комплексні сигнали:

IppStatus ippsFFTFwd_CToC_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDstRe, Ipp32f* pDstIm, const IppsFFTSpec_C_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTFwd_CToC_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDstRe, Ipp64f* pDstIm, const IppsFFTSpec_C_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція над комплексними масивами (out-of-place):

IppStatus ippsFFTFwd_CToC_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, const IppsFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTFwd_CToC_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, const IppsFFTSpec_C_64fc* pFFTSpec, Ipp8u* pBuffer);

Випадок 3. Операція на місці над дійсними масивами:

IppStatus ippsFFTFwd_CToC_32f_I(Ipp32f* pSrcDstRe, Ipp32f* pSrcDstIm, const IppsFFTSpec_C_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTFwd_CToC_64f_I(Ipp64f* pSrcDstRe, Ipp64f* pSrcDstIm, const IppsFFTSpec_C_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 4. Операція на місці над комплексними масивами:

IppStatus ippsFFTFwd_CToC_32fc_I(Ipp32fc* pSrcDst, const IppsFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTFwd_CToC_64fc_I(Ipp64fc* pSrcDst, const IppsFFTSpec_C_64fc* pFFTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pFFTSpec	Вказівник на структуру специфікації ШПФ.
pSrc	Вказівник на вхідний масив комплексних значень.
pDst	Вказівник на вихідний масив комплексних значень.
pSrcRe	Вказівник на вхідний масив, що містить дійсні частини сигналу.
pSrcIm	Вказівник на вхідний масив, що містить уявні частини сигналу.
pDstRe	Вказівник на вихідний масив, що містить дійсні частини сигналу.

pDstIm Вказівник на вихідний масив, що містить уявні частини сигналу.

pSrcDst Вказівник на масив для операції на місці, який містить комплексні значення.

pSrcDstRe Вказівник на масив дійсних частин сигналу для операції на місці.

pSrcDstIm Вказівник на масив уявних частин сигналу для операції на місці.

pBuffer Вказівник на зовнішній робочий буфер.

Опис:

Ця функція обчислює пряме швидке перетворення Фур'є (FFT) комплексного сигналу відповідно до параметрів специфікації ***pFFTSpec***, які включають порядок перетворення (***order***), тип нормалізації (***flag***) та алгоритмічну підказку (***hint***).

Перед використанням цієї функції необхідно ініціалізувати структуру специфікації ШПФ за допомогою функції ***ippsFFTInit_C***.

Функції з суфіксом ***fc*** працюють із комплексними масивами ***pSrc*** і зберігають результат у ***pDst***, а їх варіанти ***_I*** виконують операцію на місці з використанням ***pSrcDst***.

Функції з суфіксом ***f*** або ***d*** працюють із дійсними типами даних (відповідно float або double) і обробляють сигнали, представлені окремими масивами дійсних (***pSrcRe***) та уявних (***pSrcIm***) частин. Результат зберігається окремо у ***pDstRe*** та ***pDstIm***. Для операцій на місці використовуються масиви ***pSrcDstRe*** та ***pSrcDstIm***.

Для підвищення продуктивності рекомендується використовувати зовнішній робочий буфер ***pBuffer***. Його розмір можна визначити за допомогою функцій ***ippsFFTGetBufSize_C*** або ***ippsFFTGetSize_C***. Якщо ***pBuffer=NULL***, функція самостійно виділяє необхідну пам'ять, що може зменшити швидкодію.

Зовнішній буфер можна багаторазово використовувати для кількох викликів FFT, що особливо ефективно для невеликих розмірів перетворення.

FFTInv_CtoC

Обчислює обернене швидке перетворення Фур'є (ШПФ) комплексного сигналу.

Синтаксис:

Випадок 1. Операція з дійсними масивами, що представляють комплексні сигнали:

IppStatus ippsFFTInv_CToC_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDstRe, Ipp32f* pDstIm, const IppsFFTSpec_C_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_CToC_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDstRe, Ipp64f* pDstIm, const IppsFFTSpec_C_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція над комплексними масивами (out-of-place):

IppStatus ippsFFTInv_CToC_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, const IppsFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_CToC_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, const IppsFFTSpec_C_64fc* pFFTSpec, Ipp8u* pBuffer);

Випадок 3. Операція на місці над дійсними масивами:

IppStatus ippsFFTInv_CToC_32f_I(Ipp32f* pSrcDstRe, Ipp32f* pSrcDstIm, const IppsFFTSpec_C_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_CToC_64f_I(Ipp64f* pSrcDstRe, Ipp64f* pSrcDstIm, const IppsFFTSpec_C_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 4. Операція на місці над комплексними масивами:

IppStatus ippsFFTInv_CToC_32fc_I(Ipp32fc* pSrcDst, const IppsFFTSpec_C_32fc* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_CToC_64fc_I(Ipp64fc* pSrcDst, const IppsFFTSpec_C_64fc* pFFTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pFFTSpec	Вказівник на структуру специфікації ШПФ.
pSrc	Вказівник на вхідний масив, що містить комплексні значення.
pDst	Вказівник на вихідний масив, що містить комплексні значення.
pSrcRe	Вказівник на масив, що містить дійсні частини вхідного сигналу.

pSrcIm Вказівник на масив, що містить уявні частини вхідного сигналу.

pDstRe Вказівник на масив, що містить дійсні частини вихідного сигналу.

pDstIm Вказівник на масив, що містить уявні частини вихідного сигналу.

pSrcDst Вказівник на масив для операції на місці, який містить комплексні значення.

pSrcDstRe Вказівник на масив дійсних частин для операції на місці.

pSrcDstIm Вказівник на масив уявних частин для операції на місці.

pBuffer Вказівник на зовнішній робочий буфер.

Опис:

Ця функція обчислює зворотне швидке перетворення Фур'є (IFFT) комплексного сигналу згідно з параметрами специфікації ***pFFTSpec***, які включають порядок перетворення (***order***), тип нормалізації (***flag***) та алгоритмічну підказку (***hint***).

Перед викликом цієї функції структура специфікації ШПФ повинна бути ініціалізована за допомогою функції ***ippsFFTInit_C***.

Функції з суфіксом ***fc*** працюють із комплексними масивами ***pSrc*** і зберігають результат у ***pDst***. Варіанти ***_I*** виконують операцію на місці, використовуючи ***pSrcDst***.

Функції з суфіксом ***f*** або ***d*** обробляють сигнали, представлені окремими дійсними (***pSrcRe***) та уявними (***pSrcIm***) частинами, зберігаючи результат окремо у ***pDstRe*** і ***pDstIm***. Для операцій на місці використовуються ***pSrcDstRe*** та ***pSrcDstIm***.

Для підвищення продуктивності допускається використання зовнішнього робочого буфера ***pBuffer***. Його розмір слід попередньо визначити за допомогою функції ***ippsFFTGetSize_C***. Після виділення буфера його можна повторно використовувати для наступних викликів FFT-функцій, що значно зменшує витрати на динамічне виділення пам'яті, особливо для невеликих розмірів перетворення.

Якщо ***pBuffer = NULL***, функція самостійно виділяє необхідну пам'ять для виконання обчислень.

Примітка:

Довжина перетворення (FFT) повинна дорівнювати степеню двійки:

$$N = 2^{\text{order}}$$

FFTFwd_RToPack, FFTFwd_RToPerm, FFTFwd_RToCCS

Обчислює пряме швидке перетворення Фур'є (ШПФ) для дійсного сигналу та зберігає результат у форматах Pack, Perm або CCS.

Синтаксис:

Випадок 1. Робота не на місці, результат у Pack форматі:

IppStatus ippsFFTFwd_RToPack_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToPack_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція на місці, результат у Pack форматі:

IppStatus ippsFFTFwd_RToPack_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToPack_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 3. Робота не на місці, результат у Perm форматі:

IppStatus ippsFFTFwd_RToPerm_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToPerm_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 4. Операція на місці, результат у Perm форматі:

IppStatus ippsFFTFwd_RToPerm_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToPerm_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 5. Робота не на місці, результат у CCS форматі:

IppStatus ippsFFTFwd_RToCCS_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToCCS_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 6. Операція на місці, результат у CCS форматі:

IppStatus ippsFFTFwd_RToCCS_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToCCS_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pFFTSpec Вказівник на структуру специфікації ШПФ.

pSrc Вказівник на вхідний масив дійсних значень.

pDst Вказівник на вихідний масив, що містить упаковані комплексні значення.

pSrcDst Вказівник на вхідний і вихідний масив для роботи на місці.

pBuffer Вказівник на зовнішній робочий буфер.

Опис:

Ці функції обчислюють пряме швидке перетворення Фур'є (FFT) дійсного сигналу відповідно до параметрів специфікації pFFTSpec, що включають порядок перетворення (***order***), тип нормалізації (***flag***) та алгоритмічну підказку (***hint***).

Перед викликом будь-якої з цих функцій потрібно ініціалізувати структуру специфікації ШПФ за допомогою функцій ***ippsFFTInit_R***.

Довжина перетворення повинна дорівнювати степеню двійки:

$$N = 2^{\text{order}}$$

Результат зберігається у форматі ***Pack***, ***Perm*** або ***CCS*** залежно від обраної функції.

ippsFFTFwd_RToPack — зберігає результат у Pack форматі.

ippsFFTFwd_RToPerm — зберігає результат у Perm форматі.

ippsFFTFwd_RToCCS — зберігає результат у CCS форматі.

Для підвищення продуктивності функції можуть використовувати зовнішній робочий буфер ***pBuffer***, щоб уникнути внутрішнього розподілу пам'яті.

Розмір цього буфера визначається за допомогою функції ***ippsFFTGetSize_R***.

Якщо ***pBuffer = NULL***, функція самостійно виділяє пам'ять для виконання обчислень. Використання зовнішнього буфера рекомендовано для швидкодії, особливо при малих розмірах перетворення.

Приклад:

Наведена нижче програма генерує 8-точковий реальний сигнал косинуса, обчислює його пряме БПФ у форматі ***CCS***, далі рахує модуль (Magnitude) перших 4 комплексних спектральних коефіцієнтів і друкує їх. Усі обчислення виконуються за допомогою Intel® IPP (Signal Processing).

```
#include <stdio.h>
#include <math.h>
#include "ipp.h"

#define EXIT_MAIN exitLine:                    /* Мітка для виходу */
```

```
#define check_sts(st) if((st) != ippStsNoErr) goto exitLine; /* Перейти до виходу, якщо функція Intel(R) IPP повернула статус, відмінний від ippStsNoErr */

/* Результати ippMalloc() тут не перевіряються, оскільки функції Intel(R) IPP
   виконують перевірку аргументів і повертають відповідний статус у разі помилки */

int main(void)
{
   int i;
   IppStatus status = ippStsNoErr;
   Ipp32f pSrc[8], pDst[10];       /* Вказівники на вхідний/вихідний масиви */
   Ipp32f pMagn[4];            /* Вказівник на обчислені значення модуля (Magnitude) */
   IppsFFTSpec_R_32f* pSpec = NULL;  /* Вказівник на структуру специфікації FFT */
   /* Вказівники на робочі буфери */
   Ipp8u *pMemInit = NULL, *pBuffer = NULL, *pSpecMem = NULL;
  /* Розміри структури FFT, буфера ініціалізації та робочого буфера */
   int sizeSpec = 0, sizeInit = 0, sizeBuf = 0;

   check_sts( status = ippsFFTGetSize_R_32f(3, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone,
                              &sizeSpec, &sizeInit, &sizeBuf) )

   /* Виділення пам'яті */
   pSpecMem = (Ipp8u*) ippMalloc(sizeSpec);
   pBuffer  = (Ipp8u*) ippMalloc(sizeBuf);
   pMemInit = (Ipp8u*) ippMalloc(sizeInit);

   check_sts( status = ippsFFTInit_R_32f(&pSpec, 3,
IPP_FFT_DIV_INV_BY_N, ippAlgHintNone,
                     pSpecMem, pMemInit) )

   for (i = 0; i < 8; ++i) {
      pSrc[i] = (float)cos(IPP_2PI * i * 16.0/64);
   }

   check_sts( status = ippsFFTFwd_RToCCS_32f(pSrc, pDst, pSpec, pBuffer) )

   check_sts( status = ippsMagnitude_32fc((Ipp32fc*)pDst, pMagn, 4) )

   printf("FFT Magnitude =  %f  %f  %f  %f\n", pMagn[0], pMagn[1], pMagn[2], pMagn[3]);

EXIT_MAIN
   ippFree(pMemInit);
   ippFree(pSpecMem);   /* звільняємо пам'ять, виділену під специфікацію */
   ippFree(pBuffer);
   printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
   return (int)status;
}
```

FFTInv_PackToR, FFTInv_PermToR, FFTInv_CCSToR

Обчислює обернене швидке перетворення Фур'є (ШПФ) для дійсного сигналу, коли вхідні спектральні дані задані в упакованих форматах ***Pack***, ***Perm*** або ***CCS***.

Синтаксис:

Випадок 1. Робота не на місці, вхідні дані у Pack форматі:

IppStatus ippsFFTInv_PackToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_PackToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція на місці, вхідні дані у Pack форматі:

IppStatus ippsFFTInv_PackToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_PackToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 3. Робота не на місці, вхідні дані у Perm форматі:

IppStatus ippsFFTInv_PermToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_PermToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 4. Операція на місці, вхідні дані у Perm форматі:

IppStatus ippsFFTInv_PermToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_PermToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 5. Робота не на місці, вхідні дані у CCS форматі:

IppStatus ippsFFTInv_CCSToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_CCSToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 6. Операція на місці, вхідні дані у CCS форматі:

IppStatus ippsFFTInv_CCSToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_CCSToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pFFTSpec Вказівник на структуру специфікації ШПФ.
pSrc Вказівник на вхідний масив (упакований спектр у форматі ***Pack / Perm / CCS***).
pDst Вказівник на вихідний масив із дійсними значеннями (результат IFFT).
pSrcDst Вказівник на буфер для операції на місці (вхід—вихід).
pBuffer Вказівник на зовнішній робочий буфер.

Опис:
Ці функції виконують обернене ШПФ (IFFT) дійсного сигналу з упакованого спектра у форматах Pack, Perm або CCS та відновлюють часовий ряд дійсних значень. Обчислення виконуються згідно з параметрами ***pFFTSpec*** (порядок ***order***, нормалізація ***flag***, підказка ***hint***). Перед використанням необхідно ініціалізувати специфікацію за допомогою ***ippsFFTInit_R***. Довжина перетворення має бути степенем двійки:

$$N = 2^{\text{order}}.$$

Для підвищення продуктивності рекомендується використовувати зовнішній буфер ***pBuffer***; його розмір визначається функцією ***ippsFFTGetSize_R***. Якщо ***pBuffer = NULL***, функція виділить пам’ять внутрішньо (це повільніше, особливо для малих розмірів).

ippsFFTInv_PackToR — виконує IFFT, коли вхід у ***Pack*** форматі.
ippsFFTInv_PermToR — виконує IFFT, коли вхід у ***Perm*** форматі.
ippsFFTInv_CCSToR — виконує IFFT, коли вхід у ***CCS*** форматі.

FFTFwd_RToPack, FFTFwd_RToPerm, FFTFwd_RToCCS

Ці функції виконують пряме швидке перетворення Фур'є (ШПФ) дійсного сигналу та зберігають результат у різних форматах представлення комплексних коефіцієнтів — ***Pack***, ***Perm*** або ***CCS***.

Синтаксис

Випадок 1. Робота не на місці, результат у форматі Pack:

IppStatus ippsFFTFwd_RToPack_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToPack_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція на місці, результат у форматі Pack:

IppStatus ippsFFTFwd_RToPack_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToPack_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 3. Робота не на місці, результат у форматі Perm:

IppStatus ippsFFTFwd_RToPerm_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToPerm_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 4. Операція на місці, результат у форматі Perm:

IppStatus ippsFFTFwd_RToPerm_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToPerm_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 5. Робота не на місці, результат у форматі CCS:

IppStatus ippsFFTFwd_RToCCS_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToCCS_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 6. Операція на місці, результат у форматі CCS:

IppStatus ippsFFTFwd_RToCCS_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTFwd_RToCCS_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pFFTSpec вказівник на структуру специфікації ШПФ.
pSrc вказівник на вхідний масив із дійсними значеннями сигналу.
pDst вказівник на вихідний масив, який міститиме упаковані комплексні значення.
pSrcDst вказівник на спільний буфер для операцій на місці (in-place).
pBuffer вказівник на зовнішній робочий буфер.

Опис:
Ці функції виконують пряме ШПФ дійсного сигналу та формують спектр у форматах ***Pack***, ***Perm*** або ***CCS*** залежно від вибраної функції. Обчислення здійснюються згідно з параметрами структури pFFTSpec, яка містить порядок перетворення (***order***), нормалізацію (***flag***) та допоміжний код (***hint***).

Перед використанням необхідно ініціалізувати специфікацію за допомогою ***ippsFFTInit_R***. Довжина перетворення повинна бути степенем двійки:

$$N = 2^{\text{order}}$$

Функції можуть використовувати зовнішній робочий буфер pBuffer, щоб уникнути динамічного виділення пам’яті всередині викликів. Використання зовнішнього буфера помітно покращує продуктивність, особливо для малих розмірів FFT. Його розмір визначається за допомогою функції ***ippsFFTGetSize_R***. Якщо ***pBuffer=NULL***, функція автоматично виділить необхідну пам’ять.

Призначення функцій:
ippsFFTFwd_RToPack — виконує пряме ШПФ і зберігає результат у форматі Pack.
ippsFFTFwd_RToPerm — виконує пряме ШПФ і зберігає результат у форматі Perm.
ippsFFTFwd_RToCCS — виконує пряме ШПФ і зберігає результат у форматі CCS.

Приклад
```
#include <stdio.h>
```

```
#include <math.h>
#include "ipp.h"
#define EXIT_MAIN exitLine:                          /* Мітка для виходу з програми */
#define check_sts(st) if((st) != ippStsNoErr) goto exitLine; /* Перейти до виходу, якщо функція Intel(R) IPP повернула статус, відмінний від ippStsNoErr */

/* Результати виклику ippMalloc() не перевіряються,
   оскільки функції Intel(R) IPP виконують перевірку аргументів
   і повертають відповідний статус у разі помилки. */

int main(void)
{
   int i;
   IppStatus status = ippStsNoErr;
   Ipp32f pSrc[8], pDst[10];       /* Масиви для вихідних і результуючих даних */
   Ipp32f pMagn[4];              /* Масив для обчислених значень амплітуди */
   IppsFFTSpec_R_32f* pSpec = NULL;  /* Вказівник на структуру специфікації ШПФ */

/* Вказівники на робочі буфери */
   Ipp8u *pMemInit = NULL, *pBuffer = NULL, *pSpecMem = NULL;
/* Розміри структури специфікації, ініціалізаційного та робочого буферів */
   int sizeSpec = 0, sizeInit = 0, sizeBuf = 0;

   check_sts( status = ippsFFTGetSize_R_32f(3, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, &sizeSpec, &sizeInit, &sizeBuf) )

   /* Виділення пам’яті */
   pSpecMem = (Ipp8u*) ippMalloc(sizeSpec);
   pBuffer  = (Ipp8u*) ippMalloc(sizeBuf);
   pMemInit = (Ipp8u*) ippMalloc(sizeInit);
   check_sts( status = ippsFFTInit_R_32f(&pSpec, 3, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone, pSpecMem, pMemInit) )

   /* Формування вхідного сигналу — косинусоїди */
   for (i = 0; i < 8; ++i)
   {
      pSrc[i] = (float)cos(IPP_2PI * i * 16.0 / 64);
   }

   /* Виконання прямого ШПФ (FFT) дійсного сигналу */
   check_sts( status = ippsFFTFwd_RToCCS_32f(pSrc, pDst, pSpec, pBuffer) )

   /* Обчислення амплітуд спектра */
   check_sts( status = ippsMagnitude_32fc((Ipp32fc*)pDst, pMagn, 4) )

   /* Вивід результатів на екран */
   printf("FFT Magnitude =  %f  %f  %f  %f\n", pMagn[0], pMagn[1], pMagn[2], pMagn[3]);

EXIT_MAIN
```

```
    /* Звільнення виділеної пам’яті */
    ippFree(pMemInit);
    ippFree(pSpec);
    ippFree(pBuffer);

    /* Вивід статусу завершення програми */
    printf("Exit status %d (%s)\n", (int)status, ippGetStatusString(status));
    return (int)status;
}
```

FTInv_PackToR, FFTInv_PermToR, FFTInv_CCSToR

Ці функції виконують обернене швидке перетворення Фур’є (ШПФ) дійсного сигналу, використовуючи вхідні дані у форматах ***Pack***, ***Perm*** або ***CCS***, та повертають результат у вигляді дійсного сигналу.

Синтаксис:

Випадок 1. Операція не на місці, вхідні дані у форматі Pack:

IppStatus ippsFFTInv_PackToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_PackToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція на місці, вхідні дані у форматі Pack:

IppStatus ippsFFTInv_PackToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_PackToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 3. Операція не на місці, вхідні дані у форматі Perm:

IppStatus ippsFFTInv_PermToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_PermToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 4. Операція на місці, вхідні дані у форматі Perm:

IppStatus ippsFFTInv_PermToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);
IppStatus ippsFFTInv_PermToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 5. Операція не на місці, вхідні дані у форматі CCS:

IppStatus ippsFFTInv_CCSToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTInv_CCSToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Випадок 6. Операція на місці, вхідні дані у форматі CCS:

IppStatus ippsFFTInv_CCSToR_32f_I(Ipp32f* pSrcDst, const IppsFFTSpec_R_32f* pFFTSpec, Ipp8u* pBuffer);

IppStatus ippsFFTInv_CCSToR_64f_I(Ipp64f* pSrcDst, const IppsFFTSpec_R_64f* pFFTSpec, Ipp8u* pBuffer);

Включити файли:***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pFFTSpec вказівник на структуру специфікації ШПФ.
pSrc вказівник на вхідний масив із даними у форматі ***Pack***, ***Perm*** або ***CCS***.
pDst вказівник на вихідний масив, який міститиме відновлений дійсний сигнал.
pSrcDst вказівник на спільний буфер для операцій на місці (in-place).
pBuffer вказівник на зовнішній робочий буфер.

