МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Магістерської підготовки
Кафедра Інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Моделювання та програмування цифрових і мікропроцесорних систем на VHDL»

Виконав студент 2 курсу групи МІС-18 спеціальності 122 Комп'ютерні науки
Ходжаєв Довлетмират

Керівник к.геогр.н.,доцент
Коваленко Л.Б

Консультант

Рецензент д.т.н., проф.д.
Ковальчук В.В.

Одеса 2020

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

АС – автоматизована система;

ВІС – великі інтегральні схеми

ЕОМ – електронно-обчислювальна машина

НВІС – надвеликі інтегральні схеми

ОС – операційна система

ПЛІС – програмовані логічні інтегральні схеми

ПЗ – програмне забезпечення;

САПР – система автоматизованого проектування

PLD – програмовані логічні матриці

ВСТУП

Використання в сучасних цифрових системах великих інтегральних схем (ВІС) і надвеликих інтегральних схем (НВІС) приводить до необхідності структуризації процесу проектування, тому особлива увага приділяється створенню відповідних інструментальних засобів проектування. Конкретним прикладом подібних інструментальних засобів є мови опису апаратури, CDL (computer discription langauage − мова опису (архітектури) ЕОМ), ISP (Interactive System Language, Интерактивна системна мова) і AHPL (A Hardware Programming Language – мова програмування апаратних засобів, розробниками якої є Hill та Peterson). Проте ці мови не дозволяють створити адекватну модель з урахуванням часових параметрів. Такі мови, як, наприклад, HHDL, ISP і VHDL ((Very high speed integrated circuits) Hardware Description Language – мова опису інтегральних схем), передбачають побудову універсальних моделей з часовими параметрами і не орієнтовані на конкретні апаратні структури. Перевага цих мов полягає в тому, що вони документують проекти і здійснюють моделювання. Хороша документація необхідна для забезпечення транспортабельності проекту.

Всі широко поширені мови опису апаратури підтримуються відповідними системами моделювання. Отже, макетування систем замінюється моделюванням, і процес проектування відрізняється від традиційного.

При підготовці проектів, розробники використовують автоматизовані робочі місця (АРМ), де проект вводиться як початковий файл на мові опису апаратури у вигляді принципової електричної схеми. Паралельно проводиться розробка програмного забезпечення (якщо це необхідно) для проектованої системи. Після виготовлення дослідного або експериментального зразка прототипу системи проводиться автономне тестування програмної і технічної частини проекту. На завершуючому етапі проектування проводиться комплексне тестування системи з використанням внутрішньосхемних емуляторів і логічних аналізаторів.

Актуальність роботи зростає з кожним днем тому, що програмовані логічні інтегральні схеми (ПЛІС) – зручна в освоєні і застосуванні елементна база. Різко зростає кількість елементів у кристалі, а ціни падають.

Сучасні обчислювальні машини починають свою історію від універсаль-ного комп'ютера фон Неймана, в якому процеси рішення задачі організовано як послідовне в часі виконання простих операцій. Кожна операція виконується за своєю командою, сукупність команд реалізує програму, розроблену для рішення даної задачі. Склад апаратних засобів фіксовано (процесор, пам'ять, пристрої вводу/виводу), і зміна задачі, що розв'язується відбивається лише на кількості та складі команд програми. Такий процес розв'язку відповідає програмній інтерпретації алгоритмів. Але в сучасних обчислювальних архітектурах широко використовується одночасна (паралельна або розподілена) реалізація декількох команд, конвеєрна та суперскалярна обробка, паралельне виконання декількох програм в багатопроцесорних системах.

Програмний метод інтерпретації алгоритмів не єдиний спосіб рішення задачі. Можна застосовувати структурну або, інакше кажучи, апаратну інтер-претацію алгоритму. В даному випадку також виконується множина окремих простих операцій. Але для виконання окремих дій застосовують свої функціо-на-льні блоки, з'єднані в схему визначеної структури. Функціональні характери-стики блоків і характер з'єднання між ними відповідають алгоритму рішення задачі. Іншими словами, для отримання рішення створюється структура, що відображає алгоритм якій інтерпретується. Алгоритм реалізується за рахунок проходження даних і їх перетворення по шляхах обробки (асинхронно або при тактуванні), від входів до виходів схеми, з яких результат зчитується. Команди для виконання окремих операцій відсутні. Це забезпечує розподілене рішення задачі не тільки в часі, але і в просторі. Природним чином досягається висока ступінь паралелізму. В даному випадку склад апаратних засобів тісно зв'язаний із задачею, яка вирішується. Ускладнення задачі приводить до збільшення кіль-кості обладнання, яке використовується. Зміна задачі потребує зміни складу компонентів і способів їх з'єднання.

Дякуючи програмуванню структур для електронної промисловості відбулася універсалізація ВІС/НВІС, зробивши їх придатними для широкого кола споживачів. Це було досягнуто виготовленням промисловістю деяких загото-вок, які в подальшому перетворюються споживачем в необхідні їм пристрої. Заготовки можна виробляти масовим тиражем, що дозволяє виконати їх у вигляді ВІС/НВІС. Очевидно, необхідна простота доведення заготовок до закінченої мі-кросхеми, тобто простота програмування структур ВІС/НВІС за специфікація-ми замовника. В даному випадку застосування ВІС/НВІС навіть для малотира-ж-ного виробу стає можливим і ефективним.

Таким чином, програмованість ВІС/НВІС, іншими словами, створення програмованих логічних інтегральних схем, названих ПЛІС, дозволила використовувати їх там, де раніше приходилося застосовувати МІС/СІС. Більш пізніше досягнення – репрограмованість мікросхем – забезпечила можливість багатора-зової зміни налаштування. Це дозволяє змінити функціонування мікросхеми, тобто на одному і тому ж кристалі отримати пристрої різного призначення, сти-раючи стару конфігурацію її структури і записуючи нову. Для деяких різнови-дів НВІС програмованої логіки можлива ре конфігурація в оперативному режимі, тобто без виймання мікросхеми із працюючої системи і з високою швидкістю. Оперативне програмування мікросхем – новий шлях побудови апаратури.

1 ОСНОВНІ ПИТАННЯ МОДЕЛЮВАННЯ ЦИФРОВИХ ОБЧИСЛЮВАЛЬНИХ СИСТЕМ

Мова паралельного програмування повинна, з одного боку, забезпечувати користувача-програміста звичною і зручною програмною моделлю, з другого боку, алгоритм, описаний цією мовою повинен мати нескладне і ефективне відображення в архітектуру цільової ОС і мати властивість переносимості між ОС з неоднаковою архітектурою. З цієї точки зору перспективною моделлю ОС є модель з віртуальною топологією. Застосовуючи таку модель, програміст складає програму для віртуальної структури ОС, а компілятор відображає цю програму, в якій явно задана віртуальна структура, в структуру конкретної ОС.

Останнім часом набуває поширення розробка ОС з програмованою архітектурою (Configurable Computer). Поява програмованих логічних інтегральних схем (ПЛІС) з об’ємом до 1 млн. вентилів і більше (таких як Xilinx Virtex) дала потужний поштовх для розвитку цього напрямку. Слід зауважити, що ОС з програмованою архітектурою найбільш відповідає задачі відображення програм для віртуальної топології.

Мова VHDL була розроблена в Міністерстві оборони США на початку вісімдесятих років як мова для опису моделей апаратних засобів обчислювальної техніки (Hardware Description Language). В середині вісімдесятих років виробився особливий стиль написання програм, так званий стиль синтезу. Алгоритм, описаний стилем синтезу може бути відображений в конкретну логічну схему за допомогою компілятора-синтезатора логічних схем. Такі компілятори-синтезатори набули широкого розповсюдження в наш час як невід’ємна складова систем автоматизованого проектування (САПР) для ПЛІС та замовлених НВІС. Найбільш відомими компіляторами – синтезаторами є Synopsys фірми Synopsys, Synplify фірми Synplicity, Leonardo фірми Exemplar, Metamor фірми Metalogic.

Відображення алгоритму на VHDL в логічну схему є взаємно однозначним і ін’єктивним, причому досягається найбільш високий ступінь розпаралелювання обчислень. Крім того, алгоритм на VHDL може відображати всі тонкі властивості обчислювального алгоритму, такі, як точно задана розрядність вхідних і проміжних даних, пряма логічна обробка бітів операндів, які неможливо безпосередньо відобразити засобами інших алгоритмічних мов. Це означає, що структура, одержана в результаті синтезу відображає всі особливості початкового паралельного алгоритму.

Особливістю алгоритму, описаному на VHDL стилем для синтезу є те, що він, як правило, може бути без змін відображений в декілька різних програмованих обчислювальних середовищ – ними є ПЛІС, які мають різну технологію і архітектуру. Таким чином, алгоритм на VHDL є переносимим між різними обчислювальними середовищами. [1][1)]

Основою технології програмування ОС з програмованою архітектурою є відображення алгоритмів на мові VHDL в логічну структуру, яка занурюється в ПЛІС. В той час, коли більшість сучасних паралельних ОС не забезпечують гідної реалізації дрібнозернистого паралелізму періодичних алгоритмів з локальними зв’язками, застосування з програмованою ОС може бути ефективним вирішенням проблеми реалізації дрібнозернистого паралелізму. [1]

У системах автоматизованого проектування цифрових обчислювальних систем необхідно передбачати реальну поведінку елементів і вузлів окремих пристроїв розроблюваної системи. Для попередньої оцінки поведінки в більшості випадків виявляється достатнім моделювання на логічному рівні, де оперують з логічними значеннями "1" і "0". В деяких випадках додатково

[1)] [1] Сергиенко А.М. VHDL для проектирования вычислительных устройств / А.М. Сергиенко. К.: ЧП "Корнейчук", ООО "ТИД "ДС", 2003. 208 с.

розглядаються перехідні процеси і затримки сигналів для точнішого прогнозування роботи схеми [2] – [6] [1)].

При логічному моделюванні на ЕОМ емулюється робота функціональної схеми за принципом проходження інформації у вигляді логічних станів сигналів. Процес моделювання виконується в три етапи: подача на вхід схеми тестової комбінації; обчислення станів елементів схеми по відповідних логічних моделях послідовно від входу до виходу всієї схеми, аналіз реакції схеми на отримане збурення.

Для моделювання схем необхідно мати спеціальний опис схеми, бібліотеку математичних (логічних) моделей елементів, набір методів моделювання, тестові послідовності і відомі або прогнозовані реакції схеми.

Моделювання систем відрізняється ступенем деталізації схем. Як елементи можуть використовуватися модулі цифрових систем, що включають десятки ВІС високої степені інтеграції, і навіть цілі обчислювальні системи у вигляді офісних і промислових комп'ютерів, серверів і так далі Моделі таких систем дуже складні і можуть бути функціональним об'єктно-орієнтованим описом.

1.1 Логічне моделювання функціональних вузлів

Опис функціональних схем. Одному з найважливіших завдань в моделюванні є розробка і використання в процесі функціонально-логічного

[1)] [2] Томашевський В. М. Моделювання систем. К: Видавнича група ВНУ, 2005. 352 С.: іл. ISBN 966-552-120-9

[3] Бубенніков А. Н. «Моделирование интегральных микротехнологий, приборов и схем». М, «Высш. шк.», 1989. 320 с.

[4] Моделювання. Конспект лекцій з дисципліни «Моделювання систем» для студентів напряму підготовки 6.050103 – „Програмна інженерія”. / Укл.: Скітер І.С. Чернігів: ЧІБіП, 2015. 139 с.

[5] Дьяков И.А. Моделирование цифровых и микропроцессорных систем. Язык VHDL / И.А. Дьяков. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2001. 68 с.

[6] Стеценко, І.В. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В. Стеценко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. Черкаси : ЧДТУ, 2010. 399 с. ISBN 978-966-402-073-9

проектування мови опису функціональних схем. Мова повинна бути зрозумілою, достатньо простою і мати зручний інтерфейс. Такі вимоги приводять до спеціалізованої орієнтації на різні типи схем (цифрові, аналогові) і рівні деталізації моделей (логічні елементи, ВІС, НВІС, програмовані логічні контролери). Особливості мов виявляються в описі змінних типу "сигнал", в описі затримки розповсюдження сигналів, а також у використанні процесів для моделювання паралельного проходження сигналів в схемах. Це такі моделюючі системи, як VHDL і ін. [7][1)].

Для роботи моделюючих програм необхідна інформація і відомості про топологію схеми, вхідні і вихідні вузли, моделі елементів, моделі несправностей елементів і різна додаткова інформація.

Ранжирування елементів функціональної схеми. Велике значення при моделюванні має порядок розгляду описів схеми. При пошуку значень виходів елементів необхідно, щоб значення входів цих елементів були вже визначені.

Процесу логічного моделювання передує процедура ранжирування описів елементів функціональної схеми. Елементи повинні розміщуватися в такому порядку, в якому відбувається послідовне перемикання елементів схеми при вхідній дії X(τ). Елемент схеми має ранг p + 1, якщо його входи підключені до елементу, що має максимальний ранг p. Елементу, на входи якого поступають сигнали від зовнішніх джерел, привласнюється ранг p=1.

Ранжирування комбінаційних схем виконується по наступному алгоритму:

1 Визначити елементи, всі входи яких підключені до джерела зовнішніх сигналів X(τ), і привласнити ранг p=1.

2 Визначити елементи, всі входи або більше половини входів яких підключені до виходів елементів, що мають ранг.

3 Збільшити ранг на одиницю і привласнити вибраним елементам.

4 Перевірити, чи є ще вільні, без рангу, елементи. Якщо є, то здійснити

[1)] [7] «Леонов С.Ю. VHDL-технології проектування електронних пристроїв : навч. посіб. / С.Ю. Леонов, Т.В. Гладких, О.І. Баленко. К. : Вид-во "КАФЕДРА", 2014. 423 с.

перехід до п. 2, інакше виконати наступний крок.

5 Якщо всі елементи впорядковані і найвищий ранг мають елементи, виходи яких є виходами схеми, то завершити процес ранжирування. Інакше видати повідомлення "Схема не комбінаційна" або "Помилка в описі схеми".

Приведений алгоритм може бути модифікований і покращений. У нім позначені тільки основні кроки ранжирування і найпростіші умови вибору елементів. Перевіримо його працездатність на прикладі нескладної схеми (Рис. 1.1). Маємо опис зв'язків функціональної схеми формування імпульсу, затриманого і укороченого щодо вхідного.

Хай ранг вхідного сигналу Xl(τ) дорівнює нулю. Тоді отримуємо, що ранг елементу 1 дорівнює одиниці, елементів 2 − 4 відповідно два, три і чотири.

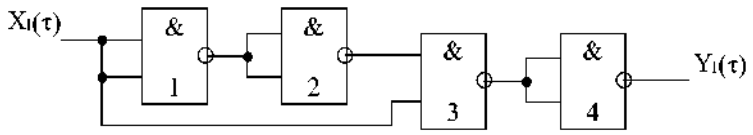


Рисунок 1.1 − Функціональна схема формувача імпульсу

Результат очевидний і не порушує послідовного проходження сигналу.

