Лабораторна робота №1

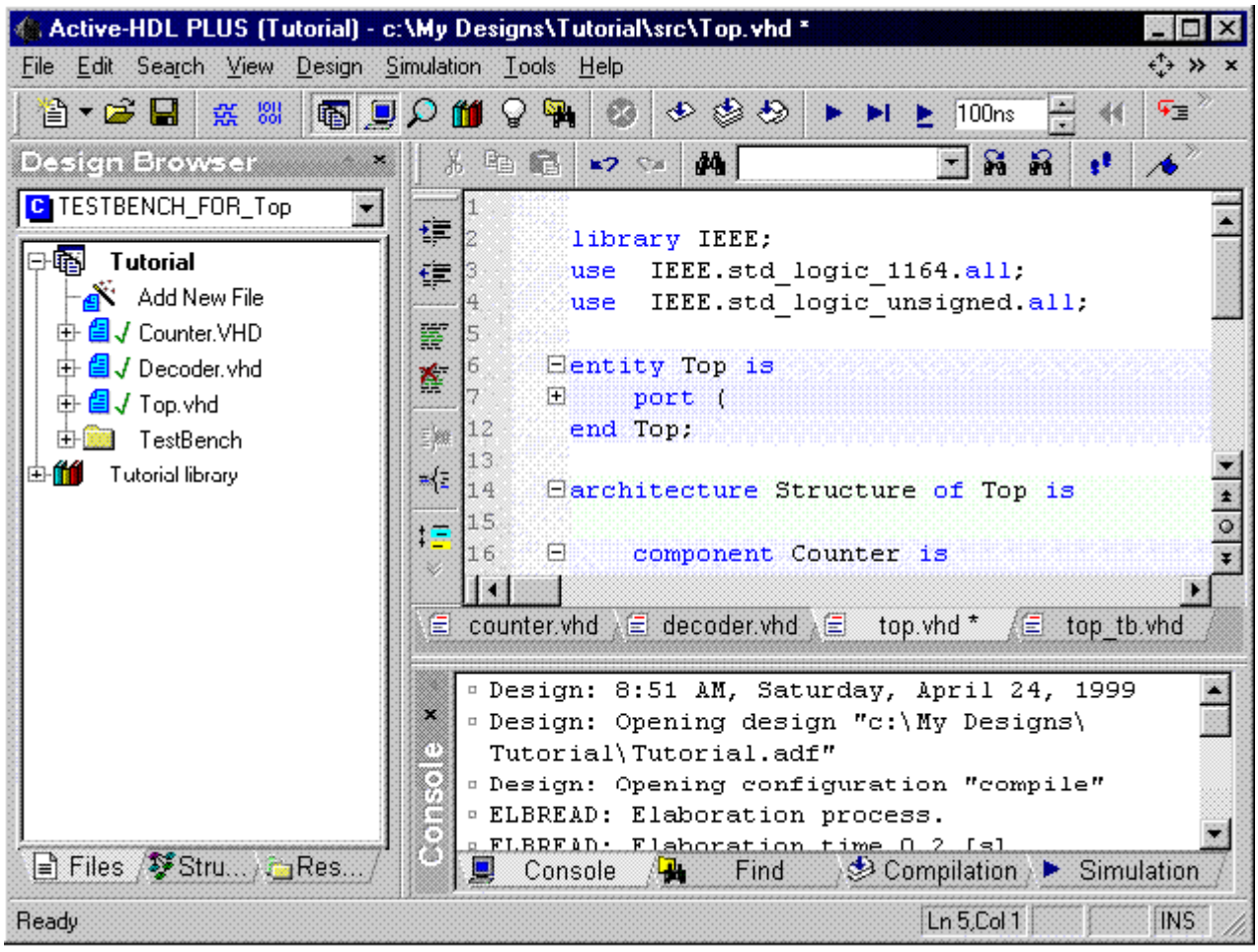
Вивчення інтегрованого середовища автоматизованого проектування Active-HDL фірми Aldec Inc

Мета роботи: ознайомитись з принципами автоматизованого проектування ПЛІС за допомогою пакету Active-HDL, вивчити структуру VHDL-проекту, навчитися працювати з засобами управління проектом.

Теоретична частина

Одним із світових лідерів в розробці систем автоматизованого проектування (САПР) для створення програмованих логічних інтегральних схем на базі мов описання апаратного забезпечення є корпорація Aldec Inc.

Структура Active-HDL базується на стандартному GUI-інтерфейсі. Загальний вигляд екрана при роботі з САПР Active-HDL:



Головними частинами Active-HDL є:

Design Browse

вікно перегляду проекту, призначене для для відображення інформації про складові елементи (компоненти) проекту:
- файли опису пристрою,

- використані в проекті бібліотеки,
- допоміжні файли проекту, до яких належать макроси та скрипт-файли, файли результатів симуляції роботи пристрою, допоміжні текстові файли,
- структуру проекту,
- сигнали та змінні, декларовані в проекті.

HDL Editor

редактор HDL-тексту з можливістю відображення заданих синтаксичних конструкцій мови різними кольорами; завдяки інтеграції редактора із симулятором компонент дозволяє виконувати зручне покрокове відлагодження пристрою і швидко виявляти помилки.

Console

вікно призначене для інтерактивного виводу текстової інформації, зокрема повідомленнь середовища; компонент також призначений для вводу команд середовища (Active-HDL commands).

Waveform Editor

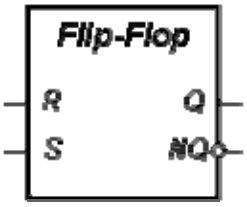
редактор, призначений для графічного відображення та редагування результатів симуляції - часових діаграм.

Language Assistant

компонент є зручним засобом, який дозволяє розробнику використовувати бібліотеку шаблонів опису стандартних примітивних конструкцій та функціональних блоків; Language Assistant дозволяє розміщувати вибрані шаблони безпосередньо в редагованому файлі та створювати свої власні шаблони.

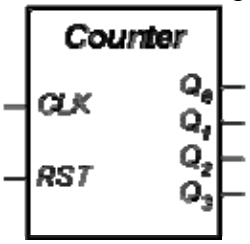
Порядок виконання роботи:

1. Запустити **Start -> Programs -> Active-HDL**. У діалоговому вікні **Getting Started** вибрати **Cancel**.
2. У вікні **Active-HDL** відкрити підручник по роботі з пакетом: **Help \ On-line Documentation \ HDL Entry and Simulation Tutorial**. Використовуючи цей підручник, ознайомитись з інтегрованим середовищем пакету, процесом створення проекту, редагування коду і його симуляції (отримання часових діаграм).
3. Відкрити проект Modulator з наборуі VHDL-прикладів (**File \ Open Design \ Samples \ Modulator**) та вивчити його склад і структуру за допомогою Design Browser. Скласти повний перелік об'єктів, що входять до складу проекту Modulator, описати їх інтерфейси.
4. Вивчити порядок застосування та функціональні можливості Майстра Нового Проекту (**New Design Wisard**). Створити за допомогою New Design Wisard порожній проект.
5. Описати на VHDL об'єкт, що являє собою RS-тригер. Інтерфейс цього об'єкта:
   2 вхідні порти *R* і *S* типу **std_logic**,
   2 вихідні порти *Q* та *NQ* типу **std_logic**:



6. Згенерувати такий же об'єкт в окремому файлі за допомогою Майстра (**Design Browser \ Add New File \ Wizards \ VHDL Source Code Wizard**). Порівняти об'єкт, створений вручну із згенерованим автоматично об'єктом.
7. Згенерувати за допомогою Майстра об'єкт, що реалізує 4-бітний лічильник. Його інтерфейс:
   2 вхідних порти *CLK* і *RST* типу **std_logic**,

один вихідний 4-розрядний порт *Q* типу **std_logic_vector**:



8. Підготувати звіт до захисту.

Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Опис засобів Active-HDL для управління проектами.
3. Опис структури VHDL-проекту.
4. Перелік об'єктів проекту Modulator та їх інтерфейси.
5. Склад та структура проекту, сформованого в результаті роботи.
6. Перелік об'єктів проекту, сформованого в результаті роботи, та їх інтерфейси.
7. Висновки.

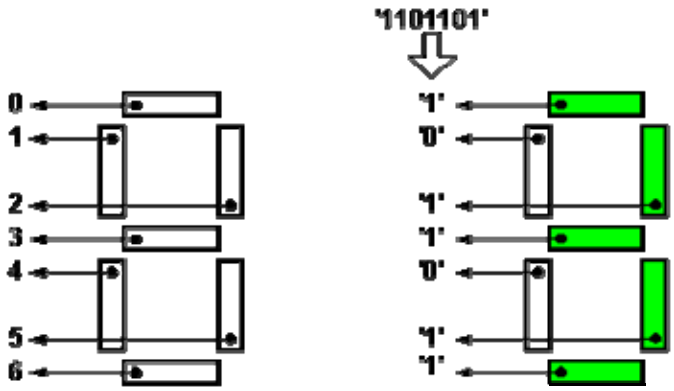
Лабораторна робота №2

Розробка та моделювання декодера для семисегментного індикатора

Мета роботи: вивчити методи опису поведінки об'єктів за допомогою архітектур та процесів, шляхи застосування оператора вибору **case**, навчитись моделювати поведінку об'єктів в САПР Active-HDL.

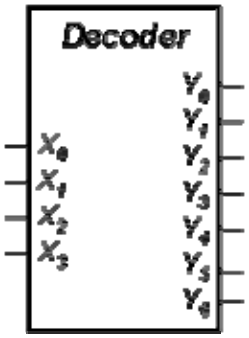
Теоретична частина

Для візуалізації процесів, що протікають в складному цифровому пристрої, необхідно виводити значення сигналів на 7-сегментний індикатор. Необхідно розробити на VHDL пристрій "декодер", що перетворює вхідний 4-розрядний двійково-десятковий код у сигнал для одного розряду 7-сегментного індикатора. Кожний розряд індикатора являє собою 7 сегментів, подання логічної одиниці на які викликає їх свічення. Наприклад, подання на індикатор числа "1101101" приведе до висвічення цифри "3":



Інтерфейс декодера:

- 4-розрядний вхідний порт *X* типу **std_logic_vector** (3 **downto** 0),
- вихідний 7-розрядний порт *Y* типу **std_logic_vector** (6 **downto** 0):



Якщо на вхід *X* пристрою об'єкта поданий код від 0 до 9, декодер сформувати на виході *Y* сигнал, який задає зображення цього числа. Для кодів 10..15 на виході формується сигнал "0000000" (жоден сегмент індикатора не світиться). Декодер повинен оновлювати сигнал *Y* кожний раз, коли змінюється значення вхідного порту *X*.

Рекомендації до створення VHDL-опису:

- Можливі значення вихідного сигналу задавати за допомогою констант.
- Для перетворення вхідного сигналу з типу **std_logic_vector** в тип **integer** використовувати функцію ***Conv_Integer***. Приклад її застосування:
  n := ***Conv_Integer***(*S*) , де *n* - змінна типу **integer**, *S* - змінна або сигнал типу **std_logic_vector**.
  Для того, щоби використовувати цю функцію, необхідно підключити відповідний пакет підпрограм, додавши рядок
  use IEEE.STD_LOGIC_UNSIGNED.all;
  на початку файлу опису.
- При застосуванні оператора **case** використати можливість вибору альтернативи для всіх можливих значень змінної або сигналу.

Порядок виконання роботи:

1. Створити новий проект в Active-HDL.
2. Створити об'єкт декодера та описати на VHDL його поведінку.
3. Скомпілювати створений об'єкт (меню **Design** \ **Compile**, або клавіша <F11>).
4. У вікні **Design Browser** для встановлення верхнього рівня моделювання (**Top Level**) вибрати об'єкт-декодер.
5. Ініціалізувати моделювання об'єкта (меню **Simulation** \ **Initialize Simulation**).
6. Створити в проекті новий файл **Waveform Viewer** (меню **File** \ **New** \ **Waveform**).
7. Вставити в вікно **Waveform Viewer** вхідний та вихідний сигнали декодера (меню **Waveform** \ **Add Signals**).
8. Призначити вхідному сигналу декодера стимулятори (меню **Waveform** \ **Stimulators**).
9. Запустити процес симуляції декодера (меню **Simulation** \ **Run**).
10. Вивчити отримані часові діаграми роботи декодера.
11. Змінюючи стимулятори на вхідних портах декодера, перевірити коректність його роботи для всіх можливих значень на вході.
12. Підготувати звіт до захисту.

Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Опис засобів Active-HDL для моделювання об'єктів.
3. Текст VHDL-опису декодера.
4. Часові діаграми роботи декодера.
5. Висновки.

Лабораторна робота №3

Моделювання інерційної та транспортної затримок часу

Мета роботи: Навчитися описувати інерційну та транспортну затримки часу за допомогою Active-HDL, з'ясувати принципові відмінності між ними.

Теоретична частина

Інерційна затримка є типовою для більшості реальних систем, в зв'язку з чим у VHDL ця модель використовується за замовчуванням. Оператор **after** автоматично вважає затримку інерційною, тому ключове слово **inertial** є необов'язковим. Характерною властивістю моделі цієї затримки є те, що дві послідовних зміни вхідного сигналу будуть проігноровані, якщо час між ними коротше, ніж задана затримка.

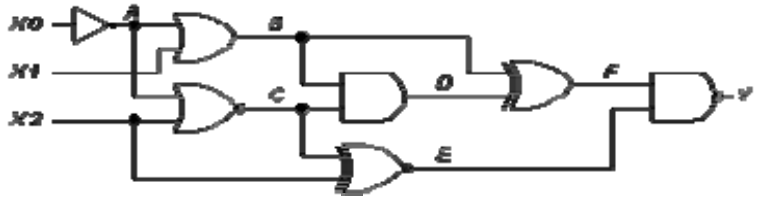
Для опису поведінки деяких пристроїв інерційна затримка непридатна. Прикладом може бути лінія передачі. Транспортування сигналів по лінії передачі відбувається без будь-яких змін, отже затримка в цьому випадку називається *транспортною*. Для того, щоби відрізнити її від інерційної затримки, прийнятої у VHDL за замовчуванням, використовується ключове слово **transport**, яке вказується перед описом значення затримки.