Опис:
Ці функції виконують обернене перетворення Фур'є для сигналів, представлених у форматах ***Pack***, ***Perm*** або ***CCS***, і відновлюють початковий дійсний сигнал. Обчислення здійснюється згідно з параметрами, заданими у структурі ***pFFTSpec***, яка визначає порядок перетворення (***order***), нормалізацію (***flag***) та допоміжний код (***hint***).

Перед використанням потрібно ініціалізувати структуру специфікації за допомогою функції ***ippsFFTInit_R***.
Довжина перетворення повинна бути степенем двійки:

$$N = 2^{\text{order}}$$

Для уникнення розподілу пам'яті всередині функції можна використовувати зовнішній робочий буфер ***pBuffer***. Його розмір слід визначити за допомогою функції ***ippsFFTGetSize_R***. Зовнішній буфер може використовуватися повторно для кількох викликів, що підвищує продуктивність, особливо при малих розмірах FFT. Якщо ***pBuffer=NULL***, пам'ять буде виділена автоматично.

Призначення функцій:

ippsFFTInv_PackToR обчислює обернене ШПФ для вхідних даних у форматі ***Pack***.

ippsFFTInv_PermToR обчислює обернене ШПФ для вхідних даних у форматі ***Perm***.

ippsFFTInv_CCSToR обчислює обернене ШПФ для вхідних даних у форматі ***CCS***.

5.5. Функції дискретного перетворення Фур'є

Функції, описані в цьому розділі, обчислюють пряме і обернене дискретні перетворення Фур'є дійсних і комплексних сигналів. DFT менш ефективний, ніж швидке перетворення Фур'є, проте довжина вектора, перетвореного DFT, може бути довільною.

hint аргумент, переданий функціям ініціалізації, пропонує використовувати спеціальний алгоритм, швидший або точніший. ***flag*** аргумент задає метод нормалізації результату. Комплексний сигнал може бути представлений у вигляді єдиного масиву, що містить комплексні елементи, або двох окремих масивів, що містять дійсну та уявну частини. Результат ШПФ можна запакувати ***Pack*** , ***Perm*** , або ***CCS*** формати.

Щоб використовувати функції DFT, вам слід ініціалізувати структуру специфікації, яка містить такі дані, як таблиці коефіцієнтів двійника. Використовувати ***ippsDFTInit_R*** і ***ippsDFTInit_C*** функції для ініціалізації структури специфікації як для прямого, так і для оберненого перетворення. Перед використанням цих функцій обчисліть розмір структури специфікації DFT, використовуючи ***ippsDFTGetSize_R*** або ***ippsDFTGetSize_C*** функції і заздалегідь виділити пам'ять для структури.

Ви можете прискорити DFT, використовуючи зовнішній буфер. Використання зовнішнього буфера може покращити продуктивність, уникаючи розподілу та вивільнення внутрішніх буферів та зберігання даних у кеші. Розмір зовнішнього буфера повертається ***ippsDFTInit_R*** і ***ippsDFTInit_C*** функції.

Для отримання додаткової інформації про швидкі обчислення дискретного перетворення Фур'є див. [5], розділ 8-2, Швидке обчислення DFT.

Спеціальний набір функцій Intel IPP забезпечує так званий "непрацюючий" DFT комплексного сигналу. У цьому випадку елементи в частотній області як для прямого, так і для оберненого перетворення можуть бути впорядковані, щоб пришвидшити обчислення перетворень. Це повторне впорядкування приховано від користувача і може відрізнятися в різних

реалізаціях функцій. Однак забезпечується оборотність кожної пари функцій для прямого / оберненого перетворень.

Опис:

Ці функції обчислюють розміри, необхідні для роботи з ДПФ (дискретним перетворенням Фур'є):

розмір структури специфікації DFT, розмір робочого буфера, потрібного для ініціалізації структури (***ippsDFTInit_R***, ***ippsDFTInit_C***), а також розмір буфера для безпосереднього виконання перетворень у функціях ***ippsDFTFwd*** та ***ippsDFTInv***.

Отримані розміри (у байтах) зберігаються у змінних ***pSpecSize***, ***pSizeInit*** і ***pSizeBuf*** відповідно.

Функція ***ippsDFTGetSize_R*** використовується для дійсних варіантів DFT-функцій,

а ***ippsDFTGetSize_C*** — для комплексних варіантів.

DFTFwd_CToC

Обчислює пряме дискретне перетворення Фур'є (DFT) комплексного сигналу.

Синтаксис:

Випадок 1. Операція з дійсним типом даних:

IppStatus ippsDFTFwd_CToC_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDstRe, Ipp32f* pDstIm, const IppsDFTSpec_C_32f* pDFTSpec);

IppStatus ippsDFTFwd_CToC_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDstRe, Ipp64f* pDstIm, const IppsDFTSpec_C_64f* pDFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція з комплексним типом даних:

IppStatus ippsDFTFwd_CToC_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, const IppsDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);

IppStatus ippsDFTFwd_CToC_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, const IppsDFTSpec_C_64fc* pDFTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pDFTSpec вказівник на структуру специфікації DFT.

pSrc вказівник на вхідний масив, що містить комплексні значення.

pDst вказівник на вихідний масив, що містить комплексні значення.

pSrcRe вказівник на вхідний масив, що містить дійсні частини сигналу.

pSrcIm вказівник на вхідний масив, що містить уявні частини сигналу.

pDstRe вказівник на вихідний масив, що містить дійсні частини сигналу.

pDstIm вказівник на вихідний масив, що містить уявні частини сигналу.

pBuffer вказівник на зовнішній робочий буфер.

Опис:

Ці функції обчислюють пряме DFT згідно з параметрами, визначеними у структурі ***pDFTSpec***: довжина перетворення ***len***, нормалізація ***flag***, та оптимізаційний код ***hint***.

Функції, які працюють із комплексним типом даних, приймають вхідний масив ***pSrc*** і зберігають результат у ***pDst***.

Функції, що обробляють сигнали, представлені окремими дійсними (***pSrcRe***) та уявними (***pSrcIm***) частинами, зберігають результати відповідно у ***pDstRe*** та ***pDstIm***.

Функції можуть використовувати зовнішній робочий буфер ***pBuffer***, щоб уникнути внутрішнього виділення пам’яті.

Розмір буфера потрібно обчислити за допомогою ***ippsDFTGetBufSize_C*** перед викликом функцій DFT.

Якщо ***pBuffer = NULL***, пам’ять виділяється автоматично.

Примітка:

Вектори даних для цих функцій повинні бути вирівняні по кордону SIMD, який підтримується апаратною платформою.

Для забезпечення правильного вирівнювання рекомендується використовувати функцію ippMalloc(), яка гарантує коректне розташування даних у пам’яті.

Математичний опис:

Пряме дискретне перетворення Фур’є можна подати у вигляді:

$$X(k) = A \sum_{n=0}^{N-1} x(n)\, e^{-j\frac{2\pi}{N}kn}$$

де:

$X(k)$ — спектр сигналу (частотна область);
$x(n)$ — вхідний сигнал (часова область);
N — довжина DFT (len);
k — індекс елемента у частотній області;
n — індекс елемента у часовій області;
A — коефіцієнт нормалізації, що визначається параметром ***flag***.

Додаткова інформація:

Функції оптимізовані під мікроархітектури Intel® з підтримкою SIMD-інструкцій (SSE2, AVX, AVX-512 тощо).

Їхня ефективність на процесорах інших виробників може відрізнятися, оскільки оптимізації орієнтовані на архітектуру Intel®.

DFTInv_CtoC

Обчислює обернене дискретне перетворення Фур'є (DFT) комплексного сигналу.

Синтаксис:

Випадок 1. Операція з дійсним типом даних:

IppStatus ippsDFTInv_CToC_32f(const Ipp32f* pSrcRe, const Ipp32f* pSrcIm, Ipp32f* pDstRe, Ipp32f* pDstIm, const IppsDFTSpec_C_32f* pDFTSpec);

IppStatus ippsDFTInv_CToC_64f(const Ipp64f* pSrcRe, const Ipp64f* pSrcIm, Ipp64f* pDstRe, Ipp64f* pDstIm, const IppsDFTSpec_C_64f* pDFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція з комплексним типом даних:

IppStatus ippsDFTInv_CToC_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, const IppsDFTSpec_C_32fc* pDFTSpec, Ipp8u* pBuffer);

IppStatus ippsDFTInv_CToC_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, const IppsDFTSpec_C_64fc* pDFTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pDFTSpec вказівник на структуру специфікації DFT.

pSrc вказівник на вхідний масив, що містить комплексні значення.

pDst вказівник на вихідний масив, що містить комплексні значення.

pSrcRe вказівник на вхідний масив, що містить дійсні частини сигналу.

pSrcIm вказівник на вхідний масив, що містить уявні частини сигналу.

pDstRe вказівник на вихідний масив, що містить дійсні частини сигналу.

pDstIm вказівник на вихідний масив, що містить уявні частини сигналу.

pBuffer вказівник на робочий буфер.

Опис:

Ці функції виконують обернене дискретне перетворення Фур'є (IDFT) згідно з параметрами, визначеними у структурі ***pDFTSpec***: довжина перетворення ***len***, нормалізація ***flag*** та допоміжний код ***hint***.

Функції, що працюють із комплексним типом даних (наприклад, із суфіксом ***32fc***), приймають вхідний комплексний масив ***pSrc*** і зберігають результат у ***pDst***.

Функції, які працюють із дійсними типами даних, обробляють сигнали, представлені окремими дійсними (***pSrcRe***) та уявними (***pSrcIm***) частинами, і записують результат у ***pDstRe*** та ***pDstIm*** відповідно.

Для уникнення внутрішнього виділення пам'яті функції можна викликати із зовнішнім робочим буфером ***pBuffer***. Після виділення такого буфера його можна використовувати повторно для подальших викликів DFT-функцій.

Розмір робочого буфера необхідно попередньо обчислити за допомогою функції ***ippsDFTGetBufSize_C***.

Якщо ***pBuffer=NULL***, функція самостійно виділить необхідну пам'ять.

Примітка:

Для забезпечення максимальної продуктивності вектори даних мають бути вирівняні по межі SIMD, підтримуваній апаратною платформою. Для цього рекомендується використовувати ***ippMalloc()***, що гарантує правильне вирівнювання пам'яті.

Математичний опис:

Обернене дискретне перетворення Фур'є описується рівнянням:

$$x(n) = A \sum_{k=0}^{N-1} X(k)\, e^{j\frac{2\pi}{N}kn}$$

x(n) — відновлений сигнал у часовій області;
X(k) — спектр сигналу (частотна область);
N — довжина перетворення;
k — індекс елементів у частотній області;
n — індекс елементів у часовій області;
A — коефіцієнт нормалізації, що визначається параметром flag.

DFTFwd_RToPack, DFTFwd_RToPerm, DFTFwd_RToCCS

Виконує пряме дискретне перетворення Фур'є (DFT) дійсного сигналу та зберігає результат ***y*** різних форматах представлення комплексних коефіцієнтів — ***Pack***, ***Perm*** або ***CCS***.

Синтаксис:

Випадок 1. Робота не на місці, результат у форматі Pack:

IppStatus ippsDFTFwd_RToPack_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippsDFTFwd_RToPack_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);

Випадок 2. Робота не на місці, результат у форматі Perm:

IppStatus ippsDFTFwd_RToPerm_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippsDFTFwd_RToPerm_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);

Випадок 3. Робота не на місці, результат у форматі CCS:

IppStatus ippsDFTFwd_RToCCS_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);
IppStatus ippsDFTFwd_RToCCS_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);

Включити файли:***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pDFTSpec вказівник на структуру специфікації DFT.
pSrc вказівник на вхідний масив, що містить дійсні дані сигналу.
pDst вказівник на вихідний масив, що містить упаковані комплексні значення результату DFT.
pBuffer вказівник на зовнішній робочий буфер.

Опис:
Функції ***ippsDFTFwd_RToPack***, ***ippsDFTFwd_RToPerm*** і ***ippsDFTFwd_RToCCS*** виконують пряме дискретне перетворення Фур'є (DFT) дійсного сигналу та записують результат у відповідний формат представлення комплексних коефіцієнтів.

Обчислення здійснюється згідно з параметрами, вказаними у структурі ***pDFTSpec***, яка задається під час ініціалізації функцією ***ippsDFTInit_R***. Параметри визначають довжину перетворення (***len***), нормалізацію (***flag***) і алгоритмічні підказки (***hint***).

Використання зовнішнього буфера ***pBuffer*** дозволяє уникнути динамічного виділення пам'яті під час виконання функцій, що значно підвищує швидкодію, особливо для невеликих розмірів DFT. Розмір буфера визначається за допомогою функції ***ippsDFTGetSize_R***. Якщо ***pBuffer*** дорівнює ***NULL***, функція самостійно виділить пам'ять, необхідну для роботи.

Призначення функцій:
ippsDFTFwd_RToPack — обчислює пряме DFT і зберігає результат у форматі Pack.
ippsDFTFwd_RToPerm — обчислює пряме DFT і зберігає результат у форматі Perm.

ippsDFTFwd_RToCCS — обчислює пряме DFT і зберігає результат у форматі CCS.

DFTInv_PackToR, DFTInv_PermToR, DFTInv_CCSToR

Виконує обернене дискретне перетворення Фур'є (IDFT) дійсного сигналу, зберігаючи результат у звичайному (real) форматі.

Синтаксис:

Випадок 1. Операція не на місці, вхідні дані у форматі Pack:

IppStatus ippsDFTInv_PackToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

IppStatus ippsDFTInv_PackToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);

Випадок 2. Операція не на місці, вхідні дані у форматі Perm:

IppStatus ippsDFTInv_PermToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

IppStatus ippsDFTInv_PermToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);

Випадок 3. Операція не на місці, вхідні дані у форматі CCS:

IppStatus ippsDFTInv_CCSToR_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDFTSpec_R_32f* pDFTSpec, Ipp8u* pBuffer);

IppStatus ippsDFTInv_CCSToR_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDFTSpec_R_64f* pDFTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pDFTSpec вказівник на структуру специфікації DFT.

pSrc вказівник на вхідний масив, що містить дані у форматі ***Pack***, ***Perm*** або ***CCS***.

pDst вказівник на вихідний масив, що містить відновлений дійсний сигнал.

pBuffer вказівник на зовнішній робочий буфер.

Опис:

Ці функції виконують зворотне дискретне перетворення Фур'є (IDFT) дійсного сигналу, перетворюючи дані з упакованих форматів (***Pack***, ***Perm***, ***CCS***) у дійсний часовий ряд.

Обчислення виконується відповідно до параметрів, визначених у структурі pDFTSpec: довжина перетворення (***len***), нормалізація (***flag***) та алгоритмічні підказки (***hint***).

Перед використанням функцій необхідно ініціалізувати структуру специфікації DFT за допомогою ***ippsDFTInit_R***.

Використання зовнішнього буфера ***pBuffer*** дозволяє уникнути динамічного виділення пам'яті під час виконання функцій, що підвищує продуктивність, особливо для малих розмірів DFT.

Розмір буфера має бути попередньо обчислений функцією ***ippsDFTGetSize_R***. Якщо ***pBuffer*** дорівнює ***NULL***, функція автоматично виділяє необхідну пам'ять.

Призначення функцій:

ippsDFTInv_PackToR — обчислює зворотне DFT для даних у форматі Pack.

ippsDFTInv_PermToR — обчислює зворотне DFT для даних у форматі Perm.

ippsDFTInv_CCSToR — обчислює зворотне DFT для даних у форматі CCS.

5.6. DFT для функцій заданої частоти (Герцеля)

Функції, описані в цьому розділі, виконують дискретне перетворення Фур'є (DFT) для заданої частоти.Слід зазначити, що DFT визначається лише для нормалізованих частот виду :

$$0, \frac{1}{N}, \frac{2}{N}, \ldots, \frac{N-1}{N},$$

де N — кількість відліків у часовій області.

Тому під час обчислень потрібно вибирати частоту саме з цього набору допустимих значень.

Функції Intel IPP реалізують алгоритм Герцеля [6], який є ефективним тоді, коли потрібно обчислити лише кілька окремих значень DFT, а не повне перетворення.

Деякі функції розраховані на обчислення двох значень DFT одночасно, що робить їх особливо корисними в задачах, де потрібно проаналізувати кілька частот — наприклад, при виявленні двотональних багаточастотних (DTMF) сигналів, де такі функції забезпечують суттєве підвищення швидкодії.

oertz

Обчислює дискретне перетворення Фур'є (DFT) для сигналу на заданій частоті за допомогою алгоритму Герцеля.

Синтаксис:

IppStatus ippsGoertz_32f(const Ipp32f* pSrc, int len, Ipp32fc* pVal, Ipp32f rFreq);

IppStatus ippsGoertz_64f(const Ipp64f* pSrc, int len, Ipp64fc* pVal, Ipp64f rFreq);

IppStatus ippsGoertz_32fc(const Ipp32fc* pSrc, int len, Ipp32fc* pVal, Ipp32f rFreq);

IppStatus ippsGoertz_64fc(const Ipp64fc* pSrc, int len, Ipp64fc* pVal, Ipp64f rFreq);

IppStatus ippsGoertz_16s_Sfs(const Ipp16s* pSrc, int len, Ipp16sc* pVal, Ipp32f rFreq, int scaleFactor);

IppStatus ippsGoertz_16sc_Sfs(const Ipp16sc* pSrc, int len, Ipp16sc* pVal, Ipp32f rFreq, int scaleFactor);

Включити файли:***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc вказівник на вхідний вектор даних.

len кількість елементів у вхідному векторі.

pVal вказівник на результат — комплексне значення DFT для заданої частоти.

rFreq нормалізована частота у діапазоні [0, 1.0).

scaleFactor масштабний коефіцієнт (див. Цілісне масштабування).

Опис:

Функція ***ippsGoertz*** обчислює одне значення DFT для сигналу довжини ***len***, переданого у векторі ***pSrc***, на заданій нормалізованій частоті ***rFreq***.

Алгоритм базується на методі Герцеля, який є ефективною альтернативою класичному FFT, коли необхідно обчислити DFT лише для кількох частот, а не для всього спектра.

Функції ***ippsGoertzQ15*** працюють із частотою, представленою у фіксованому форматі ***Q15***, тобто значення частоти з діапазону [0, 1.0) кодується у 16-бітному представленні (де 1.0 ≈ 32768).

Алгоритм Герцеля описується такими рівняннями:

$$s[n] = x[n] + 2\cos(2\pi k/N)s[n-1] - s[n-2]$$

$$P(k) = s[N-1]^2 + s[N-2]^2 - 2\cos(2\pi k/N)s[N-1]s[N-2]$$

де $k/N = rFreq$ — нормалізоване значення частоти, для якої обчислюється DFT.

Приклад:

Наведений приклад демонструє використання функції ***ippsGoertz_32fc*** для обчислення спектральної складової сигналу частоти 60 Гц, дискретизованого з частотою 400 Гц:

```
IppStatus goertzel(void) {
#define LEN 100
   IppStatus status;
   Ipp32fc* x = ippsMalloc_32fc(LEN), y;
   int n;

   /* Генеруємо сигнал із частотою 60 Гц, дискретизований на 400 Гц */
   for (n = 0; n < LEN; ++n) {
      x[n].re = (Ipp32f)sin(IPP_2PI * n * 60 / 400);
      x[n].im = 0;
   }

   status = ippsGoertz_32fc(x, LEN, &y, 60.0f / 400);
   printf_32fc("Goertz =", &y, 1, status);
   ippsFree(x);
   return status;
}

Result:
Goertz = {0.000090, -50.000008}
```

5.7. Функції дискретного косинусного перетворення

У цьому розділі наведені функції, які обчислюють дискретне косинусне перетворення (DCT) сигналу. Функції DCT, що використовуються в області обробки сигналів Intel IPP, реалізують модифікований алгоритм обчислень, запропонований у [2].

DCTFwdInit

Ініціалізує структуру для прямого дискретного косинусного перетворення (DCT).

Синтаксис:

IppStatus ippsDCTFwdInit_32f(IppsDCTFwdSpec_32f** ppDCTSpec, int len, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);

IppStatus ippsDCTFwdInit_64f(IppsDCTFwdSpec_64f** ppDCTSpec, int len, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);

Включити файли: **ipps.h**

Доменні залежності:
Заголовки: **ippcore.h, ippvm.h**
Бібліотеки: **ippcore.lib, ippvm.lib**

Параметри:

ppDCTSpec подвійний вказівник на створювану структуру специфікації DCT.

len кількість відліків (довжина сигналу) для виконання DCT.

Hint алгоритмічна підказка; параметр застарілий, потрібно встановлювати значення ippAlgHintNone.

pSpec вказівник на область пам'яті для структури специфікації DCT.

pSpecBuffer вказівник на додатковий робочий буфер; може бути NULL, якщо буфер не потрібен.

Опис:

Функція ***ippsDCTFwdInit*** ініціалізує структуру специфікації DCT, на яку посилається ***ppDCTSpec***, задаючи такі параметри, як довжина перетворення (len) та тип алгоритму (hint).

Перед викликом цієї функції необхідно виділити пам'ять для структури специфікації DCT і, за потреби, для робочого буфера. Розмір обох областей пам'яті повинен бути заздалегідь обчислений за допомогою функції ***ippsDCTFwdGetSize***.

Якщо робочий буфер не використовується, параметр ***pSpecBuffer*** може бути ***NULL***. Якщо ж буфер необхідний для виконання ініціалізації, значення ***pSpecBuffer*** має вказувати на дійсну область пам'яті.

Ця функція є початковим етапом підготовки до виконання прямого DCT і повинна бути викликана перед будь-якою з функцій ***ippsDCTFwd***.

DCTInvInit

Ініціалізує структуру для оберненого дискретного косинусного перетворення (Inverse Discrete Cosine Transform, IDCT).

Синтаксис:

IppStatus ippsDCTInvInit_32f(IppsDCTInvSpec_32f** ppDCTSpec, int len, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);

IppStatus ippsDCTInvInit_64f(IppsDCTInvSpec_64f** ppDCTSpec, int len, IppHintAlgorithm hint, Ipp8u* pSpec, Ipp8u* pSpecBuffer);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

ppDCTSpec подвійний вказівник на створювану структуру специфікації оберненого DCT.

len кількість відліків (довжина сигналу), для яких виконується обернене DCT.