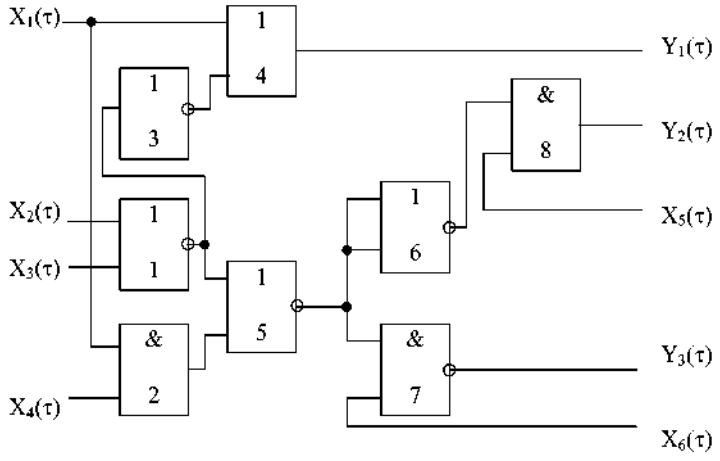


Рисунок 1.2 − Приклад функціональної схеми

У випадку (Рис. 1.2) результат не очевидний. Виконаємо процедуру ранжирування. Ранг елементів 1, 2 рівний 1 ($p_{1\,2} = 1$). Ранг решти елементів також визначається по вищеописаному алгоритму і рівний $p_{3,5} = 2$; $p_{4,6,7} = 3$; $p_8 = 4$.

Якщо в схемі є зворотні зв'язки (рис.1.3), то спочатку ранжирують її комбінаційну частину описаним способом, а потім удаються до умовного ранжирування, вибирають із списку зворотних зв'язків один і вважають його умовно ранжированим (зворотний зв'язок розмикають). Якщо можливо продовжити процес прямого ранжирування, то елемент з умовно розімкненим зворотним зв'язком отримує черговий ранг і процес продовжується. Інакше вибирають інший зворотний зв'язок, розмикають і намагаються продовжити ранжирування. Перед виконанням ранжирування вважають, що опис зв'язків елементів вже виконаний, тобто визначена множина $G = (G^1, G^2)$ компонентів схеми, де G^1 – множина логічних елементів; G^2 – множина зв'язків. [8][1)]

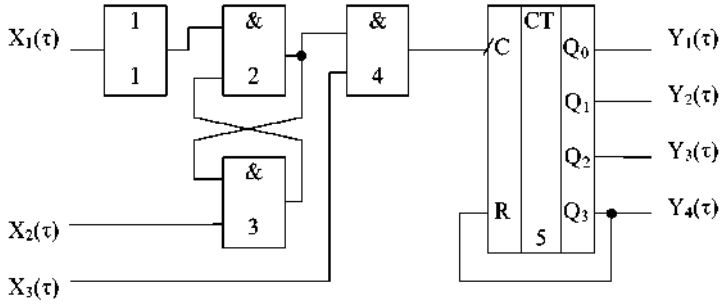


Рисунок 1.3 – Схема із зворотними зв'язками

Таким чином, алгоритм ранжирування послідовнісних схем і схем із зворотними зв'язками має наступний вигляд:

1 Визначити елементи множини G^1, всі входи якої підключені до джерел зовнішніх сигналів $X(\tau)$ і привласнити $p = 1$.

[1)] [8] Преснухін Л. Н., Воробйов И. В., Шішкевіч А. А. «Расчет элементов цифровых устройств.» М, «Высш. Шк»., 1991. 526 с.

2 Визначити елементи $g_i \in G^1$, всі входи або більше половини входів яких підключені до виходів елементів, що мають ранг.

3 Якщо входи елементів g_i, що залишилися, підключені до виходів елементів, що ще не мають рангу, тобто знайдені зв'язки $s_i \in G^2$, то буде перехід до п. 4, інакше збільшити ранг $p = p + 1$ і привласнити елементам g, Тимчасово виключити елементи g_i із списку G^1.

4 Вибрати з s_i зв'язок s_j з найменш віддаленим не ранжированим елементом, виключити його з розгляду і привласнити елементу $p = p + 1$. Виключити зв'язок $s_j s_1$, а також із списку зворотних зв'язків G^2 на час обробки.

5 Якщо є ще зворотні зв'язки в G^2, то обробка повторюється з п. 3.

6 Якщо є ще елементи без рангу в G^1, то процес повторюється з п. 2.

7 Якщо всі елементи відранжировані і найвищий ранг мають елементи, виходи яких являються виходами схеми, то ранжирування завершити. Інакше видати повідомлення "Помилка в описі схеми".

Проведемо ранжирування схеми, показаної на Рис. 1.3, по описаному вище алгоритму. Елементи схеми матимуть наступний ранг: $p_1 = 1$; $p_{2,3} = 2$; $p_4 = 3$; $p_5 = 4$.

Ранжировані елементи заносяться в список в порядку зростання рангів. Елементи, що мають однаковий ранг, можуть розташовуватися в списку в довільному порядку.

При моделюванні список елементів обробляється відповідно до встановлених рангів. Процес починається з елементів, $p = 1$. Порядок запису елементів і послідовність обробки співпадають, що дозволяє значно економити час.

Логічна модель. Завдання логічного моделювання зводиться до вирішення логічних рівнянь для окремих елементів.

$$O_i(\tau) = M_i[I_i(\tau)]$$

де M_i − логічна модель (функція); O_i − вектор логічних станів виходів; I_i − вектор логічних станів входів і-го елементу.

Логічна функція в деяких випадках може бути задана у вигляді таблиці. Це може пояснюватися складністю функціонування схеми і неможливістю скласти функцію через логічні операції. Таблиці застосовують для спрощення опису роботи елементів, що мають складні і громіздкі логічні вирази.

Для спрощення моделювання іноді застосовують поведінкові моделі схем. Такі моделі не показують принцип організації схеми на логічному рівні, а ілюструють поведінку схеми на високому рівні абстракції. Наприклад, проводиться моделювання нагромаджуючого лічильника. Декомпозиція схеми приводить до тригерів і зв'язків між ними. Кожен тригер можна представити набором елементарних логічних елементів. Отримаємо достатньо складну схему. Але якщо вимоги до моделювання нижчі, то цілком достатньою може бути модель виду i = i + 1, це і є поведінкова модель.

В процесі роботи з моделями і функціями потрібно ввести деякі відповідності. Якщо на входи елементів поступає дія $X(\tau)$, то вектор входів $I(\tau)$ для елементів і схеми набуває значень цієї дії $I(\tau) = X(\tau)$. У схемі входи елементів, підключені до виходів інших елементів, мають значення $I(\tau) = O_j(\tau)$ для $i \neq j$. Присутність зворотних зв'язків в і-му елементі означає, що i=j:

Процес однопрохідного логічного моделювання проводиться послідовною обробкою моделей елементів в порядку зростання рангів. Враховуючи наявність декількох елементів в схемі і існуючі зв'язки, логічні моделі об'єднуються в систему рівнянь:

$$\begin{cases} O_1^p = M_1\left[O_1^{p-1}(\Delta\tau), O_2^{p-1}(\Delta\tau), \ldots, O_{n-1}^{p-1}(\Delta\tau), O_n^{p-1}(\Delta\tau), \mathbf{X}(\tau)\right]; \\ O_2^p = M_2\left[O_1^p(\tau), O_2^{p-1}(\Delta\tau), \ldots, O_{n-1}^{p-1}(\Delta\tau), O_n^{p-1}(\Delta\tau), \mathbf{X}(\tau)\right]; \\ \ldots \\ O_{n-1}^p = M_{n-1}\left[O_1^p(\tau), O_2^p(\tau), \ldots, O_{n-1}^{p-1}(\Delta\tau), O_n^{p-1}(\Delta\tau), \mathbf{X}(\tau)\right]; \\ O_n^p = M_1\left[O_1^p(\tau), O_2^p(\tau), \ldots, O_{n-1}^p(\tau), O_n^{p-1}(\Delta\tau), \mathbf{X}(\tau)\right]. \end{cases}$$

Система рівнянь може бути розв’язана відомими методами розв’язку систем лінійних рівнянь. У комбінаційних схемах після одноразового проходу, за умови правильності опису, розв’язок буде знайдено. Схема в цьому випадку буде остаточною. Для послідовнісних схем із зворотними зв'язками розв’язок буде знайдено тільки за декілька проходів.

Логічні моделі, залежно від складності опису елементів, можна розділити на декілька типів: булеві, трійкові, враховуючі затримки сигналів і багатозначні.

Булеві моделі працюють з двома логічними станами "1" або "0". Використовується математичний апарат двійкової алгебри.

У трійкових моделях додатково до двох логічних станів елементу використовують третій стан невизначеності Z. Застосовуються при описі елементів, стани яких в початкові моменти часу не визначені.

Моделювання з урахуванням часових затримок елементів дає достовірніші результати в порівнянні з бінарним моделюванням. При цьому враховується черговість надходження сигналів і одночасний логічний стан цих сигналів. Такий спосіб моделювання дає детальну картину обробки інформації і дозволяє визначити невідповідності, що викликають змагання або гонки сигналів.

При багатозначному моделюванні, додатково до описаних вище, враховуються статичні і динамічні характеристики елементів, з яких складається логічний елемент. Це дає інформацію про протікання реальних фізичних процесів в схемі.

Результатом моделювання цифрових систем є часові діаграми, які будуються автоматично по відомих сталих значеннях сигналів на входах і виходах елементів. Часові діаграми в більшості випадків будуються по двох осях. На горизонтальній осі розмічаються часові інтервали моделювання, по вертикальній осі відкладаються логічні стани сигналу. Таким чином, виходить розгорнена в часі зміна логічних станів. В деяких випадках на діаграмі стрілками показують фронти або спади сигналів, що впливають на поведінку інших

сигналів. Сигнали на часових діаграмах зображають у вигляді прямокутників або трапецій. Це пов'язано з необхідністю показати час фронтів і спадів. Приклад часової діаграми роботи ВІС паралельного вводу-виводу в режимі 0 показаний на рис. 1.5. Інформація передається з шини даних мікропроцесорної системи на зовнішні пристрої через порти А, В або С. В нашому прикладі ВІС працює в режимі виводу по всіх портах, хоча може мати і інші настройки. Комбінація сигналів А0 і А1 визначає порт запису, а $\overline{CS}$ визначає доступ до ВІС. Безпосередньо запис в порт відбувається по сигналу $\overline{WR}$. На діаграмі відмічені основні часові параметри: $\tau_{\overline{WR}}$ – тривалість сигналу запису, не менше 450 нс; $\tau_{A\overline{WR}}$ – час зсуву сигналу запису щодо сигналів адреси, не менше 50 нс; $\tau_{\overline{WRA}}$ – час збереження сигналів адреси після сигналу запису, не менше 50 нс; τ_{out} – час вибірки запису, не більше 550 нс. [9][1)], [10][2)].

Аналіз часових діаграм дозволяє перевірити коректність схеми, оцінити ефективність функціонування, частотні властивості. Особливі труднощі виникають при аналізі цифрових пристроїв на елементах з різними затримками, наприклад, для усунення ризиків збоїв, гонок і змагань сигналів. Для пристроїв, виконаних на елементах, що мають одну і ту ж затримку, шляхи проходження сигналу в схемі можуть бути різними. Це приводить до неодночасного приходу сигналів на входи окремих елементів. Такий ефект називається гонками сигналів. В результаті виникають ризики збоїв. Ризики збою можуть виникнути із-за наявності затримки вхідних сигналів, а також для деяких елементів з прямими і інверсними виходами.

Іноді схема із зворотними зв'язками може бути непрацездатна із-за змагання сигналів. Змагання сигналів – це поява короткочасових імпульсів під час

---

1) [9] Abstrakte Modellierung digitaler Schaltungen (VHDL vom funktionalen Modell bis zur Gatterebene) // K. ten Hagen. Springer, 1995.

2) [10] Peter J. Ashenden. University of Adelaide, South Australia: ftp: // ftp.cs.adelaide.edu.au / pub / VHDL-Cookbook (Mac, PC, PS); ftp: // bears.ece.ucsb.edu pub / VHDL; ftp: // du9ds4.fb9dv.uni-duisburg.de/pub/cad.

перемикання схеми з одного стану в інший. Виникає це явище унаслідок накладення сигналів з сумірними величинами затримок.

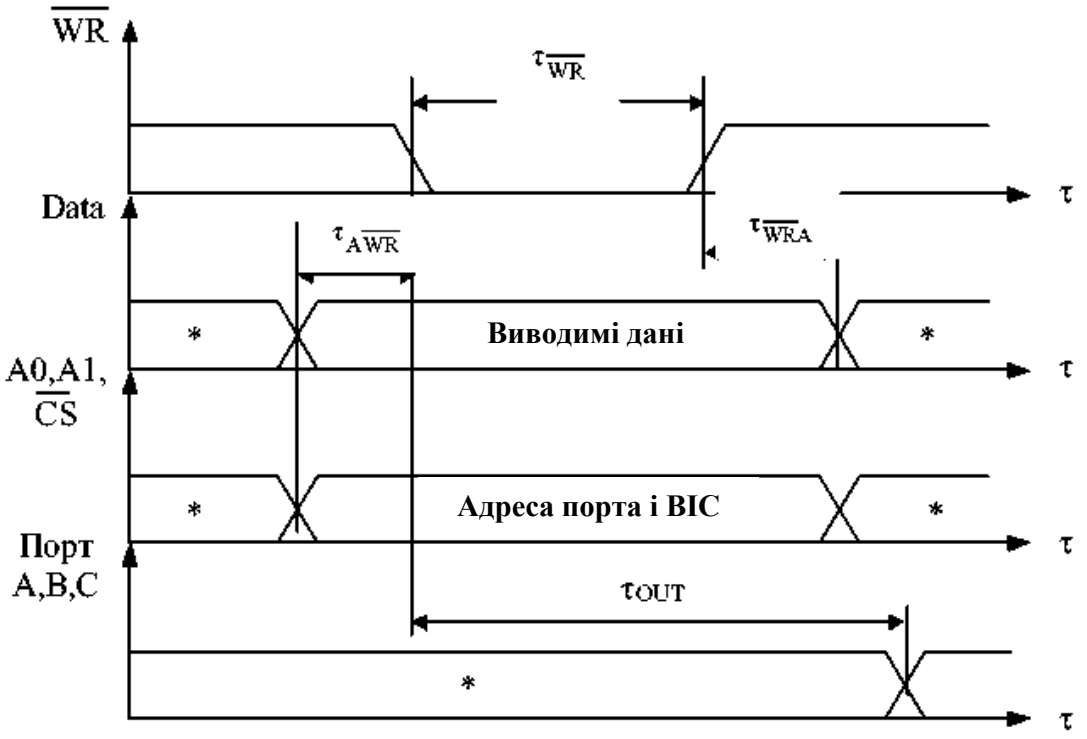


Рисунок 1.5 − Часова діаграма роботи ВІС 8255 на вивід в режимі 0

1.2 Способи логічного моделювання

Залежно від способу організації обчислювального процесу розрізняють два основні способи логічного моделювання цифрових пристроїв: потактовий (синхронний і асинхронний) і подієвий. [2] – [6] [1)]

[1)] [2] Томашевський В. М. Моделювання систем. К: Видавнича група ВНУ, 2005. 352 С.: іл. ISBN 966-552-120-9

[3] Бубенніков А. Н. «Моделирование интегральных микротехнологий, приборов и схем». М, «Высш. шк.», 1989. 320 с.

[4] Моделювання. Конспект лекцій з дисципліни «Моделювання систем» для студентів напряму підготовки 6.050103 – „Програмна інженерія”. / Укл.: Скітер І.С. Чернігів: ЧІБіП, 2015. 139 с.

[5] Дьяков И.А. Моделирование цифровых и микропроцессорных систем. Язык VHDL / И.А. Дьяков. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2001. 68 с.