Моделі інерційної і транспортної затримки є достатніми для опису довільної фізичної системи. Вони мають наступні головні подібності та відмінності:

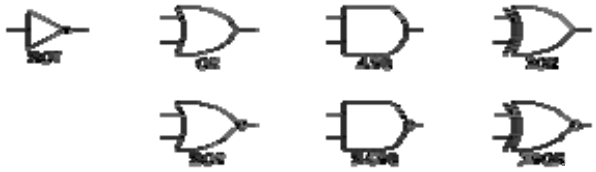
Інерційна затримка	*Транспортна затримка*
є затримкою за замовчуванням у VHDL і не вимагає ніяких додаткових декларацій	вимагає використання ключового слова **transport**
не поширює імпульси, коротші ніж задана затримка	поширює всі зміни вхідного сигналу, незалежно від того, як швидко і як часто вони відбуваються
описується за допомогою оператора **after** після якого вказується значення часу	
може застосовуватись до сигналів довільного типу	

Порядок виконання роботи:

1. Створити новий проект в Active-HDL.
2. Описати інтерфейс (**entity**) та архітектуру (**architecture**) для наступної схеми (вхідний порт - це 3-розрядний вектор *X*, вихідний порт - сигнал *Y*, сигнали *A, B, C, D, E, F* використовуються як проміжні):



де



3. Для всіх елементів схеми задати часові затримки: для інвертора 5 ns, для решти логічних елементів 10 ns.
4. Промоделювати роботу схеми (в якості стимуляторів використовувати *лічильники* (*Counters*) з відповідним періодом перерахунку)
   - при тривалості вхідних сигналів, більшої за інерційну затримку логічних елементів,
   - при тривалості вхідних сигналів, меншої за інерційну затримку логічних елементів.
5. Порівняти отримані часові діаграми.
6. Замінити інерційні затримки на транспортні.
7. Промоделювати роботу схеми
   - при тривалості вхідних сигналів, більшої за інерційну затримку логічних елементів,
   - при тривалості вхідних сигналів, меншої за інерційну затримку логічних елементів.
8. Порівняти отримані часові діаграми.
9. Підготувати звіт до захисту.

Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Визначення інерційної та транспортної затримок.
3. Принципова схема включення логічних елементів, що моделюється в лабораторній роботі.
4. Опис схеми у VHDL-коді.
5. Результати моделювання (пп. 4 і 7).
6. Результати порівняння отриманих часових діаграм при використанні інерційних та транспортних затримок.
7. Висновки.

Лабораторна робота №4

Моделювання режиму очікування в цифрових пристроях за допомогою оператора wait

Мета роботи: отримати практичні навички застосування оператора **wait** при моделюванні різних режимів очікування у Active-HDL. Навчитися виконувати порівняння часових діаграм.

Теоретична частина

Присвоєння нових значень сигналам в процесах відбувається тільки після припинення процесу. Для задання умов припинення процесу використовується оператор **wait**. Якщо процес містить оператор **wait**, він виконує послідовно всі оператори до оператора **wait**. Після цього процес припиняється і очікує виконання умови, заданої в операторі **wait**. Як тільки це відбувається, процес поновлюється і виконує всі оператори, поки знову не зустріне оператор **wait**.

Існує три типи оператора **wait**:

- **wait for** *time_expression* - очікування на протязі певного часу;
- **wait until** *condition* - очікування виконання логічного виразу;
- **wait on** *sensitivity_list* - очікування зміни значення сигнала із списку:

Якщо змішуються декілька таких умов, вони утворюють четверту конструкцію, яка називається складною умовою.

Тип оператора **wait**	Опис	Приклади
wait for *tyme_expression*	Припиняє процес на визначений час; Час може бути заданий безпосередньо, або як вираз, результатом якого є значення часу. Використовуються в основному для моделювання на випробувальних стендах.	**wait for** 10 ns; **wait for** ClkPeriod / 2;
wait until *condition*	Припиняє процес доти, доки задана умова не стане істинною завдяки зміні будь-якого сигналу, що містяться в умові. При цьому, якщо сигнал не змінюється, wait until не поновить процес, навіть якщо сигнал задовільняє умову.	**wait until** CLK='1'; **wait until** CE and (not RST); **wait until** IntData > 16;
wait on *sensitivity_list*	Припиняє процес доти, доки не відбудеться подія з будь-яким з сигналів, вказаних в списку чутливості. Іншими словами, процес поновлюється тільки тоді, коли зміниться значення хоча б одного сигналу з списку чутливості.	**wait on** CLK; **wait on** Enable, Data;
складний **wait**	Містить комбінацію двох або трьох різних форм оператора wait.	**wait on** Data **until** CLK='1'; **wait until** Clk='1' **for** 10 ns;

Якщо симулятор знаходить оператор **wait** відразу на початку процесу, то процес буде негайно припинений і жодний оператор не буде виконаний. Якщо ж оператор **wait** розташований в кінці процесу, спочатку будуть виконані всі оператори, що розташовані перед оператором **wait**, після чого процес буде припинений.

Порядок виконання роботи:

1. Створити новий проект в Active-HDL.
2. Створити об'єкт з наступним інтерфейсом:

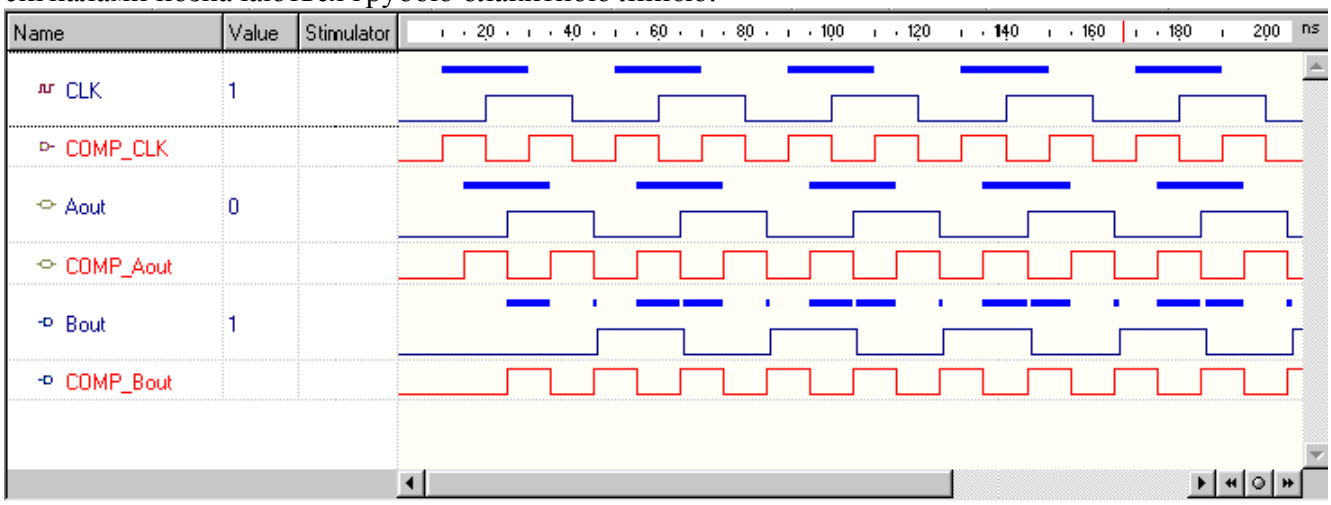
- вхідний порт *CLK* типу **std_logic**,
- вхідний-вихідний порт *Ainout* типу **std_logic**:
- вихідний порт *Bout* типу **std_logic**.

