НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Кулініч О.М., Касаткін Д.Ю., Лахно В.А., Сагун А.В.

НАВЧАЛЬНИЙ ПОСІБНИК

ТЕОРІЯ І ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ

(Частина 1. ПРОЕКТУВАННЯ СИСТЕМ ОБРОБКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ)

Київ - 2023

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Кафедра комп'ютерних систем, мереж та кібербезпеки

Кулініч О.М., Касаткін Д.Ю., Лахно В.А., Сагун А.В.

ТЕОРІЯ І ПРОЕКТУВАННЯ КОМП'ЮТЕРНИХ СИСТЕМ І МЕРЕЖ (ЧАСТИНА 1. ПРОЕКТУВАННЯ СИСТЕМ ОБРОБКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ)

Рекомендовано до друку Вченою радою факультету інформаційних технологій (протокол № 5 від 22 листопада 2023 р.)

Київ - 2023

УДК 621.3.049.77.001.63

П79

Теорія і проектування комп'ютерних систем і мереж (Частина 1. Проектування систем обробки та захисту інформації) Навчальний посібник / **Кулініч О.М., Касаткін Д.Ю., Лахно В.А., Сагун А.В.** – К.: Видавництво, 2023 – 410 с.

Рекомендовано до друку Вченою радою факультету інформаційних технологій (протокол № 5 від 22листопада 2023 р.)

Рецензенти:

С. В. Толюпа, д.т.н., професор (Київський національний університеи ім. Тараса Шевченка);

В. С. Чевардін, д.т.н., с.н.с. (Військовий інститут телекомунікацій та інформатизації).

В навчально посібнику описані загальні принцири проектування та створення систем обробки та захисту інформації, представлені елементи спеціалізованої мови програмування AHDL та приклади реалізації сучасних функціональних вузлів і блоківрадіоелектронних систем.

Посібник буде у нагоді для студентів, що вивчають проектування та побудову засобів оробки та захисту інформації.

Призначені для студентів спеціальності 122 «Комп'ютерні науки» та 125 «Кібербезпека».

Видання здійснено за авторським редагуванням.

ЗМІСТ

ВСТУП

Спроби формалізації роботи конструкторів і нав'язування їм суворої програми дій у загальному вигляді для творчості шкідливі. Коли ж ми прагнемо уподібнити роботу інформаційних машин дії нашого мозку - це розумно. Але було б сумною помилкою нашу діяльність зробити подібною до функціонування обчислювальної машини. Однак виявлення загальних закономірностей проектно-конструкторського процесу, виділення загальних етапів і процедур, розробка різних методів вирішення завдань на цих етапах - справа, необхідна як для стимулювання творчої діяльності досвідчених проектувальників або навчання молодих фахівців.

Процес проектування є багатоетапним та повторюваним, що передбачає повернення та перегляд прийнятих рішень. До даного процесу потрібно додати можливості сучасної бази елементів, розповсюдження якої завершується отриманням типових функцій, що відповідають окремим мікросхемам або елементам функціональних бібліотек, програмованих ВІС/НВІС.

Стандартні ВІС/НВІС відрізняються рівнем інтеграції, оскільки висока вартість проектування досягає сотень мільйонів доларів, виявляється прийнятною в цьому випадку, оскільки вона розподілена на велику кількість вироблених мікросхем.

Окрім стандартних деталей, система також включає спеціальні частини, специфічні для цього рішення. Це стосується схем управління блоками, забезпечення їх взаємодії і т. д. Реалізація нестандартної частини системи історично пов'язана з використанням малих і середніх інтегральних мікросхем. Використання МІС та СІС супроводжується швидким зростанням кількості корпусів інтегральних схем, ускладненнями монтажу та зниженням надійності та швидкості системи.

Крім того, між завершенням розробки схеми і здачею перших дослідних зразків проходить тривалий період часу, і люба помилка або заміна схеми буде коштує дуже дорого.

Виникле протиріччя знайшло рішення у великих інтегральних схем і надвеликих інтегральних схем з програмованою та репрограмованою структурою. Одними з перших, представниками цього напрямку були програмовані логічні матриці, програмована логічна матриця і базові матричні кристали. Де користувач організовує власну логіку, навіть в одному екземплярі, використовуючи просте та дешеве обладнання порівняно з програмістами.

Програмовані логічні матриці засновані на технологіях, подібних до тих, що використовуються в програмованих пристроях запам’ятовування, але використовують дані технології для незвичних цілей.

Програмована матриця - це мікросхема, що містить велику кількість елементів, які використовують основні логічні функції. Користувач може організовувати ці функції в тій чи іншій логічній схемі відповідно до своїх вимог і довільним чином «розділяти» не тільки зв'язки між логічними елементами буфера «І/АБО», але й визначати призначення кристалічних терміналів і корпус мікросхеми.

На теперішній час існує багато версій потужного програмного забезпечення, для сумісних ПК, з їхньою допомогою здійснюється власне програмування системи за описом користувача логічних функцій і рівнянь за допомогою комп'ютерного моделювання, в ході якого перевіряється правильність підготовленого проекту.

Сьогодні можна написати щось на зразок «програми», а потім на її основі створити спеціалізовану інтегральну схему. Такий підхід дає низку переваг: збільшення розміру та простоти друкованих плат, прискорення розробки новітніх конструкцій, збільшення надійності, зменшення кількості необхідних стандартних типів цифрових мікросхем та вдосконалення власних навичок.

РОЗДІЛ І. ЗАГАЛЬНІ ПОНЯТТЯ ЩОДО ПРОЕКТУВАННЯ

1.1. Основні поняття та визначення

Проектування нових типів і моделей машин, апаратів, пристроїв, пристроїв, приладів та інших виробів є складним і тривалим процесом, що включає розробку вихідних даних, креслень, технічної документації, необхідної для виготовлення дослідних зразків і подальшого виробництва та експлуатації. об'єктів проектування.

Проектування - це комплекс робіт, спрямованих на отримання характеристик нового або модифікованого технічного об'єкта, достатніх для створення або виготовлення об'єкта в необхідних умовах.

У процесі проектування виникає необхідність створити опис, необхідний для побудови об’єкта, якого ще не існує. Описи, отримані під час проектування, є остаточними або непрямими. Остаточні описи — це збірник конструкторської та технологічної документації у вигляді креслень, специфікацій, програм для ЕОМ і систем автоматизації тощо.

Процес проектування, яким повністю керує людина, називається неавтоматизованим.

Автоматизованим називають проектування, де відбувається взаємодія людини з комп’ютером.

Автоматизована система проектування — організаційно-технічна система, яка складається з комплексу засобів автоматизації створення, яка взаємодіє з підрозділами проектної організації та здійснює автоматизоване проектування.

Об’єкт проектування (ОП) — новий (модернізований) технічний об’єкт, неіснуючий у природі (у такому вигляді) до початку проектної діяльності, створений у процесі та отриманий у результаті проектування.

життєвим циклом зазвичай розуміють етапи «існування» технічного об'єкта (системи) від моменту усвідомлення необхідності його виробництва до моменту його знищення за непотрібністю.

У короткий список, як правило, входять проектування, виробництво, виконання технічного завдання, експлуатація (включаючи ремонтопридатність і модифікація) і, нарешті, утилізація.

Складність технічних об'єктів з точки зору їх проектування зростатиме у зв'язку з необхідністю врахування специфіки всіх інших етапів життєвого циклу вже на етапі проектування. Цей компонент ускладнює як процес проектування, так і сам об’єкт. Ступінь складності зазвичай пропорційний початковій складності цього об'єкта. Відмова від урахування в процесі проектування тих характеристик ОП, які актуальні на інших етапах життєвого циклу, може (і зазвичай призводить) призвести до значних додаткових матеріальних витрат.

Що стосується технічних об'єктів (систем), то складність прийнято оцінювати за кількістю елементів, що утворюють систему, і за кількістю зв'язків, що визначають спосіб взаємодії елементів. В якості альтернативної оцінки часто використовується потужність множини можливих (спостережуваних) станів системи і множина залежностей, що визначають можливі переходи від стану до стану (зазвичай перша оцінка може бути зведена до другої).

Дві складові складності проекту: **суб'єктивна та об'єктивна.**

Перша складова визначає ступінь сприйняття людиною поведінки системи. Людина давно навчилася поводитися з цією стихією, просто абстрагуючись від неї.

Другий пов'язаний з розмірністю опису (репрезентації) системи, і зокрема з розмірністю завдань проектування. Цей компонент можна представити оцінками складності, наведеними раніше, і він менш актуальний для автоматизації проектування, оскільки міри складності людини та комп’ютера відрізняються.

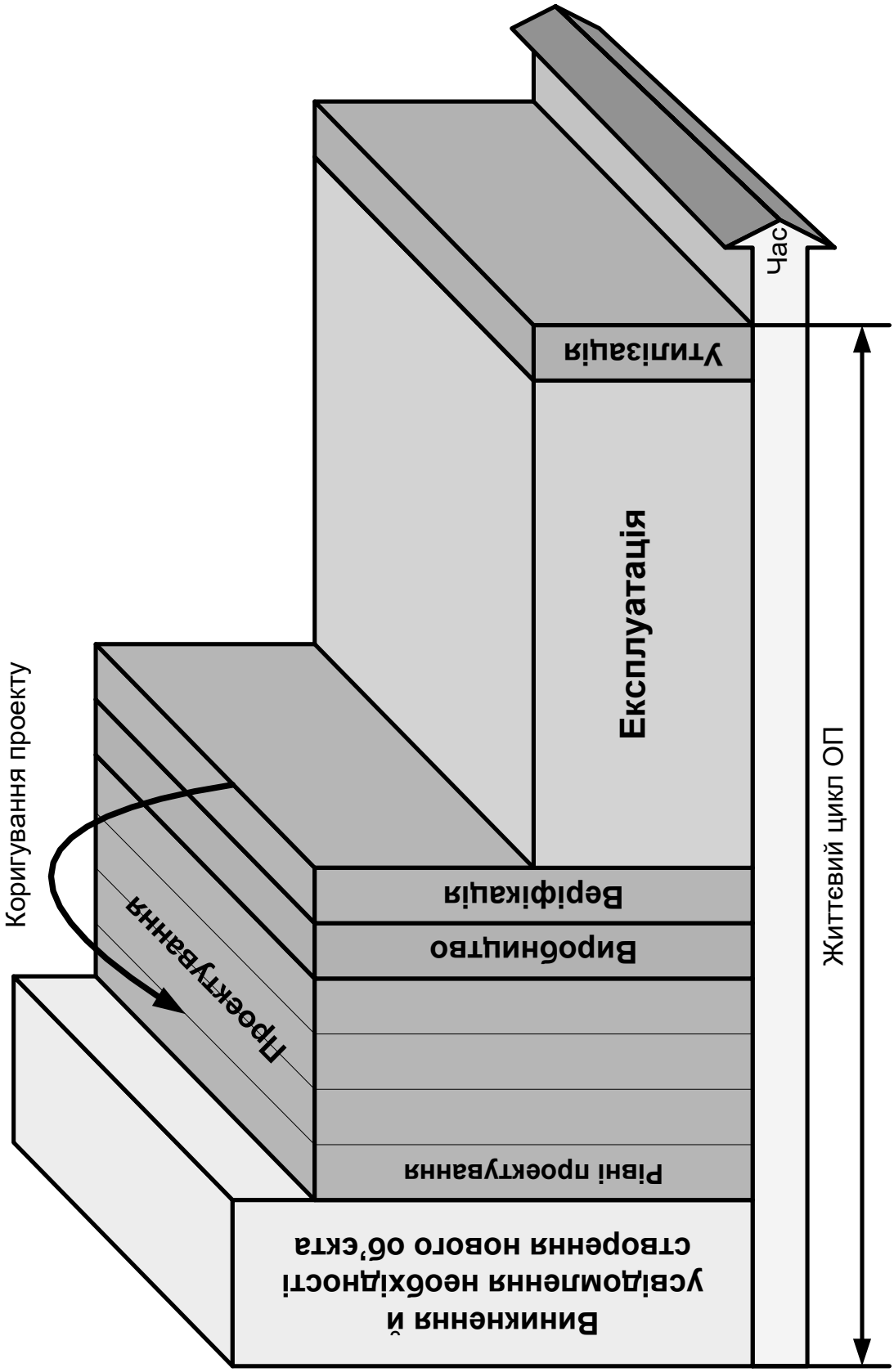


Рис. 1.1 - Життєвий цикл об'єкта проектування

Спосіб подолання «комплексу складності» для другої складової також відомий людству з давніх часів і базується на принципі «розділяй і володарюй». Принцип передбачає поділ (декомпозицію) системи на ряд менш

складних, взаємопов'язаних підсистем, кожна з яких має меншу розмірність і може бути зрозуміла людині, якщо сформулювати відповідні цим підсистемам абстракції.

У загальному випадку вплив «комплексу складності» на завдання проектування машин і обчислювальних систем обумовлено:

- складність самої предметної області;

- необхідність урахування на етапі проектування особливостей інших стадій життєвого циклу ОП;

- складність управління процесом проектування;

- складність опису поведінки дискретних систем;

- необхідність і складність перевірки проектних рішень.

Якщо система не є дискретною, тобто описується неперервною функцією, то невеликі зміни у вхідних параметрах завжди викликатимуть невеликі зміни у вихідних параметрах. Навпаки, дискретні системи за своєю природою мають кінцеве число можливих станів, хоча у великих системах це число дуже велике за правилами комбінаторики.

У той же час переходи між дискретними станами неможливо змоделювати за допомогою неперервних функцій. Будь-яка зовнішня по відношенню до системи подія може перевести її в новий стан, і перехід з одного стану в інший не завжди детермінований. У несприятливих умовах зовнішня подія може порушити поточний стан системи, оскільки її творці не змогли передбачити всі можливі варіанти. Це ще одна (можливо, основна) причина, чому при декомпозиції дискретних систем, навіть в межах одного рівня абстракції, підсистеми повинні мало залежати одна від одної, забезпечуючи при цьому цілісність функцій усієї системи. У розривних системах така поведінка малоймовірна, але в дискретних системах будь-яка зовнішня подія може вплинути на будь-яку частину внутрішнього стану системи. Це, звичайно, основна причина обов'язкового тестування дискретних систем. Але насправді, за винятком самих тривіальних випадків, повністю перевірити ці системи неможливо.

За відсутності як математичних інструментів, так і інтелектуальних можливостей для повної перевірки поведінки великих дискретних систем можна розраховувати лише на прийнятний рівень впевненості щодо їх правильності. Ця довіра в першу чергу повинна базуватися на довірі до проектних рішень, тобто вона повинна бути забезпечена вже в процесі проектування.

Звичайне тестування інструментів зводиться до тестування окремих програм на вибіркових прикладах, що є не способом демонстрації коректності програм в цілому, а лише показником можливої відсутності помилок.

1.2. Класифікація процесу проектування

Ідеї складних технічних об'єктів у процесі їх створення поділяються на аспекти та рівні ієрархії. Дані характеризують групу пов'язаних властивостей об'єкта. Типовими даними в описі технічних об'єктів є: технологічний, функціональний і конструктивний.

– Функціональний аспект відтворює фізичні та інформаційні процеси, що відбуваються в об'єкті під час його експлуатації.

– Конструктивний аспект характеризує конструкцію, розміщення в просторі і форми складових об'єкта.

– Технологічний визначає доцільність, можливості і способи виготовлення об'єкта в заданих умовах.

Блоково -ієрархічний підхід до проектування полягає в розподілі описів проектованих об'єктів на ієрархічні рівні за ступенем деталізації властивостей об'єктів.

Типовими ієрархічними рівнями функціонального проектування є: функціонально-логічний (функціональна та логічна діаграми); схеми (схеми підключення вузлів і окремих блоків); компонент (конструкція елементів та їх розташування).

Проектування також поділяється на стадії, стадії та процедури. Є етапи:

– науково-дослідні роботи (НДР),

– дослідно-конструкторські роботи (НДК),

– ескізний дизайн,

– технічний проект,

– робочий проект,

– тестовий зразок.

Проектне рішення - опис проекту або його частини, для прийняття заявки на завершення проекту або шляхів його продовження.

Процедура проектування - це частина проектування, яка завершується отриманням проектного рішення.

Шлях проектування - це серія проектних процедур, що ведуть до отримання необхідних проектних рішень.

процедури **аналізу та синтезу** .

Аналітична процедура є об'єктно-орієнтованим дослідженням.

Справжнє завдання аналізу формулюється як завдання встановлення відповідності між двома різними описами одного предмета.

Процедура синтезу полягає у створенні описів проектованого об'єкта, які відображають структуру та параметри об'єкта.

Проектування як окремих об'єктів, так і систем починається з розробки технічного завдання (ТЗ) на проектування. ТЗ містить основну інформацію про об'єкт проекту, умови його експлуатації, а також вимоги замовника до проектованого виробу. Типова схема конструкції наведена на рис. 1.2.

Найважливішою вимогою до ТЗ є його повнота.

Виконання цієї вимоги визначає своєчасність і якість виконання проекту. Наступний крок – ескізний проект.

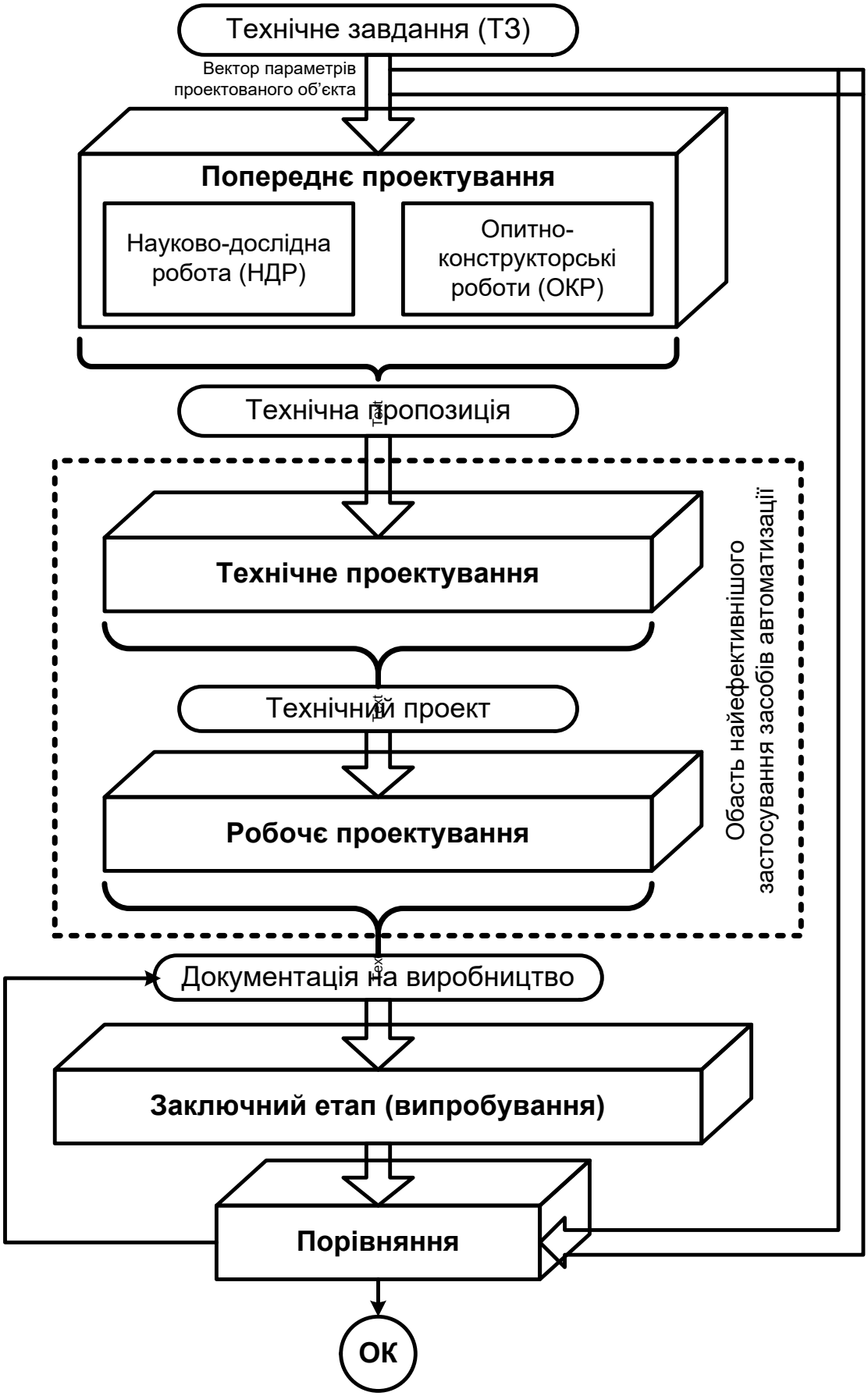


Рис. 1.2 Типова проектна схема

Початковий проект - це пошук основних можливостей побудови системи, вивчення нових правил, структур і обґрунтування найймовірніших загально прийнятних рішень.

Результатом даного етапу є технічна пропозиція.

На етапі ескізного створенн проводиться деталізоване вивчення доцільності побудови системи.

Результатом цього етапу є оформлення проекту.

Технічне проектування - це виконання презентації всіх конструкторських і технологічних рішень, які в сукупності.

Результатом цього етапу є технічний проект.

Детальне проектування - це етап детальної обробки блоків, вузлів та деталей системи, що проектується, а також технологічних процесів виготовлення деталей і складання їх у вузли і блоки.

Останнім етапом є виконання прототипу, за результатами випробувань вносяться необхідні зміни в конструкторську документацію.

У неавтоматизованому проектуванні найбільш трудомісткими є етапи технічного та експлуатаційного проектування. Втілення автоматизації на даних етапах призводить до більш ефективних результатів.

У процесі створення складної системи формуються деякі уявлення про систему, важливі властивості з різним ступенем деталізації, що відображають цю систему. У цих уявленнях можна виділити складові частини - рівні дизайну.

Один рівень зазвичай включає уявлення, які мають спільну фізичну основу і дозволяють використовувати один і той же математичний апарат для їх опису. Рівні проектування можна виділити за ступенем деталізації, з якою відображаються властивості проектованого об'єкта. Тоді вони називаються горизонтальними (ієрархічними) рівнями проектування.

Блочно-ієрархічний підхід до проектування полягає в поділі на горизонтальні рівні . Горизонтальні рівні характеризуються такими особливостями:

– при переході від певного рівня $К_1$, на якому розглядається система S, до сусіднього, більш низького рівня $К_2$, система S розбивається на блоки і замість системи S розглядаються її окремі блоки;

– розгляд кожного з блоків на рівні $К_2$ з більшим ступенем деталізації, ніж на рівні $К_1$, призводить до завдань приблизно однакової складності з точки зору людського сприйняття та можливості вирішення з використанням доступних засобів проектування;

– застосовуючи його концепції системи та елемента на кожному рівні ієрархії, тобто якщо елементами проектованої системи S вважалися блоки S_k, то на сусідньому, нижчому рівні $К_2$ ті ж блоки S_k вже вважаються системами.

Рівні проектування також можна відрізнити від властивостей об'єктів, які враховуються природою. У цьому випадку вони називаються вертикальними дизайнерськими рівнями.

При проектуванні пристроїв автоматизації основними вертикальними рівнями є функціональний (схемний), конструктивний і технологічний проекти.

При проектуванні автоматизованих комплексів до цих рівнів додається алгоритмічне проектування (програмне забезпечення).

Функціональне проектування передбачає розробку структурних, функціональних і принципових схем. Функціональний проект визначає основні особливості будови, принципи роботи, найважливіші параметри та особливості створюваних об'єктів.

Алгоритмічне проектування пов'язане з розробкою алгоритмів функціонування ЕОМ і комп'ютерних систем (КС), зі створенням їх загального системного і прикладного програмного забезпечення.

Конструктивне проектування охоплює питання реалізації результатів функціонального проектування, тобто вибір форм і матеріалів вихідних деталей, вибір типорозмірів уніфікованих деталей, просторове розташування компонентів, що забезпечується врахуванням взаємодії між дизайном елементів.

Технологічний проект охоплює питання впровадження результатів конструктивного проектування, тобто розглядаються питання створення технологічних процесів виробництва виробів.

На етапі НДР для проведення наукових досліджень і експериментів рекомендується використовувати спеціальні системи автоматизації. Ці системи використовують багато елементів математичного та САПР для підтримки інших етапів проектування.

Залежно від послідовності реалізації етапів проектування розрізняють проектування «зверху вниз» і «знизу вверх».

Проектування знизу вгору (проектування знизу вгору) характеризується вирішенням завдань нижніх рівнів ієрархії перед вирішенням завдань вищих рівнів.

Дизайн зверху вниз (дизайн зверху вниз) — це зворотна послідовність.

В даний час проектування складних пристроїв, їх вузлів і вузлів здійснюється на різних підприємствах з використанням різних програм САПР, у тому числі типових, наприклад САПР для проектування електронної та обчислювальної техніки, САПР для проектування електричних машин.

Функціональне проектування в САПР включає два великих горизонтальних рівня - системний і функціонально-логічний (рис. 1.3). Дизайн зверху вниз зазвичай використовується для виконання завдань на цих рівнях.

На системному рівні проектуються структурні схеми пристроїв. На цьому рівні вся система агрегації розглядається як єдине ціле, а елементами системи є такі пристрої, як процесори, канали зв'язку, різні датчики, старанні пристрої та ін.

Функціональні та принципові схеми пристроїв розроблені на функціональному та логічному рівнях. Тут є підрівні - реєстровий і логічний:

– Блокові пристрої розроблені на регістровому підрівні.

– На логічному рівні пристрої або їх складові блоки складаються з окремих логічних елементів.

На схемотехнічному рівні проектуються принципові електричні схеми пристроїв. Компонентами тут є елементи електронних схем.

На рівні компонентів розробляються окремі компоненти пристрою, які розглядаються як системи, що складаються з елементів.

Функціональне проектування в САПР може бути як зверху вниз, так і зверху вниз.

Вищі, ієрархічні рівні алгоритмічного проектування використовуються для створення комп'ютерного програмного забезпечення. Для складних програмних систем зазвичай виділяють два ієрархічні рівні. На найвищому з них відбувається планування програмної системи та розробляються діаграми алгоритмів; програмні модулі є елементами схем. На наступному рівні ці модулі програмуються на деякій алгоритмічній мові. Тут використовується дизайн зверху вниз.

Архітектурне проектування - вибір архітектури системи, тобто визначення таких структурних і алгоритмічних особливостей , як формати даних і команд, система команд, правила роботи, умови виникнення і дисципліна обробки переривань і т.д.

мікропрограмування призначене для розробки мікропрограм для операцій і процедур, що виконуються на комп'ютері апаратним способом. Цей рівень тісно пов'язаний з функціонально-логічним рівнем проектування.

Основними завданнями системного та архітектурного рівнів проектування є:

– визначення принципів організації системи;

– вибір архітектури, специфікація функцій системи та їх поділ на апаратні та програмні функції;

– розробка конструктивної схеми, тобто визначення складу пристроїв і способів їх взаємодії;

– визначення вимог до вихідних параметрів пристроїв та складання технічних завдань (ТЗ) на розробку окремих компонентів системи.

Технічні умови для розробки окремих пристроїв САПР включають:

– перелік функцій, які виконує пристрій;

– умови роботи пристрою,

– вимоги до його вихідних параметрів, дані про зміст і форму інформації, якою цей пристрій обмінюється з іншими пристроями системи.

Крім того, на етапі функціонального проектування пристроїв вже відоме прийняте на етапі ескізного проектування рішення щодо характеру основи елемента.

На логічному підрівні рівня функціонально-логічного проектування вирішуються наступні задачі:

– синтез функціональної та принципової схем виділених блоків;

– перевірка працездатності синтезованих блоків з урахуванням затримок і сигнальних обмежень обраної бази елементів або розробка вимог до елементів у складі САПР;

– синтез контрольно-діагностичних тестів;

– Формулювання ТК для рівня проектування схеми.

Основною частиною ТЗ на рівні схемотехніки є вимоги до вихідних параметрів електронних схем: затримки поширення сигналу, потужності розсіювання , рівні вихідної напруги, запас опору тощо. Крім того, ТЗ визначає умови експлуатації у вигляді вказівки допустимих діапазонів зміни зовнішніх параметрів (температури, напруги живлення тощо).

На рівні схемотехніки основні завдання проектування такі:

– синтез структури принципової схеми;

– розрахунок параметрів пасивних елементів та визначення вимог до параметрів активних елементів;

– обчислення ймовірності виконання вимог ТЗ за вихідними параметрами;

– Формулювання ТЗ для конструкції компонентів.

На компонентному рівні завдання функціонального, конструктивного і технологічного проектування тісно пов'язані між собою. Це:

– вибір фізичної структури та розрахунок параметрів напівпровідникових елементів;

– вибір топології елемента та розрахунок геометричних розмірів;

– розрахунок електричних параметрів і характеристик компонентів;

– розрахунок параметрів технологічних процесів, що забезпечують бажаний кінцевий результат;

– розрахунок ймовірності виконання вимог до вихідних параметрів елементів і пристроїв.

При низхідному проектуванні поєднання ієрархічних рівнів проявляється у створенні технічних специфікацій на розробку елементів з урахуванням вимог системи.

У проектуванні «знизу вгору» розробка елементів передує розробці системи, тому ТЗ для елементів зазвичай створюються на основі думки експертів на тому ж рівні, на якому ці елементи розробляються. Зв'язок між рівнями проявляється насамперед у тому, що при проектуванні системи враховуються властивості вже спроектованих елементів шляхом використання макромоделей елементів.

Конструктивне проектування включає в себе ієрархічні рівні проектування стійок , панелей, типових змінних елементів (ТЕЗ). Дизайн зверху вниз характерний для вирішення проектних завдань.

Конструктивне проектування включає рішення завдань із груп: проектування розподільних пристроїв і установок; забезпечення прийнятних теплових режимів; проектування електромеханічних вузлів зовнішніх пристроїв; підготовка проектної документації.

Основні завдання проектування та складання комутаторів у CAD включають розміщення компонентів на підкладці та відстеження електричних з’єднань між компонентами. Ці завдання вказані в наступному списку:

– проектні розрахунки геометричних розмірів елементів (це завдання іноді вважають функціональним завданням на проектування);

– визначення взаємного розташування компонентів на конструктивному елементі;

– розташування компонентів на елементі конструкції з урахуванням геометрії пристрою, схемних і технологічних обмежень;

– відстеження дзвінків;

– підготовка креслень із зображенням загального вигляду пристрою та визначенням основних габаритних розмірів.

Завдання розміщення елементів і простежуваності електричних з'єднань також вирішуються в програмному забезпеченні САПР електронних пристроїв RSAD. Таким чином, на рівні типових змінних елементів (ТЕЗ) необхідно розташувати корпуси мікросхем і прокласти друковані дроти в один або кілька шарів друкованої плати. Крім того, завдання проектування роз'ємів і інсталяцій включають завдання компонування елементів у блоки.

Виготовлення проектної документації включає в себе автоматичне оформлення результатів проекту вищевказаного. завдання в необхідній формі (наприклад, у вигляді малюнків, схем, таблиць тощо). Так, для отримання фотооригіналів друкованих плат і фотошаблонів інтегральних схем (ІС) зараз використовується програмно-кероване обладнання – координатні епюри та фотонабірні пристрої.

1.3. Автоматизація проектування

Завдання, які вирішуються на кожному етапі блочно- ієрархічного проектування, поділяються на задачі синтезу та аналізу (рис. 1 , 2). Завдання синтезу пов'язані з отриманням варіантів конструкції, аналітичні – з їх оцінкою.

Синтез — це створення опису об'єкта, який відповідає заданим функціям і відповідає заданим обмеженням.

Задача синтезу виконується в обраному класі елементарних об'єктів, що утворюють об'єкт, що реалізує заданий клас функцій.

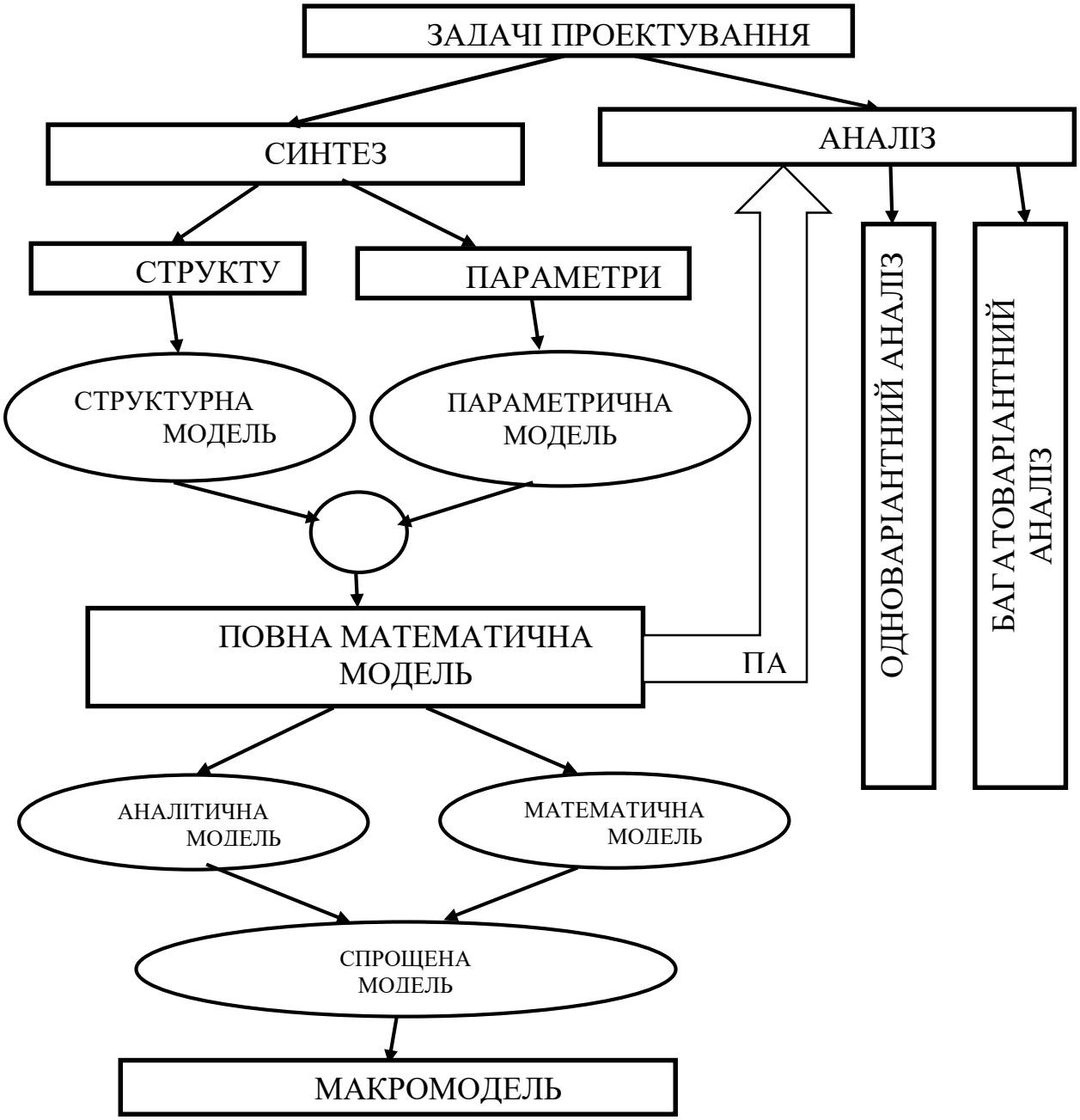


Рисунок 1.3 - Рішення проектних завдань

Вихідні дані - опис функцій, які виконує проектований об'єкт; перелік параметрів якості та їх межі.

Результатом є деяка структура, яка реалізує заданий клас функцій.

Під структурою об'єкта розуміють сукупність

$$S = \{C,H\},$$

де С – набір елементів, що входять до структури об’єкт, а Н – набір асоціацій між ними.

Рівноправні структури – такі структури, які виконують однакові функції ($F_1 = F_2$), складаються з однакових елементів ($\{C_1\} = \{C_2\}$) і з'єднані однаковими зв'язками ($\{H_1\} = \{H_2\}$).).

Еквівалентними структурами є структури, де $F_1 = F_2$, але $C_1 <> C_2$ та/або $H_1 <> H_2$.

Задача синтезу може мати формальні методи розв'язування – така задача алгоритмічно розв'язувана, в іншому випадку вона алгоритмічно нерозв'язувана. Алгоритмічно - нерозв'язні задачі розв'язуються вручну або евристичними методами (повний перелік).

Розрізняють структурний і параметричний синтез.

Метою структурного синтезу є отримання структурних схем об'єкта, які містять інформацію про склад елементів і спосіб їх зв'язку.

Оптимізація - це визначення найкращих в певному сенсі значень структури і (або) параметрів.

Оптимізація, пов'язана виключно з параметричним синтезом, тобто з обчисленням оптимальних значень параметрів для заданої структури об'єкта, називається параметричною оптимізацією. Завдання вибору оптимальної структури полягає в оптимізації структури.

Завданнями аналізу при проектуванні є завдання дослідження моделі проектованого об'єкта.

Аналіз полягає у визначенні функціонального та параметричного опису системи на основі заданого структурного опису.

Метою вирішення аналітичної задачі є дослідження властивостей описів F, S і P, отриманих на певному етапі сходження з дерева рішень проекту. Метою такого дослідження є оцінка якості отриманого варіанту рішення або перевірка F-описів на відповідність заданому.

На відміну від задачі синтезу, задача аналізу завжди вирішується алгоритмічно. Твердження є правильним, оскільки вже отримано варіант вирішення задачі синтезу і відомі принаймні відповідні описи F і S.

Задача аналізу вирішується моделюванням.

Найбільш загальними методами аналізу є одновимірний (дослідження об'єкта в даній точці траєкторії поведінки) і багатовимірний (дослідження властивостей об'єкта в околицях даної точки траєкторії поведінки).

Результатом аналізу є визначення адекватності.

Адекватність є показником відповідності моделі аналізованому об'єкту.

Формалізація задачі на проектування є обов’язковою умовою її вирішення на ЕОМ. (Рис. 1.4)

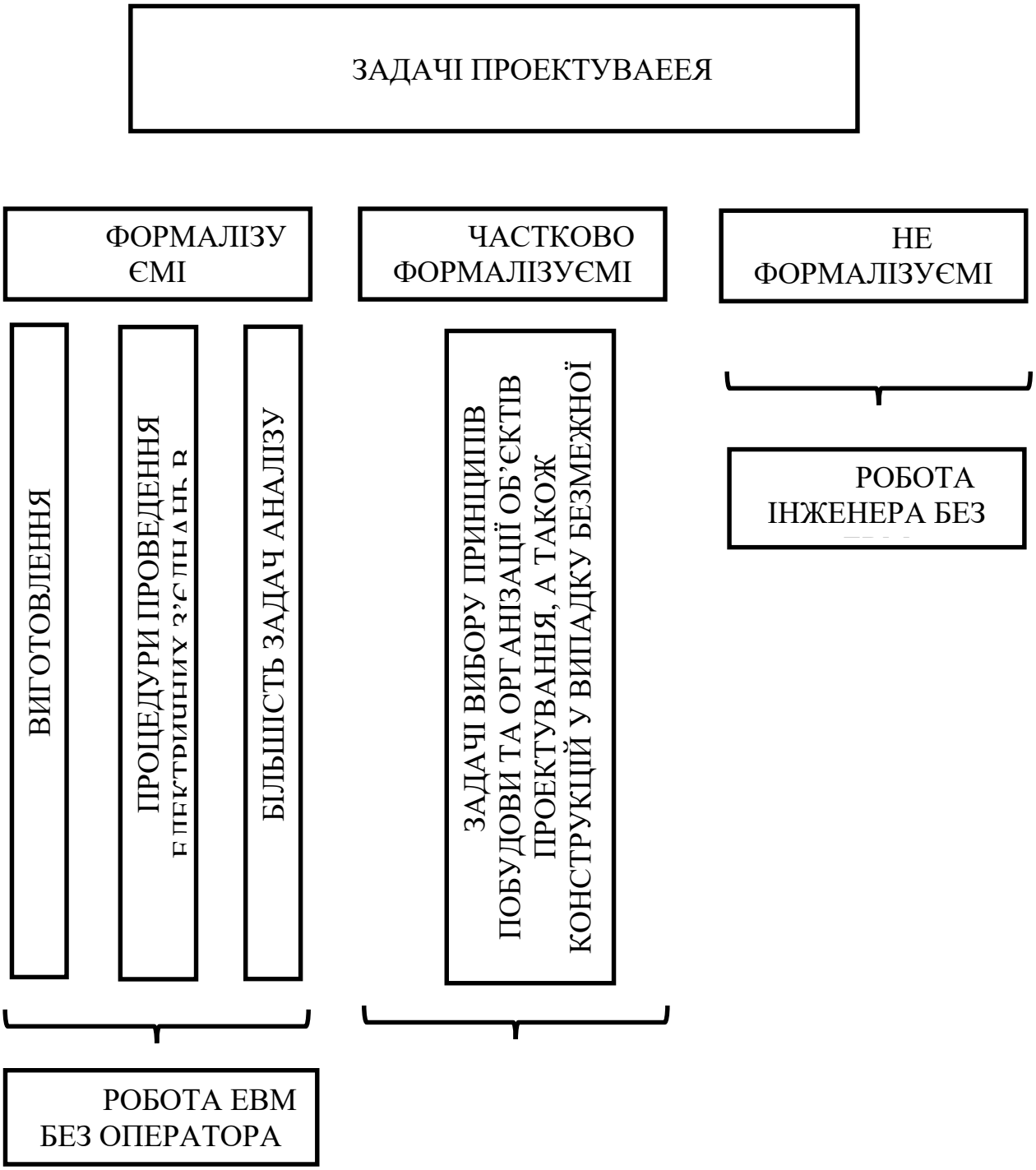


Рис 1.4. Формалізація проектних завдань

Формалізовані завдання - це насамперед завдання, які завжди вважалися рутинними і не потребують значних творчих зусиль інженерів. Ось процедури:

– складання конструкторської документації (КД) в умовах, в яких зміст КД вже повністю встановлено, але ще не має форми, прийнятої для зберігання та подальшого використання (наприклад: форма креслень, схем, схем, алгоритмів, зв'язків). , таблиці);

– процедури виконання електричних з’єднань на друкованих платах або виготовлення фотоформ у поліграфії.

Крім рутинних завдань, формальні завдання включають більшість завдань, пов'язаних з аналізом проектованих об'єктів. Їх формалізація досягається розвитком теорії та методів автоматизованого проектування перед будь-яким моделюванням.

Перша група завдань - це повністю формалізовані завдання, найчастіше вони виконуються на комп'ютері, без втручання людини в процес прийняття рішень.

Частково формалізовані завдання

У той же час існує багато проектних завдань творчого характеру, для яких методи формалізації невідомі. Ці завдання пов'язані з вибором правил побудови і організації об'єкта, синтезом схем і конструкцій в умовах, коли вибір варіантів здійснюється з необмеженої кількості варіантів, і можливістю отримання нових, раніше невідомих рішень. не виключено.

Друга група завдань – це частково формалізовані завдання, які виконуються на комп’ютері за активної участі людини, тобто робота з комп’ютером відбувається в інтерактивному режимі.

Неформальні задачі, які складають третю групу задач, розв'язуються інженером без допомоги комп'ютера.

В даний час одним із напрямків розвитку математичного забезпечення автоматичного проектування є розробка методів синтезу та алгоритмів на різних рівнях ієрархічного проектування.

Моделювання – у цьому контексті воно визначає процес вимірювання на моделях властивостей, необхідних для оцінки якості проектного рішення на конкретному рівні проектування.

Моделі можуть бути фізичними (різні моделі, стелажі) і математичними.

Математична модель — це сукупність математичних об’єктів (чисел, змінних, векторів, множин тощо) і зв’язків між ними.

Модель відображає деякі властивості природи, але вона завжди відрізняється від природи. Модель - це найчистіше знання предмета.

У САПР використовуються наступні типи моделей:

– **Імітаційна модель** Застосовуючи деякі правила, імітується робота РО;

– **Алгоритмічна модель** - імітаційна модель, описана за допомогою алгоритмів і алфавіту (набору правил і синтезу);

– **Аналітична модель** - імітаційна модель, представлена системою рівнянь, що описують задану функцію функціонування об'єкта;

– **Функціональна модель** - описує функції, які виконує система в цілому або функції окремих елементів і представляє фізичні або інформаційні процеси, що відбуваються в модельованому об'єкті (зазвичай це системи рівнянь);

– **Структурна модель** відображає структуру, елементи та зв'язки між ними і показує лише структурні (у деяких випадках геометричні) властивості об'єктів (як правило, графів, матриць тощо).

Нехай об'єкт проекту (ОП) характеризується виразом типу

$$OP = \{F, S, P\},$$

де F, S і P – функціональний, структурний і параметричний описи об’єкта відповідно.

Функціональний опис відображає траєкторію ПО у просторі часових станів як певну функцію, аргументами якої є керуючі дії та пасивні дії зовнішнього середовища. Керуючі дії можуть бути як зовнішніми, так і внутрішніми, тобто сформованими за певними правилами роботи, прихованими всередині об'єкта.

Як частина формального визначення завдання проекту пов’язане з декомпозицією вихідних F-описів на певні підфункції (компоненти): F0 =S(Fj1), j =1,2, ., n, де S – оператор визначення такого складу Fi $_{1\ ,\ який}$ забезпечує попередній функціональний опис (F_0). Оператор S називається структурним описом (S description) і визначає структуру РО на цьому рівні деталізації. Відомі об'єкти, які називаються елементами, можуть відповідати деяким функціональним описам компонентів (Fi), отриманих шляхом декомпозиції. Елемент може бути досить складною технічною системою. У цьому випадку важливо, що під час проектування F-опис елемента не потребує подальшої декомпозиції, і тому він не має S-описів.

Для решти Fi знову потрібна декомпозиція і так далі, $_{поки}$ всі описи F не будуть відповідати елементам.

Вибір варіанту розкладання зазвичай визначається якістю отриманого розчину.

Нехай якість — це набір властивостей ОР, які називаються параметрами P={ p_i }, i =1,2, ., k . При цьому p_i є обчислюваними функціями, значення аргументів яких визначаються параметричними описами термів наступного рівня (P_j +1), оскільки інші рівні ще не визначені. Підкреслення курсивом слова «розраховується» означає, що значення параметрів обов’язково має бути скалярним значенням. В іншому випадку значення параметрів неможливо порівняти, тобто до них можна застосувати реляційні операції.

Абстракція є одним із найпотужніших інтелектуальних способів розуміння складних явищ, доступних людині.

Парадигма (вихідна точка) абстракції полягає у визначенні подібності певних об'єктів, ситуацій або процесів і вирішенні підкреслити істотні ознаки, тимчасово ігноруючи відмінності.

Таким чином, під час абстрагування можна виділити такі етапи:

– Абстрагування - вибір значущих властивостей для розгляду;

– Специфікація - визначення символічного представлення абстрактних понять:

– Перетворення - визначення скінченного набору операцій над символічним представленням з метою прогнозування поведінки реальних об'єктів відповідно до абстракції;

– Аксіоматизація - це суворе формулювання властивостей об'єктів і правил перетворення.

Для кожної нової версії конструкції модель має бути адаптована або зібрана заново, а параметри оптимізовані.

Процес проектування повторюється. Ітерації можуть охоплювати більше одного рівня проекту. Таким чином, в процесі проектування необхідно багато разів проводити процедуру аналізу об'єкта. Тому існує очевидне бажання зменшити складність кожного варіанту аналізу без шкоди для якості остаточного дизайну. За цих умов найпростіші та економічні моделі доцільно використовувати вже на початкових етапах процесу проектування, коли не потрібна висока точність результатів. На завершальному етапі використовуються найбільш точні моделі, проводиться багатофакторний аналіз, завдяки якому отримують достовірні оцінки роботи об'єкта.

Процедура синтезу об’єкта – це набір процедур для синтезу структури, складання моделі та оптимізації параметрів.

Суть цього методу полягає у використанні циклу, який полягає в послідовному чергуванні етапів генерації чергового варіанту та аналізі його якості (Р-описів) у міру наближення до бажаного результату. Такий аналіз дозволяє визначити «напрямок», в якому можна отримати S-описи з більшим ступенем узгодження з бажаним результатом, а протилежний напрямок можна виключити з розгляду.

Процедури (алгоритми) розв’язування задачі синтезу евристичними методами, у тому числі методом послідовних наближень, зазвичай називають ітераційними. Ітераційні алгоритми послідовно створюють варіанти рішення і можуть бути перервані на будь-якому етапі ітерації, якщо в результаті аналізу якості результуючого варіанту буде визначено, що варіант прийнятний. Проте конструктивний алгоритм на основі методу послідовних наближень можна

побудувати, якщо ознаки «відсікання» безперспективних варіантів достатньо сильні, щоб дати єдине рішення.

Оптимальними вважаються ті, які відповідають ТЗ і є найкращими для досягнення.

Завдання оптимізації САПР зводиться до перетворення фізичного представлення об'єкта, його призначення та ступеня корисності в математичну формулювання екстремальної задачі. Мета оптимізації виражається в критеріях оптимізації.

Критерії - правила переваги порівнюваних варіантів. Основою критеріїв оптимізації є цільова функція F(x), де x – множина контрольованих параметрів. Вектори x з фіксованими значеннями визначають один з варіантів об'єкта і його характеристик.

Цільова функція повинна бути такою, щоб за її значеннями можна було визначити ступінь досягнення мети, тобто найкращий варіант повинен характеризуватися великим значенням F(X), тоді оптимізація полягає в максимізації F (X) або навпаки, з мінімізацією F(X) найкращий варіант повинен мати менші значення параметрів.

Крім цільової функції F(X) і списку контрольованих параметрів (X), у постановці оптимізаційної задачі можуть бути враховані обмеження типу рівності H(X) = 0 і нерівності H(X)<> 0 . Прямі обмеження $a_i <= x_i <= b_i$, де a_i і b_i є максимально можливими значеннями $x_{i,\epsilon}$ частковим випадком обмежень нерівності .

Об'єкт називається строго оптимальним, якщо значення всіх параметрів знаходяться в межах допустимого діапазону значень параметрів.

Об'єкт називається квазіоптимальним, якщо деякі параметри вектора (X) виходять за межі, але при цьому межі повинні бути строго визначеними.

За наявності обмежень завдання оптимізації називається умовною оптимізацією, інакше – безумовною.

Область, де виконуються як безпосередні обмеження, так і умови працездатності, називається областю працездатності.

Таким чином, остаточна формулювання задачі оптимізації проектування виглядає наступним чином: екстремізація цільової функції F(X) в області XD, що визначається обмеженнями H(X) = 0 і Ф(X)> 0.

Задача оптимізації в такій установці є задачею математичного програмування. Якщо функції F(X), H(X), F(X) лінійні, це задача лінійного програмування. Якщо хоча б одна з них нелінійна, ми маємо проблему нелінійного програмування.

Задача оптимізації структури зводиться до побудови оптимальної структури S = (E, H). При цьому під оптимальним розуміється варіант конструкції, параметри якого відповідають усім системним, будівельним, технологічним, електротехнічним та економічним вимогам ТЗ. При цьому надзвичайного значення набуває критерій оптимальності, який описує якість проектованої конструкції.

При проектуванні за приватними критеріями найважливішим вихідним параметром проектованого об'єкта вважається цільова функція F(X), всі інші параметри у вигляді відповідних умов експлуатації відносяться до обмежень. У цьому випадку оптимальною задачею проектування є однокритеріальна задача математичного програмування: екстремізувати значення цільової функції F(X) за наявності системи обмежень на параметри об’єкта проектування. Складність такого завдання невелика.

Приватні критерії вибираються, коли є потреба порівняти кілька еквівалентних рішень або існує заздалегідь визначена потреба оптимізувати один чи більше приватних критеріїв (без значних обмежень щодо інших критеріїв).

Якщо у функції можна задати характеристики всіх необхідних критеріїв, то такий критерій називається узагальненим. В якості узагальнених критеріїв найчастіше використовуються адитивний , мультиплікативний і мінімаксний критерії .

Якщо критерій не враховує дисперсію ймовірностей параметрів, він є детермінованим, в іншому випадку він є статистичним.

У адитивних критеріях цільова функція створюється шляхом підсумовування нормалізованих значень приватних критеріїв. Нормовані критерії - це відношення фактичного значення приватного критерію до деякої нормуючої величини, виміряної в тих самих одиницях, що і сам критерій (результатом є безрозмірна величина).

Можливі кілька підходів до вибору нормуючого дільника.

Перший підхід передбачає, що нормативні дільники є максимальними значеннями критеріїв, досягнутими в рамках існуючих проектних рішень.

Другий підхід передбачає взяти в якості нормуючих дільників оптимальне значення, вказане в технічних умовах.

Третій підхід пропонує використовувати різницю між максимальним і мінімальним значенням критерію в області компромісу як нормалізуючі дільники.

Цільова функція: $F(x) = \Pi Fi(x)$

Недоліки: формальне сприйняття, не випливає з об'єктивної ролі приватного критерію; можлива взаємна компенсація приватних критеріїв.

Мультиплікативний критерій. Іноді при вирішенні якоїсь задачі важливо враховувати не абсолютне значення критерію, а його зміну.

Цільова функція : $F(x) = \Pi F_i(x)$

У разі нерівності приватних критеріїв введіть ваговий коефіцієнт C_i тоді мультиплікативний критерій набуде вигляду:

$$F(x)=\prod_{i=1}^{n} F_i^{C_i n}(x) \quad \text{Або} \quad F(x)=\prod C_i F_i(x)$$

Перевагою мультиплікативного критерію є те, що його використання не потребує нормалізації приватних критеріїв.

Недоліком є те, що критерій може компенсувати надмірні зміни одних критеріїв у результаті змін інших.

Формально принцип максиміну формулюється так:

Слід вибрати таку множину *X0* е X, на якій реалізується максимальне з мінімальних значень приватних критеріїв $F(x_0) = \max \min \{c_i(x)\}$.

Якщо приватні критерії $f_i(x)$ *повинні бути мінімізовані, тоді* застосовується правило мінімаксу

$$F(x0) = мін.\ max\ \{fi(x)\}.$$

Адитивні критерії вибираються, коли абсолютні числові значення критеріїв з обраним вектором *X мають значуще значення* .

вектора відіграють зміни абсолютних значень приватних критеріїв X, то доцільно використовувати мультиплікативний критерій.

Якщо стоїть завдання досягти рівності нормованих значень суперечливих приватних критеріїв, то оптимізацію слід проводити за максимальним критерієм (мінімакс).

Метод оцінки – кожному параметру експерти присвоюють бали в діапазоні {1,10}. Також можливі дробові значення і ті ж оцінки.

Тоді $H_i, k = h_i, k /((i = 1, n)\ h_i, k$;

$c_i = (((k=1, L)\ H_i, k) /(((i =1, n)\ ((k=1, L)\ H_i, k)$

Метод ранжування - кожен з *I* експертів ранжує n критеріїв (у порядку зменшення важливості). На основі цієї оцінки кожному параметру присвоюється ранг n - 1. Це значення називається перетвореним рангом i -го критерію, тоді

$$c\ i = ((k =1, L)(ri, k /((i =1, n)((k =1, L)\ ri, k).$$

Серед властивостей об'єкта, відображених в описах на певному рівні ієрархії, виділяють властивості систем, компонентів системи та зовнішнього середовища, в якому об'єкт повинен функціонувати.

Параметри - це кількісне вираження величин, що відображають властивості систем в цілому, компонентів систем і зовнішнього середовища, в якому повинен функціонувати об'єкт.

Параметри відповідно вихідні, внутрішні та зовнішні.

Позначимо кількість початкових параметрів - внутрішніх і зовнішніх через m, n, t, а вектори цих параметрів через $Y = (y_1, y_2, .., y_m)$, $X = (x_1, x_2, .., x_n)$, $Q = (q_1, q_2, .., q_t)$. Очевидно, що властивості системи залежать від внутрішніх і зовнішніх параметрів, тобто існує функціональна залежність

Система відношень $F = (y, x, t)$ є прикладом математичної моделі (ММ) об’єкта. Наявність такого ММ полегшує оцінку початкових параметрів на основі відомих значень векторів Y і X. Однак існування зв'язку не означає, що воно відоме програмісту і може бути точно представлене в такому однозначний вигляд по відношенню до векторів Y і X. Як правило, отримати математичну модель можна тільки для дуже простих об'єктів.

Слід відзначити наступні особливості модельних параметрів проектованих об'єктів.

Внутрішні параметри (параметри елементів) в моделях *k-го* рівня ієрархії стають вихідними параметрами в моделях нижнього (k +1)-го рівня ієрархії. Таким чином, у випадку електронного підсилювача параметри транзистора є внутрішніми під час проектування підсилювача та одночасно виводяться під час проектування самого транзистора.

Вихідні параметри, тобто фазові змінні, що з'являються в моделі однієї з підсистем (в одному аспекті опису), часто виявляються зовнішніми параметрами в описах інших підсистем (інших аспектів). Таким чином, максимальні температури корпусів електронних пристроїв в електричних моделях підсилювача відносяться до зовнішніх параметрів, а в теплових моделях цього ж об'єкта - до початкових параметрів.

Попередні описи проектованих об'єктів часто є КТ на проектування. Ці описи містять значення, які називаються технічними вимогами та вихідними параметрами (стандартами вихідних параметрів). Технічні вимоги утворюють вектор $TT = (TT_1 , TT_2 ,..., TT_n)$, де значення TT є межами діапазонів зміни вихідних параметрів.

1.4. Автоматизована система проектування

Система автоматизованого проектування (САПР) — це набір засобів і методів автоматизованого проектування.

САПР складається з кількох складових частин, які називаються витратними матеріалами (рис. 1.5).

Технічна підтримка САПР — це набір взаємопов’язаних і сумісних технічних засобів, призначених для виконання завдань автоматизованого проектування.

Технічна підтримка поділяється на групи заходів:

– програмне забезпечення обробки даних, яке представлено процесорами і пристроями пам'яті, тобто комп'ютерними пристроями, в яких здійснюється перетворення даних і програмне управління обчисленнями;

– підготовка та введення даних, а також

– відображення та документування, вони використовуються для зв'язку комп'ютера із зовнішнім середовищем і з оператором;

– архівація проектних рішень, представлених зовнішніми носіями інформації, базами даних;

– передача даних використовується для організації зв'язків між територіально рознесеними комп'ютерами і терміналами (прикордонними пунктами).

Програмне забезпечення САПР поєднує власне програми засобів обробки даних на автоматизованих носіях і програмну документацію, для роботи програми.

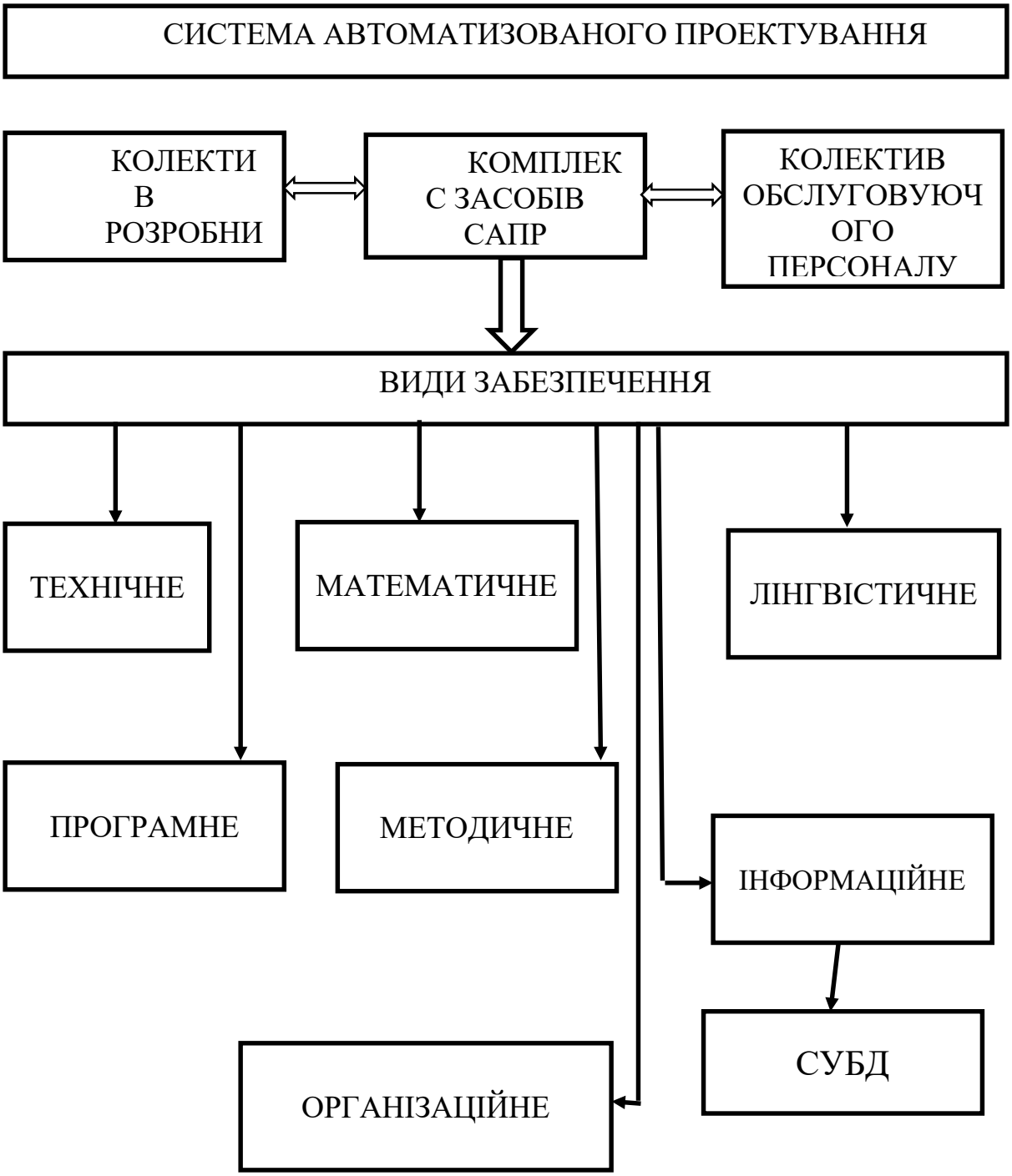


Рис. 1. 5 - Компоненти САПР

Програмне забезпечення (ПЗ) поділяється на загальносистемне, базове та прикладне (спеціальне). Системне програмне забезпечення призначається для організації роботи технічних засобів, отже для планування і керованя обчислювальним процесом, розподілу певних ресурсів і представлене комп'ютером і операційними системами комп'ютера. Системне програмне забезпечення зазвичай розробляється для багатьох застосувань і не відображає специфіки САПР. Для цілей САПР створюється базове та прикладне

програмне забезпечення. Основне програмне забезпечення включає програми, які забезпечують правильну роботу прикладних програм.

Прикладне програмне забезпечення реалізує математичне забезпечення безпосереднього виконання проектних процедур. Прикладне програмне забезпечення зазвичай має форму пакетів прикладних програм (APP), кожен з яких обслуговує певний крок у процесі проектування або групу завдань одного типу в межах різних кроків.

Інформаційне забезпечення САПР об'єднує всі види даних, необхідні для автоматизованого проектування.

Визначені дані можуть бути зображені у вигляді тих чи інших документів на різних носіях, що містять довідкову інформацію про матеріали, комплектуючі вироби, типові конструктивні рішення, параметри елементів, зведення про поточний стан розробки у вигляді проміжних і остаточних проектних рішень, конструкції та параметри проектованих об'єктів тощо. Основною частиною інформаційного забезпечення САПР є банк даних, який є сукупністю засобів централізованого збору та колективного використання даних у САПР. Банк даних (BND) складається з бази даних і системи управління базами даних.

База даних (БД) - самі дані, що зберігаються в запам'ятовуючих пристроях комп'ютера і впорядковані за правилами, прийнятими в цій базі даних. Система керування базами даних (СУБД) — це комплекс програмних засобів, що забезпечують функціонування бази даних. При використанні РСУБД дані зберігаються в базі даних, здійснюється їх відбір за запитом користувачів і прикладних програм, забезпечується захист даних від спотворення, несанкціонованого доступу тощо

Підтримка мови САПР представлена набором мов, які використовуються для опису процедур автоматизованого проектування та проектних рішень.

Основною частиною мовного забезпечення є мова спілкування людини з комп’ютером.

Методичне забезпечення САПР складають документи, що характеризують склад, правила вибору та функціонування засобів автоматизованого проектування.

Припустимим є більш широке тлумачення поняття методичного забезпечення, в якому під методичним забезпеченням розуміють сукупність математичного та лінгвістичного забезпечення та іменованих документів, що реалізують правила використання засобів проекту.

Організаційне забезпечення САПР включає положення, інструкції, накази, штатні розклади, вимоги кваліфікації та інші документи, що визначають організаційну структуру підрозділів проектної організації та їх взаємодію з комплексом засобів автоматизованого проектування.

Спеціалізація деякої частини САПР для реалізації проектних завдань одного етапу проектування призводить до виділення цієї частини як підсистеми САПР. Така спеціалізація стосується програмного забезпечення, математичного забезпечення, мовної підтримки, а іноді й технічної підтримки. Як правило, в САПР пристроїв автоматизації є підсистеми функціонально-логічного проектування та структурного проектування.

У САПР виділяють підсистеми проектування схем, проектування компонентів і структурного (топологічного) проектування.

Складність сучасних технічних об’єктів зумовлює появу в САПР підсистем, які виконують функції незалежних, автоматизованих систем проектування (САПР електронних систем, САПР електричних машин тощо).

Як правило, кожна підсистема має власну мову введення та прикладний пакет.

Сучасні САПР створені за такими принципами:

1) САПР – система людина-машина. Команда розробників є невід'ємною частиною системи проектування, що виконує проектні роботи у взаємодії з комп'ютерами.

2) Комплексна автоматизація всіх рівнів проектування

Це дає можливість впроваджувати такі зміни в структуру проектних підприємств, форми документації, які відповідають цілям автоматизації -

скорочення матеріальних і часових витрат, підвищення якості проектування, збереження чисельності інженерно-технічних працівників на поточному рівні, незважаючи на складність проектованих об'єктів.

3) Інформаційна узгодженість підсистем і програм проектування, виконується за таких умов:

– програми створюються для роботи з однією базою даних і не вимагають ручної перестановки числових таблиць, які вводяться для однієї та надсилаються для іншої з підключених програм;

– призначення початкової інформації про об'єкт або необхідні проектні операції виконується єдиною мовою введення.

4) Відкритість САПР.

Властивість відкритості системи означає можливість внесення змін до системи в процесі її функціонування. Зміни можуть складатися з додавання нових або заміни старих елементів програмного забезпечення, інформації та, можливо, також технічної та мовної підтримки. Внесення змін має бути максимально простим і доступним для користувачів САПР. Властивість відкритості призводить до продовження терміну служби системи і підвищує її універсальність.

5) Сумісність традиційного та автоматизованого проектування.

Цей принцип актуальний у випадках, коли автоматизоване проектування реалізується на вже діючому підприємстві з існуючою структурою, відомчими зв'язками, формами і методами використання проектної документації. Саме в цих умовах доцільним є еволюційний шлях впровадження САПР, при якому зміни, продиктовані характеристиками автоматизованого проектування, протягом тривалого часу не порушуватимуть нормальне функціонування підприємства.

Для технічної підтримки CAD застосовуються такі вимоги:

– зручність використання інженерами-конструкторами, можливість оперативної взаємодії інженерів з ЕОМ;

– достатня продуктивність і обсяг оперативної пам'яті комп'ютера для вирішення завдань усіх етапів проектування в прийнятний час;

– можливість одночасної роботи з технічними засобами необхідної кількості користувачів для ефективної роботи всього колективу програмістів;

– відкритість комплексу технічних засобів до розширення та модернізації системи в міру вдосконалення та розвитку техніки;

– висока надійність, прийнятна вартість і так далі

Виконання вищезазначених вимог можливе лише за умови організації технічного забезпечення у вигляді спеціалізованої системи ППО, що забезпечує роботу в декількох режимах. Таке технічне забезпечення називається комплексом технічних засобів САПР (комплекс ТС).

Форма використання комп’ютера, при якій обчислювальні ресурси зосереджені в центрі обробки даних і працюють виключно в пакетному режимі, не підходить для сучасних САПР.

Тоді комп’ютер стає ефективним, регулярно використовуваним інструментом проектування, коли інженер має можливість швидко звертатися до машини та так само швидко отримувати результати рішення .

Тому в комплексі ТС має бути розроблена група зовнішніх пристроїв введення-виведення інформації. При цьому ефективна взаємодія інженера з комп’ютером буде забезпечена лише тоді, коли форма введення та виведення інформації буде зручною для людини і не призведе до необхідності ручного виконання громіздких і помилкових операцій кодування або декодування. повідомлення .

Залежно від характеру завдань, що вирішуються, зручними формами подання інформації можуть бути таблиці, малюнки, діаграми, текстові повідомлення тощо.

Так, перша із зазначених на початку розділу вимог до технічних засобів САПР передбачає включення до складу комплексу ТС як стандартного набору зовнішніх обчислювальних пристроїв, так і додаткових пристроїв оперативного введення та виведення інформації, у тому числі в графічній формі. . Цей комплект зовнішніх пристроїв встановлюється в приміщеннях

конструкторського відділу і називається автоматизованим робочим місцем конструктора (АРМ).

АРМ — це комплекс спеціальних технічних пристроїв, які за підтримки програмних, інформаційних та мовних засобів забезпечують вирішення конкретних завдань проекту.

Склад АРМ залежить від характеру завдань, які виконуються в проектному підрозділі.

Наявність багатьох БТР в одній САПР, можливість одночасної роботи на обладнанні БТР декількома користувачами та розподіл БТР за зонами конструкторських підрозділів диктують необхідність ієрархічної побудови бригади ТЗ з виділенням не менше двох рівнів комп’ютерів.

На верхньому рівні є один або кілька високопродуктивних комп'ютерів. Ці комп'ютери утворюють центральний обчислювальний комплекс (ЦПК), призначений для вирішення складних проектних завдань, що потребують великої кількості машинного часу та пам'яті. На нижчому рівні знаходяться міні-комп’ютери (термінальні комп’ютери), які є частиною ARM. Міні-ЕОМ в АРМ керує роботою комплексу зовнішніх пристроїв, обміном інформацією між АРМ і ЦВК; вирішує відносно прості задачі проектування з точки зору споживання машинного часу та пам'яті.

Математичне забезпечення (МО) САПР – це набір математичних методів (ММет), математичних моделей (ММ) і алгоритмів проектування (АЛП), які необхідні для виконання завдань проектування і представлені в заданій формі.

Математичні опори можна розділити на два види: інваріантні МО; спеціальні МО.

Спеціальне МЗ - включається в САПР спеціально для проектування технічних об'єктів, на які орієнтована дана САПР.

Інваріантне МЗ включає в себе методи й алгоритми, слабо пов'язані з характеристиками математичних моделей і застосовувані на багатьох ієрархічних рівнях.

Вимоги до математичного забезпечення:

1) уніфікованість;
2) алгоритмічна надійність;
3) обгрунтованість математичних моделей і задач;
4) досконалість;
5) економічність.

Універсальність МЗ полягає в його застосовності в широкому класі проектованих об'єктів.

Однією з відмінностей методів розрахунку CAD від методів розрахунку вручну є високий ступінь універсальності. Наприклад, у підсистемі проектування схем CAD використовуються математичні моделі транзистора, дійсні для будь-якого робочого поля (активного, насичення, відсічення, інверсно активного), а також методи отримання та аналізу моделей для будь-якої аналогової чи комутаційної схеми. на використовуються елементи з дозволеного списку; моделі та алгоритми використовуються в підсистемі структурного проектування САПР, обробки інформації, що дозволяє досліджувати стаціонарні та нестаціонарні процеси, з довільними правами роботи в пристроях ВС та з довільними вхідними потоками.

Для того, щоб САПР можна було застосувати до будь-якого або більшості об’єктів, проектованих на підприємстві, необхідний високий ступінь універсальності МО.

Алгоритмічна надійність — це властивість компонента МЗ, яка забезпечує правильні результати при використанні.

Алгоритми, для яких неможливо чітко визначити межі застосовності, називаються евристичними. Такі алгоритми можуть бути використані неправильно , що або призведе до неправильного рішення, або рішення не буде знайдено. У САПР такі алгоритми та методи використовуються у випадках, коли ненормальні результати є очевидними.

Кількісна оцінка алгоритмічної надійності - ймовірність отримання правильного результату при збереженні певних обмежень використання методу. Якщо ця ймовірність дорівнює або близька до одиниці, то кажуть, що метод алгоритмічно надійний

Умовністю математичних моделей і задач є властивість збільшення похибок вихідних параметрів при малих похибках вхідних параметрів. Для аналізу та оптимізації об'єктів з погано визначеними математичними моделями необхідно використовувати спеціальні методи з підвищеною алгоритмічною надійністю.

Точність - ступінь збіжності розрахункових результатів з реальними. Точність оцінюється розв’язуванням спеціальних тестових завдань, покликаних підкреслити внесок кожного елемента в загальну похибку.

У більшості випадків рішення проектних завдань характеризується:

– сумісне використання багатьох компонентів МО, що ускладнює визначення внеску в сумарну похибку кожного компонента;

– векторний характер результатів (наприклад, при аналізі є вектор початкових параметрів, при оптимізації - координати екстремальної точки), тобто результатом рішення є не один параметр, а багатопараметричне значення.

Економія - витрати машинного часу та пам'яті.

Способи скорочення часу роботи машини:

1. Паралелізм обчислювального процесу;

2. Застосування макромоделювання .

Підтримка інваріантної математики включає:

1. Методи та алгоритми оптимізації;

2. Багатофакторний аналіз;

3. Логіко -комбінаторні методи прийняття рішень.

Методи та алгоритми оптимізації

Методи оптимізації відрізняються кінцевим набором показників якості

Багатовимірний аналіз

Основними методами багатовимірного аналізу є аналіз чутливості та статистичний аналіз.

Метою аналізу чутливості є визначення факторів чутливості (факторів впливу).

Коефіцієнт абсолютної чутливості визначає вплив i -го внутрішнього параметра на j-й зовнішній параметр: $a_j, i = dy_{yj} / d_{xi}$;

Коефіцієнт відносної чутливості вихідного параметра y_j до змін внутрішнього параметра x_i має вигляд: $b_j, i = a_j, i \times x_{ja\,\mathrm{nom}} / y_{j\,\mathrm{nom}}$.

Мета статистичного аналізу - отримання оцінок розкиду вихідних параметрів і ймовірності виконання заданих умов експлуатації проектованого об'єкта.

Результати статистичного аналізу такі:

- гістограма вихідних параметрів;

- оцінки математичних сподівань і середніх квадратичних відхилень кожного з уj ;

- максимально можливі відхилення;

- оцінити коефіцієнти кореляції між y_j і x_j і початковими параметрами.

Як вихідні дані використовуються відомості про розкид внутрішніх параметрів і допустимі діапазони змін або закони розподілу зовнішніх параметрів.

Підтримка мов охоплює всі мови, які використовуються в CAD. Структуру підтримки мови можна представити так (рис. 1.6)

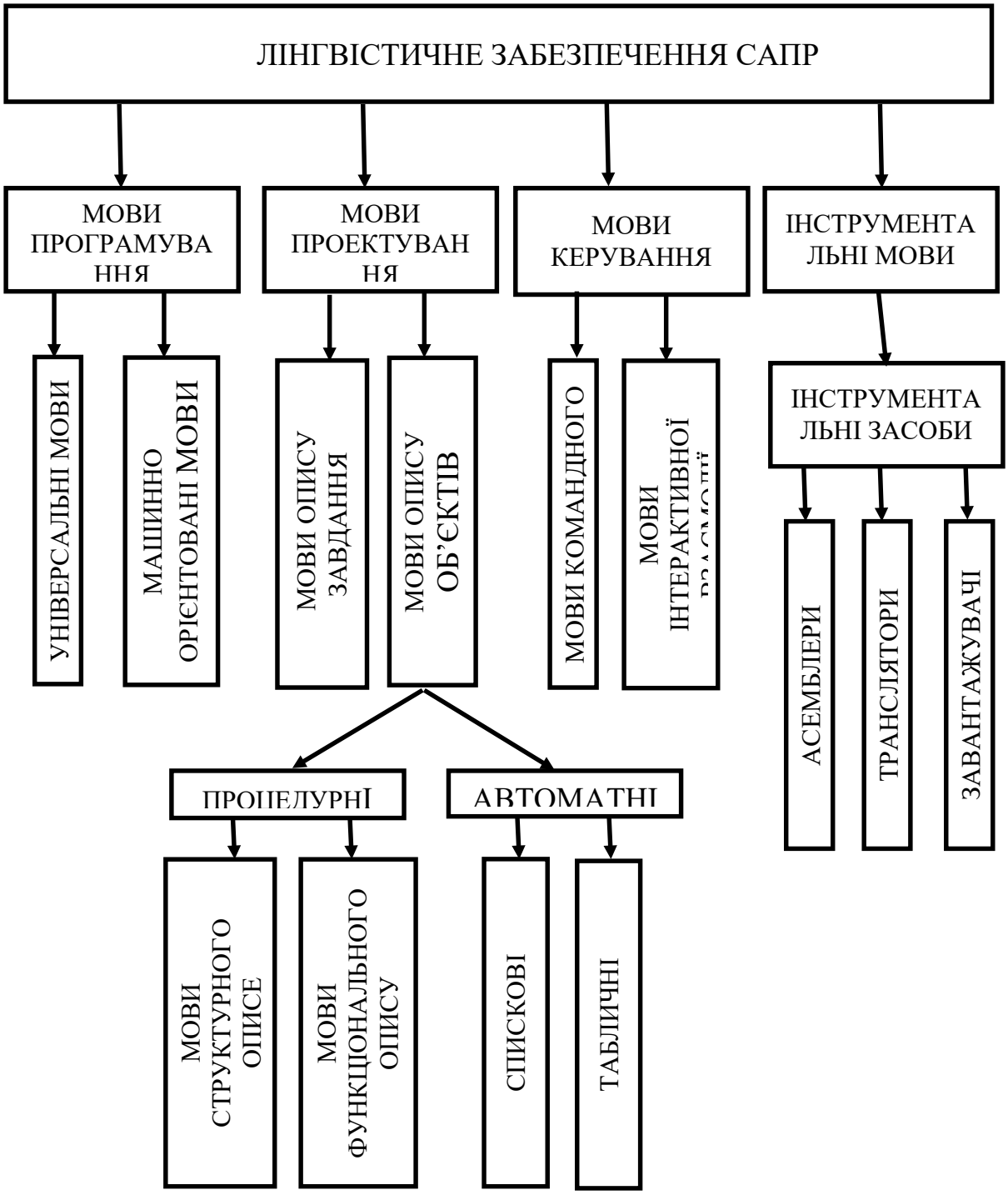


Рисунок 1.6 - Структура підтримки мови САПР

Підтримка мови поділяється на такі компоненти:

- мови програмування (універсальні, машинні);
- мови проектування (мови введення);
- мови управління;
- інструментальні мови.

Мови введення діляться на:

- мова опису завдання;

- мови опису об'єктів.

Для опису шляху проекту використовуються мови опису завдань. Подібні до операційних систем, але орієнтовані на САПР.

Мови опису об'єктів використовуються для опису функції параметрів і структурного опису об'єкта, вхідних, проміжних і результатних даних.

Вимоги до мови введення:

- універсальність;

- зручність сприйняття алфавіту та синтаксису мови;

- максимальна стисліть опису;

- однозначне тлумачення мовних елементів і конструкцій;

- можливість розвитку та розширення.

Мови опису об'єктів поділяються на:

- процедурні мови;

- автоматичні мови.

Процедурні мови:

- мови функціонального опису;

- мови структурного опису.

Функціональний опис визначає динамічні функції об’єкта (функція розвивається з часом) і може бути виражений аналітично, графічно, таблично.

Оскільки моделлю структурного опису є граф (орієнтований, мультиграф тощо), для їх завдання найчастіше використовуються графові форми представлення. Можна використовувати матриці суміжності та інцидентності , але для структурного опису реальних об’єктів ці матриці будуть по суті порожніми, що призведе до невиправданої вартості або високої складності, тому ці матриці використовуються рідко , і переважна більшість мов використовують список. структури даних.

Інструментальні мови - це особлива категорія програмних засобів, за допомогою яких створюються всі інші програми.

У категорію інструментів входять не тільки перекладачі мов високого рівня, а й асемблери, завантажувачі, відладчики та інші системні програми.

Інструменти використовуються для створення як прикладного програмного забезпечення, так і нових засобів системного програмування, включаючи перекладачі з мов високого рівня.

Вимоги до мови можна сформулювати так:

1) Усі мовні конструкції мають бути природно та просто (елегантно) визначені.

2) Для вирішення конкретного завдання повинна бути можливість використовувати комбінацію структур, щоб уникнути необхідності введення нової структури.

3) Повинна бути мінімальна кількість неочевидних структур спеціального призначення.

у програмі, яка його не використовує, додаткових витрат не виникло.

5) Достатньо, щоб користувач знав лише той набір конструкцій, які безпосередньо використовуються в його програмі.

Мови керування завданнями — це умовні мови з певного набору команд, які можна вказати в командному рядку, створити налаштування або викликати дії.

Мови керування завданнями забезпечують функції керування завданнями, а саме:

– адаптація компонентів програми для вирішення конкретного проектного завдання;

– інтерактивне управління прийняттям рішень;

– конфігурація бази даних САПР для заданого завдання проектування.

GUI - візуальне представлення мови керування завданнями.

Програмне забезпечення САПР включає:

– системне програмне забезпечення, яке зазвичай використовується операційною системою;

– тематичне програмне забезпечення, що підтримує сам процес проектування.

Засоби підтримки процесу проектування належать до класу складних програмних систем, які використовують бази даних.

Існуючі та повторювані системи САПР, як правило, страждають від ряду суттєвих недоліків, включаючи зосередженість на фізичній архітектурі систем САПР у певній предметній області.

Історично склалося, що програми САПР складаються з двох рівнів:

– рівень попередньої обробки (ARM з GUI в основі)

– завершальний рівень обробки (деяка база даних, з якою працюють деякі програми, причому база даних і програми тісно пов’язані).

Така організація програми не дуже гнучка і погано підтримується. Такі програми надзвичайно важко завантажити в розподілене середовище.

Щоб пом’якшити ці труднощі, вони використовують логічну архітектуру додатків із такими рівнями замість фізичного рівня:

– рівень документа (настільні та графічні програми)

– принципи проектування, принципи прийняття проектних рішень та управління дизайн-проектами

– управління даними

За умови, що міжрівневі інтерфейси стандартизовані , логічні рівні можна створювати незалежно. Це означає, що кожен із цих рівнів може бути реалізований як незалежний компонент і розподілений. У цьому сенсі вони говорять про сервер бази даних, сервер додатків і настільні програми.

Інформаційне забезпечення САПР – це сукупність відомостей, необхідних для реалізації завдань проектування, поданих у певній формі.

Основну частину ІО складають автоматизовані банки даних, які складаються з баз даних САПР (БД) і систем управління базами даних (СУБД).

База даних — це структурований набір пов’язаних даних із певної предметної області з різними цілями, який відображає стани об’єктів, їхні властивості та зв’язки.

ІО включає нормативно-довідкові документи, завдання державних планів, прогнози розвитку техніки, типові проектні рішення, системи класифікації та кодування техніко-економічної інформації, системи документації типу ЕСКД, ЕСТД, файли та блоки даних на машинних носіях,

нормативні. , планування, прогнозування фондів , типові рішення, алгоритми та програми тощо

Інформаційне забезпечення визначається базами даних САПР і характеризується характеристиками цих баз даних. При цьому під базою даних можна розуміти набір наступних елементів: їх концептуальної моделі, інструментів, що забезпечують занурення даних у певне сховище, побудоване на основі концептуальної моделі та функції вилучення необхідних даних (СУБД), сукупність засобів і методів, що забезпечують фізичне представлення даних на певних носіях.

Вимоги до бази даних САПР **:**

1) інформація має бути центральною концепцією, додатки повинні будуватися навколо бази даних;

2) можливі виключення надмірності (інформація повинна зберігатися в одному місці, інформація змінюється лише один раз, конфліктні копії виключаються);

3) способи внесення змін у дані повинні враховувати відносність часу.

Наприклад, зміни та обробка даних відрізняються між транзакційними та аналітичними базами даних.

Транзакційні бази даних підтримують процес проектування. Аналітичні - ті, на основі яких створюються звіти.

База даних — це структурований набір пов’язаних даних із певної предметної області з різними цілями, який відображає стани об’єктів, їхні властивості та зв’язки.

Усі ознаки об’єкта є атрибутами об’єкта. Інформація, що міститься в кожному атрибуті, називається значенням цього атрибута.

Будь-який об’єкт, який характеризується записом, що містить ідентифікатор об’єкта та його атрибути, а також ключ запису, який використовується для пошуку цього об’єкта. Ключем може бути адреса запису , ідентифікатор, індексована адреса запису .

Система керування базами даних (СУБД) — це спеціальна програма, призначена для запису, систематизації, пошуку інформації в базі даних і надання її споживачеві.

Сукупність баз даних і СУБД називається банком даних.

Є два представлення бази даних:

– логічний;

– фізичний

Логічне представлення — це представлення розробників додатків за допомогою готових баз даних. Він відображає склад інформації та зв'язки між її елементами, що зберігаються в базі даних, і не враховує питання розміщення та зберігання інформації на матеріальних носіях.

Фізичне представлення бази даних відображає те, як інформація відображається на комп’ютерному носії (як і де інформація зберігається та як вона використовується).

використовуються поняття елементів і зв'язків між ними . Існує три види спілкування:

– просте підключення (як правило, одностороннє)

– складний зв'язок (багато зв'язків)

– умовні посилання (які можуть бути розірвані).

Вимоги до бази даних ***:***

1) Адекватність даних, послідовність і надійність.
2) Організація БД повинна забезпечувати узгодження часу завантаження даних програмами з частотою їх використання.
3) Універсальність, тобто наявність усіх необхідних даних у базі та можливість доступу до них у процесі вирішення завдань.
4) Відкритість бази даних.
5) Наявність мов з високим рівнем взаємодії користувача з базою даних.
6) Конфіденційність.
7) Безпека (захист від втрати).
8) Оптимальна організація даних (мінімальна надмірність).

База даних складається як мінімум з трьох рівнів:

– нерухома частина;

– напівпостійний ;

– мінливий

Постійна частина містить довідкову інформацію (фіксовані, отримані раніше рішення). Напівпостійні - ті дані, які використовуються для вирішення проблеми. Змінна - це дані, з якими в даний момент працює будь-яка програма.

Ієрархічна база даних складається з упорядкованого набору дерев; точніше, з упорядкованого набору кількох екземплярів одного типу дерева.

Тип дерева складається з одного «кореневого» типу запису та впорядкованого набору нуля або більше типів піддерева (кожен з яких є різновидом дерева). Дерево типів зазвичай є ієрархічно організованим набором типів записів.

тип нащадка із загальним екземпляром типу предка називаються близнюками. Для бази даних визначено повний порядок обходу - зверху вниз, зліва направо.

Недолік: отримати доступ до інформації можна тільки через root-доступ, а можливі типи підключення фіксуються заздалегідь.

Концептуальна модель даних є основою для побудови бази даних. Найперші ієрархічні бази даних базуються на аналізі даних із вибором зв’язків «один до багатьох» .

Недоліком бази даних є те, що доступ до інформації можливий лише через кореневий запис, а можливі типи підключення фіксуються заздалегідь. Отримання нестандартної інформації вимагає багато роботи, що випливає з типу опису підключення.

Мережеву базу даних також називають семантичною або кадровою базою даних.

Семантичні бази даних побудовані на основі зв'язків багато-до-багатьох і практично містять кілька ієрархічних рівнів з повторюваними даними та зв'язками між ними.

Ці бази даних не характеризуються збереженням даних в одному місці. При цьому виникають труднощі з реплікацією (відкат і збереження даних у всіх місцях).

Реляційні бази даних — це двовимірні таблиці, у яких кожен рядок містить один елемент даних.

Будь-який стовпець може бути ключем. Зв'язок між таблицями відбувається через будь-який елемент цих таблиць.

Послідовність атрибутів, пов'язаних з одним елементом, називається кортежем . Кожен стовпець називається доменом.

Таблиці реляційної бази даних мають такі властивості:

– кожен елемент таблиці є одним елементом даних;

– немає дублюючих записів;

– всі стовпці в таблиці однакові (кожен стовпець містить елементи одного типу);

– стовпцям присвоюються унікальні імена;

– відсутність двох однакових рядків у таблицях;

– в операціях над таблицями рядки та стовпці можна побачити в будь-якому порядку, незалежно від інформації, змісту та значення, яке вони містять.

Поняття з'єднання інтерпретується як запит у SQL.

У цих базах даних використовується строгий математичний апарат - «реляційна алгебра», що дозволяє моделювати дані і створювати на основі цих моделей запити практично будь-якої форми.

РОЗДІЛ ІІ. ПРИНЦИПИ ПРОЕКТУВАННЯ РАДІОЕЛЕКТРОННИХ ПРИСТРОЇВ

2.1. Загальні положення

Проектування — це розробка технічної документації, що дозволяє реалізувати заданий пристрій у заданих умовах.

Стратегія проектування – це функціональний розподіл, який включає опис зовнішнього блоку (входи та виходи) і внутрісистемний опис – тобто функцію, алгоритм роботи: F = Ф(X , t), де X – вектор вхідні значення; F – вектор початкових значень; час . Під час декомпозиції функція Ф розбивається на простіші функції $Ф_1$,..., $Ф_k$, між якими мають існувати певні зв'язки, які відповідають вибраному алгоритму створення функції Ф. В результаті розкладання системи остаточно виходить структура. Перехід від функціоналу до структури – синтез.

Оптимальний варіант вибирається за результатами аналізу, в ході якого перевіряється правильність роботи і деякі показники, що характеризують пристрій. Декомпозицію функцій блоку проводять до отримання типових функцій, кожна може бути реалізована якоюсь мікросхемою.

Процес проектування є багатоетапним та повторюваним, що передбачає повернення та перегляд раніше прийнятих рішень. До цього процесу варто додати можливості сучасної бази елементів, розповсюдження якої закінчується при отриманні типових функцій, що відповідають окремим мікросхемам або елементам функціональних бібліотек, програмованих VIS / HVIS.

Характер конструкції багато в чому залежить від типу використаної елементної бази. Класифікація інтегральних схем за ознаками, пов'язаними зі способами їх проектування, наведена на рис. 2.1.

До стандартних мікросхем відносяться інтегральні схеми низького та середнього рівня (MIC і CIC). Ці мікросхеми випускаються серійно і реалізують стандартні елементи і вузли, функціонування яких не визначається

конкретними споживачами. Мікропроцесори (МП), мікроконтролери (МК) і пристрої пам'яті (ПП) належать до стандартних високоінтегрованих схем (ВІС та НВІС), які залишаються незмінними після виробництва, незалежно від пристроїв і систем, у яких вони використовуються. Стандартні інтегральні схеми мають великий ринок, що сприяє зниженню їх вартості.

Спеціалізовані інтегральні схеми (СпІС) — це ті, конструкція яких, на відміну від стандартних інтегральних схем масового виробництва, адаптована до конкретних вимог конкретного проекту. Серед SPIS є напівнастроювані та налаштовані класи . Типи нестандартних мікросхем абсолютно нестандартні і розроблені «за стандартними схемами».

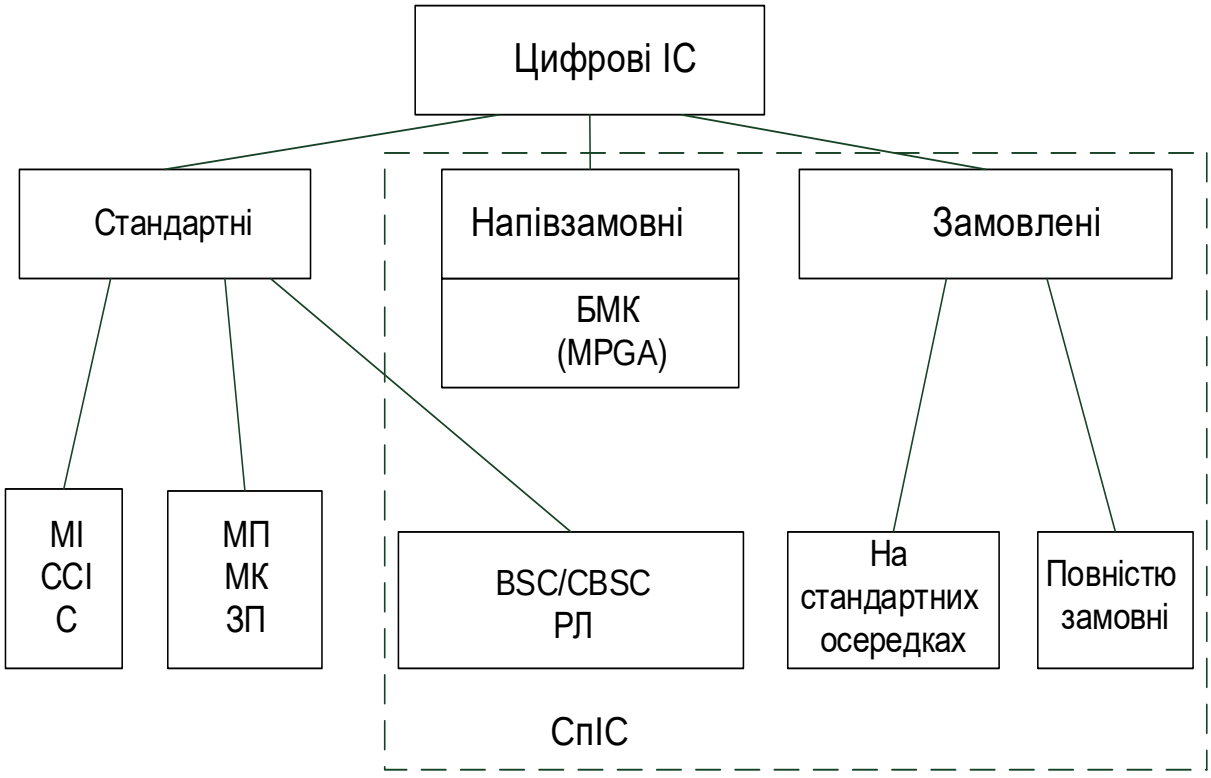


Рис.2.1. Класіфікація цифрових інтегральних схем

Повністю персоналізовані проекти розроблені відповідно до вимог конкретного клієнта. Розробник має повну свободу дій, визначаючи схему на свій розсуд аж до рівня компонентів системи (окремих транзисторів тощо). Схематичне виготовлення вимагає розробки цілого комплекту фотошаблонів,

перевірки та налагодження всіх окремих фрагментів. Такі схеми дуже дорогі і мають довгі цикли проектування.

Схеми на стандартних кристалах відрізняються від повністю кастомних тим, що їх фрагменти походять з раніше розробленої бібліотеки схемних рішень. Фрагменти вже добре розроблені, а вартість і тривалість проекту скорочуються. Виготовлення схем також вимагає виготовлення повного комплекту фотошаблонів, але їх розробка полегшується. Втрати порівняно із загальною кількістю замовлень ІС полягають у тому, що проектувальник має менше можливостей у побудові схеми, тобто результатів її оптимізації за критеріями кристалічної простоти, швидкодії та ін. менш ефективні.

Найвищі технічні характеристики досягаються при абсолютно нестандартних схемах, але популярним є метод стандартних кристалів, так як при невеликих втратах технічних характеристик з його допомогою можна значно спростити конструкцію схеми. Повністю індивідуальні схеми розробляються приблизно вдвічі швидше, ніж стандартні кристали. Напівнестандартні схеми включають кристали базової матриці (БМК). В даному випадку ми маємо справу зі стандартним напівфабрикатом, який доводиться до готового продукту за допомогою окремих роз’ємів. Реалізація потребувала виготовлення лише невеликої кількості шаблонів. Вартість і тривалість проекту в порівнянні з повністю нестандартними схемами знижується в 3..4 рази, але результат все одно далекий від оптимального, оскільки в БМК менш раціонально використовується поверхня кристала (невикористані елементи залишаються для кристала тощо), довжини зв’язків не є мінімальними, а швидкість – не максимальною.

Подібність методів проектування на ВМС і стандартних кристалах полягає у використанні бібліотек функціональних елементів. Відмінність полягає в тому, що для схем, розроблених стандартним кристалічним методом, набір елементів бібліотеки має більш виражену топологічну ширину. Наприклад, стандартизована тільки висота клітин, а їх довжина може бути різною. При проектуванні спочатку з колекції елементів бібліотеки

вибираються необхідні функціональні блоки, а потім розглядаються завдання їх компонування та відстеження.

САПР для проектування стандартної комірки є більш складною, ніж для конструкції БМК. Повністю персоналізовані проекти розроблені відповідно до вимог конкретного клієнта. Розробник має повну свободу дій, визначаючи схему на свій розсуд аж до рівня компонентів системи (окремих транзисторів тощо). Схематичне виготовлення вимагає розробки цілого комплекту фотошаблонів, перевірки та налагодження всіх окремих фрагментів. Такі схеми дуже дорогі і мають довгі цикли проектування, яка має більш жорсткі топологічні обмеження. Також були введені обмеження для стандартного методу комірки (постійна висота комірки, визначення геометричних розмірів і положення силових рейок, синхронізація тощо), але оскільки використовуються більш потужні системи САПР, обмеження послаблюються.

Важливе місце в класифікації займають надвеликі інтегральні схеми програмуємої логіки. З одного боку, вони належать до СпІС, тому що в кінцевому підсумку адаптуються під вимоги конкретного проекту. При цьому цей процес (конфігурація схеми) не впливає на виробника, для якого схема є стандартним продуктом.

На думку експертів, висловлених в журналі ElectronicDesign , «обладнання для програмування зробить таку ж революцію в найближчі роки, як мікрокомп'ютери на початку 70-х».

2.2. Сфери застосування різних типів НВІС

Всі типи СпІС мають свої області застосування. Кожен вид характеризується певним співвідношенням таких параметрів, як складність (досяжний рівень інтеграції), швидкість і вартість. Основне поняття можна пояснити з точки зору економіки, звернувшись до формули для значення SI, виробленого в уже освоєному технологічному процесі:

$$C_{\text{IC}} = C_{\text{виг}} + \frac{C_{\text{пр}}}{N},$$

Де $C_{\text{виг}}$вартість виготовлення ІМС (вартість кристала та інших матеріалів, вартість технологічних операцій з виготовлення ІМС, контрольні випробування). Витрати на виробництво стосуються кожної ІМС, тобто вони повторюються стільки разів, скільки виготовляється ІМС; $C_{\text{пр}}$– вартість проектування ІС, тобто одноразові витрати на цей тип ШІ; N - обсяг виробництва (тираж), який є кількістю інтегральних схем, які будуть виготовлені.

Вартість проектування ВІС/НВІС висока і може сягати сотень мільйонів доларів. У разі дорогих варіантів конструкції ВІС/НВІС виробництво стає рентабельним лише при високих обсягах продажів.

Витрати $C_{\text{пр}}$взаємопов'язані $C_{\text{виг}}$. Збільшення витрат на проектування зазвичай призводить до зменшення $C_{\text{виг}}$, тому що чим досконаліша конструкція, тим раціональніше використовується поверхня кристала та інші ресурси.

Наступні положення застосовуються до логіки програмування. Прості пристрої зі складністю сотні еквівалентних вентилів доцільно реалізовувати на PLD (PAL, GAL, PLA). Зі збільшенням складності конструкції природний перехід до FPQA та CPLD, якщо зусилля ІС відносно невеликі. Збільшення тиражу (близько десятків тисяч) призводить до переваг реалізацій на БМК, оскільки вартість виготовлення невеликої кількості шаблонів для створення з'єднань буде розподілена на велику кількість мікросхем, а вартість виготовлення кожної ІС буде зниження за рахунок виключення програмних зв'язків зі схеми та методів їх програмування.

При ще більшому тиражі рентабельним виявляється метод стандартних комірок, що дозволяє в подальшому поліпшити параметри схеми, щільніше розташувати її елементи на кристалі, тобто зменшити і підвищити швидкодію $C_{\text{виг}}$. При цьому $\frac{C_{\text{пр}}}{N}$ термін , у формулі вартості ІС, не буде занадто великим через високе значення N, хоча необхідність проектування цілого набору шаблонів для технологічних процесів призводить до великих витрат.

Повністю структурований дизайн не характерний для СНД . Він настільки дорогий, що практично використовується лише для створення стандарту ВІС/НВІС масового виробництва. Наприклад, розробка першого 32-розрядного мікропроцесора коштувала тоді 140 мільйонів доларів, а 1 Мбіт оперативної пам’яті – 395 мільйонів доларів.

Варто відзначити, що провідна комп'ютерна компанія IBM використовує методологію проектування змішаного типу, розміщуючи стандартну комірку ВІС/НВІС в одних і тих же областях. У схемах обробки критичних сигналів використовуються більш щільні і швидкодіючі схеми, інший напрямок - транзистори БМК.

Проектування стандартних, серійних ІС, як і проектування нестандартними методами в цілому - це доля великих спеціалізованих компаній. Схемотехніки в основному займаються іншими розробками: малоскладними цифровими пристроями на базі МІС та СІС, мікропроцесорними системами технічних засобів і технологічних процесів, малогабаритним обладнанням або прототипами систем на основі програмованої логіки ІС .

Проектування на основі МІС, СІС є найбільш традиційним процесом, який використовує як евристичні підходи, так і формалізовані методи. Конструктор визначає конструкцію пристрою, виходячи зі своїх знань, ідей і засвоєного досвіду попередників, а при визначенні функцій окремих блоків використовує також формальні методи. У цьому випадку якісне проектування вимагає знання типових функціональних вузлів, їх властивостей і параметрів.

Мікропроцесорна система створюється в результаті розробки комплексу апаратних і програмних засобів. Розробка апаратної частини зводиться до складання системи з типових модулів: центрального блоку, різних типів пам'яті, адаптерів, контролерів і зовнішніх пристроїв.

Основною частиною інженерного розвитку обладнання в умовах сучасної України, безумовно, є застосування програмованої логіки для побудови необхідних пристроїв або для їх налагодження.

Проектування на основі дуже складних логічних схем програмування здійснюється виключно за допомогою систем автоматизованого проектування. Спрощена структура алгоритмів проектування наведена на рис. 2. 2.

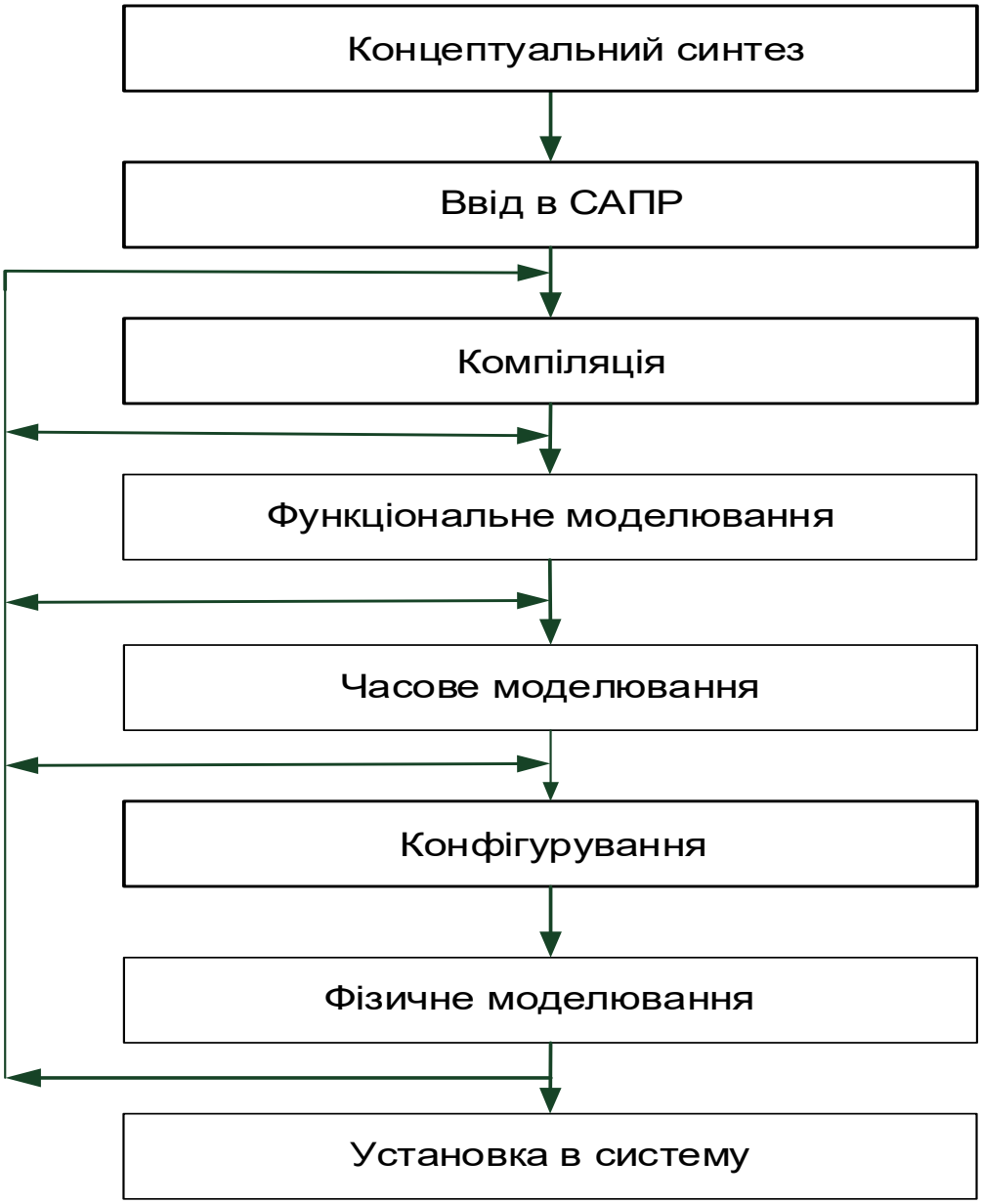


Рис. 2.2. Структура алгоритму автоматичного проектування

Проектування на концептуальному рівні – це робота дизайнера, яка має слабке відношення до автоматизації. На цьому рівні визначається необхідне функціонування пристрою, вхідні та вихідні сигнали, їх природа та зв'язки, поділ проекту на частини тощо. Результати концептуального синтезу вводяться в програму САПР, яка компілює проект, тобто синтезує пристрій у базі даних бібліотеки його моделей. Отриманий проект потребує ретельної

перевірки, тому після етапу синтезу слід етап аналізу, який проводиться через моделювання та теоретичну перевірку. Створення має декілька рівнів з різним ступенем відтворення властивостей об'єкта. Він може бути функціональним, який перевіряє доцільність логічної структури пристрою, часовим, що враховує затримки сигналу в схемах пристрою без топології кінцевого шляху тощо. У результаті моделювання можуть бути помилки, які необхідно виправити, що збагачує ітеративний процес проектування. характеру з поверненням до попередніх етапів і внесенням необхідних коректив у проект.

конфігурація мікросхеми програмованої логіки , після чого можливий фактичний контроль роботи пристрою - фізичне моделювання проекту. Після успішного завершення фізичного моделювання пристрій готовий до встановлення в системі.

2.3. Методи та засоби автоматизованого проектування цифрових пристроїв

Усі сучасні методи проектування процесорів на основі комплексного програмування BIC/HBIC базуються на використанні САПР. Правильний вибір САПР є найважливішою передумовою ефективного проектування та прискорення виробництва.

Методи та інструменти проектування тісно пов'язані з вибором САПР і навпаки, вибір САПР визначає прийнятні та відповідні методи інструменту проектування, тому ці питання не можна розглядати ізольовано один від одного

Способи опису проекту

Використання САПР вимагає ефективних, візуальних, керованих і контрольованих засобів опису проекту. Розроблений пристрій можна описати різними способами, і найпоширенішим методом є опис конструкції в цілому.

В даний час найпоширенішими універсальними методами опису, які використовуються на кожному рівні ієрархії проекту, є графіка і текст. Рідше використовується пряме поділ схем ПЛІС в редакторі топології, опис у вигляді необхідних часових діаграм і т. д . Кожен з методів опису конструкції має свої

переваги і недоліки. Схожість способу опису, внутрішньої організації та роботи викличного пристрою значно зменшує час створення проекту, спрощує його внутрішнє тестування і, як правило, виявляється більш природним та зрозумілим.

Графічне зображення проекту створюється в базі даних елементів бібліотеки, прийнятних для обраної САПР, наприклад, в базі даних елементів стандартної серії TTL. Головною перевагою графічного методу є його традиційність і наочність, які пов'язані з ознайомленням дизайнерів зі сприйняттям схематичних зображень. Звичайно, ця перевага виявляється лише після належного ієрархічного та структурного аналізу проекту.

Сучасні мови опису обладнання (HDL, HardwareDescriptionLanguages) дозволяють описати розроблений пристрій, як з точки зору його поведінки, так і з точки зору його конструкції. Ці можливості роблять все більш поширеним подання проекту у вигляді текстового опису алгоритмів його фрагментів у поєднанні з текстовим описом міжблочних зв'язків для складних проектів. Перевагами текстового методу опису проекту є його компактність і відносна легкість автоматизації всіх перетворень, що включає початкову генерацію самого проекту. Дуже важлива можливість широкого використання стандартних універсальних мов, таких як HDL, що забезпечує легке перенесення проекту з деякого апаратного застосунку на іншу і перехід від однієї САПР до другої.

На відміну від текстових, графічні методи подання проекту зазвичай вузькоспеціалізовані і вимагають спеціальних засобів передачі інформації про проект в інше середовище, для чого можуть використовуватися спеціальні універсальні мови передачі інформації про проект, які набувають все більшого поширення. Текстові описи діляться на два основних різновиди - мови низького рівня та мови високого рівня.

Мови низького рівня ближче до апаратного забезпечення, що створює потенційні можливості для компіляторів створювати проекти з більш сприятливими параметрами. Ціна за це, звичайно, сильна орієнтація на конкретне обладнання та компанію, яка його виробляє. Приклади таких мов

включають AHDL (AltéraHDL) і ABEL (Xilinx). Використовуючи мови низького рівня, легше створювати проекти з найкращими параметрами синхронізації, оскільки проекти будуть враховувати специфіку архітектури конкретного CPLD або FPGA.

Мови високого рівня менш прив'язані до апаратних платформ і тому більш універсальні. Серед них найпоширенішими мовами є VHDL і Verilog . Дані мови, як і другі алгоритмічні мови більш високого рівня, в основному дозволяють описувати будь-який алгоритм у послідовній формі, тобто через послідовність операторів присвоювання та прийняття рішень. Основною відмінністю є можливість відображення дій, що виконуються паралельно на апаратному забезпеченні, представлених окремими паралельними процесами.

При проектуванні процесора найефективніше розділити його на два блоки: операційний і керуючий. Операційний блок (ОБ) виконує перетворення даних і складається зі стандартизованих частин, а блок керування (БК) повинен забезпечувати необхідну послідовність операцій, що виконуються в ОБ (одна або декілька). Для цього БК подає керуючі сигнали на входи ОВ. Порядок дій і сигналів управління залежить від результатів операції в ОВ. З цього випливає, що ВС зручно визначити в термінах кінцевого автомата з пам'яттю (ЗП) того чи іншого типу. У складних конструкціях можливе розбиття процесора на кілька функціональних слабозв'язаних пар ОВ-ВС на одному рівні ієрархії або створення пари ієрархічних зв'язків цієї системи.

Операційний блок зазвичай представлений набором регістрів, логічних схем (зазвичай багатофункціональних і керованих), буферних схем і комутаційних зв'язків між ними.

2.4. Етапи проектних процедур

У загальному вигляді процедура розробки процесора представлена вище (див. рис. 1.2). Більш детально проектування за допомогою САПР ми розглянемо нижче. Розробка відбувається в такому порядку:

Складання змістовної графічної схеми або функціонального алгоритму структурної схеми пристрою. Перше завдання полягає в переході від технічного завдання проекту (ТЗ) до формального опису проектованого пристрою. Технічне завдання є сумою словесного і технічного опису, як його формалізація призводить до ідентифікації основних блоків пристрою (або алгоритму) і визначення їх зв'язків і взаємодій . Спосіб і манера поділу найчастіше і в першу чергу визначаються уподобаннями дизайнера і фіксуються лише зрідка. Сама форма КТ може визначати певні заходи для проектувальника, хоча, можливо, більш ефективним буде інший спосіб опису проекту чи його фрагментів. По суті, на даний момент здійснюються перші кроки першого етапу . Формально перший етап - це поділ завдання на функціонально відмінні підзадачі , етап декомпозиції.

Декомпозицію можна звести до поєднання схем алгоритмів роботи фрагментів або функціональної схеми пристрою та його частин. Можливим варіантом для досить складних систем було б розумне поєднання поведінкового та структурного розбиття дизайну. Поділ відбувається не тільки в межах одного рівня ієрархії. Більшість проектів також поділяються на ієрархічно організовані рівні. Важливим завданням , яке необхідно вирішити на цьому етапі, є уточнення та узгодження з клієнтом функцій інтерфейсу проекту . Визначається впровадження протоколів зовнішнього обміну . Тимчасові характеристики та правила взаємодії із зовнішніми пристроями визначають прийнятну організацію та структурну модель внутрішніх вузлів.

Використання САПР на даному етапі проектування ще досить рідкісне, хоча для реалізації сучасних, дуже складних проектів (кілька сотень тисяч арматури) все частіше використовуються спеціальні блокові редактори, які дозволяють декомпозицію проекту без легалізації складових частин . Прикладом є CAD Quartus від Altera , який містить спеціальний інструмент редагування на рівні блоку .

Розробка загальної структури проекту

Основним завданням є вибір прийнятних елементів для реалізації кожного рівня ієрархії, визначення зв’язків між ними, а якщо параметри

елементів регульовані , то їх коригування. Кілька моментів є визначальними для сцени: з одного боку, це джерело набору прийнятних елементів, з іншого боку, це засіб опису зв'язків елементів один з одним і, якщо необхідно, можливість опис новітніх (специфічних для даного проекту) елементів.

При розробці засобів з цифровим відображенням інформації їх природно розділити на два блоки: оперативний і контрольний. Операційний блок (ОВ) виконує перетворені дані і побудований зі стандартних частин (поведінкових частин), а блок управління (control device, ВС) забезпечує необхідну послідовність операцій, що виконуються в ОВ (одна або декілька). Для цього БК подає керуючі сигнали на входи ОВ. Послідовність дій та/або керуючих сигналів залежить від результатів роботи в ОВ і зовнішніх впливів. Звідси видно, що БК зручно встановлювати у вигляді кінцевого автомата з пам’яттю (ЗП) того чи іншого типу.

У складних конструкціях можна розділити ВС на кілька функціонально слабко пов’язаних пар ОВ-ВС на одному ієрархічному рівні або створити пару, ієрархічно вбудовану в ОВ (рідше в ВС). Ресурси та можливості сучасних програм САПР змушують нас дещо по-новому підходити до проектування машин. Тут слід звернути увагу на дві основні проблеми. По-перше, проблема оптимального кодування станів і перехідних умов автомата представляє для програмістів більше теоретичний інтерес, ніж практичний. Розробник найчастіше визначає лише спосіб кодування машини, а конкретне кодування виконує CAD. Лише в деяких випадках (зазвичай на основі розуміння швидкості виконання конкретних переходів або створення вихідного сигналу) доцільно вручну закодувати машину і накласти прийнятий варіант на САПР. Можливість такого певного визначення стандартної відповідності підтримується або призначенням постійних присвоєнь окремим елементам проектованої конструкції в програмі САПР, або використанням більш детального методу опису проектної структури.

По-друге, організація та структура комірок ПЛІС (а також практична відсутність обмежень на складність зв’язків автоматів) дозволяють значно розширити спектр синтезованих структурних діаграм автоматів SAIIR,

структур канонічних автоматів типу Майлі та Мура під час синтез виявляються переплетеними в різних комбінаціях.

Змістовний опис проекту та його частин

Впровадження САПР дозволяє створювати ефективно керовані та контрольовані описи проектів та їх частин. Крім того, один і той же пристрій може бути описаний різними засобами САПР. Використовувані методи зазвичай придатні як для опису проекту в цілому, так і для опису окремих його фрагментів. Більше того, більшість програм САПР дозволяють перетворювати один тип опису в інший. В даний час найбільш поширеним, універсальним способом опису проекту, який використовується на будь-якому рівні його ієрархії, є графіка і текст. Рідше використовується пряме малювання діаграм в редакторі топології, опис у вигляді необхідних часових діаграм і т. д. Кожен із способів має свої переваги і недоліки.

Графічне зображення проекту в сучасній САПР може бути створено як на основі графічних позначень конструктора, так і на основі елементів бібліотеки, прийнятних для вибраної САПР, наприклад елементів стандартної серії ТТL(Ш). Припустимо змішувати ці дві основи в різних комбінаціях. Основними перевагами графічного методу є його традиційність і прозорість, які пов'язані зі знанням творцями способу сприйняття зображень діаграм. Звичайно, ці переваги виявляються тільки при правильному ієрархічно-структурному розподілі проекту.

Залежно від мети, програміст повинен вибрати той чи інший редактор. Результатом роботи редакторів є створення опису проекту та/або його фрагментів на одній із мов опису обладнання. Навіть якщо CAD не підтримує графічне введення, воно може відображати текстовий опис у графічній формі.

До недоліків графічного представлення можна віднести відсутність точних стандартів відповідності між текстовим і графічним представленням. На відміну від текстових, графічні методи: презентації проекту зазвичай вузькоспеціалізовані і вимагають спеціальних засобів передачі інформації про систему в інше середовище, для чогось можуть використовуватися спеціальні універсальні мови передачі інформації про проект.

Сучасні мови опису обладнання (Hardware опис Мови) дозволяють описати розроблений пристрій як з точки зору його поведінки, так і з точки зору його конструкції. Ці можливості роблять поширеним для складних проектів представлення проекту у вигляді текстового опису алгоритмів його фрагментів у поєднанні з текстовим описом міжблочних зв’язків. Перевагами текстового методу опису проекту є його компактність, відносна легкість автоматизації всіх перетворень, у тому числі початкової генерації опису проекту. Дуже важливо мати можливість використовувати стандартні, універсальні мови, такі як HDL, які забезпечують легке перенесення проекту з вибраної апаратної платформи на другу і перехід від однієї САПР до другої.

В основі архітектурно-конструктивного опису операційного блоку лежить завдання побудови зв'язків вибраних елементів. Традиційний графічний метод представлення структури з'єднань, який залишається найбільш наочним способом, супроводжується текстовим способом опису в сучасних САПР. Інструменти, включені в програму CAD, дозволяють автоматичне, пряме та зворотне перетворення описів.

Склад елементів операційного блоку залежить від складу використовуваної бібліотеки. Більшість САПР підтримує ієрархічний опис проектів. На кожному рівні проект представлений як набір елементів цього рівня. Набір функціональних можливостей бібліотечних елементів, пропонованих стандартною САПР, надзвичайно широкий, а бібліотеки за своїм складом і походженням поділяються на:

- стандартні бібліотеки фірми-розробника САПР, вміст яких відповідає тій чи іншій поширеній серії MIS або SIS (наприклад, серії IS типу 74);
- стандартні елементи обчислювальної техніки (логічні елементи, дешифратори, мультиплексори , лічильники та ін.), параметри яких постійні;
- типові елементи обчислювальної техніки (лічильники, регістри, мультиплексори тощо) , детальні параметри яких (розрядність, поляризація керуючих сигналів тощо) можуть вільно задаватися проектувальником;
- типові вузли комп'ютерних систем (Периферійні пристрої, апаратні ядра мікроконтролерів і мікропроцесорів), деякі параметри яких змінюються

проектувальником (як правило, конфігурація цих вузлів розробляється або компанією, що розробляє вузол, або у співпраці з нею);

- елементи, створені дизайнером і пов’язані в бібліотеці дизайнера.

Кожен проект можна використовувати як підпроект у більш складному проекті. Створити вбудовану бібліотеку модулів конструктора неможливо, хоча бажано мати власні модулі дизайну або базу готових проектів.

Як правило, на кожному рівні ієрархії базовий набір елементів операційного блоку (на цьому рівні) доповнюється набором регістрів, логічних систем (багатофункціональних і керованих), буферних схем і необхідних комутаційних зв'язків між ними. . для його функціонування. Важливо, щоб на нижніх ієрархічних рівнях опису проекту була однозначна інтерпретація функціонування всіх елементів ОВ.

На цьому етапі визначається функціонування керуючого блоку, що забезпечує необхідну взаємодію елементів операційного блоку. Слід підкреслити, що останні два етапи сильно взаємозалежні і, якщо не розвиваються паралельно, реалізуються ітераційно .

Існують різні форми і способи опису автомата. Сучасна тенденція полягає в переході від нотації логічних виразів, обмежених положеннями Конституційного Трибуналу, до графічної форми. Опис у вигляді графа переходів (діаграми станів) стає одним із найпоширеніших варіантів автоматних задач. Графічні редактори для створення автоматів входять до складу засобів завдання початкових проектів сучасних САПР.

Редактори від різних виробників NVIS PL мають свою специфіку, але всі вони відрізняються винятковою простотою, природністю та зручністю, а також відсутністю абсолютної необхідності знати вихідну мову редактора. Найбільш якісні версії дизайнерських програм мають повний набір інструментів для виконання всієї конструкторської процедури розробки проекту, що дозволяє реалізувати наступні операції:

- малювати граф переходів, що містить назви станів, напрямок, умови та пріоритети умов переходу, генеровані сигнали та способи їх формування;

- перевірити правильність підготовленого графіка переходів (повторення назв, неоднозначність переходів, некоректність переходів тощо);
- скомпілювати проект (з вихідного текстового файлу) у вибраному режимі;
- моделювати роботу автомата в інтерактивному режимі та режимі компіляції.

Важливою перевагою StateCAD є можливість широкого вибору форм представлення результатів (опис на мовах високого рівня VHDL і Verilog і на мовах низького рівня ABEL, AHDL).

Специфіка продуктів тієї чи іншої компанії відбивається і на мовах високого рівня, виражаючи, перш за все, різницю в необхідних для роботи бібліотеках, а також у складності та різноманітності прийнятних для компіляторів синтаксичних конструкцій. Кінцеві результати компіляції однієї і тієї ж вихідної графової схеми або автомата наступної компіляції тієї ж програми з мови високого рівня в початковий файл чіпсета PL, отримані від компіляторів різних компаній, можуть істотно відрізнятися і мати різні продуктивність. StateCADVersion 3.2 пакета WorkviewOffice зручний тим, що перед перекладом діаграми переходів необхідно вказати не тільки бажане представлення мови (VHDL , AHDL , Verilog , ABEL тощо), а й зарезервовані атрибути, що дозволяє оптимізувати позначення автомата і уникати використання синтаксичних конструкцій. не допускається для укладачів відповідних компаній.

Як уже зазначалося, при використанні графічних редакторів користувачеві не обов'язково знати вихідну мову редактора. Однак у деяких випадках мати це надзвичайно корисно. Корисність орієнтування в мовних конструкціях виявляється, наприклад, у ситуаціях, коли автомат потрібно мінімізувати за тим чи іншим параметром, насамперед за інтервалами часу між генерованими вихідними сигналами , що може призвести до тимчасових збоїв сигналу. Саме в таких випадках знання мови та майстерність дизайнера полегшують отримання найкращих результатів.

Коли проект і всі його частини готові, можна переходити до

найважливішого етапу проектування - складання проекту. Компіляція може включати як весь проект, так і його компоненти. Компіляція окремих фрагментів, з одного боку, спрощує проект, оскільки зменшує розмір аналізованих проблем, з іншого боку, вимагає врахування різних способів функціонування внутрішніх ресурсів і зовнішніх елементів. Під час компіляції виявляються всі приховані помилки та невідповідності. Компіляція ділиться на ряд підетапів: збір бази даних проекту, контроль з'єднання, логічна мінімізація проекту, створення файлу запуску (конфігурації) тощо. На кожному етапі можуть виникати помилки, які вимагають повторної компіляції після того, як вони були виправлені.

Процес компіляції в САПР виробників ПЛ ВІС відрізняється від компіляції в САПР інших компаній, основною відмінністю є відсутність кроку створення стартового файлу. Як правило, ефектом зовнішнього компілятора є структура проекту на основі обраного сімейства ПЛІС. Класичним прикладом такого компілятора є пакет LeonardoSpektrum, пакет, який підтримує проектування для FPGA від Actel , Altera , AgireSystems , QuickLogik , Xilinx та інших. Забезпечує введення тексту у VHDL , Verilog , вичитку на рівні передач реєстрів, оптимізацію на основі обмежень, встановлених програмістом, аналіз часових характеристик проекту, створення вихідної інформації, необхідної для розміщення та підключення проекту до обраної VIS, перегляд проект на рівні воріт.

Результатом компіляції за допомогою САПР від виробників НВІС є стартовий файл, тобто конфігураційна інформація для обраної програмованої мікросхеми. Крім того, створюється файл звіту, який містить всю інформацію про процес збирання та його результати. Існує значна різниця в процедурах побудови схем CPLD і FPGA . За винятком автоматичного розміщення та відстеження з'єднань FPGA, як правило, коригування, внесені розробником, допускаються на будь-якому етапі процесу. Ця дія використовується редакторами топології, які дозволяють змінювати структуру проекту кристалів і підвищення ефективності розроблених пристроїв. Для CPLD вплив дизайнера на структуру скомпільованої схеми можливий лише через непряме

втручання, шляхом зміни опису проекту або зміни параметрів, встановлених перед компіляцією. Після успішного завершення проекту або його частини можна переходити до його перевірки.

Тестування розробленого пристрою або його фрагментів є одним з найважливіших етапів проектування, оскільки бездефектних проектів практично не буває. Пошук недоліків дизайну – складне завдання, швидкість і якість якого залежить від досвіду розробників.

