А.М. Сергієнко, Ю.М.Виноградов, Т.М.Лесик

Цифрова обробка сигналів

Комп'ютерний практикум мовою VHDL

Київ –2012

ББК 32.811
УДК 621.391

Рецензенти: О.М.Романкевич, доктор технічних наук, професор, кафедра спеціалізованих компьютерних систем НТУУ "КПІ"; В.П.Широчин, доктор технічних наук, професор, кафедра обчислювальної техніки НТУУ "КПІ".

А. М. Сергієнко, Ю. М. Виноградов, Т. М. Лесик Цифрова обробка сигналів. Комп'ютерний практикум мовою VHDL. К.: НТУУ«КПІ» , 2012. –104 с.

Комп'ютерний практикум охоплює побудову моделей генераторів сигналів, дослідження властивостей Z-перетворення, цифрових фільтрів та швидкого перетворення Фур'є з використанням VHDL-симулятора. Приведені основи мови VHDL. Для студентів, аспірантів, викладачів вузів, а також спеціалістів у галузі електроніки, вимірювальної і обчислювальної техніки та зв□язку.

ББК 32.811

A. M. Sergiyenko, Ju. M. Vinogradov, T. M. Lesyk Digital signal processing. Computer tutorial using VHDL. K.: НТУУ«КПІ» , 2012. − 104 p.

Computer tutorial includes signal generator modeling, Z-transform investigation, digital filter, fast Fourier transform modeling at the base of the VHDL simulator. VHDL basics are considered as well. For students, aspirants, lecturers, and specialists in computer science, electronics and measurements.

ББК 32.811

А. М. Сергиєнко, Ю. М. Виноградов, Т. М. Лесик Цифровая обработка сигналов. Компьютерный практикум на языке VHDL. К.: НТУУ«КПИ» , 2012. –104 с.

Компьютерный практикум охватывает построение моделей генераторов сигналов, исследование свойств Z-преобразования, цифровых фильтров и быстрого преобразования Фурьє с использованием VHDL-симулятора. Приведены основы языка VHDL. Для студентов, аспирантов, преподавателей вузов, а также специалистов в области электроники, измерительной и вычислительной техники и связи.

ББК 32.811

А.М. Сергієнко, Ю.М.Виноградов, Т.М.Лесик

Цифрова обробка сигналів

Комп'ютерний практикум мовою VHDL

ЗМІСТ

Передмова

Зараз важко назвати галузь науки і техніки, де не використовується цифрова обробка сигналів (ЦОС). Найбільш активно ЦОС застосовується у транспорті, радіотехніці, зв'язку, телебаченні, телекомунікаціях, вимірювальній та побутовій техніці. Це передбачає масовий випуск та широку номенклатуру засобів ЦОС та комп'ютерних застосунків. Задачі ЦОС відрізняються високою продуктивністю засобів, що їх вирішують, а також рядом особливостей алгоритмів ЦОС, таких, як високий рівень паралелізму, потокова обробка даних, особливий математичний апарат, широке розповсюдження. ЦОС у комп'ютерах реалізована програмно у переважній більшості випадків.

Алгоритми ЦОС мають періодичну природу і полягають в обробці майже безперервних потоків даних. Найбільш природно такі алгоритми задавати на потоковій моделі. На цій моделі основана універсальна високорівнева мова VHDL. Досі ця мова використовувалась лише для моделювання та опису дискретних систем. Але вона має ряд неоціненних переваг у порівнянні з традиційними мовами, такими як Matlab, C, Pascal, якими часто описують алгоритми ЦОС. Опис обробки потоків даних мовою VHDL є натуральним і зрозумілим. Від програми на VHDL нескладно перейти як до програми для мікропроцесора, так і до опису схеми спеціалізованого процесора, який апаратно реалізує алгоритм.

У книзі пропонується цикл лабораторних робіт з вивчення алгоритмів ЦОС та їх реалізації в обчислювачах з потоковою структурою. Основу лабораторних робіт становить опис алгоритмів мовою VHDL з їх поведінковим моделюванням у симуляторі. Студент, що виконує курс лабораторних робіт, вивчає не тільки основні алгоритми ЦОС, але й сучасну технологію проектування їх реалізації в ПЛІС.

У першому розділі викладені основи мови VHDL. У другому розділі викладені основи програмування алгоритмів цифрової обробки сигналів мовою VHDL У третьому розділі наведені відомості про САПР Active HDL. У четвертому розділі описані лабораторні роботи. Кожна наступна лабораторна робота відрізняється наростаючою складністю і використовує результати попередніх лабораторних робіт.

1. Основи мови VHDL

1.1 Загальна характеристика

Обчислення, які запрограмовані мовою VHDL, ґрунтуються на множині ***процесів***, що виконуються паралельно й обмінюються між собою ***сигналами***. Тому VHDL − це мова паралельного програмування.

Відмінність VHDL від інших мов паралельного програмування полягає в тому, що типи й об'єкти мови мають пряму аналогію з фізичними сутностями та елементами цифрової електроніки. Наприклад, у класі ***фізичних типів***, основним типом є час − time. На відміну від числових типів, фізичний тип має одиниці виміру. Так, тип time веде відлік в секундах і частках секунди, тобто в наносекундах, мікросекундах і т.д.

Проектування дискретних схем мовою VHDL ґрунтується на тому, що під сигналами типу bit або bit_vector маються на увазі реальні одно- або багаторозрядні лінії зв'язку, а під операторами процесу або просто процесами − елементи та вузли комп'ютерів, функціонування яких у часі збігається з поводженням цих процесів.

За визначенням, потоки даних або сигнали є ***синхронними***, якщо між даними в них є стала взаємна відповідність, тобто вони можуть бути перенумерованими натуральними числами. Наприклад, якщо в одному потоці дані $y(i-1)$, а в іншому $-x(i-2)$, то ці потоки є синхронними. Відповідно, граф, у якого всі потоки в дугах є синхронними, є ***графом синхронних потоків даних***. Алгоритми ЦОС, які описані мовою VHDL, виконуються саме моделлю графа синхронних потоків даних.

Кожна вершина такого графа представляє собою процес. Дуга графа, яка з'єднує вихід однієї вершини зі входом іншої, представляє собою потік даних, тобто сигнал. Процес запускається кожного разу, якщо відбувається подія зміни будь-якого його вхідного сигналу. В момент зупинки процес викликає зміну свого вихідного сигналу, призначаючи йому нове значення. Це значення є певною функцією від значень вхідних сигналів та внутрішнього стану процесу. Причому запуски процесів у вершинах мають сталий період, через що потоки даних в дугах є синхронними.

На рис.1.1 показаний приклад графа синхронних потоків даних, що описує обчислення за різницевим рівнянням фільтру:

$$y(i) = a \cdot y(i-1) + x(i-2) .$$

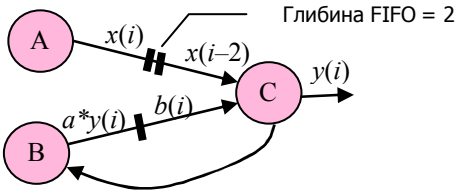


Рис. 1.1. Приклад графа синхронних потоків даних

У ньому вершина A видає сигнал *x*, вершина B виконує дію: $b = a \cdot y$, а вершина C – дію: $y = b + x$. Для виконання затримок на 1 та 2 цикли, дуги мають буферні затримки типу FIFO на 1 та 2 даних, відповідно. Вершина C спрацьовує негайно, як тільки є готові сигнали — дані *b* та *x* на її входах і видає на свій вихід результат *y*. Ця вершина описується наступним оператором процесу:

```
Process begin
   y <= b + x;
   wait for b, x;
end process;
```

Тут оператор процесу обчислює перший послідовний оператор присвоювання сигналу y і зупиняється на другому послідовному операторі **wait**. Як тільки змінюється сигнал b або x, виконання оператора процесу відновлюється з першого оператора. Аналогічно описуються решта вершин, причому буфери FIFO входять у склад відповідних вершин та їх синхронізація виконується за допомогою загального синхросигналу.

В програмах на VHDL ***сигнал*** служить як носієм інформації між процесами, так і синхронізуючим впливом, що запускає виконання процесів. Крім того, сигнал має пряму аналогію з реальними сигналами, які моделюються. При виконанні VHDL— програми, як і при обробці реальних сигналів, моделювання може тривати невизначено довгий термін.

При програмуванні на VHDL, як правило, не використовуються ресурси, що поділяються між декількома процесами, як, наприклад, загальні змінні. Завдяки цьому, зникає проблема синхронізації доступу до ресурсів і як результат, VHDL-програми виконуються значно швидше, ніж інші паралельні програми, а число процесів може досягати сотень тисяч.

Отже, серед поширених мов моделювання обробки сигналів, таких як C, Matlab та інші, мова VHDL виявляється найбільш адаптованою до цієї предметної галузі.

1.2 Основні конструкції мови

Основною програмною одиницею мови VHDL є ***об'єкт проекту***. Об'єкт - це складова частина проекту, яка представляє собою деяку закінчену дискретну систему або її компонент. Об'єкт може бути використаний в іншому проекті як компонент або може бути об'єктом більш високого рівня ієрархії у даному проекті.

Об'єкт проекту описується набором складових частин: оголошення об'єкту, тіло архітектури об'єкту (або просто архітектура), оголошення пакета, тіло пакета й оголошення конфігурації. Складові частини проекту зберігаються в одному або декількох файлах з розширенням VHD. Як правило, в одному файлі описується лише один об'єкт, який до того ж має таку саму назву, як і файл. Об'єкт проекту зазвичай описується відповідно до синтаксису:

\об'єкт проекту\::= [\опис **library**\]
[\опис **use**\]
\оголошення об'єкта\
\тіло архітектури\
[\оголошення конфігурації\]

де ідентифікатори \опис **library**\ – назви бібліотек, які використовуються в об'єкті проекту й указують транслятору місце розташування цих бібліотек. Опис **use** указує, які пакети і які елементи цих пакетів можуть бути використані в об'єкті. Тут і далі жирним шрифтом виділені ключові слова мови. Слово у зворотні похилих означає синтаксичну одиницю мови і представляє собою розширений ідентифікатор VHDL, що може містити будь-які символи. Фраза, яка вкладена у квадратні дужки, може бути опущена, а у фігурних дужках – може повторюватися кілька разів.

Оголошення об'єкта

Оголошення об'єкта вказує, як об'єкт проекту виглядає зовні і яким чином його можна включити в іншому об'єкті проекту як компонент, тобто він описує зовнішній інтерфейс об'єкта. Синтаксис оголошення об'єкта:

\оголошення об'єкта\::= **entity** \ідентифікатор\ **is**
[**generic**(\оголошення настроювальної константи\
{; \оголошення настроювальної константи\});]
[**port** (\оголошення порту\ {;\оголошення порту\});]
end \ідентифікатор\;

Тут \ідентифікатор\ - ім'я об'єкту. Оголошення портів, позначене ключовим словом **port**, – це набір інтерфейсних сигналів об'єкта проекту. Настроювальні константи **generic** кодують певні власти-

вості об'єкта проекту, наприклад, розрядність ліній зв'язку, параметри затримки, кодування структури пристрою, що моделюється. Слід врахувати, що в VHDL ідентифікатори з однаковим написанням представляють той самий об'єкт, тобто вони «нечутливі» до висоти букв. На рис. 1.2 показане зображення об'єкта фільтра нижніх частот, що відповідає наступному оголошенню об'єкта

```
entity LPF1 is
generic(bypass: natural:=0);-- при 1 - вимкнути
port (C:in bit;              -- синхросигнал
    R:in bit;                --встановлення в 0
    DI:in real ;             --вхідне дане
    DO:out real);            --вихідне дане
end LPF1;
```

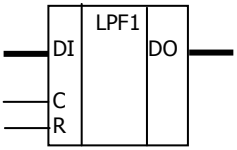


Рис.1.2. Об'єкт фільтра нижніх частот

Входи синхросигналу C, встановлення в 0 R мають бітовий тип, а вхід DI і вихід DO – тип числа з плаваючою комою. Ключові слова **in**, **out** позначають напрямок передачі інформації, відповідно, в об'єкт та з об'єкту. Напрямок **inout** в оголошенні порту означає двонаправлену шину. Настроювальна константа bypass типу natural задає режим роботи фільтра. Текст після подвійного дефісу до кінця рядка представляє собою коментар.

Архітектура об'єкта

Архітектура об'єкта – це частина об'єкта, у якій описано, яким чином реалізований об'єкт. Її синтаксис:

```
\тіло архітектури\::= architecture \ідентифікатор\ of \ім'я об'єкта\ is
                        {\оголошення в архітектурі\}
                    begin
                      { \паралельний оператор\}
                    end [architecture][\ідентифікатор\];
```

Ідентифікатором позначається ім'я тіла архітектури, а ім'я об'єкта вказує, який саме об'єкт описаний у цьому тілі архітектури. Одному об'єкту проекту може відповідати кілька архітектур, у кожній з яких описаний один з варіантів реалізації об'єкта. Оголошені в ***декларативній частині*** архітектури типи, сигнали, підпрограми видимі тільки в межах цієї архітектури. ***Виконавчу частину*** архітектури становлять паралельні оператори, такі як процес, паралельне присвоювання сигналу, вставка компонента та ін. Через те, що всі оператори у виконавчій частині тіла архітектури паралельні, їхній взаємний порядок – довільний.

Одна з архітектур для **entity** LPF1 виглядає так:

```
architecture SIMPLE of LPF1 is
   signal d1,d2,d3:real; -- декларативна частина
begin
 -- Виконавча частина -----
   process(C,R) begin      -- оператор процесу, який моделює
      if R='1' then            -- ланцюжок регістрів  d1,d2,d3
            d1<=0.0;
            d2<=0.0;       -- початкове встановлення
            d3<=0.0;
      elsif C='1' and C'event then
            d1<=DI;
            d2<=d1;        -- запис у регістри
            d3<=d2;
      end if;
   end process;
 DO<= d2 when bypass = 1    -- оператор умовного присвоювання
        else  0.25*d1 + 0.5*d2 + 0.25*d3;   -- власне обчислення
end SIMPLE;
```

1.3 Типи даних, об'єкти та вирази мови

Те, що VHDL — це строго типізована мова, означає, що кожен об'єкт мови має тип, що вказує, яке значення може бути присвоєне об'єкту і які операції можуть із ним виконуватися. Під час компіляції перевіряються відповідності між типом об'єкта та типами аргументів операцій і виразів. При виконанні програм перевіряється відповідність між типами сигналів та змінних в присвоюваннях. У пакеті STANDARD задані стандартні типи: bit, bit_vector, integer, real, boolean, character, string, time. Інші типи визначені в стандартних бібліотеках, таких, як IEEE, або задаються користувачем.

Типи даних

В VHDL використовуються чотири класи типів даних: скалярні, композитні типи, типи доступу за посиланням та файлові типи. ***Скалярні типи*** мають прості й одиночні значення, що набувають значення із упорядкованої множини значень. З наступними об'єктами скалярного типу можна виконувати операції порівняння.

Перелічний тип визначається як список (перелік) всіх можливих значень даного типу. До нього відноситься ***символьний тип*** – character , який представлено таблицею 256 символів ASCII.

Наприклад, оголошення перелічного типу STD_ULOGIC з пакета STD_LOGIC_1164 бібліотеки IEEE виглядає так:

```
type STD_ULOGIC is ( 'U',  -- Неініціалізовано
                     'X',  -- Сильне невідоме
                     '0',  -- Сильний  0
                     '1',  -- Сильна  1
                     'Z',  -- Високий імпеданс
                     'W',  -- Слабке невідоме
                     'L',  -- Слабкий    0
                     'H',  -- Слабка    1
                     '-'   -- Байдуже          );
```

Булевський тип boolean представляє множину (true, false) і найчастіше використовується в логічних операціях та операціях порівняння. На противагу йому ***бітовий тип*** bit застосовується безпосередньо для моделювання логічних схем.

Цілочисельний тип визначає множину значень цілих чисел у заданому діапазоні:

```
type ADDRESS is range 0 to 65535;
```

Тип integer являє собою наперед визначений цілочисельний тип у діапазоні від $-2^{31}+1$ до $2^{31}-1$.

Тип з плаваючою комою — real — це машинне наближення дійсних чисел. Він визначається аналогічно до цілого типу, але з діапазоном чисел з плаваючою комою.

Фізичний тип - це числа, що виражають фізичні величини, такі як час, довжина, напруга і т.п. Найчастіше використовується визначений тип time, що задає затримки сигналів, відлік часу моделювання. Значення цього типу відраховують як цілі числа у фемтасекундах, fs, пікосекундах, ps, наносекундах, ns, мікросекундах, us і т.д.

Композитні типи використовуються для визначення наборів значень. У мові існують регулярний тип і комбінований тип.

Регулярний тип являє собою множину елементів однакового типу. Розрізняють необмежені та обмежені регулярні типи. Необмежений тип масиву чисел із плаваючою комою:

```
type REAL_ARR is array (integer range<>) of real;
```

Вираз integer **range**<> означає, що в масиві діапазон індексів виражається цілими числами, але тут він не визначений. Цей діапазон має бути визначений потім, при оголошенні об'єктів або під час компіляції та зв'язування об'єктів, наприклад, при виклику функції.

Приклади оголошення обмеженого регулярного типу:

```
type MY_STRING is array (0 to 79) of character;
type ARR_4K is array (0 to 4095) of bit_vector(7 downto 0);
type ARR_4Kx8 is array (0 to 4095, 7 downto 0) of bit;
```

У першому випадку заданий тип рядка довжини 80, а в другому — тип масиву байтів обсягом 4К, що застосовується, наприклад, для побудови чотирикілобайтного ОЗП. У третьому прикладі оголошений тип масиву також розміром 4 кілобайти, але тут масив двовимірний і складається з бітів.

Строковий тип string є також регулярним типом, який визначається як одновимірний масив символів.

Комбінований тип визначає множину значень як і регулярний тип, але ці значення можуть бути різнотипними. Наприклад, тип комплексного числа:

```
type COMPLEX is record
        RE:real;
        IM:real;
end record;
```

Підтип

Підтипом називається тип з додатковими обмеженнями. Підтип використовують для ототожнювання групи об'єктів базового або родоначального типу. У прикладах оголошення підтипів:

```
subtype Z01_U is STD_ULOGIC range '0' to 'Z';
subtype BYTE_I is integer range -128 to 127;
```

у першому випадку задається підтип, що складається з елементів '0', '1' та 'Z', а в другому випадку — підтип цілих чисел, які представляють байти.

Обмеження підтипу дає змогу виявити помилки на етапі моделювання, коли помилковий вихід за межі підтипу фіксується симулятором. Об'єкти різних підтипів, що мають один родоначальний тип, можуть брати участь у обчисленнях без конфліктів типів.

Підтипи natural та positive — це наперед визначені підтипи, що означають цілі позитивні числа й цілі числа більше нуля, відповідно.

Об'єкти мови

Об'єкт мови - це програмна одиниця з ім'ям, якій можна присвоювати значення певного типу або підтипу. Об'єкти оголошуються в декларативній частині архітектури, підпрограми, процесу. Вони є формальними параметрами портів, підпрограм, циклів. У VHDL використовуються чотири класи об'єктів: константи, сигнали, змінні та файли.

У момент створення ***константа*** ініціалізується з присвоюванням заданого значення, яке не може бути змінене в процесі виконання програми. Наприклад, при оголошенні константи:

constant MINUS1: bit_vector(31 **downto** 0):=(31=>'1', **others** =>'0');

їй присвоюється агрегат (31=>'1', **others** =>'0'), в якому найлівішому біту 32-розрядного вектора надається значення 1, а іншим бітам (**others**) – значення 0. ***Агрегатом*** називається операція, що поєднує одне або кілька значень у значення композитного типу.

Змінна створюється при оголошенні. Присвоювання змінній відбувається негайно при виконанні оператора присвоювання. Змінну можна використати тільки в операторах процесу та в описах підпрограм. Тому змінні не мають доступу до інших процесів і підпрограм. Це обов'язкова умова детермінованої поведінки VHDL-моделей. При оголошенні змінної їй можна надавати діапазон зміни та присвоювати початкове значення, як, наприклад:

variable ct2: integer **range** 0 **to** 1023:=0;

Глобальна змінна – це особливий тип змінної, якій може бути присвоєне значення будь-коли та будь-де в об'єкті проекту. Також вона видима скрізь у межах проекту. Вона оголошується у декларативній частині архітектури, як наприклад:

shared variable result: real;

Через те, що мова не передбачає вирішення конфліктів при паралельному доступі до глобальної змінної, їх використання бажано обмежити випадками, коли задіяний лише один оператор присвоювання такій змінній.

Сигнал використовується для з'єднання частин проекту в ціле за допомогою інформаційного потоку. Тому сигнал – це об'єкт, за допомогою якого інформація передається між процесами та компонентами, тобто між паралельними операторами. Сигнал можна запам'ятовувати у його історії та відтворювати у симуляторі у вигляді графіка або таблиці. Оголошення сигналу виглядає так само, як оголошення змінної:

signal ct2: natural **range** 0 **to** 511;

Сигнал не можна оголосити в процесі або в підпрограмі. Порти об'єкта проекту неявно представляють собою сигнали.

Якщо сигнал або змінна не мають явно заданого початкового значення, то симулятор присвоює їм початкове значення як найліше з множини значень його типу. Наприклад, якщо сигнал цілого типу, то його початкове значення буде $-2^{31}+1$, типу real – значення $-1\cdot10^{308}$ (задається у пакеті STANDARD), типу bit – значення 0, типу boolean – значення false.

Тип ***файл*** призначений лише для позначення імені файла, до якого звертається програма, щоб прочитати дані, які потрібні для

моделювання або записати дані, які виникли при моделюванні. Файл оголошується як у прикладі:

```
type RealFile is file of real;
file dataFile : RealFile open WRITE_MODE is "MY_DATA_FILE.DAT";
```

Тоді файл на жорсткому диску з ім'ям MY_DATA_FILE.DAT буде відкритий для запису в нього чисел з плаваючою комою за допомогою процедури write, яка звертається до файлової змінної dataFile.

Літерали

Літерал — це такий лексичний елемент, якому присвоюється певне значення під час компіляції. Літералами є ідентифікатори, константи, операнди, мітки та інші елементи, які не є зарезервованими словами або коментарями. Особливістю мови VHDL є те, що синтаксис багатьох літералів строго відповідає їхньому типу.

Десятковий літерал — це ціле число за основою 10. Ним представляються константи цілого типу, наприклад:

```
constant thousand: natural:=1000;
```

Реальний літерал — це число, яке представлене з плаваючою комою. В його записі завжди повинна бути крапка, яка відділяє цілу частину від дробової, наприклад:

```
constant zero: real:=0.0;
constant a1_div_pi: real:=3.183e-1;
```

Символьний літерал — це елемент перелічуваного типу, який представляється символом ASCII. Він повинен бути занесений в одиночні лапки: '1', 'F', '#' . Біти представляються таким літералом.

Рядковий літерал — це ланцюжок символів, який занесений у подвійні лапки. Він відноситься до перелічуваного типу, наприклад:

```
constant name: string:= "Текст";
constant a128: bit_vector(7 downto 0):= "10000000";
```

Ідентифікатор використовується як ім'я об'єкта мови, підпрограми, мітки, тощо. Він складається з латинських літер, цифр, а також символа підкреслення між ними. Першим символом має бути літера. Ідентифікатори, які мають однакове прочитання, означають одне і те саме ім'я. Наприклад, A_One і a_one — це одне і те саме ім'я.

Розширений ідентифікатор — це будь-яка послідовність символів, обмежена зворотніми похилими, як наприклад: \<I@Ю<E\. Кожен розширений ідентифікатор є унікальним.

Вираз

Вирази у VHDL представляють собою формули, що визначають процес обчислення значень. У виразах скомбіновані операції та виклики функцій зі змінними, сигналами, константами, літералами та виразами в дужках. Наступні операції можуть бути використані у виразах. Вони наведені в порядку збільшення свого пріоритету.

Логічні операції **and**, **or**, **nand**, **nor**, **xor**, **xnor**. Ці операції визначені для типів bit та boolean, а також для одномірних масивів (векторів) елементів типу bit і boolean.

Операції порівняння = | /= | < | <= | > | >= . Рівність та нерівність визначені для будь-якого типу, крім файлового. Інші операції порівняння визначені для будь-якого скалярного типу або для одномірного регулярного типу. Масиви порівнюються поелементно, починаючи з найлівішого елемента. Результатом порівняння завжди є об'єкт типу boolean.

Оператори зсуву **sll**, **srl**, **sla**, **sra**, **rol**, **ror**. Лівим операндом має бути вектор бітів або булевських змінних, а правим — ціле число.

Оператори додавання + | − | &. Операнди для додавання та віднімання мають бути числового типу. Результат оператора конкатенації '&' — це вектор, який є з'єднанням векторів — операндів одного, хоча і довільного типу.

Оператори знаку **+|−** визначені для будь-якого числового типу.

Оператори добутку ***** | **/** | **mod** | **rem** визначені для цілих чисел. Оператори множення та ділення ще визначені для типу з плаваючою комою. Але не допускається, щоб обидва операнди були фізичного типу, оскільки тоді не визначений тип результату.

Інші операторы **abs** | ****** визначені для будь-яких числових типів, але при піднесенні до степеня a**b операнд b має бути цілого типу. Сюди відноситься також оператор інверсії **not.**

Об'єкти у виразах

Об'єктами у виразах можуть бути не тільки змінні, сигнали, константи та літерали, але і їхні частини та сукупності. і-й елемент масиву a задається як a(i). ***Вирізка*** послідовності елементів об'єкта регулярного типу з заданим діапазоном, наприклад, бітового вектора Data, задається як Data(15 **downto** 8). Елемент об'єкта комбінованого типу виділяється так само, як і у інших мовах — за допомогою крапки. Наприклад, якщо задано сигнал такого типу

```
signal sign:COMPLEX;
```

то його частини виділяються як sign.RE та sign.IM.

Однотипні об'єкти можна об'єднувати у спеціальний об'єкт композитного типу, який називається ***агрегат***. Наприклад, об'єкту комбінованого типу COMPLEX присвоюється агрегат:

```
sign<= (Re => 1000.0, Im => 0.0);
```

Зв'язування елементів може відбуватися в порядку нумерації елементів. Тоді воно називається позиційним зв'язуванням. Наприклад, в оголошенні:

```
variable v_5: bit_vector (0 to 4):=('0','0','0','1','1');
```

бітам 0, 1, 2 присвоєне значення 0, а бітам 2, 3 — значення 1. Якщо елемент зв'язується зі значенням за іменем, то таке зв'язування називається пойменованим. Наприклад, для тієї самої змінної v_5:

```
(3|4 => '1', others => '0')  або  (0 to 2 => '0'; 3 to 4 => '1').
```

Тут ключове слово **others** означає інші елементи значення й воно повинне стояти останнім у списку зв'язувань.

Для регулярного типу можна використовувати навіть простий вираз, який визначає номер елемента з діапазону цього типу. Наприклад, агрегат (i => '1', **others** => '0') задає вектор у якому на i-му місці стоїть 1, а інші біти - нульові.

Перетворення типів

Для того, щоб різнотипні об'єкти брали участь в обчисленні виразів без конфліктів, необхідно за допомогою перетворення типів привести ці об'єкти до одного типу — типу виразу. Якщо типи близькі, то можна виконати ***перехід типу***. Такими типами вважаються типи real та integer, регулярні типи з елементів того ж самого типу та з однаковими діапазонами індексів. Перехід від real до integer виконується як у прикладі:

```
B:=integer(10.5);
```

де результатом буде округлення до одиниць, тобто 11.

Якщо типи не є близькими, то слід викликати функцію перетворення типу, багато з яких входять до складу стандартних пакетів бібліотеки IEEE. Для підтипів, що мають один родоначальний тип, наприклад, для natural та positive, перетворення типу не потрібно. Часто програмісти складають власні функції перетворення типу.

Результат виразу може належати кільком типам одночасно. Тоді його необхідно позначити як ***кваліфікований вираз***, який кваліфіковано з заданим типом. Наприклад, для агрегата vect'(3=>'1',**others** => '0'), завдяки кваліфікації, буде чітко позначено, що він має тип vect.

1.4 Процеси та послідовні оператори

Основним елементом опису дискретної моделі є такий паралельний оператор, як процес. Тіло процесу утворюють послідовні оператори, до яких відносяться: присвоювання, оператори **if, case, loop, next, exit, assert, report, wait, return, null,** виклик процедури.

Оператор процесу

Оператор процесу описується у відповідності до наступного спрощеного синтаксису:

```
\оператор процесу\ ::= process [(\список чутливості)] [is]
                           {\оголошення в процесі\}
                       begin
                           {\послідовний оператор\}
                       end process;
```

Процес – це неявно заданий нескінченний цикл. Після ініціалізації процесу на початку моделювання процес виявляється або в активному стані, або призупиненим для очікування заданих подій. В активному процесі послідовні оператори в тілі процесу виконуються в заданому порядку, і після виконання останнього оператора керування повертається до першого оператора.

Процес призупиняється при виконанні оператора **wait**. Якщо в заголовку процесу є список чутливості, то всередині його не допускаються оператори **wait**. У цьому випадку сам компілятор вставляє оператор **wait** наприкінці ланцюжка послідовних операторів. Процес продовжує своє виконання після оператора **wait**, якщо відбудеться зазначена в цьому операторі подія або зміниться якийнебудь сигнал у списку чутливості процесу.