3. В тіло архітектури включити наступний код:

```
architecture comp of comp is
-- signal CLK : std_logic :='0';
begin
 Pr_CLK: process(CLK)
  begin
    -- формування сигналу CLK
 end process Pr_CLK;
 Pr_A: process(CLK)
  begin
   if CLK'event and CLK ='1' then Ainout<='1' after 5 ns;
    elsif CLK'event and CLK='0' then Ainout<='0' after 5 ns;
   end if;
 end process Pr_A;
 Pr_B: process (Ainout)
  begin
   if Ainout'event then Bout<= not Ainout;
   end if;
 end process Pr_B;
end comp;
```

23. Скомпілювати створений об'єкт і промоделювати його роботу. Отримати часові діаграми для сигналів *CLK* (задається як джерело з синхронізуючим імпульсом частотою 50 МГц), *Ainout*, *Bout*.
24. Зберегти результати моделювання як файл з ім'ям Wform1.awf у біжучій директорії.
25. За допомогою оператора **wait for** сформувати синхронізуючий сигнал *CLK* самостійно (попередньо виключивши його з інтерфейсу пристрою) в тілі процесу *Pr_CLK*. Рівень цього сигналу повинен змінюватись кожні 10 ns.
26. Вивчити роботу процесу *Pr_A* та переписати його з використанням оператора **wait on**.
27. Вивчити роботу процесу *Pr_B* та переписати його з використанням оператора **wait until**.
28. Промоделювати роботу створених за допомогою оператору **wait** процесів, отримати часову діаграму.
29. Порівняти поточну часову діаграму та базову Wform1, для чого
    - ввійти до меню **Waveform \ Compare Waveforms** - на екрані з'явиться вікно "Open";
    - вибрати файл Wform1.awf та натиснути кнопку "Open" - в результаті до поточної часової діаграми будуть додані часові залежності, які зберігаються у Wform1.awf, темно-синім кольором позначені поточні часові залежності, червоним - додані, розбіжності між порівнюваними

сигналами позначаються грубою блакитною лінією.



- якщо базова та біжуча діаграми співпадають, перейти до наступного пункту виконання роботи.

30. Вийти з режиму порівняння. Для цього необхідно вибрати у вікні часових діаграм будь-яку лінію розбіжності сигналів (грубу блакитну лінію), та натиснути праву кнопку миші, і у меню, що з'явилося, вибрати пункт "Remove Difference Marks".
31. Додати до послідовності операцій процесу *Pr_A* вираз **wait for** 30 ns.
32. Промоделювати роботу процесу, записати отриману часову діаграму, як файл Wform2.awf.
33. У виразі, який був доданий у п.12, змінити час затримки на 20 ns.
34. Промоделювати роботу процесу. Отриману часову діаграму порівняти з Wform2.awf.
35. Підготувати звіт до захисту.

Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Основні види застосування оператору **wait**.
3. Варіант базового коду та відповідні результати моделювання (п. 4).
4. Варіанти коду із застосуванням оператора **wait** (пп. 6, 7, 8).
5. Результати порівняння (п. 9).
6. Результати моделювання (п. 15).
7. Висновки.

Лабораторна робота №5

Розробка та моделювання паралельного регістру та регістру зсуву

Мета роботи: Засвоїти принципи роботи паралельного та зсувного регістрів. Навчитися описувати синхронізовані процеси та використовувати циклічні оператори при описі поведінки об'єктів за допомогою VHDL.

Теоретична частина

У VHDL існує два оператора цикла:

- оператор циклу з умовою **while** ... **loop** ...
- оператор циклу з лічильником **for** ... **loop** ... .

Оператор умовного циклу починається з логічної умови. Цикл повторюється до тих пір, поки виконується умова на початку циклу. Спочатку перевіряється умова, і, якщо вона істинна, виконуються оператори всередині циклу. Якщо ж умова хибна, цикл вважається закінченим і керування передається першому оператору, що знаходиться після циклу.

Як правило, такі цикли використовуються для повторення виконання групи операторів до тих пір, поки сигнал або змінна відповідають вибраному критерію (наприклад, дорівнюють деякому значенню).

Приклад використання умовного оператора циклу:

```
process
variable Count : integer := 0;
begin
  wait until CLK = '1';

  while Level = '1' loop
    Count := Count + 1;
    wait until CLK = '0';
  end loop;

end process;
```

Цикл з лічильником не містить явної булевої умови. Замість цього задається дискретний лічильник із діапазоном значень і цикл повторюється до тих пір, поки цей лічильник не вийде за межі діапазона. Після кожної ітерації циклу лічильнику присвоюється наступне значення із заданого діапазону. Лічильник, який не обов'язково декларувати (його специфікація в заголовку циклу прирівнюється до декларації) всередині циклу вважається константою і може використовуватись в присвоєннях, індексах виразів, але не може бути змінене. Більше того, лічильник існує тільки всередині циклу, в якому він задекларований.

Діапазон лічильника можна задавати не тільки в класичній формі вигляду **from** … **to** …. Він може бути також заданий як підтип або перечислимий тип. В такому випадку тільки (під)тип задається в якості діапазону лічильника.

Приклад використання оператора циклу з лічильником:

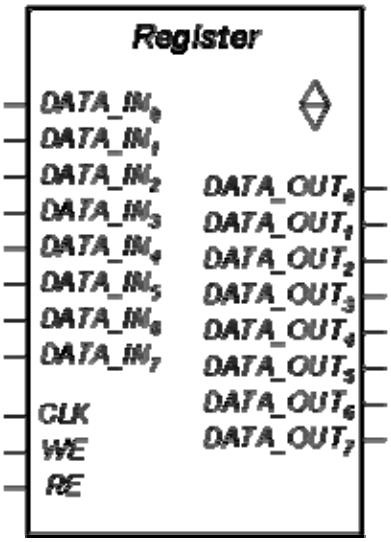
```
constant MSB : natural := 7;
signal SourceBus : std_logic_vector (MSB downto 0);

ReverseBus: process (SourceBus)
 variable ResultBus : std_logic_vector (MSB downto 0);
begin

  for Cntr in 1 to MSB loop
    ResultBus(MSB - Cntr) := SourceBus(Cntr);
  end loop;

end process ReverseBus;
```

Постановка задачі

Необхідно описати на VHDL роботу паралельного та зсувного регістрів, промоделювати їх та отримати часові діаграми.

