**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ**

**Методичні вказівки**
**до виконання лабораторних робіт з дисципліни**

**ЦИФРОВІ ПРИСТРОЇ ТА МІКРОПРОЦЕСОРИ**

для студентів з напряму підготовки 6.050901 «Радіотехніка»
(всіх форм навчання)

# ПРОГРАМОВАНІ ЛОГІЧНІ ІНТЕГРАЛЬНІ СХЕМИ

**2015**

Методичні вказівки до виконання лабораторних робіт з дисципліни “Цифрові пристрої та мікропроцесори ” для студентів з напряму підготовки 6.050901 «Радіотехніка» всіх форм навчання. Програмовані логічні інтегральні схеми /Укл: С.В. Морщавка, С.С. Самойлик . – Запоріжжя: ЗНТУ, 2015. – 66с.

Укладачі: С.В. Морщавка, доцент, к.т.н.
С.С. Самойлик, ст. викладач кафедри РТТ

Рецензент: В.С. Кабак, доцент, д.т.н.

Відповідальний
за випуск: С.С. Самойлик, ст. викладач кафедри РТТ.

Затверджено
На засіданні кафедри «РТТ»
Протокол №7 від 20.02.2015р.

Виконано у рамках проекту 544091-TEMPUS-1-2013-1-BE-TEMPUS-JPCR, "Розробка курсів з Вбудованих Систем зі застосуванням новітніх підходів віртуалізації для поєднання досліджень, освіти та виробництва в Україні, Грузії та Арменії " (DesiRE)

**ЗМІСТ**

# ВСТУП

Програмовані логічні інтегральні схеми (ПЛІС) стають останнім часом усе більш розповсюдженою й звичною елементною базою для цифрових пристроїв. Завдяки різкому збільшенню щільності пакування елементів на кристалі багато провідних виробників або почали серійне виробництво, або анонсували ПЛІС із еквівалентною ємністю більше 1 мільйона логічних вентилів.

Основними перевагами ПЛІС при застосуванні в засобах обробки сигналів є:

- висока швидкодія;
- можливість реалізації складних паралельних алгоритмів;
- наявність засобів автоматизованного проектування (САПР), що дозволяють провести повне моделювання системи;
- можливість програмування або зміни конфігурації безпосередньо в системі;
- сумісність алгоритмів на рівні мов опису апаратури (VHDL, AHDL, Verilog та ін.);
- архітектурні особливості ПЛІС як не можна краще пристосовані для реалізації таких операцій, як множення, згортка тощо.

У цей час тактові частоти ПЛІС досягли величин понад 500 МГц, що дозволяє реалізовувати багато алгоритмів майже на радіочастотах. Тому актуальним є ознайомлення студентів з базовими можливостями цифрових систем на ПЛІС і способами їх проектування.

У даних методичних рекомендаціях розглядаються основи проектування цифрових пристроїв на базі ПЛІС сімейства MAX3000A фірми Altera. Хоч це й не сама нова розробка фірми Altera, але її клони продовжують випускатися й зараз. Її логічної місткості цілком достатньо для реалізації як комбінаційних пристроїв так і пристроїв з пам'яттю.

Ще одним фактором при виборі ПЛІС Altera є наявність достатньо розвинених безкоштовних версій САПР, зокрема пакета MAX+PLUS II, який можна безкоштовно завантажити із сайту www.altera.com або одержати на CD Altera Digital Library, на якому міститься також і повний набір документації по архітектурі й застосуванню ПЛІС.

# 1 ЗАГАЛЬНА ХАРАКТЕРИСТИКА ПЛІС MAX3000 І СПОСОБИ РОЗРОБКИ ЦИФРОВИХ ПРИСТРОЇВ НА НИХ ЗА ДОПОМОГОЮ ПРОГРАМНОГО ПАКЕТА ALTERA MAX+PLUS II

## 1.1 Загальні теоретичні відомості

### 1.1.1 Архітектура й особливості сімейства MAX3000

Мікросхеми сімейства MAX3000 виконані по CMOS EPROM технології, при дотриманні технологічних норм 0.35 мкм, що дозволило їх суттєво здешевити. Усі ПЛІС MAX3000 підтримують технологію програмування в системі (ISP, In-system programmability) і периферійного сканування (boundary scan) у відповідності зі стандартом IEEE Std. 1149.1 JTAG. Елементи вводу-виводу (ЕВВ) дозволяють працювати в системах з рівнями сигналів 5В, 3.3В, 2.5В. Матриця з'єднань має безперервну структуру, що дозволяє реалізувати час затримки поширення сигналу менше 4.5 нс. ПЛІС MAX3000 мають можливість апаратної емуляції виходів з відкритим колектором (open-drains pin) і задовольняють вимогам стандарту PCI. Є можливість індивідуального програмування ланцюгів скидання, встановлення й тактування тригерів, що входять у макрокомірку. Передбачений режим зниженого енергоспоживання. Програмуваний логічний розширник дозволяє реалізувати на одній макрокомірці функції до 32 змінних. Є можливість завдання біта таємності (security bit) для захисту від несанкціонованого тиражування розробки.

Реалізація функції програмування в системі підтримується з використанням стандартних засобів завантаження, таких як ByteBlasterMV, BitBlaster, MasterBlaster, а також підтримується формат JAM.

ПЛІС MAX3000 випускаються в корпусах, що мають від 44 до 208 виводів.

На рис.1.1 представлена функціональна схема ПЛІС сімейства MAX3000.

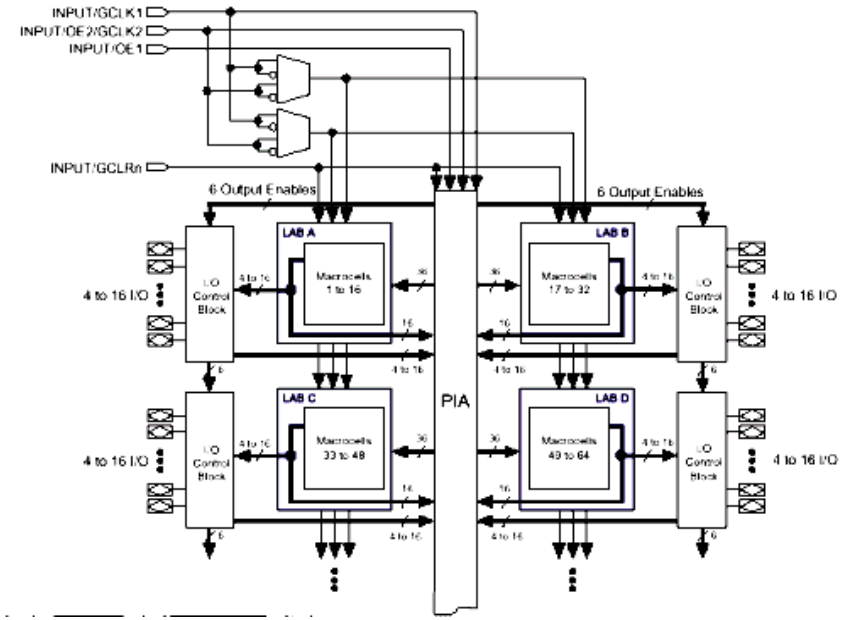



Рисунок 1.1 – Функціональна схема ПЛІС сімейства MAX3000

Основними елементами структури ПЛИС сімейства MAX3000 є:

- логічні блоки (ЛБ) (LAB, Logic Array Blocks);
- макрокомірки (МК) (MacroCells);
- логічні розширники (expanders) (паралельні (parallel) і поділювальні (shareble));
- програмувальна матриця з'єднань (ПМЗ)(programmable interconnect array, PIA);
- елементи вводу-виводу (ЕВВ)(I/O Control Block).

ПЛІС сімейства MAX3000 мають чотири виводи, що закріплені за глобальними ланцюгами (dedicated inputs). Це глобальні ланцюги синхронізації скидання й встановлення в третій стан кожної макрокомірки. Крім того, ці виводи можна використовувати як входи або виходи користувача для "швидких" сигналів, що обробляються у ПЛИС.

Як видно з рис. 1.1 в архітектура ПЛІС сімейства MAX3000 базується на логічних блоках, що складаються з 16 макрокомірок

кожний. Логічні блоки з'єднуються за допомогою ПМЗ. Кожний логічний блок має 36 входів із ПМЗ.

На рис 1.2 наведена структурна схема однієї макрокомірки ПЛІС сімейства MAX3000.

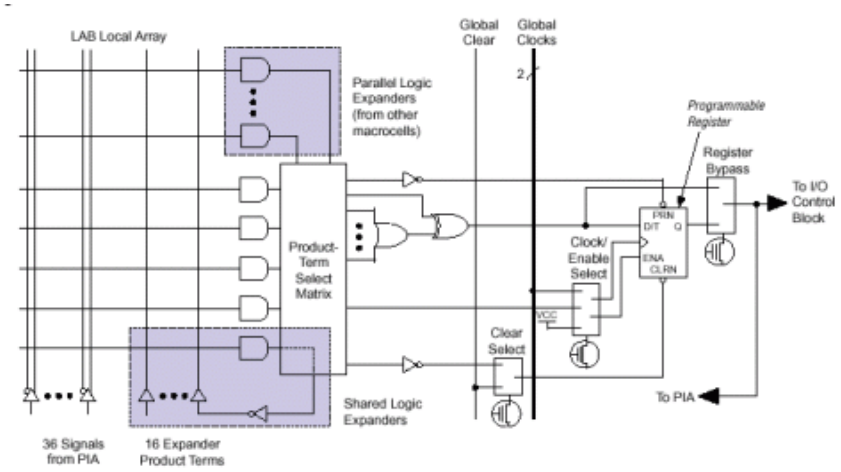



Рисунок 1.2 – Структурно-функціональна схема макрокомірки ПЛІС сімейства MAX3000

МК ПЛІС сімейства MAX3000 складається з трьох основних частин:

- локальної програмованої матриці (LAB Local Array);
- матриці розподілу термів (Product-Term Select Matrix);
- програмувального регістру (Programmable Register).

Комбінаційні функції реалізуються за допомогою локальної програмованої матриці й матриці розподілу термів, що дозволяє поєднувати логічні добутки або по АБО (OR) або по виключному АБО (XOR). Крім того, матриця розподілу термів дозволяє комутувати ланцюги керування тригером МК.

Режим тактування й конфігурація тригера вибираються автоматично під час синтезу проекту в САПР MAX+PLUS II залежно від обраного розроблювачем типу тригера в описі проекту.

У ПЛІС сімейства MAX3000 доступно два глобальних тактових сигнали, що дозволяє проектувати схеми з двофазною синхронізацією.

Для реалізації логічних функцій великого числа змінних використовуються логічні розширювачі. Поділюваний логічний розширювач (рис. 1.3) дозволяє реалізувати логічну функцію з більшим числом змінних (входи макрокомірки), дозволяючи об'єднати МК, що входять до складу одного ЛБ. Таким чином, поділюваний розширювач формує терм, інверсне значення якого передається матрицею розподілу термів у локальну програмувальну матрицю й може бути використане будь-якою МК даного ЛБ. Як видно з рис. 1.3, є 36 сигналів локальної ПМЗ, а також 16 інверсних сигналів з поділюваних логічних розширювачів, що дозволяє в межах одного ЛБ реалізувати функцію до 52 термів рангу 1.

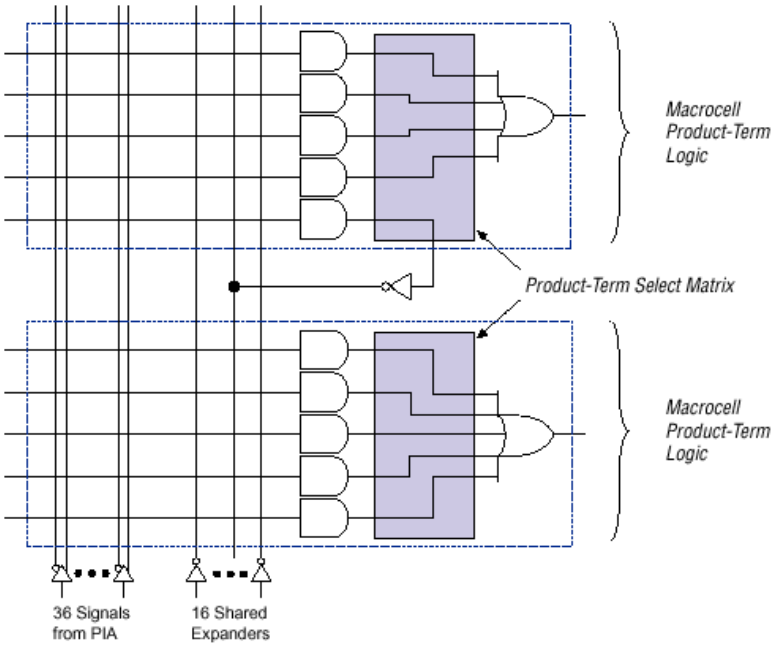



Рисунок 1.3 – Паралельний логічний розширювач

На рис 1.4 наведена схема одного елементу вводу-виводу (ЕВВ) ПЛИС сімейства MAX3000. ЕВВ дозволяє організувати режими роботи з відкритим колектором і третім станом.

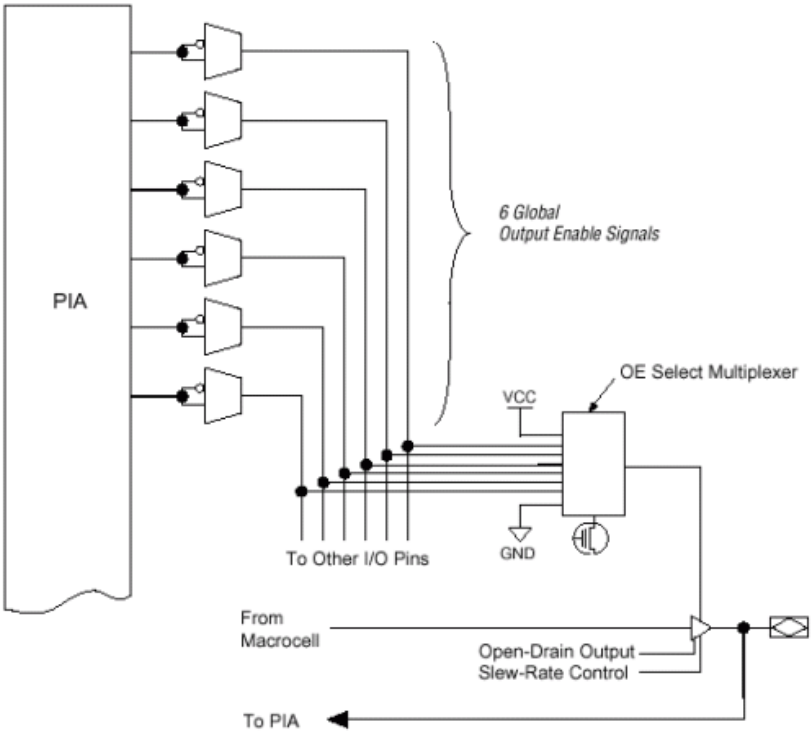



Рисунок 1.4 Елемент вводу-виводу

### 1.1.2 Процедура розробки й основні підходи до роботи в системі MAX+plus II

Назва системи MAX+plus II є абревіатурою від Multiple Array Matrix programmable Logic User System. Система MAX+plus II має засоби зручного введення проекту, компіляції й налагодження, а також безпосереднього програмування пристроїв.