[6] Стеценко, І.В. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В. Стеценко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. Черкаси : ЧДТУ, 2010. 399 с. ISBN 978-966-402-073-9

Потактовий спосіб моделювання. За цим способом вісь часу розбивається на рівні дискретні інтервали (такти). Поведінка функціональної схеми пристрою розглядається на цій тактованій осі часу. Усередині кожного такту будь-який елемент схеми знаходиться тільки в одному з двох станів: нульовому або одиничному. Елементи функціональної схеми заносяться у впорядкований або довільний список з вказівкою його номера і типу. Стан кожного елементу фіксується, як функція часу в пам'яті моделюючої системи.

В процесі моделювання здійснюється програмне сканування списку в певному порядку. У одному такті виконується декілька ітерацій, на кожній з яких формується вектор логічних станів елементів схеми. Кожна наступна ітерація характеризує стан схеми при зміненому стані елементів в поточному такті.

Багатократний циклічний перегляд списку проводиться до тих пір, поки дві сусідні ітерації не співпадуть. Отриманий збіг є ознакою закінчення моделювання на даному такті. Потім аналогічні дії проводяться на наступному такті. Моделювання завершується після вичерпання вхідних дій і часового інтервалу моделювання.

Особливості потактового способу:

1) вибір тривалості такту. Чим менша тривалість, тим ефективніша робота моделюючої програми. Час такту повинен бути кратним для всіх значень часових параметрів. Це залежить від тривалості фронтів сигналів , їх затримки, інтервалів між змінами вхідних сигналів. Тривалість такту повинна бути такою, щоб отримана часова діаграма була коректна при можливо меншому часі моделювання.

2) принцип побудови списку елементів і послідовності сканування. Отже, бажано використовувати послідовний принцип або такі алгоритми, які дають найменший час моделювання.

Синхронне потактове моделювання. Вважають, що для всіх елементів комбінаційної частини схеми затримки дорівнюють нулю. Затримки всіх елементів пам'яті однакові і рівні або кратні тривалості такту роботи схеми. В

цьому випадку емулюється тільки логіка роботи схеми. Моделювання здійснюється методом простих ітерацій по формулі

$$O^{(k)}=F[O^{(k-1)},X(\tau)],$$

де $O^{(k)}$ − значення вектора на к-й ітерації. Якщо $O^{(k)}=O^{(k-1)}$, те розв'язок знайдено, інакше виконується наступна ітерація. Нестійкий стан схеми (генерування) або помилки, допущені при проектуванні, характеризуються відсутністю збіжності ітераційного процесу. Критерій виходу з ітерацій − велика кількість тактів, що повторюються, або перевищення часового інтервалу моделювання.

Асинхронне потактове моделювання. У даному способі враховують, що елементи схеми мають різний час спрацьовування, який визначається затримкою τ проходження сигналу з входу на вихід елементу. Кожен такт роботи схеми розбивається на ряд мікротактів з тривалістю, відповідній затримці τ. Моделювання проводиться по мікротактах ітераційно. Для такого моделювання необхідно ввести два масиви логічних значень виходів елементів: $MO=[MO_1, M0_2 ,..., MO_n]$ − на попередньому мікротакті і $MN = [MN_1 ,MN_2 ,..., MN_n]$ − на подальшому мікротакті. Масиви мають однакові розмірності. В процесі моделювання, після закінчення кожної ітерації, відбувається збереження значень масиву MN_i в MO_i, На наступній ітерації за початковими даними масиву MO_i, вирішується система булевих рівнянь і результат записується в масив MN_i , Рішення вважається знайденим, коли значення масивів на попередньому і поточному мікротакті співпадають, ітераційний процес закінчується.

Асинхронний спосіб використовують як для аналізу на рівні інтегральних мікросхем, так і на рівні дискретних елементів, що входять до складу логічних елементів (діоди, транзистори, резистори).

Подієвий спосіб моделювання. Основна ідея полягає в обчисленні рівнянь тільки активізованих елементів, тобто таких, у яких хоч би на одному вході відбулася подія (змінилася вхідна змінна).

Алгоритм подієвого способу на кожному кроці встановлює, які елементи є активізованими, і викликає їх моделі. Це скорочує машинний час на аналіз схем.

Подією вважається будь-яка зміна стану схеми, наприклад, перехід її з одного логічного стану в інший. Якщо сигнали, подані на входи елементу, приводять до зміни його стану, то це подія. Як при потактовому способі складається список з інформацією про всі елементи.

Процес моделювання наступний. Серед елементів знаходять ті, котрі спрацьовують першими, наприклад, від вхідних сигналів. Кожен такий елемент зміною свого стану може викликати нові події. Це визначається існуючими зв'язками між елементами. Всі події можна розділити на дві групи: миттєві і події в майбутньому. Миттєві події відбуваються без затримки в часі. Події в майбутньому є множиною миттєвих подій. Коли всі події в множині миттєвих реалізовані, то вибирається така подія з множини майбутнього, котра найближча за часом до теперішнього моменту. Множина майбутніх подій переноситься в множину миттєвих, а час моделювання набуває значення часу цієї майбутньої події. Цей процес повторюється циклічно.

Моделювання можна вважати завершеним, якщо повністю реалізована група миттєвих подій і жодна подія з множини миттєвих не викликає події в майбутньому.

Алгоритм подієвого асинхронного моделювання розглянемо на прикладі абстрактної схеми (Рис. 1.6).

Перед виконанням моделювання повинні бути визначені елементи $g_i \in G^1$, зв'язки $s_i \in G^2$, вектори I_i ,O_i входів і виходів елементів g_i. Для кожного елементу повинні існувати логічна модель або опис набору логічних функцій M_i.

В процесі моделювання виконуються наступні кроки.

1. На входах схеми встановлюються початкові значення тестової послідовності $X(\tau) = X(\tau_0)$.

2. Перевіряється список елементів G1 і формується список поточних подій елементів gi, на виходах яких відбулася зміна стану сигналів у зв'язку з дією X(τ).

3. Із списку поточних подій вибираються для елементів gl відповідні їм моделі M. По моделі обчислюється реакція виходів Oi, із затримкою $\Delta\tau$. Формується список елементів gj, входи яких сполучені з виходами елементів gl. Потім вибираються Mj і проводяться обчислення, визначаються нові елементи g_k і так далі до тих пір, поки перестане виконуватися умова розповсюдження дії X(τ). Розповсюдження дії завершене, якщо немає більше елементів, на входах яких відбулася зміна станів.

4 Після переглядання списку поточних подій час моделювання збільшується на один такт $\tau = \tau + T$.

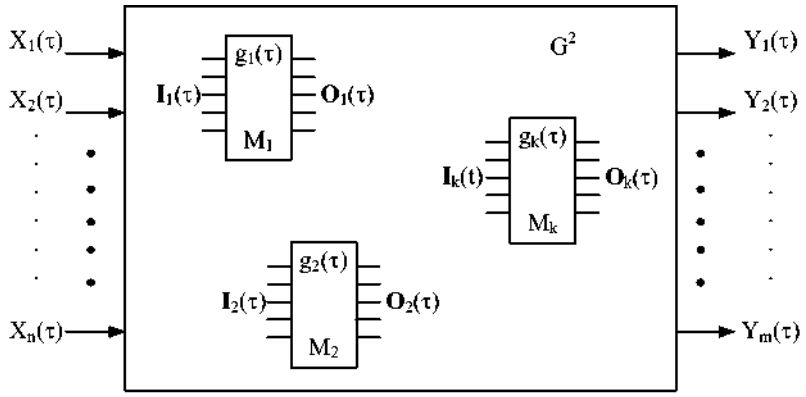


Рисунок 1.6 – Абстрактна схема цифрової системи

2 МОВИ ОПИСУ АПАРАТНИХ ЗАСОБІВ

Головний недолік традиційних методів проектування – це необхідність вручну транслювати опис проекту в набір логічних рівнянь. Цей крок може бути повністю виключений за допомогою мов опису апаратних засобів (HDL – hardware description language). Наприклад, більшість засобів HDL дозволяють використовувати кінцеві автомати (finite state machines) для послідовнісних систем і таблиці істинності для комбінаційних модулів. Такий опис проекту може бути автоматично конвертований в HDL-код, який реалізується засобами синтезу.

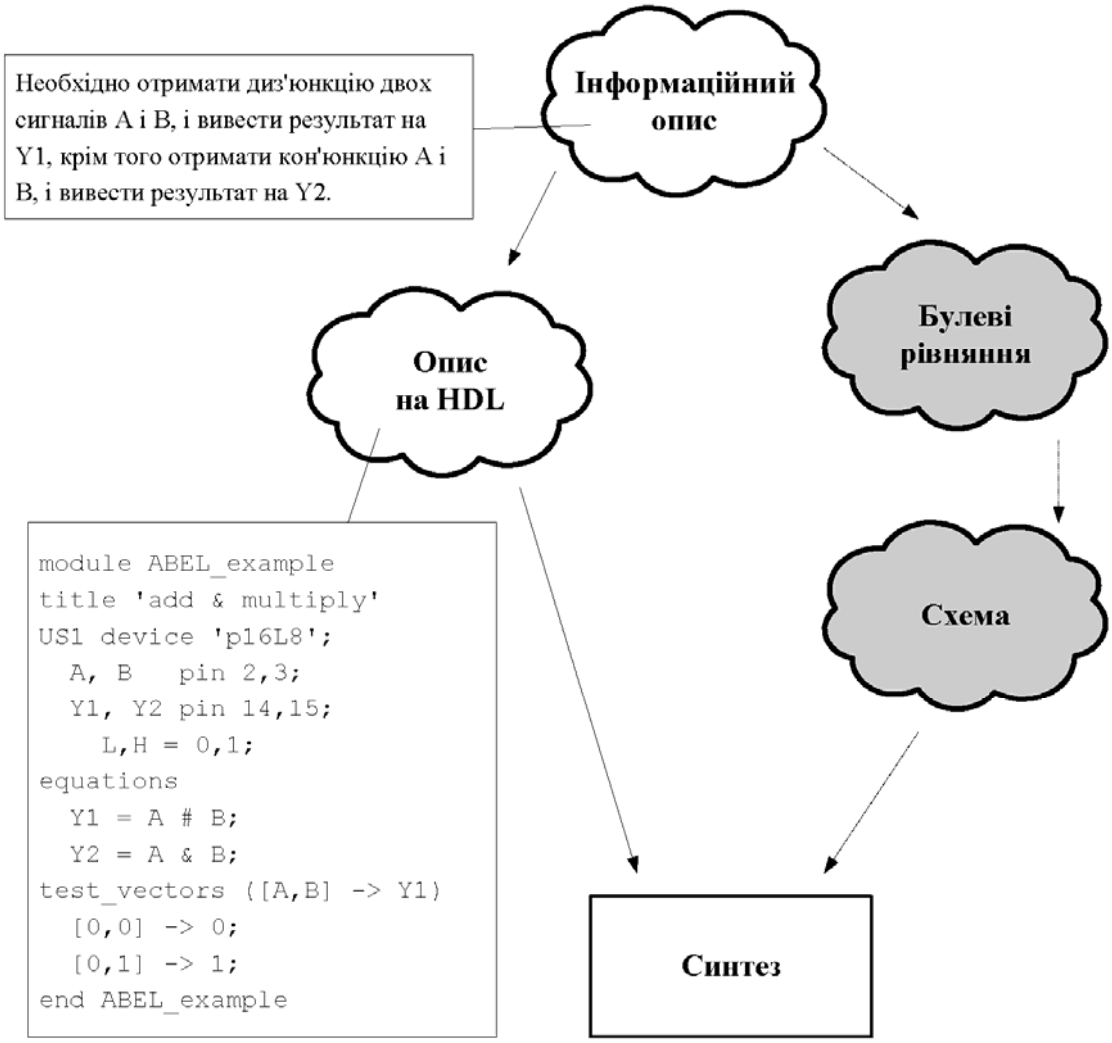


Рисунок 2.1 − Опис апаратних засобів

Мови опису апаратних засобів знайшли принципове застосування для програмованих логічних матриць (PLD) різної складності, від простих PLD до CPLD (complex PLD) та FPGA (Field Programmable Gate Array). Найбільш популярними мовами апаратних засобів є VHDL, Verilog та Abel. [7][1)], [11] [2)], [12][3)]

2.1 Оперування різними рівнями опису

Оскільки схеми стають дедалі складнішими, мають місце дві паралельні тенденції: перехід до більш загальних форм вхідних даних, таких як опис поведінки системи, використання комп'ютерних систем автоматизації проектування.

Існує декілька рівнів опису системи: від рівня кремнієвого кристала до складної специфікації системи. Ці рівні можна аналізувати як в термінах структури системи, так і в термінах її поведінки. Сьогоднішні технології настільки складні, що традиційні методи проектування не охоплюють усього спектру рівнів проектування. Засоби автоматизації проектування, які з'явились в останні роки, орієнтовані на найнижчий рівень (кремній) і на інтегрування процесу проектування.

Як система співпрацює із середовищем

Яку б функцію не виконувала система, вона має отримувати деякі вхідні дані і виводити деякі вихідні результати. Іншими словами, система має спілкуватись із середовищем.

Комунікаційна частина системи називається інтерфейсом. Системний інтерфейс описується у VHDL через блок інтерфейсу (entity) або просто

[1)] [7] «Леонов С.Ю. VHDL-технології проектування електронних пристроїв : навч. посіб. / С.Ю. Леонов, Т.В. Гладких, О.І. Баленко. К. : Вид-во "КАФЕДРА", 2014. 423 с.

[2)] [11] Ващишак, С. П. Мікропроцесорні системи : конспект лекцій / С. П. Ващишак. – Івано-Франківськ : ІФНТУНГ, 2013. 147 с.

[3)] [12] Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС Xilinx с применением языка VHDL / И.Е. Тарасов – Горячая Линия – Телеком, 2005. 256 с.

інтерфейс, який є базовим елементом проектування будь-якої системи. Як неможливо створити систему без інтерфейсу, так і неможливо створити VHDL-систему без блоку інтерфейса.

Для досягнення певної функціональності дані повинні якось перетворюватись всередині системи. Ця трансформація даних і вивід очікуваних функцій виконується внутрішньою частиною системи, або тілом системи, яке називається архітектурою (architecture).

Функціонування системи може бути як дуже простим (включення/виключення деякого перемикача), так і дуже складним (автопілот пасажирського літака). Однак в будь-якому випадку система може бути представлена набором тіла та інтерфейса.

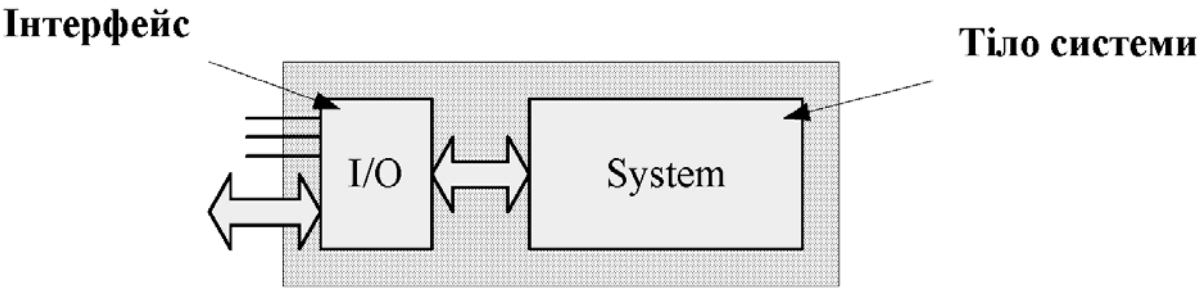


Рисунок 2.2 − Системний інтерфейс

2.2 Специфікація інтерфейсу

Оскільки починати будь-який проект необхідно з аналізу його середовища, не дивно, що блок інтерфейсу (entity), який описує інтерфейс між системою та її середовищем, є головною частиною кожного опису.