Оператор очікування події wait

Оператор **wait** призупиняє виконання процесу. За оператором

```
wait on C , R;
```

оператор процесу продовжить виконання за подією зміни сигналів C або R. За оператором

```
wait  until C = '1';
```

продовження відбудеться при зміні значення сигналу C з '0' на '1'. А оператор

```
wait  for  10 ns;
```

зупиняє процес на 10 наносекунд.

Оператори послідовного присвоювання

Ці оператори виконують присвоювання змінної:

\змінна\ := \вираз\;

або сигналу:

\сигнал\ <= \вираз\;

Присвоювання змінної виконується негайно. При виконанні присвоювання сигналу обчислюється вираз справа, який лише призначається сигналу. Остаточне присвоювання сигналу відбувається під час призупинки процесу.

Оператор if

Цей умовний оператор виконує ланцюжки послідовних операторів, причому від умови залежить, який саме ланцюжок операторів виконуватиметься. Спрощений синтаксис оператора:

```
\оператор if\::=if \умова 1\ then
        {\послідовний оператор 1\}
[ { elsif  \умова 2\ then
        {\послідовний оператор 2\}]
   [else
        {\послідовний оператор 3\}]
    end if;
```

Кожна з умов має бути виразом, що обчислює результат булевського типу. При виконанні оператора **if** умови перевіряються послідовно. Якщо результат — **true**, тоді виконується відповідний ланцюжок операторів і виконання оператора **if** припиняється.

Оператор case

Цей оператор дозволяє виконання одного з ланцюжків послідовних операторів залежно від виразу селектора. Його синтаксис:

```
\оператор case\::=case \простий вираз\ is
        when \альтернативи\ => {\послідовний оператор\}
       {when \альтернативи\ => {\послідовний оператор\}}
    end case;
\альтернативи\:= \альтернатива\{ | \альтернатива\}
```

У виразі селектора \простий вираз\ має стояти об'єкт цілого, перелічуваного або регулярного типу. Кожна з альтернатив \альтернатива\ має бути того самого типу, що і \простий вираз\ і представлена виразом або діапазоном (наприклад, 0 **to** 4 у випадку, якщо селектор — перелічуваного типу), або множиною, (наприклад, 5|7|9). Ніякі два значення, що одержуються з виразів альтернатив,

не повинні дорівнювати одне одному, тобто множини альтернатив не перекриваються. Останньою альтернативою може бути слово **others**, що вказує на альтернативи, які не перераховані. Якщо слово **others** не застосовується, то в альтернативах повинні бути перераховані всі можливі значення селектора.

Ось так можна запрограмувати перетворення цілого числа i в представлення у вигляді шістнадцятиричної цифри c :

```
case i is
   when 0 to 9 =>        c<= character'val(i+48);
   when 10 to 14 =>      c<=character'val(i+55);
   when others =>        c<='F';
end case;
```

Тут у виразі character'val(i+48) атрибут val видає значення символу зі стандартної таблиці символів ASCII за позицією i+48, тобто у 48-й позиції знаходиться символ 0, у 49-й — символ 1 і т.д.

Оператор циклу

Цей оператор кілька разів виконує послідовність операторів:

```
\оператор циклу\::=[\схема ітерації\]loop
            {\послідовний оператор\}
            {next  [when \умова\];}
            {exit   [when \умова\];}
    end loop;

     \схема ітерації\::=while \умова\    | for \змінна циклу \ in \діапазон\
```

Цикл за першою схемою ітерацій виконуватиметься, поки умова \умова\ не прийме значення false. Причому умова перевіряється до виконання циклу, і якщо вона дорівнює false, то цикл не виконується. У прикладі:

```
i:=1;  sum:= 0;
while i<=n loop
        sum:= sum or vec(i);
         i:=i+1;
end loop;
```

обчислюється змінна sum, що дорівнює сумі елементів масиву цілих чисел vec розмірності n. Якщо n = 0, то цикл не обчислюється. Цей приклад записується за допомогою другої схеми ітерації так:

```
sum:='0';
for i in 1 to n loop
        sum:= sum + vec(i);
end loop;
```

Тут змінна циклу i послідовно приймає значення 1,2,... з діапазону 1 **to** n. Змінну циклу не потрібно декларувати, як інші змінні та їй не можна виконувати присвоювання.

Якщо необхідно завершити чергову ітерацію до її закінчення, то застосовують оператор **next** запуску наступної ітерації. У прикладі:

```
numb:=0;
for i in 1 to n loop
   next when vec(i) /= 0;
   numb:=numb+1;
end loop;
```

обчислюється число нулів numb у масиві vec.

Якщо потрібно закінчити оператор циклу до завершення усіх ітерацій, застосовують оператор **exit** виходу із циклу. Оператор **loop** іноді застосовується без схеми ітерації, тобто коли цикл може виконуватися невизначено велику кількість разів, а саме протягом усього періоду моделювання.

Оператори assert і report

Ці оператори були введені в мову VHDL для виявлення помилок моделювання та повідомлення про них на консоль. У оператора пастки **assert** наступний синтаксис:

```
\оператор assert\::= assert \булевський вираз\
                [report \рядок повідомлення\][severity \вираз\];
```

Тут \булевський вираз\ — це перевірка певної умови правильності моделювання, що дорівнює false, якщо знайдено помилку, та true, якщо моделювання вірне; \рядок повідомлення\ - вираз типу string, що представляє рядок повідомлення про причину помилки. Вираз \вираз\ має наперед визначений тип severity_level, що складається з елементів note, warning, error та failure. Він означає рівень критичності помилки. При найвищому рівні failure моделювання зупиняється.

1.5 Підпрограми

До підпрограм відносяться процедури та функції. Специфікація підпрограми описує дії при виклику підпрограми й має наступний спрощений синтаксис:

```
\специфікація підпрограми\::=procedure \ім'я процедури\
                        [(\список параметрів\)] |
function \ім'я функції\[(\список параметрів\)] return \тип результату\ is
        {\оголошення в підпрограмі\}
```

begin
{\послідовний оператор\}
end [**procedure**][\ім'я процедури\]

\список параметрів\::=(\елемент списку\ {; \елемент списку\})

\елемент списку\::=[**constant** | **variable** | **signal**]
\ідентифікатор\{,\ідентифікатор\}:
[**in** | **out** | **inout**] \тип параметра\[:= \статичний вираз\]

Тут \ім'я процедури\, \ім'я функції\ — ідентифікатор процедури або функції. У списку параметрів вказується інформація про формальні параметри підпрограми. Замість формальних параметрів підставляються фактичні параметри під час виклику підпрограми. У елементах списку параметрів може оголошуватися, який це параметр (константа, змінна або сигнал) напрямок передачі параметра (**in**, **out** або **inout**), його тип і його значення за замовчуванням.

Ось приклад опису процедури сортування двох чисел:

```
procedure SORT2(variable x1,x2:inout integer) is
    variable t:integer;  -- оголошення в підпрограмі
begin
    if x1>x2 then       -- послідовний оператор
      return;     -- негайне припинення виклику процедури
    else
      t:=x1; x1:=x2; x2:=t;
    end if;
end procedure;
```

Список параметрів може бути відсутнім, тобто він тоді заданий неявно. При цьому імена фактичних параметрів збігаються з іменами формальних параметрів, оголошених у підпрограмі.

Виклик підпрограми містить ім'я підпрограми та список фактичних параметрів для передачі значень у підпрограму та з неї. Зв'язування формальних і фактичних параметрів може бути позиційним, пойменованим або комбінованим. Виклик процедури з позиційним зв'язуванням:

```
SORT2(a1,a2);
```

Той же самий виклик з асоціативним зв'язуванням:

```
SORT2(x2=>a2,x1=>a1);
```

При виклику підпрограм типи фактичних і формальних параметрів повинні строго збігатися. Якщо фактичний параметр — константа, то вона може бути задана виразом.

Функції

Виклик функції повертає одиночне значення. Тому він використовується у виразах. Допускаються функції, які мають параметр in (вважається за замовчуванням), який є сигналом або константою (константа — за замовчуванням). Зазвичай виклик функції не дає ***побічного ефекту***. Це означає, що ніякий об'єкт, оголошений зовні функції, не може бути змінений при її виклику, і для таких самих аргументів функція завжди повертає той самий результат.

У викликаній функції оператори виконуються послідовно. Виклик функції завжди закінчується оператором **return** з виразом, який обчислює величину, що повертається.

Крім ідентифікатора, позначенням функції може бути символ операції. Так, функція додавання векторів може бути оголошена як:

```
function "+"(a,b:bit_vector) return bit_vector;
```

Виклик такої функції не відрізняється від використання оператора додавання. Не допускається застосування у функції оператора **wait**. Тому функція завжди виконується негайно і повертає значення відразу після свого виклику.

Процедури

Використовуючи параметри в режимі **out** або **inout**, процедури можуть повертати довільну кількість значень. Виклик процедури — це окремий оператор, що буває як послідовним, так і паралельним. Якщо не вказувати режим параметра процедури, то мається на увазі, що він — в режимі **in**. Параметрами можуть бути константи, змінні, сигнали або файли. Якщо клас параметра не зазначений, то вхідний параметр — константа, а вихідний параметр — змінна.

Процедури можуть використовувати оператор **wait**, і тому паралельний виклик процедури з **wait** виконується як оператор процесу. У викликаній процедурі оператори виконуються послідовно, поки не виконається останній оператор або не трапиться оператор **return**.

Перезавантаження підпрограм

Допускається, щоб різні підпрограми мали однакове ім'я. Тоді говорять, що така процедура або функція перезавантажує іншу процедуру або функцію. При компіляції та реалізації виклику підпрограми з набору підпрограм з однаковим ім'ям вибирається відповідна підпрограма, у якій є належні число, порядок, імена (якщо поіменоване зв'язування параметрів) і типи параметрів.

1.6. Паралельні оператори

Описову частину архітектури становлять паралельні оператори. Такі оператори виконуються паралельно і тому їх порядок у програмі довільний.

Поведінковий опис проекту задається паралельним оператором — процесом, описаним у розділі **1.4.** Також паралельними операторами є паралельне присвоювання сигналу, паралельний виклик процедури, оператор вставки компонента паралельний оператор **assert** та інші.

Паралельне присвоювання сигналу

Присвоювання сигналу, яке вставлене в програму поза оператором процесу або підпрограми, вважається паралельним оператором. Такий оператор еквівалентний процесу із цим присвоюванням, після якого вставлено оператор **wait**. Так, наступні оператори еквівалентні.

```
OP1: S<=A+B+1;
OP2: process(A,B) begin
       S<=A+B+1;
     end process;
```

Логічні функції можна задавати різновидами цього оператора, такими як умовне присвоювання сигналу та селективне присвоювання сигналу.

Присвоювання одному і тому самому сигналу у різних паралельних операторах, як правило, ***не допускається***.

Умовне присвоювання сигналу

Воно означає вибір одного з виразів для присвоювання залежно від умови та має спрощений синтаксис:

```
\умовне паралельне присвоювання\::= \ім'я\<=
                                {\вираз\ when \булевський вираз\ else}
                                    \вираз\[when \булевський вираз\];
```

Цей оператор описує еквівалентний процес з оператором **if**. Поведінка тривходового мультиплексора описується таким умовним присвоюванням:

```
Y<=A when SEL= 0
     B when SEL= 1 else C;
```

Селективне присвоювання сигналу

Таке присвоювання еквівалентно процесу з оператором **case**. Воно має наступний спрощений синтаксис:

```
\селективне паралельне присвоювання\::= with \вираз\ select
                                        \ім'я\<=  {\вираз\ when \альтернативи\,}
                                                  \вираз\[when others ];
```

де \альтернативи\ мають те саме значення, що й в операторі **case**. Приклад оператора **case** у розділі 1.4 може бути представлений наступним селективним присвоюванням:

```
with i  select
      C<= character'val(i+48)  when 0 to 9 ,
          character'val(i+55) when 10 to 14 ,
          'F' when others;
```

Паралельний виклик процедури

Цей оператор еквівалентний процесу, що містить тільки цю процедуру, за якою стоїть оператор **wait**. Така процедура активізується, як тільки виникне зміна якого-небудь параметра — сигналу з режимом **in** або **inout**. Процедура, що використана в паралельному виклику, не повинна мати параметрів класу змінних.

Оператор вставки компонента

Цей оператор реалізує основний механізм задавання структури проекту. Компоненти, які вставляються в архітектуру, повинні бути заздалегідь перелічені в її декларативній частині. Так, при вставці компонента на рис.1.1. він оголошується як:

```
component LPF1 is generic(bypass: natural:=0);
                  port (C, R:in bit;    DI:in real ;
                        DO:out real);
end component;
```

Тоді цей компонент можна вставити в архітектуру за допомогою оператора вставки компонента:

```
U_F: LPF1 generic map(0)
          port map(CLK, RST, DIn, DOut);
```

Кожен оператор вставки компонента повинен мати унікальну мітку — тут U_F. У фразі зв'язування настроювальних констант **generic map**(0) виконується передача настроювальної константи. У фразі зв'язування портів і сигналів **port map** сигнали й порти компонентів з'єднуються відповідно до графа структури. Тут виконано

асоціативне зв'язування, коли сигнал співвідноситься з портом у тому самому порядку, який заданий в оголошенні компонента.

Частіше асоціативного зв'язування зустрічається ***поіменоване зв'язування***, коли сигнал призначається порту явним чином, а порядок зв'язувань портів байдуже який, як наприклад:

```
U_F: LPF1 port map(R=> RST, C=>CLK, DI=>DIn, DO=>DOut);
```

Тут зв'язування настроювальних констант не виконується. Тоді компілятор приймає до уваги настроювальну константу, приведену у декларації компонента, тобто, у даному випадку — 0. При вставці компонента усім його входам повинні бути призначені певні сигнали або константи. Деякі виходи можуть залишитись вільнмми. Тоді таким виходам призначається ключове слово **open**.

Оператор generate

Якщо необхідно неодноразово повторити один або кілька паралельних операторів, то використовують оператор **generate**:

```
\ опер.generate\::= \мітка\: for \ідентифікатор\ in \діапазон\ generate
                        [{\оголошення\}
                    begin]
                      { \паралельний оператор\}
                    end generate [\мітка\];
```

Мітка оператора **generate** необхідна для позначення згенерованої структури, \ідентифікатор\ - це параметр оператора **generate**, а фраза \діапазон\ - діапазон його зміни. Вони мають такі самі синтаксис і семантику, як в операторі **loop**.

Нехай треба запрограмувати ланцюжок з 10 фільтрів, таких як на рис.1.2. Тоді фрагмент програми виглядатиме як наступний:

```
type TArr is array(0 to 10) of real;
signal dt:TARR:=(others=>0.0);
. . .
dt(0)<=DI;
CHAIN:  for i in 1 to n generate
        LPF1: port map(R=> RST,C=>CLK, DI=>dt(i-1), DO=> dt(i));
          end generate;
```

Тут в оголошенні сигналу присвоєно нульове початкове значення усім його елементам за допомогою агрегата. У ланцюжку фільтрів на вхід і-го фільтра подається сигнал dt(i-1), а на вихід — dt(i). Результуючий сигнал — це сигнал dt(10).

Для того, щоб керувати структурою проектованого пристрою, застосовується умовний оператор **generate**:

```
\умовний generate\::= \мітка\: if \булевський вираз\ generate
```

```
    [  {\оголошення\}
 begin]
   { \паралельний оператор\}
 end generate [\мітка\];
```

Залежно від умови, заданої булевським виразом, оператор вставляє чи ні в структуру вузли, які представлені паралельними операторами.

1.7 Атрибути

У мові VHDL сигнали, змінні та інші об'єкти крім свого значення також мають множину атрибутів. Атрибути бувають різного типу: атрибут – тип, значення, сигнал, функція, діапазон.

Атрибут об'єкта записується як:

\ім'я об'єкта\' \ім'я атрибута\ .

Нижче розглядаються деякі наперед визначені атрибути.

Атрибути для скалярного типу

T'left — найлівіше значення множини елементів скалярного типу T.

T'right — найправіше значення множини елементів скалярного типу T.

T'image(X) — строкове представлення виразу X типу T.

T'value(X) — функція значення типу T від рядкового представлення X.

T'pos(X) — функція номера позиції елемента X у множині типу T.

T'val(X) – функція значення елемента типу T , що стоїть в позиції X.

Приклади атрибутів:

```
    type st is (one,two,three);
st'right = three,                st'pos(three) = 2,      st'val(1) = two,
positive'left =  1,              positive'right =2147483647,
integer'value("1_000") =1000,    integer'image(330) ="330".
```

Атрибути для регулярного типу

A'left — ліве значення діапазону індексів.

A'right - праве значення діапазону індексів.

A'range — діапазон індексів.

A'reverse_range — зворотній діапазон індексів.

A'length — довжина діапазону індексів.

Наприклад, якщо заздалегідь невідомі розміри масиву vec, то суму його членів можна обчислити так:

```
sum:='0';
for i in vec'range loop  -- або так:  for i in vec'left to vec'right loop …
        sum:= sum + vec(i);
end loop;
```

Атрибути сигналів

S'event – сигнал, дорівнює true, якщо відбулася подія в сигналі S у даному циклі моделювання.

S'last_value – сигнал такого самого типу, що й S, який містить значення S до останньої події в ньому.

Прикладом застосування атрибутів сигналів є наступний процес, що моделює синхронний регістр q1, у якому за переднім фронтом синхросигналу CLK записується сигнал a1:

```
process(CLK) begin
    if CLK='1' and CLK'event then -- D-тригер
       q1<=a1;
     end if;
end process;
```

1.8 Пакети для проектування та моделювання обчислювальних пристроїв

Стандартна технологія проектування та моделювання обчислювальних пристроїв за допомогою VHDL основана на застосуванні бібліотеки IEEE. Ця бібліотека складається з більш ніж десятка пакетів і призначена, в основному, для дискретного моделювання мікросхем на всіх рівнях, крім рівня транзисторів, а також для синтезу логічних схем. Далі будуть розглянуті особливості пакетів MATH_REAL та MATH_COMPLEX, які призначені для моделювання цифрової обробки сигналів.

Пакети

Пакет — це набір оголошень типів, констант, підпрограм, компонент і т.і., які оформлені таким чином, щоб бути доступними для різних проектів і використовуватися ними спільно. Інтерфейс пакету виконаний у вигляді оголошення пакету відповідно до синтаксису:

```
\оголошення пакету\::= package \ідентифікатор\ is
                          {оголошення в пакеті}
                           end [package][\ідентифікатор\];
```

З іншого боку, тіло пакету містить приховані деталі, які є часто невидимими для користувачів пакету. Він має наступний синтаксис:

\тіло пакету\::= **package body** \ідентифікатор\ **is**
{оголошення в тілі пакету}
end [**package body**][\ідентифікатор\];

Пакет MATH_REAL

У пакеті визначені основні константи і функції, які необхідні при математичних обчисленнях. Константи MATH_E та MATH_PI дорівнюють числу e та π, відповідно.

Визначена також функція '**' піднесення до ступеня для реальних аргументів.

Далі перелічуються елементарні функції від реальних аргументів з реальними результатами.

SIGN(X) — знакова функція, повертає 1.0 або −1.0 в залежності від знаку X, або 0.0 при $X = 0$.

SQRT(X) — функція квадратного кореня.

EXP(X) — експонента, дорівнює e^X.

LOG(X) — функція логарифму.

SIN(X) — синус від аргументу в радіанах.

COS(X) — косинус від аргументу в радіанах.

TAN(X) — тангенс від аргументу в радіанах.

ASIN(X) — арксинус.

ACOS(X) — арккосинус.

ATAN(X) — арктангенс.

ATAN2(X,Y) — арктангенс відношення X/Y.

В пакеті визначена функція UNIFORM, призначена для генерації випадкових чисел з рівномірним розподілом у діапазоні від 0.0 до 1.0. Такий генератор, що видає рандомізований сигнал rand, реалізується наступним процесом:

```
process(CLK)
        variable s1,s2:integer:=123;
        variable r:real;
begin
        if CLK='1' and CLK'event then
                UNIFORM(s1,s2,r);
                rand<=r;
        end if;
end process;
```

У цьому процесі в кожному такті синхросигналу сигнал rand приймає нове випадкове значення. Ініціалізація генератора встановлюється початковим значенням змінних s1,s2.

Пакет MATH_COMPLEX

У пакеті визначені основні типи, константи і функції, які необхідні при математичних обчисленнях з комплексними даними. Основний тип пакету визначений як наступний:

type COMPLEX **is record** RE, IM: real; **end record**;

Комплексні числа в полярних координатах задані типом:

type COMPLEX_POLAR **is record** MAG: real; ARG: real; **end record**;

Визначений також масив комплексних чисел COMPLEX_VECTOR з діапазоном, заданим цілими числами.

Задані основні константи:

MATH_CBASE_1 = **(1.0 + *j*·0.0),**

MATH_CBASE_j = **(0.0 + *j*·1.0),**

MATH_CZERO = **(0.0 + *j*·0.0)** типу COMPLEX.

Крім додавання, множення, ділення комплексних чисел визначені наступні функції.

ABS(Z) — функція модуля комплексного вектора Z;

ARG(Z) — функція фази (в радіанах) комплексного вектора Z;

CMPLX(X,Y) — повертає комплексне число $X + jY$;

CONJ(Z) — повертає комплексно спряжене значення — $\dot{Z}$;

EXP(Z) — повертає комплексну експоненту e^Z

COMPLEX_TO_POLAR(Z) — перетворює Z з декартових координат у полярні;

POLAR_TO_COMPLEX(Z) — перетворює Z з полярних координат у декартові.

2. Програмування алгоритмів цифрової обробки сигналів на VHDL

2.1 Чому, власне, VHDL?

Уже протягом двадцяти років для моделювання алгоритмів ЦОС традиційно застосовується пакет Matlab. Багато підручників, присвячених ЦОС, ґрунтуються на використанні цього пакету [9,13]. У більшості випадків за допомогою Matlab виконується дослідницьке числове моделювання явищ різної природи.

Для рішення завдань ЦОС до системи Matlab було додано бібліотеку Signal, а потім для моделювання систем авторегулювання — підсистему Simulink. При цьому в середовищі Matlab не тільки гарно вимальовуються графіки результуючих сигналів, але, наприклад, розраховуються коефіцієнти фільтрів, пропонуються типові рішення. Сьогодні програма, що описана мовою Matlab, сприймається як вхідні дані деякими системами синтезу логічних схем для ПЛІС, а також компіляторами для сигнальних мікропроцесорів. Таким чином, *сучасний розроблювач апаратних систем обробки сигналів нарівні з мовою VHDL має користуватися Matlab'ом. Але чи насправді так? І чи може він обійтися лише VHDL?*

Щоб відповісти на ці питання далі проведено порівняльний аналіз мов та систем Matlab і VHDL. Критерієм порівняння є придатність і ефективність мови для моделювання завдань ЦОС, які описані на верхньому — системному рівні.

Принцип моделювання

Основні операції в Matlab — це операції над векторами й матрицями. Наприклад, фільтрація — це множення вектора відліків сигналу на теплицеву матрицю імпульсної реакції, яке виконується підпрограмою згортки. Тому моделювання в Matlab — синхронне й переривчасте. Синхронне, тому що передбачається, що дані надходять з постійним періодом, а переривчасте, тому що порції даних і результатів, шо обробляються, обмежені заздалегідь заданими розмірами масивів. Моделювання виконується поетапно — перш ніж почати виконання наступного етапу обробки, необхідно завершити попередній етап і зберегти його результати в проміжному масиві.

При цьому рідко, коли розмір масиву перевершує 10^6. Це означає, що без додаткових зусиль по "зрощуванню" сусідніх порцій моделювання можна моделювати процеси, що тривають не більше за 10^6 тактів. Причому ця величина скорочується пропорційно

складності алгоритму, тому що для зберігання проміжних масивів може не вистачити пам'яті комп'ютера.

Якщо потрібне моделювання системи з різними частотами дискретизації (multirate system), то слід приводити частоти дискретизації до однієї кратної частоти. Наприклад, якщо два суміжних блоки працюють із частотами дискретизації 8 і 11,025 кГц, то для одержання одного відліку результату необхідно прорахувати не менш ніж 320 тактів з частотою, яка є кратною обом цим частотам.

Проблема безперервного моделювання вирішена в підсистемі Matlab-Simulink. В ній алгоритм представляється графом синхронних потоків даних (ГСПД), який відображається на екрані у вигляді граф-схеми. Такий алгоритм виконується програмно, отже, за один такт моделювання обробляючі блоки — вершини графа — виконують свої функції й обмінюються порціями даних, що відносяться до цього такту. Тобто тут моделювання відбувається за подіями. Однак при цьому губиться швидкодія, яка притаманна швидкісній обробці векторів і матриць у системі Matlab.

VHDL-моделювання виконується також за подіями. Процеси запускаються, як тільки приходять для них вхідні дані. Часові співвідношення можна відслідковувати з точністю до фемтасекунди. Моделювання — безперервне з будь-якими частотами дискретизації даних. Під проміжні змінні й сигнали виділяється невеликий об'єм пам'яті, який необхідний для зберігання даних лише протягом одного такту моделювання. Тільки під сигнали, які виводяться у вигляді графіків, симулятором заводяться динамічні масиви, максимальний розмір яких визначається об'ємом віртуальної пам'яті. Тому VHDL-модель можна моделювати довго — багато мільйонів та мільярдів тактів.

Швидкість моделювання

Синхронне моделювання — найшвидше часове моделювання. В Matlab такі процедури, як згортка, швидке перетворення Фур'є (ШПФ), перемножування й обернення матриць, виконуються швидкодіючими вбудованими процедурами. Але є причини, що стримують ріст швидкості моделювання.

Система Matlab не відчувала конкуренції з боку аналогічних систем, таких як Mathcad, Maple, тому що у всіх таких систем різні мови програмування й вони зайняли окремі споживацькі ніші. Тому не було необхідності оптимізувати швидкодію системи при моделюванні обробки сигналів. Але фірми-постачальники VHDL-симуляторів, завдяки конкуренції, змушені постійно підвищувати швидкість своїх симуляторів. З іншого боку, індустрія HBIC має гостру потребу у прискоренні моделювання і тому наполегливо стимулює процес підвищення швидкодії VHDL-симулятіров.

Кожна VHDL-програма виконується в симуляторі тільки після прямої оптимизуючої компіляції. Також сучасний симулятор розпізнає, що даний модуль може бути промодельований синхронно й переводить його в режим прискореного синхронного моделювання. Деякі VHDL-симулятори забезпечують моделювання проекту на паралельній системі з декількох комп'ютерів, з'єднаних у локальну мережу, що також прискорює цей процес.

Якщо VHDL-модель описана синтезованим стилем, то її моделюють за допомогою апаратного прискорювача, на базі ПЛІС. Таке моделювання дає прискорення від десятків до десятків тисяч разів.

Представлення даних

У переважній більшості випадків в Matlab використовуються дані з плаваючою комою з подвійною точністю. Тільки деякі модулі підсистеми Simulink працюють із фіксованою комою з розрядністю 8,16 або 32 біта, щоб наблизити моделювання алгоритмів ЦОС до їх виконання в сигнальних мікропроцесорах.

Моделювання з плаваючою комою рятує розроблювача від проблем масштабування, переповнення й втрати точності при обчисленнях. Але наприкінці розробки необхідно прикладати серйозні зусилля для заміни у відпрацьованому алгоритмі операцій з плаваючою комою на цілочисельні операції.

VHDL забезпечує моделювання з даними у будь-якому виді, у тому числі із плаваючою й фіксованою комою. Числа з фіксованою комою представляються як цілі зі знаком або як вектори бітів. В останньому випадку можливе моделювання обробки даних довільної розрядності.

Обробка цілих чисел в VHDL має наступні особливості:

— можна змінну або сигнал задати з діапазоном, тоді вихід змінної за межі діапазону, наприклад, при переповненні, викличе зупинку симулятора. Це відповідає задаванню розрядності операндів з точністю до біта. Таке обмеження також зручно при налагодженні алгоритму й корекції масштабних коефіцієнтів. Причому відповідність діапазонів операндів і результатів операцій, як у випадку векторів бітів, необов'язкова, тому

— програмування із цілими числами простіше, ніж з векторами бітів, що підтверджується також тим, що не потрібно використовувати функції перетворення типів і/або типи signed, unsigned;

— моделювання із цілими числами відбувається помітно швидше, ніж з векторами бітів або із числами з плаваючою комою;

— цілі числа займають у пам'яті істотно менше місця, ніж вектори бітів або числа з плаваючою комою;

— проекти зі змінними й сигналами цілого типу, особливо з діапазонами, можуть бути без змін странсльовані в логічну схему й

далі — у ПЛІС, тоді їх моделювання багаторазово прискорюється апаратними прискорювачами;

— цілі числа ближчі до архітектури більшості сигнальних процесорів, ніж числа із плаваючою комою, тому модель із цілими змінними легше переробити в програму для такого процесора.

На жаль, не завжди досить розрядності 32-розрядних цілих чисел, які застосовуються у більшості симуляторів. Тільки деякі із симуляторів використовують 64-розрядні цілі числа.

Як в Matlab, так і в VHDL застосовуються одномірні й багатомірні масиви. Істотною відмінністю є те, що в Matlab розріджені матриці можна представляти в компактній формі. Але такі масиви не застосовуються у переважній більшості завдань ЦОС.

Зате в VHDL можна вводити всілякі типи користувача. Так, у відомому пакеті IEEE_Math_Complex уведений комплексний тип.