Процедуру розробки нового проекту від концепції до завершення можна спрощено представити в такий спосіб:

1) створення нового файлу проекту або ієрархічної структури декількох файлів проекту за допомогою будь-якої комбінації редакторів у системі MAX+plus II, тобто графічного, текстового й сигнального редакторів;

2) завдання імені файлу – проекту верхнього рівня як імені проекту;

3) призначення сімейства ПЛІС для проекту;

4) відкриття вікна компілятора Compiler і вибір кнопки Start для початку компіляції проекту. За бажанням користувача можна підключити модуль виділення проміжних параметрів проекту Timing SNF Extractor для створення файлу, використовуваного при моделюванні;

5) у випадку успішної компіляції у подальшому можна виконати часовий аналіз, для чого слід виконати наступне:

- для проведення часового аналізу затримок відкрити вікно Timing Analyzer, обрати режим аналізу й нажати кнопку Start;

- для проведення симуляції потрібно спочатку створити тестовий вектор у файлі каналу тестування (.scf), користуючись сигнальним редактором, або у файлі вектора (.vec), користуючись текстовим редактором. Потім відкрити вікно відладчика Simulator і нажати кнопку Start;

6)відкриття вікна програматора «programmer» з наступним вибором одного з двох шляхів програмування: використання програматора MPU (Master Programming Unit) або підключення завантажувальних пристроїв Bitblaster, Byteblaster або FLEX Download Cable до пристрою, що програмується в системі, без відключення від схеми;

7) вибір кнопки program для програмування пристроїв з пам'яттю типу Eprom або Eeprom або вибір кнопки Configure для конфігурації пристрою з пам'яттю типу SRAM.

На рис.1.5 представлений вигляд вікна пакету MAX+plus II із завантаженою схемою.

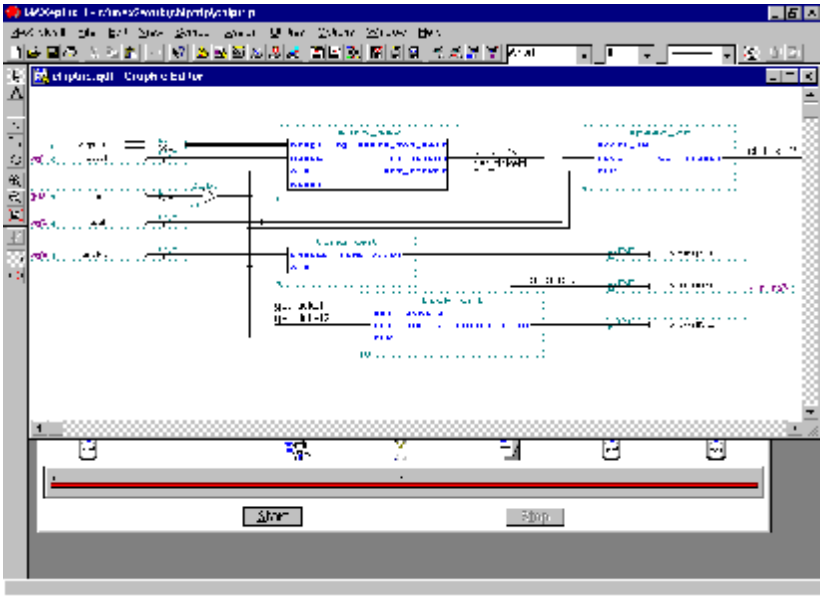


Рисунок 1.5 – Головне вікно системи MAX+plus II

ПО системи MAX+plus II містить з 11 окремих програм і головної керуючу програму. Різні програми, що забезпечують створення проекту, можуть бути активізовані миттєво, що дозволяє користувачеві перемикатися між ними клацанням миші або за допомогою команд меню. Одночасно можуть працювати кілька окремих частин, наприклад компілятор, симулятор, аналізатор синхронізації й програматор. Однакові команди різних програм працюють однотипно, що полегшує завдання розробки логічного дизайну.

У таблиці 1.1 наведений опис програм.

Таблиця 1.1

| **Програма** | **Функція, що виконується** |
|---|---|
| **Hierarchy Display** | **Огляд ієрархії** – відображає поточну ієрархічну структуру файлів у вигляді дерева з гілками, що є підпроектами |
| **Graphic Editor** | **Графічний редактор** – дозволяє розроблювати схемотехнічний логічний проект у форматі реального відображення на екрані у режимі WYSIWYG |
| **Symbol Editor** | **Символьний редактор** – дозволяє редагувати існуючі символи та створювати нові |
| **Text Editor** | **Текстовий редактор** – дозволяє створювати та редагувати текстові файли логічного дизайну, що написані мовами AHDL, VHDL и Verilog HDL |
| **Waveform Editor** | **Сигнальний** редактор – виконує подвійну функцію: інструмент для розробки дизайну та інструмент для введення текстових векторів и перегляду результатів тестування |
| **Floorplan Editor** | **Порівневий планувальник** – дозволяє графічними засобами присвоювати призначення контактам пристрою та здійснювати розподіл ресурсів логічних елементів |
| **Compiler** | **Компілятор** – оброблює логічні проекти |
| **Simulator** | **Симулятор** – дозволяє тестувати логічні операції та внутрішню синхронізацію проектованого логічного ланцюга |
| **Timing Analyizer** | **Часовий аналізатор** – аналізує роботу проектованого логічного ланцюга після синтезу та оптимізації компілятором |
| **Programmer** | **Програматор** – дозволяє програмувати, конфігурувати, проводити верифікацію та тестувати ПЛИС фірми Altera |
| **Message processor** | **Генератор повідомлень** – виводить на екран повідомлення про помилки, попереджувальні та інформативні повідомлення |

Перед тим як почати працювати в системі MAX+plus II, слід зрозуміти різницю між файлами проекту, допоміжними файлами й проектами.

*Файл проекту* - це графічний, текстовий або сигнальний файл, створений за допомогою графічного або сигнального редакторів системи MAX+plus II або в будь-якому іншому, що використовує промислові стандарти схемному або текстовому редакторі або за допомогою програми netlist writer, наявною в пакетах, що підтримують EDIF, VHDL і Verilog HDL. Цей файл містить логіку для проекту MAX+plus II і оброблюється компілятором. Компілятор може автоматично обробляти наступні файли проекту: графічні файли проекту (.gdf); текстові файли проекту мовою AHDL (.tdf); сигнальні файли проекту (.wdf); файли проекту мовою VHDL (.vhd); файли проекту мовою Verilog (.v); схемні файли Orcad (.sch); вхідні файли EDIF (edf); файли формату Xilinx Netlist (.xnf); файли проекту Altera (.adf); файли цифрового автомата (.smf).

*Допоміжні файли* – це файли, пов'язані із проектом MAX+plus II, але, що не є частиною ієрархічного дерева проекту. Більшість таких файлів не містить логіки проекту. Деякі з них створюються автоматично засобами системи MAX+plus II, інші – користувачем. Прикладами допоміжних файлів є файли призначень і конфігурації (.acf), символьні файли (.sym), файли звіту (.rpt) і файли тестових векторів (.vec).

Проект складається із усіх файлів ієрархічної структури проекту, у тому числі допоміжних і вихідних файлів. Ім'ям проекту є ім'я файлу проекту верхнього рівня без розширення. Система MAX+plus II виконує компіляцію, тестування, аналіз синхронізації й програмування відразу цілого проекту, хоча користувач може в цей час редагувати файли цього проекту в рамках іншого проекту. Для кожного проекту бажане створювати окремий підкаталог у робочому каталозі системи MAX+plus II.

Система MAX+plus II підтримує легкий доступ до всіх інструментів для створення проекту. Розробка проекту прискорюється за рахунок наявних стандартних функцій, у тому числі примітивів, мегафункцій, бібліотеки параметризованих модулів (Lpm) і макрофункцій застарілого типу мікросхем 74 серії. У системі MAX+plus II є три редактори для розробки проекту: графічний,

текстовий і сигнальний, а також два допоміжні редактори: порівневий планувальник і символьний редактор.

Усі п'ять редакторів мають спільні функції, такі як, наприклад, створення, збереження й відкриття файлу. Крім того, програмні додатки редактора MAX plus II мають наступні загальні функції: створення файлів символів і файлів із прототипами функцій ( Include-Файли); пошук вузлів; траверс ієрархічного дерева; спливаючі вікна меню, що залежить від контексту; сигнально - часовий аналіз; пошук і заміна фрагментів тексту; скасування останнього кроку редагування, його повернення, вирізка, копіювання, вставку й видалення обраних фрагментів, обмін фрагментами між програмними додатками MAX plus II або додатками Windows; друк.

**Графічний редактор** (Graphic Editor) забезпечує проектування в реальному форматі зображення (WYSIWIG). Графічні файли проекту (.gdf) або схемні файли Orcad (.sch), створені в даному графічному редакторі, можуть включати будь-яку комбінацію символів примітивів, мегафункцій і макрофункцій. Символи можуть являти собою будь-який тип файл проекту (.gdf .sch .tdf .vhd .v .wdf .edf .xnf .adf .smf).

Інструмент вибору ("стрілка") полегшує розробку дизайну. Він дозволяє зміщувати й копіювати об'єкти, а також додавати нові символи. Коли навести його на контакт або закінчення лінії, він автоматично перетвориться в інструмент малювання ортогональних ліній. Якщо їм клацнути на тексті, він автоматично перетвориться в інструмент редагування тексту.

Графічний редактор забезпечує також ряд інших можливостей. Наприклад, можна збільшити або зменшити масштаб відображення на екрані, обирати гарнітуру й розмір шрифту, задавати стилі ліній, встановлювати й відображати напрямні, отримувати дзеркальне відображення, повертати виділені фрагменти на 90, 180 або 270 градусів; задавати розмір, орієнтацію поточного аркушу схеми.

**Символьний редактор** (Symbol Editor) дозволяє переглядати, створювати й редагувати символ. Символьний файл має те ж ім'я, що й проект але з розширенням .sym. Команда Creat Default Symbol меню File, яка є в графічному, текстовому й сигнальному редакторах, створює символ для будь-якого файлу проекту. Символьний редактор має наступні характеристики: дозволяє перевизначити символ, який представляє файл проекту, створювати й редагувати виводи з їх

іменами, використовуючи вхідні, вихідні й двоспрямовані виводи, а також задавати варіанти вводу символу у файл графічного редактора, задавати значення параметрів та їх значення за замовчуванням, задавати сітка й напрямні, що допомагають виконати точне вирівнювання об'єктів, вводити коментарі до символів.

**Текстовий редактор** (Text Editor) є інструментом для створення текстових файл проекту на мовах опису апаратури: AHDL (.tdf), VHDL (.vhd), Verilog HDL (.v). У текстовому редакторі можна працювати також з довільним файлом формату ASCII. Усі вищеозначені файли проекту можна створювати у будь-якому текстовому редакторі, однак даний редактор має вбудовані можливості введення файлів проекту, їх компіляції й налагодження з видачею повідомлень про помилки з їх локалізацією у вихідному тексті або в тексті допоміжних файлів; крім того, існують шаблони язикових конструкцій для AHDL, VHDL та Verilog HDL, виконується акцентування кольором синтаксичних конструкцій. У цьому редакторі можна вручну редагувати файли призначень і конфігурації (.acf), а також коректувати параметри конфігурації для компілятора, симулятора й часового аналізатору.

Користуючись даним текстовим редактором, можна створювати тестові вектори (.vec), що використовуються для тестування, настроювання функцій та введення сигнального проекту. Можна також створювати командні файли (.cmd – для симулятора й .edc – для EDIF), а також макробібліотеки (.lmf).

**Сигнальний редактор** (Waveform Editor) слугує інструментом створення опису проекту, введення тестових векторів та перегляду результатів тестування. Користувач може створювати сигнальні файли проекту (.wdf), які містять часові діаграми, що описують логіку роботи проекту, а також файли каналів тестування (.scf), які містять вхідні вектора для тестування й функціонального налагодження. Розробка опису проекту в сигнальному редакторі є альтернативою його створенню в графічному або текстовому редакторах. Тут можна графічним способом задавати комбінації вхідних логічних рівнів і необхідних вихідних сигналів. Створений у такий спосіб файл WDF може містити як логічні входи, так і входи цифрового автомата, а також виходи комбінаторної логіки, лічильників й цифрових автоматів. Спосіб розробки дизайну за допомогою сигнального редактору краще підходить для ланцюгів з чітко визначеними

послідовними входами й виходами, тобто для цифрових автоматів, лічильників і регістрів.

За допомогою сигнального редактора можна легко перетворювати часові діаграми сигналів повністю або частково створюючи й редагуючи вузли й групи. Простими командами можна створювати файл таблиці Ascii-Символів (.tbl) або імпортувати файл тестових векторів у форматі ASCII (.vec) для створення файлів каналів SCF, що тестуються і сигнального дизайну WDF. Можна також зберегти файл WDF як SCF для проведення тестування або перетворити SCF в WDF для використання його в якості файлу проекту.

Сигнальний редактор має наступні відмінні риси: можна створити або відредагувати вузол задавши його тип, при розробці WDF можна задати тип логіки вузла, задати значення за замовчуванням у логічному вузлі, а також ім'я стану за замовчуванням у вузлі цифрового автомата, для спрощення створення тестового вектора можна легко додати у файл каналів SCF, що тестуються, кілька вузлів або все з інформаційного файлу симулятора (.snf), що існує для повністю відкомпільованого проекту, можна об'єднувати від 2 до 256 вузлів для створення нової групи (шини) або розгрупувати об'єднані раніше в групу вузли. Можна також об'єднувати групи з іншими групами. Значення групи може бути відображене у двійковій, десятковій, шістнадцятковій або восьмеричній системі числення з перетворенням або без у код Грэя, можна копіювати, вставляти, переміщувати або видаляти обрану частину ("інтервал") сигналу, а також весь вузол або групу. Можна також інвертувати, вставляти, переписувати, повторювати, розширювати або стискати інтервал сигналу будь-якої довжини з будь-яким логічним рівнем, тактовим сигналом, послідовністю рахунку або іменем стану, задавати сітку для вирівнювання переходів між логічними рівнями, у будь-якому місці файлу можна вводити коментарі до сигналів, змінювати масштаб відображення.

Для полегшення тестування можна зробити накладення будь-яких виходів у поточному файлі або накласти другий файл сигнального редактора для порівняння сигналів його вузлів і груп з відповідними сигналами поточного файлу.

**Порівневий планувальник** (Floorplan Editor) призначений для призначення ресурсів фізичних пристроїв і перегляду результатів

розведення, зроблених компілятором. У вікні порівневого планувальника можуть бути представлено два типи зображення:

Device View (Вид пристрою) показує всі контакти пристрою і їх функції.