Не може існувати VHDL-специфікації якоїсь системи без декларації інтерфейсу. Крім того, все, що описано в інтерфейсі, автоматично стає видимим для усіх інших елементів проекту, зв’язаних з цим інтерфейсом. Більше того, ім’я системи завжди співпадає з іменем її інтерфейсу. Отже, проектування довільної системи у VHDL завжди необхідно починати з декларації інтерфейсу.

Склад інтерфейсу

Блок інтерфейсу забезпечує специфікацію інтерфейсу системи і звичайно складається з трьох елементів:

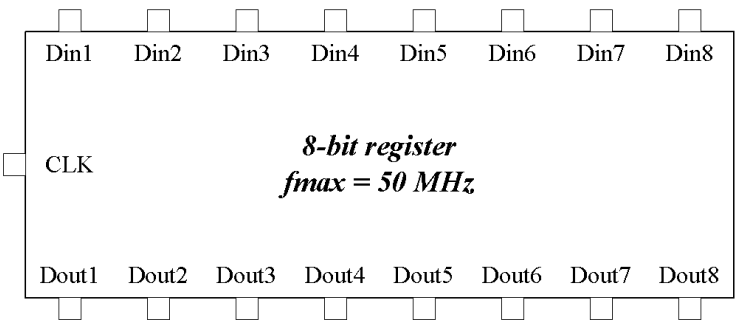
заголовку інтерфейсу,

параметрів системи (наприклад, ширина шини даних процесора або максимальна тактова частота),

з'єднання, за допомогою яких інформація передається в систему та з неї (входи та виходи системи).

Елементи інтерфейсу

Два ключових елементи довільного інтерфейсу (параметри та з'єднання) у кожному інтерфейсі декларуються окремо (рис. 2.3):




```
entity Eight_bit_register is

    LENGTH = 8
    fmax = 50 MHz
                              параметри

    D_IN  eight-bit input
    D_OUT eight-bit output
    CLK   one-bit input
                              з'єднання

end entity Eight_bit_register;
```

```
entity Eight_bit_register is

    generic ( LENGTH = 8
              fmax   = 50 MHz
            );
                              параметри

    port ( D_IN  eight-bit input
           D_OUT eight-bit output
           CLK   one-bit input
         };
                              з'єднання

end entity Eight_bit_register;
```

Рисунок 2.3 – Параметри та з'єднання інтерфейсів

2.3 Архітектура як опис тіла системи

Архітектура пристрою описується у VHDL як архітектура інтерфейсу:

```
entity  Tvset  is
end  entity  Tvset;
architecture  TV2000  of  Tvset  is
end  architecture  TV2000;
```

Типи архітектурних описів

Кожна система може бути описана як в термінах її функціональних можливостей (поведінки), так і в термінах її структури, відповідно тому, яка інформація про систему потрібна.

Звичайно засоби синтезу працюють з обидвома типами архітектурного опису: перший, який задає очікуване функціонування, має бути специфікований і формалізований, після чого він трансформується в структурний еквівалент, що більш придатний для засобів синтезу. Частина цієї трансляції може бути проведена автоматично, однак повний функціональний (поведінковий) синтез на сьогодні ще не реалізований.

Поведінковий опис

Функціональний опис визначає те, що система, як очікується, буде робити. Наприклад, необхідно описати, як телевізор відреагує на вхідні сигнали від кнопок пульта дистанційного керування.

Поведінкова специфікація взагалі – це опис виходів як реакцій на вхідні дані.

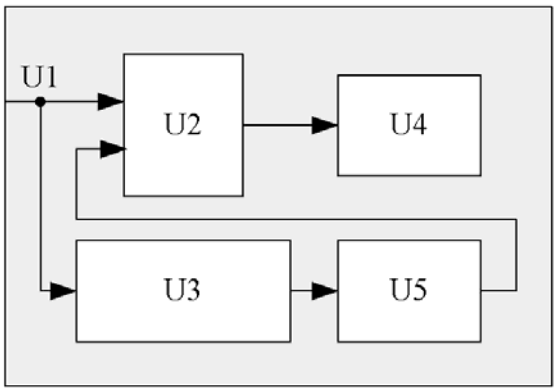
Необхідно зазначити, що поведінковий опис не відповідає, як ці функціональні можливості системи повинні бути виконані.

```
entity  Tvset  is
port  (ON,OFF  :  one-bit  input
...
) end  entity  Tvset;
architecture  TV2000  of  Tvset  is
...
if  ON  then  turn  on  the  system
if  OFF  then  turn  off  the  system
... end  architecture  TV2000;
```

Структурний опис

Структурний опис не зачіпає функціонування системи, а замість цього визначає компоненти, що повинні використовуватися, і як вони повинні бути з'єднані, щоби досягти очікуваних результатів. Іншими словами, він описує внутрішню структуру системи.

Структурний проект набагато легше синтезувати, ніж поведінковий, тому що він звертається до конкретних фізичних компонентів. Однак, створення структурного опису системи (рис. 2.4) дещо складніше, оскільки вимагає детального знання компонентів і принципів їх дії для підвищення його ефективності.




```
entity Tvset is
   port (
         ...
         )
end entity Tvset;

architecture TV2000 of Tvset is
   ...
   U1,U5 -> U2 -> U4
   U1 -> U3 -> U5
   U3 -> U5 -> U2
   U2 -> U4
   ...
end architecture TV2000;
```

Рисунок 2.4 – Структурний опис

Інерційні затримки

Інерційна затримка (рис. 2.5) є типовою для більшості реальних систем, в зв’язку з чим у VHDL ця модель є затримкою за замовчуванням. Оператор after автоматично вважає затримку інерційною. Характерною властивістю цієї моделі є те, що дві послідовних зміни вхідного сигналу будуть проігноровані, якщо час між ними коротший, ніж задана затримка.

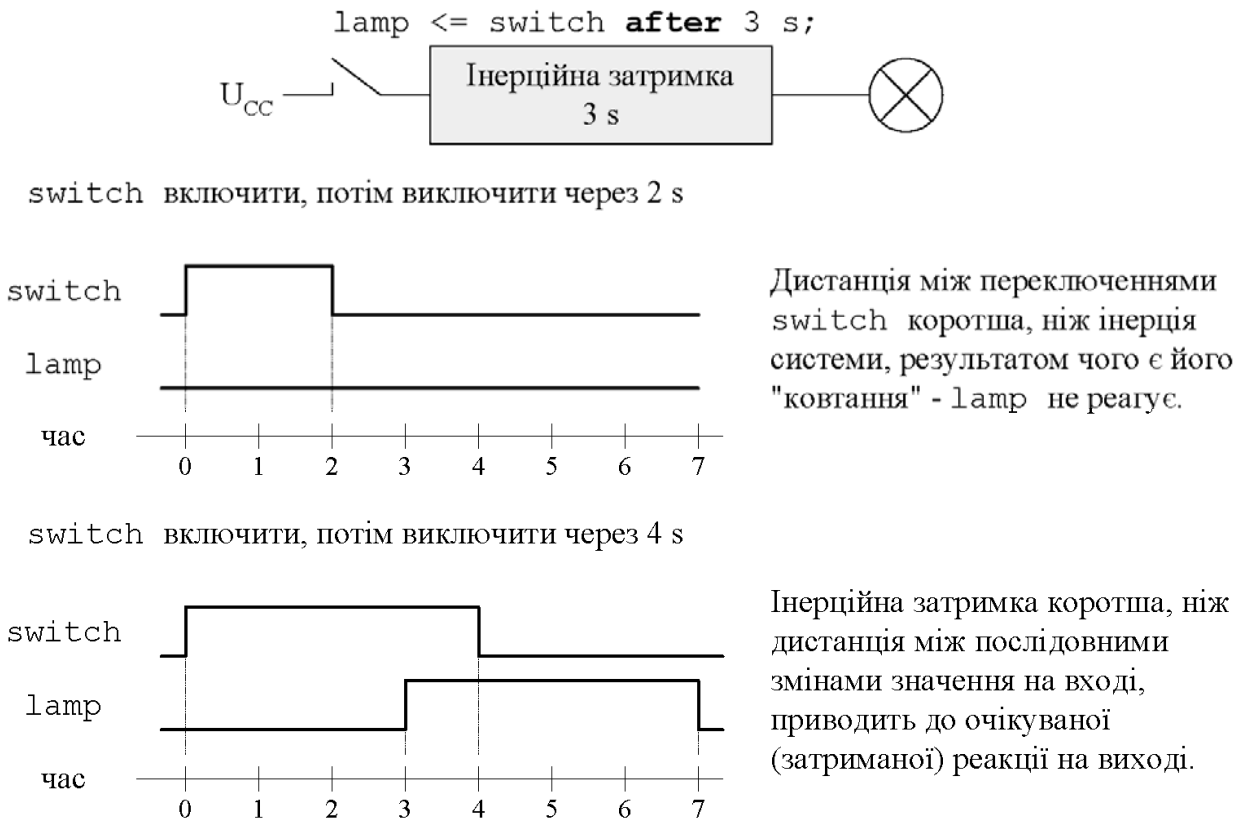


Рисунок 2.5 – Інерційна затримка

Транспортні затримки

Лінія передачі, по якій передаються імпульси довільної ширини ніякої інерціїї не має. Виходячи з поведінки лінії передачі, необхідно мати іншу модель затримки. Ця модель (рис. 2.6) застосовується для транспортування сигналів без будь-яких змін, називається моделлю транспортної затримки. Щоб відрізнити її від інерційної , прийнятої за замовчуванням, використовується слово transport.

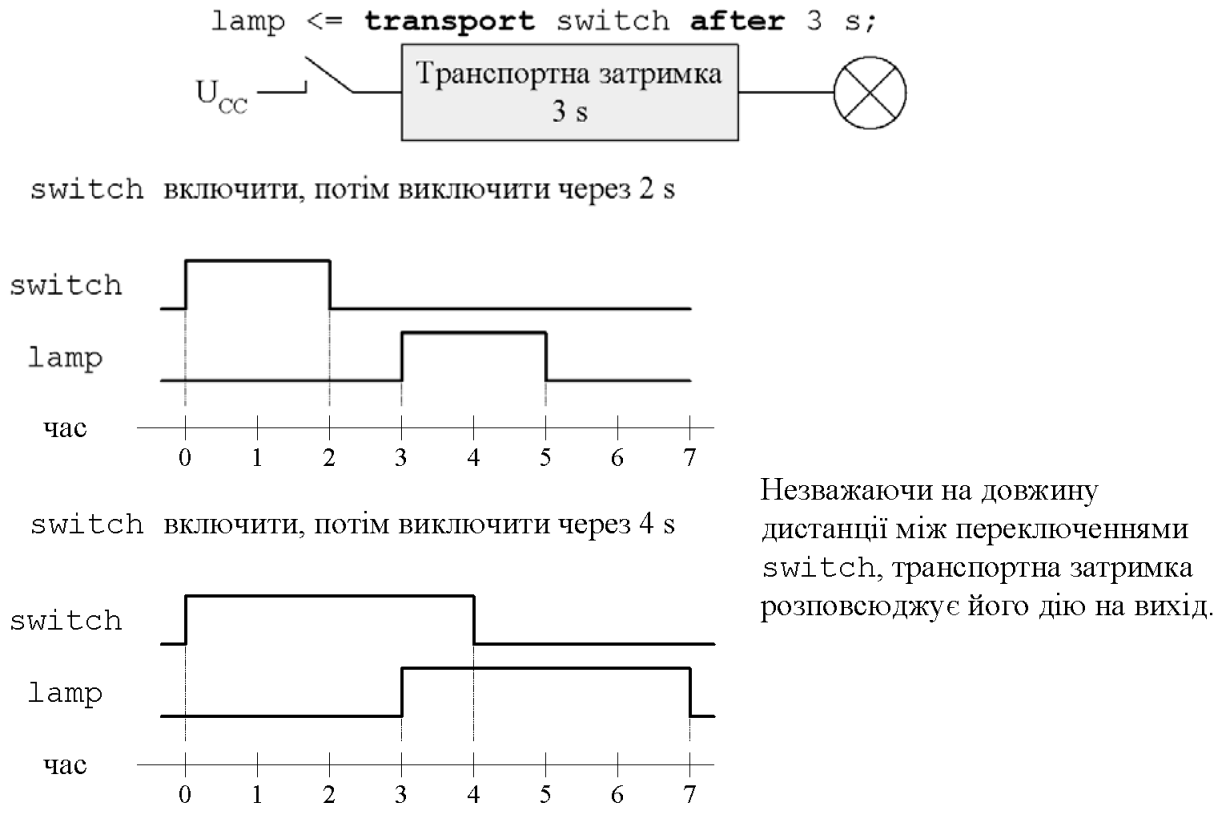


Рисунок 2.6 – Транспортна затримка

Порівняння інерційних і транспортних затримок

Моделі інерційної і транспортної затримки є достатніми для опису у VHDL довільної фізичної системи. Вони мають наступні головні подібності та відмінності (табл. 2.1), (рис. 2.7):

Таблиця 2.1 − Подібності та відмінності інерційної і транспортної затримки

Інерційна затримка	Транспортна затримка
Є затримкою за замовчуванням у VHDL і не вимагає ніяких додатко-вих декларацій	Вимагає використання ключового слова transport

Інерційна затримка	Транспортна затримка
Не поширює імпульси, коротші, ніж задана затримка	Поширює всі зміни вхідного сигналу, незалежно від того як швидко і як часто вони відбуваються
Описується за допомогою оператора after після якого вказується значення часу	
Може застосовуватись до сигналів довільного типу	

```
Out1 <= Input after 3 s;
Out2 <= transport Input after 3 s;
```

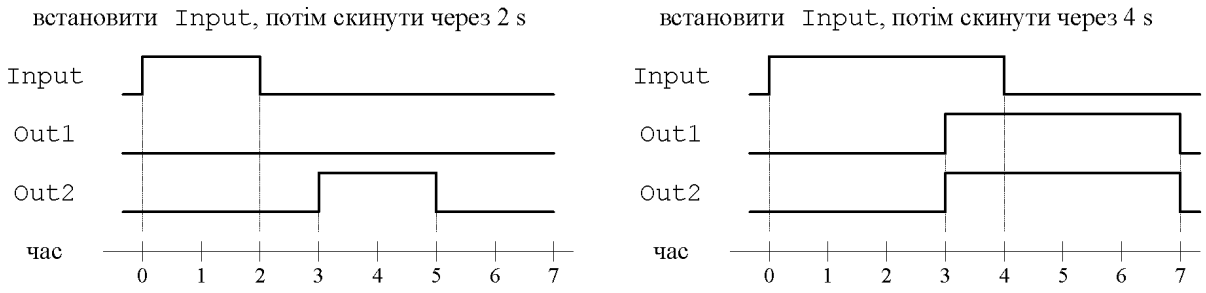


Рисунок 2.7 − Подібності та відмінності інерційної і транспортної затримки

3 МОДЕЛЮВАННЯ ЛОГІЧНИХ СХЕМ

Вимоги, що пред'являються до моделювання логічних схем, включають паралельне виконання операторів, використання затримок сигналів і розділення методів моделювання, що використовуються до комбінаційних і послідовнісних схем.