Бібліотеки функцій і процедур

Немає такої загальноприйнятої математичної функції, якої б не було в бібліотеці Matlab. Але бібліотека процедур і функцій підсистеми Simulink — обмежена. Якщо потрібно приєднати до Simulink блок власної конструкції, то його слід описувати мовами C, Fortran або Ада із застосуванням особливих інтерфейсних структур і після трансляції підключити до бібліотеки особливим чином.

Варто сказати, що VHDL — це молодший брат мови Ада. Відміни в цих мовах, зокрема, в тому, що швидкодія VHDL —програм і кількість паралельних процесів у них набагато більші. Це досягнуто завдяки тому, що VHDL, на відміну від Ада, не витрачається час на рішення проблеми конфліктів при доступі до глобальних змінних.

Для математичних розрахунків, необхідних у моделюванні обробки сигналів, VHDL має бібліотеки IEEE_Math_Real і IEEE_Math_Complex. Однак, у них немає таких процедур, як рішення систем рівнянь, знаходження коренів поліномів, розрахунку коефіцієнтів фільтрів, ШПФ, які часто необхідні при розробці систем цифрової обробки сигналів і які є в Matlab'і.

Зате в VHDL немає проблем з розширенням і додаванням будь-яких бібліотек. Сама система VHDL — це дуже маленьке ядро з визначеними типами й операціями та великим числом бібліотек і пакетів. Деякі з них прийняті як міжнародні стандарти. У останній версії VHDL з'явились пакети fixed_matrix_pkg, real_matrix_pkg, complex_matrix_pkg, призначені для математичної обробки матриць чисел з фіксованою, плаваючою комою, та комплексних чисел, відповідно. Серед функцій над матрицями є також вирішення систем лінійних рівнянь.

Функція, процедура або об'єкт, позначені в VHDL-програмі атрибутом **foreign**, можуть бути виконані будь-яким чином, наприклад, як програма на Сі або як зовнішній спеціалізований пристрій.

Останнім часом для моделювання аналого-цифрових (Analog/ Mixed Signal) систем поширюється мова VHDL-AMS. Це VHDL з розширеним синтаксисом, що дає змогу додатково вести моделювання аналогових частин проекту [14].

Графічне подання даних

У Matlab зроблено майже все, щоб можна було представити будь-які дані зручними для сприйняття. Особливо це стосується подання багатомірних масивів.

На жаль, VHDL-симулятори надають скромні засоби подання чисельних даних. Тільки порівняно недавно стало можливим показувати чисельні сигнали у вигляді графіків з віссю аргументів як віссю часу, а двовимірні масиви - у вигляді матриць — як дамп пам'яті. Зате такі графіки можуть містити до сотен мільйонів точок, та їх масштабування й перегляд є зручними. Причому, ці точки можуть бути нерівновіддаленими, завдяки чому графік зберігається в комп'ютері упакованим.

Завдяки нескладному програмуванню поводження з файлами, користувачеві VHDL-симулятора надається можливість записати дані у файл й відобразити їх за допомогою якої-небудь сторонньої системи відображення даних, наприклад, того самого Matlab.

Засоби графічного програмування

Як Matlab, так і симулятори VHDL мають у своєму складі засоби графічного програмування. Структурний стиль програмування VHDL взаємно однозначно відповідає програмуванню у вигляді вимальовування блок-схеми деякого віртуального пристрою. Користувачеві надається можливість самому створити дизайн його нових графічних елементів. Також автоматні алгоритми можуть представлятися у вигляді діаграми своїх станів.

Придбання симулятора

Користування системою Matlab передбачає обов'язкове придбання ліцензії, яка коштує доволі дорого. У той же час більшість симуляторів VHDL є безкоштовними або умовно безкоштовними, чи надаються зі значними знижками для університетів та студентів.

Таким чином, можна зробити наступні висновки.

У VHDL багато можливостей для проектування систем ЦОС без використання Matlab. Це швидкісне моделювання тривалих і трудомістких алгоритмів, подання даних із плаваючою й фіксо-

ваною комою, у тому числі з екстремально великою розрядністю, наявність бібліотек математичних функцій та можливість графічного подання даних.

VHDL-симулятор виконує обробку найбільш ефективно при моделюванні алгоритмів ЦОС із цілими числами. Від такого представлення сигналів простіше здійснити перехід до апаратної або програмної реалізації цих алгоритмів.

Для ефективного використання VHDL-симулятора при розробці алгоритмів і пристроїв ЦОС необхідно створювати пакети процедур і функцій, аналогічних таким, які застосовуються в Matlab для цих цілей, а також покращити умови для відображення одно- і двовимірних сигналів у графічному вигляді.

Слід додати, що симулятори, такі як ActiveHDL, ModelSim, мають вбудовані засоби для створення, моделювання й підключення поведінкових моделей до системи Matlab-Simulink при одночасній роботі обох систем. Тобто VHDL-симулятор виконує швидкісне моделювання складних блоків Simulink-програм, а в Matlab паралельно моделюються інші блоки й відображаються результати.

2.2 Основні властивості цифрової обробки сигналів

Дискретні сигнали – це сигнали, які визначені у дискретні моменти часу і представляються послідовностями або рядами чисел. ***Цифрові сигнали*** – це дискретні сигнали у яких дискретні і час, і амплітуда. ***Цифрова обробка сигналів*** (ЦОС) має справу з перетвореннями сигналів, які є цифровими.

Основні закономірності ЦОС визначені для дискретних сигналів і уточнюються для цифрових сигналів лише на етапі реалізації ЦОС у конкретному пристрої. Такі уточнення стосуються вибору розрядності даних, точності результатів, перевірки надійності обчислень, але істотно не впливають на властивості алгоритму. Розглянемо деякі з цих закономірностей, які, у першу чергу, стосуються структурних властивостей алгоритмів ЦОС.

Дискретні послідовності, сигнали та системи

Дискретний сигнал, тобто послідовність чисел $x(n)$, записується як

$$x = \{x(n)\},\ -\infty > n > \infty\ .$$

Дискретні сигнали можна додавати: $x + y = \{x(n) + y(n)\}$, перемножувати $x \cdot y = \{x(n) \cdot y(n)\}$, множити на константу $x \cdot \alpha =$

$= \{\alpha x(n)\}$. Дискретний сигнал y є ***затриманим*** на n_0 циклів дискретизації, якщо $y(n) = x(n-n_0)$.

Дискретні системи призначені для перетворення одних сигналів $x(n)$ (вхідних) на інші $y(n)$ (вихідні, відгуки). Дискретна система математично визначається через ін'єктивне перетворення або оператор f, який відображає вхідний сигнал у вихідний $y(n) = f[x(n)]$.

Найчастіше розглядається клас ***лінійних систем***, для яких успішно застосовується математичний апарат лінійних відображень. Лінійні системи підлягають принципу суперпозиції. Якщо $y_1(n)$, $y_2(n)$ − відгуки на $x_1(n)$, $x_2(n)$, то система є лінійною, якщо

$$f[\alpha x_1(n) + \beta x_2(n)] = \alpha f[x_1(n)] + \beta f[x_2(n)] = \alpha y_1(n) + \beta y_2(n).$$

Клас систем, які ***інваріантні до зсуву***, підлягає наступній закономірності. Якщо $y(n)$ − відгук на сигнал $x(n)$, то $y(n-k)$ − є відгуком на $x(n-k)$. Будь-яку лінійну функцію $f[x(n)]$, яка інваріантна до зсуву, можна виразити через суму:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = x(n)\star h(n), \qquad (2.1)$$

Вираз (2.1) називається згорткою сигналу $x(n)$ та імпульсної характеристики $h(n)$, а знак «$\star$» – оператором згортки. Отже, функція f повністю характеризується імпульсною характеристикою $h(n)$.

Дві лінійні системи $h_1(n)$,$h_2(n)$, які інваріантні до зсуву і ввімкнуті послідовно, мають загальну імпульсну реакцію $h(n)$, що дорівнює згортці їх реакцій: $h(n) = h_1(n)\star h_2(n) = h_2(n)\star h_1(n)$. Дві такі самі системи, які ввімкнені паралельно, тобто $f = f_1 + f_2$, еквівалентні сумі їх реакцій: $h(n) = h_1(n) + h_2(n)$.

У багатьох системах ЦОС використовується підклас лінійних систем, які інваріантні до зсуву, у яких вхідний та вихідний сигнали зв'язані відношенням різницевого рівняння N-го порядку з постійними коефіцієнтами:

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{r=0}^{M} b_r x(n-r). \qquad (2.2)$$

Отже, n-е значення виходу можна обчислити на основі n-го значення входу $x(n)$ та відповідно N та M минулих значень виходу $y(n-k)$ та входу $x(n-r)$. Тоді імпульсну реакцію такої системи можна визначити як:

$$h(n) = \frac{y(n)}{x(n)} \quad . \qquad (2.3)$$

Імпульсна реакція (2.3) у загальному випадку є нескінченною і тоді це − система з нескінченною імпульсною характеристикою

(НІХ). Якщо в (2.2) $a(k) = 0$, тоді імпульсна реакція є скінченною: $h(n) = b(n)$, $n=0,...,M$, а система називається системою зі скінченною імпульсною характеристикою (СІХ).

Z-перетворення

z-перетворення $X(z)$ послідовності $x(n)$ визначається як:

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}, \tag{2.4}$$

де z – комплексна змінна. Якщо представити z у полярних координатах $z = re^{j\omega}$, причому $r = 1$, тобто $|z| = 1$, то z-перетворення виглядатиме як дискретне перетворення Фур'є. Отже, z-перетворення є узагальненим спектральним представленням дискретних сигналів, так само, як перетворення Лапласа є спектральним представленям безперервних, тобто, аналогових сигналів. Далі також буде показано, що у z-просторі можна показати спектральні властивості лінійних систем, що інваріантні до зсуву.

z-перетворення лінійне: якщо $f(n) = \alpha y_1(n) + \beta y_2(n)$, то $F(z) = \alpha Y_1(z) + \beta Y_2(z)$. Зсуву у часі $f(n-m)$ відповідає множення на ступінь у z-просторі: $z^{-m}F(z)$. Таким чином, функції z^{-m} відповідає затримка сигналу на m тактів дискретизації. Операції згортки $h_1(n)*h_2(n)$ відповідає множення: $H_1(n)H_2(n)$ у z-просторі. Слід наголосити, що ці властивості справедливі лише для області поширення змінної z, де розглянуті функції не розбігаються.

Імпульсна реакція системи (2.3) має відображення у z-просторі як ***передаточна функція***:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{r=0}^{M} b_r z^{-r}}{1 + \sum_{k=1}^{N} a_k z^{-k}}, \tag{2.5}$$

де a_k, b_r – дійсні числа. Передаточну функцію (2.5) можна розкласти на суму елементарних дробів. Частіше, наприклад, якщо $N \geq M$; $Q = N/2$, вона факторизується на добуток дробів:

$$H(z) = \prod_{k=0}^{Q} \frac{b_{0,k} + b_{1,k}z^{-1} + b_{2,k}z^{-2}}{1 + a_{1,k}z^{-1} + a_{2,k}z^{-2}}. \tag{2.6}$$

Отже, передаточна функція свідчить про спектральні властивості лінійної системи, що розглядається. На її основі знаходять ***амплітудо-частотну*** $|H(e^{-j\omega t})|$ та ***фазо-частотну*** $\arg(H(e^{-j\omega t}))$ характеристики системи.

Графічне представлення алгоритмів ЦОС

За Т'юрінгом, алгоритм — це задавання обчислювального процеса на певній моделі обчислювача, яка описана за допомогою математичних понять. Види алгоритмів відрізняються за видом моделей обчислювача, які мають різноманітне представлення. Важливою вимогою до такої моделі є зручність сприйняття алгоритму, який задано на цій моделі та ефективність виконання цього алгоритму.

Традиційно алгоритм ЦОС задають за допомогою формули, такої як (2.2). При цьому передбачається, що є джерело відліків сигналу $x(n)$, приймач сигналу результату $y(n)$ і хтось або щось, хто обчислює власне формулу. Необхідно цю формулу переобчислювати на кожному кроці, який має номер n та слідкувати за ітеративним збільшенням n. Також десь повинні зберігатись і на кожному кроці переписуватись проміжні результати $y(n-k)$, $x(n-r)$. Отже, задавання алгоритму формулою декларує, що саме слід обчислювати, але не показує, яким чином і у якому порядку це робити.

Алгоритм у вигляді формули реалізується у системі MathCAD. При цьому слід заздалегідь обмежити кількість відліків вхідних та вихідних сигалів, що обробляються, конкретним натуральним числом. Оскільки формула лише декларує правила обчислень, ефективність її реалізації у комп'ютері невисока.

Алгоритми ЦОС, як правило, передбачають свою реалізацію у ***реактивній*** обчислювальній системі (ОС), тобто в такій ОС, для якої не припустима втрата даних через неспроможність їх вчасної обробки у випадку обмеженої швидкодії ОС. На вхід пристрою ЦОС дані приходять з постійним періодом T_S, який є оберненим до частоти дискретизації $f_S = 1/T_S$. Тому, якщо пристрій не встигне виконати цикл алгоритму для даного $x(n)$ за час T_S, то наступне дане $x(n+1)$ не оброблятиметься, бо буде втрачене.

Такі алгоритми доцільно задавати на моделі графа потоків даних. Графова модель представляє собою модель обчислювача, яка обробляє потоки даних. При цьому ***потік даних*** — це назва двох категорій — як власне множини даних — сигналу, так і засобу передачі даних між елементами моделі.

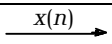
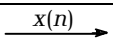
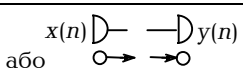
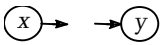
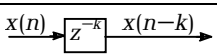
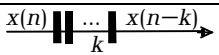
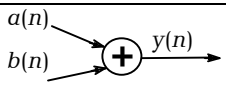
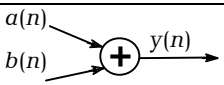
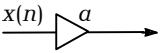
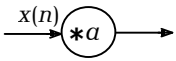
Граф потоків даних складається з вершин-акторів, які обробляють дані за елементарними операціями та потоків даних у вигляді дуг, що зв'язують між собою вершини. Кожна вершина-актор за певними правилами зчитує дані зі своїх вхідних потоків, обробляє їх і розміщує результати у свої вихідні потоки. Виконання алгоритму — це серія запусків вершин-акторів, для яких існують готові дані в потоках. Такі запуски можуть виконуватись паралельно у всіх вершинах. Граф потоків даних інтерпретується як паралельна програма, в якій підпрограми представляють собою вершини-актори, а комірки пам'яті чи буферні масиви — потоки даних.

Часто алгоритм ЦОС представляють як сигнальний граф, рідше – як граф сихронних потоків даних (ГСПД). ГСПД (рис.1.1) відрізняється від інших потокових моделей тим, що усі його потоки є синхронними. Два потоки у графовій моделі є ***синхронними,*** якщо є взаємна відповідність між усіма даними в одному та іншому потоках. Наприклад, у синхронних потоках можна перенумерувати дані й тому потоки $y(n)$ та $x(n)$ є синхронними так як є відповідність між n-ми відліками в них. Причому n розглядається як номер циклу дискретизації або тактового періоду, чи ітерації алгоритму.

Отже, у переважній більшості алгоритмів ЦОС сигнальні потоки є синхронними. Тому такі алгоритми можна представити ГСПД. Якщо у результуючому потоці наявність даних умовно залежить від вхідного потоку, то такі потоки можуть бути несинхронними. Це, наприклад, потоки в компресорі сигналів, який замінює ланцюжки нульових відліків кодами довжини цих ланцюжків.

Сигнальний граф та однорідний ГСПД є еквівалентними моделями. В табл. 2.1 показані графічні позначення елементів сигнального графа, ГСПД та їх відповідність частинам алгоритма ЦОС.

Таблиця 2.1. позначення елементів сигнального графа і ГСПД

Елемент алгоритму	Сигнальний граф	Однорідний ГСПД
Сигнал $x(n)$		
Вхідний та вихідний порти, джерело та приймач сигналів $x(n)$,$y(n)$		
Затримка на k циклів		
Додавання сигналів, вершина суматора $y(n) = a(n)+b(n)$		
Множення сигналу на константу $y(n) = a{\cdot}x(n)$, вершина блока множення		

Розглянемо приклад задавання алгоритму рекурсивного фільтру високих частот за допомогою сигнального графа і ГСПД. Передаточна характеристика такого фільтра дорівнює

$$H(z) = \frac{1-z^{-2}}{1+a_{1,k}z^{-1}+a_{2,k}z^{-2}} = (1-z^{-2})\cdot\frac{1}{1+a_{1,k}z^{-1}+a_{2,k}z^{-2}}, \qquad (2.7)$$

(порівняйте з(2.6)). Множникам передаточної функції фільтру відповідають різницеві рівняння (порівняйте з (2.2)):

$$u(n) = x(n) - x(n-2);$$
$$y(n) = u(n) - a_1 y(n-1) - a_2 y(n-2).$$

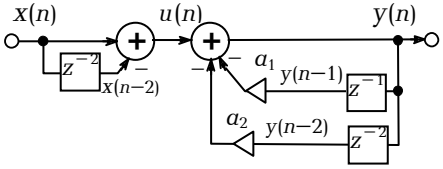


Рис. 2.1. Сигнальний граф алгоритму фільтра верхніх частот

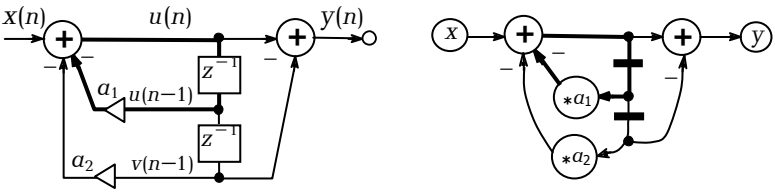
Рівняння реалізуються у сигнальному графі (рис.2.1). Вважається, що перед початком виконання усі елементи затримки зберігають нульові значення. Як тільки надходить чергове вхідне дане $x(n)$, воно зразу іде на елемент затримки на два цикли z^{-2} та на суматор "+", де воно додається до затриманого даного $x(n-2)$. Решта елементів моделі функціонують так само: як тільки є в наявності вхідні дані – одразу вони спрацьовують і видають результат на свій вихід. Таким чином, перші результати виконання моделі дорівнюють:

$$y(1) = x(1);$$
$$y(2) = x(2) - a_1 y(1);$$
$$y(3) = x(3) - x(1) - a_1 y(2) - a_2 y(1);$$
$$y(4) = x(4) - x(2) - a_1 y(3) - a_2 y(2);$$

Граф на рис.2.1 можна розглядати як структурну схему певного спеціалізованого обчислювача, у якого суматори – це відображення вершин додавання, блоки множення – вершин множення, а регістри проміжних операндів – елементів затримки. З цієї точки зору цей варіант алгоритму не є раціональним через надмірну кількість затримок. Для його оптимізації можна переставити місцями множники в формулі (2.7) та представити затримку z^{-2} як дві послідовні затримки z^{-1}:

$$u(n) = x(n) - a_1 u(n-1) - a_2 v(n-1);$$
$$v(n) = u(n-1);$$
$$y(n) = u(n) - v(n-1);$$

Результуючий сигнальний граф показано на рис. 2.2,а. Йому відповідає ГСПД на рис. 2.2,б.



а) б)

Рис. 2.2. Сигнальний граф алгоритму фільтра верхніх частот а) та відповідний ГСПД б)

Цей граф прийнято називати канонічною формою рекурсивного фільтра, так як він вміщує мінімальне число елементів затримки. Його зручно реалізувати програмним способом через те, що для нього потрібно лише дві комірки для зберігання проміжних результатів — затриманих сигналів $u(n)$ та $v(n)$.

Сигнальний граф і ГСПД можуть мати замкнені цикли. На рис. 2.2 такий цикл виділено товстою лінією. Якщо у замкненому циклі немає жодного елементу затримки, то сигнали в ньому будуть нескінченно переприсвоюватись протягом одного такту, тобто виникає ***блокування*** алгоритму. Тому обов'язковою умовою відсутності блокувань у сигнальному графі чи ГСПД є наявність принаймні по одному елементу затримки у всіх замкнених циклах. Іншою умовою відсутності блокувань є наявність початкових, наприклад, нульових даних у всіх елементах затримки, які входять у такі цикли.

Отже, сигнальний граф та ГСПД є зручною та наочною формою задавання алгоритму ЦОС. Далі алгоритми будуть часто задаватись за допомогою таких графів. Також від ГСПД можна легко перейти до VHDL-програми, що буде розглянуто далі.

2.3. Опис алгоритмів ЦОС мовою VHDL

Мовою VHDL задаються алгоритми, що представлені на графовій моделі потоків даних загального виду. При цьому відсутність блокувань алгоритму перевіряється на етапі компіляції та під час виконання алгоритма у VHDL-симуляторі. ГСПД є різновидом потокової моделі, на яку накладено певні обмеження. Ці обмеження мають відображення в описі алгоритму мовою VHDL.

Основним обмеженням є те, що вершина графа повинна видавати результат при кожному своєму запуску. Тобто результат $y(n)$ має такий самий номер або номер зі сталим зміщенням, що й вхідне дане $x(n)$. Тоді ці сигнали — вхідні дані й результат — будуть синхронними. Для цього елемент затримки повинен моделюватись оператором процеса, що запускається за синхросигналом, який є єдиним для усієї моделі. Для того, щоб елементи затримки, що стоять у замкнених циклах, мали певне початкове значення, оператори процесу повинні мати послідовні оператори початкового встановлення затриманних сигналів. Ці умови прийняті до уваги при формуванні відповідності між елементами ГСПД та операторами VHDL-програми (табл.2.2).

Опис алгоритму мовою VHDL на основі ГСПД виконується наступним чином. Вхідним та вихідним сигналам ставляться у відповідність порти вводу-виводу. Також до портів вводу додаються сигнали синхросерії clk та сигналу rst загального встановлення в 0. Усім потокам, що виходять з вершин графа та елементів затримки

призначаються відповідні сигнали певного типу, які об'являються в декларативній частині програми. Елементам ГСПД чи сигнального графу ставляться у відповідність оператори VHDL згідно табл.2.2.

Таблиця 2.2. Представлення елементів ГСПД у VHDL-програмі

Елемент алгоритму	VHDL-програма
Сигнал $u(n)$	**signal** u: real;
Вхідний та вихідний порти, джерело та приймач сигналів $x(n), y(n)$	**port**(clk, rst: **in** std_logic; x: **in** real; y:**out** real);

Затримка $u(n)$ на 1 цикл: $v(n) = u(n-1)$	**process**(clk,rst) **begin**
	if rst ='1' **then**
	v<=0.0;
	elsif clk='1' **and** clk'event **then**
	v<=u;
	end if;
	end process;

Додавання сигналів $y(n) = a(n) + b(n)$	y<=a+b;
Множення сигналу на константу $y(n) = a \cdot x(n)$	**generic**(a:real:=0,9876); або **constant** a:real:=0,9876; . . . y<=a*x;

Якщо необхідно реалізувати багатотактову затримку до n тактів, то декларується регулярний тип та сигнал цього типу, наприклад:

```
type Delay_T is array (0 to n-1) of real;
signal ud: Delay_T;
```

та в операторі процеса програмується пам'ять типу FIFO на n слів:

```
process(clk,rst) begin
    if rst ='1' then
        ud<= (others=>0.0);
    elsif clk='1' and clk'event then
        ud<=ud(1 to n-1)&u;
    end if;
end process;
```

Затриманий на $m \le n$ тактів сигнал береться до розрахунків як ud(m-1).

Опис фільтра мовою VHDL згідно з вищеописаними правилами, який виконано для ГСПД на рис. 2.2,б, наведено нижче:

```
entity BPF2_IIR is
    generic(a1:real:= - 1.3965919;  -- коефіцієнти фільтра
            a2:real:=   0.8794446);
    port(clk, rst: in std_logic; -- входи синхросигналу та встановлення в 0
            x: in real;          -- вхід фільтра
            y:out real);         -- вихід фільтра
end entity BPF2_IIR;

architecture beh of BPF2_IIR is
    signal u,ud,vd:real;          -- внутрішні сигнали фільтра
begin
    u<=x - a1*ud - a2*vd;  -- результат першої вершини суматора
    y<=u - vd;             -- результат другої вершини суматора

    process(clk,rst) begin  -- процес реалізації затримок
      if rst ='1' then
          ud<=0.0;          -- встановлення в 0 регістрів затримки
          vd<=0.0;
      elsif clk='1' and clk'event then
          ud<=u;        --перший регістр затримки
          vd<=ud;       --другий регістр затримки
      end if;
    end process;
end beh;
```

Задавання коефіцієнтів фільтра як настроювальних констант **generic** дає змогу використовувати цю модель як універсальний фільтр високих частот, що настроюється під час вставки компонента цього фільтра.

Функції, які виконуються у вершинах ГСПД, можуть бути довільними функціями, що представляються паралельними операторами присвоювання сигналу. Такі функції завжди знаходять відображення в апаратурі як деякі логічні комбінаційні схеми. Можна використовувати оператор процеса, але такий оператор також має відобразитись у логічну комбінаційну схему. Для цього послідовні умовні оператори повинні спрацьовувати при кожному запуску оператора процеса. Такі умовні оператори мають альтернативні гілки, які покривають усі комбінації вхідних умовних сигналів.

Наприклад, наступний оператор зліва не відображається у комбінаційну схему. Він описує деяку схему з пам'яттю, яка зберігає попередній результат при комбінації a = '0' та b = '1' . Правильним записом оператора буде, наприклад, такий, як справа, у якому альтернатива **else** перекриває решту комбінацій.

```
process(x,v,a,b) begin
    if a ='0' and b='0' then
        y<= x + u;
    elsif a ='1' then
        y<= x - u;
    end if;
end process;
```

```
process(x,v,a,b) begin
    if a ='0' and b='0' then
        y<= x + u;
    elsif a ='1' then
        y<= x - u;
    else
        y<= x;
    end if;
end process;
```

Вище розглядались лише одорідні ГСПД, у яких кожна вершина обчислювала стільки вихідних результатів, скільки груп вхідних даних поступило на її входи. У неоднорідного ГСПД кількість результатів, що обчислюються в деякій вершині протягом одного циклу алгоритму та кількість груп вхідних даних відносяться як взаємно прості натуральні числа. Наприклад, вершина дециматора, яка проріджує вхідну послідовність у n разів, протягом одного циклу сприймає n вхідних відліків і видає лише один відлік.

Такий ГСПД часто називають багаточастотним ГСПД (multirate SDF), тому що його частини функціонуютьз різними частотами дискретизації відповідних сигналів.

Під час опису неоднорідного ГСПД за допомогою VHDL слід додати керування прийомом даних в елементи затримки. Такий елемент описується так, як у наступному прикладі:

```
process(clk,rst) begin
    if rst ='1' then
        v<=0.0;
    elsif clk='1' and clk'event then
        if en ='1' then
            v<=u;
        end if;
    end if;
end process;
```

Тут сигнал en є сигналом дозволу запису в регістр затримки v. Цей сигнал генерується схемою керування моделі, яка лічить такти циклу обчислень і видає сигнал en у певних тактах.

3. Система Active HDL

Запуск системи

Система автоматизованого проектування Active-HDL поставляється фірмою Aldec і призначена для моделювання дискретних систем, які описані мовою VHDL або Verilog.

VHDL-проект розміщується в системі Active-HDL в окремому каталозі в робочій області (Project space). Система Active-HDL запускається при активізації криптограми (натисканням кнопки)  . Під час цього система запитує, чи відкрити існуючу робочу область (Open existing workspace), чи створити нову робочу область (Create new workspace). Якщо потрібно створити нову робочу область, то система запрошує вказати ім'я та каталог для неї. Можна створити порожню область (Create an empty design) або створити каталог проекту й помістити в нього існуючі файли проекту (Add existing resource files). В останньому випадку слід вибрати додавання файлів (Add files), знайти потрібний каталог, відзначити курсором у ньому файл або групу сусідніх файлів (тримати натиснутою кнопку Shift), або кілька файлів (тримати натиснутою кнопку Ctrl).

Керування системою

Система Active HDL керується за допомогою маніпулятора "миша". Для цього курсор установлюють на рядок меню, піктограму, об'єкт проекту тощо, які активують натисканням лівої кнопки миші. При натисканні правої кнопки миші спливає додаткове меню команд та опцій. Багато команд керування системою пов'язані з "гарячими" клавішами.

Стандартним способом керування Active HDL, як і будь-яким іншим VHDL-симулятором, є задавання керуючих команд у вікні консолі. Наприклад, команди

```
asim HPF2_IIR beh
run 1000 ns
```

активують симулятор для об'єкта HPF2_IIR з архітектурою beh і заставляють його виконати моделювання протягом 1000 нс. Втім, усі дії, що викликані мишею, стосовно керування симулятором, зводяться до автоматичної генерації керуючих команд, які відтворюються у вікні консолі. Часто для автоматизації моделювання послідовність керуючих команд збирають у скрипт, який називається DO-файлом.

Графічний інтерфейс системи

Головна панель системи Active HDL показана на рис. 3.1. Вона містить у собі область меню, вікно обходу проекту (Browser), вікно консолі, робоче вікно та багато інших, які встановлюються в області робочого вікна в тому чи іншому режимі роботи системи.