LAB View (Вид логічного структурного блоку) показує внутрішню частину пристрою, у тому числі всі логічні структурні блоки (LAB) і окремі логічні елементи

Після виконання всіх призначень й створення проекту приступають до його компіляції. Спочатку компілятор дістає інформацію про ієрархічні зв'язки між файлами проекту й перевіряє проект на прості помилки введення опису проекту.

**Компілятор** застосовує різноманітні способи збільшення ефективності проекту й мінімізації використання ресурсів мікросхеми. Якщо проект занадто великий, щоб бути реалізованим в одному пристрої, компілятор може автоматично розбити його на частині для реалізації в декількох пристроях того ж самого сімейства пристроїв, при цьому мінімізується число сполук між пристроями. У файлі звіту (.rpt) потім буде відображено яким чином проект буде реалізовано в одному або декількох пристроях.

Крім того, компілятор створює файли, які програматор буде використовувати для програмування одного або декількох пристроїв. У розроблювача також є можливість настроїти обробку проекту. Наприклад, можливо задати стиль логічного синтезу проекту за замовчуванням і інші параметри логічного синтезу в рамках усього проекту, що дозволить провести логічний синтез відповідно до ваших потреб. Крім того, ви можете ввести вимоги по синхронізації в рамках усього проекту, точно задати розбивку великого проекту на частині для реалізації в декількох пристроях і вибрати варіанти параметрів пристроїв, які будуть застосовані для всього проекту в цілому.

Завантаження готового проекту в ПЛИС або конфігураційне ПЗП виконують за допомогою **програматора.**

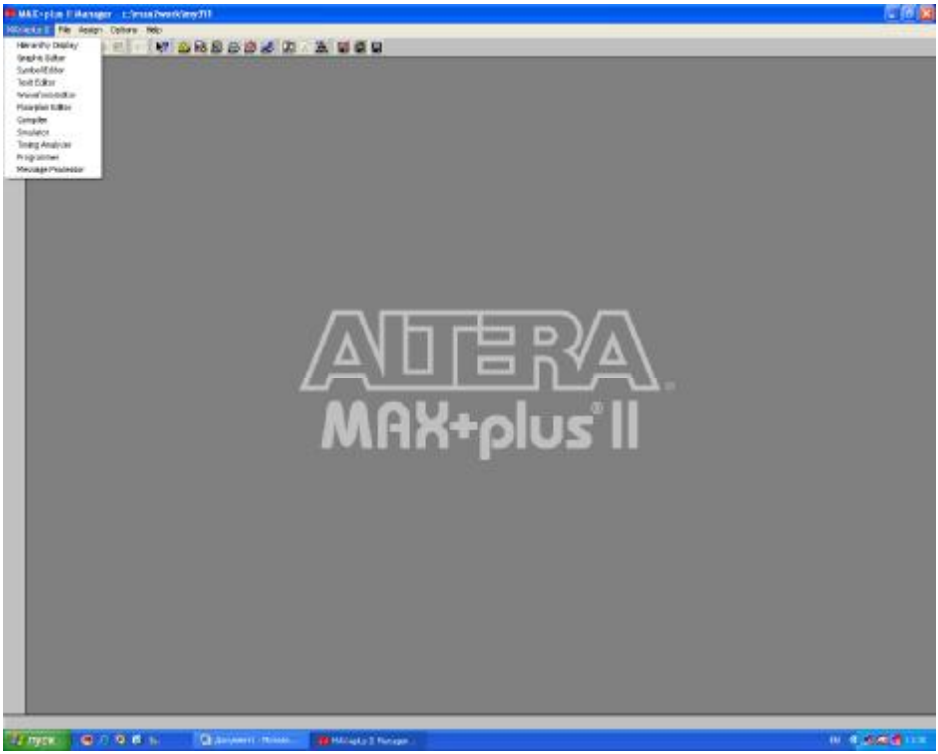
## 1.2 Лабораторна работа №1

### Створення цифрових пристроїв на базі сімейства ПЛИС MAX3000 за допомогою графічного редактора програмного пакета Altera Max+plus II

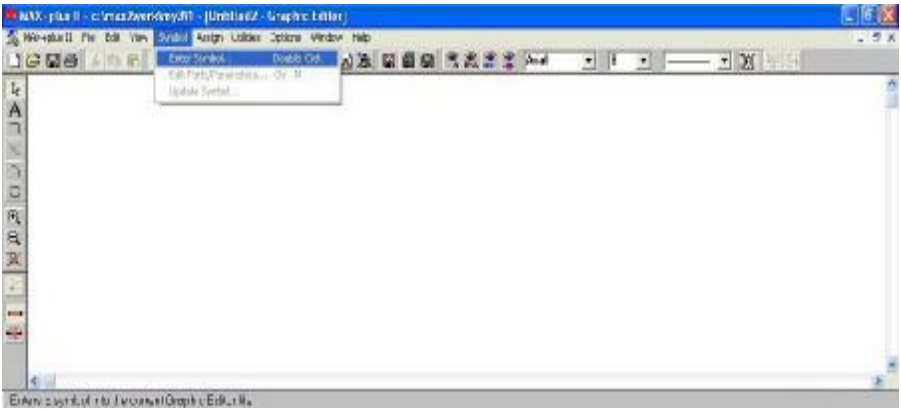
**Ціль роботи –** Вивчити основні параметри й характеристики сімейства ПЛИС MAX3000 фірми Altera та графічний редактор програмного пакету Altera Max+plusII.

### 1.2.1 Порядок виконання роботи

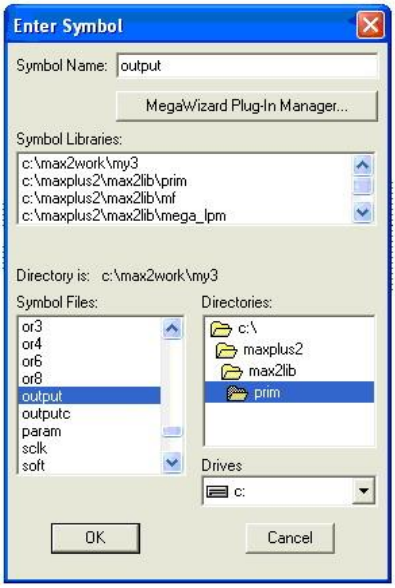
1.2.1.1 Запустити оболонку пакета та з меню Max+plus II запустити графічний редактор Graphic Editor



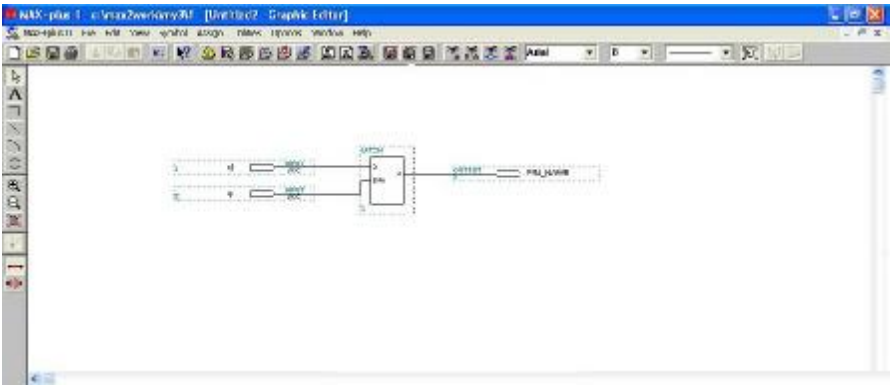
1.2.1.2 У вікні, що відкрилося, починаємо будувати схему з базових символів, що доступні через пункт меню Symbol->Enter Symbol



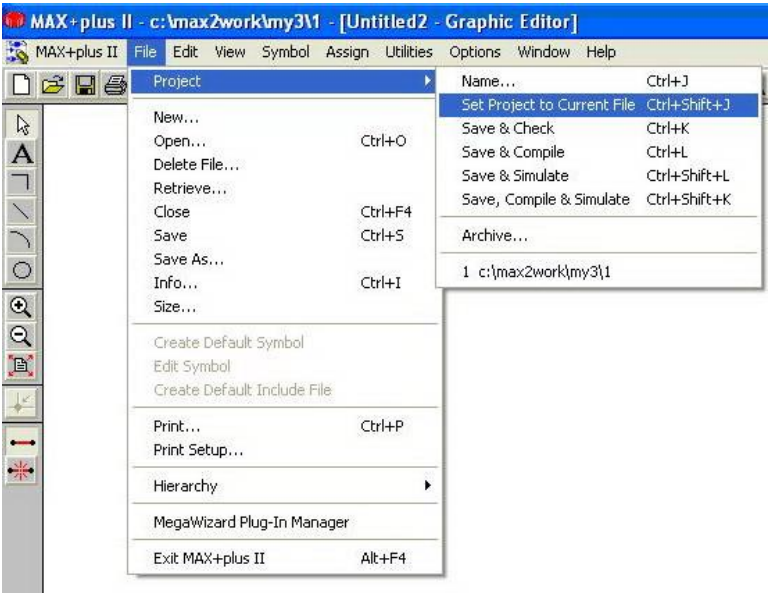
1.2.1.3 При вставці символу необхідно вибрати бібліотеку символів і сам елемент з вікна, що відкрилося.



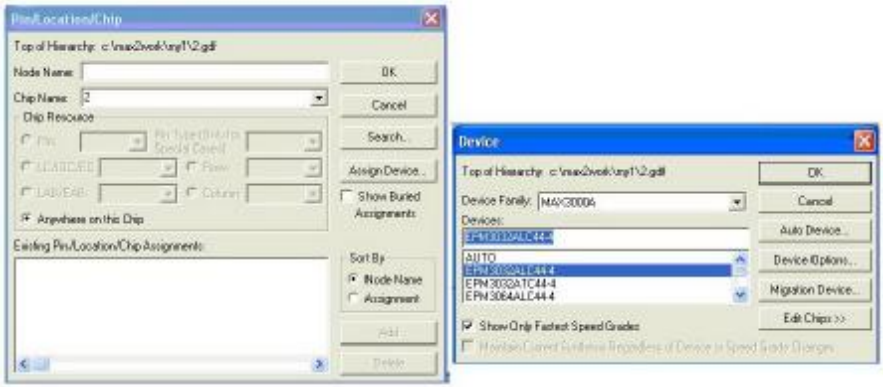
1.2.1.4 Використовуючи провідники (доступні після натискання передостанньої кнопки на панелі ліворуч) з'єднуємо елементи в схему.



1.2.1.5 Після створення схеми вказуємо, що даний файл і складає основу проекту. Для цього вибираємо меню File->Project і зберігаємо проект, задавши при цьому шлях до нього та ім'я файлу в якому буде зберігається створений проект.

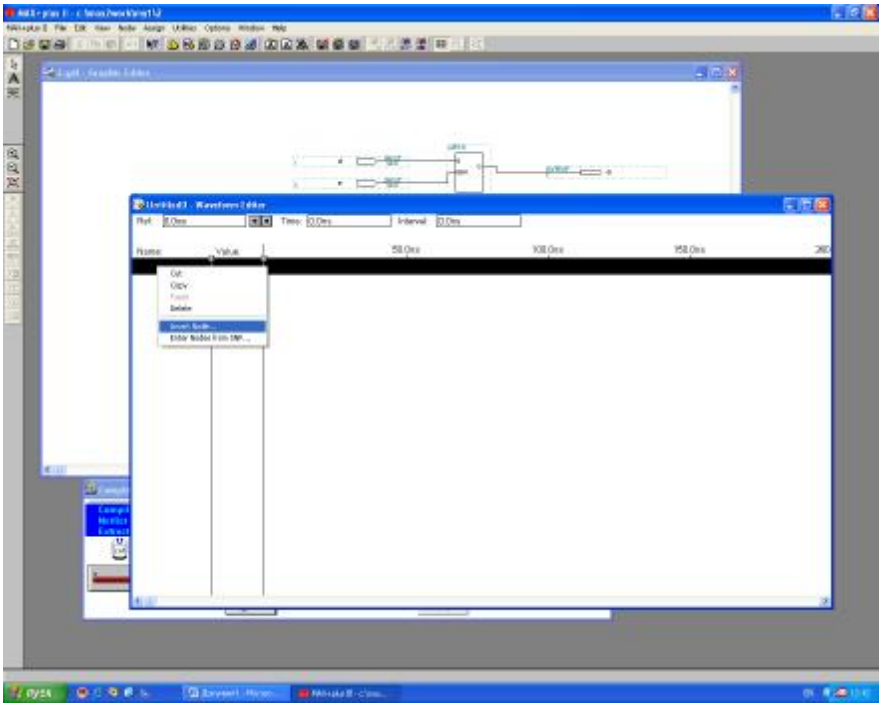


1.2.1.6 Обираємо пристрої на якому буде реалізовуватися проект. Для цього через меню властивостей кожного об'єкта, що доступне за кліком правої кнопки миші, при виділеному об'єкті входимо в меню Pin/Location/Chip і задаємо сімейство ПЛИС у цьому випадку це MAX3000A і вибираємо назву мікросхеми даного сімейства.

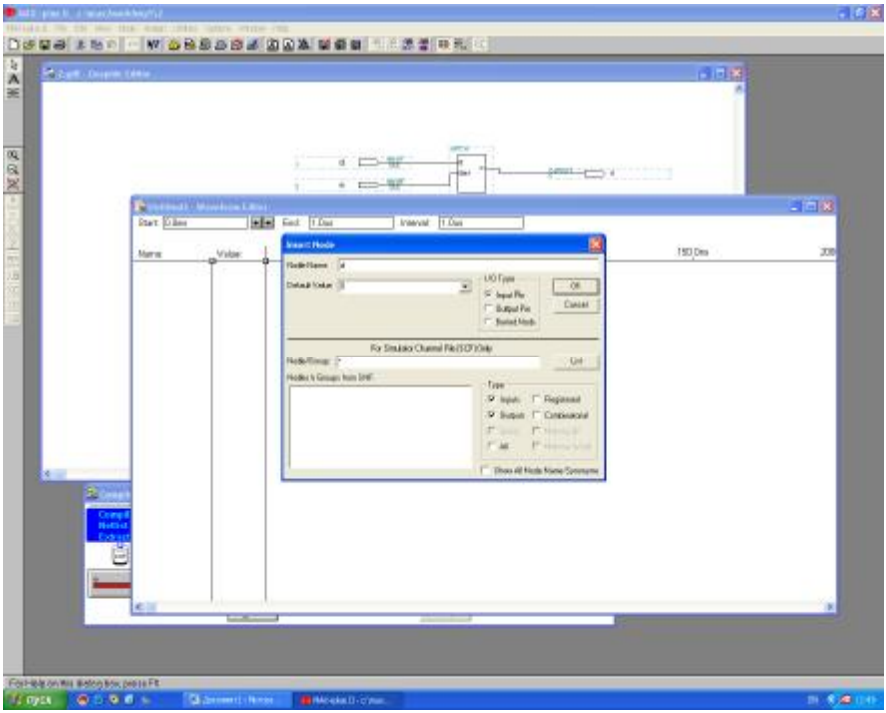



1.2.1.7 Після того, як схема буде зібрана необхідно виконати процес компіляції для цього з панелі кнопок угорі вибираємо «Compiler». Ознакою успішної компіляції є повідомлення про відсутність помилок. При наявності помилок їх необхідно усунути й повторити компіляцію проекту.