3.1 Моделювання комбінаційних схем

Схема вважається комбінаційною, якщо в ній немає зворотних зв'язків або елементів пам'яті. Моделі комбінаційних схем в мові VHDL можуть бути описані різними командами: логічними, арифметичними, з використанням операцій відношення і операторів одночасного присвоєння, умовних переходів і запуску процедур. Розглянемо деякі варіанти схем і різні способи їх представлення.

Опис логічної схеми, показаної на рис. 3.1, можна представити логічними операторами: and, or, nand, nor, xor, nxor, not. Затримка сигналу в кожному елементі складає 20 нс.

```
entity logsh_1 is
   port (x1, x2, x3, x4: in bit;  y: out bit);
end logsh_1;
architecture p1 of logsh_1 is
   signal e1,e2: bit;
begin
e1 <= x1 and x2 after 20 ns; --      паралельне виконання
e2 <= x3 xor x4 after 20 ns;
y <= e1 or e2 after 20 ns;
end p1;
```

Якщо декілька елементів працюють паралельно (рис. 3.2), то в описі можна використовувати вектор:

```
entity logsh_2 is
  port (a, b: in bit_vector (0 to 3);  y: out bit_vector (0 to 3));
end logsh_2
architecture p2 of logsh_2 is
begin
  y <= a and b after 20 ns;
end p2;
```

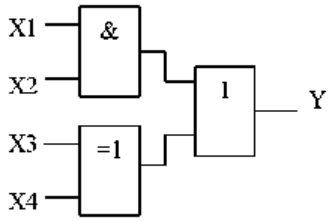


Рисунок 3.1 − Послідовна обробка

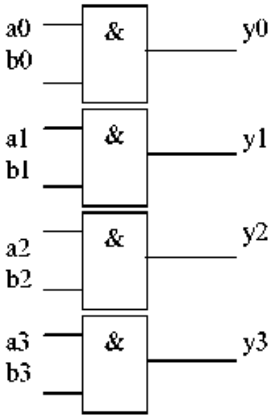


Рисунок 3.2 − Паралельна обробка

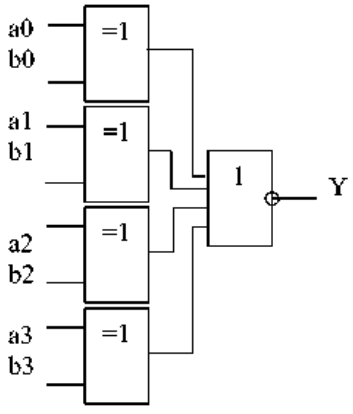


Рисунок 3.3 − Схема компаратора

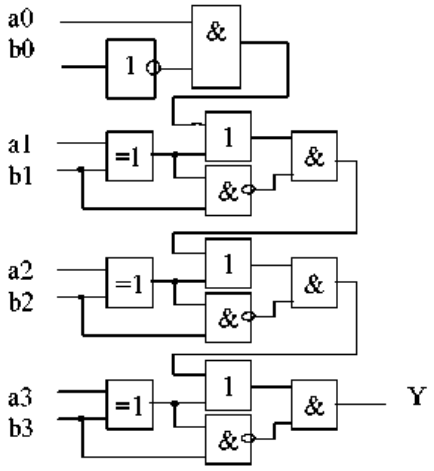


Рисунок 3.4 − Схема компаратора

У цих прикладах відбувається пряма трансляція логічної схеми в оператори мови. Щоб отримати реальну картину розповсюдження сигналів, враховуються затримки у всіх елементах схеми. Послідовність "трансляції" схеми в оператори програми співпадає з послідовністю проходження сигналів через елементи.

Схеми, призначені для порівняння сигналів, раціонально моделювати операторами відношень або впорядкування. На рис. 3.3 показана схема компаратора, що визначає порозрядне порівняння сигналів A і B. Якщо хоч би один розряд $a_i \neq b_i$, то на виході схеми − рівень логічної одиниці, інакше рівень логічного нуля.

Схема, показана на рис. 3.4, виконує порівняння $a \geq b.$. Фрагмент проекту для компараторів може бути записаний у вигляді:

```
entity relat_1 is      -- відношення a ≠ b
   port (a, b: in bit_vector (0 to 3);  y: out boolean);
end relat_1;
architecture cmp of relat_1 is
begin
   y <= a = b after 40 ns;
end cmp;
entity relat_2 is      -- відношення a ≥ b
   port (a, b: in integer range 0 to 3;  y: out boolean);
end relat_2;
architecture cmp of relat_2 is
begin
   y <= a >= b after 140 ns;
end cmp;
```

Недолік такого опису полягає в тому, що доводиться вказувати приблизну затримку сигналів, рівну сумі затримок кожного елементу. У першій схемі компаратора кожен сигнал проходить два елементи, отже, сумарне значення 40 нс (по 20 ns на кожен елемент). У другій схемі в максимальному випадку сигнал проходить сім елементів.

Логічні схеми суматора або схеми віднімання, а тим більше множення, ділення, піднесення до ступеня дуже складні в логічних елементах принципової схеми. Окрім суматора, інші схеми застосовуються украй рідко. Транслятор VHDL робить спеціальну оптимізацію, але можна застосувати її на етапі розробки проекту. [13][1)]

Проілюструємо логіку суматора (рис. 3.5) фрагментом програми з використанням пакету оператора друку оголошень:

```
package add_vect is
  type small_int is range 0 to 2;
end add_vect ;
use add_vect.all;
entity arithmetic is
   port (a, b: in small_int;  c: out small_int);
end arithmetic;
architecture exmpl of arithmetic is
signal c: bit;
begin
    c <= a + b after 80 ns;
end exmpl;
```

[1)] [13] J. Armstrong. Chip Level Modelling in VHDL. Prentice Hall, 1988. P. 148.

Як і в попередньому прикладі, недолік опису полягає в моделюванні затримки.

VHDL забезпечує одночасні твердження для створення умовної логіки, мультиплексорів , демультиплексорів, схем вибору. Схема, показана на рис. 3.6, легко представляється оператором *if*, а на рис. 3.7 − оператором вибору *case*. У приведених фрагментах програм реалізовані послідовне і одночасне твердження відповідно до логічних схем.

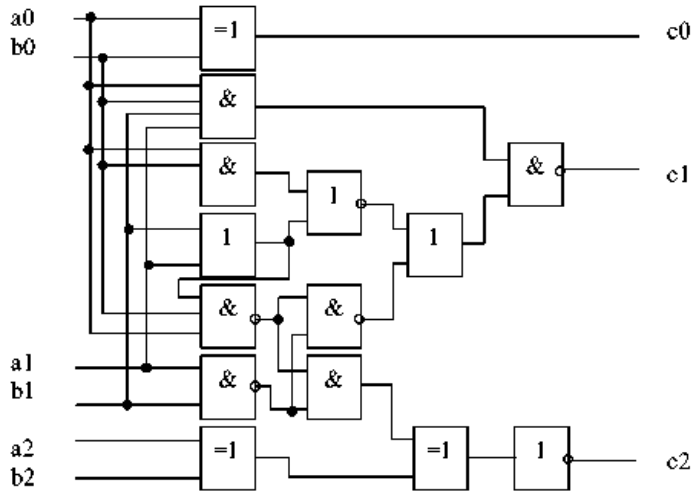


Рисунок 3.5 − Логічна схема суматора

```
entity control is
  port (a, b, c: in boolean;  y: out boolean);
end control;
architecture p of control is
begin
   y <= b when a else  c;
end p;
```

Для вибраного значення сигналу a, b, c, d за кодом sel дозволяється проходження на вихід y:

```
entity controls is
   port (sel: bit_vector (0 to 1);  a,b,c,d : bit;  y: out bit);
end controls;
architecture p of controls is
begin
  with sel select
     y <= c when b"00",
     y <= d when b"01",
     y <= a when b"10",
     y <= b when others;
end p;
```

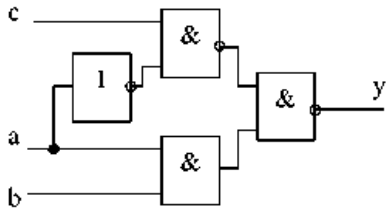


Рисунок 3.6 − Логічна схема для if

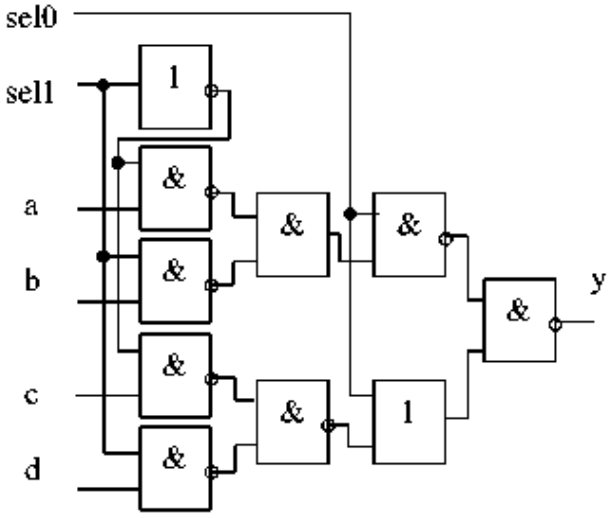


Рисунок 3.7 − Логічна схема для case

Ті ж функції можуть бути виконані з використанням послідовних тверджень і відбуватимуться усередині процесу. Умова в умовному операторові повинна бути обов'язкове булевого типу.

Наступний приклад ілюструє розв’язок тієї ж задачі, але з використанням умовного оператора:

```
entity control is
  port (a, b, c:  boolean;  y: out boolean );
end control;
architecture exmpl of control is
begin
 process (a, b, c)
 variable n: boolean;
  begin
    if a then
        n := b;
    else
        n := c;
    end if;
   y <= n after 60 ns;
end process;
end exmpl;
```

Використання оператора вибору (або вибране призначення сигналу) компілюється швидше і відпрацьовує логіку з меншою затримкою розповсюдження, чим використання вкладених умовних операторів.

VHDL вимагає, щоб всі можливі умови були представлені в операторі вибору. Щоб гарантувати це, використовуйте пропозицію "інші" в кінці оператора вибору, щоб виключити будь-які невизначені умови.

Наступний приклад ілюструє такий варіант:

```
entity control is
  port (sel: bit_vector(0 to 1);  a,b,c,d : bit;  y: out bit);
end control;
architecture p of control is
begin
process (sel,a,b,c,d)
begin
case sel is
 when b"00"  => y <= c;
 when b"01"  => y <= d;
 when b"10"  => y <= a;
 when others => y <= b;
end case;
end process;
end p;
```

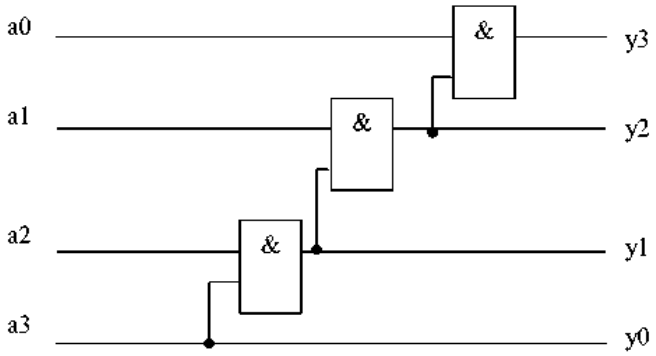


Рисунок 3.8 − Каскадне включення

Якщо є фрагменти схеми, що повторюються, або каскадне включення однакових елементів, то досить організувати цикл або виклик підпрограм, маючи у такому разі опис тільки одного елементу.

Якщо можливо, спочатку для циклу генерують діапазони. Інакше логіка усередині циклу може копіюватися для всіх можливих значень діапазонів циклу. Це може мати складну реалізацію в елементах схем.

Розглянемо приклад (рис. 3.8) каскадного включення елементів 2І.

```
entity loop_sh is
   port (a: bit_vector(0 to 3);  y: out bit_vector(0 to 3));
end loop_sh;
     architecture p of loop_sh is
   begin
   process (a)
   variable b:bit;
   begin
     b := '1';
     for i in 0 to 3 loop
       b := a(3-i) and b;
       y(i) <= b;
       report "Loop number = " &  integer'image(i);
     end loop;
    end process;
end p;
```

```
entity sr_1 is
   port (a, b,c: in bit_vector (5 downto 0 );
   ctl : integer range 0 to 2**5 -1;
    w,x,y: out bit_vector (5 downto 0));
end sr_1
architecture p of  sr_1 is
begin
  w <= a sll ctl;   --   зсув вліво і заповнення  '0'
  x <= a sra ctl;   --   зсув вправо і запис   a'left [ a(5) ]
  y <= a rol ctl;   --   циклічний зсув вліво
end p;
```

В деяких випадках при моделюванні необхідно враховувати третій стан елементу. Для цього передбачено підключення стандартних пакетів std_logic, визначених в IEEE. Std_logic_1164, або використання порожнього оператора null, щоб вимкнути драйвер. Перший метод застосовується тільки "до типу std_logic, другий застосовується до будь-якого типу. Перший метод зазвичай використовується частіше.

```
library ieee;
use ieee.std_logic_1164.all;
entity tbuf is
  port (enable : boolean;
       a : std_logic_vector(0 to 4);
       m : out std_logic_vector(0 to 4));
end tbuf;
architecture p of tbuf is
begin
  process (enable, a)
     if enable then
       m <= a;
    else
      m <= 'Z';
     end if;
 end process;
end p;
```

Якщо треба описати внутрішню шину, то можна скористатися другим методом.

```
package p_bus is
  subtype bundle is bit_vector (0 to 4);
end p_bus;
use work.p_bus.all;
entity tbuf is
  port (enable: boolean;  a: bundle;  m: out bundle bus);
end tbuf;
architecture p of tbuf is
begin
  process (enable, a)
  begin
   if enable then
     m <= a;
   else
     m <= null;
   end if;
  end process;
end p;
```

Застосування порожнього оператора до сигналу виду шини вимикає драйвер. Коли в умовному операторові виконається умова, буде сформований буфер з трьома станами.

3.2 Моделювання послідовнісних схем

Для моделювання послідовнісної логіки в мові VHDL немає спеціальних ресурсів, проте, правильне використання операторів затримок, очікування, охоронних конструкцій в блоках і процесів дозволяє досить добре описувати схеми такого типу. Динаміка зміни сигналів (фронти і зрізи) реєструється атрибутами мови. Дуже добре складати проекти, застосовуючи при описі схем їх представлення у вигляді кінцевих автоматів. У практиці програмування склалися певні підходи до опису схем тактуваня і запису-читання даних, скидання і установки станів.

Базові компоненти послідовнісних схем − це один або декілька блоків комбінаційних схем і деяка кількість зворотних зв'язків. Затримки сигналів в таких схемах можуть бути тактованими (синхронними) або залежними від часу проходження сигналу (асинхронні).

Стан синхронних схем може відображатися змінними, наприклад, накопичення значення в лічильнику. Але залежно від стану синхронізуючих сигналів значення змінній присвоюється з певною затримкою вихідного сигналу. У асинхронних схемах механізм зворотних зв'язків моделюється аналогічно синхронним схемам, міняється тільки семантика змінного зворотного зв'язку. Для синхронних схем змінна зворотного зв'язку представляє період тактової частоти, а для асинхронної схеми − затримку розповсюдження.