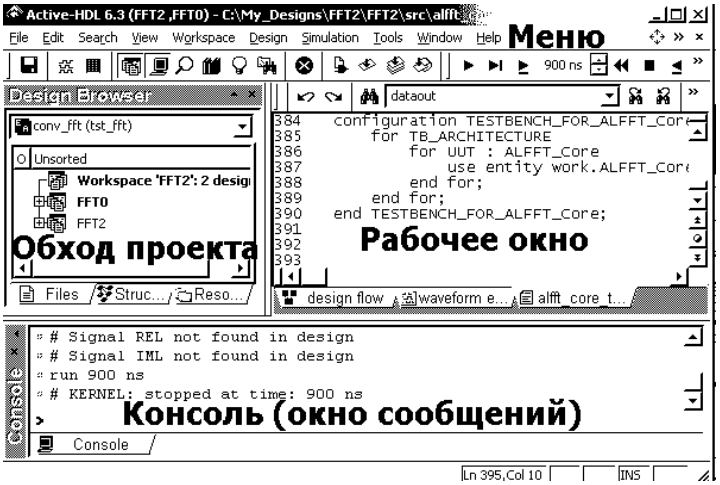


Рис.3.1. Головна панель системи Active HDL

Область меню містить у собі систему спливаючих меню й групи піктограм, пов'язаних з режимами роботи системи, які найчастіше застосовуються. Серед них вона містить піктограми керування запуском і зупинкою моделювання, вікно завдання періоду моделювання та вікно, що вказує наявний момент моделювання. У режимі настроювання (Customize) можна встановити бажаний набір піктограм й вікон. Вікно обходу проекту має вікно дерева файлів, бібліотек проекту і керування файлами (Files), вікно дерева компонентів проекту, його сигналів, констант і змінних (Structure), вікно допоміжних файлів проекту (Resources) і вікно вибору вершини проекту (рядок угорі вікна).

У вікні консолі видаються повідомлення про хід і результати компіляції, моделювання, знайдених помилках і т.п. У ньому задаються також рядкові керуючі команди для симулятора.

У робочому вікні виставляються робочі вікна проекту. Це можуть бути вікна текстових редакторів для програм на VHDL, Verilog, C, DO-файлів, графічні редактори структурних схем проекту (Block diagram), графа алгоритму автомата (State diagram), умовних графічних позначень компонентів проекту (Symbol editor), які

використовуються для введення та редагування об'єктів проекту. Для видачі результатів моделювання використовуються вікно графіків (Waveform editor), вікно протоколу зміни сигналів (List) та інші. Для керування проектом при взаємодії із зовнішніми САПР ПЛІС використовують вікно ходу проектування (Design flow manager). Одночасно можуть бути відкриті кілька робочих вікон, позначених унизу прапорцями. Необхідне вікно вибирається за прапорцем.

Додавання та редагування файлів

Для додавання VHDL-файлу до проекту вибирають курсором його місце в дереві файлів у вікні обходу проекту (Add new file), натисканням правої кнопки відкривають меню додавання файлів і вибирають або створення файлу (New VHDL-source), або починають пошук готового файлу у файловій системі (Add files to design).

При введенні й редагуванні VHDL-файлу використовують ті самі можливості, які надаються звичайними текстовими редакторами. Бібліотека шаблонів (Language assistant), що відкривається кнопкою  , надає можливість вибрати типові шаблони операторів і конструкцій VHDL та вставити їх у текст.

Для підключення до проекту стандартних бібліотек і вибору з них інтерфейсів компонентів, щоб їх вставити у свій текст, використовують керування бібліотеками (Library manager), яке відкривається кнопкою  .

Компіляція файлів

Після введення й редагування VHDL-файлу його компілюють. Компіляція викликається кнопкою  . При цьому помилки, знайдені при компіляції видаються у вікні консолі. Одночасно рядок з помилкою підкреслюється в тексті червоною хвилястою лінією.

При груповій компіляції усіх файлів проекту використовують кнопки  й . У першому випадку файли компілюються підряд, а в другому - зі зміною порядку, щоб об'єкти проекту, які є компонентами, компілювалися раніше, ніж об'єкти, в яких вони вставлені. Скомпільовані VHDL - файли, які не мають помилок і зауважень, підключаються до результуючого проекту та відзначаються позначкою на дереві файлів у вікні керування файлами. Коли всі файли проекту успішно скомпільовані, слід задати кореневий об'єкт проекту у вікні вибору вершини проекту, який позначено значком  . Лише після цього можливе моделювання проекту.

Графіки сигналів

При моделюванні проекту його поведінку зазвичай спостерігають у вікні графіків, які викликають за допомогою кнопки . У цьому вікні задають графіки сигналів, які потрібно спостерігати при моделюванні. Для цього можна використати кнопку додавання сигналів. Зручніше ці графіки задати "перетягуванням" сигналів з вікна обходу дерева компонентів проекту і його сигналів.

Для кожного графіка можна задати режим відображення - Properties, установку якого викликають натисканням клавіш Alt+Enter або з меню, яке викликане правою кнопкою миші. При установці основних властивостей (General) можна вказати основу системи числення відображуваних бітових векторів або цілих чисел. При установці властивостей самого графіка (Display) задають висоту графіка в пікселах (Heigh), його кольори й форму: Literal - вказівка значення, Logic - логічні рівні та Analog – графік кривої. В останньому випадку задають діапазон чисел, які відповідають рівням усікання сигналу при імітації аналогового посилення з насиченням.

Під час моделювання обраним сигналам можна примусово задати постійне або змінне у часі значення. Тоді до порту чи сигналу об'єкта підключається стимулятор, сигнал якого переважає над сигналом у моделі. Режим програмування стимуляторів викликають активізацією кнопки . При цьому генератором сигналу може бути генератор прямокутних імпульсів (Clock), часова послідовність різних рівнів (Formula), постійне значення (Value), натискання клавіші (Hotkey), генератор сигналу, що змінюється лінійно (Counter), попередньо заданий у редакторі графік сигналу (Custom), попередньо визначений і запам'ятований графік (Predefined).

Використовуючи меню файлів, вікно з графіками сигналів можна зберегти у файлі з розширенням AWF або відновити, прочитавши обраний AWF-файл. Відмічені графіки можна «перетягнути» у інші застосунки, наприклад, в текстовий редактор, використовуючи операції «помістити у кишеню», «викласти з кишені» (клавіші Ctrl-C, Ctrl-V).

Моделювання проекту

Перед запуском моделювання встановлюють часовий інтервал моделювання. Наприклад, інтервал 1000 нс установлюється так: 1000 ns . Якщо коренева архітектура або конфігурація проекту задана, то її моделювання запускається кнопкою . При цьому у вікні поточного часу замість повідомлення: No simulation спостерігається зміна часу в наносекундах або пікосекундах залежно від

мінімального інтервалу зміни сигналу в моделі. Крім того, у цьому ж вікні зі знаком "+" вказується поточний номер дельта-циклу моделювання.

У процесі моделювання кнопка його зупинки ⊗ має червоний колір. Моделювання можна зупинити, натиснувши на цю кнопку.

Через заданий інтервал часу моделювання зупиняється. При цьому у вікні графіків вимальовуються графіки вибраних сигналів. Моделювання можна продовжити, натиснувши кнопку ▶ або привести у початковий стан, натиснувши на кнопку ◀◀.

Після кожної компіляції будь-якого файлу проекту стан симулятора необхідно привести у початковий стан, натиснувши на кнопку ◀◀. Закінчити моделювання можна натиснувши кнопку ■.

Ввід структурної схеми

ActiveHDL та інші симулятори дають змогу ввести алгоритм у вигляді структурної або функціональної схеми. Далі ця схема автоматично транслюється у відповідну VHDL-програму, яка написана структурним стилем, і яка транслюється та підключається до проекту звичайним чином.

Щоб розпочати ввід структурної схеми у меню File-New або Add New File вибирають Block Diagram . Після цього відкривається сторінка редактора графічних структурних схем. Властивості креслення схеми, такі як розміри аркуша, назва проекту та інформація про автора креслення, задаються в меню Page Setup, що викликається при натисканні правої кнопки миші.

Меню вводу схеми вміщує кнопки вводу компонентів , ліній зв'язку , багаторозрядних шин , портів вводу-виводу , допоміжних операторів та інших конструкцій мовою VHDL , позначення глобальної шини та інші.

Під час вводу компонента відкривається меню Symbol Toolbox, яке забезпечує керування базою даних загальних компонентів та компонентів, що відносяться до даного проекту. Як правило, перед тим, як складати структурну схему, вже є розробленими і описаними мовою VHDL її компоненти. Тоді умовні символи компонентів, які розроблені власноруч, вибираються з гілки Units Without Symbols. В її елементах пропонуються початкові креслення умовних символів об'єктів проекту, які вже розроблені та входять в бібіліотеку проекту. Умовний символ, який з'являється у окремому вікні при виборі компонента, перетягується за допомогою миші у поле креслення.

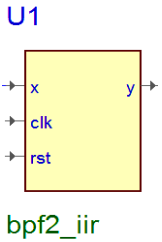


Рис.3.2. Умовне позначення фільтра

Далі цей символ вважається як введений і зафіксований. Наприклад, символ фільтра з ГСПД на рис. 2.2,б показано на рис.3.2. Символ має дві мітки — мітку компонента, яку можна відредагувати (тут U2), та назву компонента, яку змінювати не можна.

Якщо компонент креслення вибрано, то відкриваючи правою кнопкою миші меню, можна в ньому вибрати різні опції, за допомогою яких редагуються властивості компонента. В опції Symbol Properties редагуються мітка компонента, його настроювальні константи generics, ім'я архітектури, атрибути користувача, додаються коментарі, початкове значення портів тощо. Вибір опції Edit SymbolIn Separate Window відкриває вікно редактора символів, в якому можна змінити креслення символа. Опція Replase Symbol дає змогу замінити даний компонент на інший, який має аналогічні порти.

Якщо після складання креслення було змінено об'єкт проекта компонента, тобто відбулося редагування VHDL-файлу, то для корекції його символа та структури в цілому необхідно вибрати опцію Compare Symbol With Contents та підтвердити цю корекцію.

Після того, як символи компонентів та портів вводу-виводу розміщені на аркуші креслення, їх порти вводу-виводу з'єднуються між собою лініями зв'язку або багаторозрядними шинами . Для виділених курсором ліній та шин чи їх відрізків можна відредагувати властивості, такі як тип сигналу, його ім'я, початкове значення, атрибути користувача, режим відображення властивостей, колір та товщина лінії тощо за допомогою опції Properties. Якщо лінії не присвоєно певного імені, то відповідному сигналу редактором буде присвоєне довільне ім'я, наприклад, NET1234. При підведенні кінця однієї лінії дотично до іншої лінії вони з'єднуються, що автоматично помічається крапкою.

В структурній схемі багато загальних ліній зв'язку, таких як синхросигнал, загальне встановлення в нуль та інші, як правило, не зображаються. З цією метою такі сигнали відмічаються позначенням глобальної шини . При цьому назва лінії зв'язку та тип сигналу повинні співпадати з назвою та типом портів компонентів або з іменем відрізків лінії зв'язку, які під'єднані до цих портів.

Після того, як компоненти та лінії зв'язку введено, розташування компонентів та форму ліній зв'язку можна редагувати за допомогою вибора курсором та перетягування по полю креслення. Виділений елемент або групу елементів зображення можна вида-

лити, скопіювати в буфер та вставити з буфера опціями Cut, Copy, Paste, відповідно.

Остаточно сформоване креслення структури транслюється у відповідний VHDL-код за допомогою вибору копки меню Generate HDL Code або транслюється з подальшою компіляцією кнопкою . Сформований VHDL-код можна переглянути, натиснувши кнопку .

На рис.3.2. показано приклад структури чотириступеневого фільтра, яку було введено за допомогою редактора графічних структурних схем.

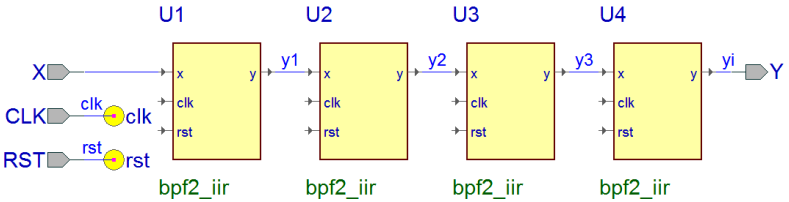


Рис.3.2. Структура чотириступеневого фільтра

4. Лабораторні роботи

4.1. Лабораторна робота 1
Основи мови VHDL. Модель генератора сигналів

Мета роботи: одержати знання та навички у розробці поведінкових моделей генераторів тестових сигналів для ЦОС, навчитись користуватись VHDL-симулятором.

Теоретичні відомості

Різноманітні генератори сигналів використовуються для діагностування, тестування та налагодження алгоритмів і пристроїв ЦОС. Часто використовується генератор синосоїдальних сигналів через те, що такий сигнал проходить через лінійну систему без зміни своєї форми. Шумовий сигнал має широкий спектр і тому дослідження його проходження через лінійну систему дає змогу оцінити її частотні характеристики. Також шум додається до сигналу, щоб виявити фільтруючі властивості системи. Пропускаючи через систему сигнал спеціальної форми, можна перевірити її якість, так як система з гідною передаточною характеристикою зберігає форму сигналу.

В лабораторній роботі слід розробити генератор сигналів спеціальної форми Gen та виконати з ним ряд дослідів, включивши його у схему на рис. 4.1.

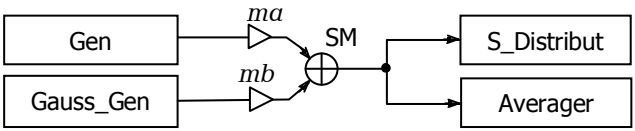


Рис. 4.1. Схема для лабораторної роботи 1

Інші блоки схеми на рис.4.1. – це генератор гаусового шуму Gauss_Gen, блок аналізу статистичного розподілення сигналу S_Distribut та блок осереднення сигналу Averager.

В основу генератора гаусового шуму покладено виконання центральної граничної теореми. Згідно з нею, якщо випадкова величина є сумою досить великого числа незалежних між собою доданків, кожен з яких є незначним за величиною, то ця сума має розподіл, який наближається до нормального.

Блок Gauss_Gen має дві частини: генератор чисел з рівномірним розподілом та суматор, який складає до nn згенерованих чисел.

Генерація чисел з рівномірним розподілом в інтервалі [0.0,1.0] виконується за конгруентним алгоритмом з використанням процедури UNIFORM. Одержана сума ділиться на nn/2 і видається як відлік шуму, який розподілено в інтервалі [-1.0,1.0]. Тут nn — це настроювальна константа. При nn = 1 генератор видає шум з рівномірним розподілом, при nn = 2 — з трикутним розподілом, а при nn ≥ 10 — практично з гаусовим розподілом.

Сигнали від генераторів Gen та Gauss_Gen складаються у суматорі SM після множення на масштабні коефіцієнти *ma, mb,* відповідно (рис. 4.1). В результаті, одержуємо сигнал $x(n)$, що досліджується.

Статистичні властивості сигналу $x(n)$ можна оцінити завдяки побудові гістограми розподілу його вірогідності. Обчислення гістограми виконують наступним чином. Інтервал зміни величини $x(n)$, наприклад, $a < x(n) < b$, ділять на k підінтервалів рівної довжини. i-му інтервалу, включаючи 0-й підінтервал $x(n) \le a$ та k+1 —й підінтервал $b \le x(n)$, ставиться у відповідність натуральне число M_i, яке дорівнює кількості відліків $x(n)$, які попадають у відповідний підінтервал. Підінтервал часто називають інтервалом групування або кишенею.

Нехай ширина підінтервалу дорівнює $c = (b - a)/k$, а границя i-го інтервалу — $d_i = a + i{\cdot}c$. Тоді після аналізу усієї послідовності $x(n)$ у i-й кишені одержимо число таких $x(n)$, що $d_{i-1} \le x(n) < d_i$. Виключеннями є 0-а кишеня з кількістю відліків $x(n) < a$ та k+1-а кишеня з числом таких $x(n)$, що $b \le x(n)$.

При обробці $x(n)$ на комп′ютері організують масив цілих чисел M розміром k+2. Якщо $a \le x(n) < b$, то обчислюють номер кишені

$$i = \left[\frac{(x(n)-a)}{c}\right] + 1. \qquad (4.1)$$

Тут дужки [] означають цілу частину числа. Одержавши індекс за формулою(4.1) у i-ту комірку, тобто кишеню $M(i)$ додають 1. Такі обчислення повторюють для усіх $x(n)$.

На основі одержаних результатів можна для послідовності $x(n)$ довжиною N обчислити вибіркову функцію щільності вірогідності за формулою

$$p_i = \frac{M(i)\ k}{N(b - a)}\ . \qquad (4.2)$$

Вибір параметрів *a, b, k* залежить головним чином від уяви про розподіл сигналу $x(n)$, способу збору даних та їх кількості. Напевне, що кількість кишень недоцільно вибирати більше за $\sigma\sqrt{N}$ чи за

кількість рівнів квантування сигналу. Параметри *a, b, k* уточнюються після циклів пробної побудови гістограми.

Побудова гістограми виконується блоком S_Distribut. Він має настроювальні константи channels, тобто *k*, dmin, тобто *a* та drange, який дорівнює $b - a$.

Якщо сигнал $x(n)$ – періодичний, то $x(n+T_o) = x(n)$, де T_o – його період. Реальний сигнал $x'(n)$ має додаткову складову у вигляді шумового сигналу $q(n)$, тобто $x'(n) = x(n) + q(n)$. Блок осереднення сигналу Averager виконує додавання періодів сигналу $x'(n)$. Якщо шум $q(n)$ є послідовністю випадкових чисел з нормальним законом розподілу, то після додавань N періодів сигналу $x'(n)$ його складова $q(n)$ збільшиться менше, ніж у N разів. Таким чином, після N додавань періодів сигналу $x'(n)$ та нормалізації результату діленням на N для $n = 0,...,T_o-1$ та $k = 0,...,NT_o-1$ одержимо результат

$$y(n) = \frac{1}{N}\sum_{i=0}^{N-1}(x(n+T_o i)+q(k)) = x(n) + \frac{1}{N}\sum_{i=0}^{N-1}q(k) =$$

$$= x(n) + m(q(k)) + \frac{1}{\sqrt{N}}\ q'(k)\ , \qquad (4.3)$$

де $m(q(k))$ – математичне очікування шуму $q(k)$, $q'(k)$ – осереднений шумовий сигнал. Отже, бачимо, що осередненням сигналу можна знизити рівень шуму до $\sqrt{N}$ разів. Тому осереднення періодичного сигналу часто використовується для його фільтрації від шуму. При цьому важливими умовами є прямування до нуля математичного очікування шуму та строга рівність довжини масиву $y(n)$, де накопичується осереднений сигнал, періоду T_o сигналу $x(n)$.

Завдання для роботи

1. Розробити модуль Gen, який генерує періодичну функцію, що має два інтервали:

$$f(i) = \begin{cases} f_1 \text{ на першому інтервалі довжиною } N_1, \\ f_2 \text{ на другому інтервалі довжиною } N_2. \end{cases}$$

Тут період функції дорівнює $T_o = N_1 + N_2$. Функції f_1, f_2 та їх параметри N_1, N_2 беруться з таблиці 4.1 в залежності від номера варіанту роботи. Модуль описується мовою VHDL як об'єкт і архітектура. Параметри N_1, N_2 повинні бути настроювальними константами, тобто generics.

2. Розробити стенд для випробувань, що відповідає структурі на рис. 4.1. Стенд описується структурним стилем мовою VHDL з застосуванням засобу Block Diagram системи ActiveHDL.

Таблиця 4.1 Функція, яка реалізується в модулі Gen

№ вар.	f_1	f_2
1	$\sin(2\pi i/N_1) + \sin(6\pi i/N_1)/3$	$\sin(2\pi i/N_2) + \sin(6\pi i/N_2)/3$
2	i/N_1	$1 - i/N_2$
3	i^2/N_1^2	$(N_2 - i)^2/N_2^2$
4	$\sqrt{i/N_1}$	$\sqrt{(N_2 - i)/N_2}$
5	$\sin^2(2\pi i/N_1)$	$\sin^2(2\pi i/N_2)$
6	$\exp(-\lvert i - N_1/2\rvert)$	$-\exp(-\lvert i - N_2/2\rvert)$
7	$1/(1 + (10\cdot(i - N_1)/N_1)^2)$	$1/(1 + (10\cdot i/N_2)^2)$
8	$1 - \exp(-10\cdot i/N_1)$	$\exp(-10\cdot i/N_2)$
9	$\frac{4\sin(8\pi(i-N_1/2)/N_1)}{i-N_1/2+\delta}$	$\frac{4\sin(8\pi(i-N_2/2)/N_2)}{i-N_2/2+\delta}$
10	$\ln(i+1)/\ln N_1$	$\ln(N_2 - i + 1)/\ln N_2$
11	i/N_1	$\sqrt{(N_2 - i)/N_2}$
12	i^2/N_1^2	$1/(1 + (10\cdot i/N_2)^2)$
13	$\sqrt{i/N_1}$	$\exp(-10\cdot i/N_2)$
14	$\sin^2(2\pi i/N_1)$	$-\sin^2(2\pi i/N_2)$
15	$\exp(-\lvert i - N_1/2\rvert)$	$\sin^2(2\pi i/N_1)$
16	$1/(1 + (10\cdot(i - N_1)/N_1)^2)$	$\sqrt{(N_2 - i)/N_2}$
17	$1 - \exp(-10\cdot i/N_1)$	$(N_2 - i)^2/N_2^2$
18	$\frac{4\sin(8\pi(i-N_1/2)/N_1)}{i-N_1/2+\delta}$	$-\frac{4\sin(8\pi(i-N_2/2)/N_2)}{i-N_2/2+\delta}$
19	$\ln(i+1)/\ln N_1$	$\exp(-10\cdot i/N_2)$
20	i/N_1	$\exp(-10\cdot i/N_2)$
21	i^2/N_1^2	$\ln(N_2 - i + 1)/\ln N_2$
22	$\sqrt{i/N_1}$	$\exp(-10\cdot i/N_2)$
23	$\sin(2\pi i/N_2) - \sin(6\pi i/N_2)/3$	$\sin(2\pi i/N_2) - \sin(6\pi i/N_2)/3$
24	$\exp(-\lvert i - N_1/2\rvert)$	$-\frac{4\sin(8\pi(i-N_2/2)/N_2)}{i-N_2/2+\delta}$
25	$1/(1 + (10\cdot(i - N_1)/N_1)^2)$	$\exp(-10\cdot i/N_2)$
26	$1 - \exp(-10\cdot i/N_1)$	$\sqrt{(N_2 - i)/N_2}$
27	$\frac{4\sin(8\pi(i-N_1/2)/N_1)}{i-N_1/2+\delta}$	$\exp(-\lvert i - N_1/2\rvert)$
28	$\ln(i+1)/\ln N_1$	$(N_2 - i)^2/N_2^2$

При цьому структура складається з розробленого модуля Gen та готових модулів генератора гаусового шуму Gauss_Gen, блока аналізу статистичного розподілення сигналу S_Distribut, блока осереднення сигналу Averager, генератора синхроімпульсів CLK_GEN та суматора синалів ADD2.

3. Встановити початкові значення настроювальних констант в модулях: період синхросигналу tclk=10 ns – в CLK_GEN, N1 $=N_1=100$, N2 $=N_2=200$ – в Gen, математичне очікування m $=0.0$, вид генерації nn $=10$ в Gauss_Gen, підсилення каналів сигналу від Gen – ma $=1.0$, сигналу від Gauss_Gen – mb $=3.0$ – в модулі ADD2. В модулі S_Distribut кількість кишень channels $=102$, значення нижньої межі dmin та діапазону зміни сигналу drange задати з урахуванням діапазону зміни заданої функції $f(i)$ та доданої до неї величини шуму з урахуванням ma та mb. Період сигналу, що осереднюється в модулі Averager, дорівнює N $=T_o=N_1+N_2$.

4. Скомпілювати побудовану модель стенду для випробувань. Виконати моделювання для 16 періодів сигналу. Номер періоду сигналу видається на вихід Naver модуля Averager. Якщо необхідно, скоректувати параметри dmin, drange.

5. Фрагменти графіків сигналів на виходах модулів ADD2, S_Distribut, та Averager зберегти у звіті лабораторної роботи разом з параметрами модулів.

6. Повторити пп. 3,4,5 з різними значеннями настроювальних констант N1, N2, N mb, m, nn та для різної тривалості моделювання (кількості періодів сигналу). Виконати також моделювання для випадку N $\neq N_1+N_2$.

7. Проаналізувати одержані графіки сигналів з виходів модулів S_Distribut та Averager. При цьому перевірити справдження формули (4.3). Сформулювати залежності між формою і характером сигналів та формою кривої їх розподілення на виході модуля S_Distribut. Зробити висновки по роботі.

Приклад виконання роботи

Нехай задано функцію

$$f(i) = \begin{cases} \text{arctg}(i/N_1) \text{ на першому інтервалі довжиною } N_1, \\ \cos(i/N_2) \text{ на другому інтервалі довжиною } N_2. \end{cases}$$

Тоді модуль, який її генерує, описується як наступний:

```
entity Gen is generic(n1:natural:=100;
            n2:natural:=200);
            port(CLK : in STD_LOGIC;
                 RST : in STD_LOGIC;
```

```
                DATA_OUT : out REAL:=0.0;
                START : out BIT);
end Gen;
architecture MODEL of Gen is
        signal ct2:natural;
begin
        process(CLK,RST)        begin
            if RST='1' then
                ct2<=0;
                START<='1';
            elsif CLK='1' and CLK'event then
                START<='0';
                if ct2=n1+n2-1 then
                        ct2<=0;
                else
                        ct2<=ct2+1;
                end if;
                if ct2<=n1 and ct2>0 then
                        DATA_OUT<= arctan(real(ct2)/real(n1));
                else
                DATA_OUT<= cos(MATH_PI*real(ct2-n1)/real(n2));
                end if;
            end if;
        end process;
end MODEL;
```

Побудована схема стенду для випробувань показана на рис.4.2.

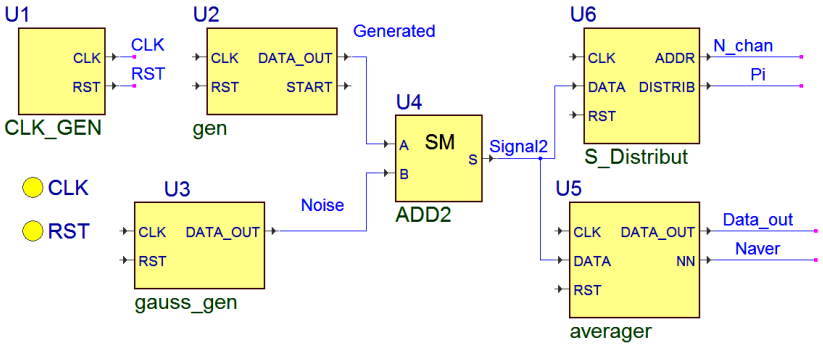


Рис. 4.2. Схема стенду для випробувань

На рис. 4.3. показані графік сигналу, згенерованого модулем Gen, графік сигналу з доданим гаусовим шумом, амплітуда якого відноситься як 3:1 та сигнал після 100 циклів осереднення у модулі

Averager. Аналіз графіків показує, що дійсно, після 100 осереднень періодичного сигналу рівень доданого гаусового шуму зменшується приблизно у 10 разів.

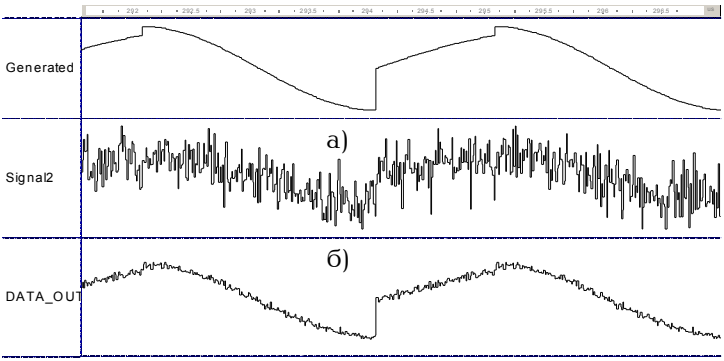


Рис. 4.3. Графіки згенерованого сигналу (а), сигналу з доданим гаусовим шумом (б), та сигналу після осереднення (в)

На рис. 4.4. показані графіки сигналів, одержаних на виході модуля S_Distribut для сигналів без шума, з доданим шумом з рівнем 0,3 та рівнем 3. Графіки показують, що сигнал має позитивну постійну складову. Перший графік має форму, характерну для сигналу без шуму, другий графік показує наявність в сигналі шуму, а третій — що в сигналі переважає шум, причому саме шум з імовірно гаусовим розподілом.

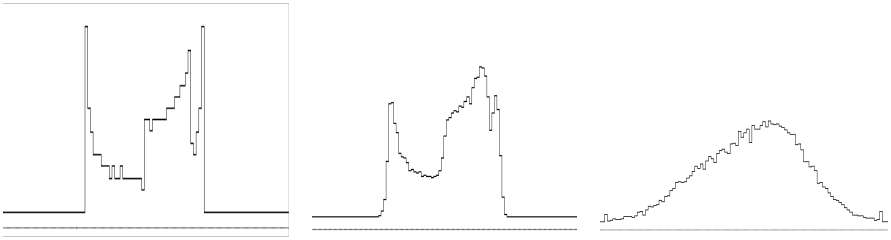


Рис. 4.4. Графіки розподілу сигналу без шуму (а) та сигналу з доданим гаусовим шумом (б), (в)

4.2. Лабораторна робота 2
Дослідження властивостей Z-перетворення

Мета роботи: одержати знання про спектральні властивості дискретних сигналів і передаточних функцій та навички у їх дослідженні за допомогою VHDL.