Паралельний 8-розрядний регістр має:

- 8-розрядний вхід *DATA_IN* (7 **downto** 0) типу **std_logic_vector** для передачі даних,
- вхід синхронізації *CLK*, типу **std_logic**,
- вхід дозволу на запис *WE* типу **std_logic**,
- вхід дозволу зчитування *RE* типу **std_logic**.
- 8-розрядний вихід *DATA_OUT* (7 **downto** 0) типу **std_logic_vector** для виводу даних.

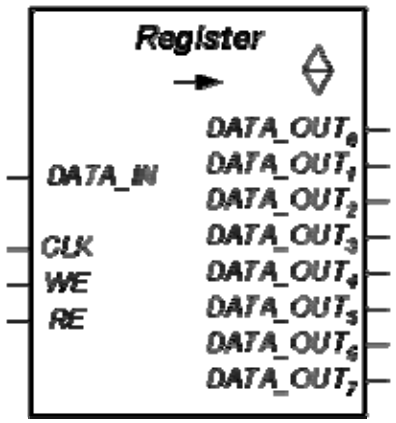


Робота регістра має здійснюватися наступним чином:

- у стані збереження байта на виході регістра постійно утримується високий імпеданс ("ZZZZZZZZ"), що дозволить організувати роботу декількох регістрів через одну шину, оскільки сигнал високого імпедансу має найнижчий пріоритет;
- якщо *WE* = '1' і *RE* = '0', то здійснюється запис інформації в регістр;
- якщо *WE* = '0' і *RE* = '1', то на вихід регістра подається значення байта, що зберігається в цьому регістрі;
- всі інші комбінації *WE* та *RE* розглядаються як стан збереження байта;
- робота регістра має бути синхронізована по сигналу *CLK*.

Зсувний 8-розрядний регістр має:

- один вхід *DATA_IN* типу **std_logic** для вводу інформації,
- вхід синхронізації *CLK* типу **std_logic**,
- вхід дозволу на запис *WE* типу **std_logic**,
- вхід дозволу зчитування *RE* типу **std_logic**,
- 8-розрядний вихід *DATA_OUT* (7 **downto** 0) типу **std_logic_vector** для паралельного виводу даних



Робота регістра має здійснюватися наступним чином:

- у стані збереження байта на виході регістра постійно утримується високий імпеданс ("ZZZZZZZZ");
- якщо *WE* = '1' і *RE* = '0', то здійснюється запис інформації в регістр, при цьому сигнал *DATA_IN* надходить в *DATA_OUT(0)*, значення *DATA_OUT(0)* переміщується в *DATA_OUT(1)* і т.д.;
- якщо *WE* = '0' і *RE* = '1', то на вихід регістра подається значення байта, що зберігається в цьому регістрі;
- всі інші комбінації *WE* та *RE* розглядаються як стан збереження байта;
- робота регістра має бути синхронізована по сигналу *CLK*.

Рекомендації до створення VHDL-опису:

- Для збереження проміжної інформації доцільно застосовувати змінні типу **std_logic_vector** (7 **downto** 0).
- Для забезпечення синхронізації в список чутливості процесів необхідно поміщати сигнал *CLK*.
- Зсув у зсувному регістрі слід реалізовувати за допомогою циклу з параметром (**for**).
- Більш ефективним є зсув від старшого (7-го) біта до молодшого (0-го).

Порядок виконання роботи:

1. Створити новий проект в Active-HDL.
2. Розробити VHDL-модель паралельного 8-розрядного регістру.
3. Промоделювати роботу розробленого паралельного регістра в режимах запису інформації, збереження байта та зчитування інформації.
4. Проаналізувати на основі отриманих часових діаграм відповідність роботи паралельного регістра заданому алгоритму.
5. Розробити VHDL-модель зсувного 8-розрядного регістру.

6. Помоделювати роботу розробленого зсувного регістра в режимах запису інформації, збереження байта та зчитування інформації.
7. Проаналізувати на основі отриманих часових діаграм відповідність роботи зсувного регістра заданому алгоритму.
8. Підготувати звіт до захисту.

Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Опис операторів циклу, що використовуються у VHDL.
3. Опис алгоритмів роботи розроблених регістрів.
4. Тексти VHDL-кодів, що описують поведінку паралельного та зсувного регістрів.
5. Часові діаграми роботи регістрів.
6. Висновки.

Лабораторна робота №6

Проектування постійного запам'ятовуючого пристрою

Мета роботи: Вивчити принципи роботи постійного запам'ятовуючого пристрою (ПЗП). Отримати навички застосування масивів при створенні проектів у Active-HDL.

Теоретична частина

В VHDL масив інтерпретується як тип, значення якого складається з ряду елементів єдиного підтипу. Кожен елемент масиву відрізняється своїм індексом або рядом індексів (в багатовимірних масивах). Індекс масиву повинен мати порядковий тип і знаходитись в області допустимих значень. Таким чином, перш ніж застосовувати масив у VHDL, необхідно задекларувати тип масиву, а потім задекларувати масив як об'єкт:

```
type array_type_name is array index_range of element_type;
variable array_name: array_type_name [:= initial_values];
```

Приклад одновимірного масиву:

```
type A is array (5 downto 0) of bit;
variable B : A := "011001";
```

Для багатовимірного масиву порядок і структура оголошення масиву залишаються така ж сама, наприклад

```
type A is array (1 to 3, 1 to 2) of integer;
variable B : A := ((0 , 1),(5 , 2),(7, 8));
```

У наведених прикладах замість ключового слова **variable** може стояти **signal** або **constant**.

При декларуванні типу масиву дозволяється не визначати його розмірність. Такі типи масивів називаються необмеженими. В цьому випадку розмірність вказується при декларуванні масиву, як об'єкта:

```
type arr_type is array (index_type range <>) of elements_type;
variable arr_name: arr_type (index_range) [:= initial_val];
```

де *index_type* - тип індексів масиву (**integer**, **natural**, і т.п.).

Багатовимірні необмежені масиви і їх типи оголошуються аналогічно. Наприклад:

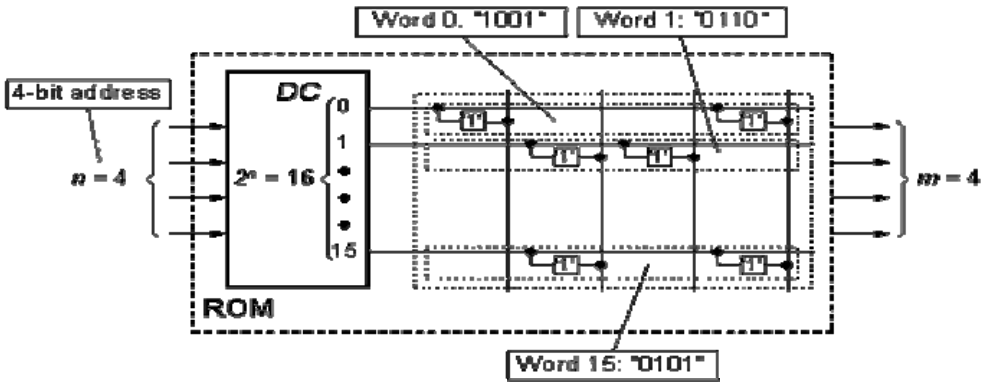
```
type multy is array (integer range <>, bit range <>, ... , boolean range <>) of
elements_type;
```

Постановка задачі

Запам'ятовуючі пристрої (ЗП) цифрової техніки призначені для запису, зберігання та видачі інформації, що представляється у вигляді цифрового коду. При цьому є і така інформація, яка не повинна змінюватись, наприклад, константи, табличні значення, коефіцієнти перетворень і т.ін. Така інформація записується у постійний запам'ятовуючий пристрій, для якого дозволяється тільки зчитування інформації, яка в нього занесена.