У сучасних САПР найпоширенішим тестуванням є робота з редактором діаграм часу, ці редактори поділяються на компілюючі та інтерпретуючі. Редактори типу інтерпретації спрощують процедуру налагодження проекту та пошуку його дефектів, пов'язаних з некоректністю творець структурного або поведінкового шляху реалізації системи або характеристики реалізації елементної бази, що використовується. У 6-віконній САПР інтерпретаційного типу результати моделювання за певний момент часу просто відображаються на всіх типах проектних презентацій (сигнали, електричні схеми, топологія), що дозволяє легко змінювати потік експерименту та композицію відображення сигналу. У випадку схематичного опису конструкції трасування сигналу спрощується.

Перевагою складання систем моделювання є мінімізація тимчасових витрат. Водночас залишається проблематичним визначення відповідності між рядками тексту і станами окремих сигналів. Розподілу було надано два підходи до генерації зовнішнього по відношенню до проекту впливу. Одним із підходів є формування цих ефектів шляхом визначення послідовності синхронізації вхідних сигналів у редакторі часових діаграм. Інший підхід (зручний для менших проектних завдань) полягає в написанні спеціалізованої тестової програми .

У більшості реальних цифрових пристроїв після введення початкових даних виконується кілька циклів. Роботу пристрою слід перевіряти на кількох наборах даних одного типу, щоб можна було використовувати наступну структуру програмного модуля, що представляє тестовий ефект: генеруються початкові сигнали заповнення, потім виконуються два вкладені цикли , і

внутрішній цикл послідовно формує тестові сигнали для виконання дій над входами одного набору даних, а їх зміна відбувається у зовнішньому циклі.

Сучасні САПР мають повну інформацію про конструкцію проектованого пристрою та часові параметри у всіх компонентах – це дозволяє автоматизувати процес розрахунку різних часових характеристик проекту.

Наприклад, CAD MAX+ PLUS II дозволяє автоматично розраховувати три основні класи параметрів часу:

- мінімальні та максимальні затримки між джерелами вхідних сигналів і приймачами вихідних сигналів, інформація про які надається у вигляді матриці затримок;
- максимальна продуктивність пристрою у вигляді максимальної тактової частоти використовуваних в конструкції елементів пам'яті;
- попередня настройка і час очікування сигналу, що гарантує надійну роботу схем при фіксації сигналів в елементах синхронної пам'яті.

Багато програм САПР також дозволяють виділяти критичні шляхи передачі інформації та трансформації для отримання схематичного або топологічного представлення проекту.

Хоча продуктивність цих обчислень не гарантує виявлення всіх помилок проектування, пов'язаних з процесами синхронізації процесора, це значно зменшує кількість таких помилок або, принаймні, дозволяє знаходити в проекті місця, небезпечні з точки зору відмови. .

Одним із завершальних етапів проектування є етап експериментальної перевірки розробленого пристрою. Незважаючи на всю ретельність попередніх етапів, завжди є далека від нуля ймовірність того, що в конструкції будуть дефекти, які можуть виявитися на етапі впровадження або навіть нормального використання пристрою та спричинити серйозні наслідки.

Проведення повномасштабних експериментів значно підвищує ймовірність бездефектних виробів. Спосіб прискорення роботи на цьому етапі та можливість переміщення її на ранні стадії розробки, тобто до закінчення виробництва кінцевого продукту, — це відомі прототипи систем і способи проведення з ними експериментів . Макетні плати широко використовувалися

і раніше, особливо при створенні мікропроцесорних систем. Аналогічна ситуація при розробці систем і пристроїв на основі логіки програмування. Різноманітні іноземні компанії виробляють і постачають широкий асортимент макетних плат , що містять логіку програмування та додаткове обладнання (в основному швидкі мікросхеми оперативної пам'яті). Тут ви можете вказати інструменти Alter (Demo дошка); Програми PLD (шина PCI _ _ _ плати оцінки дошка) ; Xilinx , віртуальний комп'ютер Corp. , Програмне забезпечення для відео (проектні плати HOT PCI набір) та інші.

Як при використанні прототипів продуктів , так і при тестуванні кінцевого продукту, необхідно залучати апаратні та програмні засоби, які будуть завантажувати конфігурацію, генерувати ефекти та контролювати коректну роботу тестованого пристрою. Важливим моментом проведення дослідів є генерація тестових розливів . У цьому плані PL VIS відкриває нові, раніше недосяжні можливості. Додатковий чіп, а в деяких випадках і деяка частина конфігурованих ресурсів VIS, може використовуватися як генератор сигналу. Водночас зміст дослідження може легко модифікуватися в процесі проведення експериментів залежно від результатів попередніх етапів експериментальної роботи. Корисним засобом налагодження може бути метод передачі даних про стан досліджуваного об'єкта під час експерименту на комп'ютер приладу для візуалізації та детального аналізу. Багато ПЛІС і відповідні їм програми САПР підтримують цю взаємодію. Однак у більшості випадків може знадобитися створення спеціальних програм забезпечення для пришвидшення аналізу поведінки об'єкта тестування. Звичайно, у певних ситуаціях прийнятно використовувати послідовні пристрої, такі як багатопроменеві осцилографи, логічні аналізатори тощо .

Після успішного завершення натурних дослідів з дослідним або макетним зразком конструктор повинен забезпечити випуск дослідної партії розробленого виробу. При цьому найважливішим завданням є створення передумов якісного супроводження продукції, що випускається у формі інтегральних схем ПЛ.

Оскільки базова функціональність проекту більше не викликає сумнівів, завдання дослідницького обладнання, що використовується на цьому етапі, відрізняється від завдань обладнання, що використовується на попередньому етапі. Обладнання для тестування та спосіб його використання повинні виявляти дефектну продукцію в найкоротші терміни та з мінімальними витратами, і виконувати їх персоналом з мінімальними професійними вимогами. І лише для невеликої партії продукції можливе використання тестових засобів, які дозволяють локалізувати несправності .

Необхідно відокремити вимоги до тестового обладнання від вимог до самого проекту. Продуктивність тестового обладнання значною мірою залежить від передбачливості розробника. Процедура розробки з перших кроків повинна бути зосереджена на необхідності тестування кінцевого продукту. Якщо конструктор (або замовник проекту) правильно вибрав мету проектування, то після виготовлення прототипів виробів може постати завдання організації випуску серійної продукції. При цьому основні зусилля розробників зосереджені на проектуванні та виготовленні апаратного та програмного забезпечення, що дозволяє здешевити та прискорити виробництво серійної продукції. За умови підвищення попиту на вироблену продукцію та реальних торгових точок можна розглядати питання модернізації продукту, в тому числі з переходом на більш дешеву елементну базу ІС. Цей напрямок пов'язаний з розробкою стратегії і тактики переведення проектів на нову елементну базу або переходу на нові технології виробництва кінцевої продукції.

2.5. Загальна інформація про САПР MAX+PLUS II

Донедавна MAX+PLUS II являлась єдиною системою проектування пристроїв на основі FPGA Altera . Лише в 1999 році з'явилася система проектування нового покоління Quartus , призначена для проектування пристроїв на системах APEX20K FPGA. Інтегроване програмне забезпечення MAX+PLUS II дає вам контроль над середовищем логічного проектування та

допомагає досягти максимальної ефективності та продуктивності. Усі пакети працюють як на платформі IBM PC, так і на платформах SUN, IBM RISC/6000 і HP9000. У майбутньому ми розглянемо роботу на платформі IBM PC.

Для нормального встановлення та роботи системи CAD MAX+PLUS II необхідний IBM PC-сумісний комп'ютер з процесором не гірше Pentium, оперативною пам'яттю не менше 16 МБ і вільним місцем на жорсткому диску приблизно 150-400 МБ, в залежності від конфігурації системи. З власного досвіду можна сказати, що для розробки великих чіпів на FLEX10K50 і вище FPGA бажано мати принаймні 64 МБ оперативної пам’яті (краще 128, ще краще 256, 384 МБ і більше – дуже добре) і Pentium II (P -3 насправді не дає великої користі). Звичайно, можна використовувати більш слабкі машини, але тоді збільшується час компіляції і збільшується навантаження на жорсткий диск в результаті заміни. Збільшення обсягу оперативної пам’яті та кешу дає кращі результати порівняно зі збільшенням тактової частоти процесора. Якщо ви не плануєте стежити за великими кристалами, тоді достатньо 32 МБ оперативної пам'яті, щоб отримати хорошу швидкість компіляції проекту. Що стосується вибору операційної системи, то безперечно краще Windows NT, гірше Windows 95 OSR2, гірше Windows 98, особливо локалізована версія. Звичайно, це пов'язано з тим, що пакет був спочатку розроблений для Unix і не використовує всі переваги всіх механізмів Windows . Це особливо помітно при тимчасовому моделюванні складних пристроїв DSP, де перемальовування екрана займає більшу частину часу. Оскільки пакунок не локалізований, краще використовувати нелокалізовані (американські або загальноєвропейські) версії Windows .

Під час встановлення системи MAX+PLUS II створюється два каталоги: \maxplus2 і \max2work. Каталог \maxplus2 включає системне програмоване забезпечення та файли даних і розділений на підкаталоги, перелічені в таблиці 2.1.

Таблиця 2.1. Структура системного каталогу \maxplus2 системи MAX+PLUS II

Підкаталог	опис
.\drivers	Включає драйвери пристроїв для WINDOWS NT (лише для встановлення на платформі ПК у WINDOWS NT)
.\edc	командні файли , надані компанією Altera (.edc) , які генерують файли вихідних даних (.edo) для певних умов тестування.
.\lmf	Файли бібліотеки макросів , надані Altera (.lmf) , які відображають функції логіки користувача на еквівалентні функції логіки MAX+PLUS II
.\max2inc	файли Include (файли заголовків) із прототипами функцій для макрофункцій, розроблених Altera . Прототипи функцій відображають список портів (пінів) для макрофункцій, реалізованих у текстових файлах проекту (.tdf), написаних у AHDL
.\max2lib\edif	Містить основні елементи та функції макросів, що використовуються в інтерфейсі користувача EDIF
.\max2lib\mega_lpm	Містить мегафункції , включаючи функції бібліотеки параметризованих модулів (LPM), а також файли з відповідними прототипами в AHDL
.\max2lib\mf	Включає спеціальні та застарілі функції макросів (74 серії)

.\max2lib\prim	основні елементи, надані Altera
.\vhdlnn\altera	Містить бібліотеку altera з програмним пакетом maxplus2. Цей пакет містить усі примітиви, мегафункції та макрофункції системи MAX+PLUS II, що підтримується VHDL
.\vhdlnn\ieee	Містить бібліотеку ieee пакетів VHDL, включаючи std_logic_1164, std_logic_arith , std_logic_signed і std_logic_unsigned
.\vhdlnn\std	Містить бібліотеку std із стандартними пакетами та інструментами введення/виведення тексту, описаними в Довіднику стандартів IEEE у VHDL IEEE Standard Language VHDL P

Каталог \max2work містить навчальні файли та приклади та розділений на підкаталоги, описані в таблиці. 2.2.

Таблиця 2.2. Структура робочого каталогу \max2work системи MAX+PLUS II

Підкаталог	опис
.\ahdl	Містить зразки файлів , які ілюструють «Як використовувати AHDL» (Як вниз Використовуйте AHDL) в електронному довіднику (довідка MAX+PLUS II) і в посібнику MAX+PLUS II AHDL
.\chiptrip	Містить усі файли проекту для підручника chiptrip , описаного в посібнику MAX+PLUS II AHDL

.\edif	Містить усі зразки файлів, що ілюструють функції EDIF в електронному довіднику (довідка MAX+PLUS II)
.\tutorial	Містить інформаційний файл read.me для навчального проекту chiptrip . Усі файли , створені в проекті chiptrip , мають бути розташовані в цьому підкаталозі
.\vhdl	Містить зразки файлів, що ілюструють тему «Як використовувати VHDL» (Як вниз Використовуйте VHDL) в онлайн-посібнику (довідка MAX+PLUS II) і в посібнику MAX+PLUS II VHDL
.\verilog	Містить зразки файлів, що ілюструють «Як використовувати мову протоколу перевірки Verilog HDL » (Як вниз використання Verilog HDL) в онлайн-посібнику (довідка MAX+PLUS II) і в посібнику MAX+PLUS II Verilog HDL

MAX+PLUS II розроблено компанією Altera та забезпечує кросплатформне середовище розробки, незалежно від архітектури, яке можна легко адаптувати до конкретних вимог користувача. Система MAX+PLUS II має інструменти, які дозволяють зручно вводити проект, швидко вводити в експлуатацію та пряме програмування пристроїв.

Показано на рис. 2.3 Зміст Системне програмне забезпечення MAX+PLUS II — це повний пакет, що дозволяє створювати логічні схеми для програмованих логічних пристроїв Altera , включаючи сімейство Classic , MAX 7000, MAX 5000, MAX 6000, FLEX 9000, FLEX 8000 і FLEX 10K . пристрої . Інформацію про інші підтримувані сімейства пристроїв Altera можна знайти у файлі read.me у системі MAX+PLUS II.

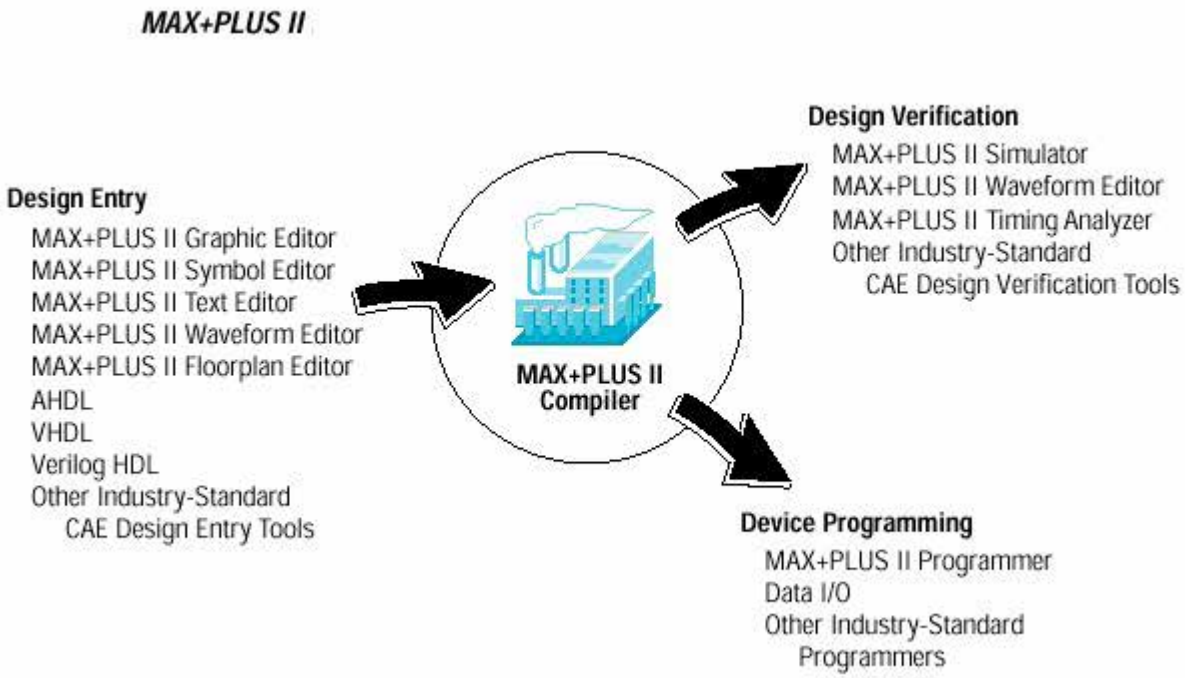


Рис. 2.3. Середовище проектування в системі МАКС+ПЛЮС II

Система MAX+PLUS II пропонує повний спектр можливостей логічного проектування: різноманітні інструменти проектування для створення ієрархічних проектів, ефективний логічний синтез, компіляція синхронізації, розділення, функціональне та часове тестування (симуляція), тестування кількох пов’язаних пристроїв, аналіз системні параметри синхронізації, автоматична локалізація несправностей та програмування та перевірка приладів. Система MAX+PLUS II може читати та записувати файли AHDL і файли трасування EDIF, файли Verilog HDL, а також файли схем OrCAD . Крім того, MAX+PLUS II зчитує файли трасування, згенеровані програмним забезпеченням Xilinx , і зберігає файли затримки у форматі SDF для зручності взаємодії з іншими галузевими стандартними пакетами.

Система MAX+PLUS II пропонує користувачеві багатий графічний інтерфейс, доповнений ілюстрованою онлайн-довідковою системою. Повна система MAX+PLUS II складається з 11 повністю вбудованих програм (рис. 2.4). (Логічний проект (project), включаючи всі підпроекти (subproject), у системі MAX+PLUS II називається проектом (project))

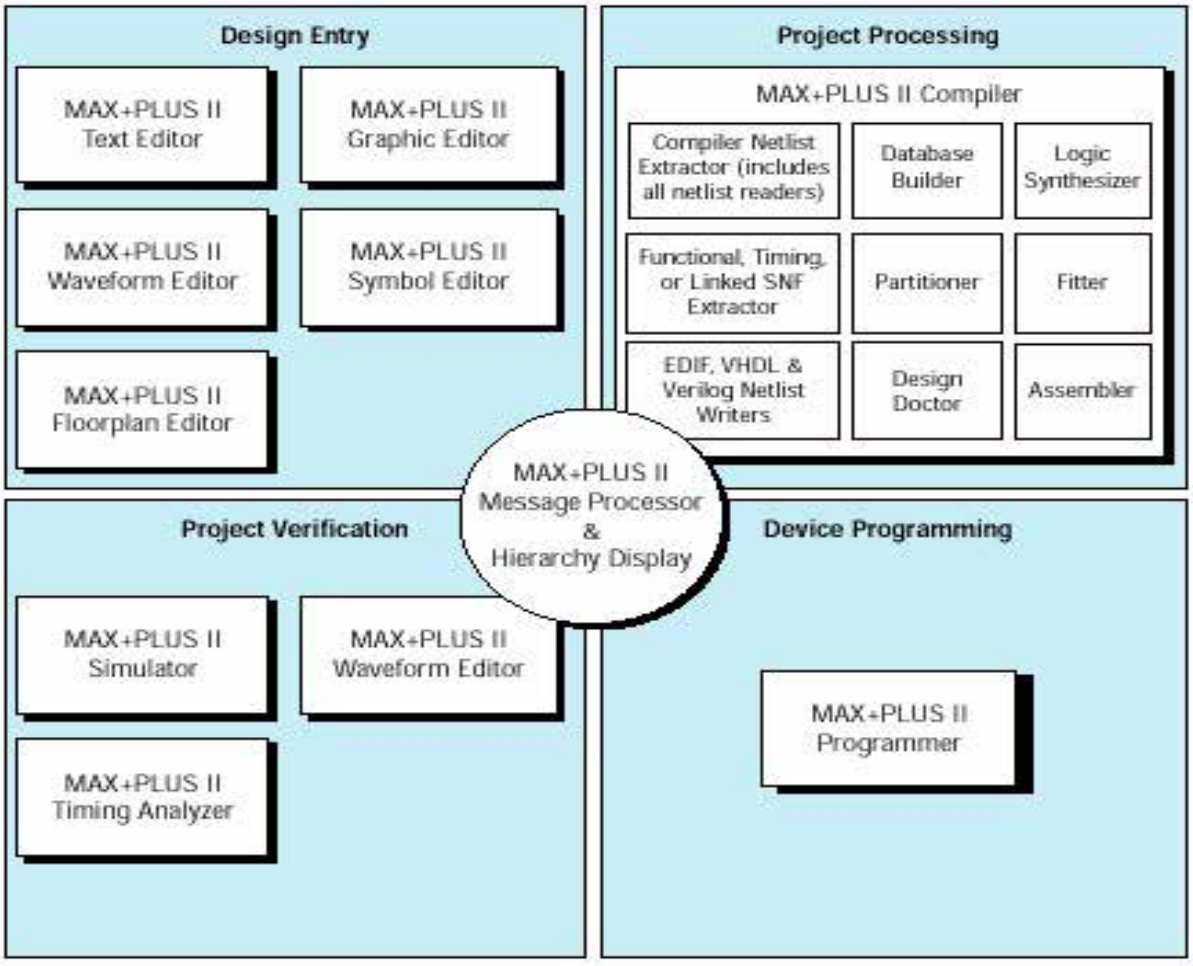


Рис. 2.4. Додатки в системі MAX+PLUS II

Щоб ввести опис дизайну (Design Entry) можна описати проект у вигляді файлу на мові опису обладнання, здісненого в зовнішньому редакторі, чи текстовому редакторі MAX+PLUS II, у вигляді електричної схеми за допомогою графічного редактора Редактор у вигляді тимчасової діаграми, створеної в редакторі сигналів Waveform редактор . Для зручності роботи зі складними ієрархічними проектами кожен підпроект може бути пов'язаний з символом, який можна редагувати за допомогою графічного редактора символів редактор . Розташування вузлів уздовж контактів LB і FPGA виконується за допомогою порівневого планувальника під назвою Floorplan редактор .

Перевірка проекту виконується за допомогою симулятора , результати якого зручно переглядати в редакторі сигналів Waveform Редактор , в якому створюються тестові ефекти.

Компіляція проекту, включаючи завантаження списку мереж Netlist Extractor), створення бази даних проекту, логічний синтез (лог синтез), виділення тимчасових функціональних параметрів дизайну (SNF Extractor), розділення (Partioner), трасування (Fitter) і генерація файлу програмування або завантаження (Assembler) виконуються за допомогою системного компілятора (Compiler)

Пряме програмування або завантаження конфігурацій пристрою за допомогою відповідного апаратного забезпечення здійснюється за допомогою модуля програматора (Programmer).

Багато функцій і команд, таких як відкриття файлів, введення пристрою, призначення контактів і шлюзів, а також компіляція поточного проекту, є схожими в багатьох програмах MAX+PLUS II. Редактори дизайну (графіки, тексту і сигналів) мають багато спільного з вторинними редакторами (рівневим плануванням і символічним). Кожен редактор створення проекту дозволяє вам виконувати подібні завдання подібним чином (наприклад, пошук сигналу або символу). Ви можете легко поєднати різні типи файлів проекту в ієрархічний проект, вибравши формат опису проекту, який найкраще підходить для кожного функціонального блоку. Велика бібліотека мега- та макрофункцій Altera, включаючи функції з бібліотеки LPM, надає широкі можливості для введення даних про дизайн

Ви можете працювати одночасно з різними програмами системи MAX+PLUS II. Наприклад, ви можете відкрити декілька файлів проекту та передавати інформацію між ними під час створення або тестування іншого проекту. Або, наприклад , відобразіть усе дерево проекту та перейдіть з одного рівня на інший у вікні попереднього перегляду, і вибраний файл з’явиться у вікні редактора, і відповідний редактор для кожного файлу буде викликано автоматично (рис. 2.5)

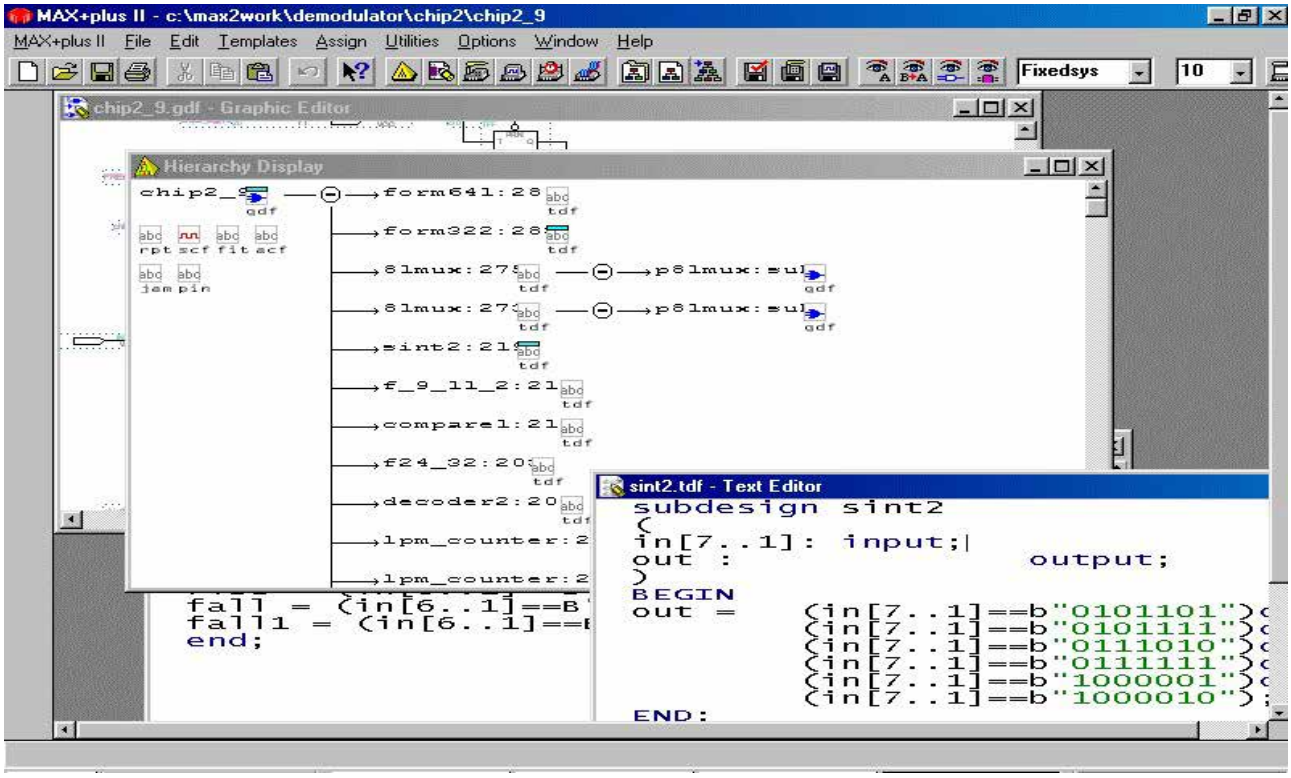


Рис. 2.5. Ієрархічний вигляд проекту

Основою системи MAX+PLUS II є компілятор, який надає потужні інструменти для обробки проектів і дозволяє задавати потрібні режими роботи компілятора. Автоматична локалізація помилок, звітування та велика документація про помилки роблять зміни в проекті швидкими та легкими. Ви можете генерувати вихідні файли в різних формах для різних задач, таких як робота функцій, синхронізація та зв’язок між декількома пристроями; аналіз часових параметрів; програмування пристрою.

2.6. Порядок розробки проекту

Процедуру розробки нового проекту (проекту) від концепції до реалізації можна спростити так:

- створення нового файлу (дизайн файл проекту або ієрархічна структура кількох файлів проекту з використанням різних редакторів створення проекту в системі MAX+PLUS II, тобто графічного, текстового та сигнального редакторів;
- вказавши назву файлу проекту верхнього рівня (Top с ієрархія) як назва проекту (Project ім'я);

- виділення сімейства FPGA для реалізації проекту. Користувач може самостійно призначити конкретний пристрій або залишити компілятору оцінку необхідних ресурсів;
- відкривши вікно компілятора (Compiler) і запустивши його, натиснувши кнопку «Пуск» , щоб розпочати компіляцію проекту. За бажанням ви можете підключити Timing SNF Extractor , щоб створити файл проводки, який використовується для тестування та аналізу синхронізації;

Якщо збірка пройшла успішно, можна перевірити та проаналізувати час, для чого слід виконати наступні кроки:

- щоб виконати аналіз часу, відкрийте вікно «Час» . Аналізатор , виберіть режим аналізу та натисніть кнопку Пуск ;
- щоб виконати тест, ви повинні спочатку створити векторний тестовий вектор у файлі тестового каналу (.scf) за допомогою редактора сигналів або у векторному файлі (.vec) за допомогою текстового редактора. Потім відкрийте вікно симулятора налагоджувача та натисніть кнопку Пуск ;
- програмування або завантаження конфігурації виконується шляхом запуску модуля програматора (Programmer) , а потім вставлення пристрою в адаптер програмування програматора MPU (Master) програмування Блок) або підключення пристроїв MasterBlaster , BitBlaster , ByteBlaster або кабелю для завантаження FLEX Кабель) до запрограмованого в системі пристрою;
- вибравши кнопку «Програмувати», щоб запрограмувати пристрої з пам’яттю EPROM або EEPROM (MAX, EPC), або вибравши кнопку «Налаштувати» , щоб завантажити конфігурацію пристрою з пам’яттю SRAM (FLEX).

Нижче ми детально розглянемо основні елементи реалізації проекту в системі MAX+PLUS II.

Систему MAX+PLUS II можна запустити двома способами: подвійним клацанням лівої кнопки миші на піктограмі MAX+PLUS II) або введенням maxplus2 у командному рядку.

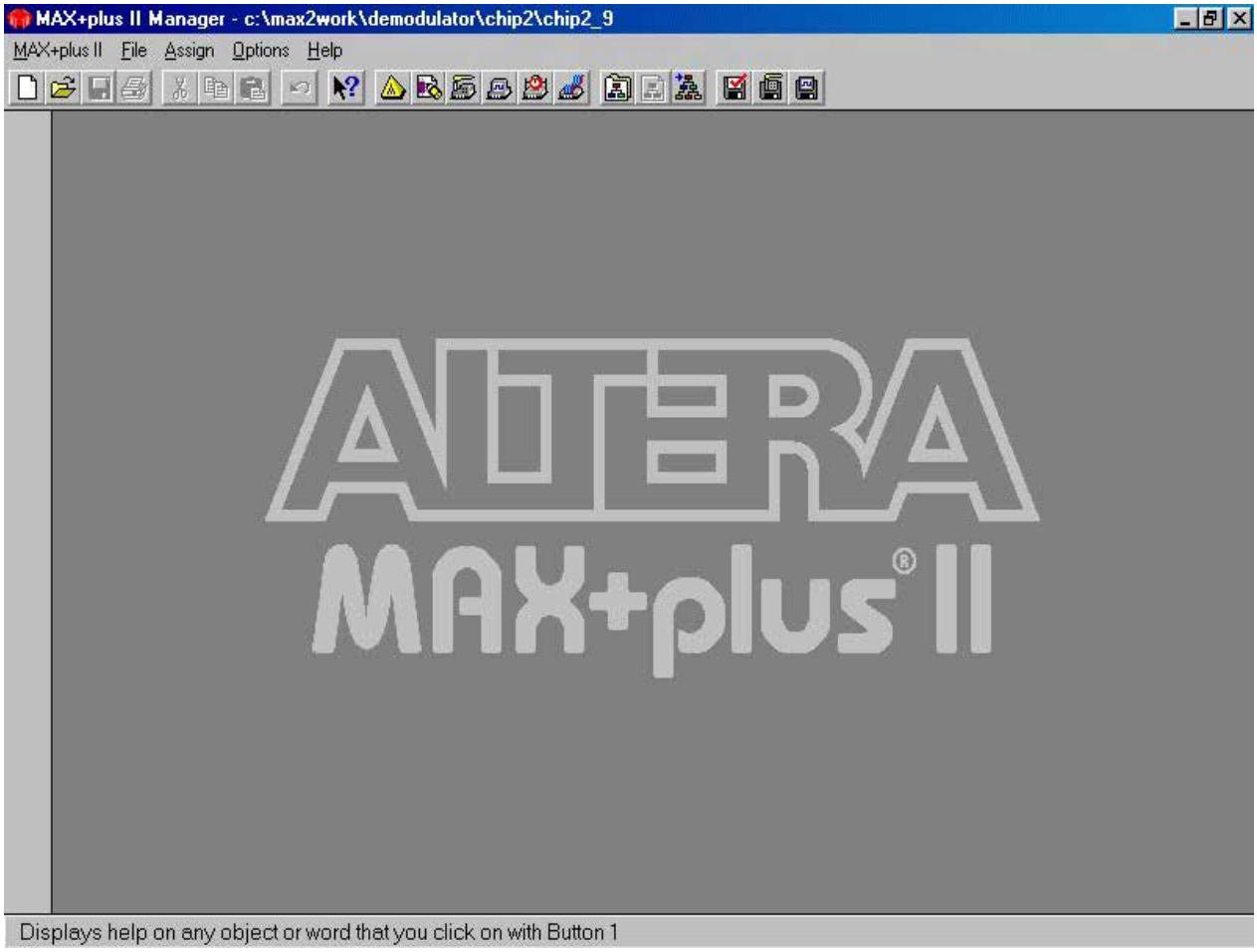


Рис. 2.6. Головне вікно системи MAX+PLUS II

Після початку роботи системи MAX+PLUS II автоматично запускається її головне вікно, меню якого містить усі програми системи MAX+PLUS II (див. рис. 2..6).

Назва проекту та поточний файл проекту відображаються у верхній частині вікна. За нею розташована панель меню, а під нею панель основних інструментів системи, що забезпечує швидкий доступ до її елементів. У нижній частині екрана є рядок підказок.

Елементи системи зручно викликати за допомогою програми MAX+PLUS II, як показано на рис. 2.7.

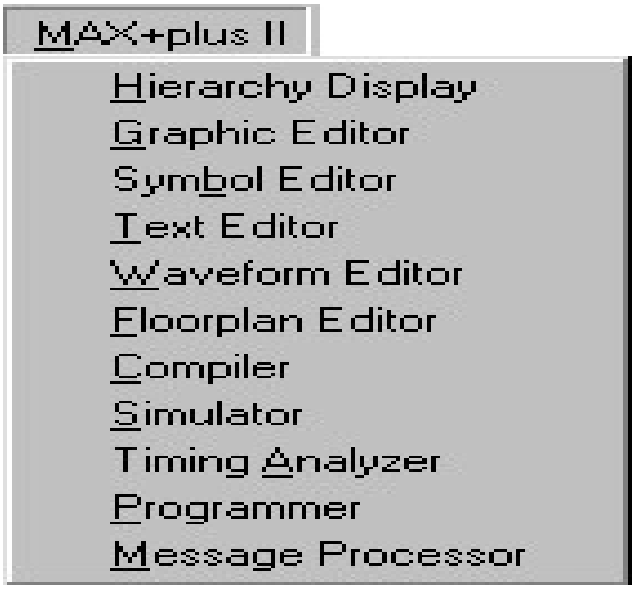


Рис. 2.7. Вікно меню MAX+PLUS II

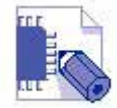
Розглянемо детальніше меню MAX+PLUS II (рис. 2.7). Програмне забезпечення системи MAX+PLUS II містить 11 додатків і основну оболонку управління. Різні програми, які дозволяють створювати файли проекту, можна запускати негайно, дозволяючи проектувальнику перемикатися між ними натисканням миші або за допомогою команд меню. Одна з фонових програм, наприклад компілятор, симулятор, аналізатор синхронізації та програматор, може працювати одночасно. Ті самі команди з різних програм працюють однаково, що полегшує завдання створення проектів. Ви можете згорнути вікно будь-якої програми до піктограми, не закриваючи програму, а потім розгорнути його знову. Це дозволяє користувачеві працювати ефективно, не захаращуючи головний екран.

У таблиці 2.3 наведено піктограми та опис програм.

Таблиця 2.3 Застосування системи MAX+PLUS II

додаток	Функцію виконано
	Огляд ієрархії – відображає поточну ієрархію папок у вигляді дерева, що представляють підпроекти . Ви можете візуально визначити, чи є файл проекту діаграмою, текстом або сигналом; які файли зараз відкриті; які допоміжні файли в проекті доступні користувачеві для

	редагування. Ви також можете безпосередньо відкрити або закрити один або кілька файлів дерева та ввести для них призначення ресурсів
	Графічний редактор - дозволяє розробити логічну схемотехніку в фактичному форматі екранного відображення WYSIWYG. Використовуючи запатентовані примітиви, мега- та макрофункції Altera як базові блоки програмування, користувач також може використовувати власні символи
	Редактор символів - дозволяє редагувати існуючі символи та створювати нові
! 	Текстовий редактор – дозволяє створювати та редагувати текстові файли проекту, написані мовами опису обладнання HDL, VHDL і Verilog HDL. Ви також можете використовувати цей редактор для створення, перегляду та редагування інших файлів ASCII, які використовуються іншими програмами MAX+PLUS II. Ви можете створювати файли в HDL та інших текстових редакторах, але цей текстовий редактор MAX+PLUS II пропонує переваги контекстно-залежної довідки, підсвічування синтаксису та готових шаблонів для AHDL, VHDL та Верило "
# 	Редактор сигналів - має подвійну функцію: інструмент проектування та інструмент для введення тестових сигналів і моніторингу результатів тестування

$ 	рівень за рівнем - дозволяє графічно призначати контакти пристрою та ресурси логічних елементів і блоків. Ви можете редагувати розпіновку на кресленні корпусу пристрою та призначати сигнали окремим логічним елементам у більш детальному поданні LAB . Ви також можете переглянути результати останньої збірки
%	Компілятор – логіка процесу, призначений для сімейств пристроїв Altera класичний . MAX 7000, MAX 5000, MAX 6000, FLEX 9000, FLEX 8000 і FLEX 10K. Більшість завдань виконується автоматично. Однак користувач може контролювати процес компіляції
	повністю або частково
	Симулятор – дозволяє тестувати логічні операції та внутрішню синхронізацію проектованої логічної системи. Можливі три режими тестування : функціональний, за часом і тестування кількох взаємопов’язаних пристроїв
!& ' 	Аналізатор часу - аналізує роботу розробленої логічної схеми після її синтезу та оптимізації компілятором, що дозволяє оцінити затримки, що виникають у схемі.
(	Програматор – дозволяє програмувати, налаштовувати, перевіряти та тестувати пристрої Altera

(	Генератор повідомлень – відображає повідомлення про помилки, попередження та інформацію про стан проекту користувача та дозволяє користувачеві автоматично знаходити джерело повідомлення у вихідному або допоміжному файлі та в плані призначення .

На рис. 2.3 ілюструється процес розробки багатовіконного проекту, коли вікно огляду ієрархії та текстовий редактор відкриті одночасно.

Перш ніж почати працювати в системі MAX+PLUS II, ви повинні зрозуміти різницю між файлами дизайну , файлами підтримки.

Програма проекту — це текстовий, графічний чи сигнальний файл, що створюється за допомогою графічних редакторів або редакторів сигналів системи MAX+PLUS II або будь-якої іншої промислової стандартної схеми або текстовий редактор або за допомогою netlist writer , доступний у пакетах , що підтримують VHDL і Verilog HDL. Даний файл містить логіку проекту MAX+PLUS II і обробляється системою. Компілятор автоматично обробляє наступні файли проекту:

графічні файли проекту (.gdf);

проектувати текстові файли в AHDL (.tdf) ;

сигнальні файли проекту (.wdf) ;

файли проекту VHDL (.vhd) ;

файли проекту Verilog (.v);

файли схем OrCAD (.sch);

вхідні файли EDIF (edf);

файли у форматі Xilinx Список мереж (.xnf) ;

файли проекту Altera (.adf) ;

цифрові машинні файли (.smf) .

Файли підтримки – це файли, які пов’язані з проектом MAX+PLUS II, але не є частиною дерева ієрархії проекту. Більшість із цих файлів не містять логіки проектування. Частина створюються автоматично програмами системи MAX+PLUS II, другі створює користувач. Прикладами файлів підтримки є

файли призначень і конфігурації (.acf) , файли символів (.sym), файли звітів (.rpt) і тестові векторні файли (.vec) .

Проект складається з усіх файлів в ієрархічній структурі проекту, включаючи файли підтримки та вихідні файли. Ім’я проекту — це ім’я файлу ghjtrnf більшого рівня без розширення. Програма MAX+PLUS II виконує перевірку, тестування, синхронізацію та програмування всього проекту одночасно, хоча користувач може одночасно редагувати файли цього проекту в іншому проекті. Наприклад, під час створення проекту project1 користувач може редагувати файл проекту TDF, який також є файлом проекту project2 , і зберегти його; однак, якщо він хоче скомпілювати його, йому доведеться спочатку встановити project2 як назву проекту.

Для кожного проекту створіть окремий підкаталог у робочому каталозі системи MAX+PLUS II (\max2work).

У системі MAX+PLUS II усі інструменти, що дозволяють створити логічний дизайн, легко доступні. Розробка дизайну прискорюється доступними стандартними логічними функціями, включаючи примітиви, мегафункції , бібліотеку параметризованих модулів LPM і застарілі макрофункції серії 74. Надзвичайно шкідливо використовувати застарілі бібліотеки та переносити стандартні схеми серії TTL на FPGA . Проект повинен бути розроблений спеціально для архітектури FPGA, щоб отримати більш-менш розумні результати.

Файли схем проекту створюються в графічному редакторі MAX+PLUS II. Ви також можете відкривати, редагувати та зберігати діаграми, створені за допомогою редактора діаграм OrCAD .

проекти на AHDL, VHDL і Verilog HDL в текстовому редакторі MAX+PLUS II або будь-якому іншому текстовому редакторі.

Дизайни сигналів створюються в редакторі сигналів MAX+PLUS II.

EDIF і Xilinx , розроблені за допомогою інших стандартних інструментів EDA, можна імпортувати в середовище MAX+PLUS II.

Програмні пакети Altera A+PLUS і SAM+PLUS можуть бути інтегровані в середовище MAX+PLUS II.

Розподіл фізичних ресурсів для будь-якого вузла або контакту в поточному проекті можна ввести в графічне середовище за допомогою багаторівневого планувальника. Зберігає завдання проекту у файлі з розширенням . acf , який зберігає всі типи призначень ресурсів, зондів (Probes) і пристроїв (Devices) , а також параметри конфігурації (Asign) для компілятора, симулятора та аналізатора часу.

який представляє будь-який тип файлу проекту, можна автоматично створити в будь-якому редакторі проекту MAX+PLUS II за допомогою команди Файл / Створити За замовчуванням символ Команда . За допомогою редактора символів MAX+PLUS II ви можете редагувати символи або створювати власні, а потім використовувати їх у будь-якому файлі діаграми у вашому проекті.

Методи визначення файлів проекту показано на рисунку 2.8.

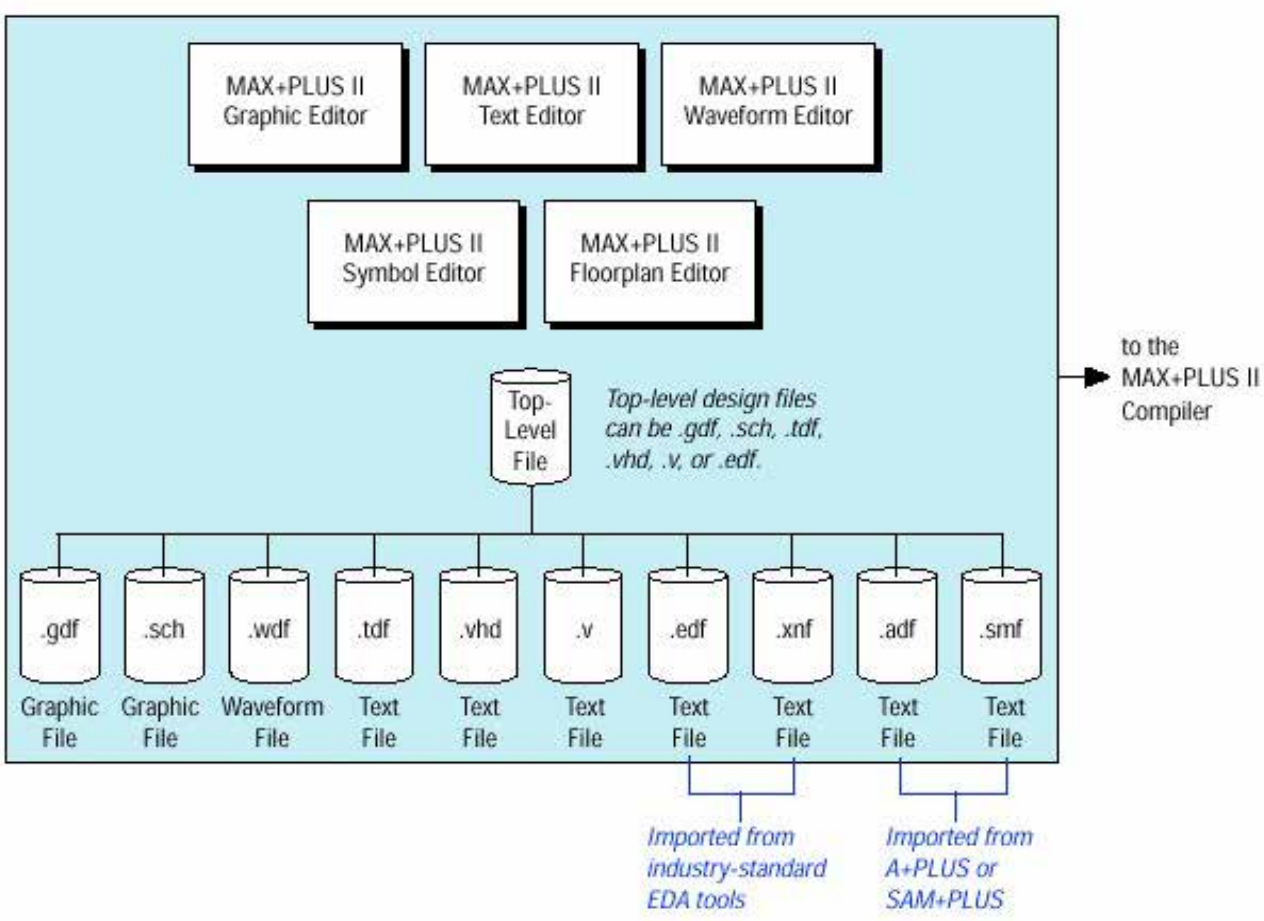


Рис. 2.8. Способи опису файлів проекту

В ієрархічній структурі проекту на кожному рівні використання файлів із розширеннями є змішаним . gdf . tdf . vhd.v. edf sch Хоча файли з розширенням . wdf . xnf . adf . смф має бути на найнижчому рівні ієрархії проекту або бути єдиним файлом.

команди меню «Призначити» для введення, редагування та видалення типів призначення ресурсів, пристроїв і параметрів, які керують компіляцією проекту, включаючи логічний синтез, розділення та вирівнювання. На рис. 2.9 показано команди меню «Призначити». Розробник може виконувати призначення для даного проекту незважаючи від того, чи відкрито файл проекту чи вікно програми. Система MAX+PLUS II зберігає інформацію про проект у файлі з розширенням . acf . Зміни до призначень, зроблені у вікні планувальника рівнів , також зберігаються у файлі ACF. Ви також можете редагувати ACF-файл проекту в текстовому редакторі.

Рис. 2.9. Меню призначення проекту призначити .

Наступні функції є спільними для всіх програм MAX+PLUS II: призначення пристрою, ресурсу та зонда; збереження попередньої версії; глобальні параметри засобу в проекті; деякі параметри проекту. загальні вимоги до часових параметрів проекту; проект глобального логічного синтезу . Давайте розглянемо їх ближче.

Пристрій Altera , контакт чи логічний пристрій, що виконує деяке, визначене користувачем завдання. Користувач може призначити логіку ресурсам пристрою, щоб гарантувати, що компілятор MAX+PLUS II адаптується до дизайну за бажанням. Розрізняють наступні види завдань.

Кліка призначення вказує, які логічні функції повинні залишатися разом. Групування логічних функцій у кліки гарантує, що вони реалізовані в одному блоці логічної структури LAB, блоці комірки пам’яті EAB, рядку чи пристрої. Для призначення клацання використовується команда Призначити / Натиснути (рис. 2.10)

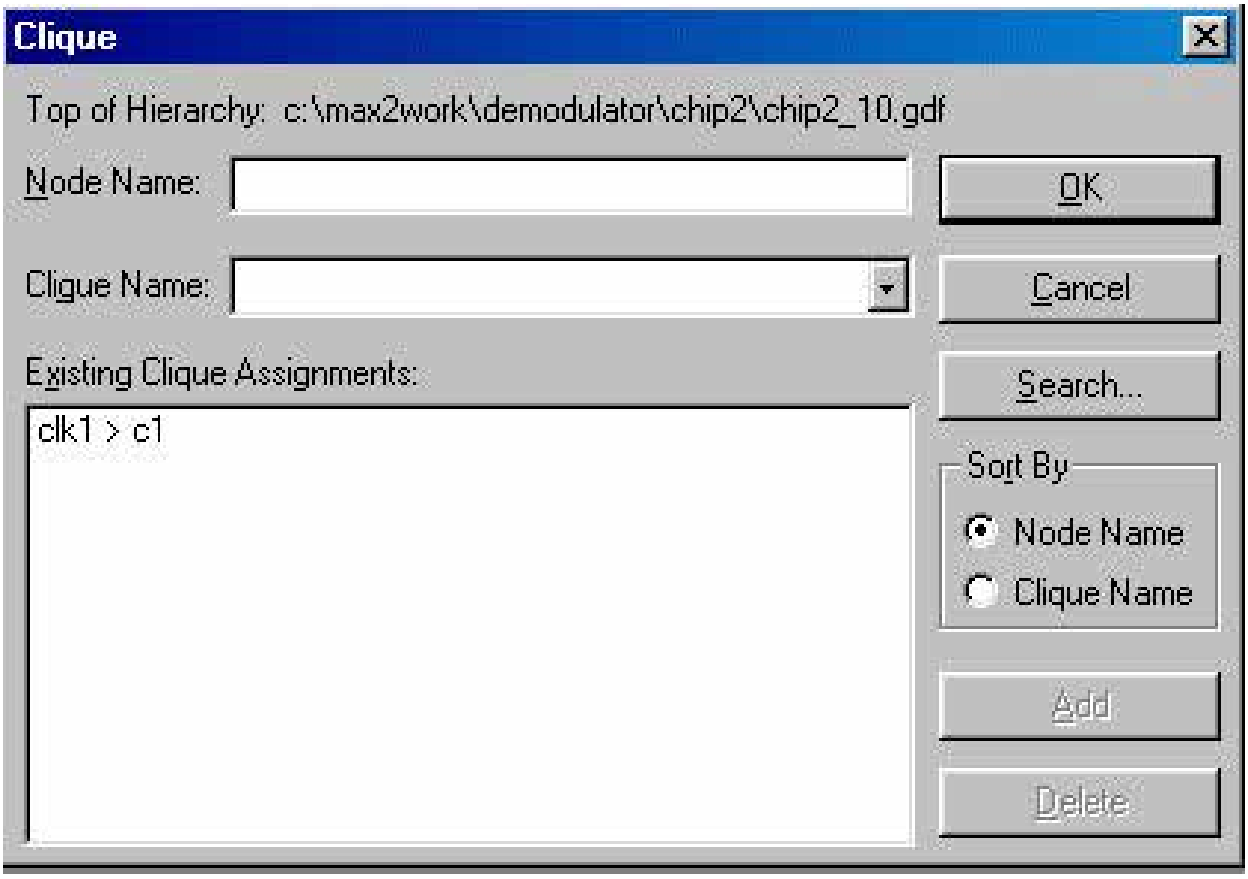


Рис. 2.10. Призначення кліка (команда Assign / Click)

У сфері клік name вказує назву кліки. У полі Вузол name вказує назву вузла. Вузли, з'єднані в кліки (іноді їх називають групами, але ця назва призводить до плутанини з концепцією групи в AHDL), будуть розміщені в тому самому LB під час компіляції

Токен assignment визначає , які логічні функції мають бути реалізовані в одному пристрої, якщо проект розділений на частини (кілька пристроїв).

Pin assignment призначає вхід або вихід однієї логічної функції, такої як примітивна функція або *мегафункція* , на певний контакт або вертикальний (горизонтальний) ряд контактів на FPGA.

Місцезнаходження призначення призначає одну логічну функцію, таку як вихід примітивної функції або мегафункції, *певній* комірці на інтегральній схемі, такій як логічний вентиль, комірка вводу/виводу, комірка пам'яті, блоки LAB і EAB, горизонтальні або вертикальні рядки .

Призначення пінів, мікросхем і комірок здійснюється за допомогою команди Assign / Pin / Location / Chip , вікно якої показано на рис. 2.11

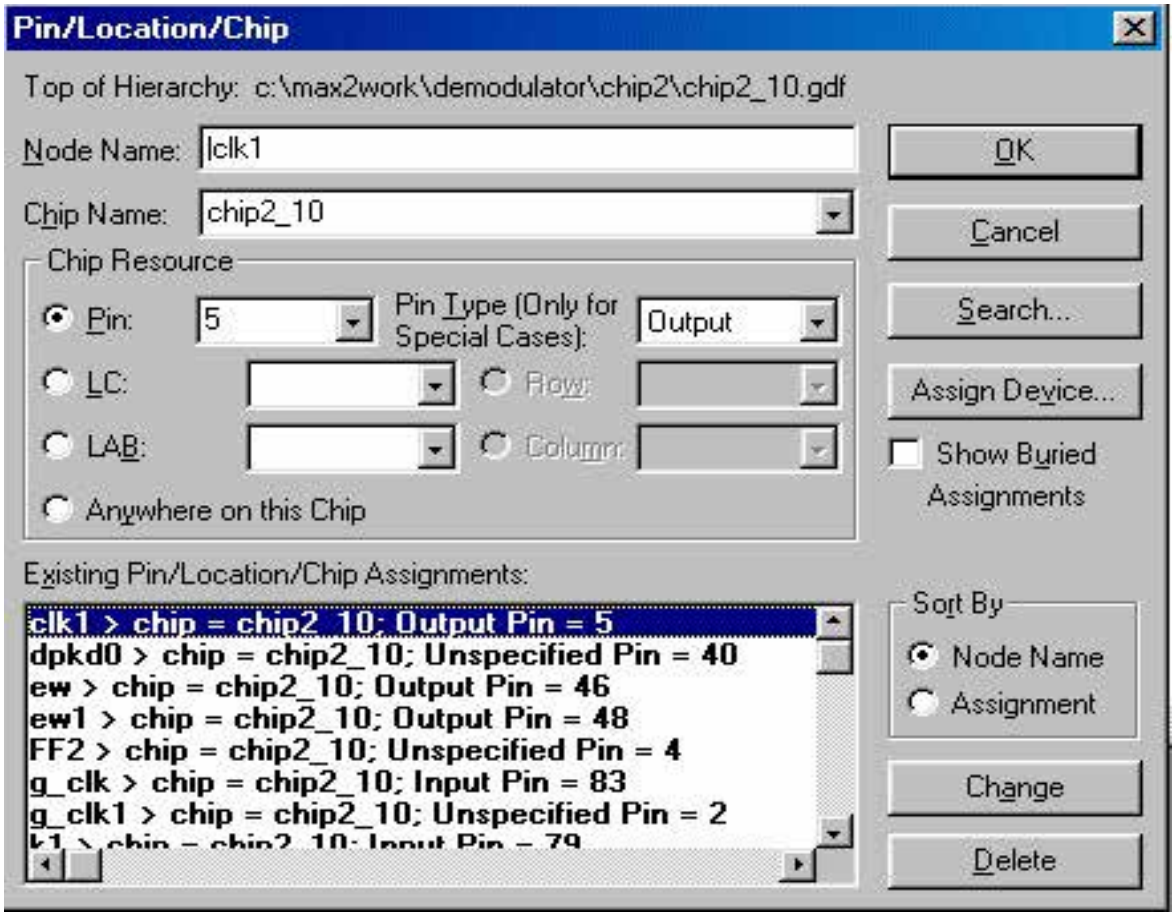


Рис. 2.11 . Вікно команд Призначити/Закріпити/Місце/Створити елемент

У полях цього вікна ви можете встановити номер PIN-коду (Pin), логічну комірку (LC) або блок (LAB), а також використовувати кнопки «Змінити» та «Видалити» , щоб змінити призначення.

Зонд assignment призначає унікальне ім'я , *яке легко запам'ятати* , входу або виходу логічної функції. Це завдання дуже корисне при моделюванні системи. Для призначення зонда використовується команда Призначити / Зонд (рис. 2.12)

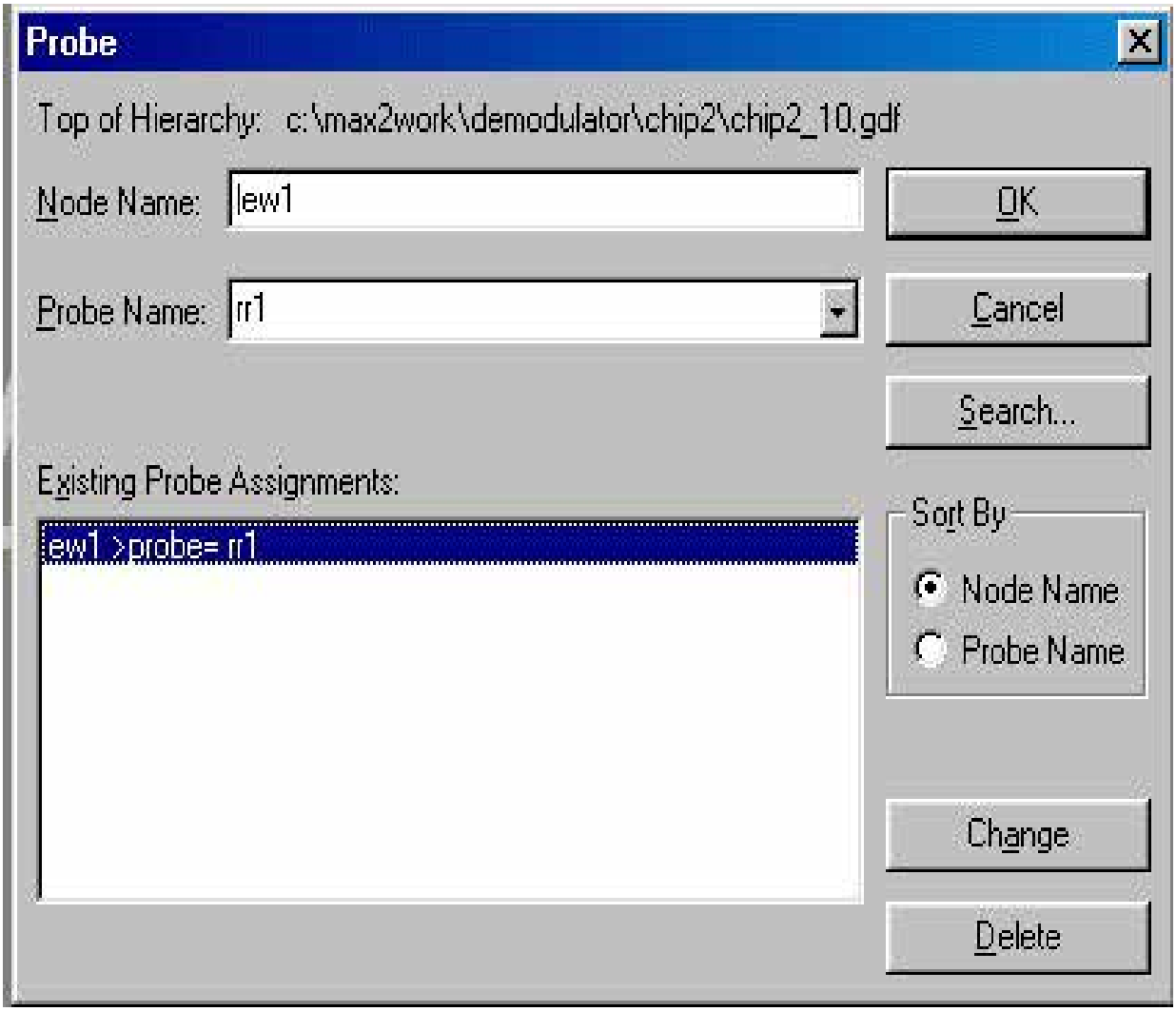


Рис. 2.12. Меню команд Призначити / Опитувати

Підключено шпилька призначення визначає зовнішнє підключення двох або більше контактів у схемі користувача. Ця інформація також корисна в режимі тестування часу та під час тестування кількох підключених проектів. Щоб призначити підключені контакти, використовуйте команду Призначити / Підключено Штифти (рис. 2.13)

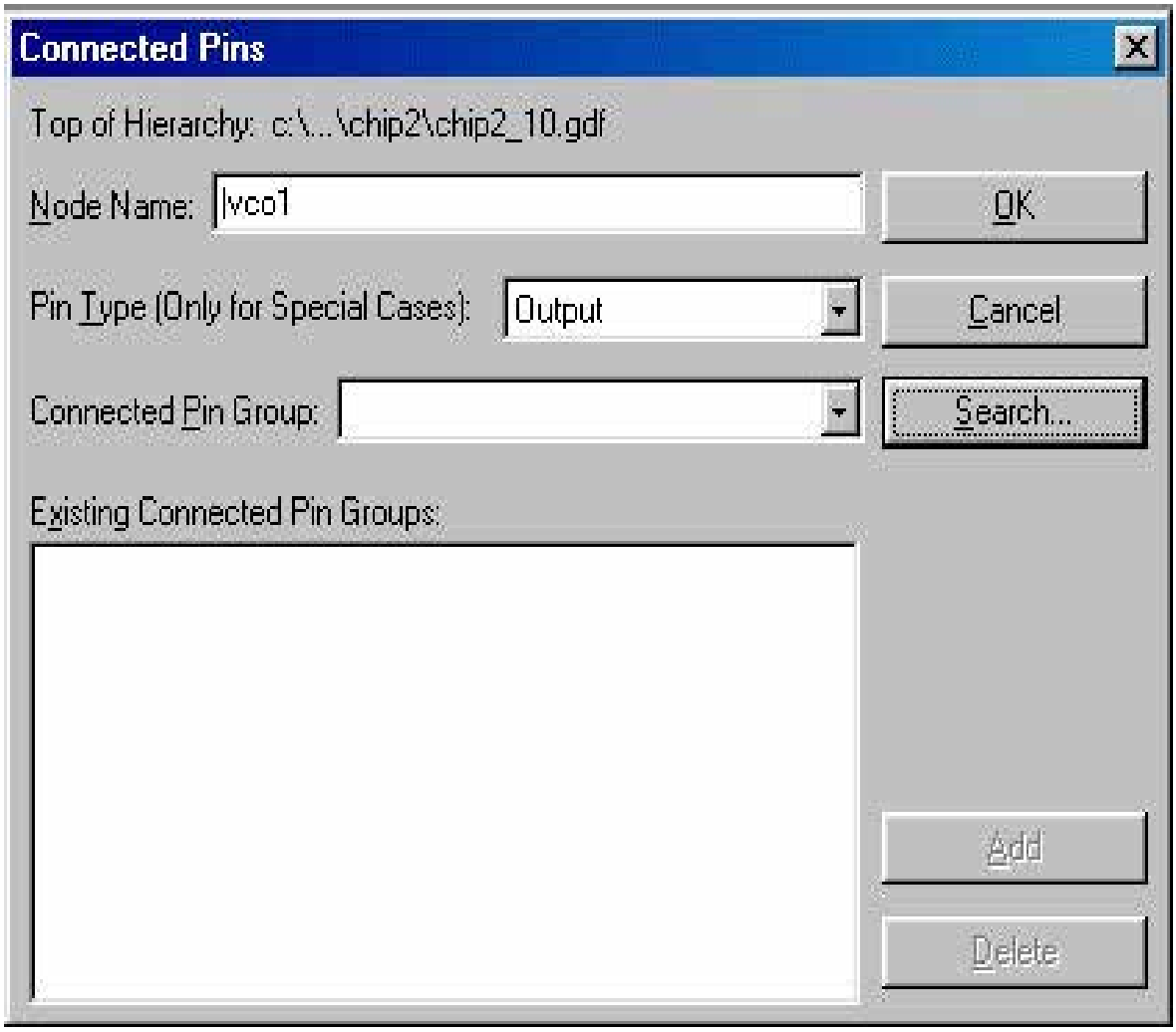


Рис.2.13 . Меню Команди Призначити/Підключити контакти .

Місцевий розгром assignment призначає вихідний коефіцієнт розповсюдження вузла логічному вентилю в тій самій LAB, що й вузол, або сусідньому LAB, що примикає до вибраного вузла, використовуючи спільні локальні з'єднання . Локальна маршрутизація також відбувається між вузлом, розташованим у блоці LAB на схемі пристрою, і вихідним висновком, до якого він підключений. Призначення локальної траси виконується за допомогою команди Призначити / Локальна розгром (Рис. 2.14)

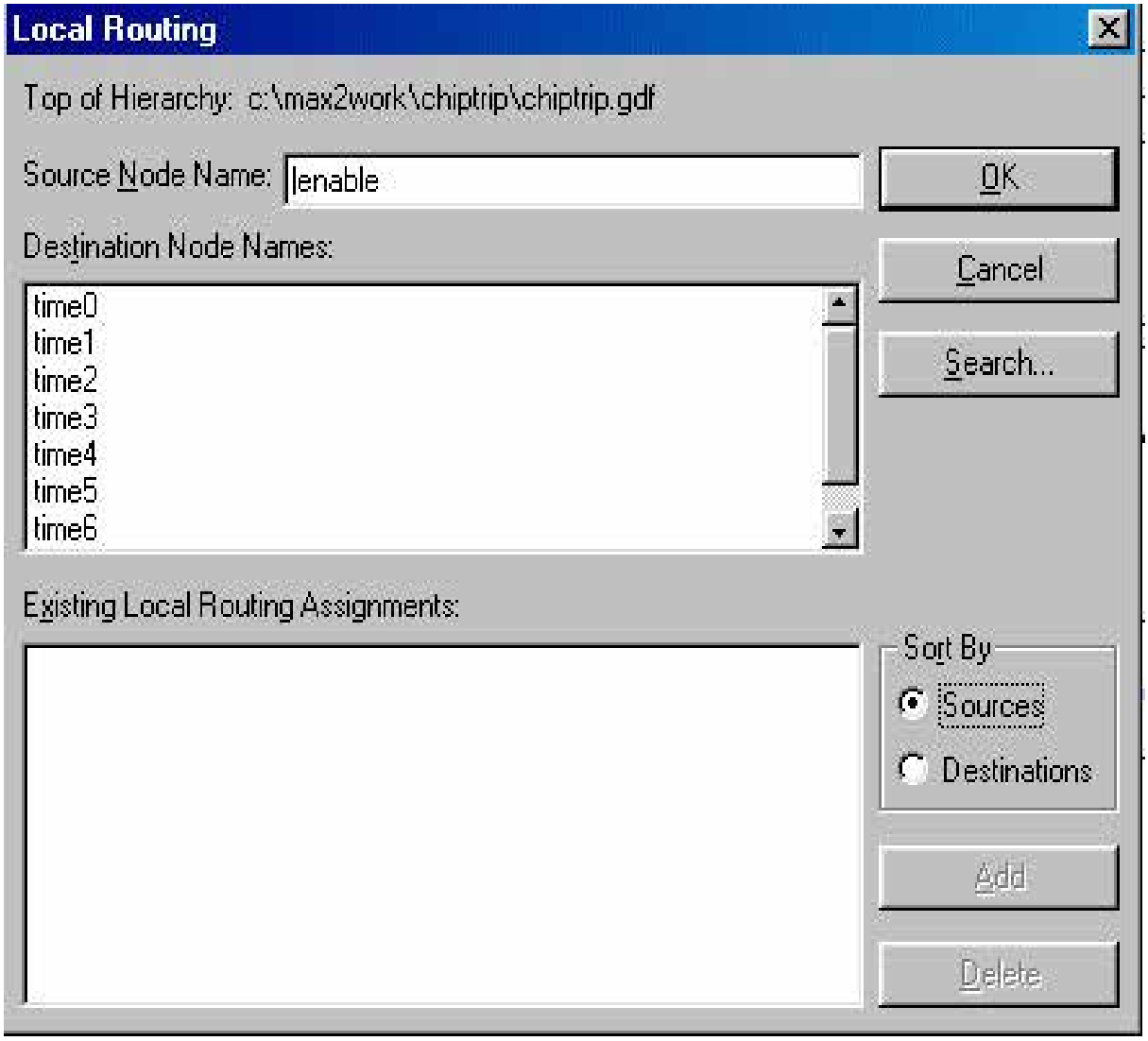


Рис. 2.14. Вікно Команди призначення/локальної маршрутизації

пристрій assignment призначає тип FPGA , в якому буде реалізовано проект. Якщо ваш проект складається з кількох пристроїв, цей тип призначення призначає мікросхеми певним пристроям. Ви також можете вибрати AUTO і дозволити компілятору вибрати пристрій із певного сімейства пристроїв. Ви можете керувати процесом автоматичного вибору пристроїв, вказавши обсяг і кількість пристроїв у сімействі. Якщо проект занадто великий для виконання на одному пристрої, ви можете вказати тип і кількість додаткових пристроїв. Щоб вибрати пристрій, використовуйте команду Призначити / Пристрій (Рис. 2.15)

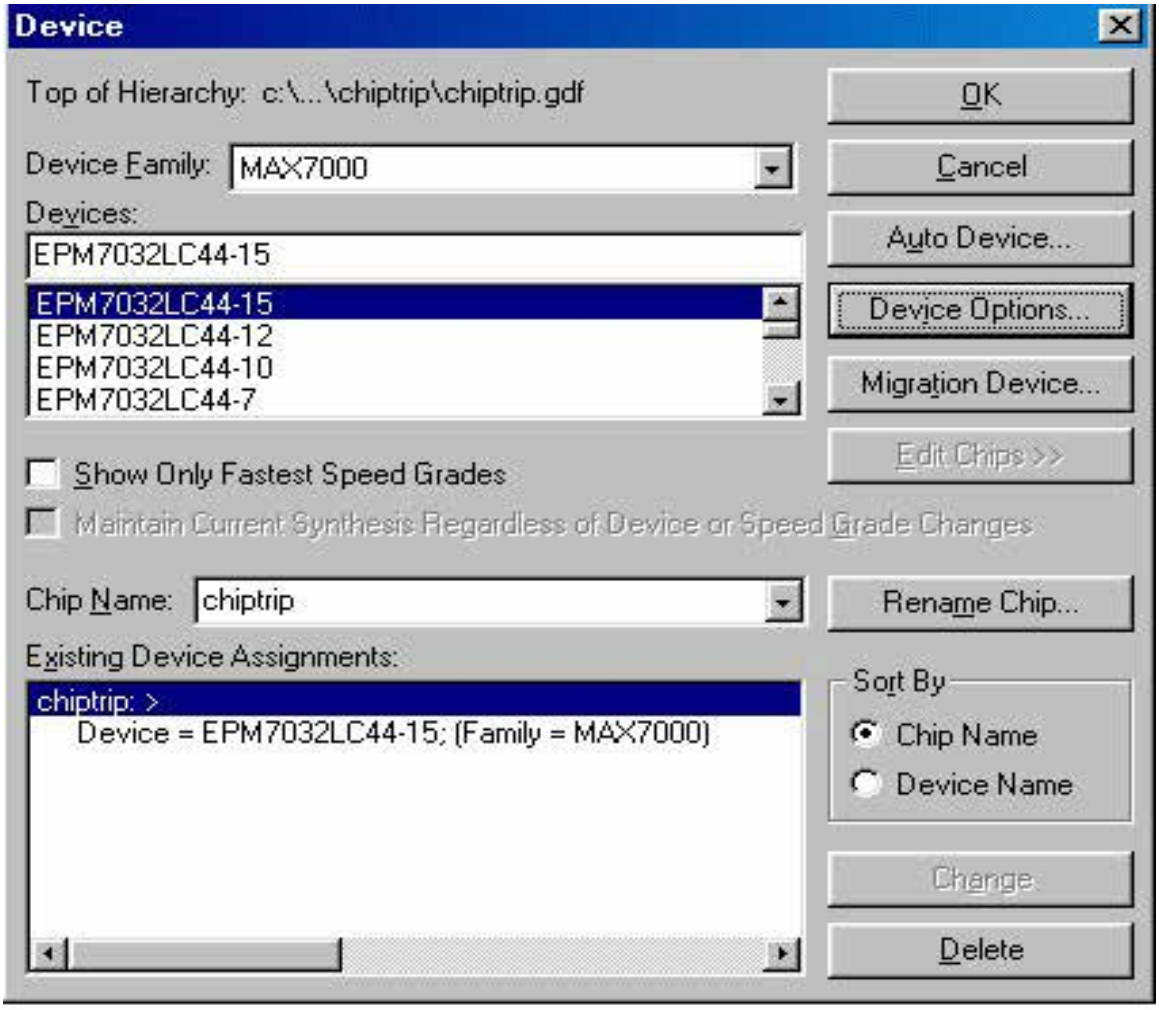


Рис. 2.15. Вікно «Призначити / команди пристрою »

Логіка варіант призначення контролює логічний синтез окремих логічних функцій під час компіляції, використовуючи стиль логічного синтезу та/або окремі параметри логічного синтезу . Altera надає велику кількість логічних параметрів, а також попередньо створених стилів, кожен з яких є набором параметрів логічних параметрів, об’єднаних єдиною назвою стилю синтезу (Synthesis стиль). Розробник в праві використовувати запропоновані стилі або створювати нові. Дані стилі синтезу дозволяють налаштувати параметри синтезу для певного сімейства пристроїв, враховуючи архітектуру сімейства. Щоб встановити стилі синтезу, використовуйте команду Призначити / Логіка Опції (Рис. 2.16)

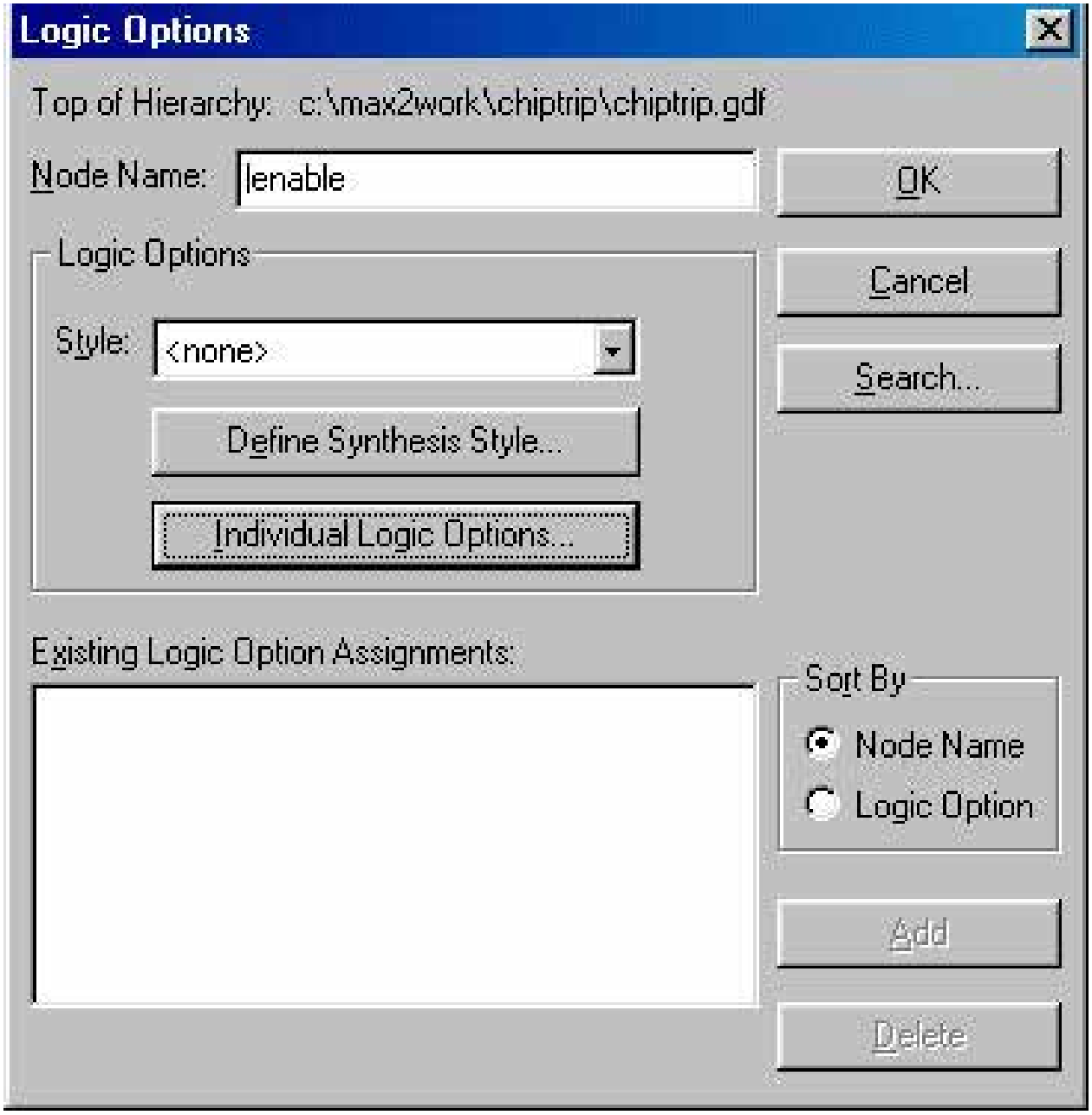


Рис. 2.16 . Вікно Призначення команд /логічні параметри

Терміни призначення контролює логічний синтез і налаштування окремих логічних функцій для отримання необхідних характеристик часу затримки t_{PD} (вхід - незареєстрований вихід), t_{CO} (тактовий сигнал - вихід), t_{SU} (тактовий сигнал - настройка часу), f_{MAX} (тактова частота). Користувач також може розірвати з'єднання між даними шляхами поширення сигналу (у системі MAX+PLUS II називаються «вузлами») та іншими вузлами в проекті. Призначення параметрів синхронізації вузла виконується за допомогою команди Призначити / Час Вимоги (Рис. 2.17)

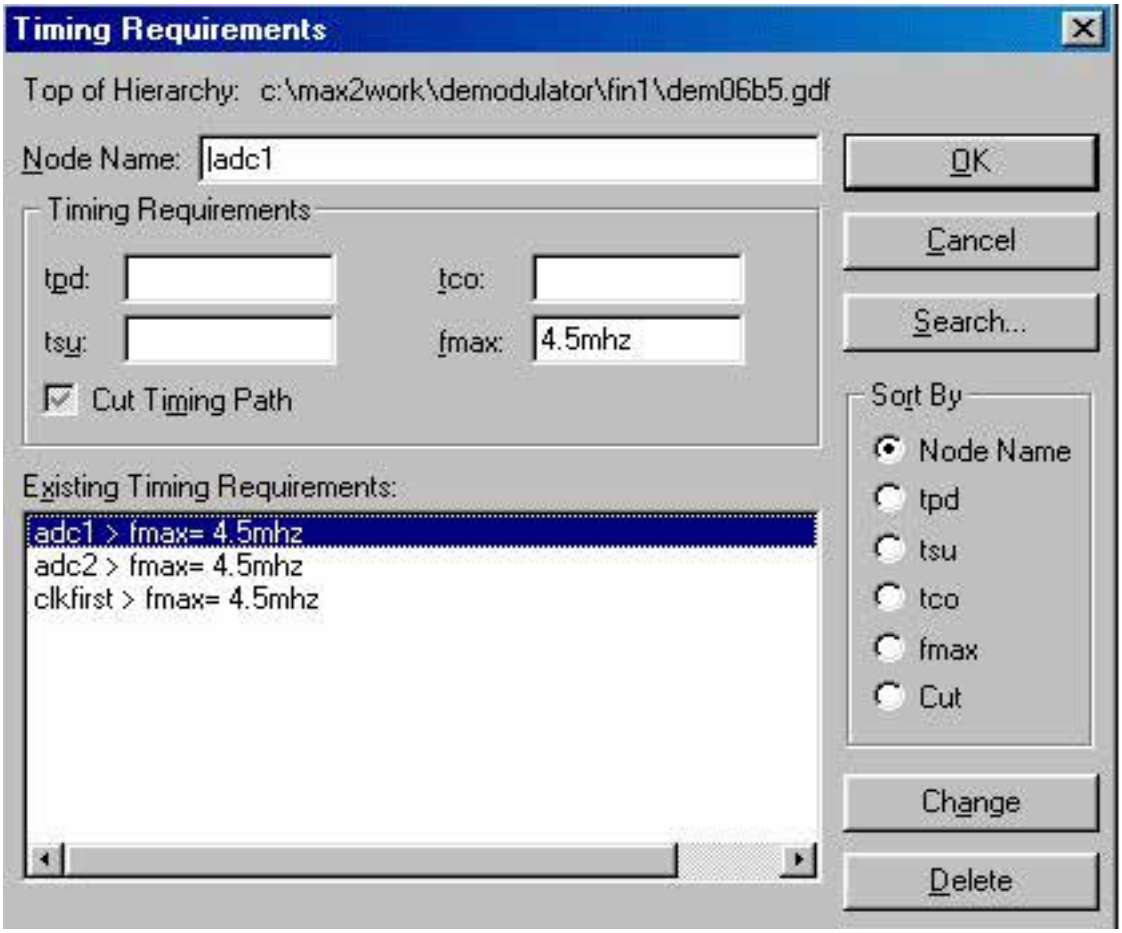


Рис. 2.17 . Вимоги до часу призначення/проектування командного вікна

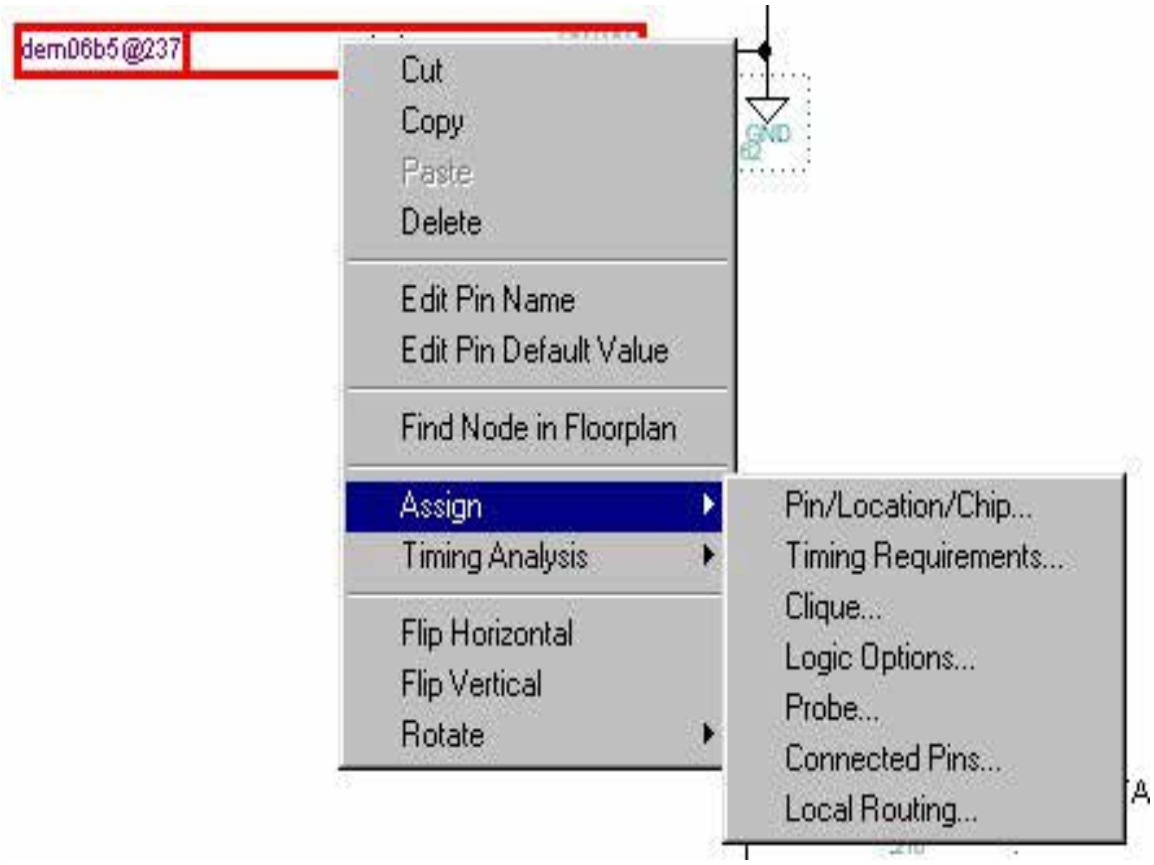


Рис.2.18.

З команд меню «Призначити» . призначення можна зробити, клацнувши правою кнопкою миші на вибраному вузлі проекту та вибравши відповідне призначення з контекстного меню (рис. 2.18)

Ви можете визначити глобальні параметри пристрою, які компілятор використовуватиме для всіх пристроїв під час обробки вашого проекту. Щоб зарезервувати додаткові логічні можливості на майбутнє, ви можете вказати відсоток контактів і шлюзів, що повинні залишитися невикористаними за час текучої компіляції. Ви також можете вказати налаштування бітів і контактів опцій пристрою в конфігурації пристрою, яка використовується для багатьох цілей. Наприклад, можна встановити біт захисту від читання (Security bit) є глобальним за замовчуванням, що запобігає піратству топології EPROM або пристроїв на основі EEPROM.

Ви можете вказати імена та глобальні налаштування, які компілятор використовуватиме для параметрів у всіх параметризованих функціях у вашому проекті.

Ви можете ввести глобальні вимоги до часу для проекту, визначивши загальні характеристики часу затримки TPD _

(вхід - вихід незареєстрований), t_{CO} (тактовий сигнал - вихід), t_{SU} (тактовий сигнал - установка часу), f_{MAX} (частота тактового сигналу). Ви також можете розірвати з’єднання між усіма двонаправленими петлями зворотного зв’язку, попередньо встановленими (набір 1) і чіткими (набір 0) шляхами сигналу та іншими синхронізуючими мережами в проекті. для ці цілі використовується команда Призначені/глобальні вимоги до часу проекту (рис. 2.19)

Рис. 2.19 . Вікно Команди Призначити/Загальні вимоги до часу проекту

Вирізати прапорець все Двонаправлений Зворотній зв'язок терміни Шляхи дозволяють усунути всі схеми зворотного зв’язку для двонаправлених контактів

Вирізати прапорець все Очистити та встановити терміни Шляхи дозволяють видалити зв’язки між усіма скинутими та запрограмованими схемами в проекті.

Ігнорувати прапорець терміни завдання Поки Фітинг дозволяє запустити маршрутизатор (Fitter) без урахування часових обмежень проекту. Після зняття цього прапорця та встановлення параметрів часу, т.зв контрольований синтез (час водіння синтез).

З власного досвіду ми помітили, що на початку, щоб прискорити компіляцію, ви повинні поставити галочку в цьому полі, а потім, «облизавши » проект, зняти його. Пам'ятайте, що контрольований синтез часу можливий лише для пристроїв FLEX, для пристроїв MAX час затримки заздалегідь визначений, а у разі призначення параметрів часу перевіряється лише відповідність параметрів, отриманих під час синтезу, заданим.

Ви можете зробити глобальні налаштування компілятора для логічного синтезу вашого проекту. Ви можете встановити стиль логічного синтезу за замовчуванням, вказати ступінь оптимізації ресурсу швидкості та вказати компілятору вибрати автоматичні глобальні керуючі сигнали, такі як Clock , Clear (встановлено на 0), Preset (встановлено на 1) і Output Увімкнути (дозволити вихід). Ви також можете вибрати стандартний або багаторівневий режим синтезу компілятора, режим цифрового машинного кодування з 1 при включенні та режим автоматичного пакування реєстрів. Ви також можете вибрати автоматичну реалізацію логіки у високошвидкісних вхідних або вихідних вентилях і комірках введення/виведення (Slew потік), відкриті зливні хомути (відкриті відтік контакти) і блоки комірок пам'яті EAB. Для призначення параметрів глобального логічного синтезу використовуйте команду Призначити / Глобальний Дизайн Логіка Синтез (Рис. 2.20)

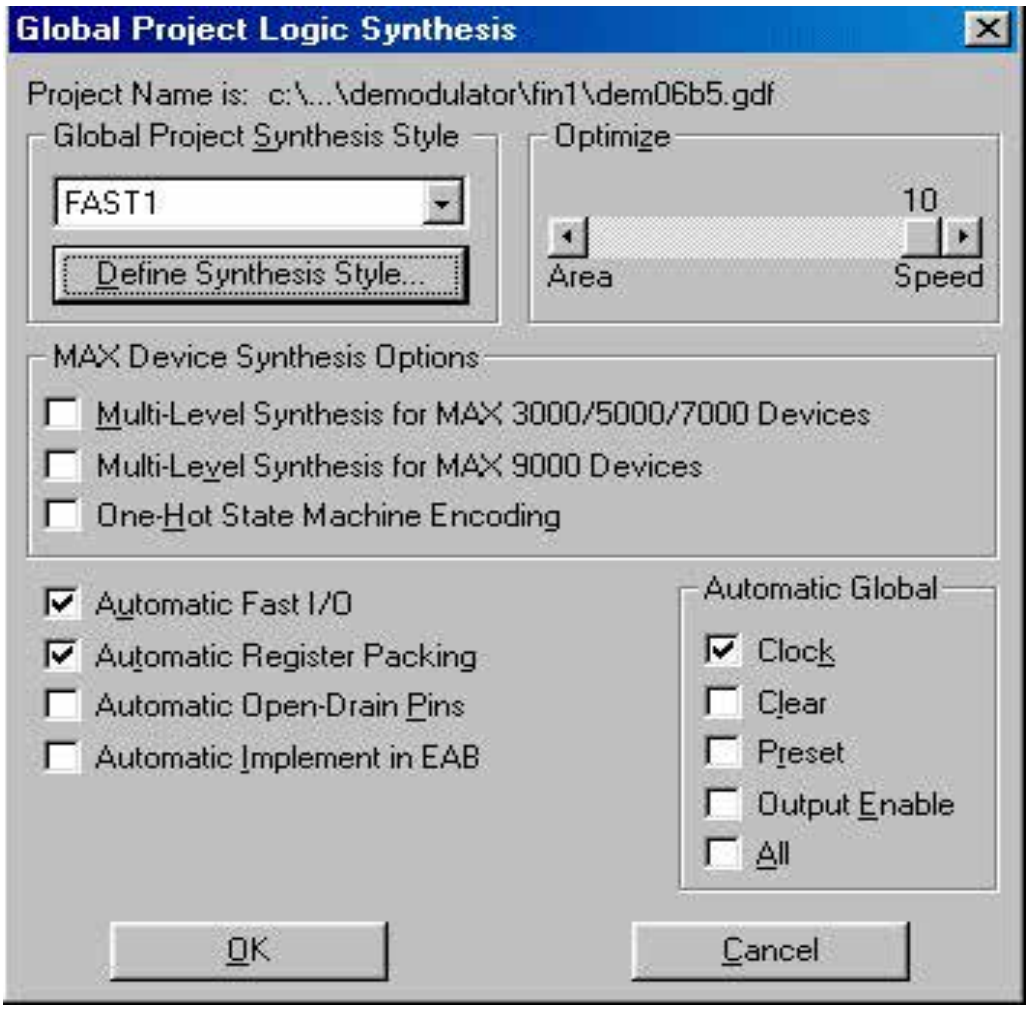


Рис.2.20 . Вікно Команди призначення/загального синтезу логіки проектування.

Визначте кнопку Синтез Стиль дозволяє вибрати більш тонкі параметри стилю синтезу (рис. 2.21)

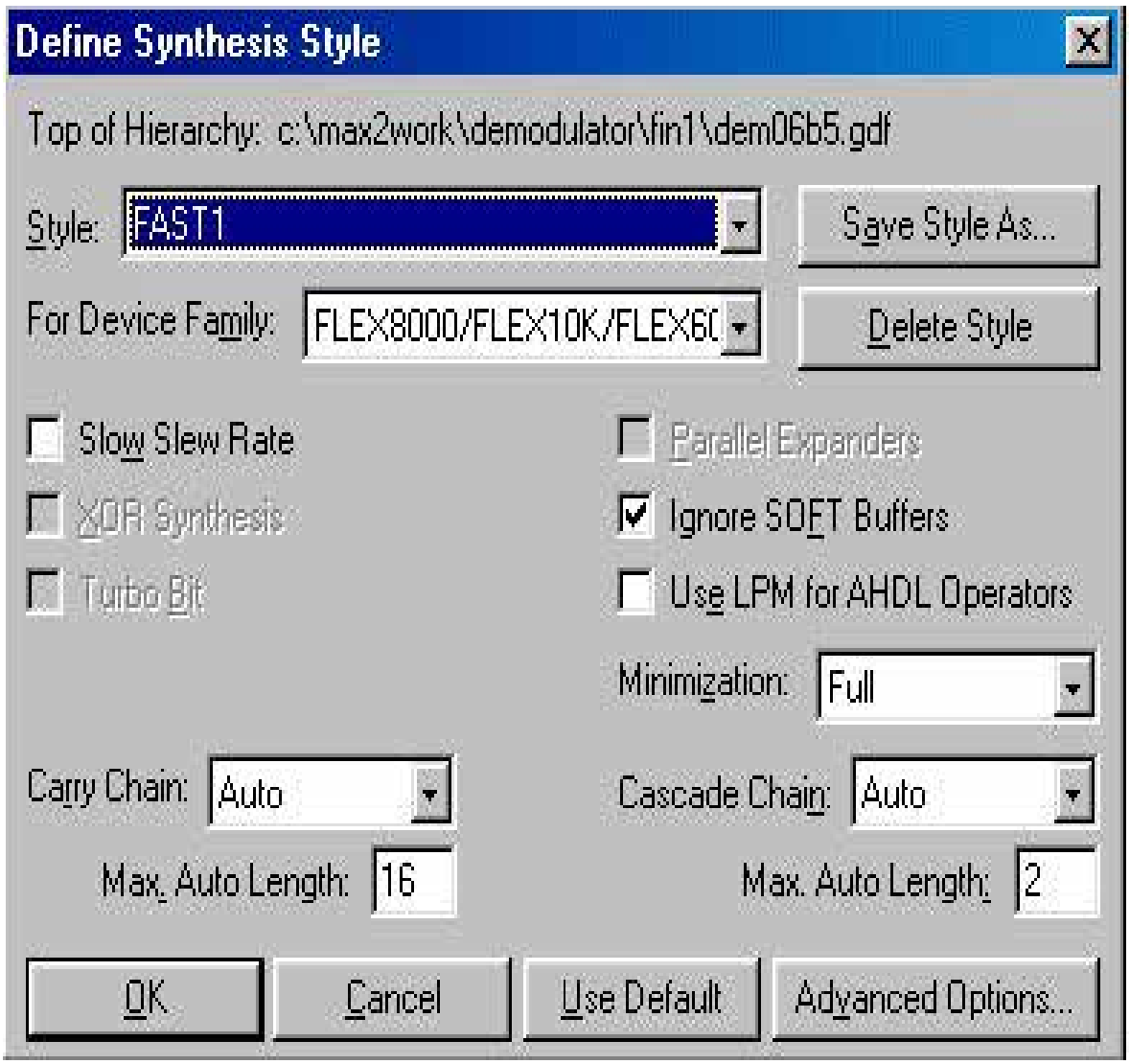


Рис. 2.21. Визначення стилю синтезу

Як видно з рис. 2.21 можна вибрати назву стилю, спосіб реалізації та максимальну довжину ланцюжків передачі та каскаду, ступінь мінімізації логічних функцій та інші параметри синтезу.

Розширена кнопка Параметри дозволяють вибрати параметри синтезу в діалоговому вікні, показаному на малюнку 2.22

Рис.2.22. Розширене вікно Опції .

2.7. Редактори MAX PLUS II

Усі п’ять редакторів MAX PLUS II і три редактори для створення файлів проекту (графіки, тексту та сигналу) мають спільні функції, такі як збереження та виклик файлу. Крім того, програми MAX PLUS II Editor мають такі спільні функції: Створення файлів символів і файлів прототипів функцій:

пошук вузлів (node розташування) ;

проходження ієрархічного дерева (ієрархія перехід) ;

контекстне меню меню команди) ; аналіз часу (Тимінг Аналіз) ;

пошук і заміна фрагментів тексту (Find & Replace Текст) ;

скасування останнього кроку редагування, повернення до нього, вирізання, копіювання, вставлення та видалення виділених фрагментів, обмін фрагментами між програмами MAX PLUS II.

На рис. На малюнку 2.23 показано вікно графічного редактора (Graphic Editor) MAX PLUS II, який забезпечує проект у форматі реального зображення (WYSIWIG). Ви можете створювати нові файли (команда New з меню File). Доступ до графічного редактора здійснюється з меню MAX PLUS II .

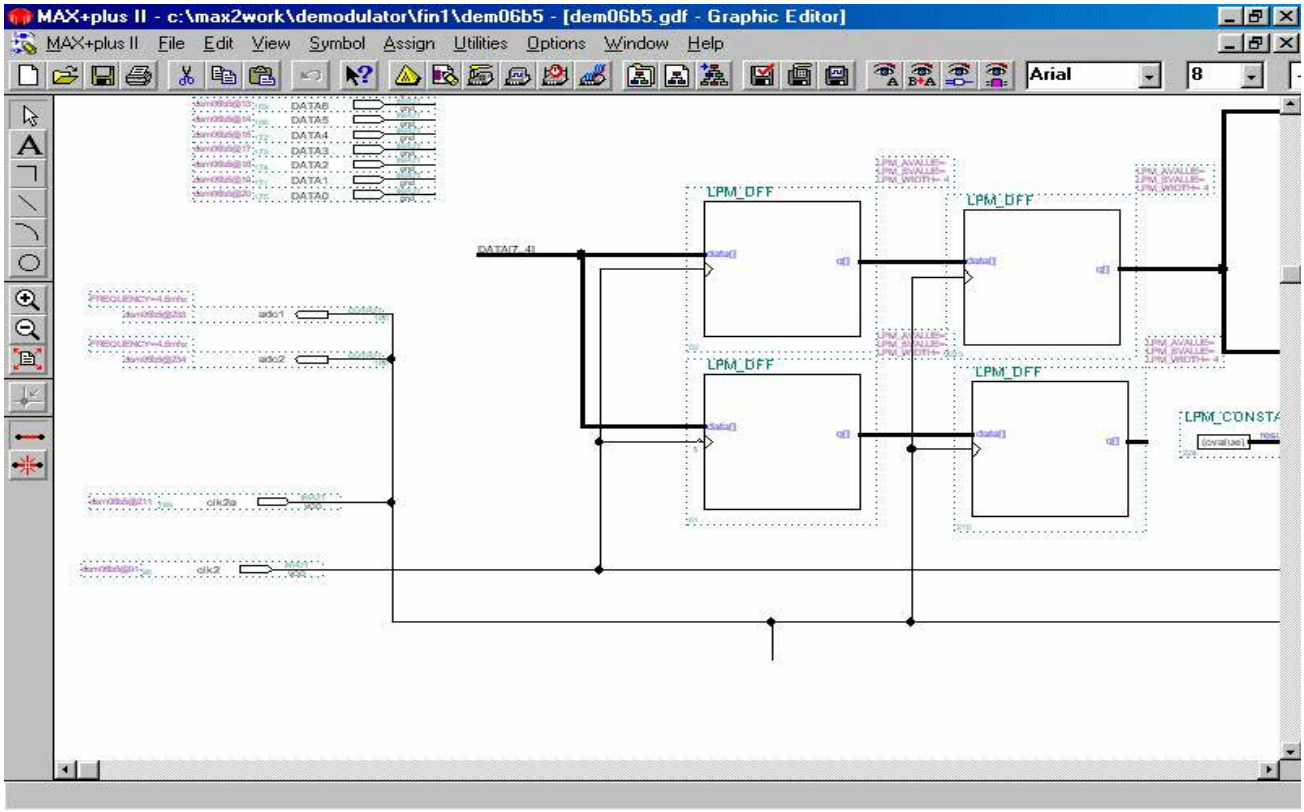


Рис. 2.23. Графічний редактор MAX PLUS II

Файли проектної графіки (.gdf) або файли схем OrCAD (.sch), створені в цьому графічному редакторі , можуть містити будь-яку комбінацію примітивних символів, мегафункцій і макрофункцій. Символи можуть бути файлами проекту будь -якого типу (.gdf. tdf.vhd..edf.xnf..smf) .

Багатоваріантність графічного редактора характеризується такими особливостями:

інструмент вибору («стрілка») полегшує розробку проекту. Він дозволяє переміщувати та дублювати об’єкти, а також створбвати нові символи. Коли ви розміщуєте його на шпильці або на кінці лінії, він автоматично перетворюється на інструмент для малювання ортогональних ліній. Якщо ви клацнете на тексті, він буде автоматично перетворено в інструмент редагування тексту;

символи з’єднані сигнальними лініями, які називаються вузлами або шинними лініями , які являють собою кілька логічно згрупованих вузлів. Давши вузлу ім’я, ви можете пов’язати його з іншими вузлами чи символами

лише за назвою. Автобуси пов’язані за назвою, але також можуть бути пов’язані графічно;

користувач може замінити порти, які використовуються в кожному окремому прикладі символу мега- чи макрофункції, а також змінити їх. У той же час з'явиться інверсійне коло, що вказує на перевернутий порт;

можна вибрати кілька об’єктів у прямокутній області та редагувати їх разом або окремо. Коли ви переміщуєте виділену область, сигнальні з’єднання збережуться;

для кожного символу ви можете переглядати призначення датчиків, штифти, розташування, чіпи, клацання, час, локальну маршрутизацію, логічні параметри та призначення параметрів. Щоб спростити тестування, ви також можете створити групи контактів, які визначають підключення зовнішніх пристроїв між контактами;

altera , мега- та макро-функції скорочують час розробки проекту. Користувач також може створювати власні бібліотеки функцій. Коли ви редагуєте символ або відновлюєте його параметри за замовчуванням, ви можете автоматично створити вибрані схеми або всі можливі варіанти цього символу у графічному редакторі.

Графічний редактор надає багато інших можливостей. Наприклад, ви можете збільшити або зменшити масштаб, щоб побачити весь проект або будь-які його деталі. Ви можете вибрати гарнітуру та розмір шрифту, встановити стилі ліній, а також встановити та відобразити напрямні. Можна копіювати, вирізати, вставляти та видаляти виділені фрагменти; отримати дзеркальне відображення (вертикально або горизонтально; повернути виділені фрагменти на задану кількість градусів; задати розмір і положення поточного аркуша діаграми по вертикалі або горизонталі).

На рис. На малюнку 2.24 показано вікно редактора символів системи MAX PLUS II, за допомогою якого можна переглядати, створювати та редагувати символ, який є логічною схемою. Ви можете створювати нові файли (команда New з меню File). Редактор символів викликається з меню MAX PLUS II .

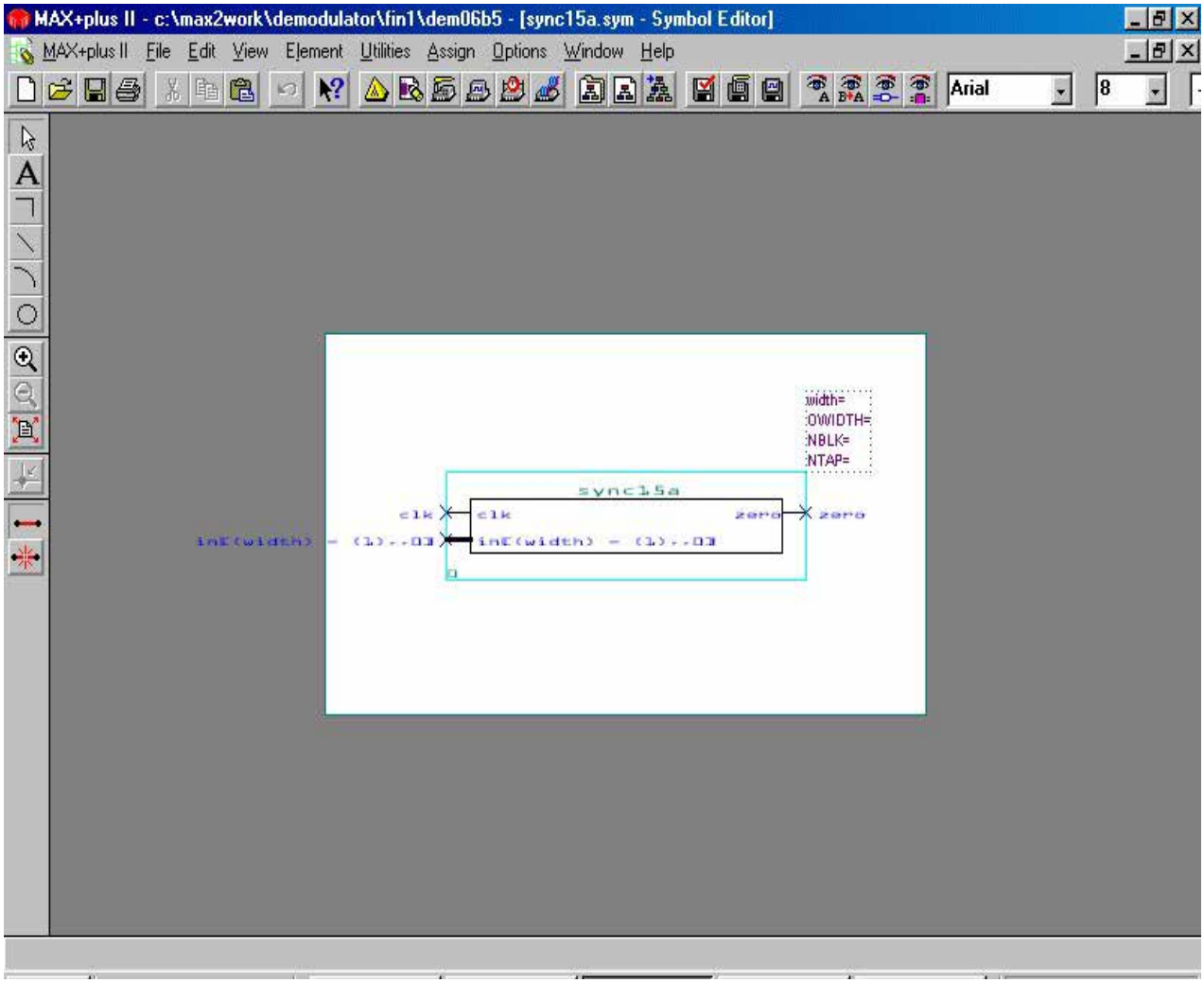


Рис. 2.24. Редактор символів MAX PLUS II

Символьний файл має таке ж ім'я, як і файл проекту, і розширення . sim . Створіть команду За замовчуванням Меню Suymbol File , доступне в графічному, текстовому та сигнальному прикладенні, створює пристрій для любого файлу проекту. Редактор символів має такі функції:

- ви можете замінити символ, що представляє файл проекту;
- ви можете створювати та редагувати піни та їх назви, розширювати вхідні, вихідні та двонаправлені піни, а також налаштовувати параметри введення символу у файл графічного редактора: з відображенням імен пінів на екрані або без нього, з відображенням повної чи скороченої назви . Тому повне ім'я порту та ім'я, яке відображається у файлі у вікні графічного редактора, можуть відрізнятися;

- назви пінів автоматично дублюються через межі символів. Редагувати можна лише імена всередині символу. Назви поза символом не можна редагувати, вони просто ілюструють, як з'єднані контакти;
- можна встановити значення параметрів і їх значення за замовчуванням;
- сітка та напрямні допомагають точно вирівняти об'єкти;
- До символу можна внести коментарі або корисні примітки, які також з'являться після введення символу у файл у графічному редакторі.

На рис. На малюнку 2.25 показано вікно текстового редактора MAX PLUS II , який є гнучким інструментом для створення файлів текстових проектів мовами опису обладнання: AHDL, VHDL, Verilog HDL. У цьому текстовому редакторі також можна працювати з будь-яким файлом ASCII. Ви можете створювати нові файли (команда New з меню File). Редактор символів викликається з меню MAX PLUS II .

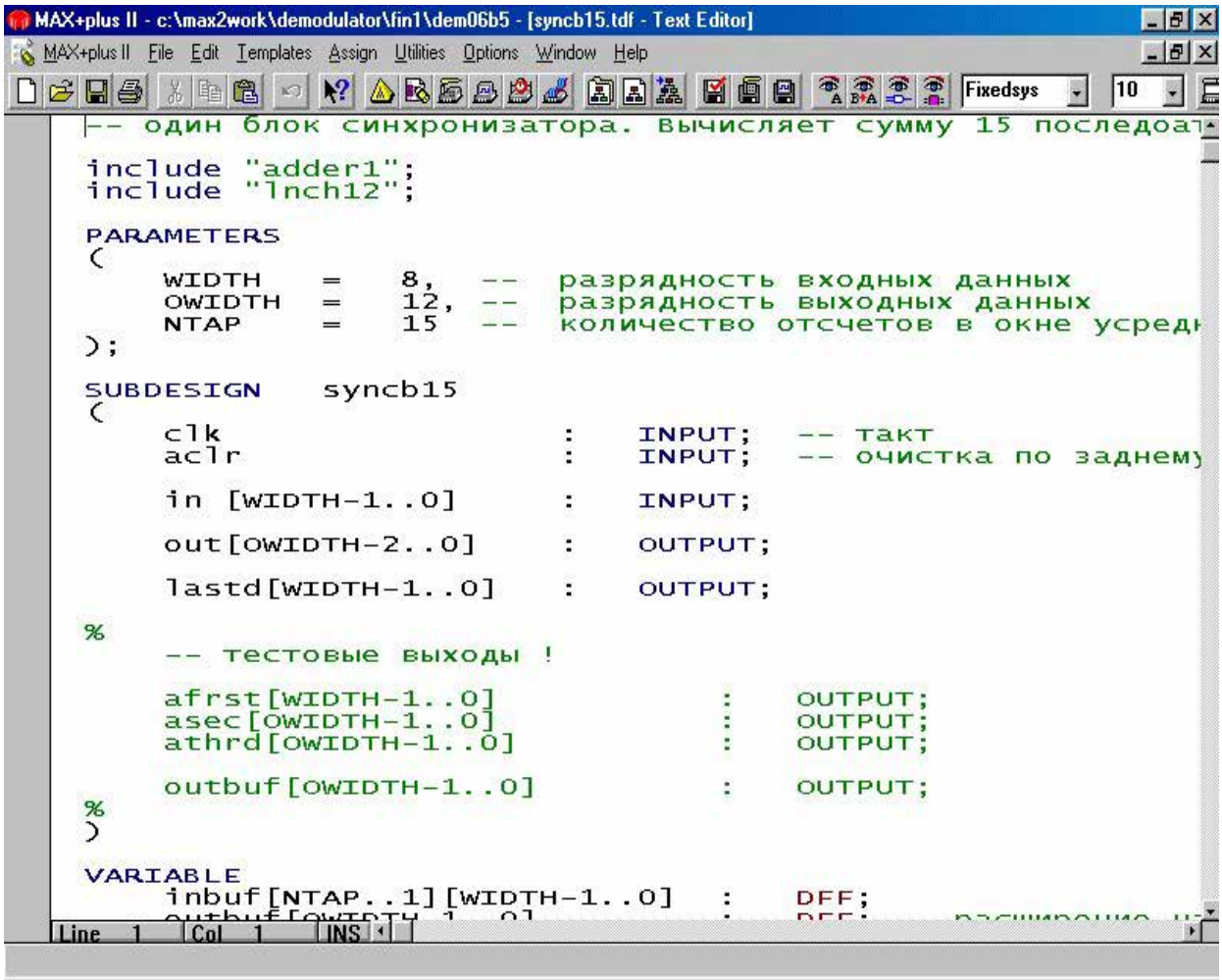


Рис. 2.25. Текстовий редактор MAX PLUS II

Усі перелічені файли проекту можна створити в будь-якому текстовому редакторі, але цей редактор має вбудовані можливості для зручного введення файлів проекту, їх компіляції та налагодження, відображення повідомлень про помилки та їх розташування у вихідному тексті або в тексті допоміжних файлів; Крім того, доступні приклади мовних конструкцій для мов програмування, а також виконано розфарбування синтаксичних конструкцій. У цьому редакторі ви можете вручну редагувати файли призначення та конфігурації (.acf), а також вводити параметри конфігурації для компілятора, симулятора та аналізатора часу .

За допомогою цього текстового редактора ви можете створювати тестові вектори (.vec), які використовуються для тестування, функцій налагодження та під час введення дизайну сигналу . Ви також можете створювати пакетні файли (.cmd для симулятора та .edc – для EDIF), а також бібліотеку макросів (.lmf) .

Текстовий редактор MAX PLUS II надає контекстно-залежну довідку.

Редактор сигналів (Waveform Редактор (рис. 2.26) виконує дві ролі: він служить інструментом для створення проекту та інструментом для створення тестових завдань і аналізу результатів тестування. Розробник може створювати файли проектних сигналів (.wdf), які містять логіку проектування проекту, а також файли тестового каналу (.scf), які містять вхідні вектори для функціонального тестування та налагодження . Новий файл створюється за допомогою команди «Новий» у меню «Файл» . Доступ до редактора сигналів здійснюється з меню MAX PLUS II .

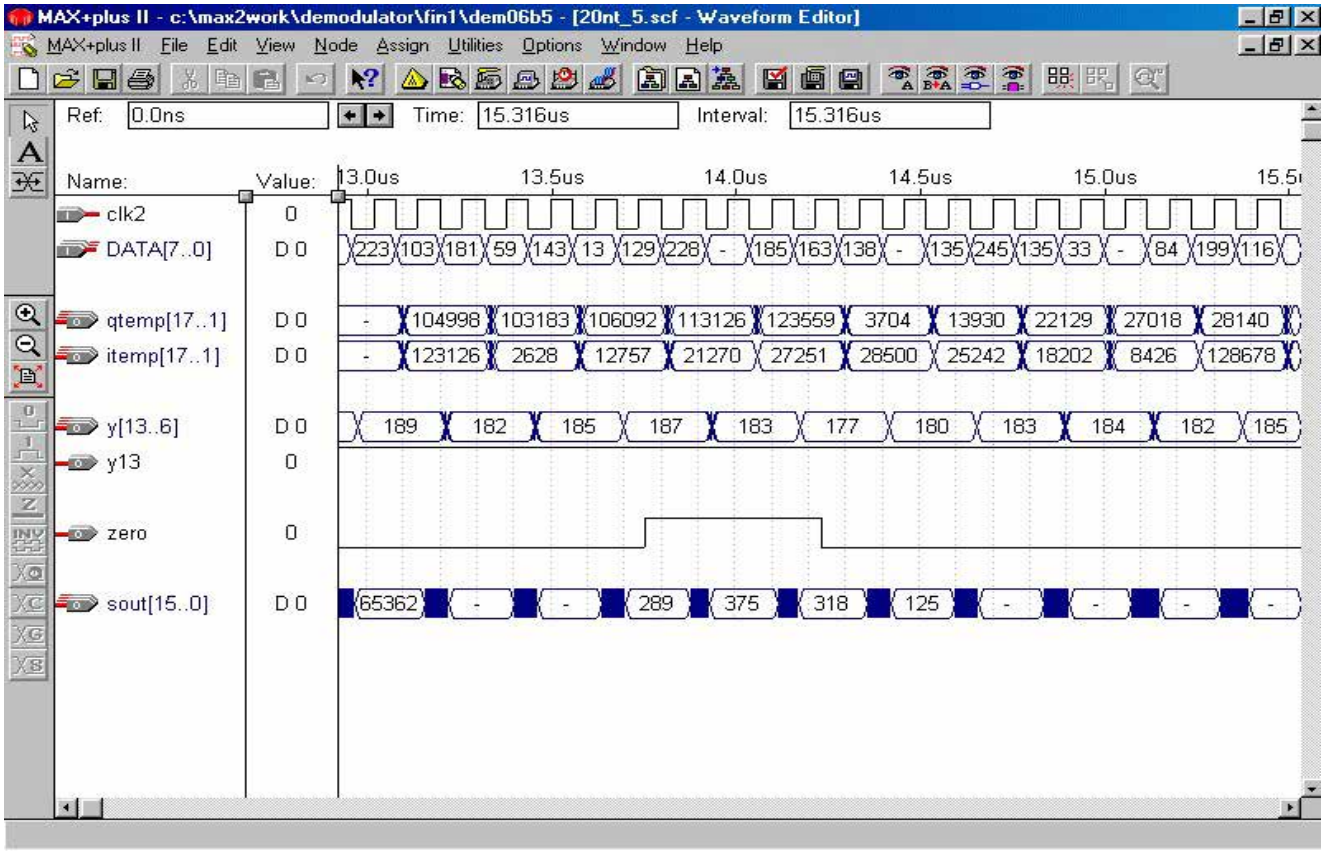


Рис. 2.26. Редактор сигналів MAX PLUS II

Розробка проекту в редакторі сигналів є альтернативою створенню проекту в графічних або текстових редакторах. Тут ви можете графічно вказати комбінації вхідних логічних рівнів і необхідних виходів. Отриманий файл має розширення WDF (Waveform дизайн файл) може містити як логічні входи, так і входи цифрової машини, а також комбінаторну логіку, лічильник і виходи цифрової машини. Ви також можете використовувати «приховані» вузли , щоб допомогти визначити необхідні результати.

Метод розробки проекту за допомогою редактора сигналів краще підходить для схем із чітко визначеними послідовними входами та виходами, тобто цифрових машин, лічильників та регістрів.

Використовуючи редактор сигналів, ви можете легко трансформувати сигнали цілком або частково, створюючи та редагуючи вузли та групи. За допомогою простих команд ви можете створити файл таблиці символів ASCII (.tbl) або імпортувати тестовий векторний файл ASCII (.vec) , щоб створити тестовий канал SCF і файли дизайну сигналу WDF . Ви також можете зберегти

файл WDF як SCF для тестування або конвертувати SCF у WDF для використання як файл проекту.

Редактор сигналів має наступні особливості:

- ви можете створити або відредагувати вузол, щоб отримати тип вводу/виводу (введення/виведення), який представляє вхідний або вихідний штифт або блочну логіку;
- при розробці WDF ви можете вказати тип логіки, яка робить кожен вузол контактом і входом, регістром, комбінаторним або цифровим автоматом;
- також можна вказати значення за замовчуванням для активного логічного рівня в логічному вузлі: високий (1), невизначений (X) або високий імпеданс (Z), а також назву стану за замовчуванням у типі цифрової машини вузол;
- файл інформації симулятора (.snf) у файл тестового каналу SCF, який існує для повністю скомпільованого та оптимізованого дизайну;
- ви можете підключити від 2 до 256 вузлів, щоб створити нову групу (шину) або розгрупувати вузли, раніше підключені до групи. Ви також можете об'єднувати групи з іншими групами. Значення групи може відображатися у двійковій, десятковій, шістнадцятковій або вісімковій системі з перетворенням коду Грея або без нього;
- можна копіювати, вставляти, переміщувати або видаляти вибрану частину («інтервал») сигналу або всю форму сигналу, а також весь вузол або групу (тобто назву вузла або групи плюс форму сигналу). Ви можете редагувати кілька інтервалів, цілі прогони, цілі вузли та групи за одну операцію. Копії цілих вузлів і груп пов’язані між собою, так що редакційні зміни в одній копії відображаються в усіх копіях. Ви також можете ревертувати, вставляти, перезаписувати, копіювати, розширювати або звужувати інтервал сигналу довільної довжини, з будь-яким логічним рівнем, годинником, послідовністю підрахунку або назвою стану;
- можна визначити та додатково відобразити сітку для вирівнювання переходів між логічними рівнями до або після їх створення ;

- можна вводити коментарі між проходами будь-де у файлі;
- можна змінити масштаб відображення;
- щоб показати різницю між тестовими виходами та виходами реального пристрою, ви можете накласти будь-який із виходів у поточному файлі або накласти другий файл редактора сигналів, щоб порівняти форми сигналів його вузлів і груп із відповідними в поточному файлі. файл .

Для налагодження пристроїв DSP часто необхідно протестувати алгоритм на реальних або змодельованих сигналах. Для цього зручно використовувати векторний сигнальний файл (Vector Файл)

Вектор Файл (текстовий формат ASCII) використовується для визначення вхідних умов моделювання та вузлів для моделювання. Вектор Файл також можна використовувати для створення хвилі Дизайн Вхідний файл проекту. Давайте детальніше розглянемо векторний формат файлу.

Усі розділи, які використовуються у Vector Файл обговорюються нижче в тому порядку, в якому вони зазвичай відображаються у файлі. Ви можете повторити будь-який розділ, щоб додати в нього додаткові умови введення

Unit Section

ключовим словом UNIT , а потім визначає одиниці вимірювання у файлі. Параметр необов'язковий. Типовими одиницями є нс . Можливі одиниці вимірювання: нс , мс, мкс , с, мГц. Розділ закінчується на ; .

Приклад: UNIT ms ;

Start Section

Починається ключовим словом START , за яким слід початкове тимчасове значення. Параметр необов'язковий. Значення за замовчуванням дорівнює нулю. Якщо одиниці вимірювання не вказані, вони беруться з розділу Одиниця Розділ . Розділ закінчується на ; .

Приклад: START 5ns;

Stop Section

Подібно до розділу «Пуск» . Розділ . За замовчуванням передбачається значення часу останнього вектора моделі.

Приклад: STOP 150ms;

Будь ласка, візьміть до уваги, що Вектор Файл має містити кратне значення Start-Stop Розділ, що представляє часові інтервали. Неприпустимо відносити до одного і того ж періоду часу різні модельні вектори.

Interval Section

ключове слово INTERVAL, за яким слідує тимчасове значення. Визначає інтервал часу введення вектора. Параметр необов'язковий. Стандартне значення становить 1 нс . Розділ закінчується на ; . *Приклад:* INTERVAL 15ns;

Group Create Section

ключове слово GROUP CREATE . Цей розділ не завжди потрібен для груп, шин або кінцевих автоматів, створених у вихідних файлах проекту. Усі вузли в групі мають бути типу введення/виведення. У векторі Файл , який використовується для моделювання, вузли повинні бути названі відповідно до імен вузлів, введених у файл проекту, включаючи ієрархічний шлях, якщо необхідно. Розділ закінчується на ; .

Приклад: GROUP CREATE groupABC = nodeA вузол Б nodeC ;

Radix Section

ключового слова RADIX , за яким слідує позначення цифрової системи. Параметр необов'язковий. За замовчуванням використовується шістнадцяткова система числення . Є чотири системи: BIN (двійкова), DEC (десяткова), HEX (шістнадцяткова), OCT (вісімкова). Розділ закінчується на ;

Приклад: RADIX DEC;

Inputs Section

ключове слово INPUTS . Нижче наведено список імен хостів та/або груп. У векторі Файл, який використовується для моделювання. Імена вузлів мають збігатися з іменами вузлів у файлі проекту, включаючи ієрархічний шлях, якщо необхідно.

Приклад: шина даних INPUT clk OEN nodeA ;

Розділ закінчується на ; .

У прикладі нижче, коли використовується наступний розділ введення, значення з попереднього розділу будуть втрачені.

Приклад : INPUTS A1 A2;

ПОЧАТОК 0 ;

Przystań 25;

ШАБЛОН % Модель 1 секції з входами A1 і A2%

0 0 0

0 0 1;

СТАРТ 26;

СТОП 50;

ШАБЛОН %Декція моделі 2 із входами A1 і A2%

0 10

1 0 0;

ВХОДИ A1 B1;

СТАРТ 51;

ТРИМАТИ 100;

ШАБЛОН % Модель 3 секції з входами A1 і B1%

1 1 0

0 1 1;

Outputs Section

ключове слово OUTPUTS . Використовується для опису результатів. Аналогія з входами Розділ .

Приклад: RCO OUTPUTS QA QB QC;

Приклад : ВИХОДИ A1 A2;

ПОЧАТОК 0 ;

Przystań 25;

ШАБЛОН %Модель 1 секція з виходами A1 і A2%

0 0 0

0 0 1;

СТАРТ 26;

СТОП 50;

ШАБЛОН %Секція моделі 2 з виходами A1 і A2%

0 10

1 0 0;

ВИХОДИ A1 B1;

СТАРТ 51;

ТРИМАТИ 100;

ШАБЛОН %Секція моделі 3 з виходами A1 і B1%

1 1 0

0 1 1;

Buried Section

Починається ключовим словом POBURIED . Використовується для опису вузлів. Аналогія з входами Розділ .

Приклад : BURIED nodeQA0 nodeQA1 nodeQB0 nodeQB1;

Pattern Section

Він починається з ключового слова PATTERN. У цьому розділі використовуються дані з попередніх розділів Inputs , Outputs і Buried Розділ . Додавання нових входів , виходів і похованих Розділ видаляє всі попередні дані цього типу, тобто старі дані замінюються новими (див. приклад для введення даних Розділ). Розділ закінчується на ; .

Вектор Файл, який використовується для створення файлу WDF, може містити додаткові розділи Комбінаторний Секція , машина Розділ , зареєстрований Розділ . Ці розділи ігноруються, якщо Vector Файл використовується для моделювання.

Розглянемо приклад створення вектора Файл для восьмирозрядного суматора. Опис на AHDL

sum8_.tdf

ПІДПРОЕКТ SUM8_

% 8-бітне беззнакове додавання %

(

clk:INPUT;

a[7..0]:INPUT = GND;

```
        b[7..0]    :       INPUT;
                sum[7..0]       :  OUTPUT;
                 cr                 :                              OUTPUT; )
        VARIABLE
        cr.prn=VCC;
        cr.clrn=VCC;
        sum[7..0]:DFF;
        c[7..1]:NODE;
                sum[7..0].clk=clk;
                sum[7..0].prn=VCC;
                sum[7..0].clrn=VCC;
        cr:DFF; BEGIN
        cr.clk=clk;
If (a[0]&b[0])==VCC then
                c[1]=VCC;
        sum[0]=GND;
        ElsIf (a[0]#b[0])==GND then
                sum[0]=GND;
                c[1]=GND; c[1]=GND;
        Else sum[0]=VCC;
        End If;
        FOR i IN 1 TO 6 GENERATE If (a[i]&b[i]&c[i])==VCC then
                c[i+1]=VCC;
                sum[i]=VCC;
                ElsIf (a[i]#b[i]#c[i])==GND then
                c[i+1]=GND;
                sum[i]=GND;
        ElsIf (((a[i]&b[i])#(a[i]&c[i])#(b[i]&c[i]))&!(a[i]&b[i]&c[i]))==VCC then
                c[i+1]=VCC; Else sum[i]=VCC;
                c[i+1]=GND; End If;
                sum[i]=GND;
```

```
        END GENERATE;
    If (a[7]&b[7]&c[7])==VCC then
            cr=VCC;
            sum[7]=VCC;
    ElsIf (a[7]#b[7]#c[7])==GND then
            cr=GND;
        sum[7]=GND;
    ElsIf (((a[7]&b[7])#(a[7]&c[7])#(b[7]&c[7]))&!(a[7]&b[7]&c[7]))==VCC
then
        cr=VCC; Else sum[7]=VCC;
        sum[7]=GND;
        cr=GND; End If;
    END;
```

У цьому прикладі буде додано 6 пар двійкових чисел. У цьому випадку синхронізований меандр clk (тактовий імпульс для D-тригерів) буде встановлено з періодом повторення 50 нс і шпаруватістю 2. Початкове значення імпульсу буде 0. Буде виконано перетворення від 0 нс до 500 нс.