Hint алгоритмічна підказка; параметр застарілий, необхідно встановлювати значення ippAlgHintNone.

pSpec вказівник на область пам'яті, у якій зберігатиметься структура специфікації DCT.

pSpecBuffer вказівник на робочий буфер; може бути NULL, якщо буфер не використовується.

Опис:

Функція ***ippsDCTInvInit*** ініціалізує у буфері ***pSpec*** структуру специфікації оберненого DCT, на яку посилається ***ppDCTSpec***, використовуючи параметри ***len*** (довжина перетворення) та ***hint*** (тип алгоритму).

Перед викликом цієї функції необхідно виділити пам’ять для структури специфікації DCT та, за потреби, для робочого буфера. Розмір цих областей пам’яті має бути попередньо обчислений функцією ***ippsDCTInvGetSize***, яка визначає необхідний обсяг для ініціалізації та виконання оберненого DCT.

Якщо робочий буфер не використовується, параметр ***pSpecBuffer*** може бути ***NULL***. Якщо робочий буфер потрібен, ***pSpecBuffer*** повинен вказувати на дійсну область пам’яті.

Ця функція є підготовчим етапом перед виконанням функцій ***ippsDCTInv***, які безпосередньо обчислюють обернене дискретне косинусне перетворення.

DCTFwdGetSize

Обчислює розміри всіх буферів, необхідних для прямого дискретного косинусного перетворення (DCT).

Синтаксис:

IppStatus ippsDCTFwdGetSize_32f(int len, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);

IppStatus ippsDCTFwdGetSize_64f(int len, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);

Включити файли:***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

len кількість відліків (довжина сигналу), для яких виконується DCT.

Hint алгоритмічна підказка; параметр застарілий, потрібно встановлювати значення ippAlgHintNone.

pSpecSize вказівник на змінну, у яку буде записано розмір структури специфікації прямого DCT.

pSpecBufferSize вказівник на змінну, у яку буде записано розмір робочого буфера, необхідного для функції ініціалізації ***ippsDCTFwdInit***.

pBufferSize вказівник на змінну, у яку буде записано розмір робочого буфера, необхідного для виконання функції ***ippsDCTFwd***.

Опис:

Функція ***ippsDCTFwdGetSize*** обчислює обсяги пам’яті, потрібні для створення та виконання операцій прямого DCT. Вона визначає:

— розмір структури специфікації DCT (***pSpecSize***),

— розмір буфера ініціалізації (***pSpecBufferSize***), який використовується під час виклику ***ippsDCTFwdInit***,

— розмір робочого буфера (***pBufferSize***) для виконання самого перетворення за допомогою ***ippsDCTFwd***.

Виклик цієї функції є обов’язковим перед ініціалізацією DCT та його виконанням. Результати, отримані у вказаних змінних, дозволяють правильно виділити пам’ять для подальших функцій ***ippsDCTFwdInit*** та ***ippsDCTFwd***, забезпечуючи ефективне керування ресурсами та запобігаючи внутрішнім алокаціям пам’яті під час обчислень.

DCTInvGetSize

Обчислює розміри всіх буферів, необхідних для оберненого дискретного косинусного перетворення (Inverse DCT).

Синтаксис:

IppStatus ippsDCTInvGetSize_32f(int len, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);

IppStatus ippsDCTInvGetSize_64f(int len, IppHintAlgorithm hint, int* pSpecSize, int* pSpecBufferSize, int* pBufferSize);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

len кількість відліків (довжина сигналу), для яких виконується обернене DCT.

Hint алгоритмічна підказка; параметр застарілий, потрібно встановлювати значення ippAlgHintNone.

pSpecSize вказівник на змінну, у яку буде записано розмір структури специфікації оберненого DCT.

pSpecBufferSize вказівник на змінну, у яку буде записано розмір робочого буфера, необхідного для функції ініціалізації ***ippsDCTInvInit***.

pBufferSize вказівник на змінну, у яку буде записано розмір робочого буфера, необхідного для виконання функції ***ippsDCTInv***.

Опис:

Функція ***ippsDCTInvGetSize*** використовується для попереднього визначення кількості пам’яті, потрібної для виконання операцій оберненого DCT. Вона обчислює:

— розмір структури специфікації (***pSpecSize***), у якій зберігаються параметри DCT;

— розмір робочого буфера ініціалізації (***pSpecBufferSize***), який використовується функцією ***ippsDCTInvInit***;

— розмір робочого буфера виконання (***pBufferSize***), який застосовується під час виклику ippsDCTInv.

Виклик цієї функції є необхідним до створення структури специфікації DCT та виконання самого перетворення. Результати, отримані у вказаних змінних, дозволяють правильно виділити пам’ять і забезпечити ефективну роботу наступних функцій ***ippsDCTInvInit*** та ***ippsDCTInv***, мінімізуючи накладні витрати на внутрішні алокації пам’яті.

DCTFwd

Обчислює пряме дискретне косинусне перетворення (DCT) сигналу.

Синтаксис:

Випадок 1: операція не на місці

IppStatus ippsDCTFwd_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);

IppStatus ippsDCTFwd_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDCTFwdSpec_64f* pDCTSpec, Ipp8u* pBuffer);

Випадок 2: операція на місці

IppStatus ippsDCTFwd_32f_I(Ipp32f* pSrcDst, const IppsDCTFwdSpec_32f* pDCTSpec, Ipp8u* pBuffer);

IppStatus ippsDCTFwd_64f_I(Ipp64f* pSrcDst, const IppsDCTFwdSpec_64f* pDCTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pDCTSpec	вказівник на структуру специфікації прямого DCT.
pSrc	вказівник на вхідний вектор.
pDst	вказівник на вихідний вектор.
pSrcDst	вказівник на вектор для операцій на місці.
pBuffer	вказівник на зовнішній робочий буфер.

Опис:

Функція ***ippsDCTFwd*** обчислює пряме дискретне косинусне перетворення (DCT) для вхідного сигналу ***pSrc*** (або ***pSrcDst*** для операцій на місці) відповідно до специфікації, заданої у структурі ***pDCTSpec***.

Перед викликом цієї функції структуру специфікації необхідно ініціалізувати за допомогою функції ***ippsDCTFwdInit***.

Результат перетворення записується у вихідний вектор ***pDst*** (або ***pSrcDst*** для операцій на місці).

Якщо довжина сигналу ***len*** є степенем двійки, функція використовує ефективний швидкий алгоритм, який значно продуктивніший за пряме обчислення DCT.

Для інших значень N використовується безпосереднє обчислення за визначенням, але з урахуванням симетрії функції косинуса, що зменшує кількість операцій множення приблизно вдвічі.

Пряме дискретне косинусне перетворення визначається за формулою:

$$y(k) = \sum_{n=0}^{N-1} x(n) \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right], \quad k = 0, 1, \ldots, N-1$$

де x(n) — вхідні дані pSrc[n], а y(k) — результат перетворення pDst[k].

Функцію можна використовувати з зовнішнім робочим буфером ***pBuffer***, щоб уникнути внутрішнього розподілу пам’яті. Після виділення робочого буфера його можна використовувати для наступних викликів функцій DCT.

Оскільки внутрішнє виділення пам’яті є дорогою операцією та залежить від системних механізмів, використання зовнішнього буфера значно підвищує продуктивність, особливо для малих розмірів перетворення.

Розмір буфера слід попередньо обчислити викликом ***ippsDCTFwdGetSize***.

Якщо зовнішній буфер не вказано (***pBuffer = NULL***), функція автоматично виділить необхідну пам’ять для своєї роботи.

DCTInv

Обчислює обернене дискретне косинусне перетворення (Inverse DCT) сигналу.

Синтаксис:
Випадок 1: операція не на місці

IppStatus ippsDCTInv_32f(const Ipp32f* pSrc, Ipp32f* pDst, const IppsDCTInvSpec_32f* pDCTSpec, Ipp8u* pBuffer);
IppStatus ippsDCTInv_64f(const Ipp64f* pSrc, Ipp64f* pDst, const IppsDCTInvSpec_64f* pDCTSpec, Ipp8u* pBuffer);
Випадок 2: операція на місці
IppStatus ippsDCTInv_32f_I(Ipp32f* pSrcDst, const IppsDCTInvSpec_32f* pDCTSpec, Ipp8u* pBuffer);
IppStatus ippsDCTInv_64f_I(Ipp64f* pSrcDst, const IppsDCTInvSpec_64f* pDCTSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pDCTSpec	вказівник на структуру специфікації оберненого DCT.
pSrc	вказівник на вхідний вектор.
pDst	вказівник на вихідний вектор.
pSrcDst	вказівник на вектор для операцій на місці.
pBuffer	вказівник на зовнішній робочий буфер.

Опис:

Функція ***ippsDCTInv*** обчислює обернене дискретне косинусне перетворення (DCT) сигналу ***pSrc*** (або ***pSrcDst*** для операцій на місці) відповідно до параметрів, визначених у структурі специфікації ***pDCTSpec***, яку необхідно попередньо ініціалізувати викликом ***ippsDCTInvInit***. Результат записується у ***pDst*** (або ***pSrcDst*** при роботі на місці).

Якщо довжина сигналу ***len*** є степенем двійки, функція використовує ефективний швидкий алгоритм, який значно продуктивніший за пряме обчислення DCT. Для інших значень ***len*** використовується пряме обчислення за визначенням, але із врахуванням симетрії функції косинуса, що дозволяє зменшити кількість операцій множення приблизно удвічі.

В оберненому DCT (де ***N = len***) вхідний вектор ***pSrc[k]*** відповідає коефіцієнтам DCT, а вихідний сигнал pDst[n] відновлюється за формулою:

$$x(n) = \frac{1}{N}\left[\frac{1}{2}y(0) + \sum_{k=1}^{N-1} y(k)\cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right)\right], \quad n = 0, 1, \ldots, N-1$$

де y(k) — коефіцієнти DCT, а x(n) — відновлений часовий сигнал.

Функцію можна викликати із зовнішнім робочим буфером ***pBuffer***, щоб уникнути внутрішнього розподілу пам’яті. Після виділення буфера його можна повторно використовувати для наступних викликів функцій обчислення DCT. Використання зовнішнього буфера підвищує продуктивність, особливо для малих розмірів перетворення, оскільки внутрішнє виділення пам’яті є витратною операцією.

Розмір буфера слід попередньо обчислити за допомогою ***ippsDCTInvGetSize***. Якщо зовнішній буфер не вказано (***pBuffer = NULL***), функція автоматично виділить необхідну пам’ять для роботи.

5.8. Функції перетворення Гільберта

Функції, описані в цьому пункті, призначені для обчислення аналітичного сигналу дискретного часу з дійсної послідовності даних за допомогою перетворення Гільберта.

Аналітичний сигнал — це комплексний сигнал, дійсна частина якого є точною копією вхідних даних, а уявна частина містить результат перетворення Гільберта. Іншими словами, уявна частина є версією вихідного сигналу зі зсувом фази на 90°.

Таке представлення дозволяє отримати інформацію не лише про амплітуду й частоту сигналу, а й про його фазу, що робить аналітичну форму зручною для аналізу огинаючих, миттєвої частоти та фазових характеристик.

Перед виконанням перетворення Гільберта необхідно визначити розміри робочого буфера та структури специфікації за допомогою функції ***ippsHilbertGetSize***, після чого ініціалізувати структуру викликом ippsHilbertInit.

HilbertGetSize

Обчислює розміри всіх необхідних буферів і структури специфікації для виконання **перетворення Гільберта**.

Синтаксис:

IppStatus ippsHilbertGetSize_32f32fc(int length, IppHintAlgorithm hint, int* pSpecSize, int* pBufferSize);

Включити файли: ***ipps.h***

Доменні залежності
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

length кількість відліків у перетворенні Гільберта.

hint підказка щодо вибору алгоритмічної реалізації функції (DFT). Зазвичай встановлюється значення ***ippAlgHintNone***.

pSpecSize вказівник на змінну, у яку буде записано розмір (у байтах) структури специфікації Гільберта.

pBufferSize вказівник на змінну, у яку буде записано розмір (у байтах) робочого буфера.

Опис:

Функція ***ippsHilbertGetSize*** обчислює обсяг пам’яті, необхідний для створення структури специфікації перетворення Гільберта та тимчасового робочого буфера, який використовується функцією ***ippsHilbert***.

Розраховані значення ***pSpecSize*** і ***pBufferSize*** застосовуються для виділення пам’яті перед ініціалізацією та подальшим виконанням перетворення.

Використання цієї функції передбачене для забезпечення оптимального розподілу пам’яті, що дозволяє уникнути внутрішніх алокацій у функції ippsHilbert і підвищує загальну продуктивність обчислень.

HilbertInit

Ініціалізує структуру специфікації для перетворення Гільберта.

Синтаксис:

IppStatus ippsHilbertInit_32f32fc(int length, IppHintAlgorithm hint, IppsHilbertSpec* pSpec, Ipp8u* pBuffer);

Включити файли: ***ipps.h***

Доменні залежності

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

length кількість відліків у перетворенні Гільберта.

hint підказка щодо вибору алгоритмічної реалізації перетворення (наприклад, на основі DFT). Зазвичай встановлюється значення ***ippAlgHintNone***.

pSpec вказівник на структуру контексту Гільберта, яка зберігає попередньо обчислені параметри для подальшого використання.

pBuffer вказівник на робочий буфер, необхідний для ініціалізації структури.

Опис:

Функція ***ippsHilbertInit*** створює і ініціалізує структуру специфікації перетворення Гільберта ***pSpec*** для подальшого використання функцією ***ippsHilbert***.

Під час ініціалізації задається довжина перетворення (***length***) і тип алгоритму (***hint***).

Перед викликом ***ippsHilbertInit*** потрібно заздалегідь визначити розміри структури ***pSpec*** і робочого буфера ***pBuffer*** за допомогою функції ***ippsHilbertGetSize***. Отримані значення використовуються для виділення необхідної пам’яті.

Функцію слід викликати один раз перед виконанням перетворення Гільберта, щоб створити контекст для обчислень. Структура ***pSpec*** містить усі необхідні коефіцієнти та параметри, що дозволяє прискорити подальші виклики ***ippsHilbert*** без повторної ініціалізації.

Hilbert

Обчислює аналітичний сигнал за допомогою перетворення Гільберта.

Синтаксис:

IppStatus ippsHilbert_32f32fc(const Ipp32f* pSrc, Ipp32fc* pDst, int length, const IppsHilbertSpec* pSpec, Ipp8u* pBuffer);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc Вказівник на вхідний дійсний вектор.

pDst Вказівник на вихідний комплексний вектор (аналітичний сигнал).

length Кількість відліків, що обробляються.

pSpec Вказівник на попередньо ініціалізовану структуру специфікації перетворення Гільберта.

pBuffer Вказівник на зовнішній робочий буфер.

Опис:

Функція обчислює комплексний аналітичний сигнал

$x_a[n] = x[n] + j\,\mathcal{H}\{x[n]\}$, де $\mathcal{H}\{\cdot\}$ —

перетворення Гільберта. Дійсна частина виходу дорівнює вхідному сигналу, уявна є його версією зі зсувом фази на 90°. Перед викликом необхідно визначити розміри буферів за допомогою ***ippsHilbertGetSize*** та ініціалізувати структуру ***pSpec*** викликом ippsHilbertInit. Розмір робочого буфера ***pBuffer*** має відповідати значенню, поверненому ***ippsHilbertGetSize***; за наявності достатнього зовнішнього буфера уникається внутрішнє виділення пам’яті та поліпшується продуктивність.

6. ФУНКЦІЇ ВЕЙВЛЕТНИХ ПЕРЕТВОРЕНЬ

У цьому розділі описуються функції вейвлет-перетворень, реалізовані в бібліотеці Intel® Integrated Performance Primitives (Intel® IPP) [1].

Під час обробки сигналів дані можуть бути представлені як у частотній, так і у частотно-часовій областях. У багатьох практичних випадках вейвлет-перетворення виступає альтернативою перетворенню Фур'є, оскільки забезпечує кращу локалізацію сигналу в часі та частоті.

Дискретне вейвлет-перетворення (DWT) подає сигнал у вигляді набору коефіцієнтів $a_{i,k}$ із двома індексами — один відповідає «частотній» характеристиці (масштабу), а інший визначає часову локалізацію. Коефіцієнт показує локальну амплітуду відповідної базисної хвилі, тобто одну з функцій основи перетворення.

Індекс, що відповідає частоті, фактично задає масштаб хвилі — зменшення базової хвилі в часі у 2^n разів дозволяє аналізувати сигнали на різних рівнях роздільності.

Такі перетворення дають змогу створювати ефективні реалізації, подібні до швидкого вейвлет-перетворення (FWT) за аналогією з швидким перетворенням Фур'є (FFT).

На графічному зображенні площина часу – частоти поділена на сегменти, що відповідають локальним амплітудам хвиль, тобто областям концентрації енергії сигналу.

У бібліотеці Intel IPP реалізовано саме цей тип перетворень — дискретне вейвлет-перетворення (DWT), який забезпечує компактне подання сигналу та зручний інструмент для його багаторівневого аналізу й компресії.

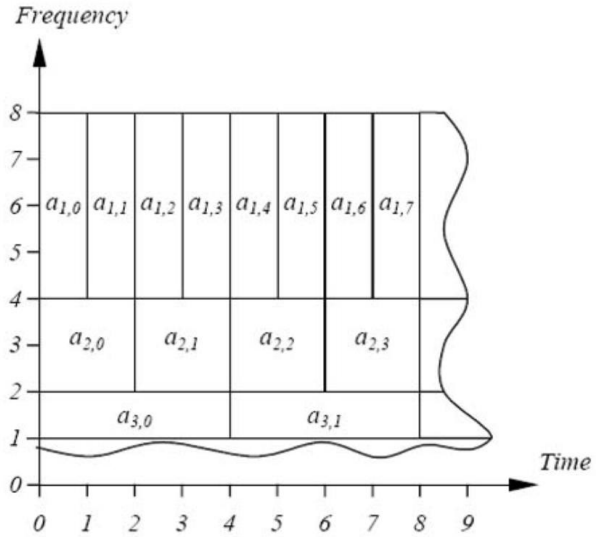


Рис. 6.1 Коефіцієнти розкладання вейвлетів у частотно-часовій області

Дискретне вейвлет-перетворення (DWT) є одним із методів вейвлет-аналізу, що базується на використанні базисних функцій, пов'язаних із

масштабним коефіцієнтом 2. Таким чином, DWT має спільну фундаментальну основу з іншими методами пакетного аналізу сигналів.

Аналогічно можна визначити ще один базовий елемент, який використовується для реконструкції (синтезу) сигналу, — це однорівневе обернене вейвлет-перетворення. На рисунку «Three-Level Discrete Wavelet Decomposition» подано схему прямого DWT, що забезпечує перехід до часово-частотного представлення, показаного на попередньому рисунку. Діаграма містить три рівні декомпозиції. Відповідна процедура відновлення сигналу за допомогою послідовного застосування елементарного однорівневого оберненого перетворення зображена на рисунку «Three-Level Discrete Wavelet Reconstruction».

Реалізація дискретних багатомасштабних перетворень ґрунтується на використанні інтерполяційних та децимаційних фільтрів із коефіцієнтом передискретизації 2. Вибір базисних функцій для декомпозиції та реконструкції сигналу однозначно визначає параметри фільтрів. Функції багатомасштабного перетворення, реалізовані в Intel IPP, використовують фільтри з кінцевою імпульсною характеристикою (FIR).

Бібліотека Intel IPP Primitives містить дві групи функцій:

1) Перетворення з фіксованими банками фільтрів, які забезпечують найвищу продуктивність.
2) Перетворення з довільно заданими фільтрами, що використовують ефективні поліфазні алгоритми фільтрації. Інтерфейс цих функцій дозволяє обробляти дані блочно, що робить їх придатними для реального часу та потокової обробки сигналів.

Трирівневе дискретне вейвлет-розкладання:

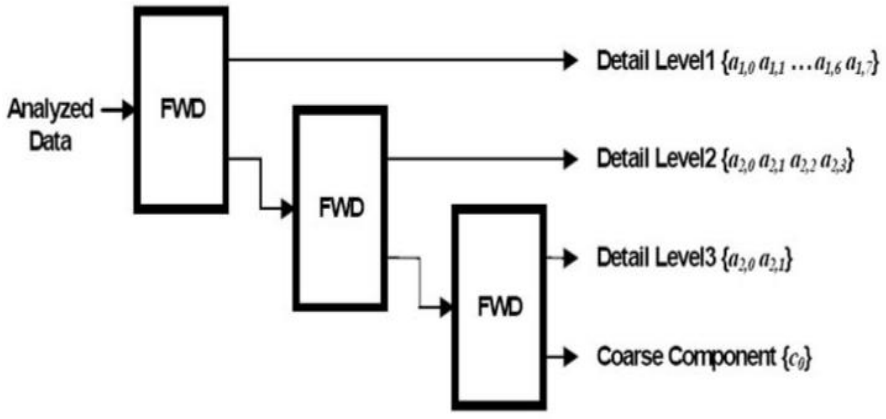


Рис. 6.2 Коефіцієнти розкладання вейвлетів у частотно-часовій області

6.1. Вейвлет - перетворення Хаара

Цей розділ описує функції, які виконують пряме або обернене перетворення вейвлетів для фіксованих банків фільтрів.

WTHaarFwd, WTHaarInv

Виконує пряме або обернене однорівневе дискретне вейвлет-перетворення Хаара.