Одним з часто використовуваних в послідовнісних схемах елементом є тактований тригер. Такий тригер має вхід даних Data, вхід тактового сигналу Clk, вхід початкової установки Reset або Set і вихід даних Q. Тригер передає дані з входу на вихід тільки по фронту (рівню або зрізу) тактового сигналу. Моделювання таких елементів доцільно проводити з використанням охоронних конструкцій в блоках. Для описаного вище тригера фрагмент програми:

```
D: block ((Clk='1' and not Clk'Stable) or Reset='1')
    begin
    Q<=quarded '0' when Reset='1'; else
    Data when Clk='1' and not Clk'Stable; else
    Q;
end block D;
```

В даному прикладі дані записуються по фронту Clk. Якщо Reset має високий рівень, то значення тригера скидається в нуль. Будь-які зміни на вході і відсутність переходу з нуля в одиницю на тактовому вході не змінюють значення, що зберігається в тригері.

Наступний приклад ілюструє запис інформації по сигналу Clk, що поступає на вхід схеми. Одночасно виконується деяка обробка вхідних даних. У всіх випадках сигнал "y" зберігає поточне значення, якщо немає змін рівня тактового сигналу.

```
process (Clk, a, b)
begin
  if Clk then
    y <= a and b;
  end if;
end process;
```

Вказаний фрагмент програми можна розмістити усередині функції або процедури і викликати при кожному записі фіксованих даних. Такий підхід зручний і тоді, коли необхідне моделювання паралельної обробки сигналів, наприклад, одночасної фіксації даних в регістрі, побудованому на декількох тригерах.

```
procedure my_latch (signal Clk, a, b :  boolean; signal y : out boolean)
begin
  if Clk then
    y <= a and b;
  end if;
end;
```

Звернення до процедур, наприклад, з ім'ям proc_A усередині процесу може бути записане так:

```
L1: proc_A ( clock, input1, input2, outputA );
L2: proc_A ( clock, input3, input4, outputB  );
```

При призначенні сигналу можна використовувати умовні вирази. Тут слід звернути увагу на те, що "y" використовується в двох напрямах:

```
y <= a and b when Clk else y;
y <= a and b when Clk;
```

Якщо тригер повинен бути чутливий до фронту тактового сигналу Clk, то умова запису доповнюється атрибутом, наприклад:

```
process (Clk)   – список сигналів чутливості
begin
 if Clk and Clk'event then  – визначення зрізу сигналу
y<=a and b;
end if;
end process;
```

Тут сигнал "y" зберігає значення, якщо Clk не змінюється (немає переходу від нуля до одиниці). В операторі процесу може бути відсутнім список сигналів чутливості, тоді застосовують оператор очікування виконання умови:

```
process    -- немає списку сигналів
begin
  wait until not Clk ;  -- очікування зрізу сигналу
    y <= a and b;
end process;
```

Якщо необхідна паралельна обробка, то, як і в попередньому прикладі, оголошуємо процедуру і викликаємо її при необхідності:

```
procedure my_ff (signal Clk, a, b :  boolean; signal y : out boolean)
begin
  if not  Clk  and Clk'event then   -- фронт тактового  сигналу
    y <= a and b;
  end if;
end;
```

Як і в попередньому прикладі, "y" використовується двонаправлено:

```
y <= a and b when Clk and Clk 'event  else y;
y <= a and b when Clk and Clk 'event;
```

Вхід скидання або установки значень тригера по зміні сигналу або по його рівню іноді описують у вигляді функції:

```
function rising_edge (signal s : bit ) return boolean is
begin
  return s = '1' and s'event;
end;
```

При використанні цієї функції D-тригер може бути описаний як:

```
q<=d when rising_edge(Clk);
```

Умова перевірки тактового сигналу, як правило, проста і може включати елементарні логічні операції, наприклад, коли важлива не тільки зміна Clk, але і додатковий сигнал дозволу. Такий же результат досягається складеним умовним оператором. Але в таких випадках існує імовірність пропуску тактового сигналу при часових обмеженнях.

Початкове значення тригера можна скинути або встановити зовнішніми вхідними сигналами Reset або Set. Така дія може бути синхронною або асинхронною. У текст програми додається ще одна змінна або константа. Нижче показаний приклад синхронної установки тригера по сигналу Set.

```
process (Clk)
begin
  if Clk and Clk'event then       -- фронт  сигналу
   if Set then
     y <= true;              -- у типа boolean
   else
     y <= a and b;
   end if;
  end if;
end process;
```

Приклад скидання і установки групи біт по сигналу init:

```
process

  begin
    wait until Clk                  -- фронт Clk
      if init then
  y <= 7;                           -- у типа integer
      else
  y <= a + b;
      end if;
  end process;
```

Поведінка асинхронного скидання або установки описується незалежно від управління по тактовому сигналу. Просто додається ще одна умова і дія з цієї умови. Розглянемо приклад скидання значення для використання в паралельній обробці по сигналу Reset:

```
y <= false when Reset else a when Clk and Clk 'event  else y;
```

Повний опис тригера з асинхронним скиданням або асинхронною установкою, де обробка сигналів незалежна і паралельна, представимо так:

```
process (Clk, Reset)
begin
  if Reset then
    q <= false;                -- у типу boolean
  else
    if Clk and Clk'event then    -- фронт clock
     q <= d;
    end if;
  end if;
end process;
procedure ff_asуnc_set (signal Clk, a, Set:  boolean;
          signal q : out boolean)
begin
  if Set then
    q <= true;
  elsif Clk  and Clk'event then  -- фронт clock
    q <= a;                      --  ввід даних
  end if;
end;
```

Щоб описувати поведінку і асинхронної установки і асинхронного скидання, ми просто додаємо другого умовного оператора:

```
    q <= false when Reset else
     true when Set else
     d when Clk and Clk 'event;
--    Reset і Set у паралельній обробці
    process (Clk, Reset, Set)
    begin
      if Reset then
        q <= false;      -- q типу  boolean
      elsif  Set then
        q <= true
      else
        if Clk and Clk'event then    -- фронт clock
          q <= d;
        end if;
              end if;
    end process;
```

Щоб описувати поведінку асинхронного завантаження, треба замінити змінну Set або Reset сигналом або виразом.

Завантаження даних може виконуватися з використанням послідовних операторів:

```
process (Clk, load_ctl,load_data)
begin
```

```
    if load_ctl = '1' then
       q <= load_data;
    elsif rising_edge(Clk) then
       q <= d;
    end if;
  end process;
```

Таким чином, опис тригера відрізняється по завантаженню даних, скиданню і установці початкових значень. В процесі моделювання необхідно мати чітке уявлення не тільки про логіку роботи схеми, але і за яким принципом: синхронному або асинхронному виконуються дії.

4 МОДЕЛЮВАННЯ НА РІВНІ МІКРОСХЕМ

При складанні проектів на елементній базі, що складається з мікросхем високого ступеня інтеграції, наприклад, IC, BIC або HBIC, необхідно користуватися моделями того ж рівня. Модель рівня інтегральної схеми − це поведінкова модель логічного блоку. Точний опис затримок сигналів в таких моделях здійснюється без використання описів нижчого рівня.

Основні вимоги до моделей рівня інтегральних схем формулюються таким чином. [14][1)]

З метою отримання правильно функціонуючої схеми часові параметри входів і виходів повинні бути точно змодельованими. Виходячи з припущення, що схема сама є базовим примітивом, не допускається її декомпозиція на простіші структурні примітиви. Модель схеми реалізується у вигляді послідовності мікрооперацій, що описують логіку роботи схеми, без описів вентильного або регістрового рівня. Розробник моделі може використовувати текстові описи, структурні схеми, специфікації і часові діаграми, а також таблиці станів і діаграми станів.

Для того, щоб мати точне представлення часових параметрів схеми, модель повинна мати достатній ступінь деталізації за часом.

Модель схеми можна умовно розділити на три частини: вхідну, внутрішню і вихідну. Вхідна частина містить вхідні специфікації на кількість входів, час встановлення і утримання сигналів, а також мінімальну тривалість імпульсів. Внутрішня частина містить специфікації на можливі мікрооперації обробки сигналів (програмовані схеми), пам'ять, затримки сигналів в основних внутрішніх колах. Вихідна частина містить специфікації на кількість вихідних сигналів і їх часові параметри. Структура моделі схеми має три важливих

[1)] [14] Brown S. Fundamentals of Digital Logic with VHDL (2nd edition) / S. Brown Z. Vranesic. McGraw-Hill, 2004. 939 p.

складових: види затримок сигналів, мінімальну енергію і функціональне розбиття схеми.

Серед затримок сигналів можна виділити: просту, у зворотному зв'язку і в колах з точкою прийняття розв’язку. Проста затримка моделюється шляхом затриманого призначення однією змінною або сигналу іншому сигналу. Затримане призначення реалізується оператором after. Затримка зворотного зв'язку представляється поєднанням оператора затриманого присвоєння і конструкції процесу. Вихідний параметр при цьому використовується як вхідний в списку сигналів запуску процесу. Затримка в колах з точкою прийняття розв’язку − складніша модель затримки. Для її моделювання застосовується механізм процесів. Проходження сигналів в таких схемах розглядається у декілька етапів. На першому і третьому етапах сигнал затримується у вхідних і вихідних колах. Другий етап відповідає "центральній" частині схеми, де ухвалюється рішення пропустити сигнал далі або блокувати його. Можлива обробка сигналу мікропрограмою схеми. Прикладом такої схеми може бути регістр з трьохстабільними виходами і сигналами стробування і дозволу запису даних. Приклади з мікропрограмною обробкою: BIC и паралельного або послідовного вводу-виводу.

Модель схеми структурно може бути представлена у вигляді графа, а загальні функції розбиті на підфункції. Кожен вузол графа реалізується в окремому процесі і виконує одну підфункцію. Дуги графа позначають шляхи проходження сигналів між вузлами-процесами Крім того, кожен вузол графа може мати своє функціональне розбиття. Іншими словами, під функція має декілька вхідних в неї функцій. Наприклад, підфункція обробляє сигнали X, Y, Z. Але дії виконуються окремо над сигналами X і Y, X і Z, Y і Z. В цьому випадку вузол графа слід розділити на три частини, а підфункцію реалізувати у вигляді трьох окремих процесів з відповідними списками сигналів чутливості.

Часові параметри вхідних н вихідних сигналів представляються інерційною або транспортною затримкою. Тим самим досягається часове моделювання. Застосування інерційних затримок можна контролювати за допомогою операторів assert і report, оскільки виникає імовірність пропустити імпульси

малої тривалості. Це, насамперед, пов'язано з часом передустановки і утримання, а також з порушенням вимог до найменшої тривалості сигналу. Параметром оператора assert є вираз булевого типу. У ньому також можна використовувати атрибути сигналів. Параметром оператора report є будь-яке повідомлення, поміщене в подвійні лапки. Як правило обидва оператори використовуються разом і записуються послідовно один за одним. Оператори контролю часових параметрів можуть включатися в опис інтерфейсу або архітектурне тіло об'єкту проекту. Зона дії в першому варіанті глобальна, в другому − локальна, тільки в рамках одного архітектурного тіла.

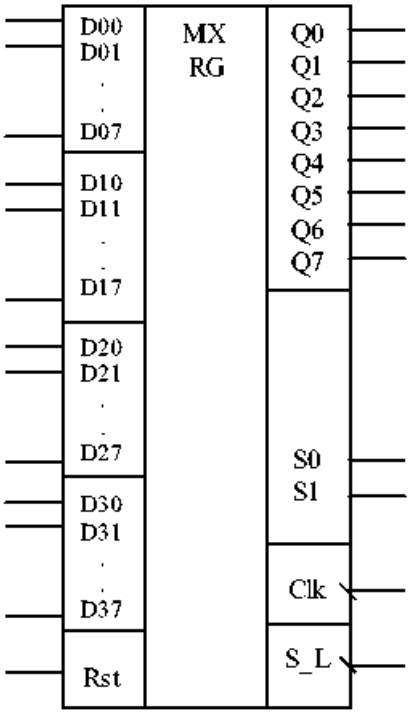


Рисунок 4.1 − Умовне графічне позначення ВІС

Розглянемо приклад розробки проекту. Хай маємо ВІС, що включає мультиплексор і регістр зсуву (рис. 4.1). Принцип роботи ВІС наступний. Чотири незалежні восьмибітові входи (D00-D07 D10-D17, D20-D27, D30-D37) комутуються на восьмибітовий вихід (Q0-Q7) по сигналах вибору SO і S1.

Комбінації сигналів вибору визначають відповідний їм вхід, наприклад, якщо

SO = 0 і S1 = 0, то Qi = DOi; SO = 1 і SI = 0, то Qi = Dli; SO = 0 і SI = 1, то Qi=D2i; SO = 1 і SI = 1, то Qi =D3i.

Дані з вибраного таким чином входу записуються у вихідний регістр Qi по фронту сигналу Clk. Зріз і рівень цього сигналу не впливають на записані дані. Зсув зафіксованих даних вліво відбувається по фронту сигналу S_L. Високий рівень сигналу Rst скидає в нульове значення вміст регістра Qi.

Для моделювання відомий інтерфейс ВІС, і на основі опису функціонування можна побудувати поведінкову модель. Проведемо функціональне розбиття мікросхеми і побудуємо граф моделі процесів. На основі аналізу функцій ВІС можна виділити наступні її частини: вхідні канали даних і мультиплексор, регістр зсуву і вихідні канали даних. У кожній з частин сигнал проходить відповідну обробку і затримується на деякий час. Один з можливих варіантів графа моделі показаний на рис. 4.2.

Процес Mux відповідає комутації одного з вхідних каналів на вхід регістра із затримкою розповсюдження Del_sel. Процес Load відпрацьовує функцію завантаження даних в регістр по сигналу Clk. Час запису даних Del_load. Регістр зсуву функціонально розбитий на схему фіксації даних і зсуву даних, оскільки операції запису і зсуву незалежні один від одного і активізуються різними сигналами. Час зсуву даних Del_shift. Специфікація вихідних каналів необов'язково приводить до окремого процесу.

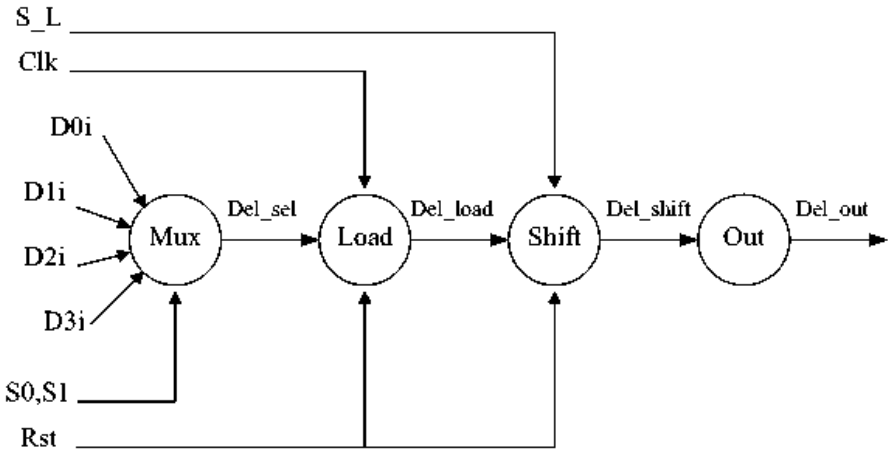


Рисунок 4.2 Граф моделі процесів ВІС

Але повторюючи загальну схему, можна ввести ще один вузол Out і відповідний йому процес. Час проходження сигналу від регістра зсуву до виходу схеми через вихідні ключі рівний Del_out.