Під час запуску симуляції необхідно, щоб відповідний файл sum8_.scf не знаходився в робочому каталозі, інакше симулятор за умовчанням використовуватиме файл, відмінний від . vec і файл із розширенням . scf . Після процесу моделювання за допомогою . vec — файл із розширенням . scf буде створено автоматично.

На рис. На малюнку 2.27 показано вікно планувальника рівнів (FloorPlan Редактор) , за допомогою якого користувач розподіляє ресурси фізичним пристроям і переглядає результати розгалуження та сплайсингу, які виконує компілятор. Вікно плану поверху відкривається після вибору параметра План поверху Редактор в меню MAX PLUS II .

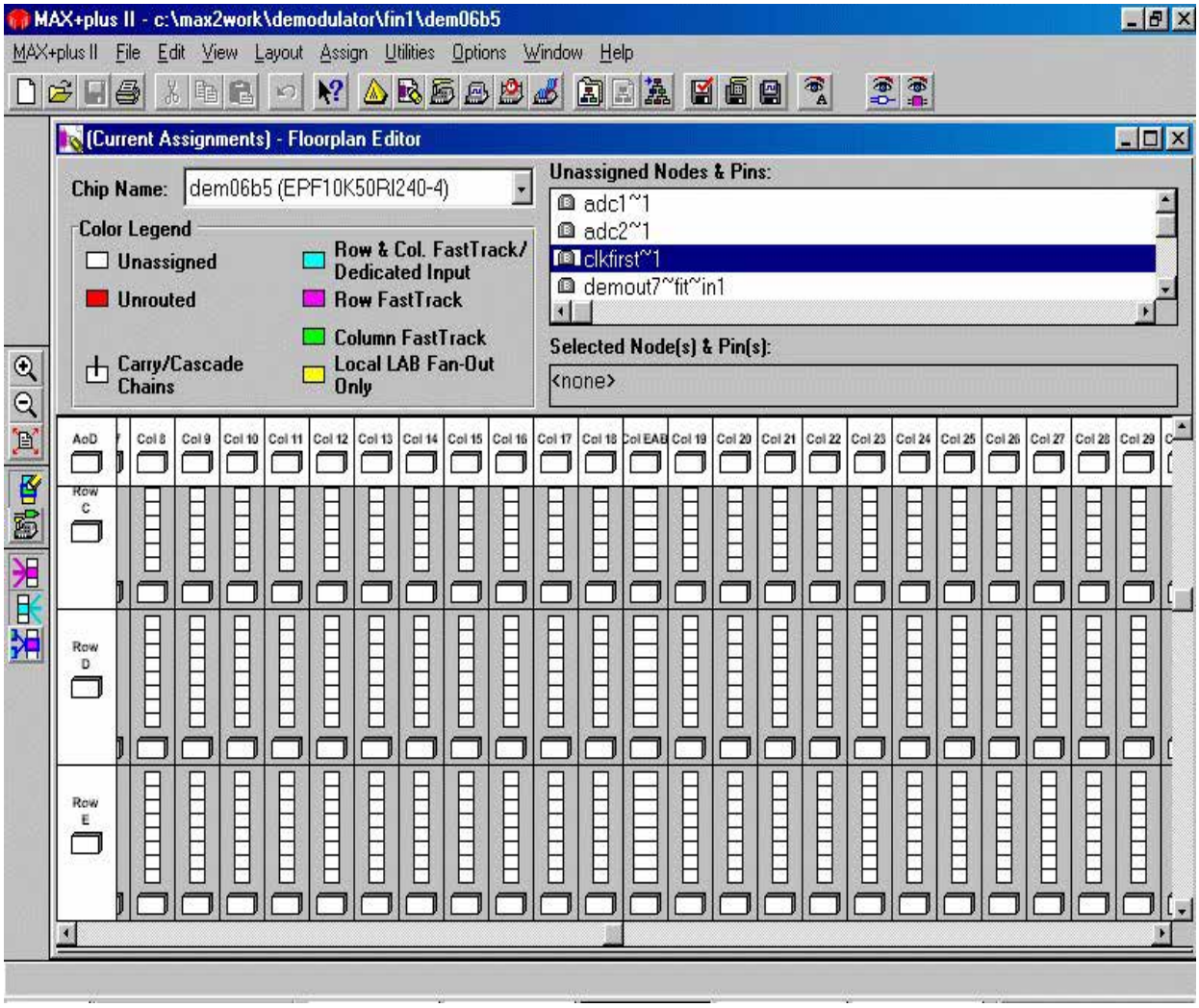


Рис. 2.27. Планувальник рівнів MAX PLUS II У вікні планувальника рівнів можуть бути представлені два типи

2.8. Процес побудови проекту

По-перше, компілятор витягує інформацію про ієрархічні зв’язки між файлами проекту та перевіряє проект на наявність простих помилок введення проекту. Він створює організаційну карту проекту, а потім об’єднує всі файли проекту в неієрархічну базу даних, яку може ефективно обробляти.

Компілятор використовує різноманітні прийоми для підвищення ефективності проектування та зменшенню використання можливостей пристрою. Якщо пристрій дуже великий для реалізації на одній FPGA, компілятор може автоматично розділити його для реалізації на кількох пристроях в одному сімействі FPGA, мінімізуючи кількість з’єднань між

пристроями. У файлі звіту (.rpt) буде показано, як проект буде реалізовано на одному чи кількох пристроях.

Хоча компілятор може автоматично скомпілювати проект, можна вказати обробку проекту відповідно до точних інструкцій програміста. Наприклад, можна встановити стандартний стиль синтезу логіки проекту та інші параметри синтезу логіки по всьому проекту. Крім того, зручно визначити часові вимоги для всього проекту, точно визначити поділ великого проекту на частини, які будуть реалізовані на кількох пристроях, вибрати параметри пристроїв, які стосуватимуться всього проекту в цілому. Ви також можете вибрати, скільки шпильок і воріт залишити невикористаними під час поточної збірки, щоб зарезервувати їх для подальших змін дизайну.

Ви можете почати компіляцію з будь-якої програми MAX PLUS II або вікна компілятора. Компілятор автоматично обробляє всі вхідні файли поточного проекту. Процес компіляції можна спостерігати у вікні компілятора у такому вигляді (рис. 2.28):

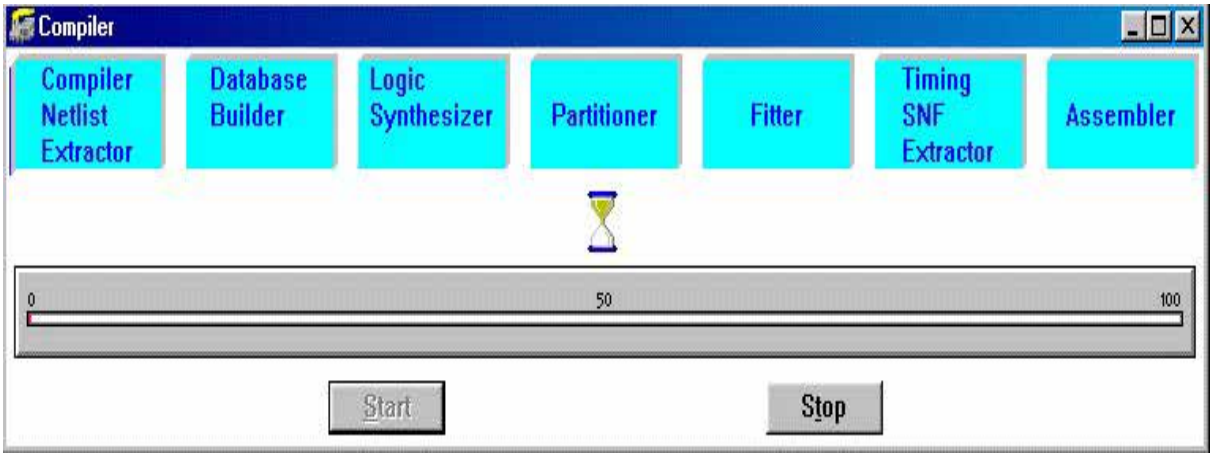


Рис. 2.28. Процес побудови

- прямокутники модуля компілятора підсвічуються один за одним, коли компілятор завершує кожен крок обробки;
- піктограма вихідного файлу, створеного цим модулем, з’являється під прямокутником модуля компілятора. Щоб відкрити відповідний файл , двічі клацніть лівою кнопкою миші на значку, і він відкриється;
- швидкість завершення збірки поступово зростає (до 100%), що також відображається на зростаючому прямокутнику

«термометр»;

- зупинка компілятора змінюється на кнопку Зупинити / Показати Статус , який ви можете вибрати, щоб відкрити діалогове вікно, що відображає поточний стан розділення та редагування проекту;
- у разі виявлення будь-яких помилок або потенційних проблем під час процесу збирання автоматично відкривається вікно обробки повідомлень, у якому відображається список повідомлень про помилки, попереджень та інформаційних повідомлень, а також негайна допомога щодо виправлення помилки. Крім того, ви можете визначити джерела новин у файлах проекту або в його плані призначення шарів .

Компілятор може працювати у фоновому режимі. Ви можете згорнути вікно компілятора під час обробки вашого проекту та продовжити роботу над іншими файлами проекту. Зростаючий прямокутний «термометр» під зменшеною піктограмою вікна компілятора дозволяє вам відстежувати хід процесу збирання, поки ви зосереджуєтесь на іншому завданні. Однак пам’ятайте, що розкіш багатозадачності можлива лише на пристойній машині. Якщо у вас мало оперативної пам'яті, краще випити чашку чаю, поки триває процес компіляції.

Системний компілятор MAX PLUS II обробляє проект за допомогою наступних модулів та інструментів:

Екстрактор містить вбудовані EDIF, VHD;

Logic Synthesizer;

Partitioner;

Verilog і XNF;

Database Builder;

Fitter;

Assembler;

Design Doctor Utility.

Linked SNF Extractor;

EDIF Netlist Writer;

Functional SNF Extractor;

Verilog Netlist Writer;

VHDL Netlist Writer VHDL;

Timing SNF Extractor;

Модуль компілятора Список мереж Extractor перетворює кожен файл проекту в один або кілька двійкових файлів із розширенням cnf. Оскільки компілятор перезаписує значення всіх параметрів, які використовуються в параметризованих функціях, вміст файлу CNF може змінитися під час послідовної компіляції, якщо зміниться значення параметрів. Цей модуль також створює файл ієрархічних зв'язків (.hif) (hierarchy підключитися до файл) , який документує ієрархічні зв'язки між файлами проекту, а також містить інформацію, необхідну для відображення ієрархічного дерева проекту у вікні ієрархії Дисплей . Крім того, цей модуль створює файл бази даних вузла (.ndb) (node база даних) , яка містить імена вузлів проекту для бази даних призначення ресурсів.

Вбудовані зчитувачі формату EDIF, VHDL, Verilog і XNF автоматично переводять інформацію про проект у відповідні формати файлів . edf , . vhd , .v , . xnf у формат, сумісний із системою MAX PLUS II. Зчитувач EDIF обробляє вхідні файли EDIF за допомогою бібліотечних файлів . lmf , (бібліотека відображення файл) , які встановлюють сумісність логічних функцій, розроблених в інших системах САПР, із функціями системи MAX PLUS II. Програма для читання XNF може створити файл експорту текстового дизайну (.tdx) (текст дизайн експорт файл) , який містить інформацію AHDL, еквівалентну інформації, що міститься у файлі XNF (. xnf); це робиться для редагування проекту в AHDL.

Модуль бази даних Конструктор (розробник бази даних) використовує файл HIF для створення створених компілятором файлів CNF, що містять опис проекту. Базуючись на ієрархічній структурі проекту, цей модуль копіює кожен файл CNF в єдину базу даних без ієрархічної структури. Таким чином база даних підтримує електричний зв'язок проекту.

Під час створення бази даних модуль перевіряє логічну завершеність і узгодженість дизайну, а також перевіряє граничну узгодженість і синтаксичні

помилки (наприклад, вузол без джерела або призначення). Більшість помилок виявляється на цій стадії компіляції, і їх можна легко та негайно виправити. Кожен модуль компілятора послідовно обробляє та оновлює цю базу даних.

Коли компілятор обробляє проект вперше, компілюються всі файли проекту. Ви можете скористатися функцією «швидкої перекомпіляції» (smart recompile) , щоб створити розширену базу даних проекту, яка допоможе прискорити наступні збірки. Ця база даних дозволяє перепризначати фізичні ресурси пристрою, такі як призначення контактів і воріт, і перекомпілювати проект без перебудови бази даних і повторного синтезу логіки проектування. Використання функції «повної перекомпіляції». Можна вибрати між перекомпіляцією лише тих файлів, які були відредаговані після останньої компіляції, або повною перекомпіляцією всього проекту.

Модуль логічного синтезатора (Logic Синтезатор використовує низку алгоритмів, які зменшують споживання ресурсів і усувають надлишкову логіку, таким чином забезпечуючи найбільш ефективне використання структури вентиля для цільової архітектури сімейства пристроїв . Цей модуль компілятора також застосовує методи логічного синтезу для вимог користувача щодо параметрів синхронізації тощо. Крім того, логічний синтезатор шукає логіку для непідключених вузлів. Якщо він знаходить неприєднаний вузол, він видаляє примітиви, пов’язані з цим вузлом.

Для керування логічним синтезом доступні три попередньо визначені стилі та велика кількість логічних опцій.

У будь-якій програмі MAX PLUS II ви можете ввести значення часу проектування та вибрати параметри логіки, а також визначити стилі синтезу логіки. Ви можете встановити глобальний логічний синтез за замовчуванням і параметри синхронізації проекту для всього проекту в цілому, а також усі додаткові параметри логіки та призначення параметрів синхронізації проекту для окремих логічних функцій.

Якщо проект не підходить при установці на одному пристрої, Partitioner (СТІЙ) ділить базу даних, оновлену логічним синтезатором, на кілька ПЛІС одного сімейства, одночасно намагаючись розділити дизайн на мінімально

можливу кількість пристроїв. Структура розділена по межах логічних елементів, а кількість виводів, які використовуються для зв'язку між пристроями, зведено до мінімуму.

Поділ може виконуватися повністю автоматично, під частковим або повністю контролем користувача. Призначення пристроїв і налаштування автоматичного вибору пристроїв дозволяють застосувати рівень контролю, який найкраще відповідає вашому проекту.

Поки запущені модулі Partitioner і Fitter , ви можете призупинити збірку. Компілятор відобразить інформацію про поточний стан процесів розділення та маршрутизації, включаючи порівняння необхідних і доступних ресурсів. Це необхідно для того, щоб прийняти рішення про продовження будівництва або внесення фундаментальних змін в проект.

Використовуючи базу даних, оновлену секціонером, програма налаштування узгоджує вимоги до дизайну з відомими ресурсами одного чи кількох пристроїв. Він призначає кожній логічній функції розташування логічного вентиля, який її реалізує, і вибирає відповідні шляхи з’єднання та призначення контактів. Цей модуль намагається узгодити розподіл ресурсів, тобто контакти, логічні вентилі, вентилі введення/виведення, комірки пам’яті, мікросхеми, клацання, пристрої, локальну маршрутизацію , синхронізацію та призначення контактів із файлу призначення та конфігурації (.acf) . файл) з доступними ресурсами.

Модуль має параметри, які дозволяють визначати методи маршрутизації, наприклад, автоматичне введення логічних елементів або обмеження коефіцієнта з’єднання входу. Якщо трасування не вдається виконати, модуль надсилає повідомлення та дає вам можливість проігнорувати деякі або всі призначення або зупинити компіляцію.

Незалежно від того, було завершено повне відстеження проекту, цей модуль генерує файл звіту (.rpt) (звіт файл) , який документує інформацію про розділення проекту, імена вхідних і вихідних контактів, таймінги проекту та невикористані ресурси для кожного пристрою в проекті. Ви можете

включити у свій файл звіту розділи, що показують призначення користувачам, ієрархію файлів, з'єднання логічних воріт і рівняння, реалізовані в LE.

Компілятор автоматично створює файл трасування (.fit), який документує призначення ресурсів і пристроїв у всьому проекті, а також інформацію трасування . Незалежно від успішності трасування користувач може переглянути інформацію про узгодження, розділення та відстеження з файлу узгодження у вікні Планувальника шарів . Також можна переписати призначення з файлу узгодження до призначень ACF і файлу конфігурації для подальшого редагування.

Можна вказати трекеру (Fitter) генерувати вихідні текстові файли проекту у форматі AHDL (.tdo). Оскільки проект із кількома пристроями створює один файл на пристрій, вам слід розділити проект на кілька проектів, кожен для одного пристрою . Якщо ви хочете захопити логіку на деяких пристроях, збережіть файл TDO для цього пристрою як текстовий файл проекту (.tdf) і перекомпілюйте логіку для цього пристрою, зберігаючи результати синтезу логіки, отримані з попередньої компіляції.

Функціональний екстрактор SNF (функціональний екстрактор тестів) створює файл функціонального тесту (.snf). Компілятор генерує цей файл перед синтезом проекту, тому він містить усі вузли, присутні у вихідних файлах проекту. Цей файл SNF не містить інформації про час. Його генерація можлива лише в тому випадку, якщо проект скомпільовано без помилок.

Екстрактор Timing SNF створює (якщо проект компілюється без помилок) файл синхронізації (.snf), який містить дані синхронізації проекту . Цей файл призначений для тестування та аналізу часу. Крім того, ці файли SNF також використовують модулі компілятора, що містять записи EDIF, Verilog і VHDL, які генерують вихідні файли в цих форматах, а також (необов'язково) стандартні файли відкладеного виведення (.sdo) (стандартні затримка формат Вихід файл).

Ви можете наказати компілятору створити оптимізований файл SNF за допомогою команди Processing / Timing SNF Extractor . Оптимізація файлу SNF

збільшує час компіляції, але допомагає заощадити час на тестування та аналіз часу.

Зв’язаний екстрактор SNF (екстрактор тестування макета) створює файл (.snf) для тестування макета під час тестування кількох дизайнів (рівень плати). Цей файл SNF поєднує інформацію з двох типів файлів SNF: тести часу та функціональні тести, які були створені окремо для цих кількох проектів. Складні проекти можуть використовувати пристрої з різних сімейств. Якщо тестовий файл збірки містить лише інформацію про час, його також можна використовувати для аналізу часу.

Список мереж EDIF Writer (програма написання EDIF). Компілятор MAX PLUS II може працювати з більшістю стандартних програм САПР, які зчитують стандартні файли EDIF 200 або 300. Цей (додатковий) модуль компілятора, що містить записувач EDIF, створює один або кілька вихідних файлів EDIF (.edo), що містять інформацію про функції та (опціонально) часові параметри, отримані після синтезу. Інформацію про час також можна записати в окремі вихідні файли у стандартному форматі затримки (.sdo) .

Verilog Список мереж Writer (програма для запису Verilog)

Записувач Verilog створює вихідні файли з розширенням . vo, що містить інформацію про функції та часові параметри, отримані після синтезу. Інформацію про час також можна записати в окремі вихідні файли у стандартному форматі затримки (.sdo) . Список мереж VHDL Записувач VHDL (записувач VHDL)

Додатковий модуль компілятора запису VHDL генерує один або кілька вихідних файлів VHDL 1987 або 1993 (.vho), що містять функцію та (необов’язкову) інформацію про час із синтезу . Інформацію про час також можна записати в окремі вихідні файли у стандартному форматі затримки (.sdo) .

Вихідні файли на мовах опису обладнання можна використовувати при перевірці конструкції за допомогою зовнішнього симулятора. Ці файли

генеруються лише після успішної компіляції проекту. Асемблер (монтажний модуль)

Модуль асемблера перетворює призначення воріт, контактів і пристроїв, зроблені модулем трасування Fitter, на образ програми для пристрою(ів) у формі одного або кількох бінарних об’єктних файлів програматора (.pof) або об’єктних файлів SRAM (.sof).).); Для деяких пристроїв компілятор також створює файли JEDEC ASCII (.jed), що містять інформацію про розробника, конфігураційні файли ASCII (.ttf) і файли ASCII у форматі Intel (.hex). Об'єктні файли POF і SOF, а також конфігураційні файли JEDEC потім обробляються системним програматором MAX PLUS II і обладнанням для програмування Altera (або іншим програматором). Файли HEX і TTF можна використовувати для різного налаштування пристроїв FLEX 8000, FLEX 6000 і FLEX 10K. Модуль асемблера створює файли для програміста лише після успішного завершення проекту.

Після завершення компіляції системний компілятор і програматор MAX PLUS II дозволяє генерувати додаткові файли програмування пристрою, які можна використовувати в інших середовищах програмування. Наприклад, ви можете створювати файли послідовного потоку бітів (.bbs) і необроблені двійкові файли (.rbf), щоб налаштувати пристрої FLEX 6000, FLEX 8000 і FLEX 10K . Ви можете створювати послідовні тестові векторні файли (.svf) або мовні файли JAM (.jam) для програмування пристроїв у автоматичному тестовому обладнанні, такому як ATE .

Дизайн лікар Інструмент (інструмент діагностики проекту). Додатковий інструмент діагностики проекту перевіряє логіку кожного файлу проекту, щоб визначити елементи, які можуть викликати проблеми з надійністю на рівні системи. Зазвичай ці проблеми виявляються лише після апаратного завантаження пристрою. Ви можете вибрати з трьох попередньо визначених наборів правил розробки проекту з різними рівнями. Крім того, ви можете розробити власний набір принципів проектування.

Правила розробки базуються на принципах надійності, які включають логіку, що містить асинхронні входи, тактові сигнали , багаторівневу логіку в

годиннику , попередньо встановлені та чіткі конфігурації та умови змагання. Налаштування правил перевірки здійснюється за допомогою команди Design Doctor Налаштування (Рис. 2.29)

Рис. 2.29 . Вікно Команди для налаштувань Design Doctor

Перевірка проекту

Для перевірки конструкції (див. рис. 2.30) система MAX PLUS II використовує три програми: симулятор (Simulator), аналізатор часу (Timing Аналізатор) і редактор сигналів (Waveform редактор).

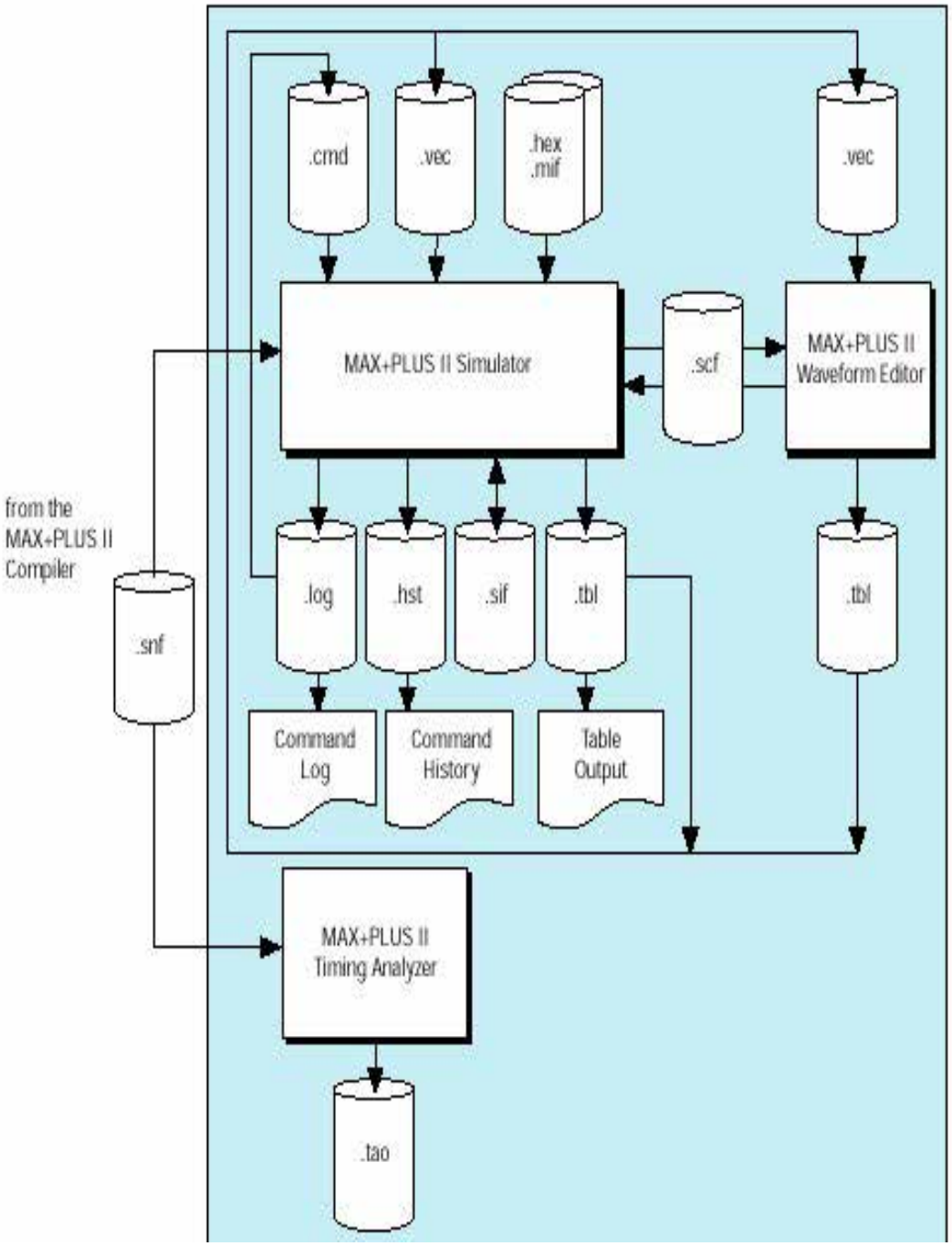


Рис. 2.28. Перевірка проекту в системі MAX PLUS II Симулятор

Симулятор системи MAX PLUS II перевіряє логічні операції та внутрішню синхронізацію проекту, дозволяючи користувачеві імітувати проект. Тренажер може працювати в інтерактивному або автоматичному (пакетному) режимі. Вікно симулятора показано на рис. 2.31.

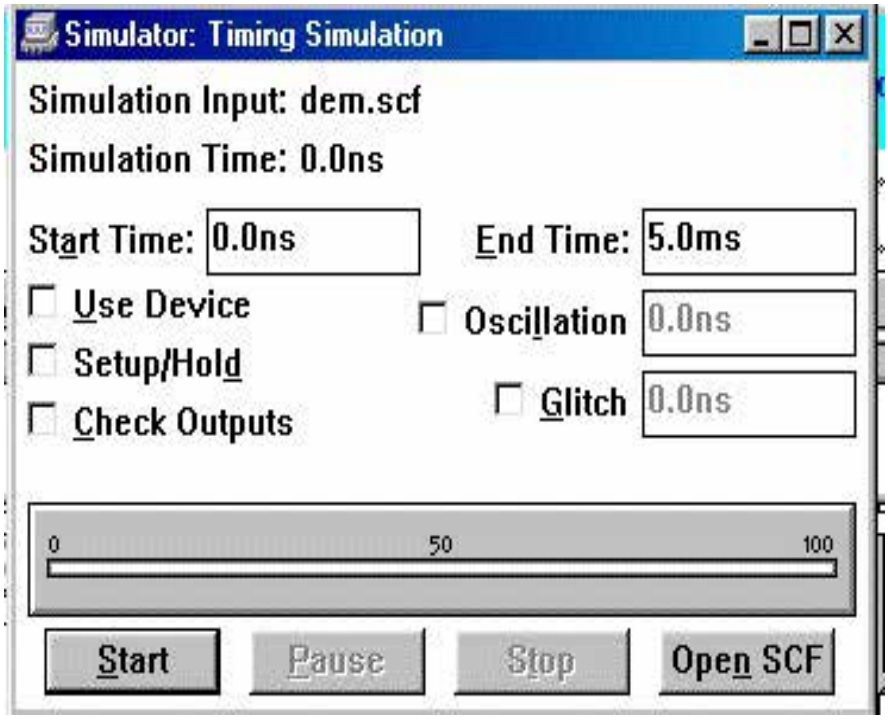


Рис. 2.31. Симулятор MAX PLUS II

Перед тестуванням проект потрібно скомпілювати, вказавши опцію компілятора для створення файлу (.snf) для функціонального тестування, тестування часу або тестування макета кількох проектів (пристроїв) . Отриманий файл SNF для поточного проекту автоматично завантажується, коли ви відкриваєте симулятор.

Для проектів, пов’язаних із пам’яттю, ви можете вказати початковий вміст пам’яті у шістнадцяткових (Intel) файлах із розширенням . hex або у файлах ініціалізації пам'яті з розширенням . mif . Редактор сигналів може автоматично створити файл SCF за замовчуванням, який користувач може редагувати, щоб отримати необхідні тестові вхідні вектори. Якщо замість цього використовується векторний текстовий файл ASCII, редактор сигналів автоматично згенерує з нього файл тестового каналу SCF.

Симулятор дозволяє перевіряти вихідні значення, отримані під час тестів, з результатами, що містяться у файлі SCF (вказані користувачем прогнозовані значення або результати попередніх тестів). Використовуючи відповідне обладнання для програмування, також можна виконувати функціональні тести для перевірки фактичних вихідних значень програмованого пристрою на основі результатів тестування.

Використовуючи різні параметри симулятора, ви можете контролювати свій проект на наявність помилок , а також установлювати порушення та затримки часу. Після завершення тестів ви можете відкрити редактор сигналів, щоб переглянути оновлений файл SCF або зберегти отримані вихідні значення у файл таблиці з розширенням . tbl , а потім переглянути результати в текстовому редакторі.

Тести на функціональність. Якщо компілятор має «завдання» створити файл SNF для функціонального тестування, він генерує його перед синтезом дизайну. Таким чином, всі вузли конструкції можуть бути змодельовані під час функціонального тестування.

Під час функціонального тестування симулятор ігнорує всі затримки поширення. Таким чином, немає затримки у файлі SNF для функціонального тестування; вихідні логічні рівні змінюються одночасно з вхідними векторами.

2.9. Тестування часових параметрів

Компілятор генерує файл SNF для тестування параметрів синхронізації після повного синтезу та оптимізації проекту. Тому цей файл містить лише ті вузли, які не були знищені в процесі логічного синтезу.

З цього файлу симулятор отримує інформацію про апаратне забезпечення, завантажену з файлів моделі пристрою (.dmf), підключених до системи MAX PLUS II.

Якщо ви розділили свій проект на кілька пристроїв, компілятор створює файл SNF для всього проекту та для кожного пристрою. Проте перевірка часу виконується лише для проекту в цілому.

Тестування синхронізації можна прискорити, вказавши компілятору створити оптимізований файл SNF, що містить динамічні моделі, що представляють різні типи комбінаторної логіки. Це збільшує час процесора компілятора, але отриманий оптимізований SNF може скоротити час тестування, оскільки симулятор може працювати з

динамічними моделями замість того, щоб інтерпретувати всю логіку в комбінаторній схемі .

Створюючи файл SNF для тестування посилань у кількох проектах, компілятор об’єднує файли SNF функціональних тестів і/або файли тестів синхронізації для кількох окремих проектів. Окремі « підпроекти » у збірці SNF можуть бути присвячені пристроям різних сімейств. Крім того, оскільки файли SNF для функціонального тестування створюються до завершення повної збірки, можна вводити підпроекти для представлення логіки, не реалізованої в пристрої Altera.

Чіп SNF можна використовувати для тестування на рівні плати. Крім того, якщо він містить лише інформацію про час, його можна використовувати для запуску аналізатора системного часу MAX PLUS II.

Симулятор разом з іншими додатками системи MAX PLUS II дозволяє виконувати такі завдання:

- встановити очікувані логічні рівні на виході, які можна порівняти з результатами тестування;
- імітувати окремі вузли або вузли, об’єднані в групи. Ви можете комбінувати біти кінцевого автомата в проекті, моделювати їх як групу та посилатися на них за назвою стану;
- Визначте період часу, який представляє вібрацію або несправності, і проаналізуйте конструкцію для обох або однієї з цих умов.
- контролювати наявність порушень початкових налаштувань реєстру та часових затримок у проекті;
- запис фактичних вихідних значень пристрою замість змодельованих;
- виконувати функціональні проби. Можна перевірити, що змодельовані вихідні значення функціонально еквівалентні фактичному виходу пристрою;

- встановіть умови точки зупинки, які змушують симулятор призупиняти роботу під час процесу тестування;
- список імен і логічних рівнів будь-якої комбінації вузлів і груп, а також ініціалізація логічних рівнів вузла або групи перед тестуванням;
- ініціалізувати вміст блоків RAM (RAM) або ROM (ROM) перед тестуванням;
- зберігати ініціалізовані значення вузла та групи, включаючи ініціалізований вміст пам’яті, у файлі ініціалізації симулятора (.sif) або перезавантажувати ініціалізовані значення, що зберігаються у файлі ;
- Записуйте команди симулятора в текстовий файл журналу тестування (.log) у тому самому форматі, що й командний файл (.cmd) , який використовується під час тестування в автоматичному (пакетному) режимі, а потім використовуйте цей файл журналу для повторення цього циклу тестування . Команди симулятора та отримані результати також можна зберегти у файлі історії тестів із розширенням . hst .

Отже, ми розглянули основні прийоми використання пакету MAX+PLUS II . Звичайно, в рамках однієї глави практично неможливо детально розглянути всі прийоми роботи з таким складним і різноманітним програмним засобом, однак зацікавлений користувач може освоїти пакет самостійно, використовуючи цю книгу і фірмове програмне забезпечення.

РОЗДІЛ ІІІ. ПРИНЦИПИ ОПИСУ АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ НА МОВІ AHDL

Загальні відомості про спеціальну мову програмування AHDL

AHDL — це низькорівнева модульна мова, повністю інтегрована в систему MAX + PLUS II . Вона добре підходить для проектування складної комбінаційної логіки, кінцевих машин, таблиць істинності параметричної логіки. Для створення текстових файлів проекту можна використовувати текстові редактори системи MAX + PLUSII або будь-які інші (AHDL текст Дизайн Файли (.tdf)). Потім можна скомпілювати. tdf для стилів виводу, придатних для подальшого моделювання, аналізу часу та програмування пристрою. Крім того, системний компілятор MAX + PLUS II дозволяє створювати експортні текстові файли (AHDL текст Дизайн Експорт Файли) і вихідні текстові файли (Текст Дизайн Вихід Файли (.tdo)), який можна зберегти як . tdf для повторного використання як файлів проекту.

Ви можете створити цілий ієрархічний проект за допомогою AHDL або mix. tdf з іншими типами файлів в один проект, використовуючи будь-який текстовий редактор для створення . tdf , але тільки текстовий редактор системи MAX + PLUSII дозволяє використовувати його при вході, компіляції та налагодженні проекту.

AHDL можна легко вставити в ієрархію проекту. У текстовому редакторі можна автоматично створити символ, що представляє . tdf і помістіть його в графічний файл проекту (Graphic Дизайн Файл (.gdf)). Подібним чином об’єднайте власні функції та понад 300 мега- та макрофункцій, наданих Altera , у будь-якому файлі . файли tdf . Ви можете скористатися командами меню «Призначити» або «Призначити та конфігурувати» . Файл (.acf) , щоб створити ресурс і вибрати пристрій, перевірити синтаксис і виконати повну збірку для налагодження та запуску проекту. Будь-які помилки автоматично визначаються MessageProcessor і виділяються у вікні текстового редактора.

Для застосування AHDL використовується текстовий редактор . Цей редактор має вбудовані можливості для зручного введення файлів проекту, їх компіляції та налагодження з повідомленнями про помилки та їх розташуванням.

3.1. Елементи мови AHDL

Зарезервовані ключові слова використовуються для керування операторами AHDL , а також визначеними константами GND і VCC . Зарезервовані ключові слова відрізняються від зарезервованих ідентифікаторів тим, що ключові слова можна використовувати як символічні імена, якщо вони взяті в одинарні лапки ('), тоді як зарезервовані ідентифікатори ні. І тих, і інших можна вільно використовувати в коментарях.

Altera рекомендує вводити всі ключові слова великими літерами, щоб їх було легше читати.

Щоб отримати довідку щодо ключових слів, спочатку переконайтеся, що файл TDF збережено з розширенням .tdf. Потім відкрийте файл у вікні текстового редактора та натисніть Shift + F1 і виберіть певну кнопку довідки на панелі інструментів.

Нижче наведено список зарезервованих слів:

AND	FUNCTION	OUTPUT
ASSERT	GENERATE	PARAMETERS
BEGIN	GND	REPORT
BIDIR	HELP_ID	RETURNS
BITS	IF	SEGMENTS
BURIED	INCLUDE	SEVERITY
CASE	INPUT	STATES
CLIQUE	IS	SUBDESIGN
CONNECTED_PINS	LOG2	TABLE

CONSTANT	MACHINE	THEN
DEFINE	NAND	TO
DEVICE	NOR	VARIABLE
DEFAULTS	MOD	TITLE
DIV	NOT	VCC
DESIGN	NODE	TRI_STATE_NODE
ELSIF	OPTIONS	WITH
ELSE	OF	WHEN
FOR	OTHERS	XOR
END	OR	XNOR

Нижче приведений список зарезервованих ідентифікаторів:

CASCADE	JKFF	SRFF
CARRY	JKFFE	SRFFE
DFFE	LCELL	TFF
CEIL	LATCH	TFFE
EXP	MEMORY	USED
DDF	MCELL	TRI
GLOBAL	SOFT	X
FLOOR	OPENDRN	WIRE

Символи

Специфічне значення в AHDL. Цей список містить символи, які використовуються як оператори та компаратори в логічних виразах і як оператори в арифметичних виразах.

Символи	функція
_(підкреслення)	Визначені користувачем ідентифікатори, які використовуються як дійсні символи в іменах символів
---(два тире)	вбудований коментар у стилі VHDL .
% (відсотки)	Обмежує коментарі у стилі AHDL .
()(дужки)	Обмежте та визначте узгоджені назви шин. приклад: шина (a , b , c) складається з вузлів a , b і c . SUBDESIGN розділи та інструкції прототипу функції. Крім того, він обмежує введення та виведення таблиць істинності в операторах TruthTable . Містить біти оголошення StateMachine і стани . Обмежити операції з найвищим пріоритетом у логічних і арифметичних виразах. Обмежте визначення параметрів у операторах Parameters, оголошеннях екземплярів та іменах параметрів у операторах FunctionPrototype і вкладених посиланнях. Крім того, обмеження умови в операторі Assert . Обмежити аргументи функції оцінки в операторах Define .
[] (дужки)	Перегляньте пропозицію шин.
" ... " (лапки)	Обмежити символічні імена

" ... " (подвійні лапки)	заголовок , параметри та оператори підтвердження . Обмежити імена файлів у операторах include . Обмежте числа недесятковими числами.
. (точка)	Відокремлює символічні імена змінних логічної функції від імен портів. Відділяє розширення від імен файлів.
..(еліпс)	Відділяє старший біт від молодшого.
; (крапка з комою)	посібники та розділи AHDL .
, (кома)	Відокремлює символічні імена від типів у оголошеннях
= (дорівнює)	Призначає значення за замовчуванням GND і VCC входам у розділі Subdesign . Призначає значення параметрам у операторі Option . Призначає значення за замовчуванням параметрам у операторі Parameters або у вкладеному посиланні. Присвоює значення станам кінцевого автомата. Присвоює значення логічним рівнянням. Підключає сигнал до порту за посиланням, яке використовує підключення за назвою порту.
= > (стрілка)	Відокремлює введення від виводу в операторах TruthTable . Відокремлює пропозиції WHEN від логічних виразів у операторах CASE .
+ (плюс)	Оператор додавання
- (мінус)	Оператор віднімання
== (два знаки рівності)	Оператор еквівалентності рядка або числа

! (знак оклику)	номер оператора
!= (знак оклику дорівнює)	Оператор нерівності
> (біліше ніж)	Для порівняння є ще щось
> = (більше або дорівнює)	Компаратор більше або дорівнює
< (менше)	Порівняння менше ніж
<= (менше або дорівнює)	Компаратор менше або дорівнює
& (амперсант)	Оператор I
!& (Ambient Apersant)	Оператор I-HI
$ (знак долара)	Ексклюзивним оператором є OR
!$ (долар зі знаком оклику)	Єдиним оператором є АБО-HI
# (знак фунта)	оператор АБО
!# (знак оклику)	оператор АБО
? (питання)	Тернарний оператор. Він використовує такий формат: <вираз 1>?<вираз 2>:<вираз 3> Якщо перший вираз не дорівнює нулю (істина), то обчислюється другий вираз і результат повертається до потрійного виразу. В іншому випадку повертається значення третього виразу.

Строкові і символьні імена

Існує три види імен. Символьні імена є ідентифікаторами користувачів. Вони використовуються для оголошення таких частин TDF :

- внутрішні та зовнішні вузли та шини;
- постійний
- машини, що змінюють стан, біти стану та імена станів

● копії

● сегменти пам'яті

● оцінні функції

● іменовані оператори

Імена підпроектів — це визначені користувачем імена для файлів проекту нижчого рівня. Ім'я підпроекту має відповідати імені файлу TDF .

Імена портів – це символічні імена, які ідентифікують входи або виходи логічної функції.

Компілятор генерує імена, що містять символ тильди (~), які можуть з’являтися у FitFile проекту . Якщо використовується зворотна анотація, ці імена також відображаються у файлі ACF проекту . Символ тильди зарезервований лише для імен, згенерованих компілятором, і може використовуватися в настроюваних іменах пінів, вузлів і шин.

Для трьох типів імен доступні два типи представлення: у лапках і без лапок. Імена рядків беруться в одинарні лапки ('), а імена символів – ні.

Під час створення токенізованого представлення файлу TDF , що містить рядки імен портів, лапки не включаються в символ pinstub .

Шини

Символічні імена та порти одного типу можуть бути оголошені як шини в логічних виразах і рівняннях.

Шина, яка може містити до 256 елементів (або бітів), розглядається як сукупність вузлів і діє як одне ціле.

константи GND і VCC можуть бути продубльовані для формування шини.

Про шини можна повідомити трьома способами:

Ім’я шини складається з символічного імені або порту, за яким слідує специфікація піддіапазону, укладена в дужки, тобто *[* 4..1]. Ім'я та найдовший номер у діапазоні можуть містити до 32 символів. приклад,

Ім'я q [MAX..0] дійсне, якщо константа MAX описана вище в операторі Constant .

Коли визначено шину, дужки [] є циліндричним методом опису всього діапазону. приклад,

i [4..1] можна вказати як *[*].

b [6..0][3..2] можна вказати як *b* [][].

Ім’я шини складається з символічного імені або імені порту, за яким слід специфікація піддіапазону, укладена в дужки, тобто *d* [6..0][2..0]. Ім'я та найдовший номер у діапазоні можуть містити до 32 символів. Один вузол на шині може *називатися* ім’ям [y] [*z*] або ім’ям *y_x* , де *yix — це числа* в *області* шини.

Ім’я послідовності шини складається зі списку розділених комами символічних імен, портів або чисел, укладених у круглі дужки, наприклад (*a , b , c*).

приклад,

Вхідні порти *змінної* DFF reg можна записати як *reg .(d , clk , clrn , prn).*

Нижче наведено два набори прикладів, які показують дві шини, описані різними методами:

B [5..0]

(b5, b4, b3, b2 , b1, b0)

B []

b[log2(256)..1+2-1]

b[2^8..3mod1]

b[2*8..div2]

Діапазони в назвах шин можуть складатися з чисел або арифметичних виразів, розділених двокрапками (..) і взятих у квадратні дужки [] . приклад,

шина [4..1] з елементами *a 4, a 3, a 2* і *a 1* .

d [B ”10”.. B ”00”] шина з *d 2, d 1* і *d 0* .

b [2*2..2-1] шина з *b 4, b 3, b 2* і *b 1* . Обмежувачі діапазонів є арифметичними виразами.

q [MAX..0] є дійсною шиною, якщо константа MAX описана в операторі *Constant* .

c [MIN (a , b)..0] є прийнятною шиною, якщо функція MIN , що оцінюється , описана в операторі *Define* .

t [WIDTH -1..0] є прийнятною шиною, якщо параметр WIDTH описано в операторі *Parameters* .

Незалежно від того, чи є роздільник діапазону числом чи арифметичним виразом, компілятор розділяє та інтерпретує роздільники як десяткові (цілі) значення.

Піддіапазони містять підмножину вузлів, визначених у оголошених напрямках і лише в напрямках ліворуч від логічного рівняння або заміненого посилання. приклад:

якщо ви оголошуєте шину *c* [5..1], ви можете використовувати такі піддіапазони цієї шини:

до [3..1], до [4..2], до 4, до [5], (до 2, , до 4)

Кома використовується в піддіапазоні *(c 2, , c 4) , щоб заощадити простір, не призначений члену шини.*

Діапазони зазвичай перераховуються в порядку спадання. Щоб вказати діапазони в іншому порядку, визначте параметр VIT0 за допомогою оператора *Options* , щоб компілятор не відображав попереджувальні повідомлення. Для дводіапазонних шин це стосується обох діапазонів.

3.2. Числа в AHDL

Ви можете використовувати десяткові, двійкові, вісімкові та шістнадцяткові числа в будь-якій комбінації. Синтаксис для кожної основи показано нижче.

база:	Значення:
Десятковий	< рядок чисел від 0 до 9 >
двійковий	B "<рядок 0, 1 і X>"(X = "будь-який стан")

Вісім років	O”<рядок цифр від 0 до 7>” або Q "<рядок цифр від 0 до 7>"
шістнадцять	X "<рядок цифр від 0 до 9, від A до F >"
	H "<рядок цифр від 0 до 9, від A до F >"

Для чисел застосовуються такі правила:

● Компілятор MAX + PLUSII завжди інтерпретує числа в логічних виразах як групи двійкових цифр, числа в діапазонах шин як десяткові значення;

● у логічних рівняннях числа не можуть бути призначені окремим вузлам, замість них використовуються GND і VCC.

Арифметичні вирази

Оператор/компаратор	приклад	опис	Пріоритет
+(унарний)	+1	Позитивний	1
-(унарний)	-1	Негативний	1
!	!I	НІ	1
^	a^2	Ступінь	1
MOD	4MOD2	Модуль	2
ВІДДІЛ	4DIV2	Розподіл	2
*	a*2	Множення	2
LOG2	LOG2(4-3)	Логарифм за основою 2	2
+	1+1	Доповнення	3
-	1-1	Віднімання	3
== (числовий)	5 == 5	Числова рівність	4
== (термін)	"a" == "b"	Рівність умов	4
!=	5!=4	Не рівні	4

>	5>4	Більш ніж	4
>=	5>=5	більше або дорівнює	4
<	a<b+2	Менше ніж	4
<=	a<=b+2	Менше або дорівнює	4
&	a&b	I	5
I	a I б		
!&	1!&0	NAND	5
NAND	!NAND0		
$	1$1	XOR	6
XOR	1XOR1		
!$	1! 1 долар	XNOR	6
XNOR	1XNOR1		
#	a#b	АБО	7
АБО	a або b		
!#	a !# b	АБО	7
АБО	HI b		
?	(5<4)?3:4	Тримісний	8

Арифметичні вирази можна використовувати для визначення обчислюваних функцій у операторах *Define* , констант у операторах *Constant* , значень параметрів у операторах *Parameters* і як розмежувачі діапазону шини.

Ці вирази використовують арифметичні оператори та компаратори для виконання базових арифметичних операцій і операцій порівняння з числами, які вони містять. В арифметичних виразах використовуються наступні оператори та компаратори:

1. Арифметичні вирази повинні давати невід’ємні числа.

2. Якщо результат LOG 2 не є цілим числом, він автоматично округляється до наступного цілого числа. Наприклад, LOG 2(257)=9.

Арифметичні оператори, які підтримуються в арифметичних виразах, є надмножиною арифметичних операторів, які підтримуються в логічних виразах.

Логічні вирази

Логічні вирази складаються з операндів, розділених логічними операторами, арифметичними операторами та компараторами, а також додатково згрупованих дужками. Вирази використовуються в логічних рівняннях, а також в інших операторах, таких як *Case* і *Якщо тоді* .

Логічний вираз може мати одну з наступних форм:

- операнд

Наприклад *a, b* [5..1], 7, VCC;

- Посилання на логічну функцію зі знаком підстановки

Наприклад, out[15..0] = 16dmux(q[3..0]);

- Префіксний оператор (! або -), застосований до логічного виразу

Наприклад *!c* ;

- Для логічних виразів, розділених двійковим оператором

Наприклад, *d 1* $ *d 3 ;*

- Логічний вираз, укладений у круглі дужки

Наприклад (! foo & bar).

Ви можете називати логічні оператори та компаратори у файлах AHDL , щоб допомогти вам вводити призначення ресурсів та інтерпретувати розділ рівнянь у файлі звіту.

Булеві оператори

У логічних виразах можна використовувати такі логічні оператори:

Оператор	приклад	опис
!	! бути	Додаток до 1
HI	Не ти	
&	хлібного масла	I

I	хліб з маслом	
!&	a[3..1] !& b[5..3]	НІ
NAND	a[3..1] NAND b[5..3]	
#	трюк #лікування	АБО
АБО	трюк або задоволення	
!#	c[8..5] !# d[7..4]	НІ
АБО	c[8..5] NOR d[7..4]	
$	foo $bar	Не включає АБО
XOR	foo strip XOR	
!$	x2 !$ x4	Ексклюзив АБО НІ
XNOR	x2XHI x4	

Кожен оператор є логічним вентилем із двома входами, за винятком оператора NOT (!), який є префіксом інверсії одного вузла. У той же час, можна використовувати ім'я або символ для представлення логічного оператора.

Вирази, які використовують ці оператори, інтерпретуються по-різному залежно від того, чи є операнди окремими вузлами, шинами чи числами.

дозволити компілятору замінювати оператори I та всі компрамери в логічних виразах *функціями* lpm_add_sub і lpm_Compare , включаючи булеву опцію UseLPMforAHDLOperators .

Логічні оператори з використанням NOT

Оператор NOT є префіксом інвертора. Поведінка оператора NOT залежить від операнда, на який він впливає.

З оператором NOT можна використовувати три типи операндів:

- якщо аргументом є один вузол, GND або VCC , виконується одна інверсія, наприклад, !a означає, що сигнал проходить через інвертор;

- якщо операндом є група вузлів, то кожен елемент цієї групи проходить через інвертор, наприклад, шина !a[4..1] інтерпретується як (!a4, !a3, !a2, !a1);
- якщо операнд є числом, воно інтерпретується як двійкове число, і кожен його біт інвертується. Наприклад, !9 інтерпретується як !B"1001", що є B "0110".

Логічні оператори з використанням AND , NAND , OR , NOR , XOR та XNOR .

Є п'ять комбінацій операндів з двійковими операторами. Кожне з цих поєднань тлумачиться по-різному:

- якщо обидва операнди є окремими вузлами або константами GND і VCC , оператор виконує логічну операцію над двома елементами, наприклад *(a $ b)* ;
- якщо обидва операнди є групами вузлів, оператор діє на відповідні вузли кожної групи, виконуючи побітові операції між групами, групи мають бути однакового розміру, наприклад (a *, b , c) # (d , e , f)* інтерпретується як *(a #d , b , #e , c , #f)* .
- якщо один операнд є окремим вузлом, GND або VCC , а інша група вузлів, окремий вузол або константа дублюються, щоб утворити групу такого ж розміру, як і другий оператор, тоді вираз тактується як групова операція, наприклад, a *& b [* 4 ..1] інтерпретується як (*a & b4, a & b 3, a & b 2, a & b 1*).
- якщо обидва операнди є числами, коротше число розширюється знаком, що відповідає величині другого числа, і потім інтерпретується як групова операція, наприклад, у виразі (3 # 8) 3 і 8 перетворюються на двійкові числа "0011 " і "1000" відповідно", результатом буде B "1011".
- якщо один операнд є числом, а інший є вузлом або групою вузлів, то число розбивається на біти відповідно до розміру групи, а вираз розглядається як групова операція, наприклад, у виразі (a, b, c) & 1 *, 1*

перетворюється на B " 001", і вираз набуває вигляду (a , b , *c*) & (0, 0, 1), результатом буде (*a & 0, b & 0, c & 1*).

Вираз, що використовує VCC як аргумент, інтерпретується залежно від виразу, який використовує 1 як операнд. Наприклад, у першому виразі 1 є числом розширеного формату зі знаком. У другому виразі вузол VCC дублюється. Потім кожен вираз інтерпретується як групова операція.

(a , b , c) і 1 = (0, 0, c)

(a , b , c) і VCC = (a , b , c)

Арифметичні оператори в булевих виразах

Арифметичні оператори використовуються для виконання арифметичних операцій додавання та віднімання над числами та шинами в булевих виразах. Вони використовують такі оператори:

Оператор	приклад	опис
+(унарний)	+1	Плюс
-(унарний)	- a[4..1]	Мінус
+	число[7..0] + дельта[7..0]	Доповнення
-	right_x[] – lcftmost_x[]	Віднімання

До бінарних операторів можуть застосовуватися такі правила:

- операції виконуються між двома операндами, які повинні бути рядками або числами;
- якщо обидва операнди є шинами, вони повинні бути однакового розміру;
- якщо обидва операнди є числами, коротше число розширюється до розміру другого операнда;
- якщо один операнд є числом, а інший вузол є групою вузлів, то число скорочується або розширюється, щоб відповідати розмірам операндів. Якщо будь-які значущі біти відкинуто, компілятор MAX + PLUSII відобразить повідомлення про помилку.

Додавши дві рейки праворуч від логічного рівняння за допомогою оператора +, ви можете поставити 0 ліворуч від групи, щоб збільшити ширину

рейки. Цей метод додає до лівої частини рівняння додатковий біт даних, який можна використовувати як вихідні дані для переносу. Наприклад, шини лічильника [7..0] і дельта [7..0] доповнюються нулями, щоб надати інформацію про сигнал лічильника :

(число , відповідь [7..0]) = (0, число [7..0]) + (0, дельта [7..0])

Компаратори

Для порівняння окремих вузлів або шин використовуються два типи компараторів: логічний і арифметичний. Наступні компаратори можна використовувати в логічних виразах.

компаратор	приклад	опис
==(логічне значення)	адреса[19..4] ==B”B800”	Дорівнює
!=(логічний)	b 1 != b3	Не рівні
< (арифметика)	слава[] < сила[]	Менше ніж
<=(арифметика)	гроші[] <= сила[]	Менше або дорівнює
>(арифметика)	кохання[] > гроші[]	Більш ніж
>=(арифметика)	дельта[] >= 0	більше або дорівнює

Логічні компаратори можуть порівнювати окремі вузли, шини та числа без невизначених значень (X). Під час порівняння шин чи номерів шини мають бути однакового розміру. Компілятор MAX + PLUSII виконує порозрядне порівняння шин, повертаючи VCC , коли порівняння істинне, і GND , коли порівняння хибне.

Арифметичні компаратори можуть порівнювати лише шини та номери; шини повинні бути однакового розміру. Компілятор виконує беззнакове порівняння значень шини, тобто кожна шина інтерпретується як позитивне двійкове число та порівнюється з іншою шиною.

Пріоритети булевих операторів і компараторів

Операнди, розділені логічними, арифметичними операторами та операторами порівняння, оцінюються відповідно до наступних правил пріоритету (пріоритет 1 є найвищим). Операції з однаковим пріоритетом оцінюються зліва направо. Ви можете використовувати parentheses(), щоб змінити порядок обчислень.

Пріоритет:	Оператор/компаратор:
1	- (мінус)
1	! (НІ)
2	+ (доповнення)
2	- (віднімання)
3	== (дорівнює)
3	!= (нерівно)
3	< (менше)
3	<= (менше або дорівнює)
3	> (більше ніж)
3	>= (більше або дорівнює)
4	& (Я)
4	!& (І НІ)
5	$ (без АБО)
5	! $ (виключно АБО НІ)
6	# (АБО)
6	! # (АБО-НІ)

3.3. Порти

Порт — це вхід або вихід логічної функції. Порт може бути розташований у двох місцях:

- вхідний або вихідний порт поточного файлу з'являється в розділі підпроекту ;
- порт, який є входом або виходом екземпляра примітивного файлу програмування або файлу програмування нижчого рівня, що використовується в розділі логіки .

Порти поточного файлу

Вхідний або вихідний порт поточного файлу відображається в розділі «Підпроект» у такому форматі :

<ім'я порту>:<тип порту>[= <за замовчуванням>]

Доступні такі типи портів

INPUT MACHINEINPUT

OUTPUT MACHINEOUTPUT

BIDIR

Ви можете імпортувати та експортувати кінцеві автомати між TDF та іншими файлами проекту, описуючи вхідні та вихідні дані як MACHINE INPUT або MACHINE OUTPUT у розділі Subproject . Прототипи функцій, які представляють файл, повинні вказувати, які порти належать кінцевому автомату. MACHINEINPUT і MACHINEOUTPUT можна використовувати лише у файлах нижчого рівня в ієрархії проекту.

Порти екземплярів

Порт, який є входом або виходом екземпляра логічної функції.

У наведеному нижче прикладі тригер D відображається як *регулярна змінна* в розділі *«reg* у розділі *Variable»* , а потім використовується в розділі *Logic.*

```
Variable
        reg: DFF;
BEGIN
```

```
        reg.clk = clk
        reg.d = d
        out = reg.q
    END;
```

Усі функції, які пропонує Altera , мають визначені імена портів (pinstub), указані в прототипі функції. Найпоширеніші примітивні імена портів наведено в таблиці нижче:

Опис назви порту

. q тригерний або фіксуючий вихід;

. d введіть дані тригера або фіксатора;

. t тригерний вхід T ;

. j J вхід JK тригер;

. k вхід K тригер JK ;

. s Вхід конфігурації тригера SR ;

. r Вхід скидання звільнення SR ;

. тактовий вхідний тригер clk;

. ena тригер синхронізації дозволу input , обмеження дозволу кінцевого автомата на запуск;

. prn активний попередньо встановлений низький вхід тригера

. clrn активний низький яскравий вхід тригера;

. reset вхід активного високого автомата скидання;

. oe дозвіл на вхід, вихід, примітив TRI;

на головному вході примітивів CARRY, CASCADE, EXP, TRI, OPNDRN, SOFT, GLOBAL і LCELL ;

. вихід примітивів TRI , OPNDRN , SOFT , GLOBAL і LCELL ;

3.4. Структура проекту

У цьому розділі описано структуру проекту в AHDL . Розділи та оператори AHDL описані в тому порядку, в якому вони з’являються у файлі текстового дизайну (TDF - TextDesignFile):

- оператор ;
- оператор ;
- Включити оператор ;
- константний оператор ;
- Визначити оператор ;
- оператор прототипу ;
- оператор ;
- оператор підтвердження ;
- розділ ;
- секція ;
- логічний розділ ;

Оператор Title

Оператор Title дозволяє ввести коментар до текстового файлу проекту, який буде розміщено у файлі звіту (ReportFile), створеному компілятором. Наступний приклад ілюструє використання оператора Title :

TITLE " DisplayController ";

При використанні *оператора заголовка необхідно дотримуватися таких правил* :

- Оператор *Title* починається з ключового слова TITLE, за яким слідує текстовий рядок, узятий у лапки. Оператор закінчується на ";";
- якщо оператор *Title* починається в текстовому файлі проекту, використаний заголовок розміщується на початку ReportFile . У наведеному вище прикладі заголовок *DisplayController* розміщено у файлі звіту;

•*заголовок може містити 255 символів* , крім того, він не може *містити розриви рядків* і *розриви* рядків . Щоб взяти в заголовок лапки, використовуйте пари подвійних лапок, наприклад:

TITLE """EMP5130"" DisplayController ";

•не більше одного оператора *заголовка* ;

•оператор *Title* повинен бути розміщений за межами інших розділів AHDL .

Оператор Parameters

Оператор *Parameters* дозволяє визначити один або кілька параметрів, які керують екземпляром параметричної мега- або макрофункції. Наступний приклад ілюструє використання оператора *Parameters* :

```
PARAMETERS
(
FILENAME ="myfile.mif", -- за замовчуванням за "=" необов'язково.
ШИРИНА,
AD_WIDTH = 8,
NUMWORD = 2^AD_WIDTH
);
```

При використанні *оператора Parameters необхідно дотримуватися таких правил* :

•оператор *Parameters* починається з ключового слова PARAMETERS , за яким слідує список з одного або кількох параметрів і необов’язкових значень за замовчуванням, весь список укладений у круглі дужки;

•параметри в списку відокремлюються комами, імена параметрів від необов'язкових значень за замовчуванням відокремлюються символом (=), у наведеному вище прикладі лише параметр WIDTH не має визначеного значення;

•імена параметрів можуть бути визначеними користувачем або *Altera* ;

•значення параметрів можуть бути текстовими рядками, взятими в лапки, в цьому випадку, якщо значення параметрів не взяті в лапки, компілятор спробує інтерпретувати їх як арифметичні вирази, якщо це не вдасться, вони будуть інтерпретовані як рядки;

•*Параметри* оператора закінчуються символом (;);

•після визначення параметра його можна використовувати у всьому текстовому файлі проекту;

•параметр можна використовувати лише після його визначення;

•імена параметрів повинні бути унікальними;

•назва параметра не повинна містити пробілів, для розділення слів і кращого сприйняття використовуйте символ підкреслення;

•оператор *Parameters* можна використовувати будь-яку кількість разів в одному текстовому файлі проекту;

•*Параметри* оператора повинні бути за межами інших розділів AHDL;

•параметри, які використовуються для визначення інших параметрів, повинні бути визначені заздалегідь;

•використання циклічних посилань заборонено, наступний приклад ілюструє використання недійсного циклічного посилання:

```
PARAMETERS
    (
    FOO = BAR,
    BAR = FOO;
    );
```

При компіляції текстового файлу проекту компілятор шукає значення параметрів у такому порядку:

• екземпляр логічної функції аналізується, наприклад, у текстовому файлі проекту, в об’єкті (екземплярі), створеному за допомогою оголошення об’єкта (декларації екземпляра) або вбудованого посилання, можна вказати параметри, які будуть використовуватися, і за бажанням вкажіть їх значення . У графічному файлі проекту ви можете вибрати символ і за допомогою

команди *«Редагувати порти/параметри»* в меню « *Символ»* призначити значення параметрів для цього об’єкта;

- виконується аналіз екземпляра логічної функції вищого рівня, значення параметрів екземпляра логічної функції вищого рівня поширюються на підфункції цієї логічної функції, якщо екземпляри цих логічних підфункцій не мають власних значення цих параметрів;
- виконується аналіз стандартних глобальних значень параметрів проекту, визначених командою *Global Project Parameters* з *меню Assign, ці значення зберігаються у файлі конфігурації (Assig* nm en t & Configuration file - .asi) проекту;
- додаткові значення за замовчуванням, указані в розділі *«Параметри»* текстового файлу проекту (TDF) або через примітив PARAM у графічному файлі проекту, що описує логічну функцію. Ці значення за замовчуванням застосовуються лише до файлу, в якому вони вказані, і не застосовуються до підпроектів, які вже є в цьому проекті.

Оператор Include

Оператор Include дозволяє імпортувати текст із файлу з розширенням .inc у поточний файл. Наступний приклад ілюструє використання оператора Include:

INCLUDE "const.inc";

Оператор Include має такі характеристики:

- оператор Include починається з ключового слова INCLUDE, за яким слідує ім’я файлу .inc, який потрібно включити, узяте в подвійні лапки;
- якщо ви явно не вкажете розширення пов’язаного файлу, » компілятор припускає, що файл має розширення .inc за замовчуванням;
- оператор Include закінчується на (,);
- на етапі компіляції оператор Include замінюється інформацією з файлу inc, у наведеному вище прикладі файл const.inc замінює текст INCLUDE "çonst.inc".

Оператор Include часто використовується для об’єднання прототипів функцій для файлів, які знаходяться на нижчому рівні ієрархії, ніж даний текстовий файл проекту. Для використання мега- і макрофункцій необхідно спочатку визначити логіку їх роботи у відповідному файлі проекту. Потім використовуйте оператор *Function Prototype* , щоб визначити порти функції. Крім того, ви можете використовувати оператор Include, щоб включити прототип функції, що зберігається у відповідному файлі .inc. Потім ви можете створити оголошення об’єкта (декларація екземпляра) або вбудоване посилання на екземпляр логічної функції .

містить прототип *функції* для поточного файлу проекту за допомогою команди *«Створити стандартний файл для включення» з меню «Файл».*

На етапі компіляції текстового файлу проекту компілятор шукає файли з розширенням .inc у такому порядку:

- спочатку йде пошук в каталозі цього проекту;
- бібліотеки користувача, визначені *командою Бібліотеки користувача в* меню *Параметри* ;
- каталоги *\maxp lu s2\max2 li b\mega_ lpm* і *\maxp lu s2\max2 inc, створені* під час встановлення.

Збережіть і перевірте проект у меню «Файл» або виконайте повну перекомпіляцію проекту, щоб відновити дерево ієрархії проекту, що відображається у вікні відображення ієрархії проекту.

При використанні оператора Include необхідно дотримуватися таких правил:

- у запущеній програмі імена файлів залежать від контексту, у документації MAX+PLUS II імена файлів можна вводити як великими, так і малими літерами, однак, якщо використовується оператор Include, імена файлів мають точно повторювати свої оригінальні імена, імена макросів а мегафункції *Altera* складаються лише з малих літер;
- Оператор *включення* має бути за межами інших розділів AHDL;

•Інструкції *Include* можна використовувати будь-яку кількість разів в одному текстовому файлі проекту.

Файли з розширенням .inc можуть містити лише такі інструкції:

- Function Prototype;
- Define;
- Parameters;
- Constant.
-

Оператор Constant

Оператор *«Constant»* дозволяє вводити назву числа або арифметичного виразу. Наступні приклади демонструють використання оператора *Constant* :

```
CONSTANT   UPPER_LIMIT =130;
CONSTANT   BAR = 1 + 2 DIV 3 + LOG2(256);
CONSTANT   FOO= 1;
CONSTANT   FOO_PLUS_ONE = FOO + 1
```

Константний оператор має такі характеристики:

•оператор *Constant* починається з ключового слова CONSTANT, за яким йде символічне ім’я, потім символ (=) і, нарешті, число (включаючи його основу, якщо необхідно) або арифметичний вираз;

•оператор *Constant* закінчується на (;);

•Після. спосіб визначення константи дозволяє використовувати її у всьому текстовому файлі проекту, у наведеному вище прикладі в розділі *«Логіка»* константа UPPER_LIMIT може використовуватися для представлення десяткового числа 130;

•константи можуть бути визначені за допомогою арифметичних виразів, ці вирази можуть містити попередньо визначені константи;

•компілятор обчислює арифметичні вирази, що використовуються в операторі *Constant* , і спрощує їх до числових значень, не створюючи логічних схем.

При використанні оператора Constant необхідно дотримуватися таких правил:

- константу можна використовувати лише після її визначення;
- імена констант повинні бути унікальними;
- ім'я константи не може містити пробілів, для розділення слів в імені константи використовуйте символ підкреслення;
- оператор *Constant* можна використовувати будь-яку кількість разів у межах одного текстового файлу проекту;
- оператор *Constant* повинен бути за межами інших розділів мови AHDL;
- константи, які використовуються для визначення інших констант, повинні бути визначені заздалегідь;
- використання циклічних посилань заборонено, наступний приклад ілюструє використання неприпустимих циклічних посилань:
  - CONSTANT FOO = BAR;
  - CONSTANT BAR = FOO;

Оператор Define

Оператор *Define* дозволяє визначити оцінювану функцію (функцію з оцінкою), яка є математичною функцією, яка повертає значення, обчислені з необов’язкових вхідних аргументів.

У наступному прикладі описується функція оцінки MAX, яка визначає наявність принаймні одного порту в розділі *підпроекту* :

```
DEFINE MAX(a,b) = (a > b) 7 a : b;
SUBDESIGN
( data [MAX (WIDTH, 0)..0] : INPUT;
Datab [MAX (WIDTH. 0)..0] : OUTPUT; )
BEGIN datab[] = dataaf];
END;
```

Оператор *Define* має такі характеристики:

- Оператор *Define* починається з ключового слова DEFINE, після якого йде символічне ім’я та список одного або кількох аргументів, укладених у дужки;
- аргументи відокремлюються комами, символ (=) відокремлює список аргументів від арифметичного виразу;
- оператор закінчується на (;);
- одного разу визначена функція оцінки може бути пізніше використана у всьому текстовому файлі проекту;
- Попередньо визначені функції оцінювання можна використовувати для визначення функції оцінювання, наприклад, така функція оцінювання MIN_ARRAY_BOUND обчислюється зі значення функції оцінювання MAX:

```
DEFINE MAX(a,b) = (a > b) ? a : b;
DEFINE MIN_ARRAY_BOUND (x) = MAX(0, x) + 1;
```

При використанні оператора *Define* необхідно дотримуватися таких правил:

- функцію оцінки можна використовувати лише після її визначення; .
- імена функцій оцінювання мають бути унікальними;
- назви функцій оцінки не повинні містити пробілів, для розділення слів у назві функції оцінки та покращення її сприйняття використовуйте підкреслення;
- оператор *Define* можна використовувати будь-яку кількість разів в одному текстовому файлі проекту;
- оператор *Define* має бути розміщений за межами інших розділів AHDL .

Оператор Function Prototype

Оператори *прототипу* виконують ту саму функцію, що й символи у файлах графічного дизайну. Обидва містять короткий опис функції, її назву, а також вхідні, вихідні та двонаправлені порти. Порти кінцевих автоматів

можна використовувати для функцій імпорту та експорту для кінцевих автоматів.

Порти введення мега- та макрофункцій не мають стандартних значень, як у файлах графічного редактора MAX+PLUS II. Тому вхідні значення для невикористаних портів повинні бути вказані явно. Крім того, у розділі *Subdesign* ви можете вказати значення за замовчуванням для двонаправлених портів.

Перед створенням об’єкта мега- чи макрофункції переконайтеся, що існує відповідний файл проекту, який описує його логічне функціонування.

Потім за допомогою оператора *прототипу функції* описуються порти функції та створюється екземпляр логічної функції шляхом оголошення об’єкта або посилання, яке потрібно замінити.

Оператор прототипу функції має такі характеристики:

- ключовим словом FUNCTION слідує назва функції;
- список вхідних портів слідує за назвою функції;
- ключове слово WIDTH , а за списком параметрів йде список у дужках, імена розділені комами;
- Ключове слово RETURNS
- під час імпорту та експорту кінцевих автоматів *оператор прототипу функції, який використовується файлом, повинен використовувати порт машини (вказаний за допомогою* ключового слова MACHINE), щоб вказати, які входи та виходи є кінцевими автоматами, наприклад:

```
FUNCTION ss_def (clock, reset, count)
RETURNS (MACHINE ss_out);
```

- оператор прототипу функції закінчується на (;);
- оператор *прототипу функції* повинен знаходитися поза іншими розділами AHDL і передувати екземпляру логічної функції, створеної оголошенням заміненого об’єкта або посилання.

Для примітивного екземпляра також варто використовувати механізм оголошення заміненого об'єкта або посилання. Однак, на відміну від мега-

та макрофункцій, логіка примітиву визначена, і немає необхідності визначати логіку примітиву в окремому файлі проекту. Крім того, немає необхідності використовувати оператор *прототипу функції* , якщо ви не хочете змінити порядок обходу портів примітиву.

Наступний приклад ілюструє прототип функції, який існує за замовчуванням для примітивів JKFF.

FUNCTION JKFF (j, k, elk, clrn, pm)

RETURNS (q);

У цьому прикладі показано модифікований прототип функції для примітиву JKFF:

FUNCTION JKFF (k, j, elk, clrn, pm)

RETURNS (q);

Альтернативою використанню оператора Function Prototype у файлі проекту є використання оператора Include для об'єднання файлів .inc, які містять прототипи функцій, які ви використовуєте. Крім того, MAX+PLUS II має *команду Create Default Include File у* меню *File* , яка автоматично створює файл із розширенням .inc, що містить прототип функції для поточного файлу проекту.

макрофункцій зберігаються у файлах із розширенням .inc у каталогах \maxplus2 \ m ax2lib \ mega_lpm та \maxplus2\max2inc відповідно .

Оператор Options

Оператор *Options* призначений для надання значення опції LiT 0, яке вказує відносно групи, чи є біт з найменшим номером Most Significant Bit (MSB) , Least Significant Bit (LSB) або ваговий індекс, що залежить від позиції цього біт з описом групи. Використання цього параметра дозволяє уникнути створення попереджувальних повідомлень, якщо найменший пронумерований біт у групі не використовується як молодший значущий біт, який є типовим.

При описі групи розмірності, визначеної діапазоном чисел, ліве число представленого діапазону завжди є індексом старшого біта; відповідно, правильний номер представленого діапазону завжди є індексом найменш зваженого біта. Якщо заданий діапазон чисел представлений у порядку зростання, а параметр В IT0=M8B не встановлено, буде згенеровано попередження. Якщо вибрано параметр BIT 0= MSB і заданий діапазон представлений у порядку спадання, також буде згенеровано попереджувальне повідомлення. Якщо опція BIT0 = ANY , можна визначити розміри групи за діапазонами чисел, представлених як у порядку зростання, так і в порядку спадання, без генерації попереджень.

Оператор Options *починається з* ключового слова OPTIONS , за яким слідує параметр VIT0 і його встановлення. Оператор *опції* закінчується на (;);

Наступний приклад ілюструє використання оператора Options :

OPTIONS BIT0 = MSB ;

У наведеному прикладі біт із найменшим номером у групі визначається як старший біт (MSB). Інші можливі варіанти LSB - найменший; вага, а будь-який — це вага залежно від позиції біта з найменшим номером в описі групи.

Оператор Options , розташований на початку текстового файлу проекту, призводить до того, що налаштування порядку переходу бітів у групах застосовуються до всього файлу проекту. Якщо поточний файл проекту є файлом проекту верхнього рівня, налаштування в операторі *Options* застосовуються до всіх підпроектів, які містяться в цьому проекті верхнього рівня. Якщо поточний файл проекту не є файлом проекту верхнього рівня, тоді дія оператора «Установити *параметри»* застосовується лише до цього файлу проекту.

Оператор Assert

Оператор *Assert* дозволяє перевіряти дійсність арбітражних виразів за допомогою параметрів, чисел, функцій оцінки, а також статусів портів (порт використовується чи ні).

Наступний приклад ілюструє використання оператора *Assert* :

```
ASSERT (WIDTH>0)
REPORT    "Ширина (%)позитивним цілим" WIDTH
SEVERITY        ERROR
HELP_ID         INTVALUE;
```

ключовим словом ASSERT слідує арифметичний вираз, укладений у дужки. Коли вираз отримує значення false, рядок повідомлення після ключового слова REPORT надсилається до текстового редактора. За відсутності умовного виразу рядок повідомлення відображається безумовно.

За ключовим словом REPORT іде рядок повідомлення та додаткові параметри, представлені змінними. Рядок повідомлення береться в лапки і може містити символи %, які замінюються значеннями відповідних змінних. Якщо ключове слово REPORT не використовується, а значення арбітражного виразу false, з’являється таке повідомлення:

<validity> : рядок <1 і номер >, файл <filename>: твердження не вдалося

Додаткові змінні, включені в повідомлення, складаються з одного або кількох параметрів, функцій оцінки або арифметичних виразів. Змінні, що містяться в повідомленні, відокремлюються комами. Значення змінних підставляються в порядку появи % символів в повідомленні. У наведеному вище прикладі значення змінної WIDTH замінює символ % у рядку повідомлення.

Додаткове ключове слово SEVERITY супроводжується рівнем серйозності ERROR, WARNING або INFO. Рівень серйозності за замовчуванням – ПОМИЛКА.

ключ HELP_ID і рядок *підказки підтримуються деякими* функціями *Altera* і зарезервовані для внутрішнього використання Altera .

Оператор *Assert* закінчується символом (;).

Оператори *Assert* можна використовувати в розділі *Logic* або поза іншими розділами AHDL.

Оператор Subdesign

Розділ *Subdesign* визначає вхідні, вихідні та двонаправлені порти цього проекту.

Наступний приклад ілюструє використання розділу *Subproject* :

```
SUBDESIGN top
foo,  bar,
 clk1,  clk2   : INPUT =VCC;
aO,    al,
a2,   a3 ,
 a4              : OUTPUT;
 b[7 .0): BIDIR
```

Розділ *Subdesign* має наступні характеристики:

- за ключовим словом SUBDESIGN йде назва проекту, назва підпроекту має збігатися з назвою текстового файлу проекту, у цьому прикладі він називається top;
- перелік сигналів у круглих дужках;

сигнали представлені символьними іменами, що вказують їх тип (наприклад, INPUT);

назви сигналів відокремлюються комами. Після назви ставиться двокрапка, потім тип сигналу та символ (;);

- можливі типи портів IN, OUT, BIDIR, MACHINE IN або MACHINE OUT. У наведеному вище прикладі сигнали *foo, bar* , *clk1* і *clk2* і сигнали *a0, al, a2, aZ* і *a4* є виходами. Шина *b* [7..0] є двонаправленою;
- ключові слова MACHINE INPUT і MACHINE OUTPUT використовуються для імпорту та експорту кінцевих автоматів між текстовими файлами проекту або іншими файлами проекту, але типи портів MACHINE INPUT і MACHINE OUTPUT не можна використовувати в текстових файлах проекту верхнього рівня;

- після вказівки типу порту ви можете додатково вказати значення за замовчуванням GND або VCC (в іншому випадку значення за замовчуванням не приймаються), у наведеному вище прикладі VCC - це значення, призначене за замовчуванням для вхідних сигналів, якщо вони не використовуються у файлах вищого рівня ієрархії (призначення, зроблені в ієрархії файлів, мають вищий пріоритет)

У файлі верхнього рівня порти INPUT, OUTPUT або BIDIR є виходами пристрою. У файлах нижчого рівня всі типи портів є точками входу та виходу для даного файлу, але не для пристрою в цілому.

Розділ Variable

Розділ *змінних* використовується для опису та/або генерації змінних, що використовуються в розділі *логіки*. Змінні AHDL схожі на змінні, які використовуються в мовах високого рівня та визначають внутрішню логіку.

Наступний приклад ілюструє використання розділу *Variable* :

```
VARIABLE
a, Ь, c :       NODE;
temp  :         halfadd;
ts_node:        TRI STATE_NODE;
IF DEVICE_FAMILY     = "FLEX8000" GENERATE
kadder :    flex_adder;
d, e    :   NODE,
ELSE GENERATE
kadder :    pterm_adder;
 f, g   :   NODE;
END GENERATE;
```

Розділ *змінних* може містити такі оператори та конструкції:

- опис об'єктів;

- опис вузлів; .
- опис реєстрів;
- опис скінченних автоматів;
- опис псевдоімен скінченних автоматів.

If Generate statements , який можна використовувати для створення об’єктів, вузлів, регістрів, кінцевих автоматів і псевдонімів кінцевих автоматів.

Розділ *змінних* має такі функції:

- розділ починається з ключового слова VARIABLE;
- визначені користувачем імена символьних змінних відокремлюються комами, а від відповідного типу символом двокрапки прийнятні типи змінних: NODE, TRI STATE_NODE, <primitive>, <megafunction>, <macrofunkciton> або <state machine declaration> у у наведеному вище прикладі внутрішні змінні *a, b* і *c* мають тип NODE; temp — екземпляр макрофункції halfadd, a ts_node — об’єкт типу TRI_STATE_NODE;
- кожен рядок визначення змінної закінчується знаком (;).

У файлі з розширенням . *придатний* для поточного проекту може містити зарезервовані компілятором імена, які включають тильду (~). Символ тильди зарезервовано лише для імен, створених компілятором; забороняється використовувати для позначення виходів, вузлів і груп (шин).

Оператор Case

Оператор *Case* визначає список альтернатив, які можна активувати залежно від значення змінної, групи або виразу, наступного за ключовим словом Case.

Наступний приклад ілюструє використання оператора Case:

```
CASE f[].q IS
WHEN H"00" =>
adr[] =0;
```

```
 s = a & b;
WHEN H"01" =>
cont[].d = cont[].q+1;
WHEN H "02", H  "03", H  "04" = >
t[3..0].d   = adr  [4..1];
WHEN OTHERS   = > f[].d  =  f[].q;
END CASE;
```

Case Operator має наступні функції:

- логічні вирази, група або кінцевий автомат розміщуються між ключовими словами CASE та IS у наведеному вище прикладі це f[].q;
- оператор Case закінчується ключовими словами END CASE, за якими йде символ (;);
- вміст інструкції Case є списком однієї або кількох унікальних альтернатив, які є результатом ключового слова WHEN, причому кожній альтернативі передує ключове слово WHEN;
- кожна альтернатива є одним або декількома постійними значеннями, розділеними комами та символом (=>);
- якщо значення логічного виразу після ключового слова CASE відповідає будь-якій альтернативі у варіанті, тоді всі оператори після відповідного символу (*>) активуються, у прикладі вище, якщо f[].q дорівнює h"01 ", тоді активується вираз count[].d = число t[].q+1;
- якщо значення логічного виразу після ключового слова CASE не дорівнює жодній з альтернатив, тоді буде активована альтернативна опція після ключових слів WHEN OTHERS у наведеному вище прикладі, якщо значення f[].q не дорівнює H "00", H "0l" або H ”CF”, тоді активується вираз f[].d = f[].q;
- оператор Defaults визначає значення за замовчуванням у випадках, коли ключові слова WHEN OTHERS не використовуються;

- якщо оператор Case використовувався у визначенні переходів кінцевого автомата, тоді ви не можете використовувати ключові слова WHEN OTHERS для виходу з неприпустимих станів, якщо стани кінцевого автомата визначені за допомогою n-вимірного коду і автомат має 2n станів, тоді це прийнятно використовувати ключові слова WHEN OTHERS ; :
- кожна альтернативна змінна повинна закінчуватися символом (;).

Оператор Defaults

Оператор *Defaults* дозволяє визначати значення за замовчуванням, які використовуються в таблицях істинності, а також в операторах If then і Case. Оскільки активні високі сигнали автоматично приймають значення GND, оператор за замовчуванням необхідний лише при використанні активних низьких сигналів.

Значення за замовчуванням, призначені змінним, не слід плутати зі значеннями за замовчуванням, призначеними портам у розділі Subdesign.

У наступному прикладі показано використання оператора Defaults;

```
BEGIN
DEFAULTS
A  =  VCC;
END   DEFAULTS;
IF   y   &   z    THEN
a   =    GND; % низький %
END IF;    END;
```

Оператор Defaults має такі характеристики:

- значення за замовчуванням розміщуються між ключовими словами DEFAULTS і END DEFAULTS. Оператор закінчується символом (;);
- вміст оператора Defaults складається з одного або кількох логічних виразів, призначених константам або змінним, у наведеному вище прикладі змінній a присвоєно значення за замовчуванням VCC;

- оператор за замовчуванням активується, якщо будь-яка змінна зі списку операторів за замовчуванням виявляється невизначеною в будь-якому з операторів, у наведеному вище прикладі змінна a є невизначеною, якщо y і z мають значення логічного нуля; це активує вираз (a = VCC) у операторі Default.

При використанні оператора за замовчуванням необхідно дотримуватися таких правил:

- в розділі *Logic* дозволено використовувати не більше одного оператора Default, а також, при його використанні, він повинен розташовуватися безпосередньо після ключового слова BEGIN;
- якщо одній і тій самій змінній в операторі за замовчуванням зроблено кілька призначень, усі призначення, крім першого, ігноруються;
- Оператор Default не можна використовувати для призначення значення X (люба) змінна.

Оператор If Then

Оператор If Then містить список операторів, які будуть виконані, якщо логічний вираз між ключовими словами IF і THEN виявиться істинним.

Приклад оператора If Then :

```
IF a   [] == b   [] then
C   [8..1] =  H  "77";
        Addr  [3..l] = f  [3..l].q;
                f[].d  = addr   [] +1;
        elsif  g3   $   g4   THEN
                T  [].d = addr   [];
        else
d  =  vcc;
end  if;
```

Команда If Then має такі характеристики:

- між ключовими словами IF і THEN знаходиться логічний вираз, в залежності від значення якого буде виконуватися чи ні буде виконуватися список операторів після ключового слова THEN, кожен оператор в цьому списку закінчується символом (;);
- між ключовими словами ELSEIF і THEN міститься додатковий логічний вираз, а після ключового слова THEN — список операторів, які виконуються в залежності від значення булевого виразу. Ці додаткові ключові слова та оператори можна повторювати кілька разів;
- оператори після ключового слова THEN активуються, якщо відповідний логічний вираз істинний, наступні конструкції ELSEIF THEN ігноруються;
- ключове слово ELSEIF, за яким слідують один або більше операторів, має подібне значення до ключового слова WHEN OTHERS в операторі Case, якщо жоден із логічних виразів не має значення true, тоді виконуються оператори, наступні за ключовим словом ELSE, як у прикладі, показаному вище, якщо жоден із логічних виразів не є істинним, виконується оператор d = VCC, використання ключового слова ELSE необов'язкове;
- послідовно обчислюються значення логічних виразів, наступних за ключовими словами IF і ELSEIF;
- Оператор IF Then закінчується ключовими символами ENDIF, за якими йде символ (;).

Оператор If then може генерувати логічні схеми, надто складні для компілятора. Якщо оператор if then містить складні булеві вирази, то включення зворотного до кожного з цих виразів, швидше за все, призведе до ще більш складних булевих виразів. Наприклад, якщо a і b є складеними виразами, то звернення цих виразів може бути ще складнішим.

Оператор If:	Інтерпретація компілятором:
IF a THEN	IF a THEN
c = d;	c = d;

```
END   IF;
ELSIF   b   THEN          IF   !a  & !b  THEN
C =  E;                   C =  E;
END    IF;
ELSE                IF   !a   & !b   THEN
C = F;                    C =  F;
END  IF;                  END   IF; .
```

На відміну від оператора If Then, який може обчислювати лише значення булевих виразів, операторів IfGenerate може обчислювати значення наборів арифметичних виразів. Основна відмінність операторів IfThen і IfGenerate полягає в тому, що в першому випадку апаратно обчислюється значення булевого виразу, а в другому — на етапі компіляції обчислюється значення набору арифметичних виразів.

Оператор If Generate

Оператор *If Generate* містить список операторів, які активуються, коли арифметичний вираз отримує позитивне значення.

Оператор If Generate має такі характеристики:

- між ключовими словами Generate є арифметичний вираз, значення якого можна оцінити, ключове слово GENERATE відображає список операторів, кожен з яких закінчується коренем (;), оператори активуються, якщо арифметичний вираз обчислюється як справжнє значення;
- ключові слова ELSE GENERATE супроводжуються одним або декількома операторами, які активуються, якщо арифметичний вираз обчислюється як хибне значення;
- оператор If Generate закінчується ключовими словами END GENERATE, за якими йде символ (;);
- Оператор If Generate можна використовувати в розділі *Logic* and *Variable*

На відміну від операторів IfThen, які можуть обчислювати лише значення булевих виразів, оператори IfGenerate можуть обчислювати значення наборів арифметичних виразів. Основна відмінність між операторами If then і If Generate полягає в тому, що в першому випадку значення булевого виразу обчислюється апаратно, а в останньому випадку значення набору арифметичних виразів обчислюється під час компіляції.

Оператор If Generate особливо часто використовується з операторами For Generate для обробки особливих ситуацій, таких як молодший біт у багатоступінчастому множнику. Цей оператор також можна використовувати для перевірки значень параметрів, як показано в останньому прикладі.

Оператор For Generate

Наступний приклад ілюструє використання ітераційного оператора For Generate :

```
CONSTANT NUM_OF_ADDERS   =   8;
SUBDESIGN gent
 ( A [NUM_OF_ADDERS..1],          b   [NUM_OF ADDERS..1];
cin                              : INPUT;
c   (NUM_OF ADDERS.. l],                cout         : OUTPUT;         )
VARIABLE
carry_out     [(NUM_OF_ADDERS+l)..l]:      NODE;
BEGIN
carry_out  [1]  =  cin;
For  i          IN  1    TO NUM_ADDERS GENERATE
c[i] = a  [i]
$
        b   [i] $ carry_out[1]
        carry_out   [1] [i+l] = a   [i]
 &
        b  [ij"# carry out   [i] & (a   [i] $ b   [i]);
```

```
    END GENERATE;
    cout = carry_out  (NUM_OF_ADDERS+l ];
END;
```

Оператор For Generate має такі характеристики:

- між ключовими словами FOR і GENERATE є такі параметри:
- тимчасова змінна, яка має символічне ім’я, змінна використовується лише в операторі For Generate і припиняє своє існування після того, як компілятор обробить оператор, у наведеному вище прикладі змінна і. Це ім’я не можна використовувати як константу, параметр або назва вузла в цьому проекті;
- за ключовим словом IN слідує діапазон, обмежений двома арифметичними виразами, арифметичні вирази розділені ключовим словом TO, у наведеному вище прикладі арифметичний вираз дорівнює 1 і NUM_OF_ADDRESS, у межах діапазонів можуть бути вирази, що складаються лише з констант і параметрів; використання змінних не допускається;
- ключове слово GENERATE супроводжується одним або декількома логічними операторами, кожен з яких закінчується на (;);
- Оператор If Generate закінчується ключовими словами END GENERATE, за якими йде (;)

3.5. Розробка проектів цифрових пристроїв в AHDL

оператор або розділ AHDL до поточного файлу. Шаблони AHDL дозволяють легко вводити синтаксичні конструкції AHDL , підвищуючи швидкість і точність введення дизайнів.

Щоб вставити шаблон AHDL у поточну позицію:

1. Відкрийте діалогове вікно шаблону AHDLT за допомогою команди меню Шаблон .

2. Виберіть назву у вікні TemplateSection .

3. Натисніть OK .

Після введення шаблону в . tdf , ви повинні замінити всі змінні в шаблоні. Кожне ключове слово AHDL пишеться з великої літери, і кожне ім’я змінної починається з двох підкреслень (_ _), щоб ідентифікувати їх.

Створений текстовий вихідний файл

Ви можете створити один або кілька вихідних файлів текстового дизайну (TextDesignOutputFiles (.tdo)), що містять еквівалент AHDL повністю оптимальної логіки для пристрою, який використовується в проекті. Крім того, компілятор також створює один або кілька вихідних файлів призначення та конфігурації (Assignment & ConfigurationOutputFiles (.aco)), збережіть . tdo як текстовий файл відредагованого проекту, визначте його як проект за допомогою команд меню ProjectName або ProjectSetProjectto поточний файл і перекомпілюйте проект.

Створити файлу проекту необхідно:

1. Увімкніть Generate AHDL . зробити Файл у команді меню Процес .

2. Запустіть збірку або одну з команд у меню «Файл» : «Проект». Збережіть і побудуйте або спроектуйте Зберігайте , компілюйте та симулюйте в будь-якій програмі MAX + PLUSII.

Числа використовуються для представлення постійних значень у логічних і арифметичних виразах, рівняннях і значеннях параметрів. *AHDL* підтримує всі комбінації десяткових, двійкових, вісімкових і шістнадцяткових чисел.

Файл *Dec. _ Tdf* нижче описує декодер адреси, який генерує активний сигнал високого рівня, коли адреса має значення 370 Hex .

```
SUBDESIGN dec
(               address [15..0]   :   INPUT;
                chip_enable       :   OUTPUT;        )
BEGIN
```

```
chip =  (address  [ 15..0 ]  ==  H  ” 0370 ”);
END;
```

У цьому простому прикладі десяткові числа 15 і 0 використовуються для визначення бітів шини адреси. Шістнадцяткове число H "0370" визначає розшифровану адресу.

Константи та функції обчислення особливо корисні, коли одне й те саме число, рядок або арифметичний вираз повторюється кілька разів у файлі: якщо воно змінюється, вам потрібно змінити лише один оператор. У AHDL константи реалізуються за допомогою оператора Constant , а функції обчислюються за допомогою оператора Define .

AHDL містить функції USED , CEIL і FLOOR .

нижче файл dec 2.tdf має ті самі функції, що й dec . tdf , але використовує константу IO_ADDRESS замість числа H "0370" .

```
Constant IO_ADDRESS= H"0370";
SUBDESIGN  dec
(          A  [15..0]  :  INPUT;
Ce  :          OUTPUT;          )
BEGIN
CE = (a  []  ==  IO_ADDRESS);
END;
```

Комбінаційна логіка

Комбінаційна логіка реалізована в AHDL за допомогою виразів логічних рівнянь, мегатаблиць істинності та макрофункцій. Приклади комбінаційних функцій включають декодери, мультиплексори та суматори.

Реалізація виразів і логічних рівнянь

Логічні вирази — це набір вузлів, чисел, констант, розділених операторами та/або додатково згрупованих дужками. Логічне рівняння встановлює вузол або шину рівними значенню виразу.

Наступний tdf демонструє два прості булеві вирази, що представляють два логічні блоки.

```
SUBDESIGN bool
(       A 0,    a 1,    b    :                  INPUT;
Out 1, out 2          :          OUTPUT;)
BEGIN
Out  2   =   out1  #   b;
Out  1   =    a1   &   ! a0;
END;
```

У цьому файлі out1 є логічним І вхідних даних a1 та інвертує a 0, а вихід out2 є логічним АБО вихід1 і b . Порядок, у якому вони передаються у файлі, не має значення.

Ви можете назвати логічні оператори у файлі звіту про проект, щоб полегшити введення призначень ресурсів та інтерпретацію рівнянь.

b файл 3.tdf *ідентичний логічному файлу 1.tdf* , але використовує іменовані оператори . Ім'я оператора відокремлюється від оператора двокрапкою: ім'я може містити до 32 символів.

```
SUBDESIGN 22
(                  A0, A1, B: INPUT;
OUT1, OUT2 : OUTPUT;        )
BEGIN
OUT1 = A1  TIGER:   &   !  A0;
OUT2 = OUT1 PANTHER:    # B;
END;
```

Оголошення вузлів

Вузол, який відображається з оголошенням вузла в розділі Variable , можна використовувати для зберігання значення проміжного виразу. Оголошення вузлів особливо корисні, коли логічні вирази використовуються повторно.

файл b 2.tdf містить ту саму логіку, що й файл bool. tdf , але має лише один вихід.

```
SUBDESIGN b 2
( A0,    A1,    B   : INPUT;
OUT              :     OUTPUT;
)
Variable
A_EQUALS_2 :   NODE;
BEGIN
A_EQUALS_2 = A1 & !A0;
OUT2 = A_EQUALS_2 # B;
END;
```

Цей файл оголошує вузол *a_equals_2* і прив’язує його до виразу *a1&! i 0*. Використання вузлів може заощадити ресурси пристрою, коли вузол використовується в кількох виразах.

Присвоєння вузлам типу *Node* об'єднати сигнали за допомогою функцій І та АБО.

Визначення шин

Шина може містити до 256 біт, інтерпретується як набір вузлів і працює як одне ціле. Ім’я шини можна вказати як ім’я з одним діапазоном, ім’я з подвійним діапазоном або ім’я в послідовному форматі.

У логічних рівняннях шина може дорівнювати mule, іншій шині, одному вузлу, VCC , GND , 1 або 0. У кожному з цих випадків значення шини різні.

Оператор Option можна використовувати, щоб вказати, чи буде молодший біт старшим (MSB), молодшим значущим бітом (LSB) чи чимось іншим.

Коли шину визначено, дужки [] є скороченим способом визначення всього діапазону. Наприклад, a[4..1] також можна вказати як a[]; b [5..4][3..2] можна представити як b [][].

груповий файл 1.tdf показує логічні вирази, які визначають кілька шин.

```
options bit0 = msb;
constant max_width = 1+2+3-3-1;
%   max_width  =  2%
SUBDESIGN  gr
( a[1..2],
 use_exp_in [1+2-2.. MAX_WIDTH]   :   INPUT;
d[1..2],
use_exp_out [1+2*2-4.. MAX_WIDTH] :   OUTPUT;
dual_range  [5..4][3..2]  :     OUTPUT; )
BEGIN
D [] =   A  [] + B  ”10”;
USE_EXP_OUT  [] = USE_EXP_IN   [];
DUAL_RANGE   [][]  =  VCC;
END;
```

У даному прикладі оператор OPTIONS використовується для визначення того, що крайній правий біт шини буде MSB , а до шини a[додається десяткова 1. Шини use _ exp _ in і use _ exp _ out показують, як константи і арифметичні вирази можна використовувати для обмеження діапазонів шини.

Наступні приклади ілюструють використання шин:

- коли шина порівнюється з шиною такого ж розміру, кожен елемент з правого боку дорівнює кожному елементу з лівого боку у відповідній позиції;

- коли шина прив’язана до VCC або GND, усі біти шини прив’язані до цих значень;

- коли для шини встановлено значення 1, лише молодший біт шини підключається до значення VCC. Інші біти шини підключені до GND;

- при вирівнюванні шин різних розмірів кількість бітів шини в лівій частині рівняння має точно ділитися на кількість бітів шини в правій частині рівняння.

Наприклад, рівняння

i [4..1] = b[2..1] є правильним.

У цьому рівнянні біти представлені таким чином:

A4 = B2 *A3 = B1*

A2 = B2 *A1 = B1*

Реалізація умовної логіки

Оператори If Then і Case використовуються для реалізації умовної логіки. Оператор If Потім він обчислює один або кілька логічних виразів і описує поведінку для різних значень виразу. Оператор Case — це список альтернатив, доступних для кожного значення виразу. Вони оцінюють вирази, а потім обирають лінію поведінки на основі значень виразів.

Умовна логіка, реалізована за допомогою оператора If . Не плутайте this і Case з логікою, створеною оператором *If* GENERATE . Ця логіка не є обов’язковою, це умова.

Файл, пріоритет . Наступний tdf показує кодувальник пріоритетів, який перетворює активний рівень введення найвищого пріоритету на значення виразу.

```
SUBDESIGN prek
( low,
      middle,
      high    : INPUT;
```

```
        highest_  level[1..0]   : OUTPUT; )
    BEGIN
        IF high THEN
            HIGHEST_LEVEL   [] = 3;
        ELSIF middle THEN
            HIGHEST_LEVEL   [] = 2;
        ELSIF low THEN
            HIGHEST_LEVEL    [] = 1;
    ELSE
            HIGHEST_LEVEL   [] = 0;
    END IF;
    END;
```

У цьому прикладі входи *є високими, середніми* і *низький* оцінюються, щоб визначити, чи дорівнюють їхні рівні *VCC. Оператор If він активує рівняння, яке слідує за активними полями If або ELSE і якщо вхідний сигнал високий* високий, тоді верхній_рівень [ ] дорівнює 3.

Якщо активовано більше ніж один вхід, то оператор *If Потім він оцінює пріоритет вхідних даних у порядку проходження через області If і* ELSIF (перша область має найвищий пріоритет).

буде викликано рівняння після ключового слова ELSE .

Файл, . tdf нижче описує 2-4-бітовий декодер . Перетворює 2-бітовий код в унарний код.

```
SUBDESIGN dec
( code[1..0] :              INPUT;
out[3..0] :          OUTPUT;    )
BEGIN
CASE code[] IS
     WHEN 3 = > out [] = B ”1000”;
```

```
        WHEN 2 = > out [] = B ”0100”;
        WHEN 1 = > out [] = B ”0010”;
        WHEN 0 = > out [] = B ”0001”;
    END CASE;
END;
```

У цьому прикладі код вхідної шини дорівнює 0, 1, 2 або 3. У операторі CASE після => йде активне рівняння. Наприклад, якщо *код []* дорівнює 1, то вихід 1 встановлюється на B "0010". Оскільки всі значення виразів різні, одночасно можна активувати лише одне поле WHEN .

Якщо Оператори *If Then і Case* подібні. У деяких випадках ви можете використовувати один із двох операторів, щоб отримати той самий результат.

Але між ними є важлива відмінність:

- будь-який вид логічного виразу можна використовувати в операторі If then, будь-який вираз після полів If або ELSIF може не бути пов’язаним з іншими виразами в операторі. Навпаки, в операторі Case лише один логічний вираз порівнюється з константою в кожній області WHEN ;
- використання ELSIF може призвести до надто складної логіки для компілятора, оскільки кожен наступний оператор ELSIF все одно повинен перевіряти, чи попередні оператори If/ ELSIF є хибними.

Створення дешифраторів

У AHDL ви можете використовувати табличні функції оператора істинності або pin_compare або lpm_decode для створення декодера .

Сегмент файлу. Наведений нижче *tdf* є комбінованим декодером світлодіодів (LED), світлодіоди відображають шістнадцяткові числа.

```
SUBDESIGN seg
(
I  [3..0]   :                  INPUT;
A, B, C, D, E, F, G    :            OUTPUT;
```

```
)
BEGIN
TABLE
        i[3..0] = > A, B, C, D, E, F, G;
        H”0” = >1, 1, 1, 1, 1, 1, 0;
        H”1”  = >0, 1, 1, 0, 0, 0, 0;
        H”2”  = >1, 1, 0, 0, 1, 1, 0;
        H”3”  = >1, 1, 1, 1, 0, 0, 1;
        H”4”  = >0, 1, 1, 0, 0, 1, 1;
        H”5”  = >1, 0, 1, 1, 0, 1, 1;
        H”6”  = >1, 0, 1, 1, 1, 1, 1;
        H”7”  = >1, 1, 1, 0, 0, 0, 0;
        H”8”  = >1, 1, 1, 1, 1, 1, 1;
        H”9”  = >1, 1, 1, 1, 0, 1, 1;
        H”A” = >1, 1, 1, 0, 1, 1, 1;
        H”B” = >0, 0, 1, 1, 1, 1, 1;
        H”C” = >1, 0, 0, 1, 1, 1, 0;
        H”D” = >0, 1, 1, 1, 1, 0, 1;
        H”E” = >1, 0, 0, 1, 1, 1, 1;
        H”F” = >1, 0, 0, 0, 1, 1, 1;
    END TABLE;
END;
```

У цьому прикладі вихідний набір для всіх 16 можливих вхідних наборів i [3..0] описано в операторі *TruthTable* .

файл dec3.*tdf* є декодером адреси для *реалізації 16-розрядної мікропроцесорної системи* .

```
SUBDESIGN dec3
(ADDR  [15..0],   M/IO     :  INPUT;
ROM,  RAM,  PRINT,   SP   [2..1]   :   OUTPUT;   )
```

```
BEGIN
TABLE
m/io,    addr [15..0]                    = >  rom,  ram,  print,   sp [];
 B ”00XXXXXXXXXXXXXX” = >1, 0, 0,  B”00”;
 B ”100XXXXXXXXXXXXX”  = >   0,      1,       0,     B”00”;
 B ”0000001010101110”          = >   0,      0,       1,     B”00”;
 B ”0000001011011110”          = >   0,      0,       0,     B”01”;
 B ”0000001101110000”          = >   0,      0,       0,     B”10”;
END TABLE;
END;
```

У цьому прикладі є тисячі наборів вхідних даних і описують їх у операторі *Truth Table* непрактично Замість цього можна використовувати логічний рівень *X* , щоб вказати, що вихід не залежить від відповідного входу. Наприклад, у першому рядку оператора TABLE вихідний сигнал rom має бути високим для всіх наборів вхідних даних, адреса [15..0], починаючи з 00. Тому слід ретельно визначити лише перетин набору вхідних даних, а X символ слід використовувати для інших вхідних даних.

Використовуючи символи X, переконайтеся, що комбінації бітів у таблиці істинності не перекриваються. AHDL припускає, що лише одна умова в таблиці істинності може бути істинною в будь-який момент часу .

Використовуйте значення за замовчуванням для змінних

Можна визначити значення за замовчуванням для використовуваного вузла або шини, якщо його значення не вказано в іншому місці файлу. AHDL дозволяє призначати значення вузлу або шині більше одного разу в одному файлі. Якщо ці призначення конфліктують, для вирішення конфліктів використовуються значення за замовчуванням. Якщо значення за умовчанням не визначено, йому присвоюється значення GND .

Значення за замовчуванням визначається за допомогою оператора Defaults для змінних, що використовуються в операторах *Truth Table, If Then i Case.*

Не плутайте значення змінних за замовчуванням зі значеннями порту за замовчуванням, призначеними в розділі Subproject .

Файл 1.tdf за замовчуванням нижче оцінює вхідні дані та вибирає один із п’яти кодів ASCII на основі вхідних даних.

```
SUBDESIGN    1
(       I  [3..0]          :          INPUT;
ascii_code  [7..0]   :          OUTPUT;   )
BEGIN
DEFAULTS
ASCII_CODE[] = I’00111111’;   %  ASCII КОД “?”   %
END DEFAULTS;
TABLE
I [3..0] = > ascii_code[];
B “1000”    = >    B”01100001”;
B “0100”    = >    B”01100010”;
B “0010”    = >    B”01100011”;
B “0001”    = >    B”01100100”;
END TABLE;
END;
```

Коли вхідний набір збігається з одним із наборів, наведених у лівій частині оператора *таблиці істинності,* вихід встановлюється відповідно до комбінації в правій частині. Якщо відповідності немає, вихідні дані за замовчуванням мають B ' 00111111'.

файл 2.tdf за замовчуванням ілюструє, як виникають конфлікти, коли одному вузлу присвоюється більше ніж одне значення, і як AHDL вирішує ці конфлікти.

```
SUBDESIGN  2
(
a,  b,  c  :  INPUT;
select_b,  select_c  select_a,  :  INPUT;
wire_and  wire_or,  :  OUTPUT;
)
BEGIN
DEFAULTS        wire_or  =  GND;
wire_and  =  VCC;       END DEFAULTS;
IF  select_a  THEN
WIRE_OR  =  A;          WIRE_AND  =  A;
END  IF;
IF select  _b  THEN
WIRE_OR  =  B;          WIRE_AND  =  B;
END  IF;
IF  select_c  THEN
        WIRE_OR  =  C;
        WIRE_AND  =  C;
END  IF;
END;
```

В цьому прикладі *wire_or призначається a ,* b *або* c *,* залежно *від* сигналів Select_a *, Select_b і Select_c* . Якщо жоден із цих сигналів не дорівнює *VCC* , тоді *провід _ або* встановлюється на GND .

Якщо кілька сигналів *select _ a* , *select _ b* або *select _ c* є *VCC* , тоді *сигнал або* є логічним АБО відповідних вхідних значень.

Сигнал *wire_and* сигнал працюють однаково, за винятком того, що вони за замовчуванням мають значення *VCC* , коли жоден із *вибраних* сигналів не дорівнює *VCC* , і логічно позначаються "I" відповідними входами, коли більше одного сигналу є *VCC* .

Логіка з активними низькими рівнями

Активний низький сигнал стає активним, коли його значення дорівнює GND. Активні низькі сигнали можуть бути корисними для керування пам'яттю, периферійними пристроями та чіпами мікропроцесорів.

Файл *daisy.tdf.* є модулем схеми арбітра. Він приймає запити доступу до шини від себе та від наступного модуля в ланцюжку. Доступ до шини здійснюється модулем з найвищим пріоритетом, який її запитав.

```
SUBDESIGN daisy
(
request _out    :   OUTPUT;  %до старшого пріоритету%
grant_out       :   OUTPUT;      % до молодшого пріоритету %
local_request   :   INPUT;
request _in     :   INPUT;
grant_ in       :   INPUT;
)
BEGIN
DEFAULTS
local_grant  =  VCC;
 request _out =      VCC;
grant_out = VCC;
END DEFAULTS;
IF  REQUEST _IN  ==  GND # LOCAL_REQUEST  ==  GND THEN
 request _out  =  GND;
END IF;
        IF  GRANT_ IN  ==  GND THEN
```

```
        IF LOCAL_REQUEST ==   GND THEN
        Local_grant  =  GND;
ELSIF  request _in  ==  GND THEN
        grant_out   =   GND;
END  IF;
END;
```

Усі сигнали в цьому файлі мають активний низький рівень. *Altera* рекомендує вибрати схему іменування вузлів, яка чітко вказує назви активних дірок, наприклад, із « n » на початку .

Якщо Оператори *If Then* визначається активність модулів, тобто чи дорівнює сигнал GND . Якщо сигнал активний, активуються рівняння після відповідного оператора *If Then*.

Реалізація двонаправлених виходів

MAX+PLUS II дозволяє налаштувати контакти введення/виведення як двонаправлені. Двонаправлені виходи можна визначити за допомогою порту BIDIR , який підключається до виходу примітиву TRI . Сигнал між контактом і примітивом TRI є двонаправленим і може використовуватися для керування іншою логікою у вашому проекті.

Двонаправлений сигнал введення/виведення, керований примітивом TRI , використовується як вхід *d* тригера D (DFF).

Також можна підключити двонаправлений вихід із файлу TDF нижчого рівня до виводу вищого рівня. Двонаправлений вихідний порт підпроекту має підключатися до двонаправленого вихідного порту верхнього рівня ієрархії. Прототип функції для низькорівневого TDF- файлу повинен містити двонаправлений вихід у реченні RETUTNS . Файл *Bidir . Наступний tdf містить чотири екземпляри згаданої вище* функції bus 2. *tdf* .

```
FUNCTION   bus2   ( clk,   oe )
```

```
RETURNS ( io );
SUBDESIGN    bidir
(    clk,   oe    :    INPUT;
io [3..0]    :   BIDIR;    )
BEGIN
 IO 0  =     BUS_REG2 (CLK, OE);
 IO 1  =     BUS_REG2 (CLK, OE);
 IO 2  =     BUS_REG2 (CLK, OE);
 IO 3  =     BUS_REG2 (CLK, OE);
END;
```

Реалізація тристабільних шин

Примітиви TRI , що керують портами OUTPUT або BIDIR , мають вхід дозволу виходу (OUTPUT Enable), який переводить вихід у стан високого імпедансу.

Створити тристабільну шину , з'єднавши примітиви TRI та порти OUTPUT або BIDIR разом за допомогою вузла TRI_STATE_NODE . Схема управління не повинна допускати більше одного виходу одночасно.

Файли *tri tdf* реалізує тристабільну шину за допомогою вузла TRI_S TATE_NODE, створеного в декларації *Node.*

```
SUBDESIGN     tri
(  in[3..1],
oe[3..1]    :    INPUT;
out1                        :          OUTPUT;  )
VARIABLE
 tnode :   TRI _ NODE;
BEGIN
tnode    =    TRI(in3, oe3);
tnode   =    TRI(in2, oe2);
```

```
tnode   =   TRI(in1, oe1);
out1  =    tnode;
END;
```

У цьому прикладі кілька призначень вузлів пов’язують сигнали разом. Для реалізації трьох стабільних шин потрібен тип TRI_STATE_NODE замість типу NODE , для типу NODE сигнали підключаються за допомогою дроту «AND » або дроту «OR», а для типу TRI_STATE_NODE сигнали підключаються з таким же вузлом. Однак, якщо лише одна змінна призначена вузлу TRI_S TATE_NODE, вона інтерпретується як звичайна змінна NODE.

3.6. Послідовна логіка

Послідовну логіку в AHDL можна реалізувати за допомогою кінцевих автоматів і регістрів або за допомогою бібліотеки параметричних модулів (LPM). Кінцеві автомати особливо зручні для реалізації послідовної логіки. Іншими прикладами є лічильники та контролери.

Оголошення рекордів

Регістри зберігають значення даних і синхронізують їх за допомогою тактового сигналу . Ви можете оголосити екземпляр реєстру за допомогою оголошення реєстру в розділі Variable (ви також можете реалізувати реєстр за допомогою посилань на функції в розділі *Logic*). AHDL пропонує деякі примітивні регістри, а також підтримує функції регістрів LPM .

Після оголошення реєстру його можна підключити до іншої логіки у файлі TDF за допомогою його портів. Порт примірника використовується в такому форматі:

"ім'я екземпляра" "ім'я порту"

Файл reg *tdf* використовує оголошення *Register* для створення байтового регістра, який фіксує значення входів *d* на передньому фронті *Clock* , коли вхідне навантаження є високим.

```
SUBDESIGN    registr
(     clk,   load,    d  [7..0]   :     INPUT;
q  [7..0]                :       OUTPUT;      )
VARIABLE
ff   [7..0]   :   DFFE;
BEGIN
FF []. CLK   =   CLK;
FF   []. ENA   =   LOAD;
FF []. D   =   D[];
Q []   =   ff[]. q;
END;
```

Регістри відображаються в розділі *VARIABLE* як DFFE. Перше логічне рівняння в розділі *«Логіка»* . підключіть вхід *clk* з портами *clk* запускає *ff* [7..0]. Друге рівняння з'єднує вхід навантаження з портами ввімкнення синхронізації. Третє рівняння з'єднує входи даних *d* [7..0] із вхідними портами тригера *ff* [7..0]. Четверте рівняння з'єднує виходи з вихідними портами тригерів. Усі чотири рівняння оцінюються разом.

Також можете оголосити тригери T , JK і SR у розділі *Variable* , а потім використовувати їх у розділі *Logic* .

Файл *lpm_rej . tdf* нижче містить посилання на реалізацію екземпляра функції *lpm_dff* , яка має ті самі функції, що й файл *reg . tdf*

```
INCLUDE   "lpm_dff.inc"
SUBDESIGN    reg
```

```
(      clk, load, d [7..0]   :    INPUT;
q [7..0]    :   OUTPUT;     )
BEGIN
Q [] =   LPM_DFF   (.CLOCK  =  CLK, ENABLE  =  LOAD,  DATA []= D
[])
   WITH  (LPM_WIDTH=8)
   RETURNS   (.q[]);
END;
```

Оголошення результатів реєстру

Вихідні дані реєстру файлів TDF , оголосивши вихідні порти як тригери в розділі *Variable* . файл . *_tdf нижче має таку* саму функціональність, як і *файл bur_reg* . *tdf* , але має зареєстрований вихід.

```
SUBDESIGN out
(      clk,  load, d [7..0] :       INPUT;
q [7..0]  :                         OUTPUT;         )
VARIABLE
Q [7..0]  : DFFE;
BEGIN
Q [].ena = load;
Q [] = d[];
Q [].clk = clk;
END;
```

Після призначення значення виходам регістру в розділі *Logic* значення з *d* входів надсилається до регістру. Виходи регістру не змінюються, доки не з'явиться наростаючий фронт *тактового сигналу* . Щоб вказати вхідний

сигнал годинника регістру, використовуйте конструкцію " ім'я регістра " .clk у розділі *"Логіка"* .

Реалізувати глобальний тактовий сигнал *за допомогою* примітиву GLOBAL з параметром *Global boolean Сигнал* у діалоговому вікні *Індивідуальна особа Логіка Опція* або використання *автоматичного Глобальний Годинник* із *глобального діалогового вікна Дизайн Логіка Синтез* (меню «Призначити »).

У наведеному вище файлі кожен тригер DFFF , оголошений у розділі *Variable* , запитує вивід із тим самим іменем, тому ви можете посилатися на виходи *q* запускає без використання порту *q* . У файлі TDF високого рівня вихідні порти синхронізовані з вихідними контактами.

Розробка лічильника

Лічильник можна визначити за допомогою тригерів D (DFF і DFFF) і оператора *If Потім або за допомогою* функції *lpm_counter* .

файл cou . *tdf* реалізує 16-розрядний суматор із навантаженням, яке можна скинути до нуля.

```
SUBDESIGN counter
(     clk, load, ena, clr, d [15..0]  :   INPUT;
q [15..0]   :   OUTPUT;    )
VARIABLE
COUNT  [15..0]  :  DFF;
BEGIN
Count [].clrn  =  !clr;
Count [].clk  =  clk;
IF  load  THEN
            Count  [].d =   d[];
      ELSIF   ena   THEN
            Count  []. d =   count   [].q + 1;
```

```
        ELSE
                Count   [].d  =  count  [].q;
    END  IF;
    END;
```

У цьому файлі *є* 16 тригерів із іменами від 0 до 15 у розділі змінних . Оператор *If* визначає значення, що завантажується в тригери під час реалізації тієї ж функції, що й файл *cou . tdf*

```
    INCLUDE “counter.inc”;
    SUBDESIGN   cnt
    (   CLK,   LOAD, ENA, CLR, D  [15..0]  :   INPUT;
    Q  [15..0] : OUTPUT;     )
    VARIABLE
    MY_CNTR:  LPM_COUNTER WITH   (LPM_WITDH  =  16);
    BEGIN
        my_cntr.cnt_en   =   ena;
        my_cntr.aclr   =   clr;
        my_cntr.clock   =   clk;
        q  []  =  my_cntr.q  [];
        my_cntr.aload   =    load;
        my_cntr.data[]   =   d[];
    END;
```

Скінченні автомати

У AHDL скінченні автомати також можна легко реалізувати як таблиці істинності в булевих рівняннях. Мова побудована таким чином, що можна самостійно призначати значення станам або дозволити компілятору MAX + PLUS II зробити це самостійно.

Компілятор використовує розширену евристику для автоматичного призначення станів, що мінімізує логічні ресурси, необхідні для реалізації кінцевих автоматів. Для цього просто намалюйте діаграму станів і побудуйте таблицю наступних станів. Компілятор автоматично виконає такі дії:

● призначає біти, вибираючи тригер T або D (TFF або DFF) для кожного біта;

● присвоює значення станам;

● використовує техніку складного логічного синтезу для отримання рівнянь збудження.

Щоб визначити кінцевий автомат у AHDL , додайте до файлу TDF наступне :

● анонс кінцевого автомата (розділ Змінні);

● Булеві рівняння керування (логічний розділ);

● переходи між станами в *таблиці* або списку *випадків* (розділ Логіка).

Також можна імпортувати та експортувати кінцеві автомати між файлами TDF та іншими файлами проекту, визначаючи входи та виходи як порти машини в розділі *Subproject* .

Реалізація скінченних автоматів

Можна створити кінцевий автомат, оголосивши його назву, стани та біти в декларації кінцевого автомата в розділі *змінних* .

Простий файл . *tdf* нижче має ту саму функціональність, що й тригер D (DFF)

```
SUBDESIGN   simple
(       Clk,   Reset,   D     :   INPUT;
Q   :   OUTPUT;       )
VARIABLE
        ss   :   MACHINE WITH STATES (s0, s1);
BEGIN
```

```
        ss.reset  =  reset;
        ss.clk  =  clk;
CASE  ss  IS
C s0  = >  q = GND;
IF  d  THEN
            Ss  =  s1;
END  IF;
WHEN  s1 = >
            q  =  VCC;
IF  !d  THEN
            ss  =  s0;
END  IF;
END  CASE;
END;
```

У файлі в розділі *Variable оголошено* кінцевий автомат з назвою *ss* . Стани автомата визначаються як *s 0* і *s 1* , а біти стану не оголошуються.

Переходи кінцевого автомата визначають умови для зміни стану. Щоб задати переходи кінцевого автомата, необхідно умовно призначити стан в рамках однієї поведінкової структури. Для цієї мети рекомендуються оператори *таблиці* або *регістру* . Наприклад, у *простих . tdf* переходи з кожного стану визначаються оператором *Case* .

Виведення стану також можна визначити за допомогою оператора *If Тоді* або *Кейс* . У свідченнях *Кейс* ці призначення використовуються в реченнях WHEN . Наприклад, у *простих . tdf* виходу *q* присвоюється значення GND , коли кінцевий автомат *ss* знаходиться на *s 0* і VCC , коли машина знаходиться на *s 1* . Вихідні значення також можна визначити в таблицях істинності.

3.7. Реалізація ієрархічних проектів

TDF , написаний у AHDL , можна змішувати з іншими файлами в ієрархії проекту. Файли низького рівня можуть бути файлами, наданими Altera , або визначеними користувачем мега- та макро-функціями.

Використання непараметричних функцій

MAX+PLUS II містить бібліотеки примітивів і непараметричних макрофункцій. Усі логічні функції MAX + PLUS II можна використовувати для створення ієрархічних структур. Мега- та макрофункції автоматично встановлюються в підкаталоги каталогу \ *maxplus 2\ max 2 lib* , створеного під час встановлення. Примітивна логіка вбудована в AHDL .

Існує два способи використання (тобто вставлення екземпляра) непараметричної функції в AHDL :

● оголосити змінну для функції, тобто назву екземпляра, у розділі *Variable* оголошення екземпляра та використовувати порти екземпляра в розділі Logic ;

● використовуйте посилання на логічну функцію в розділі LogicTDF файлу.

Входи та виходи мега- та макрофункцій необхідно вказувати за допомогою оператора функції (Function прототип). Функції прототипу не потрібні для примітивів. МАКС + ПЛЮС II містить включені файли (Include файли), з прототипу всіх мега- та макрофункцій MAX + PLUS II *у каталогах* \ *maxplus 2\ max 2 lib* \ *mega_lpm* і \ *maxplus 2\ max 2 inc* відповідно . Використовуючи оператор *Include* , можна перенести вміст файлу *Include* у файл TDF , щоб оголосити прототип мега- та макрофункцій MAX + PLUSII .

Файл *макросу tdf-файлі* показано 4-бітовий лічильник, підключений до декодера 4 на 16. Ці функції створюються за допомогою оголошень *екземплярів* у розділі *Variable* .

```
INCLUDE    “16dmux”;
INCLUDE     “4count”;
SUBDESIGN    macro
(      clk     :    INPUT;
 Out   [15..0]    :    OUTPUT;       )
VARIABLE
        COUNTER: 4COUNT;
        DECODER: 16DMUX;
BEGIN
        decoder.   (d,c,b,a) = counter.  (qd,qc,qb,qa);
        out[15..0] = decoder.q  [15..0];
        counter.clk =  clk;
        counter.dnup = GND;
END;
```

Файл використовує оператори *Include* для імпорту прототипів функцій для двох макрофункцій: 4count і 16dmux . У розділі *змінних* змінна *лічильника r* оголошено як екземпляр функції 4 count, а *декодер змінної* оголошено як екземпляр функції 16 dmux. Вхідні порти функцій у форматі <ім’я екземпляра> та <ім’я порту> визначені в лівій частині логічних рівнянь у розділі «Логіка» , а вихідні порти — у правій частині.

Посилання на *4count.inc* використовує комунікацію порт-елемент під час посилання *16 dmux. Inc* використовує зв'язок за назвою порту. Вхідні порти обох макрофункцій визначені праворуч від посилання, а вхідні порти – ліворуч.

У коментарях є еквівалентні посилання для різних типів зв’язку через порт. У посиланні порти праворуч від знака рівності (=) можуть бути перераховані за допомогою позиційного посилання або імені порту. Порти в лівій частині рівняння завжди використовують позиційний зв'язок. При використанні позиційного зв’язку порядок портів важливий, оскільки існує

однозначна відповідність між порядком портів у прототипі функції та портами, визначеними в розділі «Логіка » .

Пропозиція RETURNS необов’язкова для посилання. RETURNS можна використовувати для перерахування підмножини результатів функцій, які використовуються в екземплярі.

Примітиви та макрофункції завжди мають значення за замовчуванням для неприкріплених входів, і навпаки, мегафункції не обов’язково мають їх.

Використання параметричних функцій

MAX + PLUSII включає параметричні мегафункції та функції з бібліотеки параметричних модулів (LPM). Наприклад, параметри використовуються для вказівки ширини порту, незалежно від того, реалізований блок оперативної пам’яті як синхронний чи асинхронний. Параметричні функції можуть містити інші підпроекти, які, у свою чергу, можуть бути параметричними або непараметричними. Параметри можна використовувати з деякими непараметричними макрофункціями. Усі логічні функції можна використовувати для створення ієрархічних структур. Мега- та макрофункції автоматично встановлюються в підкаталоги \ *maxplus 2\ max 2 lib* , створені під час встановлення, примітивна логіка вбудована в мову AHDL .

Доступ до параметричних функцій можна отримати за допомогою посилання на функцію або оголошення *екземпляра* так само, як і до непараметричних функцій, але з кількома додатковими кроками:

● екземпляр логічної функції повинен містити пропозицію WITH , яка базується на реченні WITH у прототипі функції, де перераховано параметри, які використовує екземпляр . параметричне значення має бути прив’язане десь у проекті, якщо сам екземпляр виконує не містять деяких або всіх значень необхідних параметрів, компілятор шукає їх у порядку значень параметрів;

● оскільки параметричні функції не обов'язково мають початкові значення для непідключених входів, переконайтеся, що всі необхідні порти

підключені. З іншого боку, примітиви та макрофункції завжди мають початкові значення для неприєднаних входів.

Нижче *файл lpm_add 1.tdf реалізує 8-бітовий суматор* за допомогою посилання *на параметричну мегафункцію lpm_add_sub* .

```
INCLUDE   “lpm_add_sub.inc”
SUBDESIGN   lpm_add1
(         a [8..1],    b [8..1]         :    INPUT;
          C    [8..1]          :          OUTPUT;
          carry_out           :          OUTPUT;                )
BEGIN
%Екземпляр мегафункції зі зв’язком порту по по положенню%
(c[],carry_out) lpm_add_sub  (GND,   a  [], b [],   GND)
                    WITH (LPM_WIDTH  =  8,
LPM_REPRESENTATION   =   “unsigned”);
%Еквівалентний екземпляр зі зв’язком по імені%
-- (c[],carry_out)  lpm_add_sub(.data a [] = a [], data b [] = b [],
-- cin  =  GND,    add_sub  =  GND)
-- WITH (LPM_WIDTH  =  8,
LPM_ REPRESENTATION    =    “unsigned”);
END;
```

РОЗДІЛ IV. РОБОТА З ГРАФІЧНИМ РЕДАКТОРОМ

Вибравши графічний редактор, відкриваємо новий файл. Подвійним клацанням у вікні редактора викликаємо бібліотеку. Спочатку краще працювати з бібліотекою mf, яка містить стандартні елементи серії 74XX. Схему довільного елемента з бібліотеки можна подивитися, зробивши подвійне клацання на вибраному елементі. Крім того, можна переглянути довідник, виконавши такі операції:

- виділити потрібний елемент клацанням "миші";
- клацнути по ньому правою клавішею "миші";
- вибрати в меню опцію Edit Ports/Parameters;
- вибрати Help on.

У цьому довіднику є прототип функції текстового редактора і таблиця істинності для вибраного елемента.

При роботі з мегафункцією необхідно виконати такі операції:

- викликати потрібний символ із бібліотеки мегафункцій;
- правою клавішею "миші" клацнути всередині символу;
- виставити необхідні параметри, скасувати зайві входи.

Графічний редактор Altera схожий редактор PaintBrush. Але багато операцій мають свої особливості.