Синтаксис:
Випадок 1: пряме перетворення

IppStatus ippsWTHaarFwd_32f (const Ipp32f* pSrc, int len, Ipp32f* pDstLow, Ipp32f* pDstHigh);
IppStatus ippsWTHaarFwd_64f (const Ipp64f* pSrc, int len, Ipp64f* pDstLow, Ipp64f* pDstHigh);
IppStatus ippsWTHaarFwd_16s_Sfs (const Ipp16s* pSrc, int len, Ipp16s* pDstLow, Ipp16s* pDstHigh, int scaleFactor);

Випадок 2: обернене перетворення

IppStatus ippsWTHaarInv_32f (const Ipp32f* pSrcLow, const Ipp32f* pSrcHigh, Ipp32f* pDst, int len);
IppStatus ippsWTHaarInv_64f (const Ipp64f* pSrcLow, const Ipp64f* pSrcHigh, Ipp64f* pDst, int len);
IppStatus ippsWTHaarInv_16s_Sfs (const Ipp16s* pSrcLow, const Ipp16s* pSrcHigh, Ipp16s* pDst, int len, int scaleFactor);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pSrc вказівник на вхідний вектор для прямого перетворення.
len кількість елементів у векторі.
pDstLow вказівник на масив із грубими компонентами (“низькочастотні”).
pDstHigh вказівник на масив із деталями (“високочастотні”).
pSrcLow вхідний масив із грубими компонентами (“низькочастотні”) для оберненого перетворення.
pSrcHigh вхідний масив із деталями (“високочастотні”) для оберненого перетворення.
pDst вказівник на масив із відновленим сигналом.
scaleFactor масштабний коефіцієнт.

Опис:

Ці функції виконують пряме й обернене однорівневе дискретне вейвлет-перетворення Хаара. Обидва є ортогональними та забезпечують точну реконструкцію вихідного сигналу.

Пряме перетворення відповідає декомпозиції сигналу з коефіцієнтами фільтрів децимації низьких частот

$h_l = \{½, ½\}$
та високих частот
$h_h = \{½, -½\}$.

Обернене перетворення відповідає реконструкції сигналу з коефіцієнтами фільтрів інтерполяції низьких частот

$g_l = \{1, 1\}$
та високих частот
$g_h = \{-1, 1\}$.

Для забезпечення однакового діапазону значень вхідного й вихідного сигналів амплітуди частотних характеристик фільтрів нормалізуються так, що

$|H_l(0)| = 1, \quad |H_h(0.5)| = 1.$

Оскільки абсолютні значення коефіцієнтів інтерполяційних фільтрів дорівнюють 1, реконструкція сигналу виконується мінімальною кількістю операцій, що робить алгоритм придатним для застосування в системах стиснення даних. Цілочисельна реалізація може виконуватись без втрат за умови scaleFactor = −1. Для запобігання насиченню рекомендується використовувати типи з підвищеною точністю.

Якщо застосовується нормалізація за потужністю, коефіцієнти фільтрів набувають вигляду

$h_l = \{2^{-1/2}, 2^{-1/2}\}, \quad h_h = \{2^{-1/2}, -2^{-1/2}\},$
$g_l = \{2^{-1/2}, 2^{-1/2}\}, \quad g_h = \{-2^{-1/2}, 2^{-1/2}\}.$

Формули прямого перетворення (N = len):
c(k) = (x(2k) + x(2k + 1)) / 2
d(k) = (x(2k + 1) − x(2k)) / 2

де c(k) = pDstLow[k], d(k) = pDstHigh[k].

Формули оберненого перетворення:

y(2i) = c(i) − d(i)
y(2i + 1) = c(i) + d(i)

де c(i) = pSrcLow[i], d(i) = pSrcHigh[i].

Для парної довжини N: 0 ≤ k, i < N/2; компоненти мають довжину N/2, а сумарна довжина дорівнює N.

Для непарної довжини N сигнал розширюється до N + 1, при цьому
x[N] = x[N − 1],
c((N + 1)/2 − 1) = x(N − 1),
d((N + 1)/2 − 1) = 0,
а відновлений сигнал має вигляд
y(N) = c((N + 1)/2 − 1).

Особливості:

Для непарної довжини використовується симетричне продовження сигналу. У блочному режимі екстраполяція не застосовується для парних блоків; останній блок може бути непарним.

ippsWTHaarFwd — виконує пряме однорівневе дискретне перетворення Хаара для сигналу pSrc довжини len, зберігаючи низькочастотні компоненти в pDstLow і високочастотні — у pDstHigh.

ippsWTHaarInv — виконує обернене однорівневе дискретне перетворення Хаара, реконструюючи сигнал із компонентів pSrcLow та pSrcHigh.

Приклад:

```
IppStatus wthaar(void) {
   Ipp32f x[8], lo[4], hi[4];
   IppStatus status;
   ippsSet_32f(7, x, 8);
   --x[4];
   status = ippsWTHaarFwd_32f(x, 8, lo, hi);
   printf_32f("WT Haar low =", lo, 4, status);
   printf_32f("WT Haar high =", hi, 4, status);
   return status;
}
```

Вихід:

```
WT Haar low = 7.000000 7.000000 6.500000 7.000000
WT Haar high = 0.000000 0.000000 0.500000 0.000000
```

6.2. Перетворення для банку фільтрів користувачів

WTFwdGetSize, WTInvGetSize, WTFwdInit, WTInvInit, WTFwd, WTInv

Функції, описані в цьому розділі, реалізують пряме та обернене дискретне вейвлет-перетворення (DWT) для користувацьких банків фільтрів. Вони забезпечують створення та використання структур для побудови багаторівневих систем аналізу та синтезу сигналів.

Короткий опис:

WTFwdGetSize, WTInvGetSize — обчислюють розмір структур для ініціалізації прямих і обернених вейвлет-перетворень.

WTFwdInit, WTInvInit — ініціалізують відповідні структури перетворень.

WTFwd — виконує пряме вейвлет-перетворення (аналіз сигналу).

WTInv — виконує обернене вейвлет-перетворення (реконструкцію сигналу).

Синтаксис:

IppStatus ippsWTFwdGetSize (IppDataType srcType, int lenLow, int offsLow, int lenHigh, int offsHigh, int* pStateSize);

IppStatus ippsWTInvGetSize (IppDataType dstType, int lenLow, int offsLow, int lenHigh, int offsHigh, int* pStateSize);

Включити файли***: ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

srcType / dstType тип вхідних або вихідних даних (***Ipp8u, Ipp16s, Ipp32f, Ipp64f*** тощо).

lenLow довжина фільтра нижніх (низькочастотних) коефіцієнтів.

offsLow зсув для фільтра низьких частот.

lenHigh довжина фільтра верхніх (високочастотних) коефіцієнтів.

offsHigh зсув для фільтра високих частот.

pStateSize — вказівник на розмір структури стану, у байтах.

Опис:

Функції ***ippsWTFwdGetSize*** і ***ippsWTInvGetSize*** обчислюють розмір структур, необхідних для ініціалізації прямих (***ippsWTFwdInit***) і обернених (***ippsWTInvInit***) вейвлет-перетворень. Ці структури зберігають параметри фільтрів, їх коефіцієнти, довжини, зсуви та службову інформацію, потрібну для ефективної реалізації аналізу та синтезу сигналу.

Реалізація вейвлет-перетворення в Intel IPP базується на використанні фільтрів з кінцевою імпульсною характеристикою (FIR). Такі фільтри дозволяють проводити аналіз сигналу через операції згортки з подальшою децимацією (для прямого перетворення) та інтерполяцією (для оберненого).

ippsWTFwdInit створює структуру перетворення для прямого DWT, використовуючи зазначені фільтри низьких і високих частот.

ippsWTInvInit створює структуру для оберненого DWT — реконструкції сигналу.

Після ініціалізації можна використовувати:

IppStatus ippsWTFwd(...); // Пряме перетворення
IppStatus ippsWTInv(...); // Обернене перетворення

Пряме вейвлет-перетворення розкладає сигнал на низькочастотну складову (апроксимацію) та високочастотну складову (деталі), тоді як обернене — виконує повну реконструкцію сигналу з цих компонентів.

Додаткові відомості:

Фільтри користувача задають базисну функцію вейвлета (Daubechies, Symlet, Coiflet, тощо), що визначає часово-частотну локалізацію сигналу. Intel IPP дозволяє реалізовувати як ортогональні, так і біортогональні фільтри.

Для багаторівневої декомпозиції сигналу можна повторно використовувати результати попереднього рівня DWT, зменшуючи розмір сигналу вдвічі на кожному етапі.

7. ФІЛЬТРАЦІЯ

7.1. Фільтрація на основі скінченної імпульсної характеристики

SumWindow

Виконує підсумовування елементів у вікні (масці), застосованій до кожного елемента вектора.

Синтаксис:
IppStatus ippsSumWindow_8u32f(const Ipp8u* pSrc, Ipp32f* pDst, int len, int maskSize);
IppStatus ippsSumWindow_16s32f(const Ipp16s* pSrc, Ipp32f* pDst, int len, int maskSize);

Включити файли ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc	вказівник на вхідний вектор.
pDst	вказівник на вихідний вектор.
len	кількість елементів у векторі.
maskSize	розмір маски.

Опис:

Функція ***ippsSumWindow*** формує кожен елемент у вихідному векторі ***pDst*** як суму ***maskSize*** послідовних елементів із вхідного вектора ***pSrc***. Обчислення виконується за принципом ковзного вікна:

$$pDst[i] = \sum_{k=0}^{maskSize-1} pSrc[i+k]$$

Це еквівалентно простому згладжуванню сигналу або застосуванню прямого фільтра з прямокутною імпульсною характеристикою. Такий тип операції часто використовується для попередньої обробки сигналів, зменшення шуму або побудови інтегральних характеристик.

7.2. Функції фільтрації FIR

У цьому розділі описано функції, що виконують фільтрацію із кінцевою імпульсною характеристикою (FIR). Вони дозволяють ініціалізувати різні структури FIR-фільтрів, отримувати та встановлювати лінії затримки й коефіцієнти (taps), а також виконувати фільтрацію. У складі Intel IPP є функції,

які реалізують FIR-фільтри без внутрішньої лінії затримки — так звані потокові FIR-фільтри (stream FIR).

Окремий набір функцій дозволяє обчислювати коефіцієнти фільтрів різних типів (низькочастотних, високочастотних, смугових тощо).

Щоб виконати однорівневу (single-rate) FIR-фільтрацію за допомогою функції ***ippsFIRSR***, необхідно дотримуватися такої послідовності:

- Викликати ***ippsFIRSRGetSize*** для отримання розміру структури специфікації фільтра та робочого буфера.
- Викликати ***ippsFIRSRInit*** для ініціалізації структури специфікації фільтра.
- Викликати ippsFIRSR для застосування FIR-фільтра до вхідного вектора.

FIRMRGetSize

Обчислює розмір контекстної структури та робочого буфера для багатошвидкісної FIR-фільтрації (multi-rate FIR filtering).

Синтаксис:

IppStatus ippsFIRMRGetSize(int tapsLen, int upFactor, int downFactor, IppDataType tapsType, int* pSpecSize, int* pBufSize);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

tapsLen довжина FIR-фільтра.
upFactor коефіцієнт підвищення частоти дискретизації (upsampling).
downFactor коефіцієнт зниження частоти дискретизації (downsampling).
tapsType тип даних коефіцієнтів (підтримуються Ipp32f, Ipp32fc, Ipp64f, Ipp64fc).
pSpecSize вказівник на розмір структури специфікації FIR-фільтра.
pBufSize вказівник на розмір тимчасового робочого буфера.

Опис:

Ця функція обчислює:

- розмір внутрішньої структури специфікації для багатошвидкісної FIR-фільтрації, яку можна спільно використовувати кількома потоками програми;
- розмір робочого буфера, необхідного кожному потоку для виконання фільтрації.

Структура, отримана за допомогою ***ippsFIRMRGetSize***, використовується під час ініціалізації (***ippsFIRMRInit***) та подальшої фільтрації сигналів із різними частотами дискретизації (***ippsFIRMR***).

Для прикладу використання див. демонстраційний фрагмент у розділі з описом функції FIRMR.

FIRMRInit

Ініціалізує структуру специфікації багатошвидкісного FIR-фільтра.

Синтаксис:

IppStatus ippsFIRMRInit_<mod>(const Ipp<dataType>* pTaps, int tapsLen, int upFactor, int upPhase, int downFactor, int downPhase, IppsFIRSpec_<dataType>* pSpec);

Підтримувані значення ***mod***:

32f 64f 32fc 64fc

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pTaps вказівник на масив коефіцієнтів фільтра. Кількість елементів у масиві дорівнює tapsLen.

tapsLen кількість коефіцієнтів фільтра.

upFactor коефіцієнт підвищення частоти дискретизації (upsampling factor).

upPhase фаза розташування ненульового семплу у блоці довжиною upFactor під час підвищення частоти.

downFactor коефіцієнт зниження частоти дискретизації (downsampling factor).

downPhase фаза вибору ненульового семплу у блоці довжиною downFactor під час зниження частоти.

pSpec вказівник на структуру специфікації FIR-фільтра.

Опис:

Функція ***ippsFIRMRInit*** ініціалізує структуру специфікації багатошвидкісного FIR-фільтра у зовнішньому буфері. Перед викликом цієї функції необхідно обчислити розмір структури специфікації та робочого буфера за допомогою функції ***ippsFIRMRGetSize***.

Параметр ***upFactor*** визначає, наскільки збільшується частота дискретизації сигналу всередині фільтра (див. також опис функції ***SampleUp***). При цьому між кожними двома сусідніми вибірками вхідного сигналу вставляється ***upFactor*** − 1 нульових значень.

Параметр ***upPhase*** задає позицію ненульового семплу у кожному блоці довжиною upFactor у процесі підвищення частоти.

Параметр ***downFactor*** визначає, у скільки разів зменшується частота дискретизації вихідного сигналу після фільтрації (див. опис функції ***SampleDown***). З кожного блоку вихідних даних довжиною ***downFactor*** зберігається лише один елемент, а решта ***downFactor − 1*** вибірок відкидаються.

Параметр ***downPhase*** визначає, яка саме вибірка з кожного блоку зберігається під час процедури зниження частоти дискретизації.

Таким чином, ***ippsFIRMRInit*** формує повну структуру фільтра для комбінованої операції інтерполяції + фільтрації + децимації, забезпечуючи узгоджену часову фазу між етапами обробки. Структура, створена цією функцією, використовується в подальших викликах ***ippsFIRMR*** для виконання багатошвидкісної FIR-фільтрації сигналів.

FIRMR

Виконує багатошвидкісну (multirate) FIR-фільтрацію вхідного вектора.

Синтаксис:

IppStatus ippsFIRMR_32f(const Ipp32f* pSrc, Ipp32f* pDst, int numIters, IppsFIRSpec_32f* pSpec, const Ipp32f* pDlySrc, Ipp32f* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_64f(const Ipp64f* pSrc, Ipp64f* pDst, int numIters, IppsFIRSpec_64f* pSpec, const Ipp64f* pDlySrc, Ipp64f* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int numIters, IppsFIRSpec_32fc* pSpec, const Ipp32fc* pDlySrc, Ipp32fc* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int numIters, IppsFIRSpec_64fc* pSpec, const Ipp64fc* pDlySrc, Ipp64fc* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_16s(const Ipp16s* pSrc, Ipp16s* pDst, int numIters, IppsFIRSpec_32f* pSpec, const Ipp16s* pDlySrc, Ipp16s* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRMR_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int numIters, IppsFIRSpec_32fc* pSpec, const Ipp16sc* pDlySrc, Ipp16sc* pDlyDst, Ipp8u* pBuf);

Включені файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc Вказівник на вхідний вектор.

pDst Вказівник на вектор призначення.

numIters Кількість «ітерацій» обробки, що задає число вибірок для фільтрації. Функція фільтрує (***numIters·downFactor***) елементів із ***pSrc*** і записує (***numIters·upFactor***) елементів у ***pDst***.

pSpec Вказівник на внутрішню структуру специфікації FIR (ініціалізовану ***ippsFIRMRInit_<mod>).***

pDlySrc Вказівник на масив значень лінії затримки для входу; може бути ***NULL***. Якщо не ***NULL***, довжина дорівнює (***tapsLen+upFactor−1) / upFactor***.

pDlyDst Вказівник на масив значень лінії затримки для виходу; може бути ***NULL***. Якщо не ***NULL***, довжина = ***(tapsLen+upFactor−1)/ upFactor***.

pBuf Вказівник на робочий буфер.

Опис:

Перед викликом цієї функції слід ініціалізувати постійну специфікацію фільтра викликом ***ippsFIRMRInit_<mod>*** (а розміри структури й буфера — заздалегідь через ***ippsFIRMRGetSize***). Функція виконує багатошвидкісну фільтрацію: внутрішнє інтерполювання (***upsampling***), фільтрацію одношвидкісним FIR та подальшу децимацію (***downsampling***) — як єдину операцію. Коефіцієнти фільтра ***h(0)…h(tapsLen−1) (taps)*** задаються під час ініціалізації структури через масив ***pTaps***. Масиви ***pDlySrc*** і ***pDlyDst*** зберігають стан ліній затримки між викликами: якщо ***pDlySrc == NULL***, використовуються нульові затримки; якщо ***pDlyDst == NULL***, стан не зберігається. Параметри ***upFactor/upPhase*** визначають інтерполяцію: між кожними сусідніми вибірками вхідного сигналу вставляється ***(upFactor − 1)*** нулів, а ***upPhase*** задає позицію ненульового семплу в блоці довжини ***upFactor***. Параметри ***downFactor/downPhase*** визначають децимацію: з кожного вихідного блоку довжини ***downFactor*** зберігається лише один семпл, позицію якого задає ***downPhase***. Для обчислення перших вихідних значень використовується лінія затримки ***pDlySrc***; останні ***(tapsLen − 1)*** елементів обробленого входу зазвичай копіюються у ***pDlyDst*** для наступного виклику. Нижче наведено коректовані перші вихідні відліки (одношвидкісний еквівалент для ілюстрації згортки):

$$y(0) = h(tapsLen-1)\cdot d(0) + h(tapsLen-2)\cdot d(1) + \ldots + h(1)\cdot d(tapsLen-2) + h(0)\cdot x(0)$$

$$y(1) = h(tapsLen-1)\cdot d(1) + h(tapsLen-2)\cdot d(2) + \ldots + h(1)\cdot x(0) + h(0)\cdot x(1)$$

$$\ldots$$

$$y(tapsLen-1) = h(tapsLen-1)\cdot x(0) + \ldots + h(1)\cdot x(tapsLen-2) + h(0)\cdot x(tapsLen-1)$$

де ***d(·)*** — елементи масиву ***pDlySrc***. На практиці багатошвидкісна реалізація виконує ті самі кроки в узгодженому часово-фазовому інтерфейсі, без явного формування вставлених нулів.

Приклад:

Нижче наведено приклад використання ***ippsFIRMRGetSize***, ***ippsFIRMRInit_32f*** та ***ippsFIRMR_32f***.

```
int firmr() {
   IppsFIRSpec_32f* pSpec;
   Ipp32f pTaps[8] = {0.125f,0.125f,0.125f,0.125f,0.125f,0.125f,0.125f,0.125f};
   int i, numIters = 33, tapsLen = 8;
   int upFactor = 2, upPhase = 0, downFactor = 3, downPhase = 0;
   int specSize = 0, bufSize = 0;
   Ipp32f pSrc[/* downFactornumIters / 333];
   Ipp32f pDst[/ upFactornumIters / 233];
   Ipp32f pDlySrc[(tapsLen + upFactor - 1) / upFactor];
   Ipp32f pDlyDst[(tapsLen + upFactor - 1) / upFactor];
   Ipp8u pBuf;
   IppStatus status;

   status = ippsFIRMRGetSize(tapsLen, upFactor, downFactor, ipp32f, &specSize,
&bufSize);
   pSpec = (IppsFIRSpec_32f*)ippsMalloc_8u(specSize);
   pBuf = ippsMalloc_8u(bufSize);

   status = ippsFIRMRInit_32f(pTaps, tapsLen, upFactor, upPhase, downFactor,
downPhase, pSpec);
   if (status != ippStsNoErr) return -1;

   for (i = 0; i < downFactor * numIters; ++i) pSrc[i] = 1.0f;

   status = ippsFIRMR_32f(pSrc, pDst, numIters, pSpec, NULL, pDlyDst, pBuf);
   if (status != ippStsNoErr) return -1;

   /* подальша обробка/вивід pSrc та pDst за потреби */
   return 0;
}
```

Див. також:

FIRMRGetSize Обчислює розмір контекстної структури та робочого буфера для багатошвидкісної FIR-фільтрації.

FIRMRInit Ініціалізує структуру специфікації багатошвидкісного FIR-фільтра.

FIRSRGetSize

Обчислює розмір сталої (константної) структури та робочого буфера для одношвидкісної FIR-фільтрації.

Синтаксис:
IppStatus ippsFIRSRGetSize(int tapsLen, IppDataType tapsType, int* pSpecSize, int* pBufSize);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
tapsLen Довжина FIR-фільтра (кількість коефіцієнтів).
tapsType Тип даних коефіцієнтів. Підтримувані значення: ipp32f або ipp64f.
pSpecSize Вказівник на розмір внутрішньої сталої структури специфікації.
pBufSize Вказівник на розмір робочого буфера, необхідного для фільтрації.

Опис:
Функція обчислює: (1) розмір внутрішньої сталої структури специфікації для одношвидкісної FIR-фільтрації (цю структуру можна спільно використовувати всіма потоками застосунку); (2) розмір робочого буфера для кожного потоку.

FIRSRInit

Ініціалізує сталу (константну) структуру для одношвидкісної FIR-фільтрації.

Синтаксис:
IppStatus ippsFIRSRInit_32f(const Ipp32f* pTaps, int tapsLen, IppAlgType algType, IppsFIRSpec_32f* pSpec);
IppStatus ippsFIRSRInit_64f(const Ipp64f* pTaps, int tapsLen, IppAlgType algType, IppsFIRSpec_64f* pSpec);
IppStatus ippsFIRSRInit_32fc(const Ipp32fc* pTaps, int tapsLen, IppAlgType algType, IppsFIRSpec_32fc* pSpec);

IppStatus ippsFIRSRInit_64fc(const Ipp64fc* pTaps, int tapsLen, IppAlgType algType, IppsFIRSpec_64fc* pSpec);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pTaps вказівник на масив коефіцієнтів фільтра.
tapsLen кількість коефіцієнтів FIR-фільтра.
algType бітова маска, яка визначає тип алгоритму. Можливі значення:

- ***ippAlgAuto*** — автоматичний вибір оптимального алгоритму,
- ***ippAlgDirect*** — пряме (класичне) згортання,
- ***ippAlgFFT*** — фільтрація з використанням швидкого перетворення Фур'є (FFT).

pSpec вказівник на внутрішню сталу структуру специфікації FIR-фільтра.