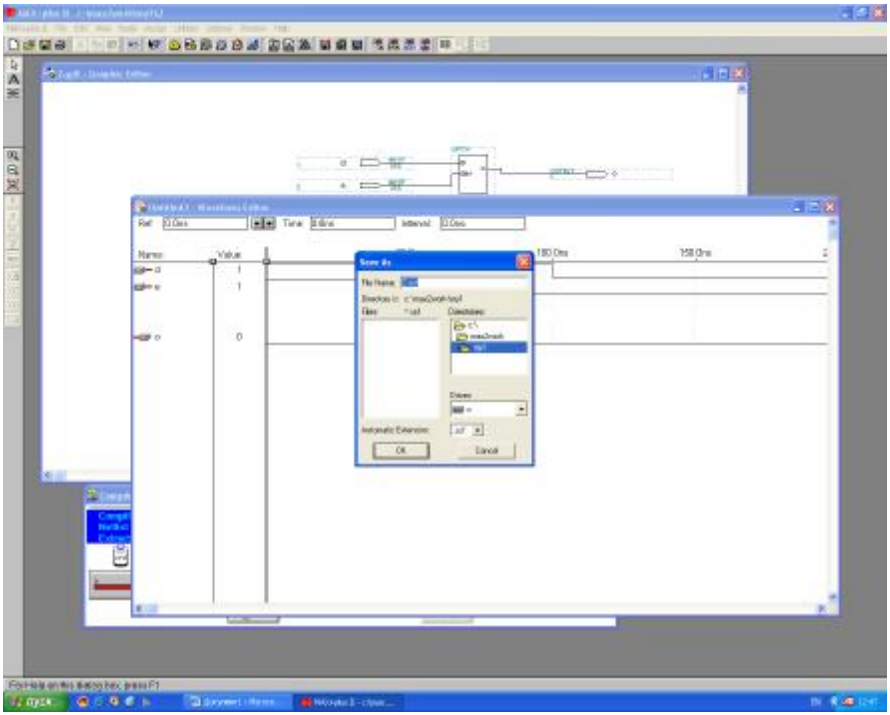
1.2.1.8 Для створення тестових векторів впливів запускаємо редактор Waveform Editor з меню File та графічно задаємо впливи на всі входи схеми за чергою.



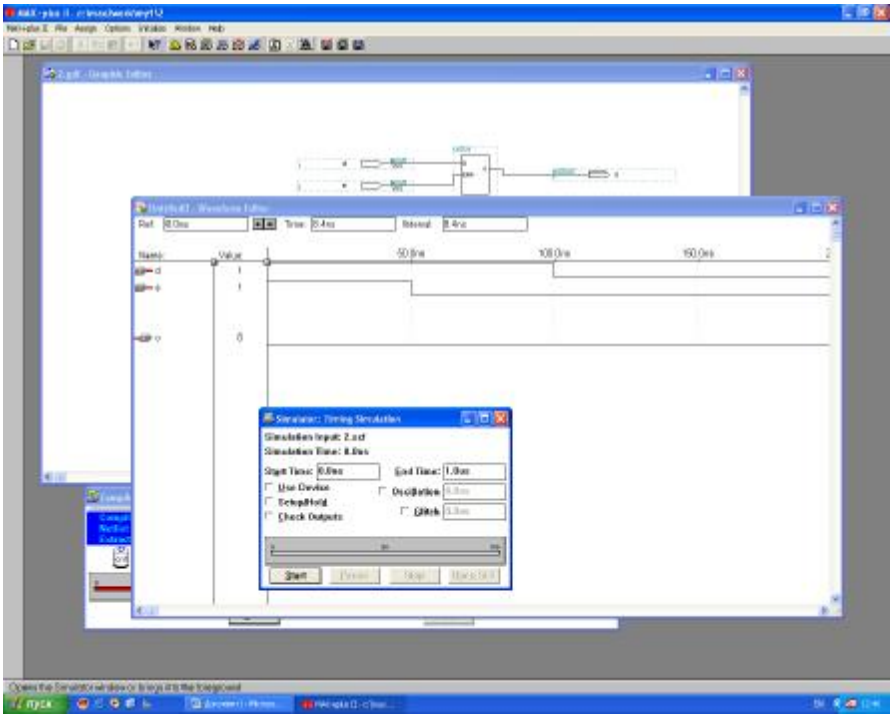
1.2.1.9 Для цього створюємо вхід і вказуємо його ім'я, тип і стан. Цю процедуру необхідно виконати для кожного входу схеми.



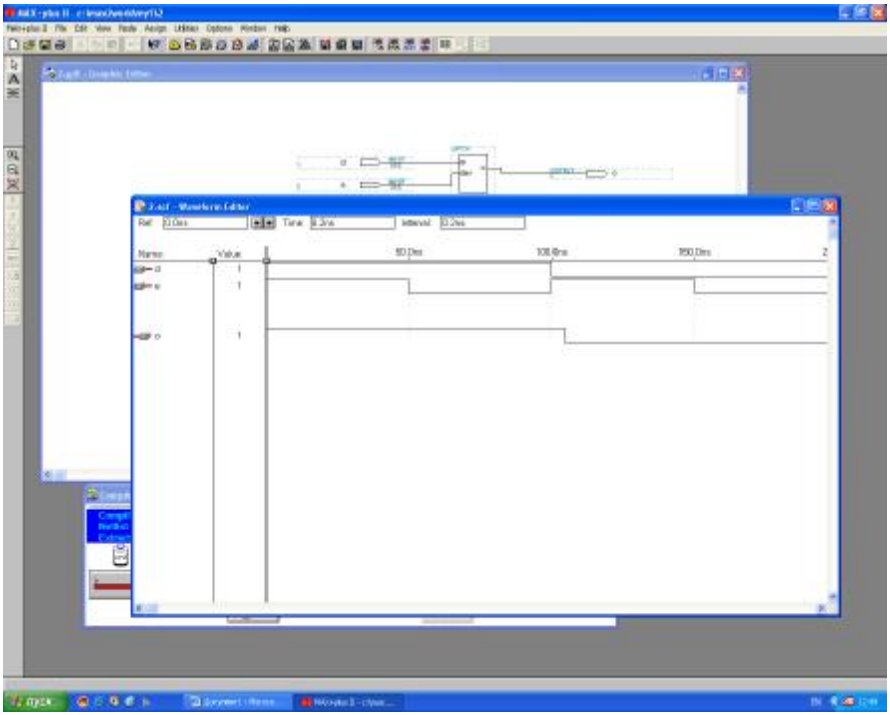
1.2.1.10 Зберігаємо вхідні вектори сигналів



1.2.1.11 Кнопкою меню запускаємо тестування проекту



1.2.1.12 У результаті тестування одержуємо вихідні сигнали, якщо результати тестування нас не влаштовують, змінюємо схему й повторюємо попередні кроки.



### 1.2.2 Лабораторне завдання

Намалюйте УГП тригера, що має входи керування, згідно з варіантом по табл. 1.1. Побудуйте таблицю станів, реалізуйте за допомогою будь-якого типу базових або універсальних елементів (можна використати будь-яку кількість елементів з будь-якою кількістю входів). За допомогою **графічного редактора програмного пакета Altera Max+plus II** переконайтеся, що отримана схема виконує свої функції.

Таблиця 1.2 – Варіанти завдань

| Параметри | Номери варіантів | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| Вхід R | п | п | п | і | і | і | - | - | - | п | п | п | і | і | і | - | - | - | п | п | і | і | п | п | і | і | п | п | і | і |
| Вхід S | п | і | - | п | і | - | п | і | - | п | і | - | п | і | - | п | і | - | п | і | п | і | п | і | п | і | п | і | п | і |
| Вхід C | п | п | п | п | п | п | п | п | п | і | і | і | і | і | і | і | і | і | п | п | п | п | і | і | і | і | - | - | - | - |
| Вхід D | п | п | п | п | п | п | п | п | п | п | п | п | п | п | п | п | п | п | - | - | - | - | - | - | - | - | - | - | - | - |

Примітка: тут R – вхід скидання; S – вхід встановлення; C – вхід синхронізації; D – інформаційний вхід; “і” – інверсний, “п” – прямий, “-” – відсутній.

### 1.2.3 Зміст звіту

Звіт по виконаній роботі має містити у собі: титульну сторінку з назвою роботи; мету роботи; УГП тригеру та схема, що його реалізує; сигнальні діаграми для перевірки роботи синтезованої цифрової схеми; допоміжна інформація; висновки.

### 1.2.4 Контрольні питання

1. Перелічте можливості графічного редактора системи MAX+PLUS II.
2. Які види типових модулів (символів) цифрових пристроїв передбачені в бібліотеці системи MAX+PLUS II для скорочення часу розробки проекту? Де вони розташовуються й чим відрізняються?
3. Призначення піктограм інструментів графічного редактора системи MAX+PLUS II.
4. Набір команд, визначений для графічного редактора в розділі меню Edit.
5. Набір команд, визначений для графічного редактора в розділі меню View.
6. Набір команд, визначений для графічного редактора в розділі меню Symbol .
7. Набір команд, визначений для графічного редактора в розділі меню Utilities .

8. Набір команд, визначений для графічного редактора в розділі меню Window .
9. Набір команд, визначений для графічного редактора в розділі меню File.
10. Набір команд, визначений для графічного редактора в розділі меню Option .
11. Введення елементів схеми в графічному редакторі системи MAX+PLUS II.
12. Переміщення елементів схеми в графічному редакторі системи MAX+PLUS II.
13. Копіювання схеми в графічному редакторі системи MAX+PLUS II.
14. Видалення елементів схеми в графічному редакторі системи MAX+PLUS II.
15. Зміна положення елементів схеми в графічному редакторі системи MAX+PLUS II.
16. Введення й редагування примітивів INPUT і OUTPUT.
17. Іменування елементів схеми в графічному редакторі системи MAX+PLUS II.
18. Графічне з'єднання ланцюгів (шин) у графічному редакторі системи MAX+PLUS II.
19. З'єднання ланцюгів і шин по імені в графічному редакторі системи MAX+PLUS II.
20. Як здійснюється масштабування й нормування зображення в графічному редакторі системи MAX+PLUS II?
21. Як задається режим нерозривності ланцюгів/шин?
22. Як задати шрифт і його розміри при роботі в графічному редакторі системи MAX+PLUS II?
23. Порядок виконання проекту в графічному редакторі системи MAX+PLUS II.

## 1.3 Лабораторна робота №2

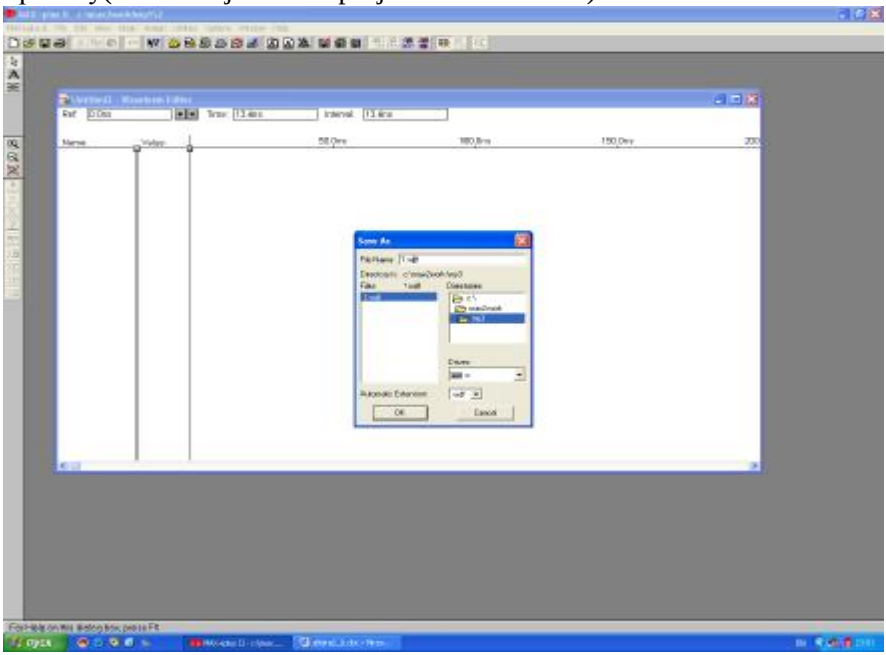
Синтез цифрових пристроїв за допомогою сигнальних описів з використанням системи MAX+PLUS II фірми ALTERA

**Ціль роботи** – Вивчення основних можливостей система проектування MAX+PLUS II фірми ALTERA. Навчитися задавати тестові вхідні сигнали й проводити аналіз вихідних сигналів при різних впливах у редакторі Waveform Editor.

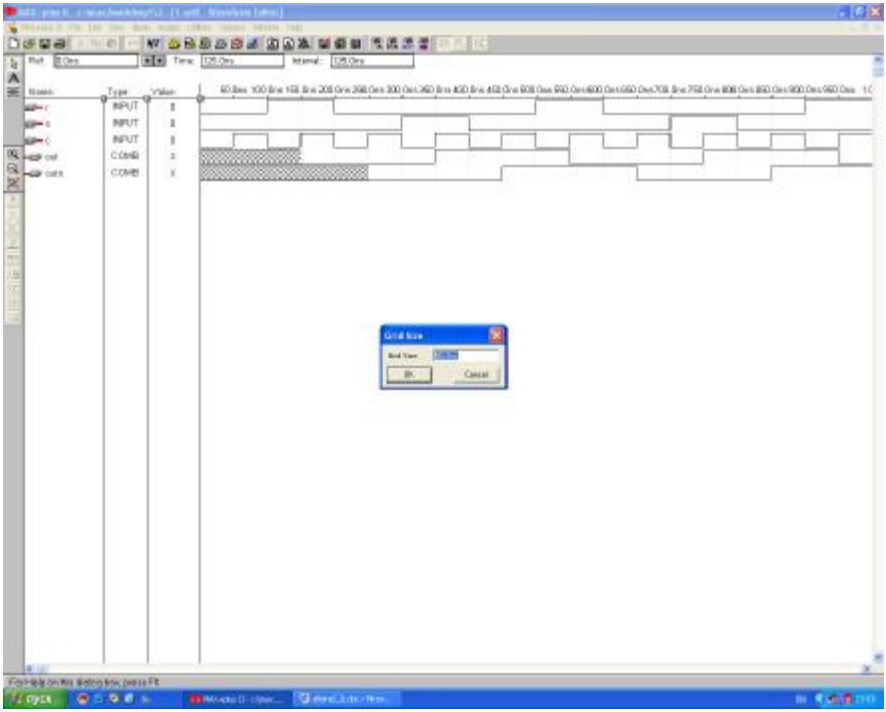
### 1.3.1 Порядок виконання роботи

1.3.1.1 Запускаємо оболонку пакета і з меню Max+plusII запускаємо Waveform Editor.