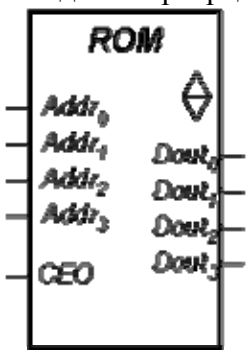
В ПЗП за кожною *n*-розрядною адресою записане одне завчасно визначене *m*-вимірне слово. Таким чином, ПЗП є перетворювачем коду адреси в код слова, тобто комбінаційною схемою з *n* входами та *m* виходами.

Накопичувач ПЗП апаратно виконується у вигляді системи взаємно-перпендикулярних шин, на перетині яких або присутній (логічна 1), або відсутній (логічний 0) елемент, що пов'язує між собою відповідні горизонтальну та вертикальну шини. Вибірка слів виконується за допомогою дешифратора.



Порядок виконання роботи:

1. Створити новий проект в Active-HDL.
2. Описати інтерфейс ПЗП, що має наступні порти:
   - вхідний 4-розрядний порт адреси ПЗП *Addr* типу **std_logic_vector**,
   - вхідний порт дозволо зчитування ПЗП *CEO* типу **std_logic**:
   - вихідний 4-розрядний порт даних *Dout* типу **std_logic_vector**.



3. Визначити тип масиву даних, які будуть зберігатись в ПЗП.
4. Сформувати масив даних ПЗП як масив констант.
5. Описати процес виборки даних з ПЗП.
6. Скомпілювати та промоделювати процес вибірки даних. Результати моделювання представити у вигляді часових діаграм та у табличній формі за допомогою **List Viewer**.
7. Підготувати звіт до захисту.

Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Функціональна схема ПЗП та його короткий опис.
3. Основні теоретичні відомості про використання масивів в Active-HDL
4. VHDL-код опису ПЗП.

5. Результати моделювання у вигляді часових діаграм.
6. Результати моделювання у табличній формі.
7. Висновки.

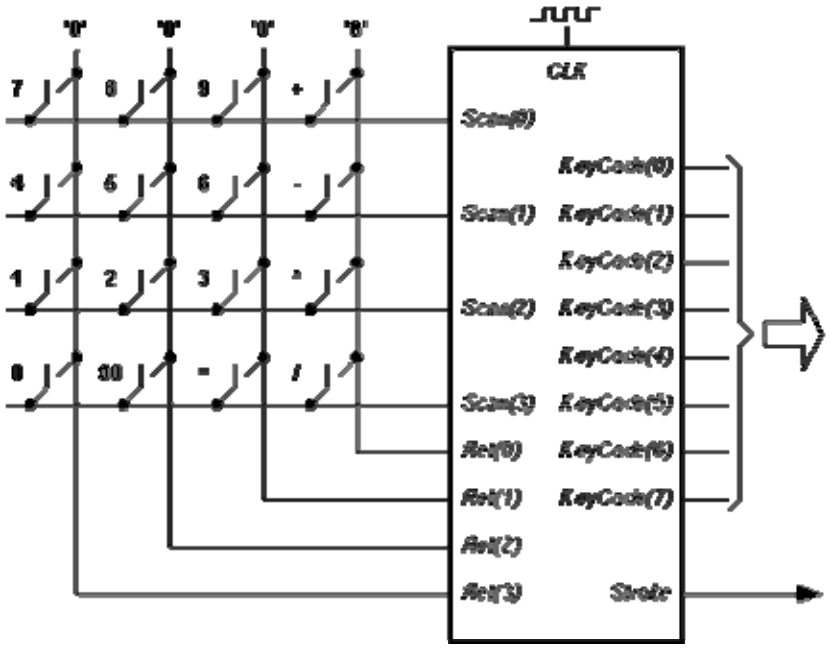
Лабораторна робота №7

Проектування сканеру клавіатури із застосуванням діаграм скінчених автоматів для опису об'єктів в САПР Active-HDL

Мета роботи: Вивчити принцип роботи сканера клавіатури. Отримати навички проектування цифрових пристроїв за допомогою скінчених автоматів засобами Active-HDL.

Теоретична частина

Клавіатура більшості обчислювальних пристроїв змонтована у вигляді прямокутної матриці, у точках перетину рядків і стовбців якої розташовуються кнопочні контакти. Скануючий пристрій, що розробляється, повинен визначати номер елемента матриці, який відповідає натисненій клавіші, і видавати відповідний йому двійковий код.



Сканер працює у двох головних режимах:

1. Режим очікування:
    1. На всі рядки матриці клавіатури, які позначені *Scan(0)*, *Scan(1)*, *Scan(2)*, *Scan(3)* подається логічна одиниця.
    2. Всі стовбці матриці клавіатури, позначені *Ret(0)*, *Ret(1)*, *Ret(2)*, *Ret(3)* мають низький рівень сигналу.
    3. Рівень сигналів Ret перевіряється з кожним синхронізуючим імпульсом *ClLK*, і, якщо один з них змінить своє значення з '0' на '1', це сигналізує про натиснення однієї з клавіш. При цьому сканер переходить в наступний режим.
2. Режим сканування:
    1. На рядок *Scan(0)* подається сигнал високого рівня, а на всі інші рядки - низького рівня.
    2. Перевіряється рівень сигналів *Ret*.
    3. Якщо рівень одного з *Ret* (наприклад, *Ret(1)*) зміниться з "низького" на "високий", це означає, що було замкнено контакт, який розташований на перетині рядка *Scan(0)* та

стовбця *Ret(1)*. Тобто номер елемента матриці клавіатури, якому відповідає натиснена клавіша, знайдено.

4. Сканер передає цю інформацію у кодер для формування відповідного коду обраної цифри або операції та знову переходить до режиму очікування.
5. Якщо рівень всіх рядків *Ret* залишається низьким, то проводиться перевірка наступного рядка матриці, і т.д.