Проведення шин виконується через команду Options/LineStyle із процесом вибором вигляду сполучної лінії. Ідентифікація шини проводиться латиницею в режимі покажчика (в лівому інструментальному полі), проте краще це робити в текстовому тедакторі, помістивши маркер у тексті входу/виходу або над лінією, що маркується. Наприклад, rg [12..1] , де 12 і 1 - це старший та молодший індекси зв'язків. Позначення окремого зв'язку в шині складається з імені шини та її індексу, наприклад rg [1]. Лінії, що йдуть від входів/виходів, можна не позначати, оскільки їм автоматично присвоюються імена входів/виходів.

Входи та виходи задаються за допомогою примітивів, які викликаються подвійним клацанням. Вони також мають бути названі.

Крапка на шині або зв'язку ставиться за допомогою інструментальної лінійки зліва.

Зв'язки краще проводити за допомогою інструмента у вигляді прямого кута.

За допомогою примітивів можна вводити GND та VCC ("землю" та живлення). Примітиви викликаються подвійним клацанням лівої кнопки "миші".

Обертання елемента виконується командою Rotate, що вибирається з меню, яке з'являється після клацання правою клавішею "миші" (контекстне меню). Найпростіше виконати команду обертання з опції Edit.

Роздрукування схем із графічного редактора:

- зменшити до краю масштаб зображення, щоб визначити необхідний розмір аркуша та спосіб розташування малюнка;
- в опції File вибрати Size і встановити необхідний розмір аркуша. Це можна зробити шляхом перебору запропонованих форматів;
- перевірити, чи міститься у цьому аркуші вся схема;
- увійти в опцію File і вибрати Print Setup для встановлення книжкового або альбомного розташування;
- клацнути по іконці принтера.

Створення бібліотечного елемента:

- створити файл .gdf та відтранслювати його;
- клацнути "мишею" за командою File/Create Default Symbol.

Реалізація ієрархічного проекту у графічному редакторі:

- зобразити всі блоки (підпрограми) як файлів *.gdf;
- провести компіляцію кожного блоку (підпрограми);

- за допомогою команди File/Create Default Symbol занести кожен блок (підпрограму) до своєї бібліотеки підпрограм у вигляді символу;
- вставити цей символ як стандартний елемент в головний (main) блок (програму);
- відобразити усі зв'язки з цим елементом у головному блоці.

Робота з текстовим редактором

Цей редактор створює файли з розширенням *.tdf. Можна використовувати мови AHDL та VHDL.

Для перегляду include-файлів потрібно натиснути на іконку із зображенням папки, вибрати директорію maxplus2max2inc при ключі All files і переглянути файли з розширенням *.inc. Їхній вміст можна побачити в директорії maxplus2 max2libmf. Щоб побачити вміст function (include), потрібно натиснути на опцію Templates.

Створення Include-файлу:

- створити файл .tdf з шапкою TITLE та відтранслювати його;
- клацнути "мишею" по команді File/Create Default Include File;
- прочитати цей файл та уточнити назви входів/виходів.

Перегляд Include-файлу зі стандартної бібліотеки:

- клацнути по Help;
- вибрати опцію Searh for Help on┘ ;
- знайти ім'я примітиву (4count, 8count тощо);
- клацнути на кнопці "Показати";
- переглянути вміст.

Перегляд Include-файлу з власної бібліотеки:

- відкрити папку з шуканим Include-файлом;
- викликати свій Include-файл;

- переглянути вміст.

Налаштування на ПЛІС серії MAX7000S:

- увійти в опцію Assign/Device;
- встановити у вікні Device Family тип ПЛІС MAX7000S;
- тільки потім дати ім'я проекту та запустити трансляцію.

4.1. Створення комбінованого проекту

Раціональне проектування пов'язане з використанням графічного та текстового редакторів одночасно. Стандартні вузли (лічильники, регістри, суматори тощо) зручно створювати в графічному редакторі. Мікропрограмні автомати (МПА), довільні логічні функції, операції з умовами, циклами тощо слід синтезувати в текстовому редакторі. За такого проектування економиться час, а схема виходить досить прозорою. Порядок розв'язання задачі в цьому випадку може бути наступним:

- розбити схему на стандартні вузли та нестандартні блоки;
- створити у текстовому редакторі проект(и), що описує всі нестандартні блоки;
- відтранслювати текстовий файл(и) з локальним ім'ям проекту та створити символ, що відповідає даному нестандартному блоку;
- у графічному редакторі створити файл із глобальним ім'ям, до якого включити всі нестандартні блоки у вигляді символів та всі стандартні вузли;
- відтранслювати отриманий файл.

Компіляція та налагодження

Перед компіляцією в опції File/Project вказати ім'я проекту (ім'я головного файлу з розширенням *.gdf). Компіляція проводиться через опцію

MAX+plusII/Compiler, або через головну інструментальну лінійку. Якщо з'явиться вікно компілятора, натисніть клавішу Start.

За наявності помилок у файлі, що компілюється, з'являються відповідні повідомлення. Виділивши потрібне повідомлення і натиснувши кнопку Location, можна локалізувати помилку.

Для скасування глобальних входів тактування, скидання та вибору кристала потрібно в меню вибрати опцію Assign, а в ній Global Project Logic Synthesis та прибрати всі "галочки" у секціях Automatic Global, MAX Device Synthesis Options та інших. Переконайтеся, що у вікні Global Project Synthesis Style встановлено режим NORMAL. У цьому випадку глобальні входи можна використовувати як звичайні.

Для визначення обсягу проекту необхідно виконати такі операції:

- клацнути "мишею" по "піраміді";
- розкрити опцію rpt;
- переглянути повідомлення до опції %LCS Utilized.

Моделювання

Відкриваємо файл із розширенням *.scf, клацнувши по відповідному значку головної лінійки інструментів. Двійним клацанням викликаємо меню вводу вхідних/вихідних перемінних.

Для встановлення інтервалу часу моделювання потрібно увійти в опцію File/EndTime та встановити необхідний час (us, ms, s) або масштаб.

Для визначення тактової частоти необхідно використовувати ліву лінійку інструментів (іконка із символом "C"), попередньо натиснувши діаграму тактової частоти.

Щоб визначити значення вхідних символів необхідно з використанням вертикального трея встановити з протягуванням тривалість сигналу, а далі його значення з використанням іконки з символом "1" на треї інструментальній панелі.

Якщо при установці частоти не спрацьовує завдання кроку, то клацнути по 3-й праворуч іконці на горизонтальній інструментальній лінійці, а потім знову на вертикальній лінійці повторити завдання частоти.

Для запуску пакета моделювання (симулятор) необхідно натиснути кнопку симулятора головної інструментальної панелі.

Призначення ресурсів

Для вказівки нумерації висновків слід виконати такі операції:

- увійти до меню Assign;
- вибрати опцію Pin/Location/Chip┘ ;
- у вікні записати ім'я вузла і номер виведення ПЛІС.

Переглянути результати своєї роботи з розподілу ресурсів можна, викликавши файл .acf, де ім'я проекту.

Програмування ПЛІС

Після призначення ресурсів можливе програмування ПЛІС. Програмування виконується у такому порядку:

- при вимкненому ПК приєднати кабель ByteBlaster до принтерного роз'єму ПК та програмованого приладу;
- увімкнути ПК, запустити MAX+PLUS II;
- подати харчування (зазвичай +5) на програмований прилад;
- увійти до опції меню Filei задати ім'я проекту;
- вибрати опцію MAX+plus II/Programmeri кнопку Program або просто клацнути "мишею" по іконці програмування;
- переконатися у успішному виконанні програмування;
- відключити живлення програмованого приладу;
- вийти із MAX+PLUS II;
- вимкнути ПК;
- від'єднати кабель ByteBlaster.

Процедура розробки нового проекту у системі АП MAX+PLUS II

Процес розробки нового проекту від концепції до завершення можна спрощено представити так:

1. створення нового файлу проекту або ієрархічної структури кількох файлів проекту що використанням різні редактори створення проекту в системі MAX+PLUS II, тобто графічного, текстового та інших редакторів;

2. привласнення імені проекту самого верхнього рівня (Top of hierarchy) як імені системи;

3. визначення програмуваних логічних мікросхем для створення проекту. Розробник сам назначає конкретний пристрій або передати дану дію компілятору для оцінки наявних ресурсів;

4. відкриттям компілятора і його завантаження кнопкою “Start” для розпочаття процесу компіляції проекту. Якщо розробник хоче, він підключає модуль виводу часових затримок Timing SNF Extractor для розробки файлу, що застосовується чи використовується при перевірці часових параметрів;

5. у разі успішної компіляції – тестування та годинний аналіз, для виконааня оного потрібно виконати слідуючі дії:

5.1. для виконання годинного аналізу відкрити вікно “Timing Analyzer”, вибрати режим аналізу та нажати кнопку “Start”;

5.2. для виконання тестування потрібно створити файл каналу тестування (*.scf), використавши сигнальний редактор, чи у файлі вектора перевірки (*.vec), зкориставшись текстовим редактором;

6. створення або копіювання любої конфігурації створеного приладу проводиться шляхом запуску програматора з даною вставкою пристрою проектування в програмувальний адаптер вибраного програматора MPU (Master Programming Unit) чи з використанням підключення засобу MasterBlaster, ByteBlaster чи проводу завантаження FLEX (FLEX Download Cable) до пристрою, який програмується у системі;

7. вибір кнопки “Program” для програмування пристрою з ЗПУ EPROM (MAX, EPC) або вибом кнопки Configure для переписування конфігурації засобу з пам'яттю типу SRAM (FLEX).

Якщо в процесі роботи в системі MAX+PLUS II відрізняти різницю між файлами проекту, допоміжними файлами та проектами.

Файл проекту – це графічний, текстовий чи сигнальний файл, створений за допомогою графічного чи сигнального редактора для редактора середовища MAX+PLUS II. Цей файл містить логіку проекту та охоплюється компілятором. Він автоматично обробляє описані файли проекту:

1) графічні схеми системи (*.gdf);

2) текстові програми проекту на мові AHDL (*.tdf);

3) сигнальні епюри проекту (*.wdf);

4) програми проекту на мові VHDL (*.vhd);

5) програми проекту на мові Verilog (*.v);

6) схемні модулі OrCAD (*.sch);

7) вхідні проекти EDIF (*.edf);

8) файли в форматі Xilinix Netlist (*.xnf);

9) файли програми Altera (*. adf);

10) цифрові проекти (*.smf).

Додаткові файли – це програми, що пов'язані з проектом MAX+PLUS II, але вони не являються частиною ієрархії проекту. Багатоь таких файлів не містять опису любих функцій проекту. Частина з них утворюються автоматично системи MAX+PLUS II, інші – оператором.

Проект містить зі усіх файлів ієрархічної структури, у тому числі допоміжних та вхідні файли. Ім'я проекту є ім'я файлу найвищого рівня без розширення. Система MAX+PLUS II проводить компіляцію, тестування, часовий аналіз та програмування всього проекту, хоча розробник може перероблювати файли даного проекту в рамках другого проекту.

Для любої схеми слід створювати окрему папку у робочому каталозі системи (\max2work).

При розпочатті роботи системи MAX+PLUS II автоматично відкривається головне вікно (Main Window) (рис. 4.1), що охоплює всі додатки системи. У верхньому рядку написано ім'я проекту, з яким працював розробник. Два слідуючі рядки є схожими для Windows: рядок головного меню та панель застосувань, у лівій стороні даної знаходяться поширені інструменти Windows (Print, Cut, Copy New, Open, Save, , Paste, Undo), в правій – спеціалізовані інструменти пакету з використаннямяких відбувається запуск основних можливостей пакету.

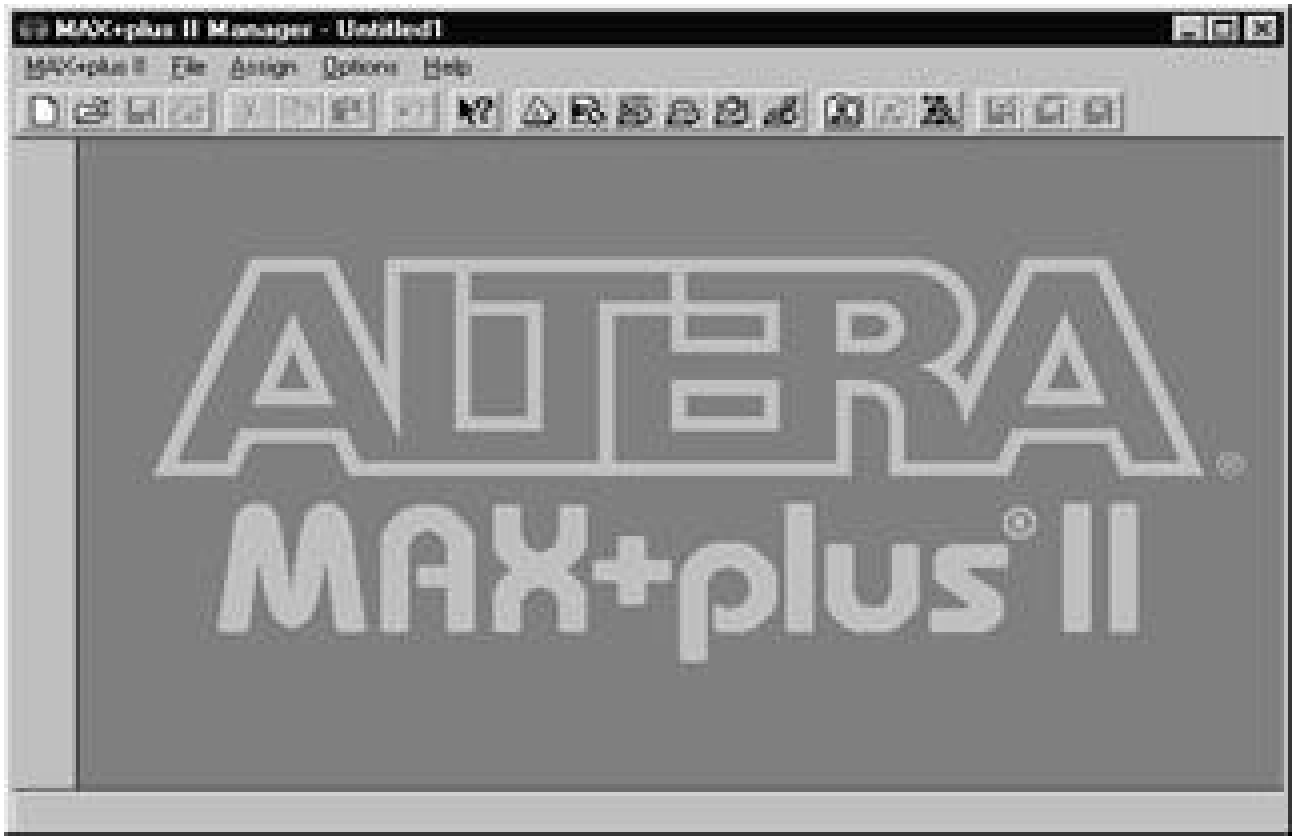


Рис. 4.1. Центральне вікно програми автоматизованого проектування MAX+PLUS II

Запуск складових системи просто проводити через вікно меню MAX+PLUS II (рис. 4.2), яке містить у собі вкладені підменю визову основних додатків: перегляд ієрархії, графічне поле, символьний планувальник, текстовий редактор, сигнальний модулятор, порівневий редактор, компілятор, симулятор, аналізатор годинних параметрів, програмуючий пристрій та

генератор сповіщень, функціональне використання яких вже було описано раніше.

В багаторівневій структурі проекту на любому рівні можливо змішати використання складових з розширеннями .gdf, .tdf, .vhd, .v, .edf, .sch. Хоча файли з розширенням .wdf, .xnf, .adf, .smf повинні бути на самому нижньому рівні ієрархії проекту, або бути одним файлом.

В усіх застосунках MAX+PLUS II повинна бути можливість за допомогою команд Призначити вводити, змінювати та знищувати види вибраних ресурсів, пристроїв та параметрів,що керують компіляцією проекту, логічним синтезом та розподілом на частини. На рис. 4.3 продемонстровано команди меню Assign. Розробник може виконувати призначення для вибраного проекту незалежно, чи відкритий якийсь файл проекту чи вікно додатків.

Рис. 4.2. Меню MAX+PLUS II

Рис. 4.3. Призначення проекту Assign

Програма MAX+PLUS II містить інформацію для проекту у копії з розширенням .acf. Зміна перевизначень, які зроблені у вікні рівневого планувальщика також зберігаються з розширенням .acf. Також, користувач може редагувати acf-файл системи у текстовому редакторі.

Слідуючі функції є загальноприйнятними для всіх додатків MAX+PLUSII: призначення пристрою, ресурсів та щупів, збереження більш старих версій, глобальні можливості пристрою в проекті, головні параметри проекту, головні вимоги до часових параметрів системи, глобальний структурний та функціональний синтез проекту.

Мікросхема є частиною пристрою Altera, наприкладом являється, контакт або базовий елемент, який виконує зазначене, визначене користувачем завдання. Управління компіляцією проекту та іншими параметрами проводиться за допомогою вбудованих додатків. Використовуються наступні методи призначень.

Clique assignment (визначення кліка) – задає які логічні функції можуть залишатися разом. Таке групування логічних функцій гарантує, що вони будуть реалізовуватися в одному блоці логічної структури мікросхеми, одному її ряді чи пристрої.

Chip assignment (визначення мікросхеми) – задає функції, що повинні бути створені в одному пристрої у разі розподілення проекту на декілька частини (декілька систем).

Pin assignment (визначення виходу) – визначає вхід або вихід одного логічного блоку функцій, як примітив або мегафункція, конкретного контакту або горизонтального (вертикального) ряду виводів ПЛМ.

Location assignment (визначення осередка) – задає розміщення логічного блоку (вузла) у конкретному логічному елементі. У полях даного додатка можна задати вибраний вивід, логічний блок, а також, за використання кнопки “Change” та “Delete”, змінити перепризначення.

Probe assignment (визначення щупів) – надає легке для зберігання унікальне ім'я входу чи виходу вибраної системи функцій.

Connected pin assignment (визначення з'єднаних виводів) – задає зовнішнє з’єднання двох чи більше виводів на проектуємій схемі. Дана інформація буде корисна і в режимі тестування необхідних часових параметрів системи і при тестуванні декількох створених проектів.

Local routing assignment (призначення місцевого трасування) – привласнює коефіцієнт розподілення за виходом вузла логічного елемента, що міститься в функціональному блоці логічних елементів або сусідньому блоці базови елементів, суміжним з вибраним вузлом, з використанням місцевих зв'язків. Місцеве трасування проводиться між вузлами логічних елементів на периферії пристрою, та вивідним контактом, з яким він поєднаний. Призначення вибраного трасування проводиться за допомогою команди Assign/Local routing.

Device assignment (визначення пристрою) – призначає тип ПЛМ, де буде втілений проект. Якщо проект містить в собі декілька пристроїв, то дана функція здійснює призначення чіпів необхідним пристроям. Можливо також вибрати команду Auto і передати компілятору право вибирати пристрій із заданої батьківщини пристроїв. Процесом автоматичного вибору пристрою можна керувати, задаючи діапазон і кількість пристроїв у батьківщині. Якщо проект є досить великим для створення в одному пристрої, можна задати тип і кількість необхідних додаткових пристроїв. Щоб вибрати пристрій використовується команда “Assign/Device”.

Logic option assignment (визначення логічної функції) – керує створенням окремих логічних компонентів під час обробки з використанням стилю базового логічного синтезу та деяких опцій логічного створенняпідтримує. Фірма Altera підтримуєє велику кількістьбазовихмоделі опцій, а також готових стилів, кожний з яких являє собою зібрання установок для логічних опцій, об'єднане одним ім'ям моделі синтезу (Synthesis style). Розробник може користуватися готовими стилями або створювати нові. Стилі синтезу можуть настроювати опції синтезу на вибрані родинні пристроїв, враховуючи при цьому архітектуру батьківщини. Для налаштування моделей синтезу вможна використовувати команду “Assign/Logic Options”.

Timing assignment (визначення часових сигналів) – управляє логічним створенням та підгонкою окремих функцій з метою отримання заданих значень для годин затримки. Розробник може знищити з'єднання між шляхами

для вибраного сигналу та іншими складовими чи блоками проекту. Визначення часових параметрів системи відбувається за командою Assign/Timing Requirements.

Можна проводити глобальні годинні вимоги для проекту, задаючи загальні характеристики для затримки, використовується команда “Assign/Global Project Timing Requirements”.

Для визначення всіх параметрів базового синтезу проекту використовується команда Assign/Global Project Logic Synthesis.

4. 2. Процедура компіляції створеного проекту у системі автоматизованого проектування MAX+PLUS II

Спочатку компілятор дістає інформацію про ієрархічні зв'язки між файлами проекту та перевіряє систему на прості помилки введення дизайнів. Компілятор створює організаційну модель проекту і потім, комбінуючи з усіма файлами проекту, перетворює їх на базу даних без ієрархії, яку він здатний повноцінно обробляти.

Компілятор застосовує різноманітні засоби зростання ефективності проекту та мінімізації використання ресурсів пристрою. Якщо проект надто великий, то він не може бути реалізованим в одній програмувальній логічній схемі – компілятор може розбити проект на необхідні частини для реалізації у декількох пристроях того чи іншого сімейства програмувальних логічних інтегральних схем, при цьому зменшується число з'єднань між системами. У файлі звітності (.rpt) буде відображено яким чином проект буде реалізовуватись: в одному або кількох системах.

Компілятор повинен автоматично компілювати систему. Існують деякі можливості задати обробку системи у відповідності з визначеними вказівками користувача. Наприклад, можна задати методи логічного синтезу системи та інші параметри базового синтезу у межах всього проекту. Крім того, зручно задавати часові вимоги у межах всього проекту, точно вказати розмежування

великого блоку на частини для реалізації в декількох системах та обрати варіанти параметрів пристроїв, які будуть використані для всього проекту в цілому. Користувач здатний вибирати кількість виводів та базових логічних блоків, які залишатимуться не задіяними під час текучої компіляції, щоб зарезервувати їх дляподальших модифікацій проекту.

Компіляцію системи можливо запустити з любого додатку MAX+PLUS II у вікні компілятора. Компілятор самостійно обробляє всі необхідні файли поточного проекту.

Процес компіляції можна побачити у вікні компілятора (рис. 4.4) у такому вигляді:

1) спустошується та перевертається пісковий час, що говорить про активність компілятора;

2) відображується чергою у вигляді прямокутників компілятора;

3) під вікном модуля компілятора міститься піктограма вихідного файлу, що був створений цим модулем;

4) відсоток завершення процесу омпіляції поступово збільшується (до 100%);

5) за час розбиття і монтажу компілятора Stop (Стоп) перетворюється на Stop/Show Status (Стоп/Показати стан), яку користувач може вибрати для відкриття діалогового вікна, в якому відтворюється стан розботи та монтажу проекту;

6) якщо виявлені в процесі роботи компілятораї будь-які помилки або можливих проблем автоматизовано відкривається вікно системи обробника повідомлень, у якому відтворюється список повідомлень про помилки, що попереджують та повідомлень, а також відразу створюється довідка з виправлення помилки. Також, користувач може визначити першопричину повідомлень у файлах проекту або порівневому планувальнику.

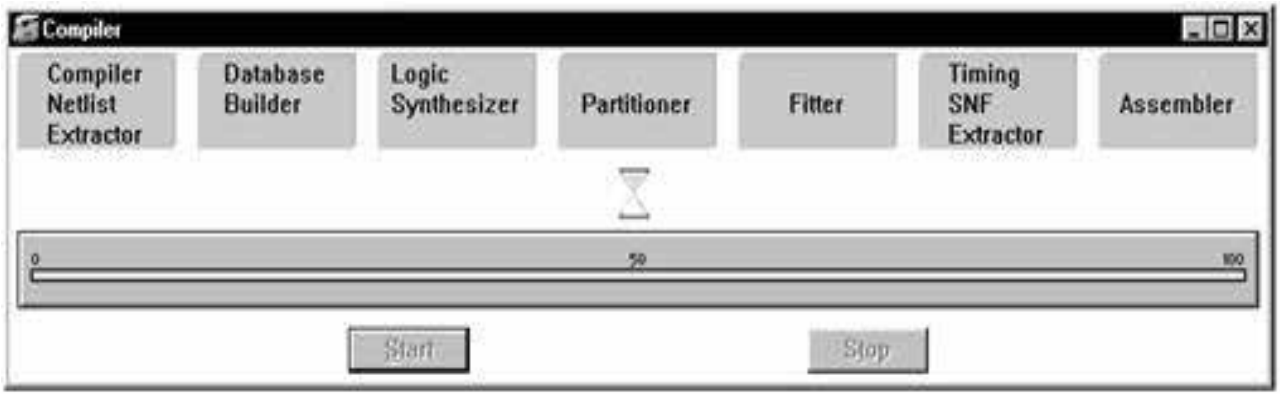


Рис. 4.4. Компіляція проекту

Компілятор САПР MAX+PLUS II обробляє проект, з використанням наступних утиліт:

1) визначник списку кіл (Compiler Netlist Extractor), що вміщує програми для читання різних форматів EDIF, VHDL, Verilog, XNF;

2) будівельник бази даних (Database Builder);

3) синтезатор базових функцій (Logic Synthesizer);

4) розподільник (Partitioner);

5) трасувальник проекту(Fitter);

6) пристрій функціонального тестування (Functional SNF Extractor);

7) визначникдля тестування часових параметрів (Timing SNF Extractor);

8) визначник для тестування компонування (Linked SNF Extractor);

9) програма для зберігання вихідного файлу EDIF (EDIF Netlist Writer);

10) система запису вихідного файлу Verilog (Verilog Netlist Writer);

11) програма запису вихідного файлу VHDL (VHDL Netlist Writer);

12) модуль ассемблера (Assembler);

13) програма діагностики системи (Design Doctor Utility).

Модуль екстрактора форматів (Компілятор Список мереж Extractor) перетворює кожен файл проекту в один або кілька двійкових файлів із розширенням. cnf (компілятор список мереж файл). Оскільки компілятор замінює значення всіх параметрів, які використовуються в параметризованих функціях , вміст файлу cnf може змінитися під час наступних компіляцій, якщо значення параметрів зміняться. Цей модуль також створює файл ієрархічних

зв’язків із розширенням. hif (ієрархія підключитися до файл). Цей файл документує ієрархічні зв’язки між файлами проекту, а також інформацію, необхідну для відображення ієрархічного дерева у вікні «Ієрархія». Дисплей ". Крім того, цей модуль створює файл бази даних вузла з розширенням . ndb (вузол база даних), яка містить імена вузлів проекту для бази даних призначення ресурсів.

Модуль конструктора баз даних (Database Builder) використовує файл ієрархічних зв’язків для створення файлів cnf , створених компілятором, які містять опис проекту. Базуючись на даних ієрархічної структури проекту, цей модуль копіює кожен файл cnf в одну базу даних без ієрархічної структури. Таким чином база даних зберігає електричні з’єднання проекту.

При створенні бази даних модуль перевіряє логічну завершеність і узгодженість оформлення, а також перевіряє граничні зв'язки і наявність синтаксичних помилок. Більшість помилок виявляються на цій стадії компіляції та можуть бути легко виправлені. Кожен модуль компілятора послідовно обробляє та оновлює цю базу даних.

Коли компілятор обробляє проект вперше, компілюються всі файли проекту. Користувач має можливість вибрати «швидку перекомпіляцію» (smart recompile), щоб створити розширену базу даних проекту, яка дозволяє пришвидшити наступні компіляції. Використання параметра повної перекомпіляції (заг recompile), ви можете вибрати між перекомпіляцією лише тих файлів, які були відредаговані після останньої компіляції, або повною перекомпіляцією.

Модуль логічного синтезу (Logic Синтезатор) використовує серію алгоритмів, які зменшують споживання ресурсів і видаляють повторювану логіку, забезпечуючи таким чином ефективне використання структури логічних елементів для архітектури всієї батьківщини пристрою. Крім того, логічний синтезатор шукає логіку для непідключених вузлів. Якщо він знаходить такий вузол , він видаляє примітиви, пов’язані з таким вузлом.

Якщо під час встановлення проект не поміщається на одному пристрої, модуль Partitioner розбиває базу даних на кілька FPGA одного батька, намагаючись при цьому розділити проект на мінімальну кількість пристроїв.

Використовуючи базу даних, оновлену модулем розділення, модуль відстеження (Fitter) узгоджує вимоги до дизайну з відомими ресурсами одного або кількох пристроїв. Він призначає кожній логічній функції розташування логічного елемента, що її реалізує, і вибирає відповідні методи підключення та призначення виходів.

Functional SNF Extractor створює функціональний тестовий файл із розширенням . snf_ _ Компілятор генерує цей файл перед синтезом проекту, він містить усі вузли, присутні у початкових файлах проекту.

Timing SNF Extractor створює, якщо проект було скомпільовано без помилок, файл для перевірки параметрів синхронізації, що містить дані про параметри синхронізації проекту. Розширення файлу також . snf_ _

Пов’язаний SNF Extractor створює файл (.snf) для тестування макета кількох проектів (рівень плати). Такий файл поєднує інформацію з двох типів файлів snf : для тестування параметрів синхронізації та функціональних тестів, які були синтезовані для цих кількох проектів окремо.

Програма для запису вихідного файлу у форматі EDIF (EDIF Netlist Письменник). Компілятор MAX+PLUS II може працювати з більшістю стандартних програм автоматичного проектування, які читають файли у стандартному форматі EDIF 200 або EDIF 300. Edo .

Програма для запису вихідного файлу в Verilog (Verilog Список мереж Письменник). Додатковий модуль Verilog Writer генерує вихідні файли з розширенням. vo , що містить інформацію про функції та їхні часові параметри, отримані після синтезу.

Програма для запису вихідного файлу у форматі VHDL (VHDL Netlist Письменник). Додатковий модуль компілятора запису VHDL генерує один або кілька вихідних файлів VHDL (.vho) із синтаксисом 1987 або 1993.

Модуль асемблера перетворює логіку, контакти та призначення пристроїв, зроблені модулем трасування, в образ програмного забезпечення пристрою у формі одного або кількох бінарних об’єктних файлів програміста (.pof) і об’єктних файлів SRAM (.sof) .

Інструмент діагностики дизайну (Design лікар Утиліта) перевіряє логіку кожного файлу проекту, щоб визначити елементи, які можуть викликати проблеми з надійністю на системному рівні. Ці проблеми стають помітні тільки після того, як пристрій включено «в залізо». Для обробки проектів різного рівня можна вибрати одне з трьох попередніх правил.

4.3. Загальні відомості про мову опису обладнання AHDL

Мова опису обладнання AHDL (Altera Обладнання опис Мова) розроблена компанією Altera і призначена для опису комбінаційних і послідовних логічних пристроїв, групових операцій, цифрових автоматів з урахуванням особливостей FPGA Altera . Повністю інтегрується з системою автоматичного проектування MAX+PLUS II. Файли опису обладнання, збережені в AHDL, мають розширення . tdf (Текст дизайн файл). Щоб створити tdf -файл , ви можете використовувати текстовий редактор MAX+PLUS II або будь-який інший текстовий редактор. Проект, виконаний у вигляді файлу tdf, компілюється та використовується для створення програмного файлу або завантаження Altera FPGA .

Оператори та елементи AHDL є досить потужним і універсальним засобом опису алгоритмів роботи цифрових пристроїв. Мова опису апаратного забезпечення AHDL дозволяє створювати ієрархічні структури в межах однієї з цих мов або використовувати файли AHDL tdf та інші типи текстових описів обладнання в ієрархічній конструкції. Звичайно, для створення проектів AHDL можна використовувати будь-який текстовий редактор, але текстовий редактор системи MAX+PLUS II надає ряд додаткових можливостей для введення, компіляції та перевірки проекту.

Файли, створені в AHDL, можна легко інтегрувати в ієрархічну структуру проекту. Система MAX+PLUS II дозволяє автоматично створити символ елемента, алгоритм роботи якого описаний у файлі tdf , а потім вставити його у файл опису діаграми (файл gdf). Крім того, користувач може вводити власні функції на додаток до приблизно 300 макро-функцій, розроблених Altera . Для всіх функцій, що входять до бібліотеки макросів системи MAX+PLUS II, Altera надає файли з розширенням . вкл . (вкл дизайн файл).

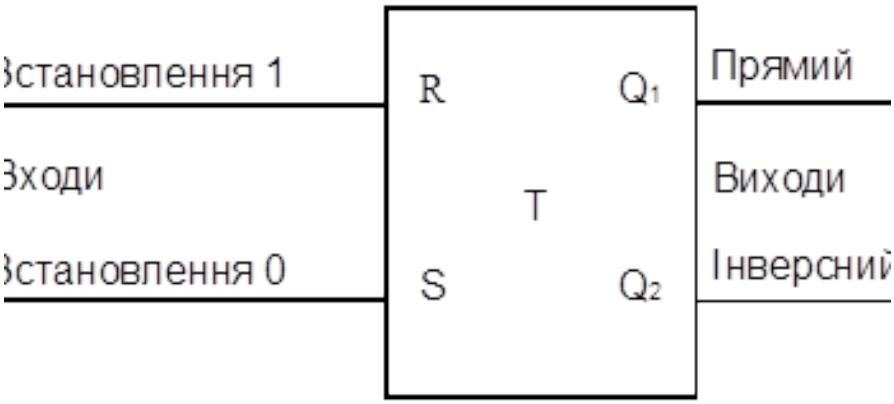
При розподілі ресурсів пристрою дизайнер може використовувати програми редагування тексту або оператори AHDL. Крім того, розробник може перевірити синтаксис і виконати повну компіляцію. Будь-які помилки автоматично реєструються обробником повідомлень, а інформація про їх виникнення з'являється у вікні текстового редактора, що оптимізує час розробки пристрою.

4.4. Реалізація базових пристроїв мікроелектроніки в інтегрованому середовищі MAX+PLUS II

Основною структурною одиницею, яка використовується для побудови комбінаційних логічних систем, є логічний елемент (клапан). У послідовних логічних схемах роль такої структурної одиниці виконує тригер. У цій частині дипломної роботи будуть розглянуті різні типи тригерів.

Умовне позначення тригера RS наведено на рис. 7.1.1: Тригер RS має два входи R і S і два виходи $_{Q1}$ і $_{Q2}$. У тригерах виходи завжди знаходяться в протилежних (компланарних) станах. Іншими словами, якщо вхід Q $_{1}$ є логічною одиницею, вихід Q $_{2}$ буде логічним нулем і навпаки.

Входи R і S даного тригера називають опорним входом 1 і опорним входом 0 відповідно.



Малюнок 4.5. Традиційне позначення тригера RS

Принцип роботи тригера RS ілюструє його таблиця істинності (табл. 4.5).

Таблиця 4.1. Таблиця істинності тригера RS

Режим роботи	Увійдіть		спускатися		
	C	P	Q1 _	Питання 2	Вплив на вихідну потужність Q 1
Заборонений стан	0	0	1	1	Заборонено - він не використовується
Встановлення 1	0	1	1	0	Щоб встановити Питання 1 в 1
Установка 0	1	0	0	1	Щоб встановити Питання 1 з 0
Безпека	1	1	Q1 _	Питання 2	Залежить від попереднього стану

логічна одиниця встановлюється на обох виходах (Q 1 = Q 2 = 1). Це заборонена умова тригера; він не використовується. Відповідно до іншого рядка таблиці істинності вихід Q 1 встановлено в логічну одиницю . У цьому випадку кажуть, що тригер встановлений у стан 1. Згідно з третім рядком, коли S=1 і R=0, сигнал на вході Q 1 скидається (скидається вихід Q 1) до логічного рівня

0 Це означає , що тригер встановлений у стан 0. Четвертий рядок таблиці істинності відповідає R =S=1 . У цьому випадку тригер залишається в стані очікування: на виходи Q_1 і Q2 записуються попередні рівні додаткових сигналів . Це режим економії.

З таблиці 4.1 видно, що встановлення тригера в стан 1 (встановлення 1 на виході Q_1) ініціалізує логічний 0 на вході S. Подібним чином, встановлення тригера в стан 0 (налаштування 0 на виході Q 1) стану тригера RS викликає через появу 0 на одному з його входів, то ймовірніше, що більш точним представленням цієї діаграми буде звичайне графічне представлення, показане на малюнку 4.6.

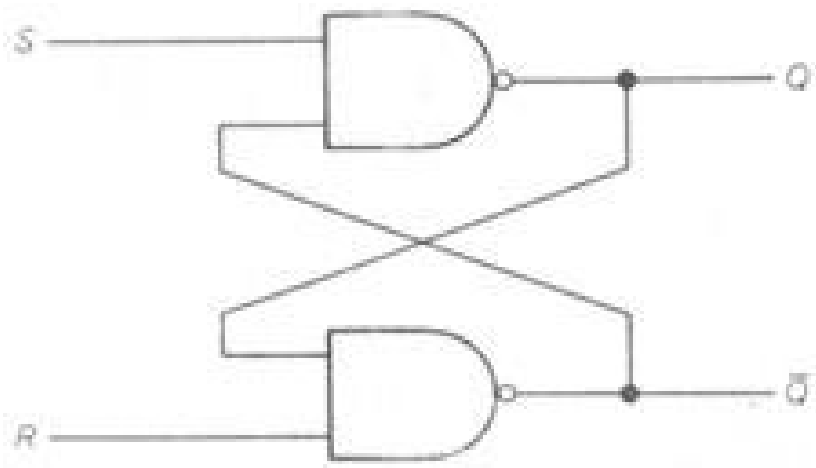


Рис. 4.6. Тригер RS побудований на логічних елементах I-HE

Особливу увагу слід приділити інвертуючим котушкам на входах R і S. Вони показують, що активним рівнем сигналу для установки тригера в стани 1 і 0 є рівень логічного 0 на одному з входів. Тригер RS часто називають засувкою RS або окремим тригером входу .

Принцип роботи синхронного RS тригера ілюструє його таблиця істинності (табл. 4.2).

Таблиця 4. 2. Таблиця дійсності синхронного тригера RS

Режим роботи	Увійдіть			спускатися		
	CLK	C	P	Q1 _	Питання $_2$	Вплив на вихідну потужність Q_1

Безпека	____	0	0	Без змін		Без змін
Установка 0	____	0	1	0	1	Щоб встановити Питання $_1$ з 0
Встановлення 1	____	1	0	1	0	Щоб встановити Питання $_1$ в 1
Заборонений стан	____	1	1	1	1	Заборонено - він не використовується

Лише три верхні рядки таблиці істинності описують фактичні режими роботи тригера RS. Нижній рядок відповідає забороненому стану і ніколи не використовується. З таблиці видно, що стан виходів синхронного тригера RS може змінюватися тільки при надходженні тактових імпульсів. У цьому випадку кажуть, що тригер працює синхронно: процес його перемикання синхронізований із тактовими імпульсами.

У багатьох цифрових схемах важливу роль грає ще одна особливість тригера RS - наявність пам'яті. Дійсно, якщо тригер встановлений на 1 або 0, він залишається в цьому стані навіть при певних змінах у вхідних сигналах.

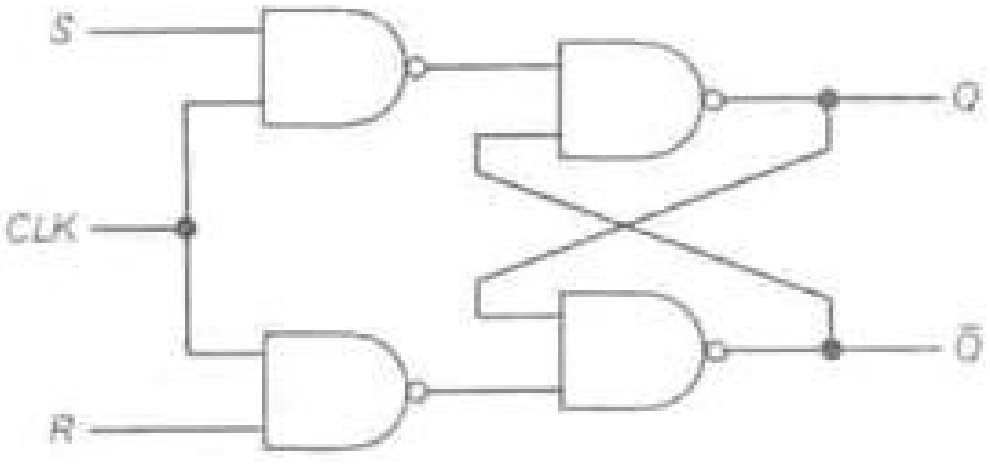


Рис. 4.7. Синхронний тригер RS побудований на логічних елементах ІН І

Щоб отримати синхронний RS-тригер, в схему звичайного RS-тригера необхідно ввести два додаткових логічних елемента І-НЕ, як показано на рис. 4. 7.

Умовне графічне позначення тригера D показано на малюнку 7.1.6. Цей тригер має тільки один вхід даних D і вхід синхронізації CLK. Тригер D часто називають тригером затримки. Слово «затримка» означає те, що відбувається з даними (інформацією), що надходять на вхід D.

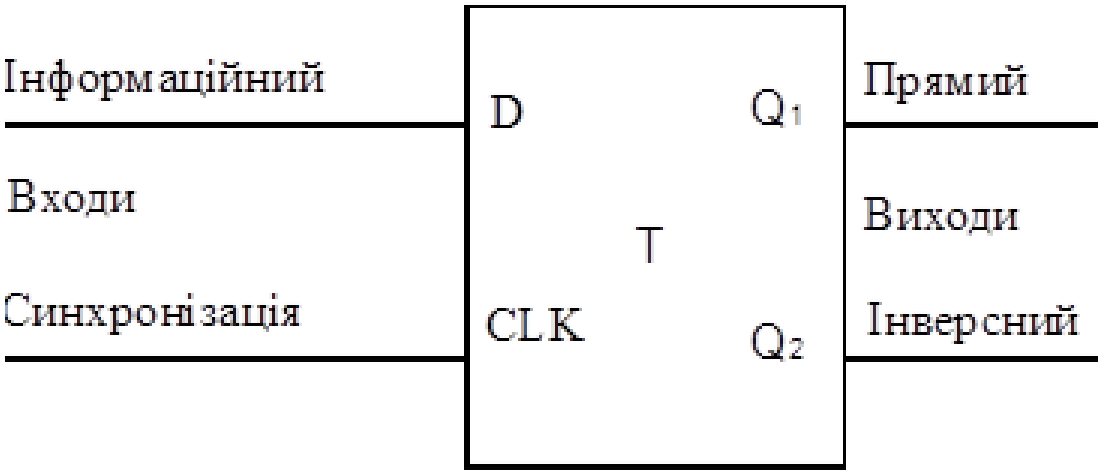


Рис. 4.8. Умовне графічне позначення тригера D

Зверніть увагу, що сигнал на виході Q у такті n+1 повторює сигнал, який був на вході D у попередньому такті n.

Тригер D можна отримати з тактованого тригера RS шляхом додавання до нього інвертора, як показано на рис. 4.8.

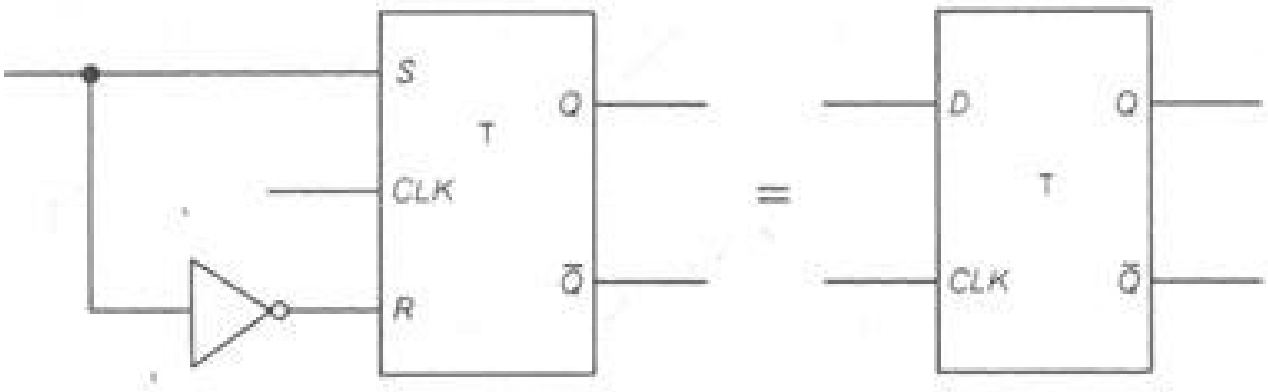
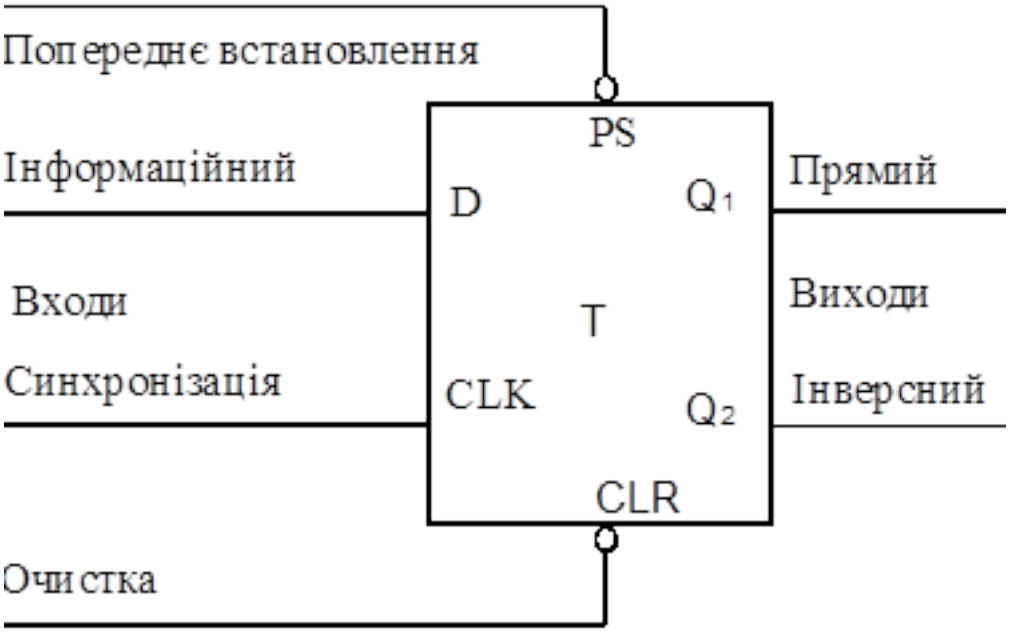


Рис. 4.9. D тригерна діаграма

На рис. 4.9. показано умовне позначення для типового, масового виробництва тригера D. Він має два додаткових входи - попередньо встановлений (PS) і очищений (CLR). Логічний 0 на вході PS ініціює логічну 1 на виході Q. Логічний 0 на вході CLR ініціює очищення на виході Q.

В активних станах входи PS і CLR блокують роботу входів D і CLK; після розблокування входи D і CLK працюють подібно до звичайного тригера D, показаного на рис. 4.8.



Малюнок 4.10. Умовне графічне позначення серії інтегральних тригерів D

Спусковий гачок JK є універсальним спусковим гачком, який має особливості всіх інших типів спускових механізмів. Умовне графічне позначення тригера JK показано на рис. 4/11. Тригер JK має два інформаційних входи: J і K і вхід синхронізації CLK і, як і всі тригери, два додаткових виходи $_{Q1}$ і $_{Q2}$. Таблиця тяжкості тригера JK наведена в таблиці. 4.3. Коли обидва входи J і K подаються на логічний рівень 0, тригер буде заблоковано, а стан його виходів не зміниться. У цьому випадку тригер знаходиться в режимі запису.

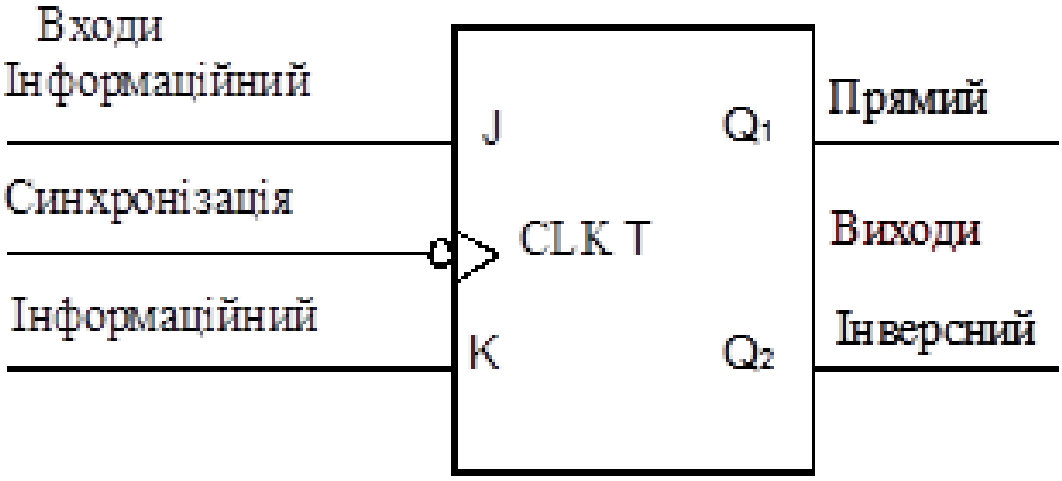


Рис. 4/11 . Умовне графічне позначення тригера JK

Таблиця 4.3. Таблиця істинності тригера JK

Режим роботи	Увійдіть			спускатися		
	CLK	Дж	К	Q1 _	Питання 2	Вплив на вихідну потужність Q 1
Безпека	____	0	0	Без змін		Без змін - блокування
Установка 0	____	0	1	0	1	Скинути або очистити Q 1 до 0
Встановлення 1	____	1	0	1	0	Щоб встановити Питання 1 в 1
Перемикання	____	1	1	Перемикання		Зміна статусу на протилежність

Рядки 2 і 3 таблиці істинності описують режими, що відповідають перемиканню тригера зі стану 0 в 1. Рядок 4 ілюструє дуже важливий режим роботи тригера JK - перемикання. Якщо обидва входи J і K встановлені на логічну 1, наступні тактові імпульси спричинять зміну виходів тригера з 1 на 0, з 0 на 1 і так далі. Ця робота схожа на перемикання перемикача, звідси й назва режиму.

Умовне графічне позначення тригера JK, що входить до складу інтегральної мікросхеми, показано на рис. 4/12 . Цей тригер має два додаткових асинхронних входи (вхід попереднього налаштування та вхід скидання).

Синхронними входами є інформаційні входи J і K і вхід синхронізації CLK.

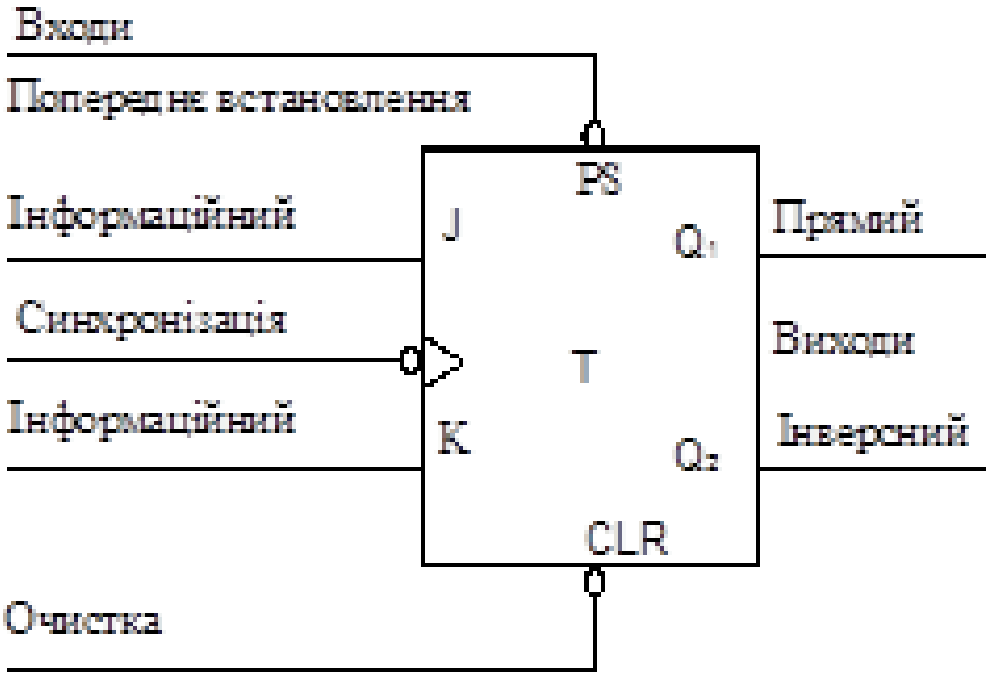


Рис. 4.12 . Умовне графічне позначення тригера з інтегралом серії JK

При реалізації тригерів за допомогою AHDL необхідно використовувати примітиви тригерів.

На таблиці 4.4. наведено всі основні тригерні елементи, які використовуються в описі роботи пристрою.

Таблиця 4.4 . Запуск примітивів у AHDL

Примітивний	Первісний прототип
DFF	ФУНКЦІЯ DFF (D, CLK, CLRN, PRN) ПОВЕРНЕННЯ (Q)
DFFE	ФУНКЦІЯ DFFE (D, CLK, CLRN, PRN, ENA) ПОВЕРНЕННЯ (Q)
TFF	ФУНКЦІЯ TFF (T, CLK, CLRN, PRN)

	ПОВЕРНЕННЯ (Q)
TFFE	ФУНКЦІЯ TFFE (T, CLK, CLRN, PRN, ENA) ПОВЕРНЕННЯ (Q)
JKFF	ФУНКЦІЯ JKFF (J, K, CLK, CLRN, PRN) ПОВЕРНЕННЯ (Q)
JKFFE	ФУНКЦІЯ JKFFE (J, K, CLK, CLRN, PRN, ENA) ПОВЕРНЕННЯ (Q)
SRFF	ФУНКЦІЯ SRFF (S, R, CLK, CLRN, PRN) ПОВЕРНЕННЯ (Q)
SRFFE	ФУНКЦІЯ SRFFE (S, R, CLK, CLRN, PRN, ENA) ПОВЕРНЕННЯ (Q)
ЗАСУВКА	ФУНКЦІОНАЛЬНА ЗАСУВКА (D, ENA) ПОВЕРНЕННЯ (Q)

Тригерні виходи:

D, T, J, K, S, R – інформаційні входи;

CLK – вхід тактового сигналу (активний спад 0-1);

CLRN – вхід скидання асинхронного тригера (активний рівень – логічний нуль);

PRN – вхід налаштування асинхронного тригера (активний рівень – логічний нуль);

ЕНП – введення дозволу на роботу (активний рівень – логічна одиниця).

Програма реалізації тригерів мовою AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином: Назва « тригери »;

Підпроект тригери

(

D,T,J,K,S,R,CLK,CLRN,PRN,ENA: вхід ;

Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9: вихід ;

)

Почніть

Q1 = DFF (D, CLK, CLRN, PRN); Q2 = DFFE (D, CLK, CLRN, PRN, ENA);

Q3 = TFF (T, CLK, CLRN, PRN); Q4 = TFFE (T, CLK, CLRN, PRN, ENA);

Q5 = JKFF (J, K, CLK, CLRN, PRN);

Q6 = JKFFE (J, K, CLK, CLRN, PRN, ENA);

Q7 = SRFF (S, R, CLK, CLRN, PRN);

Q8 = SRFFE (S, R, CLK, CLRN, PRN, ENA);

Q9 = ЗАСУВКА (D, ENA);

кінець _

На рис. 4/13. буде показано вікно редактора сигналів проекту Triggers .

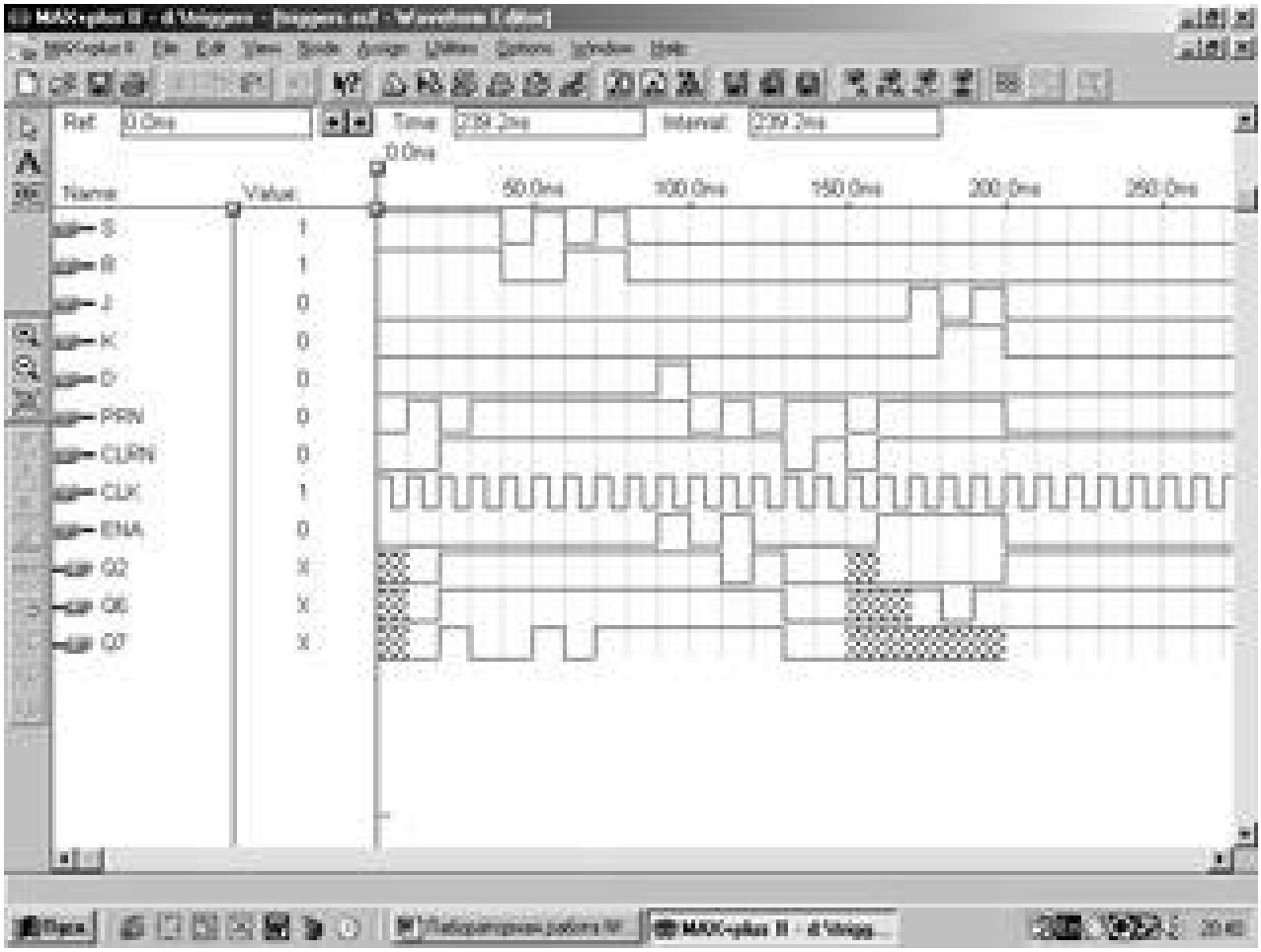


Рис. 4/13. Результати тестування тригерів RS-, D-, JK

4.5. Теоретичні відомості про регістри

Схема одного типового регістра зсуву показана на рис. 4.14 . Цей регістр реалізований на 4-х тригерах D. Такий регістр називається 4-розрядним регістром зсуву, тому що він дозволяє зберігати 4 двійкові біти даних A, B, C, D.

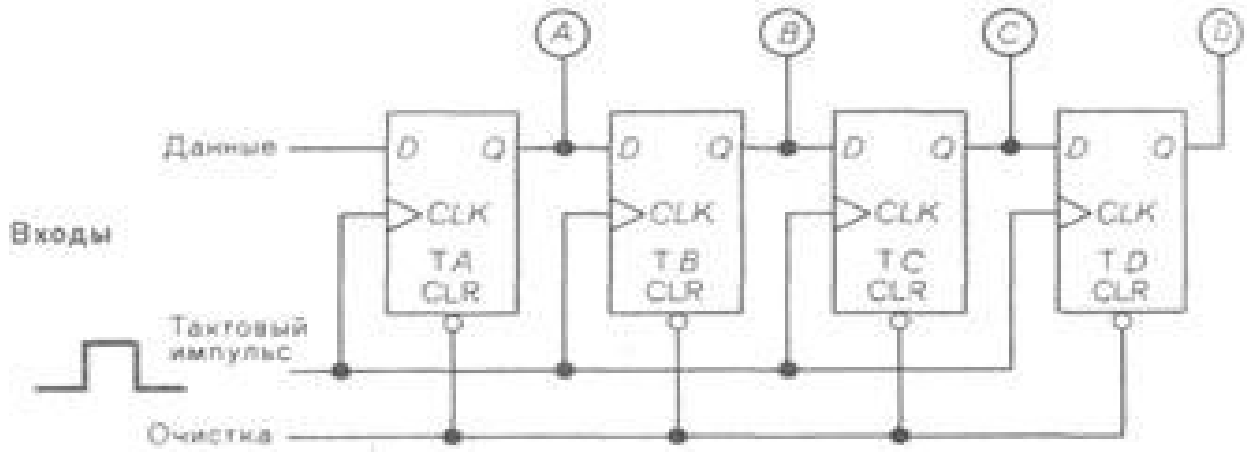


Рис. 4.14 . 4-розрядний регістр послідовного зсуву

За допомогою таблиці 4.4. та рис. 4.14. ми маємо можливість спостерігати за роботою цього приладу. Спочатку очистимо регістр (встановимо рівні логічного нуля на його виходах A, B, C, D). Для цього застосуйте логічний 0 до вхідних даних CLR clear. Рядок 1 таблиці відповідає отриманому стану регістра зсуву. 4.4. До надходження тактового імпульсу виходи регістру залишаються в стані 0000. Давайте подамо перший імпульс на вхід синхронізації CLK; індикатор покаже число 1000, тому що в момент тактового імпульсу логічна 1 з інформаційного входу тригера TA передається на його логічний вихід Q. цифра A, а введені раніше цифри зсуваються на одну позицію. (цифра) праворуч . Таким же чином, коли логічний 0 подається на інформаційний вхід, нуль буде введений в біт A на кожному тактовому імпульсі, а раніше введені нулі та одиниці будуть зсунуті вправо. Перед надходженням тактового імпульсу 9 інформаційний вхід встановлюється на 1, а перед надходженням імпульсу 10 цей вхід повертається на 0. Протягом годин роботи тактових імпульсів 9-13 блок входив у стан реєстру по імпульсу. 9

переміститься вправо на індикаторі. Рядок 15 у табл. 4.5. показує, що в імпульсі 13 він пропускає крайній правий біт регістра зсуву і втрачається.

Таблиця 4.5. Робота 4-розрядного регістра зсуву

Увійдіть				спускатися			
Номер лінії	прибирання	Дано	Число тактових імпульсів	ТА	туберкульоз	співвласник	TD
				І	Б	С	Д
1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0
3	1	1	1	1	0	0	0
4	1	1	2	1	1	0	0
5	1	1	3	1	1	1	0
6	1	0	4	0	1	1	1
7	1	0	5	0	0	1	1
8	1	0	6	0	0	0	1
9	1	0	7	0	0	0	0
10	1	0	8	0	0	0	0
11	1	1	9	1	0	0	0
12	1	0	10	0	1	0	0
13	1	0	11	0	0	1	0
14	1	0	12	0	0	0	1
15	1	0	13	0	0	0	0

Тригер D також називають тригером із затримкою . Він просто посилає інформаційний сигнал із входу D на вихід Q із затримкою в один такт.

Пристрій, схема якого зображена на рис. 4/14. називається послідовним регістром зсуву. Термін «послідовний» відображає той факт, що дані

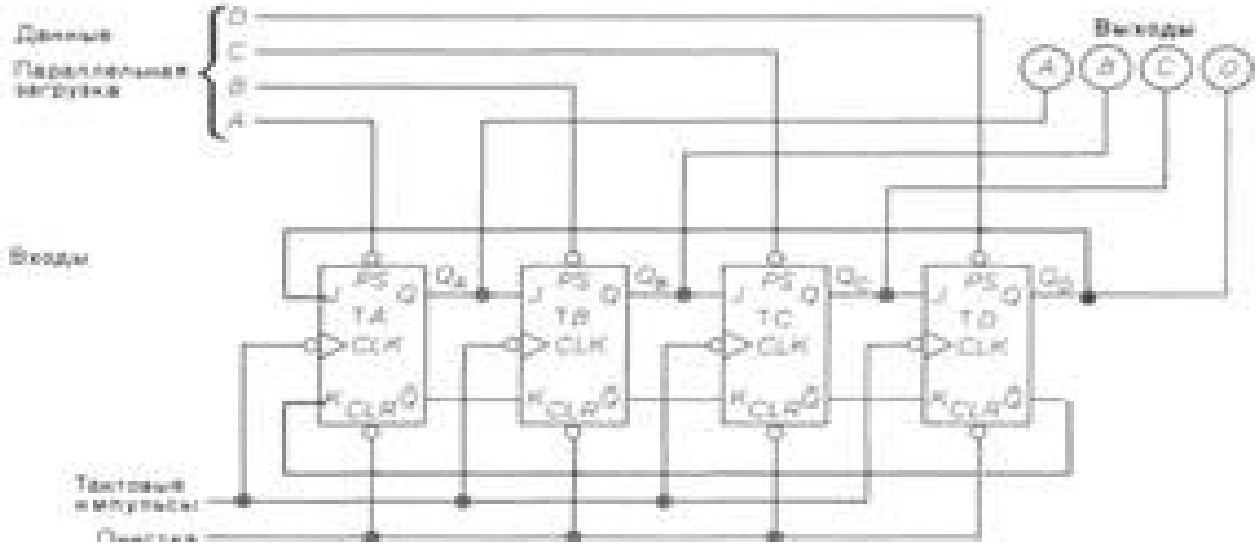
вносяться до цього реєстру крок за кроком. Наприклад, щоб занести в реєстр двійкову комбінацію 0111, необхідно пройти всю послідовність станів від рядка 1 до рядка 6 таблиці. 4.5. Послідовне завантаження 4-бітної комбінації 0111 в послідовний регістр відбувається за 5 циклів (рядок 2 можна опустити).

Іншим методом завантаження регістра є паралельне (або розширене) завантаження, при якому всі інформаційні біти вводяться в регістр одночасно «за командою» одного тактового імпульсу.

Регістр зсуву на рис. 4/14. можна перетворити на 5-розрядний, додавши до схеми ще один тригер D. Регістри зсуву частіше 4-, 5- або 8-розрядні. Вони можуть використовувати не тільки тригери типу D , але й інші типи тригерів (наприклад , тригери JK або синхронні тригери RS).

Послідовний регістр зсуву, робота якого описана вище, має два істотних недоліки: він дозволяє вводити лише один біт інформації для кожного тактового імпульсу, і, крім того, крайній правий біт втрачається при кожному зсуві вправо.

На рис. 4.15. показана схема 4-розрядного паралельного кільцевого регістра. Входи A, B, C, D у цьому пристрої призначені лише для ознайомлення.



Малюнок 4.15 . 4-розрядний паралельний кільцевий регістр зсуву

Ця система може бути оснащена ще однією корисною функцією - можливістю кругового потоку інформації, коли дані з входу пристрою будуть повертатися на його вхід і не будуть втрачені.

Цей регістр зсуву використовує чотири тригери JK. Зверніть увагу на зворотний зв’язок між виходом тригера TD та входами J і K тригера TA . Завдяки цій схемі зворотного зв'язку в регістр подається інформація, яка зазвичай втрачається на виході тригера TD, який буде циркулювати навколо регістра зсуву. Сигналом для очищення регістра (встановлення його виходів на 0000) є рівень логічного нуля на вході CLR.

Входи паралельного навантаження A, B, C, D підключені до входів налаштування тригера PRN, що дозволяє встановлювати рівень логічної одиниці на будь-якому виході (A, B, C, D). Якщо логічний 0 застосовано до одного з цих входів, тоді відповідний вихід буде логічна 1. Застосування тактових імпульсів до входів CLK всіх тригерів JK спричиняє зміщення інформації в регістрі вправо. Від тригера TD дані надсилаються до тригера TA (круговий потік інформації).

Таблиця 4.6. Робота 4-розрядного паралельного кільцевого регістра зсуву

Увійдіть							спускатися			
Номер черги	прибирання	Паралельне завантаження даних				Число тактових імпульсів	TA	туберкульоз	співвласник	TD
		I	Б	C	Д		I	Б	C	Д
1	1	1	1	1	1	0	1	1	1	0
2	0	1	1	1	1	0	0	0	0	0
3	1	1	0	1	1	0	0	1	0	0
4	1	1	1	1	1	1	0	0	1	0
5	1	1	1	1	1	2	0	0	0	1
6	1	1	1	1	1	3	1	0	0	0

7	1	1	1	1	1	4	0	1	0	0
8	1	1	1	1	1	5	0	0	1	0
9	0	1	1	1	1	0	0	0	0	
10	1	1	0	0	1	0	1	1	0	
11	1	1	1	1	1	6	0	0	1	1
12	1	1	1	1	1	7	1	0	0	1
13	1	1	1	1	1	8	1	1	0	0
14	1	1	1	1	1	9	0	1	1	0
15	1	1	1	1	1	10	0	0	1	1

Таблиця 4.6. допомагає зрозуміти, як працює регістр паралельного зсуву. Після включення живлення на виходах регістра встановлюється будь-яка двійкова комбінація , наприклад у рядку 1 таблиці. Застосування логічного 0 до входів тригера CLR ініціює очищення регістру (рядок 2). Потім (рядок 3) він завантажується в регістр двійкової комбінації 0100. Наступні тактові імпульси зсувають введену інформацію вправо (рядки 4-8). Треба звернути увагу на рядки 5 і 6: одиниця з крайнього правого тригера TD переноситься на крайній лівий тригер TA. У цьому випадку можна говорити про круговий рух агрегату в регістрі.

очищення реєстру ініціюється за допомогою запису CLR. Нова двійкова комбінація 0110 завантажується (рядок 10). Надання 5 тактових імпульсів (рядки 11-15) змушує інформацію циклічно зміщуватися на 5 позицій вправо. Слід зазначити, що для відновлення даних у початковий стан необхідні 4 тактових імпульсу. Якщо в регістрі зсуву на рис. 4.15 розірвати ланцюг зворотного зв'язку, тоді ми отримаємо звичайний паралельний регістр зсуву; можливість циклічного потоку даних буде виключена.

Програма реалізації 4-розрядного послідовного регістра зсуву мовою AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
Назва «реєстр1»;
Параметри
(ШИРИНА = 4); - встановлення швидкості передачі регістра
Assert (WIDTH > 0) - перевірка розрядності регістра на відмінність від 0
звіт « Цінність с ШИРИНА параметр маю Бути більше потім %" WIDTH
Серйозність помилка ;
Реєстр підпроектів1
(
D_INPUT, SET, RESET: вхід = GND; - вхідні сигнали
CLK: вхід ; - вхід синхронізації
ON: вхід = VCC; - внесення дозволу на роботу
Q_OUTPUT: вихід ; - вихідні сигнали
)
Змінний
FF [ШИРИНА..1]: DFFE; - оголошення змінної FF, що належить до класу DFFE
Почніть
FF[]. clk = clk;
FF[]. prn = !SET;
FF[]. clrn = !СКИДАННЯ;
FF[]. ena = УВІМКНУТИ;
FF[].d = (FF[WIDTH-1..1].q, D_INPUT);
Q_ВИВІД = FF[ШИРИНА].q;
кінець _
```

Програма, яка реалізує 4-розрядний паралельний кільцевий регістр зсуву на мові AHDL в інтегрованому середовищі MAX+PLUS II, виглядає наступним чином:

```
Реєстр підпроектів2
    Параметри
```

```
(     ШИРИНА = 4); - встановлення швидкості передачі регістра
      Assert (WIDTH > 0) - перевірка бітрейту запуску (більше нуля)
    Звіт « Значення Параметр WIDTH _ маю Бути більше потім %" WIDTH
    Серйозність помилка ;
(
I[WIDTH..1]: вхід = VCC; - вхідні сигнали (дані)
    CLK: вхід ;
    RESET: вхід ;
    O[ШИРИНА..1]: вихід ; - вихідні сигнали (дані)
)
Змінний
    FF[ШИРИНА..1]: JKFF; - оголошення змінної FF, що належить до класу
JKFF
Почніть
    FF[ШИРИНА..1].j = (FF[ШИРИНА-1..1].q, FF[ШИРИНА].q);
    FF[ШИРИНА..1].k = (!FF[ШИРИНА-1..1].q, !FF[ШИРИНА].q);
    FF[ШИРИНА..1]. clk = clk;
    FF[ШИРИНА..1]. clrn = !СКИДАННЯ;
    FF[ШИРИНА..1]. prn = I[ШИРИНА..1];
    O[ШИРИНА..1] = FF[ШИРИНА..1].q;
кінець _
```

На рис. 4.16. буде показано вікно редактора сигналів проекту Register1.

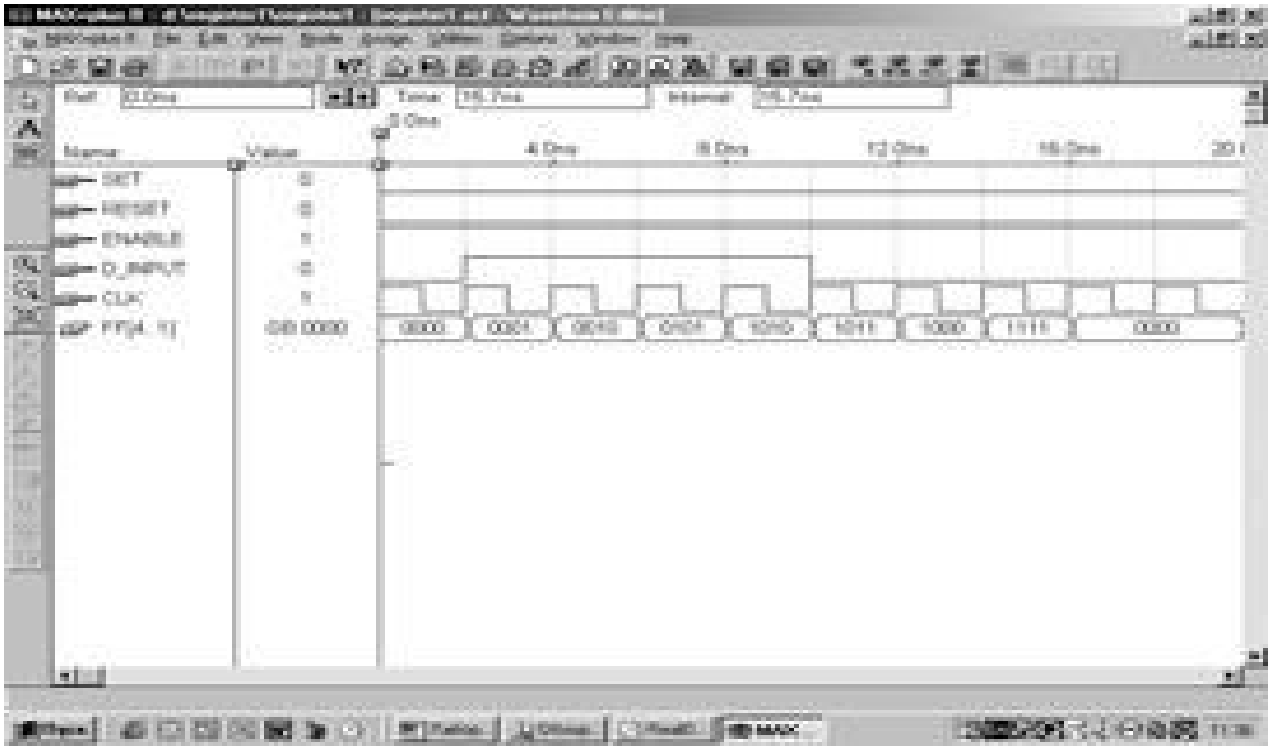


Рис. 4.16 . Результати перевірки 4-розрядного послідовного регістра зсуву

На рис. 4.17. буде показано вікно редактора сигналів проекту Register2.

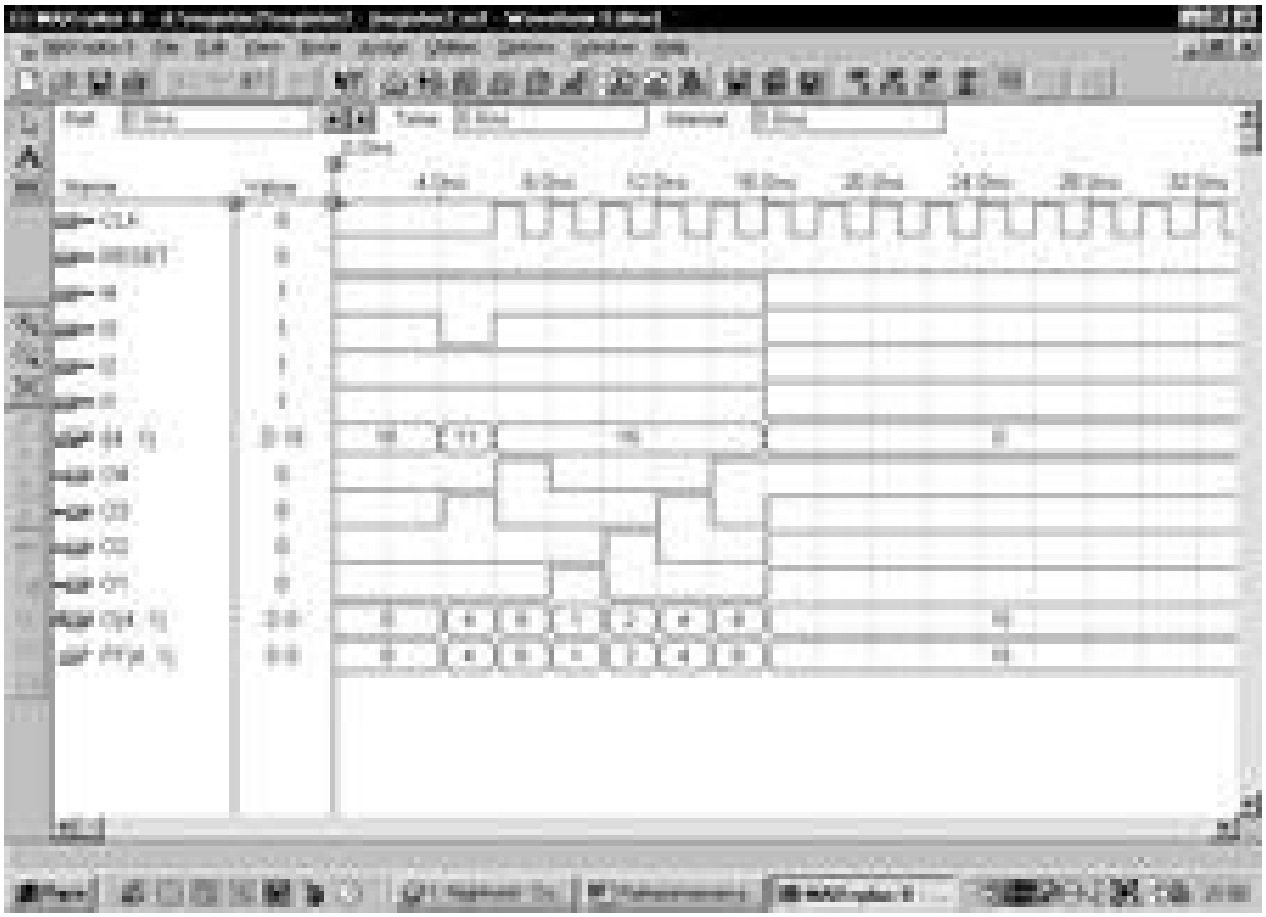


Рис. 4.17. Результати тестування для 4-розрядного паралельного кільцевого регістра зсуву

4.6. Теоретичні відомості про лічильники

Процедури для двійкових і десяткових чисел наведено в таблиці. 4. 7. Використовуючи лише чотири двійкові цифри (T4, T3, T2, T1), ми можемо порахувати від 0000 до 1111 (від 0 до 15 у десятковій системі). Стовпець T1 таблиці відповідає двійковому розряду одиниці або молодшому розряду. Зазвичай використовується термін «молодший клас». Стовпець T4 відповідає двійковій цифрі вісімки або старшій значущій цифрі. Зазвичай використовується термін «найстарший клас». Зверніть увагу, що найчастіше змінюються цифри в колонці одиниць вимірювання. Якщо ви хочете, щоб лічильник рахував від 0000 до 1111 у двійковому форматі, він повинен мати 16 різних вихідних станів. Такий лічильник називається модульним лічильником 16. Модуль лічильника — це кількість різних станів, які проходить лічильник протягом одного повного циклу підрахунку.

Таблиця 4. 7. Порядок підрахунку лічильника за модулем 16

Двійкове число				десяткове число
T4	T3	T2	T1	
8	4	2	1	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8

1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Функціональна схема лічильника модуля 16, що складається з чотирьох тригерів JK, наведена на рис. 4.18. Скін JK-тригера працює в режимі перемикання (J = K = 1). Нехай стан виходів лічильника відповідає двійковому числу 0000 на початку години - лічильник очищення. Коли тактовий імпульс 1 надходить на синхронізуючий вхід CLK тригера T1, тригер перемикається, після проходження сегмента імпульсу , і на індикаторі з'являється число 0001. Повертаємось до таблиці. 4.7 . ми бачимо, що цифри (1 або 0) у стовпці T1 (одиниці) змінюються з кожним кроком підрахунку. Це означає, що тригер T1 перемикається, коли надходить новий тактовий імпульс шкіри. Як видно зі стовпця T2, тригер T2 перемикається вдвічі частіше, ніж тригер T1. Загалом, шкіра старішого класу в столі. 4.7. перемикається вдвічі частіше, ніж раніше.

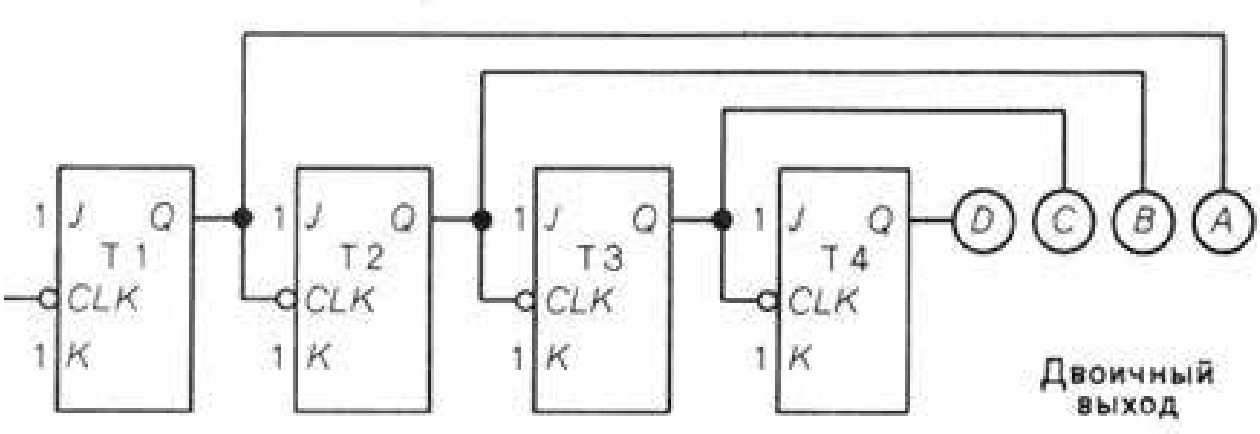


Рис. 4.18 . Логічна схема лічильника по модулю 16

Роботу лічильника за модулем 16 ілюструють погодинні графіки на рис. 4.18 . Верхня діаграма відповідає входу синхронізації. Нижче наведено діаграми виходів Q регістрів T4, T3, T2, T1.

Оскільки кожен тригер впливає лише на наступний тригер, перемикання всіх тригерів займає кілька годин.

Ми бачимо, що зміна стану проходить послідовно через ланцюжок тригерів. Тому лічильник, який ми розглядаємо, називається лічильником крос-керрі.

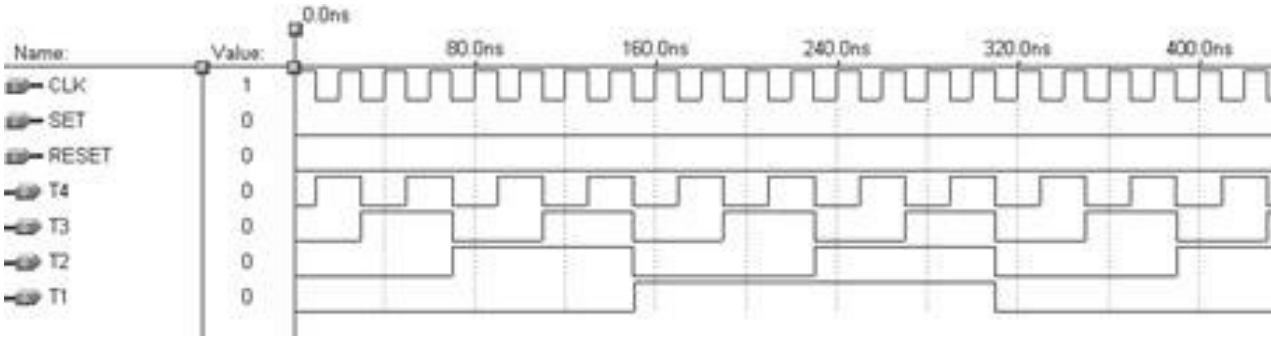


Рис. 4.19. Схема таймера Modulo 16

Лічильник, функціональна схема якого наведена на рис. 4.19. , можна назвати не тільки лічильником переносу, але також лічильником за модулем 16, 4-розрядним лічильником або асинхронним лічильником. Назва skin характеризує схему, що розглядається, з одного боку.

Визначення «перехресного обслуговування» та «асинхронного» означають, що тригери не спрацьовують одночасно. Назва «лічильник модуля 16» відображає кількість різних станів, які «проходить» лічильник за один повний цикл підрахунку. Визначення «4 біти» вказує на кількість двійкових цифр у виході лічильника.

Лічильник по модулю 10 рахує від 0000 до 1001 (від 0 до 9 в десятковій системі числення), тобто перші 10 комбінацій з таблиці. 4.7. Ми бачимо, що для цього потрібні чотири двійкові біти: біт одиниці, біт двійки, біт четвірки та біт вісімки.

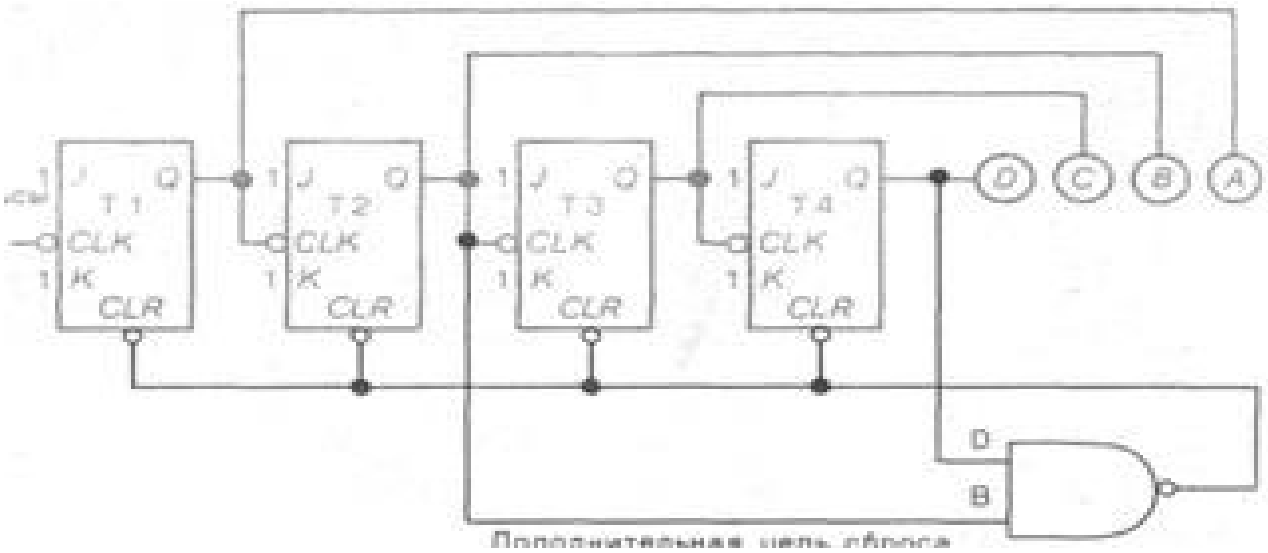


Рис. 4.20. Логічна схема лічильника по модулю 10

Такий лічильник може бути реалізований на чотирьох тригерах, з'єднаних за описаною вище схемою асинхронного лічильника. На схемі слід ввести додатковий логічний елемент ІН І для скидання всіх тригерів на нуль і очищення лічильника з надходженням десятого імпульсу (тобто з надходженням першого імпульсу після підрахунку лічильника до 1001 - 9 в десятковій системі). система).

Принцип використання такого логічного елемента стає зрозумілим, якщо розглянути, яке двійкове число стоїть після 1001. З табл. 4.7. ви бачите, що число дорівнює 1010 (10 у десятковій дробі). Якщо на вході логічного елемента І подається логічна 1, що міститься в двох і восьми цифрах двійкового числа 1010, цей елемент переведе всі тригери в стан 0. Лічильник почне відлік від 0000 до 1001. Таким чином, логічний елемент І НЕ дозволяє встановити лічильник у стан 0000. Подібне використання логічного елемента І-НЕ дозволяє створювати лічильники з іншими значеннями модуля. На рис. 4.20. показана функціональна схема асинхронного лічильника за модулем 10. Цей лічильник також можна назвати декадним (десятковим).

Прямі лічильники (0, 1, 2,...) описані вище. Однак у деяких цифрових системах необхідно рахувати у зворотному напрямку (9, 8, 7, 6,...).

Лічильники, які лічать від більших чисел до менших, називаються лічильниками віднімання або зворотними лічильниками.

Схема асинхронного лічильника віднімань за модулем 8 наведена на рис. 4.21, відповідну послідовність підрахунку чисел подано в таблиці. 4.7.

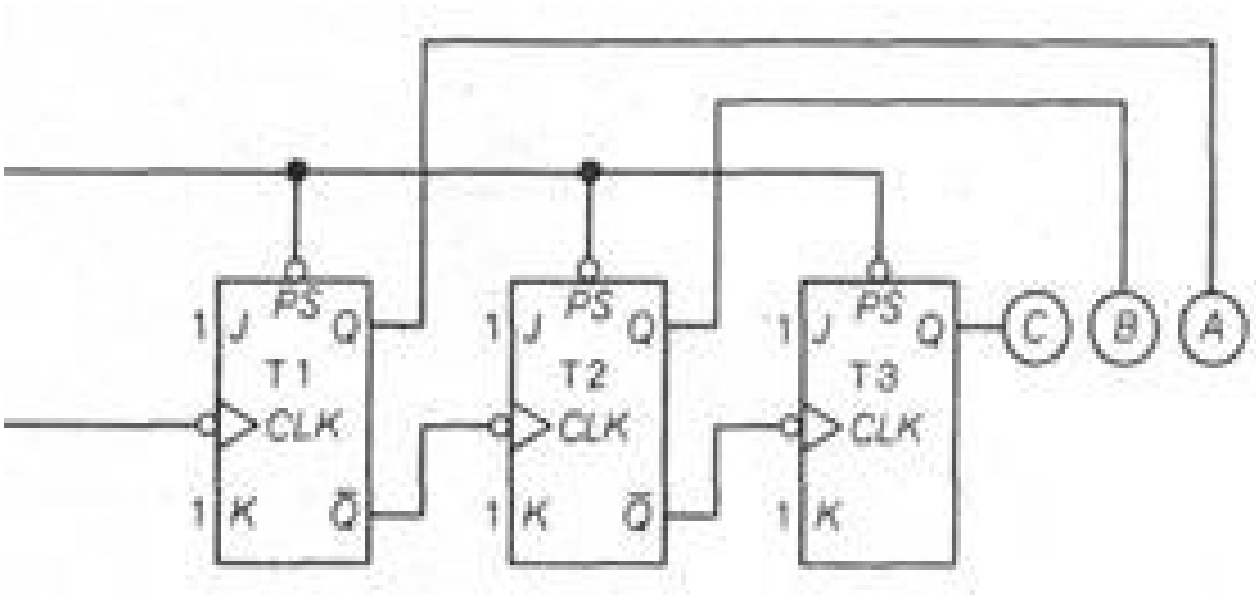


Рис. 4.21 . Логічна схема асинхронного 3-розрядного лічильника віднімання

Таблиця 4.8. Послідовність підрахунку для 3-розрядного лічильника віднімання

Номер тактовий імпульс	Двійкова послідовність підрахунку			Десяткові числа
	T3	T2	T1	
0	1	1	1	7
1	1	1	0	6
2	1	0	1	5
3	1	0	0	4
4	0	1	1	3

5	0	1	0	2
6	0	0	1	1
7	0	0	0	0
8	1	1	1	7
9	1	1	0	6

Зверніть увагу, що схема лічильника віднімання нагадує схему лічильника прямої дії на рис. 4.18 . Єдина відмінність полягає в методі передачі від тригера T1 до тригера T2 і від тригера T2 до тригера T3. У прямому лічильнику синхронізуючий вхід зовнішнього тригера з'єднаний з інверсним Q-виходом попереднього тригера . Зауважте, що лічильник зворотного відліку спочатку встановлюється на 111 (десяткове число 7) через вхід налаштування (PRN) перед його запуском. Тригер T3 – це лічильник цифр двійкових одиниць (стовпець T1). Тригер T2 є двійковим лічильником (стовпець T2). Тригер T3 — лічильник четвірок (стовпець T3).

Програма реалізації 4-розрядного асинхронного лічильника з передачею по модулю 16 мовою AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
заголовок «лічильник1»;
        параметри
                (ШИРИНА = 4); - встановлення розряду лічильника
        assert (WIDTH > 0) - перевірка лічильника швидкості передачі
        Звіт " Вартість". Параметр WIDTH _ маю Бути більше потім %" WIDTH
        серйозність помилка _
        лічильник підпроекту1
(
        CLK: вхід ; - лічильник імпульсів синхронізації (CLK).
```

```
        RESET: вхід ; - імпульси скидання (CLRN) тригерів лічильника
        SET: вхід ; - програмовані імпульси (ПРН) тригерів лічильників
        O [ШИРИНА..1]: вихід ; - вихід лічильника
)
мінливий
        ТРИГЕР [ШИРИНА..1]: JKFF;
почати
        TRIGGER [WIDTH..1].j = vcc ;
        TRIGGER [WIDTH..1].k = vcc ;
        ТРИГЕР [ШИРИНА..1]. clrn = !СКИДАННЯ;
        ТРИГЕР [ШИРИНА..1]. prn = !SET;
        ТРИГЕР [ШИРИНА]. clk = !CLK;
        ТРИГЕР [ШИРИНА-1..1]. clk = !TRIGGER [WIDTH..2].q;
        O [WIDTH..1] = TRIGGER[1..WIDTH].q;
кінець _
```

Програма реалізації асинхронного лічильника модулем 10 мовою AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
заголовок "лічильник2";
        параметри
                (ШИРИНА = 4); - встановлення розряду лічильника
                assert (WIDTH > 0) - перевірка лічильника швидкості передачі
                Звіт " Вартість". Параметр WIDTH _ маю Бути більше потім %"
        WIDTH
        серйозність помилка _
        лічильник підпроекту2
(
```

```
        CLK: вхід ; - лічильник імпульсів синхронізації (CLK).
        SET: вхід ; - програмовані імпульси (ПРН) тригерів лічильників
        O [ШИРИНА..1]: вихід ; - вихід лічильника
)
мінливий
        ТРИГЕР [ШИРИНА..1]: JKFF;
почати
        TRIGGER [WIDTH..1].j = vcc ;
        TRIGGER [WIDTH..1].k = vcc ;
        ТРИГЕР [ШИРИНА..1]. prn = !SET;
        ТРИГЕР [ШИРИНА]. clk = !CLK;
        ТРИГЕР [ШИРИНА-1..1]. clk = !TRIGGER [WIDTH..2].q;
        ТРИГЕР [ШИРИНА..1]. clrn = (TRIGGER[3].q! & TRIGGER[1].q);
        O [WIDTH..1] = TRIGGER[1..WIDTH].q;
кінець _
```

Програма, яка реалізує 3-розрядний лічильник віднімання на мові AHDL в інтегрованому середовищі MAX+PLUS II, виглядає наступним чином:

```
назва "лічильник3";
        параметри
                (ШИРИНА = 3); - встановлення розряду лічильника
                assert (WIDTH > 0) - перевірка лічильника швидкості передачі
                Звіт " Вартість". Параметр WIDTH _ маю Бути більше потім %"
        WIDTH
                серйозність помилка ;
        лічильник підпроекту3
(
        CLK: вхід ; - лічильник імпульсів синхронізації (CLK).
        RESET: вхід ; - імпульси скидання (CLRN) тригерів лічильника
        SET: вхід ; - програмовані імпульси (ПРН) тригерів лічильників
```

```
    O[ШИРИНА..1]: вихід ; - вихід лічильника
)
мінливий
    ТРИГЕР [ШИРИНА..1]: JKFF;
почати
    TRIGGER [WIDTH..1].j = vcc ;
    TRIGGER [WIDTH..1].k = vcc ;
    ТРИГЕР [ШИРИНА..1]. clrn = !СКИДАННЯ;
    ТРИГЕР [ШИРИНА..1]. prn = !SET;
    ТРИГЕР [ШИРИНА..1]. clk = (CLK, !TRIGGER [WIDTH..2].q);
    O [WIDTH..1] = !TRIGGER [1..WIDTH].q;
кінець _
```

Програма реалізації 3-розрядного універсального лічильника мовою AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

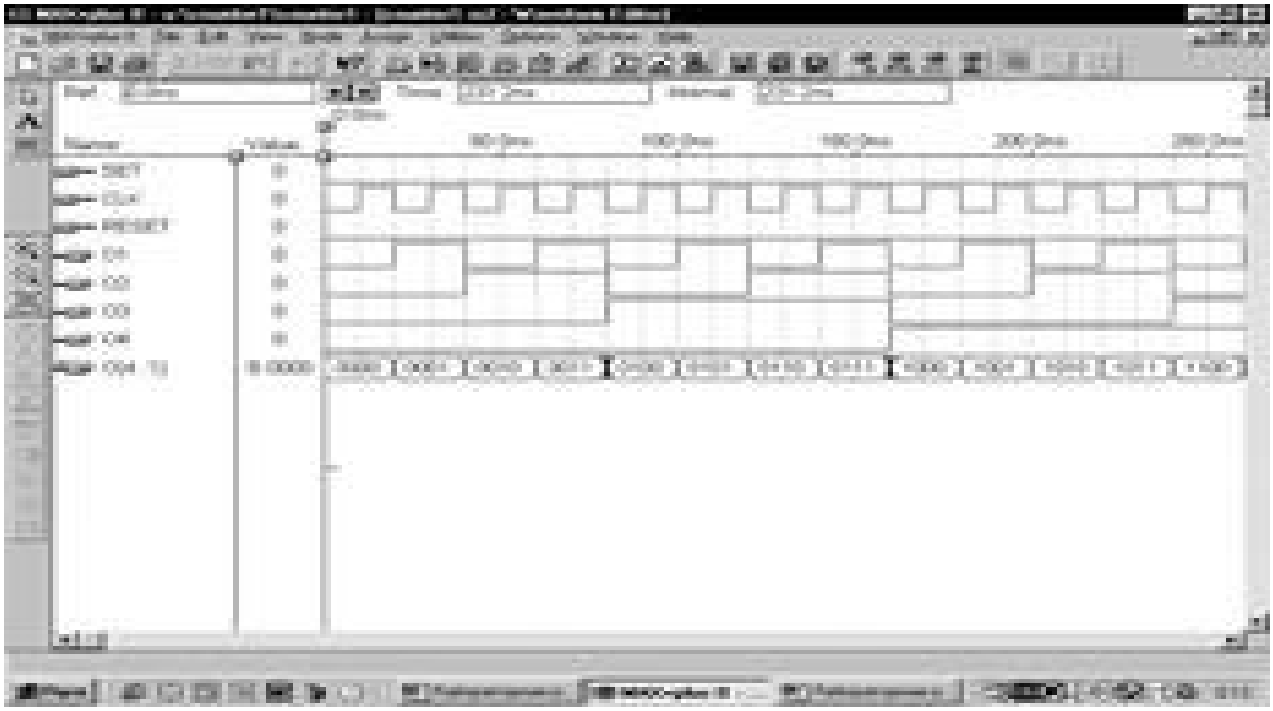
```
    назва «лічильник 4»;
    параметри
        (ШИРИНА = 3); - встановлення розряду лічильника
        assert (WIDTH > 0) - перевірка лічильника швидкості передачі
        Звіт " Вартість". Параметр WIDTH _ маю Бути більше потім %"
    WIDTH
        серйозність помилка ;
    лічильник підпроекту4
(
    CLK: вхід ; - лічильник імпульсів синхронізації (CLK).
    SET: вхід ; - програмовані імпульси (ПРН) тригерів лічильників
    RESET: вхід ; - імпульси скидання (CLRN) тригерів лічильника
    FWC: вхід ;
    BWC: вхід ;
    O[ШИРИНА..1]: вихід ; - вихід лічильника
```

```
)
мінливий
        ТРИГЕР [ШИРИНА..1]: JKFF;
почати
        TRIGGER[].j = vcc ;
        TRIGGER[].k = vcc ;
        ТРИГЕР []. prn = !SET;
        ТРИГЕР []. clrn = !СКИДАННЯ;
        ТРИГЕР [ШИРИНА]. clk = !CLK;
        ТРИГЕР [ШИРИНА-1..1]. clk = !((TRIGGER[WIDTH..2].q & !FWC) &
        !(!TRIGGER[WIDTH..2].q & !BWC));
        O[WIDTH..1] = TRIGGER[1..WIDTH].q;
кінець _
```

Примітка:

- Підраховуються вхідні значення, що відповідають вставці лічильника
- напрямок росту:
- FWC = 1;
- BWC = 0.
- Підраховуються вхідні значення, що відповідають вставці лічильника
- спадний напрямок:
- FWC = 0;
- BWC = 1.

На рис. 4.22. буде показано вікно редактора сигналів проекту counter1.



Малюнок 4.22. Результати тестування 4-розрядного асинхронного лічильника з переносом за модулем 16

На рис. 4.23. буде показано вікно редактора сигналів проекту counter2.

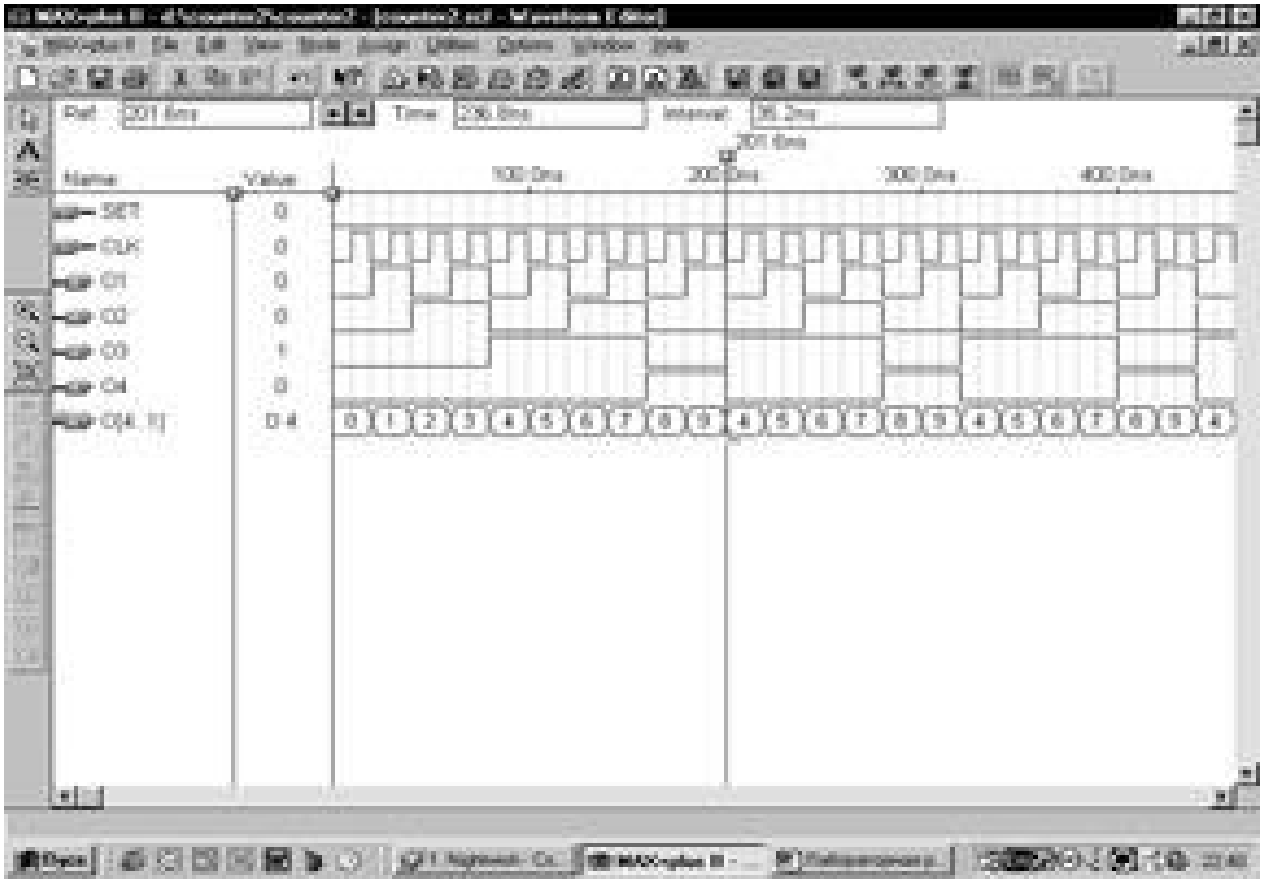


Рис. 4.23 . Результати перевірки асинхронного лічильника за модулем 10

На рис. 4.24. буде показано вікно редактора сигналів проекту counter3.

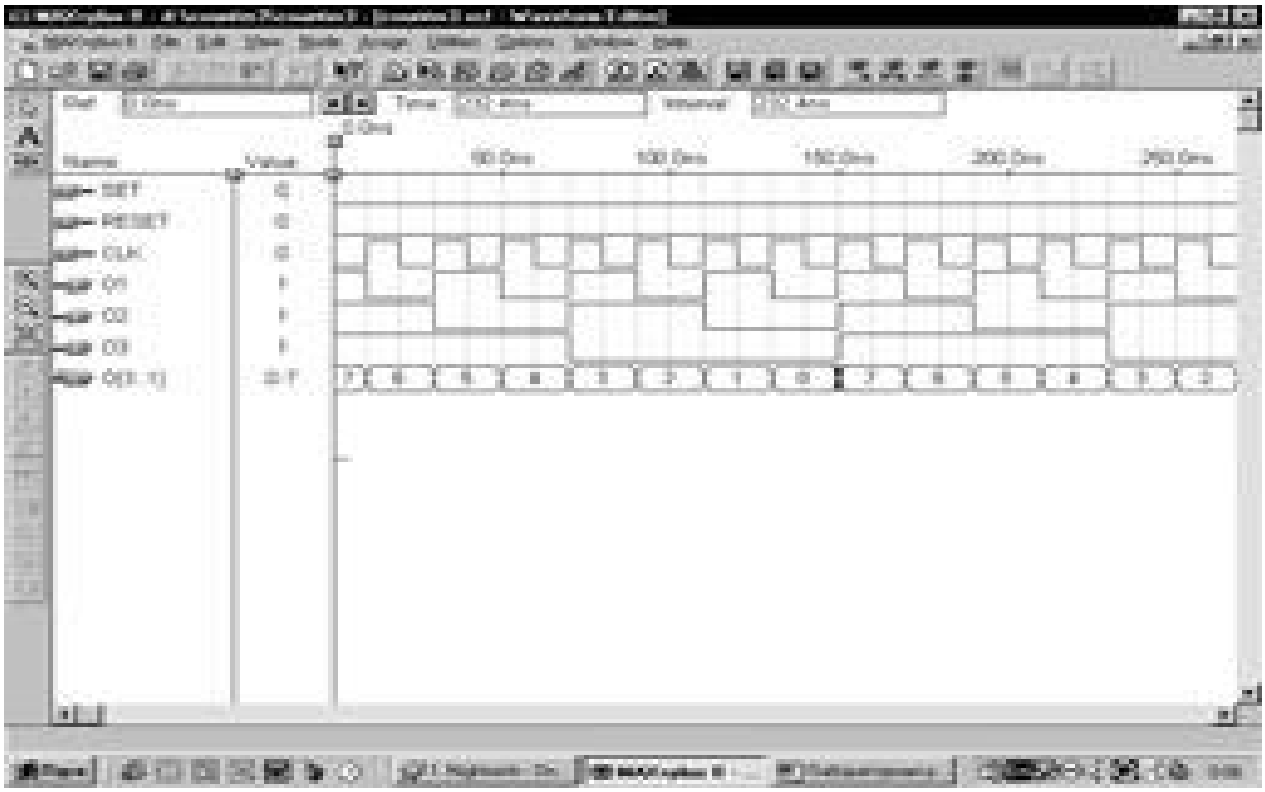


Рис. 4.24 . Результати тестування 3-розрядного синхронного лічильника

На рис. 4.25. буде показано вікно редактора сигналів проекту counter4.

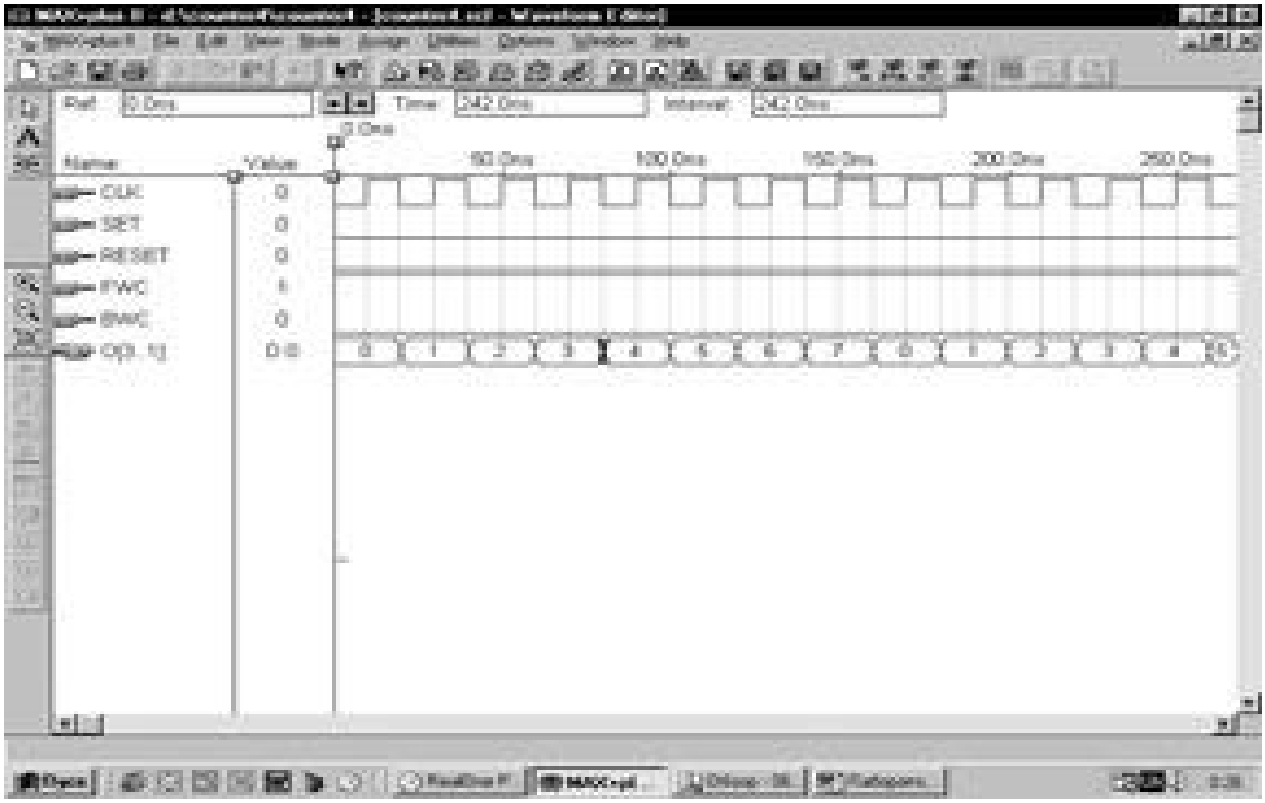


Рис. 4.25 . Результати тестування 3-розрядного універсального лічильника

4.7. Теоретичні відомості про мультиплексори , демультиплексори , кодери, декодери

Мультиплексор — це комбінаційний логічний пристрій, призначений для керування передачею даних від кількох джерел в один вихідний канал.

Типовим застосуванням мультиплексорів є передача даних від кількох просторово рознесених джерел інформації на вхід приймача. Припустимо, що температура навколишнього середовища вимірюється в кількох кімнатах і результати цих вимірювань потрібно вивести на комп'ютер. У той же час, оскільки температура змінюється повільно , немає необхідності вимірювати її постійно, щоб отримати достатню точність. І достатньо того, що вимірювання будуть проводитися з рівними погодинними інтервалами. Найважливішим є те, що інтервали між двома вимірюваннями значно коротші, ніж постійна година, яка характеризує зміну температури в контрольованому приміщенні. Цю функцію, тобто підключення різних джерел інформації до одного приймача за заданою командою, виконує мультиплексор . Він перетворює інформацію, розкидану в просторі, на зображення поперечного перерізу протягом години.

Відповідно до визначення, мультиплексор повинен мати один вихід і дві групи входів: інформаційні та адресні. Код, застосований до адресних входів, визначає, який з інформаційних входів у даний момент підключений до вихідного контакту. Оскільки n-розрядний двійковий код може приймати $2n$ значень, якщо кількість адресних входів мультиплексора дорівнює n, то кількість його інформаційних входів має бути 2^n.

Таблиця істинності, яка ілюструє роботу мультиплексора з двома адресними входами, виглядає наступним чином (табл. 7.4.1).

Таблиця 4.9. Таблиця істинності для мультиплексора з двома адресованими входами

для мене	1 _	0 _	Q	$\overline{Q}$

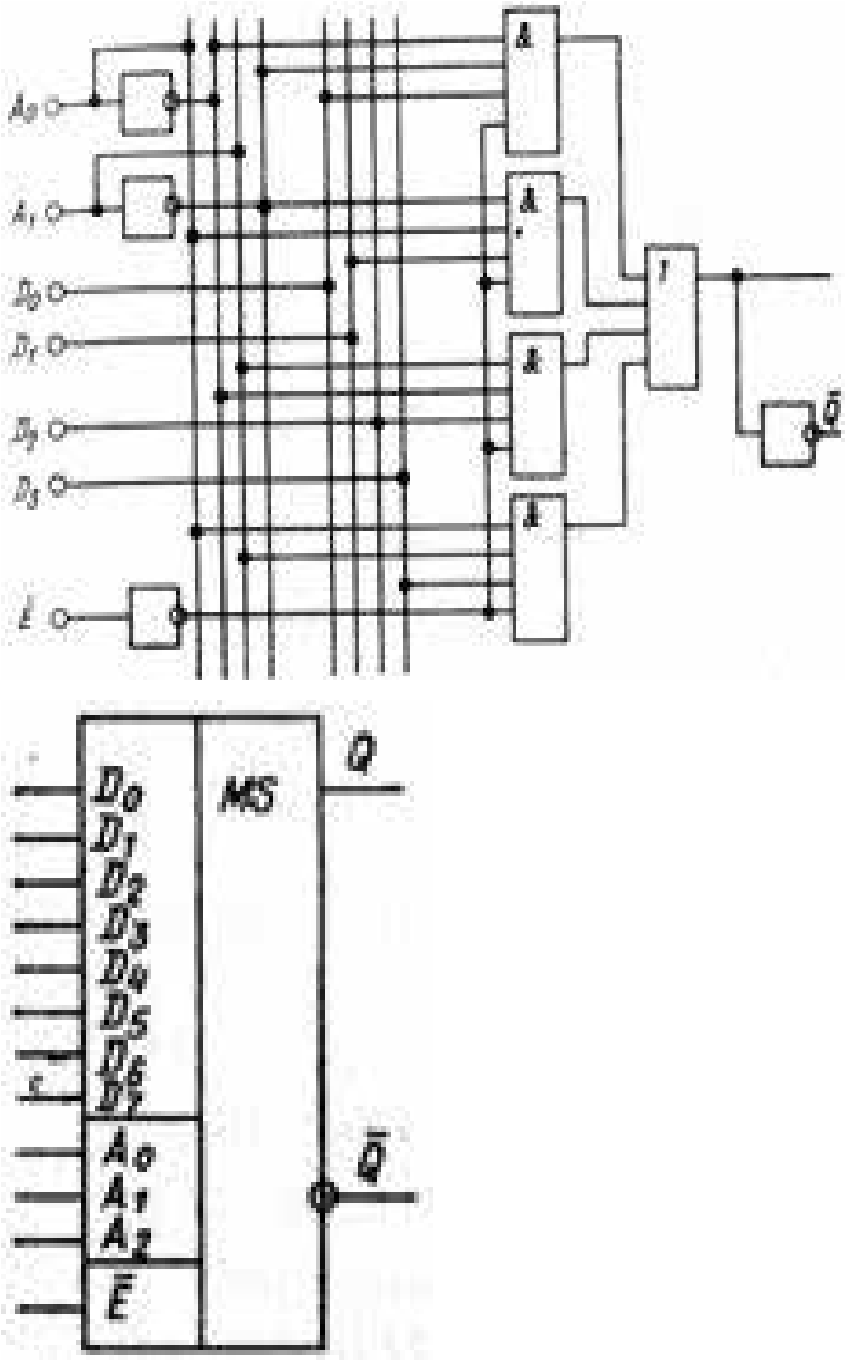
1	X	X	0	1
0	0	0	D_0	$\overline{D}_0$
0	0	1	D_1	$\overline{D}_1$
0	1	0	D_2	$\overline{D}_2$
0	1	1	D_3	$\overline{D}_3$

Ця таблиця враховує, що мультиплексор зазвичай має додатковий зворотний вихід $\overline{Q}$ і вхід, що дозволяє виконувати операцію Е (у програмах AHDL вхід для операції, у прикладі примітивних тригерів, називається ENA). Якщо на вхід дозволу Е подається активний логічний сигнал (Е=1), вихідний сигнал є константою мультиплексора і не залежить від його вхідних сигналів.

Функція логічної алгебри, що описує роботу мультиплексора, має вигляд:

$$O = D_0 \overline{A}_1 \overline{A}_0 \overline{E} + D_1 \overline{A}_1 A_0 \overline{E} + D_2 A_1 \overline{A}_0 \overline{E} + D_3 A_1 A_0 \overline{E}.$$

Логічна схема мультиплексора , що відповідає заданій функції логічної алгебри та умовному позначенню мультиплексора , на прикладі інтегральної мікросхеми 555КП7 наведена на рис. 2. 4.27. а, б



І) Б)

Рис. 4.27 . Логічна схема мультиплексора (а) та її умовне графічне позначення (б)

При передачі інформації з кількох джерел по звичайному каналу з тимчасовою розкладкою потрібні не тільки мультиплексори , а й пристрої зворотної дії, які розподіляють отриману з одного каналу інформацію між декількома приймачами. Цю задачу вирішують демультиплексори .

середовищі MAX+PLUS II з використанням мови AHDL мультиплексор можна описати двома способами:

1) таблиця істинності;

2) на поведінковому рівні.

Опис пристрою за допомогою таблиці істинності є найпростішим, оскільки він вимагає від розробника знання лише таблиці істинності мультиплексора . Обсяг отриманої програми порівняно з обсягом програми, описаної на поведінковому рівні, значно менший, але сама архітектура (логічна схема) пристрою залишається невідомою для проектувальника.

Спеціаліст вибирає метод опису, виходячи з технічного завдання, заданого обсягу програми, кількості елементарних вентилів на мікросхемі та власного досвіду.

У цій роботі наведено приклади опису мультиплексора , який має два адресних входи, чотири інформаційних входи та один дозволяючий вхід як за допомогою таблиці істинності, так і на поведінковому рівні.

Демультиплексор — це комбінаційний логічний пристрій, призначений для керування передачею даних від одного джерела інформації до кількох вихідних каналів. Згідно з визначенням, демультиплексор в загальному випадку має один інформаційний вхід, n адресних входів і 2^n виходів. Таблиця істинності, що описує роботу демультиплексора з двома адресними входами і входом, що дозволяє операцію E, має вигляд (табл. 4.10):

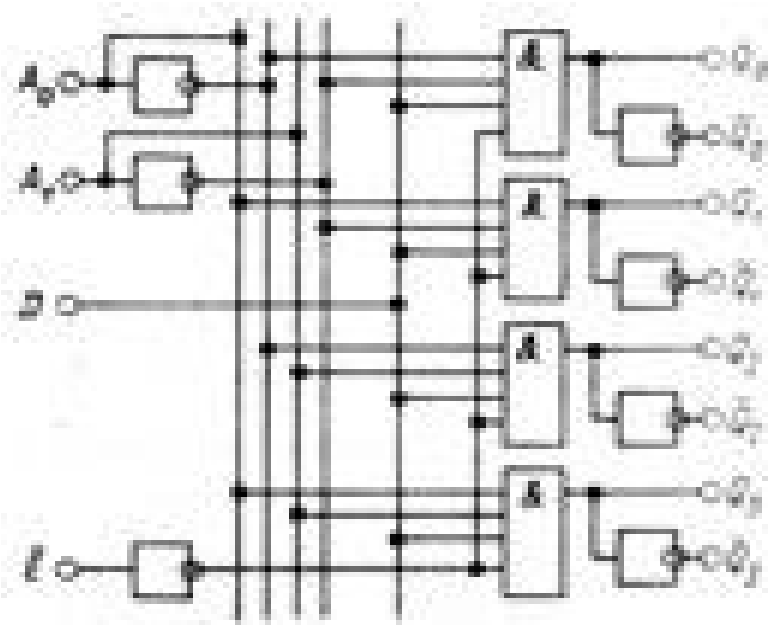
Таблиця 4.10 . Таблиця істинності для демультиплексора з двома адресованими входами

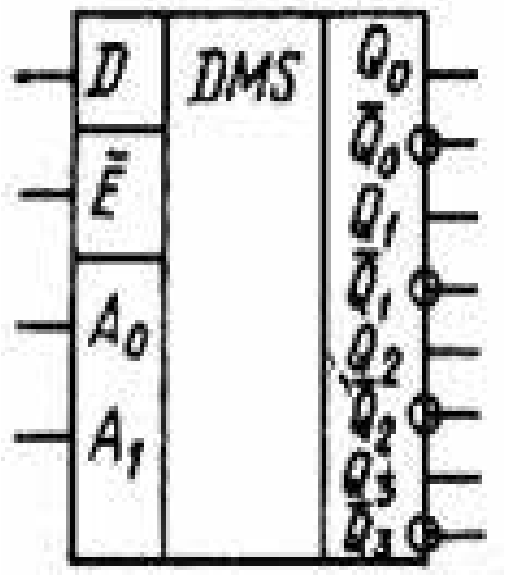
для мене	1 _	0 _	Питання $_0$	Q1 _	Питання $_2$	Q3 _
1	X	X	0	0	0	0
0	0	0	Д	0	0	0
0	0	1	0	Д	0	0
0	1	0	0	0	Д	0
0	1	1	0	0	0	Д

Цій таблиці відповідає така функція логічної алгебри:

$$Q_0 = D\overline{A}_1\overline{A}_0\overline{E} = \overline{D} \downarrow A_1 \downarrow A_0 \downarrow E,$$
$$Q_1 = D\overline{A}_1 A_0\overline{E} = \overline{D} \downarrow A_1 \downarrow \overline{A}_0 \downarrow E,$$
$$Q_2 = DA_1\overline{A}_0\overline{E} = \overline{D} \downarrow \overline{A}_1 \downarrow A_0 \downarrow E,$$
$$Q_3 = DA_1A_0\overline{E} = \overline{D} \downarrow \overline{A}_1 \downarrow \overline{A}_0 \downarrow E. \quad (2)$$

На рис. 4.28. і наведена логічна схема демультиплексора, що виконує функцію логічної алгебри (2), а на рис. 4.28б наведено її умовне графічне зображення.





І) Б)

Рис . 4.28 . Логічна схема демультиплексора (а) і його умовне графічне позначення (б)

Комбінаційний логічний пристрій для перетворення чисел з десяткової системи в двійкову називається шифратором або шифратором. Входам кодера послідовно призначаються значення десяткових чисел, тому подача активного логічного сигналу на один із входів сприймається кодувальником як надання відповідного десяткового числа. Цей сигнал перетворюється на вихідний сигнал кодера у двійковому коді . Згідно зі сказаним, якщо кодер має n виходів, то його входів не повинно бути більше ^{2n} . Кодувальник з 2n $^{\text{входами}}$ і виходами називається повним. Якщо кількість входів кодера менше 2n , то він називається неповним.

Розглянемо роботу шифру на прикладі перетворювача десяткових чисел від 0 до 9 в двійково -десятковий код. Відповідна таблиця істинності в цьому випадку (табл. 4.11).

Оскільки кількість входів цього пристрою менше $2^n = 16$, ми маємо неповний кодер. Використовуючи таблицю для Q_3, Q_2, Q_1, Q_0, можна записати такі вирази:

$$Q_3 = x_8 + x_9;$$

$$Q_2 = x_4 + x_5 + x_6 + x_7;$$

$$Q_1 = x_2 + x_3 + x_6 + x_7;$$

$$Q0 = x1 + x3 + x5 + x7 + x9$$

Таблиця 4.11 . Таблиця істинності для десяткового конвертера від 0 до 9 у двійковий

x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	Q3 _	Питання 2	Q1 _	Питання 0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1

0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

Отримана система (3) характеризує роботу шифру. Логічна схема пристрою, що відповідає системі (3), наведена на рис. 4.29 .

Легко помітити, що в цьому типі перетворювача сигнал, що подається на вхід x_0, не використовується. Отже, немає сигналу на жодному вході $x_0 x_1$ інтерпретується діаграмою як наявність нульового сигналу.

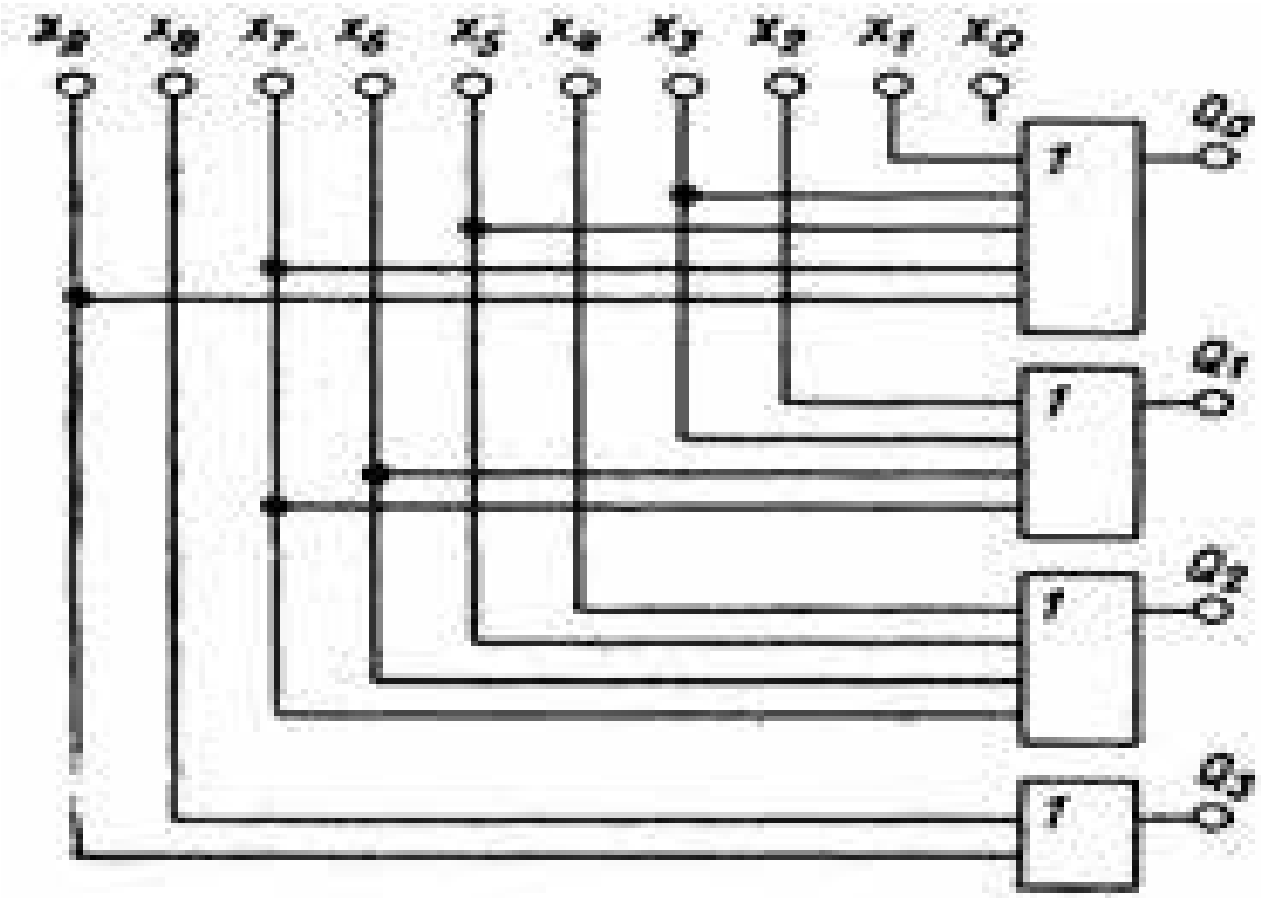


Рис . 4.29 . Логічна схема десяткового шифратора

Основним напрямком використання шифру в цифрових системах є введення вихідної інформації з клавіатури.

Після натискання будь-якої клавіші на відповідний вхід кодера надходить сигнал «логічна одиниця», який потім перетворюється в двійково - десятковий код. Можливість введення інформації показано на рис. 2. 4.30 .

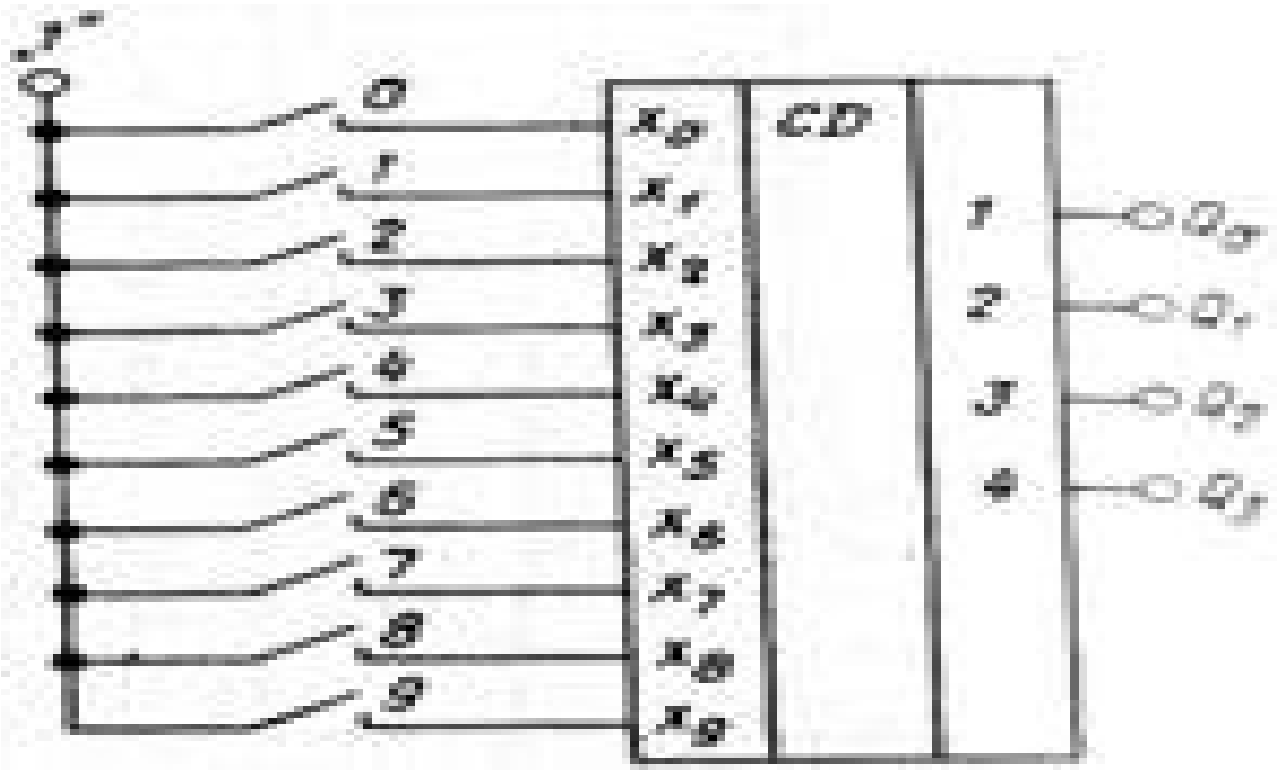


Рис. 4.30 Пристрій введення з клавіатури

У середовищі MAX+PLUS II , що використовує мову AHDL, кодер можна описати двома способами:

- таблиця істинності , емульована за допомогою оператора CASE;
- на поведінковому рівні.

Опис пристрою, емулюваного за допомогою оператора CASE, є найпростішим, оскільки він вимагає від розробника знання лише таблиці істинності кодера. Обсяг отриманої програми порівняно з обсягом програми, описаної на поведінковому рівні, значно менший, але сама архітектура (логічна схема) пристрою залишається невідомою для проектувальника.

Спеціаліст вибирає метод опису, виходячи з технічного завдання, заданого обсягу програми, кількості елементарних вентилів на мікросхемі та власного досвіду.

Декодер — це комбінаційний логічний пристрій, який використовується для перетворення чисел із двійкової системи в десяткову. За визначенням декодер належить до класу перетворювачів коду. Зрозуміло, що кожне двійкове число узгоджується з сигналом, який генерується на виході пристрою. Таким чином дешифратор виконує операцію, зворотну шифру. Якщо кількість адресних входів дешифратора віднести до числа його виходів m співвідношенням $m = 2^n$, то дешифратор називається повним. У протилежному випадку, якщо $m < 2^n$, дешифратор називається неповним.

Поведінка декодера описується таблицею істинності, подібною до таблиці істинності кодера (див. систему 3), але в цій таблиці вхідний і вихідний сигнали інвертовані. Відповідно до цієї таблиці, оскільки вихідний сигнал дорівнює 1 тільки на одному окремому наборі вхідних змінних, тобто для однієї одиничної композиції, алгоритм декодера описує систему рівнянь у вигляді:

$$x_0 = \overline{Q}\,\overline{Q}\,\overline{Q}\,\overline{Q};$$

$$x_1 = \overline{Q}\,\overline{Q}\,\overline{Q}\,Q;$$

$$x_2 = \overline{Q}\,\overline{Q}\,Q\,\overline{Q};$$

і так далі, де Q_i - значення логічної змінної на і-му вході пристрою.

У загальному випадку система (4) має вигляд:

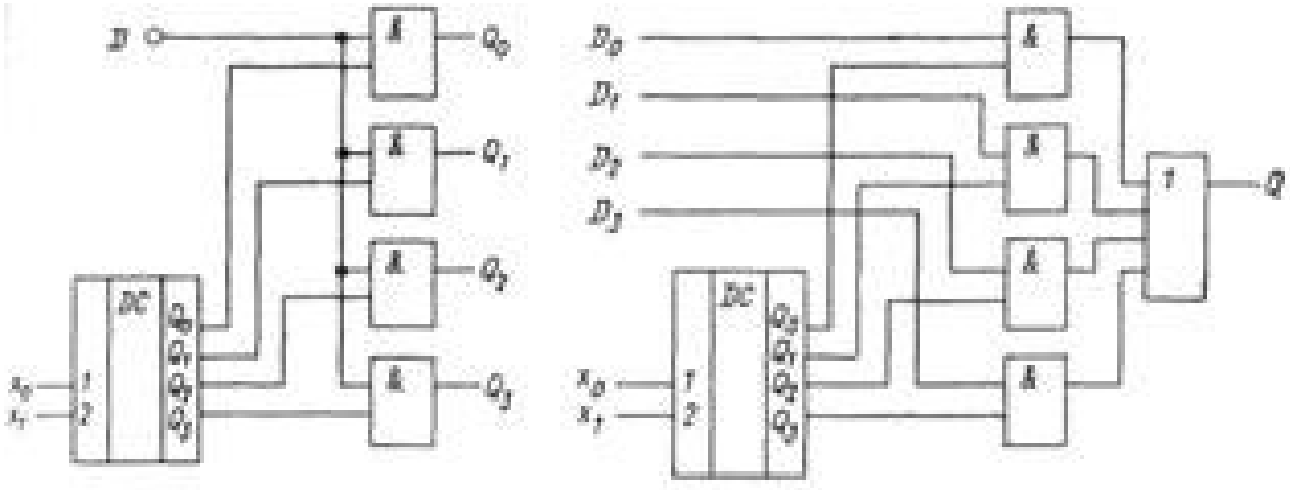
$$x_i = (Q_3 Q_2 Q_1 Q_0)_i$$

де xi – сигнал на виході і-го кодера ; $(Q_3 Q_2 Q_1 Q_0)_i$ є компонентом одиниці, що відповідає двійковому коду і-го десяткового розряду .

Неважко помітити, що алгебраїчна логічна функція декодера відрізняється від алгебраїчної логічної функції демультиплексора лише

наявністю в останньому додаткового множника, відповідного значенню сигналу на інформаційному вході D. Тому при D = 1, демультиплексор працює як декодер. Зворотне перетворення декодера в демультиплексор вимагає введення двох допоміжних логічних елементів І, які виконують операцію логічного множення між загальним інформаційним вхідним сигналом D і відповідним логічним результатом множення адресних сигналів (Q 3 Q 2 Q 1 Q 0) .

мультиплексора за допомогою декодера . Для цього використовується діаграма на рис. 7.4.5,а доповнити чотирма вихідними логічними елементами АБО (рис. 7.4.5,б).



І) Б)

Рис. 4.31. Реалізація демультиплексора (а) і мультиплексора (б) за допомогою дешифратора

У розробці інтегральних схем використовується кілька логічних структур декодера. Основна відмінність полягає в швидкості роботи і кількості використовуваних елементарних логічних елементів.

Найшвидшим і складним є дешифратор, який безпосередньо реалізує систему алгебраїчних логічних функцій (4). Такий дешифратор називають однокаскадним або паралельним. Його структурна схема подібна до схеми демультиплексора за умови D = 1.

Припускаючи, що для обробки одного вхідного логічного сигналу необхідний певний звичайний апаратний блок, кількість звичайних апаратних блоків для n-розрядного декодера визначається виразом:

$$N_1 = n2^n.$$

На рис. 4.32. наведено умовне графічне зображення дешифратора. Відповідає інтегральній схемі двійково -десяткового декодера типу 564ID1. Якщо основною вимогою при проектуванні є простота системного рішення, використовуються інші структурні схеми дешифраторів. Однак спрощення конструкції було досягнуто ціною зниження ефективності.

Рис. 4.32. Звичайне графічне позначення декодера

Мікросхеми декодера часто мають вхід, що дозволяє E (вхід затвора). Наявність цього входу дає змогу на основі готових інтегральних схем створювати деревоподібні структури дешифрування , які необхідні для збільшення швидкості передачі вхідного коду .

Програма реалізації шифру 10 на 4 (опис таблиці істинності шифру) з використанням мови AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

Шифр підпроекту 1

```
        (
    XIP[9..0]: вхід ; - вхідні сигнали
    QOP[3..0]: вихід ; - вихідні сигнали
        )
    Почніть
        Таблиця
            XIP[] => QOP[];
            b"0000000001" => b"0000";
            b"0000000010" => b"0001";
            b"0000000100" => b"0010";
            b"0000001000" => b"0011";
            b"0000010000" => b"0100";
            b"0000100000" => b"0101";
            b"0001000000" => b"0110";
            b"0010000000" => b"0111";
            b"0100000000" => b"1000";
            b"1000000000" => b"1001";
    Кінець таблиця _
кінець _
```

Програма реалізації кодера 10 на 4 (опис на поведінковому рівні кодера) з використанням мови AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
    Шифр підпроекту2
            (
    XIP[9..0]: вхід ; - вхідні сигнали
    QOP[3..0]: вихід ; - вихідні сигнали
            )
    Почніть
```

```
        QOP[3] = XIP[8] + XIP[9];
        QOP[2]= XIP[4] + XIP[5] + XIP[6] + XIP[7];
        QOP[1]= XIP[2] + XIP[3] + XIP[6] + XIP[7];
        QOP[0]= XIP[1] + XIP[3] + XIP[5] + XIP[7] + XIP[9];
    кінець _
```

Програма реалізації 3-бітного декодера з інверсними виходами на мові AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
    дешифратор підпроекту1
    (
        XIP[3..1]: вхід ; - вхідні сигнали
        QOP[7..0]: вихід ; - вихідні сигнали
    )
    почати
        case XIP[] є
            коли 0 => QOP = b"11111110";
            коли 1 => QOP = b"11111101";
            коли 2 => QOP = b"11111011";
            коли 3 => QOP = b"11110111";
            коли 4 => QOP = b"11101111";
            коли 5 => QOP = b"11011111";
            коли 6 => QOP = b"10111111";
            коли 7 => QOP = b"01111111";
        кінець справа _
    кінець _
```

Програма впровадження мультиплексора з 2 адресними входами, 4 інформаційними входами та операцією дозволу входу (опис із використанням емульованої таблиці дійсності мультиплексора) з використанням мови AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
підпроект 1 мультиплексор
(
        INFIN[4..1]: вхід ; - інформаційні входи
        ADRIN[2..1]: вхід ; - адресні входи
        EAW: запис ; - дозвіл на роботу ( вхід на ворота )
        P: вихід ; - Вихід мультиплексора
)
почати
        якщо ENA == 0 , то - емуляція таблиці істинності
                випадок ADRIN[2..1] є
                        коли 0 => Q = INFIN[1];
                        коли 1 => Q = INFIN[2];
                        коли 2 => Q = INFIN[3];
                        коли 3 => Q = INFIN[4];
                кінець справа _
        кінець якщо _
кінець _
```

Примітка. Компілятор AHDL не допускає присутності

- таблиці валідності змінних (параметрів), навіть якщо
- змінні (параметри) раніше призначалися як константи
- Позначаючи. Тому, за логікою таблиці істинності, виходячи з
- створюється послідовність перевірки для оператора вибору CASE
- значення вхідних сигналів системи.