Опис:
Функція ***ippsFIRSRInit*** ініціалізує сталу структуру специфікації FIR-фільтра для одношвидкісної фільтрації сигналів. Перед її викликом необхідно обчислити розмір структури та робочого буфера за допомогою функції ***ippsFIRSRGetSize***.
Після ініціалізації структура ***pSpec*** використовується в наступних викликах фільтрації, зокрема у функції ***ippsFIRSR***, що виконує одношвидкісну обробку даних.
Тип алгоритму, переданий через параметр ***algType***, визначає, чи буде використовуватися прямий метод згортки, чи FFT-реалізація, що дозволяє оптимізувати швидкодію для довгих фільтрів.

FIRSR

Виконує одношвидкісну (single-rate) FIR-фільтрацію вхідного вектора.

Синтаксис:
IppStatus ippsFIRSR_32f(const Ipp32f* pSrc, Ipp32f* pDst, int numIters, IppsFIRSpec_32f* pSpec, const Ipp32f* pDlySrc, Ipp32f* pDlyDst, Ipp8u* pBuf);
IppStatus ippsFIRSR_64f(const Ipp64f* pSrc, Ipp64f* pDst, int numIters, IppsFIRSpec_64f* pSpec, const Ipp64f* pDlySrc, Ipp64f* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int numIters, IppsFIRSpec_32fc* pSpec, const Ipp32fc* pDlySrc, Ipp32fc* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int numIters, IppsFIRSpec_64fc* pSpec, const Ipp64fc* pDlySrc, Ipp64fc* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_16s(const Ipp16s* pSrc, Ipp16s* pDst, int numIters, IppsFIRSpec_32f* pSpec, const Ipp16s* pDlySrc, Ipp16s* pDlyDst, Ipp8u* pBuf);

IppStatus ippsFIRSR_16sc(const Ipp16sc* pSrc, Ipp16sc* pDst, int numIters, IppsFIRSpec_32fc* pSpec, const Ipp16sc* pDlySrc, Ipp16sc* pDlyDst, Ipp8u* pBuf);

Включені файли: ***ipps.h***

Доменні залежності
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pSrc вказівник на вхідний вектор.

pDst вказівник на вихідний вектор.

numIters кількість елементів у вихідному векторі.

pSpec вказівник на внутрішню сталу структуру специфікації FIR-фільтра.

pDlySrc вказівник на масив значень лінії затримки для вхідного сигналу.

pDlyDst вказівник на масив значень лінії затримки для вихідного сигналу.

pBuf вказівник на робочий буфер.

Опис:

Перед використанням цієї функції потрібно ініціалізувати сталу специфікацію фільтра викликом ***ippsFIRSRInit***.

Функція виконує фільтрацію вхідного сигналу ***pSrc*** за допомогою одношвидкісного FIR-фільтра, зберігаючи результат у ***pDst***.

Фільтрація виконується за формулою:

$$y(n) = \sum_{k=0}^{tapsLen-1} h(k) \cdot x(n-k)$$

де

x(0)...x(numIters−1) — вибірки вхідного сигналу

h(0)...h(tapsLen−1) — коефіцієнти фільтра.

Для обчислення перших вихідних значень ***y(0)...y(tapsLen−1)*** функція використовує елементи лінії затримки ***pDlySrc***, довжина якої дорівнює ***tapsLen−1***.

Перші tapsLen−1 вихідних значень обчислюються наступним чином:

y(0) = h(tapsLen−1)·d(0)+h(tapsLen−2)·d(1)+…+h(1)·d(tapsLen−2) + h(0)·x(0)
y(1) = h(tapsLen−1)·d(1)+h(tapsLen−2)·d(2)+…+h(1)·x(0) + h(0)·x(1)
y(tapsLen−1) = h(tapsLen−1)·x(0) + … + h(1)·x(tapsLen−2) + h(0)·x(tapsLen−1)

де: ***d(0), d(1), d(2), …, d(tapsLen−2)*** — елементи з ***pDlySrc***.

Після обробки останні ***tapsLen−1*** елементів із ***pSrc*** копіюються у ***pDlyDst***, щоб зберегти стан фільтра між викликами функції.

Обидва масиви ***pDlySrc*** та ***pDlyDst*** можуть бути ***NULL***:
— якщо ***pDlySrc==NULL***, використовується нульова лінія затримки;
— якщо ***pDlyDst == NULL***, дані не копіюються у вихідну лінію затримки.

Ця функція є базовою для реалізації потокової FIR-фільтрації у режимі реального часу.

FIRSparseInit

Ініціалізує структуру розрідженого (sparse) FIR-фільтра.

Синтаксис:
IppStatus ippsFIRSparseInit_32f(IppsFIRSparseState_32f** ppState, const Ipp32f* pNZTaps, const Ipp32s* pNZTapPos, int nzTapsLen, const Ipp32f* pDlyLine, Ipp8u* pBuffer);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pNZTaps Вказівник на масив ненульових коефіцієнтів (***taps***); кількість елементів — ***nzTapsLen***.
pNZTapPos Вказівник на масив позицій ненульових коефіцієнтів; кількість елементів — ***nzTapsLen***.
nzTapsLen Кількість ненульових коефіцієнтів (довжина масивів ***pNZTaps*** і ***pNZTapPos***).
pDlyLine Вказівник на масив значень лінії затримки.
ppState Подвійний вказівник на структуру стану розрідженого FIR-фільтра.
pBuffer Вказівник на зовнішній буфер для розміщення структури стану розрідженого FIR-фільтра.

Опис:

Функція ініціалізує структуру стану розрідженого FIR-фільтра ***ppState*** у зовнішньому буфері ***pBuffer***. Розмір цього буфера має бути попередньо обчислений викликом функції ***FIRSparseGetStateSize***. Під час ініціалізації значення ненульових коефіцієнтів з масиву ***pNZTaps*** (***nzTapsLen*** елементів) та їхні позиції з масиву ***pNZTapPos*** копіюються до структури стану ***ppState***. Масив ***pDlyLine*** задає значення лінії затримки; кількість елементів у цьому масиві дорівнює ***pNZTapPos[nzTapsLen−1]***. Якщо ***pDlyLine ≠ NULL***, його вміст копіюється у структуру стану; якщо ***pDlyLine == NULL***, лінія затримки в структурі стану ініціалізується нулями.

Примітка:

Значення ***nzTapsLen*** і ***pNZTapPos[nzTapsLen−1]*** мають збігатися з тими, що були передані у виклик ***FIRSparseGetStateSize***.

FIRGenHighpass

Обчислює коефіцієнти високочастотного (high-pass) FIR-фільтра.

Синтаксис:

IppStatus ippsFIRGenHighpass_64f(Ipp64f rFreq, Ipp64f* pTaps, int tapsLen, IppWinType winType, IppBool doNormal, Ipp8u* pBuffer);

Включені файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

rFreq Нормалізована частота зрізу; має лежати в діапазоні (0, 0.5).

pTaps Вказівник на масив, у який записуються обчислені коефіцієнти (***taps***); кількість елементів — ***tapsLen***.

tapsLen Кількість коефіцієнтів у масиві ***pTaps***; має бути не меншою ніж 5.

winType Тип вікна, що використовується під час обчислень:

- ***ippWinBartlett*** (вікно Бартлетта),
- ***ippWinBlackman*** (Блекмана),
- ***ippWinHamming*** (Геммінга),
- ***ippWinHann*** (Ганна).

doNormal Вибір нормалізації коефіцієнтів: ippTrue — обчислюються нормалізовані коефіцієнти; ippFalse — ненормалізовані.

pBuffer Вказівник на буфер для внутрішніх обчислень; розмір буфера отримуйте функцією ippsFIRGenGetBufferSize.

Опис:

Функція обчислює ***tapsLen*** коефіцієнтів високочастотного FIR-фільтра з частотою зрізу ***rFreq*** шляхом віконування ідеальних (нескінченних) коефіцієнтів фільтра. Тип вікна задається параметром ***winType***. Обчислені коефіцієнти записуються в масив ***pTaps***. Для визначення необхідного розміру внутрішнього буфера використовуйте функцію ***ippsFIRGenGetBufferSize***. Значення параметра ***doNormal*** визначає, чи будуть коефіцієнти нормалізовані.

FIRGenBandpass

Обчислює коефіцієнти смугового (band-pass) FIR-фільтра.

Синтаксис:

IppStatus ippsFIRGenBandpass_64f(Ipp64f rLowFreq, Ipp64f rHighFreq, Ipp64f* pTaps, int tapsLen, IppWinType winType, IppBool doNormal, Ipp8u* pBuffer);

Включені файли: і***pps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

rLowFreq Нормалізована нижня частота зрізу; має лежати в діапазоні (0, 0.5) і бути меншою за ***rHighFreq***.

rHighFreq Нормалізована верхня частота зрізу; має лежати в діапазоні (0, 0.5) і бути більшою за ***rLowFreq***.

pTaps Вказівник на масив, у який записуються обчислені коефіцієнти (***taps***); кількість елементів — ***tapsLen***.

tapsLen Кількість коефіцієнтів у масиві ***pTaps***; має бути не меншою ніж 5.

winType Тип вікна, що використовується в обчисленнях:

- ***ippWinBartlett*** (Бартлетта),
- ***ippWinBlackman*** (Блекмана),
- ***ippWinHamming*** (Геммінга),
- ***ippWinHann*** (Ганна).

doNormal Вибір нормалізації коефіцієнтів:

- ***ippTrue*** — обчислюються нормалізовані коефіцієнти;
- ***ippFalse*** — ненормалізовані.

pBuffer Вказівник на буфер для внутрішніх обчислень; розмір буфера отримуйте функцією ***ippsFIRGenGetBufferSize***.

Опис:

Функція обчислює ***tapsLen*** коефіцієнтів смугового FIR-фільтра з частотами зрізу ***rLowFreq*** і ***rHighFreq*** шляхом віконування ідеальних (нескінченних) коефіцієнтів фільтра. Тип вікна задається параметром ***winType***. Обчислені коефіцієнти записуються в масив ***pTaps***. Значення параметра ***doNormal*** визначає, чи будуть коефіцієнти нормалізовані. Для визначення необхідного розміру внутрішнього буфера використовуйте функцію ***ippsFIRGenGetBufferSize***.

FIRGenBandpass

Обчислює коефіцієнти смугового (band-pass) FIR-фільтра.

Синтаксис:

IppStatus ippsFIRGenBandpass_64f(Ipp64f rLowFreq, Ipp64f rHighFreq, Ipp64f* pTaps, int tapsLen, IppWinType winType, IppBool doNormal, Ipp8u* pBuffer);

Включені файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

rLowFreq Нормалізована нижня частота зрізу; має лежати в діапазоні (0, 0.5) і бути меншою за rHighFreq.

rHighFreq Нормалізована верхня частота зрізу; має лежати в діапазоні (0, 0.5) і бути більшою за rLowFreq.

pTaps Вказівник на масив, у який записуються обчислені коефіцієнти (***taps***); кількість елементів — ***tapsLen***.

tapsLen Кількість коефіцієнтів у масиві ***pTaps***; має бути не меншою ніж 5.

winType Тип вікна, що використовується в обчисленнях:

- ***ippWinBartlett*** (Бартлетта),
- ippWinBlackman (Блекмана),
- ippWinHamming (Геммінга),
- ippWinHann (Ганна).

doNormal Вибір нормалізації коефіцієнтів:

- ippTrue — обчислюються нормалізовані коефіцієнти;
- ippFalse — ненормалізовані.

pBuffer Вказівник на буфер для внутрішніх обчислень; розмір буфера отримуйте функцією ***ippsFIRGenGetBufferSize***.

Опис:

Функція обчислює ***tapsLen*** коефіцієнтів смугового FIR-фільтра з частотами зрізу ***rLowFreq*** і ***rHighFreq*** шляхом віконування ідеальних (нескінченних) коефіцієнтів фільтра. Тип вікна задається параметром ***winType***. Обчислені коефіцієнти записуються в масив ***pTaps***. Значення параметра ***doNormal*** визначає, чи будуть коефіцієнти нормалізовані. Для визначення необхідного розміру внутрішнього буфера використовуйте функцію ***ippsFIRGenGetBufferSize***.

7.3. Фільтрація на основі IIR

IIR-фільтри

У цьому розділі описуються функції, які ініціалізують фільтри з нескінченною імпульсною характеристикою (IIR) та виконують фільтрацію.

Intel IPP підтримує два типи IIR-фільтрів: фільтр довільного порядку та бікуад-фільтр.

Структура фільтра довільного порядку

На рисунку 7.1 показано загальну схему IIR-фільтра довільного порядку:

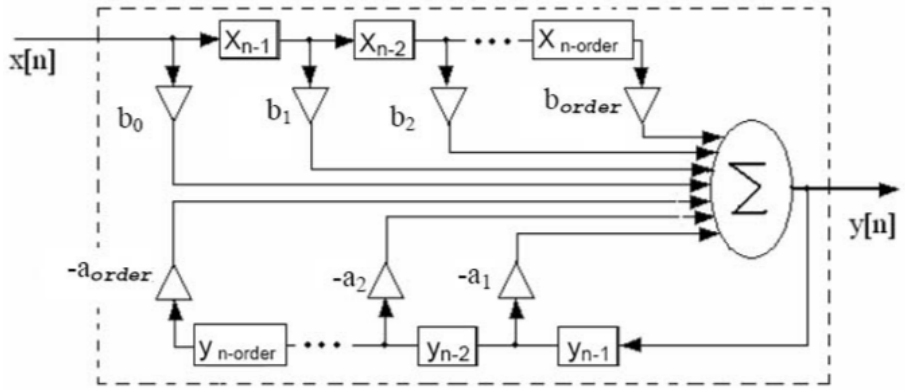


Рис.7.1 - Загальна схема IIR-фільтра довільного порядку

Тут ***x[n]*** — це вхідний сигнал, ***y[n]*** — вихідний сигнал, ***order*** — порядок фільтра, а коефіцієнти ***b_0, b_1, …, b_order, a_1, …, a_order*** — це зведені коефіцієнти фільтра.

Вихідний сигнал обчислюється за формулою:

$$y[n] = \sum_{k=0}^{\text{order}} b_k \, x[n-k] \; - \; \sum_{k=1}^{\text{order}} a_k \, y[n-k]$$

$x[n]$ — вхідний сигнал,

$y[n]$ — вихідний сигнал,

b_k — коефіцієнти чисельника (feed-forward частина),

a_k — коефіцієнти знаменника (feedback части…)

order — порядок фільтра.

Зведені коефіцієнти визначаються як:

$a_k = A_k / A_0$, $b_k = B_k / A_0$,

де ***A_0, A_1, …, A_order, B_0, B_1, …, B_order*** — початкові коефіцієнти (taps) фільтра.

Біквад-фільтр (BiQuad IIR Filter)

Біквад-фільтр — це каскад друго-порядкових фільтрів. На рисунку «Structure of a BiQuad IIR Filter» показано структуру такого фільтра з k каскадів. Кожен каскад виконує другопорядкову фільтрацію, забезпечуючи гнучкість і чисельну стабільність при реалізації фільтрів високого порядку:

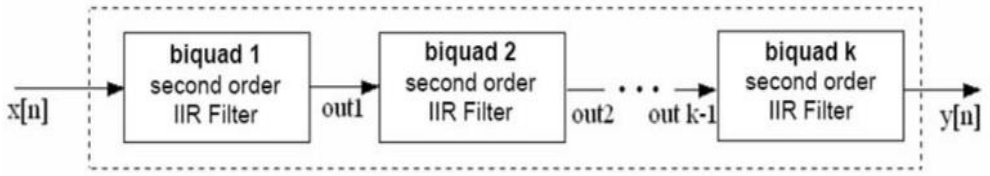


Рис.7.2 – Структура біквад-фільтра

Форми реалізації DF1 та DF2

За замовчуванням усі функції Intel IPP IIR-фільтрів, які не мають суфікса ***_DF1_,*** використовують затримувальну лінію прямої форми ***2 (DF2).***

Різниця між ***DF1*** і ***DF2*** полягає в тому, що ***DF1*** зберігає затримані значення вхідного та вихідного сигналів, тоді як ***DF2*** є вдвічі коротшою і містить попередньо обчислені значення згідно з алгоритмом [Opp75]:

```
for (i = 0; i < order; i++) {
   pDly[i] = 0;
   for (n = order - i; n > 0; n--) {
      pDly[i] += pTaps[n + i] * pSrc[len - n];  /* коефіцієнти b */
   }
}
for (i = 0; i < order; i++) {
   for (n = order - i; n > 0; n--) {
      pDly[i] -= pTaps[order + n + i] * pDst[len - n];  /* коефіцієнти a */
   }
}
```

Перетворити ***DF2*** назад у ***DF1*** неможливо. Тому, якщо необхідно отримати форму ***DF1***, слід скопіювати останні order значень вхідного та вихідного сигналів у буфер ***DF1***.

Функції ***ippsIIRSetDlyLine*** і ***ippsIIRGetDlyLine*** також працюють із затримувальною лінією у форматі ***DF2***.

Загальна схема використання IIR-фільтра:

Викличте ***ippsIIRInit*** для ініціалізації IIR-фільтра довільного порядку у зовнішньому буфері, або ***ippsIIRInit_BiQuad*** — для ініціалізації біквад-фільтра (каскаду другого порядку фільтра).

Розмір необхідного буфера обчислюється функціями ***ippsIIRGetStateSize*** або ***ippsIIRGetStateSize_BiQuad*** відповідно.

Викличте ***ippsIIR*** для фільтрації послідовності вибірок.

За потреби використовуйте ippsIIRGetDlyLine та ippsIIRSetDlyLine для отримання чи задання значень затримувальної лінії у структурі стану IIR-фільтра.

IIRInit

Ініціалізує стан довільного IIR-фільтра.

Синтаксис:

Випадок 1: робота з цілими вибірками

IppStatus ippsIIRInit32f_16s(IppsIIRState32f_16s** ppState, const Ipp32f* pTaps, int order, const Ipp32f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_16s(IppsIIRState64f_16s** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_32s(IppsIIRState64f_32s** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit32fc_16sc(IppsIIRState32fc_16sc** ppState, const Ipp32fc* pTaps, int order, const Ipp32fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_16sc(IppsIIRState64fc_16sc** ppState, const Ipp64fc* pTaps, int order, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_32sc(IppsIIRState64fc_32sc** ppState, const Ipp64fc* pTaps, int order, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

Випадок 2: робота з дійсними вибірками з плаваючою комою

IppStatus ippsIIRInit_32f(IppsIIRState_32f** ppState, const Ipp32f* pTaps, int order, const Ipp32f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_32f(IppsIIRState64f_32f** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_64f(IppsIIRState_64f** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_32fc(IppsIIRState_32fc** ppState, const Ipp32fc* pTaps, int order, const Ipp32fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_32fc(IppsIIRState64fc_32fc** ppState, const Ipp64fc* pTaps, int order, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_64fc(IppsIIRState_64fc** ppState, const Ipp64fc* pTaps, int order, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pTaps Вказівник на масив коефіцієнтів фільтра (taps). Кількість елементів дорівнює 2*(order+1).

order Порядок IIR-фільтра.

pDlyLine Вказівник на масив значень лінії затримки; кількість елементів дорівнює order.

ppState Вказівник на вказівник структури стану довільного IIR-фільтра, яка створюється.

pBuf Вказівник на зовнішній робочий буфер.

Опис:

Функція ініціалізує структуру стану довільного IIR-фільтра у зовнішньому буфері. Розмір цього буфера потрібно попередньо обчислити за допомогою функції ***IIRGetStateSize***.

Під час ініціалізації коефіцієнти з масиву ***pTaps*** копіюються у структуру стану ***pState***. Масив ***pDlyLine*** містить початкові значення лінії затримки. Якщо ***pDlyLine ≠ NULL***, то його значення копіюються в контекст фільтра; інакше лінія затримки ініціалізується нулями.

Порядок фільтра визначається параметром ***order***, який дорівнює 0 для нульового порядку. Масив коефіцієнтів довжиною ***2*(order+1)*** задається у вигляді:

$B_0, B_1, \ldots, B\_order, A_0, A_1, \ldots, A\_order,$

де $A_0 \neq 0.$

Якщо структуру стану не створено, функція повертає статус помилки.

Функції з суфіксом ***32s_32f***, викликані з коефіцієнтами з плаваючою комою, автоматично перетворюють їх у цілі значення з масштабуванням для підвищення точності.

Усі реалізації виконують відповідне масштабування при переході до цілочисельного формату для досягнення максимальної точності розрахунків.

IIRInit_BiQuad

Ініціалізує стан біквад-фільтра (IIR-фільтра, реалізованого як каскад біквадів).

Синтаксис:

Випадок 1: робота з цілими вибірками

IppStatus ippsIIRInit32f_BiQuad_16s(IppsIIRState32f_16s** ppState, const Ipp32f* pTaps, int numBq, const Ipp32f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_BiQuad_16s(IppsIIRState64f_16s** ppState, const Ipp64f* pTaps, int numBq, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_BiQuad_32s(IppsIIRState64f_32s** ppState, const Ipp64f* pTaps, int numBq, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit32fc_BiQuad_16sc(IppsIIRState32fc_16sc** ppState, const Ipp32fc* pTaps, int numBq, const Ipp32fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_BiQuad_16sc(IppsIIRState64fc_16sc** ppState, const Ipp64fc* pTaps, int numBq, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_BiQuad_32sc(IppsIIRState64fc_32sc** ppState, const Ipp64fc* pTaps, int numBq, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_BiQuad_DF1_32s(IppsIIRState64f_32s** ppState, const Ipp64f* pTaps, int numBq, const Ipp32s* pDlyLine, Ipp8u* pBuf);

Випадок 2: робота з дійсними вибірками з плаваючою комою

IppStatus ippsIIRInit_BiQuad_32f(IppsIIRState_32f** ppState, const Ipp32f* pTaps, int numBq, const Ipp32f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64f_BiQuad_32f(IppsIIRState64f_32f** ppState, const Ipp64f* pTaps, int numBq, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_BiQuad_64f(IppsIIRState_64f** ppState, const Ipp64f* pTaps, int numBq, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_BiQuad_32fc(IppsIIRState_32fc** ppState, const Ipp32fc* pTaps, int numBq, const Ipp32fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit64fc_BiQuad_32fc(IppsIIRState64fc_32fc** ppState, const Ipp64fc* pTaps, int numBq, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_BiQuad_64fc(IppsIIRState_64fc** ppState, const Ipp64fc* pTaps, int numBq, const Ipp64fc* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRInit_BiQuad_DF1_32f(IppsIIRState_32f** pState, const Ipp32f* pTaps, int numBq, const Ipp32f* pDlyLine, Ipp8u* pBuf);

Включені файли:***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pTaps	Вказівник на масив коефіцієнтів фільтра. Кількість елементів у масиві дорівнює ***6 * numBq.***
numBq	Кількість каскадів біквадів.
pDlyLine	Вказівник на масив значень лінії затримки. Кількість елементів у масиві дорівнює ***2 * numBq.***
ppState	Вказівник на вказівник структури стану біквад-фільтра.
pBuf	Вказівник на зовнішній буфер.
pState	Вказівник на структуру стану фільтра.