Приклад роботи сканера у випадку натиснення клавіши '3':

№	Дії	Результати	Висновки
1.	*Scan* = "1111"	*Ret* = "0000"	Сканер знаходиться в стані очікування.
2.	*Scan* = "1111"	*Ret* = "0010"	Одержано інформацію про натиснення клавіши. Сканер переходить в режим сканування.
3.	*Scan* = "0001"	*Ret* = "0000"	Натиснена клавіша не знаходиться в першому рядку матриці.
4.	*Scan* = "0010"	*Ret* = "0000"	Натиснена клавіша не знаходиться в другому рядку матриці.
5.	*Scan* = "0100"	*Ret* = "0010"	Натиснена клавіша знаходиться на перетині третього рядка і другого стовбця матриці.
6.	*KeyCode* = *Scan* & *Ret*	*KeyCode* = "01000010" *Strobe* = '1'	Сканер переходить в режим очікування.

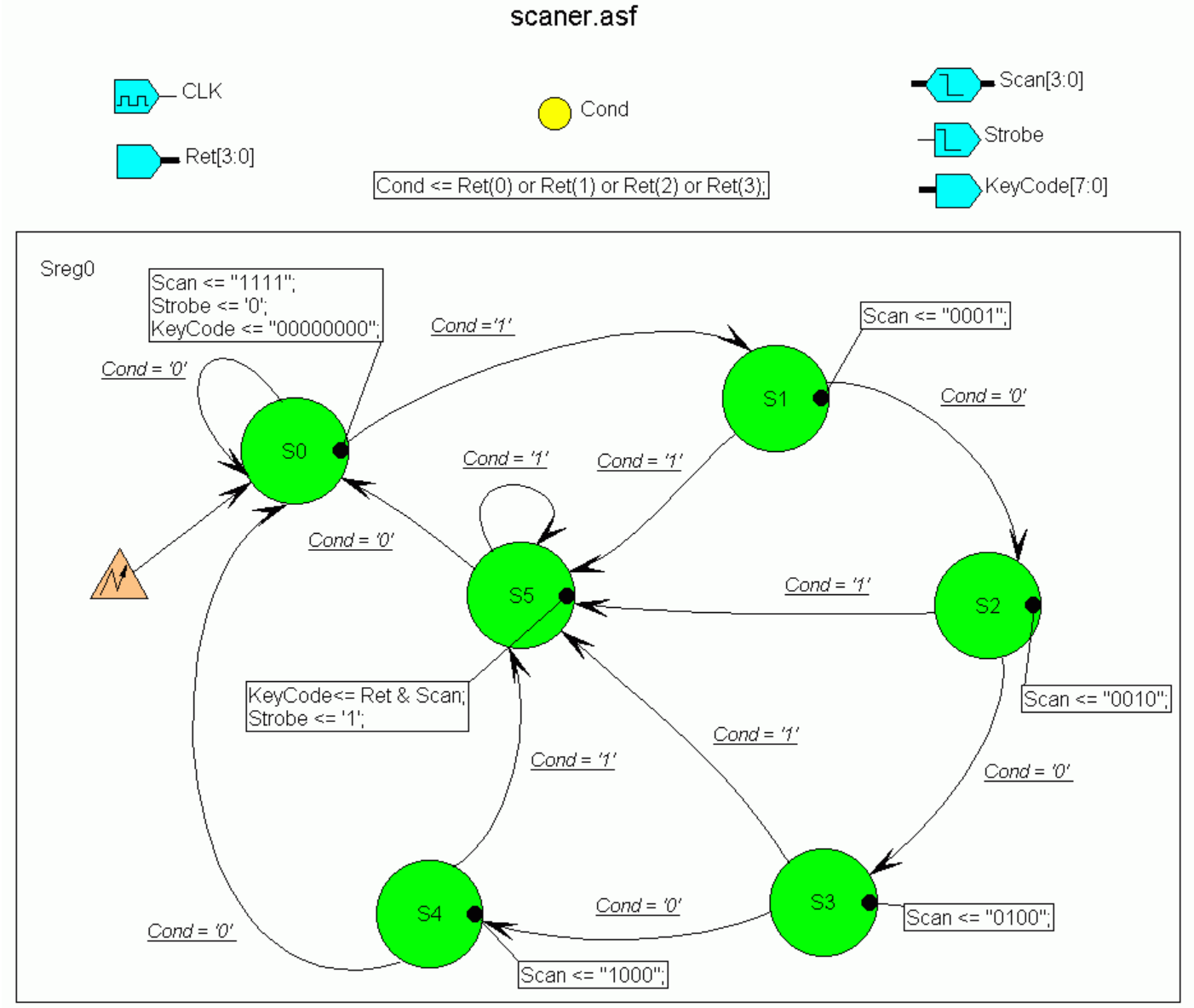
Порядок виконання роботи:

1. Створити новий порожній проект в Active-HDL.
2. Виходячи з основного принципу роботи сканера клавіатури, визначити основні стани пристрою і умови переходу від одного стану до іншого.
3. Визначити інтерфейс системи.
4. За допомогою **Add New File** і **State Diagram Wizard** створити макет діаграми скінчених автоматів.
5. Скласти діаграму скінчених автоматів.
    1. Використовуючи кнопку  робочої панелі **FSM State Editor**, створити в робочій області необхідну кількість кружків відповідно до кількості можливих станів сканера (S0, S1, ...), які були визначені в п.2.
    2. Використовуючи кнопку  робочої панелі **FSM State Editor**, відобразити за допомогою стрілок зв'язки між станами.
    3. Оскільки як в режимі очікування, так і в режимі сканування потрібно перевіряти, чи має хоча б один з сигналів *Ret* значення '1', то буде зручно ввести локальний сигнал *Cond* = (*Ret(0)* **or** *Ret(1)* **or** *Ret(2)* **or** *Ret(3)*) за допомогою кнопки  робочої панелі **FSM State Editor**.
    4. Використовуючи кнопку  робочої панелі **FSM State Editor**, визначити умови переходів між станами.
    5. Використовуючи кнопку  робочої панелі **FSM State Editor**, визначити дії, що виконуються в кожному стані скінченого автомата.
    6. Використовуючи кнопку  робочої панелі **FSM State Editor**, додати джерело діаграми, встановивши його на стан S0.

7. Ознайомитись з властивостями кожного об'єкту (параметрами, діями, умовами). Для відкриття відповідного діалогового вікна слід навести курсор на об'єкт, клацнути правою кнопкою миші і вибрати пункт "Properties" з локального меню.

6. Послідовним натисненням кнопок  і  сформувати і відобразити VHDL-код, що відповідає створеній діаграмі скінчених автоматів.
7. Вивчити відповідність графічних об'єктів діаграми і VHDL-коду.
8. Скомпілювати пристрій та промоделювати його роботу.
9. Підготувати звіт до захисту.

Діаграма скінченого автомата, створеного за допомогою Active-HDL для сканера клавіатури має вигляд:



Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Функціональна схема сканера клавіатури.
3. Основні стани сканеру клавіатури і умови переходу від одного стану до іншого.
4. Опис інтерфейсу системи.

5. Розроблена діаграма скінчених автоматів.
6. Відповідний VHDL-код для розробленої діаграми скінчених автоматів.
7. Результати моделювання в графічному та табличному вигляді та їх аналіз.
8. Висновки.