Програма реалізації мультиплексора з 2 адресними входами, 4 інформаційними входами та операцією дозволу входу (опис на поведінковому рівні мультиплексора) мовою AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
підпроектний мультиплексор2
        (
                INFIN[4..1]: вхід ; - інформаційні входи
                ADRIN[2..1]: вхід ; - адресні входи
                EAW: запис ; - дозвіл на роботу ( вхід на ворота )
                P: вихід ; - Вихід мультиплексора
)
початі
        Q = INFIN[1] & !ADRIN[2] & !ADRIN[1] & !ENA #
        INFIN[2] & !ADRIN[2] & ADRIN[1] & !ENA #
        INFIN[3] & ADRIN[2] & !ADRIN[1] & !ENA #
        INFIN[4] & ADRIN[2] & ADRIN[1] & !ENA;
кінець _
```

Примітка. Q — це функція логічної алгебри, яка описує роботу мультиплексора .

Програма реалізації демультиплексора з 3 адресними входами, 1 інформаційним входом і входом дозволу з використанням мови AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
підпроект demultiplexer1
        (
                ADRIN[3..1]: вхід ; - введення адреси
                INFIN: вхід ; - введення інформації
                EAW: запис ; - дозвіл на роботу ( вхід на ворота )
                Q[7..0]: вихід ; - Вихід демультиплексора
        )
початі
        якщо EAW == 0 , тоді
                випадок ADRIN[] є
                        коли 0 => Q[0] = INFIN;
```

коли 1 => Q[1] = INFIN;

коли 2 => Q[2] = INFIN;

коли 3 => Q[3] = INFIN;

коли 4 => Q[4] = INFIN;

коли 5 => Q[5] = INFIN;

коли 6 => Q[6] = INFIN;

коли 7 => Q[7] = INFIN;

кінець справа _

кінець якщо _

кінець _

На рис. 4.33 показано вікно редактора сигналів проекту Shifrator1.

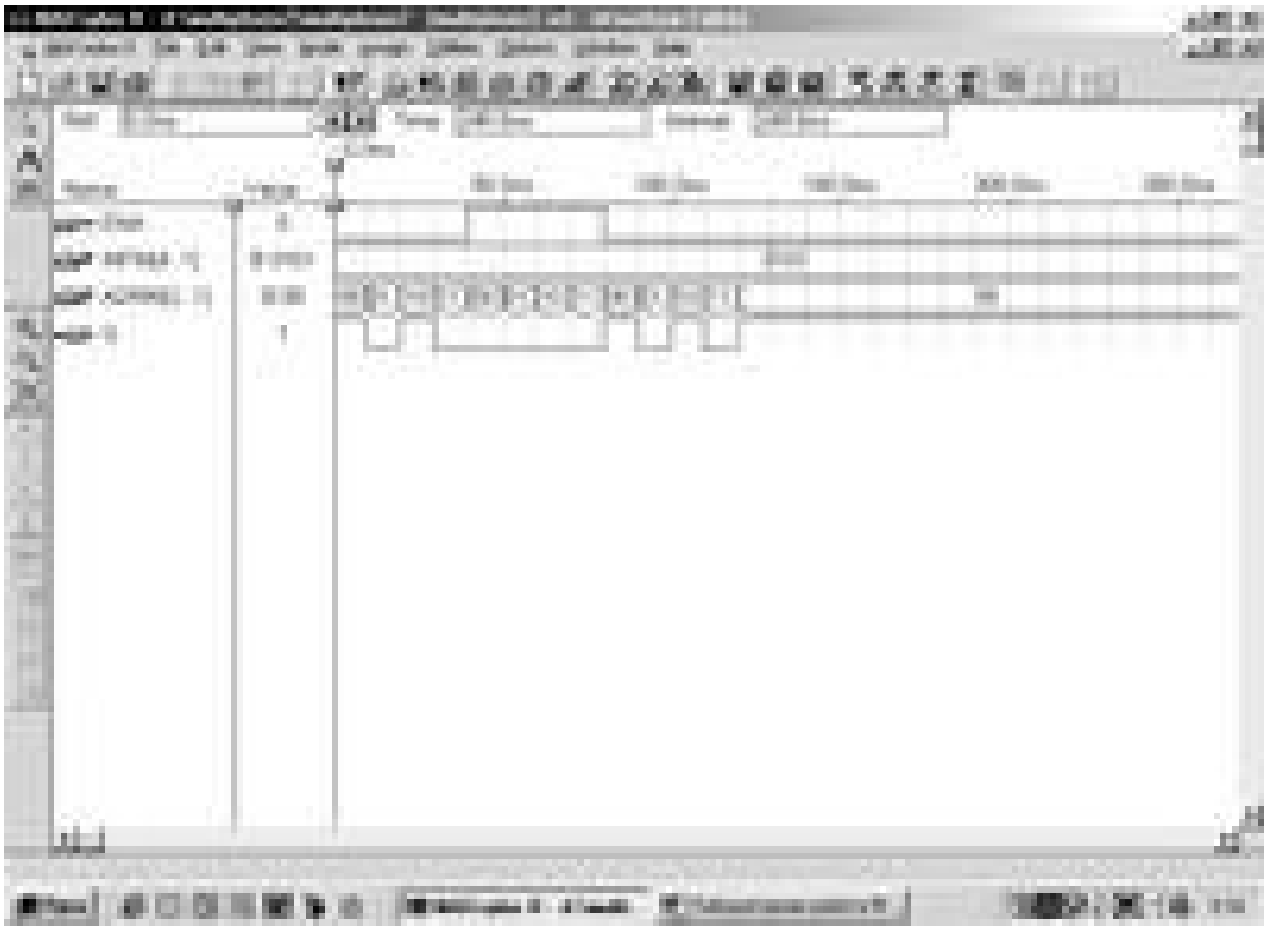


Рис. 4.33. Результати тесту кодера 10 з 4

На рис. 4.34. буде показано вікно редактора сигналів проекту "decipherer1".

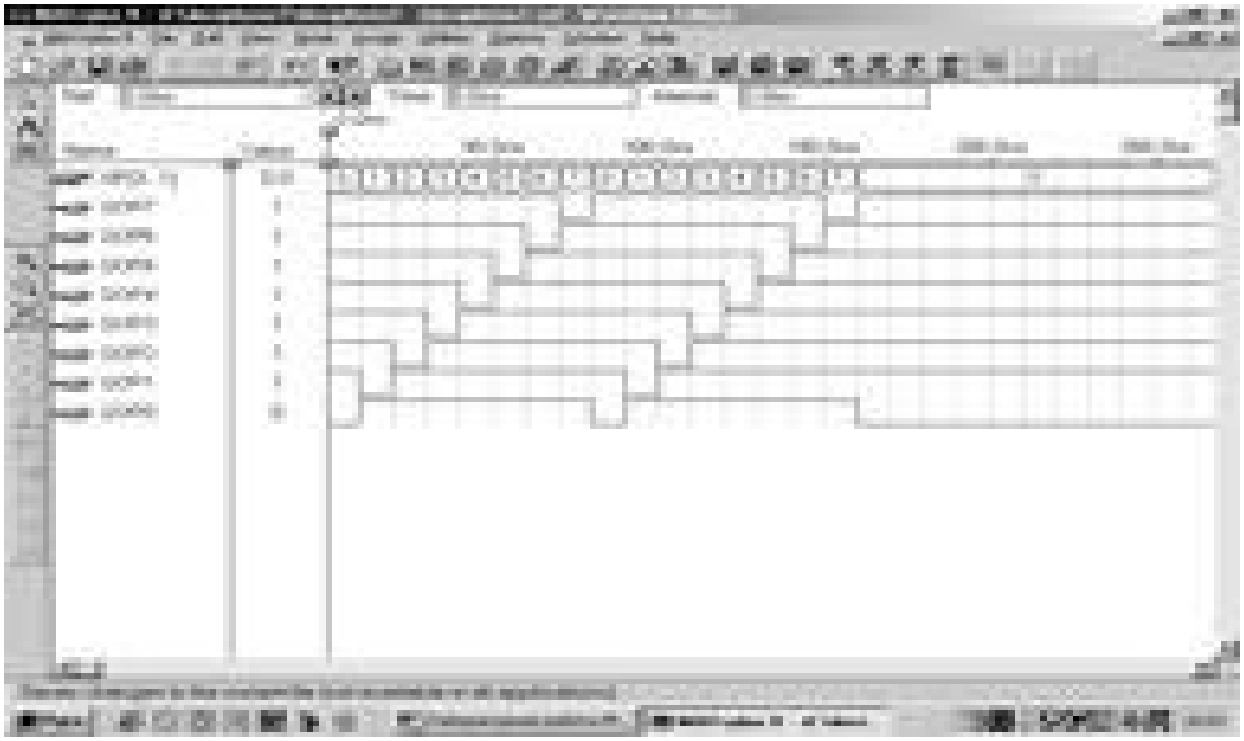


Рис . 4.34 . Повні результати тестування 3-розрядного декодера з інверсними виходами

На рис. 4.35 показано вікно редактора сигналів проекту Multiplexer2.

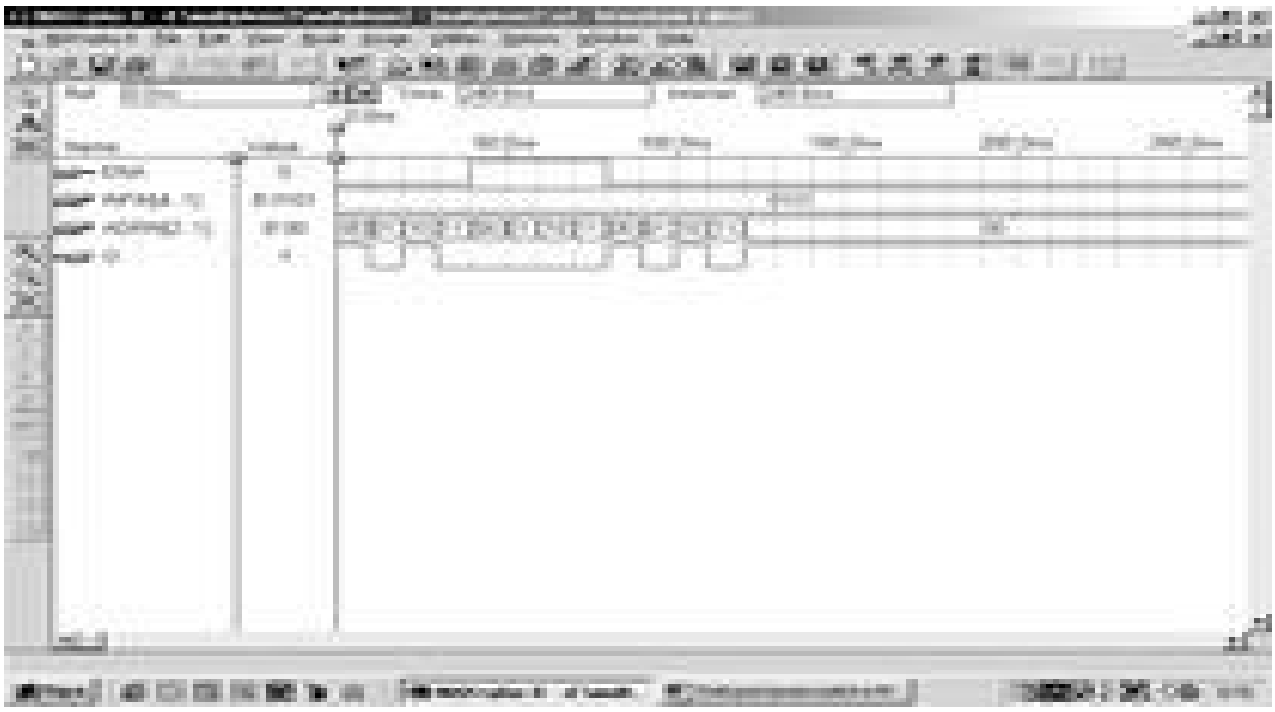


Рис. 4.35 . Результати тестування мультиплексора з 2 адресними входами, 4 інформаційними входами та операційним входом

На рис. 4.36. буде показано вікно редактора сигналів проекту "demultiplexer1".

Рис. 4.36 . Результати тестування демультиплексора з 3 адресними входами, 1 інформаційним входом і входом дозволу роботи

4.8. Теоретичні відомості про доданки та від’ємники

Крайня ліва цифра двійкового числа, наприклад 101011, називається найстаршою цифрою (OLD), а крайня права цифра називається наймолодшою цифрою (LOL). Нагадаємо, що цифри даного двійкового числа в порядку зростання старшинства мають ваги (справа наліво) 1, 2, 4, 8, 16, 32.

двійкові числа складаються лише з двох цифр (0 і 1), таблиця додавання досить проста. Це показано на рис. 7.5.1. Як і у випадку додавання десяткових чисел, перші три результати зрозумілі. Щодо останньої задачі (1+1), то під час додавання десяткових чисел у цьому випадку відповіддю буде число 2. Отже, у двійковому додаванні 1+1=0 плюс перенесення 1 до найближчої старшої двійкової цифри .

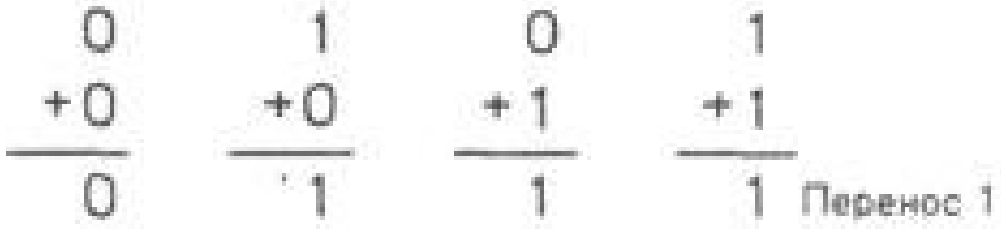
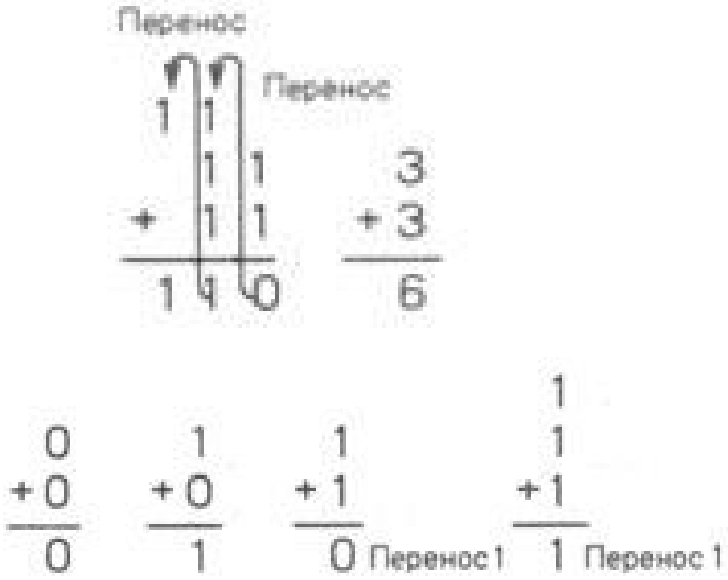


Рис. 4.37 . Двійкова таблиця додавання

Інший приклад додавання двійкових чисел наведено на рис. 7.5.2, а.



1.1 *Рис. 4.38. Двійкове додавання а) – приклад двійкового додавання; б) - форма двійкової таблиці додавання скорочена*

Рішення здається простим, поки ми не досягнемо двійкового біта, де нам потрібно знайти двійкову суму. У десятковій системі числення ця сума дорівнює 3, що відповідає числу 11. Цей випадок не показано на рис. 4.37 . Сума 1+1+1 може бути на будь-якому розряді, крім розряду одиниць. У новій (скороченій) таблиці на рис. 4.38 буде врахована ще одна можлива комбінація 1+1+1.

Ця таблиця дійсна для всіх розрядів двійкових чисел , крім розряду одиниць.

двійкові значення наведені у відповідній таблиці істинності (табл. 7.5.1). однозначні компоненти A, B і сигнал передачі $_C$.

Таблиця 4.12 . Повна таблиця істинності суматора

1.1.1.1 Увійдіть			***1.1.1.2 спускатися***	
від $_v$	IN	I	C	Від $_0$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1
Трансфер + Б + А			сума	рухатися

Повне додавання використовується для додавання всіх двійкових цифр , крім одиниць. Вони повинні мати додатковий прохідний вхід.

Повний суматор - це схема з 3 входами. Сигнали на його виходах S і C_0 є результатом додавання трьох вхідних сигналів (на входах A, B і $C_в$).

На рис. 4.39 показана розширена логічна схема повного суматора. Він заснований на структурній схемі з двома напівсуматорами.

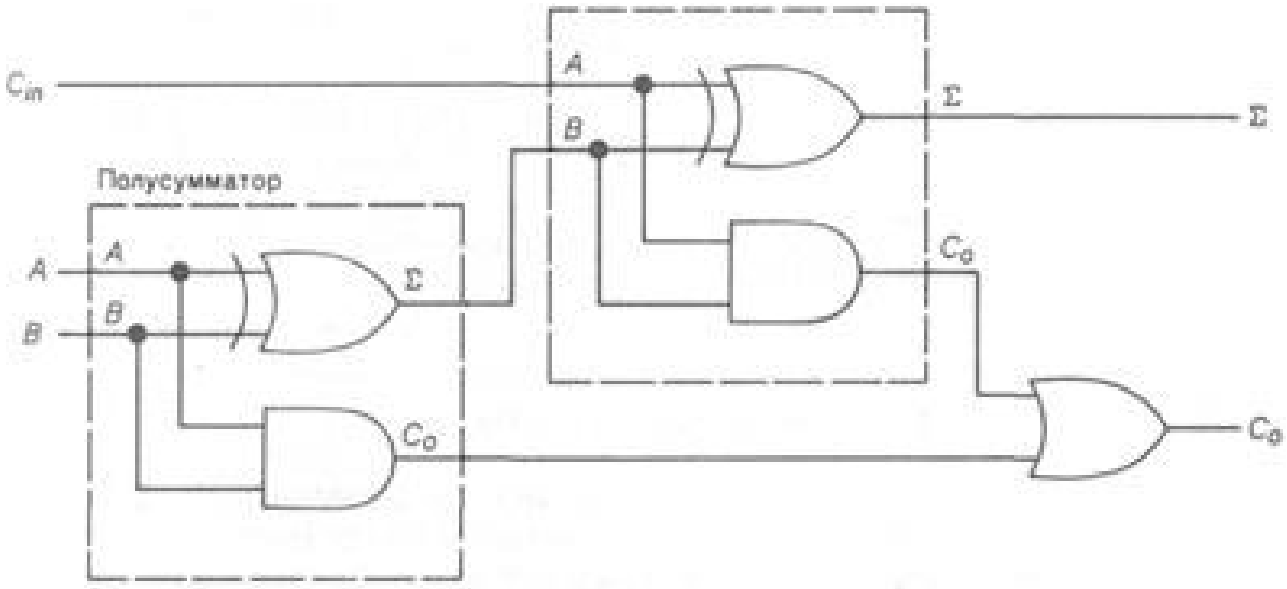


Рис. 4.39 . Логічна схема повного суматора

На рис. 4.40 . Наведена інша схема для повного суматора з використанням двох виняткових логічних елементів або трьох логічних елементів І-Н І. Зверніть увагу, що схеми показані на рис. 4.39 . і 4,40 . вони відрізняються лише заміною логічних елементів І та АБО на логічні елементи І-НЕ.

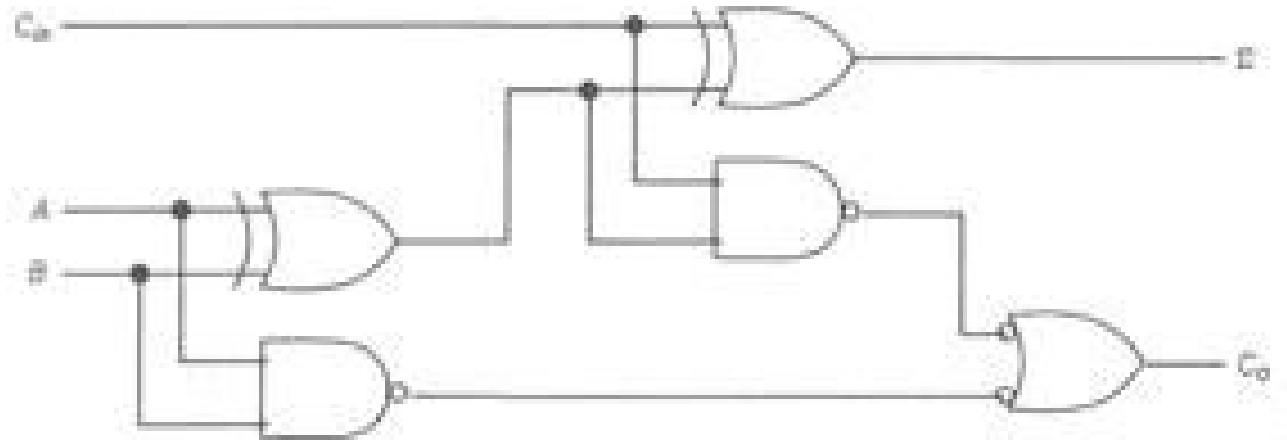


Рис . 4.40 . Використання логічної схеми повного суматора унікальні OR і IN *I*

Напівсуматори та суматори зазвичай використовуються разом. Пристрої арифметико-логічного мікропроцесора (АЛП) містять велику кількість напівсуматорних і повносуматорних схем. Мікросхеми мікропроцесора ALP

також можуть виконувати операції віднімання, використовуючи ті ж напівсуматори і суматори.

Поєднуючи певним чином напівсуматори і повні суматори, отримують пристрої, які одночасно виконують додавання кількох двійкових розрядів. Пристрій, схема якого зображена на рис. 7.5.5. виконує операцію додавання двох 3-розрядних чисел. Числа — доданки $A_2A_1A_0$ і $B_2B_1B_0$.

На вхід суматора розряду одиниць (напівсуматора) надходять сигнали, що відповідають значенням розряду одиниць в категоріях. Вхідними сигналами повного двійкового суматора є сигнали переносу з виходу половинного суматора (подаються на вхід C_y) і А значення $_1$ і $_1$ цифри двійки і слова. Потім суматор четвірок додає A_2 і B_2 і сигнал перенесення від суматора двійок. На двійковому виході пристрою (показано в правому нижньому кутку рис. 4.41) встановлюється двійкова сума.

Додавання двох 3-розрядних двійкових чисел може призвести до 4-розрядного числа, тому ми маємо додатковий вісімковий біт у покажчику суми. Зверніть увагу, що цей біт підключений до виходу (C_0) чотирьохсуматора.

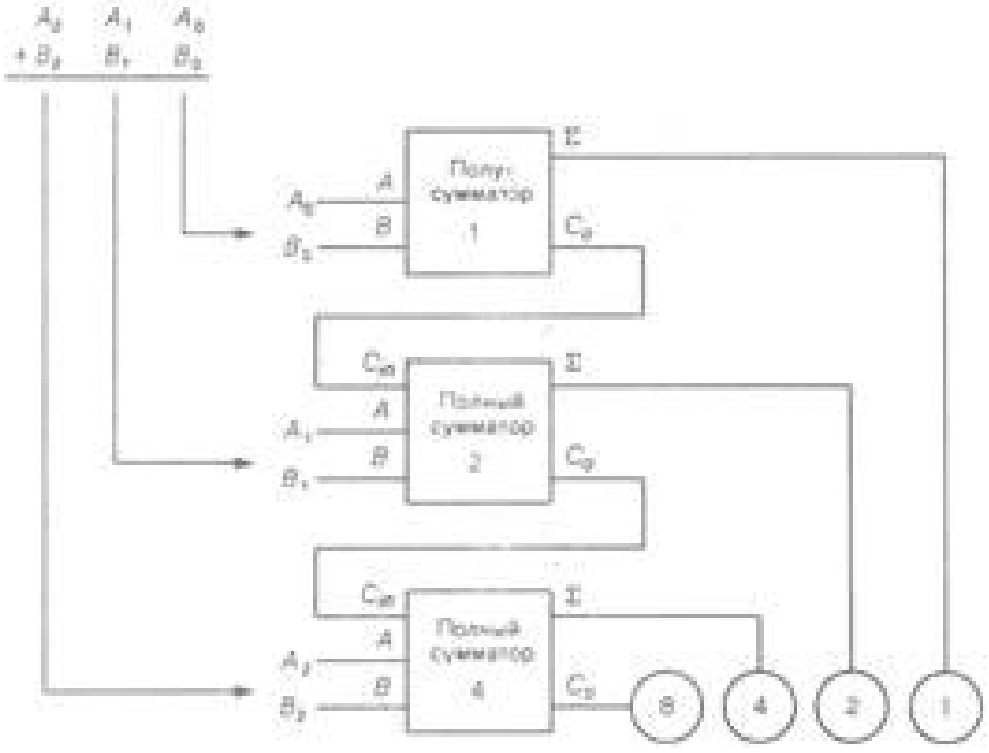


Рис. 4.41 . 3-розрядний паралельний суматор

Логіка роботи розглянутого 3-розрядного суматора практично не відрізняється від послідовності операцій, що виконуються при ручному додаванні (додавання однозначних чисел плюс перехід до наступного розряду). Однак електронний суматор виконує ці операції набагато швидше.

Ще раз зауважте, що в багаторозрядних суматорах напівсуматори використовуються лише для додавання одиниць у розряді; всі інші цифри використовують повні суматори. Наведений вище опис 3-розрядного суматора називається паралельним суматором.

У паралельному суматорі інформаційні розряди всіх розрядів надходять на входи одночасно. Результат (кількість) з'являється на виході практично відразу. Паралельний суматор на рис . 4.41. належить до класу комбінаційних логічних схем. Для визначення даних на входах і виходах суматорів зазвичай використовуються різні додаткові регістри.

Пізніше буде показано, що додавання та віднімання подібні, і що половинне віднімання та повне віднімання використовуються так само, як половинне додавання та повне додавання. Нижче наведена таблиця двійкового віднімання, яка показує правила віднімання двійкових чисел , зображених на рис. 4.41 . у вигляді таблиці істинності. Ми бачимо, що B віднімається з A (A і B – вхідні сигнали), результат (різниця) з’являється на виході D_1 . Якщо їх більше, ніж A (як у рядку 2 таблиці), ви повинні запозичити 1 до наступної старшої цифри. Сигнал позики вказується в стовпці $_0$.

Таблиця 4.13 . Двійкова таблиця віднімання

Увійдіть		спускатися	
I	IN	Д мені	B_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0
AB		Різниця	Кредит

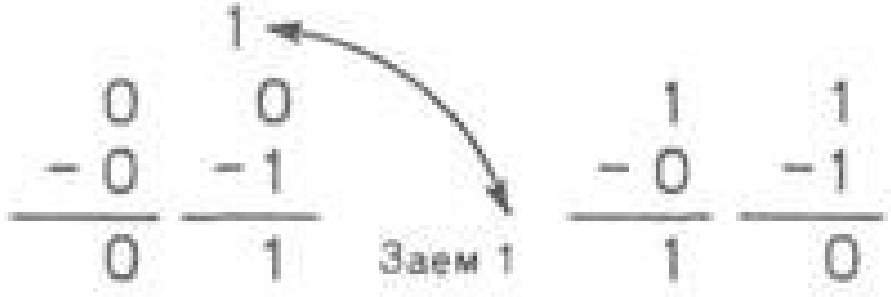


Рис. 4.42. Двійкова таблиця віднімання

Під час віднімання кількох бітів із двійкових чисел слід враховувати перевагу «одиниць» у старших цифрах.

Нижче наведено таблицю істинності, яка містить усі можливі комбінації, отримані в результаті віднімання двійкових чисел.

Таблиця 4.14. Повна таблиця важливості віднімання

1.1.1.3 Увійдіть			***1.1.1.4 спускатися***	
I	IN	$Б_{в}$	$Д_{мені}$	$В_{0}$
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1
$A – B – B_{в}$			Різниця	Кредит

Умовне позначення повного віднімання показано на рис. 4,43 п. З лівого боку - входи А, Б, Б $_{в}$, праворуч – виходи D $_{i}$, В $_{0}$. На рис. 4.43b показано , як об'єднати половину від'ємників і логічний елемент АБО, щоб отримати повний від'ємник . Детальна логічна схема повного віднімання показана на рис. 4.43 ст. Ця схема працює відповідно до таблиці істинності. Якщо необхідно, логічні елементи І та АБО можна замінити трьома логічними елементами І-Н І. У цьому випадку ми отримуємо схему повного віднімання , подібну до схеми повного суматора.

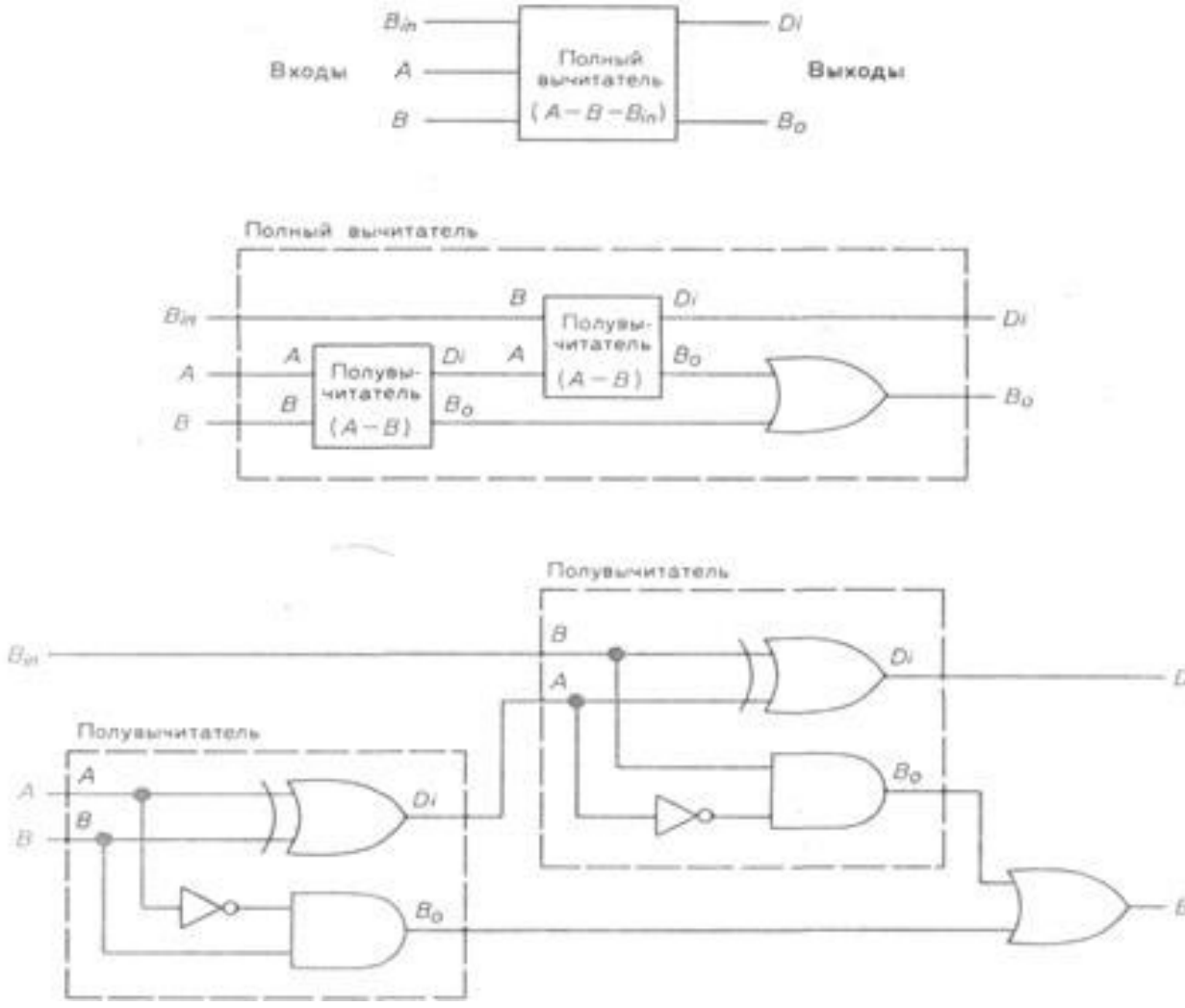


Рис. 4.43 . Повне віднімання а) – умовний графічний запис; б) – структурна схема для випадку використання двох напіввід'ємників і логічного елемента АБО; в) є діаграмою

Поєднання напіввіднімачів і повного віднімання створює пристрої, які називаються паралельним відніманням . Подібним чином паралельне віднімання поєднується з трирозрядним суматором, розглянутим вище . Додаток на рис. Паралельним число 4,42 називають тому, що до цього суматора одночасно надходять інформаційні розряди всіх розрядів доданків.

На рис. 4.44 показана структурна схема, отримана об'єднанням одного напівстоку і трьох повних стоків . Це 4-розрядна схема паралельного віднімання , яка виконує операцію віднімання одного двійкового числа $B_3 B_2 B_1 B_0$ від двійкового числа $A_3 A_2 A_1 A_0$. Зауважте, що верхнє (на діаграмі) віднімання виконує віднімання цифр одиниць (SMP). Вихід B_0 цього віднімання з'єднаний з відніманням двійки.

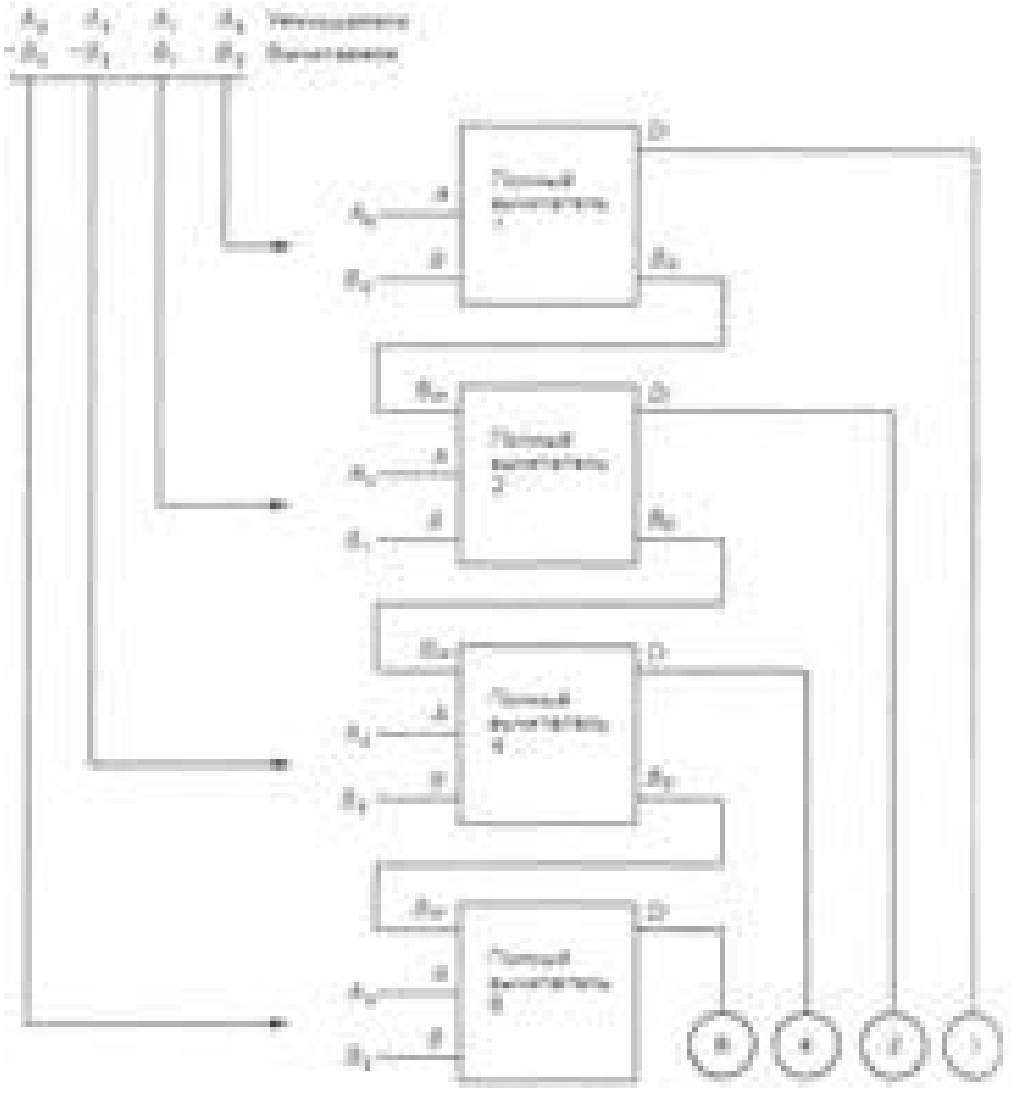


Рис. 4.44. 4-розрядне паралельне віднімання

Взагалі кажучи, вихід позики B_0 кожного віднімання з'єднаний із входом позики B_{at} віднімання сусідньої старшої цифри. Ці лінії зв'язку «слідують» за титрами в процесі віднімання двійкових чисел .

Програма реалізації 4-розрядного суматора мовою AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
ПІДПРОЕКТ add_gate
        (
                A[4..1], B[4..1], cin : введення ;
                C[4..1], cout : вихід ;
        )
ЗМІННИЙ
        carry_out [5..1]: вузол ;
ПОЧАТИ
        carry_out [1] = cin ;
                FOR i IN 1 TO 4 GENERATE
        C[i] = A[i] $B[i] $ переказ [i];
carry_out [i + 1] = CARRY ( A[i] & B[i] # carry_out [i] & ( A[i] # B[i] ));
        КІНЕЦЬ ПОКОЛІННЯ;
                cout = переміщення [5];
КІНЕЦЬ;
```

Програма реалізації 4-бітного віднімання мовою AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
ПІДПРОЕКТ add_sub
        (
                A [4..1], B [4..1]: вхід = GND;
                Res [4..1], Cout : вихід ;
```

)

ЗМІННИЙ

S[4..1]: вузол ;

Cout_int : вузол ;

ПОЧАТИ

(Cout_int , S[]) = (GND, A[]) - (GND, B[]);

(Cout , Res []) = (Cout_int , S[]);

КІНЕЦЬ;

На рис. 4.45 показано вікно редактора сигналів проекту add_gate .

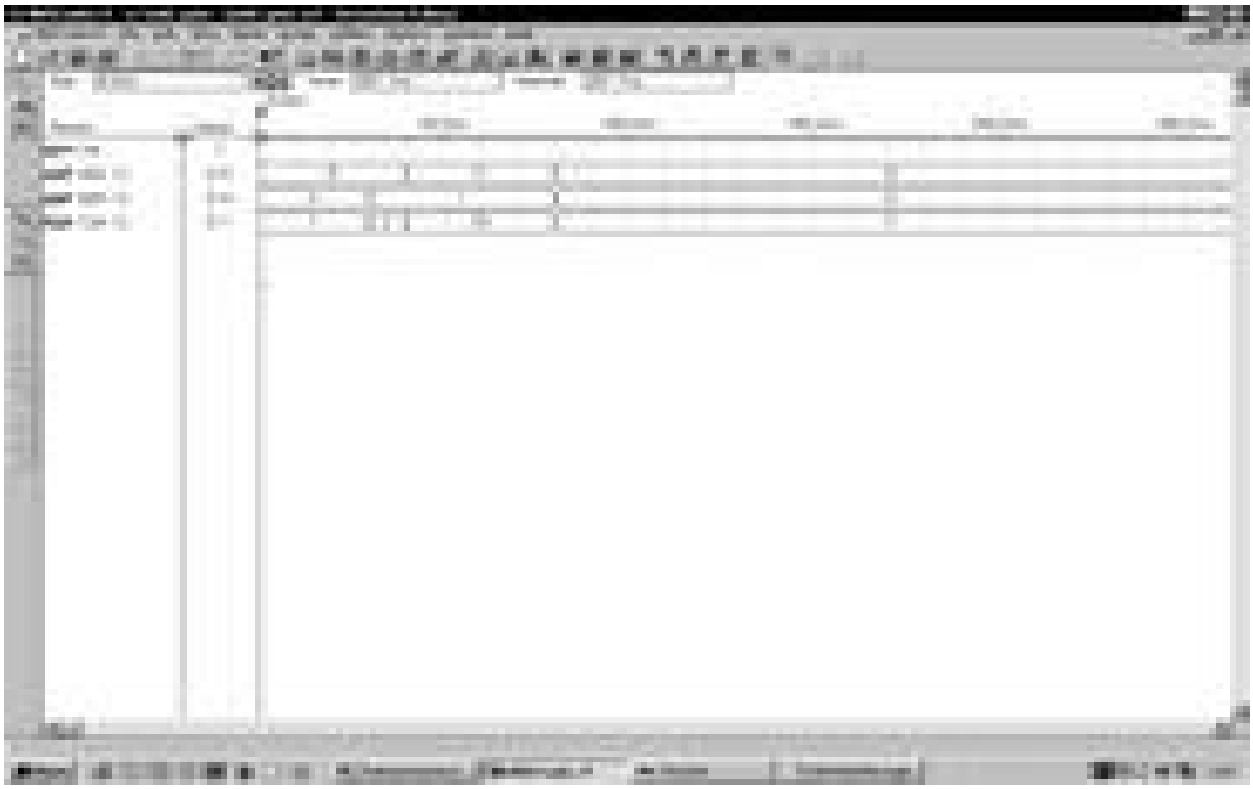


Рис. 4.45. Результати тестування 4-розрядного суматора

На рис. 4.46 показано вікно редактора сигналів проекту add_sub .

Рис. 4.46 . Результати тесту на 4-бітове віднімання

4.9. Проект впровадження COM-порту в програму CAD MAX+PLUS II

Послідовний інтерфейс для односпрямованої передачі даних використовує одну сигнальну лінію, по якій один за одним передаються інформаційні біти. Цей метод передачі визначає назву інтерфейсу та порт, який його реалізує. Ці назви відповідають англійським термінам Serial Інтерфейс і серії Порт . Послідовна передача даних може відбуватися як в асинхронному, так і в синхронному режимах.

При асинхронній передачі кожному біту передує початковий біт, який сигналізує одержувачу почати наступний пакет, за яким зазвичай слідують біти даних і, можливо, біт паритету. Пакет закінчується стоп-бітом, який гарантує певну затримку між сусідніми пакетами (рис. 4.47).

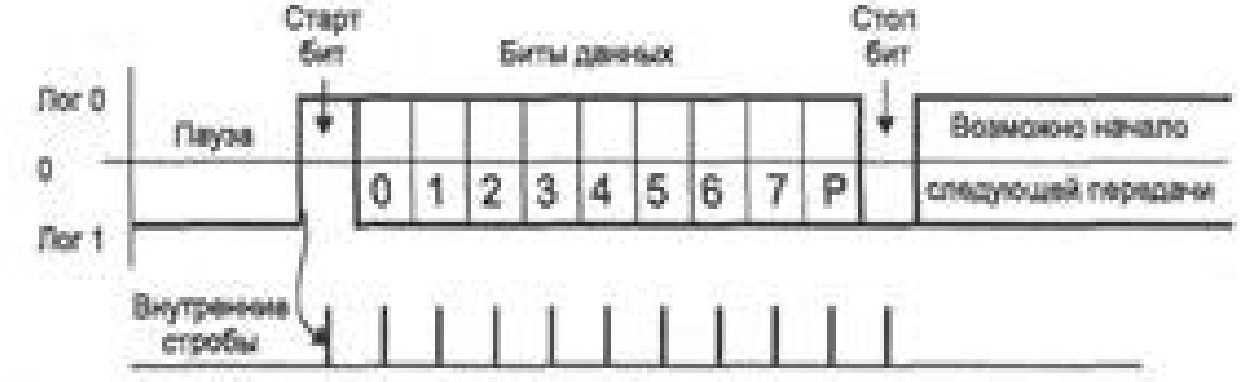


Рис. 4.47 . Асинхронний формат передачі

Початковий біт наступного надісланого байта може бути надісланий у будь-який час після закінчення стоп-біта, що означає, що між передачами можливі паузи невизначеної тривалості. Початковий біт, який завжди має точно визначене значення (log 0), забезпечує простий механізм для синхронізації приймача. Приймач і передавач працюють з однаковою швидкістю обміну, яка вимірюється кількістю бітів, що передаються за секунду. Генератор внутрішньої синхронізації приймача використовує дільник лічильника опорної частоти, який скидається на нуль, коли приймається початок (наростаючий фронт) початкового біта. Цей лічильник генерує внутрішні вентилі , які приймач використовує для запису отриманих бітів. В ідеальному варіанті затвори розташовані в середині бітових інтервалів, що забезпечує можливість прийому даних при певних розбіжностях швидкостей приймача і передавача.

Неважко помітити, що при передачі 8 бітів даних, одного керуючого біта і одного стопового біта максимально допустима розбіжність швидкості приймача і передавача, при якій дані будуть правильно розпізнані, не може перевищувати 5%. Враховуючи фазові зміни (затримані фронти сигналу) і дискретність внутрішнього лічильника синхронізації, фактично допускаються менші відхилення частоти. Чим менший коефіцієнт ділення внутрішньої частоти внутрішнього генератора (частота передачі), тим більша помилка при прив’язці стробів до центру бітового інтервалу, і тому вимоги до узгодження частоти є найсуворішими. Причому, чим вище частота передачі, тим сильніше впливають фактори, що призводять до помилок.

Асинхронний формат пакету дозволяє виявити можливі помилки передачі:

1) Якщо отримано стартову краплю пакета та рівень LU встановлено після стробу стартового біта, стартовий біт вважатиметься помилковим, і приймач повернеться в режим очікування. Одержувач може не повідомляти про цю помилку формату;

2) якщо рівень логічного нуля виявлено в момент часу, призначений стоп-бітам, записується помилка стоп-біта (також помилка формату);

3) якщо використовується перевірка парності, то після передачі бітів даних (перед стоп-бітом) надсилається контрольний біт. Цей біт доповнює кількість окремих бітів даних, щоб вони були парними чи непарними, залежно від прийнятого контракту. Прийняття біта з хибним значенням контрольного біта з увімкненою перевіркою парності призводить до виправлення помилки отриманих даних.

Перевірка формату дозволяє знайти розрив рядка: у цьому випадку зазвичай приймається логічний нуль, який спочатку інтерпретується як початковий біт і нульові біти даних, але потім запускається перевірка стопового біта.

Кількість бітів даних може бути 5, 6, 7 або 8 (5- і 6-бітні формати зустрічаються рідше). Кількість стоп-бітів може бути 1, 1,5 і 2 ("біт і половина" означає просто довжину інтервалу зупинки).

Асинхронний обмін в персональному комп'ютері здійснюється за протоколом RC-232.

Інтерфейс RC-232 призначений для з'єднання обладнання, що приймає або передає дані (OOD - один кінцевий засіб передачі даних або ADP - обладнання передачі даних) з одним кінцевим пристроєм каналу даних. Комп'ютер, принтер, плотер або інший периферійний пристрій може виступати в якості ADP. Цьому обладнанню відповідає абревіатура DTE – Data рухатися Обладнання . У ролі ACD зазвичай виступає модем, цьому обладнанню відповідає абревіатура DCE - Data спілкування Обладнання . Кінцевою метою підключення є з’єднання двох пристроїв DTE, повна схема підключення показана на рис. 2. 4.48 . Інтерфейс дозволяє відключити канал зв’язку разом із парою пристроїв DTE (модемів) шляхом прямого з’єднання пристроїв за допомогою нульмодемного кабелю (рис. 4.49).

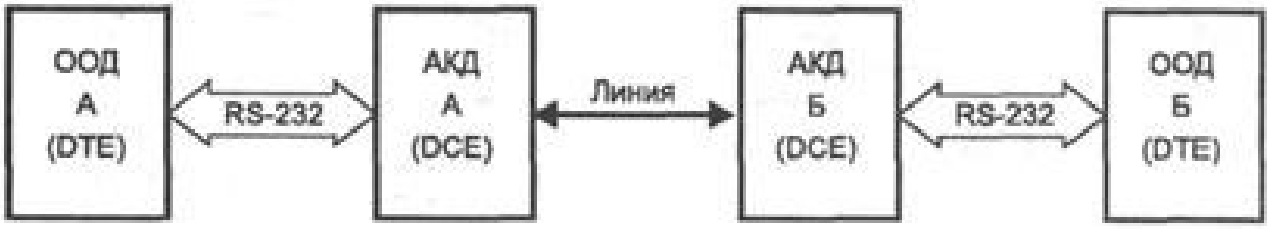


Рис. 4.48 . Повна схема підключення, сумісна з RC-232

Рис. 4.49. Підключення через нуль-модемний кабель RC-232

Стандарт описує сигнали керування інтерфейсом, передачу даних, електричний інтерфейс і типи з'єднань. Стандарт описує асинхронний і синхронний режими обміну, але COM-порти підтримують лише асинхронний режим.

Принципова структурна схема COM-порту, реалізованого в системі автоматизованого проектування MAX+PLUS II, наведена на рис. 4.48 . Це розширена версія рис. 4.49 , де пульт дистанційного керування з 20 параметрами вихідного сигналу виступатиме як один кінцевий об'єкт передачі даних.

Приймемо стоповий біт рівним 1, а кількість бітів даних рівним 8. Частота сигналу синхронізації може бути 115200 біт на секунду або 9600 біт на секунду. Щоб забезпечити схему можливістю перемикання частот, введемо сигнал перемикання (switch сигнал), який буде перемикати частоту сигналів синхронізації, які будуть подаватися на вхід синхронізації .

Усі елементи схеми можна окремо описати мовою AHDL у текстовому редакторі, а потім за допомогою редактора символів перетворити елементи на символи та зібрати всю схему в графічному редакторі.

Розглянемо функціональне призначення кожного з пристроїв, наведених на схемі.

Кодер 20 на 10 має 20-бітний вхід (Encipherer_1_input[20..1]) і 10-бітний вихід (Encipherer_1_output[10..1]). Кодер опису, що використовує таблицю істинності, виконує дві функції.

По-перше, залежно від натиснутої клавіші на пульті дистанційного керування, він виводить двійковий код у діапазоні від 0000000011 до 0000101001. Ця послідовність вихідного двійкового коду завжди починається з 0 і закінчується на 1, ці числа є не чим іншим, як початком і кінцем фрагмента . Це означає, що кодер посилає готовий код, який можна передати по лінії зв'язку.

По-друге, можливе натискання двох клавіш одночасно, що може призвести до збою системи. Тому ця влада
WHEN OTHERS => Encipherer_1_output[] = b"1111111111";
наказує шифру виводити двійковий код 1111111111 в інших випадках, не передбачених таблицею істинності шифру - це захисна функція.

5-бітний лічильник на D-тригерах має вхід (Counter_1_input), який отримує сигнал синхронізації від зовнішнього генератора імпульсів, вхід скидання (Reset) і функціонує як дільник частоти. Цей лічильник має два виходи (Counter_1_out[1] і Counter_1_out[1]), що відповідають вхідній частоті, поділеній на 2, і вхідній частоті, поділеній на 12.

Мультиплексор має 4 інформаційні входи (Multiplexer_input_1[4..1]), 2 адресні входи (Switch_signal [2..1]) і вхід дозволу (Turn on). Пристрій реалізовано на основі емульованої таблиці істинності і має завдання перемикати вихід (вихід мультиплексора_1__) на один з інформаційних входів залежно від стану адресних входів. Оскільки робоча частота системи може становити 115200 біт в секунду або 9600 біт в секунду, то для управління комутацією достатньо одного адресного входу, а іншого адресного входу (Switch_Signal [2]), що відповідає назві SWITCH [2] у вікні графічного редактора заземлено .

З тієї ж причини сигнали Multiplekser_1_input[4] і Multiplekser_1_input[3], що відповідають іменам MUX[4] і MUX[3] у вікні графічного редактора, заземлені. Вам може знадобитися повторно ввести адресу, якщо кількість можливих змін частоти збільшиться до чотирьох у майбутньому під час модифікації схеми.

надсилається сигнал синхронізації, вибраний мультиплексором, і вхід скидання (Reset) . Цей лічильник контролює завантаження послідовного регістра, встановлюючи власний вихід (LOAD) на 0 або 1: 0 - завантаження заборонено, 1 - завантаження дозволено.

10-бітний паралельний тригерний регістр D зберігає та зсуває з надходженням нового імпульсу синхронізації, який згенерував вихідний код і передав йому мультиплексор . Регістр має 10-бітний вхід для завантаження «інформаційного» коду (Register_1_input[9..0]), вхід дозволу (Enable), програмований вхід (Set), вхід синхронізації (Clk), вхід керування навантаженням (Load) і однобітовий вихід (Register_1_output).

Програма реалізації кодера 20 на 10 (опис таблиці валідності кодера) з використанням мови AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
Підпроект cipher_1
(
        Encipherer_1_input[20..1]: вхід ;
        Encipherer_1_output[10..1]: вихід ;
)
Почніть
        CASE Encipherer_1_input[] IS
WHEN b"00000000000000000001" = > Encipherer_1_output[] = b"0000000011";
КОЛИ b"00000000000000000010" = > Encipherer_1_output[] = b"0000000101";
КОЛИ b"00000000000000000100" = > Encipherer_1_output[] = b"0000000111";
КОЛИ b"00000000000000001000" = > Encipherer_1_output[] = b"0000001001";
```

```
WHEN b"00000000000000010000" = > Encipherer_1_output[] = b"0000001011";
КОЛИ b"00000000000000100000" = > Encipherer_1_output[] = b"0000001101";
WHEN b"00000000000001000000" = > Encipherer_1_output[] = b"0000001111";
WHEN b"00000000000010000000" = > Encipherer_1_output[] = b"0000010001";
КОЛИ b"00000000000100000000" = > Encipherer_1_output[] = b"0000010011";
WHEN b"00000000001000000000" = > Encipherer_1_output[] = b"0000010101";
КОЛИ b"00000000010000000000" = > Encipherer_1_output[] = b"0000010111";
WHEN b"00000000100000000000" = > Encipherer_1_output[] = b"0000011001";
WHEN b"00000001000000000000" = > Encipherer_1_output[] = b"0000011011";
КОЛИ b"00000100000000000000" = > Encipherer_1_output[] = b"0000011101";
WHEN b"00001000000000000000" = > Encipherer_1_output[] = b"0000011111";
WHEN b"00010000000000000000" = > Encipherer_1_output[] = b"0000100001";
WHEN b"00100000000000000000" = > Encipherer_1_output[] = b"0000100011";
КОЛИ b"01000000000000000000" = > Encipherer_1_output[] = b"0000100101";
КОЛИ b"10000000000000000000" = > Encipherer_1_output[] = b"0000100111";
КОЛИ ІНШІ = > Encipherer_1_outpu [] = Б  "1111111111";
    Кінець CASE;
        кінець _
```

Програма реалізації мультиплексора з 4 інформаційними входами, 2 адресними входами та однією активаційною операцією (опис із емульованою таблицею істинності мультиплексора) з використанням мови AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
Мультиплексор підпроект_1
    (
        Мультиплексор_1_вхід[4..1]: вхід ;
        Switch_signal [2..1]: вхід ;
        Увімкнути : введення ;
```

```
        Мультиплексор_1__вихід: вихід ;
    )
    Почніть
        Якщо Увімкнути == 0
            річ Switch_signal [2..1] присутній
    коли 0 => мультиплексний вихід_1__ = мультиплексний вхід_1_[1];
    коли 1 => мультиплексний вихід_1__ = мультиплексний вхід_1_[2];
    коли 2 => мультиплексний вихід_1__ = мультиплексний вхід_1_[3];
    коли 3 => мультиплексний вихід_1__ = мультиплексний вхід_1_[4];
        кінець справа _
    кінець якщо _
кінець _
```

Програма, яка реалізує 10-розрядний послідовний регістр зсуву на мові AHDL в інтегрованому середовищі MAX+PLUS II, виглядає наступним чином:

```
    Реєстр підпроектів_1
        (
        Register_1_input[9..0]: вхід ;
        Enable , Set , Clk , Load : вхід ;
        Регістр_1_вихід: вихід ;
        )
    Змінний
        Тригери [9..0]: DFFE;
    Почніть
        Тригери [9..0]. clk = Clk ;
        Тригери [9..0]. prn = Набір ;
        Тригери [9..0]. ena = Увімкнути ;
    ЯКЩО навантаження == 0
```

```
        ПОТІМ
                Тригери [].d = ( Тригери [8..0].q, VCC);
        ІНАКШЕ
                Тригери [].d = Register_1_input[];
        END IF;
        register_1_output = Тригери [9].q;
кінець _
```

Програма реалізації 4-розрядного лічильника на мові AHDL в інтегрованому середовищі MAX+PLUS II виглядає наступним чином:

```
Лічильник підпроекту_2
        (
                CLK: вхід ;
                Скидання : введення ;
                НАВАНТАЖЕННЯ: вихід ;
        )
        Змінний
                TRIG[3..0]: DFF;
        Почніть
                TRIG []. clrn = Скинути ;
                TRIG []. clk = clk;
                IF (TRIG[].q == B"1011")
                THEN TRIG[].d = B"0000";
                НАВАНТАЖЕННЯ = B"1";
                Інакше TRIG[].d = TRIG[].q + 1;
                НАВАНТАЖЕННЯ = B"0";
        END IF;
кінець _
```

Програма, яка реалізує 5-бітний лічильник методом по модулю 12 на мові AHDL в інтегрованому середовищі MAX+PLUS II, виглядає наступним чином:

```
Лічильник підпроекту_1
    (
        Лічильник_1_введення: вхід ;
        Скидання : введення ;
        Counter_1_output[2..1]: вихід ;
    )
    Змінний
        Тригери : JKFFE;
        TRIG[4..0]: DFF;
    Почніть
        Triggers.j = vcc ;
        Triggers.k = vcc ;
        Triggers.clrn = Скинути ;
        Triggers.clk = вхід Counter_1;
        TRIG []. clrn = Скинути ;
        TRIG []. clk = вхід Counter_1;
        IF (TRIG[].q == B"11000")
        THEN TRIG[].d = B"00000";
        Інакше TRIG[].d = TRIG[].q + 1;
    END IF;
    Counter_1_output[2..1] = (TRIG[4].q, Triggers.q );
кінець _
```

Схема конструкції COM-порту, що складається з окремих символів підпрограм у вікні графічного редактора, наведена на рис. 2. 4.50 . Символи

підпрограм, представлені у вигляді блоків з іменованими входами та виходами, з'єднані лініями зв'язку.

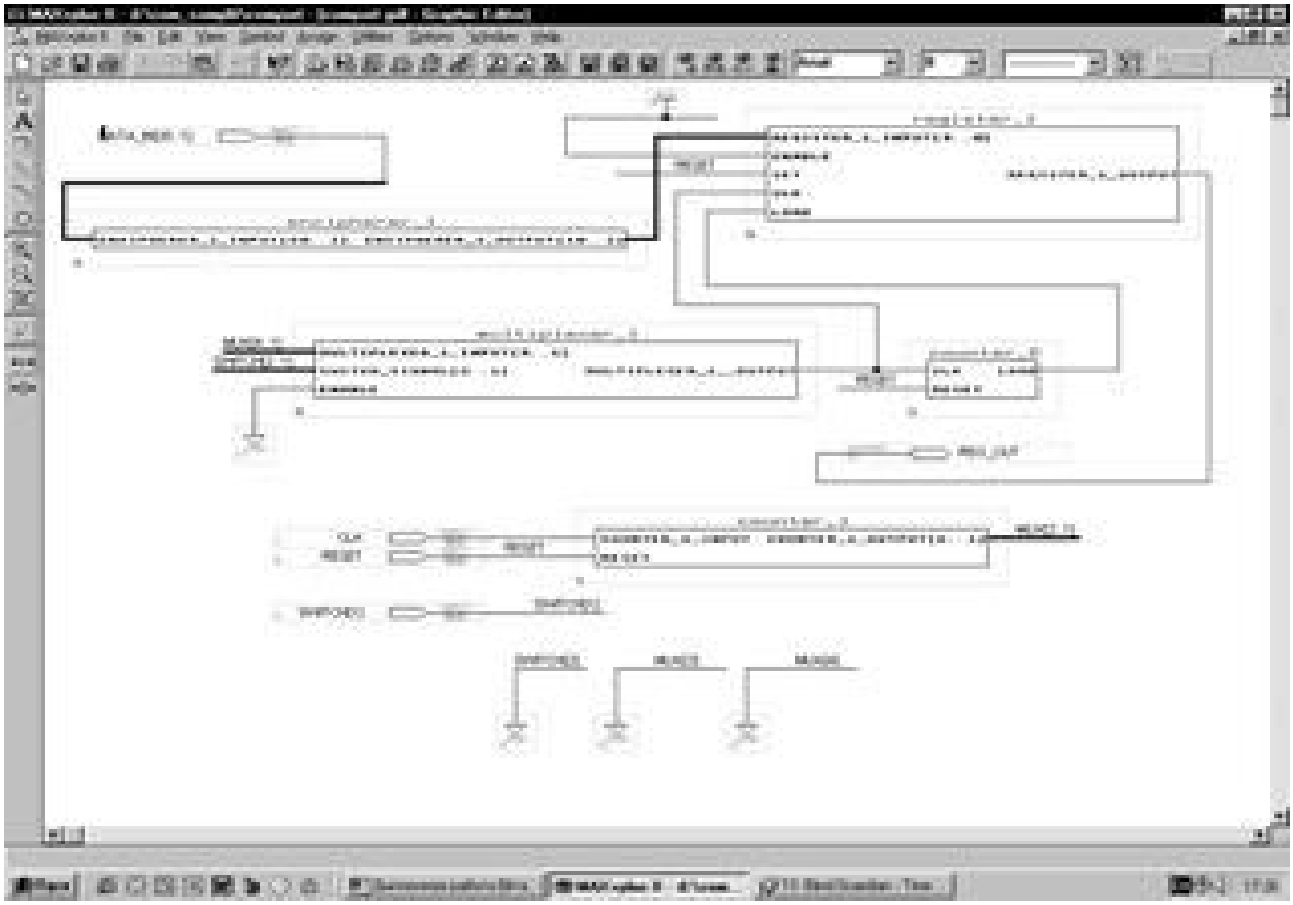


Рис. 4,50 . Реалізація проекту COM-порту в графічному редакторі

Результати програмного тестування реалізації COM-порту в CAD MAX+PLUS II наведено на рис. 4.51 . Детальний процес компіляції всього проекту та його елементів описано в розділі 4 цієї дипломної роботи.

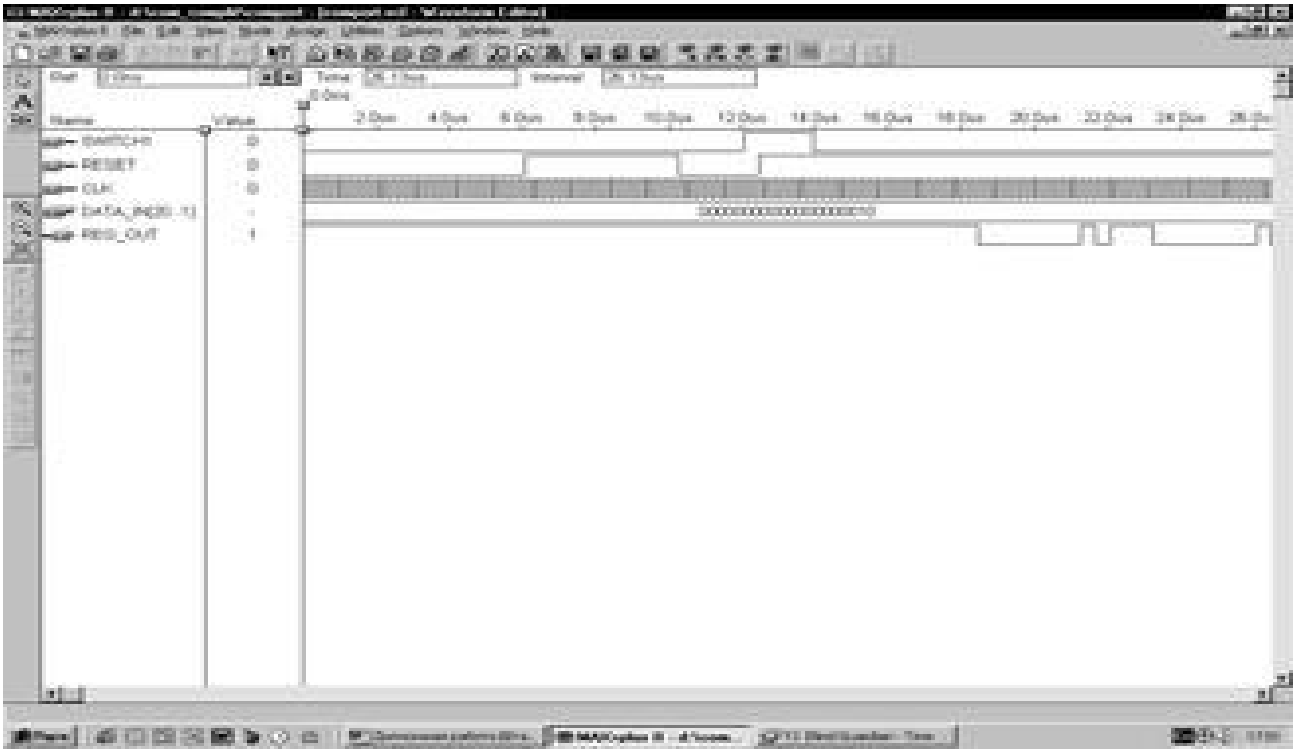


Рис. 4.51 Результати тестування конструкції COM-порту

інформаційною складовою робочої бази з інтегрованим середовищем MAX + PLUS II є опис структури розглянутого середовища, призначення його основних додатків та їх зв'язків, детальний порядок розробки нового проекту в інтегрованому MAX + середовище PLUS II; процес складання створеного проекту дозволить зменшити вартість години, відведеної на навчання спеціалістів, які займаються програмуванням ПЛІС та набуття практичних навичок.

Щоб зробити процес програмування більш зрозумілим, у роботі класифіковано та розкрито архітектуру найпопулярніших інтегральних схем з логікою програмування.

База даних програмних компонентів для роботи з інтегрованим середовищем MAX+PLUS II – це теоретичний опис основних пристроїв мікроелектроніки разом із програмами для їх реалізації:

1) тригер JK;

2) Тригер D;

3) тригер RS;

4) синхронний тригер RS;

5) 4-розрядний послідовний регістр зсуву;

6) 4-розрядний паралельний кільцевий регістр зсуву;

7) 4-розрядний асинхронний лічильник з перехресним переносом за модулем 16;

8) асинхронний лічильник за модулем 10;

9) асинхронний трирозрядний лічильник віднімань;

10) 3-розрядний універсальний лічильник;

11) шифратор 10 на 4;

12) 3-розрядний дешифратор з інвертованими входами;

13) мультиплексор з двома адресними входами, чотирма інформаційними входами і входом дозволу на роботу;

14) демультиплексор з трьома адресними входами, одним інформаційним входом і дозволом на роботу;

15) 4-ярдовий суматор;

Віднімання 4 ярдів .

Оскільки архітектури програмування логічних інтегральних схем останнім часом стрімко розвиваються та вдосконалюються, а засновані на них методи проектування залишаються незмінними, викладений у роботі матеріал може бути використаний як у навчальних, так і в наукових цілях.

РОЗДІЛ V. СИСТЕМА АВТОМАТИЗОВАНОГО ПРОЕКТУВАННЯ OrCAD

5.1. Загальні відомості про систему OrCAD

OrCAD — це набір комп'ютерних програм, призначених для автоматизації проектування електроніки. В основному використовується для створення електронних версій друкованих плат для виробництва друкованих плат, а також для виробництва електронних схем та їх моделювання.

OrCAD Capture - редактор графічних схем;

OrCAD Capture CIS (Component Information System) - графічний редактор діаграм, доповнений інструментом ведення компонентних баз даних; при цьому зареєстровані користувачі отримують доступ через Інтернет (за допомогою ICA, Internet Component Assistant) до каталогу компонентів, який містить понад 200 тис. позицій;

PSpice Schematics - редактор графічних схем, запозичений з пакету DesignLab;

OrCAD PSpice A/D - програма для моделювання аналогових і змішаних аналого-цифрових пристроїв, на яку надсилаються дані як з PSpice Schematics, так і з OrCAD Capture;

OrCAD PSpice Optimizer - програма для параметричної оптимізації;

OrCAD Layout - графічний редактор для друкованих плат;

OrCAD Layout Plus - програма OrCAD Layout, доповнена бездротовим SmartRoute Autorouter, що використовує методи оптимізації нейронної мережі (також використовується в системах Protel 99 SE і P-CAD 2000);

OrCAD Layout Engineer's Edition – програма для перегляду друкованих плат, створених за допомогою Layout або Layout Plus, інструменту для загального розташування компонентів на платі та компонування найбільш важливих схем, які виконує схемотехнік перед випуском друкованої плати. проектне завдання дизайнеру;

OrCAD GerbTool - програма для створення та модифікації керуючих файлів для фотоплотера (розроблена компанією WISE Software Solutions спеціально для OrCAD, аналог програми SAM350);

Загальна характеристика програми OrCAD Capture

OrCAD Capture призначений для створення проекту, частину якого можна вказати у вигляді електричної схеми, а частину можна описати у VHDL високого рівня. Крім того, за допомогою оболонки OrCAD Capture запускаються програми моделювання аналогових, цифрових і змішаних аналого-цифрових пристроїв PSpiee, а також параметрична оптимізація PSpiee Optimizer. В OrCAD Capture проекти діляться на кілька типів.

При створенні проекту відповідно до його типу автоматично завантажуються необхідні бібліотеки компонентів (їх список можна змінити вручну пізніше), при цьому для всіх спеціалізованих проектів можлива передача інформації в OrCAD Layout для створення друкованих плат. На рис. 5.1 показує зв'язок між OrCAD Capture та іншими програмами OrCAD. При створенні діаграм проектів необхідна інформація знаходиться у вбудованій базі даних, яка постачається разом із системою та заповнюється користувачами. Крім того, коли доступна опція Component Information Systems (CIS), офіційні користувачі отримують онлайн-доступ до великої бази даних, що містить інформацію про приблизно 200 000 компонентів від різних компаній (їх символи та корпуси надаються).

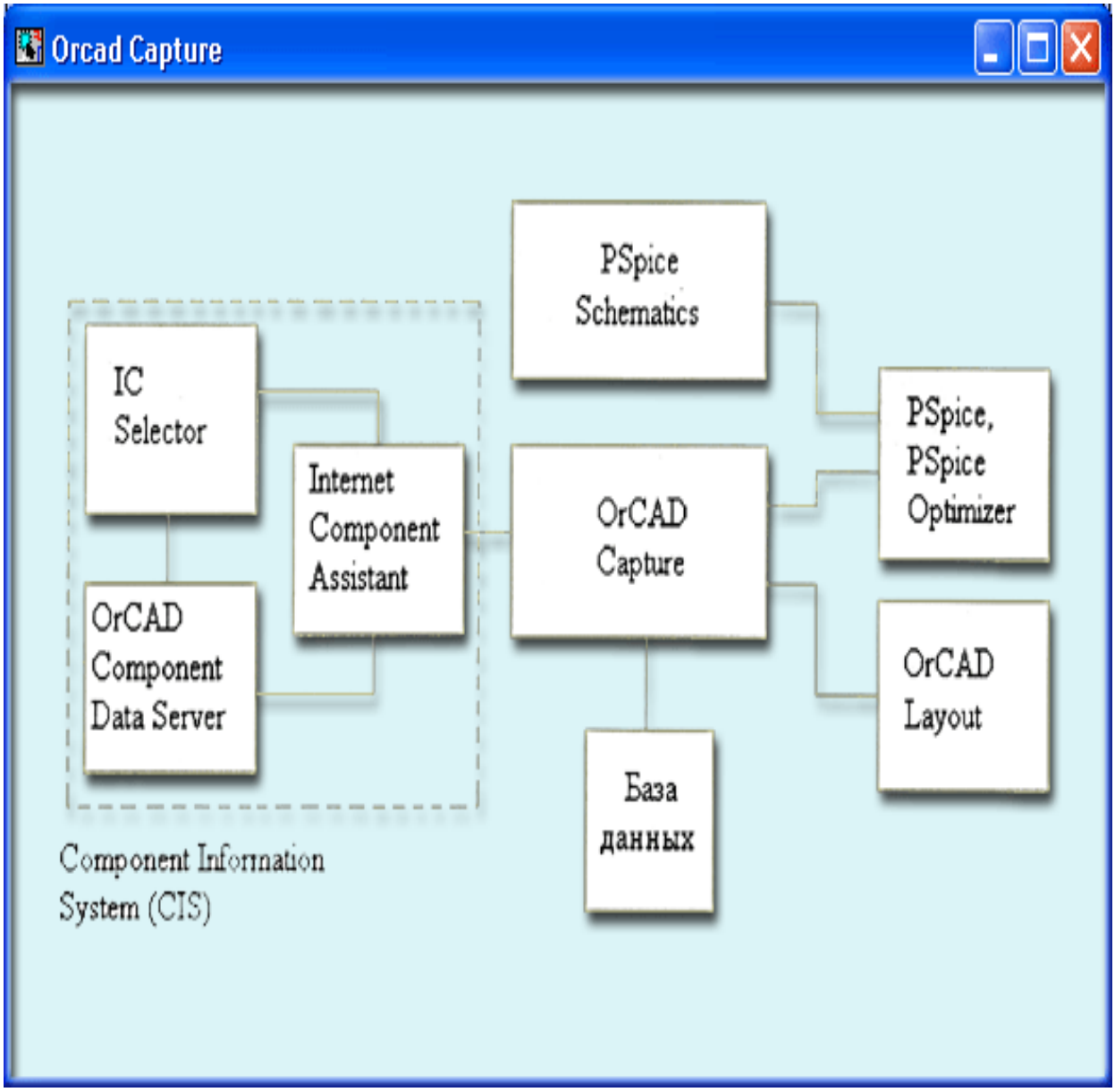


Рис. 5.1. Взаємодія OrCAD Capture з іншими програмами

На рис. 5.2 показано екран OrCAD Capture 9.2. У верхній частині є командне меню, а під ним — панель інструментів.

Меню команд і панель інструментів. Розташування піктограм панелі інструментів залежить від обраного режиму роботи та типу поточного проекту, їх розташування показано на рис. 2.5.3 і наведено в таблиці. 5.1.

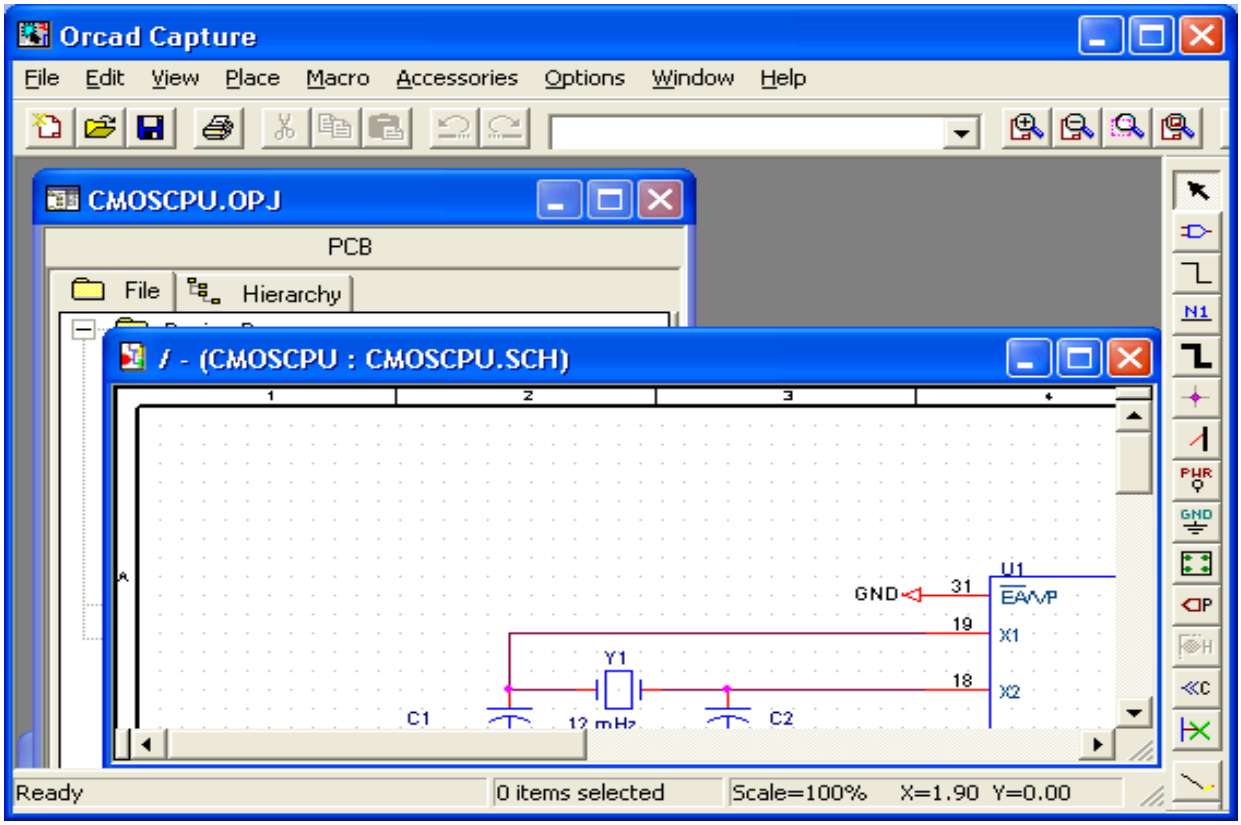


Рис. 5.2. Екран захоплення OrCAD

Таблиця 5.1. Піктограми панелі інструментів

з начок	Еквівалентна команда	Опис команди
	новий	Створення нового документа
	ВІДЧИНЕНО	Відкрийте наявний документ
	Порятунок	Зберегти зміни, внесені в поточний проект
	Роздрукувати	Надрукуйте поточне зображення схеми або символу компонента
	Нарізка	Видалити виділений об'єкт, скопіювавши його в буфер обміну

	Копія	Скопіюйте виділений об’єкт у буфер обміну
	Вставити	Вставити об’єкт із буфера обміну
	Скасувати	Скасовує результат останньої команди
	Повторити	Скасувати результат останньої команди «Скасувати».
	Збліження	Збільшення зображення
	Зона масштабування	Виведення виділеної частини зображення на весь екран
	Збільшити все	Відображення повного зображення сторінки схеми на екрані
	коментар	Присвоєння позиційних позначень елементам виділеної сторінки діаграми
	Назад Анотація	Виконання перестановок логічно еквівалентних складових ділянок і висновків у процесі зворотного регулювання
	Контроль принципів проектування	Перевірка відповідності принципам проектування DRC та принципам електричної схеми ERC

	Створіть список мереж	Компіляція файлу списку підключень вибраної сторони схеми у форматах EDIF 200, SPICE, VHDL, Verilog, Layout тощо.
	Перехресне посилання	Компіляція файлу посилання
	Перелік матеріалів	Підготовка звіту за проектом або обраним століттям
	Прив’язка до сітки	Прив’язка курсору до вузлів сітки у вікні редагування схеми та символу компонента (подібно до «Параметри» > «Параметри» > «Відображення сітки»)
	Керівник проекту	Менеджер проектів Завантажити
	Теми допомоги	Виведення змісту, предметного покажчика та засобів пошуку термінів вбудованих інструкцій

Склад командного меню залежить від обраного режиму роботи та типу поточного проекту. На рис. 5.4 показано вміст командного меню при активації менеджера проектів.

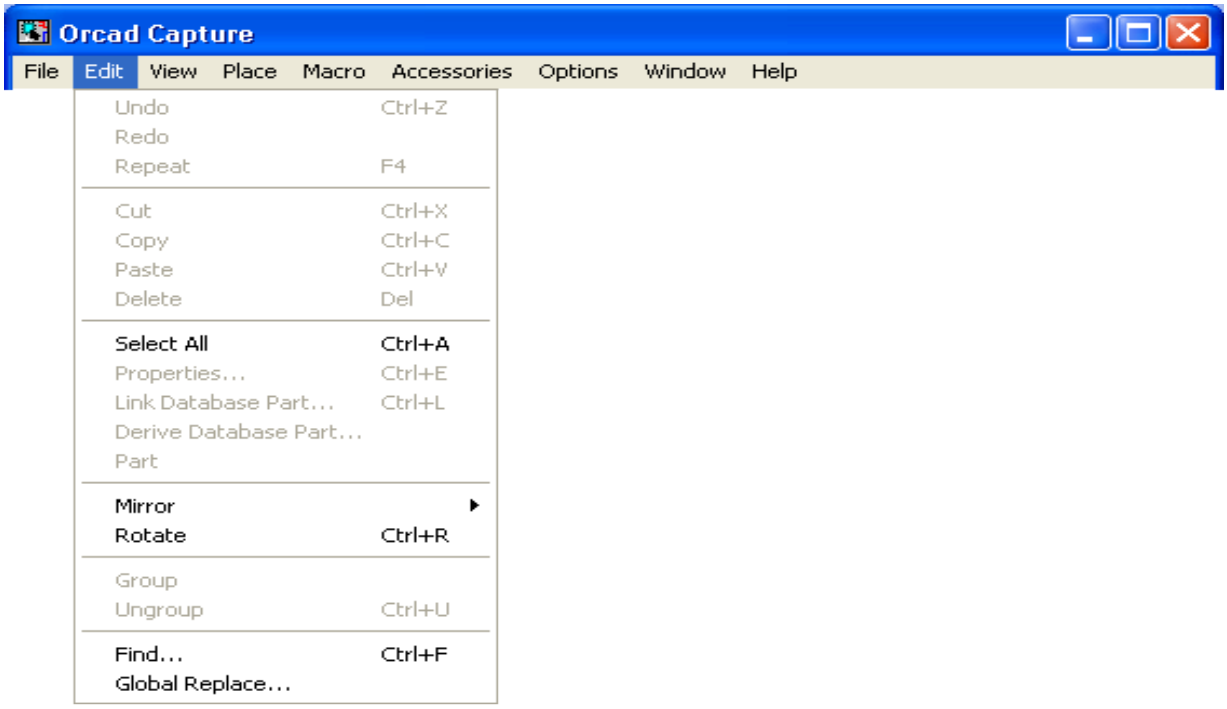


Рис. 5.4. Склад меню команд менеджера проекту

Керівник проекту розташований у лівій частині екрана захоплення. Режим «Файл» реалізує плоску файлову структуру проекту, тоді як режим «Ієрархія» реалізує його ієрархічну структуру. Структура файлу проекту містить кілька розділів:

Design Resource - опис проекту (файл проекту *.dsn, окремі сторінки схеми, список компонентів Design Cache, файли VHDL, список використовуваних бібліотек компонентів *.olb);

Results - результати проектування;

Ресурси PSpice – інформація про моделювання за допомогою PSpice (додані файли, бібліотека моделей, профілі моделювання, файли стимулів) і інші

–

Подвійне клацання лівою кнопкою миші на назві певного файлу або його піктограмі завантажує його у відповідний редактор (при виділенні файлу діаграми завантажується редактор діаграм, при виділенні текстового файлу -

вбудований текстовий редактор). Клацання правою кнопкою миші на значку окремого файлу або каталогу відкриває меню, склад якого залежить від типу вибраного об'єкта:

Add file - додавання файлу;

Parts Manager для завантаження менеджера компонентів;

Правка - редагування файлу;

Властивості - перегляд і редагування властивостей об'єкта;

Нова схема - створення нової схеми;

Властивості проекту – редагування параметрів проекту;

Зберегти – зберегти внесені зміни;

Зберегти як... - зберігає зміни, внесені в проект, під новою назвою;

Імітація вибраного(их) профілю(ів) - виконання моделювання за використовуючи PSpice. Робіть належним чином обраний профіль (файл завдання УВІМКНЕНО моделювання) ;

Переглянути результати моделювання - див графічний результати моделювання ;

Переглянути Вихід Файл – перегляд текстового файлу з результатами моделювання ;

Редагувати параметри симуляції - редагування завдання УВІМКНЕНО моделювання ;

Зробити Active – активація вибраного профілю ;

новий Сторінка - додавання нової сторінки схеми;

Редагувати Сторінка - редагування сторінки схеми ;

Схематичний сторона Properties - редагування параметрів конфігурації редактора схем;

Редагувати обраний об'єкт властивості - редагування атрибутів виділеного об'єкта на діаграмі;

Зробити Корінь - перемістити вибрану схему на вищий рівень ієрархії ;

Перейменувати – змінити назву файлу.

Редактор схем. На рис. На рис. 5.5 показано схематичне вікно редактора сторінки з додатковими панелями інструментів (рис. 5.6), команди яких наведено в таблиці. 2.2 і 2.3.

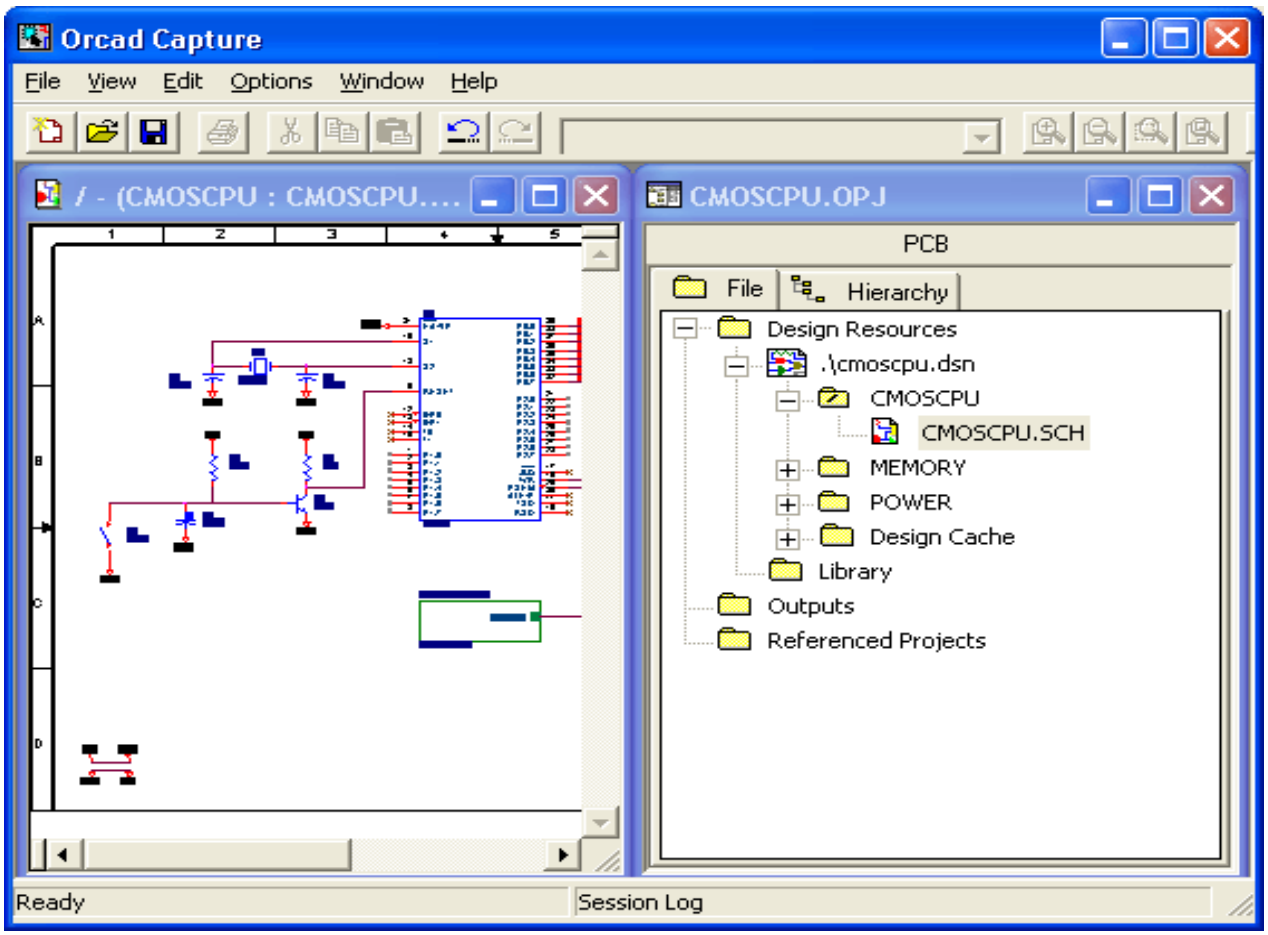


Рис. 5.5. Вікно редактора сторінки схеми

Рис. 5.6. Панелі інструментів редактора схем

Таблиця 5.2. Значки панелі інструментів режиму редагування схеми

значок	Еквівалентна команда	Опис команди
	Вибирати	Режим вибору об'єкта
	Привіт	Вибір компонента в бібліотеці для розміщення його символу на діаграмі

	Дріт	Креслення електричних кіл. Натискання клавіші Shift дозволяє вводити неортогональні рядки
	Мережевий псевдонім	Проставлення псевдонімів (додаткових імен) для ланцюгів і шин
	Автобус	Зображення автобуса (групова лінія зв'язку)
	Вузол	Побудова точки електричного з’єднання двох кіл
	Вхід автобуса	Накладення мітчиків осі шини під кутом 45°
	потужність	Розміщення символів для висновків про джерела живлення та «землю»
	Земля	клем джерела живлення та «заземлення»
	Ієрархічний блок	Розташування ієрархічних блоків
	Ієрархічний порт	Розташування портів в ієрархічних блоках
	Ієрархічний кілок	Розташування висновків ієрархічних блоків
	З’єднувач поза сторінкою	Розташування символів з’єднувача сторінки
	Немає з'єднання	Підключення до виходу символу відсутності підключення
	лінія	Штриховий малюнок

	Ломана лінія	Малювання пунктирною лінією
	Прямокутник	Креслення прямокутника
	Еліпс	Малювання еліпса/кола
	Лук	Малювання дуги
	текст	Розмістіть один або кілька рядків тексту, вказавши його розмір, колір, орієнтацію та шрифт
	Новий профіль симуляції	Створення нового файлу завдання моделювання
	Змінити параметри симуляції	Редагування імітаційного завдання
	Запустіть PSpice	Запустіть Pspice
	Переглянути результати моделювання	Перегляньте результати графічного моделювання
	Маркер напруги/рівня	Розташування маркера напруги/логічного рівня
	Мітки різниці напруг	Розміщення двох маркерів різниці напруг
	Поточний тег	Розміщення двох маркерів різниці напруг

	Маркер розсіювання потужності	Встановлення маркера розсіювання
	Увімкнути відображення напруги зміщення	Відображення вузлових напруг у робочій точці на графіку
	Перемкніть напругу в обраній мережі	Показати/видалити значення потенціалу постійного струму вибраного кола
	Увімкнути відображення поточного відхилення	Відображення струмів відгалужень у робочій точці на графіку
	Перемикання струму на вибрану частину/контакт	Показати/видалити значення постійного струму вибраного контакту компонента
	Увімкнути відображення сили повороту	Відображення гілки розподіленої потужності в робочій точці на графіку
	Перемкніть живлення вибраної частини	Показати/видалити значення втрати потужності постійного струму вибраного компонента

Текстовий редактор. Текстовий редактор дозволяє створювати та переглядати файли VHDL та інші текстові файли. На рис. 5.7 показано фрагмент файлу VHDL, ключові слова та коментарі виділені для ясності

різними кольорами, вказаними в розділі «Параметри» меню «Параметри». Завантаження файлу VHDL в редактор здійснюється подвійним клацанням лівої кнопки миші при наведенні курсору на назву файлу в менеджері проектів, текстові файли інших типів відкриваються звичайним способом за допомогою команди Файл > Відкрити > Текстовий файл .

Рядок стану. У нижній частині екрана захоплення знаходиться рядок стану (рис. 5.8), у якому відображається назва вибраного інструменту або меню, назва поточного стану програми (у лівому полі), кількість виділених об’єктів (у середнє поле), масштаб зображення та поточні координати курсору (у правому полі).

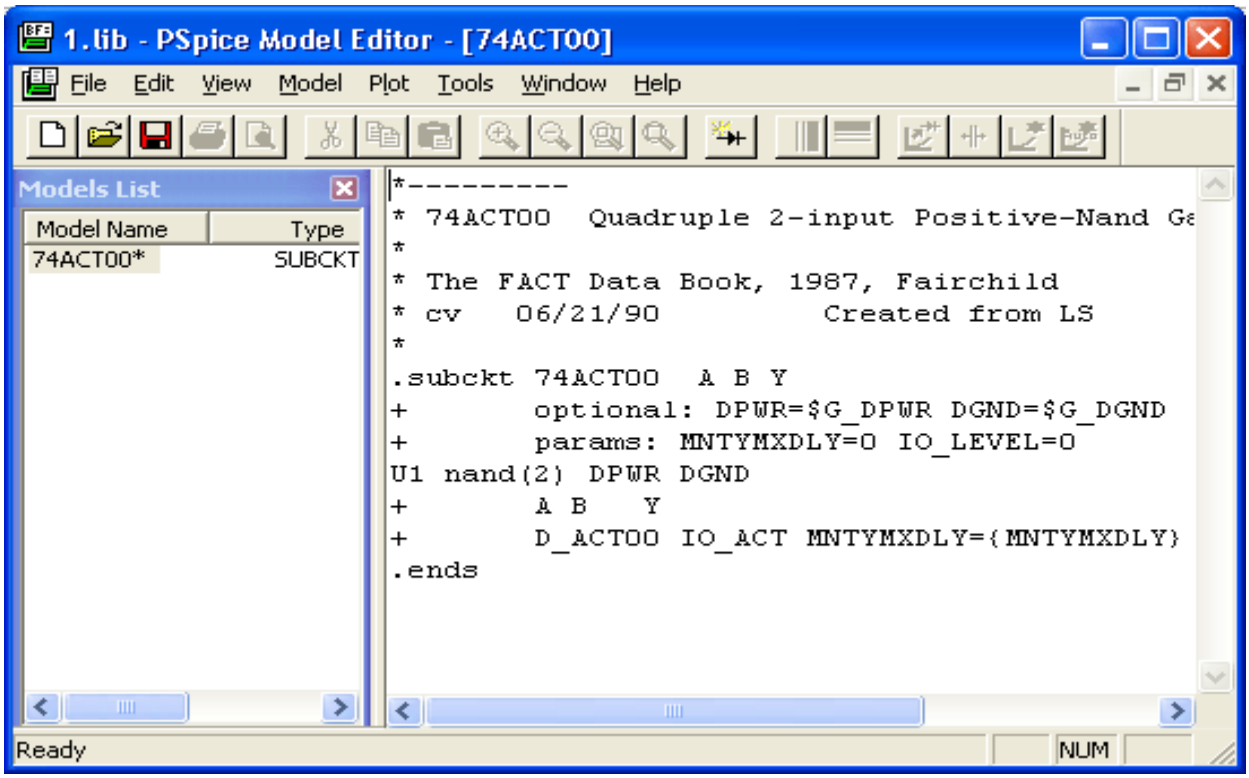


Рис. 5.7. Панель інструментів редактора символів

Вибір об'єктів. Вибравши об’єкт або групу об’єктів, ви можете виконувати різні операції, зокрема переміщення, копіювання, видалення, дзеркальне відображення, обертання, масштабування та редагування. При редагуванні текстових файлів, у тому числі файлів VHDL, використовуються стандартні методи виділення об'єктів, прийняті в MS Word і подібних програмах.

Під час редагування графічних файлів (схем і умовних позначень окремих елементів) виділення окремого об’єкта здійснюється клацанням лівої кнопки миші при наведенні курсору на виділений об’єкт (перехід у режим виділення автоматично позначається на панелі інструментів підсвічуванням значка). Об’єкт скасовується клацанням миші, встановлюючи курсор у порожнє місце на екрані. Додавання об’єкта до виділеної групи об’єктів здійснюється клацанням лівої кнопки миші з утриманою клавішею CTRL. Видалення об’єкта з виділеної групи також здійснюється утримуванням клавіші CTRL.

Додатково є можливість виділяти об'єкти у вікні (при цьому перед тим, як «розтягнути» вікно переміщенням курсору з натиснутою лівою кнопкою миші, необхідно активувати режим виділення, клацнувши на значку). Вибір усіх об’єктів на схемному аркуші виконується за допомогою команди «Правка» > «Виділити все». Щоб вибрати об’єкти, що накладаються, утримуйте клавішу Tab.

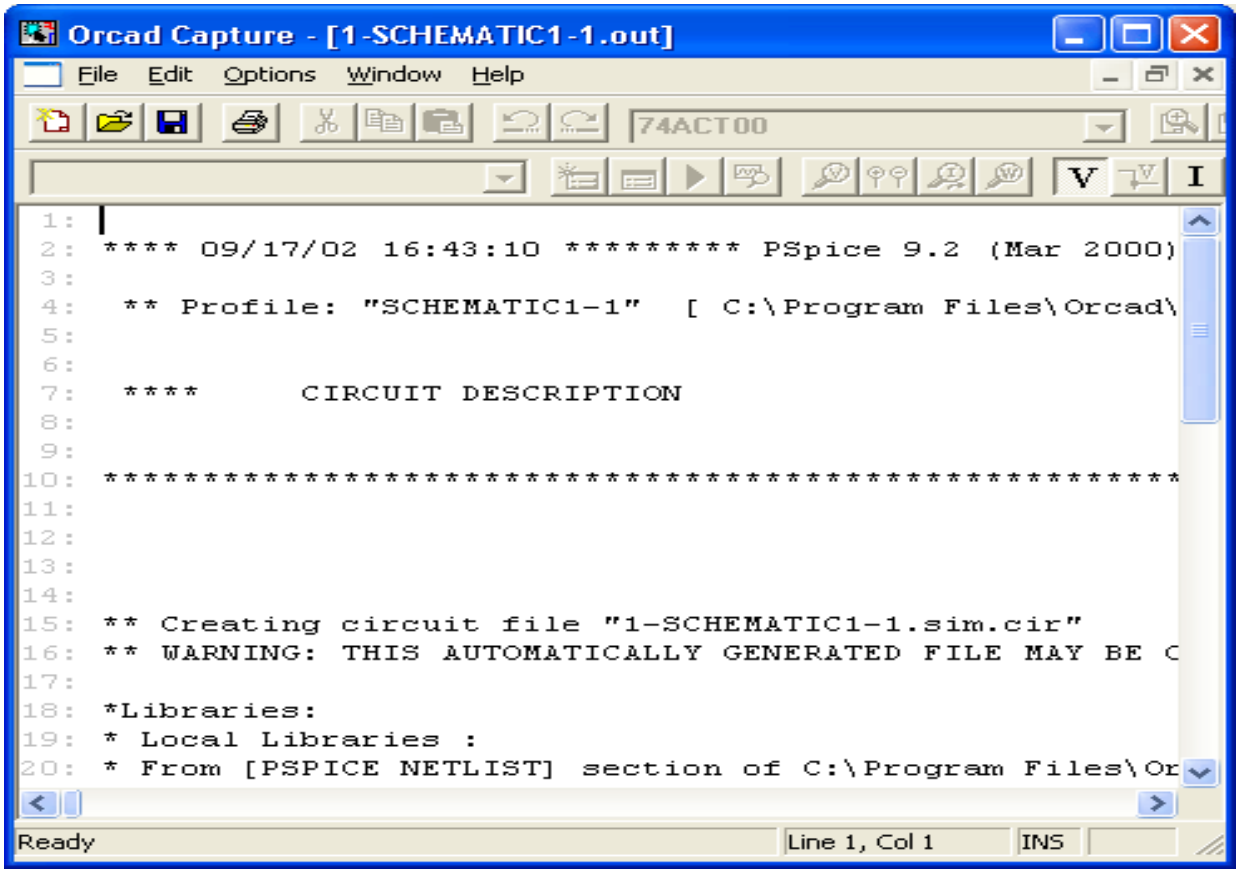


Рис. 5.8. Перегляд і редагування файлів VHDL

Редагування властивостей об'єкта. Кожен об’єкт схеми має набір властивостей (Properties), які повністю визначають його характеристики. Ці зручності включають:

Ієрархічні порти – висновки з ієрархічної складової;

Disabled page _ connectors - з'єднувачі сторінки діаграми;

символи DRC - символи помилок;

Закладки – закладки;

Частини - складові символи (включаючи ієрархічні блоки)

Сітки - ланцюги;

Піни - складові висновки;

Титульна табличка - основний напис листа схеми (кутовий штамп).

Кожна функція компонента (або атрибут у термінології DesignLab) має назву та відповідне значення. Наприклад, біполярний транзистор з позиційним позначенням Q1 має атрибут RSV Footprint (case type), який приймає значення TO206AA, атрибут Implementation Type = PSpice Model (тип математичної моделі PSpice), атрибут Implementation (назва математична модель), яка приймає значення КТ315 та інші (див. рис. 5.9).

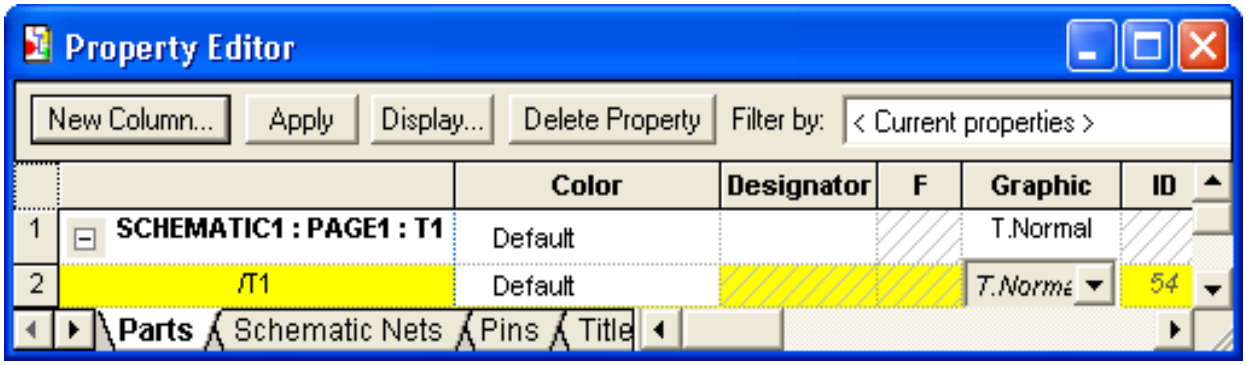


Рис. 5.9. Список характеристик транзистора Q1

Властивості (атрибути) одного або кількох елементів схеми переглядаються та редагуються за допомогою Редактора властивостей, який викликається командою Правка>Властивості (він також активується подвійним клацанням курсора на зображенні символу елемента або праворуч

натисніть контекстне меню). Перегляд електронних таблиць властивостей об’єктів проекту різних типів здійснюється за допомогою команд менеджера проекту Правка > Огляд > Частини, Мережі (рис. 5.10). Перед переглядом електронної таблиці користувач повинен вибрати тип об'єктів:

екземпляр - об'єкти, які можна використовувати кілька разів у проекті

екземпляр - окремі об'єкти, розміщені в поточному проекті (переважно).

В електронних таблицях можна редагувати лише властивості об’єктів екземплярного типу; властивості об’єктів екземплярного типу можна редагувати лише за допомогою редактора властивостей.

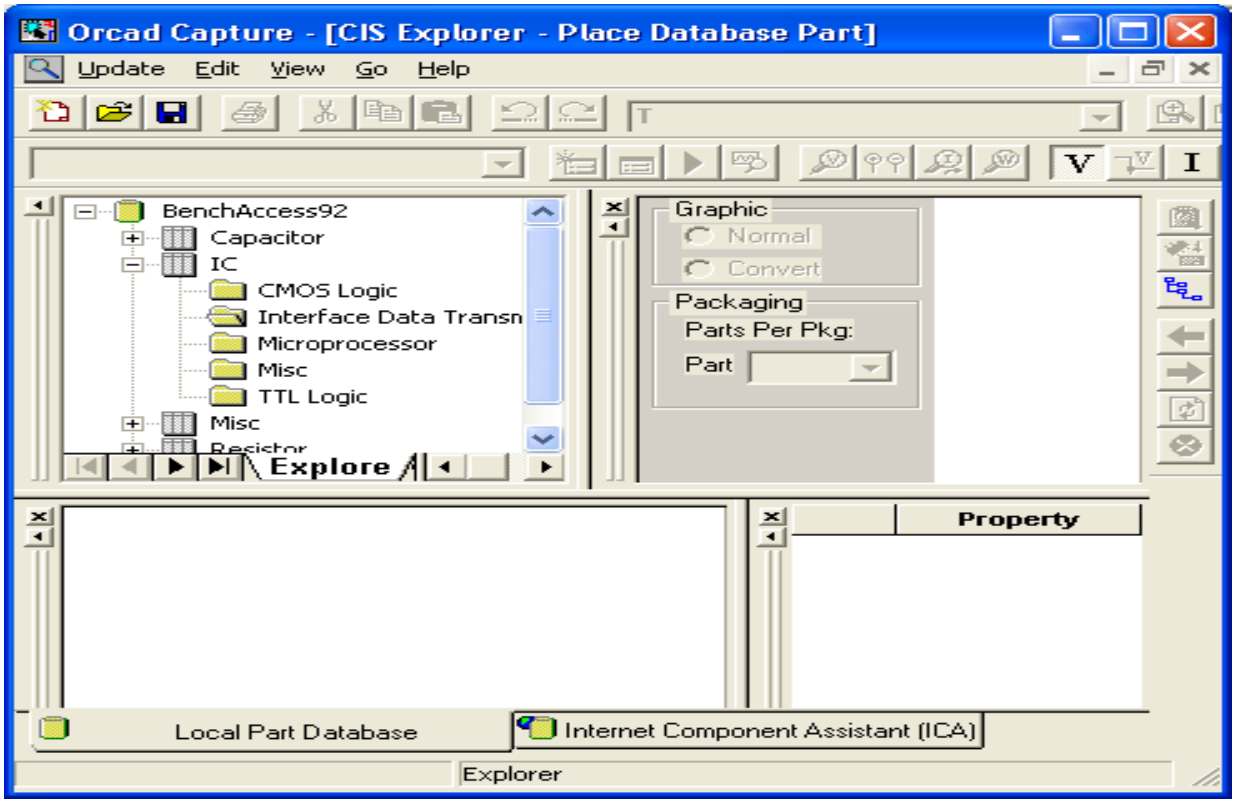


Рис. 5/10. Переглянути характеристики проекту в електронній таблиці

Переміщення та зміна розмірів графічних об'єктів. Ви можете змінювати розмір і форму деяких графічних об’єктів, таких як напрямні, автобуси (лінії групового транспорту), лінії, еліпси (особливо кола), прямокутники та багатокутники. Усі інші об’єкти можна лише переміщувати, повертати, віддзеркалювати та видаляти. Об’єкти для редагування необхідно виділити заздалегідь — у результаті для кожного виділеного графічного об’єкта на екрані відображаються спеціальні значки (рис. 5.11). Щоб змінити форму або розмір графічних об’єктів, клацніть лівою кнопкою миші на одному з цих

значків, а потім, не відпускаючи кнопку, перемістіть курсор відповідно; редагування закінчується, коли ви відпускаєте ліву кнопку миші.

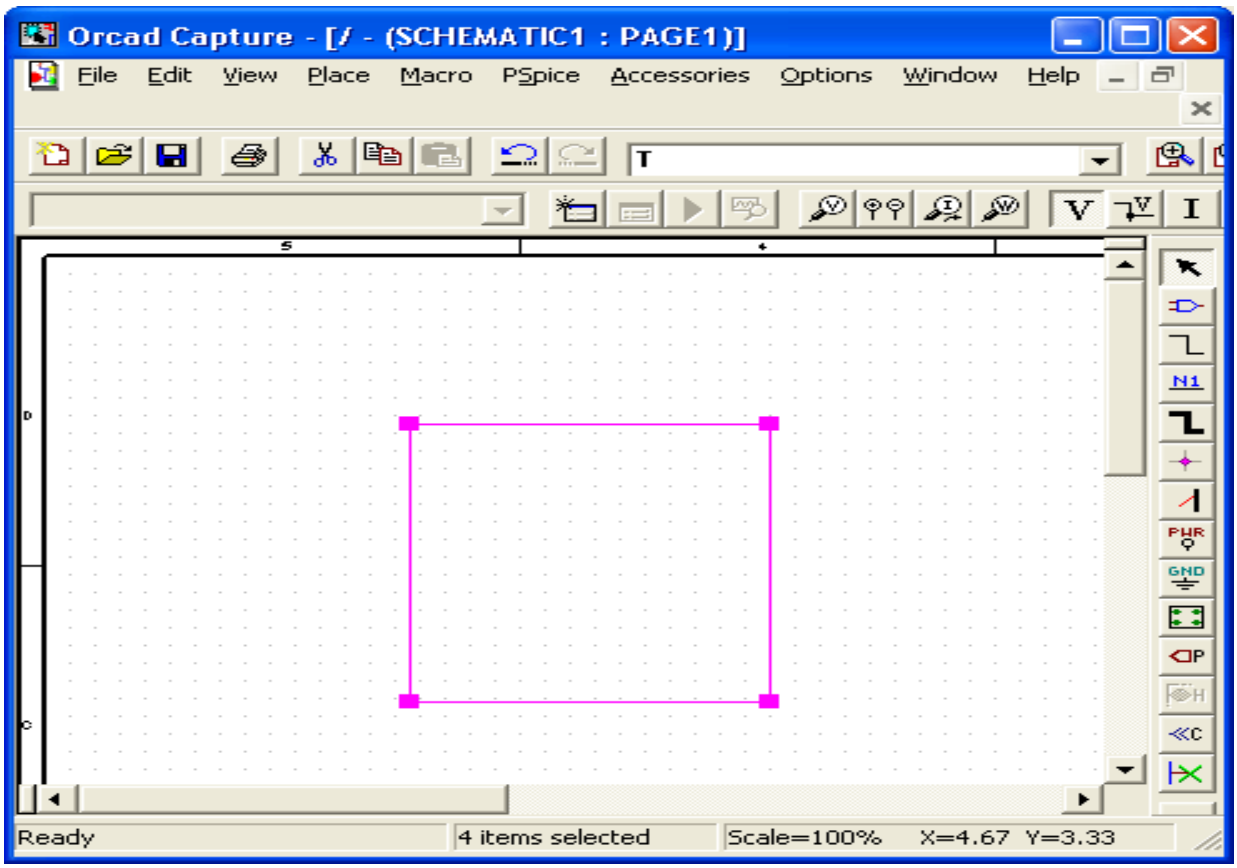


Рис. 5.11. Зміна розміру об'єктів

Щоб перемістити будь-який виділений об’єкт, клацніть курсором лівою кнопкою миші будь-де на контурі об’єкта, крім цих піктограм, а потім, не відпускаючи кнопку, перемістіть. Під час переміщення групи об'єктів курсор змінює свою форму (набуває вигляду зірки) і може бути розміщений у будь-якому місці контуру, що оточує виділену групу.