Теоретичні відомості

У підрозділі 2.2 вже розглядались деякі властивості z-перетворення, які використовуються для формування алгоритму обробки сигналу. Далі розглядатимуться деякі його уточнення. z-перетворення широко використовується в ЦОС через те, що сигнал виду $x(n) = z^n$ є ***власною функцією*** для лінійної системи, яка інваріантна до зсуву. Наведемо деякі приклади такого сигналу.

При $z = 1$: ...,$x(-2) = 1$, $x(-1) = 1$, $x(0) = 1$, $x(1) = 1$, $x(2) = 1$,...
При $z = 0.5$: ...,$x(-2) = 4$, $x(-1) = 2$, $x(0) = 1$, $x(1) = 0.5$, $x(2) = 0.25$,...
При $z = e^{-j\pi/2}$: ...,$x(-2) = -1$, $x(-1) = -j$, $x(0) = 1$, $x(1) = j$, $x(2) = -1$,...

Те, що функція є власною для системи, означає, що після обробки сигналу, який має форму цієї функції, одержимо сигнал, який має таку саму форму, але затриманий у часі та має іншу амплітуду. Покажемо, що z^n є власною функцією для системи, що описується імпульсною реакцією. Якщо такий сигнал подати на вхід системи, то одержимо, згідно з (2.1) та властивостями лінійної системи, що інваріантна до зсуву, вихідний сигнал

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)z^{n-k} = \sum_{k=-\infty}^{\infty} h(k)z^{-k}z^n = z^n \sum_{k=-\infty}^{\infty} h(k)z^{-k} \qquad (4.4)$$

Отже, бачимо, що функція z^n пройшла незмінною на вихід системи. Згідно з (2.5) та (4.4), ***передаточна функція*** системи дорівнює

$$H(z) = \sum_{k=-\infty}^{\infty} h(k)z^{-k}. \qquad (4.5)$$

Так само, як і z, функція $H(z)$ є комплексною функцією з амплітудою $|H(z)|$ та фазою $\arg(H(z))$. Функція $H(z)$ має дві множини особливих точок. Перша множина – це такі точки $q_1,...,q_M$, для яких $|H(q_i)| = 0$ і тому називаються ***нулями*** функції $H(z)$. Друга множина – це такі точки $r_1,...,r_N$, для яких $|H(r_i)| = \infty$ і тому вони називаються ***полюсами*** функції $H(z)$.

Щоб знайти нулі та полюси $H(z)$, вираз (2.5) перетворюють у дріб з позитивними степенями при z. Якщо $M - N = L$, то чисельник (2.5) множать на z^M, а знаменник — на z^N. Після цього одержують дріб

$$H(z) = z^{-L} \cdot \frac{\sum_{r=0}^{M} b_r z^{M-r}}{\sum_{k=0}^{N} a_k z^{N-k}}, \qquad (4.6)$$

чисельник та знаменник якого є поліномами з позитивними ступенями при z, де $a_0 = 1$, множник z^{-L} представляє собою затримку на L тактів і, як правило, може бути усунений з розгляду. Ці поліноми прирівнюють до нуля та знаходять корені цих рівнянь $q_1,...,q_M$ та $r_1,...,r_N$, відповідно. Маючи множини нулів та полюсів, функцію $H(z)$ можна виразити через них за допомогою формули

$$H(z) = K \cdot \frac{(z - q_1)...(z - q_M)}{(z - r_1)...(z - r_N)} = K \cdot \frac{(1 - q_1 z^{-1})...(1 - q_M z^{-1})}{(1 - r_1 z^{-1})...(1 - r_N z^{-1})}, \qquad (4.7)$$

де K — ***коефіцієнт підсилення*** системи. Отже, множини нулів та полюсів визначають передаточну функцію системи.

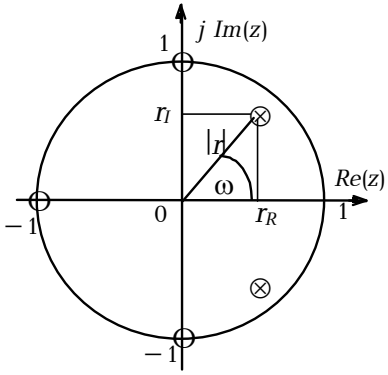


Рис. 4.5. Нулі та полюси фільтра нижніх частот

Графічно нулі та полюси зображають як точки на комплексній площині у вигляді кружечків та хрестиків, відповідно. На рис. 4.5 показано приклад розміщення нулів та полюсів деякого фільтра нижніх частот. Нуль чи полюс, як корінь рівняння, знаходять як пару реального та уявного чисел: $r = r_R + jr_I$. Для зручності корінь представляють у полярних координатах парою амплітуди $|r|$ та фази ω: $r = |r|e^{-j\omega}$. При цьому фаза ω відповідає певній частоті сигнала чи передаточної функції. Амплітуда полюса, як правило, не перевищує одиниці, тобто $|r| \le 1$, а кількість полюсів з ненульовими уявними координатами завжди парна.

Модуль передаточної функції $|H(z)|$ у тривимірному просторі можна представити як нескінченну мембрану, що пружно розтягується, яка натягнута над комплексною площиною на одиничній висоті. Точки мембрани, які відповідають нулям функції, притягнуті до комплексної площини, а точки, що означають полюси, відтягнуті вгору до нескінченності. Якщо на мембрані зобразити концентрич-

ні кола, то після її деформації згідно з розміщенням нулів та полюсів як на рис. 4.5, одержимо зображення, як на рис. 4.6. Товста лінія на поверхні $|H(z)|$ (рис. 4.6) відображається на комплексну площину як коло з одиничним радіусом. Вона будується при

$$z^{-n} = e^{-j\omega n} = \cos(-\omega n) + j\sin(-\omega n). \qquad (4.8)$$

При підстановці послідовності $e^{-j\omega n}$ в (2.4) одержимо дискретне перетворення Фур'є, а при підстановці в (4.5) чи (4.6) — передаточну функцію

$$|H(\omega)| = |H(e^{-j\omega n})|, \quad (4.9)$$

тобто залежність коефіцієнту передачі системи від частоти ω. Цю функцію називають ***амплітудо-частотною характеристикою*** (АЧХ) системи. При цьому крива, яка на рис. 4.6 розміщена на циліндрі одиничного радіуса з віссю о$|H(z)|$, розкручується на площині графіка АЧХ у вигляді безкінечної періодичної кривої з періодом 2π, як на рис. 4.7.

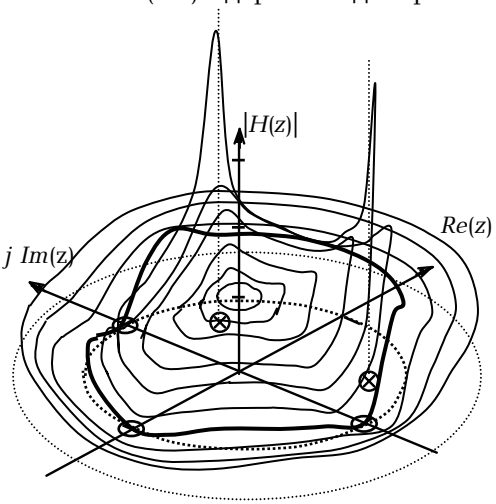


Рис. 4.6. Функція $|H(z)|$ у трьохвимірному просторі

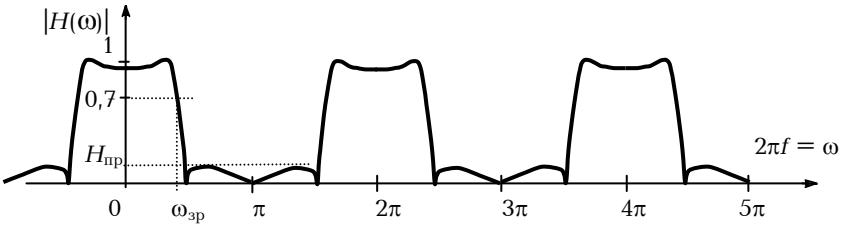


Рис. 4.7. Амплітудо-частотна характеристика фільтра нижніх частот

Для зручності вісь частот графіка АЧХ нормують, поділивши аргумент на 2π, так що одиниця на цій осі відповідає частоті дискретизації сигналу. Аналогічно АЧХ, обчислюють ***фазо-частотну функцію*** (ФЧХ) системи

$$\arg(H(\omega)) = \operatorname{arctg}\left(\frac{\operatorname{Im}(H(e^{-j\omega}))}{\operatorname{Re}(H(e^{-j\omega}))}\right) . \qquad (4.10)$$

Для АЧХ цифрових фільтрів домовлено, що у ***смузі пропускання*** АЧХ має рівень не нижче $\sqrt{0{,}5} \approx 0{,}7$. Це означає, що у цій смузі потужність фільтрованого сигналу не знижується менше за рівень 0,5 від номінальної потужності. Відповідна частота границі смуги називається ***частотою зрізу*** ($\omega_{зр}$ на рис. 4.7). Максимальний рівень фільтрованого сигналу у смузі непропускання називається ***рівнем заглушення*** фільтру ($H_{пр}$ на рис. 4.7).

Через те, що більшість систем ЦОС є системами з послідовно з'єднаних блоків, стало зручним представляти АЧХ у логарифмічному масштабі, тобто $H(\omega) = 20 \cdot \lg\left|H(e^{-j\omega n})\right|$. Тоді результуюча АЧХ системи дорівнює сумі логарифмічних АЧХ її складових. При цьому відлік рівнів АЧХ ведуть у децибелах. Наприклад, рівню 1 відповідає 0 дб, рівню 0,7 — рівень −3дб., рівню 0,1 — рівень −20 дб.

У цій лабораторній роботі досліджуються характеристики передаточної функції

$$H_1(z) = \frac{b + a(b+1)z^{-1} + z^{-2}}{1 + a(b+1)z^{-1} + bz^{-2}} \qquad (4.11)$$

та інших передаточних функцій на її основі. $H_1(z)$ відповідає ***фазовому фільтру***, нулі та полюси якого показані на рис. 4.8. Нулі та полюси функції (4.8) мають однаковий кут ω. Якщо модуль її полюса r, то модуль нуля дорівнює оберненій величині, тобто $1/r$. В результаті цього нулі повністю компенсують вплив полюсів і тому АЧХ дорівнює 1 у всьому діапазоні частот.

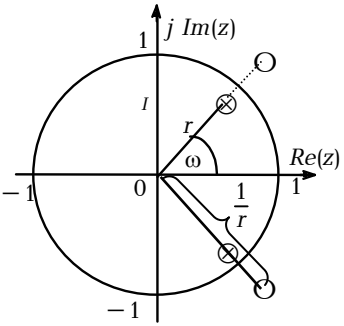


Рис. 4.8. Нулі та полюси фазового фільтра

Положення полюсів та нулів визначається коефіцієнтами a,b, причому

$$a = -\cos(\omega); \qquad b = r^2. \qquad (4.12)$$

Фазовий фільтр не змінює амплітуди сигналу, але перекручує його фазу. Причому чим ближче параметр b до одиниці, тим різкіше перекручування фази на частоті ω. Якщо з'єднати паралельно два фазових фільтра з характеристиками $H_1(z)$, $H_2(z)$, наприклад:

$$H_3(z) = (H_1(z) + H_2(z))/2, \qquad (4.13)$$

то в їх загальній передаточній характеристиці $H_3(z)$ нулі перестають компенсувати полюси. Обидва канали $H_1(z)$, $H_2(z)$ пропускають два сигнали без зміни їх амплітуд. Але на частотах, де різниця фаз дорівнює π, 3π,..., сигнали компенсують один одного і результуючий сигнал є нульовим. Навпаки, на частотах, де різниця фаз 0,2π, 4π,...,

цей сигнал залишається без змін. Завдяки цьому, в результуючій АЧХ $|H(\omega)|$ з'являються смуги пропускання та непропускання.

У будь-якій лінійній системі, що інваріантна до зсуву, можна збільшити кожну затримку в k разів. Тоді замість передаточної функції $H(z)$ розглядається функція з ***кратними затримками*** $H(z^k)$. Ефект такого збільшення затримок полягає в тому, що масштаб графіків АЧХ, ФЧХ вздовж осі частот зменшується у k разів.

Нехай $k=2$, тоді на графіку на рис. 4.7 слід замінити відліки π на $\pi/2$, 2π на π і т.д. В результаті, кількість смуг пропускання та непропускання в частотній області, що розглядається, подвоюється.

Нехай маємо $H_4(z) = H_3(z^k)$. Якщо АЧХ $|H_4(\omega)|$ має кілька смуг пропускання, то робочу смугу можна виділити за допомогою ***маскуючого фільтра*** $H_M(z)$, який з'єднаний з ним послідовно. Тобто результуюча функція

$$H(z) = H_4(z)\cdot H_M(z) \ . \qquad (4.14)$$

Для цього АЧХ маскуючого фільтра повинна мати смуги непропускання у тих діапазонах частот, де знаходяться смуги пропускання в АЧХ $|H_4(\omega)|$, які необхідно заглушити.

Для обчислення передаточних функцій бажано використати пакети IEEE.MATH_REAL та IEEE.MATH_COMPLEX. В останньому пакеті визначені типи, константи та функції, які оперують з комплексними даними.

Завдання для роботи

1. Розробити програму на VHDL, яка будує графіки $|H_1(z)|$ для різних значень $|z|$ для експериментального визначення положення нулів та полюсів функції (4.11) з параметрами, заданими у табл. 4.2. Обчислити нулі та полюси функції (4.11) і порівняти їх з практично одержаними значеннями. Для визначених положень нулів та полюсів перевірити відношення (4.12) та те, що модуль нуля дорівнює оберненій величині модуля полюса, а їх кути співпадають.

2. Розробити програму на VHDL, яка будує графіки $|H_3(\omega)|$, $\arg(H_3(\omega))$ для заданої функції (4.13) і одержати ці графіки. Визначити ширину смуги пропускання, ширину перехідної смуги, рівень заглушення у смузі непропускання.

3. За допомогою програми на VHDL побудувати графіки $|H_4(\omega)|$, $\arg(H_4(\omega))$, $H_4(z)=H_3(z^k)$ для заданого параметра k. Визначити ширину смуги пропускання, ширину перехідної смуги, рівень заглушення у смузі непропускання і порівняти їх з параметрами, які одержані в п.2.

Таблиця 4.2. Параметри та функції до лабораторної роботи

№ вар.	a	b	$H_2(z)$	k	$H_{\text{M}}(z)$
1	−0,3125	0,81	-1	2	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4})/5$
2	−0,125	0,81	-1	2	$(1-z^{-1}+z^{-2}-z^{-3}+z^{-4})/5$
3	−0,625	0,81	-1	2	$(1+4z^{-1}+6z^{-2}+4z^{-3}+z^{-4})/16$
4	−0,875	0,81	-1	2	$(1-4z^{-1}+6z^{-2}-4z^{-3}+z^{-4})/16$
5	−0,3125	0,49	$-z^{-1}$	2	$(1+4z^{-1}+6z^{-2}+4z^{-3}+z^{-4})/16$
6	−0,75	0,49	$-z^{-1}$	2	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4})/5$
7	−0,5	0,49	$-z^{-1}$	2	$(-1+3z^{-1}+5z^{-2}+3z^{-3}-z^{-4})/9$
8	−0,25	0,49	$-z^{-1}$	2	$(1+5z^{-1}+10z^{-2}+10z^{-3}+5z^{-4}+z^{-5})/32$
9	−0.8	0,49	z^{-1}	2	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4}+z^{-5}+z^{-6}+z^{-7})/8$
10	0.5	0,49	z^{-1}	2	$(1+z^{-1})(1+z^{-1}+z^{-2}+z^{-3})/8$
11	−0,25	0,25	z^{-1}	3	$(2+5z^{-1}+7z^{-2}+5z^{-3}+2z^{-4})/21$
12	−0,5	0,25	z^{-1}	3	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4}+z^{-5})/6$
13	−0,125	0,25	z^{-1}	3	$(1+z^{-1}+2z^{-2}+z^{-3}+z^{-4})/6$
14	−0,75	0,25	z^{-1}	3	$(1-z^{-1}+z^{-3}-z^{-4})/4$
15	−0,75	0,25	z^{-1}	3	$(1+0.7z^{-1}-0.7z^{-3}-z^{-4})/3.4$
16	−0,625	0,25	z^{-1}	3	$(1-z^{-1}+z^{-2}-z^{-3}+z^{-4}-z^{-5})/6$
17	−0,75	0,25	z^{-1}	3	$(1-0.7z^{-1}+0.7z^{-3}-z^{-4})/3.4$
18	−0,5	0,25	z^{-1}	4	$(1+z^{-1}+z^{-3}+z^{-4})/4$
19	0.25	0,25	z^{-1}	4	$(-1+z^{-1}+z^{-3}-z^{-4})/4$
20	−0,625	0,25	z^{-1}	4	$(1-z^{-1}+z^{-3}-z^{-4})/4$
21	−0,5	0,25	z^{-1}	4	$(-1+2z^{-2}-z^{-4})/4$
22	−0,75	0,25	z^{-1}	4	$(1+2z^{-1}+2z^{-2}+2z^{-3}+z^{-4})/8$
23	−0,6	0,25	z^{-1}	4	$(1-2z^{-1}+2z^{-2}-2z^{-3}+z^{-4})/8$
24	0	0,25	z^{-1}	4	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4}+z^{-5}+z^{-6})/7$
25	−0.1	0,25	z^{-1}	4	$(1-z^{-2}+z^{-4}-z^{-6})/4$
26	−0.15	0,25	z^{-1}	4	$(1+1.4z^{-1}+z^{-2}-z^{-4}-1.4z^{-5}-z^{-6})/6.8$
27	−0,5	0,25	z^{-1}	5	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4})/5$
28	−0,6	0,25	z^{-1}	5	$(-3-2z^{-1}+5z^{-2}+5z^{-3}-2z^{-4}-3z^{-5})/20$
29	−0,75	0,25	z^{-1}	5	$(3-2z^{-1}+5z^{-2}+5z^{-3}-2z^{-4}+3z^{-5})/20$

4. Для заданої функції $H_{\text{M}}(z)$ за допомогою програми на VHDL побудувати графіки $|H_{\text{M}}(\omega)|$, $\arg(H_{\text{M}}(\omega))$, а також графіки $|H(\omega)|$, $\arg(H(\omega))$ для функції (4.14). Проаналізувати ширину смуги пропускання, рівень заглушення у смузі непропускання для різних значень параметра a. Дослідним шляхом визначити діапазон зміни

параметра *a*, у якому забезпечується заглушення не менше 20 дб і такий параметр *a*, при якому досягається максимальне заглушення. Намалювати структуру фільтра.

5. Проаналізувати одержані графіки. Зробити висновки по роботі.

Приклад виконання роботи

Нехай задано $a = -0.5,\ b = 0.64,\ k = 2,\ H_2(z) = z^{-1}$,

$$H_{\text{M}}(z) = \frac{(1+z^{-1})(1-z^{-7})}{7(1-z^{-1})}.$$

Для обчислення передаточних функцій розроблено ряд функцій з використанням пакетів IEEE.MATH_REAL та IEEE.MATH_COMPLEX. Обчислення послідовності z^n виконується з використанням наступних функцій, які перевантажують одна одну

```
--число Z радіусом MAG в степені кута j*fi
function Z(mag,fi:real)  return COMPLEX_POLAR is
begin
   return mag*exp(COMPLEX_TO_POLAR(MATH_CBASE_J)*(fi));
end Z;
--число Z радіусом 1 в степені кута j*fi
function Z(fi:real)  return COMPLEX_POLAR is
begin
   return exp(COMPLEX_TO_POLAR(MATH_CBASE_J)*(fi));
end Z;
```

Наступна функція обчислює $H_1(z)$ за формулою (4.11), але для позитивних степенів z, тобто з урахуванням (4.6). Вона необхідна для сканування простору $|H_1(z)|$ при пошуку нулів та полюсів, коли встановлюється різна величина mag амплітуди та фази fi комплексного вектора z. Також вона використовується для розрахунків АЧХ та ФЧХ.

```
-- обчислює (bZ^2+a(b+1)Z+1)/(Z^2+c(b+1)Z+b)
function Allpass2(b, a: real;  --коефіцієнти
                  mag, fi: real)  -- амплітуда, кут для z
                  return COMPLEX_POLAR is
                  variable tn,td:   COMPLEX_POLAR;
begin
                  td:=COMPLEX_TO_POLAR(COMPLEX'(b,0.0))
                  +a*(b+1.0)*Z(mag,fi) + Z(mag**2,2.0*fi);
                  tn:=COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))
                  +a*(b+1.0)*Z(mag,fi) + b*Z(mag**2,2.0*fi);
                  return tn/td ;
end Allpass2;
```

Наступна функція обчислює $H_1(z^2)$ (4.11) для від'ємних степенів z з одиничною амплітудою, тобто вона обчислює $H_1(\omega)$, де ω = fi . Тому вона використовується лише для розрахунків АЧХ та ФЧХ.

```
function Allpass2x2(b,c:real; fi:real)  return COMPLEX_POLAR is
   variable tn,td:   COMPLEX_POLAR;
begin
            tn:=COMPLEX_TO_POLAR(COMPLEX'(b,0.0))
            +c*(b+1.0)*Z(-2.0*fi) + Z(-4.0*fi);
            td:=COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))
            +c*(b+1.0)*Z(-2.0*fi) + b*Z(-4.0*fi);
            return tn/td ;
end Allpass2x2;
```

Для сканування простору $|H_1(z)|$ використовується наступний фрагмент VHDL-програми.

```
   signal clk:bit;
   signal m,ph:real:=0.0;
   signal a,b,logm,phase,mag:real:=0.0;
begin
   clk<=not clk after 5ns; --генератор синхросигналів
   process(CLK)
            variable p,phas:real:=0.0;
            variable Hz: COMPLEX_POLAR;
   begin
            a<=–0.5;  b<=0.64;    -- параметри H(z)
            if clk='1' and clk'event then
                     phas:= phas+0.001;        -- лічильник фази (частоти)
                     p:=phas*MATH_PI*2.0;    -- нормована фаза
                     m<= trunc(phas)*0.1+0.1; -- амплітуда Z
                     ph<=phas;                 -- сигнал фази
            end if;
            Hz:=Allpass2(b,a,m,p);             --власне H(z)
            mag<=abs(Hz);                      -- амплітуда H(z)
            phase<=Hz.ARG;                     -- фаза H(z)
            logm<=20.0*log10(abs(Hz));         -- H(z) у децибелах
   end process;
```

Після запуску програми на моделювання одержано графік $|H_1(z)|$ (рис. 4.9), у якому на кількох інтервалах, позначених m, розміщено криві $|H_1(m,ph)|$, де m – довжина, ph – нормований кут (фаза, частота) вектора z, причому m = 0.1, 0.2,..., повний оберт вектора z відбувається коли ph – ціле число. Перша половина $|H_1(m,ph)|$ – для m = 0.1,...0.9, а друга половина – для m = 1.0,...1.9. Таким чином, кожен інтервал відповідає замкненій кривій, як на рис. 4.6. Там же аналогічно показано графік arg($H_1(z)$) для m = 0.6,...1.1.

Аналіз графіків на рис. 4.9 показує, що полюси функції $H_1(z)$ мають фазу $\pm 2\pi \cdot ph = \pm 2\pi \cdot 0.1645 = \pm 1.0336$ і знаходяться в точках $r_1 = 0.8\ \angle 1.0336$ та $r_2 = 0.8\ \angle -1.0336$. Нулі функції $H_1(z)$ мають таку саму фазу і дорівнюють $q_1 = 1.3\ \angle 1.0336$, $q_2 = 1.3 \angle -1.0336$. Слід відмітити, що $-\cos(1.0336) = -0.512 \approx a$, $1/0.8 = = 1.25 \approx 1.3$ та $0.8^2 = 0.64 = b$, тобто справджуються відношення (4.12).

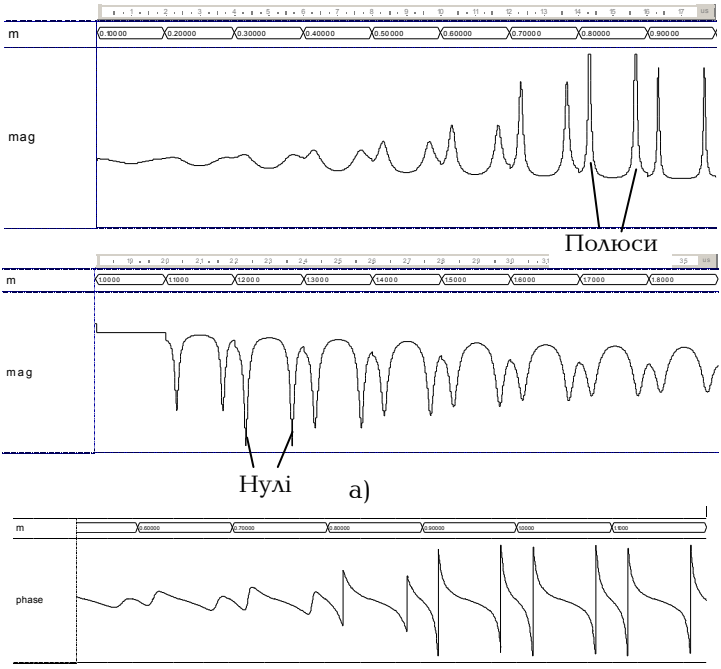


Рис. 4.9. Розгортка функції $|H_1(z)|$ (а) та функції $\arg(H_1(z))$ (б)

Аналіз графіка $\arg(H_1(z))$ показує, що при куті $ph = 0$ фаза дорівнює 0, при поступовому збільшенні ph фаза плавно зменшується. Але якщо ph наближається до положення полюсів, фаза стрімко досягає значення $\pm\pi$.

Для обчислення $|H_3(\omega)|$ у попередній програмі один оператор присвоювання змінній замінюється на наступний оператор, що відповідає формулі (4.13) .

```
Hz:=(Allpass2(b,a,1.0,p)+Z(-1.0*p))/2.0 .
```

Тут $H_2(z) = z^{-1}$ — теж передаточна функція фазового фільтра, тому що $|H_3(z)| = 1$ для усіх z.

Після запуску програми на моделювання одержано графіки $|H_3(\omega)|$ та $\arg(H_3(\omega))$ (рис. 4.10) для частот $\omega = (0, 2\pi)$ або для нормованих частот $ph = (0, 1)$. Як бачимо, додавання передаточної функції $H_2(z)$ створює АЧХ фільтра нижніх частот з частотою зрізу $f_{зр} = 0{,}156 \approx \arg(r_1) = \frac{1}{2\pi}1.0336 = 0{,}164$ частки частоти дискретизації. Виміряний за графіком logm рівень заглушення складає 8.5 дб.

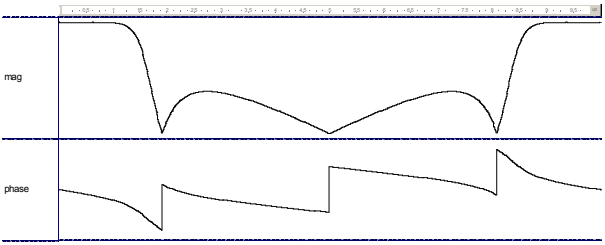


Рис. 4.10. Графіки АЧХ $|H_3(\omega)|$ та ФЧХ $\arg(H_3(\omega))$

Так само, як і у попередньому випадку, графік $|H_4(\omega)|$ (рис.4.11,а) будується за допомогою заміни оператора на наступний оператор

```
Hz:=(Allpass2x2(b,a,p)+Z(-2.0*p))/2.0;
```

Графік $|H_M(\omega)|$ (рис.4.11,б) будується за допомогою оператора

```
Hz:=(COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))+Z(-p))*
    (COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))-Z(-7.0*p))
 /(COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))-Z(-p))/14.0;
```

Видно, що цей графік представляє собою повторений двічі графік на рис.4.10, що є характерним для передаточних функцій з кратними затримками.

Графік результуючої АЧХ $|H_M(\omega)|$ (рис.4.11,в) будується за допомогою оператора

```
Hz:=(Allpass2x2(b,a,p)+Z(-2.0*p))*
     (COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))+Z(-p))*
     (COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))-Z(-7.0*p))
     /(COMPLEX_TO_POLAR(COMPLEX'(1.0,0.0))-Z(-p))/28.0;
```

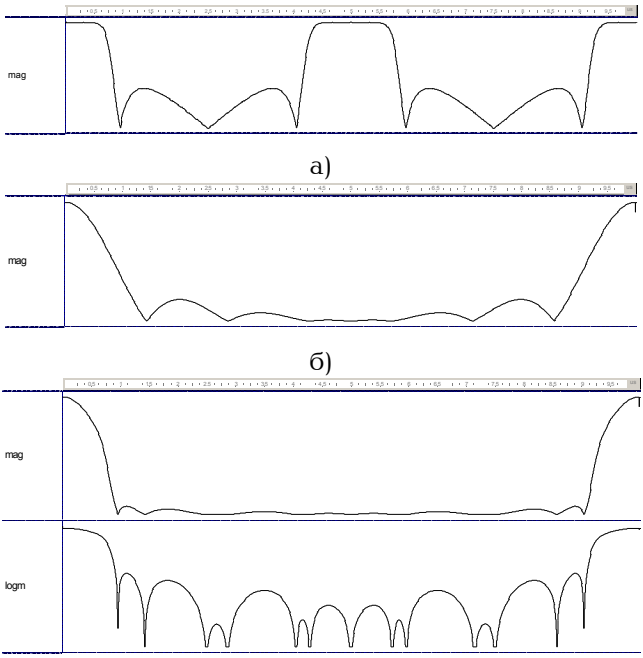


Рис. 4.11. Графік АЧХ $|H_4(\omega)|$ (а), $|H_M(\omega)|$ (б) та результуючої АЧХ $|H(\omega)|$ в лінійному та логарифмічному масштабах

Як свідчить рис. 4.11, функція $H_M(z)$ маскує $H_4(z)$, в результаті чого одержано передаточну функцію $H(z)$ (4.5) фільтра нижніх частот з частотою зрізу $f_{зр} = 0{,}06$ від частоти дискретизації та рівнем заглушення більше 23 дб.