Опис:

Функція ***ippsIIRInit_BiQuad*** ініціалізує стан біквад-фільтра (каскаду друго-порядкових IIR-фільтрів) у зовнішньому буфері. Розмір буфера необхідно попередньо обчислити за допомогою функції ***ippsIIRGetStateSize_BiQuad***.Під час ініціалізації коефіцієнти з масиву ***pTaps*** копіюються в структуру стану ***ppState***. Масив ***pDlyLine*** визначає початкові значення лінії затримки. Для функцій з суфіксом ***_DF1*** кількість елементів у ***pDlyLine*** дорівнює ***4 * numBq***, а для інших варіантів — ***2 * numBq***.

Якщо ***pDlyLine ≠ NULL***, то його значення копіюються до структури фільтра; інакше всі значення лінії затримки ініціалізуються нулями.

Варіант функції ***ippsIIRInit_BiQuad_DF1*** працює з лінією затримки, розташованою в пам'яті у вигляді:

$x_{0,-2}, x_{0,-1}, y_{0,-2}, y_{0,-1}, x_{1,-2}, x_{1,-1}, y_{1,-2}, y_{1,-1}, \ldots, x_{n-1,-2}, x_{n-1,-1}, y_{n-1,-2}, y_{n-1,-1}$

Біквад-фільтр визначається як послідовне з'єднання ***numBq*** каскадів. Масив коефіцієнтів довжиною ***6 * numBq*** має структуру:

$B_{0,0}, B_{0,1}, B_{0,2}, A_{0,0}, A_{0,1}, A_{0,2};$
$B_{1,0}, B_{1,1}, B_{1,2}, A_{1,0}, A_{1,1}, A_{1,2};$
$\ldots$
$B_{n-1,0}, B_{n-1,1}, B_{n-1,2}, A_{n-1,0}, A_{n-1,1}, A_{n-1,2}$

де $A_{n,0} \neq 0$ та $B_{n,0} \neq 0$.

Якщо структуру стану не створено, функція повертає статус помилки. Функції з суфіксом **32s_32f**, викликані з дійсними коефіцієнтами з плаваючою комою, автоматично конвертують їх у цілочисельний формат із масштабуванням для підвищення точності.

У всіх випадках дані, що зберігаються у структурі, масштабуються для збереження максимальної чисельної стабільності.

IIRGetStateSize

Обчислює розмір зовнішнього буфера для структури стану довільного IIR-фільтра.

Синтаксис:

```
IppStatus ippsIIRGetStateSize32f_16s(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_16s(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_32s(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize32fc_16sc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_16sc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_32sc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize_32f(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_32f(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize_64f(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize_32fc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_32fc(int order, int* pBufferSize);
IppStatus ippsIIRGetStateSize_64fc(int order, int* pBufferSize);
```

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри**:**
order порядок IIR-фільтра.
pBufferSize вказівник на змінну, у яку буде записано обчислений розмір буфера (у байтах).

Опис:
Функція ***ippsIIRGetStateSize*** обчислює необхідний розмір зовнішнього буфера для збереження структури стану довільного IIR-фільтра та записує результат у змінну, на яку вказує параметр ***pBufferSize***.

Для обчислення розміру потрібно задати параметр ***order***, який визначає порядок фільтра.

Повертає:
ippStsNoErr — помилок не виявлено;
ippStsNullPtrErr — помилка, якщо вказівник **pBufferSize** дорівнює NULL;
ippStsIIROrderErr — помилка, якщо **order ≤ 0**.

IIRGetStateSize_BiQuad

Обчислює розмір зовнішнього буфера для структури стану біквад-фільтра (IIR-фільтра, реалізованого як каскад біквадів).

Синтаксис:

IppStatus ippsIIRGetStateSize32f_BiQuad_16s(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_BiQuad_16s(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_BiQuad_32s(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize32fc_BiQuad_16sc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_BiQuad_16sc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_BiQuad_32sc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_32f(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_BiQuad_32f(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_64f(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_32fc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64fc_BiQuad_32fc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_64fc(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize64f_BiQuad_DF1_32s(int numBq, int* pBufferSize);
IppStatus ippsIIRGetStateSize_BiQuad_DF1_32f(int numBq, int* pBufferSize);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
numBq кількість каскадів біквадів (кількість секцій другого порядку у фільтрі).
pBufferSize вказівник на змінну, у яку буде записано обчислений розмір буфера (у байтах).

Опис:
Функція **ippsIIRGetStateSize_BiQuad** обчислює необхідний розмір зовнішнього буфера для збереження структури стану біквад-фільтра (IIR-фільтра, побудованого як каскад друго-порядкових секцій). Результат записується у змінну, на яку вказує **pBufferSize**.
Щоб коректно визначити розмір буфера, потрібно задати параметр **numBq**, який визначає кількість каскадів біквадів у фільтрі.

Повертає:
- **ippStsNoErr** — помилок не виявлено;
- **ippStsNullPtrErr** — помилка, якщо вказівник **pBufferSize=*NULL***;
- **ippStsIIROrderErr** — помилка, якщо **numBq ≤ 0**.

IIRGetDlyLine

Отримує вміст лінії затримки зі структури стану IIR-фільтра.

Синтаксис:

IppStatus ippsIIRGetDlyLine32f_16s(const IppsIIRState32f_16s* pState, Ipp32f* pDlyLine);
IppStatus ippsIIRGetDlyLine64f_16s(const IppsIIRState64f_16s* pState, Ipp64f* pDlyLine);
IppStatus ippsIIRGetDlyLine64f_32s(const IppsIIRState64f_32s* pState, Ipp64f* pDlyLine);
IppStatus ippsIIRGetDlyLine32fc_16sc(const IppsIIRState32fc_16sc* pState, Ipp32fc* pDlyLine);
IppStatus ippsIIRGetDlyLine64fc_16sc(const IppsIIRState64fc_16sc* pState, Ipp64fc* pDlyLine);
IppStatus ippsIIRGetDlyLine64fc_32sc(const IppsIIRState64fc_32sc* pState, Ipp64fc* pDlyLine);
IppStatus ippsIIRGetDlyLine_32f(const IppsIIRState_32f* pState, Ipp32f* pDlyLine);
IppStatus ippsIIRGetDlyLine64f_32f(const IppsIIRState64f_32f* pState, Ipp64f* pDlyLine);
IppStatus ippsIIRGetDlyLine_64f(const IppsIIRState_64f* pState, Ipp64f* pDlyLine);
IppStatus ippsIIRGetDlyLine_32fc(const IppsIIRState_32fc* pState, Ipp32fc* pDlyLine);
IppStatus ippsIIRGetDlyLine64fc_32fc(const IppsIIRState64fc_32fc* pState, Ipp64fc* pDlyLine);
IppStatus ippsIIRGetDlyLine_64fc(const IppsIIRState_64fc* pState, Ipp64fc* pDlyLine);
IppStatus ippsIIRGetDlyLine64f_DF1_32s(const IppsIIRState64f_32s* pState, Ipp32s* pDlyLine);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Опис:
Функція ***ippsIIRGetDlyLine*** копіює поточні значення лінії затримки з відповідної структури стану ***pState*** у масив ***pDlyLine***.
Якщо вказівник ***pDlyLine*** дорівнює ***NULL***, то лінія затримки у структурі стану ініціалізується нульовими значеннями. Перед викликом цієї функції необхідно попередньо ініціалізувати стан фільтра за допомогою однієї з функцій ініціалізації (***ippsIIRInit*** або ***ippsIIRInit_BiQuad***).Таким чином, ***ippsIIRGetDlyLine*** дає змогу зчитати поточний вміст внутрішньої пам'яті фільтра, що містить попередні вхідні та вихідні вибірки, необхідні для подальшої фільтрації сигналу.

IIRSetDlyLine

Встановлює вміст лінії затримки у структурі стану IIR-фільтра.

Синтаксис:

IppStatus ippsIIRSetDlyLine32f_16s(IppsIIRState32f_16s* pState, const Ipp32f* pDlyLine);
IppStatus ippsIIRSetDlyLine64f_16s(IppsIIRState64f_16s* pState, const Ipp64f* pDlyLine);
IppStatus ippsIIRSetDlyLine64f_32s(IppsIIRState64f_32s* pState, const Ipp64f* pDlyLine);
IppStatus ippsIIRSetDlyLine32fc_16sc(IppsIIRState32fc_16sc* pState, const Ipp32fc* pDlyLine);
IppStatus ippsIIRSetDlyLine64fc_16sc(IppsIIRState64fc_16sc* pState, const Ipp64fc* pDlyLine);
IppStatus ippsIIRSetDlyLine64fc_32sc(IppsIIRState64fc_32sc* pState, const Ipp64fc* pDlyLine);
IppStatus ippsIIRSetDlyLine_32f(IppsIIRState_32f* pState, const Ipp32f* pDlyLine);
IppStatus ippsIIRSetDlyLine64f_32f(IppsIIRState64f_32f* pState, const Ipp64f* pDlyLine);
IppStatus ippsIIRSetDlyLine_64f(IppsIIRState_64f* pState, const Ipp64f* pDlyLine);
IppStatus ippsIIRSetDlyLine_32fc(IppsIIRState_32fc* pState, const Ipp32fc* pDlyLine);
IppStatus ippsIIRSetDlyLine64fc_32fc(IppsIIRState64fc_32fc* pState, const Ipp64fc* pDlyLine);
IppStatus ippsIIRSetDlyLine_64fc(IppsIIRState_64fc* pState, const Ipp64fc* pDlyLine);
IppStatus ippsIIRSetDlyLine64f_DF1_32s(IppsIIRState64f_32s* pState, const Ipp32s* pDlyLine);

Включені файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pState	вказівник на структуру стану IIR-фільтра.
pDlyLine	вказівник на масив, що містить значення лінії затримки.

Кількість елементів у масиві становить order для довільних IIR-фільтрів або ***2*numBq*** для біквад-фільтрів (каскад другого порядку).

Якщо вказівник ***pDlyLine*** дорівнює ***NULL***, значення лінії затримки у структурі стану ініціалізуються нулями.

Опис:

Функція ***ippsIIRSetDlyLine*** копіює вказані значення лінії затримки з масиву ***pDlyLine*** у структуру стану ***pState***. Якщо вказівник ***pDlyLine*** не задано (тобто ***NULL***), усі значення лінії затримки у структурі стану встановлюються рівними нулю.

Перед викликом цієї функції необхідно попередньо ініціалізувати структуру стану IIR-фільтра за допомогою однієї з функцій ініціалізації, таких як ippsIIRInit або ***ippsIIRInit_BiQuad***.

Ця функція зазвичай використовується для оновлення внутрішніх затриманих вибірок фільтра після часткової обробки сигналу або для відновлення стану фільтра з попереднього сеансу обчислень.

IIR

Виконує фільтрацію вхідного сигналу за допомогою IIR-фільтра.

Синтаксис:

Випадок 1 (не на місці, цілі числа):

IppStatus ippsIIR32f_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, IppsIIRState32f_16s* pState, int scaleFactor);

IppStatus ippsIIR64f_16s_Sfs(const Ipp16s* pSrc, Ipp16s* pDst, int len, IppsIIRState64f_16s* pState, int scaleFactor);

IppStatus ippsIIR64f_32s_Sfs(const Ipp32s* pSrc, Ipp32s* pDst, int len, IppsIIRState64f_32s* pState, int scaleFactor);

IppStatus ippsIIR32fc_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, IppsIIRState32fc_16sc* pState, int scaleFactor);

IppStatus ippsIIR64fc_16sc_Sfs(const Ipp16sc* pSrc, Ipp16sc* pDst, int len, IppsIIRState64fc_16sc* pState, int scaleFactor);

IppStatus ippsIIR64fc_32sc_Sfs(const Ipp32sc* pSrc, Ipp32sc* pDst, int len, IppsIIRState64fc_32sc* pState, int scaleFactor).

Випадок 2 (не на місці, числа з плаваючою крапкою):

IppStatus ippsIIR_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, IppsIIRState_32f* pState);
IppStatus ippsIIR_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, IppsIIRState_64f* pState);
IppStatus ippsIIR64f_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, IppsIIRState64f_32f* pState);
IppStatus ippsIIR_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, IppsIIRState_32fc* pState);
IppStatus ippsIIR_64fc(const Ipp64fc* pSrc, Ipp64fc* pDst, int len, IppsIIRState_64fc* pState);

IppStatus ippsIIR64fc_32fc(const Ipp32fc* pSrc, Ipp32fc* pDst, int len, IppsIIRState64fc_32fc* pState).

Випадок 3 (на місці, цілі числа):

IppStatus ippsIIR32f_16s_ISfs(Ipp16s* pSrcDst, int len, IppsIIRState32f_16s* pState, int scaleFactor);
IppStatus ippsIIR64f_16s_ISfs(Ipp16s* pSrcDst, int len, IppsIIRState64f_16s* pState, int scaleFactor);
IppStatus ippsIIR64f_32s_ISfs(Ipp32s* pSrcDst, int len, IppsIIRState64f_32s* pState, int scaleFactor);
IppStatus ippsIIR32fc_16sc_ISfs(Ipp16sc* pSrcDst, int len, IppsIIRState32fc_16sc* pState, int scaleFactor);
IppStatus ippsIIR64fc_16sc_ISfs(Ipp16sc* pSrcDst, int len, IppsIIRState64fc_16sc* pState, int scaleFactor);
IppStatus ippsIIR64fc_32sc_ISfs(Ipp32sc* pSrcDst, int len, IppsIIRState64fc_32sc* pState, int scaleFactor).

Випадок 4 (на місці, числа з плаваючою крапкою):

IppStatus ippsIIR_32f_I(Ipp32f* pSrcDst, int len, IppsIIRState_32f* pState);
IppStatus ippsIIR_64f_I(Ipp64f* pSrcDst, int len, IppsIIRState_64f* pState);
IppStatus ippsIIR64f_32f_I(Ipp32f* pSrcDst, int len, IppsIIRState64f_32f* pState);
IppStatus ippsIIR_32fc_I(Ipp32fc* pSrcDst, int len, IppsIIRState_32fc* pState);
IppStatus ippsIIR_64fc_I(Ipp64fc* pSrcDst, int len, IppsIIRState_64fc* pState);
IppStatus ippsIIR64fc_32fc_I(Ipp32fc* pSrcDst, int len, IppsIIRState64fc_32fc* pState).

Випадок 5 (багатоканальна обробка):

IppStatus ippsIIR_32f_P(const Ipp32f** ppSrc, Ipp32f** ppDst, int len, int nChannels, IppsIIRState_32f** ppState);
IppStatus ippsIIR64f_32s_PSfs(const Ipp32s** ppSrc, Ipp32s** ppDst, int len, int nChannels, IppsIIRState64f_32s** ppState, int* pScaleFactor);
IppStatus ippsIIR_32f_IP(Ipp32f** ppSrcDst, int len, int nChannels, IppsIIRState_32f** ppState);
IppStatus ippsIIR64f_32s_IPSfs(Ipp32s** ppSrcDst, int len, int nChannels, IppsIIRState64f_32s** ppState, int* pScaleFactor).

Включені файли: ***ipps.h***
Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pState Вказівник на структуру стану IIR-фільтра.
ppState Вказівник на масив вказівників на структури станів (для багатоканальної обробки).
pSrc, ppSrc Вказівники на вхідниі вектори.
pDst, ppDst Вказівники на вихідні вектори.
pSrcDst, ppSrcDst Вказівники на вектори для операцій на місці.

len Кількість елементів у векторі.

nChannels Кількість векторів (каналів), що фільтруються одночасно.

scaleFactor, pScaleFactor Коефіцієнт масштабування для цілочисельних варіантів (див. Integer Scaling).

Опис:

Функції сімейства ***ippsIIR*** фільтрують ***len*** елементів вхідного вектора ***pSrc*** (або ***pSrcDst*** для операцій на місці) за параметрами, збереженими у структурі стану ***pState***, і записують результат у ***pDst*** (або назад у ***pSrcDst***). Для цілочисельних реалізацій вихід масштабують згідно зі ***scaleFactor***; у разі переповнення застосовується насичення. Перед викликом потрібно ініціалізувати стан фільтра однією з функцій ініціалізації (***ippsIIRInit*** або ***ippsIIRInit_BiQuad***), визначивши порядок/кількість біквадів, значення тапів ***pTaps*** та початкову лінію затримки ***pDlyLine***. У багатоканальних варіантах (Випадок 5) обробляються ***nChannels*** векторів однакової довжини ***len***; кожен канал має власну ініціалізовану структуру стану. Не змінюйте ***scaleFactor*** без зміни (переініціалізації) структури стану.

IIRSparseInit

Ініціалізує розріджену (sparse) структуру стану IIR-фільтра.

Синтаксис:

IppStatus ippsIIRSparseInit_32f(IppsIIRSparseState_32f** ppState, const Ipp32f* pNZTaps, const Ipp32s* pNZTapPos, int nzTapsLen1, int nzTapsLen2, const Ipp32f* pDlyLine, Ipp8u* pBuf);

Включені файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pNZTaps Вказівник на масив ненульових коефіцієнтів (тапів) фільтра.

pNZTapPos Вказівник на масив позицій відповідних ненульових коефіцієнтів. Кількість елементів дорівнює (nzTapsLen1 + nzTapsLen2).

nzTapsLen1 Кількість ненульових b-коефіцієнтів (чисельник).

nzTapsLen2 Кількість ненульових a-коефіцієнтів (знаменник, без a0).

pDlyLine Вказівник на масив значень лінії затримки; може бути NULL.

ppState Подвійний вказівник на створювану структуру стану розрідженого IIR-фільтра.

pBuf Вказівник на зовнішній буфер для розміщення структури стану.

Опис:

Функція ініціалізує у зовнішньому буфері ***pBuf*** структуру стану розрідженого IIR-фільтра ***ppState***. Розмір цього буфера слід попередньо обчислити викликом ***ippsIIRSparseGetStateSize***. Масив довжини (***nzTapsLen1 + nzTapsLen2) pNZTaps*** задає ненульові коефіцієнти у такому порядку:

B0, B1, …, B_{nzTapsLen1−1}, A1, A2, …, A_{nzTapsLen2}

(для розрідженого подання A0 не задається, вважається 1).

Масив довжини ***(nzTapsLen1 + nzTapsLen2) pNZTapPos*** задає відповідні позиції цих ненульових коефіцієнтів у різницевому рівнянні:

BP0, BP1, …, BP_{nzTapsLen1−1}, AP0, AP1, …, AP_{nzTapsLen2−1},

де ***BP**** — зсуви для ***b***-коефіцієнтів (зазвичай ***0,1,2,***…), а ***AP0 ≠ 0*** (позиції для ***a***-коефіцієнтів відповідають членам ***y[n−k], k ≥ 1***).

Під час ініціалізації значення коефіцієнтів з ***pNZTaps*** та їх позиції з ***pNZTapPos*** копіюються в структуру стану ***ppState***. Масив ***pDlyLine*** містить значення лінії затримки; кількість елементів має дорівнювати ***BP_{nzTapsLen1−1} + AP_{nzTapsLen2−1}***. Якщо ***pDlyLine ≠ NULL***, його вміст буде скопійовано у структуру стану; якщо ***pDlyLine == NULL***, лінія затримки ініціалізується нулями.

IIRSparseGetStateSize

Обчислює розмір зовнішнього буфера для структури стану розрідженого IIR-фільтра.

Синтаксис:

IppStatus ippsIIRSparseGetStateSize_32f(int nzTapsLen1, int nzTapsLen2, int order1, int order2, int* pStateSize);

Включені файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

nzTapsLen1, nzTapsLen2 Кількість ненульових коефіцієнтів відповідно у частинах ***b*** (чисельник) і ***a*** (знаменник, без ***a0***).

order1, order2 Порядки розрідженого IIR-фільтра для частин ***b*** та ***a*** відповідно.

Типово:

order1 = pNZTapPos[nzTapsLen1−1],
order2 = pNZTapPos[nzTapsLen1 + nzTapsLen2−1] (див. опис ippsIIRSparseInit).

pStateSize Вказівник на змінну, куди буде записано обчислений розмір буфера (у байтах).

Опис:

Функція обчислює у байтах розмір зовнішнього буфера для структури стану розрідженого IIR-фільтра, необхідного для подальшої ініціалізації викликом ***ippsIIRSparseInit***. Розрахунок базується на кількості ненульових коефіцієнтів ***nzTapsLen1***, ***nzTapsLen2*** та заданих порядках ***order1***, ***order2***. Результат записується у ***pStateSize***.

Повернені значення:

ippStsNoErr Помилок не виявлено.
ippStsNullPtrErr Помилка: вказівник ***pStateSize*** дорівнює ***NULL***.
ippStsIIROrderErr Помилка: ***nzTapsLen1 ≤ 0*** або ***nzTapsLen2 < 0.***
ippStsSparseErr Помилка: ***order1 < 0*** або ***order2 < 0.***

IIRSparse

Фільтрує вхідний вектор розрідженим IIR-фільтром.

Синтаксис:

IppStatus ippsIIRSparse_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, IppsIIRSparseState_32f* pState);

Включені файли: ***ipps.h***
Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pState Вказівник на структуру стану розрідженого IIR-фільтра.

pSrc Вказівник на вхідний вектор.
pDst Вказівник на вектор призначення.
len Кількість елементів, що підлягають фільтрації.