По побудованому графові тепер можна скласти проект схеми на мові VHDL. Найбільш простий варіант проекту, де немає часових специфікацій сигналів і всі процеси об'єднані в один, може виглядай так:

```
package typedef is
  subtype byte is bit_vector (7 downto 0);
 end;
use work.typedef.all;
entity Bis_data is
   port   (clk,rst,s_l : boolean;
       s0, s1 : bit;
       d0, d1 ,d2, d3 : byte;
       q : out byte);
end Bis_data;
architecture BIS of Bis_data is
    constant Del_sel, Del_load, Del_shift, Del_out : time := 10 ns;
    signal reg,shft : byte;
begin
    process (clk,rst,s_l)
    begin
        if rst then                    -- асинхронне скидання
              reg <= x"00";
              shft <= x"00";
elsif clk and clk'event then    -- такт завантаження
    case s0 & s1 is              --  мультеплексування
    when b"00" => reg <= d0 after Del_sel;
    when b"10" => reg <= d1 after Del_sel;
    when b"01" => reg <= d2 after Del_sel;
    when b"11" => reg <= d3 after Del_sel;
    end case;

  if s_l and s_l'event then              -- такт зсуву
        shft <= shft rol 1 after Del_shift;
  else
        shft <= reg after Del_load;
  end if;
end if;

 end process;
 q <= shft after Del_out;
end BIS;
```

Слід зазначити, що розглянутий вище приклад хороший для розуміння логіки роботи ВІС і не повинен застосовуватися для моделювання роботи. Щоб правильно виконати моделювання, треба відповідно до графа визначити три

процеси. Якщо буде потрібно контроль часових параметрів, то спочатку необхідно привести часову діаграму з вказаними в ній часовими характеристиками сигналів (рис. 4.3) і "вбудувати" її в проект.

На часові й діаграмі показані значення основних часових характеристик, одночасно діаграма ілюструє принцип роботи мікросхеми. У початковий момент часу на вхід асинхронного скидання мікросхеми поступає сигнал Rst, що має найменшу тривалість τ_{Rst}. Через деякий час, не показано на діаграмі, на входи D поступають дані. Найменший час утримання даних позначений як τ_{DATA}. За цей час повинне виконатися завантаження даних в регістр, через мультиплексор. Після того, як дані на вході матимуть стійке значення, інтервал τ_{DS}, по сигналах SO і S1 відбувається вибір каналу, комутованого на вхід регістра. Протягом часу τ_{Sei} (часу утримання), при незмінних значеннях SO і S1 по фронту сигналу Clk відбувається фіксація даних в регістрі.

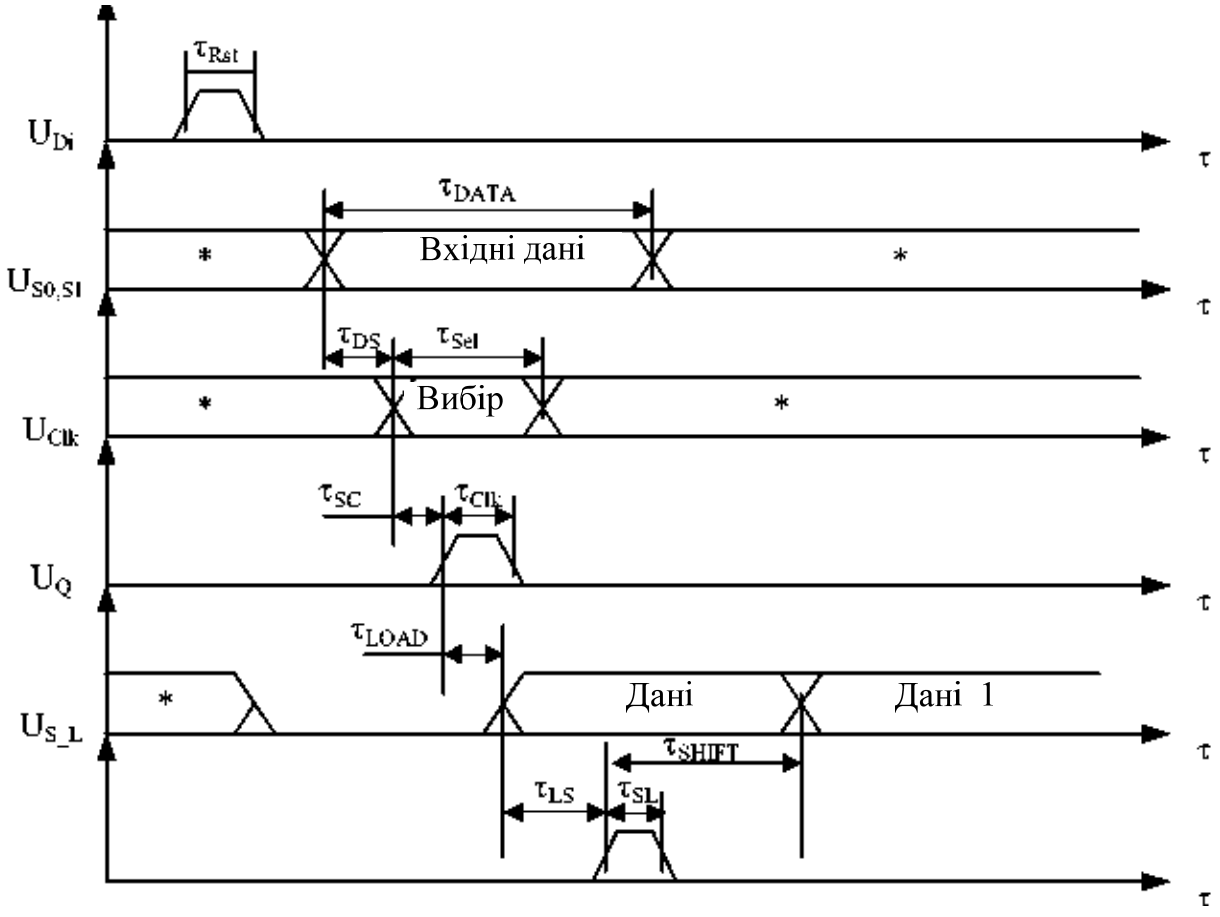


Рисунок 4.3 − Часова діаграма роботи ВІС

Найменший час між надходженням сигналів SO і S1 фронтом Clk позначено як τ_{sc}. Тривалість імпульсу запису повинна бути не менше τ_{Clk}. Протягом часу τ_{LOAD} дані записуються в регістр. Через інтервал часу, найменше значення якого рівне τ_{LS} допускається зсув даних по фронту S_L. Операція зсуву може бути проведена не обов'язково тільки після запису нових даних. Мінімальна тривалість сигналу зсув − τ_{SL}. Оброблені таким чином дані з'являться на виходах мікросхеми не швидше, ніж через час τ_{SHIFT}.

Проект на мові VHDL відповідно до графа моделі і перевірки деяких часових параметрів приведений у додатках.

Оскільки немає проміжного зберігання даних після мультиплексора, то дві функції мультиплексування і завантаження об'єднано в одну і оформлено в проекті в одному процесі. Окремо записаний процес асинхронного скидання. Процес Out не має сигналів запуску і тому завжди активний. В процесі моделювання перевіряються тільки вагомі часові характеристики. Це тривалість імпульсів і стабільність сигналів комутації.

5 МОДЕЛЮВАННЯ СИСТЕМ

При моделюванні систем всі моделі представляються на вищому поведінковому рівні. Це відповідає таким компонентам, як мікропроцесори, мікросхеми пам'яті, інтерфейсні BIC і так далі. Всі компоненти системи об'єднані схемою з'єднань, локальними і загальними шинами, системними магістралями і окремими лініями зв'язку. Таким чином, при моделюванні системи повинні враховуватися специфіка компонентів і цілісність всієї системи. [15][1], [16][2]

5.1 Моделювання схемних з'єднань

У мові VHDL модель з'єднань компонентів для передачі і прийому інформації представляється у вигляді архітектурного тіла з описом інтерфейсу компонентів. Розглянемо приклад системи (рис. 5.1), що складається з двох компонентів A і B, сполучених локальною шиною L_{AB}. Хай локальна шина має вісім провідників. Система має вісім вхідних ліній і 16 вихідних.

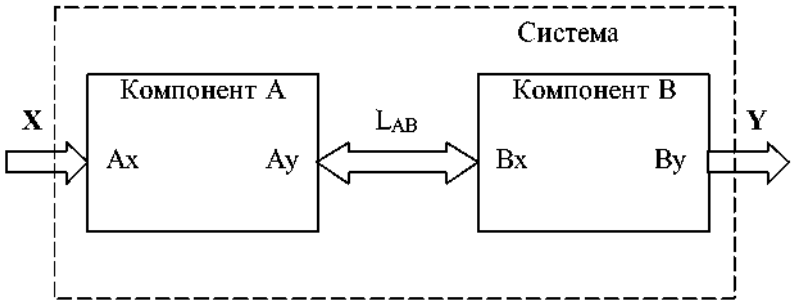


Рисунок. 5.1 − Структурна схема системи з двох компонентів

Тоді інтерфейс компонентів A і B має наступний опис:

---

[1] [15] Скляр В. В. Тестирование и разработка диверсных программируемых логических контроллеров на базе ПЛИС с использованием среды функционального программирования / В. В. Скляр, В. С. Харченко, А. С. Панарин // Радіоелектронні і комп'ютерні системи. 2014. № 1. С. 29–41. Режим доступу: http://nbuv.gov.ua/UJRN/recs_2014_1_6.

[2] [16] Скляр В. В. Тестирование программируемых логических контроллеров на базе ПЛИС с использованием среды функционального программирования / В.В. Скляр, В.С. Харченко, А.С. Панарин // Системи обробки інформації. 2014. № 1(117). С. 44-55.

```
entity A is
   port (Ax : in bit_vector(0 to 7); Ay : out bit_vector(0 to 7));
end A;
entity B is
   port (Bx : in bit_vector(0 to 7); By : out bit_vector(0 to 15));
end B;
```

Опис системи включає її інтерфейс і архітектурне тіло, в якому записується інтерфейс компоненту і конкретизація зв'язків між компонентами. Для системи, що розглядається в прикладі, можна записати:

```
entity System is
   port (X : in bit_vector(0 to 7); Y : out bit_vector(0 to 7));
end System;
architecture System_AB of System is
  signal Lab : bit_vector(0 to 7);  ----    опис ліній зв'язку
component  A is                     ----    опис компоненти A
   port (Ax : in bit_vector(0 to 7); Ay : out bit_vector(0 to 7));

 end component;
 component  B is                     ----    опис компоненти B
    port (Bx : in bit_vector(0 to 7); By : out bit_vector(0 to 15));
 end component;
 begin
   ----       виконання процесів, блоків
   A port map(X,Lab);   ----  конкретизація компоненти системи
   B port map(Lab,Y);   ----  конкретизація компоненти системи
end System_AB;
```

Таким чином, об'єднання компонентів в систему моделюється описом інтерфейсів і передачею відповідних значень сигналів компонентам.

Якщо для з'єднання використовуються окремі одиничні лінії, то відповідні їм змінні можуть мати бітовий тип. Всі подальші дії аналогічні. Групове представлення сигналів, коли воно відповідає схемі, вигідніше, ніж бітове. Це пов'язано з тим, що замість однієї події при одночасній зміні декількох сигналів в групі доведеться обробляти окремо всі N подій.

У тексті програми оператори port map записуються усередині архітектурного тіла за операторами процесів. Приведений в прикладі запис оператора визначається позиційним зіставленням елементів відображення портів в описі компонентів. Те ж саме можна зробити за допомогою ключової відповідності:

S1: A **port map**(Ay=>Lab, Ax=>X);

S2: B **port map**(By=>Y, Bx=>Lab);

Позиції тут не грають ролі, оскільки відповідність забезпечується зіставленням імен.

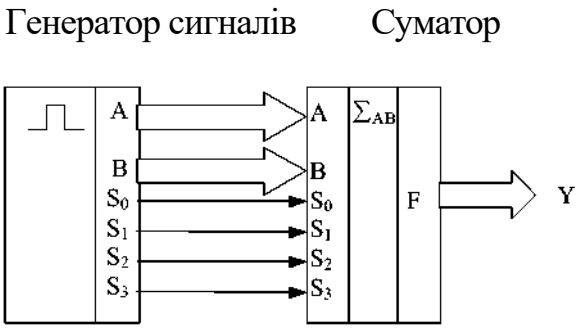


Рисунок 5.2 − Приклад системи генератор-суматор

При моделюванні схемних зв'язків на високому рівні для представлення інформації можна використовувати інші типи даних (напр. перечислимий тип_:

type control **is** (mov, lda, ani,…, hlt);
signal data_bus : control;

Інформація управління представляється в мнемонічному вигляді. Такий підхід полегшує моделювання і звільняє від контролю низькорівневих деталей. В деяких випадках застосовують числовий тип, наприклад, subtype control is integer range 0 to 31. Це зручніше в порівнянні з цілим типом із-за спрощення контролю помилок.

Розглянемо приклад моделювання системи, що складається з двох компонентів: суматора і генератора сигналів. Принцип роботи системи достатньо простий (рис. 5.2).

Генератор формує значення для операндів A і B з сигналами управління функцією роботи суматора S0-S3. На виході системи маємо результат Y. Компонент "суматор" названий так умовно і це видно з таблиці істинності 5.1.

Таблиця 5.1

S_3	S_2	S_1	S_0	Функция F
0	0	0	0	$\overline{A}$
0	0	0	1	$\overline{A+B}$
0	0	1	0	$\overline{A}B$
0	0	1	1	0
0	1	0	0	$\overline{AB}$
0	1	0	1	$\overline{B}$
0	1	1	0	$A \oplus B$
0	1	1	1	$A\overline{B}$
1	0	0	0	$\overline{A} + B$
1	0	0	1	$\overline{A+B}$
1	0	1	0	B
1	0	1	1	AB
1	1	0	0	1
1	1	0	1	$A+\overline{B}$
1	1	1	0	$A + B$
1	1	1	1	A

Затримка проходження сигналів через схему складає 57 нс. Проект системи на мові може бути записаний у вигляді (див. додатки).

Сигнали Si представлені як фізичні провідники з часом розповсюдження сигналів від одного компоненту до іншого 5 нс. Слід відмітити, що опис схемних зв'язків має певні труднощі, особливо при моделюванні багатозначної логіки.

6 ОЦІНКА ПРОЕКТНОГО РІШЕННЯ

В даному розділі проводиться обґрунтування доцільності розробки програмного забезпечення.

Метою дипломної роботи є розгляд питань моделювання цифрових і мікропроцесорних систем. Показані традиційні методи моделювання і автоматизованого проектування з використанням процедурних мов програмування і спеціалізованої мови VHDL САПР «Xilinx». Викладений опис основних компонентів мови VHDL і методів моделювання для різних рівнів представлення схем, від логічних елементів до систем, що складаються з набору ВІС. Приведені приклади схем і представлення їх на мові VHDL.

Результатом розробки є програмний продукт, який реалізує ці підходи.

Програмування здійснюється на персональному комп'ютері з характеристиками: Celeron 2,4 MHz/512 Mb RAM/80Gb HDD/Video 64 Mb Ge-Force 2MX400 / Monitor Samsung 17".