4.3. Лабораторна робота 3
Фільтри зі скінченною імпульсною характеристикою

Мета роботи: одержати знання про фільтри зі скінченною імпульсною характеристикою та навички їх розрахунку, програмування та моделювання за допомогою VHDL.

Теоретичні відомості

Такі лінійні системи, як фільтри, поділяються на СІХ- та НІХ-фільтри (див. підрозділ 2.2). СІХ-фільтри мають ряд наступних переваг:

– СІХ-фільтри завжди стабільні, тому що у них немає полюсів і зворотніх зв'язків, через які відбувається збудження;

– завжди гарантована побудова фільтра з лінійною фазою;

– помилки округлення в СІХ-фільтрах мають значно менший вплив на результат, ніж у НІХ-фільтрах;

– СІХ-фільтри ефективно реалізуються в мікропроцесорах, які мають команди множення з додаванням до акумулятора.

АЧХ $|H(\omega)|$ фільтра задає конкретний коефіцієнт підсилення на певних частотах, а ФЧХ $\varphi(\omega) = \arg(H(\omega))$ – вплив на затримку чи зсув фази сигналу на цих частотах. Функція ***групової затримки*** визначена як

$$T_d(\omega) = -\frac{d\varphi(\omega)}{d\omega}. \qquad (4.15)$$

Фільтр з лінійною фазою має ФЧХ, яку можна виразити формулою:

$$\varphi(\omega) = -a\omega \text{ або } \varphi(\omega) = \pi - a\omega\ . \qquad (4.16)$$

Тому, згідно з (4.15), у фільтра з лінійною фазою групова затримка сигналу є сталою a для усіх частот. Такий фільтр не має перекручування фази сигналу і, відповідно, не втрачає своєї форми (якщо його спектр попадає у смугу пропускання), тому що усі його частотні компоненти затримуються на одну і ту саму затримку.

За формою АЧХ фільтри діляться на фільтри нижніх частот (ФНЧ), високих частот (ФВЧ), смугові фільтри (СФ) та режекторний (смугово-непропускний) фільтр (РФ). Через те, що АЧХ фільтра з реальними коефіцієнтами є симетричною функцією від ω, така АЧХ задається у діапазоні $0 \le \omega \le \pi$.

АЧХ ідеального ФНЧ показана на рис. 4.12,а. Діапазони $0 \le \omega \le \omega_{зр}$ та $\omega > \omega_{зр}$ називаються ***смугами пропускання*** та

непропускання, відповідно, а частота $\omega_{зр}$, яка розділяє ці смуги — ***частотою зрізу***. Ідеальний фільтр нижніх частот у діапазоні $0 \le \omega \le \omega_{зр}$ має АЧХ $|H(\omega)| = 1$, а у діапазоні $\omega > \omega_{зр}$ — $|H(\omega)| = 0$ (рис. 4.12,а).

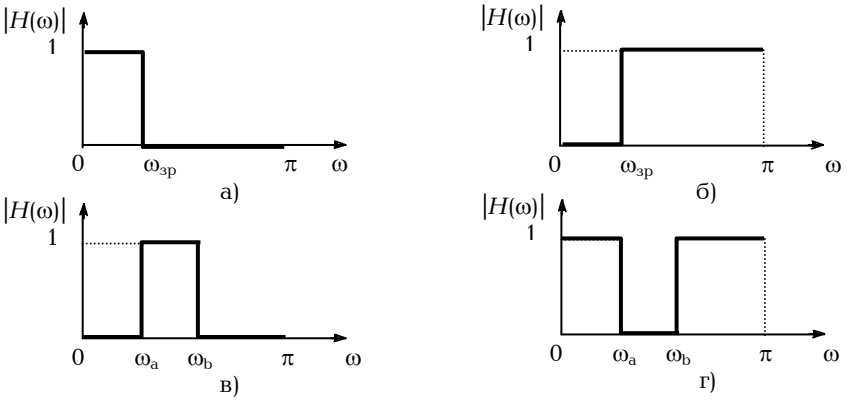


Рис. 4.12. Графіки АЧХ фільтрів нижніх частот (а), верхніх частот (б), смугового (в) та режекторного (г) фільтрів

В АЧХ смугового фільтра частоти ω_a, ω_b називаються нижньою та верхньою частотами зрізу. Такий фільтр пропускає частоти у діапазоні (ω_a, ω_b) і заглушує частоти в інших діапазонах (рис. 4.12,в). Режекторний (смугово-непропускаючий) фільтр — навпаки — заглушує частоти в діапазоні (ω_a, ω_b) (рис. 4.12,г).

У попередній лабораторній роботі приділялась увага фазовому фільтру, у якого $|H(\omega)| = 1$ для усіх частот. Особливим випадком фазового фільтру є ***фільтр Гільберта***, який виконує зсув фази вхідного сигналу на кут $\pi/2$.

Існують багатосмугові фільтри, які мають кілька смуг пропускання та непропускання. Гребінчастий фільтр — це різновид багатосмугового фільтра, який має кілька смуг, що рівновіддалені за частотою, і тому його АЧХ за формою нагадує гребінець.

Насправді неможливо досягнути ідеальної форми АЧХ, як на рис. 4.12. На практиці АЧХ задається вихідними даними смуг пропускання ($0,\omega_п$), непропускання ($\omega_н,\pi$), ***перехідної смуги*** ($\omega_п,\omega_н$) рівнів пульсацій у смузі пропускання δ_1 та у смузі непропускання δ_2 (рис. 4.13).

Ці рівні пульсацій, як правило, задаються у відносних одиницях — децибелах як похибка амплітуди у смузі пропускання A_1 та мінімальний рівень заглушення фільтра A_2:

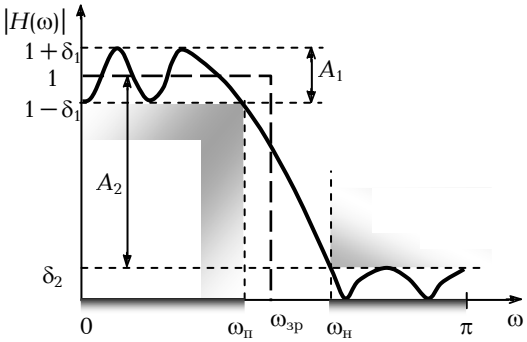


Рис. 4.13. Графік АЧХ реального фільтра нижніх частот

$$A_1 = 20 \log\left(\frac{1+\delta_1}{1-\delta_1}\right), \text{ дБ}; \quad A_2 = 20 \log \delta_2, \text{ дБ}. \tag{4.17}$$

Згідно з (2.5), передаточна функція СІХ-фільтра визначається формулою

$$H(z) = \sum_{r=0}^{M} b_r z^{-r} \tag{4.18}$$

Якщо M – парне число, тобто $N = 2M$, то (4.16) можна переписати як

$$H(z) = \sum_{r=-N}^{N} b_{r+N} z^{-r-N} = z^{-N} \sum_{r=-N}^{N} h_r z^{-r} \ . \tag{4.19}$$

Нехай коефіцієнти h_r мають симетрію: $h_r = h_{-r}$. Тоді АЧХ, згідно з (4.8) та (4.19) обчислюється за формулою

$$H(\omega) = e^{-j\omega N} \sum_{r=-N}^{N} h_r e^{-j\omega r} = e^{-j\omega N} \left(h_0 + 2 \sum_{r=1}^{N} h_r \cos(\omega r) \right) , \tag{4.20}$$

де множник $e^{-j\omega N}$ означає лише затримку на N тактів і може бути усунений з аналізу.

Якщо M – непарне число, то в (4.20) буде відсутній окремий член h_0. Якщо коефіцієнти імпульсної реакції h_r – реальні, то і $H(\omega)$ – реальна функція. ФЧХ такого фільтра дорівнює

$$\varphi(\omega) = \begin{cases} -\omega N & \text{при } H(\omega) \geq 0, \\ \pi - \omega N & \text{при } H(\omega) < 0 \ . \end{cases} \tag{4.21}$$

Отже при умові симетрії імпульсної реакції ФЧХ є лінійною, і через кожні π радіан зміни кута $\varphi(\omega)$ значення АЧХ змінює знак. Навіть коли імпульсна реакція – антисиметрична, тобто коли

$h_0 = 0$, $h_r = -h_{-r}$, ФЧХ є лінійною. Згідно з (4.15), групова затримка такого СІХ-фільтра дорівнює $T\mathrm{d}(\omega) = M/2$, тобто вона відповідає середньому члену послідовності імпульсної реакції.

Властивості симетрії імпульсної реакції, як правило, використовуються для зменшення обсягів обчислень. Згідно з (2.2), СІХ-фільтр обчислюється як

$$y(n) = \sum_{r=0}^{M} b_r x(n-r). \tag{4.22}$$

При урахуванні симетрії коефіцієнтів b_r вираз (4.20) можна переписати як

$$y(n) = \sum_{r=0}^{M/2-1} b_r[x(n-r) + x(n-M+1+r)]. \tag{4.23}$$

Порівнявши (4.22) і (4.23), видно, що кількість множень у останньому випадку зменшена удвічі. Сигнальний граф такого фільтра для випадку непарного M показано на рис. 4.14. При парному M сигнальний граф фільтра має на один помножувач менше.

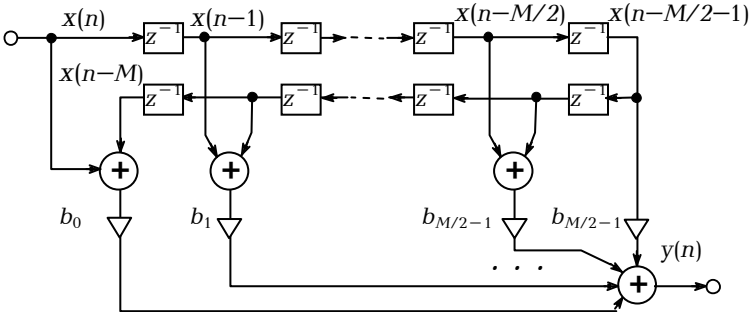


Рис. 4.14. Сигнальний граф СІХ-фільтра з симетричною імпульсною реакцією непарної довжини

СІХ-фільтри з різними АЧХ відрізняються значеннями та кількістю своїх коефіцієнтів b_i. Існують кілька методів синтезу набору цих коефіцієнтів. Найбільш універсальним та ефективним є метод Паркса й Маклеллана, оснований на процедурі поліноміальної оптимізації Ремеза.

Початковими даними для синтезу коефіцієнтів b_i або h_i (4.19) цим методом є їх число M, діапазони частот пропускання $(0,\omega_п)$ та непропускання $(\omega_н,\pi)$, як наприклад, для ФНЧ, похибка амплітуди у смузі пропускання A_1, мінімальний рівень заглушення фільтра A_2 (4.17) або ступінь важливості мінімізації похибок δ_1, δ_2 (рис. 4.12).

Згідно з методом, АЧХ (4.20) представляється як поліном

$$H(\omega) = \sum_{i=0}^{N} a_i \cos^i(\omega). \qquad (4.24)$$

Метод полягає у виборі ряду $N+1$ частот ω_k, складанні та вирішенні системи рівнянь (4.24). Результатами рішень цих рівнянь є коефіцієнти a_i, які на частотах ω_k дають похибку $\delta_1 = 0$ або $\delta_2 = 0$. Після цього обчислюється АЧХ для інших частот, що знаходяться між ω_k та ω_{k+1}. Якщо похибка такої поліноміальної апроксимації не переважає A_1 та A_2, то за коефіцієнтами a_i знаходяться шукані b_i. Якщо ні — то за певним алгоритмом вибираються інші частоти ω_k і процес оптимізації повторюється, починаючи з вирішення системи рівнянь. При використанні замість A_1, A_2 ступенів важливості мінімізації похибок, наприклад, w_1, w_2, процес оптимізації націлений на мінімізацію критерія $w_1 \max(\delta_1) + w_2 \max(\delta_2)$.

Попереднє значення M можна оцінити за емпіричною формулою [4]:

$$M \approx \frac{-10 \lg(\delta_1 \cdot \delta_2)}{2{,}324 \Delta\omega} - 13, \qquad (4.25)$$

де $\Delta\omega = |\omega_н - \omega_п|$ — ширина перехідної смуги. Якщо після обчислення коефіцієнтів фільтру не досягаються гідні рівні A_1, A_2, то збільшується число M або збільшується дистанція між $\omega_п$ та $\omega_н$. Оскільки поведінка АЧХ у перехідній смузі не контролюється за методом, то при надмірних значеннях $|\omega_н - \omega_п|$, можливі небажані перекручення АЧХ.

Метод Паркса й Маклеллана реалізований програмно у багатьох засобах математичних САПР. У даній роботі рекомендується застосувати систему Scilab, яка функціонально еквівалентна системі Matlab, але на відміну від останньої, є безкоштовною. Обчислення коефіцієнтів смугового фільтра виконується за наступними командами.

```
hn=eqfir(M, [0 f1; f2 f3; f4 0.5], [0 1 0], [1 1 1]);
[hm,fr] = frmag(hn, 256);
plot(fr,hm);
hn=
```

За першою командою знаходиться вектор коефіцієнтів h_i, число яких M. Діапазони частот задані двійками 0 f1; f2 f3; f4 0.5, де частоти fi задаються частками повного кута, який нормований до одиниці, тобто розрахункова частота $f = \omega/2\pi$. Числа у векторі [0 1 0] дорівнюють $H(f)$ у відповідних діапазонах частот, а у векторі [1 1 1] — коефіцієнтам важливості w_i. Другий оператор обчислює таблицю з 256 аргументів і значень $H(f)$, а третій — виконує вивід графіка $H(f)$. За четвертим оператором виводиться вектор знайдених коефі-

цієнтів фільтра. Кількість елементів векторів у першому операторі залежить від виду фільтра. Так, у ФНЧ їх два, а у гребінчастого фільтра — подвоєна кількість смуг пропускання.

У попередній лабораторній роботі АЧХ системи обчислювалась на основі алгоритму ЦОС, заданого формулою (2.2), шляхом підстановки (4.9) у формулу відповідної передаточної функції (2.5). Однак такий спосіб не підходить для визначення АЧХ готового фільтра. АЧХ такого фільтра, як правило, відрізняється від АЧХ, яка визначена за формулою, через квантування даних та коефіцієнтів, додавання похибок обчислень, нарешті, через несправність або відмову елементів фільтра. Тому визначення АЧХ фільтра є типовою процедурою його діагностики та тестування.

Для аналізу чи вимірювання АЧХ та ФЧХ на вхід системи, що перевіряється, слід подавати комплексний сигнал $e^{-j\omega n}$ (4.8), який називають ***аналітичним***. Відповідно, система повинна бути спроможною обробляти такий сигнал. Але переважна більшість технічних систем ЦОС призначені для обробки лише реальних сигналів.

Часто використовують простий спосіб перевірки, коли замість аналітичного сигналу подають лише його складову — $\cos(\omega n)$, а АЧХ вимірюють на виході системи, як максимум результуючого сигналу $Re(H(\omega))$, тобто в моменти, коли друга складова — $\sin(\omega n) = 0$. Недолік цього способу — у неточності вимірювання максимуму сигналу.

Більш точний спосіб полягає у одержанні уявної складової $Im(H(\omega))$ після пропускання результату $Re(H(\omega))$ через фільтр Гільберта. Але такий фільтр вносить суттєві перекручення в частотну характеристику при $\omega \to 0$.

Тому для аналізу систем ЦОС слід застосувати сигнальний граф на рис. 4.15. В ньому використовуються два ідентичних екземпляри системи $H(z)$, на які подаються синусна та косинусна складова аналітичного сигналу. Відповідно, з виходів систем знімаються складові аналітичного сигналу відгуку системи. На відміну від аналогових систем, для такого досліду завжди можна одержати дві ідентичні цифрові системи.

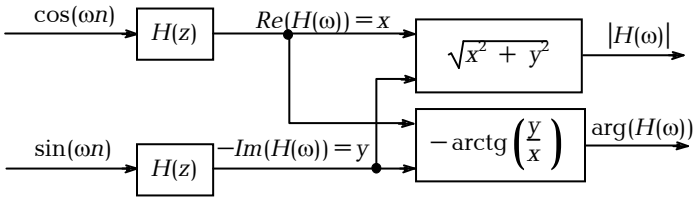


Рис. 4.15. Вимірювання АЧХ, ФЧХ реальної системи $H(z)$

Переважна більшість СІХ-фільтрів обчислюється в комп'ютерах у арифметиці цілих чисел. При програмуванні такого фільтру синтезують набір коефіцієнтів, що задовольняють задані вимоги і представлені з плаваючою комою. Потім вибирають кількість бітів квантування коефіцієнтів $n_к$, вхідних даних $n_х$ та результатів $n_у$.

Як правило, $n_х, n_у \geq \log_2 10 \cdot D/20$, де D – динамічний діапазон зміни сигналу, dB. Тобто на кожні 6 децибел динамічного діапазону припадає не менше одного розряду. Коефіцієнти масштабують та округлюють, так що цілі коефіцієнти дорівнюють

$$b'_i =]2^{n_х} \cdot b_i + 0.5[\ . \qquad (4.26)$$

Результати фільтра обчислюють за формулою (4.22) чи (4.23) з вибором такої розрядності суми, щоб не виникало переповнення. Причому розрядність добутку дорівнює $n_д = n_к + n_х$, а розрядність суматора повинна бути не меншою за $n_с = \log_2 S + n_к + n_х$, де S – теоретично можливий максимальний результат формули (4.22). Нескладно довести, що S дорівнює сумі модулів усіх коефіцієнтів фільтра, тобто

$$n_с = \log_2 \left(\sum_{i=0}^{M} |b_i| \right) \ + \ n_к + n_х. \qquad (4.27)$$

Оскільки $n_с$ може бути доволі великим числом, а вірогідність досягнення результатом (4.22) максимального значення досить мала, то на практиці $n_с$ вибирають дещо меншим, а додавання в (4.22) виконують за алгоритмом накопичення з ***насиченням***. За цим алгоритмом, якщо виникає переповнення суми, то результат замінюється максимальним числом розрядності $n_с$ з відповідним знаком. Результат фільтра $y(n)$ береться як старші n_y розрядів суми (4.22) з відкиданням молодших розрядів, тобто з усіканням.

Округлення коефіцієнтів та усікання результату незворотньо перекручують АЧХ фільтра. Тому слід обчислити АЧХ та ФЧХ за сигнальним графом на рис. 4.15 і порівняти їх з початковими характеристиками. Якщо характеристики фільтра не задовольняють задані вимоги, то слід збільшити значення $n_к$ та $n_с$, можливо – M і повторити цикл синтезу коефіцієнтів. Іноді вдається покращити АЧХ, завдяки корекції молодших розрядів коефіцієнтів b'_i.

Завдання для роботи

1. Синтезувати за допомогою системи Scilab набір коефіцієнтів СІХ-фільтра з симетричною імпульсною характеристикою, який задовольняє задані параметри АЧХ. Параметри фільтра вибрати з табл. 4.3 за варіантом. Порядок фільтра визначити за (4.25), але не більше 60. Показати на графіку АЧХ, що фільтр задовольняє завдання.

2. Написати програму моделі СІХ-фільтра з синтезованим набором коефіцієнтів, що виконує обчислення з плаваючою комою. Ввімкнути модель у стенд для випробувань, згідно з рис. 4.15 та одержати графіки АЧХ і ФЧХ фільтра. Порівняти графік АЧХ з графіком, який одержано в п. 1.

3. Представити коефіцієнти фільтра цілими числами згідно з (4.24). Написати програму моделі СІХ-фільтра з синтезованим набором коефіцієнтів, що виконує обчислення з цілими числами. Перевірити відношення (4.27), яке не повинне бути більшим за 32. Ввімкнути модель у стенд для випробувань, згідно з рис. 4.15 та одержати графіки АЧХ і ФЧХ фільтра. Порівняти графіки з графіками, які одержано в п. 2.

4. Підключити фільтр до виходу генератора сигналів, який розглядався у першій лабораторній роботі і одержати графік результуючого сигналу. Пояснити, чому він має перекручену форму.

5. Проаналізувати одержані графіки. Зробити висновки по роботі.

Для реалізації стенда для випробувань доцільно використати готові віртуальні модулі FilterTB, які додаються до роботи. Наприклад, модуль FilterTB_r призначений для тестування модулів фільтрів порти яких пропускають сигнали з плаваючою комою. Такий модуль має наступний інтерфейс.

```
entity FilterTB is
 generic(fsampl:integer := 1000;
        fstrt: integer:=0;
        deltaf:integer:=20;
        maxdelay:integer:=100;
        slowdown:integer:=3;
        magnitude:real:=1000.0);
 port( CLK    : in STD_LOGIC;
        RST    : in STD_LOGIC;
        START  : in STD_LOGIC;
        RERSP  : in real;
        IMRSP  : in real;
        REO    : out real;
        IMO    : out real;
        FREQ   : out integer;
        MAGN   : out real;
        LOGMAGN: out real;
        PHASE  : out real;
        ENA    : inout STD_LOGIC);
end FilterTB;
```

Аналогічний інтерфейс мають модулі FilterTB_I, FilterTB_V для тестування фільтрів сигналами, що представляються цілими числами та векторами бітів. Призначення настроювальних констант і портів роз'яснено у таблиці 4.4.

Таблиця 4.3. Параметри фільтра до лабораторної роботи

№ вар.	Вид фільтра	A_1,дБ	A_2,дБ	$f_п$,$f_{п1}$	$f_н$,$f_{н1}$	$f_{п2}$	$f_{н2}$	$n_х$, $n_у$	$n_к$
1	ФНЧ	0,1	60	0,3	0,42			16	12
2	ФНЧ	0,2	57	0,25	0,4			16	12
3	ФНЧ	0,4	54	0,25	0,35			16	12
4	ФНЧ	1	51	0,2	0,4			15	11
5	ФНЧ	2	48	0,2	0,3			14	10
6	ФНЧ	3	45	0,15	0,25			13	9
7	ФНЧ	4	42	0,15	0,2			12	8
8	ФВЧ	0,1	60	0,42	0,3			16	12
9	ФВЧ	0,2	57	0,4	0,25			16	12
10	ФВЧ	0,4	54	0,35	0,25			16	12
11	ФВЧ	1	51	0,4	0,2			15	11
12	ФВЧ	2	48	0,3	0,2			14	10
13	ФВЧ	3	45	0,25	0,15			13	9
14	ФВЧ	4	42	0,2	0,1			12	8
15	СФ	0,5	51	0,3	0,2	0,35	0,45	16	12
16	СФ	1	48	0,25	0,15	0,35	0,45	16	12
17	СФ	2	45	0,25	0,15	0,3	0,4	16	12
18	СФ	3	42	0,2	0,1	0,3	0,4	15	11
19	СФ	4	51	0,2	0,1	0,3	0,4	14	10
20	СФ	5	48	0,15	0,05	0,25	0,35	13	9
21	СФ	6	45	0,1	0,03	0,2	0,28	12	8
22	РФ	0,5	51	0,3	0,39	0.49	0,41	16	12
23	РФ	1	48	0,25	0,35	0.49	0,40	16	12
24	РФ	2	45	0,25	0,35	0.45	0,38	16	12
25	РФ	3	42	0,2	0,3	0.49	0,4	15	11
26	РФ	4	51	0,2	0,3	0.45	0,35	14	10
27	РФ	5	48	0,15	0,25	0.40	0,30	13	9
28	РФ	6	45	0,1	0,2	0.4	0,3	12	8
29	РФ	0,5	42	0,1	0,15	0.35	0,25	12	8

Таблиця 4.4. Віртуальний модуль для випробувань фільтрів

fsampl	Частота дискретизації f_o, наприклад 1000 кГц
fstrt	Початкова частота діапазону аналізу
deltaf	Приріст частоти d, так що через k кроків генератор видаватиме частоту $f_o + k \cdot d$
maxdelay	Затримка у кількості відліків від початку генерації частоти, після якої виконується оцінка результатів фільтрації. Має бути більшою за подвоєну максимальну групову затримку фільтра, який тестується.
slowdown	Коефіцієнт зменшення швидкості фільтра відносно тактової частоти. Якщо вхідні відліки поступають у кожному такті, то slowdown=1, якщо вони приходять у кожному парному такті, то slowdown=2 і т.д.
magnitude	Амплітуда синусного і косинусного сигналів, що генеруються.
REO,IMO	Синусний та косинусний сигнали, що генеруються
RERSP, IMRSP	Сигнали з виходів фільтрів, які є відгуками на синусний та косинусний сигнали, відповідно
FREQ	Код частоти сигналів, що генеруються, який дорівнює $f_o + k \cdot d$
MAGN	Обчислена амплітуда комплексного сигналу RERSP, IMRSP на частоті FREQ
LOGMAGN	Обчислена амплітуда комплексного сигналу RERSP, IMRSP на частоті FREQ у логарифмічному масштабі, тобто у децибелах, причому сигнал з амплітудою magnitude приймається за 0 дБ.
PHASE	Обчислена фаза комплексного сигналу RERSP, IMRSP на частоті FREQ, представлена у діапазоні $\pm\pi$
RST,START	Сигнали початкового встановлення
ENA	Сигнал дозволу прийому RERSP, IMRSP при slowdown >1

Приклад виконання роботи

Нехай задано: ФНЧ, $A_1 = 1$ дб, $A_2 = 48$ дб, $f_п = 0.21$, $f_н = 0.29$, $n_х = 12$, $n_к = 10$.

Знайдено $\Delta\omega = 2\pi(0.29 - 0.21) = 0.5027$. Вирішивши рівняння (4.17), знайдено $\delta_1 = 0.05$, $\delta_2 = 0.004$. Після підстановки цих значень в (4.25) одержано $M = 21$.

Сформовано команду для Scilab щоб знайти набір коефіцієнтів

```
hn=eqfir(21, [0 0.21; 0.29 0.5], [1 0], [4 50]);
```

де вагові коефіцієнти [4 50] − відповідають відношенню δ_1 та δ_2.

Після невдалої спроби одержати коефіцієнти, що задовольняють вимоги, було збільшено кількість коефіцієнтів $M = 22$.