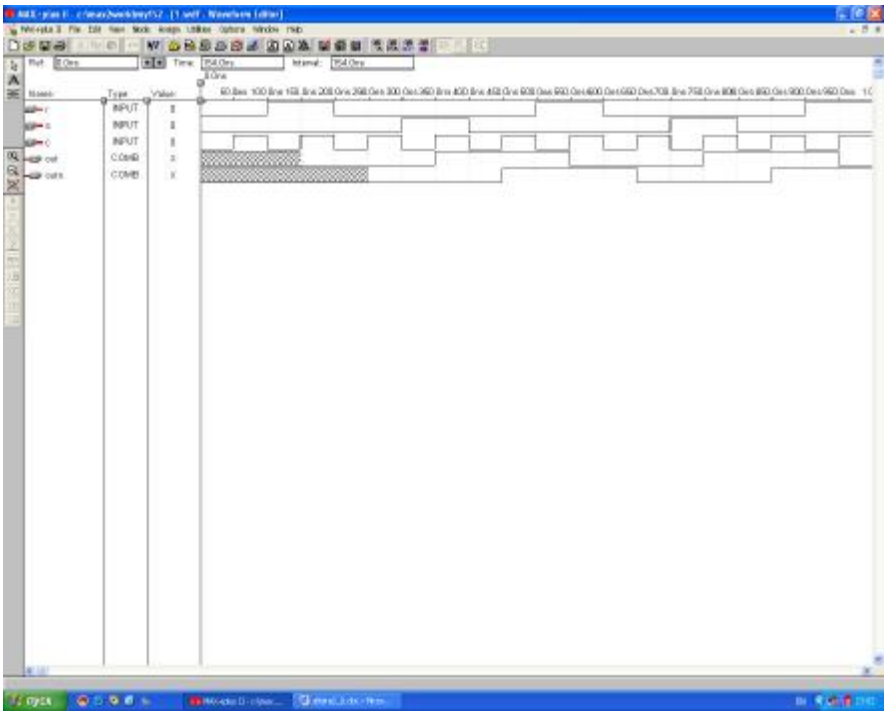
1.3.1.2 У вікні, що відкрилося, таким же чином як і в попередній роботі створюємо набір сигналів, але зберігаємо його з розширенням wdf (не scf). Після цього необхідно вказати, що даний файл є основою проекту(File->Project->Set project to current file).



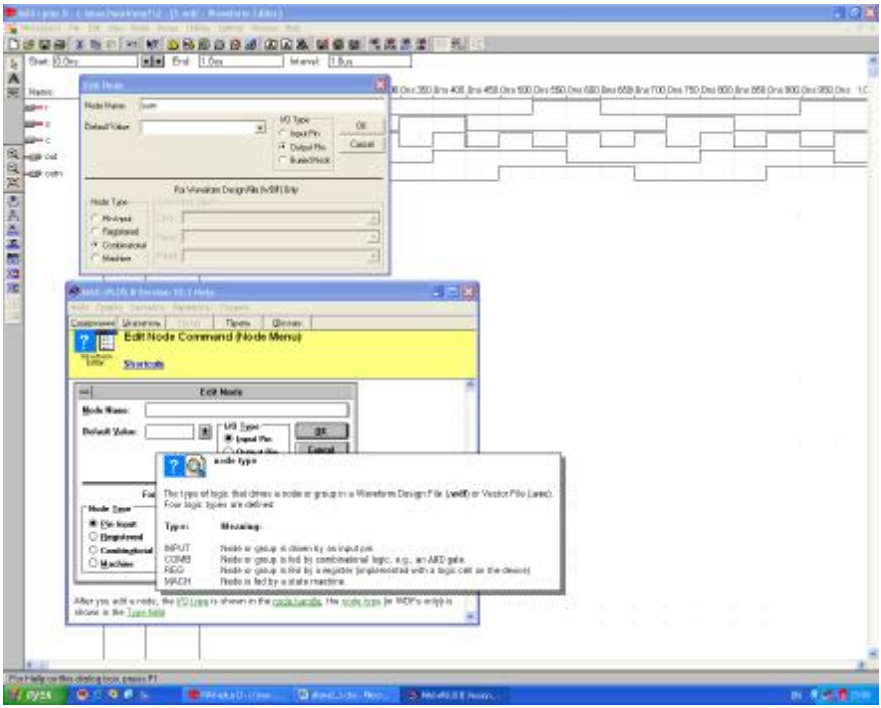
1.3.1.3 При створенні сигналів можна регулювати розмір кроку часової шкали
(Option->grid size) і сам розмір шкали (File->End time).



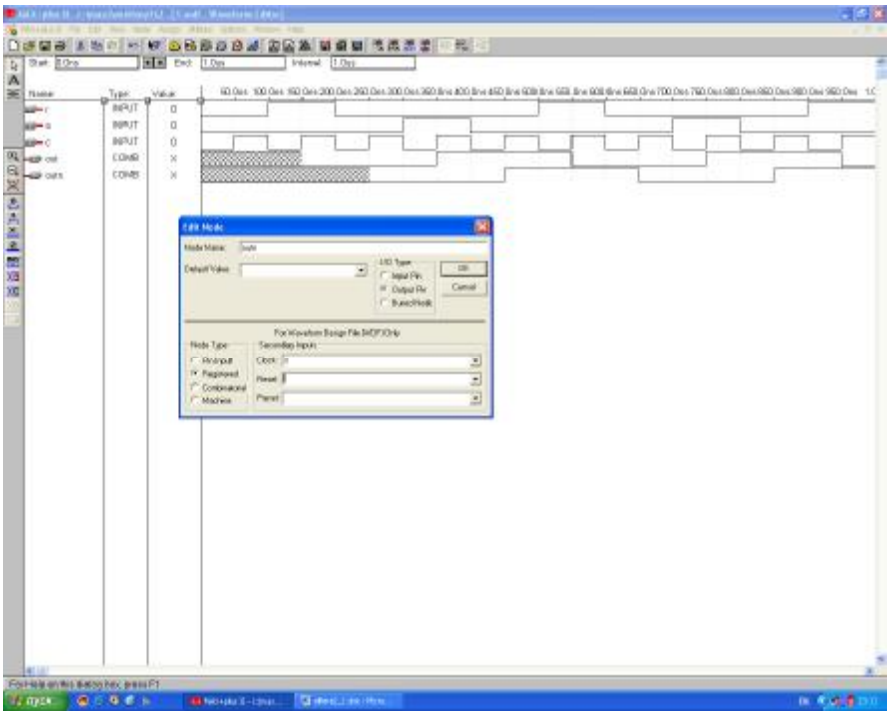
1.3.1.4 При створенні набору сигналів необхідно, щоб вони найбільш повно описували відповідність між вхідними й вихідними впливами. Для синхронного RSC – тригера набір сигналів виглядає в такий спосіб.



1.3.1.5 Обрати тип виходів. Вони бувають Combinatorial ( для комбінаційних пристроїв) і Registered ( для послідовних автоматів)



1.3.1.6 Для тригерів було б правильним вибрати тип registered, але тоді необхідно вказати який вхід використовується для синхронізації, скидання і т.д. . Вважається, що синхронізація може бути тільки по фронту. У результаті такого вибору (якщо не прийняти яких-небудь заходів), виявиться зайнятою ніжка глобальної синхронізації.



1.3.1.7. З панелі кнопок угорі вибираємо «Compiler» та стартуємо компіляцію.
Задавши новий scf файл можна перевірити розроблений пристрій на інших вхідних впливах за допомогою симулятора.

### 1.3.2 Лабораторне завдання

Намалюйте УГП тригера, що має входи керування, згідно з варіантом по табл. 1.3. Побудуйте таблицю станів, задайте сигнальний опис, що повністю описує всі можливі стани та переходи тригеру. За допомогою системы MAX+pLUS II переконайтеся, що отримана схема виконує свої функції.

Таблиця 1.3 – Варіанти завдань

| Параметри | Номери варіантів | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| Вхід R (асинхр.) | п | п | п | п | і | і | і | і | п | п | п | п | і | і | і | і | п | п | п | п | і | і | і | і | п | п | - | і | і | - |
| Вхід С | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с | ф | с |
| Входи R і S (синхронні) | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | п | п | п | п | п | п | п | п | - | - | - | - | - | - |
| Вхід D | - | - | - | - | - | - | - | - | п | п | п | п | п | п | п | п | | | | | - | - | - | - | - | - | - | - | - | - |
| Входи J K | п | п | п | п | п | п | п | п | | | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - | - |

Примітка: тут R – вхід скидання; S – вхід скидання; З – вхід синхронізації; D – інформаційний вхід; J і K – входи дозволу переходів 0→1 і 1→0, відповідно; "і" – інверсний, "п"- прямий, "-" – відсутній, "ф" – прямий динамічний, "с"- інверсний динамічний.

### 1.3.3 Зміст звіту

Звіт по виконаній роботі має містити у собі: титульну сторінку з назвою роботи; мету роботи; УГП мультиплексору та демультиплексору, сигнальний опис тригеру, сигнальні діаграми для перевірки роботи синтезованої цифрової схеми; допоміжна інформація; висновки

### 1.3.4 Контрольні запитання

1. Структурна організація ПЛІС.
2. Основні характеристики сучасних ПЛІС.
3. Маршрут проектування спеціалізованих СБІС на основі ПЛІС із використанням системи MAX+plus II.

4. Призначення й особливості графічного редактора (Graphic Editor).
5. Призначення й особливості символьного редактора (Symbol Editor).
6. Призначення й особливості текстового редактора (Text Editor).
7. Призначення й особливості сигнального редактора (Waveform Editor).
8. Призначення й особливості редактора базового плану (Floorplan Editor).
9. Підготовка й порядок проведення компіляції проекту.
10. Моделювання проекту в системі MAX+plus II.
11. Використовуючи результати моделювання, поясніть роботу логічного елемента 2I.
12. Використовуючи результати моделювання, поясніть роботу дешифратора.
13. Використовуючи результати моделювання, поясніть роботу шифратора.
14. Як задати сітку масштабу на часових діаграмах результатів моделювання?
15. Як підключити періодичну послідовність на вході пристрою, що моделюється й задати параметри цієї послідовності?
16. Як задати часову діаграму довільного імпульсного сигналу?
17. Опишіть процес групування сигналів у шину при відображенні результатів моделювання на тимчасових діаграмах.
18. Як відобразити результати моделювання у двійковому, десятковому, восьмеричному й десятковому кодах?

# 2 МОВА ОПИСУ АПАРАТУРИ AHDL

## 2.1 Загальні теоретичні відомості

### 2.1.1 Загальні відомості про мову опису апаратури AHDL

Мова опису апаратури AHDL розроблена фірмою Altera і призначена для опису комбінаційних та послідовних логічних пристроїв, групових операцій, цифрових автоматів (state machine) і таблиць дійістинності з урахуванням архітектурних особливостей ПЛИС фірми Altera. Вона повністю інтегрується із системою автоматизованого проектування ПЛИС MAX+PLUS II. Файли опису апаратури, написані мовою AHDL, мають розширення *.tdf (Text design file). Для створення tdf-файлу можна використовувати як текстовий редактор системи MAX+PLUS II, так і будь-який інший. Проект, виконаний у вигляді tdf-файлу, компілюється, налагоджується й використовується для формування файлу програмування або завантаження ПЛИС фірми Altera.

Оператори й елементи мови AHDL є досить потужним і універсальним засобом опису алгоритмів функціонування цифрових пристроїв, зручним у використанні. Мова опису апаратури AHDL дає можливість створювати ієрархічні проекти в рамках одної цієї мови або ж в ієрархічному проекті використовувати як tdf-файли, розроблені мовою AHDL, так і інші типи файлів. Для створення проектів на AHDL можна, звичайно, користуватися будь-яким текстовим редактором, але текстовий редактор системи MAX+PLUS II надає ряд додаткових можливостей для введення, компіляції й налагодження проектів.

Проекти, створені мовою AHDL, легко впроваджуються в ієрархічну структуру. Система MAX+PLUS II дозволяє автоматично створити символ компонента, алгоритм функціонування якого описується tdf-файлом, і потім вставити його у файл схемного опису ( Gdf-Файл). Подібним же чином можна вводити власні функції у будь-який tdf-файл, крім того є понад 300 макрофункцій, є вже розробленими фірмою Altera. Для всіх функцій, що входять у макробібліотеку системи MAX+PLUS II, фірма Altera надає файли з розширенням *.inc, які використовуються у операторі включення INCLUDE.

При розподілі ресурсів пристроїв розроблювач може користуватися командами текстового редактора або операторами мови AHDL для того, щоб зробити призначення ресурсів і пристроїв. Крім того, розроблювач може або тільки перевірити синтаксис або виконати повну компіляцію для налагодження й запуску проекту. Будь-які помилки автоматично виявляються оброблювачем повідомлень і відображаються у вікні текстового редактора.

При роботі з AHDL слід дотримувати так званих "Золотих правил" (Golden Rules). Виконання цих правил дозволить ефективно застосовувати мову AHDL і уникнути багатьох помилок:

- дотримуйте форматів і правил присвоєння імен, описані в посібнику зі стилів AHDL, щоб програма легко читалася й містила менше помилок;

- незважаючи на те, що мова AHDL не розрізняє прописні й малі літери, Altera рекомендує для поліпшення читаності використовувати прописні букви для ключових слів;

- не застосовуйте вкладені конструкції умовного оператора If, якщо можна використовувати оператор вибору Case;

- рядок в tdf-файлі може бути довжиною до 255 символів. Однак слід прагнути до того, щоб довжини рядка вміщувалась на екрані. Рядки закінчуються натисканням клавіші Enter;

- новий рядок можна починати в будь-якому вільному місці, тобто на місцях порожніх рядків, табуляцій, пробілів. Основні конструкції мови відділяються порожнім простором (пробіл);

- ключові слова, імена й числа повинні розділятися відповідними символами або операторами й/або одним або більше пробілами;

- коментарі повинні бути вкладені в символи відсотка (%). Коментар може включати будь-який символ, крім символу %, оскільки компілятор системи MAX+PLUS II ігнорує все що міститься поміж символів відсотків. Коментарі не можуть бути вкладеними;

- при з'єднанні одного примітива з іншим використовуйте тільки "дозволені" зв'язки між ними, не всі примітиви можуть з'єднуватися один з одним.

- використовуйте тільки макрофункції EXPDFF, EXPLATCH, NANDLTCH і NORLTCH системи, що входять у макробібліотеку, MAX+PLUS II. Не створюйте свої власні структури перехресних зв'язків. Уникайте багаторазового зв'язування разом EXPDFF,

EXPLATCH, NANDLTCH і NORLTCH. Численні приклади цих макрофункцій повинні завжди розділятися примітивами LCELL.