Назад і назад, повторюючи останню операцію. Команда «Правка» > «Скасувати» скасовує виконання останньої команди («скасувати»), тоді як ім’я останньої виконаної команди «Помістити», «Видалити», «Копіювати», «Назад» автоматично додається до назви команди «Скасувати» у підменю «Правка» Змінити розмір, повернути, віддзеркалити, наприклад, команда «Правка» > «Повторити» скасовує команду «Правка» > «Скасувати» («скасувати» вперед).

Повторне виконання останньої команди «Помістити», «Копіювати», «Вставити», «Перемістити», «Змінити розмір», «Повернути», «Дзеркало» виконується за допомогою команди «Правка>Повторити», тоді як назва останньої виконаної команди автоматично додається до назви команди «Повторити» у вікні «Правка». вікно підменю. Щоб перемістити скопійований об’єкт на певну відстань, перед виконанням «Правка» > «Повторне копіювання» виділіть скопійований об’єкт, натисніть клавішу CTRL і, не відпускаючи, перемістіть скопійований об’єкт на потрібну відстань. Потім, наступне виконання команди «Правка» > «Повторне копіювання» (F4) створює масив скопійованих об’єктів, віддалених один від одного на задану відстань (зручно, наприклад, при створенні шин).

Проекти, створені за допомогою програми OrCAD Capture, зберігаються у файлах з розширенням .opj (за прийнятою в програмі термінологією проект називається Project), які містять посилання на імена всіх використовуваних файлів: файли окремих схем (* .dsn, відповідно до прийнятої термінології файли схем називаються проектами, також перекладається як «проект»), бібліотеками, текстовими файлами VHDL, файлами звітів про проект тощо. Файл проекту може містити посилання на одну або кілька папок (ці папки відображаються у вікні менеджера проекту (див. рис. 5.2), які пов’язані з основними файлами схеми. Папка зі схемами містить одну або кілька сторінок зі схемами. Файл схеми також містить кеш дизайну - кеш дизайну, який містить копії символів компонентів, що використовуються на схемі. Проект може посилатися на декілька бібліотек. Однак він може мати лише одну схему (файл із розширенням .dsn), що складається з однієї чи кількох сторінок. Ви можете створити новий проект, а потім створити нові схеми, бібліотеки та файли VHDL. Для створення нового проекту виконується команда Файл > Новий проект, після чого в діалоговому вікні (рис. 5.12) у рядку Назва вказується назва проекту (у випадку моделювання припущень кириличні символи не допускаються).), а ім’я підкаталогу розташування вказується в рядку проекту Location (при цьому для перегляду файлової структури зручно

використовувати кнопку Browse). Потім у середній частині цього вікна вибирається тип проекту.

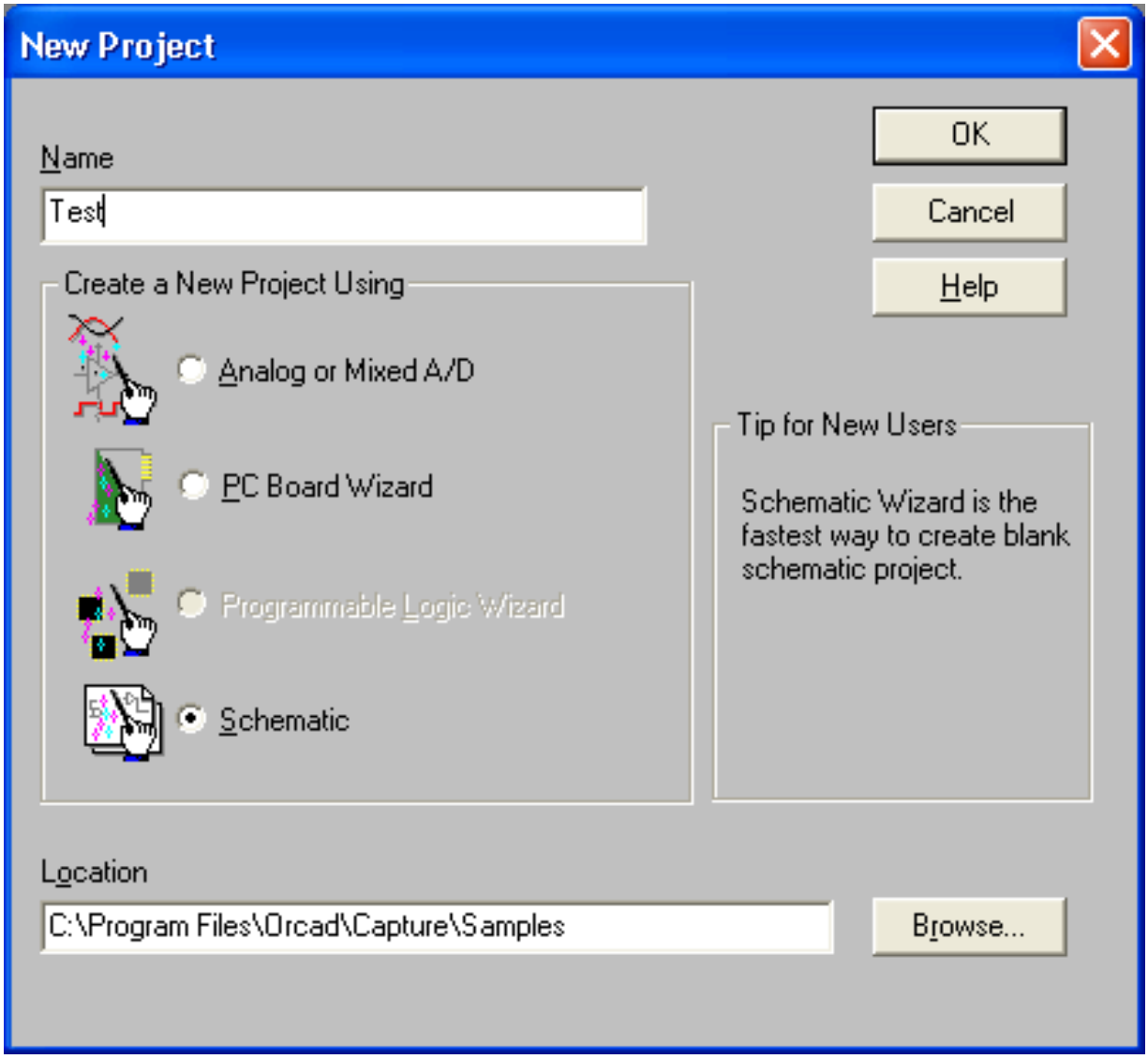


Рис. 5/12 Відображення невидимих контактів живлення

Аналогова або змішана схема - аналогові, цифрові або змішані аналого-цифрові пристрої, змодельовані за допомогою програми PSpice A/D (подальше розширення друкованої плати також можливе за допомогою програми OrCAD Layout). На початку створення проекту планується завантажити прототип, вказавши його назву на зображенні, наведеному на рис. 5.13, а в діалоговому вікні (опція «Створити на основі існуючого проекту») ви можете завантажити один із 4 стандартних прототипів або будь-який раніше створений проект.

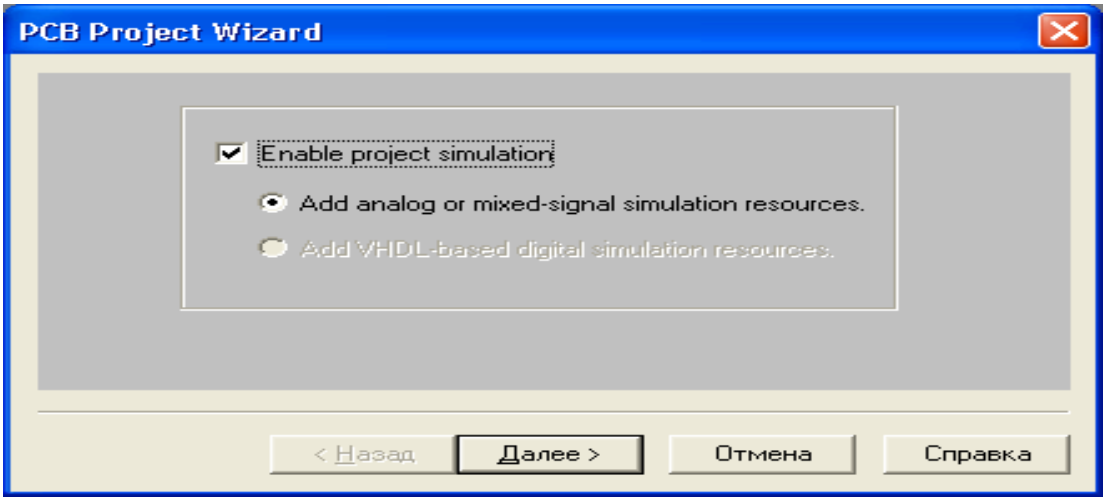


Рис. 5.13, а) Вибір прототипу дизайну (a) або можливості моделювання дизайну RSV (b) з вибором бібліотек символів Pspice

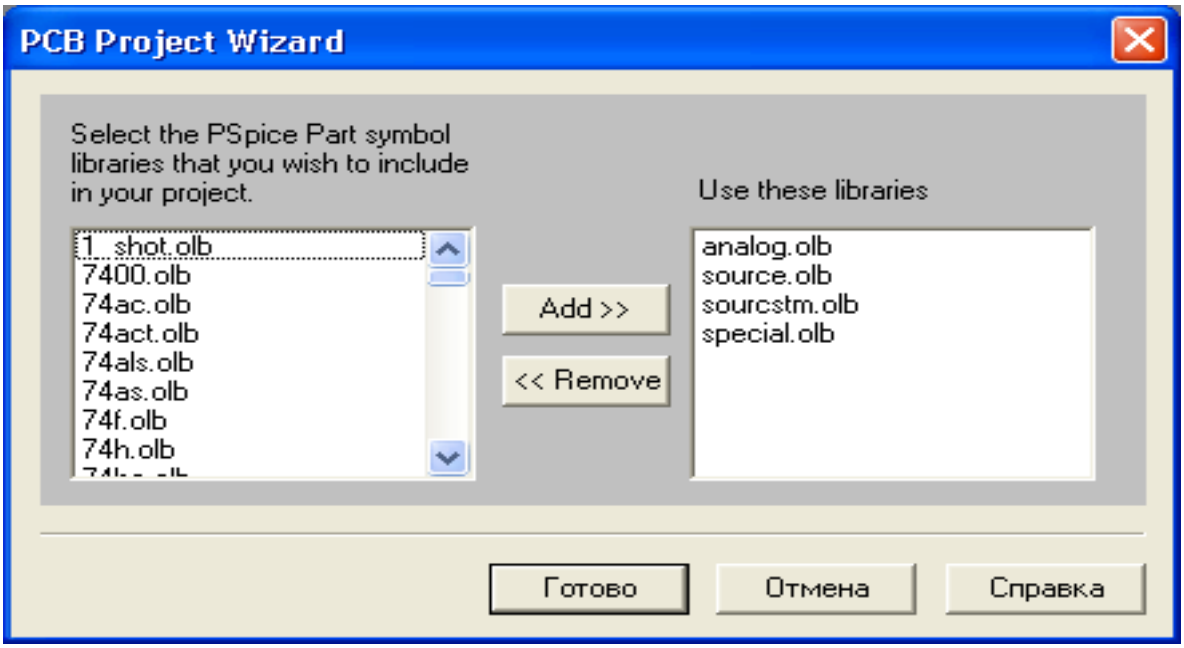


Рис. 5.13, б)

PC Board - друковані плати (моделювання змішаних аналого-цифрових пристроїв виконується за допомогою PSpice). Необхідність виконання моделювання вказується в діалоговому вікні, зображеному на рис. 5.13, b - для цього виберіть опцію Enable project simulation, щоб додати до проекту список бібліотек символів PSpice (Add analog or mixed signal simulation resources).

Схематичні – неспеціалізовані проекти (можливе лише створення та документування принципових схем, моделювання та розробка друкованих плат не передбачено).

Увага

Вибір типу проекту визначає набір команд меню OrCAD Capture, який не є настільки принциповим, тому що обмін даними можливий між будь-якими проектами.

Стандартні діалогові вікна Windows Print Setup, Print Preview і Print використовуються для надсилання даних на принтер, плотер або файл PostScript (*.prn). Команди виводу можна вибрати в меню «Файл» менеджера проекту або в OrCAD Capture, OrCAD PSpice тощо. Ви можете надрукувати сторінку схеми, символ компонента, інформацію про упаковку, текстові файли тощо в такому порядку:

1. Команда Print Setup у меню File налаштовує принтер/плотер;
2. За допомогою команди Print Preview з меню File встановлюються параметри (рис. 5.14):

   Масштабувати до розміру паперу: автоматично масштабує зображення таким чином, щоб усі вибрані сторінки діаграми проекту повністю помістилися на аркуші паперу вказаного нижче розміру. Масштабувати до розміру сторінки: автоматично масштабувати зображення таким чином, щоб вибрана сторінка діаграми повністю розміщувалася на аркуші паперу з розмірами, зазначеними нижче;

   Розмір сторінки: формат паперу - A4, A3, A2, A1, A0, Custom (задається користувачем);

   Продаж: масштаб зображення;

   X, Y print shifts: зміщення зображення по горизонталі та вертикалі;

   Якість друку: якість друку (роздільна здатність);

   Копії: кількість копій;

   Друкувати у файл: надсилати зображення у файл;

   Сортування копій: порядок друку копій - спочатку виводяться всі копії першої сторінки, потім виводяться всі копії другої сторінки і т.д . ;

   Force Black & White: виведення чорно-білого зображення.
3. Вибір друкованих сторінок схеми здійснюється різними способами:

у вікні менеджера проекту вибрано одну або кілька сторінок схеми;

потрібна сторінка схеми буде завантажена в редактор схем;

Щоб надрукувати всі сторінки діаграми дизайну, просто виберіть назву діаграми дизайну (Design) у менеджері проектів.

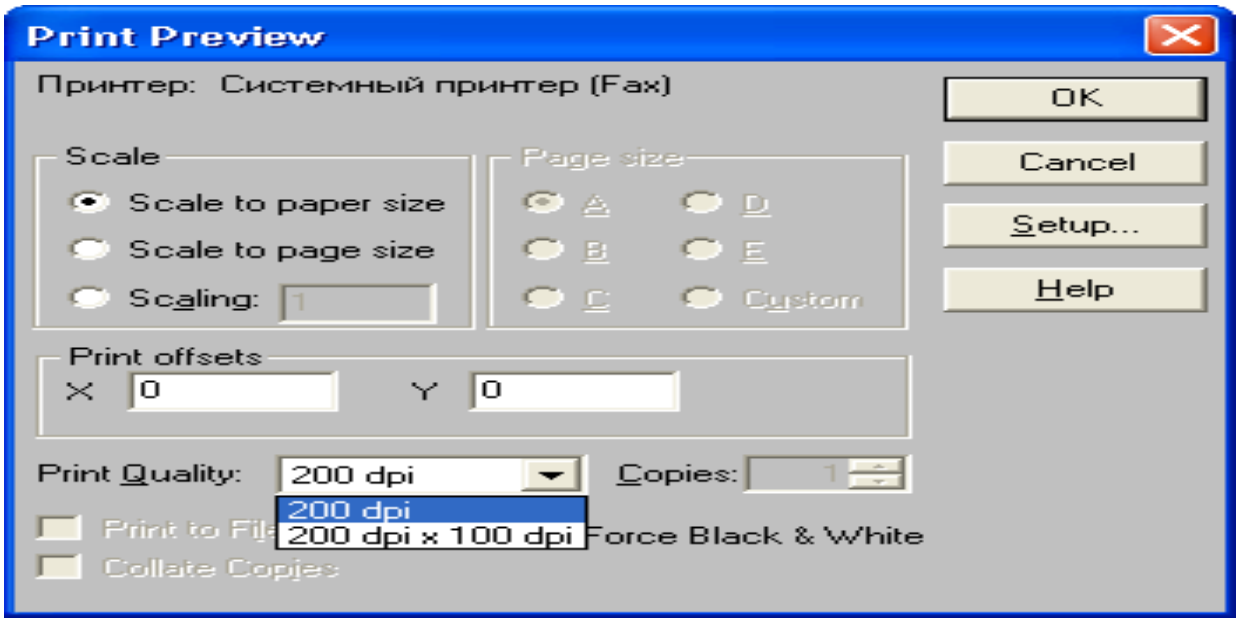


Рис. 5.14. Встановіть параметри друку

Електричні схеми більшості проектів розміщені на кількох сторінках не дуже великого формату. Існує два способи організації діаграм великого обсягу: плоскі звичайні багатосторінкові структури та ієрархічні структури.

Електричні кола, розташовані по різні боки багатосторінкової схеми, з'єднуються між собою за допомогою т. зв позасторінкові з’єднувачі з однаковою назвою. Усі сторінки таких діаграм розташовані в одній папці на одному рівні. Їх структура відображається в менеджері проектів після натискання кнопки Файл.

Діаграми ієрархічного проектування містять спеціальні символи, які називаються ієрархічними блоками. Схема кожного такого блоку розміщується у вигляді окремої схеми, яка розміщується в папці на тому ж рівні ієрархії, що й основна схема.

Перш ніж створювати новий проект за допомогою OrCAD Capture, ви повинні встановити його параметри конфігурації за допомогою трьох команд у меню «Параметри менеджера проекту»:

- за допомогою команди Preferences задаються параметри схеми, які зберігаються в конфігураційному файлі Capture.ini і які не запускаються при кожному запуску програми OrCAD Capture; зміни цих параметрів вносяться у вже існуючі схеми; якщо проект створюється в іншій системі OrCAD, будуть враховані параметри, що містяться в поточному файлі Capture.ini;

- команда Design Template задає параметри схеми, які встановлюються за замовчуванням при створенні всіх нових проектів (вони вводяться в розділ [Design Template] файлу Capture.ini); зміни цих параметрів не вносяться в існуючі схеми, тому перед створенням нових схем варто переглянути і при необхідності змінити їх значення;

- параметри окремої поточної схеми задаються за допомогою команди Властивості проекту або Властивості сторінки Схеми.

Розглянемо ці способи налаштування проекту докладніше.

На вкладці «Колір / Друк» ви можете переглядати та встановлювати кольори всіх об’єктів на діаграмі (за допомогою палітри, яка відкривається клацанням лівою кнопкою миші на намальованому прямокутнику) та вибирати об’єкти, які потрібно надрукувати (для для цього в колонці навпроти назви об’єкта Друк стоїть позначка); колір основного тексту (кутовий штамп, головний блок) також призначається рамці креслення (Border) і лініям сітки із засічками на рамці креслення (Grid Reference); кольори графічних об'єктів (ліній, багатокутників і дуг) задаються на вкладці «Різне», якщо на цій вкладці визначені кольори за замовчуванням, вони встановлюються відповідно до кольору графіки (Графіка) на вкладці «Колір / Друк».

На вкладці Відображення сітки окремо для редактора діаграм та символів виберіть стиль зображень сітки у вигляді точок (крапок) або ліній (ліній); панель Display підкреслює необхідність відображення сітки на екрані дисплея, а панель Snap to Grid підкреслює необхідність «прив’язки» курсора до вузлів сітки під час розміщення об’єктів на діаграмі.

Вкладка «Панорамування та масштабування» визначає коефіцієнт збільшення зображення (коефіцієнт масштабування) і коефіцієнт

панорамування (відсоток автоматичного прокручування) для редактора діаграм і символів (діаграма панорамується, тобто переміщується без зміни масштабу, коли курсор наближається до межі робоче вікно, якщо натиснути й утримувати ліву кнопку миші, залишиться).

На вкладці «Вибір» можна встановити, чи будуть виділені об’єкти, якщо вони перетинаються межею прямокутника виділення (Перетин), або якщо вони повністю знаходяться всередині області виділення (Повністю закриті); На панелі «Максимальна кількість об’єктів високої роздільної здатності під час перетягування» вказується максимальна кількість об’єктів, які відображаються на екрані, коли ви вибираєте їх у вікні та переміщуєте.

На вкладці «Різне» виберіть стиль заливки для закритих фігур (Стиль заливки), стиль і ширину лінії (Стиль і ширина лінії), колір графічних об’єктів (Колір), а також шрифт, який використовується в менеджері проектів і файл журналу сеансу; додатково встановлюються такі параметри:

- Візуалізація тонів True Type зі штрихами - Зображення шрифтів True Type як векторні шрифти штрихів (для друку);

- Fill text - заливка шрифту;

- Увімкнути автоматичне відновлення - автоматичне збереження файлів проекту, діаграм і текстів VHDL в каталозі \WINDOWS\TEMP\AUTOSAVE;

- Оновлення кожні xxx хвилин - автоматичний інтервал збереження файлів у хвилинах;

- Автоматично посилатися на розміщені частини - автоматичне призначення позиційних позначок для елементів, розміщених на схемі;

- Enable Intertool Communication (ITC) - включення режиму перевірки та відображення результатів на екрані при передачі даних з інших системних програм OrCAD, таких як OrCAD Layout і OrCAD PSpice; наприклад, коли режим ITC увімкнено між OrCAD Capture та OrCAD Layout, між схемою та друкованою платою встановлюється «гаряче» з’єднання (перехресне тестування).

Текстовий редактор, який використовується при роботі з файлами VHDL, налаштовується на вкладці Текстовий редактор. Панель підсвічування синтаксису визначає виділені кольори для ключових слів, коментарів, рядків у подвійних лапках та ідентифікаторів. На панелі «Поточний параметр шрифту» при натисканні кнопки «Установити» встановлюється розмір шрифту тексту, а колір об’єктів не підсвічується. Конкретні об’єкти будуть виділені, якщо ви виберете панель «Виділити ключові слова», «Коментарі», «Ідентифікатори» та «Рядки в лапках». Інтервал табуляції текстового редактора вказується на панелі Інтервал табуляції.

Команда «Шаблон проекту» визначає набір параметрів для нових проектів, деякі з яких можна перезаписати для окремих сторінок схеми. Вкладки діалогу для цієї команди показано на рис. 2. 2.21.

На вкладці Шрифти (рис. 5.15, а) визначаються шрифти для текстів різних об’єктів, розташованих на схемі.

У вкладці Дошка заголовка (рис. 5.15, б) визначається текст, що вводиться в різні графи основного напису (кутового штампа). Загалом, існує два основних типи субтитрів: прийняті за замовчуванням та окремі субтитри. Діалогове вікно команди «Шаблон проекту» за замовчуванням визначає інформацію, введену в головний заголовок; при цьому основний напис розташовується в нижньому правому кутку нового аркуша схеми (якщо на вкладці основного блоку правильно введено повне ім'я Бібліотеки Бібліотека і це бібліотека CAPSYM.OLB, то її назва не потрібно надавати).

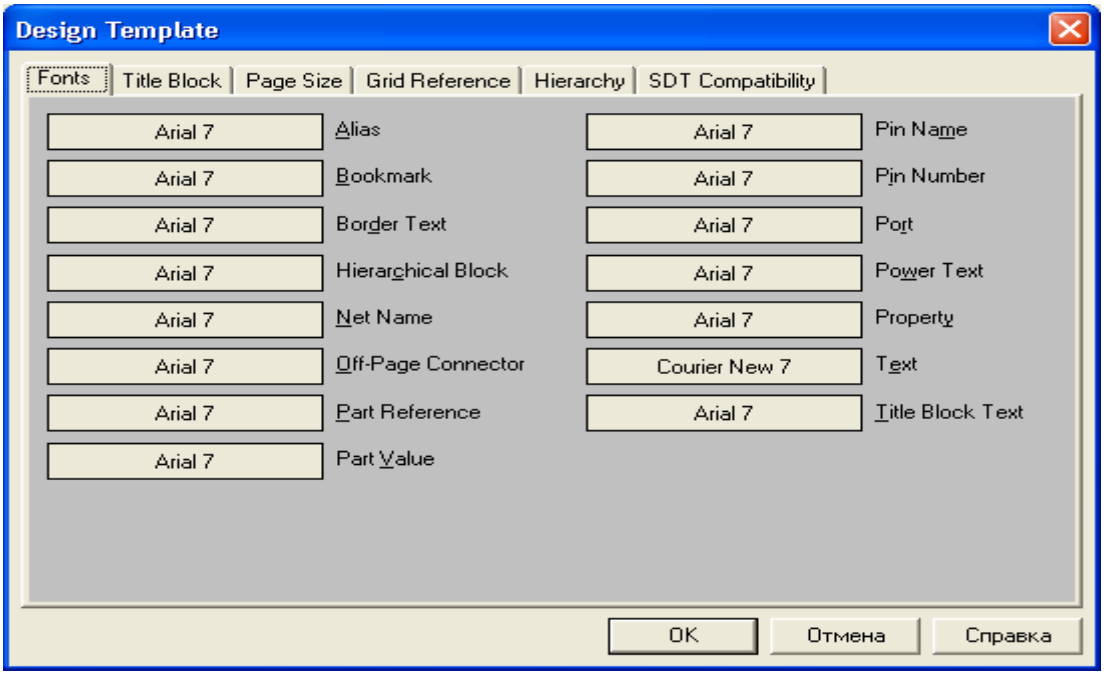


Рис . 5.15, а) Діалог вікна команда Параметри>Шаблон проекту

Назва графічного символу основного напису подається в рядку назви основної частини. Редагування тексту, введеного в основний текст кожної діаграми, виконується в редакторі діаграм за допомогою команди Правка>Властивості. Окремо основні тексти розміщуються на схемі за допомогою команди «Розмістити» > «Блок заголовка».

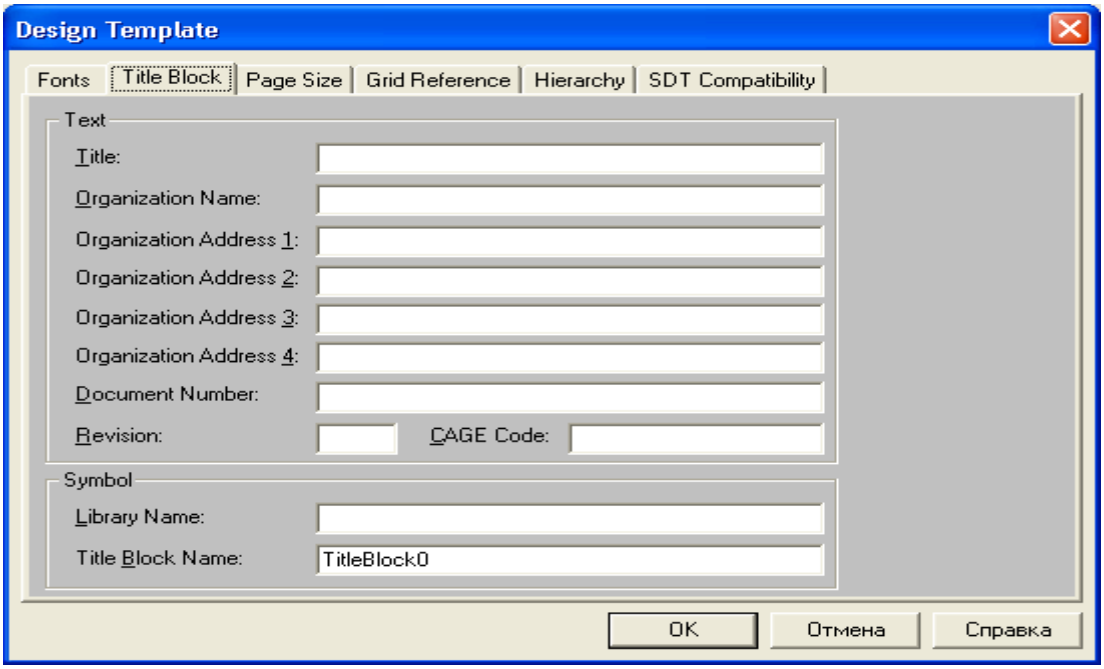


Рис. 5.15, б)

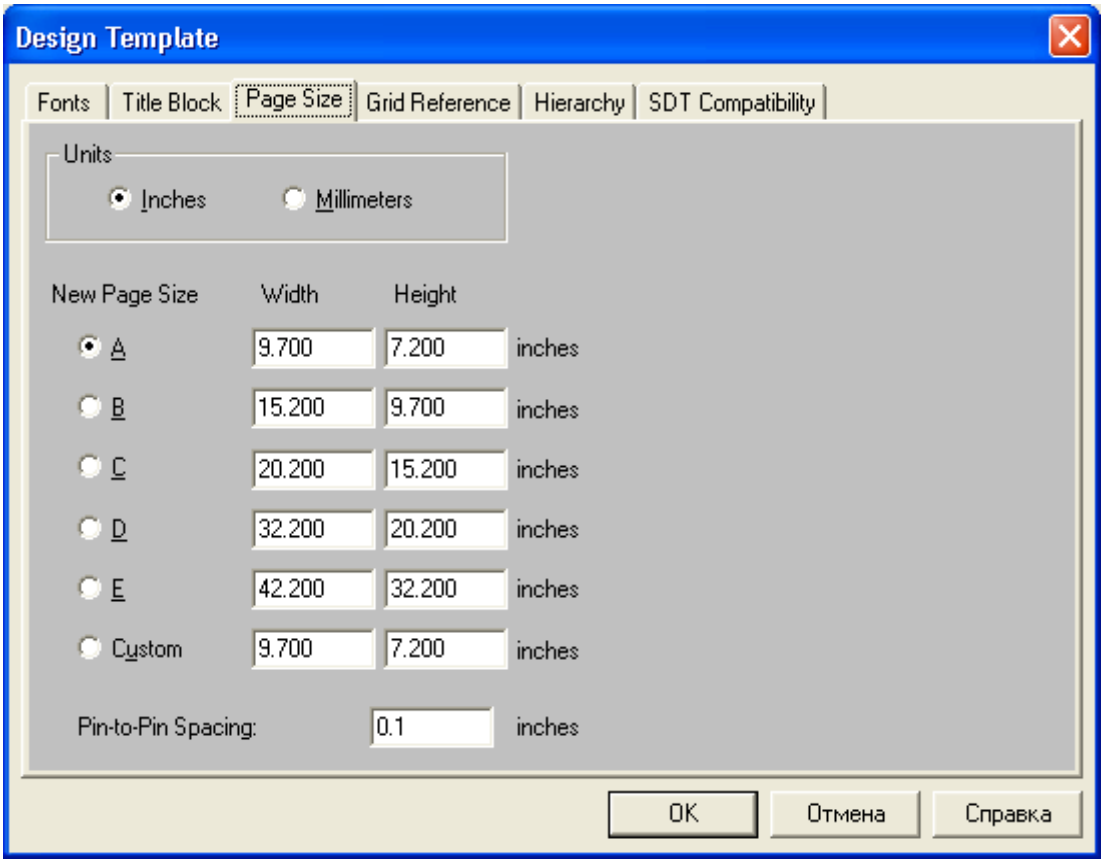


Рис. 5.15, дюйми)

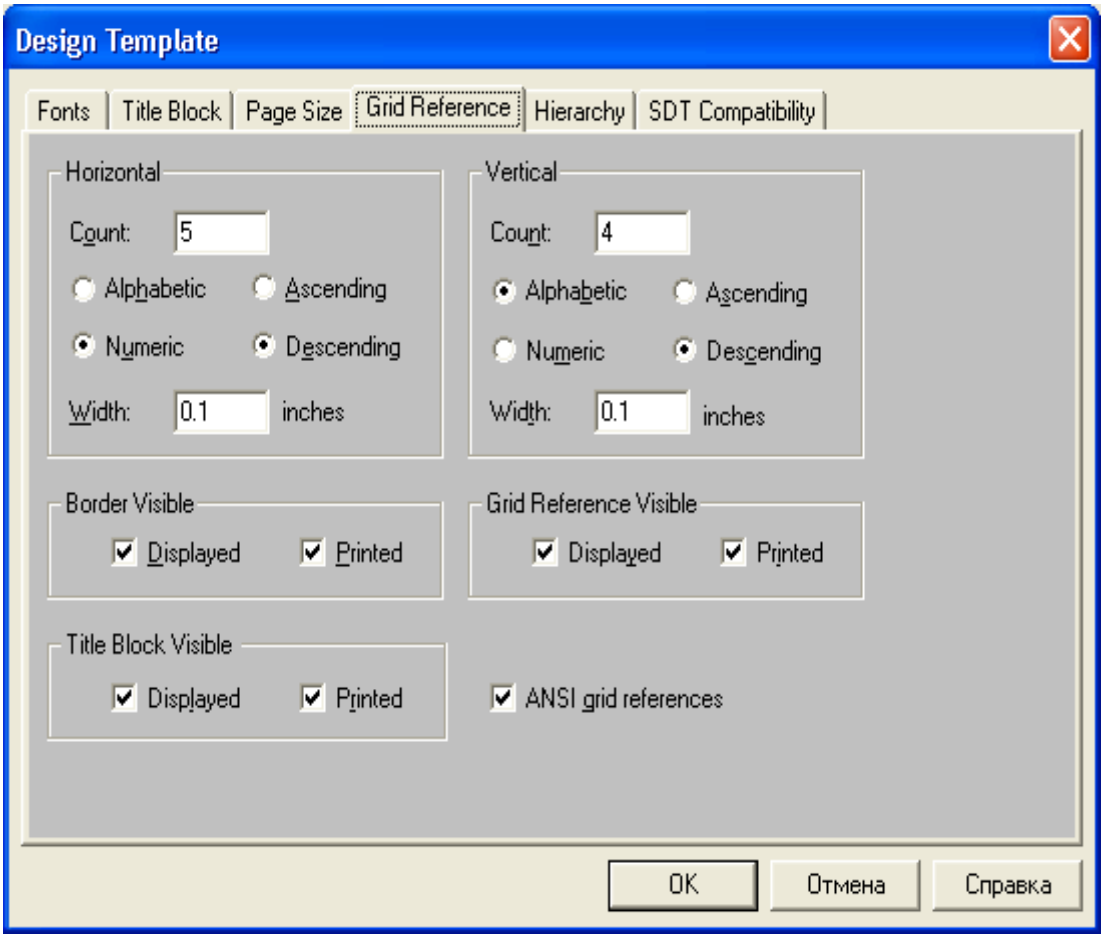


Рис. 5.15, d)

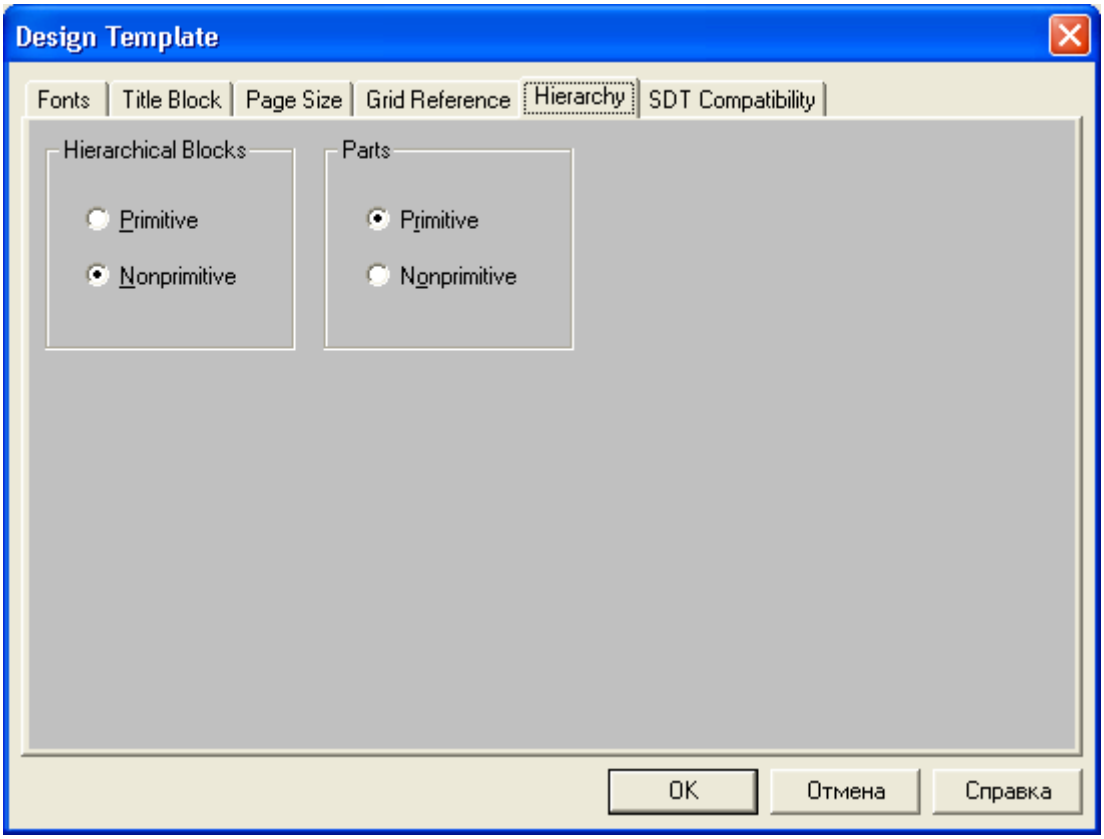


Рис. 5.15, д)

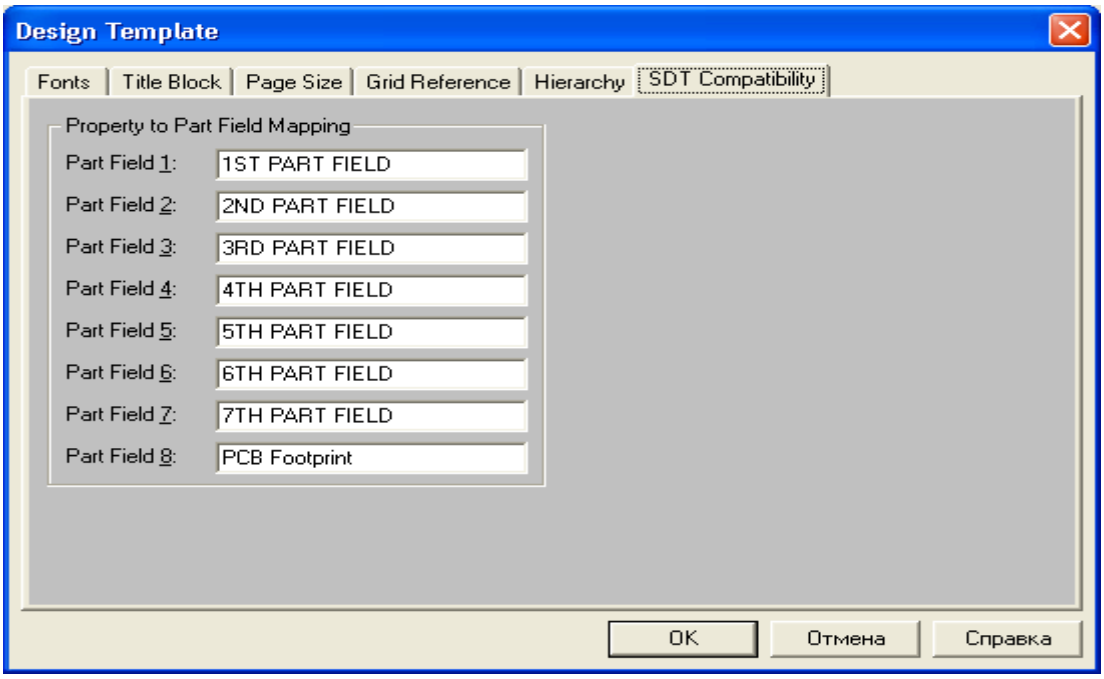


Рис. 5.15, д)

На вкладці Розмір сторінки (рис. 5.15, в) система одиниць вимірювання за замовчуванням (дюйми або міліметри) і розмір аркуша схеми A, B, C, D, E (в англійській системі), A4, A3, A2. , A1, A0 (в метричній системі) або Custom

(розміри визначає користувач). Стовпець Pin-to-Pin Spacing вказує мінімальну відстань між клемами елементів при їх розміщенні на схемі, водночас цей параметр визначає розмір кроку сітки (зверніть увагу, що для існуючого проекту або окремої схеми , крок сітки змінити не можна, він задається перед створенням проекту, що досить незручно). Зауважте, що коли ви змінюєте параметр Pin Spacing, розміри символів змінюються автоматично, коли ви розміщуєте їх на новій діаграмі.

На вкладці Grid Reference (рис. 5.15, г) задаються параметри рамки, розташованої навколо аркуша діаграми:

- Count - кількість діаграм на кадрі по горизонталі і вертикалі;
- Алфавітний - нумерація діаграм в алфавітному порядку;
- Числовий - нумерація діаграм в порядку номерів;
- Ascending - встановлення номерів граф кадрів у порядку зростання;
- За спаданням - встановлення номерів графа кадрів у порядку спадання;
- Ширина - горизонтальна і вертикальна ширина кадру;
- Border Visible - видимість меж сторінки на дисплеї (Displays) і під час друку (Printed);
- Grid Reference Visible - видимість рамки аркуша діаграми на дисплеї (Displayed) і після друку (Printed);
- Title Block Visible - видимість основного заголовка на дисплеї (Displayed) і після друку (Printed);
- Посилання на сітку ANSI - зображення рамки аркуша діаграми за стандартом ANSI.

На вкладці Ієрархія (рис. 5.15, д) задаються параметри, прийняті за замовчуванням при створенні нових ієрархічних блоків (Hierarchical Blocks) і компонентів (Parts):

- Примітивні - примітивні компоненти, які не мають ієрархічної структури;
- Непримітивні - компоненти з ієрархічною структурою.

Параметри символів компонентів OrCAD Capture із полями символів компонентів у форматі DOS-версії OrCAD Schematic Design Tools (SDT 386+), який використовується під час збереження проекту у форматі SDT, має значення .

Зміна параметрів поточного проекту здійснюється за допомогою команди Параметри>Властивості проекту, яка містить вкладки Шрифти, Ієрархія, Відповідність SDT, Різне (рис. 5.15, а, г, д); зміна параметрів поточної сторінки схеми здійснюється за допомогою команди Параметри> Властивості сторінки схеми має Розмір сторінки, посилання на сітку, різні вкладки .

Зауважте , що _ Увімкнено Це УВІМКНЕНО Ви можете скористатися вкладкою зміни розміру сторінки розмір діаграми І система одиниць І НІ Ти можеш Зміна Параметр контактної відстані , а УВІМКНЕНО Ви можете використовувати лише вкладку «Різне» . огляд інформації приблизно дизайн Або білизна діаграми .

Упорядкування символів компонентів Бібліотеки захоплення містять символи для компонентів, джерел живлення та заземлення. Вони розміщуються на діаграмі за допомогою команди Place > Part, яка також активується клацанням піктограми меню інструментів. У діалоговому вікні цієї команди (рис. 5.16, а) спочатку в списку Бібліотеки вибирається назва однієї або кількох бібліотек, вміст яких відображається на панелі Частина (для вибору кількох бібліотек натисніть і утримуйте клавіша CTRL).

Потім на панелі Part вибирається назва компонента, символ якого потрібно розмістити на схемі (якщо виділено кілька бібліотек, то після назви кожної з них ставиться символ /, а потім назва бібліотеки частини). У розділі «Графіка» вибрано звичайне або еквівалентне зображення логічних компонентів у стилі ДеМоргана (Перетворити). У розділі «Упаковка» вказується номер розділу компонента, після чого у вікні знизу з’являється зображення вибраного розділу компонента із зазначенням номерів роз’ємів його контактів (рядок «Деталі в упаковці» вказує загальну кількість штук частини пакета

Натискання кнопки «Додати бібліотеку» відкриває діалогове вікно, у якому можна додавати бібліотеки до списку «Бібліотеки», натискання кнопки «Видалити бібліотеку» видаляє вибрану бібліотеку зі списку. Кнопка «Пошук частини» використовується для пошуку певного компонента в бібліотеках зі списку «Бібліотеки». Після натискання кнопки ОК символ вибраного елемента переноситься на діаграму. Після переміщення курсору компонент переміщується в потрібне місце на діаграмі і фіксується натисканням лівої кнопки миші.

Потім на діаграмі можна розмістити іншу копію того самого символу. При натисканні правої кнопки миші відкривається спливаюче меню (рис. 5.16, б), яке дублює виклик команд головного меню повороту (Rotate), дзеркального відображення (Mirror), зміни масштабу зображення (Zoom), редагування параметрів компонента (Edit Properties).) та багато інших Розміщення символу виділеного елемента на діаграмі завершується після вибору в цьому меню команди Режим завершення або натискання клавіші Esc.

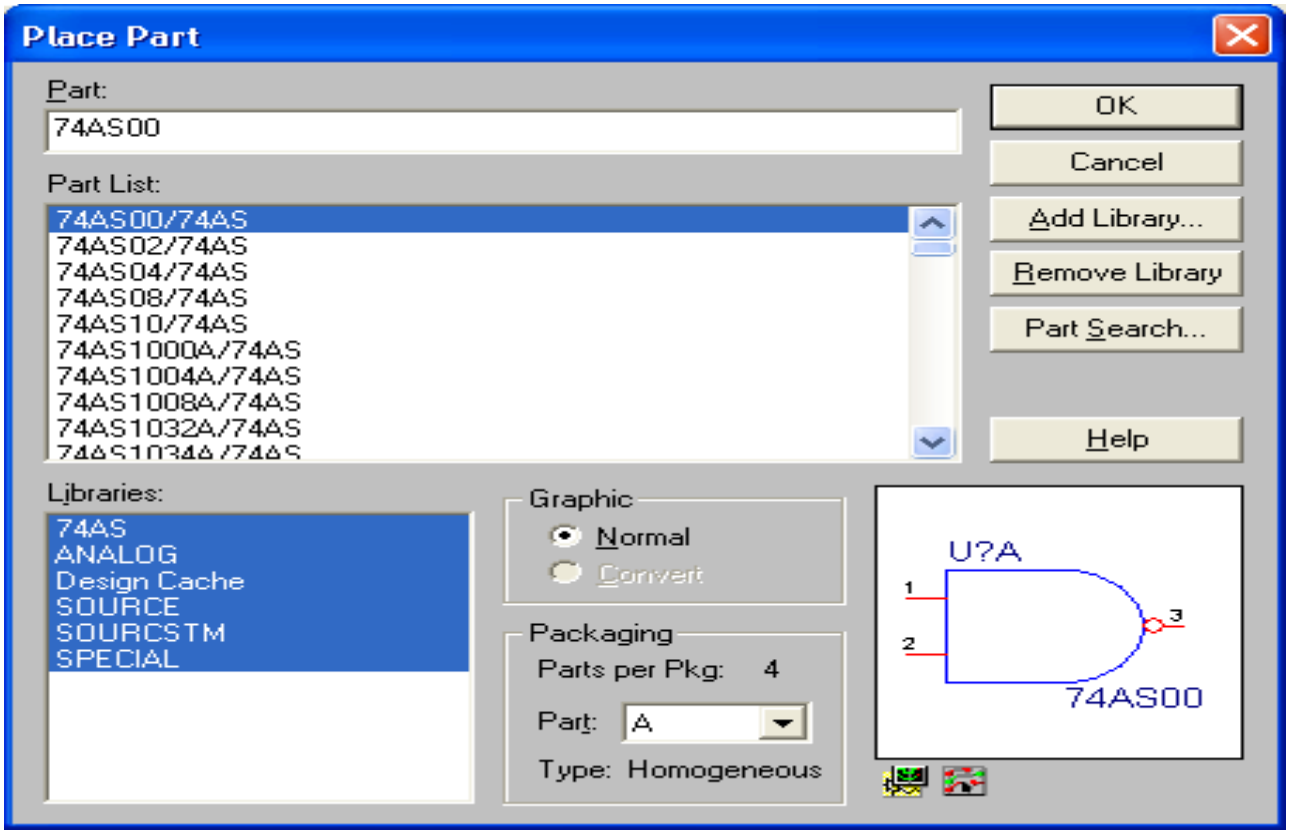


Рис. 5.16, а) Діалогове вікно Place > Part (a) і спливаюче меню (b), що викликається після вибору компонента натисканням правої кнопки миші

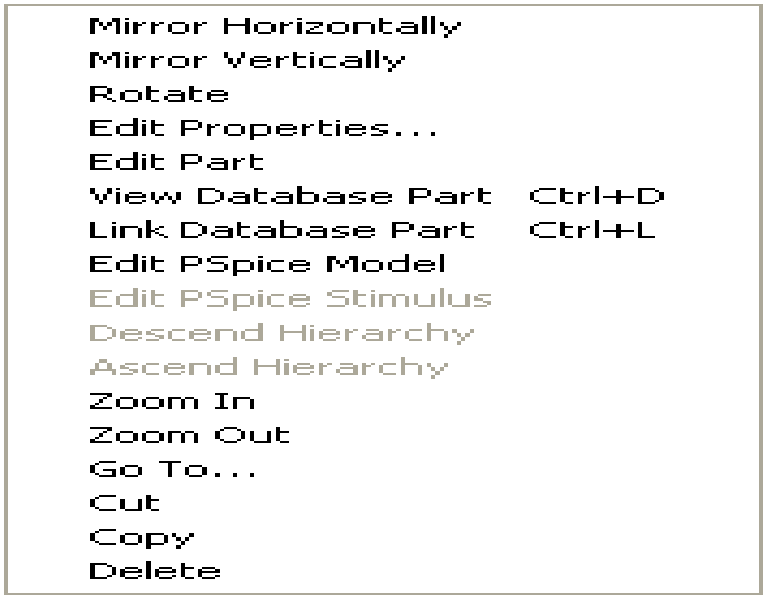


Рис. 5.16, b)

Якщо, не перериваючи режим розміщення символів компонентів на діаграмі, у спливаючому меню на рис. 5.16 b вибрати команду Редагувати властивості, то відобразиться діалогове вікно, у якому можна редагувати параметри поточного символу (рис. 5.17, а). Він має такі поля:

Частина - номінальне значення параметра простого елемента (опір, ємність тощо, які враховуються при моделюванні) або найменування складного елемента (не враховується програмою моделювання);

Позначення деталі - позначення місця розташування компонента. Він вставляється сюди вручну, якщо на вкладці Різне для команди Параметри > Параметри (рис. 2.20, д) не встановлено параметр Автоматично посилатися на розміщені частини - автоматичне призначення позиційних позначок для елементів, розміщених на схемі (подробиці нижче). На панелі RSV Footprint ви можете вибрати або налаштувати назву корпусу компонента. Вибір видимої панелі Power Pins вказує на необхідність відображення контактів заземлення та живлення на схемі. На панелі Примітив вибирається тип компонента: Так - елементарний компонент (примітив); Ні – компонент з ієрархічною структурою, За замовчуванням – встановлено за замовчуванням (відповідно до налаштування конфігурації на вкладці Ієрархія команди Параметри>Шаблон проекту (рис. 2.21, д)) . Панель Упаковка показує загальну кількість однакових

розділів компонента та назву (номер) поточного розділу (на жаль, номер розділу розташований на символі компонента і не може бути змінений на цій вкладці).

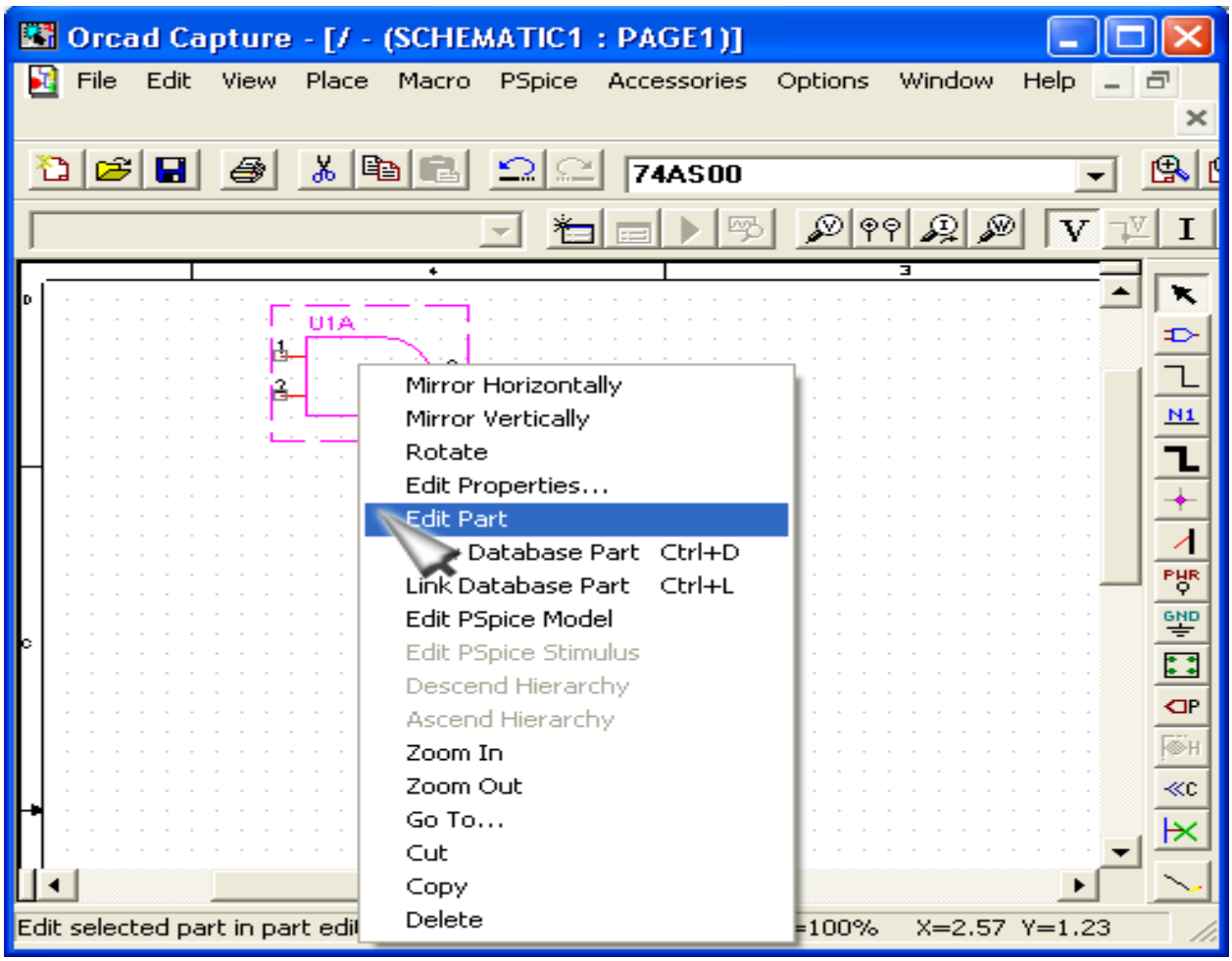


Рис. 5.17, а) Діалогові вікна для редагування параметрів символу елемента, розміщеного на схемі (а), параметрів редагування (б), їх видимості на схемі (в) і типу їх моделей (г)

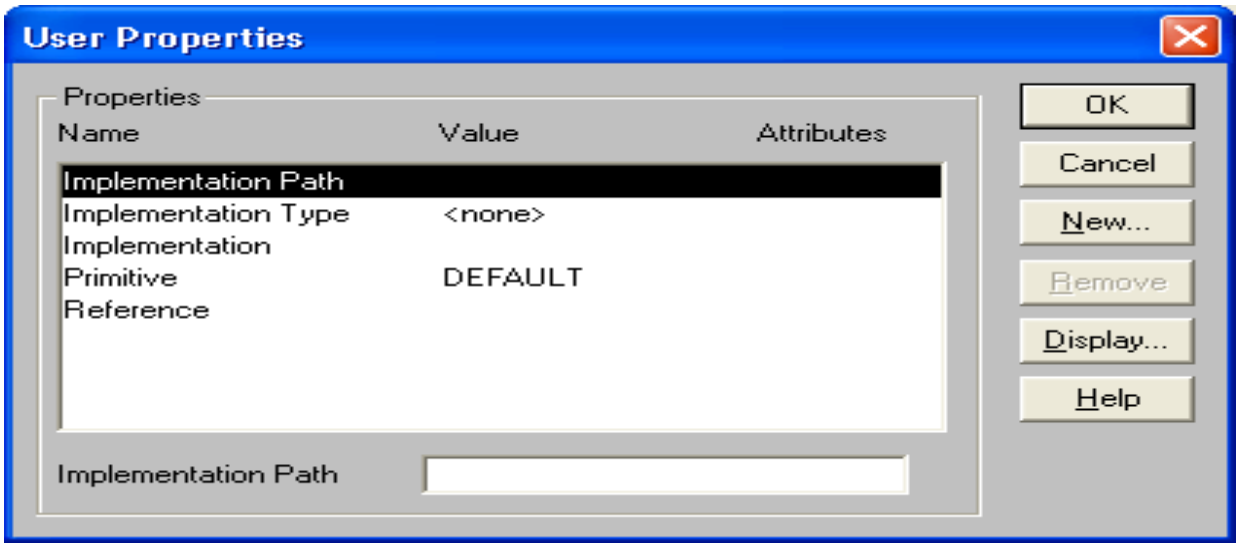


Рис. 5.17, б)

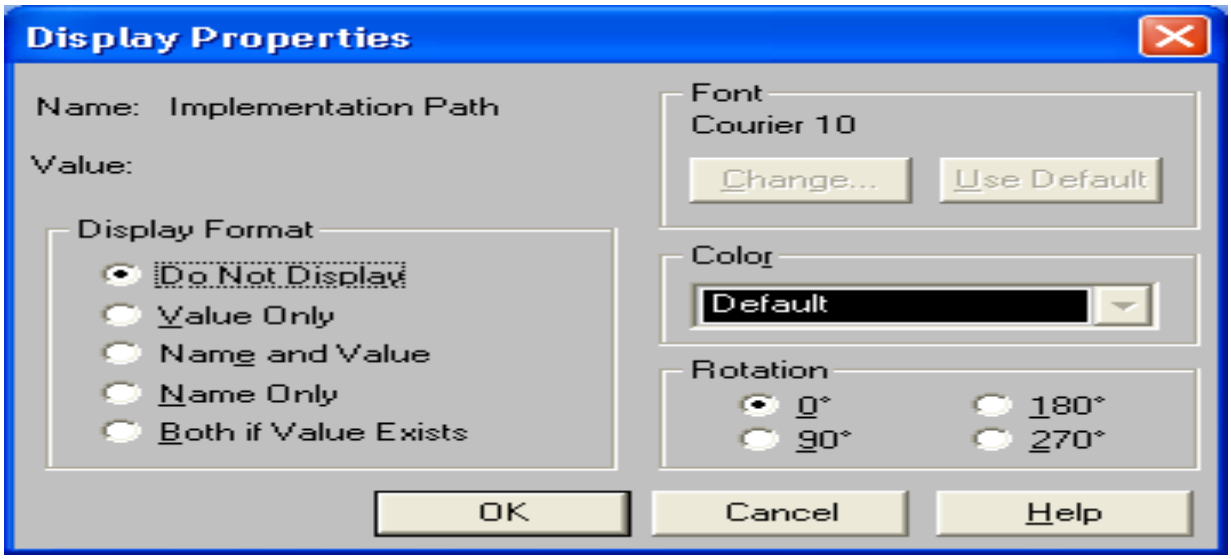


Рис. 5.17, дюйми)

Натискання панелі Властивості користувача відкриває діалогове вікно для перегляду та редагування параметрів компонента (рис. 5.17, б): у стовпці Ім’я вказується ім’я параметра, у стовпці Значення – його значення, а характеристики (атрибути) – його відображення. на діаграмі в колонці Атрибути (R - тільки для читання, V - видно на діаграмі, остання функція встановлюється на панелі Display, див. нижче). Після вибору параметра його назва відображається в нижній частині вікна, а його значення вводиться на сусідній панелі (після натискання клавіші Enter введене значення відображається в колонці Значення) – таким чином, серед ін. : у вигляді компонентів вводяться значення параметрів, необхідні для моделювання за допомогою PSpice (на рис. 5.17, б показано діалог параметра джерела гармонічної напруги VSIN); їх можна ввести або відредагувати пізніше за допомогою команди Edit>Properties (після створення діаграми за допомогою діалогових вікон, показаних на малюнку 5.18).

Натискання панелі «Дисплей» відкриває діалогове вікно, яке дозволяє встановити видимість вибраного параметра на діаграмі:

- Не відображати - нічого не відображати на схемі;

- Лише значення - відображає лише значення параметра

- Ім'я та значення - відображає як ім'я, так і значення параметра;

- Тільки назва - відображає тільки назву параметра;

- Обидва, якщо значення існує - відображає як назву, так і значення параметра, якщо його значення існує.

Натискання панелі «Приєднати реалізацію» відкриває діалогове вікно, яке дозволяє переглядати та редагувати тип об’єкта, прикріпленого до поточного компонента:

- Реалізація type - тип прикріпленого об'єкта, який приймає значення:
    - Модель PSpice - математична модель компонента для програми PSpice;
    - PSpice Stimulus - опис зовнішнього сигналу для програми PSpice;
    - Схематичний вигляд - схема об'єкта;
    - VHDL - опис компонентів у VHDL;
    - EDIF - список викликів у форматі EDIF;
    - Проект - діаграма проекту (для цього необхідно додатково задати висновки з ієрархічних блоків);
- Реалізація - назва прикріпленого об'єкта;
- Шлях і ім'я файлу, - повне ім'я файлу прикріпленого об'єкта.

Поля Graphics і Packaging цього вікна (рис. 5.17, а) такі ж, як і на рис. 5.16, о.

Розмістивши компоненти на діаграмі, ви можете переглянути параметри одного або кількох компонентів. Для цього виберіть компоненти, які вас цікавлять, і двічі клацніть або скористайтеся командою «Правка>Властивості», щоб відкрити електронну таблицю зі списком параметрів вибраних компонентів, приклади яких показано на рис. 5.18. Аналогічні таблиці містять параметри обраних ланцюгів схеми (Network Diagrams), висновки з компонента (Pins) і основний напис (Title Blocks). У цих таблицях можна редагувати лише ті параметри, які не мають атрибута R (тільки для читання), див. рис. 17.05 р.н Заштрихованим клітинкам присвоєні

параметри, значення яких не визначені; як тільки їх значення визначено, штрихування автоматично видаляється.

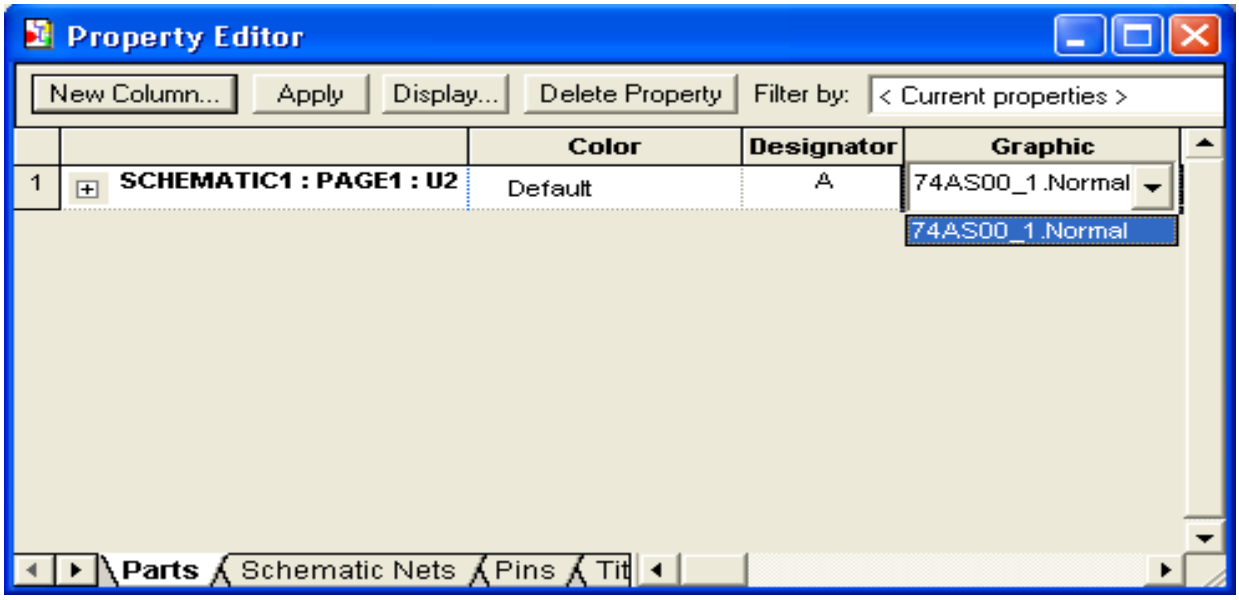


Рис. 5.18, а) Таблиці параметрів джерела гармонічного сигналу (а) та цифровий IC 7412 (б)

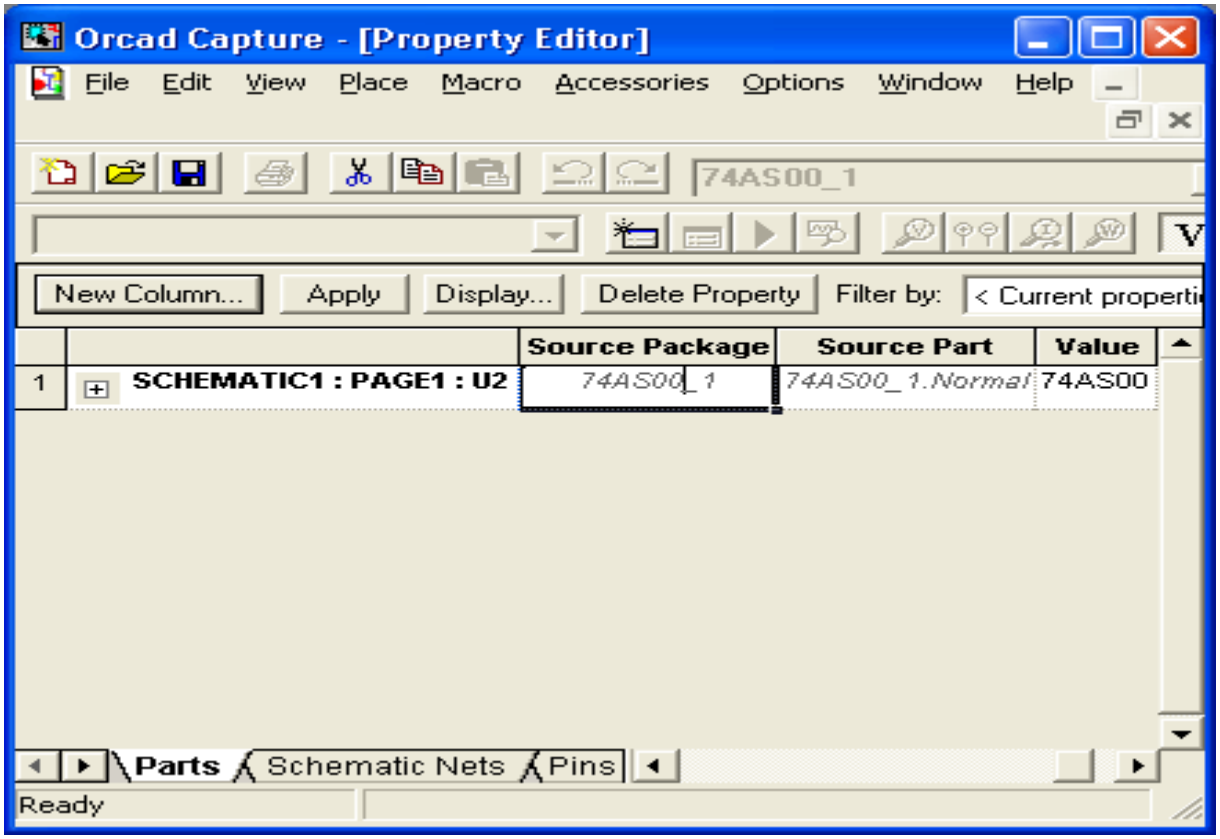


Рис. 5.18, б)

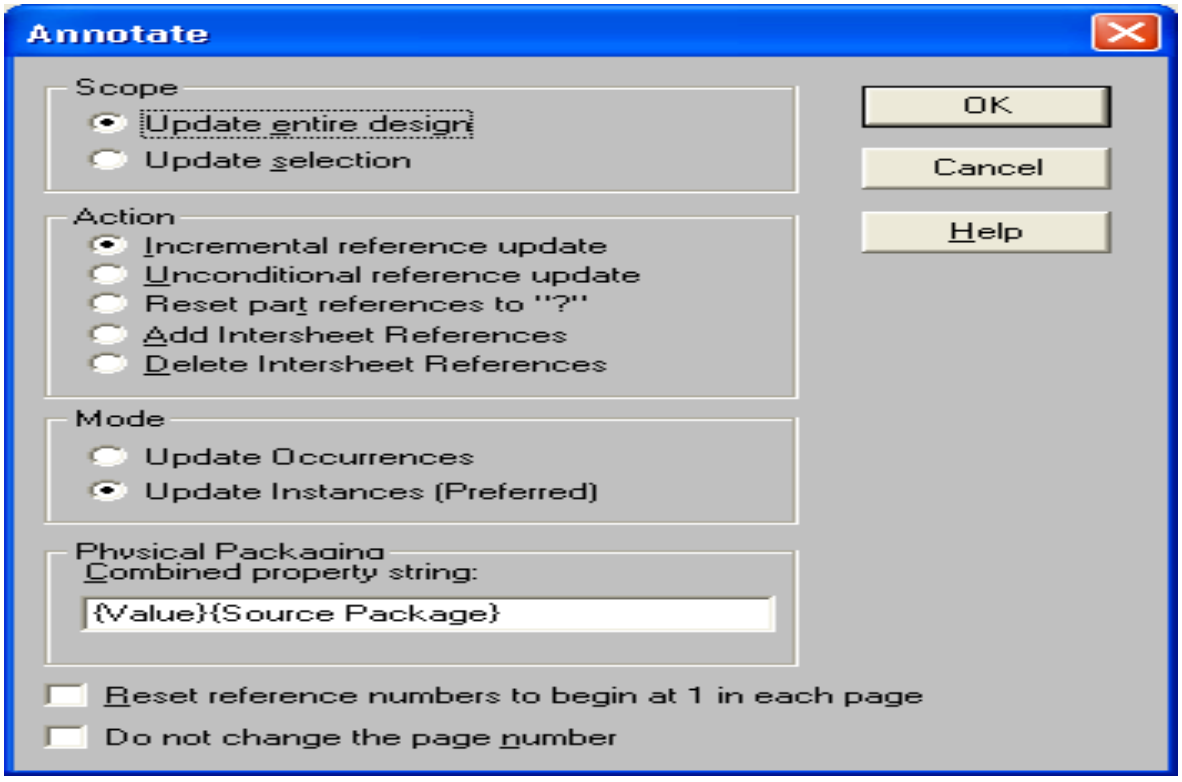
5.2. Розташування символів електричних компонентів і ланцюгів

Розташування позначок положення елементів. Позначення розташування компонентів (Part Reference) і номери секцій (Designator) вводяться вручну або при введенні компонентів (див. рис. 5.16, а і 5.17, а), або при редагуванні їх параметрів (див. рис. 5.4). В автоматичному режимі позначки розташування компонентів і упаковки секцій компонентів у корпусі розміщуються на схемі за допомогою команди «Інструменти» > «Анотувати менеджер проекту» або натисканням кнопки. Діалогове вікно для цієї команди показано на малюнку 2.5.19, яке містить такі поля:

- Обсяг (обласне завдання):
  - Оновити весь проект — оновити теги елементів і інформацію про упаковку всього проекту;
  - Оновити виділення - оновлення позиційних позначок та інформації про упаковку вибраної частини проекту;
- Дії:
- Авто › Назад Анотація
  - оновлення - оновлення маркування позиції та інформації про упаковку комплектуючої продукції, в якій замість номера вставляється знак питання «?», номери комплектуючих збільшуються на одиницю;
  - Безумовне довідкове оновлення - оновлення маркування позиції та інформації про упаковку всіх компонентів у вибраній галузі;
  - Скинути посилання на частину до "? - обмін кімнати інгредієнти на "?";
  - Додавати Проміжний лист Посилання - додавання посилань на інші сайти;

- Видалити Проміжний лист Посилання – видалення посилань на інші сайти;

- Режим (Режим оновлення):
  - оновлення Події - оновлення параметрів усіх окремих зразків компонентів;
  - Update Instances — оновлення параметрів компонента та всіх посилань на нього (цей режим кращий при роботі з проектами PSpice);
- Фізична упаковка (автоматична упаковка компонентів відповідно до конкретних властивостей, наприклад, у конкретному випадку упаковка конденсаторів, ємність яких знаходиться в заданих межах):
  - Зв'язана властивість рядок - рядок властивості ;
- Скинути номери посилань, щоб починати з 1 на кожній сторінці - починати з 1 нумерацію позиційних позначок для елементів одного типу на кожній сторінці;
  - Не міняйте номер сторінки - ні зрадити номер сторінки .



o

Рис. 5.19. Автоматичне позиціонування компонентів

За допомогою команди Annotate символи суміжних перерізів багатосекційних компонентів упаковуються в рамки (рис. 5.20, а), а позначки позицій компонентів розміщуються в напрямку зліва направо і зверху вниз (рис. 5.20, Б). Крім того, символи компонентів можна зіставляти з конкретними випадками, які відповідають ряду характеристик, указаних у рядку властивості Combined.

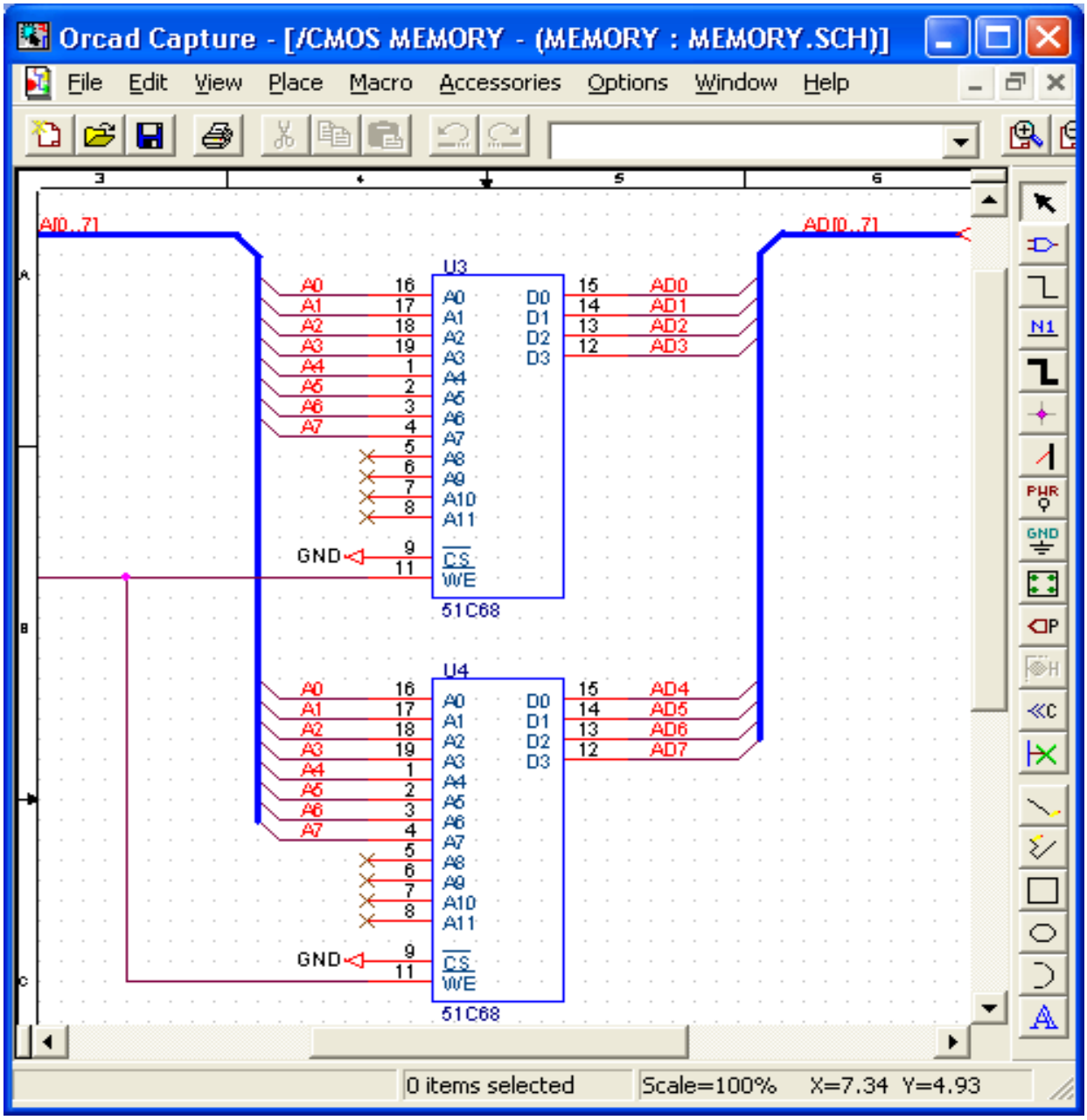


Рис. 5.20, а) Автоматичне пакування компонентів (а) і маркування позиції (b)

Розміщення символів «землі» та джерел живлення. Команди Place>Ground і Place>Power або натискання кнопок інструментів відкривають діалогові вікна, приклад яких показано на рис. 5.21, подібні до діалогу введення компонентів (див. рис. 5.16, а). Перелік символів «землі» та джерел живлення, що входять до стандартних бібліотек CAPSYM.OLB та SOURCE.OLB, наведено в таблиці. 2.3. Крім того, ці символи можна розмістити на діаграмі лише за допомогою команд PlacoGround і Place>Power. Обидві ці команди еквівалентні. При цьому символи блоків живлення мають видимі атрибути назви, які можна змінювати на панелі «Назва», наприклад, можна ввести назву + 5V (за замовчуванням назва, що відображається на схемі, збігається з назвою). Назви не мають основного значення, вони використовуються лише для того, щоб зробити схему більш наочною.

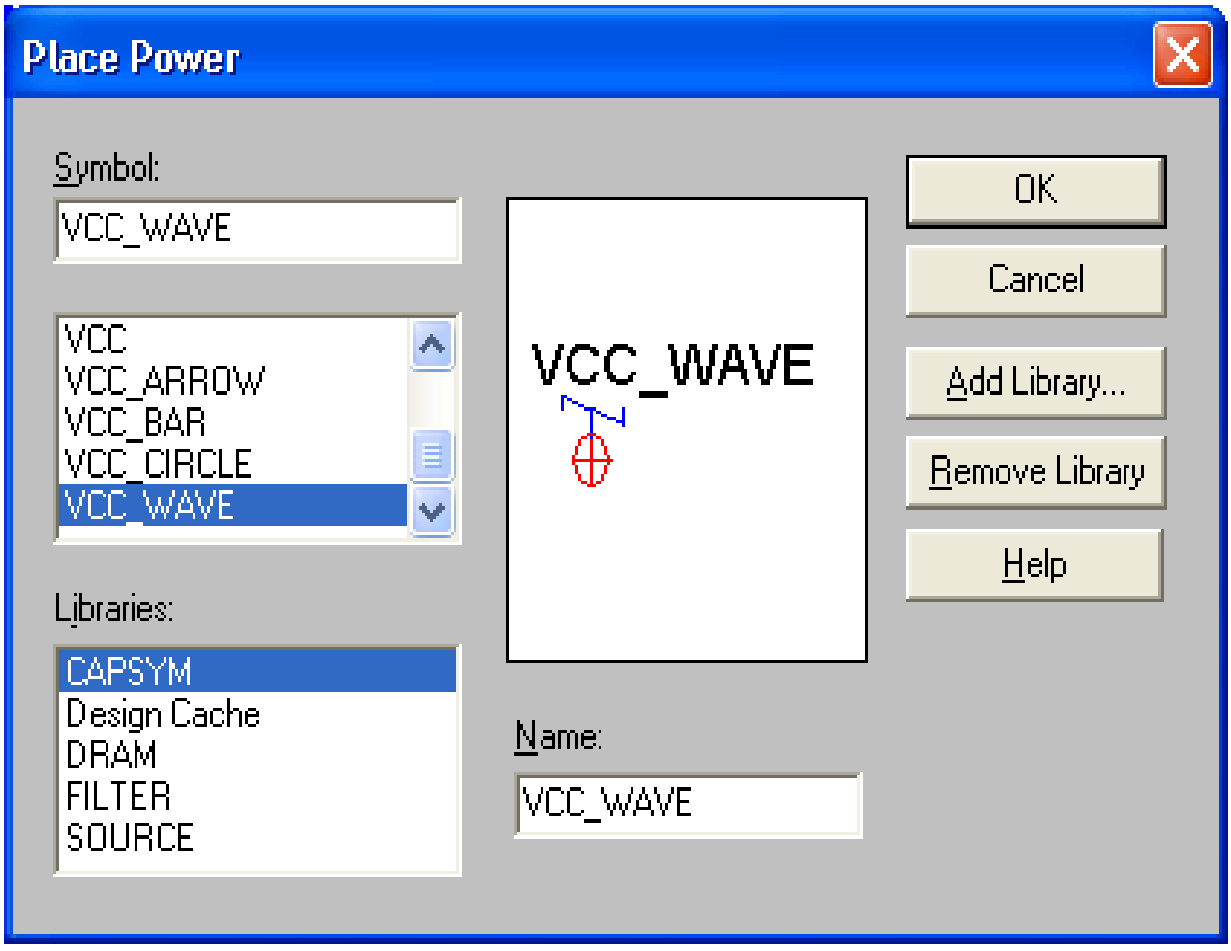


Рис. 5.21. Діалогове вікно для введення символів потужності

Таблиця 5.3. Список символів «землі», джерел живлення та постійних логічних сигналів.

Библиотека символов	Имя символа	Назначение
CAPSYM.OLB	GND_EARTH	"Земля"
	GND_FIELD SIGNAL	"Земля"
	GND_POWER	"Земля"
	GND_SIGNAL	"Земля"
	VCC_ARROW	Источник питания
	VCC_BAR	Источник питания
	VCC_CIRCLE	Источник питания
	VCC_WAVE	Источник питания
SOURCE.OLB (для PSpice)	0	Глобальная "земля"
	$D_HI	Логическая "1"
	$D_LO	Логический "0"

Символи «заземлення» та живлення підключаються до вузла, що називається 0, або до клем елементів, до яких вони повинні бути підключені (щоб побачити це, просто перегляньте список з’єднань *.net або завдання моделювання *.cir). Таким чином, при моделюванні за допомогою PSpice ви не можете підключити символи потужності, ви можете використовувати лише символ «землі», який має назву «О». Крім символу «земля», бібліотека SOURCE.OLB також містить символи для постійних логічних сигналів «1» і «О». Щоб створити власні символи землі та їжі, скористайтеся командою «Дизайн» > «Новий символ» у меню диспетчера команд.

Розміщення символів відсутності зв'язку. Команда «Розмістити > Не підключати» або клацання кнопки на панелі інструментів розміщує символи «Не підключати» (NC), які показано на схемі як символи «Х», прикріплені до клем компонентів. Виходи, позначені цими символами, не включаються до звітів про помилки та списків підключень. Символи NC не можна видалити, натиснувши клавішу [Delete]; щоб видалити їх, необхідно розмістити інший символ NC поверх символу NC.

Розташування символів з’єднувача сторінки. Команда «Розмістити» > «З’єднувач поза сторінкою» або клацання кнопки на панелі інструментів відкриває діалогове вікно, у якому можна малювати символи з’єднувача сторінки на схемі. Стандартна бібліотека CAPSYM.OLB містить два символи з’єднувача сторінки, L і R. На панелі «Ім’я» діалогового вікна ви вводите назви з’єднувачів сторінок, яким автоматично призначаються назви рядків, які до них додаються. Кола, які знаходяться на одній або різних сторонах кола і мають однакову назву, вважаються електрично з’єднаними.

Схема розміщення електричних кіл. Круглі дроти розміщуються за допомогою команди Place > Wire, натискаючи SHIFT + W або натискаючи кнопку на панелі інструментів. Початок входу в ланцюжок відзначається клацанням лівої кнопки миші, поле курсору змінює свою форму, набуваючи вигляду хрестика. Ланцюжок впорядковується рухами курсору. Кожен злом провідника виправляється лівою кнопкою миші. Таким чином можна робити ортогональні розриви ланцюга під кутами, кратними 90°. Вставлення кабелю під будь-яким кутом здійснюється утримуванням клавіші SHIFT. Вхід кола струму є повним, якщо його кінець збігається з виходом елемента або будь-якої точки іншого кола. Подвійне натискання лівої кнопки миші призведе до примусового замикання ланцюга, після чого можна намалювати інший провідник. Завершення режиму введення схеми здійснюється натисканням клавіші Esc або вибором лінії Кінець проводу у спливаючому меню, що відкривається клацанням правої кнопки миші.

Якщо ланцюги починаються або закінчуються в будь-якій точці на ділянці іншого дроту або на виході компонента, між ними встановлюється електричний зв’язок. Ознакою того, що ланцюг пов'язаний до кінця, є зміна його форми - зникнення квадрата на його кінці. Поперечні ділянки кабелів не з'єднуються між собою. Вони з'єднані двома способами:

- при прокладанні пересічних проводів зупиніться на точці з'єднання і двічі клацніть лівою кнопкою миші - в результаті з'єднання буде позначено спеціальною точкою (перетином);

- для з’єднання проводів, що перетинаються, встановіть курсор у точку перетину та виконайте команду Місце>Перетин, натисніть комбінацію клавіш SHIFT + J або кнопку на панелі інструментів; щоб скасувати електричне з’єднання, помістіть іншу таку точку поверх з’єднання.

Якщо під час розміщення елементів на схемі одна або кілька клем зіткнуться, між ними буде встановлено електричне з’єднання, а потім, розсуваючи елементи один від одного, дріт автоматично прокладеться.

Якщо під час переміщення елемента або частини кола відбувається замикання кількох ланцюгів, як показано на рис. Відображаються попередження та короткі замикання 5.22. Щоб скасувати це переміщення, натисніть «ОК», а потім виконайте «Правка» > «Скасувати переміщення». Переміщення ланцюгів без урахування їх електричних з'єднань здійснюється за допомогою клавіші ALT.

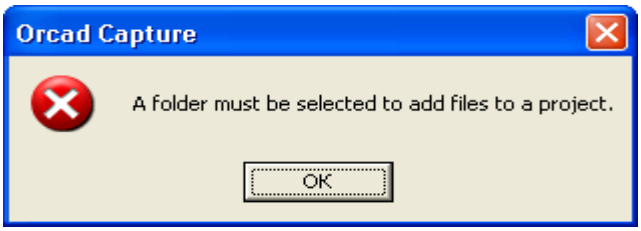


Рис. 5.22. Попередження про коротке замикання

Після розміщення рядків їм автоматично призначаються системні імена. наприклад N01049, який не можна змінити. Однак у списках викликів є т. зв псевдоніми ланцюга, які для вибраного ланцюга визначаються командою Місце>Мережевий псевдонім, також ініціюється натисканням комбінації SHIFT+N або клацанням кнопки на панелі інструментів. Кожен ланцюжок може мати кілька псевдонімів, з яких поточний псевдонім вибирається в таблиці Properties і використовується для складання списку підключень.

На схемі дроти зображені лініями стандартної ширини 0,2 мм в масштабі 1:1 (на жаль, цю ширину змінити не можна). Лінії однакової товщини представляють контурні лінії символів компонентів та їх води (див. рис. 2.4).

Розташування групових транспортних ліній (автобусів). Групові лінії зв'язку (автобуси) вводяться командою Місце > Шина (SHIFT + B) або клацанням кнопки на панелі інструментів. На схемі вони зображені більш широкими лініями, ніж дроти (рис. 5.23). Відгалуження окремих контурів, нахилені під кутом 45°, вводяться командою Location>Bus Entry (SHIFT+B) або натисканням кнопки так само, як окремі контури. При цьому сегменти ланцюжка зручно копіювати, перетягуючи їх, утримуючи натиснутою клавішу CTRL, зберігаючи вихідний об'єкт без змін. Імена (псевдоніми) шин і каналів, які вони включають, призначаються за допомогою команди Place > Alias Netto, а при розміщенні назв окремих каналів їх номери, запропоновані в діалоговому вікні команд, автоматично збільшуються на одиницю, наприклад ADDR1 , ADDR2, ADDR3, ADDR4. Назва шини, що складається з цих проводів, записується у форматі: ADDR [1..4]. На схемі шини позначені лініями стандартної ширини 0,8 мм (масштаб 1:1).

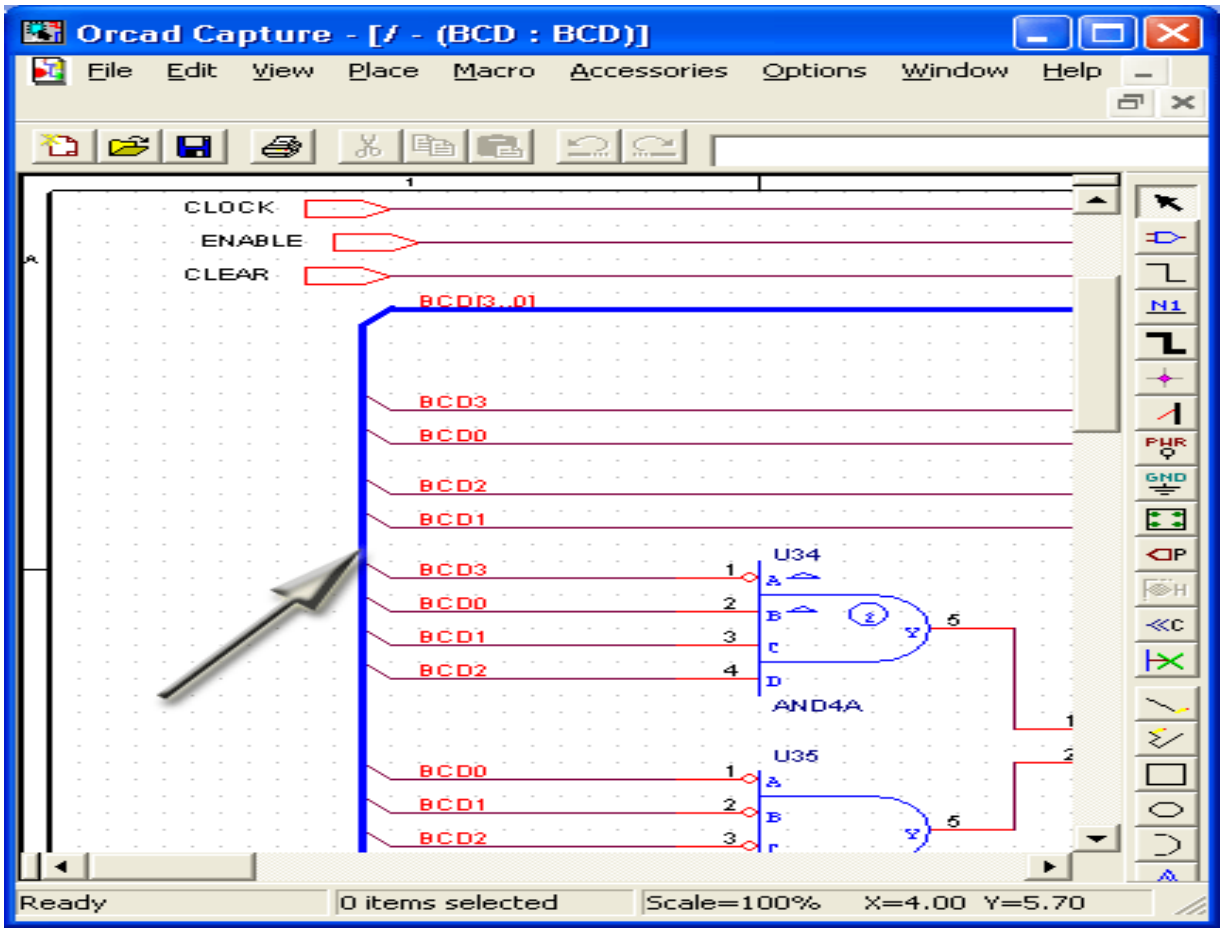


Рис. 5.23. Зображення шин

Будь-яка частина діаграми може бути оформлена у вигляді ієрархічного блоку, символом якого є прямокутник, а потім розміщена на діаграмі, що дозволяє зменшити її розмір. Інше використання ієрархічних блоків полягає в представленні повторюваних частин схем: різноманітних фільтрів, підсилювачів, випрямлячів, суматорів тощо. Ієрархічний блок розміщується на діаграмі за допомогою команди Place Hierarchical Block або клацанням кнопки на панелі інструментів. На рис. 5.24 і представлено діалогове вікно для цієї команди, яке складається з таких панелей:

- Посилання - позиційне позначення ієрархічного блоку;
- Тип реалізації - тип ієрархічного блоку, який приймає такі значення:
    - Схематичний вигляд - діаграма об'єкта,
    - VHDL - опис компонентів у VHDL,
    - EDIF - список викликів у форматі EDIF,
    - Проект - проект FPGA,
    - Модель PSpice - файл математичної моделі у форматі PSpice; і в цьому блоці потрібно вручну розмістити ієрархічні висновки,
    - PSpice Stimulus - файл зовнішнього впливу у форматі PSpice; крім того, в цьому блоці необхідно вручну розмістити ієрархічні висновки;
- Назва розгортання - ім'я ієрархічний блокувати ;
- Шлях і ім'я файлу - повне ім'я файлу, в якому знаходиться опис ієрархічного блоку (не вказується, якщо файл знаходиться в поточному каталозі проекту, в цьому випадку передбачається ім'я ієрархічного блоку - назва його папки);
- Примітивний - тип блоку: Так - елементарний блок; Ні – блок з ієрархічною структурою, За замовчуванням – встановлено за замовчуванням (відповідно до налаштування конфігурації на вкладці Ієрархія команди Параметри > Шаблон проекту;

- Властивості користувача - відкриває діалогове вікно, у якому можна ввести додаткові параметри блоку.

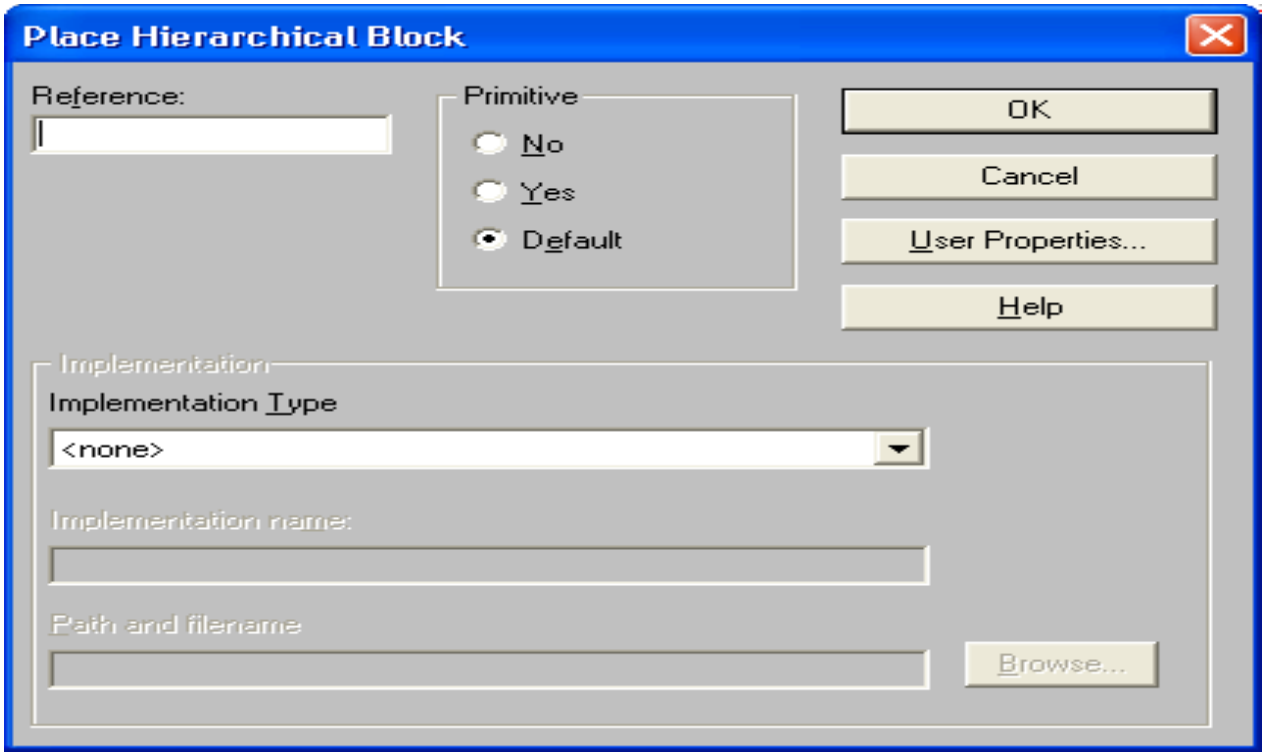


Рис. 5.24, а) Діалоги для створення ієрархічного блоку (а) і побудови його висновків (б)

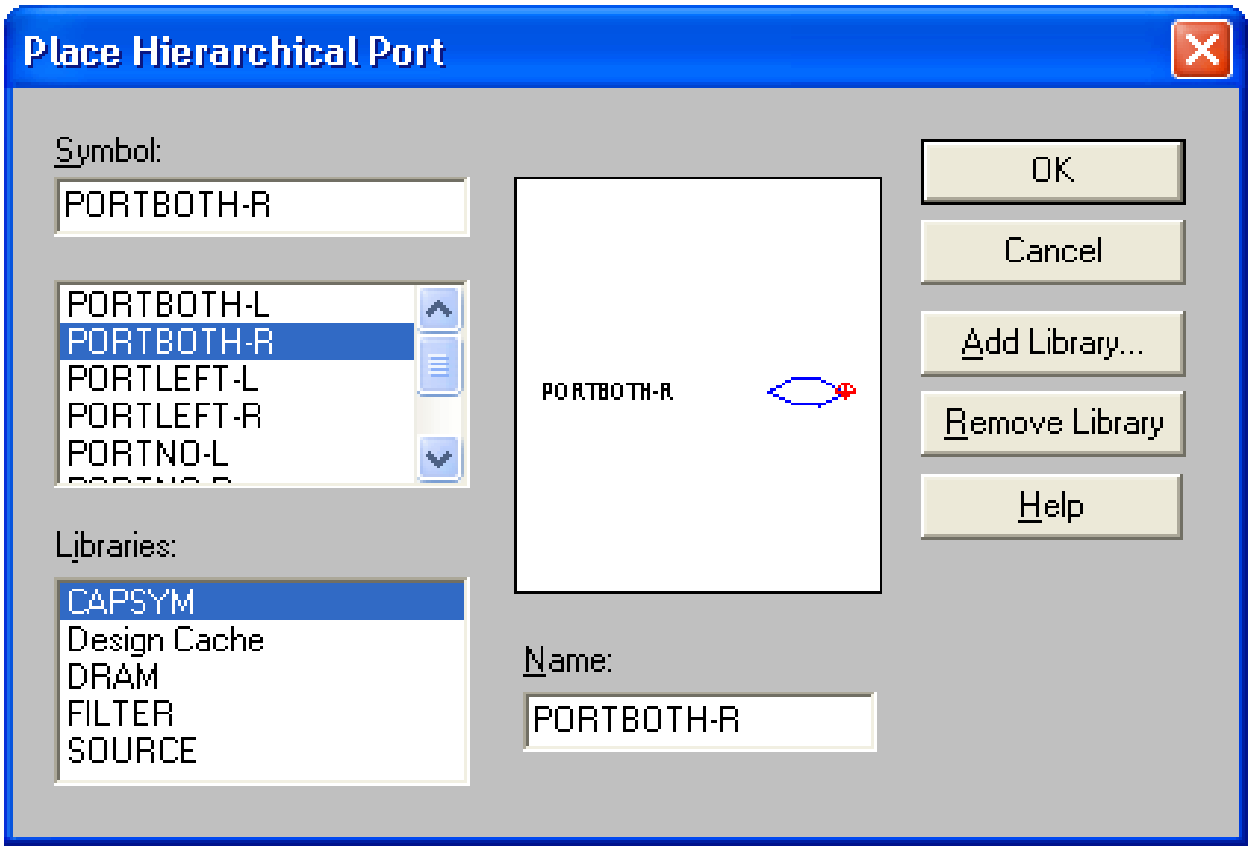


Рис. 5.24, б)

Після закриття цього вікна курсором на схемі малюються прямокутні контури символу ієрархічного блоку, а висновки з цього блоку вводяться за допомогою команди Place > Hierarchical Pin або натисканням кнопки на панелі інструментів . У діалоговому вікні команд (рис. 5.24, б) вказується:

- на панелі Ім'я - ім'я виведення;
- у колонці Тип - тип виведення:
  - 3 State – вихід цифрового елемента, який має три стани;
  - Bidirectional - двонаправлений цифровий компонентний вихід;
  - Під'їзд - під'їзд;
  - Open Collector - вихід цифрового елемента з відкритим колектором;
  - Open Emiter - вихід цифрового елемента з відкритим емітером;
  - Вывод – вихід;
  - Пасив - закінчення пасивного елемента;
  - Power - вихід для підключення до джерела живлення;
- на панелі Width вибирається тип схеми, підключеної до терміналу:
  - Скалярний - одноланцюговий;
  - Автобус є автобус.

Щоб не відкривати це вікно кожного разу при розміщенні нового виходу, можна розташувати всі виходи блоку одного типу, а потім редагувати таблицю всіх виходів (рис. 5.25), вибравши рядок Змінити властивості у спливаючому вікні. -up window - верхнє меню.

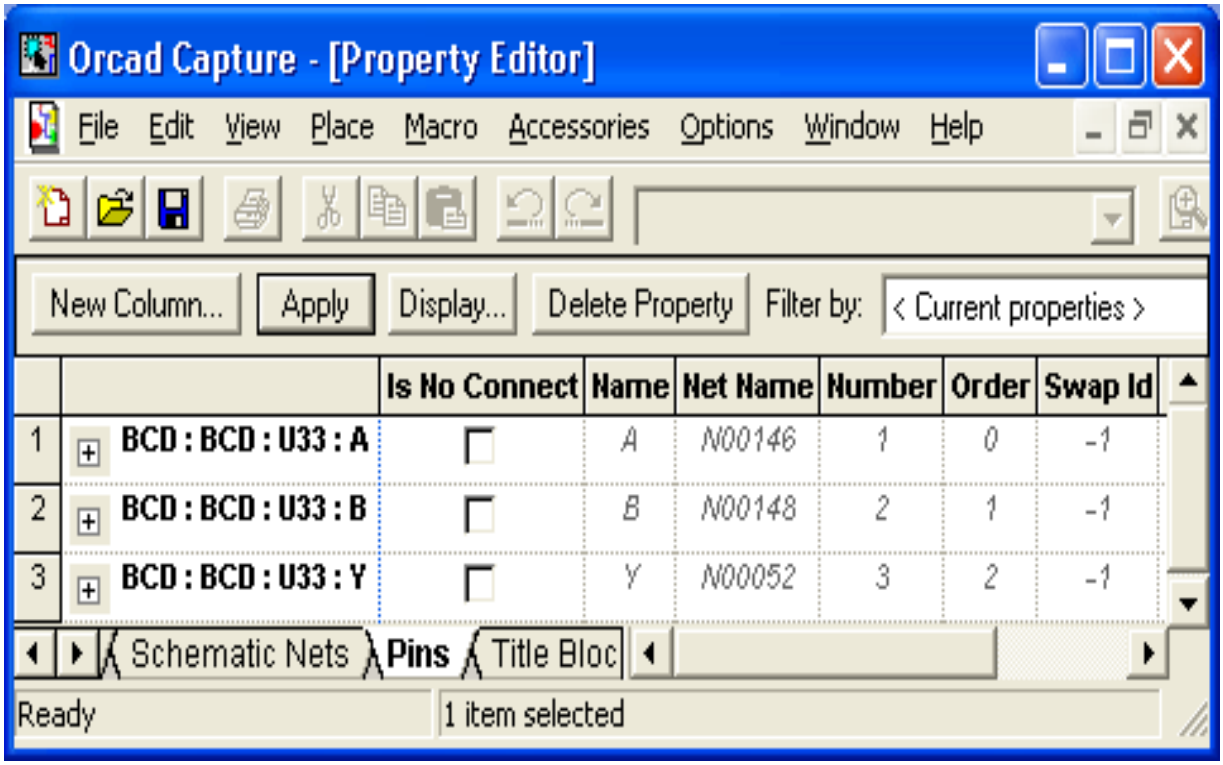


Рис. 5.25. Таблиця властивостей програми

Після виконання команди «Розмістити» > «Ієрархічний блок» автоматично створюється папка з заданим ім'ям. У цій папці слід розмістити опис ієрархічного блоку у вигляді діаграми його заміни (якщо вибрано тип блоку Схематичний вигляд) або текстовий опис у VHDL. На рис. На рис. 5.26 наведено приклад схематичного опису ієрархічного блоку. Ланцюгам, що з'єднуються з терміналами ієрархічного блоку, присвоюються імена, що відповідають іменам відповідних терміналів, або зовнішні порти схеми цього блоку вводяться за допомогою команди Розташування > Ієрархічний порт або натисканням кнопки на панелі інструментів (назви портів також мають збігатися з назвами відповідних терміналів, щоб забезпечити їх електричне підключення).

Графічна інформація вводиться на діаграму за допомогою команд «Розмістити» > «Лінія», «Лахана», «Прямокутник», «Еліпс» і «Дуга». Ця інформація носить допоміжний характер, тому з неї можна, наприклад, створювати електричні схеми. Стилі графіки за замовчуванням встановлюються на вкладці «Різне» у вікні команд «Параметри» >

«Параметри». Після малювання відрізків або дуг за допомогою команд Місце>Лінія, Місце>Лахана та Місце>Дуга їх можна відредагувати за допомогою діалогового вікна Редагування графіки (рис. 5.27, а). Вибирає:

- лінія Стиль і ширина - тип лінії (суцільна, штрихова та ін.) і її товщина (0,2, 0,8 і 2 мм);
- Колір - колір лінії.

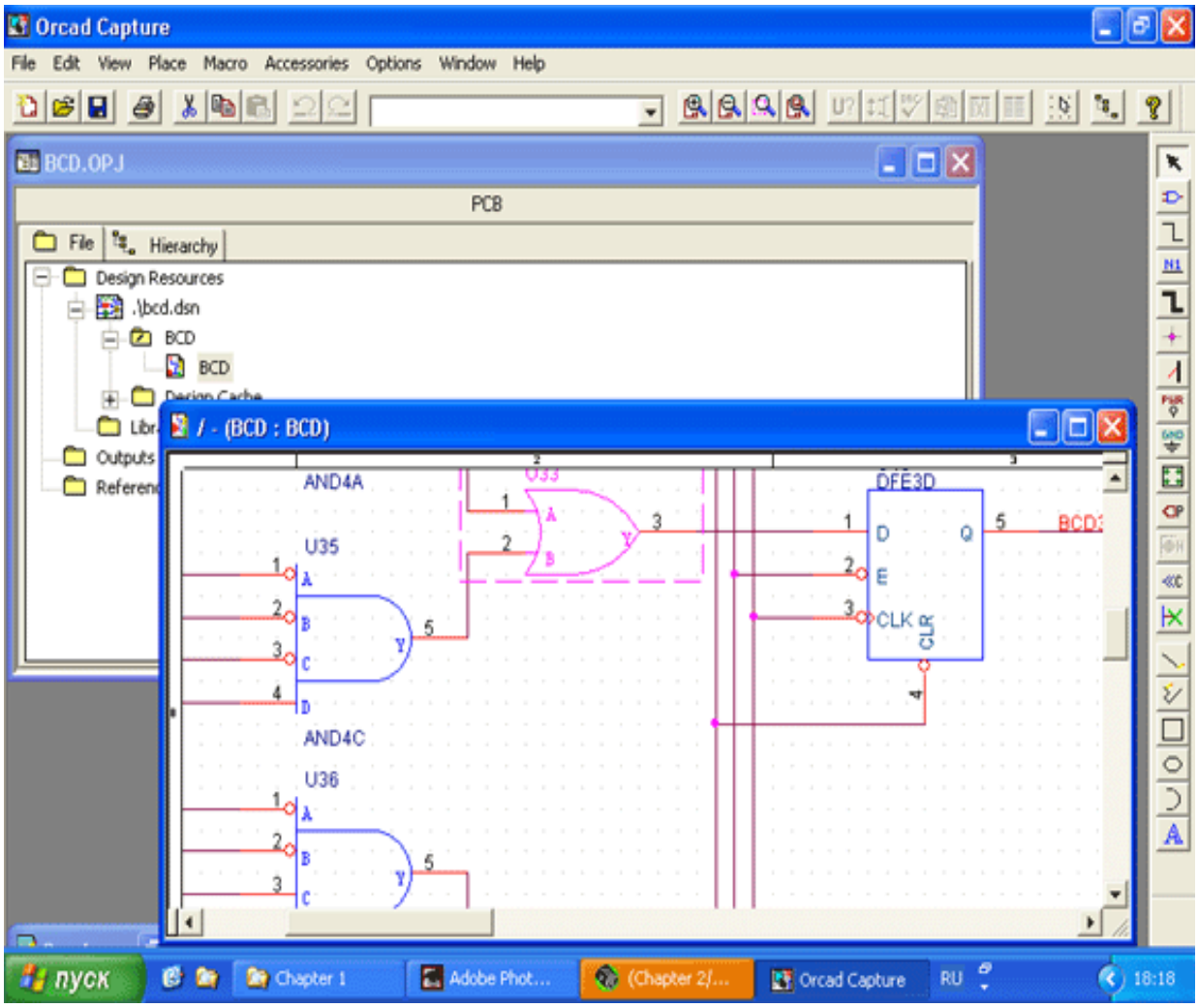


Рис. 5.26. Схематичний опис ієрархічних блоків

У діалогових вікнах Edit Filled Graphics, що відкриваються під час редагування замкнутих фігур, крім зазначених вище параметрів, вибирається тип заливки Fill Style (див. рис. 5.27, б).

OrCAD Illustrated Tutorial > Створення проекту в OrCAD Capture > Розміщення графічних об’єктів і тексту

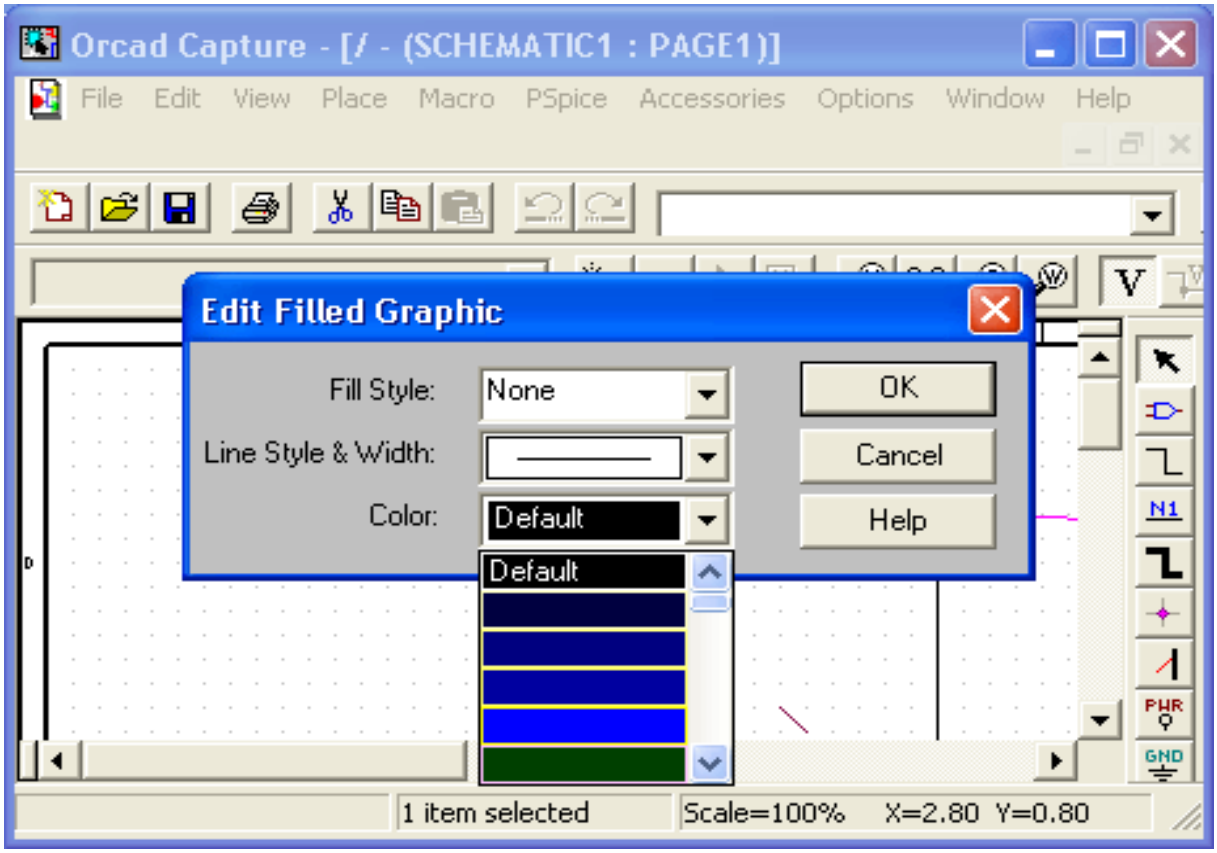


Рис. 5.27, б)

Щоб збільшити роздільну здатність під час малювання графіки, ви можете вимкнути режим курсора до сітки (параметр «Прив'язати до сітки» на вкладці «Відображення сітки» у вікні команд «Параметри» > «Параметри»), утримуючи крок переміщення курсора на 0,1 від кроку сітки.

Малюнки, попередньо введені в графічні файли *.bmp, розміщуються на діаграмі за допомогою команди «Розмістити» > «Зображення».

Текст розміщується на схемі за допомогою команди Place>Text або натисканням кнопки на панелі інструментів. Спочатку текст вводиться в діалоговому вікні, зображеному на рис. 5.28, а) (примусове перенесення тексту на новий рядок виконується натисканням клавіш CTRL + Enter), де вказано орієнтацію тексту та колір шрифту. Тип і розмір шрифту вибираються у вікні (рис. 5.28, б), яке відкривається після натискання панелі Змінити (доступні

кириличні шрифти, див. рис. 5.26). Встановлення шрифту здійснюється через «Параметри» > «Шаблон оформлення».

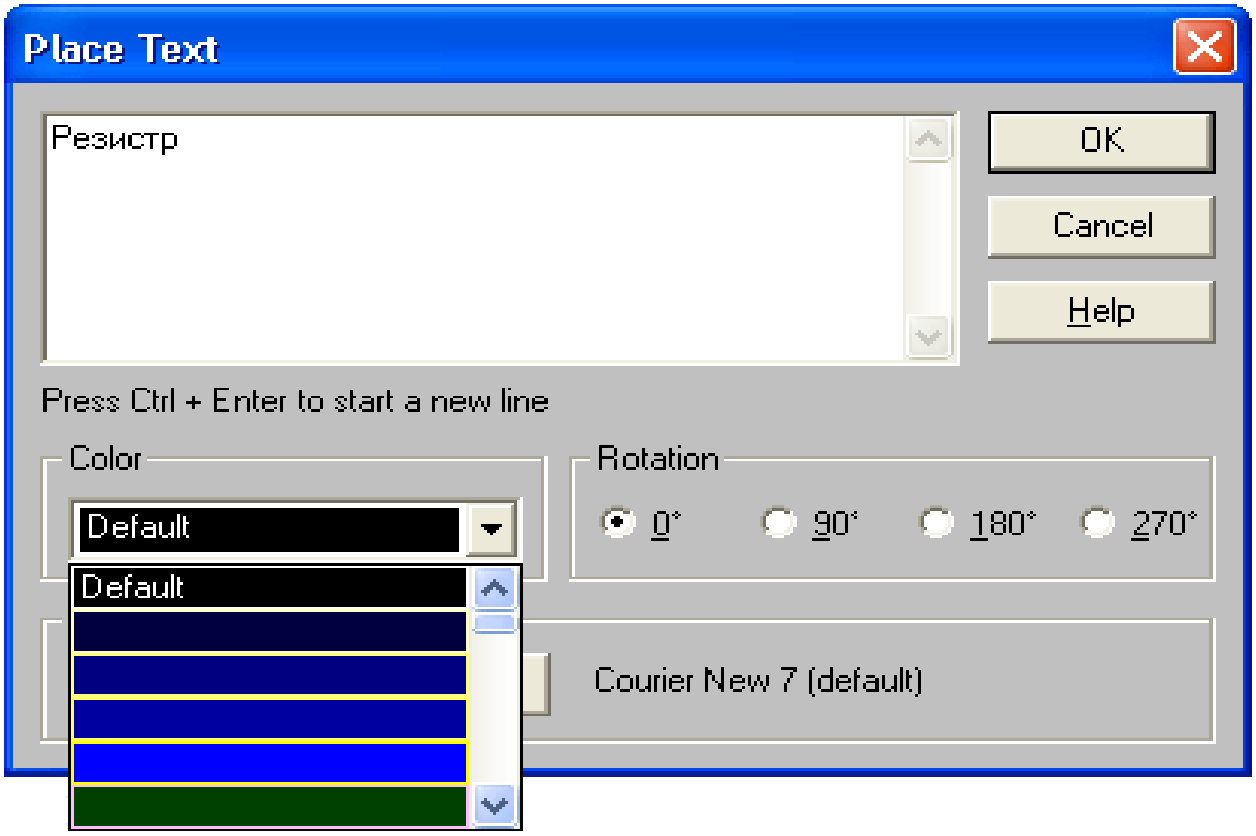


Рис. 5.28, а) Діалогові вікна для введення тексту (а) і вибору шрифту (б)

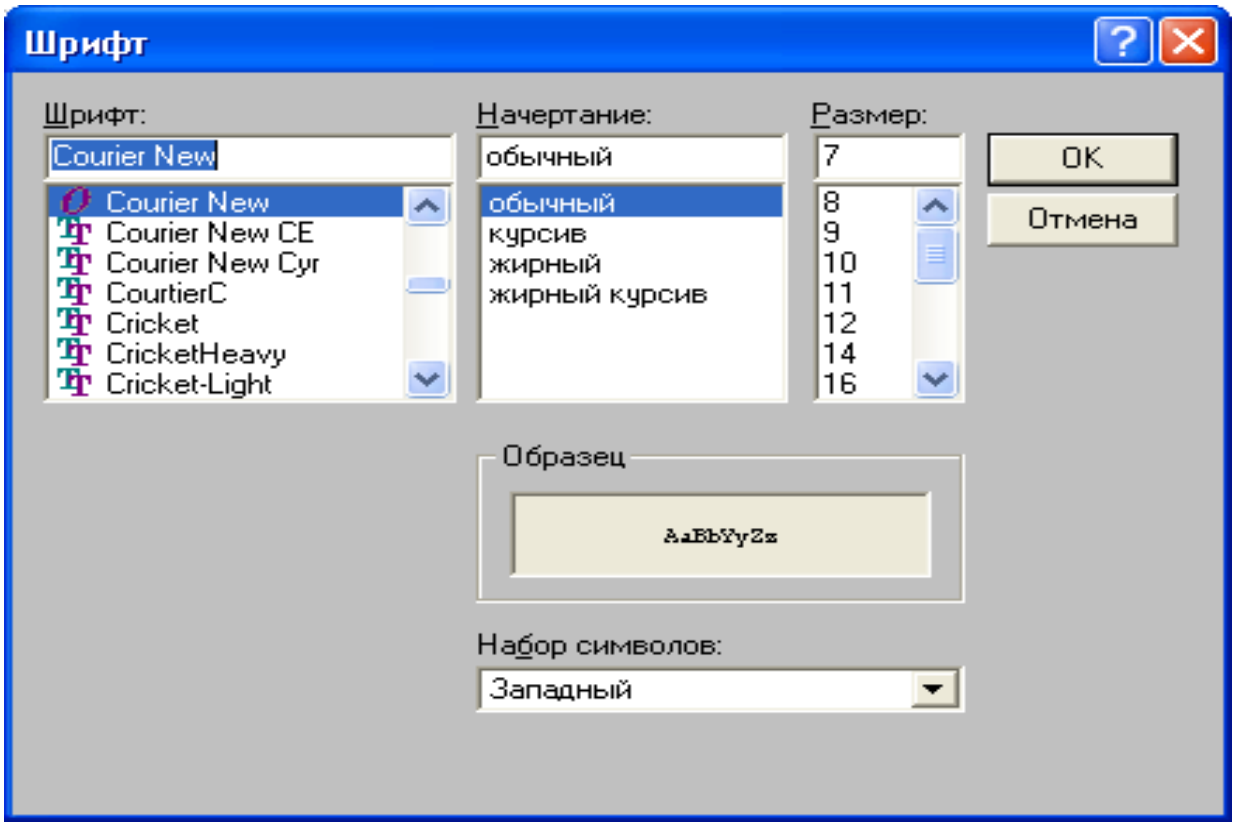


Рис. 5.28, б)

Імпорт тексту в діалогове вікно (рис. 5.28, а) з інших програм Windows здійснюється звичайним способом натисканням клавіш CTRL + V (фрагмент тексту попередньо необхідно помістити в Буфер обміну). Щоб експортувати виділений рядок тексту на схемі в буфер обміну для передачі в інші програми, натисніть CTRL + C або CTRL + X.

У редакторі схем можна зберегти порядок виконання окремих команд у файлі, який називається файлом макрокоманд, а потім виконати їх знову. Наприклад, в такому файлі можна зберегти інструкцію з компонування схеми і розміщення її назви. Створений командний файл макросу зберігається у тимчасовій пам’яті. Такий файл можна запустити лише під час поточного сеансу захоплення. Щоб надати цьому файлу унікальне ім’я, вкажіть його в діалоговому вікні «Налаштувати макрос». Оскільки файли макрокоманд можна використовувати лише на одній сторінці схеми, у них не можна зберігати такі команди:

- перехід на інший рівень ієрархії сходження та спуску ;
- редагування Компоненти Місце > Редагувати частину.

Координати об'єктів, збережених у файлі макросу, обчислюються відносно положення курсору під час виконання останньої команди перед записом у цей файл. Запис у файл макросу відбувається в такому порядку:

1. Клацання лівою кнопкою миші вибирає точку на сторінці діаграми, відносно якої будуть розраховані координати файлу макросу;
2. Після команди Макрос > Запис , яка дублюється натисканням клавіші F 7, відкривається рядок інструментів для запису файлів макросів, який містить три кнопки і показаний на рис. 5.29;
3. Натискання правої кнопки на панелі інструментів дозволяє писати команди у файл макросу та виконувати ці команди; натискання середньої кнопки призупиняє запис команди. Створення макрофайлу завершується натисканням лівої кнопки.