Розроблено наступний модуль СІХ-фільтра, в якому використані знайдені коефіцієнти фільтра.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity FIR_filter is              -- СІХ-фільтр парного порядку
    generic(nn:natural:=22); -- порядок фільтра
    port(CLK : in STD_LOGIC;
            RST : in STD_LOGIC;
            X : in REAL;        -- вхід фільтра
            Y : out REAL);    -- вихід фільтра
end FIR_filter;
architecture model of FIR_filter is
    constant m: natural:=nn/2;  -- кількість коефіцієнтів
    type Tarr is array (0 to m-1) of real;
    constant b: Tarr:= (                -- набір коефіцієнтів
    0.0099895,    0.0238865,    0.0071111,  - 0.0255226,  - 0.0145117,
    0.0407322,    0.0307317,  - 0.0684982,  - 0.0734177,    0.1589393,
    0.4370358);
    signal xf,xb:Tarr;   -- ланцюжки затримок вхідного сигнала
begin
    DELAY:process(CLK,RST) begin
            if RST='1' then
                    xf<=(others=>0.0);
                    xb<=(others=>0.0);
            elsif CLK='1' and CLK'event then
                    xf<=X & xf(0 to m-2);
                    xb<=xb(1 to m-1) & xf(m-1);
            end if;
    end process;

    ADD_MUL:process(xf,xb)
            variable acc:real;
    begin
            acc:=0.0;
            for i in 0 to m-1 loop
                    acc:=acc + b(i)*(xf(i)+xb(i));
            end loop;
            Y<=acc;
    end process;
end model;
```

Модуль фільтра був вставлений у сенд для випробувань, що показаний на рис. 4.16. Задопомогою цього стенда було побудовано графіки АЧХ та ФЧХ, які показано на рис.4.17.

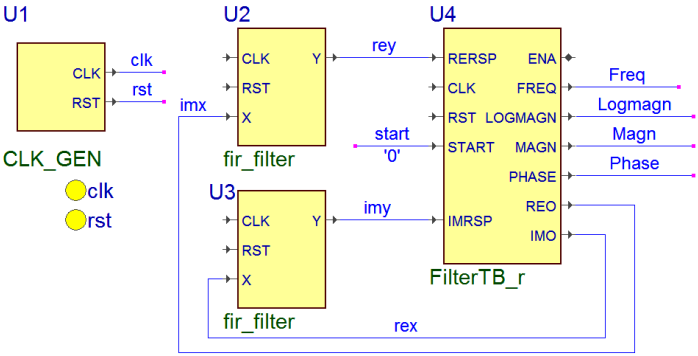


Рис.4.16. Стенд для іспитів СІХ-фільтра

За допомогою фільтра було профільтровано сигнал з генератора, який було побудовано в першійлабораторній роботі. Результат фільтрації показано на рис. 4.18.

Як видно, фільтр типу ФНЧ дещо згладжує різкі переходи сигналу. Невеликі коливання в межах різких переходів пов'язані з недостатнім притисненням сигналу з частотою, яка близька до 0,5 частоти дискретизації.

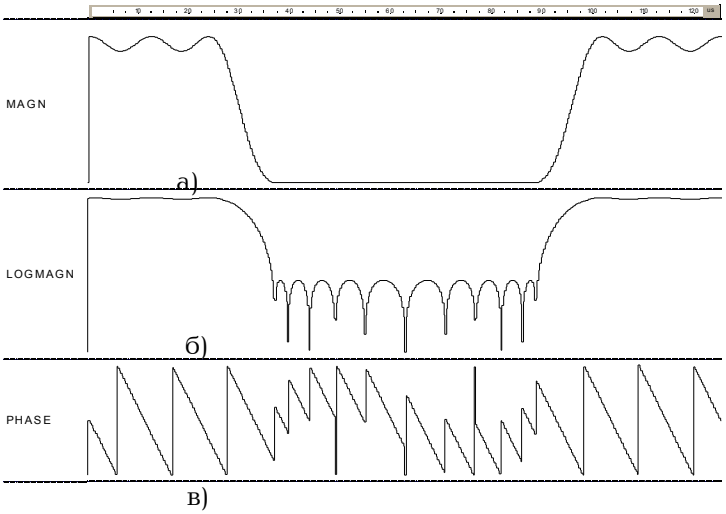


Рис.4.17. АЧХ (а), АЧХ у логарифмічному масштабі (б) та ФЧХ (в) синтезованого СІХ-фільтра

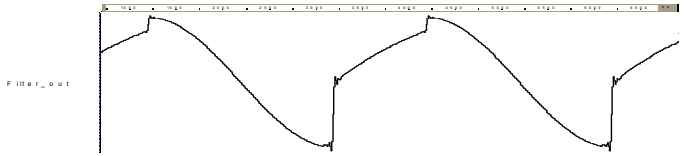


Рис. 4.18. Результат фільтрації сигналу СІХ-фільтром

Далі модель FIR_filter, яка працює з плаваючою комою, була перероблена для обчислень з цілими числами у модель FIR_filter_I. Для цього, крім заміни типів сигналів та змінних, були змасштабовані коефіцієнти згідно з (4.24):

```
constant b: Tarri:= ( 10, 24, 7, -26, -15, 42, 31, -70, -75, 162, 448);
```

Також результат був поділений на масштабний коефіцієнт:

```
Y<=acc/2**10;
```

Після включення в стенд для випробувань і подання тестової послідовності з амплітудою 2048, що відповідає розрядності n_x, була одержана АЧХ, як на рис. 4.16.

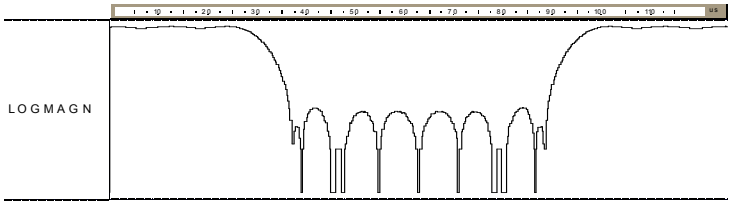


Рис.4.19. АЧХ у логарифмічному масштабі для СІХ-фільтра з $n_x = 12$ і $n_к = 10$

Порівняння АЧХ на рис. 4.17 та 4.19 показує, що округлення коефіцієнтів до 10 розрядів погіршило A_2 (43 дб замість 48 дб) і не змінило інших показників. Отже, синтез СІХ-фільтра пройшов успішно. Для підвищення його якості слід збільшити розрядність коефіцієнтів.

4.4. Лабораторна робота 4
Фільтри з нескінченною імпульсною характеристикою

Мета роботи: одержати знання про фільтри з нескінченною імпульсною характеристикою та навички їх розрахунку, програмування та моделювання за допомогою VHDL.

Теоретичні відомості

НІХ-фільтри (див. підрозділ 2.2) мають наступні переваги над СІХ-фільтрами:

− у кілька разів менші обчислювальні витрати та об′єм пам′яті;

− менша групова затримка сигналу;

− має пряму аналогію з фізичними моделями та аналоговими фільтрами.

До недоліків НІХ-фільтрів належать нелінійність ФЧХ, необхідність забезпечення стабільності обчислень і як результат − підвищена точність розрахунків. Наявність зворотніх зв′язків в ГСПД НІХ-фільтра обмежує можливості розпаралелювання обчислень.

Реалізація НІХ-фільтра − це обчислення різницевого рівняння (2.2). Якщо одиничний імпульс подати на вхід x фільтра, то на його виході y одержимо його імпульсну реакцію $h(n)$ (2.3). Через те, що результат $y(n)$ залежить від попередніх результатів $y(n-k)$, імпульсна реакція виходить нескінченною. Це стало причиною назви таких фільтрів.

На відміну від СІХ-фільтрів, НІХ-фільтри мають крім нулів також полюси передаточної функції. Наявність пари комплексних полюсів $r_1 = \mathrm{Re}(r) + j\mathrm{Im}(r)$, $r_2 = \mathrm{Re}(r) - j\mathrm{Im}(r)$ задає резонансні властивості фільтра на частоті, яка визначається фазовою координатою полюсів $\varphi = \arg(r)$. Причому чим ближче полюс до одиничного кола, тобто чим більше $|r|$ наближається до 1 − тим значніше проявляється резонанс (див. рис. 4.5). При цьому проміжні результати фільтра можуть зростати до θ разів у порівнянні з амплітудою вхідного сигналу, де θ − ***коефіцієнт добротності***. У крайньому випадку, коли $|r| = 1$, $\theta \to \infty$, тобто фільтр стає нестійким. Отже, нерівність $|r| < 1$ є ***критерієм стійкості*** НІХ-фільтра, якого обов′язково слід дотримуватись.

У попередній лабораторній роботі можна було пересвідчитись, що якість фільтрації зростає зі збільшенням довжини імпульсної реакції. Отже, добравши якісні коефіцієнти для (2.2) можна одер-

жати НІХ-фільтр, ефективність якого буде суттєво вища ніж у СІХ-фільтра.

Пошук коефіцієнтів — це задача апроксимації передаточної функції (2.5), яка повинна наближатись до заданої форми на рис. 4.12. За методом апроксимації НІХ-фільтри розділяють на фільтри Бесселя, Батерворта, Чебишева та еліптичні. Ці методи були розроблені для синтезу аналогових фільтрів і були адаптовані до розробки цифрових фільтрів. Фільтр Бесселя має гладку АЧХ у смугах пропускання та непропускання і є аналогом ланцюгового RC-фільтра. Фільтр Батерворта має незначну пульсацію у смузі пропускання. Фільтр Чебишева першого роду має мінімізовані рівновеликі пульсації у смузі пропускання і гладку АЧХ в смузі непропускання, а фільтр Чебишева другого роду — навпаки — мінімізовані рівновеликі пульсації у смузі непропускання.

Еліптичний фільтр має мінімізовані рівновеликі пульсації як у смузі пропускання, так і непропускання. Серед усіх фільтрів він має мінімальну кількість коефіцієнтів. Але він має найбільше перекручування фази і, відповідно, нерівномірну групову затримку (4.15). Як результат, сигнал, який профільтровано еліптичним фільтром, суттєво змінює форму.

Як правило, НІХ-фільтр роблять багатоланковим, тобто його передаточна функція визначається формулою (2.6), а його структура — як на рис. 3.2. Такий спосіб реалізації фільтра дає змогу дещо зменшити вимоги до точності обчислень, тому що загальний коефіцієнт добротності фільтра θ розділяється між ступенями. Крім того, кожна наступна ланка фільтрує похибки обчислень попередніх ланок і результуючий шум округлення стає меншим, ніж у фільтрів, реалізованих іншим способом.

k-та ланка фільтра може бути пешого чи другого порядку з передаточною функцією:

$$H_1(z) = \frac{b_{0,k} + b_{1,k}z^{-1}}{1 + a_{1,k}z^{-1}} \; ; \; H_2(z) = \frac{b_{0,k} + b_{1,k}z^{-1} + b_{2,k}z^{-2}}{1 + a_{1,k}z^{-1} + a_{2,k}z^{-2}} \; .$$

Для зменшення кількості множень коефіцієнти чисельників нормалізують і чисельник стає рівним

$$1 + b_{1,k}z^{-1}/b_{0,k} + b_{2,k}z^{-2}/b_{0,k}. \qquad (4.28)$$

При цьому результат фільтрації множать на загальний коефіцієнт підсилення $b = b_{0,1} \cdot b_{0,2} \cdot \ldots$.

Ланка першого порядку має реальні нуль та полюс. Ланка другого порядку має, як правило, комплексні полюси. Найчастіше у цього фільтра нулі реальні, але у еліптичного фільтра та фільтра Чебишева другого роду — нулі завжди комплексні. Якщо фільтр — ФНЧ з реальними нулями $q_{1,2} = 1$, то чисельник $H_2(z)$ дорівнює

$1+2z^{-1}+z^{-2}$, у фільтра високих частот нулі $q_{1,2} = -1$ і чисельник дорівнює $1-2z^{-1}+z^{-2}$. У смугового фільтра нулі $q_1 = -1$, $q_2 = 1$, які відповідають чисельнику $1-z^{-2}$. Якщо смуговий фільтр – багатоланковий то у однієї ланки може бути чисельник $H_2(z)$ як у ФНЧ, а у іншої – як у ФВЧ.

Згідно з вимогою критерія стійкості НІХ-фільтра, коефіцієнти знаменника $H_2(z)$ повинні задовольняти нерівність

$$1 > a_{2,k} > |a_{1,k}| - 1. \tag{4.29}$$

Умовою того, що корені – комплексні, є

$$|0{,}5a_{1,k}/\sqrt{a_{2,k}}\ | < 1. \tag{4.30}$$

Для пошуку коеффіцієнтів можна використати наступну функцію системи Scilab:

```
[cells,fact,zzeros,zpoles]=eqiir(ftype,approx,[om1,om2,om3,om4],deltap,deltas);
```

де ftype – тип фільтра: 'lp' – ФНЧ, 'hp' – ФВЧ, 'bp' – СФ, 'sb' – РФ; approx – вид фільтру: 'butt' – Баттерворта, 'cheb1' – Чебишева першого типу, 'cheb2' – Чебишева другого типу, 'ellip' – еліптичний; 0 ≤ om1 ≤ ≤om2 ≤ om3 ≤ om4 ≤ pi – частоти початку і кінця смуг пропускання чи непропускання у радіанах; deltap, deltas – δ_1,δ_2 – коливання АЧХ у смугах пропускання та непропускання (рис. 4.12); cells – вказує, що слід знайти коефіцієнти багатоланкового фільтру; fact – коефіцієнт нормалізації, тобто загальний коефіцієнт підсилення b; zzeros, zpoles – множини нулів та полюсів, відповідно. Крім того, ця функція виводить формулу передаточної функції у формі (2.6) і (4.28).

Завдання для роботи

1. Синтезувати за допомогою системи Scilab набір коефіцієнтів НІХ-фільтра, який задовольняє задані параметри АЧХ. Параметри фільтра вибрати з табл. 4.3 за варіантом. Вид апроксимації – Батерворта (0), Чебишева першого роду (1), Чебишева другого роду (2) еліптичного фільтра (3) вибрати за формулою N mod 4, де N – номер варіанта. Одержані набори нулів та полюсів відобразити на графіку z-площини.

2. Розробити модель ланки фільтра другого порядку, описану мовою VHDL. Коефіцієнти фільтра повинні задаватись як настроювальні константи.

3. Розробити модель багатоланкового НІХ-фільтра структурним стилем. Настроїти ланки моделі згідно з набором коефіцієнтів фільтра. Виміряти АЧХ у лінійному та логарифмічному масштабі, а також ФЧХ так само, як це робилось у лабораторній роботі 3.

Виміряти внутрішні сигнали ланок і оцінити величину коефіцієнта добротності ланок θ_i. Порівняти загальний коефіцієнт добротності θ фільтра з величиною, оберненою до fact.

4. Підключити фільтр до виходу генератора сигналів, який розглядався у першій лабораторній роботі і одержати графік результуючого сигналу. Пояснити його форму.

5. Округлити коефіцієнти фільтра до двох розрядів після коми і знов виміряти АЧХ. Пояснити її форму.

6. В одній з ланок змінити коефіцієнти так, щоб не справджувалась нерівність (4.29) і щоб $|r| \approx 1$. Виміряти вихідний сигнал і дослідити ефект збудження.

7. Проаналізувати одержані графіки. Зробити висновки по роботі.

Приклад виконання роботи

Нехай задано: СФ, $A_1 = 0.3$ дб, $A_2 = 54$ дб, $f_{н1} = 0.05$ $f_{п1} = 0.11$, $f_{п2} = 0.14$, $f_{н2} = 0.2$.

Вирішивши рівняння (4.17), знайдено $\delta_1 = 0.02$, $\delta_2 = 0.002$.

Сформовано команду для Scilab, щоб знайти набір коефіцієнтів

```
[cells,fact,zzeros,zpoles]=
        eqiir('bp','cheb1',[0.1*%pi,0.22*%pi,0.28*%pi, 0.4*%pi],0.02,0.002)
```

За цією командою одержано формулу передаточної функції:

$$\frac{1 + 2z + z^2}{0.9487295 - 1.2250177z + z^2} \quad \frac{1 + 2z + z^2}{0.8857866 - 1.2789722z + z^2}$$

$$\frac{1 - 2z + z^2}{0.8948900 - 1.401471z + z^2} \quad \frac{1 - 2z + z^2}{0.9582494 - 1.5204099z + z^2},$$

причому fact = 0.0000507.

Графік нулів та полюсів (рис. 4.20) одержано за командами Scilab:

```
ze =[- 1.,  - 1.,  - 1.,  - 1.,    1.,    1.,    1. ,   1. ];
pol =[0.6125089 + %i*0.7573390, 0.6125089 - %i*0.7573390,
      0.6394861 + %i*0.6905390, 0.6394861 - %i*0.6905390,
      0.7007355 + %i*0.6354996, 0.7007355 - %i*0.6354996,
      0.7602049 + %i*0.6167154, 0.7602049 - %i*0.6167154];
n= [0:0.01:2*%pi]';
plot(cos(n),sin(n),'k', real(ze),imag(ze),'ko',real(pol),imag(pol),'kx')
```

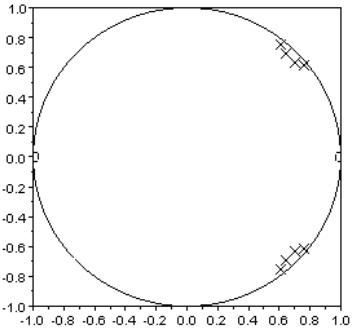


Рис. 4.20. Графік розміщення нулів та полюсів

Опис VHDL-моделі ланки фільтра приведено у розділі 2.3. Структура фільтра показана на рис. 3.2.

Одержані графіки АЧХ та ФЧХ показані на рис. 4.21 при встановленій затримці maxdelay=400.

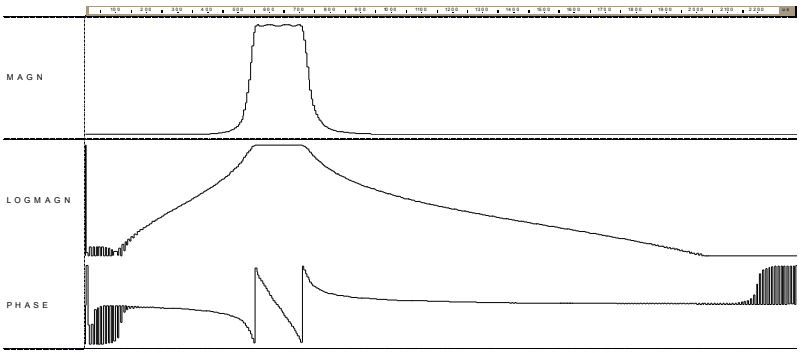


Рис. 4.21. Графіки АЧХ у лінійному та логарифмічному масштабі та ФЧХ НІХ-фільтра

Аналіз графіків на рис. 4.21 показує, що одержані характеристики задовольняють задані, причому $f_{н1} = 0.05$, $f_{п1} = 0.109$, $f_{п2} = 0.141$, $A_2 = 58$ дб на межі смуги непропускання і плавно спадає до рівня 100 дБ.

Шляхом вимірювання амплітуди максимальних сигналів на входах і виходах ланок та обчислення їх відношень визначені добротності ланок $\theta_1 = 40$, $\theta_2 = 13$, $\theta_3 = 7$, $\theta_4 = 5.4$. Загальна доброт-

ність $\theta = \theta_1 \cdot \theta_2 \cdot \theta_3 \cdot \theta_4 = 19656$. Вона доволі точно збігається з величиною 1/fact = 19723.

На рис. 4.22 показаний відгук фільтра на тестовий сигнал, такий як на рис. 4.3. Видно, що фільтр виділив частоти зі спектром, який має ширину смуги пропускання. Сплески сигналу відповідають різким переходам у тестовому сигналі, тобто тим місцям, де зосереджені високі частоти.

Після округлення коефіцієнтів до двох десяткових знаків після коми одержано АЧХ, як на рис. 4.23.

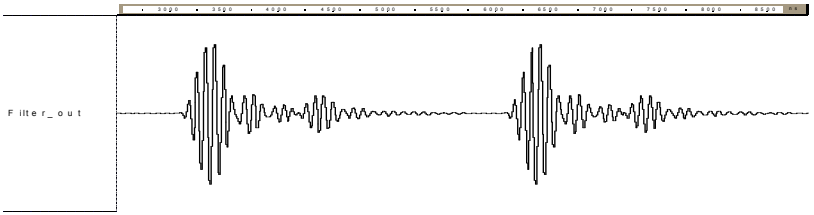


Рис. 4.22. Графік відгуку НІХ-фільтра на тестовий сигнал.

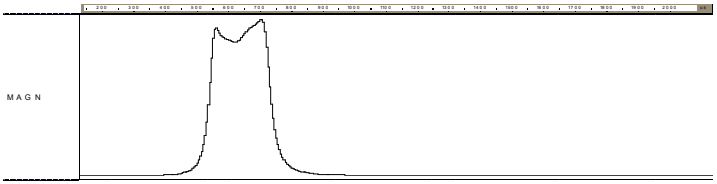


Рис. 4.23. Графік АЧХ НІХ-фільтра з округленими коефіцієнтами

Як видно з рис. 4.23, неточне представлення коефіцієнтів привело до спотворення АЧХ у смузі пропускання, через те, що передаточна функція була сформована з чебишевською оптимізацією пульсацій саме у смузі пропускання.

4.5. Лабораторна робота 5
Швидке перетворення Фур'є

Мета роботи: одержати знання про методи спектрального оцінювання, алгоритм ШПФ та навички їх програмування та моделювання за допомогою VHDL.

Теоретичні відомості

У розділі 2.2 розглядалось z-перетворення дискретних сигналів $x(n)$, що дає представлення сигналу через суму власних функцій. Згідно з (2.4), при $z = e^{j\omega} = e^{j2\pi f}$ z-перетворення представляє собою дискретне перетворення Фур'є:

$$X(f) = T\sum_{n=-\infty}^{\infty} x(n)e^{-j2\pi fnT}, \tag{4.31}$$

де T – період дискретизації сигналу $x(n)$, $X(f)$ – спектр сигналу. Слід відмітити, що спектр $X(f)$ – періодична безперервна функція, період якої дорівнює частоті дискретизації $F_s = 1/T$. Для неї існує обернена функція – зворотнє перетворення Фур'є

$$x(n) = \frac{1}{T}\int_0^{F_s} X(f)e^{j2\pi fn}\,\mathrm{d}f\,. \tag{4.32}$$

Вирази (4.29) і (4.30) для кінечної послідовності $x(n)$ довжиною N відліків виглядають як зрізані ряди Фур'є

$$F(x) = X(k) = T\sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}, \tag{4.33}$$

$$F^{-1}(X) = x(n) = \frac{1}{NT}\sum_{k=0}^{N-1} X(k)e^{j2\pi kn/N}. \tag{4.34}$$

Власне, (4.33) і (4.34) називаються в ЦОС як пряме і зворотнє ***дискретне перетворення Фур'є*** (ДПФ). Обидві послідовності $x(n)$ та $X(k)$ є періодичними з періодом N. В них параметр T введено для можливості обчислення каліброваної спектральної щільності енергії

$$E(k) = X^2(k) = \left(T\sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}\right)^2;$$

та спектральної щільності потужності

$$\tilde{S}(k) = \frac{E(k)}{NT} = \frac{T}{N}\left(\sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}\right)^2. \tag{4.35}$$

Функцію (4.34) часто використовують для вимірювання спектральних властивостей сигналу. Метод оцінки спектру на основі обчислення ДПФ виборок сигналу довжиною N називається методом періодограм, а одержаний спектр — періодограмою.

Періодограма $\tilde{S}(k)$ дорівнює спектральній щільності потужності періодичного сигналу з періодом N. Для дійсного сигналу це лише наближена оцінка спектральної щільності $S(k)$. При цьому, якщо $x(n)$ називають відліками, то $X(k)$ чи $S(k)$ — називають ***бінами***. Безпосереднє обчислення (4.35) означає, що у доволі довгому дискретному сигналі $x(n)$, так званим, прямокутним ***часовим вікном*** $w(n)$ вирізається фрагмент довжиною N, який вважається періодичним:

$$x'(n) = x(n)\cdot w(n),$$

причому $w(n) = 1$ при $0 \le n < N$ та $w(n) = 0$ — при інших n. Як результат, сигнал, що аналізується, стає несхожим на початковий сигнал. В ньому з'являються розриви в точках iN. Отже, результуючий спектр є згорткою справжнього спектру та спектру вікна

$$F(x'(n)) = F(x(n)) \star F(w(n)) \ . \qquad (4.36)$$

Через це у спектральної щільності $\tilde{S}(k)$ крім основної пелюстки складової спектру з'являються бокові пелюстки. Поява цих пелюсток називається розтіканням або просочуванням потужності і призводить до зміщення оцінок спектру сусідніх бінів відностно біна основної пелюстки.

Важливим параметром методу спектрального оцінювання є його роздільна здатність, тобто здатність роздільно вимірювати спектральні відгуки двох синусоїдальних сигналів, які близькі за амплітудою та частотою. Вважається, що спектральні піки розділені, якщо провал між ними складає величину не менше 3дБ.

Згідно з (4.33), інтервал часу дослідження сигналу дорівнює $T_e = NT$. Відповідно, відстань між бінами $B_e = 1/NT$ характеризує максимальну роздільну здатність. Звідси видно, що спектральна роздільна здатність B_e в герцах обернено пропорційна інтервалу часу сигналу T_e, що обробляється в секундах, тобто $T_e B_e = 1$.

Для зменшення впливу просочування потужності використовуються вікна спеціальної форми. Це еквівалентно тому, що ввімкнуто фільтр, який фільтрує спектр за формулою (4.36) і своїм впливом мінімально спотворює спектр. Усі вікна симетричні відносно центру і мають спадання підсилення на своїх краях, що призводить принаймні до минімізації розривів у сигналі, що аналізується.

При застосуванні часового вікна роздільна здатність зменшується в α разів у порівнянні з еквівалентною шириною смуги B_e. Отже, при обробці випадкових сигналів слід розглядати ефективну

статистичну ширину смуги $B_S = \alpha B_e$. Вона дорівнює ширині смуги основної пелюстки на рівні 3 дБ.

У табл. 4.5. представлені часові вікна, які застосовуються при спектральному аналізі методом періодограм для $-N/2 \le n < N/2$. Максимальний рівень бокових пелюсток ΔS_S показує, синусоїду якого рівня неможливо розпізнати у періодограмі, якщо у сигналі є синусоїда з амплітудою 0 дБ у сусідніх бінах.

Таблиця. 4.5. Часові вікна для спектрального аналізу

№ пп.	Вікно	Функція $w(n)$	ΔS_S, дб	α	A
1	Прямокутне	1	$-13{,}3$	0,88	1
2	Трикутне	$1 - 2\lvert n\rvert/N$	$-26{,}5$	1,28	$\sqrt{3}$
3	Макса, Фока, Берт'є	$\frac{\sin(2\pi n/N)}{2\pi n/N}$	-26	1,24	1,49
4	Ханна	$0.5+0.5\cos\frac{2\pi n}{N}$	$-31{,}5$	1,44	1,63
5	Хеммінга	$0.54+0.46\cos\frac{2\pi n}{N}$	-43	1,3	1,59
6	Парзена	$1-6\frac{4n^2}{N^2}+6\frac{8\lvert n\rvert^3}{N^3}$, $\lvert n\rvert \le N/4$ $2\left(1-\frac{2\lvert n\rvert}{N}\right)^3$, $N/4 < \lvert n\rvert$	-51	1,84	1,93
7	Блекмана	$0.42+0.5\cos\frac{2\pi n}{N}+ \ +0.08\cos\frac{4\pi n}{N}$	-55	1,64	1,81
8	Лапласа – Гауса	$\exp\left(-\frac{n^2}{18N^2}\right)$	-55	1,64	1,84
9	Наттола	$0{,}36358+0{,}48918\cos\frac{2\pi n}{N}+$ $+0{,}1366\cos\frac{4\pi n}{N}+ \ 0{,}01064\cos\frac{6\pi n}{N}$	-98	1,7	
10	Карре–Руйє	$\frac{\sin(2\pi n/N)}{2\pi n/N}\cdot(1 - 2\lvert n\rvert/N)$	–	1,64	1,98

Оскільки множення сигналу на вікно дещо заглушує сигнал, амплітуда спектральних складових зменшується в A разів. Тому для корекції результатів аналізу функцію вікна або сигнал чи результат ДПФ слід помножити на A.

При аналізі випадкових сигналів слід враховувати статистичний показник якості оцінки Q, який дорівнює відношенню середніх

рівнів шуму і сигналу. Доведено, що для сигналу з гаусовим шумом спектральне оцінювання є слушним, якщо

$$QT_e B_S \geq 1 \tag{4.37}$$

Нехай $Q = 0{,}1$, тоді роздільна здатність, що досягається, оцінюється як $B_S \approx 10/T_e = 10B_e$. Тому для слушної оцінки спектру слід збільшувати величину T_e . Для цього можна збільшувати довжину виборки N у $M = 1/Q$ разів, а потім результат одержати як осереднення M сусідніх бінів.

Інший метод — метод періодограм Велча — оснований на тому, що $N{\cdot}M$ відліків сигналу розділяється на M виборок, для яких обчислюється спектр $\tilde{S}_i(k)$, i=1,...,M. Результуючий спектр є осередненням спектрів виборок:

$$\tilde{S}(k) = \frac{1}{M}\sum_{i=1}^{M} \tilde{S}_i(k) \tag{4.38}$$

ДПФ (4.33) основане на операції повороту комплексного вектора. Поворот виконується як множення комплексного числа на коефіцієнт $e^{-j2\pi pn/N}$, який скорочено зображається як W_N^{pn}.

Нехай вхідний сигнал $x(n)$ представляє собою комплексну синусоїду — вектор довжиною r_p, що в кожному такті повертається на кут $2\pi p/N$, тобто у n-му такті він дорівнює $r_p W_N^{-pn}$. Якщо ці вектори-відліки повертати у протилежний бік на кут $-2\pi k/N$ і накопичувати в акумуляторі, то результат $\Sigma r_p W_N^{-pn}{\cdot}W_N^{kn}$ досягне максимума Σr_p при умові, що $p = k$, а якщо $p \neq k$, то результат прямує до нуля. Недолік ДПФ полягає у великій кількості обчислень $-N^2$ комплексних множень та додавань, якщо необхідно знайти результати для усіх N частотних каналів або бінів.

Отже, у нульовому частотному каналі вхідні відліки повертають на кут 0, у $k = 1$ —му каналі — на кут $2\pi n/N$, у 2-му каналі — на кут $4\pi n/N$ і т.д. Причому у різних каналах трапляються однакові повороти тих самих даних, наприклад, у всіх каналах — на кут 0, у половині каналів — на кут $\pi/2$. Крім того, якщо непарні відліки у непарних каналах повернути на кут $2\pi n/N$, то решта поворотів у парах сусідніх каналів співпадають. Будь-який поворот для ДПФ довжини $2^{\log_2 N}$ можна представити як послідовність $\log_2 N$ поворотів на кут $2^q\pi/N$, $q = 1,2,...$. У парних каналах кількість поворотів можна скоротити удвічі, якщо додати відліки $x(n)$ та $x(n{+}N/2)$.

На таких особливостях ДПФ ґрунтуються алгоритми швидкого перетворення Фур'є (ШПФ). Так, ДПФ можна представити як додавання двох сум від парних та непарних вхідних даних:

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) W_N^{2nk} + \sum_{n=0}^{N/2-1} x(2n+1) W_N^{(2n+1)k} = \quad (4.39)$$
$$= \sum_{n=0}^{N/2-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_{N/2}^{nk}.$$

Тут множник T з (4.33) усунено як загальний нормуючий коефіцієнт. Як видно, (4.39) – це два ДПФ для $N/2$ даних кожне, причому результати другого ДПФ повернуті на кут $-2\pi k/N$. Аналогічно розкладаються $N/2$-точкові ДПФ на $N/4$-точкові ДПФ. З кожним таким розкладанням кількість операцій зменшується приблизно удвічі. Найменшим ДПФ є двоточкове ДПФ:

$$X(0) = x(0) + x(1);$$
$$X(1) = x(0) - x(1).$$

З урахуванням множення на коефіцієнт повороту $W_N^q = W^q$ одного з операндів базова операція ШПФ записується як

$$X(0) = x(0) + x(1)W^q;$$
$$X(1) = x(0) - x(1)W^q.$$

ГСПД базової операції ШПФ та її умовне позначення показані на рис. 4.24. За форму графа базової операції її часто називають "метеликом" (butterfly). За розмір ДПФ базової операції таке ШПФ називають ШПФ за основою 2. Таке ШПФ є найпоширенішим. Менш поширеним є ШПФ за основою 4, яке має на чверть меншу кількість операцій. ШПФ, які мають інші основи, включаючи комбіновані основи, використовуються рідко.

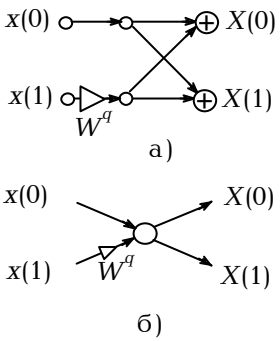


Рис. 4.24. ГСПД базової операції ШПФ (а) та її умовне позначення (б)

Один з графів алгоритмів ШПФ для $N = 8$ показано на рис. 4.25. На ньому пунктиром виділено графи 4-точкового ШПФ, які ілюструють обчислення за формулою (4.39). Виділення серед вхідних даних, що представляють сигнал у часовому просторі, в (4.39) парних чи непарних відліків називається проріджуванням. Через це даний алгоритм називається ШПФ з проріджуванням по часу. У інших алгоритмів ШПФ виконується проріджування, починаючи з частотних результатів і тому вони називаються ШПФ з проріджуванням по частоті.

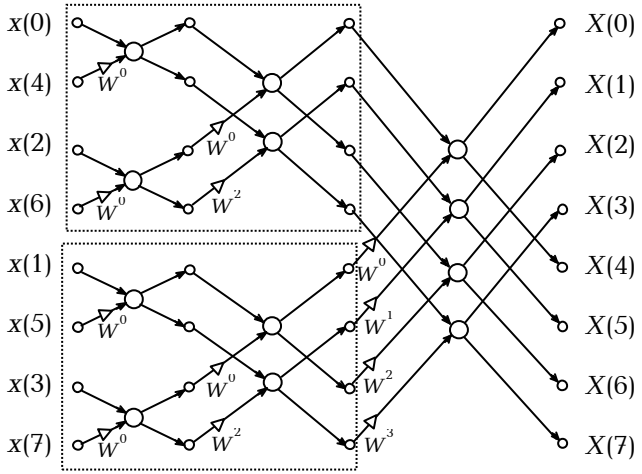


Рис. 4.25. Граф алгоритму ШПФ для $N = 8$

Стовпці точок на графі можна розглядати як масиви даних, що обробляються. Якщо номер даного у масиві представити двійковим числом, то щоб з натурального створити ***двійково-інверсний порядок*** масиву, слід дзеркально переставити двійкові розряди номеру даних. Наприклад, номер $a_3a_2a_1a_0$ перетворюється на номер $a_0a_1a_2a_3$. Тоді за нумерацією даних у стовпцях видно, що результуючий масив на рис. 4.25 має натуральний порядок, а вхідні дані — двійково-інверсний. Стовпець базових операцій називається ***ітерацією*** алгоритму ШПФ.

Нескладно порахувати, що у 8-точковому ШПФ лише 12 комплексних додавань та множень. N-точкове ШПФ має $0{,}5N \log_2 N$ комплексних додавань та множень замість N^2 таких операцій при обчисленні ДПФ за формулою (4.33). Отже, ШПФ є ефективним алгоритмом.

Алгоритм зворотнього ШПФ одержують при заміні поворотних коефіцієнтів на W^{-q} та корекції знаків у базовій операції згідно з формулою оберненого ДПФ (4.34). Але для обчислення оберненого ДПФ можна використати також пряме ШПФ, якщо усі результати помножити на N та, крім $X(0)$, поміняти місцями: $X(1)$ з $X(N-1)$, $X(2)$ з $X(N-2)$ і т.д. [1].

Досі малась на увазі обробка за допомогою ДПФ комплексного, тобто аналітичного сигналу. На практиці найчастіше ДПФ застосовується до реального сигналу. Для цього уявна частина вхідних даних представляється нулем: $\mathrm{Im}(x(n)) = 0$. Спектр такого сигналу для парного N має симетрію відносно спектрального відліку $X(N/2)$:

$$X^{*}(N-k) = X(k) ; \qquad (4.40)$$

тобто

$$\begin{aligned} &\mathrm{Re}(X(N-k)) = \mathrm{Re}(X(k)) ; \qquad k = 1,...,N/2-1; \\ &\mathrm{Im}(X(N-k)) = -\mathrm{Im}(X(k)); \\ &\mathrm{Im}(X(0)) = \mathrm{Im}(X(N/2)) = 0. \end{aligned}$$

Існують алгоритми обчислення ДПФ реальних послідовностей, в яких завдяки використанню властивості (4.40) загальна кількість операцій зменшена майже удвічі.

У лабораторній роботі ШПФ виконується програмою FFTbehR.VHD, яка описана нижче. У VHDL-симуляторах зазвичай цифрові сигнали спостерігаються у вигляді графіків, які одержуються у процесі моделювання алгоритму. Але алгоритм ШПФ припускає, що дані надходять і видаються відразу у вигляді масивів. Тому доцільно обчислювати ШПФ як ланцюжок процедур. Перша з них називається Gather_C і накопичує вхідний масив комплексних даних. Друга процедура — FFT — виконує алгоритм ШПФ, а заключна процедура Scatter_C — видає результуючий масив у вигляді послідовності даних.

Зазначені процедури виконуються об'єктом FFTbeh. Інтерфейс цього об'єкта й перелік використовуваних стандартних бібліотек виглядає так:

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_ARITH.all;
use IEEE.MATH_REAL.all;
use IEEE.MATH_COMPLEX.all;
entity FFTbeh is
  generic(NN:positive:=8;              -- 2**NN -довжина перетворення
          IFFT:natural:=0; -- 0 - пряме, 1- зворотнє пертворення
          pt:integer;          -- масштаб результату
          overs:integer:=0 -- 0,1 – без перекриття вхідних масивів;
                   -- 2 – перекриття 1/2,  4 – перекриття 1/4...
                   );-- -1, -2 – половина масиву занулюється для згортки
  port (
          CLK: in STD_LOGIC;        -- синхросерія
          RST: in STD_LOGIC;        -- встановлення в 0
          START: in STD_LOGIC;      -- початок накопиченя масиву
          EDIN: in STD_LOGIC;       -- строб даних
          DIN_RE: in real;          -- реальна частина даних
          DIN_IM: in real;          -- уявна частина даних
          DOUT_RE: out real:=0;     -- реальная частина спектру
          DOUT_IM: out real:=0;     -- уявна частина спектру
          NUM: out integer:=0;      -- номер вихідного відліку
          READY: out STD_LOGIC);-- початок виводу результатів
  end FFTbeh;
```