- якщо численні двоспрямовані або вихідні виводи зв'язані разом, розробник не може використовувати оператор Pin Connection для з'єднання виводів при функціональному моделюванні з апаратною підтримкою або функціональному тестуванні;

- немає необхідності створювати прототипи функцій для примітивів. Однак розроблювач може перевизначити примітиви в оголошеннях прототипів функцій для зміни порядку виклику входів у вашому tdf-файлі;

- не редагуйте файл Fit. Якщо розроблювач бажає відредагувати призначення для проекту, необхідно зберегти спочатку файл Fit як tdf-файл або зробити зворотне призначення за допомогою команди Project Back-Annotate і відредагувати їх за допомогою команд Chip to Device, Pin/LC/Chip і Enter Assignments;

- якщо розроблювач прагне завантажити регістр по певному фронту глобального тактового сигналу Clock, фірма Altera рекомендує використовувати для керування вхід Clock Enable одного із тригерів типу Enable: DFFE, TFFE, JKFFE або SRFFE;

- коли розроблювач починає працювати з новим файлом проекту, відразу ж необхідно задати сімейство ПЛІС, на які орієнтований проект, за допомогою конструкції Family для того, щоб надалі мати можливість скористатися макрофункціями, специфічними для даного сімейства. Якщо розроблювач не задасть сімейство, воно буде вважатися таким же, як і в поточному проекті;

- використовуйте опцію Design Doctor для перевірки надійності логіки проекту під час компіляції.

Надавані за замовчуванням фірмою Altera стилі для логічного синтезу мають різні налаштування для різних сімейств пристроїв, що забезпечує більш ефективне використання архітектури кожного пристрою. Коли розроблювач використовуєте який-небудь із цих стилів, його налаштування зміняться, при перехід до іншого сімейства пристроїв. Після зміни сімейства необхідно перевірити нові налаштування стилю.

### 2.1.2 Використання чисел і констант у мові AHDL

Числа використовуються для подання констант в булевих виразах та рівняннях. Мова AHDL підтримує всі комбінації десяткових, двійкових, восьмеричних і шіснадцяткових чисел.

Нижче наведений файл decode1.tdf, який являє собою дешифратор адреси, що генерує високий активний рівень сигналу дозволу доступу до шини, якщо адреса рівна шіснадцятковому числу 370h.

```
subdesign decode1
   (
      address[15..0] : INPUT;
      chip_enable : OUTPUT;
   )
BEGIN
   chip_enable = (address[15..0] == H"0370");
END;
```

У цьому прикладі десяткові числа використані для визначення розмірності масиву біт, у якому міститься адреса шини. Шіснадцятковим числом H"0370" записане значення адреси, при якій забезпечується високий рівень сигналу.

У файлі AHDL можна використовувати константи для описових імен чисел. Таке ім'я, використовуване протягом усього файлу, може бути більш інформативним, ніж число; наприклад, ім'я UPPER_LI несе більше інформації, ніж число 103. У мові AHDL константи вводяться оголошенням CONSTANT.

Перевага використання констант особливо помітна, якщо те саме число використовується у файлі кілька раз. Тоді, якщо його потрібно змінити, змінюють його тільки один раз в оголошенні константи.

### 2.1.3 Комбінаційна логіка

Логічна схема називається комбінаційною, якщо в заданий момент часу виходи є тільки функціями входів у цей момент часу. Комбінаційна логіка в мові AHDL реалізована булевими виразами й рівняннями, таблицями дійстинності й великою кількістю макрофункцій. У число прикладів комбінаторних логічних функцій входять: дешифратори, мультиплексоры й суматори.

Булеві вирази - це аналіз вузлів, чисел, констант і інших булевих виразів, що виділяються операторами, компараторами й, можливо, згруповані в круглих дужках. Булеві рівняння встановлюють рівність між вузлом або групою й булевих виразів.

Як приклад наведений файл boole1.tdf, у якому дано два простих булевих вирази, що представляють два логічні елементи.

```
subdesign boole1
     (
        a0, a1, b : INPUT;
        out1, out2 : OUTPUT;
     )
BEGIN
   out1 = a1 & !a0;
   out2 = out1 # b;
END;
```

Тут вихід out1 утворюється у результаті логічної операції I, що застосована до входу a1 і інвертованого входу a0, а вихід out2 утворюється у результаті застосування логічної операції АБО до виходу out1 та входу b. Оскільки ці рівняння обробляються одночасно, послідовність їх проходження у файлі не важлива.

Вузол, який оголошується в секції змінних VARIABLE в оголошенні NODE, можна використовувати для зберігання проміжних виразів.

Це корисно робити, якщо булеві вирази повторюється кілька раз і їх доцільно замінити іменем вузла. Наведений вище файл boole1.tdf можна переписати по-іншому:

```
subdesign boole2
   (
      a0, a1, b : INPUT;
      out : OUTPUT;
   )
VARIABLE
   a_equals_2 : NODE;
BEGIN
   a_equals_2 = a1 & !a0;
   out = a_equals_2 # b;
END;
```

Тут оголошується вузол a_equals_2 і йому привласнюється значення виразу a1 & !a0. Використання вузлів допомагає заощаджувати ресурси пристроїв, якщо вузол використовується в декількох виразах.

Важливим поняттям AHDL є група. Група може містити в собі до 256 елементів (біт), розглядається як сукупність вузлів і бере участь у різних діях як єдине ціле. У булевих рівняннях група може бути прирівняна булевому виразу, іншій групі, одному вузлу, VCC, GND,1 або 0. У кожному випадку значення групи різні.

Якщо група визначена, для короткого запису всього діапазону ставлять дві квадратні дужки [ ]. Наприклад, групу a[4..1] можна коротко записати як a[ ].

Умовна логіка робить вибір між режимами залежно від логічних входів. Для реалізації умовної логіки використовуються оператори IF або CASE:

В операторові IF оцінюється одне або декілька булевих виразів і потім описуються режими для різних значень цих виразів.

В операторові CASE дається список альтернатив, які є для кожного можливого значення деякого виразу. Оператор оцінює значення виразу й по ньому вибирає режим у відповідності зі списком.

### 2.1.4 Логіка оператора IF

Як приклад розглянемо файл priority.tdf, у якому описаний кодувальник пріоритету, який перетворить рівень самого пріоритетного активного входу в значення. Він генерує дворазрядний

код, що показує вхід з найвищим пріоритетом, на який надійшов сигнал VCC.

```
subdesign priority
   (
      low, middle, high : INPUT;
      highest_level[1..0] : OUTPUT;
   )
BEGIN
   IF high THEN
      highest_level[] = 3;
   ELSIF middle THEN
      highest_level[] = 2;
   ELSIF low THEN
      highest_level[] = 1;
   ELSE
      highest_level[] = 0;
   END IF;
END;
```

Тут оцінюються входи low, middle і high, щоб визначити, чи дорівнюють вони VCC. На виході вийде код, відповідний до пріоритету того входу, на який був запущений VCC. Якщо жоден вхід не запущений, значення коду буде рівно 0.

Логіка оператора CASE. Як приклад розглянемо файл decoder.tdf, що реалізує функції дешифратора, що перетворює код із дворазрядного двійкового в чотириразрядний позиційний. У результаті його роботи два дворазрядні двійкові входи перетворяться в один "температурний код", який так називається тому, що чотири його припустимі значення містять по одній одиниці: 0001, 0010, 0100, 1000.

```
subdesign decoder
   (
      code[1..0] : INPUT;
      out[3..0] : OUTPUT;
   )
BEGIN
```

```
    CASE code[] IS
    WHEN 0 => out[] = B"0001";
    WHEN 1 => out[] = B"0010";
    WHEN 2 => out[] = B"0100";
    WHEN 3 => out[] = B"1000";
    END CASE;
END;
```

Тут група входу code[1..0] може приймати значення 0, 1, 2, 3. Залежно від реального коду активізується відповідна гілка оператора й тільки вона одна в цей момент часу. Наприклад, якщо значення на на входах code[] дорівнює 1, на виході out встановлюється значення B"0010".

Оператори IF і CASE схожі. Іноді використання кожного з них приводить до тих самих результатів. Однак між цими двома операторами існують кілька важливих відмінностей:

В операторові IF можна використовувати будь-які булеві вирази. Кожний вираз, що записаний у речені , що йде за IF або ELSEIF, може не бути пов'язаним з іншими виразами в операторі. В операторі CASE один єдиний вираз порівнюється зі значенням, що перевіряється.

У результаті інтерпретації оператора IF може бути згенерирована логіка, занадто складна для компілятора системи MAX+PLUS II. У наведеному нижче прикладі показано, як компілятор інтерпретує оператор IF. Якщо a й b - складні вирази, то інверсія кожного з них дасть ще більш складні вирази.

Розглянемо опис типових комбінаційних схем на AHDL. Дешифратор містить комбінаторну логіку, яка перетворить вхідні схеми у вихідні значення або задає вихідні значення для вхідних схем. Для створення дешифратора в мові AHDL використовується оголошення таблиці істинності TABLE.

Нижче наведений файл дешифратора адреси decode3.tdf для шеснадцяткової мікропроцесорної системи.

```
subdesign decode3
(
    addr[15..0], m/io : INPUT;
    rom, ram, print, sp[2..1] : OUTPUT;
```

```
      )
      BEGIN
         TABLE
            m/io, addr[15..0] => rom, ram, print,
sp[];
            1, B"00XXXXXXXXXXXXX" => 1, 0, 0,
B"00";
            1, B"100XXXXXXXXXXXX" => 0, 1, 0,
B"00";
            0, B"0000001010101110" => 0, 0, 1,
B"00";
            0, B"0000001011011110" => 0, 0, 0,
B"01";
            0, B"0000011101110000" => 0, 0, 0,
B"10";
         END TABLE;
      END;
```

У даному прикладі існують тисячі можливих варіантів входу (адреси), тому було б непрактично зводити їх усі в таблицю. Замість цього, можна позначити символом "X" несуттєві розряди або ті, що не впливають на вихід. Наприклад, вихідний сигнал rom (ПЗП) буде високим для всіх 16384 варіантів адреси addr[15..0], які починаються з 00. Тому вам потрібно тільки вказати загальну для всіх варіантів частину коду, тобто 00, а в інших розрядах коду поставити "X". Такий прийом дозволить зробити проект, що вимагає менше пристроїв і ресурсів.

Наведений нижче приклад decode4.tdf показує використання стандартної параметризуючої функції lpm_decode для вирішення задачі розробки дешифратора, аналогічного прикладу decode1.tdf

```
      INCLUDE "lpm_decode.inc";
      subdesign decode4
      (
         address[15..0] : INPUT;
         chip_enable : OUTPUT;
      )
      BEGIN
```

```
        chip_enable =
lpm_decode(.data[]=address[])
        WITH (LPM_WIDTH=16, LPM_DECODES=210)
        RETURNS (.eq[H"0370"]);
     END;
```

Іноді для змінних зручно використовувати значення за замовчуванням. Можна визначити значення за замовчуванням для вузла або групи, які будуть автоматично використовуватися для них, якщо у файлі їх значення не будуть задані. Мова AHDL дозволяє неодноразово надавати значення вузлу або групі у файлі. Якщо при цьому відбудеться конфлікт, система автоматично буде використовувати значення за замовчуванням. Якщо значення за замовчуванням не були задані, використовується значення GND.

Оголошення значень за замовчуванням DEFAULTS можна використовувати для завдання змінних у таблиці дійсності, операторах IF і CASE.

Нижче приводиться файл default1.tdf, у якому відбувається оцінка входів і вибір відповідного ASCII коду.

```
     subdesign default1
        (
           i[3..0] : INPUT;
           ascii_code[7..0] : OUTPUT;
        )
     BEGIN
        DEFAULTS
        ascii_code[] = B"00111111";% ASCII
question mark"?" % END DEFAULTS;
        TABLE
        i[3..0] => ascii_code[];
        B"1000" => B"01100001"; % "a" %
        B"0100" => B"01100010"; % "b" %
        B"0010" => B"01100011"; % "c" %
        B"0001" => B"01100100"; % "d" %
        END TABLE;
     END;
```

Якщо значення входу збігається з одним зі значень у лівій частині таблиці, код на виході набуває відповідного значення ASCII коду в правій частині таблиці. Якщо вхідне значення не збігається з жодним з (лівих) табличних, виходу буде привласнене значення за замовчуванням B"00111111", у вищенаведеному прикладі – знак питання.

У наведеному нижче файлі default2.tdf показано, як при багаторазовому присвоюванні вузлу різних значень виникає конфлікт і як він вирішується засобами AHDL.

```
subdesign default2
   (
      a, b, c : INPUT;
      select_a, select_b, select_c : INPUT;
      wire_or, wire_and : OUTPUT;
   )
BEGIN
   DEFAULTS
      wire_or = GND;
      wire_and = VCC;
   END DEFAULTS;
   IF select_a THEN
      wire_or = a;
      wire_and = a;
   END IF;
   IF select_b THEN
      wire_or = b;
      wire_and = b;
   END IF;
   IF select_c THEN
      wire_or = c;
      wire_and = c;
   END IF;
END;
```

У даному прикладі вихід wire_or установлюється рівним a, b, або із залежно від вхідних сигналів select_a, select_b і select_c. Якщо

жоден з них не рівний VCC, то вихід wire_or приймає значення за замовчуванням, рівне GND.

Якщо більше одного сигналу (select_a, select_b або select_c) рівні VCC, то wire_or дорівнює результату логічної операції АБО над відповідними вхідними сигналами. Наприклад, якщо select_a і select_b рівні VCC, те wire_or рівно a АБО b.

Із сигналом wire_and проводяться аналогічні дії, але він стає рівним VCC, коли вхідні сигнали select рівні VCC, і рівний логічному «І» від відповідних вхідних сигналів, якщо більш, ніж один з них рівний VCC.

Слід нагадати ще про реалізацію логіки з активним низьким рівнем. Значення сигналу з низьким активним рівнем рівно GND. Сигнали з низьким активним рівнем можуть бути використані для керування пам'яттю, периферійними пристроями й мікропроцесорними чипами. Фірма Altera рекомендує позначати як-небудь імена сигналів з низьким активним рівнем, наприклад, першим символом в імені ставити символ "/", який не є оператором, і використовувати його постійно.

Система MAX+PLUS II дозволяє конфігурувати порти вводу/виводу (I/O) у пристроях Altera як двоспрямовані порти. Двоспрямований вивід задається як порт BIDIR, який приєднується до виходу примітива tri. Сигнал між цим виводом і буфером із трьома станами є двоспрямованим, і його можна використовувати в проекті для інших логічних схем.

Приклад, що приводиться нижче, bus_reg2.tdf реалізує регістр, який робить вибірку значення, отриманого на шині із трьома станами, а також може передати назад на шину збережене значення.

```
subdesign bus_reg2
(
      clk : INPUT;
      oe : INPUT;
      io : BIDIR;
)
BEGIN
      io = tri(DFF(io, clk,, ), oe);
END;
```

Двоспрямований сигнал io, що запускається примітивом tri, використовується в якості входу d для D-Тригера (DFF). Коми наприкінці списку параметрів відокремлюють місця для сигналів тригера clrn і prn. Ці сигнали за замовчуванням установлені в неактивний стан.