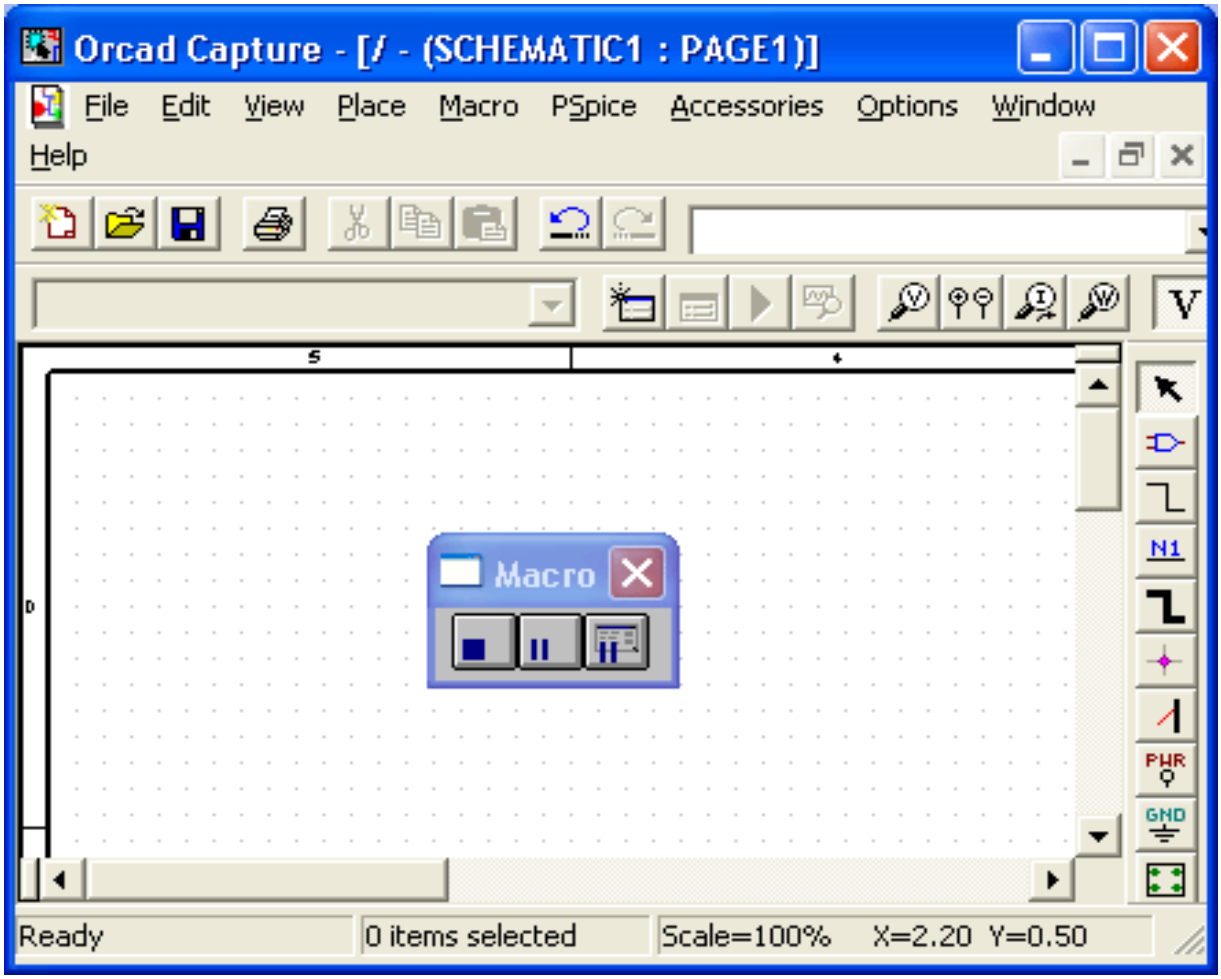


Рис. 5.29. Кнопки керування записом макрофайлів (зупинка, пауза, початок запису)

Щоб виконати останній файл макросу, клацнувши лівою кнопкою миші, виберіть на діаграмі точку, від якої буде зв’язано початок координат файлу макросу, і виконайте команду Musgo>Plau, повторивши натискання клавіші F8. Щоб назвати файл макросу та вибрати файл макросу для виконання за допомогою команди Macro>Configure, дубльованої натисканням F9, з’явиться діалогове вікно налаштування файлу макросу, показане на малюнку 5.30. Це вікно містить такі панелі:

- Macro Name - ім'я файлу макросу;
- Налаштовані макроси - відображення списку доступних макрофайлів, де вказано назву виконуваного макрофайлу;
- Закрити - закриває діалогове вікно;
- Запис - закриває діалогове вікно і починає запис команд у файл макросу;

- Play - виконання файлу макросу;
- Додати - додавання іншої назви до списку макрофайлів;
- Видалити - видаляє імена файлів макросів зі списку;
- Зберегти - зберегти зміни в поточному файлі макросу під тим же ім'ям;
- Зберегти як - зберігає зміни в поточному файлі макросу під новим ім'ям;
- Призначення клавіатури - призначення «гарячих» клавіш для виконання макросу, наприклад Ml, M2 або CTRL+1;
- Призначення меню - специфікація меню, пов'язана з поточним файлом макросу;
- Опис - опис файлу макросу.

Зміна вигляду поточної сторінки діаграми виконується зміною масштабу зображення за допомогою команди «Перегляд » > «Масштаб», переміщенням (зміна центру зображення без зміни масштабу) за допомогою команди « Перегляд » > «Масштаб» > «Вибрати» або переміщенням до певної точки за допомогою команду Перегляд > Перейти . Команди Zoom не потребують особливих пояснень. Давайте детальніше розглянемо команду переходу Go Do , який має три діалогові вікна, показані на малюнку 5.31.

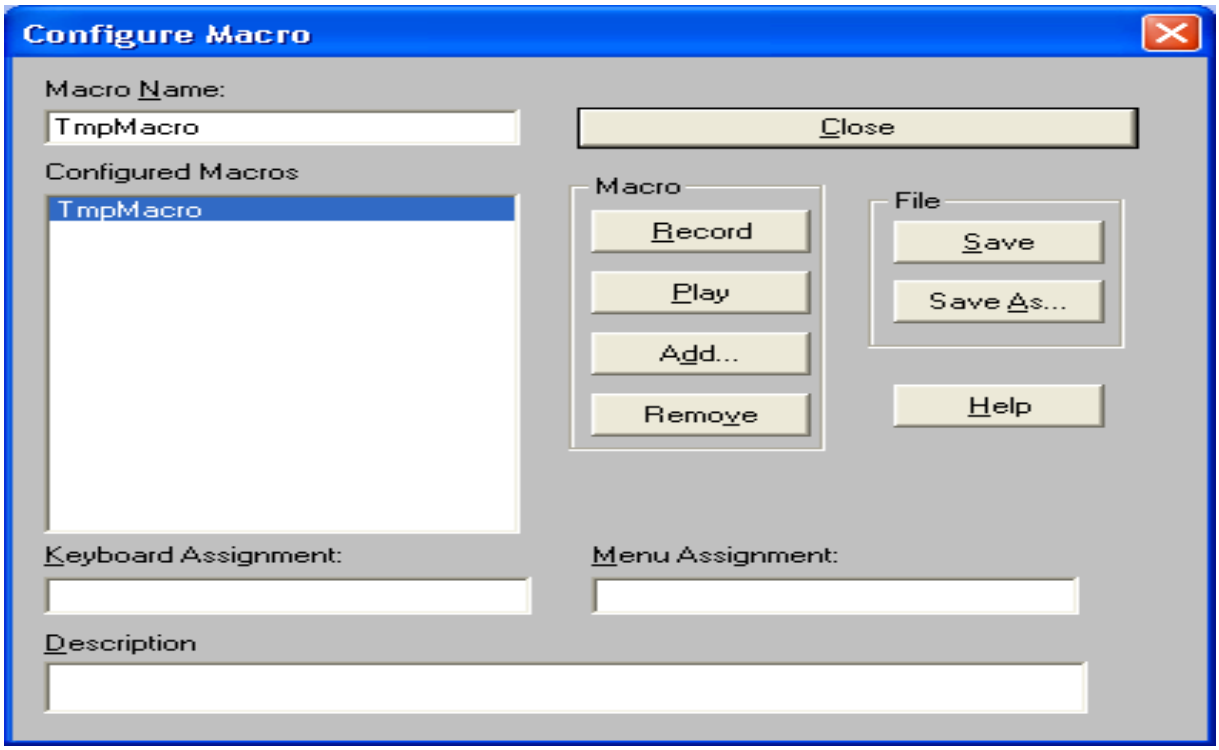


Рис. 5.30. Діалог конфігурації макрофайлу

Примітка

Приклади корисних файлів макросів знаходяться в каталозі \CAPTURE\MACROS.

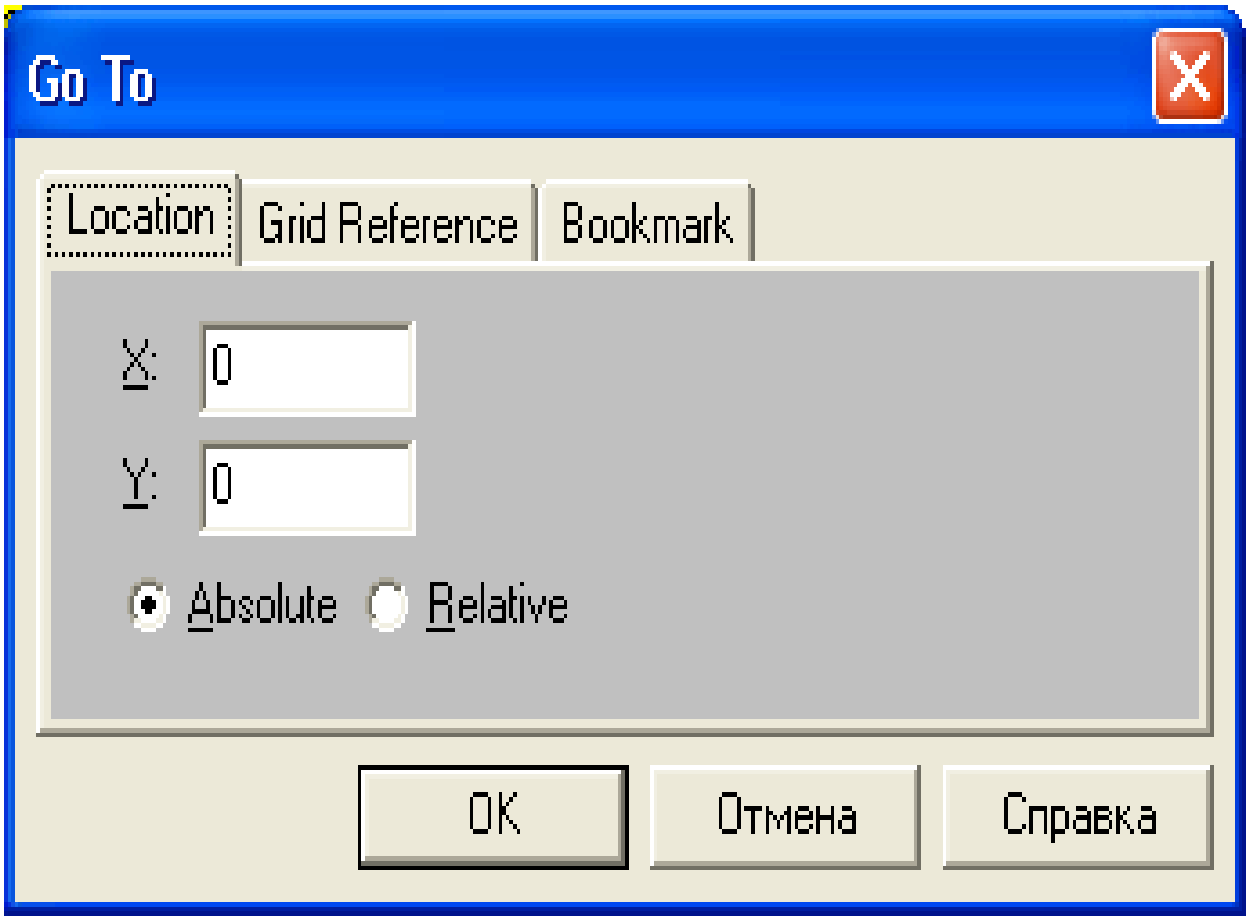


Рис. 5.31. Перейти до діалогових вікон команд

Переміщення до точки з певними координатами X, Y здійснюється за допомогою діалогового вікна Розташування (рис. 5.31). Переміщення до точки, координати якої обчислюються на рамці креслення, здійснюється за допомогою діалогового вікна Прив'язка сітки. Нарешті, переміщення до точки, попередньо визначеної командою «Помістити» > «Закладка», здійснюється за допомогою діалогового вікна «Закладка».

Крім того, команда Правка>Знайти (рис. 5.32) використовується для пошуку різноманітних об'єктів на діаграмі.

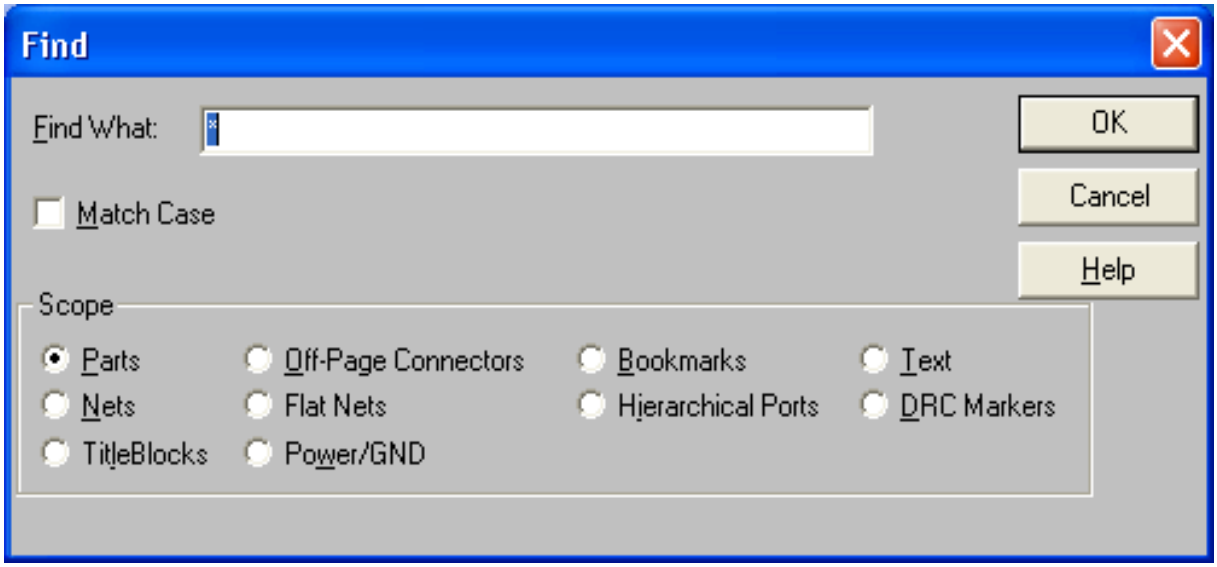


Рис. 5.32. Діалогове вікно для пошуку об'єктів на схемі

Бібліотеки символів (файли *.olb) Capture в OrCAD 9.2 містять понад 30 000 елементів. При створенні проекту слід заздалегідь продумати, які бібліотеки можна використовувати в конкретному випадку. В іншому випадку, наприклад, після створення схеми пристрою, неможливо розробити друковану плату через невідповідності в бібліотеках символів і корпусах компонентів. У таблиці 5.4 наведена коротка інформація про розміщення входить до стандартної поставки бібліотеки.

Каталог Capture\Library\PSpice містить бібліотеки символів *.olb і математичні моделі компонентів *.lib, які використовуються в моделюванні за допомогою PSpice, а також майже всі символи графічного редактора PSpice Schematics і відповідні їм математичні моделі.

Ряд бібліотек символів з каталогу \Capture\Library\PSpice не містить інформації про упаковку компонентів, з'єднання на їх корпусах або числові значення параметрів математичної моделі (ці значення вводяться безпосередньо в Схему):

- Abm . олб - функціональні блоки (суматор, множник, лінійна інерційна ланка, інтегратор, диференціатор, обмежувач та ін.);
- Аналоговий . olb – дискретні аналогові компоненти (R , R_var , C, L , E тощо);

- Breakout.olb - місця для символів напівпровідникових приладів та інших компонентів;
- Source.olb - джерела аналогових і цифрових сигналів, параметри яких подано в текстовому вигляді;
- Sourcestm.olb - джерела аналогових і цифрових сигналів, створені за допомогою програми Stimulus Editor;
- Special.olb - символи, які використовуються для призначення спеціальних директив моделювання (включаючи специфікацію параметра PARAM, мітку WATCH тощо).

Решта бібліотек відповідають компонентам конкретних типів, узгоджені з бібліотеками математичних моделей і компонентів твердих тіл (ці бібліотеки знаходяться в підкаталогах \Capture\Library\PSpice і \Capture\Library):

- Anlg_dev . _ _ olb - операційні підсилювачі та інші інтегральні схеми від Analog Пристрої
- Bipolar.olb - біполярні транзистори;
- CD400.olb - цифровий для MOSFET вентилів;
- Lin_tech.olb - операційні підсилювачі компанії лінійних технологій ;
- Siemens.olb - напівпровідникові прилади від Siemens
- 7400.olb, 74ac.olb тощо - цифровий TTL-IS;

Таблиця 5.4. Стандартні бібліотеки OrCAD.

Этап проектирования	Расширения имен файлов библиотек	Имя подкаталога расположения библиотек
Создание схем (OrCAD Capture)	**olb** – символы компонентов	\Capture\Library\PSpice
Создание схем (**PSpice Schematics**)	**sib** – символы компонентов **plb** – упаковочная информация	\PSpice\Library
Моделирование схем (**OrCAD PSpice**)	**lib** – математические модели компонентов	\Capture\Library\PSpice
Разработка печатных плат (**OrCAD Layout**)	**lib** – типовые корпуса (**Footprints**) компонентов	\Layout\Library (см. каталог библиотек, в файлах Liblist.txt, Laylib.txt)

Бібліотеки символів компонентів — це файли з розширенням .olb, які містять усю інформацію, необхідну для створення діаграм і передачі даних до інших програм OrCAD. Перш ніж перейти до опису принципів роботи з бібліотеками, пояснимо основні поняття, які використовуються в OrCAD.

Компонентами називаються фізично існуючі транзистори, конденсатори, інтегральні схеми (IC) тощо. Частина - умовне графічне зображення (символ) елемента на схемі. Деякі компоненти є багатосекційними і складаються з кількох секцій. Якщо всі секції такого компонента однакові, наприклад, цифрова інтегральна схема 4NI-I, його називають однорідним, інакше – гетерогенним. Інформацію про упаковку елемента, яка включає кількість секцій елемента, кількість висновків з окремих секцій, наявність логічно еквівалентних секцій і висновків (допускається їх перекомпонування при автоматичній маршрутизації з'єднань ПП), називають Пакетом. . OrCAD Capture припускає, що термін Part означає як символ для окремої частини компонента, так і символ для всього компонента в цілому. Бібліотеки символів компонентів є окремими файлами з розширенням olb.

Графічна проекція фізичного тіла компонента на друкованій платі називається «відбитком». Бібліотеки відбитків тіла компонентів є окремими файлами з розширенням lib.

Файли бібліотеки символів відкриваються в менеджері проектів за допомогою Файл>Відкрити>Бібліотека. Після натискання значка «+» у рядку з назвою бібліотеки відобразиться її каталог, як показано на рис. 5.33. Вибравши окремі компоненти в цьому каталозі, їх можна видалити та перемістити до інших бібліотек, як зазвичай; Щоб перемістити компонент з однієї бібліотеки в іншу, одночасно відкрийте каталоги обох бібліотек у менеджері проектів і перетягніть піктограму символу з однієї бібліотеки в іншу, розмістивши її відповідно до її назви.

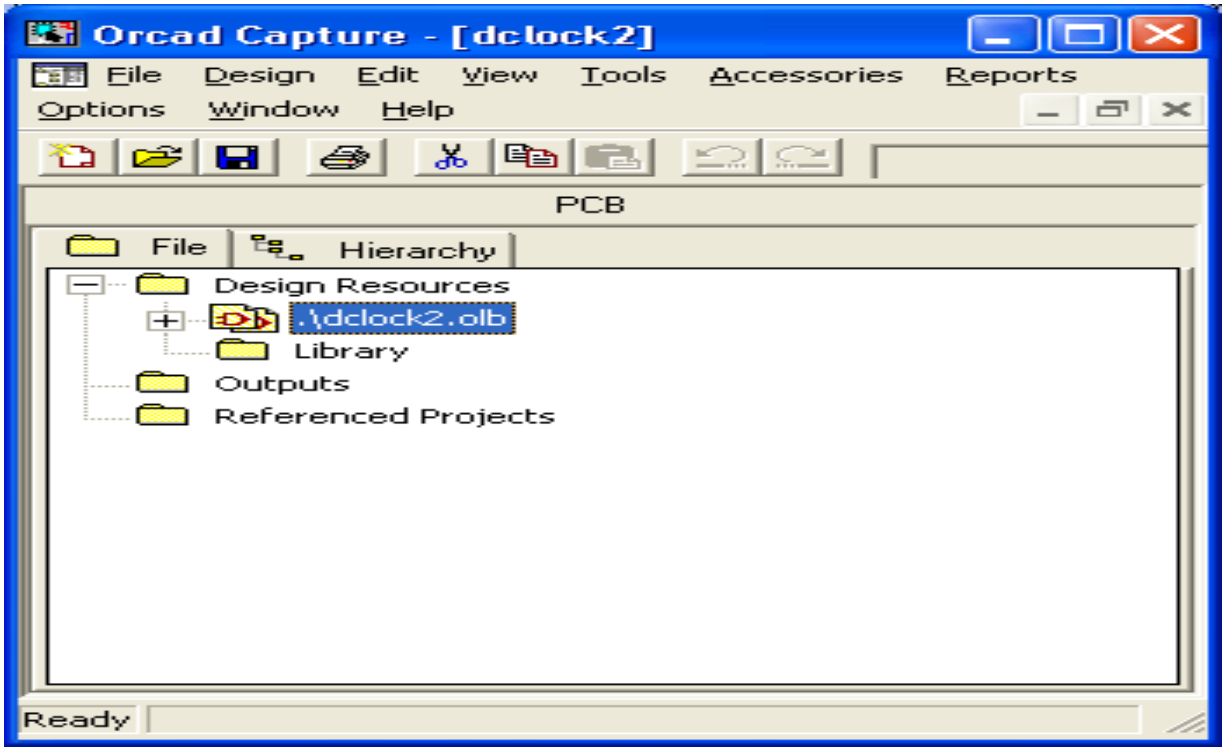


Рис. 5.33. Каталог бібліотеки

Після розміщення першого символу на діаграмі т. зв кеш проекту, де цей символ скопійовано з файлу бібліотеки. У результаті символи для всіх компонентів, розміщених на діаграмі проекту, додаються до розділу Design Cache менеджера проекту, зберігаючи їх зв'язок із бібліотеками символів. Це дозволяє вам синхронно змінювати всі входження символу у вашому проекті, змінюючи його в бібліотеці. Для цього виберіть символ компонента в розділі «Дизайн кешу» та виконайте «Дизайн» > «Замінити кеш». Ім'я вибраного символу відображається в рядку «Назва частини» діалогового вікна для цієї команди, як показано на малюнку 5.34. Потім у рядку «Бібліотека частин» вказується назва бібліотеки (за допомогою браузера перегляду), де вона розташована. Якщо натиснути «ОК», усі входження цього символу в поточному проекті буде замінено на символ бібліотеки. Команда Design>Replace Cache оновлює вибраний символ і всі параметри, введені користувачем,

зберігаються.

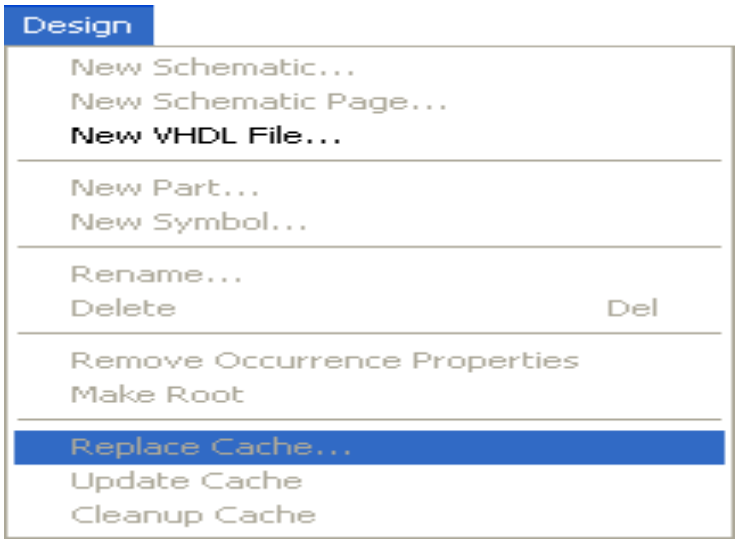


Рис. 5.34. Заміна кеша проекту

В OrCAD Capture можна створювати символи компонентів, а потім розміщувати їх у існуючих або нових бібліотеках. Щоб створювати або редагувати символи, ви використовуєте редактор частин, доступ до якого можна отримати одним із трьох способів.

1. Щоб створити новий символ, створіть новий або відкрийте існуючу бібліотеку, а потім виберіть « Дизайн» > «Створити» > «Частина» .
2. Щоб відредагувати існуючий символ, відкрийте бібліотеку символів у менеджері проектів (рис. 5.33), а потім подвійним клацанням курсору виберіть потрібний символ.
3. Щоб відредагувати символ, розміщений на діаграмі, виділіть його одним клацанням курсора, а потім виконайте команду Edit>Part.

Створіть новий символ. Залежно від призначення символи створюються за допомогою двох різних команд.

За допомогою команди Дизайн > Новий створюються допоміжні символи (рис. 5.35) чотирьох типів:

- Сила - символ для з'єднання ланцюжків «маса» і «потужність»;
- Вимкнено - сторінка Роз'єм - символ роз'єму збоку схеми;
- Ієрархічний Порт - символ ієрархічного блоку;

- Титульна табличка - умовне позначення основного напису («кутовий штамп»), її приклад, виготовлений за ЕСКД, наведено на рис. 5.36.

У стовпчику Назва (рис. 5.35) вказується назва символу, а в стовпчику Тип символу вибирається його тип. Умовні позначення цього типу розміщуються на схемах і не відповідають фізично існуючим елементам. Типи цих допоміжних символів враховуються лише під час виконання команд Place>Power, Place>Ground, Place>Off-Page Connector, Place>Hierarchical Port, Place>Title Block - розміщується лише список компонентів відповідного типу у вибраному каталозі бібліотеки в діалогових вікнах команд.

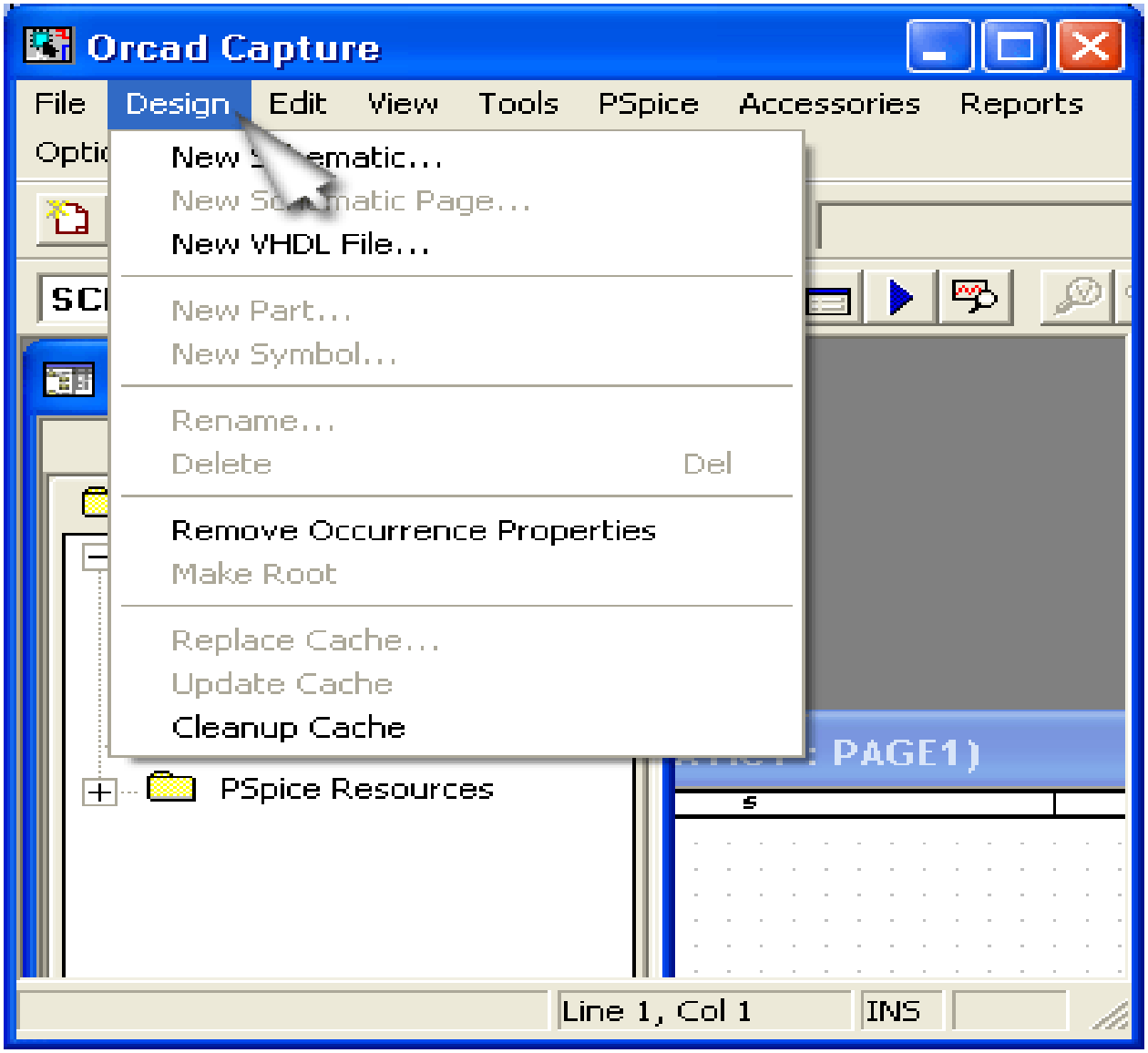


Рис . 5 . 35 . Діалог вікно Команди Design > New Symbol і Design > New Part

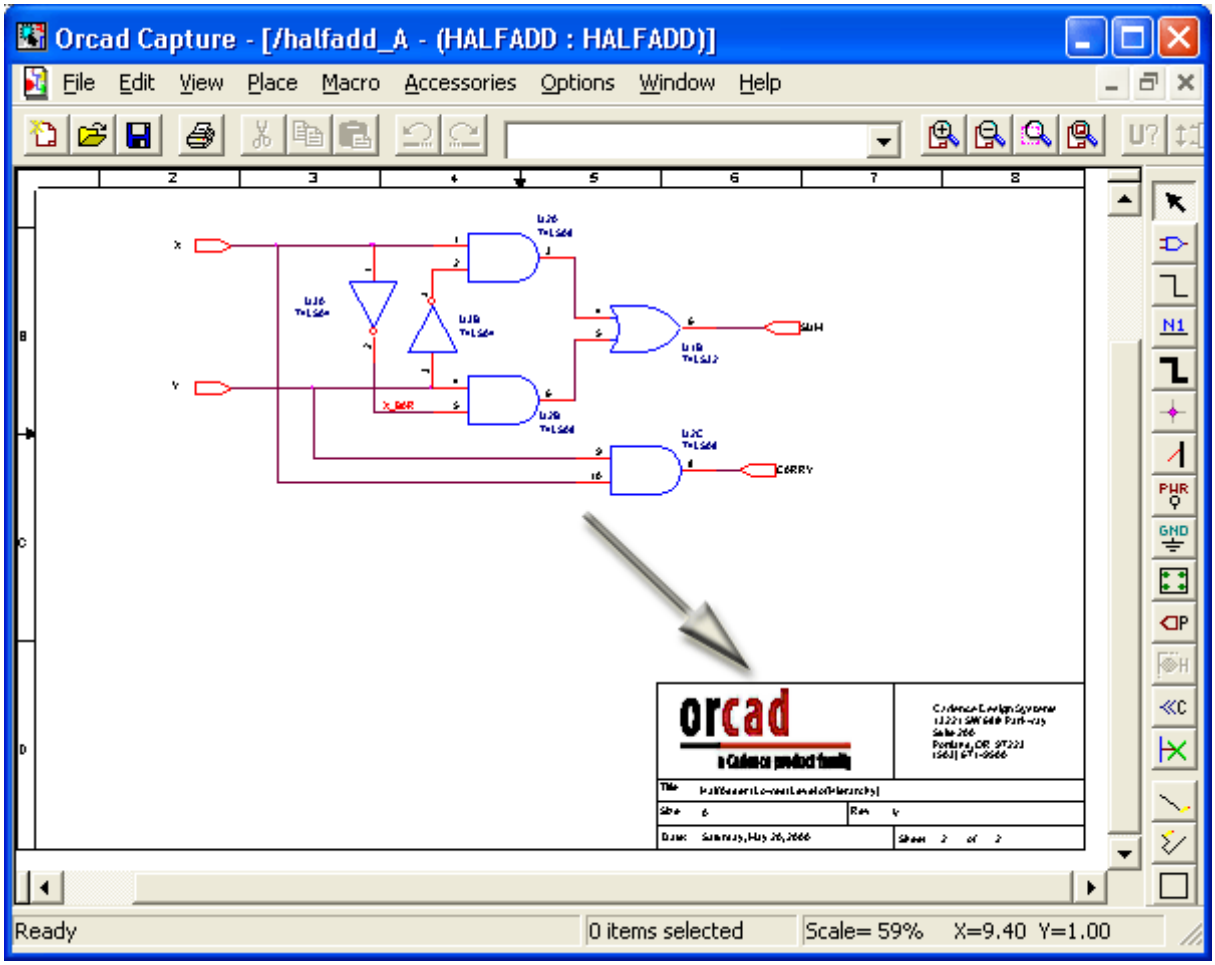


Рис. 5.36. Приклад символу основного напису. (Блок заголовка)

Символи для всіх інших компонентів, більшість із яких мають фізично існуючі тверді тіла, вводяться за допомогою команди «Дизайн» > «Нова частина». У діалоговому вікні цієї команди вводяться такі дані:

- Назва - назва символу;
- Привіт довідка Префікс - префікс позначення позиції (наприклад, R для резистора, C для конденсатора, DA для аналогової IMC, DD для цифрової IMC і т.д.);
- RSV Footprint - назва типового корпусу компонента, наприклад, DIP 16, SOI 24, якщо він існує (цей параметр є обов'язковим лише при передачі схеми розробки друкованої плати, він не потрібен під час моделювання);
- Створити конвертувати Перегляд – необхідність створення другого зображення символу (наприклад, еквівалент ДеМоргана для цифрових логічних елементів);

- Запчастини ззаду Пакет - загальна кількість секцій в корпусі компонента;
- Homogenouse або Heterogenouse - вибір між компонентами з перерізами одного або різних типів (наприклад, IC 133LAZ, що містить 4 логічні елементи 2I-NOT, належить до класу Homogenouse, а IC 564LP2, що містить 2 логічні елементи ZILy-NOT і елемент NOT, належить до класу Homogenouse. до класу гетерогенних);
- Алфавітний або цифровий - вибір між позначеннями перерізів багатосекційних елементів латинськими літерами, наприклад DD 1 A , DD 1 B , DD 1 C тощо (латинськими літерами можна позначати складові перерізи до 26 секцій в одному регістр) або числа, наприклад .DD 1-1, DD 1-2, DD 1-3 ;
- Привіт Псевдоніми - визначення псевдонімів символів для - зменшення гучності. бібліотеки (наприклад, ви можете створити компонент LA 3 і надати йому псевдоніми 133 LA 3, K 155 LA 3, 530 LA 3);
- Прикріпити Реалізація - підключення додаткового опису символу за допомогою діаграми-замінника, VHDL-файлу, списку підключень, іншого проекту або у вигляді моделі PSpice ; Pin Числа Видимий – відображення вихідних чисел на діаграмі.

Після натискання панелі ОК у діалогових вікнах команди «Дизайн» > «Нова деталь» або «Дизайн» > «Новий символ» відкривається робоче середовище «Редактор деталей» (рис. 5.37), де розміри символу обмежені пунктирним прямокутником (розміри цього прямокутника є змінений звичайним способом шляхом «перетягування» його кутів).

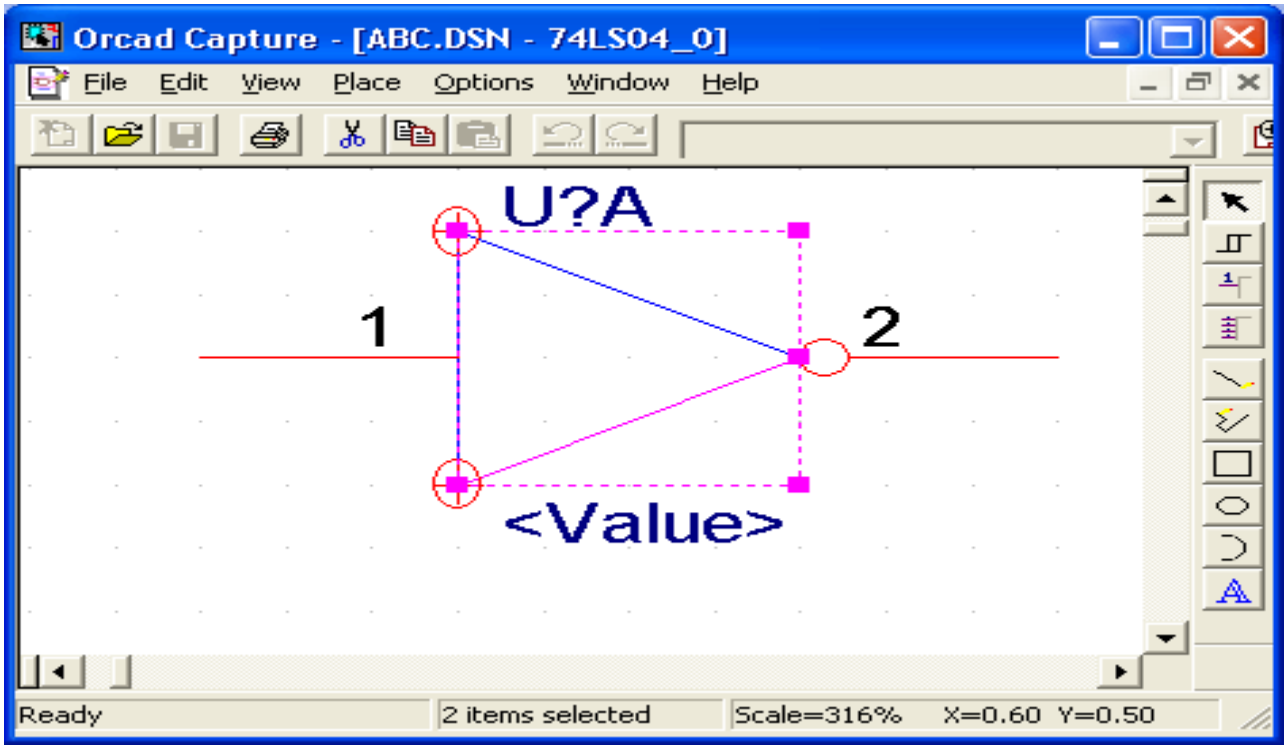


Рис. 5.37. Підготовка символу _

Піни компонентів повинні бути розміщені за межами цього прямокутника, торкаючись його. Вставлення пінів компонента виконується за допомогою команди Place > Pin, діалогове вікно якого показано на рис.5.38, о.В ньому відображається така інформація:

- Ім'я - ім'я виводу;
- Число - вихідне число;
- Форма - початкова форма (див. табл. 5.5);
- Тип - тип виводу (див. табл. 5.6), який використовується тільки при перевірці правильності складання схеми за допомогою команди Інструменти>Перевірка правил проектування (DRC);
- Скаляр або шина - вибір між одним виходом або шиною;
- Pin Visible - відображення виходу на схемі (тільки для виходів типу Power), у вікні Edit Parts ці виходи відображаються без введення їх назв і номерів;

- Властивості користувача – відкриває діалогове вікно, яке дозволяє переглядати та редагувати вихідні характеристики компонента перед розміщенням його в робочій області.

Атрибут <Value> автоматично розміщується під контуром елемента (його положення можна змінити, помістивши всередину контуру), якщо його значення не визначено, то ім'я елемента автоматично вказується на діаграмі як його значення.

Таблиця 5.5 Графік висновків.

Форма (Shape)	Описание
Clock	Вход синхронизации
Dot	Признак логического отрицания
Dot-Clock	Вход синхронизации с инвертированием
Line	Стандартный вывод, длина которого равна трем шагам сетки
Short	Короткий вывод, длина которого равна одному шагу сетки
ZeroLength	Стандартный вывод нулевой длины

Таблиця 5.6. Типи додатків.

Тип вывода	Описание
3-State	Тристабильный вывод, имеющий три возможных состояния: логическое состояние низкого уровня, догическое состояние высокого уровня и состояние большого выходного сопротивления (Z-состояние, это состояние эквивалентно разрыву цепи). Например, 8-разрядный регистр-защелка 74LS373 (КР1533ИР22) имеет тристабильные выводы
Bidirectional	Двунаправленный вывод (может быть как входом, так и выходом компонента)
Input	Вывод подачи входного сигнала
Open Collector	Выход вентиля с открытым коллектором (к нему подключается резистор нагрузки)
Open Emitter	Выход вентиля с открытым эмиттером (к нему подключается резистор нагрузки)
Output	Выход компонента
Passive	Вывод пассивного компонента (резистора, конденсатора, диода и т.п.)
Power	Выводы для подключения цепей "земли" и "питания". Например, для ИС серии 133 питание подключаются к выводу 14, а "земля" – к выводу 7. Имена этих выводов должны совпадать с именами соответствующих цепей

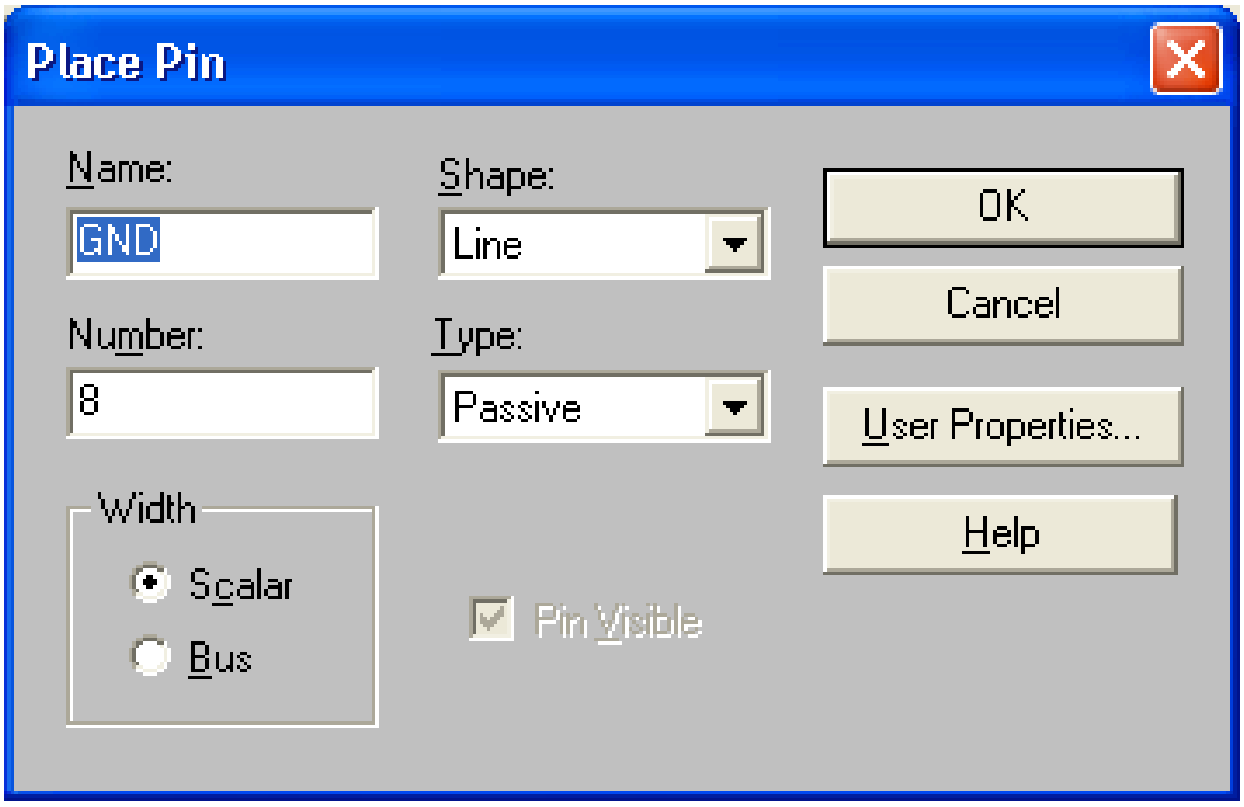


Рис. 5.38, а) Діалогове вікно для розміщення окремого виводу компонента. (A) і набір висновків (b)

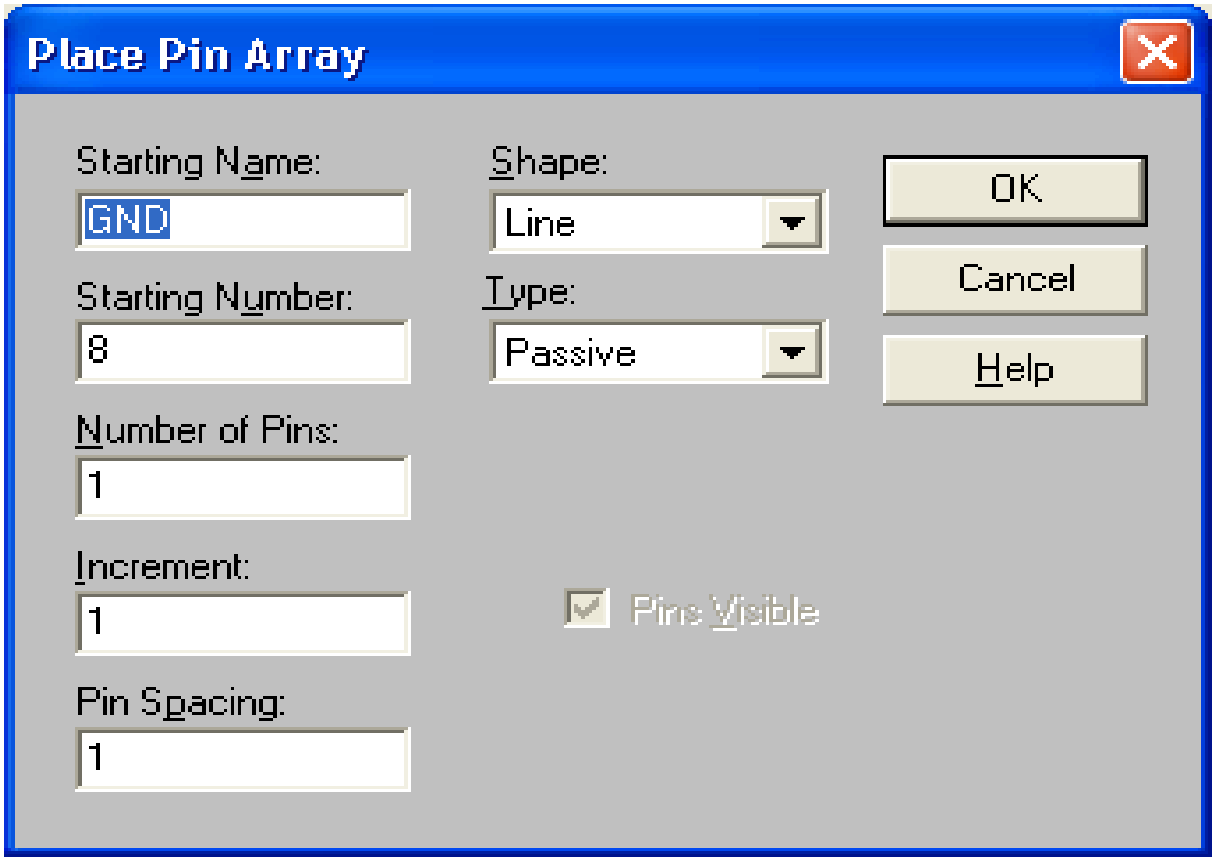


Рис. 5.38, б)

Розміщення масивів Pin здійснюється за допомогою команди Place › Pin Array, діалогове вікно якої показано на рис. 5.38, б). Містить таку інформацію:

- Запуск Ім'я – ім'я першого виводу масиву. Якщо назва виходу закінчується однією з цифр 0...9, то значення, вказане в параметрі Інкремент , додається послідовно до назв наступних виходів . Якщо над найменуванням необхідно поставити знак мінус у вигляді горизонтальної риски, то після кожного символу такого найменування ставиться коса риска «». Наприклад, введення символів R \ E \ S \ E \ T \ визначає назву RESET ;
- Запуск Number – номер першого виводу масиву;
- Номер с Pins – кількість кеглів на дошці;
- Приріст – приріст імен виходів таблиці, що вставляються автоматично (якщо ім’я першого виходу закінчується цифрою);
- Pin Інтервал – відстань між сусідніми виходами масиву в одиницях кроку сітки;
- Форма – початкова форма (табл. 5.5);
- Тип – тип виходу (табл. 5.6);
- Кілочки Видимий – відображення виходів схеми (лише для виходів типу Power).

Ділянки як однорідних, так і різнорідних елементів можуть мати загальні виводи, найчастіше це виходи для підключення ланцюгів «земля» і «силова», тобто виходи силового типу. Зазвичай ці контакти невидимі та, як вважають, підключені до схем, назви яких збігаються з назвами контактів. У разі різнорідних елементів клеми «земля» і «питання» достатньо розташувати хоча б на одній секції, у разі однорідних елементів ці клеми автоматично розміщуються у всіх секціях (одночасно всі їхні копії мають однакові назви та номери), тому їх завжди не видно на схемі. Щоб відобразити всі «основні» та «питальні» результати (з метою документації), виберіть потрібну назву проекту в менеджері проектів, клацнувши курсор і вибравши «Властивості

проекту» в меню «Параметри», а потім виберіть параметр «Показати невидимі контакти живлення» в розділі «Різне». вкладка.

Після складання висновків з розрізу наносять його контур і наносять додаткові текстові написи (див. рис. 5.39, а). У той же час функціональні символи, наведені в таблиці, зручно застосовувати за допомогою команди Plase>IEEE Symbols. 5.7. Команда Вид>Наступная деталь відкриває зображення наступного перерізу - при однорідних компонентах просто нанесіть номери додатків (послідовно виділивши їх клацанням курсора), як показано на рис. 5.39, б; для однорідних компонентів зображення кожної ділянки перемальовується. Перегляд зображень усіх перерізів багатопрофільних компонентів здійснюється за допомогою команди Вид>Пакет (див. рис. 5.39, в), перехід до редагування окремого перерізу здійснюється клацанням курсора.

Параметри компонента вводяться за допомогою команди Параметри > Властивості частини, діалогове вікно якої показано на рис. 5.16, народження. Параметри пакета компонентів вводяться за допомогою команди Options>Package Properties, діалогове вікно якого показано на рисунку 5.40. Перекомпіляція всіх цих параметрів є досить виснажливим завданням, тому при створенні нового компонента доцільніше скопіювати компонент того ж типу в бібліотеку символів за допомогою засобів Windows і потім редагувати його параметри.

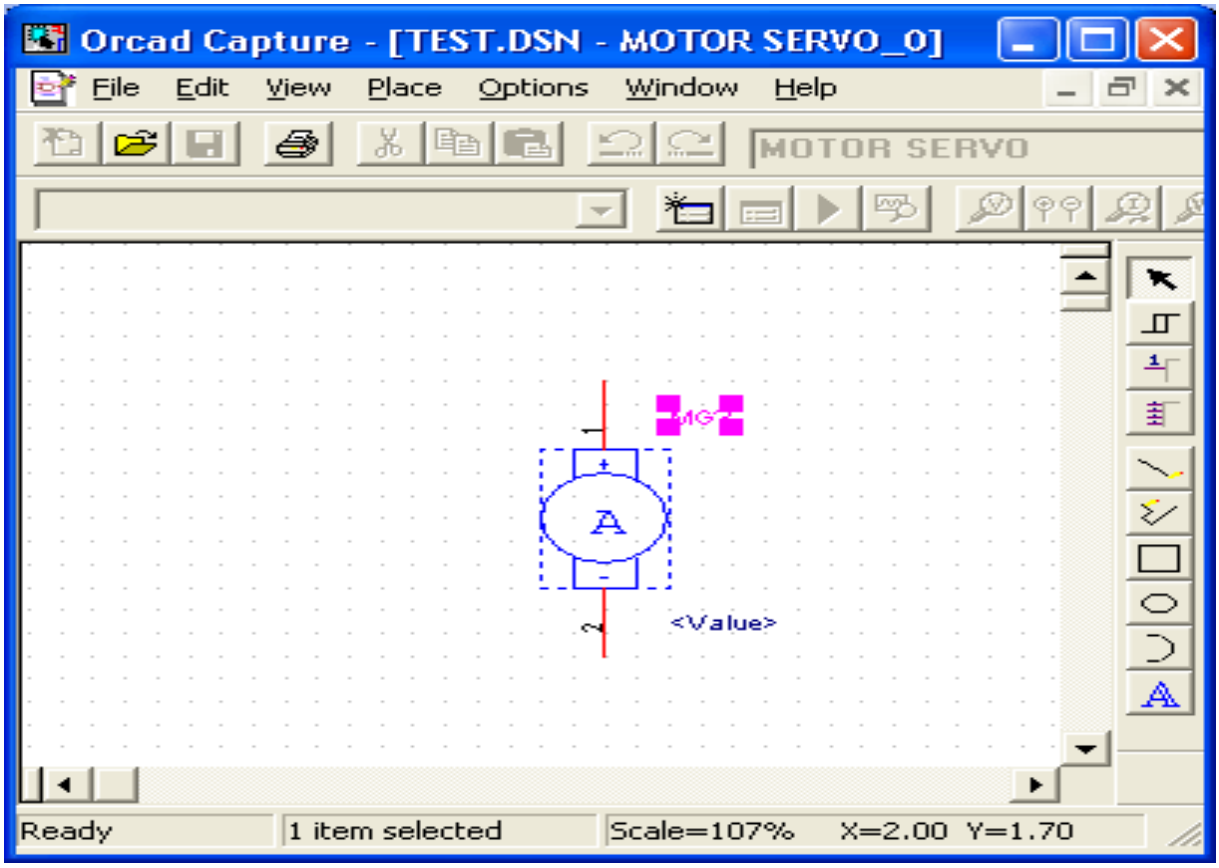


Рис. 5.39, а) Введення графічних зображень і номерів заявок окремих ділянок однорідного компонента (а, б) і перегляд упаковки трикомпонентного компонента (в)

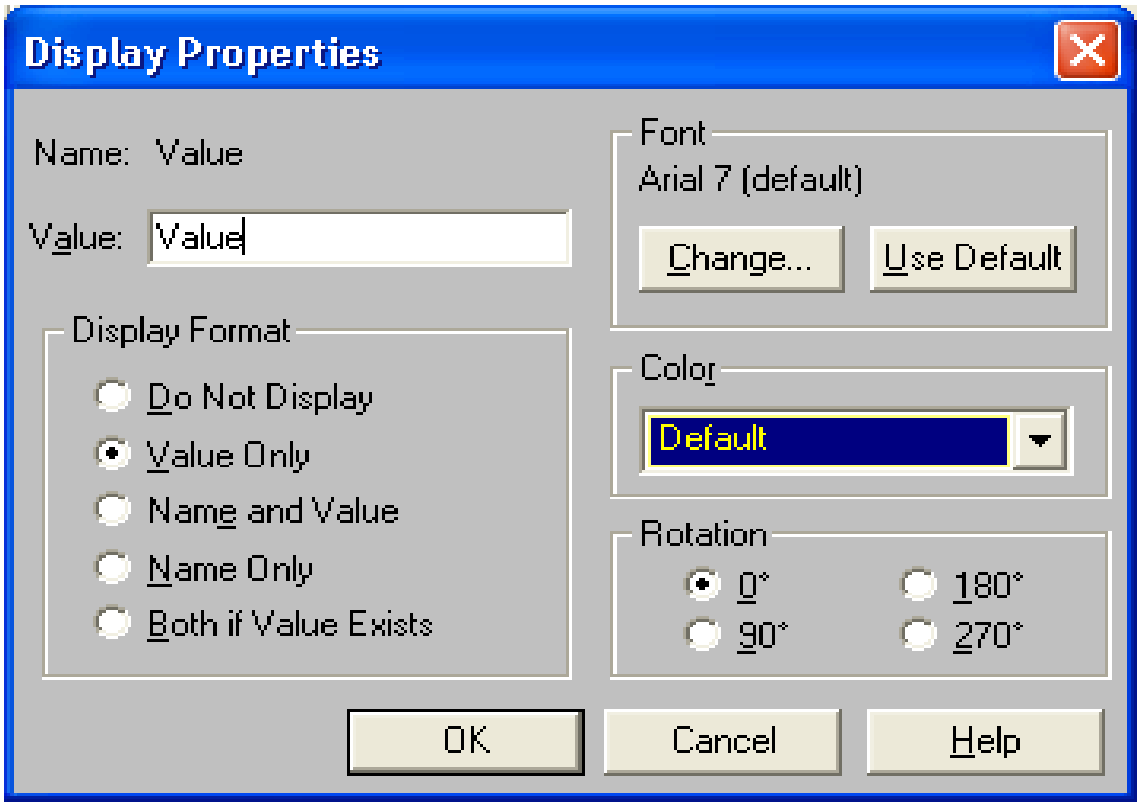


Рис. 5.39, б)

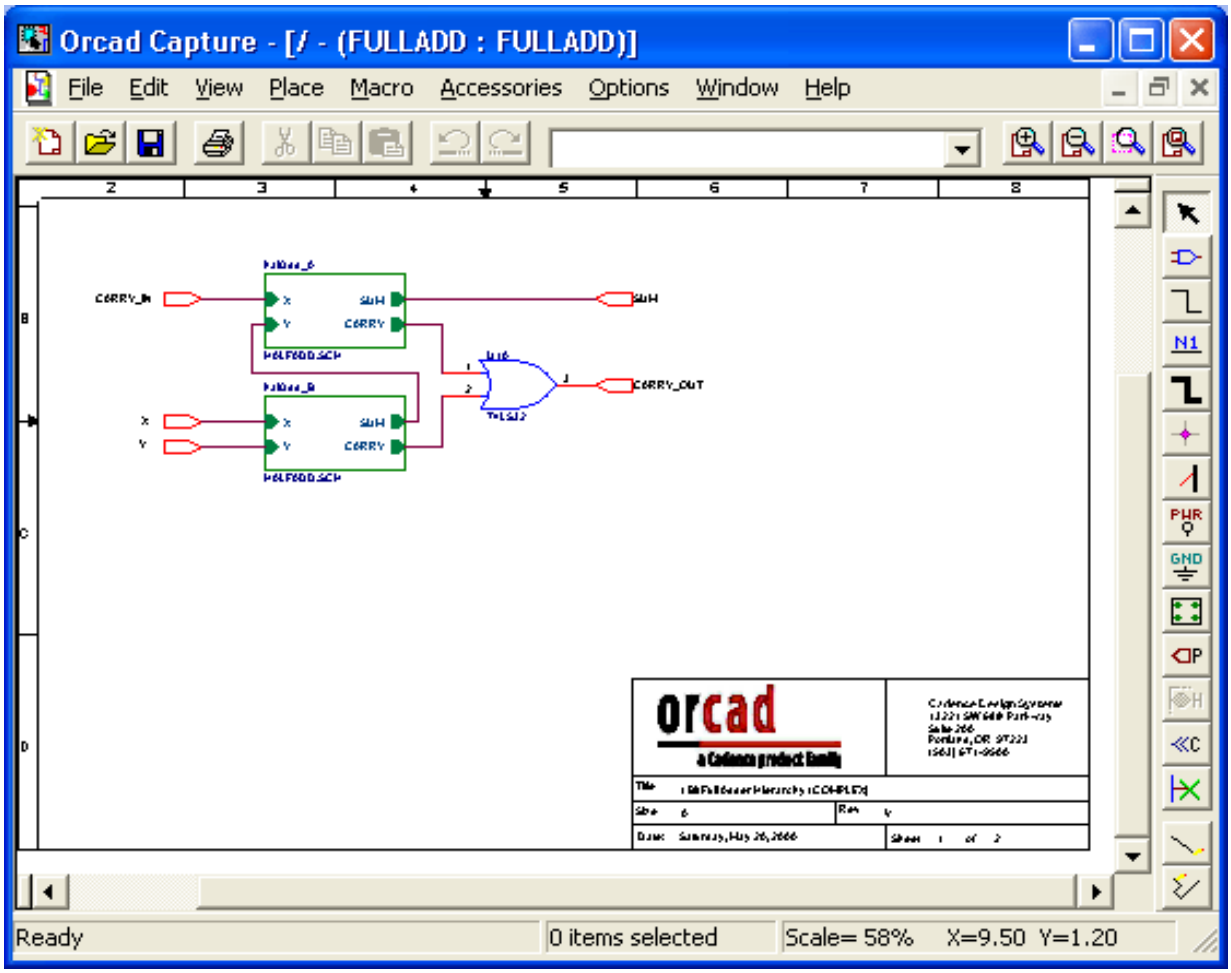


Рис. 5.39, дюйми)

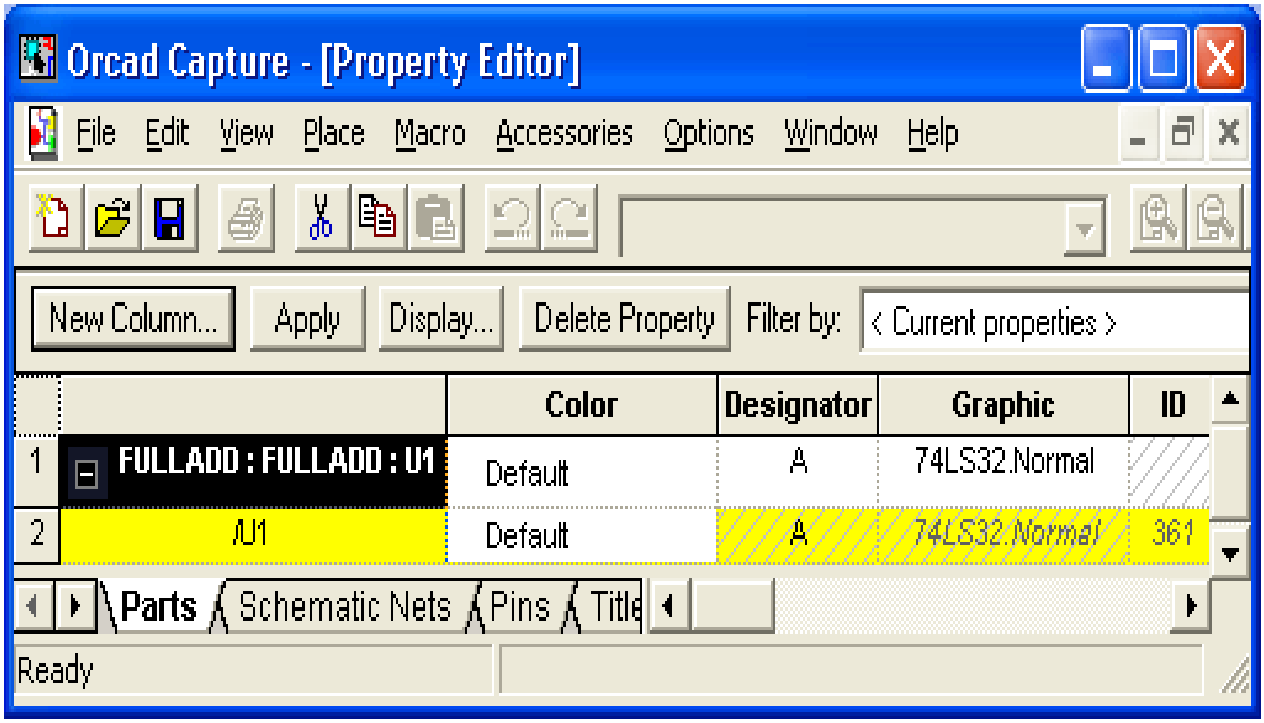


Рис. 5.40. Введення параметрів упаковки компонентів

Геометричні розміри символу пов'язані з кроком сітки, прийнятим при його побудові. Якщо при введенні символу на діаграмі змінити цей крок (він вважається рівним параметру Pin-to-Pin Spacing на вкладці Page Size, рис. 5.15, в), то розміри всіх символів зміняться пропорційно. Тому при створенні бібліотек символів рекомендується спочатку вибрати і встановити однакове значення для параметра Pin Spacing.

Увага

Символи кирилиці дозволені як імена компонентів в OrCAD Capture, наприклад, 133ИР7, але це не рекомендується, оскільки немає гарантії, що помилка не виникне в майбутньому, наприклад, під час надсилання даних до іншого модуля OrCAD або до іншої системи проектування. Що стосується назв додатків, то в них заборонено використовувати кириличні символи. Загалом, щоб уникнути плутанини, рекомендується використовувати кириличні символи лише в текстових рядках в імпортованій САПР.

Таблиця 5.7. Символи IEEE.

Символ	Символ
3 State	LE
Active Low Left	NE
Active Low Right	Non Logic
Amplified Left	Open Circuit H-type
Amplified Right	Open Circuit L-type
Analog	Open Circuit Open
Arrow Left	Passive Pull Down
Arrow Right	Passive Pull Up
BiDirectional	Pi
Dynamic Left	Postponed
Dynamic Right	SHIFT Left
GE	SHIFT Right
Generator	Sigma
Hysteresis	

Щоб продовжити проектування після створення опису схеми, виконайте команду «Інструменти>Створити список мереж» менеджера проекту. Під час моделювання за допомогою OrCAD PSpice ця команда завантажується автоматично; для передачі даних в програму розробки PP OrCAD Layout та

інші (всього доступно близько 40 вибраних користувачем форматів для створення списку підключень), ця команда виконується вручну, попередньо виділивши назву проекту в менеджері проектів.

Перед моделюванням слід виключити повторення позначок положення елементів, а перед розробкою ПП секції елементів додатково упакувати в корпус. Ці операції виконуються за допомогою команди «Інструменти» > «Анотувати менеджер проекту», діалогове вікно якої показано на малюнку 5.19.

Перед створенням списку підключень рекомендується запустити команду Tools>Design Rule Checker (DRC) для виявлення помилок на діаграмі (при запуску програми моделювання PSpice ця команда завантажується автоматично, але в будь-якому випадку необхідно вказати її конфігурацію наперед). Протокол керування зберігається у файлі *.drc і копіюється у файл протоколу Session Log (за бажанням користувача місця помилок не позначаються на схемі спеціальними маркерами DRC).

У звітах повідомляється про два типи порушень дизайну:

- Помилки - помилки, які необхідно виправити;
- Попередження - попередження, які можуть призвести до помилок при моделюванні проекту (на них не потрібно реагувати).

Після виконання команди Design Rule Checker відкривається діалогове вікно завдання Rule Checker, яке складається з двох вкладок (рис. 5.41).

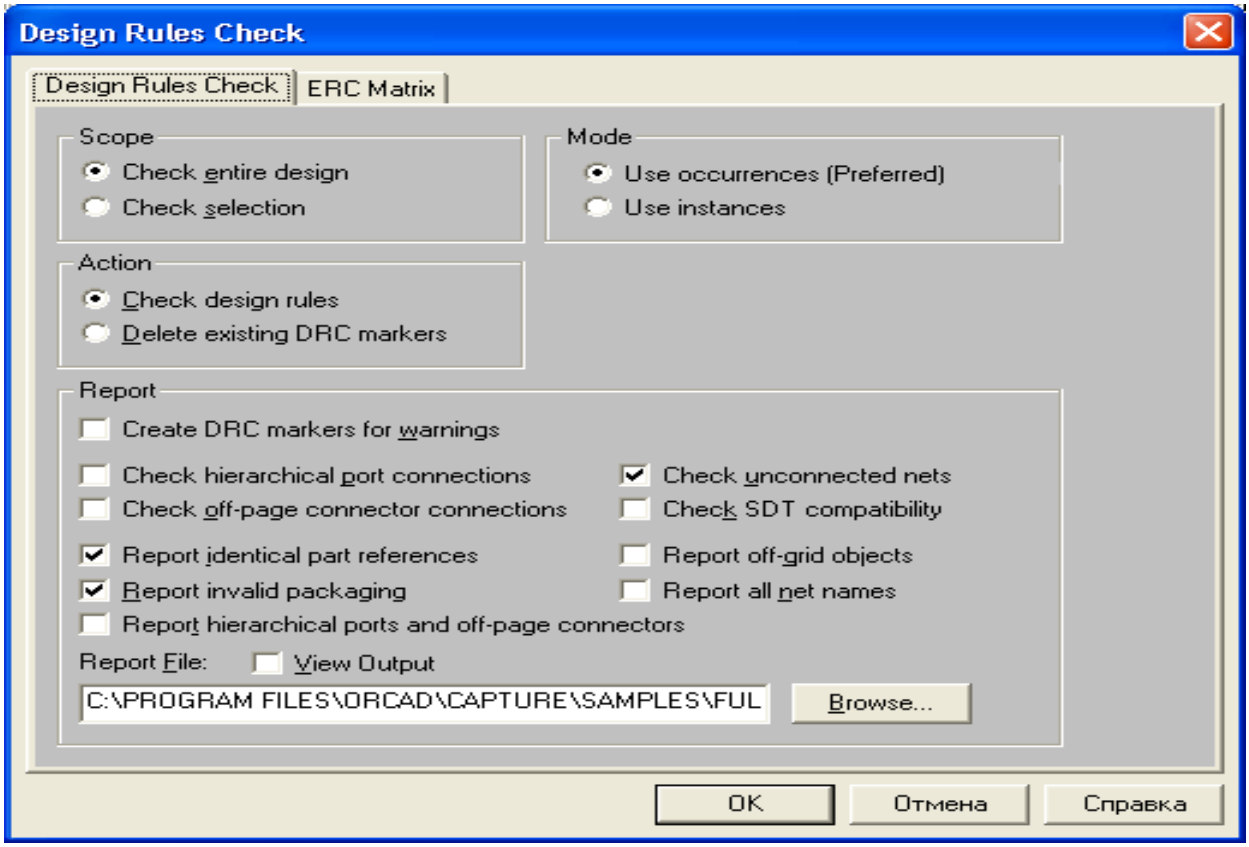


Рис. 5.41, а) Інструменти > завдання конфігурації Design Policy Checker

Вкладка «Контроль правил проектування» (рис. 5.41, а) визначає, яка інформація повинна бути включена в акт перевірки:

- Обсяг - перевірка всього проекту (Перевірити ціле проект), вибрану сторінку або кілька сторінок (перевірте вибір);
- Дія - перевірити дотримання всіх правил оформлення (Перевірити дизайн правила) або видалення раніше використаних тегів DRC з програми ;
- Звіт (вибір інформації, включеної до звіту про перевірку):
- Створення тегів DRC для попереджень - розміщення символів DRC, що попереджають про можливі помилки, відповідно до правил, зазначених у таблиці ERC (теги DRC завжди розміщуються в місцях безумовних помилок):
    - Перевірка з'єднань ієрархічних портів - перевірка збігу імен ієрархічних запитів з відповідними ієрархічними портами в

схемах їх заміщення, а також збігу їх загальної кількості та типів усіх запитів;

- Перевірити з'єднання позасторінкових з'єднувачів - перевірити узгодженість імен міжсторінкових з'єднувачів (підключених до однойменних рядків), розташованих на різних сторінках схеми;
- Повідомити про ідентичні посилання на частини – включити у звіт список компонентів з однаковими позначками розташування;
- Повідомити про неправильне пакування - включити до звіту перелік компонентів, які мають однакову форму, але різну інформацію на упаковці
- Звіт про ієрархічні порти та з’єднувачі за межами сторінки – складання списку всіх портів ієрархічних блоків і з’єднувачів за межами сторінки;
- Перевірка на наявність непідключених мереж — виявлення ланцюгів, які не підключені принаймні до двох компонентних виходів або не підключені до зовнішніх джерел сигналу, а також ланцюгів, які мають однакові назви на різних сторінках схем, але не мають міжсторінкових роз’ємів як порти комбінований або ієрархічний;
- Перевірити сумісність SDT - перевірити сумісність з редактором графічних схем OrCAD SDT для DOS (ця сумісність необхідна, якщо ви плануєте зберегти схему проекту у форматі OrCAD SDT);
- Повідомити об’єкти поза сіткою – список назв і координат об’єктів, розташованих поза вузлами сітки;
- Повідомити про всі імена мереж - створіть список імен усіх мереж.

- Файл звіту - присвоєння імені файлу звіту (за замовчуванням його ім'я відповідає назві проекту, розширення імені drc);
- Перегляд результатів - перегляд результатів перевірки на екрані.

На вкладці ERG Matrix правила перевірки задаються та зберігаються у вигляді матриці керування електричними правилами (ERC, рис. 5.41, б). У

рядках і стовпцях матриці вказуються типи запитів компонентів і різні порти (див. табл. 2.5). Незафарбована комірка означає дозвіл на об'єднання відповідних додатків, попередження позначені символом W, помилки - символом E. Наприклад, згідно з малюнком, наведеним на рис. 5.41, b матриці ERC, з’єднання Вихід-Вхід дозволено, буде відображено попередження про з’єднання Відкритий емітер-Відкритий колектор, а з’єднання Потужність-Вихід вважатиметься помилкою.

Тому перед виконанням команди Інструменти > Перевірка політики дизайну необхідно відредагувати вміст матриці ERC відповідно до характеристик поточного дизайну.

Наведемо приклад файлу звіту про результати аудиту проекту.

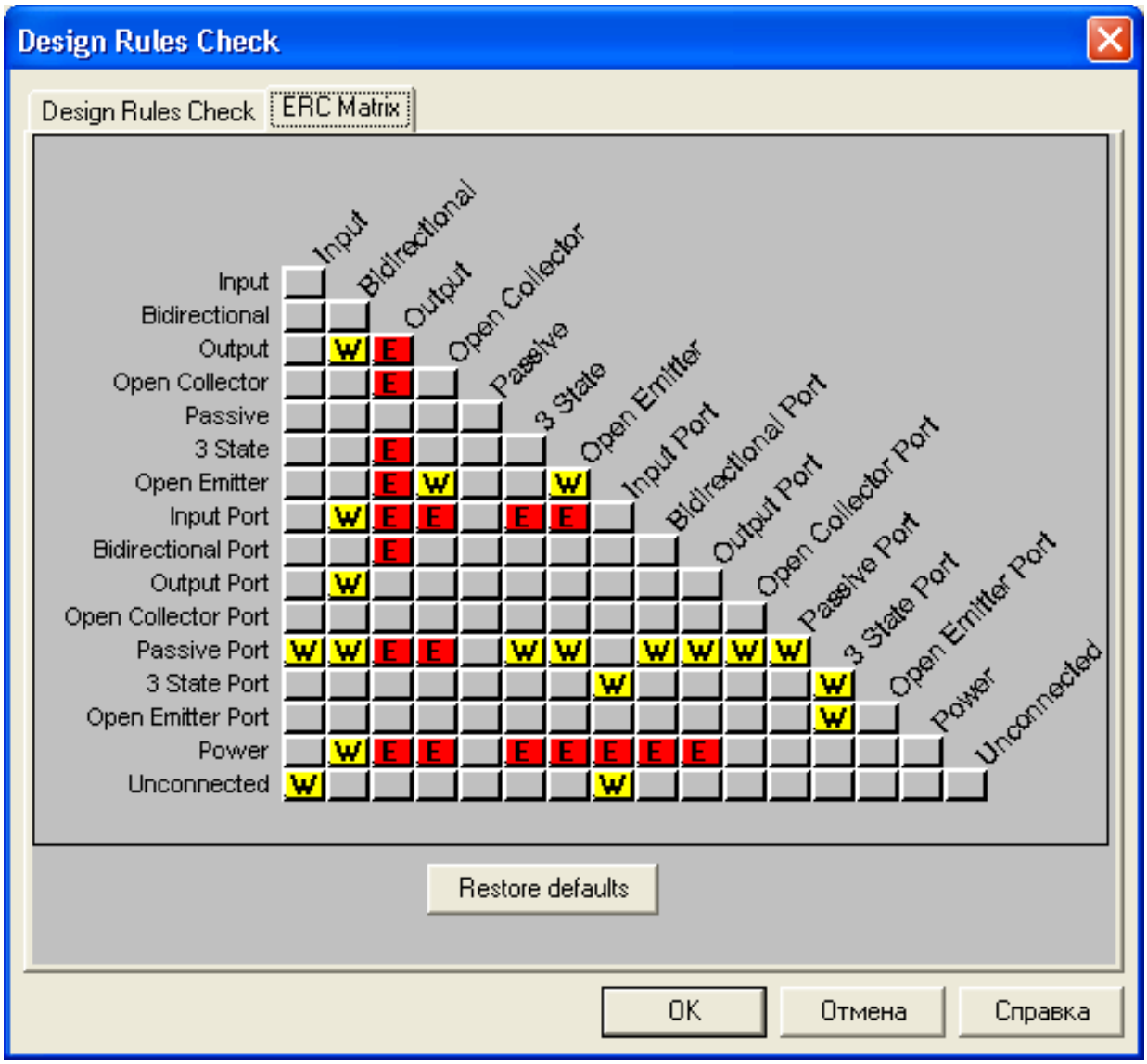


Рис . 5.4 1

Після того як ви виправите виявлені помилки, ви можете виконати Інструменти > Створити список мереж, щоб створити список підключень вашого проекту. У діалоговому вікні цієї команди доступно 9 вкладок для вибору формату списку викликів (рис. 5.42). Перші 8 вкладок пов’язані з певними форматами:

- EDIF 200 - електронний формат обміну даними, який має два різновиди; з підтримкою (ієрархічний netlist) або без підтримки (flat netlist) з ієрархічною структурою (залежно від обраної конфігурації системи);
- PSpice (файли *.net), SPICE (файли *.cir) - формат програми моделювання PSpice і SPICE (ієрархічні блоки представлені у вигляді макромоделей за допомогою директиви SUBCKT);
- VHDL (файли *.vhd), Verilog (файли *.v) - опис цифрових пристроїв мовами VHDL і Verilog;
- Layout - список зв'язків проекту у форматі програми OrCAD Layout для створення друкованих плат (*.mnl бінарні файли);
- INF - передача даних у стару версію програми моделювання DOS OrCAD Digital Simulation Tools 386+ (файли *.inf).

Залежно від обраного формату користувач повинен адаптувати форму вихідного файлу відповідно до специфіки проекту (наприклад, для формату OrCAD Layout див. рис. 5.42, увімкнення параметра Run ECO to layout призведе до автоматичного перенесення новий (змінений) файл до програми розробки друкованих плат OrCAD Layout.mnl, стару плитку необхідно попередньо завантажити в Layout). Натиснувши дев'яту кнопку «Інше», ви можете вибрати ще близько 40 форматів, найвідоміші з яких: Allegro, EEDesigner, Intergraph, HiLo, Mentor, PADS, P-CAD, Scicards, Tango, VST. Варто зазначити, що список підключень у форматі програми Cadence Allegro PCB Layout (Unix і NT) також можна скомпілювати за допомогою команди Accessories > Allegros > Allegro Netlist.

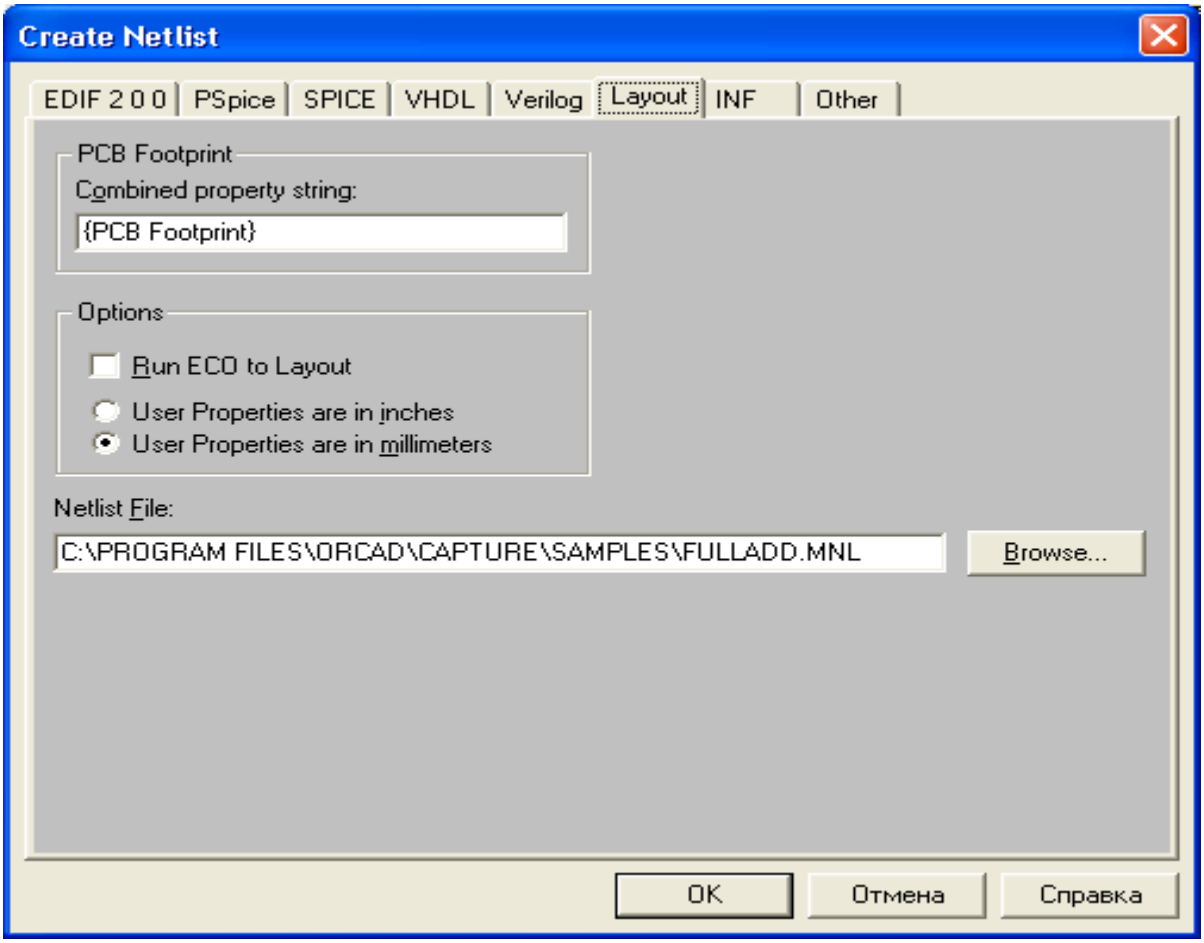


Рис. 5.42. Вибір формату списку підключень у діалоговому вікні команди «Створити список мереж».

OrCAD Capture має три команди створення технічні характеристики проект : Інструменти > Список матеріалів, Інструменти > Посилання та звіти > Список матеріалів CIS. IN власні черга команда Звіти>Список матеріалів CIS має два Підкоманди Standard і Crystal Reports (остан Є Просто IN версії Capture CIS). Діалог вікна команди редакція звіти представлені УВІМКНЕНО рис _ 5,43.

IN діалогічний вікно вибрано команду Сервис>Описание (рис . 5.43, а). наступні варіанти :

- Область застосування - збірка звіт вниз все проект (обробка всього проекту) або його обраний частини (процес виділення);
- Визначення окремого елемента (призначення списку змінних, включених до звіту):

- Заголовок - назви діаграм звіту (мають відповідати даним, що містяться в рядку властивості Combined); назви діаграм відокремлюються символом "\t" і встановлюються як текстові змінні, в яких допускаються символи кирилиці;
- Linked Property String - завдання перерахування значень, які розміщені в стовпцях звіту; назви величин беруть у дужки та відокремлюють символами "\t"; наприклад, щоб включити позначення та назви елементів компонентів до звіту, компілюється {Reference}\t{Value}; у звіті ці значення розділені табуляціями;
- Розмістити кожен запис частини в окремому рядку - вивести для кожного компонента в окремому рядку;

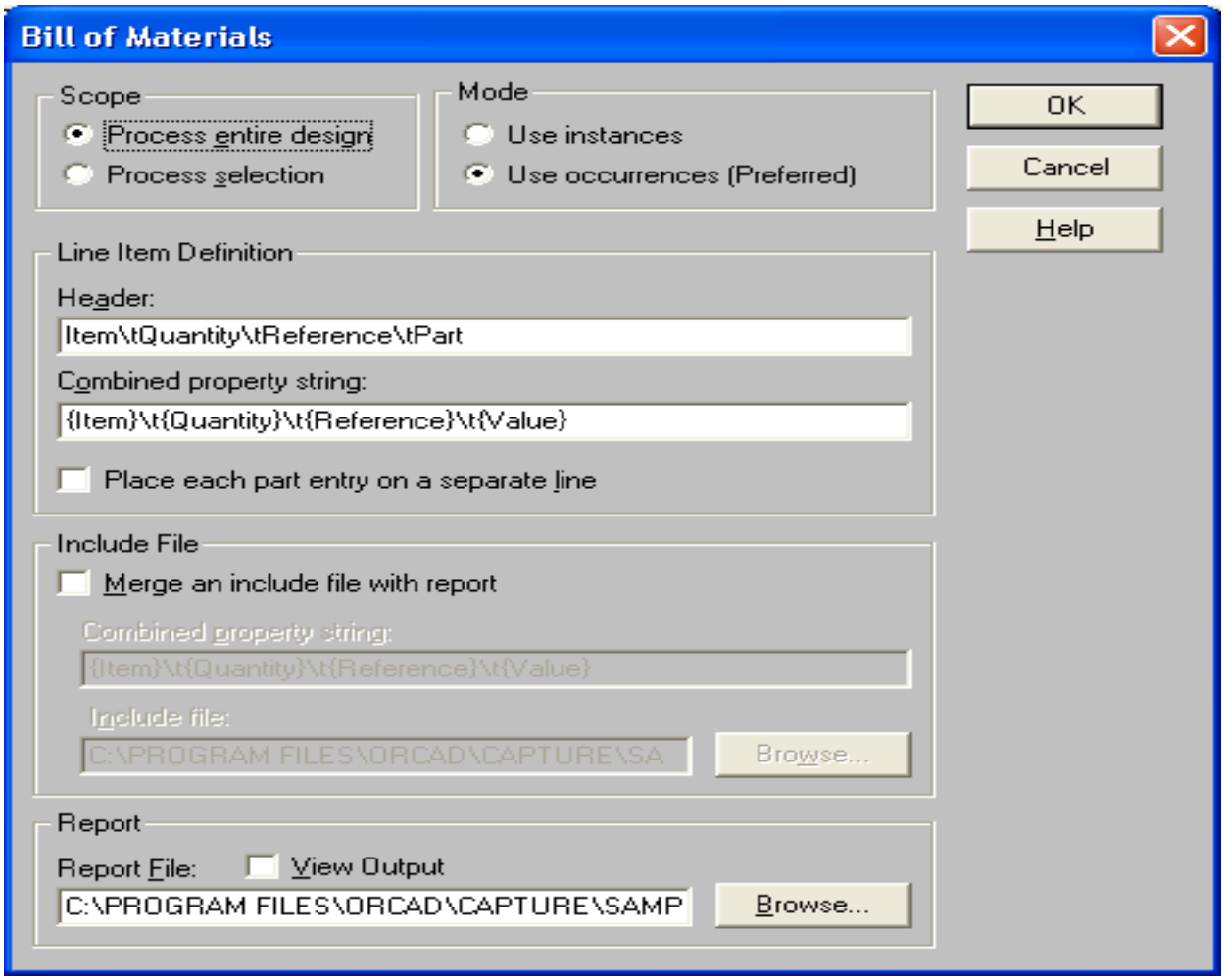


Рис . 5.43, a) Діалог вікна Команди Інструменти > Перелік матеріалів (a), Інструменти > Перехресні посилання (b), Звіти > Перелік матеріалів CIS > Стандарт (c) і Звіти > Перелік матеріалів CIS > Звіти Crystal (d)

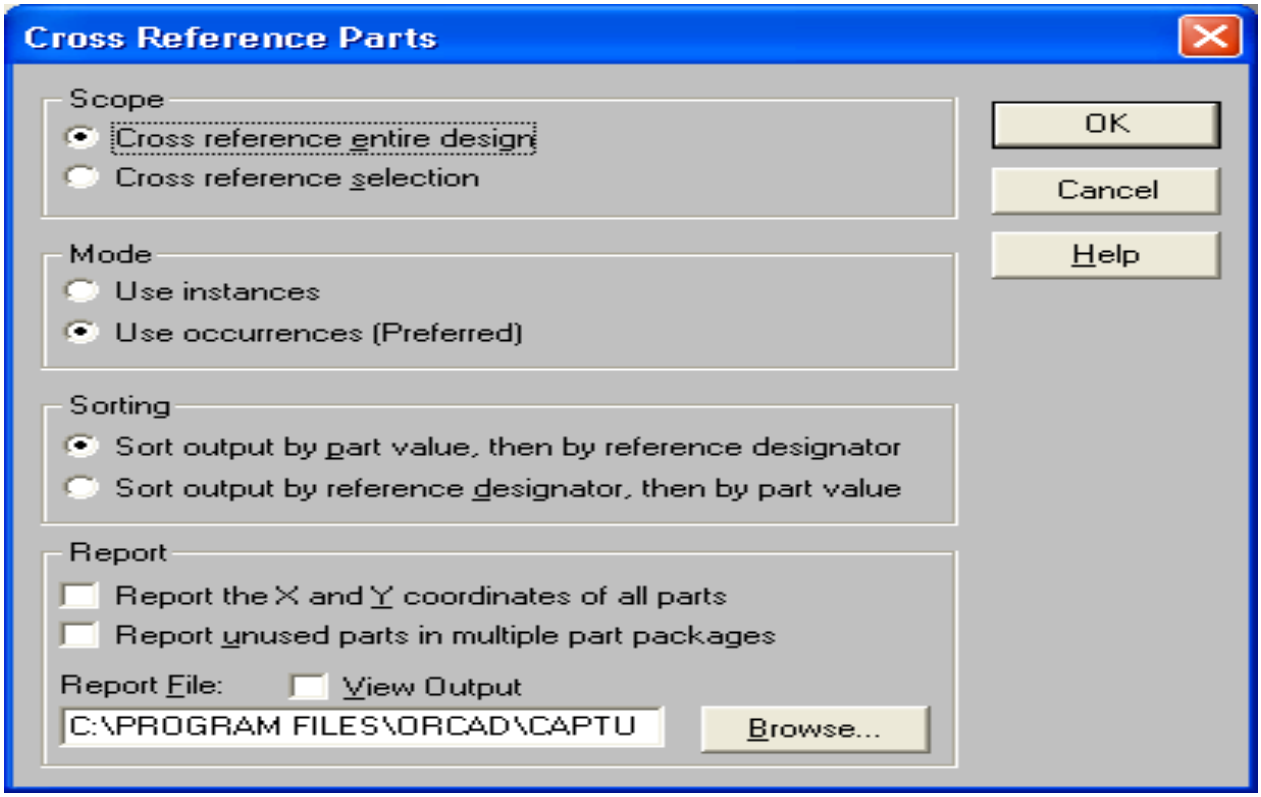


Рис. 5.43, б)

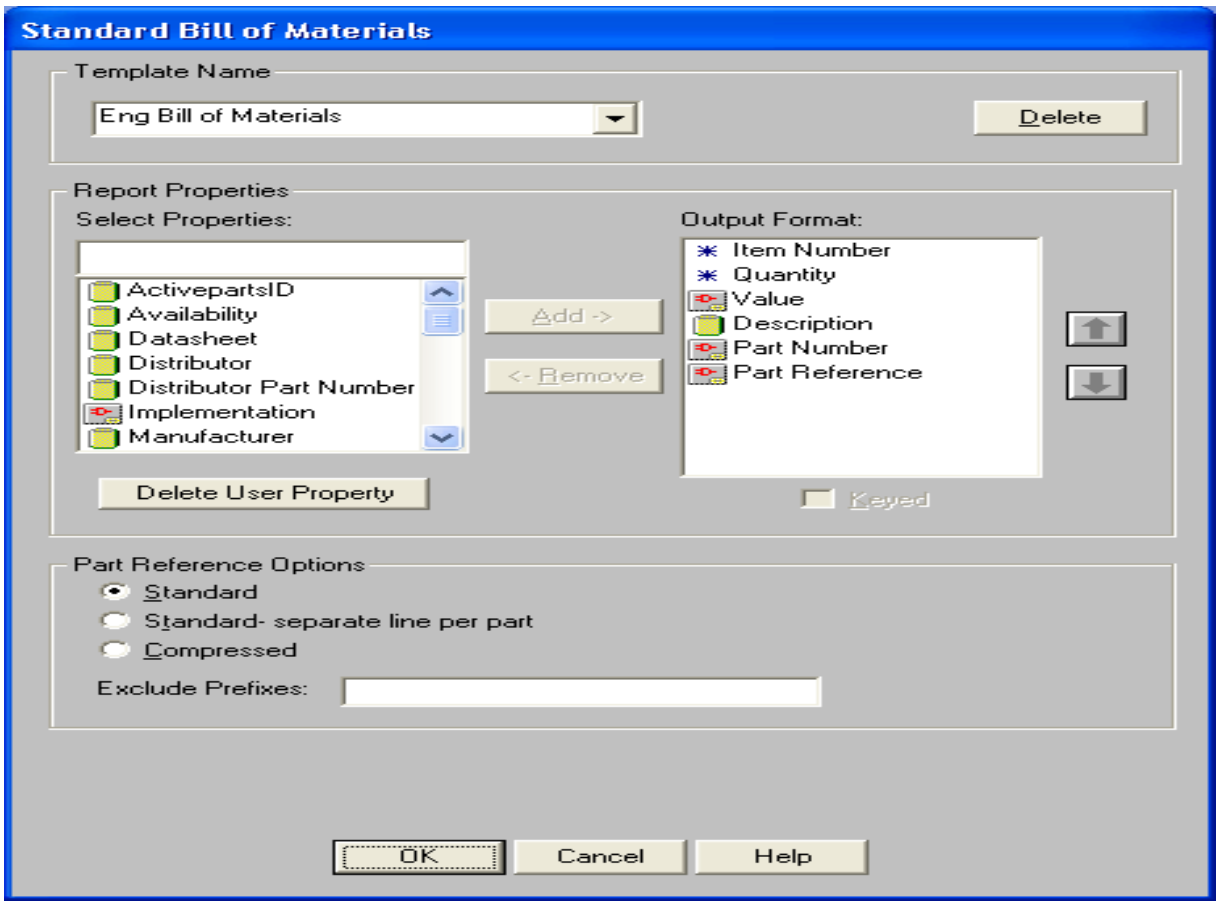


Рис. 5,43 дюйма)

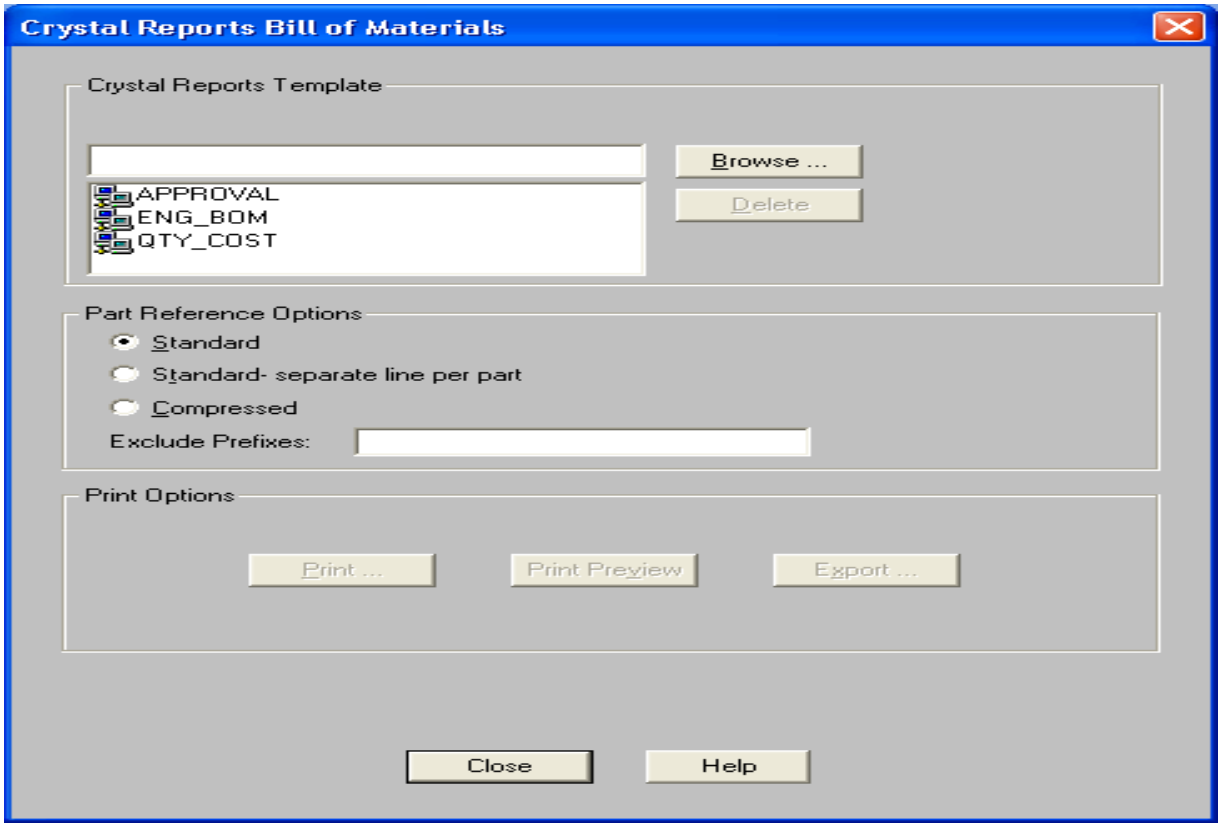


Рис. 5.43, d)

- Прикріпити файл (прикріпити файл):
  - Пов’язати файл включення зі звітом – включити у звіт додаткову інформацію, включену у файл включення;
  - Рядок зв'язаної властивості - завдання перерахування значень, що містяться у файлі Include, в одинарних лапках (див. нижче);
  - Включений файл - ім'я включного файлу;
- Файл звіту - ім'я файлу звіту (за замовчуванням це ім'я проекту з розширенням .bom);
- Переглянути Вивід - виведення на екран вихідного файлу для попереднього перегляду.

Якщо вміст першого стовпця будь-якого рядка в цьому файлі відповідає даним у властивості «Включити зв’язаний файл», тоді вміст другої частини цього рядка у файлі включення додається до файлу звіту.

Наведемо приклад витягу зі звіту про проект, складеного командою Інструменти>Список матеріалів (див. рис. 5.43, а) з використанням файлу Include (його вміст наведено вище):

Bill Of Materials September 22.2000 21:21:33 Pagel		
№п/п (Item)	**Поз. обозн. (Reference)**	**Номинал Описание компонента ГОСТ (Value)**
1	C1	1 uF Керамический конденсатор
2	R1	1k МЛТ-0.255%
3	U1	74107 Warning: The part is not in the include file
4	DD1	133LA3 4 логических элемента 2И-НЕ
		И63.088.023ТУ7

У діалоговому вікні Сервіс>Команда посилання (рис. 5.43, б) вибираються такі параметри:

- Обсяг (вибір проекту або його частини):
  - Посилання на весь проект - збірка звіт вниз все проект ;
  - Вибір літератури – підготовка звіту для обраної частини проекту;
- Сортування (сортування):
  - Сортувати вихідні дані за значенням частини, а потім за тегами посилань - сортувати рядки звіту спочатку за параметрами <Value> компонентів, а потім за тегами розташування;
  - Сортувати вихідні дані за посиланнями, потім за значеннями деталей - сортувати рядки звіту спочатку за позначеннями розташування компонентів, потім за їх параметрами <Value>;
- Звіт: _
  - Provide the X and Y координати всіх частин - завершити звіт координат X, Y розташування всіх елементів на схемі;
  - Report unused parts in multi-part packages - повний звіт інформації про невикористані частини багатокомпонентних компонентів;
- Файл звіту - ім'я файлу звіту (за замовчуванням це ім'я проекту з розширенням .xrf);

- Переглянути вихідні дані – вивести на екран вихідного файлу для попереднього перегляду.

IN діалогічний вікно команди Звіти > СНД Білл с Вибрано Materials > Standard (рис . 5.4 3 , в). наступні параметри стандарт звіт (команда доступний Просто Увімкнено Робота IN Виловити СНД):

- назву шаблону звіту зі списку або введення назви нового шаблону (для створення нових шаблонів потрібен Seagate Crystal Reports (див. веб-сайти www.orcad.com/cis/kryształ.htm , www.seagatesoftware.com , www.softline . ru) , а версії цієї програми, сумісні з OrCAD 9 не молодше 6;
- Властивості звіту (налаштування параметрів звіту):
    - Вибрані властивості (вибір параметрів, включених до звіту): наявність, таблиця даних, дистриб'ютор, номер деталі дистриб'ютора, рейтинг, схематичний шлях деталі, вихідна бібліотека, вихідний пакет, допуск, напруга;
    - Вихід Формат - список обраних параметрів;
- Keyed – вибрані параметри групуються в одному рядку звіту (див. нижче);
- Параметри посилань на деталі (вибір формат файл звіт):
    - Стандарт - стандартна форма - групування обраних параметрів в одному рядку;
    - Стандарт - окремо лінія ззаду частина - кожен елемент розміщується в окремому рядку;
    - Compressed – стиснута форма – інформація про однакові елементи розміщена в одному рядку;
    - Виключити Префікси - перелік префіксів для позначення розташування елементів, які не увійшли до звіту.

На рис. 5.44 та приклад фрагменту звіту про проект, підготовленого за допомогою команди Звіти > Специфікація CIS > Стандарт.

IN діалогічний вікно вибрано команду Reports>Bill of Materials CIS>Crystal Reports (рис . 5.43, г) наступним чином . варіанти :

- Шаблон звіту Crystal Reports – двійковий файл шаблон звіт підготовлено _ програма Crystal Reports (стандартні шаблони мають імена ENG_BOM, QTY_COST, APPROVAL);

Позиц. обозн.	Наименование	Номинал	Документ	Группа
C1	ОС К53-18	-3В 33 икФ+-2%	ОЖО.454.136 ТУ ОЖШ4201 ТУ	Конденсаторы
C2	К10-73-16.М47	2.5 мкФ	ЯАВЦ.673511.004ТУ	Конденсаторы
DA1	3ОД101АОСМ		ТТ0.336Ш2ТУП0.070.052	Микросхема аналоговая
DA2	3ОД101А ОСМ		ТТ0.336.012ТУ ПО.070.052	Микросхема аналоговая
DD1	ОСМ 537РУ8А		ВК0.347.243-08ТУ ПО.070.052	Микросхема цифровая

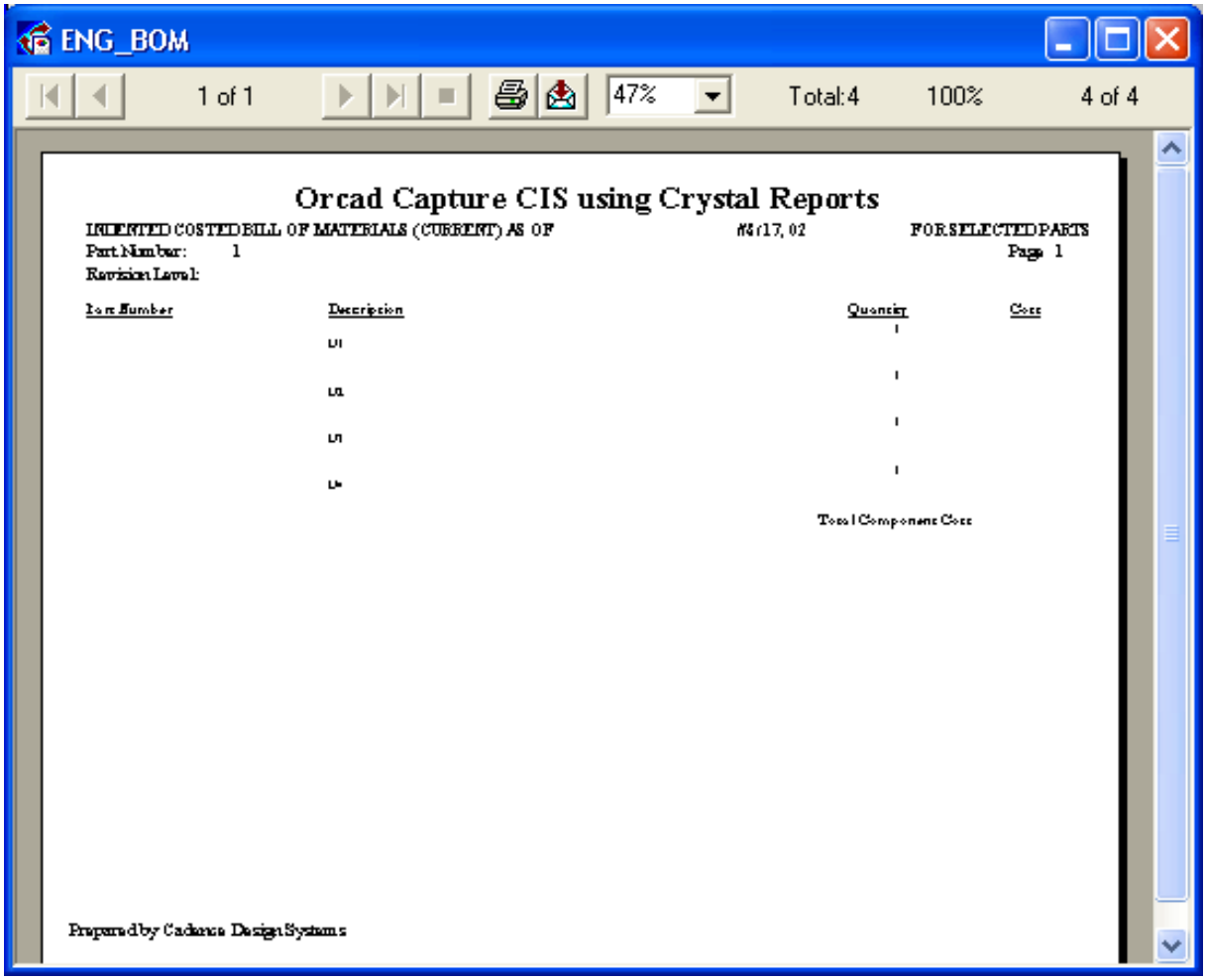


Рис. 5.44, а) Звіти про проект у стандартній формі OrCAD CIS (а), у формі Crystal Reports з шаблоном ENG_BOM (б) та шаблоном, розробленим згідно ЕСКД (в)

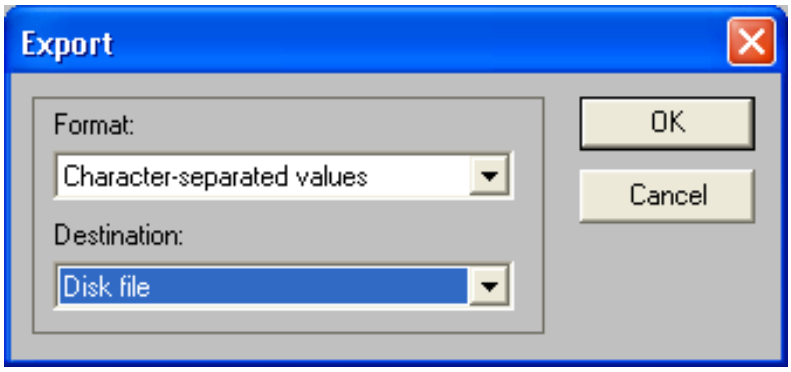


Рис . 5 . 44 , б)

Параметри посилань на деталі (вибір формат файл звіт):

- Стандартний – стандартний форма ;
- Стандарт - окремий рядок для кожної частини Привіт Він розташований УВІМКНЕНО окремо уряд ;
- Compressed – стиснута форма – інформація про однакові елементи розміщена в одному рядку;

Параметри друку (вибір формату друку для принтера):

- Print – прямий вихід на принтер;
- Print preview - попередній перегляд;
- Експорт (експорт звіту):

Формат (формат): значення, розділені символами, значення, розділені комами, Crystal Reports, формат обміну даними, Excel, HTML, Lotus 1-2-3, ODBC, розбитий на сторінки текст, стиль запису (стовпці значень), форматований текстовий формат, текст із роздільниками табуляцією , значення, розділені табуляцією, текст, документ Word для Windows;

Призначення: Дисковий файл - диск файл ; Папка Exchange - папка Exchange ; Microsoft mail - електронна пошта _

Приклад фрагмента звіту про проект, складеного за допомогою команди Reports > CIS Bill of Materials > Crystal Reports з використанням шаблону ENG_BOM, наведено на рис. 5.44, нар. Нагадуємо, що зміна шаблону звіту CIS

Bill of Materials>Standard і Crystal Reports здійснюється за допомогою програми Crystal Reports, яка поставляється окремо.

Команда File>Import Design імпортує схеми або бібліотеки з PSpice Schematics в OrCAD Capture, а також схеми, збережені у форматах EDIF 2.0.0 і PDFIF (P-CAD 8.x - 2001). У діалоговому вікні цієї команди (рис. 5.45) є три вкладки: PSpice, EDIF і PDIF.

Схеми, імпортовані з PSpice Schematics (вкладка PSpice). підходить для моделювання без додаткового редагування. На панелі Відкрити вказується ім'я вихідного файлу схеми *.sch, а на панелі Зберегти як - ім'я проекту *.obj у форматі OrCAD Capture. Крім того, на панелі MSIM.INI необхідно вказати назву файлу конфігурації *.ini програми Schematics. Коли ви вибираєте параметр «Перекласти ієрархію», на кожній сторінці створюються ієрархічні блок-схеми; коли ви вибираєте Об’єднати всі файли схем в один файл проекту, усі сторінки схем буде включено в окремий проект.

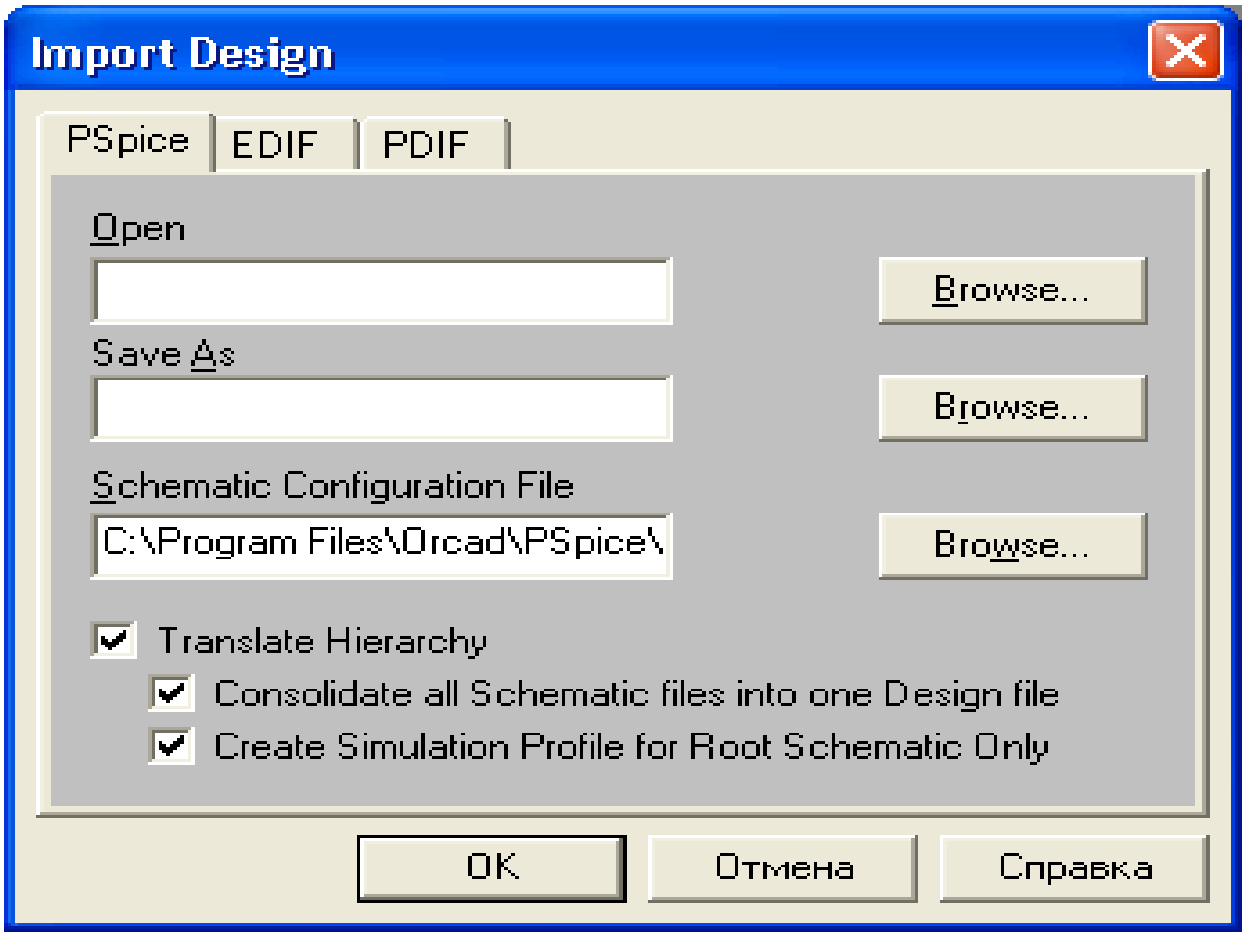


Рис. 5.45, a) Імпорт діаграм із PSpice Schematics (a) та P-CAD (b)

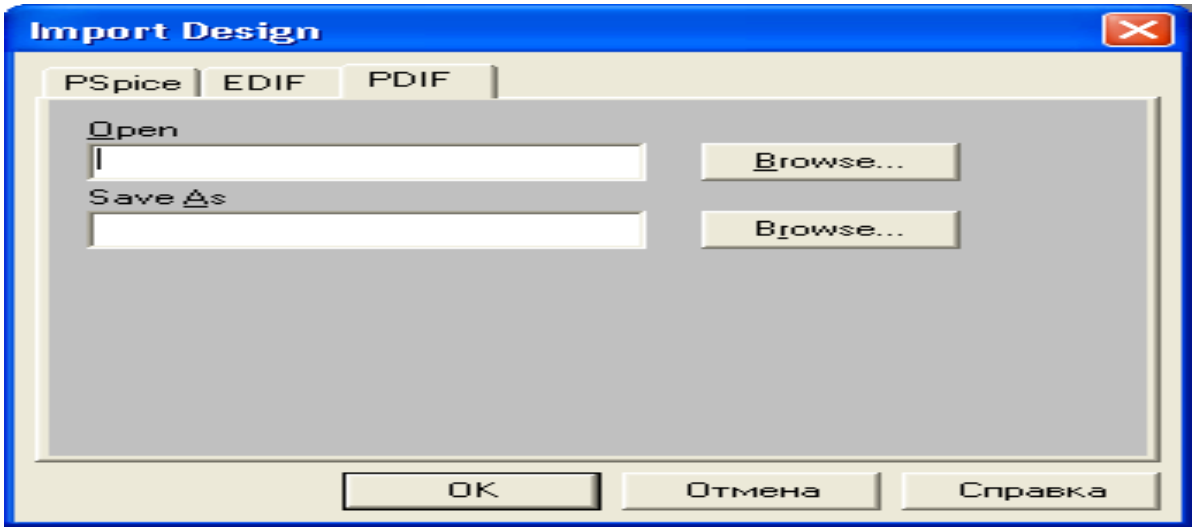


Рис. 5.45, б)

Щоб імпортувати діаграми з P-CAD 8.x, спочатку за допомогою P-CAD створіть текстовий файл *.pdf і вкажіть його назву на панелі «Відкрити» (див. рис. 5.45, б). Крім того, у файлі конфігурації перетворення pcadi.ini необхідно вказати список прив'язок імен схем P-CAD та OrCAD. Приклад такого перетворення діаграм з P-CAD 8.7 в OrCAD Capture 9.2 наведено на рис. 5.46.

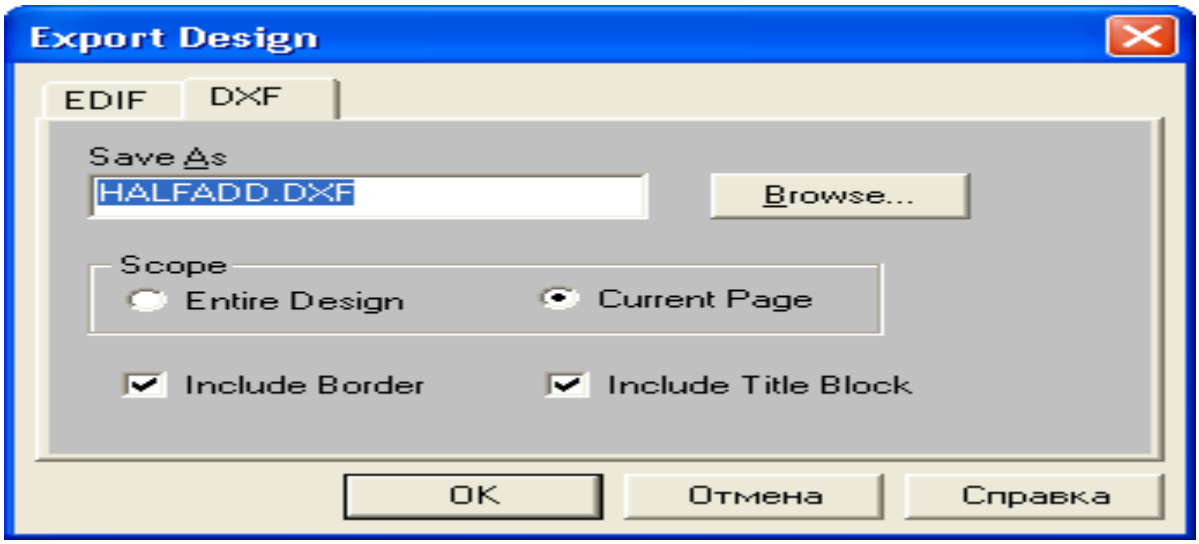


Рис. 5.46. Схематичне перетворення з P-CAD на OrCAD Capture

Схеми експортуються через «Файл» > «Експортувати проект» у форматах EDIF 2.0.0 і DXF (AutoCAD).

Експорт та імпорт параметрів компонентів, що входять до проекту, здійснюється за допомогою команди Сервіс>Експорт (Імпорт) властивостей.

РОЗДІЛ. IV. ПРИКЛАДИ РЕАЛІЗАЦІЇ РІЗНОМАНІТНИХ ФУНКЦІОНАЛЬНИХ ПРИСТРОЇВ.

6.1. Суматори.

Сумatором називається комбінаційний засіб, призначений для здійснення операції арифметичного складання чисел, представлених у вигляді двійкових чисел.

Суматори є одним з основних вузлів арифметико-логічного пристрою (АЛП). Термін «акумулятор» охоплює широкий спектр пристроїв, починаючи з найпростіших логічних схем до найскладніших цифрових пристроїв. Загальним для цих засобів є арифметичне складання чисел, представлених в двійковому вигляді.

Суматор складає два числа, працюючи за тими ж правилами, що і ми, коли виконуємо додавання в "стовпчик" (рис. 6.1).

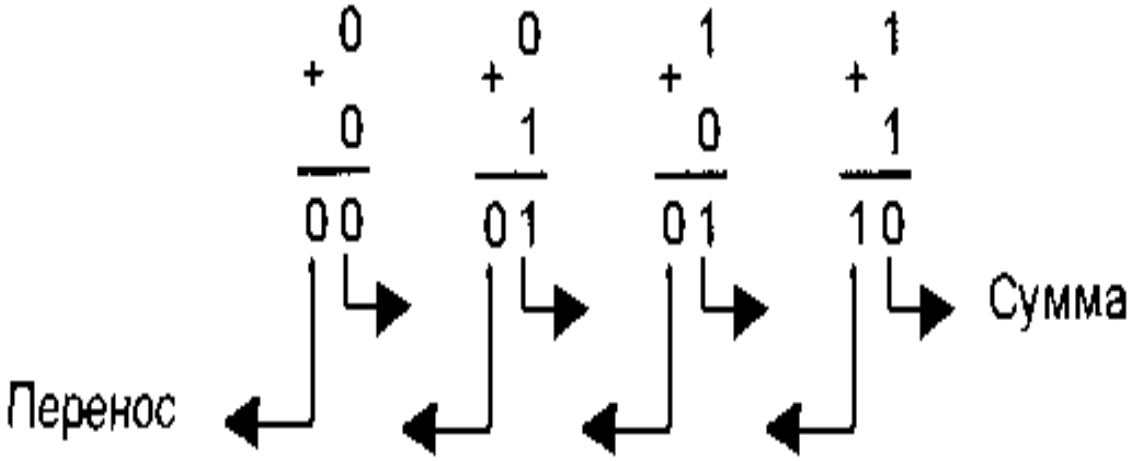


Рис. 6.1. Принцип сумування двох чисел.

При складанні двох чисел в будь-який позиційної системі в кожному розряді виробляється складання трьох чисел:

- Цифри першого доданка;
- Цифри даного розряду другого доданка;
- Цифри 1 або 0 (перенесення з сусіднього молодшого розряду).

В результаті складання для кожного розряду, виходять цифри суми і цифри I або 0 перенесення в наступний старший розряд.

Класифікація суматорів може бути виконана за різними ознаками:

1) За кількістю виходів розрізняють:
   - напівсуматори, призначені для складання двох однорозрядних кодів;
   - однорозрядні, призначені для складання двох однорозрядних кодів (на відміну від напівсуматора враховують значення перенесення із старшого розряду);
   - багаторозрядні, призначені для складання двох багаторозрядних кодів.
2) За структурою можуть бути:

- послідовні, в яких операція додавання виконується за розрядами, починаючи з молодшого;
- паралельні, в яких всі розряди вхідних кодів підсумовуються одночасно;
- комбінаційні – не мають власної пам'яті;
- накопичуючі – забезпечені власною пам'яттю, в якій зберігаються результати обчислень; при цьому кожен новий доданок додається до вже наявного в пристрої значення.

3) За способом тактування:

- синхронні;
- асинхронні.

Двійковий напівсуматор.

Півсуматор містить два входи A і B для двох складових і два виходи: S (сума) і P (перенесення). Роботу приладу відображає таблиця істинності 6. 1.

Табл. 6.1. Таблиця істинності напів суматора

Входи		Виходи	
A	B	P	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Логічна структура напівсуматора така, що стан виходу S відображає біт суми, а виходу P – біт перенесення. Це випливає і з табл. 4.1. Робота напівсуматора описується наступними рівняннями:

$$S = A\,B \vee A\,B = A + B\ ;\ P = A * B$$

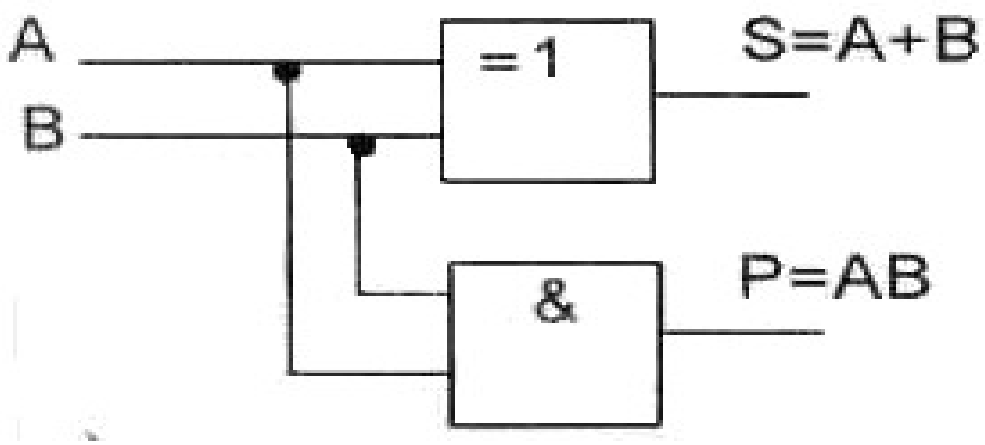


Рис. 6.2. Логічна структура напівсуматора

Повний однорозрядний двійковий суматор.

Повний однорозрядний двійковий суматор (рис. 6.3.) містить три входи: a, b для двох складових і **p** для переносу з молодшого розряду та два виходи: **S** – сума, **P** – перенесення в старший розряд. Позначають повний двійковий суматор буквами SM. Робота показана в таблиці істинності (табл. 6.2).