Організаційний ефект від використання проектного рішення полягає в полегшенні навчання студентів університету.

Соціальний ефект від використання проектного рішення полягає у вивільненні часу, для інших предметів та можливості самостійно освоювати методи корекції слідкуючих систем.

Економічний ефект полягає в підвищенні ефективності навчання студентів конкретному предмету, а саме в:

- скороченні термінів навчання;
- покращенні якості навчання;
- зростання віддачі студентів кафедри.

6.1 Визначення комплексного показника якості

В якості аналогу для проектованої системи була вибрана система проектування «Matlab & Simulink Products v.6». Дана система має ряд переваг : добре розроблений інтерфейс; широкі можливості застосування.

Комплексний показник якості визначається шляхом порівняння показників якості проектованої системи та обраного аналогу. Вибір показників якості здійснюється експертним методом. Номенклатура основних груп показників якості наступна:

- показники надійності;
- ергономічні показники;
- показники стандартизації та уніфікації;
- показники призначення.

Комплексний показник якості проектованої підсистеми визначаємо методом арифметичного середньозваженого з формули:

$$П_к = \sum_{i=1}^{N} \frac{K_i}{100} q_i, \qquad (6.1)$$

де N − кількість одиничних показників, прийнятих для оцінки якості проектованої системи;

K_i коефіцієнт вагомості i-го одиничного показника якості, який визначає його відносну вагомість, % (коефіцієнти вагомості визначаються експертним методом, сума їх повинна дорівнювати 100 %);

q_i − відносні безрозмірні показники якості, визначені порівнянням числових значень одиничних показників проектованої підсистеми і аналога за формулами:

$$q_i = \frac{П_{1i}}{П_{2i}}, \qquad (6.2)$$

або

$$q_i = \frac{П_{1i}}{П_{2i}}, \qquad (6.3)$$

де $П_{1i}$, $П_{2i}$ − кількісні значення i-го одиничного показника якості відповідно проектованої підсистеми і аналога.

З попередніх двох формул вибирається та, в якій збільшення відповідає покращенню показника якості проектованої підсистеми.

Таблиця 6.1 − Визначення комплексного показника якості проектованої системи

Показники	Числове знач. показників		Відносн. показник якості, q	Коефіцієнвагомості K_i, %	Kq, %
	Аналог	Проект, система			
1 .Показники призначення 1.1. Швидкість завантаження проекту, (сек.)	10	2	5	10	50
1.2. одночасне обслуговування користувачів, (осіб) *	1	1	1	5	5
1.3. Можливість застосування програми при навчанні, (бали)	5	15	3	40	120
2.Показники надійності					
2.1. Стійкість системи до некоректних дій користувача, (бали)	5	10	2	10	20
2.2. Стійкість до збоїв системи	5	5	1	5	5
3.Ергономічні показники 3.1. Зручність представлення даних, (бали)	5	10	2	10	20
3.2. Час, що витрачає користувач на вивчення системи, (год.)	24	12	2	5	10
3.3 Інформативність, (бали)	5	7.5	1.5	3	4.5
Наявність системи підказок (HELP), (одиниць)	2	1	1	2	2
4.Естетичні показники 4.1. Наочність і зрозумілість пунктів меню і підказок, (бали)	2	4	2	5	10

Показники	Числове знач. показників		Відносн. показник якості, q	Коефіцієнвагомості $K_{i,\%}$	$Kq,\%$
	Аналог	Проект, система			
5.Показники стандартизації і уніфікації 5.1. Відповідність структури системи згідно з встановленими стандартами, (%)	50	90	1.8	5	9
5.2. Використання процедур вводу-виводу, (%)	75	75	1	5	5
6.Показники безпеки 6.1. Показник електробезпеки, (бали)	5	5	1	5	5
Всього				110	265,5

Отже, комплексний показник якості програмного продукту *Пяк*=2,65

В даному розділі дипломної роботи проведена оцінка проектного рішення. Розроблена програма є кращою за вибраний аналог по показниках якості та ефективності. В результаті проведення розрахунків визначено показник конкурентоздатності, який є більшим 1. Крім того, дослідження передбачає отримання суттєвого економічного ефекту в процесі експлуатації. Отже, розробка і впровадження даного програмного забезпечення є економічно доцільним.

ВИСНОВКИ

У дипломній роботі розглянуті практичні питання реалізації конкретних алгоритмів і пристроїв, питання моделювання цифрових і мікропроцесорних систем. Проведено порівняння традиційних методів моделювання з автоматизованим проектуванням з використанням процедурних мов програмування і спеціалізованої мови VHDL.

У роботі проведено огляд основних компонентів мови VHDL і методів моделювання для різних рівнів представлення схем, від логічних елементів до систем, що складаються з набору ВІС. Приведені приклади схем і представлення їх на мові VHDL. Мова VHDL має засоби для опису паралельних асинхронних процесів, регулярних структур. VHDL зручна для створення великих проектів вищого і системного рівнів.

Було проведено поведінковий опис системи, що складається з двох компонентів: суматора і генератора сигналів на мові VHDL, що дало можливість виконати автоматичний синтез. При моделюванні схемних зв'язків на високому рівні для представлення інформації використано перечислимий тип даних. Об'єднання компонентів в систему моделюється описом інтерфейсів і передачею відповідних значень сигналів компонентам.

Групове представлення сигналів, коли воно відповідає схемі, вигідніше, ніж бітове. Тому що замість однієї події при одночасній зміні декількох сигналів в групі доведеться обробляти окремо всі N подій, а це швидкодія.

При моделюванні систем всі моделі представляються на вищому поведінковому рівні. Це мікропроцесори, мікросхеми пам'яті, інтерфейсні ВІС і так далі. Всі компоненти системи об'єднані схемою з'єднань, локальними і загальними шинами, системними магістралями і окремими лініями зв'язку. Таким чином, при моделюванні системи повинні враховуватися специфіка компонентів і цілісність всієї системи.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1 Сергиенко А.М. VHDL для проектирования вычислительных устройств / А.М. Сергиенко. К.: ЧП "Корнейчук", ООО "ТИД "ДС", 2003. 208 с.

2 Томашевський В. М. Моделювання систем. К: Видавнича група ВНУ, 2005. 352 С.: іл. ISBN 966-552-120-9

3 Бубенніков А. Н. «Моделирование интегральных микротехнологий, приборов и схем». М, «Высш. шк.», 1989. 320 с.

4 Моделювання. Конспект лекцій з дисципліни «Моделювання систем» для студентів напряму підготовки 6.050103 – „Програмна інженерія”. / Укл.: Скітер І.С. Чернігів: ЧІІБіП, 2015. 139 с.

5 Дьяков И.А. Моделирование цифровых и микропроцессорных систем. Язык VHDL / И.А. Дьяков. Тамбов: Изд-во Тамб. гос. техн. ун-та, 2001. 68 с.

6 Стеценко, І.В. Моделювання систем: навч. посіб. [Електронний ресурс, текст] / І.В. Стеценко ; М-во освіти і науки України, Черкас. держ. технол. ун-т. Черкаси : ЧДТУ, 2010. 399 с. ISBN 978-966-402-073-9

7 «Леонов С.Ю. VHDL-технології проектування електронних пристроїв : навч. посіб. / С.Ю. Леонов, Т.В. Гладких, О.І. Баленко. К. : Вид-во "КАФЕДРА", 2014. 423 с.

8 Преснухін Л. Н., Воробйов И. В., Шішкевіч А. А. «Расчет элементов цифровых устройств.» М, «Высш. Шк»., 1991. 526 с.

9 Abstrakte Modellierung digitaler Schaltungen (VHDL vom funktionalen Modell bis zur Gatterebene) // K. ten Hagen. Springer, 1995.

10 Peter J. Ashenden. University of Adelaide, South Australia: ftp: // ftp.cs.adelaide.edu.au / pub / VHDL-Cookbook (Mac, PC, PS); ftp: // bears.ece.ucsb.edu pub / VHDL; ftp: // du9ds4.fb9dv.uni-duisburg.de/pub/cad.

11 Ващишак, С. П. Мікропроцесорні системи : конспект лекцій / С. П. Ващишак. − Івано-Франківськ : ІФНТУНГ, 2013. 147 с.

12 Тарасов И.Е. Разработка цифровых устройств на основе ПЛИС

Xilinx с применением языка VHDL / И.Е. Тарасов. Горячая Линия Телеком, 2005. 256 с.

13 J. Armstrong. Chip Level Modelling in VHDL. Prentice Hall, 1988. P. 148.

14 Brown S. Fundamentals of Digital Logic with VHDL (2nd edition) / S. Brown Z. Vranesic. McGraw-Hill, 2004. 939 p.

15 Скляр В. В. Тестирование и разработка диверсных программируемых логических контроллеров на базе ПЛИС с использованием среды функционального программирования / В. В. Скляр, В. С. Харченко, А. С. Панарин // Радіоелектронні і комп'ютерні системи. 2014. № 1. С. 29–41. Режим доступу: http://nbuv.gov.ua/UJRN/recs_2014_1_6.

16 Скляр В. В. Тестирование программируемых логических контроллеров на базе ПЛИС с использованием среды функционального программирования / В.В. Скляр, В.С. Харченко, А.С. Панарин // Системи обробки інформації. 2014. № 1(117). С. 44-55.

ДОДАТОК А.

Проект системи генератор-суматор на мові VHDL

```
library vector;
use std.textio.all,vector.functions.all;
entity MFChip is
   port(s3:in bit;s2:in bit;s1:in bit;s0:in bit;
       A:in bit_vector(1 to 8);B:in bit_vector(1 to 8);
       Fun:out bit_vector(1 to 8));
end MFChip;
architecture Chip of MFChip is
 begin
  Proc: process(s3,s2,s1,s0,A,B)
    begin
      if (s3='0') and (s2='0') and (s1='0') and (s0='0') then Fun<=not(A) after     57 ns;end if;
if (s3='0') and (s2='0') and (s1='0') and (s0='1') then Fun<=not(A or B) after 57 ns;end if;
if (s3='0') and (s2='0') and (s1='1') and (s0='0') then Fun<=not(A) and B after 57 ns;end if;
if (s3='0') and (s2='0') and (s1='1') and (s0='1') then Fun<="00000000" after 57 ns;end if;
if (s3='0') and (s2='1') and (s1='0') and (s0='0') then Fun<=not(A and B) after 57 ns;end if;
if (s3='0') and (s2='1') and (s1='0') and (s0='1') then Fun<=not(B) after 57 ns;end if;
if (s3='0') and (s2='1') and (s1='1') and (s0='0') then Fun<=A xor B after 57 ns;end if;
if (s3='0') and (s2='1') and (s1='1') and (s0='1') then Fun<=A and not(B) after 57 ns;end if;
if (s3='1') and (s2='0') and (s1='0') and (s0='0') then Fun<=not(A) or B after 57 ns;end if;
if (s3='1') and (s2='0') and (s1='0') and (s0='1') then Fun<=not(A or B) after 57 ns;end if;
if (s3='1') and (s2='0') and (s1='1') and (s0='0') then Fun<=B after 57 ns;end if;
if (s3='1') and (s2='0') and (s1='1') and (s0='1') then Fun<=A and B after 57 ns;end if;
if (s3='1') and (s2='1') and (s1='0') and (s0='0') then Fun<="00000001" after 57 ns;end if;
if (s3='1') and (s2='1') and (s1='0') and (s0='1') then Fun<=A or not(B) after 57 ns;end if;
if (s3='1') and (s2='1') and (s1='1') and (s0='0') then Fun<=A or B after 57 ns;end if;
if (s3='1') and (s2='1') and (s1='1') and (s0='1') then Fun<=A after 57 ns;end if;
        end process Proc;
end Chip;
```

```
library vector;
use std.textio.all,vector.functions.all;
entity Generator is
      port (O1,O2:out Bit_Vector(1 to 8);os0,os1,os2,os3:out bit);
end Generator;
architecture Gen of Generator is
   signal O11: bit_vector(1 to 8):="10101010";
   signal O21: bit_vector(1 to 8):="01010101";
   signal os01:bit:='1';signal os11:bit:='0';signal os21:bit:='1';signal os31:bit:='0';
begin
     process begin
          O11<=O11;O21<=O21;
```

```
        os01<=os01;os11<=os11;os21<=os21;os31<=os31;
        wait for 100 ns;
    end process;
    O1<=O11;O2<=O21;
    os0<=os01;os1<=os11;os2<=os21;os3<=os31;
end Gen;
entity shema is end shema;
architecture Sh of shema is
  component MFChip
    Port(s3:in bit;s2:in bit;s1:in bit;s0:in Bit;
    A,B,Fun:in bit_vector(1 to 8));
  end component;
  component Generator
    port (O1,O2:out bit_vector(1 to 8);os0,os1,os2,os3:out bit);
  end component;
  signal A,B,O1,O2,Fun:bit_vector(1 to 8);
  signal s1,s2,s3,s0,os0,os1,os2,os3:bit;
  begin
  process (O1,O2,os0,os1,os2,os3) begin
     A<=transport O1 after 5 ns;B<=transport O2 after 5 ns;
     s0<=transport os0 after 5 ns;s1<=transport os1 after 5 ns;
     s2<=transport os2 after 5 ns;s3<=transport os3 after 5 ns;
  end process;
  Generator port map(O1,O2,os0,os1,os2,os3);
  MFChip port map(s3,s2,s1,s0,A,B,Fun);
end Sh;
```

ДОДАТОК Б.

Проект на мові VHDL відповідно до графа моделі і перевірки часових параметрів

```
package typedef is
   subtype byte is bit_vector (7 downto 0);
end;
use work.typedef.all;

entity Bis_data is

    generic(Trst,Tsel,Tclk,Tsl : time);
    port (clk,rst,s_l : boolean;
        s0, s1 : bit;
        d0, d1 ,d2, d3 : byte;
        q : out byte);
  end Bis_data;
  architecture BIS of Bis_data is
     constant Del_sel, Del_load, Del_shift, Del_out : time := 10 ns;
     variable F : integer;
     signal reg,shft : byte;
  begin F:=0;
  Reset: process(rst)
  assert rst'stable(Trst) report "Error Trst"
  if rst then                    -- асинхронне скидання
     reg <= x"00";
     shft <= x"00";
  end if;
  end process Reset;
Load: process (clk)
  begin
   if clk'delay(Tclk)='1' and          --Clk=1    за час    Tclk
```

```
        not clk'delay(Tclk)'stable  then -- і  перед ним не стабільний, то фронт
        assert s0'stable(Tsel) and          --  якщо сигнали вибору стабільні, то
          s1'stable(Tsel)              -- завантаження, інакше ввід повідомлення
        report "Error Tsel"
       if  s0'stable(Tsel) and s1'stable(Tsel) then
          case s0 & s1   is                          --   мультиплексування
          when b"00" => reg <= d0 after (Del_sel+Del_load);
          when b"10" => reg <= d1 after (Del_sel+Del_load);
          when b"01" => reg <= d2 after (Del_sel+Del_load);
          when b"11" => reg <= d3 after (Del_sel+Del_load);
          end  case; F:=1;
       end if;
    end if;
end process  Load;

     Shift:   process (s_l)
        if s_l='1' and s_l'stable(Tsl) then              --  такт зсуву
             shft <= shft rol 1 after Del_shift;
        end  if;
       F:=3;
     end  process Shift;

     Out: process
            if F=3 then  q <= shft after Del_out;

else q <= reg after Del_out;
        end if;
        F:=0;
     end process Out;
end BIS;
```