Двоспрямований вивід можна також використовувати для приєднання tdf-файлу більш низького рівня до виводу з високим рівнем. Прототип функції для tdf-файлу більш низького рівня повинен містити двоспрямований вивід у контексті RETURNS. У наведеному нижче файлі bidir1.tdf дано чотири приклади використання вищенаведеної макрофункції bus_reg2.

```
FUNCTION bus_reg2 (clk, oe) RETURNS (io);
subdesign bidir1
     (
        clk, oe : INPUT;
        io[3..0] : BIDIR;
     )
BEGIN
     io0 = bus_reg2(clk, oe);
     io1 = bus_reg2(clk, oe);
     io2 = bus_reg2(clk, oe);
     io3 = bus_reg2(clk, oe);
END;
```

### 2.1.5 Послідовна логіка в AHDL

Логічна схема називається послідовною, якщо виходи в заданий момент часу є функцією входів не тільки в той же момент, але й в усі попередні моменти часу. Таким чином, у послідовну схему повинні входити деякі елементи пам'яті (тригери). У мові AHDL послідовна логіка реалізована цифровими автоматами з пам'яттю (state machines), регістрами й тригерами. При цьому засоби опису цифрових автоматів займають особливе місце. Крім того, до послідовних логічних схем відносять різні лічильники й контролери.

Оголошення регістрів. Регістри використовуються для зберігання значень даних і проміжних результатів лічильника,

тактування здійснюється синхросигналом. Регістр створюється його оголошенням у секції VARIABLE.

Для приєднання прикладу примітива, макрофункції або цифрового автомата до іншої логіки в Tdf-Файлі можна використовувати порти. Порт прикладу описується в наступному форматі: <ім'я прикладу>.<ім'я порту>.

Ім'я порту - це вхід або вихід примітива, макрофункції або цифрового автомата, що є синонімом імені виводу у файлах проектів ( Gdf-Файл), *.WDF і інших.

Нижче приводиться файл bur_reg.tdf, що містить байтовий регістр, який фіксує значення зі входів d на виходах q на фронті синхроімпульсу, коли рівень завантаження високий.

```
SUBDESIGN bur_reg
   (
      clk, load, d[7..0] : INPUT;
      q[7..0] : OUTPUT;
   )
VARIABLE
   ff[7..0] : DFFE;
BEGIN
   ff[].clk = clk;
   ff[].ena = load;
   ff[].d = d[];
   q[] = ff[].q;
END;
```

Як видно з файлу, регістр оголошений у секції VARIABLE як D-Тригер з дозволом (DFFE). У першому булевому виразі в логічній секції відбувається під'єднання входу тактового сигналу підпроекту до портів тактового сигналу тригерів ff[7..0]. У другому вмразі тактовий сигнал, що синхронізує, з'єднується із входом завантаження. У третьому – входи даних підпроекту з'єднуються з портами даних тригерів ff[7..0]. У четвертому – вмпазі виходи підпроекту з'єднуються з виходами тригерів. Усі чотири вирази працюють одночасно (не послідовно), тому їх порядок не має значення.

Можна також у секції VARIABLE оголосити T-, JK- і RS-тригери й потім використовувати їх у логічній секції. При використанні T-Тригерів доведеться в третьому рівнянні змінити порт d на t. При використанні JK- і RS-тригерів замість третього рівняння прийдеться записати два рівняння, у яких відбувається з'єднання портів j і k або s і r зі сигналами.

При завантаженні регістру за фронтом глобального тактового сигналу фірма Altera рекомендує використовувати вхід дозволу одного з регістрів: DFFE, TFFE, JKFFE або SRFFE, щоб контролювати завантаження регістру.

Можна оголосити регістрові виходи, якщо оголосити виходи підпроекта як D-тригер(и) в секції VARIABLE.

Наведений нижче файл reg_out.tdf забезпечує ті ж функції, що й попередній файл bur_reg.tdf, але має регістрові виходи.

```
SUBDESIGN reg_out
   (
      clk, load, d[7..0] : INPUT;
      q[7..0] : OUTPUT;
   )
VARIABLE
   q[7..0] : DFFE;
BEGIN
   q[].clk = clk;
   q[].ena = load;
   q[] = d[];
END;
```

При присвоєнні значення регістровому виходу в логічній секції це значення формує вхідні d сигнали регістрів. Вихід регістру не змінюється доти, поки не прийде фронт синхросигналу. Для визначення тактуючого сигналу регістру потрібно використовувати опис у наступному форматі: <ім'я вихідного виводу>.clk для входу тактуючого сигналу регістру в логічній секції. Глобальний синхросигнал можна реалізувати примітивом GLOBAL або вибором у діалоговому вікні компілятора Logic Synthesis (логічний синтез) опції Automatic Global Clock.

Кожний D-тригер, що оголошений у секції VARIABLE, збуджує вихід з таким же іменем, тому можна звертатися до q виходам оголошених тригерів без використання q порту цих тригерів.

Створення лічильників.

Лічильниками називаються послідовні логічні схеми для підрахунку тактових імпульсів. У деяких лічильниках реалізований підрахунок вперед та назад (реверсивні лічильники), у деякі лічильники можна завантажувати дані, а також обнуляти їх. Лічильники звичайно визначають як D-тригери (DFF і DFFE) і використовують оператори IF.

Нижче наведено вміст файлу ahdlcnt.tdf, який реалізує 16-бітовий лічильник зі скиданням, що також може завантажуватися.

```
SUBDESIGN ahdlcnt
   (
      clk, load, ena, clr, d[15..0] : INPUT;
      q[15..0] : OUTPUT;
   )
VARIABLE
   count[15..0] : DFF;
BEGIN
   count[].clk = clk;
   count[].clrn = !clr;
   IF load THEN
         count[].d = d[];
   ELSIF ena THEN
         count[].d = count[].q + 1;
   ELSE
         count[].d = count[].q;
   END IF;
   q[] = count[];
END;
```

У цьому описі в секції VARIABLE оголошено 16 D-Тригерів та їм привласнені імена від count0 до count15. В операторі IF визначається значення, що завантажується в тригери по фронту синхросигналу (наприклад, якщо завантаження запускається VCC, те тригерам привласнюється значення d[ ]).

### 2.1.6 Цифрові автомати з пам'яттю (state mashine)

Цифрові автомати так само, як таблиці дійсності й булеві рівняння, легко реалізуються в мові AHDL. Мова структурована, тому користувач може або сам призначити біти й значення станів, або дати цю роботу компіляторові системи MAX+PLUS.

Компілятор використовує патентовані перспективні евристичні алгоритми, що дозволяють зробити такі автоматичні призначення станів, які мінімізують логічні ресурси, що потрібні для реалізації цифрового автомата.

Користувачеві просто потрібно намалювати діаграму і побудувати таблицю станів. Потім компілятор виконує автоматично наступні функції:

- призначає біти, вибираючи для кожного біта або T-тригер, або D-тригер;

привласнює значення станів;

- застосовує складні методи логічного синтезу для одержання рівнянь.

За бажанням користувача можна задати в Tdf-файлі переходи в машині станів за допомогою оголошення таблиці дійсності переходів.

У мові AHDL для завдання цифрового автомата потрібно включити в Tdf-файл наступні елементи:

- оголошення цифрового автомата (у секції VARIABLE);
- булеві рівняння керування (у логічній секції);
- переходи між станами (у логічній секції).

Цифрові автомати в мові AHDL можна також експортувати й імпортувати, роблячи обмін між файлами типу TDF і ( Gdf-файл) або *.WDF; при цьому вхідний або вихідний сигнал задається як порт цифрового автомата в секції SUBDESIGN.

Цифровий автомат задають у секції VARIABLE шляхом оголошення імені цифрового автомата, його станів і, можливо, вихідних бітів.

Нижче наведений файл simple.tdf, який реалізує функцію D-тригера.

```
SUBDESIGN simple
   (
      clk : INPUT;
      reset : INPUT;
      d : INPUT;
       q : OUTPUT;
   )
VARIABLE
   ss : MACHINE WITH STATES (s0, s1);
BEGIN
   ss.clk = clk;
   ss.reset = reset;
   CASE ss IS
   WHEN s0 =>
      q = GND;
         IF d THEN
            ss = s1;
         END IF;
   WHEN s1 =>
      q = VCC;
      IF !d THEN
         ss = s0;
      END IF;
   END CASE;
END;
```

У даному файлі в секції VARIABLE оголошений цифровий автомат (state machine) ss. Стани автомату визначаються як s0 і s1. Біти станів не визначені. Сигнали Clock, Reset і Enable керують тригерами регістру станів у цифровому автоматі. Ці сигнали задаються булевими виразами керування в логічній секції.

У попередньому прикладі (simple.tdf) синхросигнал цифрового автомата (Clock) формується входом clk. Асинхронний сигнал скидання цифрового автомата (Reset) формується входом reset, що має високий активний рівень. Для підключення сигналу відмикання (Enable) потрібно додати в даний файл проекту рядок "ena : INPUT;" у секцію SUBDESIGN, а також додати в логічну секцію булевий вираз "ss.ena = ena;".

Для завдання вихідних значень можна використовувати оператори IF і CASE. У наведеному вище прикладі (файл simple.tdf) значення виходу q встановлюється рівним GND, якщо цифровий автомат ss перебуває в стані s0, і рівним VCC, коли він перебуває в стані s1. Ці присвоювання робляться в реченнях WHEN оператора CASE.

Вихідні значення можна також задавати в таблицях дійсності переходів. Переходи між станами визначають умови, при яких машина переходить у новий стан. Переходи в машині станів задаються шляхом умовного присвоєння стану в рамках однієї конструкції, що описує режим. Для цієї мети рекомендується використовувати оператор CASE або таблицю дійсності.

У наведеному вище прикладі (файл simple.tdf) переходи для кожного стану визначені в конструкціях WHEN оператора CASE.

Біт стану - це вихід тригера, використовуваний для зберігання одного біта значень цифрового автомата. У більшості випадків для мінімізації логічних ресурсів слід надати компіляторові системи MAX+PLUS II право присвоєння бітів і значень стану. Однак користувач може зробити це самостійно в оголошенні цифрового автомата, якщо, наприклад, він прагне, щоб певні біти були виходами цифрового автомата.

Якщо виходи цифрового автомата залежать тільки від його стану, їх можна задати в пропозиції WITH STATES оголошення цифрового автомата. Це зробить їх менш підданими помилкам. Крім того, у деяких випадках для логічних операцій буде потрібно менше логічних комірок.

Нижче наведений приклад, у якому реалізований автомат Мура із чотирма станами.

```
SUBDESIGN moore1
  (
    clk : INPUT;
    reset : INPUT;
    y : INPUT;
    z : OUTPUT;
  )
VARIABLE
 % current current %
```

```
 % state   output %
  ss :MACHINE OF BITS (z) WITH STATES (
   s0   =   0,
   s1   =   1,
   s2   =   1,
   s3   =   0);
BEGIN
  ss.clk = clk;
  ss.reset = reset;
  TABLE
% current current next %
% state   input   state %
  ss,      y=>     ss;
  s0,      0=>     s0;
  s0,      1=>     s2;
  s1,      0=>  s0;
  s1,      1=>  s2;
  s2,      0=>  s2;
  s2,      1=>  s3;
  s3,      0=>  s3;
  s3,      1=>  s1;
END TABLE;
END;
```

У даному прикладі стани визначені в оголошенні цифрового автомата. Переходи між станами визначені в таблиці next_state, яка задана в оголошенні таблиці істинності. У даному прикладі машина має чотири стани та тільки один біт стану z. Компілятор системи MAX+PLUS II автоматично додає ще один біт і робить відповідні присвоєння цієї синтезованої змінної для того, щоб вийшла машина із чотирма станами. Такий цифровий автомат (state machine) вимагає, принаймні, двох бітів.

Якщо значення станів використовуються як виходи ( як у файлі moorel.tdf), для проекту буде потрібно менше логічних комірок, але, можливо, логічні комірки зажадають більше логіки, щоб збудити входи тригера. У цьому випадку модуль логічного синтезатора компілятора, можливо, не зможе повністю мінімізувати автомат.

Інший спосіб побудови цифрового автомата полягає в тому, щоб не робити присвоєння станів і явно оголосити вихідні тригери.

Цей альтернативний метод використаний у наведеному нижче файлі moore2.tdf.

```
SUBDESIGN moore2
(
  clk : INPUT;
  reset : INPUT;
  y : INPUT;
  z : OUTPUT;
)
VARIABLE
  ss : MACHINE WITH STATES (s0, s1, s2, s3);
  zd : NODE;
BEGIN
  ss.clk = clk;
  ss.reset = reset;
  z = DFF(zd, clk, VCC, VCC);
TABLE
% current current next  next %
% state   input   state output %
  ss,      y    => ss,  zd;
  s0,      0    => s0,  0;
  s0,      1    => s2,  1;
  s1,      0    => s0,  0;
  s1,      1    => s2,  1;
  s2,      0    => s2,  1;
  s2,      1    => s3,  0;
  s3,      0    => s3,  0;
  s3,      1    => s1,  1;
END TABLE;
END;
```

У даному прикладі замість того, щоб задати виходи присвоєнням значень станам в оголошенні цифрового автомата, в оголошення таблиці дійсності додано один стовпець з назвою "next output (наступний вихід)". У цьому методі для синхронізації виходів синхросигналом використовується D-Тригер, виклик якого записаний за допомогою безпосереднього посилання.