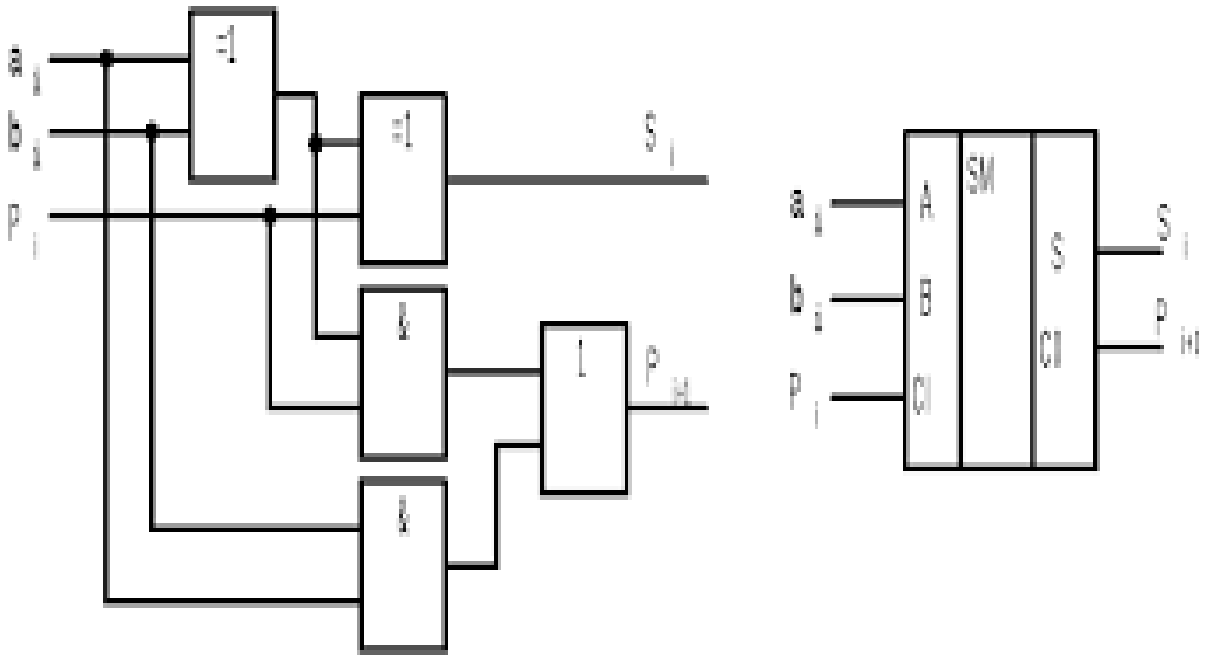


Рис 6.3. Логічна структура повного суматора

Таблиця 6.2. Таблиця істинності провного суматора

a_i	b_i	P_i	P_{i+1}	S_i
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0
1	0	1	1	0
1	1	1	1	1

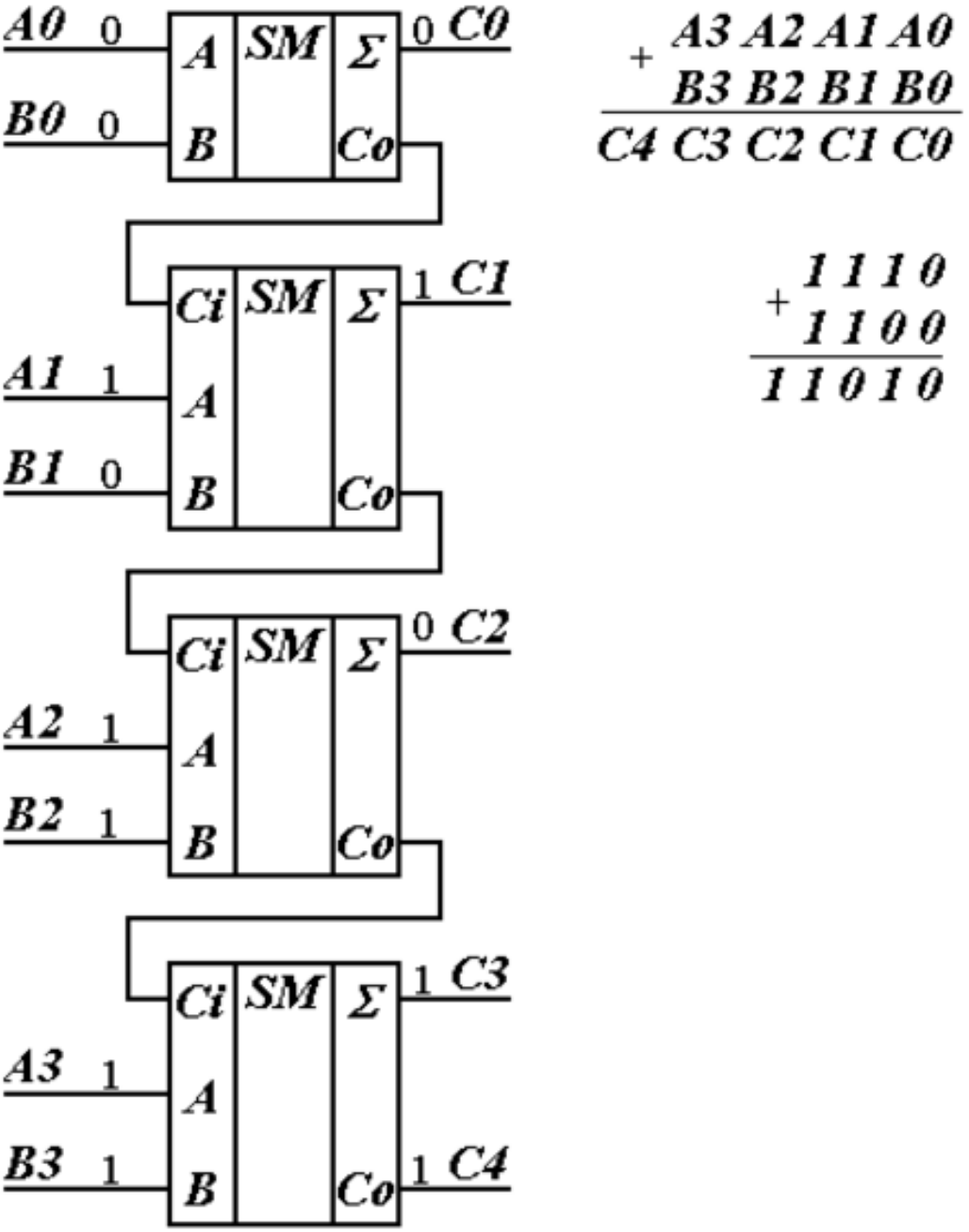


Рис. 6.4. Принцип роботи багаторозрядного суматора

Реалізація 16-розрядного суматора за допомогою САПР ALTERA MAX + plus II 10.2

Загальна принципова схема реалізації 16-розрядного суматора побудована на основі основних 3-х елементів (які надалі містять різні логічні елементи):

- Вхідний буфер;
- 16-розрядний суматор;

- Вихідний буфер.

На схемі показано:

aa[16..1] – перше 16-розрядне число;

bb[16..1] – друге 16-розрядне число;

clk – тактова частота;

yy[16..1] – результат додавання;

y17 – біт переносу.

На вхід дається глобальна тактова частота clk. Два числа aa і bb подаються на vhbufer, який розділяє 16-розрядні числа побітово. В блоці sum16 ввідбувається саме побітове додавання, і надалі в vyhbufer відбувається протилежна дія від vhbufer (Рис 6.5).

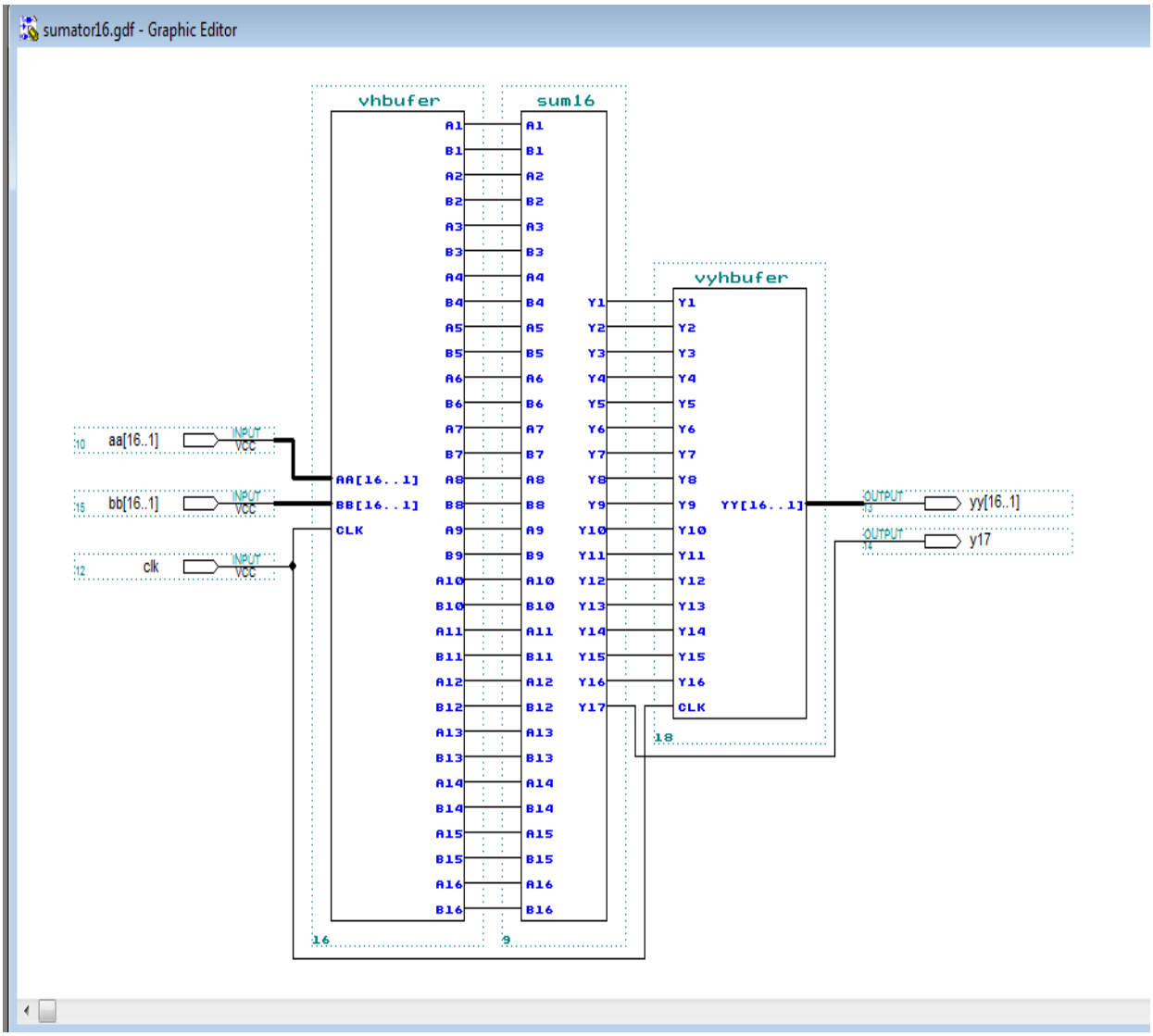


Рис. 6.5. функціональна схема 16-розрядного суматора.

Опис блоків вхідного і вихідного буферів наведений нижче. Реалізація цих блоків засновується на роботі D-тригерів (*DFF* – це синхронний D-тригер (D-FlipFlop)). Кожен біт 16-розрядного числа записуємо як кожний біт тригера.

Блоки вхідного і вихідного буферів необхідні для спрощення реалізації 16-розрядного суматора за допомогою САПР ALTERA MAX + plus II 10.2 BASELINE в графічному редакторі (замість громіздкого зображення із багатьма лініями використовуємо шини) (Рис 6.6).

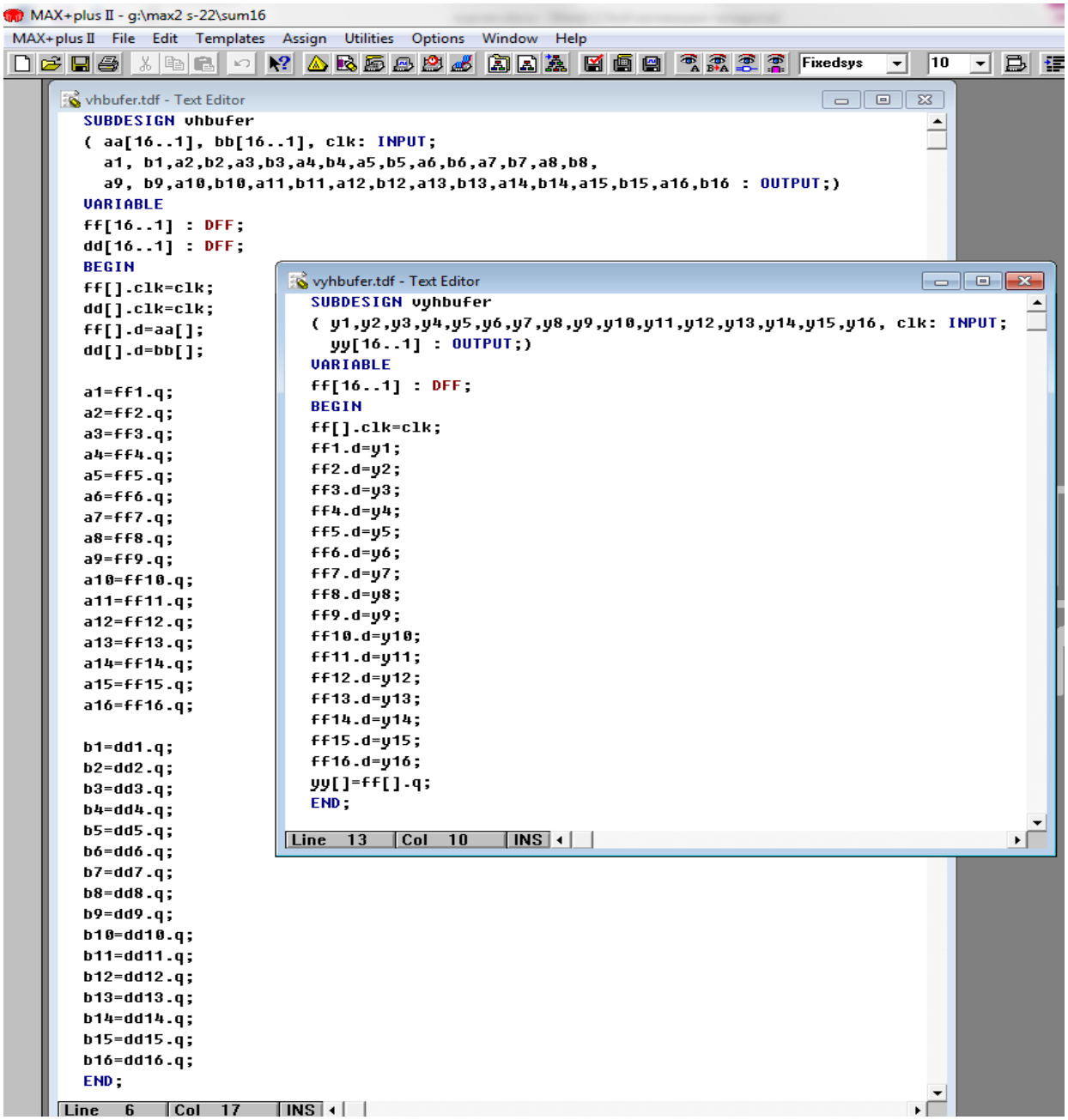


Рис. 6.6. Блоки вхідного та вихідного буферів.

Блок sum16 реалізований за допомогою суматорів та напівсуматора (Рис

4.7). Як було сказано раніше напівсуматор має два входи два виходи: сума і перенесення. Суматор має 3 входи (біт перенесення з попереднього розряду) і 2 виходи.

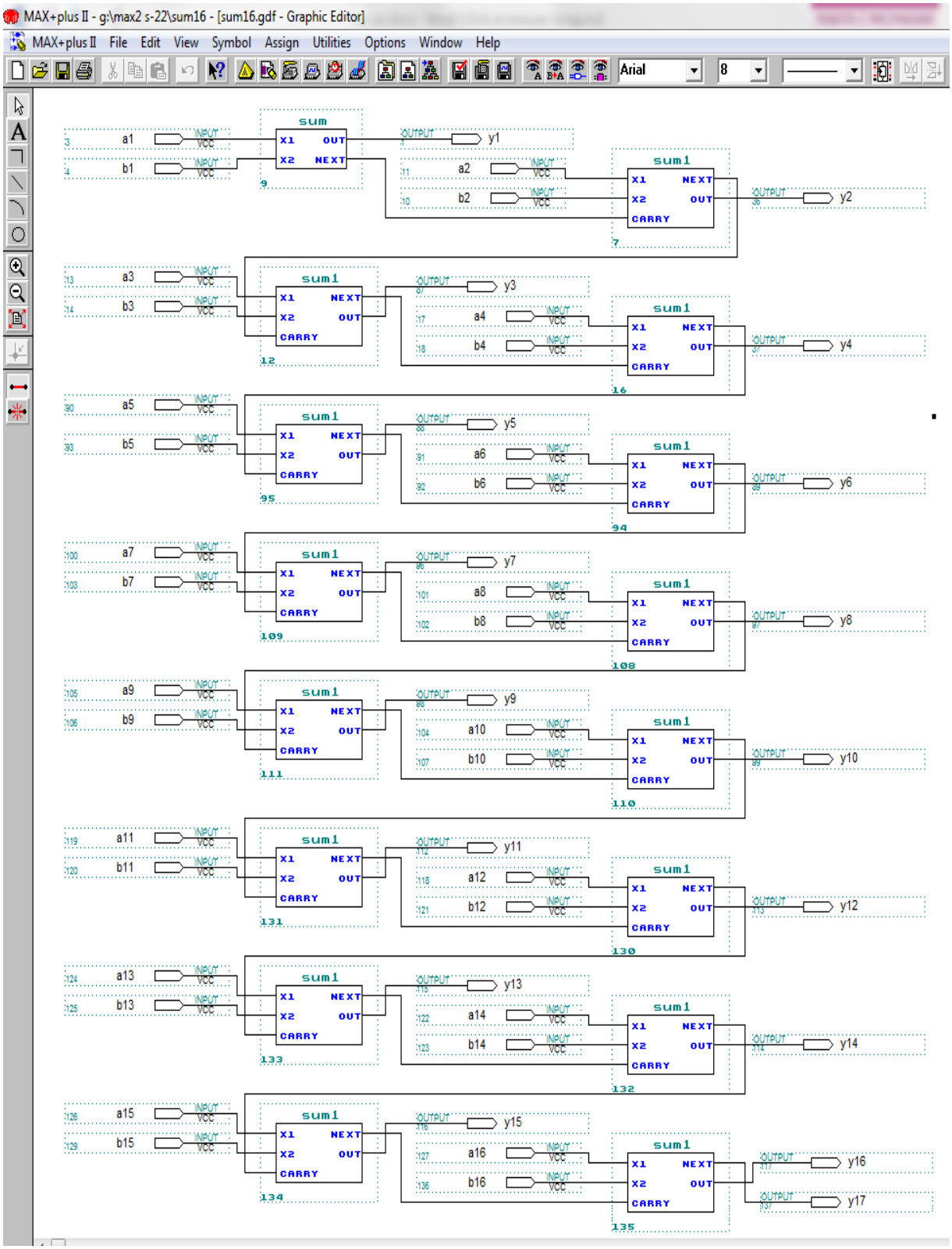


Рис. 6.7. 16-розрядний суматор

Напівсуматор працює за допомогою роботи 2-х логічних елемантів: AND

(логічне І) і XOR. Правильність роботи можна перевірити за допомогою таблиці істинності. Повний суматор можливо побудувати на основі напівсуматорів (Рис. 6.8).

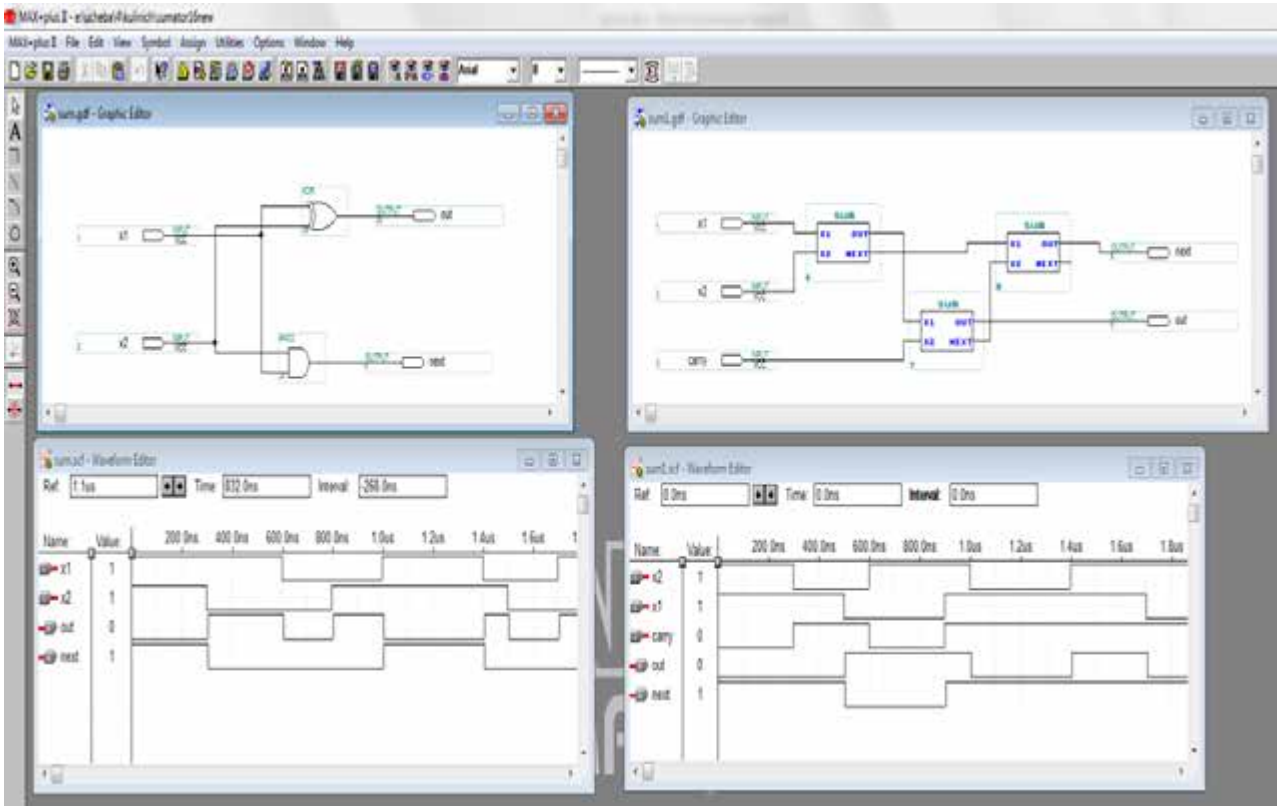


Рис. 6.8. Принципові схеми напівсуматора та повного суматора.

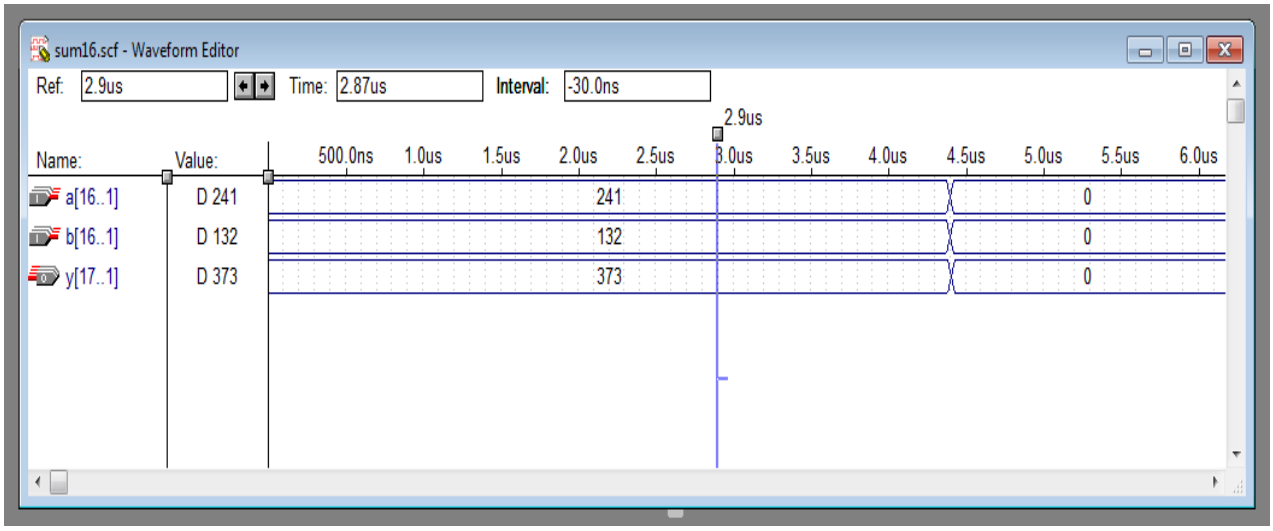


Рис. 6.9. Епюри функціонування 16-розрядного суматора.

6.2. Помножувачі.

Множення в двійковій системі виконується аналогічно множенню в десятковій згідно з наведеною схемою (Рис. 6.10):

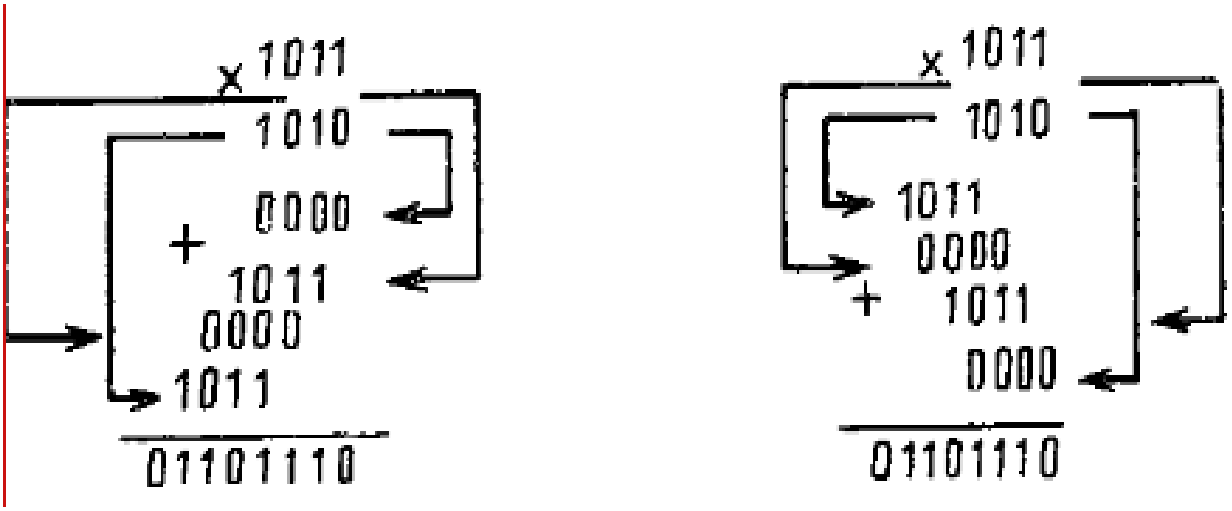


Рис. 6.10. Принцип двійкового множення.

Розглянемо структуру багатотактного помножувача, який здійснює перемноження 24-розрядних двійкових чисел. При множенні необхідно сформувати 4 рядки часткової суми. Рядки формуються за допомогою елемента І. Наприклад, X1Y0 означає логічне І між X1 і Y0. Для формування множення необхідні також суматори і пристрій забезпечує зсуву часткових сум один щодо одного, як на рис. 6. 11.

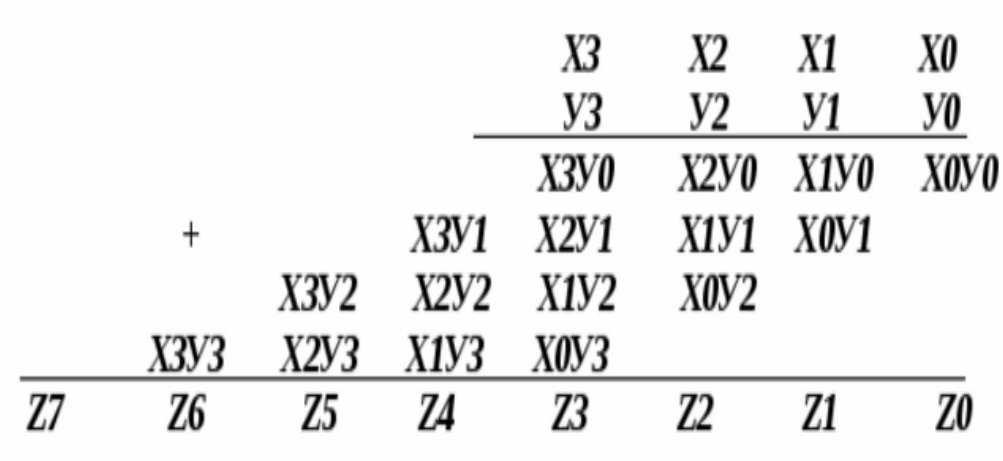


Рис. 6.11. Помножувач 4-розрядних чисел.

Для реалізації вищевказаних функцій можна застосувати наступну схему двотактного помножувача:

Як видно з малюнка багататактний помножувач складається з регістру зсуву 1 го множника, призначеного для зберігання другого множника, сигнал ТАКТ 1, який здійснює зрушення регістра зсуву 1 на 1 розряд вправо. Схеми І призначені для формування рядків часткових сум. Суматори призначені для складання часткових сум. Регістр зсуву твори призначений для зберігання множення і зсуву часткових сум на 1 розряд вправо (Рис. 6.12).

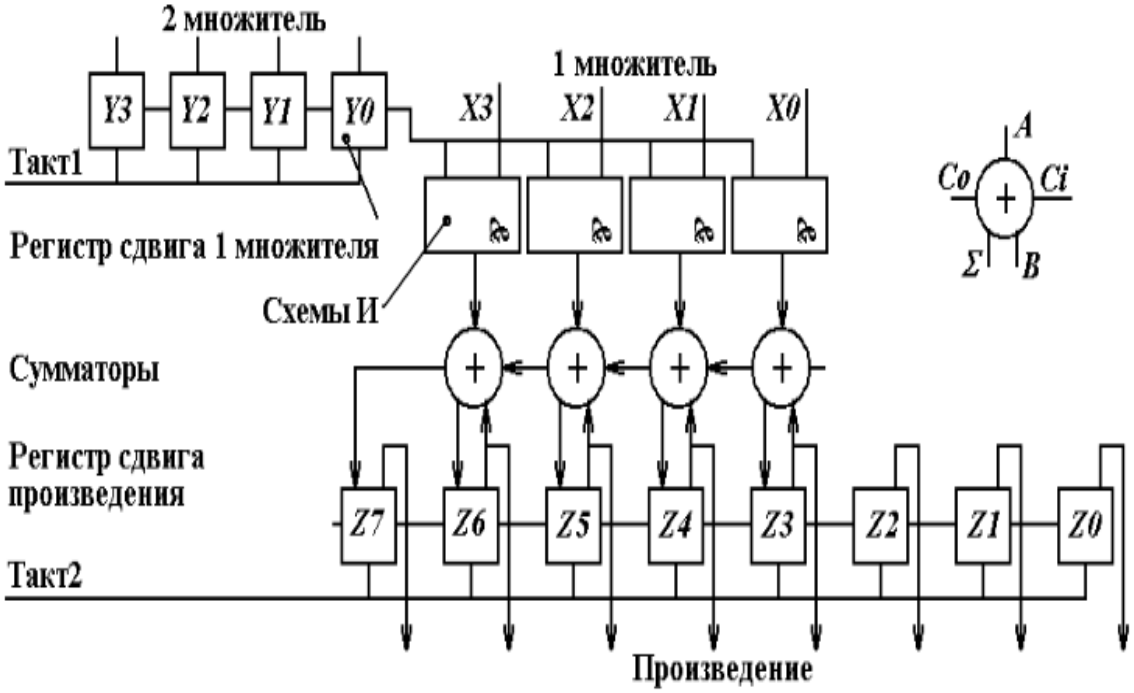


Рис. 6.12. Функціональна схема багататактного помножувача.

Реалізація помножувача 8-розрядних чисел за допомогою САПР ALTERA MAX + plus II 10.2 BASELINE

Реалізація помножувача 8-розрядних чисел у графічному вигляді досить громіздка і складна задача (надалі буде продемонстровано). Набагато простіше запрограмувати дану задачу. Це займе менше часу, зусиль, пам’яті.

Для початку варто сказати, що на вхід ми подаємо два 8-розрядні числа aa[7..0] і bb[7..0], тактову частоту clk, та шину code[3..0], за допомогою якої ми полегшимо собі задачу в написанні програмної реалізації.

У variable ми задаємо внутрішні змінні модуля, призначені для використання в розділі опису логіки. Ми будемо працювати за допомогою D-тригерів (*DFF* – це синхронний D-тригер): 2 регістри на D-тригерах, куди ми власне записуємо наші 8-розрядні числа побітно, і 3 регістри для запису проміжних результатів (проміжних множень). Рисунок 6.13.

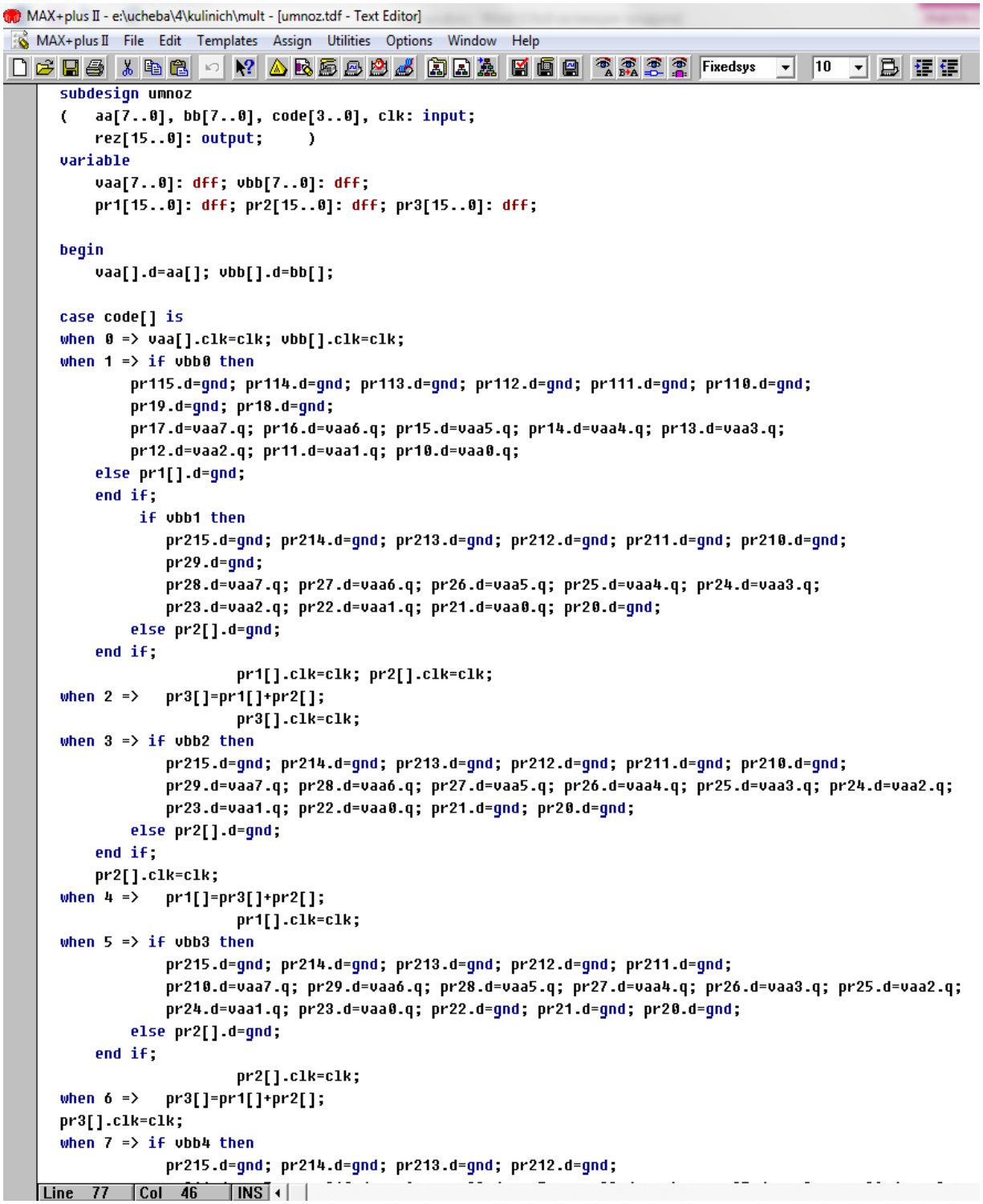
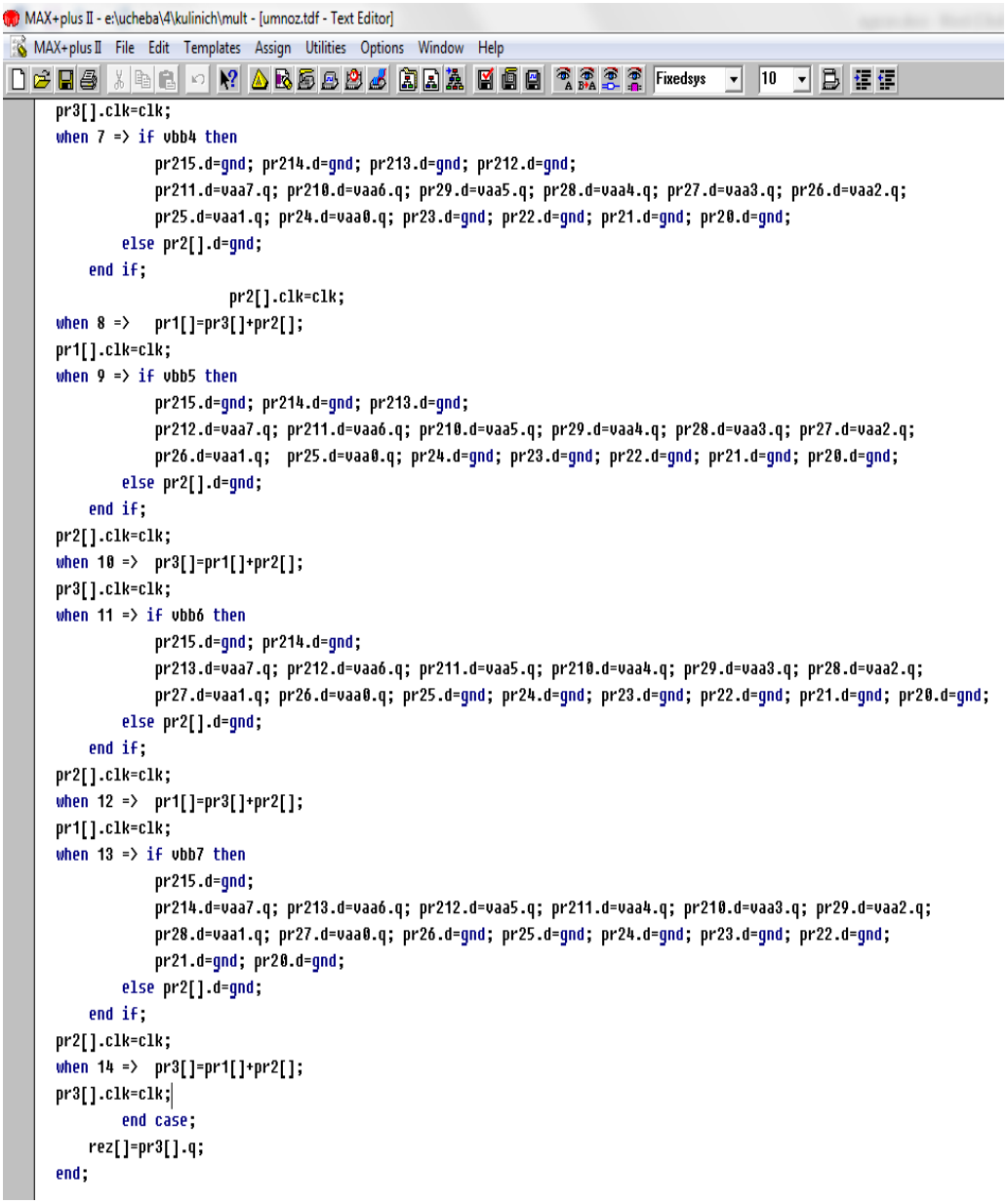


Рис. 6.13. Реалізація помножувача на мові AHDL.

За допомогою оператора *case* ми послідовно описуємо всі наші дії. На вхід дається глобальна тактова частота clk. Далі за допомогою логічного оператора if і кожного розряду 2го множника, починаючи з найменшого, ми записуємо в D-тригер результат (побітно).

Коли отримали 2 проміжних результати сумуємо їх і перезаписуємо в інший тригер. Таким чином 3-х допоміжних D-тригерів досить, щоб не заплутатися і не перевантажувати програму. Аналогічно робимо далі з кожнім розрядом 2го множника. Сумуємо всі проміжні результати і отримуємо результат (Рис. 6.14). На епюрах функціонування можемо побачити правильність роботи програми (Рис. 6.15).




```
pr3[].clk=clk;
when 7 => if vbb4 then
            pr215.d=gnd; pr214.d=gnd; pr213.d=gnd; pr212.d=gnd;
            pr211.d=vaa7.q; pr210.d=vaa6.q; pr29.d=vaa5.q; pr28.d=vaa4.q; pr27.d=vaa3.q; pr26.d=vaa2.q;
            pr25.d=vaa1.q; pr24.d=vaa0.q; pr23.d=gnd; pr22.d=gnd; pr21.d=gnd; pr20.d=gnd;
        else pr2[].d=gnd;
    end if;
                    pr2[].clk=clk;
when 8 =>   pr1[]=pr3[]+pr2[];
pr1[].clk=clk;
when 9 => if vbb5 then
            pr215.d=gnd; pr214.d=gnd; pr213.d=gnd;
            pr212.d=vaa7.q; pr211.d=vaa6.q; pr210.d=vaa5.q; pr29.d=vaa4.q; pr28.d=vaa3.q; pr27.d=vaa2.q;
            pr26.d=vaa1.q;  pr25.d=vaa0.q; pr24.d=gnd; pr23.d=gnd; pr22.d=gnd; pr21.d=gnd; pr20.d=gnd;
        else pr2[].d=gnd;
    end if;
pr2[].clk=clk;
when 10 =>  pr3[]=pr1[]+pr2[];
pr3[].clk=clk;
when 11 => if vbb6 then
            pr215.d=gnd; pr214.d=gnd;
            pr213.d=vaa7.q; pr212.d=vaa6.q; pr211.d=vaa5.q; pr210.d=vaa4.q; pr29.d=vaa3.q; pr28.d=vaa2.q;
            pr27.d=vaa1.q; pr26.d=vaa0.q; pr25.d=gnd; pr24.d=gnd; pr23.d=gnd; pr22.d=gnd; pr21.d=gnd; pr20.d=gnd;
        else pr2[].d=gnd;
    end if;
pr2[].clk=clk;
when 12 =>  pr1[]=pr3[]+pr2[];
pr1[].clk=clk;
when 13 => if vbb7 then
            pr215.d=gnd;
            pr214.d=vaa7.q; pr213.d=vaa6.q; pr212.d=vaa5.q; pr211.d=vaa4.q; pr210.d=vaa3.q; pr29.d=vaa2.q;
            pr28.d=vaa1.q; pr27.d=vaa0.q; pr26.d=gnd; pr25.d=gnd; pr24.d=gnd; pr23.d=gnd; pr22.d=gnd;
            pr21.d=gnd; pr20.d=gnd;
        else pr2[].d=gnd;
    end if;
pr2[].clk=clk;
when 14 =>  pr3[]=pr1[]+pr2[];
pr3[].clk=clk;
        end case;
    rez[]=pr3[].q;
end;
```

Рис. 6.14. Реалізація помножувача на мові AHDL.

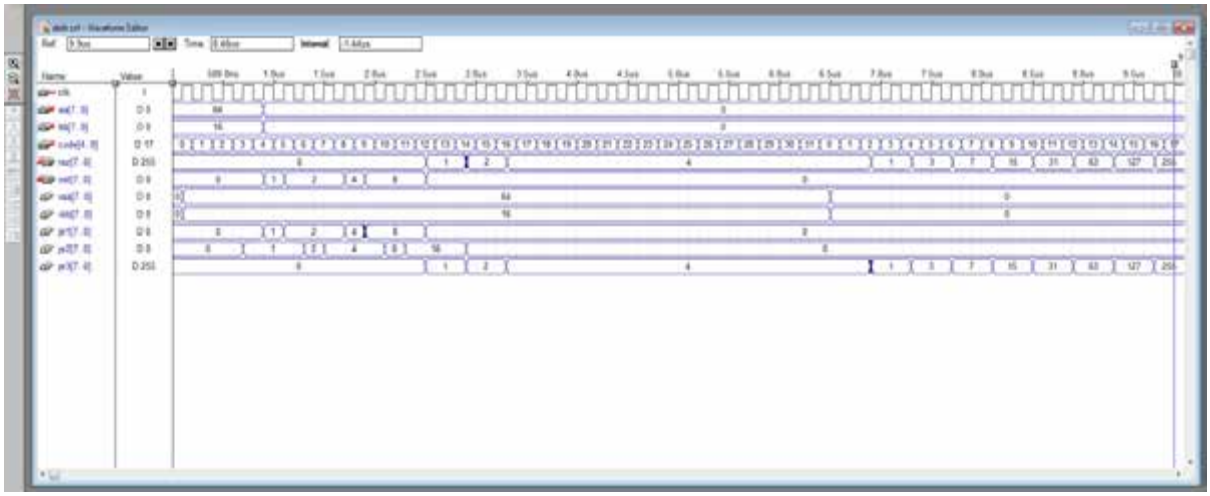


Рис. 6.15. Епюри функціонування помножувача.

Реалізація помножувача 8-розрядних чисел за допомогою графічного редактора

Інший варіант побудови помножувача набагато громіздкіший і оснований він на використанні графічного редактора. Для прикладу буде реалізовано множення 2-х 16-розрядних чисел. На (Рис. 4.16) представлено, як проект виглядає в цілому. Він складається з bufer1616255, mnsum1616, mnsum16161, bufer162, bufer1627. Але це тільки початок, Кожен з цих блоків має ряд субблоків.

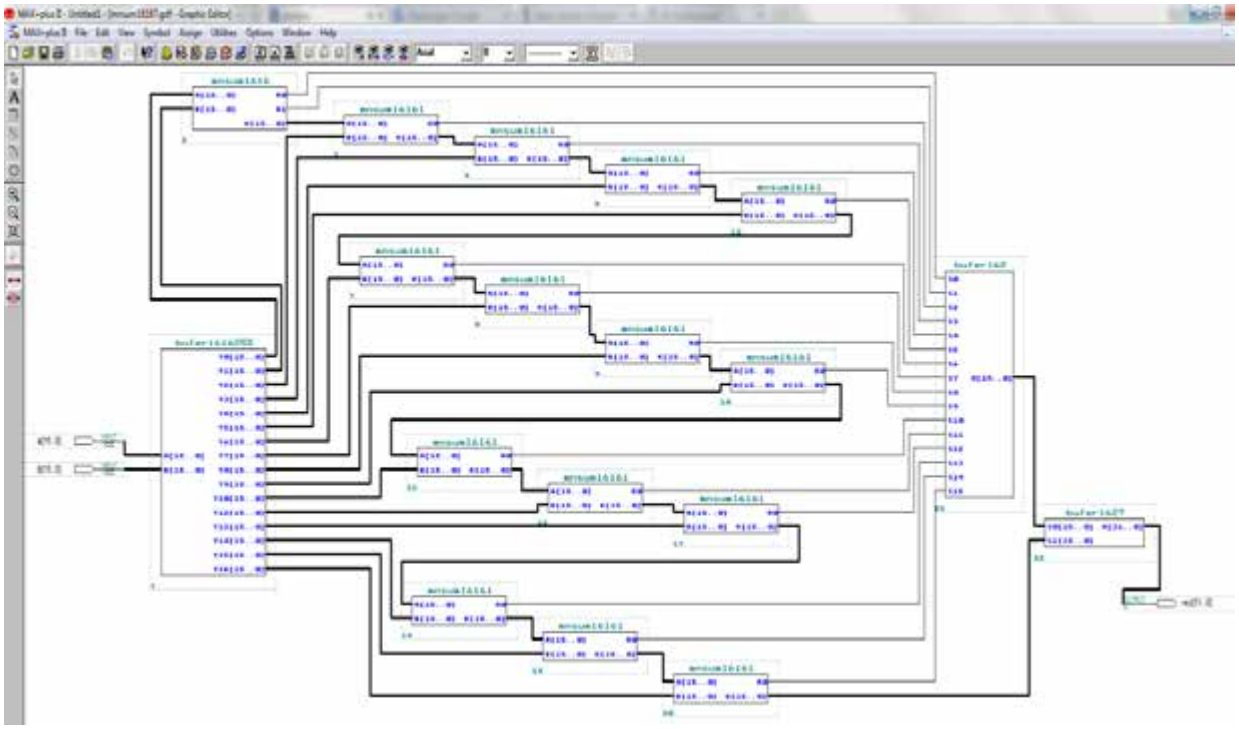


Рис. 6.16. Структурна схема помножувача.

Нижче показано, що bufer1616255 складається з bufer1614 і bufer163. Вони призначені: bufer1614 – для побітного розподілення на 16 різних виходів (та сама функція, як і в суматорі вище) і записаний як програмка, bufer163 – вона демонструє множення 1-ї комірки 2го множника на цілий рядок і містить в собі ще bufer161 і bufer162 (Рис. 6.17, 6.18).

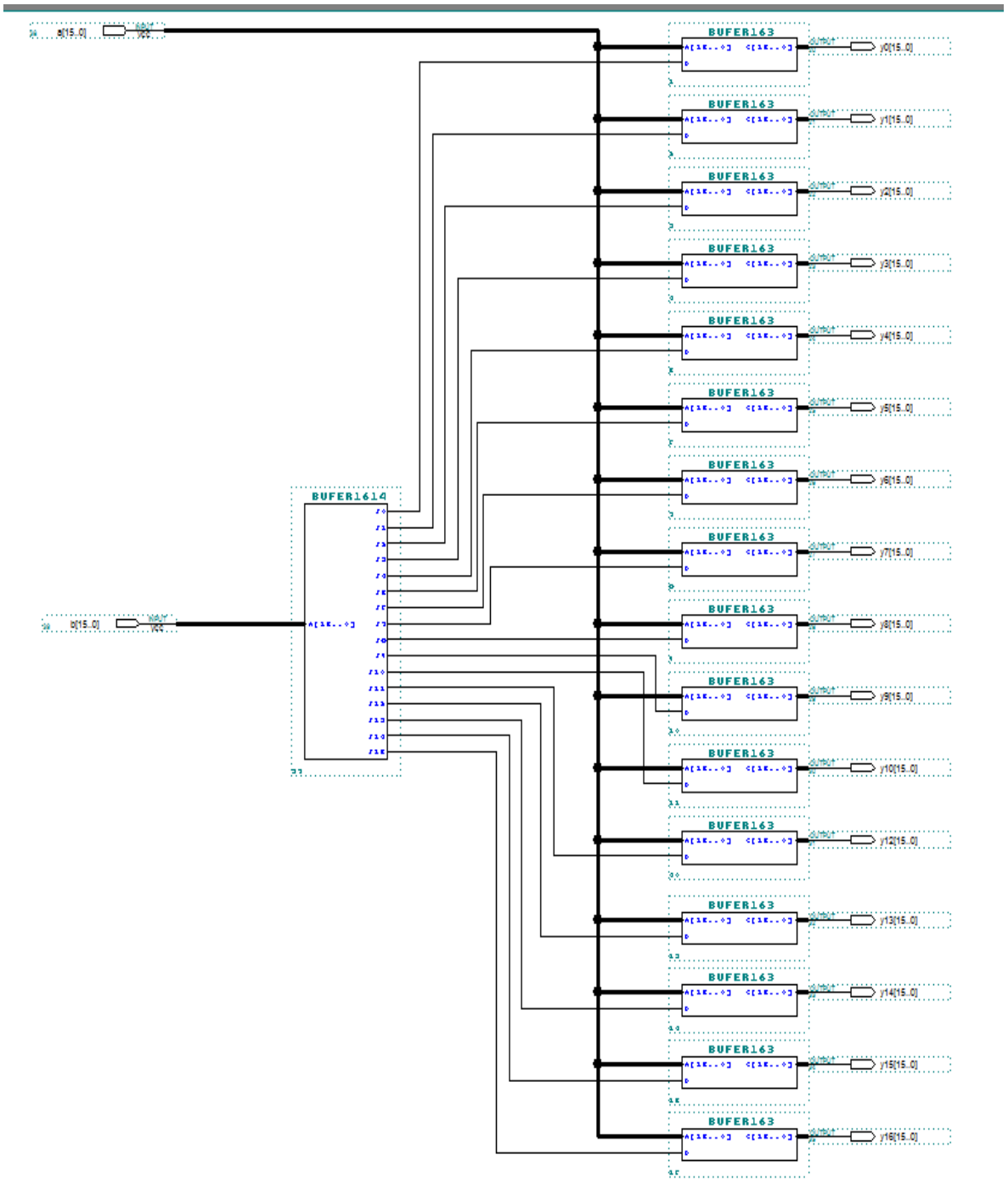


Рис. 6.17. Функціональна схема буфера.

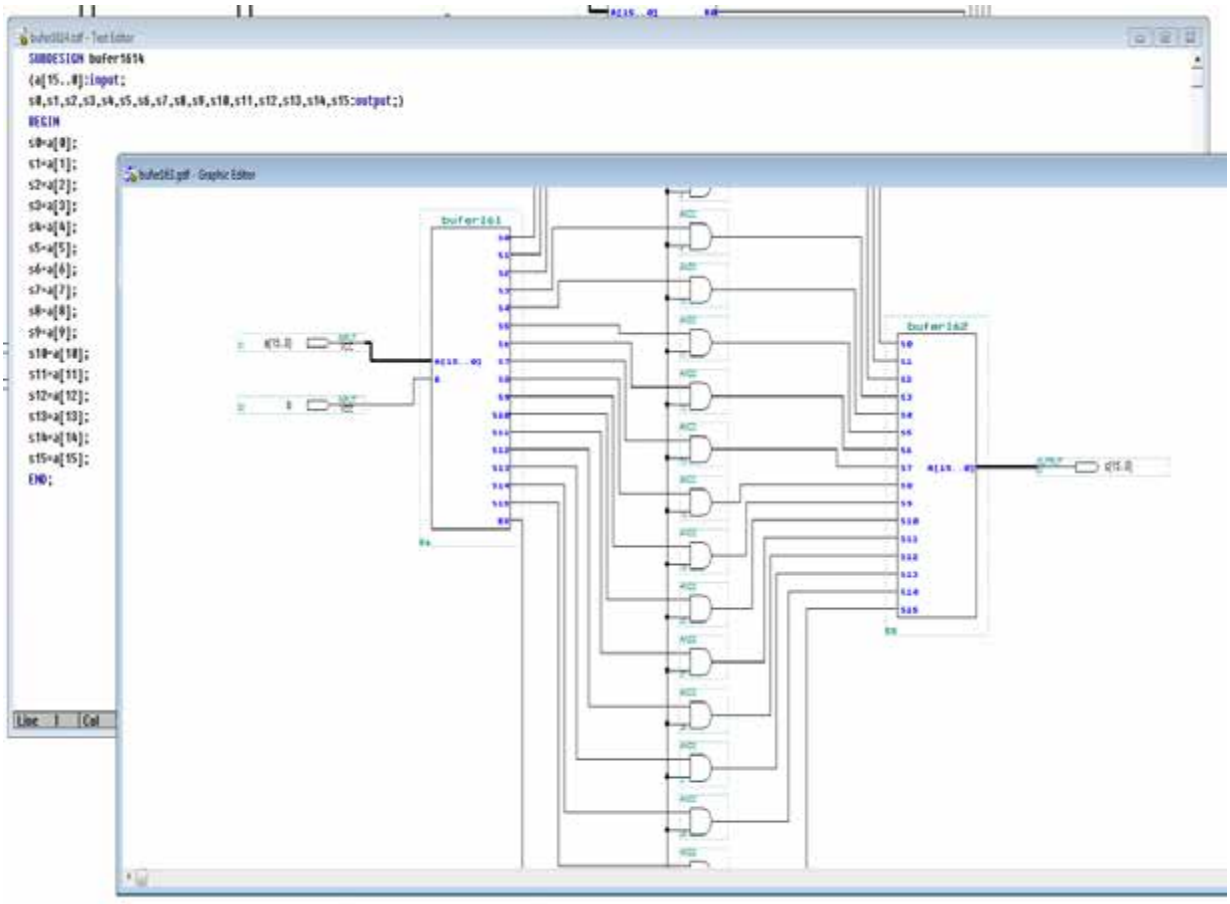


Рис. 6.18. Функціональна схема буфера.

bufer161 і bufer162 виконуєть роль вхідного і вихідного буферів (Рис. 6.19) (розділення і об’єднання).

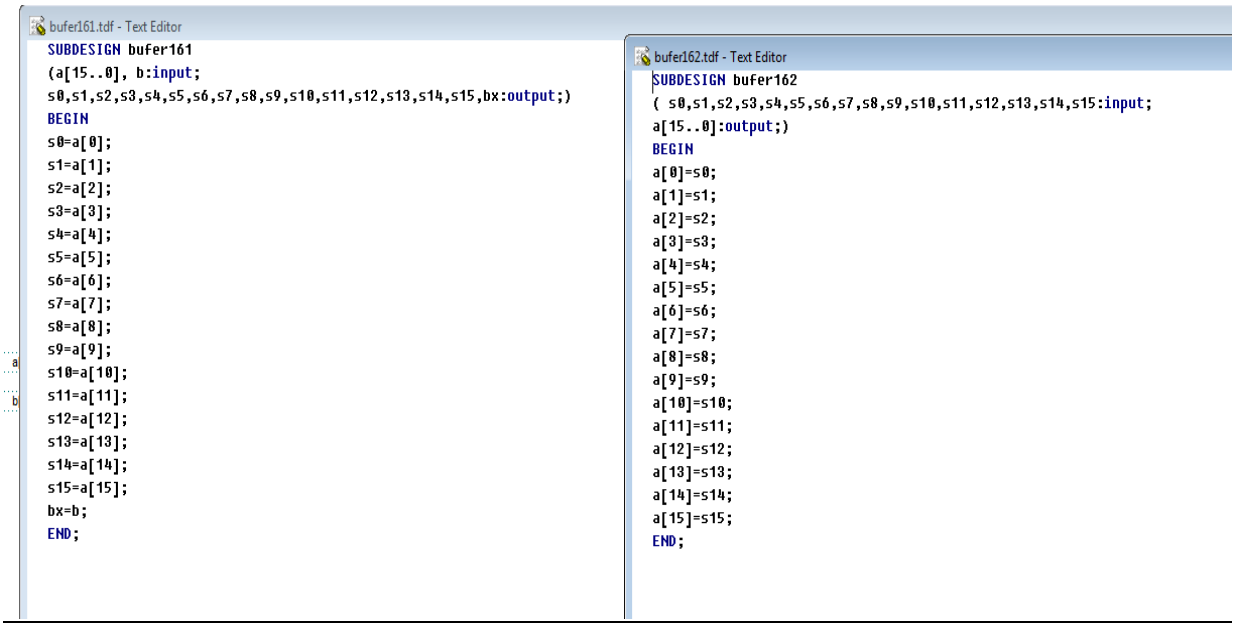


Рис. 6.19. Буфери розділення і об’єднання.

mnsum1616 і mnsum16161 здійснює додавання проміжних результатів. Побудовані вони за допомогою напівсуматорів і суматорів (Рис. 6.20).

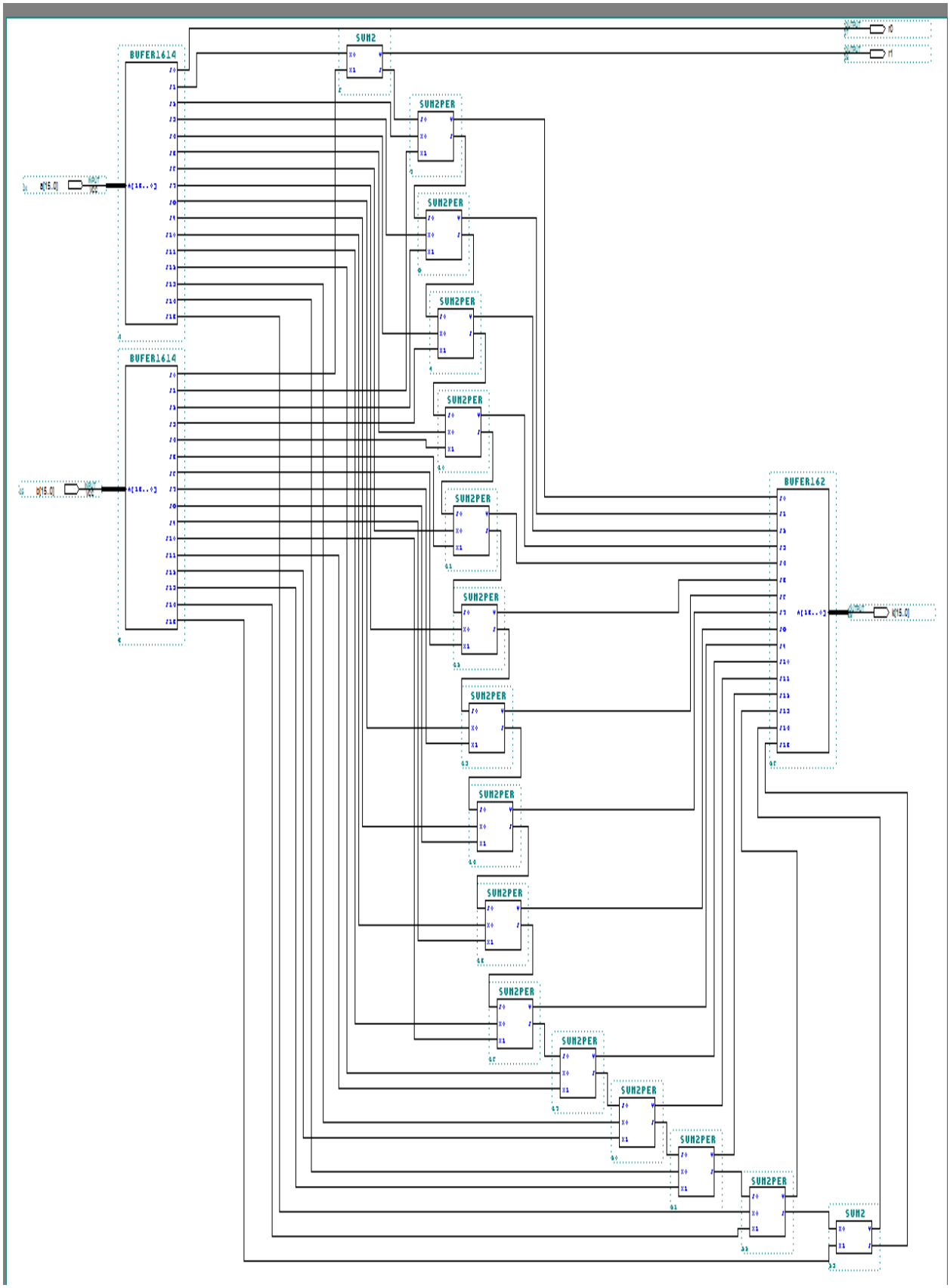


Рис. 6.20. Блок отримання проміжних результатів.

Структура bufer1627 (Рис. 6.21) отримує в результаті перемноження 32-

рзрядне число.

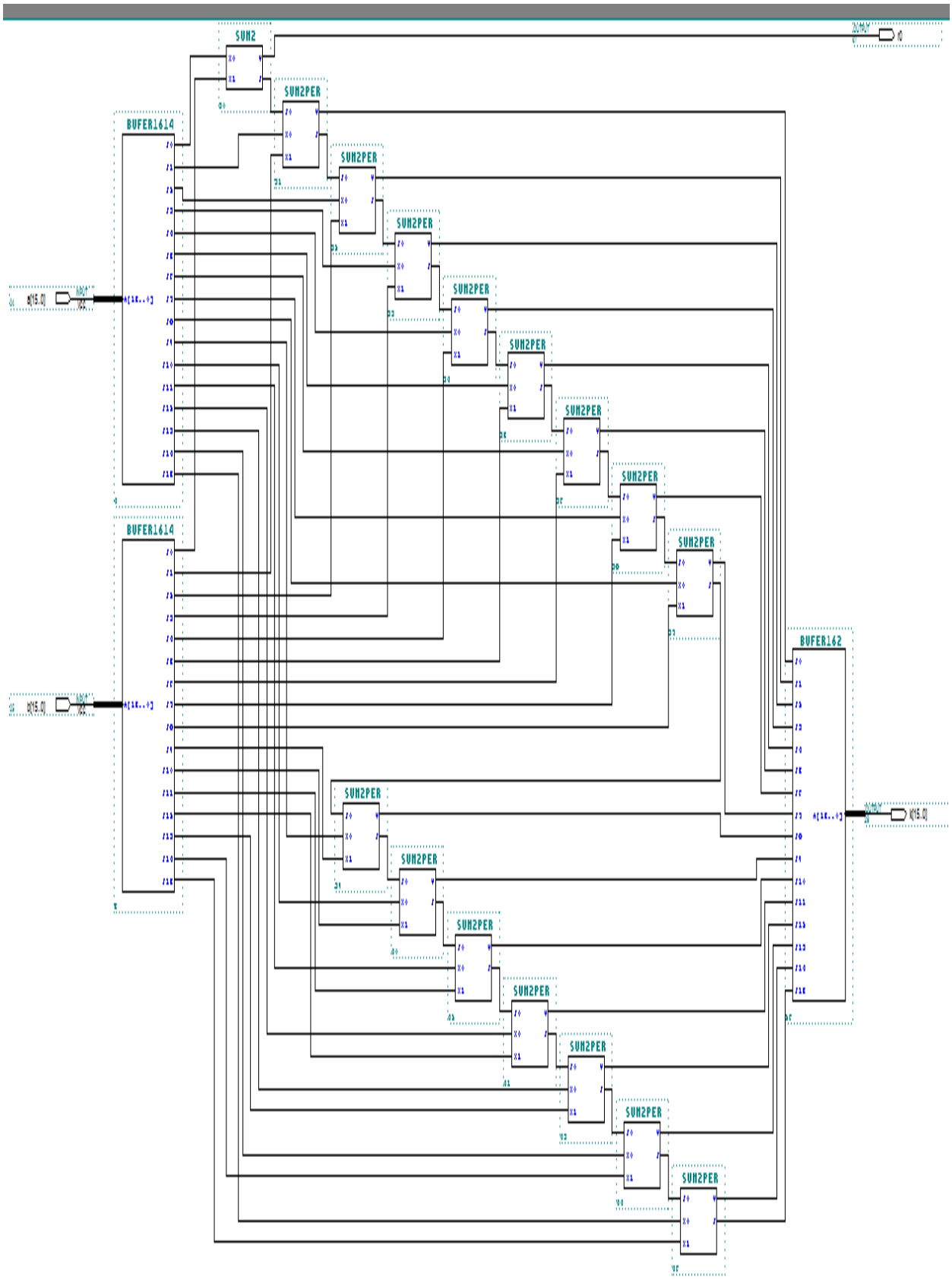


Рис. 6.21. Структурна схема пристрою.

6.3. Дільники чисел.

Пристрої множення і ділення грають велику роль в цифрових пристроях обробки інформації. Найчастіше зустрічаються завдання, в яких необхідно значення будь-якої величини множити і ділити на різні коефіцієнти, які також можуть бути змінними. В загальному випадку пристрій ділення можливо представити у вигляді функціональної схеми (Рис. 6.22).

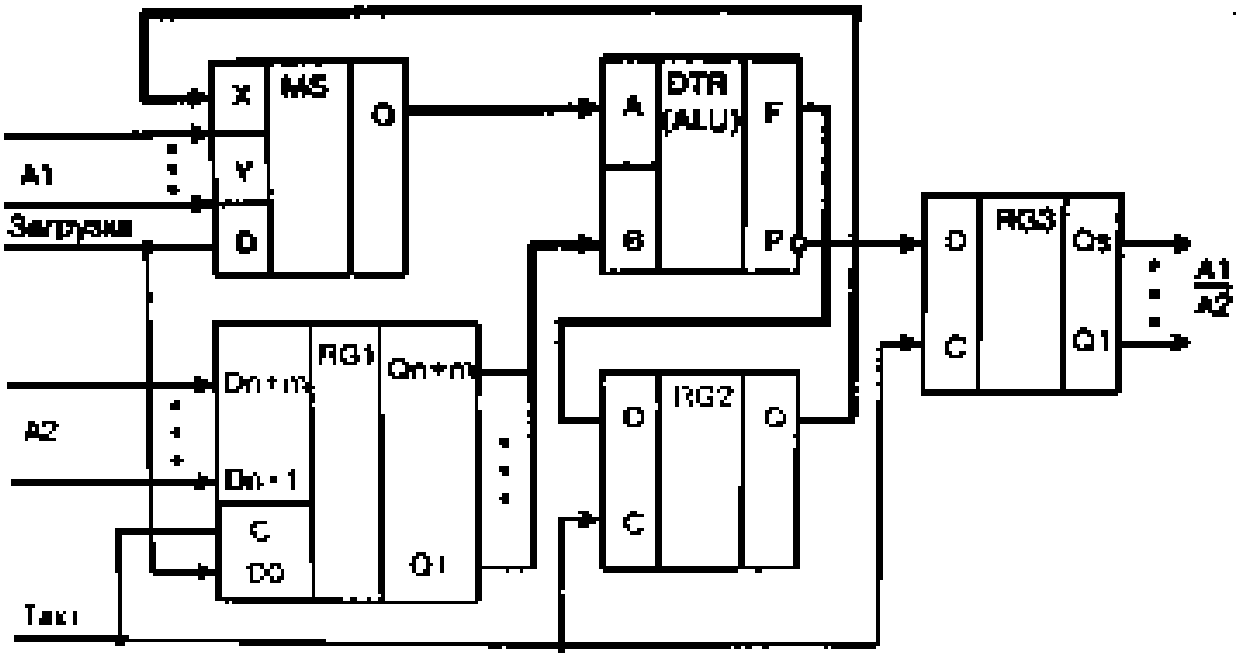


Рис. 6.22. Функціональна схема пристрою ділення

Побудова пристроїв ділення складніше за пристрої множення. Розглянемо приклад поділу чисел в двійкових кодах: число 56 (двійковий код 111000) розділимо на 8 (двійковий код 1000).

111000	1000
1000	111
1100	
1000	
1000	
1000	
0000	

Як видно, в результаті вийшло число 111 (тобто 7). Алгоритм ділення виглядає так: Зі старших розрядів діленого віднімається дільник, якщо різниця більше нуля, в результат заноситься «0», дільник зсувається на один розряд вниз (якщо різниця менше нуля, в результат заноситься «0»). В принципі (на відміну від множення), цей процес може тривати нескінченно (згадаємо нескінченні дробу).

На рис. 6.22 показана функціональна схема пристрою ділення. Основним елементом його є віднімаючий пристрій DTR. Мультиплексор MS є перемикачем з двох входів на один вихід. При подачі імпульсу «Завантаження» мультиплексор MS включається в положення, коли на його вихід надходить ділене A1, тому ділене надходить на вхід A віднімаючого пристрою DTR (в решту часу на вихід мультиплексора MS надходить різниця з регістра зберігання RG2). Одночасно в регістр RG1 по старшим розрядам записується цілитель A2, який надходить на входи B віднімаючого пристрою DTR.

На виході F віднімаючого пристрою DTR утворюється різниця. Якщо ця різниця більше нуля, то на виході P знаходиться «1», яка по тактовому імпульсу заноситься в регістр RG3. Якщо є переповнення (різниця менше нуля), то на виході P знаходиться «0», який і заноситься в регістр RG3. З кожним тактом роботи послідовно зсувається дільник A2 на один розряд вниз в регістрі RG1, а результат на один розряд вгору в регістрі RG3. Число тактів роботи визначається розрядністю результату, але при цьому потрібно пам'ятати, що розрядність регістрів RG1, RG2 і віднімаючого пристрою DTR визначається як сума числа розрядів діленого і числа розрядів результату.

Реалізація 8-розрядного дільника двійкових чисел за допомогою САПР ALTERA MAX + plus II 10.2 BASELINE

Реалізація 8-розрядного дільника чисел реалізовано в текстовому редакторі Text Editor File. Для початку варто сказати, що на вхід ми подаємо 2 8-розрядні числа aa[7..0] і bb[7..0], тактову частоту clk, та шину code[4..0], за допомогою якої ми полегшимо собі задачу в написанні програмної реалізації.

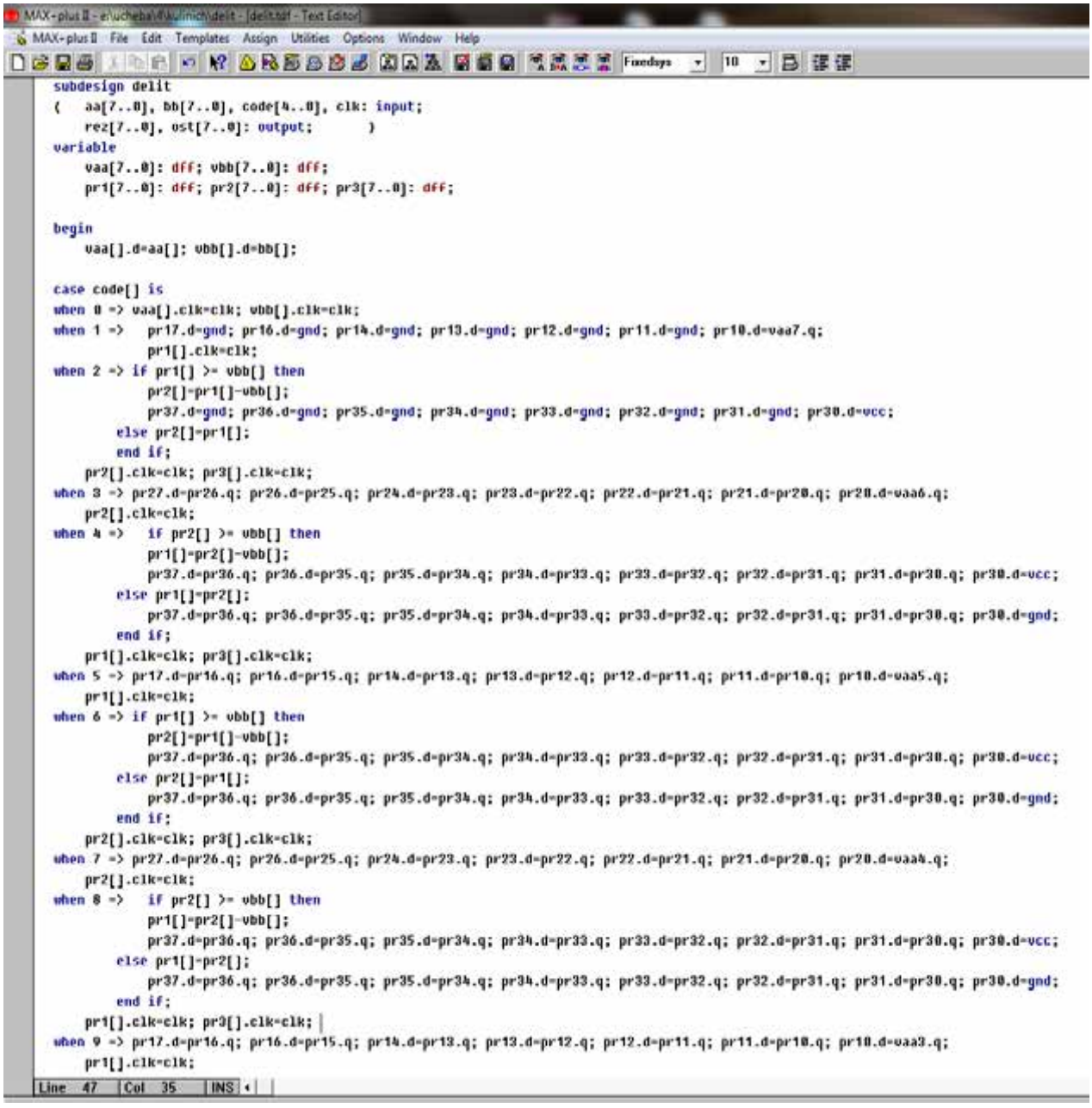
У розділі variable ми задаємо внутрішні змінні модуля, призначені для використання в розділі опису логіки. Ми будемо працювати за допомогою D-тригерів (*DFF* – це синхронний D-тригер), на основі яких будуються регістри: 2 D-тригери куди ми власне записуємо наші 8-розрядні числа побітно, і 3 для запису проміжних результатів (проміжних множень). В кінці роботи програми

ми отримаємо 2 значення: це сам наш результат rez[7..0] и остача від ділення ost[7..0].

В програмі використовуються такі позначення, як gnd та vcc:

- gnd - це ground (земля) - точка нульового потенціалу мікросхеми;
- vcc - это «+» живлення мікросхеми відносно gnd.

За допомогою оператора case ми послідовно описуємо всі наші дії. 3-х допоміжних регістрів досить, щоб не заплутатися і не перевантажувати програму. На епюрах функціонування можемо побачити правильність роботи програми (Рис. 6.23, 6.24,64.25).




```
subdesign delit
(   aa[7..0], bb[7..0], code[4..0], clk: input;
    rez[7..0], ost[7..0]: output;       )
variable
    vaa[7..0]: dff; vbb[7..0]: dff;
    pr1[7..0]: dff; pr2[7..0]: dff; pr3[7..0]: dff;

begin
    vaa[].d=aa[]; vbb[].d=bb[];

case code[] is
when 0 => vaa[].clk=clk; vbb[].clk=clk;
when 1 =>   pr17.d=gnd; pr16.d=gnd; pr14.d=gnd; pr13.d=gnd; pr12.d=gnd; pr11.d=gnd; pr10.d=vaa7.q;
            pr1[].clk=clk;
when 2 => if pr1[] >= vbb[] then
            pr2[]=pr1[]-vbb[];
            pr37.d=gnd; pr36.d=gnd; pr35.d=gnd; pr34.d=gnd; pr33.d=gnd; pr32.d=gnd; pr31.d=gnd; pr30.d=vcc;
        else pr2[]=pr1[];
        end if;
    pr2[].clk=clk; pr3[].clk=clk;
when 3 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q; pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa6.q;
    pr2[].clk=clk;
when 4 =>   if pr2[] >= vbb[] then
            pr1[]=pr2[]-vbb[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
        else pr1[]=pr2[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
        end if;
    pr1[].clk=clk; pr3[].clk=clk;
when 5 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q; pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa5.q;
    pr1[].clk=clk;
when 6 => if pr1[] >= vbb[] then
            pr2[]=pr1[]-vbb[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
        else pr2[]=pr1[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
        end if;
    pr2[].clk=clk; pr3[].clk=clk;
when 7 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q; pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa4.q;
    pr2[].clk=clk;
when 8 =>   if pr2[] >= vbb[] then
            pr1[]=pr2[]-vbb[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
        else pr1[]=pr2[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
        end if;
    pr1[].clk=clk; pr3[].clk=clk;
when 9 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q; pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa3.q;
    pr1[].clk=clk;
```

Рис 6. 23. Дільник чисел

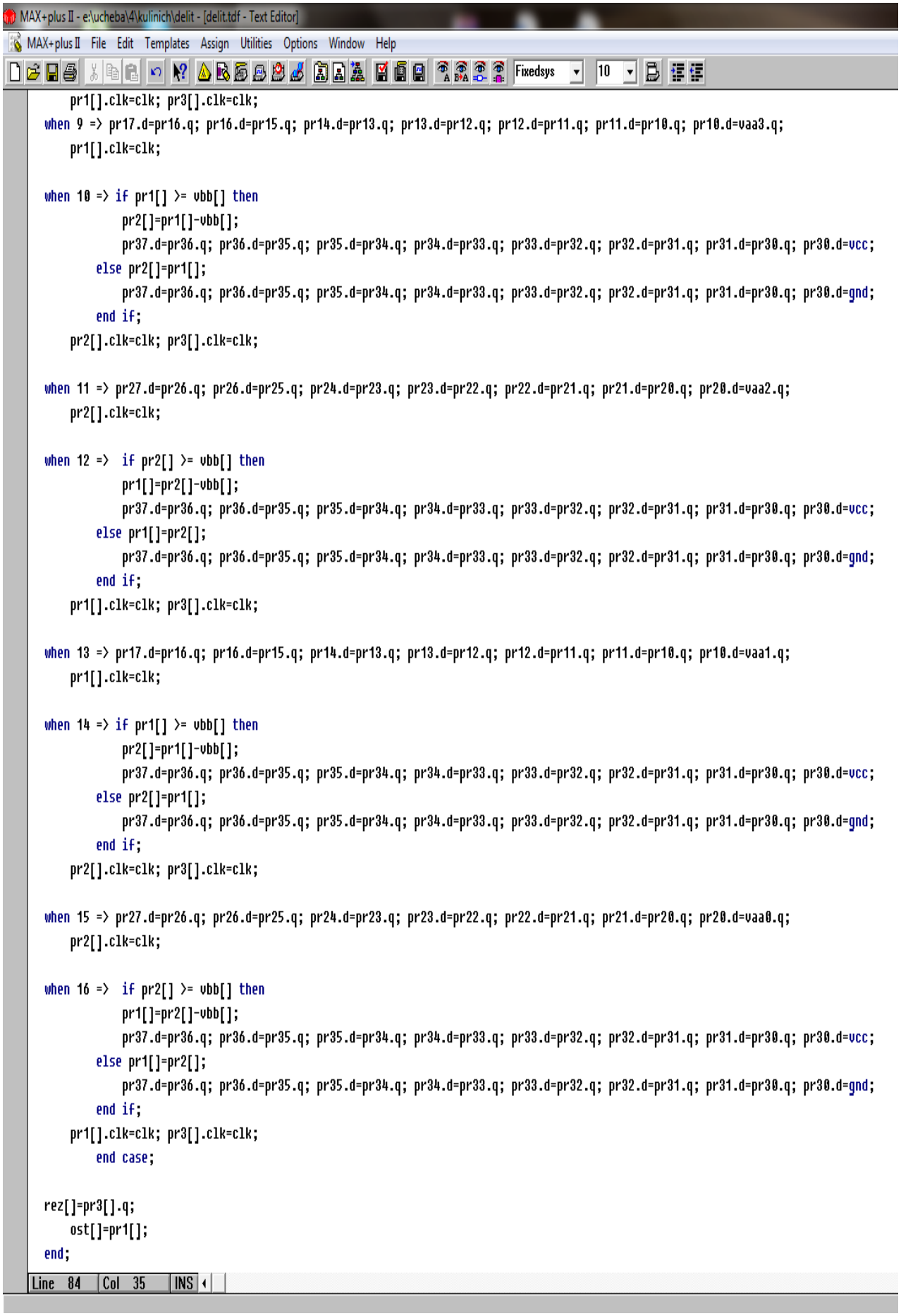


Рис 6. 24. Дільник чисел (продовження)

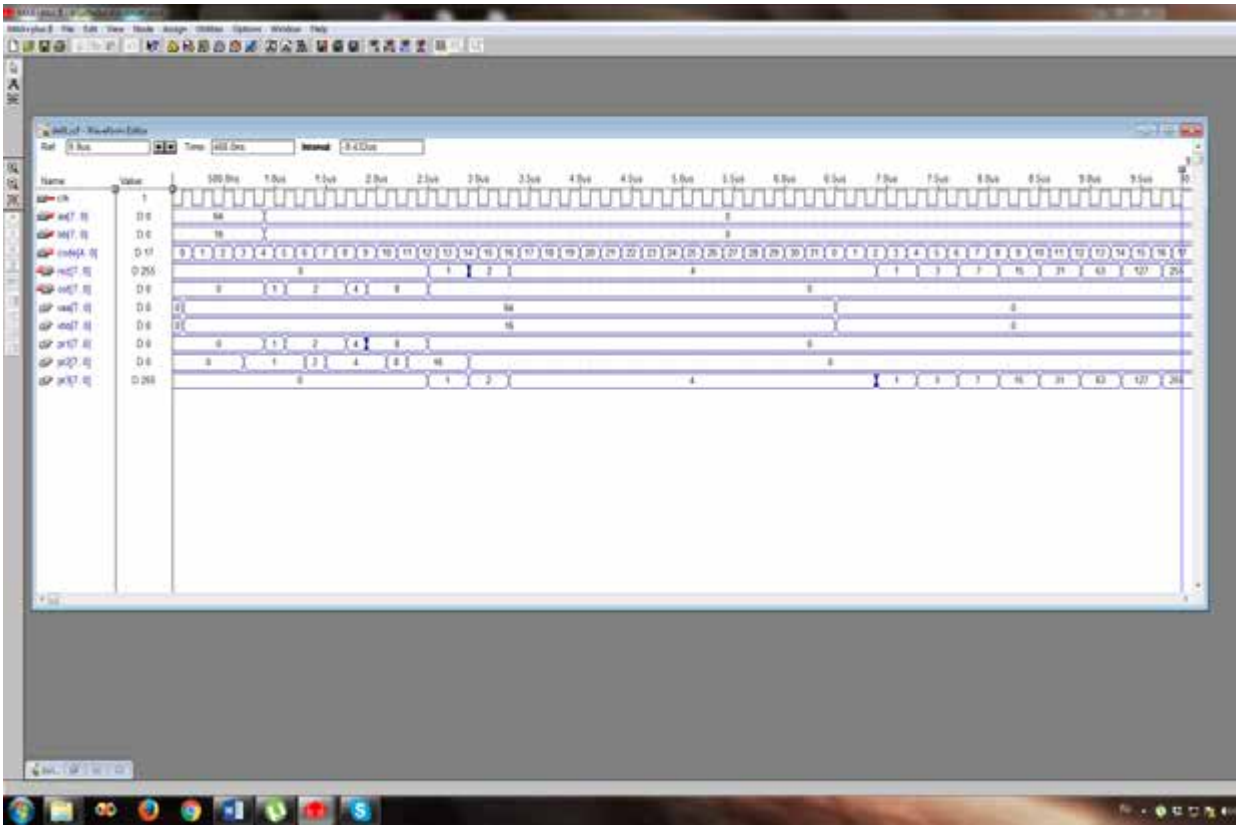


Рис 6. 25. Епюри функціонування дільника чисел

Загальний лістинг програми, що описує робботу пристрою приведений нижче.

```
subdesign delit
(       aa[7..0], bb[7..0], code[4..0], clk: input;
        rez[7..0], ost[7..0]: output;                   )
variable
        vaa[7..0]: dff; vbb[7..0]: dff;
        pr1[7..0]: dff; pr2[7..0]: dff; pr3[7..0]: dff;
begin
        vaa[].d=aa[]; vbb[].d=bb[];
case code[] is
when 0 => vaa[].clk=clk; vbb[].clk=clk;
when 1 =>   pr17.d=gnd; pr16.d=gnd; pr14.d=gnd; pr13.d=gnd; pr12.d=gnd;
pr11.d=gnd; pr10.d=vaa7.q;
                        pr1[].clk=clk;
when 2 => if pr1[] >= vbb[] then
```

```
                    pr2[]=pr1[]-vbb[];
                    pr37.d=gnd; pr36.d=gnd; pr35.d=gnd; pr34.d=gnd; pr33.d=gnd;
pr32.d=gnd; pr31.d=gnd; pr30.d=vcc;
                else pr2[]=pr1[];
                end if;
        pr2[].clk=clk; pr3[].clk=clk;
when 3 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q;
pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa6.q;
        pr2[].clk=clk;
when 4 =>   if pr2[] >= vbb[] then
                    pr1[]=pr2[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr1[]=pr2[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr1[].clk=clk; pr3[].clk=clk;
when 5 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q;
pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa5.q;
        pr1[].clk=clk;
when 6 => if pr1[] >= vbb[] then
                    pr2[]=pr1[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr2[]=pr1[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr2[].clk=clk; pr3[].clk=clk;
when 7 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q;
pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa4.q;
```

```
        pr2[].clk=clk;
when 8 =>   if pr2[] >= vbb[] then
                        pr1[]=pr2[]-vbb[];
                        pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr1[]=pr2[];
                        pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr1[].clk=clk; pr3[].clk=clk;
when 9 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q;
pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa3.q;
        pr1[].clk=clk;

when 10 => if pr1[] >= vbb[] then
                        pr2[]=pr1[]-vbb[];
                        pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr2[]=pr1[];
                        pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr2[].clk=clk; pr3[].clk=clk;
  when 11 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q;
pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa2.q;
        pr2[].clk=clk;

when 12 => if pr2[] >= vbb[] then
                        pr1[]=pr2[]-vbb[];
                        pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr1[]=pr2[];
```

```
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr1[].clk=clk; pr3[].clk=clk;
when 13 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q;
pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa1.q;
        pr1[].clk=clk;
when 14 => if pr1[] >= vbb[] then
                    pr2[]=pr1[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr2[]=pr1[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr2[].clk=clk; pr3[].clk=clk;
  when 15 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q;
pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa0.q;
        pr2[].clk=clk;
when 16 => if pr2[] >= vbb[] then
                    pr1[]=pr2[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr1[]=pr2[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr1[].clk=clk; pr3[].clk=clk;
                end case;
rez[]=pr3[].q;
        ost[]=pr1[];
end;
```

6.4. Лінійний регістр зсуву.

Регістр зсуву з лінійної зворотним зв'язком (LFSR) – це регістр зсуву бітових слів, у якого значення вхідного (всувають) біта дорівнює лінійній булевій функції від значень інших бітів регістра до зсуву. Може бути організований як програмними, так і апаратними засобами. Застосовується для генерації псевдовипадкових послідовностей бітів, що знаходить застосування, зокрема, в криптографії.

У регістрі зсуву з лінійної зворотним зв'язком виділяють дві частини (модуля):

- власне регістр зсуву;
- схему (або підпрограму) зворотного зв'язку.

Регістр складається з функціональних комірок пам’яті, в кожній з яких зберігається поточний стан (значення) одного біта. Кількість комірок L називають довжиною регістру. Біти (комірки) зазвичай нумеруються числами i=0, 1, 2, …, L-1. Функцією зворотного зв’язку для ЛРЗ є лінійна булева функція від значень всіх або декількох біт регістру. Функція виконує множення бітів регістру на коефіцієнти c_i, де i=0, 1, 2, …, L.

Протягом кожного такту регістр зсуву з лінійним зворотним зв'язком (Рис 4.26) виконує наступні операції:

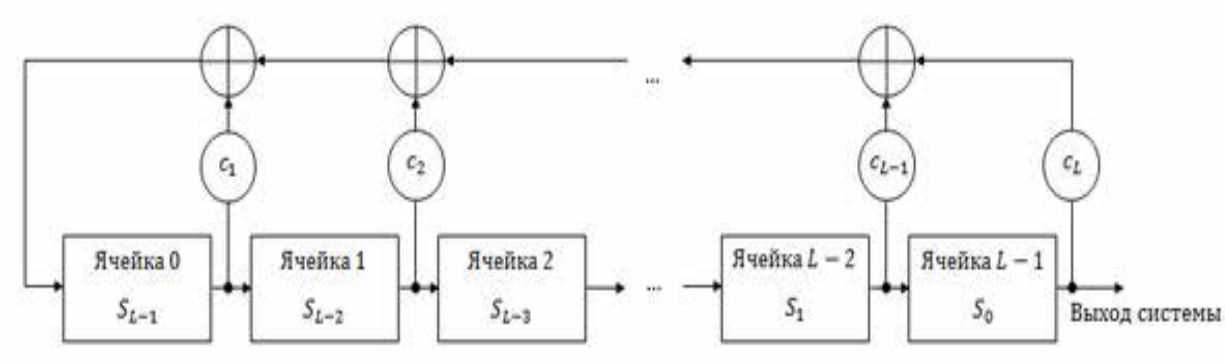


Рис 6.26. Структура лінійного рекурентного регістру

- читається біт, який розташований в комірці L-1; цей біт є черговим бітом вихідної послідовності;
- функція зворотного зв'язку обчислює нове значення для комірки 0,

використовуючи поточні значення комірок;

- вміст кожної i-ї комірки переміщується в наступну комірку i+1, де i=0, 1, 2, …, L-2;
- в комірку 0 записується біт, який раніше обчислювався функцією зворотного зв’язку.

Реалізація ЛРР (лінійного рекурентного регістру) за допомогою САПР ALTERA MAX + plus II 10.2

Реалізація лінійного рекурентного регістру може бути здійснена, як в текстовому Text Editor File так і в графічному редакторі. На вхід ми подаємо тактову частоту clk, початкове заповнення kl, та сигнал управління upr. За допомогою зворотного зв’язку os формується рекурентна послідовність. Будується сам регістр на основі D-тригерів. На епюрах функціонування зображено принцип формування вихідної послідовності (Рис. 6.27).

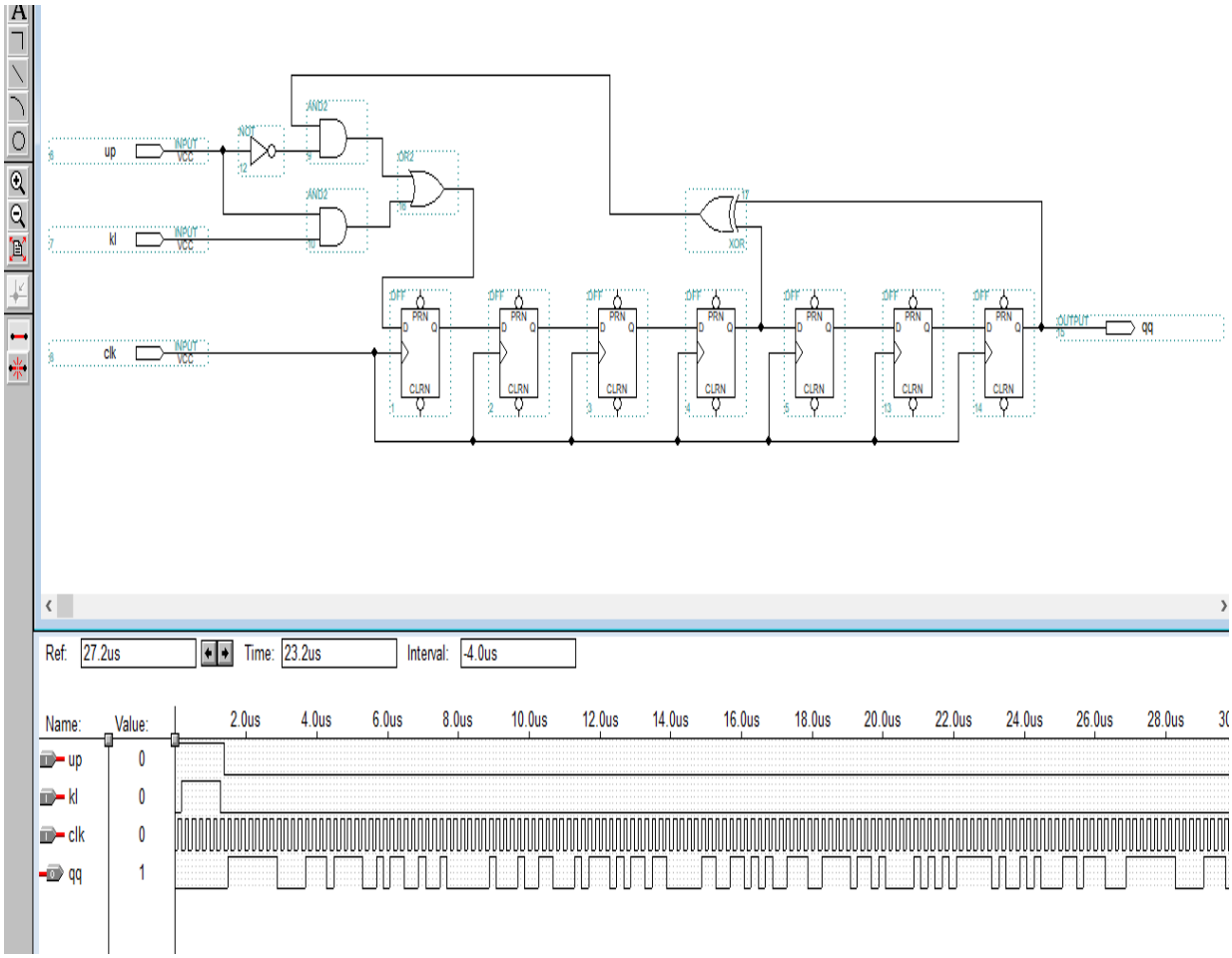


Рис. 6.27. Схема ЛРР та епюри функціонування.

В текстовому редакторі САПР *MAX + plus II* схему ЛРР виконано за допомогою двох модулів, які об’єднуються через оператор INCLUDE, (Рис. 6.28).

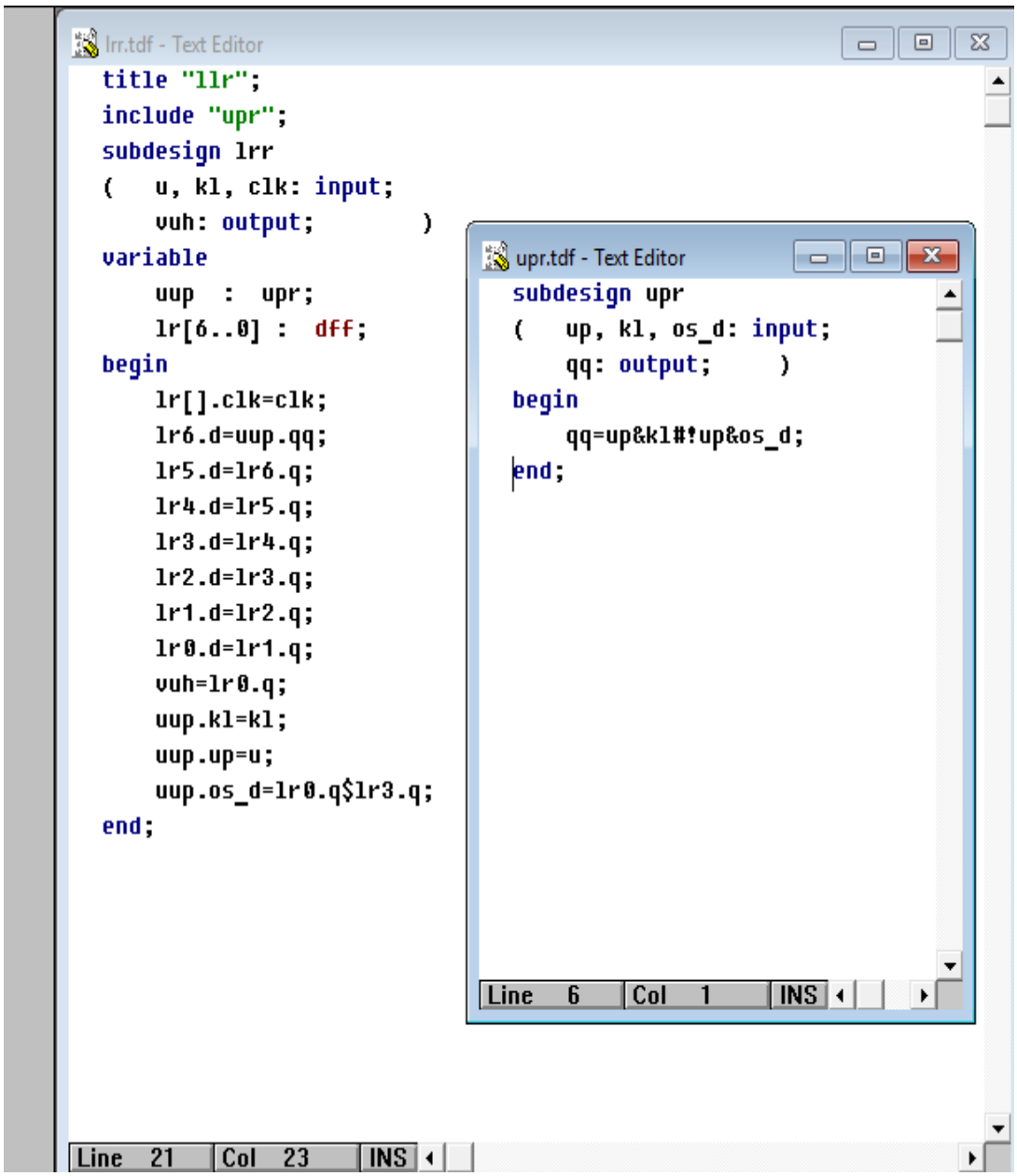


Рис. 6.28. Програма, що реалізує ЛРР.

Також, лінійний рекурентний регістр можливо створити в графічному редакторі з окремих модулів, а потім об’єднати їх за допомогою сформованих символів з допомогою команди CREATE DEFAULT SYMBOL (Рис. 6.29, 6.30).

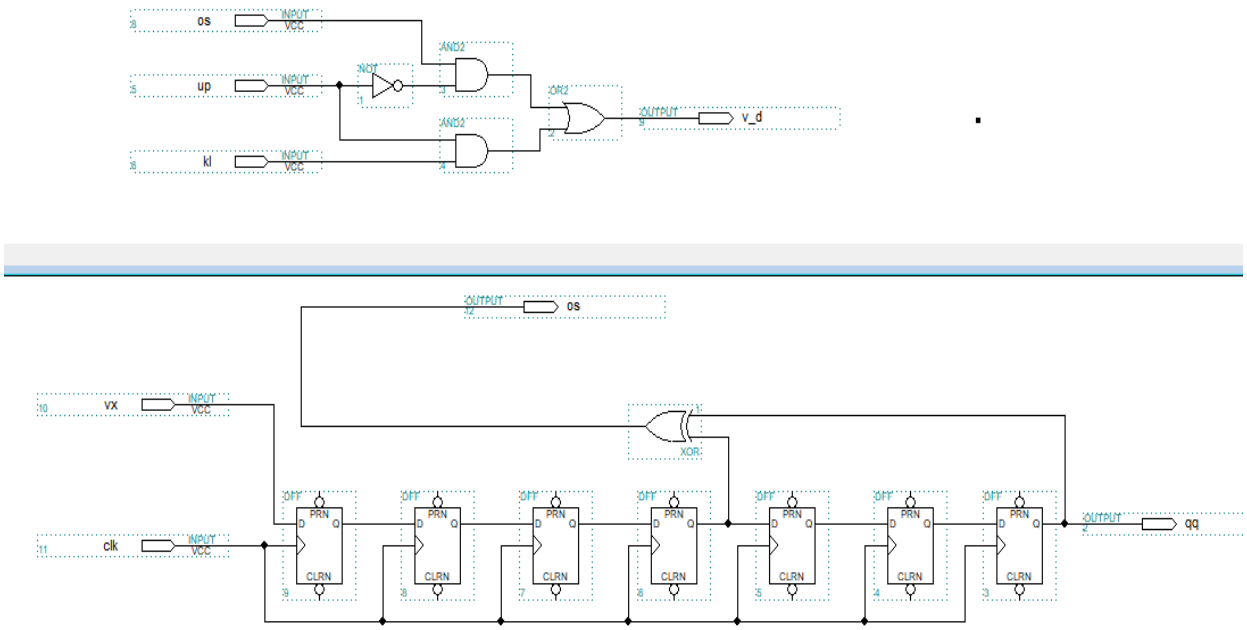


Рис. 6.29. Схема ЛРР.

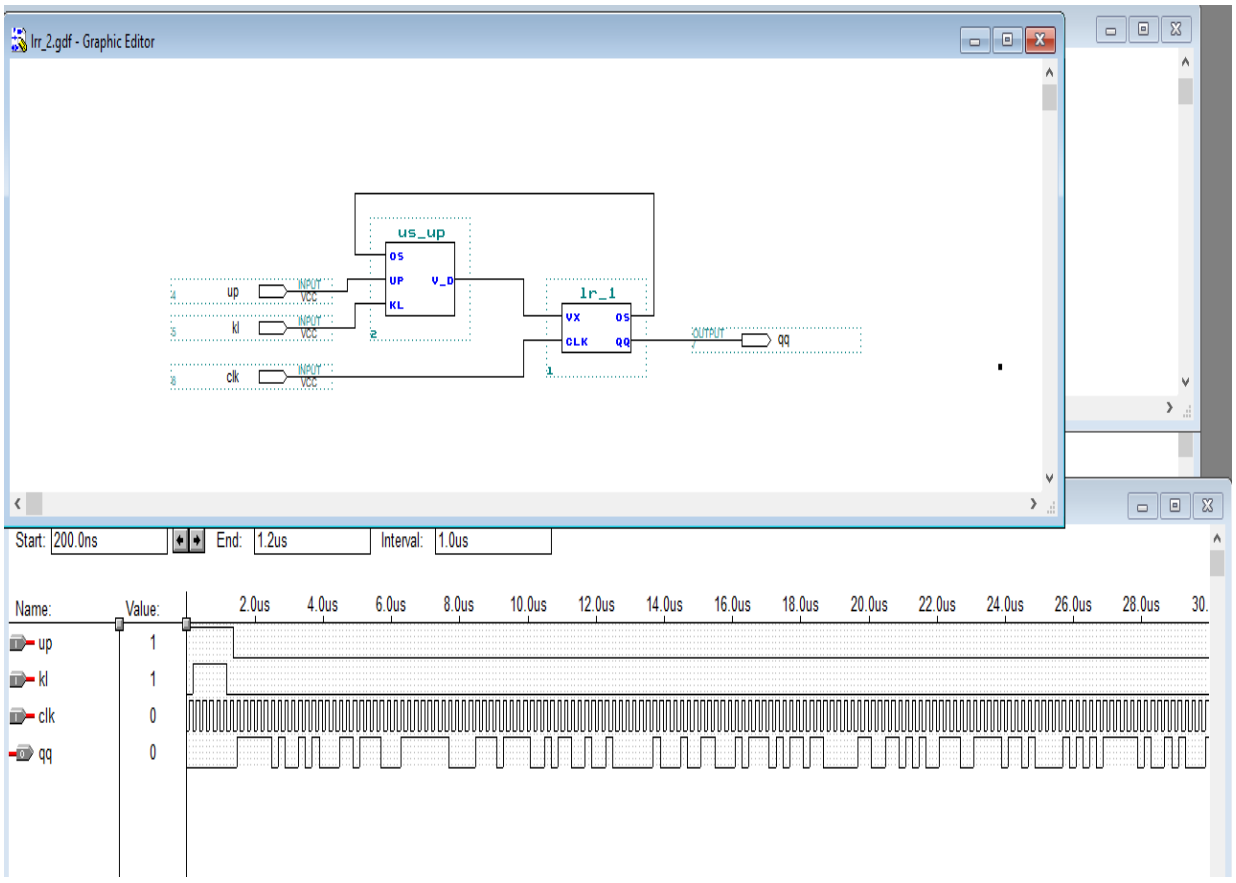


Рис. 6.30. Схема ЛРР та епюри функціонування.

6.5. ДСТУ ГОСТ 28147:2009 в режимі простої заміни

ДСТУ 28147-89 — блоковий шифр, що працює зі 256 — бітним ключем на 32 циклами перетворення та шифрує і розшифровує 64-бітні блоки. Основа даного шифру — це мережа Фейстеля. Базовий режимом шифрування за ДСТУ 28147-89 є режим гамування (визначені також складніші режими гамування зі зворотним зв'язком та імітовставки). Для перетворення в цьому режимі відкритий текст заздалдегідь розбивається на дві половини (молодші біти, старші біти). На кожному циклі роботи використовується сумування молодших бітів з 32 бітами ключової послідовності, що вибирається в заданому порядку.

Результат поділяється на вісім 4-бітних підпослідовностей, кожна з яких поступає на вхід свого блоку *таблиці змін* (у черзі зростання старшинства біт), званого нижче *S-блоком*. Повна кількість S-блоків ДСТУ — вісім, тобто така ж, скільки і підпослідовностей. Любий *S-блок* представляє собою перестановку числа від 0 до 15. Перша 4-бітна послідовність надходить на вхід першого блоку, друга — на вхід другого і т. д.

Виводи всіх S-блоків об'єднуються в 32-бітне слово, потім все слово циклічно зміщується вліво (в сторону старших розрядів) на 11 біт. Потім результат зсуву сумується за модулем два з інформацією з другого накопичувача, а результат записується в перший і т. д. на протязі 32-х циклів. Потім зашифрована інформація подається на вихід.

Приклад реалізації ДСТУ ГОСТ 28147:2009 в режимі простої заміни за допомогою САПР ALTERA MAX + plus II 10.2

Реалізація ДСТУ ГОСТ 28147:2009 здійснена в текстовому редакторі Text Editor File. На вхід ми подаємо відкриту інформацію, ключову інформації, сигнали синхронізації та службові команди (key[31..0], inf, clk, code[7..0], kl, vvod).

У розділі variable задаємо внутрішні змінні у вигляді регістрів для зберігання тимчасової інформації, використовується для цього регістри на D-тригерах. В кінці роботи програми ми отримаємо 64 біти зашифрованого тексту, який подається на вихід (Рис. 4.31, 4.32, 4.33, 4.34). Всі команди, символи, оператори та інше, що використано в реалізації було детально розглянуто в 3 розділі посібника.

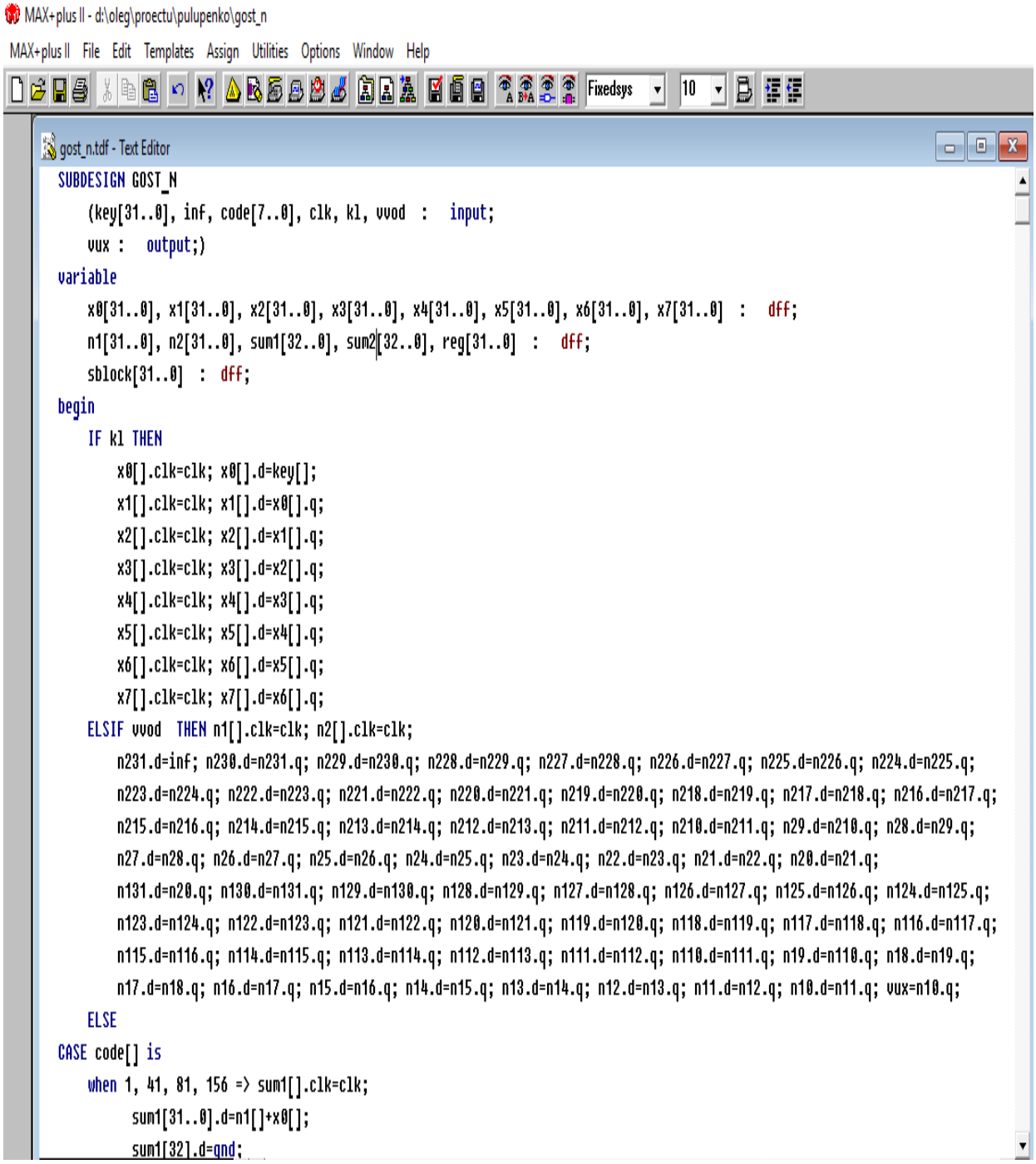


Рис. 6.31. Програмна реалізація ГОСТ 28147-89 на мові AHDL в режимі простої заміни.

```
    when 2, 7, 12, 17, 22, 27, 32, 37, 42, 47, 52, 57, 62, 67, 72, 77, 82, 87, 92, 97, 102, 107,
        112, 117, 122, 127, 132, 137, 142, 147, 152, 157 => sblock[].clk=clk;
TABLE
    sum1[3..0].q => sblock[3..0].d;
    H"0"=>H"1"; H"1"=>H"F"; H"2"=>H"D"; H"3"=>H"0"; H"4"=>H"5"; H"5"=>H"7"; H"6"=>H"A"; H"7"=>H"4";
    H"8"=>H"9"; H"9"=>H"2"; H"A"=>H"3"; H"B"=>H"E"; H"C"=>H"6"; H"D"=>H"B"; H"E"=>H"8"; H"F"=>H"C";
    END TABLE;
TABLE
    sum1[7..4].q => sblock[7..4].d;
    H"0"=>H"4"; H"1"=>H"A"; H"2"=>H"9"; H"3"=>H"2"; H"4"=>H"D"; H"5"=>H"8"; H"6"=>H"0"; H"7"=>H"E";
    H"8"=>H"6"; H"9"=>H"B"; H"A"=>H"1"; H"B"=>H"C"; H"C"=>H"7"; H"D"=>H"F"; H"E"=>H"5"; H"F"=>H"3";
    END TABLE;
TABLE
    sum1[11..8].q => sblock[11..8].d;
    H"0"=>H"E"; H"1"=>H"B"; H"2"=>H"4"; H"3"=>H"C"; H"4"=>H"6"; H"5"=>H"D"; H"6"=>H"F"; H"7"=>H"A";
    H"8"=>H"2"; H"9"=>H"3"; H"A"=>H"8"; H"B"=>H"1"; H"C"=>H"0"; H"D"=>H"7"; H"E"=>H"5"; H"F"=>H"9";
    END TABLE;
TABLE
    sum1[15..12].q => sblock[15..12].d;
    H"0"=>H"5"; H"1"=>H"8"; H"2"=>H"1"; H"3"=>H"D"; H"4"=>H"A"; H"5"=>H"3"; H"6"=>H"4"; H"7"=>H"2";
    H"8"=>H"E"; H"9"=>H"F"; H"A"=>H"C"; H"B"=>H"7"; H"C"=>H"6"; H"D"=>H"0"; H"E"=>H"9"; H"F"=>H"B";
    END TABLE;
TABLE
    sum1[19..16].q => sblock[19..16].d;
    H"0"=>H"7"; H"1"=>H"D"; H"2"=>H"A"; H"3"=>H"1"; H"4"=>H"1"; H"5"=>H"8"; H"6"=>H"9"; H"7"=>H"F";
    H"8"=>H"E"; H"9"=>H"4"; H"A"=>H"6"; H"B"=>H"C"; H"C"=>H"B"; H"D"=>H"2"; H"E"=>H"5"; H"F"=>H"3";
    END TABLE;
```

```
TABLE
    sum1[23..20].q => sblock[23..20].d;
    H"0"=>H"6"; H"1"=>H"C"; H"2"=>H"7"; H"3"=>H"1"; H"4"=>H"5"; H"5"=>H"F"; H"6"=>H"D"; H"7"=>H"8";
    H"8"=>H"4"; H"9"=>H"A"; H"A"=>H"9"; H"B"=>H"E"; H"C"=>H"0"; H"D"=>H"3"; H"E"=>H"B"; H"F"=>H"2";
    END TABLE;
TABLE
    sum1[27..24].q => sblock[27..24].d;
    H"0"=>H"4"; H"1"=>H"B"; H"2"=>H"A"; H"3"=>H"0"; H"4"=>H"7"; H"5"=>H"2"; H"6"=>H"1"; H"7"=>H"D";
    H"8"=>H"3"; H"9"=>H"6"; H"A"=>H"8"; H"B"=>H"5"; H"C"=>H"9"; H"D"=>H"C"; H"E"=>H"F"; H"F"=>H"E";
    END TABLE;
TABLE
    sum1[31..28].q => sblock[31..28].d;
    H"0"=>H"D"; H"1"=>H"B"; H"2"=>H"4"; H"3"=>H"1"; H"4"=>H"3"; H"5"=>H"F"; H"6"=>H"5"; H"7"=>H"9";
    H"8"=>H"0"; H"9"=>H"A"; H"A"=>H"E"; H"B"=>H"7"; H"C"=>H"6"; H"D"=>H"8"; H"E"=>H"2"; H"F"=>H"C";
    END TABLE;

when 3, 8, 13, 18, 23, 28, 33, 38, 43, 48, 53, 58, 63, 68, 73, 78, 83, 88, 93, 98, 103, 108,
        113, 118, 123, 128, 133, 138, 143, 148, 153, 158 => reg[].clk=clk;
        reg[10..0].d=sblock[31..21].q;
        reg[31..11].d=sblock[20..0].q;

when 4, 9, 14, 19, 24, 29, 34, 39, 44, 49, 54, 59, 64, 69, 74, 79, 84, 89, 94, 99, 104, 109, 114,
        119, 124, 129, 134, 139, 144, 149, 154, 159 =>  sum2[].clk=clk;
        sum2[31..0]=n2[31..0].q$reg[31..0].q;
        sum2[32]=gnd;

when 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110,
        115, 120, 125, 130, 135, 140, 145, 150, 155 =>  n2[].clk=clk;
        n2[31..0].d=n1[31..0].q;
        n1[].clk=clk;
        n1[31..0].d=sum2[31..0].q;
```

Рис. 6.32. Програмна реалізація (продовження).

```
when 6, 46, 86, 151 => sum1[].clk=clk;   sum1[31..0].d=n1[]+x1[];   sum1[32].d=gnd;

when 11, 51, 91, 146 => sum1[].clk=clk;   sum1[31..0].d=n1[]+x2[];  sum1[32].d=gnd;

when 16, 56, 96, 141 => sum1[].clk=clk;   sum1[31..0].d=n1[]+x3[];  sum1[32].d=gnd;

when 21, 61, 101, 136 => sum1[].clk=clk;   sum1[31..0].d=n1[]+x4[];  sum1[32].d=gnd;

when 26, 66, 106, 131 => sum1[].clk=clk;    sum1[31..0].d=n1[]+x5[];   sum1[32].d=gnd;

when 31, 71, 111, 126 => sum1[].clk=clk;    sum1[31..0].d=n1[]+x6[];   sum1[32].d=gnd;

when 36, 76, 116, 121 => sum1[].clk=clk;    sum1[31..0].d=n1[]+x7[];   sum1[32].d=gnd;

when 160 =>     n2[].clk=clk;           n2[31..0].d=sum2[31..0].q;
        end case;
    end if;
end;
```

Рис. 6.33. Програмна реалізація (продовження).

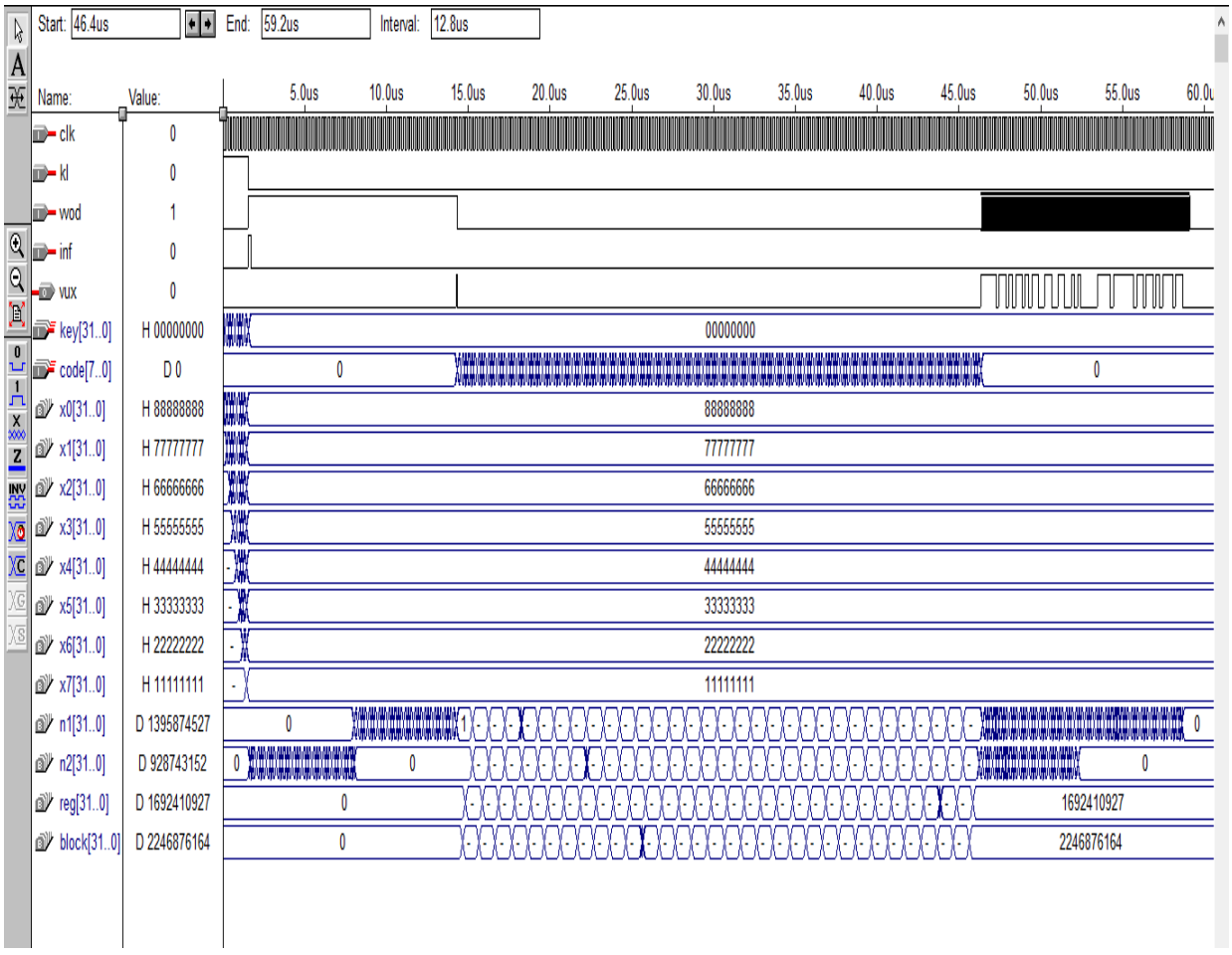


Рис. 6.34. Епюри функціонування програмної реалізації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. ДСТУ 3.321-96. Системи конструкторської документації. Терміни та визначення основних понять. Київ, 1997.
2. Проектування радіоелектронного вузла та розробка конструкторсько-технологічної документації для його виготовлення в середовищі інтегрованої САПР .Методичні вказівки до курсової роботи з дисципліни «Адміністрування комп'ютерних систем проектування" для студентів спеціальності 122 "Системне проектування". Львів,НУЛП.2019. с.22
3. Скрипник Л.В., Корнейко О.В., Кулініч О.М. Сучасна елементна база для побудови систем захисту інформації. – Ч. І. Принципи побудови інтегральних схем із структурою, що програмується. – К.: 2004. 100 с.
4. Скрипник Л.В., Ярманов С.С., Кулініч О.М. Сучасна елементна база для побудови систем захисту інформації. – Ч. ІІ. Проектування цифрових пристроїв на основі інтегральних схем із структурою, що програмується. – К.: 2004. 161 с.
5. Проектування елементів мехатронних систем у середовищі SolidWorks [Текст]: метод. вказівки до викон. комп. практикуму для студентів напряму підготовки 6.050702 «Електромеханіка» / Уклад.: Л.К. Лістовщик, В.О. Поліщук, М.П. Калюш. – К.: НТУУ «КПІ», 2013. - 76 с.
6. Tickoo S. SolidWorks 2017 for Designers / S. Tickoo // CADCIM Technologies, 2017. - 2223 p.
7. Методичні вказівки до виконання комп'ютерних практикумів з дисципліни "Пакети прикладних програм для конструювання електричних машин", використання систем автоматизованого проектування AutoCAD та SolidWorks для конструювання електричних машин / Уклад.: Ю.М. Васьковський, Ю.А. Гайденко, С.С. Цивінський. – К.: НТУУ "КПІ ІМ. І.СІКОРСЬКОГО"
8. Методичні вказівки до виконання практичних робіт з дисципліни "Основи проектування та САПР" для підготовки бакалаврів за напрямом 6.050502

“Інженерна механіка”, 6.050503 “Машинобудування” / укладач: Г.І. Танцура – Дніпродзержинськ, ДДТУ, 2011 р. – 80 с

9. Сучасні технології конструювання РЕА. Методичні вказівки до виконання лабораторних робіт для студентів спеціальності 172 «Телекомунікації та радіотехніка» усіх форм навчання. – Чернігів: ЧНТУ, 2019. – 67 с
10. Холодняк Ю. В. Комп’ютерне проектування промислових виробів: навчально-методичний посібник з виконання практичних робіт / Ю. В. Холодняк; ТДАТУ. – Мелітополь: ТДАТУ, 2020. – 152 с
11. Кофанов В.Л. Математичні та схемотехнічні основи цифрових пристроїв: Навч. посібник. – Вінниця: «УНІВЕРСУМ-Вінниця», 2005. – 165 с. 2. Справочник по цифровой схемотехнике / В.И. Зубчук, В.П. Сигорский, А.Н. Шкуро – К.: Техніка, 1990. – 448 с.
12. Радіотехніка: Енциклопедичний навчальний довідник: Навч. посібник / За ред. Ю.Л. Мазора, Є.А. Мачуського, В.І. Правди. – К.: Вища школа, 1999. – 838 с.
13. Рудик А.В. Радіоавтоматика. Частина 3. Дискретні та цифрові системи радіоавтоматики. Навчальний посібник. – Вінниця: ВНТУ, 2003. – 152 с. 9. University Program UP2 Education Kit. – Altera Corporation, v. 3.1, 2004.
14.
15. Bethune J.D. Engineering Design and Graphics with SolidWorks 2016 / J.D. Bethune // Peachpit Press, 2016. - 784 p.
16. Onwubolu G.C. Introduction to SolidWorks: A Comprehensive Guide with Applications in 3D Printing / G.C. Onwubolu // CRC Press, 2017. - 1193 p.
17. Tickoo S. SolidWorks 2017 for Designers / S. Tickoo // CADCIM Technologies, 2017. - 2223 p.
18. Verma G. SolidWorks 2017 Black Book / G. Verma, M. Weber // CADCAMCAE Works, 2017. - 518 p.
19. Beginning AutoCAD® 2019 Exercise Workbook. Cheryl R. Shrock, Steve Heather. Industrial Press, Inc.; Workbook edition (May 22, 2018) ISBN: 9780831136260./ 648p.

20. AutoCAD 2019 Tutorial First Level 2D Fundamentals - Randy H. Shih;Luke Jumper SDC Publications (May 28, 2018) ISBN-10: 1630571881 /.450 p.
21. AutoCAD 2019 Tutorial Second Level 3D Modeling Randy H. Shih SDC Publications (July 2, 2018) ISBN-10: 1630571946/ 400 pp.
8. AutoCAD 2019. Полное руководство . Николай Жарков, Финков М.– Наука и техника, 2019., –640 с.

Інформаційні ресурси

1. https://classroom.google.com/c/MTU5MjI2NDY3Nzcw?cjc=wimltvm
2. http://emoev.kpi.ua/sapr-elektromexanichnix-sitem-ta-kompleksiv/