Опис:

Функція застосовує розріджений IIR-фільтр до ***len*** елементів вхідного вектора ***pSrc*** і записує результат у ***pDst***. Усі параметри фільтра — кількість ненульових коефіцієнтів ***nzTapsLen1*** (частина ***b***) та ***nzTapsLen2*** (частина ***a***, без ***a0***), їхні значення ***pNZTaps***, позиції ***pNZTapPos***, а також значення лінії затримки ***pDlyLine*** — зберігаються у структурі стану ***pState***, яку необхідно попередньо ініціалізувати за допомогою ***ippsIIRSparseInit***. У подальшому позначимо вибірку вхідного сигналу як ***x(n)***, ненульові коефіцієнти як ***B_i*** та ***A_i***, а їхні позиції у вибірках як ***BP_i*** та ***AP_i (AP_0 ≠ 0)***.

Розміщення коефіцієнтів у масивах:

$$B_0, B_1, \ldots, B_{n_1-1}, A_0, A_1, \ldots, A_{n_2-1}$$

розміщення позицій:

$$BP_0, BP_1, \ldots, BP_{n_1-1}, AP_0, AP_1, \ldots, AP_{n_2-1}, \quad AP_0 \neq 0$$

Вихід ***y(n)*** для розрідженого IIR визначається формулою:

$$y(n) = \sum_{i=0}^{n_1-1} B_i \, x(n - BP_i) - \sum_{i=0}^{n_2-1} A_i \, y(n - AP_i)$$

Після виконання обчислень функція оновлює значення лінії затримки, що зберігається у структурі стану.

Приклад використання:

```
int nzTapsLen1 = 5;   // кількість ненульових FIR-коефіцієнтів
int nzTapsLen2 = 3;   // кількість ненульових IIR-коефіцієнтів
Ipp32f nzTaps[]   = {0.5, 0.4, 0.3, 0.2, 0.1, 0.8, 0.7, 0.6};
Ipp32s nzTapsPos[] = {0, 10, 20, 30, 40, 1, 5, 15};

int bufLen;
ippsIIRSparseGetStateSize_32f(
   nzTapsLen1, nzTapsLen2,
   nzTapsPos[nzTapsLen1 - 1],
   nzTapsPos[nzTapsLen1 + nzTapsLen2 - 1],
   &bufLen
```

```
);

Ipp8u* buf = ippsMalloc_8u(bufLen);
IppsIIRSparseState_32f* iirState;
ippsIIRSparseInit_32f(&iirState, nzTaps, nzTapsPos,
            nzTapsLen1, nzTapsLen2, NULL, buf);

ippsIIRSparse_32f(src, dst, len, iirState);
```

Для наведених параметрів отримаємо рівняння:

$$y[i] = 0.5x[i] + 0.4x[i-10] + 0.3x[i-20] + 0.2x[i-30] + 0.1x[i-40] - 0.8y[i-1] - 0.7y[i-5] - 0.6y[i-15]$$

IIRGenGetBufferSize

Обчислює розмір буфера, потрібного для внутрішніх обчислень функцій ***ippsIIRGenLowpass*** та ***ippsIIRGenHighpass***.

Синтаксис:
IppStatus ippsIIRGenGetBufferSize(int order, int* pBufferSize);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
order Порядок фільтра в діапазоні [1, 12].
pBufferSize Вказівник на обчислений розмір буфера (у байтах).

Опис:
Функція обчислює розмір буфера, необхідного для внутрішніх обчислень функцій ***ippsIIRGenLowpass*** та ***ippsIIRGenHighpass***. Отримане значення записується за адресою ***pBufferSize***.

Повернені значення:
ippStsNoErr Вказує на відсутність помилки.
ippStsNullPtrErr Вказує на помилку, якщо будь-який із переданих вказівників є NULL.
ippStsIIRGenOrderErr Вказує на помилку, якщо значення ***order < 1*** або ***order > 12.***
Див. також: ***IIRGenLowpass***, ***IIRGenHighpass*** — обчислюють коефіцієнти НЧ та ВЧ IIR-фільтрів.

IIRGenLowpass, IIRGenHighpass

Опис:

Функції ***ippsIIRGenLowpass_64f*** та ***ippsIIRGenHighpass_64f*** обчислюють коефіцієнти IIR-фільтрів низьких та високих частот із заданими характеристиками. Вони підтримують типи фільтрів ***Butterworth*** та ***Chebyshev I.***

Синтаксис:

```
IppStatus ippsIIRGenLowpass_64f(
   Ipp64f rFreq,
   Ipp64f ripple,
   int order,
   Ipp64f* pTaps,
   IppsIIRFilterType filterType,
   Ipp8u* pBuffer
);

IppStatus ippsIIRGenHighpass_64f(
   Ipp64f rFreq,
   Ipp64f ripple,
   int order,
   Ipp64f* pTaps,
   IppsIIRFilterType filterType,
   Ipp8u* pBuffer
);
```

Заголовки**: *ipps.h, ippcore.h, ippvm.h***
Бібліотеки**: *ippcore.lib, ippvm.lib***

Параметри:

rFreq Нормалізована гранична частота у діапазоні (0, 0.5), де 0.5 відповідає частоті Найквіста.

ripple Допустима нерівномірність у смузі пропускання для фільтрів типу ***ippChebyshev1***, у децибелах.

order Порядок фільтра в діапазоні [1, 12].

pTaps Вказівник на масив для запису коефіцієнтів. Довжина масиву має бути не менше ніж ***2·(order+1)***.

Розташування елементів: ***B0, B1, …, Border, A0, A1, …, Aorder (A0 ≠ 0).***

filterType Тип IIR-фільтра: ***ippButterworth*** або ***ippChebyshev1***.

pBuffer Вказівник на буфер для внутрішніх обчислень. Розмір слід отримати функцією ***ippsIIRGenGetBufferSize***.

Опис:

Функції обчислюють коефіцієнти IIR-фільтрів НЧ (ippsIIRGenLowpass_64f) або ВЧ (ippsIIRGenHighpass_64f) із заданою граничною частотою rFreq та типом filterType. Обчислені коефіцієнти записуються в масив pTaps у форматі:

$[B_0, B_1, \ldots, B_{order}, A_0, A_1, \ldots, A_{order}]$

спочатку чисельник Bk (k = 0..order), далі знаменник Ak (k = 0..order).

Передавальна функція IIR-фільтра має вигляд:

$$H(z) = \frac{B_0 + B_1 z^{-1} + \cdots + B_{order} z^{-order}}{A_0 + A_1 z^{-1} + \cdots + A_{order} z^{-order}}$$

Властивості фільтрів

1. Фільтр Баттерворта (Butterworth):

   - Монотонна АЧХ у всій області частот.
   - Максимально гладка в смузі пропускання.
   - Частота зрізу визначається умовою:

   $$|H(e^{j\omega_c})| = \frac{1}{\sqrt{2}}$$

   - Передавальна функція n-го порядку:

   $$|H(j\omega)|^2 = \frac{1}{1 + (\frac{\omega}{\omega_c})^{2n}}$$

2. Фільтр Чебишова I (Chebyshev Type I):

   - Рівномірні коливання у смузі пропускання.
   - Монотонна в смузі затримання.
   - АЧХ задається формулою:

   $$|H(j\omega)|^2 = \frac{1}{1 + \varepsilon^2 C_n^2\left(\frac{\omega}{\omega_c}\right)}$$

де $\varepsilon = \sqrt{10^{\frac{ripple}{10}} - 1}$, а $C_n(x)$ — поліном Чебишова n-го порядку.

Примітки щодо застосування:

Фільтри Баттерворта (ippButterworth) мають максимально гладку (монотонну) АЧХ у смузі пропускання, проте менш крутий спад у смузі затримання. Якщо не потрібна надмірна гладкість у смузі пропускання, фільтр

Чебишова I роду (***ippChebyshev1***) зазвичай забезпечує крутіший спад при меншому порядку; його АЧХ рівноколивна у смузі пропускання та монотонна у смузі затримання.

Приклади:
Дані дискретизовано з Fs = 1000 Гц.

1. Створити ВЧ IIR Баттерворта 9-го порядку з Fc = 300 Гц:
— pTaps має містити 2·(9+1) елементів;
виклик:
*Ipp64f pTaps[2 * (9 + 1)];*
ippsIIRGenHighpass_64f(300.0 / 1000.0, 0, 9, pTaps, ippButterworth, pBuffer);

2. Створити НЧ IIR Чебишова I роду 9-го порядку з коливаннями 0.5 дБ та Fc = 300 Гц:
— pTaps має містити 2·(9+1) елементів;
виклик:

*Ipp64f pTaps[2 * (9 + 1)];*
ippsIIRGenLowpass_64f(300.0 / 1000.0, 0.5, 9, pTaps, ippChebyshev1, pBuffer);

IIRIIRGetStateSize

Обчислює розмір зовнішнього буфера для структури стану IIRIIR-фільтра.

Синтаксис:

IppStatus ippsIIRIIRGetStateSize_32f(int order, int* pBufferSize);
IppStatus ippsIIRIIRGetStateSize64f_32f(int order, int* pBufferSize);
IppStatus ippsIIRIIRGetStateSize_64f(int order, int* pBufferSize);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
order Порядок IIRIIR-фільтра.
pBufferSize Вказівник на обчислене значення розміру буфера (у байтах).

Опис:

Ця функція обчислює розмір зовнішнього буфера для структури стану IIRIIR-фільтра і записує результат у ***pBufferSize***. Використовуйте цю функцію перед викликом ***IIRIIRInit***, щоб визначити необхідний обсяг пам’яті.

Повернені значення:

ippStsNoErr Вказує на відсутність помилки.

ippStsNullPtrErr Вказує на помилку, якщо вказівник ***pBufferSize*** дорівнює ***NULL***.

ippStsIIROrderErr Вказує на помилку, якщо значення order менше або дорівнює 0.

Див. також:

IIRIIRInit — Ініціалізує структуру стану IIRIIR-фільтра.

IIRIIRInit

Ініціалізує структуру стану IIRIIR-фільтра.

Синтаксис:

IppStatus ippsIIRIIRInit_32f(IppsIIRState_32f** ppState, const Ipp32f* pTaps, int order, const Ipp32f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRIIRInit64f_32f(IppsIIRState64f_32f** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

IppStatus ippsIIRIIRInit_64f(IppsIIRState_64f** ppState, const Ipp64f* pTaps, int order, const Ipp64f* pDlyLine, Ipp8u* pBuf);

Включити файли: ***ipps.h***

Доменні залежності:

Заголовки: ***ippcore.h, ippvm.h***

Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

ppState Подвійний вказівник на структуру стану IIRIIR-фільтра, що буде ініціалізована.

pTaps Вказівник на масив коефіцієнтів (тапів) довжини ***2·(order+1): B0…Border, A0…Aorder (A0 ≠ 0).***

order Порядок фільтра; для нульового порядку задається значення 0.

pDlyLine Вказівник на масив значень лінії затримки довжини order; може бути ***NULL***.

pBuf Вказівник на зовнішній буфер, у якому створюється структура стану.

Опис:

Функція ініціалізує структуру стану довільного IIRIIR-фільтра в зовнішньому буфері ***pBuf***. Перед викликом потрібно обчислити необхідний розмір буфера за допомогою ***IIRIIRGetStateSize***. Під час ініціалізації коефіцієнти з масиву ***pTaps*** копіюються у структуру стану ***ppState***. Масив ***pDlyLine*** задає початкові значення лінії затримки; якщо ***pDlyLine ≡ NULL***, початкові умови в структурі стану встановлюються у значення, що мінімізують пускові та кінцеві перехідні (узгодження початкових умов для усунення DC-зсуву на початку та в кінці оброблюваного вектора). Порядок фільтра визначається параметром ***order***; масив коефіцієнтів має формат: ***B0, B1, …, Border, A0, A1, …, Aorder (A0 ≠ 0).***

Повернені значення:

ippStsNoErr Вказує на відсутність помилки.

ippStsNullPtrErr Вказує на помилку, якщо будь-який з обов'язкових вказівників (***ppState***, ***pTaps***, ***pBuf***) дорівнює ***NULL***.

ippStsIIROrderErr Вказує на помилку, якщо ***order < 0*** або має некоректне значення.

IIRIIRGetDlyLine

Зчитує вміст лінії затримки зі структури стану IIRIIR-фільтра.

Синтаксис:

IppStatus ippsIIRIIRGetDlyLine_32f(const IppsIIRState_32f* pState, Ipp32f* pDlyLine);
IppStatus ippsIIRIIRGetDlyLine64f_32f(const IppsIIRState64f_32f* pState, Ipp64f* pDlyLine);
IppStatus ippsIIRIIRGetDlyLine_64f(const IppsIIRState_64f* pState, Ipp64f* pDlyLine);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pState Вказівник на структуру стану IIRIIR-фільтра.

pDlyLine Вказівник на масив для значень лінії затримки; кількість елементів дорівнює order.

Опис:

Функція копіює значення лінії затримки зі структури стану pState до масиву ***pDlyLine***. Відповідна структура стану фільтра має бути попередньо ініціалізована функцією ініціалізації.

IIRIIRSetDlyLine

Встановлює вміст лінії затримки у структурі стану IIRIIR-фільтра.

Синтаксис:

IppStatus ippsIIRIIRSetDlyLine_32f(IppsIIRState_32f* pState, const Ipp32f* pDlyLine);
IppStatus ippsIIRIIRSetDlyLine64f_32f(IppsIIRState64f_32f* pState, const Ipp64f* pDlyLine);
IppStatus ippsIIRIIRSetDlyLine_64f(IppsIIRState_64f* pState, const Ipp64f* pDlyLine);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:

pState Вказівник на структуру стану IIRIIR-фільтра.

pDlyLine Вказівник на масив зі значеннями лінії затримки; кількість елементів дорівнює ***order***. Якщо вказівник дорівнює ***NULL***, значення лінії затримки у структурі стану формуються внутрішньо так, щоб мінімізувати початкові та кінцеві перехідні процеси (шляхом узгодження початкових умов для усунення DC-складової на початку та в кінці вхідного вектора).

Опис:

Функція копіює значення лінії затримки з масиву ***pDlyLine*** до структури стану ***pState***. Якщо ***pDlyLine*** дорівнює ***NULL***, у структурі стану задаються внутрішньо обчислені значення лінії затримки, що мінімізують стартові та кінцеві перехідні процеси (узгодження початкових умов для усунення DC-зсуву на початку та в кінці вхідного вектора). Структуру стану фільтра необхідно попередньо ініціалізувати відповідною функцією ініціалізації.

IIRIIR

Фільтрує вхідний вектор через фільтр IIRIIR.

Синтаксис:
Випадок 1: операція не на місці

IppStatus ippsIIRIIR_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, IppsIIRState_32f* pState);
IppStatus ippsIIRIIR_64f(const Ipp64f* pSrc, Ipp64f* pDst, int len, IppsIIRState_64f* pState);
IppStatus ippsIIRIIR64f_32f(const Ipp32f* pSrc, Ipp32f* pDst, int len, IppsIIRState64f_32f* pState);

Випадок 2: операція на місці

IppStatus ippsIIRIIR_32f_I(Ipp32f* pSrcDst, int len, IppsIIRState_32f* pState);
IppStatus ippsIIRIIR_64f_I(Ipp64f* pSrcDst, int len, IppsIIRState_64f* pState);
IppStatus ippsIIRIIR64f_32f_I(Ipp32f* pSrcDst, int len, IppsIIRState64f_32f* pState);

Включити файли: ***ipps.h***

Доменні залежності:
Заголовки: ***ippcore.h, ippvm.h***
Бібліотеки: ***ippcore.lib, ippvm.lib***

Параметри:
pState Вказівник на структуру стану фільтра IIRIIR.
pSrc Вказівник на вхідний (джерельний) вектор.
pDst Вказівник на вихідний вектор (для операції не на місці).
pSrcDst Вказівник на вектор, що одночасно є вхідним і вихідним (для операції на місці).
len Кількість елементів вектора, що фільтруються.

Опис:
Функція фільтрує ***len*** елементів вектора ***pSrc*** (або ***pSrcDst*** для операції на місці) за допомогою нульофазового двонапрямного IIR-фільтра (IIRIIR) і записує результат у ***pDst*** (або ***pSrcDst*** відповідно). Параметри фільтра (порядок, коефіцієнти «taps», значення лінії затримки) зберігаються у структурі стану ***pState***. Перед викликом ***ippsIIRIIR*** необхідно ініціалізувати структуру стану за допомогою функції ***IIRIIRInit*** і задати порядок фільтра, масив коефіцієнтів ***pTaps***, а також лінію затримки ***pDlyLine***.