Настроювальна змінна N = 2^{NN} задає довжину перетворення. При IDCT = 0 виконується пряме перетворення, а при 1 — зворотнє. Змінна PT задає масштаб результату. Отже, програма виконує обчислення (4.33), (4.34):

$$X(k) = \frac{1}{2^{PT}} \sum_{n=0}^{N-1} x(n) e^{-j(-1)^{IDCT} 2\pi kn/N}.$$

де $x(n)$ = DIN_RE +j*DIN_IM —відлік комплексного вхідного даного, $X(k)$ = DOUT_RE +j*DOUT_IM— відлік комплексного спектру.

За сигналом START починається накопичення масиву вхідних даних. Закінчення обчислень ШПФ вказується сигналом READY. Вихідні відліки спектру супроводжуються номером NUM, за котрим знаходиться значення частоти певної спектральної лінії. Сигнал EDIN стробує прийом вхідних даних.

Поведінка об'єкта FFTbeh_R описана в архитектурі FFT_R. Для представлення масивів даних в ній оголошені наступні типи масивів комплексных і реальних даних.

```
constant N: integer:=2**NN;                  -- довжина перетворення
 type MEMOC is array (0 to N) of complex;  --комплексний масив
 type MEMOC_2 is array (0 to N/2-1 ) of complex;
 type MEMOR is array (0 to N-1) of real;    -- реальний масив
```

Далі описана процедура, за якою накопичується масив вхідних комплексних даних.

```
 procedure GATHER_C(signal CLK,START,EDIN:in std_logic;
        signal DR,DI: in integer;            --реальна та уявна частини
                -- відліків вхідних даних
        signal rdy: out std_logic;           --сигнал готовності
        signal datact: inout natural;        --номер даного
        signal RAM:out MEMOC)                -- робочий масив
 is
 begin
        wait until  CLK= '1';  -- запуск процедури за фронтом CLK
          if EDIN = '1' then  -- дозвіл прийому
                RAM(datact)<=(DR,DI);  -- запис комплексного даного
                if datact < RAM'right then
                        datact<=datact+1;    -- лічильник даних
                end if;
                if START='1' then
                   datact<=RAM'left; --встановлення лічильника даних
                end if;
                rdy<='0';
                if datact =RAM'right-1 then
                    rdy<='1';       --масив готовий
```

```
                end if;
        end if;
end procedure;
```

У цій процедурі при кожному її запуску при EDIN = '1' в масив RAM за лічильником datact записується чергова пара даних .

Процедура SCATTER_C виглядає як

```
procedure SCATTER_C(signal CLK,START,EDIN : in std_logic;
        pt: natural;                    --масштаб результату
        signal RAM: in MEMOC;           --вхідний масив
        signal datact: inout natural;  --лічильник даных
        signal rdy: out std_logic;     -- кінець виводу масиву
        signal DR,DI: out integer)  -- вихідні дані
is
        variable s:real:=real(2**pt); -- масштабний коефіцієнт
begin
        wait until CLK='1';
        if START='1' then
                datact<=RAM'left;
        elsif EDIN='1' then
                RDY<='0';
                DR<= integer(RAM2(datact).RE/s);
                DI<= integer(RAM2(datact).IM/s);
                if  datact< RAM'right then
                        datact<=datact+1;
                end if;
                if datact=RAM'right-1 then
                        RDY<='1';
                end if;
        end if;
end procedure;
```

Ця процедура виконується так само, як і попередня. Але навпаки, накопичені у масиві дані послідовно видаються на її вихід.

Нарешті, процедура, що обчислює ШПФ масиву даних виглядає так:

```
procedure FFT(N:positive;              -- розмір перетворення =2**N
        signal rdy:in std_logic;      -- запуск перетворення
        signal RAM_I: in MEMOC;  -- вхідний масив даних
        signal RAM_O:out MEMOC)  -- вихідний масив бінів
is
        constant NN:positive:=2**N;
        variable TWIDDLE:MEMOC_2;  -- массив поворотних коефіцієнтів
        variable RAM:MEMOC;                -- робочий масив
        variable a,b,c,d,w: complex;
```

```
        variable al:real;
        variable base,itera,datact,twiddlect,twiddleinc: natural;
        variable delta:integer:=1;
        variable addrf: std_LOGIC_VECTOR(N-1 downto 0);
        variable addri: std_LOGIC_VECTOR(N-1 downto 0);
begin
-- формування таблиці поворотних коефіцієнтів
   for i in 0 to NN/2-1 loop
                al:=   (MATH_PI*real(2*i))/real(NN);
                if IDCT=0 then
                  TWIDDLE(i):=(COS(al),-SIN(al)); -- для прямого ШПФ
                else
                   TWIDDLE(i):=(COS(al), SIN(al));-- для зворотнього ШПФ
                end if;
        end loop;

        loop    -- основний цикл
                wait until RDY='1';         -- початок ШПФ
-- двійково-інверсна перестановка вхідних даних
                addrf:=(others=>'0'); -- вектор адреси для прямого порядку
                        datact:=0;
                for i in 1 to NN loop
                        for ind in 0 to N-1 loop -- інверсія порядку бітів
                          addri(ind):=addrf(N-1-ind);  -- інверсна адреса
                        end loop;
     RAM(conv_integer(unsigned(addri))):=RAM_I(conv_integer(unsigned(addrf)));
                        addrf:=unsigned(addrf)+1;
                end loop;
-- власне ШПФ
                itera:=0;
                delta:=1;
                twiddleinc:=0;
                for itera in 1 to N loop -- початок ітерації
                        base:=0;
                        twiddlect:=0;
                        for butterfly in 0 to NN/2 - 1 loop
                                a:=RAM(base);   -- початок базової операції
                                base:=base+delta;
                                b:=RAM(base);
                                w:=TWIDDLE(twiddlect);
                                c:=a + b * w;     -- власне метелик
                                d:=a - b * w;
                                RAM(base-delta):=c;
                                RAM(base):=d;   -- кінець базової операції
                        -- модифікація параметрів базової операції
                    base:= base + delta;
```

```
                    if base >= NN then
                        base:=base-NN+1;
                        twiddlect:=twiddlect+twiddleinc;
                    end if;

                end loop;       --кінець ітерації
            -- модифікація параметрів ітерації
                delta:=delta*2;
                if itera=1 then
                    twiddleinc:=NN/4;
                else
                    twiddleinc:=twiddleinc/2;
                end if;
            end loop;
            RAM_O<=RAM;             -- результати ШПФ
        end loop;
end procedure;
```

Тут виконується алгоритм ШПФ за основою 2 із проріджуванням у часі із заміщенням даних, як на рис. 4.25. Процедура має дві ділянки: перша з них виконується однократно на початку обчислень, а друга — власне ШПФ — виконується періодично. При виконанні першої ділянки формується таблиця обертаючих коефіцієнтів, розмір якої дорівнює довжині перетворення і яка потім використовується алгоритмом ШПФ.

Друга ділянка являє собою нескінченний цикл. Цикл запускається, як тільки готовий масив вихідних даних, який підготовлено процедурою GATHER_C. Після запуску дані пересилаються із вхідного масиву у робочий масив. При цьому виконується двійкова інверсія адрес запису. Слід зазначити просту й оригінальну можливість двійкової інверсії адреси, що надає мова VHDL. Адреса представляється бітовим вектором і біти в ньому переставляються в інверсному порядку безпосередньо. У звичайних мовах програмування для цього необхідно виконати досить складний і не завжди очевидний алгоритм.

Як і всі алгоритми ШПФ, даний алгоритм являє собою гніздо циклів. У внутрішньому циклі виконуються базові операції ШПФ, а зовнішній цикл утворений NN ітераціями алгоритму ШПФ.

Виконавча частина архітектури містить виклики вищеописаних процедур:

```
begin
  INPUT:GATHER_C(CLK,START,EDIN, DR=>DIN_Re,DI=>DIN_IM,
                 rdy=>idatardy,datact=>datact_I,RAM=>RAM1);
  SPECTRUM:FFT(N,idatardy,RAM_I=>RAM1,RAM_O=>RAM2);
  OUTPUT: SCATTER_C(CLK,idatardy,EDIN,PT,RAM=>RAM2,DATACT=>datact_o,
```

```
                    rdy=>rdy,DR=>DOUT_RE,DI=>DOUT_IM);
  DEL:process(CLK) begin
        if CLK='1' and CLK'event and EDIN='1' then
                READY<=idatardy;
                NUM<= datact_o;
        end if;
   end process;
end FFT_int;
```

Для розуміння виконання програми варто нагадати, що паралельний виклик процедури, у якій застосовується оператор **wait**, еквівалентний оператору процесу, тіло якого складається з тіла процедури. Таким чином, всі три процедури, як і еквівалентні їм процеси, виконуються паралельно в деякому конвеєрному режимі. При цьому дані й керування передаються від процедури до процедури як між ланками конвеєра. У результаті, об'єкт FFTbeh обробляє безперервний потік даних, сегментуючи його на блоки довжиною 2**NN.

Процес DEL виконує затримку сигналу готовності READY і потоку номерів NUM відліків результату на один такт, щоб вони відповідали вихідним відлікам сигналів DOUT_RE та DOUT_IM .

Завдання для роботи

1. Розробити VHDL-модуль множення на часове вікно. Модуль виконує додавання трьох вхідних сигналів: $x(n)=A(n)+B(n)+C(n)$. Результуючий потік реального сигналу $x(n)$ помножується на часове вікно заданого типу або на прямокутне вікно (пропускає сигнал без змін). Тип вікна і його довжина N береться з таблиці 4.6. Модуль повинен також видавати сигнал START, який позначує початок вікна і призначений для запуску модуля ШПФ. Графік вікна занести у звіт.

2. Розробити VHDL-модуль, який осереднює M сусідніх вибірок спектру $X^2(k) = \tilde{S}_i(k)$ довжиною N відліків за формулою (4.38) або пропускає $\tilde{S}_i(k)$ без обробки. Також модуль повинен видавати результат у логарифмічному масштабі, тобто у децибелах.

3. Скласти стенд для випробувань, який має два генератори синусоїдальних сигналів, генератор шумового сигналу, генератор спеціального сигналу, модуль множення навікно, модуль ШПФ FFTbeh та модуль осереднення. Генераторами спеціального та синусоїдального сигналів мають бути модуль, побудований у першій лабораторній роботі та модуль FilterTB_r, відповідно. Довжина ШПФ встановлюється рівною N (табл.4.6).

Таблиця 4.6. Варіанти завдань

№ вар.	Вид вікна	N	M	№ вар.	Вид вікна	N	M
1	Трикутне	512	8	16	Блекмана	512	49
2	те саме	1024	9	17	те саме	1024	16
3	—	2048	10	18	—	2048	8
4	Макса, Фока, Берт'є	512	16	19	Лапласа — Гауса	512	100
5	те саме	1024	25	20	те саме	1024	64
6	—	2048	36	21	—	2048	49
7	Ханна	512	49	22	Наттола	512	16
8	те саме	1024	16	23	те саме	1024	25
9	—	2048	8	24	—	2048	36
10	Хеммінга	512	100	25	Карре—Руйє	512	49
11	те саме	1024	64	26	те саме	1024	64
12	—	2048	49	27	—	2048	8
13	Парзена	512	16	28	Хеммінга	512	9
14	те саме	1024	25	29	те саме	1024	10
15	—	2048	36	30	—	2048	16

4. Підключити до реального входу модуля FFTbeh вихід модуля множення на вікно, а до уявного входу — нуль. Входи модуля множення на вікно підключити до виходів двох генераторів синусоїдального сигналу з однаковими амплітудами вихідних сигналів, що дорівнюють 1.0. Частота першого генератора дорівнює $f_1 = 16/N$, частота другого — $f_2 > f_1$ встановлюється близькою до f_1. Частота генератора регулюється відношенням $f =$ fstrt/fsampl при f_2 fsampl $\neq N$. Параметр генераторів Maxdelay $> NM$.

Дослідити спектр сигналу для прямокутного вікна при відсутності осереднення. Підібрати таку частоту f_2, щоб її різниця $\Delta f = f_2 - f_1$ була мінімальною і щоб піки частот у спектрі можна було розрізнити, тобто між піками має бути провал не менше 3 дб. Записати Δf як роздільну здатність спектральної оцінки.

Встановити $f_1 = 13/(0.8N)$, $f_2 = 16/(0.73N)$. Встановити таку амплітуду синусоїди з частотою f_2, пік якої можна розрізнити у спектрі рядом з піком f_1. Виміряти різницю амплітуд сигналів з частотами f_2, f_1 у децибелах і записати її як динамічний діапазон спектральної оцінки D_S. Виміряти також різницю амплітуд сигналів з частотами 0 та f_1 у децибелах і записати її як динамічний діапазон вікна D_W.

5. Повторити п. 4 для заданого часового вікна. Перевірити справдження параметрів ΔS_S, α, A для заданого вікна.

6. Повторити п. 5 з доданим шумом, максимальна амплітуда якого у 30 разів менша за амплітуду першого синусоїдального сигналу.

7. Повторити п. 6, виконавши M-кратне осереднення спектру .

Порівняти результати п. 5 і п. 7 та перевірити справдження формули (4.33).

8. Підключити до входу модуля множення на вікно лише генератор спеціального сигналу, його параметри N_1,N_2 зменшти у 20 разів і виміряти спектр сигналу, застосувавши задане вікно. Перевірити формулу (4.40).

9. Проаналізувати одержані графіки. Зробити висновки по роботі.

Приклад виконання роботи

Нехай задано: вікно $w(n) = (2{,}18 - (8n/N)^2 e^{-|4n/N|})$, $N = 1024$, $M = 64$.

Графік вікна показано на рис. 4.26.

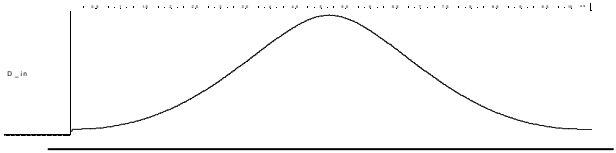
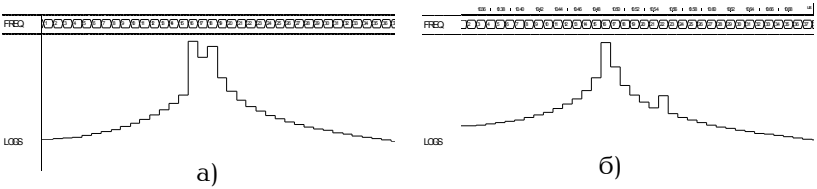


Рис. 4.26. Часове вікно для спектрального аналізу

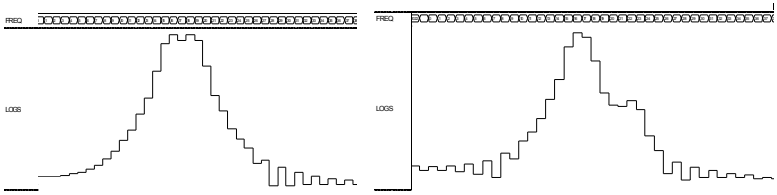
Для прямокутного вікна одержано $\Delta f = 1{,}60$ біна, динамічний діапазон $D_S = 21{,}6$ дБ, $D_W = 32{,}7$ дБ.



а) б)

Рис. 4.27. Спектр при мінімальній різниці Δf (а) та максимальній різниці амплітуд (б) при прямокутному вікні

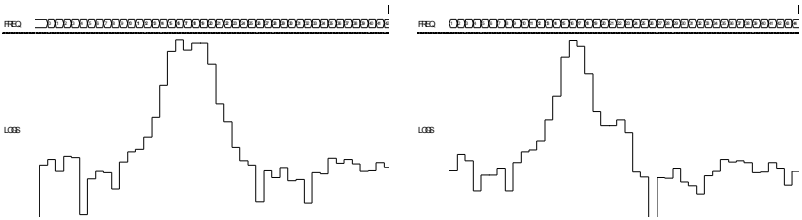
Для заданого вікна одержано Δf = 2,16 біна, динамічний діапазон D_S= 34,5 дБ, D_W= 68,5 дБ.



а) б)

Рис. 4.28. Спектр як на рис. 4.27 при застосуванні спеціального вікна

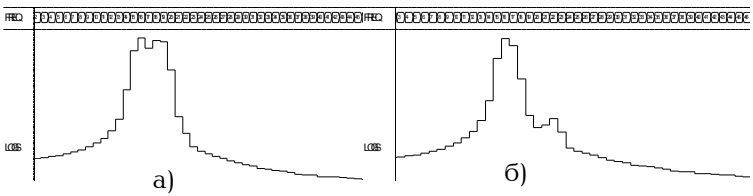
Для заданого вікна при наявності гаусового шуму одержано Δf = 2,5 біна, динамічний діапазон D_S= 34,7 дБ, D_W= 53,9 дБ.



а) б)

Рис. 4.29. Спектри як на рис. 4.28 при доданому шумові

Для заданого вікна при наявності гаусового шуму і 64 ітераціях осереднення одержано Δf = 2,45 біна, динамічний діапазон D_S= 28,4 дБ, D_W= 42,1 дБ.



а) б)

Рис. 4.30. Спектри як на рис. 4.29 після 64 осереднень

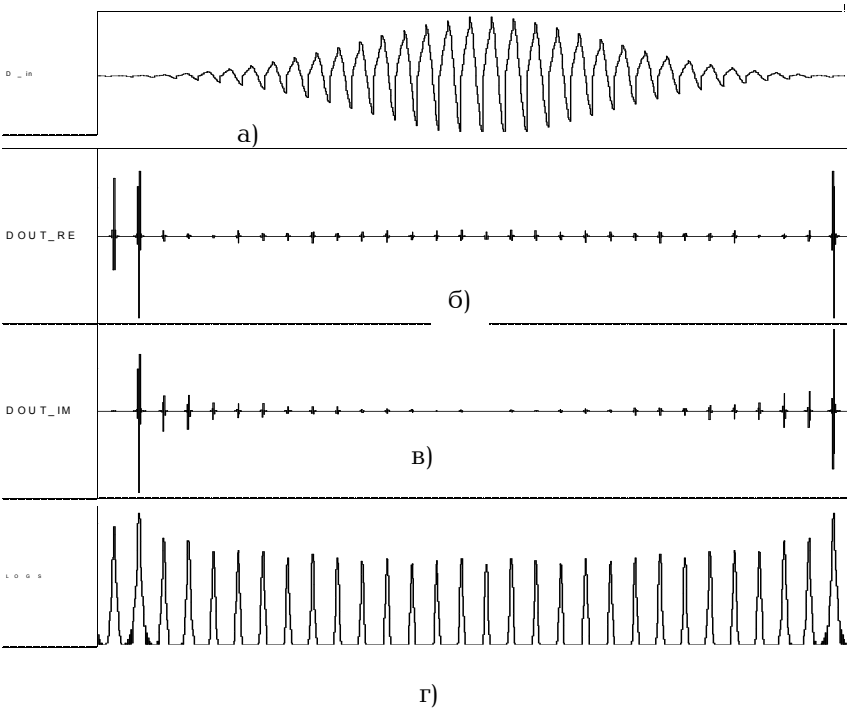


Рис. 4.31. Сигнал після множення на вікно (а), реальна (б) та уявна (в) частини його спектру й модуль спектру в логарифмічному масштабі (г)

Результати вимірювання спектру сигналу спеціальної форми: амплітуда нульової частоти (постійна складова) $\tilde{S}(0) = 43{,}2$ дБ, амплітуда першої гармоніки $\tilde{S}(34) = 51{,}3$ дБ, амплітуда другої та третьої гармонік $\tilde{S}(68) = 36{,}5$ дБ, $\tilde{S}(102) = 35{,}2$ дБ. Аналіз вимірів показує, що період сигналу дорівнює $1024/34 = 30{,}1$ відліків, що доволі точно співпадає з заданим у генераторі періодом. Частоти гармонік сигналу кратні частоті першої гармоніки, що свідчить про незмінність періоду сигналу. Сигнал має постійну складову, потужність якої у 6 разів (на 8,1 дб) менше потужності основної (першої) гармоніки. Друга та третя гармоніки за потужністю у 30 та у 40 разів, тобто на 14,8 і 16,1 дБ менші за першу гармоніку. Це означає, що сигнал є далеким від синусоїдального і має несиметричну форму.

Загальні висновки наступні. Застосування спеціального часового вікна збільшує динамічний діапазон вимірювання спектру з 21,6 дБ до 34,5 дБ, а при доволі великій відстані між бінами, у яких шукаються сигнали — до 68,5 дБ. Роздільна здатність такого вікна складає від 2,16 до 2,5 біна в залежності від наявності шума у сигналі. Застосування осереднення суттєво покращує слушність оцінки спектру. Після 64 осереднень стає видно, що генератор шуму має рівномірний спектр, рівень якого на 53 дб нижче за рівень сигналу, тобто його спектральна щільність потужності у $2{,}4 \cdot 10^5$ менша.

Література

1. Введение в цифровую обработку сигналов / Под ред. Р.Богнера и А. Константинидиса. — М.:Мир. — 1976. — 216 с.
2. Макс Ж. Методы и техника обработки сигналов при физических измерениях. Т1. Основные принципы и классические методы. — М.:Мир. —1983. —311 с.
3. Марпл С. Л. мл. Цифровой спектральный анализ и его приложения. —М.:Мир. —1990. —584 с.
4. Оппенгейм А., Шафер Р. Цифровая обработка сигналов. —М: Техносфера. — 2006. — 856с.
5. Поляков А.К. Языки VHDL и VERILOG в проектировании цифровой аппаратуры. — М.: Солон, 2003.- 320 с.
6. Сато Ю. Без паники! Цифровая обработка сигналов. —М.: ИД Додэка—XXI. —2005. —175 с.
7. Сергиенко А.М. VHDL для проектирования вычислительных устройств. –К.: –"ДиаСофт". –2003. –210 с.
8. Сэломон Д. Сжатие данных, изображений и звука. — М: Техносфера. — 2004. — 368с.
9. Яне Б. Цифровая обработка изображений. —М.: Техносфера. — 2007. — 583 с.
10. Aldec's New Co-Simulation Wizard for Simulink Offers Support for Mixed HDL/System Design. October 22, 2003. Режим доступу: http://www.aldec.com
11. Kuo S. M., Lee B. H. Wenshun T. Real-Time Digital Signal Processing. Implementations and Applications. — Wiley. — 2006. — 646 p.
12. Regalia P., Mitra S.K., Vaidyanathan P.P. The Digital All-Pass Filter: A Versatile Signal Processing Building Block // Proc. IEEE. —1988. —V.76. —№1. — p.19–37.
13. Сайт Matlab. Режим доступу: http://matlab.exponenta.ru
14. Сайт VHDL-AMS. Режим доступу: http://www.vhdl-ams.org