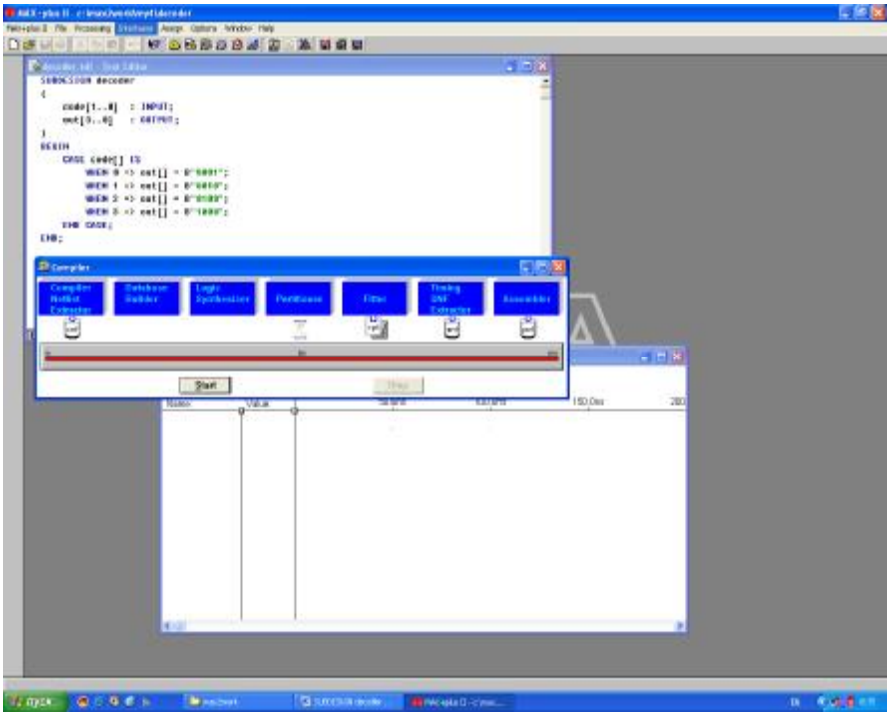
## 2.2 Лабораторна робота №3
## Створення цифрових пристроїв за допомогою мови ADHL та програмного пакета Altera Max+plus II

### 2.2.1 Порядок виконання роботи

2.2.1.1 Описати розроблений пристрій мовою AHDL, текст програми зберегти у файлі decoder.tdf, наприклад:

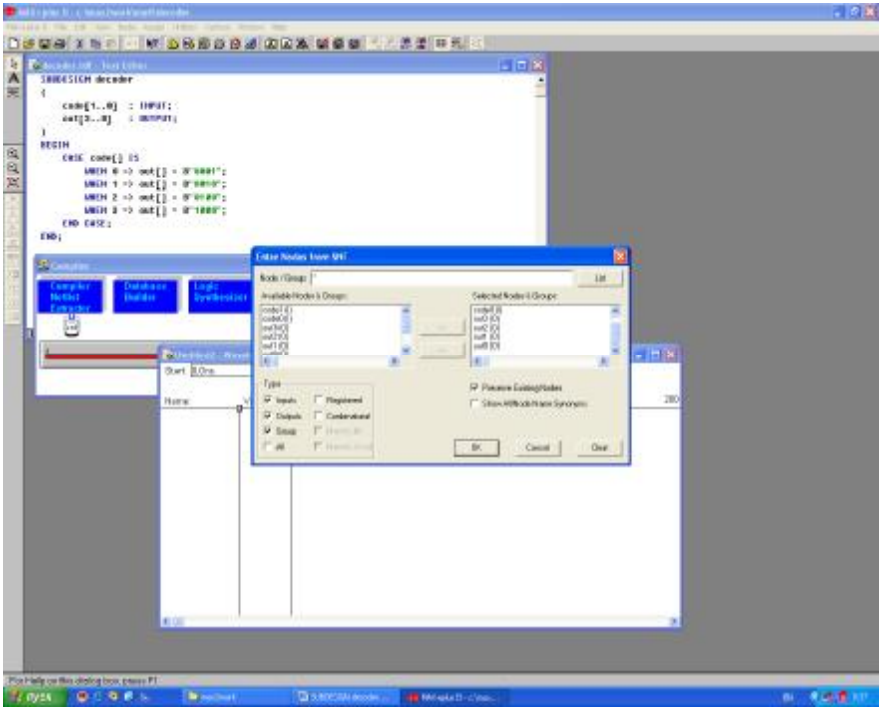
```
SUBDESIGN decoder
(
        code[1..0]      : INPUT;
        out[3..0]       : OUTPUT;
)
BEGIN
        CASE code[] IS
                WHEN 0 => out[] = B"0001";
                WHEN 1 => out[] = B"0010";
                WHEN 2 => out[] = B"0100";
                WHEN 3 => out[] = B"1000";
        END CASE;
END;
```

2.2.1.2. Задати мікросхему на якій буде виконуватися проект та поставити до вершини ієрархії створений нами decoder.tdf.

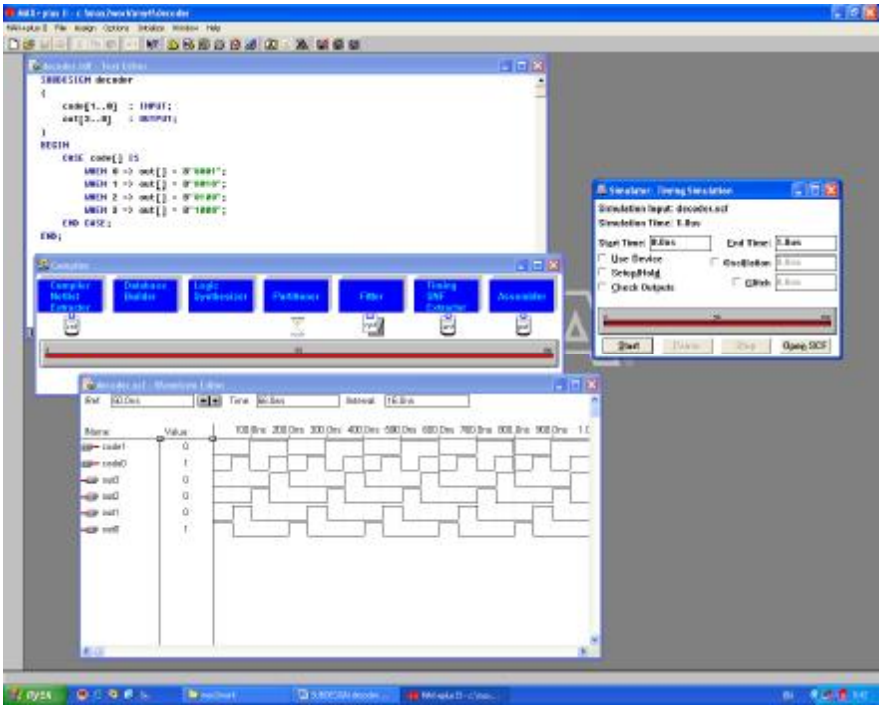


2.2.1.3 Виконати компиляцію програми.

2.2.1.4 Після вдалої компіляції відкрити Waveform editor та з контекстного меню (по кліку правої кнопки) обираємо Enter nodes from SNF. Це спростить створення тестових комбінацій. У віконці, що з'являється жмемо List та обираємо входи та виходи, які слід імпортувати з SNF файлу, що створюється на етапі компіляції.



2.2.1.5 Задати тестові сигнали та запустити симуляцію.



## 2.2.2 Лабораторне завдання

2.2.2.1 Синтезуйте демультиплексор, для котрого кількість інформаційних виходів та вид інформаційного входу коду беруться з Таблиця 2.1, згідно з номером варіанту.

Таблиця 2.1 – Варіанти завдань

| Параметри | Номери варіантів | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| Кількість виходів | 3 | 4 | 5 | 6 | 7 | 3 | 4 | 5 | 6 | 7 | 3 | 4 | 5 | 6 | 7 | 3 | 4 | 5 | 6 | 7 | 3 | 4 | 5 | 6 | 7 | 3 | 4 | 5 | 6 | 7 |
| Вид інф. входу: | і | і | і | і | і | п | п | п | п | п | і | і | і | і | і | п | п | п | п | п | і | і | і | і | і | п | п | п | п | п |

Примітка: "і" – інверсний; "п"- прямий; "+" – можна використовувати; "-" – не використовувати.

2.2.2.2 Синтезуйте мультиплексор, що має стільки ж інформаційних входів як демультиплексор (по Таблиця 2.4) виходів і такий тип виходів, як тип входу цього демультиплексору.

2.2.2.3 За допомогою симулятора дослідіть отриману схему та переконайтеся, що вона виконує свої функції.

### 2.2.3 Вміст звіту

Звіт по виконаній роботі має містити у собі: титульну сторінку з назвою роботи; мету роботи; УГП мультиплексору та демультиплексору, опис мовою AHDL, сигнальні діаграми для перевірки роботи синтезованої цифрової схеми; допоміжна інформація; висновки.

### 2.2.4 Контрольні запитання

1 Структура модуля Subdesing мовою AHDL.
2 Алфавіт і вирази мови AHDL.
3 Конструкція IF … ELSE мови AHDL.
4 Конструкція CASE мови AHDL.
5 Конструкція TABLE мови AHDL.
6 Описати мовою AHDL базові елементи цифрової схемотехніки.
7 Описати мовою AHDL елементи Шеффера й Пірса.
8 Описати мовою AHDL мажоритарність.
9 Описати мовою AHDL 3х вхідну непарність.
10 Описати мовою AHDL шифратор з 5-ти розрядного позиційного коду в бінарний.
11 Описати мовою AHDL дешифратор з 3х розрядного бінарного коду в позиційний.
12 Описати мовою AHDL пріоритетний шифратор на 4 входу.
13 Описати мовою AHDL дешифратор з бінарного в код 7-мі сегментного індикатору.
14 Описати мовою AHDL перетворювач із 4-х розрядного бінарного у двоїчно-десятковий.
15 Описати мовою AHDL 6 ти вхідний мультиплексор.
16 Описати мовою AHDL 5-ти вхідний демультиплексор.
17 Описати мовою AHDL 4 розрядний суматор.
18 Архітектура ПЛИС MAX 3000A.

19 Архітектура сімейства Apex 10k.
20 Архітектура сімейства Flex.
21 Архітектура системі-на-кристалі (SOC).
22 Історія розвитку архитектур ПЛИС.
23 Способи опису комбінаційних пристроїв мовою AHDL.

## 2.3 Лабораторна робота №4
## Розробка послідовних автоматів мовою описів пристроїв AHDL

### 2.3.1 Порядок виконання роботи

Опис послідовних автоматів мовою AHDL можна робити за допомогою:

- примітивів тригерів та регістрів;
- особливих конструкцій – машини становищ.

Приклад реалізації паралельного 8-розрядного регістра за допомогою примітивів наведено нижче.

```
SUBDESIGN reg_out
(
        clk, load, d[7..0]          : INPUT;
        q[7..0]                             : OUTPUT;
)
VARIABLE
        q[7..0]                             : DFFE;             % also declared
as outputs %
BEGIN
        q[].clk = clk;
        q[].ena = load;
        q[] = d[];
END;
```

Приклад реалізації лічильника з 4-ми виходами для керування кроковим двигуном за допомогою машини становищ наведено нижче.

```
SUBDESIGN stepper
(
        clk, reset      : INPUT;
        ccw, cw         : INPUT;
        phase[3..0]     : OUTPUT;
)
VARIABLE
        ss: MACHINE OF BITS (phase[3..0])
                WITH STATES  (
                        s0 = B"0001",
                        s1 = B"0010",
                        s2 = B"0100",
                        s3 = B"1000");
BEGIN
        ss.clk   = clk;
        ss.reset = reset;

        TABLE
                ss,     ccw,    cw      =>      ss;

                s0,     1,      x       =>      s3;
                s0,     x,      1       =>      s1;
                s1,     1,      x       =>      s0;
                s1,     x,      1       =>      s2;
                s2,     1,      x       =>      s1;
                s2,     x,      1       =>      s3;
                s3,     1,      x       =>      s2;
                s3,     x,      1       =>      s0;
        END TABLE;
END;
```

Роботу з проектування слід виконувати згідно з попередньою лабораторною роботою (підрозділ 2.3.1).

### 2.3.2 Лабораторне завдання

2.3.2.1 . Намалюйте УГП та розробіть за допомогою описів мовою AHDL, регістр згідно з варіантом по таблиці. 2.2.

Таблиця 2.2 – Варіанти завдань

| Параметри | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| Вид | Паралельний регістр | | | | | | | | | | | | | | | | | | | | Послідовний | | | | | | | | | |
| Вхід С | ф | ф | ф | ф | ф | с | с | с | с | с | п | п | п | п | п | і | і | і | і | і | ф | ф | ф | ф | ф | с | с | с | с | с |
| Входи R | п | і | п | і | - | п | і | п | і | - | п | і | п | і | - | п | і | п | і | - | п | і | п | і | - | п | і | п | і | - |
| Кількість інформації | 4 | 5 | 6 | 7 | 8 | 4 | 5 | 6 | 7 | 8 | 4 | 5 | 6 | 7 | 8 | 4 | 5 | 6 | 7 | 8 | 4 | 5 | 6 | 7 | 8 | 4 | 5 | 6 | 7 | 8 |

Примітка: тут R – вхід скидання; С – вхід синхронізації; “і” – інверсний, “п” – прямий; “-” – відсутній, “ф” – прямий динамічний, “с” – інверсний динамічний.

2.3.2.2 . За допомогою симулятора переконайтеся, що отримана схема виконує свої функції.

2.3.2.3. Намалюйте часові діаграми.

### 2.3.3 Вміст звіту

Звіт по виконаній роботі має містити у собі: титульну сторінку з назвою роботи; мету роботи; УГП регістру, опис мовою AHDL, сигнальні діаграми для перевірки роботи синтезованої цифрової схеми; допоміжна інформація; висновки.

### 2.3.4 Контрольні запитання

1. Використовуючи результати моделювання, опишіть роботу D-тригера з роздільним входом (dffe).
2. Використовуючи результати моделювання, опишіть роботу паралельного регістру.
3. Використовуючи результати моделювання, опишіть роботу зсувного регістру з паралельним завантаженням.
4. Використовуючи результати моделювання, опишіть роботу дешифратора.
5. Використовуючи результати моделювання, опишіть роботу двійкового лічильника.

6. Створіть проект трирозрядного двійкового лічильника на D-тригерах і опишіть його роботу з результатів моделювання.
7. Створіть проект чотирирозрядного двійкового лічильника на D-тригерах і опишіть його роботу з результатів моделювання.
8. Описати мовою AHDL D-тригер зі статичним керуванням.
9. Описати мовою AHDL D-Тригер з динамічним керуванням.
10. Описати мовою AHDL RSC-тригер зі статичним керуванням.
11. Описати мовою AHDL RSC-тригер з динамічним керуванням.

## ПЕРЕЛІК ЛІТЕРАТУРИ

1. Антонов А.П. Язык описания цифровых устройств AlteraHDL [Практический курс] / А.П. Антонов. – М.: ИП «Радиософт», 2001. – 224 с.

2. Соловьев В. В. Проектирование цифровых систем на основе программируемых логических интегральных схем / В.В. Слолвьев. – М.: Горячая линия-Телеком, 2001. – 636 с.

3. Стешенко В.Б. ПЛИС фирмы ALTERA: проектирование устройств обработки сигналов / В.Б. Стшенко. – М.: Додека, 2000. – 128 с.

4. ALTERA Altera Programming Hardware Data Sheet [Электронный ресурс]. – Режим доступу: https://www.altera.com/

en_US/pdfs/literature/ds/dspghd.pdf

5. ALTERA ACEX 1K Programmable Logic Family Data Sheet [Электронный ресурс]. – Режим доступу: https://www.altera.com/

en_US/pdfs/literature/ds/copy-of-acex.pdf, вільний.

6. ALTERA APEX 20K Programmable Logic Device Family Data Sheet [Электронный ресурс]. – Режим доступу: https://www.altera.com/

en_US/pdfs/literature/ds/apex.pdf , вільний.

7. ALTERA APEX 20KC Programmable Logic Device Family Data Sheet [Электронный ресурс]. – Режим доступу: https://www.altera.com/en_US/pdfs/literature/ds/ds_apex20kc.pdf , вільний.