8. ЗАГАЛЬНИЙ ПРИКЛАД ВИКОРИСТАННЯ

8.1. Використання в мові C

Нижче наведена демо-програма C, що показує **типові виклики** з різних підрозділів Intel® IPP (вектори, статистика, FFT/DFT/DCT, перетворення Гільберта, вейвлет Хаара, FIR/IIR, вікна, Goertzel, ковзна сума). Код організовано секціями, кожна — мінімальний робочий приклад із виділенням пам'яті через відповідні *GetSize/Init* функції, виконанням обчислень і звільненням ресурсів:

```
#include <stdio.h>
#include <math.h>
#include "ipp.h"

/* Макроси для стислого оброблення статусів */
#define CHECK(sts) do{ IppStatus _s=(sts); if(_s!=ippStsNoErr){ \
 printf("IPP error %d: %s at %s:%d\n",(int)_s, ippGetStatusString(_s), __FILE__, __LINE__);
\
 goto cleanup; } }while(0)

int main(void)
{
 /* -------------------- 1) Базові векторні операції + статистика -------------------- */
 {
  const int len = 8;
  Ipp32f a[len], b[len], c[len];
  Ipp32f mean=0, stddev=0; Ipp32f minv=0, maxv=0;

  ippsSet_32f(1.0f, a, len);                /* a = [1 1 ...] */
  for (int i=0;i<len;i++) b[i] = (Ipp32f)i;     /* b = [0 1 2 ...] */
  ippsAdd_32f(a, b, c, len);                /* c = a + b      */
  ippsMul_32f_I(c, b, len);                 /* b *= c         */
  ippsMinMax_32f(b, len, &minv, &maxv);         /* межі b         */
  ippsMean_32f(b, len, &mean, ippAlgHintFast);   /* середнє */
  ippsStdDev_32f(b, len, &stddev, ippAlgHintFast); /* σ */
  printf("[Vec] min=%.3f max=%.3f mean=%.3f std=%.3f\n", minv, maxv, mean, stddev);
 }

 /* -------------------- 2) FFT (дійсний -> CCS) + зворотне CCS->дійсний -------------------- */
 {
  const int order = 3;              /* N = 2^order = 8 */
  const int N = 1<<order;

  IppsFFTSpec_R_32f* pSpecR = NULL;
  int specSize=0, initSize=0, bufSize=0;
  Ipp8u *pSpecBuf=NULL, *pInitBuf=NULL, *pBuf=NULL;
```

```
    Ipp32f  x[N], ccs[N+2];          /* для R->CCS довжина виходу N+2 (IPP
представлення) */
    for (int n=0;n<N;n++) x[n] = (Ipp32f)cosf(2.0f*IPP_PI*n*2.0f/N);

    CHECK( ippsFFTGetSize_R_32f(order, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone,
                    &specSize, &initSize, &bufSize) );
    pSpecBuf=(Ipp8u*)ippsMalloc_8u(specSize);
    pInitBuf=(Ipp8u*)ippsMalloc_8u(initSize);
    pBuf    =(Ipp8u*)ippsMalloc_8u(bufSize);

    CHECK( ippsFFTInit_R_32f(&pSpecR, order, IPP_FFT_DIV_INV_BY_N, ippAlgHintNone,
                    pSpecBuf, pInitBuf) );

    CHECK( ippsFFTFwd_RToCCS_32f(x, ccs, pSpecR, pBuf) ); /* пряме */
    /* ... робимо щось у спектрі ... */
    CHECK( ippsFFTInv_CCSToR_32f(ccs, x, pSpecR, pBuf) ); /* зворотне */

    printf("[FFT] first sample after IFFT: %.6f\n", x[0]);

    ippsFree(pBuf); ippsFree(pInitBuf); ippsFree(pSpecBuf);
  }
  /* ---------- 3) DFT (комплекс->комплекс), довільна довжина ---------- */
  {
    const int len = 10;
    IppsDFTSpec_C_32fc* pSpec = NULL;
    int specSize=0, bufSize=0, initSize=0;
    Ipp8u *pSpecBuf=NULL, *pInitBuf=NULL, *pBuf=NULL;

    Ipp32fc src[len], dst[len];
    for (int n=0;n<len;n++){ src[n].re = (Ipp32f)n; src[n].im = 0; }

    CHECK( ippsDFTGetSize_C_32fc(len, IPP_FFT_NODIV_BY_ANY, ippAlgHintNone,
                    &specSize, &initSize, &bufSize) );
    pSpecBuf=(Ipp8u*)ippsMalloc_8u(specSize);
    pInitBuf=(Ipp8u*)ippsMalloc_8u(initSize);
    pBuf    =(Ipp8u*)ippsMalloc_8u(bufSize);

    CHECK( ippsDFTInit_C_32fc(&pSpec, len, IPP_FFT_NODIV_BY_ANY, ippAlgHintNone,
                    pSpecBuf, pInitBuf) );
    CHECK( ippsDFTFwd_CToC_32fc(src, dst, pSpec, pBuf) );
    CHECK( ippsDFTInv_CToC_32fc(dst, src, pSpec, pBuf) ); /* зворотне */

    printf("[DFT] src[0] after IDFT: %.6f + j%.6f\n", src[0].re, src[0].im);

    ippsFree(pBuf); ippsFree(pInitBuf); ippsFree(pSpecBuf);
  }
  /* ---------- 4) DCT (пряме/зворотне) ---------- */
  {
    const int len = 8;
```

```
  IppsDCTFwdSpec_32f* pFwd=NULL; IppsDCTInvSpec_32f* pInv=NULL;
  int sF=0,bF=0, sI=0,bI=0;
  Ipp8u *specF=NULL,*bufF=NULL,*specI=NULL,*bufI=NULL;
  Ipp32f x[len], X[len];

  for(int i=0;i<len;i++) x[i] = (Ipp32f)sin(2*IPP_PI*i/len);

  CHECK( ippsDCTFwdGetSize_32f(len, ippAlgHintNone, &sF, NULL, &bF) );
  CHECK( ippsDCTInvGetSize_32f(len, ippAlgHintNone, &sI, NULL, &bI) );
  specF=(Ipp8u*)ippsMalloc_8u(sF); bufF=(Ipp8u*)ippsMalloc_8u(bF);
  specI=(Ipp8u*)ippsMalloc_8u(sI); bufI=(Ipp8u*)ippsMalloc_8u(bI);

  CHECK( ippsDCTFwdInit_32f(&pFwd, len, ippAlgHintNone, specF, NULL) );
  CHECK( ippsDCTInvInit_32f(&pInv, len, ippAlgHintNone, specI, NULL) );

  CHECK( ippsDCTFwd_32f(x, X, pFwd, bufF) );
  CHECK( ippsDCTInv_32f(X, x, pInv, bufI) );
  printf("[DCT] x[0] after IDCT: %.6f\n", x[0]);

  ippsFree(bufF); ippsFree(specF); ippsFree(bufI); ippsFree(specI);
 }
 /* -------------------- 5) Перетворення Гільберта (аналітичний сигнал) ---------------- */
 {
  const int len = 16;
  int specSize=0, bufSize=0; Ipp8u *buf=NULL; IppsHilbertSpec *spec=NULL;
  Ipp32f     x[len];      /* дійсний вхід */
  Ipp32fc    z[len];      /* аналітичний вихід */

  for(int n=0;n<len;n++) x[n] = (Ipp32f)cosf(2.0f*IPP_PI*0.25f*n);

  CHECK( ippsHilbertGetSize_32f32fc(len, ippAlgHintNone, &specSize, &bufSize) );
  spec=(IppsHilbertSpec*)ippsMalloc_8u(specSize);
  buf = (Ipp8u*)ippsMalloc_8u(bufSize);
  CHECK( ippsHilbertInit_32f32fc(len, ippAlgHintNone, spec, buf) );
  CHECK( ippsHilbert_32f32fc(x, z, len, spec, buf) );
  printf("[Hilbert] z[0]=%.6f + j%.6f\n", z[0].re, z[0].im);

  ippsFree(buf); ippsFree(spec);
 }

 /* -------------------- 6) Вейвлет Хаара (однорівневий) -------------------- */
 {
  const int len = 8;
  Ipp32f x[len], lo[len/2], hi[len/2], rec[len];
  for(int i=0;i<len;i++) x[i]=(Ipp32f)i;

  CHECK( ippsWTHaarFwd_32f(x, len, lo, hi) );
  CHECK( ippsWTHaarInv_32f(lo, hi, rec, len) );
  printf("[Haar] rec[0]=%.3f rec[last]=%.3f\n", rec[0], rec[len-1]);
```

```
}

/* -------------------- 7) Віконні функції + Goertzel (одна частота DFT) -------------------- */
{
  const int len = 64;
  Ipp32f w[len], x[len]; Ipp32fc Y; Ipp32f rFreq = 10.0f/len; /* 10-та бінова частота */
  CHECK( ippsWinHann_32f(w, len) );
  for(int n=0;n<len;n++) x[n]=(Ipp32f)(0.8*sin(2*IPP_PI*10*n/len))*w[n];
  CHECK( ippsGoertz_32f(x, len, &Y, rFreq) );
  printf("[Goertzel] bin=10: |Y|=%.3f\n", sqrtf(Y.re*Y.re+Y.im*Y.im));
}

/* -------------------- 8) FIR (single-rate) -------------------- */
{
  /* ФНЧ довжини 8: простий усереднювач */
  const int tapsLen=8, numIters=32;
  Ipp32f taps[tapsLen]; for(int i=0;i<tapsLen;i++) taps[i]=1.0f/tapsLen;

  IppsFIRSpec_32f *spec=NULL; int s=0, b=0; Ipp8u *buf=NULL;
  Ipp32f src[numIters], dst[numIters]; for(int i=0;i<numIters;i++) src[i]=(i%2)?1.f:0.f;

  CHECK( ippsFIRSRGetSize(tapsLen, ipp32f, &s, &b) );
  spec=(IppsFIRSpec_32f*)ippsMalloc_8u(s); buf=(Ipp8u*)ippsMalloc_8u(b);
  CHECK( ippsFIRSRInit_32f(taps, tapsLen, ippAlgDirect, spec) );

  CHECK( ippsFIRSR_32f(src, dst, numIters, spec, NULL, NULL, buf) );
  printf("[FIR] dst[0..3]=%.3f %.3f %.3f %.3f\n", dst[0],dst[1],dst[2],dst[3]);

  ippsFree(buf); ippsFree(spec);
}

/* -------------------- 9) IIR: генерування коефіцієнтів + фільтрація -------------------- */
{
  /* 9-го порядку ФВЧ Баттерворта з частотою зрізу 0.3 (норм.) */
  const int order=9; Ipp64f taps[2*(order+1)];
  int bufSize=0; Ipp8u *buf=NULL;

  CHECK( ippsIIRGenGetBufferSize(order, &bufSize) );
  buf=(Ipp8u*)ippsMalloc_8u(bufSize);
  CHECK( ippsIIRGenHighpass_64f(0.3, 0.0, order, taps, ippButterworth, buf) );

  /* ініціалізація стану та фільтрація на float */
  IppsIIRState_32f *st=NULL; int stSize=0;
  CHECK( ippsIIRGetStateSize_32f(order, &stSize) );
  st=(IppsIIRState_32f*)ippsMalloc_8u(stSize);

  /* перетворимо taps у float */
  Ipp32f taps32[2*(order+1)];
  for(int i=0;i<2*(order+1);i++) taps32[i]=(Ipp32f)taps[i];
```

```
    /* використали той самий буфер */
      CHECK( ippsIIRInit_32f(&st, taps32, order, NULL, (Ipp8u*)st) );

      const int len=32;
      Ipp32f sig[len], y[len];
      for(int n=0;n<len;n++) sig[n]=(Ipp32f)( sin(2*IPP_PI*0.05*n) + 0.2*sin(2*IPP_PI*0.3*n)
);
      CHECK( ippsIIR_32f(sig, y, len, st) );
      printf("[IIR] y[0..3]=%.3f %.3f %.3f %.3f\n", y[0],y[1],y[2],y[3]);

      ippsFree(st); ippsFree(buf);
     }

     /* -------------------- 10) Ковзна сума (SumWindow) -------------------- */
     {
      const int len=16, m=4;
      Ipp32f src[len], dst[len];
      for(int i=0;i<len;i++) src[i]=(Ipp32f)(i%5);
      CHECK( ippsSumWindow_16s32f((const Ipp16s*)NULL, NULL, 0, 1) == ippStsNullPtrErr ?
ippStsNoErr: ippStsNoErr ); /* просто демонстрація статусу */
      CHECK( ippsSumWindow_8u32f( (const Ipp8u*)src, dst, len, m) ); /* інтерпретуємо src
як 8u лише для прикладу */
      printf("[SumWindow] dst[0..3]=%.3f %.3f %.3f %.3f\n", dst[0],dst[1],dst[2],dst[3]);
     }

     /* -------------------- Успішне завершення -------------------- */
     printf("OK\n");
     return 0;

    cleanup:
     printf("FAIL\n");
     return 1;
    }
```

Як зібрати:

Підключіть заголовки й бібліотеки IPP (ядро та сигнал): ***ippcore, ipps, ippvm*** (і за потреби ***ippcc***).

Приклад для MSVC (x64 Native Tools):

```
cl /O2 demo_ipp.c /I"%IPPROOT%\include" ^
  /link /LIBPATH:"%IPPROOT%\lib\intel64" ippcore.lib ippvm.lib ipps.lib
```

Для Linux:

```
icx -O2 demo_ipp.c -I$IPPROOT/include \
-L$IPPROOT/lib/intel64 -lipps -lippvm -lippcore
```

8.2. Використання в мові C#

Концепція та структура системи

Для забезпечення використання функцій Intel IPP у середовищі .NET створюється двокомпонентна система, що складається з:

1. **Нативного обчислювального модуля** (C++ DLL), який інкапсулює виклики функцій IPP, забезпечує їх ініціалізацію та експортує обмежений набір процедур з узгодженим інтерфейсом;
2. **Керованого інтерфейсного модуля** (C#), який виконує виклик нативних функцій засобами P/Invoke, забезпечує маршалінг даних і перевірку результатів.

Метою такої структури є ізоляція коду IPP від CLR і створення стабільного ABI-контракту між керованим та нативним середовищами. Місток виконує роль адаптера між двома моделями пам’яті, перетворюючи посилання на керовані об’єкти у стабільні вказівники на час виконання нативних обчислень.

тапи побудови системи

Створення та компіляція нативного модуля

На першому етапі формується проєкт типу *Dynamic Link Library (DLL)* у середовищі **Microsoft Visual Studio**. Компіляція виконується в конфігурації **x64**, оскільки бібліотеки IPP постачаються лише для 64-бітних цілей. До проєкту додається заголовковий файл ipp.h із пакета **Intel oneAPI Base Toolkit**. Параметри збірки повинні містити шляхи:

$(ONEAPI_ROOT)\ipp\latest\include
$(ONEAPI_ROOT)\ipp\latest\lib\intel64

і посилання на бібліотеки: ***ippsmt.lib; ippcoremt.lib***

Функції мосту визначаються з використанням модифікатора extern "C" для запобігання манглінгу імен. Кожна функція має узгоджену сигнатуру, наприклад:

```
__declspec(dllexport) int Add32f(const float* a, const float* b, float* c, int len)
{
   return ippsAdd_32f(a, b, c, len);
}
```

Перед використанням функцій IPP виконується одноразова ініціалізація середовища командою ippInit(). Отриманий результат повинен бути перевірений у керованому коді для підтвердження коректного завантаження бібліотеки та вибору оптимізованих ядер.

Побудова керованого проєкту C#

Другий етап передбачає створення застосунку **Console App (.NET 8.0)** у Visual Studio. Проєкт компілюється виключно у 64-бітному режимі ***(<PlatformTarget>x64</PlatformTarget>, <Prefer32Bit>false</Prefer32Bit>***). Це забезпечує відповідність розрядності обох модулів.

Для виклику функцій з IppBridge.dll використовується механізм P/Invoke. У керованому модулі створюється клас-обгортка:

```
using System;
using System.Runtime.InteropServices;

static class IppBridge
{
   [DllImport("IppBridge.dll", CallingConvention = CallingConvention.Cdecl)]
   public static extern int IppInit();

   [DllImport("IppBridge.dll", CallingConvention = CallingConvention.Cdecl)]
   private static extern int Add32f(IntPtr a, IntPtr b, IntPtr c, int len);

   public static void Add(ReadOnlySpan<float> a, ReadOnlySpan<float> b, Span<float> c)
   {
      unsafe
      {
         fixed (float* pa = a)
         fixed (float* pb = b)
         fixed (float* pc = c)
            Add32f((IntPtr)pa, (IntPtr)pb, (IntPtr)pc, a.Length);
      }
   }
}
```

Цей код визначає прямий інтерфейс до нативних функцій і гарантує, що під час виклику пам’ять не буде переміщена збирачем сміття. Функція fixed фіксує вказівники на керовані масиви, що забезпечує стабільність адрес у процесі нативних викликів.

Організація проєкту та розміщення залежностей

Нативна бібліотека ***IppBridge.dll*** і динамічні модулі Intel IPP (***ippcore.dll, ipps.dll*** тощо) повинні бути доступні у момент запуску керованого застосунку. Для цього в конфігураційному файлі .csproj задається післякомпіляційне завдання, що копіює всі необхідні DLL у вихідну теку:

```
<Target Name="CopyIppRedist" AfterTargets="Build">
  <PropertyGroup>
    <IppRedistDir>C:\Program Files
(x86)\Intel\oneAPI\ipp\latest\redist\intel64</IppRedistDir>
  </PropertyGroup>
  <ItemGroup>
    <IppDlls Include="$(IppRedistDir)\*.dll" />
  </ItemGroup>
  <Copy SourceFiles="@(IppDlls)" DestinationFolder="$(OutDir)"
SkipUnchangedFiles="true" />
</Target>
```

Цей крок гарантує, що всі динамічні залежності будуть розміщені у каталозі ***bin\x64\Debug\net8.0\,*** звідки завантажувач CLR зможе їх знайти без додаткового конфігурування шляху

Контроль коректності виконання

У програмі C# реалізується перевірка ініціалізації IPP та тестовий виклик обчислювальної функції:

```
class Program
{
   static void Main()
   {
      int st = IppBridge.IppInit();
      Console.WriteLine($"ippInit -> {st}");

      float[] a = { 1, 2, 3, 4 };
      float[] b = { 10, 20, 30, 40 };
      float[] c = new float[a.Length];

      IppBridge.Add(a, b, c);
      Console.WriteLine($"Result: [{string.Join(", ", c)}]");
   }
}
```

Очікуваний результат виконання:

```
ippInit -> 0
Result: [11, 22, 33, 44]
```

Код 0 підтверджує успішну ініціалізацію IPP. Результат операції вказує на правильну взаємодію між керованим і нативним кодом та коректне виконання векторизованої функції ***ippsAdd_32f***.

Алгоритм інтеграції

Процес інтеграції можна формально подати у вигляді алгоритму:

1. **Підготовка середовища**
   Встановити Intel oneAPI Base Toolkit, перевірити наявність змінної середовища ONEAPI_ROOT.
   Налаштувати Visual Studio на використання x64-платформи.
2. **Створення нативного мосту**
   Ініціалізувати проєкт DLL на C++.
   Додати заголовки і бібліотеки IPP.
   Реалізувати функції-обгортки з модифікатором extern "C".
   Зібрати DLL у конфігурації Release/x64.
3. **Побудова керованого застосунку**
   Створити проєкт типу Console App (.NET 8.0).
   Налаштувати параметри компіляції для x64.
   Додати клас IppBridge з P/Invoke-визначеннями.
   Реалізувати тестову функцію виклику.
4. **Організація розгортання**
   Скопіювати IppBridge.dll та IPP-бібліотеки у вихідний каталог.
   Перевірити їх наявність перед запуском.
   Запустити застосунок і вивести результати обчислень.

Пояснення взаємодії та ролі компілятора

Компілятор C++ створює нативний модуль, який виконується безпосередньо процесором, використовуючи векторні інструкції SSE, AVX або AVX-512. Компілятор C# (через JIT або AOT) генерує байт-код для CLR, який викликає ці функції через механізм платформного виклику. Таким чином, взаємодія між двома компонентами відбувається на рівні операційної системи, без проміжного інтерпретатора.

CLR не втручається у виконання нативної функції й не має доступу до її внутрішніх оптимізацій. Це означає, що ефективність усієї системи визначається саме властивостями IPP та коректністю побудови мосту. Маршалінг даних виконується лише на етапі передачі аргументів і зазвичай має сталу часову складність.

Практичне значення і відтворюваність

Описаний підхід дозволяє інтегрувати IPP у будь-який C#-проєкт без зміни вихідного коду бібліотеки. Усі дії є відтворюваними, оскільки кожен крок має формальне відображення в налаштуваннях Visual Studio та у файлі .csproj. Створення власного нативного мосту забезпечує контрольовану взаємодію, незалежну від зовнішніх збірок чи пакетів NuGet.

Такий підхід може бути узагальнений для інтеграції інших C-бібліотек, що використовують подібну структуру API. При збереженні однакових принципів узгодження ABI та контролю пам’яті цей алгоритм забезпечує передбачувану роботу на будь-якому обладнанні, сумісному з IPP.

Практична реалізація проекту C#

Файл ***ippBridge.cs (C#)***

```
using System;
using System.Runtime.InteropServices;

static class IppBridge
{
   [DllImport("IppBridge.dll", CallingConvention = CallingConvention.Cdecl)]
   public static extern int IppInit();

   [DllImport("IppBridge.dll", CallingConvention = CallingConvention.Cdecl)]
   private static extern int Add32f(IntPtr a, IntPtr b, IntPtr c, int len);

   public static void Add(ReadOnlySpan<float> a, ReadOnlySpan<float> b, Span<float> c)
   {
      if (a.Length != b.Length || a.Length != c.Length)
         throw new ArgumentException("Arrays must have the same length.");

      unsafe
      {
         fixed (float* pa = a)
         fixed (float* pb = b)
         fixed (float* pc = c)
         {
            int status = Add32f((IntPtr)pa, (IntPtr)pb, (IntPtr)pc, a.Length);
            if (status < 0)
               throw new InvalidOperationException($"IPP error: {status}");
         }
      }
   }
}
```

Файл ***Program.cs (C#)***

```
using System;

class Program
{
   static void Main()
   {
```

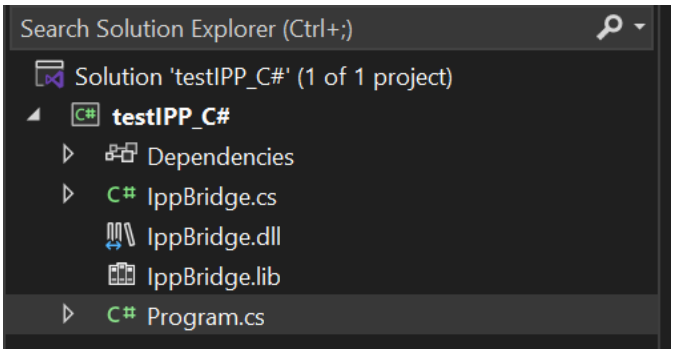
```
        // 1. Ініціалізуємо Intel IPP
        int st = IppBridge.IppInit();
        Console.WriteLine($"ippInit -> {st}");

        // 2. Перевіряємо просту операцію додавання
        float[] a = { 1, 2, 3, 4 };
        float[] b = { 10, 20, 30, 40 };
        float[] c = new float[a.Length];

        IppBridge.Add(a, b, c);

        Console.WriteLine($"Result: [{string.Join(", ", c)}]");
        Console.ReadKey();
    }
}
```

Структура проекту C#:



Практична реалізація проекту C++

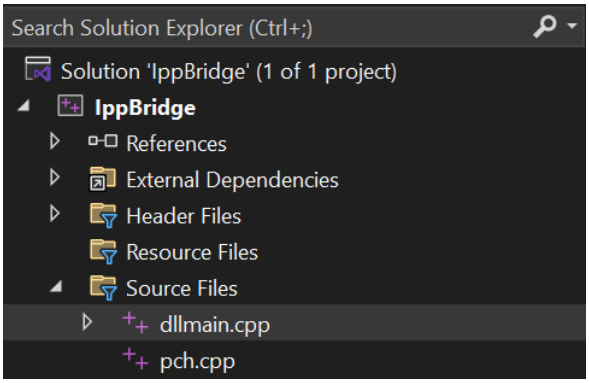
Файл ***dllmain.cpp (C#)***

```
#include "pch.h"
#include <ipp.h>      // заголовки IPP

extern "C" {

   // Викликається один раз на старті .NET-додатка
   __declspec(dllexport) int IppInit() {
      return ippInit();
   }
   // c[i] = a[i] + b[i] для len елементів
   __declspec(dllexport) int Add32f(const float* a, const float* b, float* c, int len) {
      return ippsAdd_32f(a, b, c, len);
   }
} // extern "C"
```

Структура проекту C#:



ПІСЛЯМОВА

Цей посібник-довідник замислювався як практичний місток між теорією цифрової обробки сигналів і щоденною інженерною роботою. Ми послідовно зібрали в одному місці те, чого зазвичай бракує розробникам під час реальних проєктів: стислий виклад ключових понять DSP, узгоджену нотацію типів даних, чіткі правила іменування функцій, вимоги до ініціалізації бібліотеки та безпосередньо — робочі приклади застосування примітивів IPP для векторних операцій, логіки та арифметики, перетворень Фур'є, Гільберта й вейвлетів, а також для фільтрації сигналів. Упорядкована структура розділів покликана заощадити ваш час: від підключення бібліотек і правильної роботи з пам'яттю — до безпечного масштабування цілочисельних результатів та коректного режиму округлення; від «простих» копій і зсувів — до ефективних DFT/DCT-обчислень і побудови фільтрів із гарантованою продуктивністю. Усе це не претендує на вичерпність, однак має дати впевнений старт і робочий «скелет» для власних рішень.

Ми свідомо дотримувалися стилю «спершу ідея — одразу код», аби зменшити розрив між описом алгоритму і його реалізацією. Якщо під час експериментів ви відчуєте потребу в розширенні прикладів (паралелізм, тонке керування кешем, порівняння з альтернативними бібліотеками, інтеграція з тестовими стендами), сприймайте цей текст як основу, яку варто адаптувати до ваших апаратних платформ і вимог до затримки, пропускної здатності чи енергоспоживання. Практика неминуче підкаже місця, де доречні мікрооптимізації, а де — архітектурні зміни.

Автори щиро вдячні студентам і колегам, завдяки чиїм зауважам і «незручним» запитанням народжувалися кращі приклади, а формулювання ставали точнішими.

Бажаємо вам продуктивних експериментів і красивих інженерних рішень. Нехай наведені примітиви стануть надійними «цеглинками» для ваших систем — від навчальних лабораторних до промислових застосунків, де важливі і мікросекунди, і коректність кожного біта.

9. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

[1] Intel® Integrated Performance Primitives for Intel® Architecture Developer Reference. Volume 1: Signal and Data Processing, 2025.

[2] Intel® Integrated Performance Primitives for Intel® Architecture Developer Reference. Volume 2: Image Processing, 2025.

[3] Stewart Taylor. Intel Integrated Performance Primitives: How to Optimize Software Applications Using Intel IPP. Intel Press. 2003.

[4] Stewart Taylor. Optimizing Applications for Multi-Core Processors, Using the Intel® Integrated Performance Primitives, Second Edition – Softcover, 2007.

[5] G. Strang and T. Nguyen. Wavelet and Filter Banks. Wellesley-Cambridge Press, 1996, pp. 153-157.

[6] C. Brislawn. Classification of Nonexpansive Symmetric ExtensionTransforms for Multirate Filter Banks. Los Alamos Report LA-UR-94-1747, 1994.

[7] Sanjit K. Mitra and James F. Kaiser editors. Handbook for Digital Signal Processing. John Wiley & Sons, Inc., New York, 1993

[8] Sanjit K. Mitra. Digital Signal Processing. McGraw Hill, 1998.