Лабораторна робота №8

Використання блок-діаграм для декомпозиції складних пристроїв в САПР Active-HDL

Мета роботи: Навчитися використовувати блок-діаграми (**Block Diagrams**) для декомпозиції складних об'єктів на структурні складові. Отримати навички застосування констант ***generic*** для проектування структурних елементів. Розробити модель 8-розрядного 7-сегментного індикатора.

Теоретична частина

Постановка задачі

Створити блок - генератор сигналів для 8-розрядного 7-сегментного індикатора. Декодер розряду індикатора та паралельний регістр (моделі яких було розроблено в лабораторних роботах №2 і №5) мають входити до проекту як структурні складові.

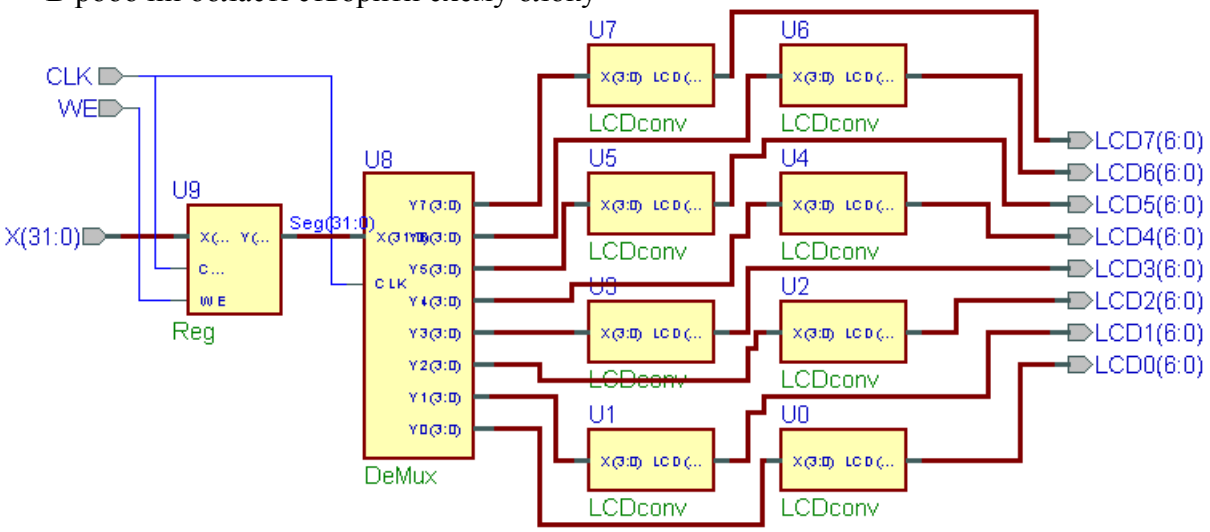
Інтерфейс генератора сигналів містить:

- вхідний 32-розрядний порт *X* типу **std_logic_vector** (31 **downto** 0), на який у двійково-десятковій формі подається 8-розрядне десяткове число (на кожний десятковий розряд по 4 двійкові розряди);
- вхідний порт *WE* типу **std_logic**, подання '1' на який дозволяє запис у проміжний регістр блоку;
- сигнал синхронізації *CLK* типу **std_logic**,
- вісім 7-розрядних вихідних портів *LCD7* … *LCD0* типу **std_logic_vector** (6 **downto** 0), що підключаються безпосередньо до відповідних розрядів 7-сегментного індикатора.

Порядок виконання роботи:

1. Створити новий порожній проект в Active-HDL.
2. Додати до проекту файл з описом декодера розряду 7-сегментного індикатора (розробленого в лабораторній роботі №2), для чого в менеджері проекту вибрати команду **Add New File**, і потім у діалоговому вікні - **Add Existing File** (додати існуючий файл), після чого вибрати необхідний файл.
3. Додати до проекту файл з описом паралельного регістра (розробленого в лабораторній роботі №5).
4. Введенням узагальнення ***generic*** до 8-розрядного регістру перетворити його на універсальний регістр.
5. Додати до проекту новий файл - блок-діаграму (**Add New File -> Block Diagram**).
6. Додати до діаграми вхідні та вихідні порти (за допомогою відповідної кнопки на панелі інструментів).

7. В робочій області створити схему блоку



8. Промоделювати роботу структурних складових та блоку в цілому (шляхом переключення **Top Level** на панелі Менеджера Проекту).
9. Проаналізувати отримані часові діаграми роботи розробленого пристрою та зробити висновки про його адекватність поставленому завданню.
10. Зконвертувати розроблену діаграму у VHDL-код, проаналізувати текст опису.
11. Підготувати звіт до захисту.

Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Опис методики застосування констант ***generic*** в VHDL.
3. Характеристика застосування блок-діаграм в САПР Active-HDL.
4. Схема блоку, створеною за допомогою блок-діаграм САПР Active-HDL.
5. Тексти VHDL-описів, що були розроблені в роботі.
6. Часові діаграми роботи структурних складових та блоку в цілому.
7. Висновки.

Лабораторна робота №9

Синтез та моделювання спеціалізованого пристрою

Мета роботи: Навчитися створювати завершені VHDL-проекти, використовувати тестові стенди для симуляції та верифікації їх роботи за допомогою пакету Active-HDL і проектувати завершені пристрої на базі ПЛІС фірми Xilinx, використовуючи пакет WebPACK ISE для синтезу готового VHDL-проекту.

Теоретична частина

1. Створення VHDL-моделі спеціалізованого пристрою

VHDL-код описує поведінку проектованої цифрової системи і являє собою звичайний текстовий файл. Виконання VHDL-опису проводиться за допомогою спеціальної програми - системи моделювання.

Система моделювання включає засоби, призначені для:

- *організації проекту* - визначення директорії проекту, розташування в ній необхідних файлів з вихідними VHDL-кодами, необхідними пакетами та бібліотеками VHDL-описів;
- *компіляції* - перетворення VHDL-кодів у внутрішнє представлення, яке і виконується (моделюється);
- *зборки* (лінкування);
- *моделювання* - виконання VHDL-кодів, представлених у внутрішній формі;
- *візуалізації* вихідних описів та результатів моделювання в різних формах - текстовій або графічній (часові діаграми).

Для моделювання цифрової системи необхідно створення спеціального тестового опису (теста) - оболонки, основне призначення якої:

- організувати подавання вхідних сигналів,
- отримати реакцію тестованої системи,
- порівняти, якщо потрібно, реакцію схеми з очікуваною.

Тестовий стенд - це середовище, в якому проект перевіряється за допомогою сигналів-стимуляторів з відображенням його реакцій.

Тестовий стенд складається з наступних елементів:

- сокет для тестованого пристрою (UUT - unit under test),
- генератор стимуляторів (підсистема, що застосовує стимулятори до UUT, генеруючи їх автономно, або читаючи із зовнішнього джерела),
- засоби відображення реакцій UUT на стимулятори.

Ідея тестових стендів адаптована до проектів у формі VHDL-специфікації. При цьому тестовий стенд не є самостійною системою, а тільки VHDL-специфікацією, що симулюється VHDL-симулятором. Він складається з реалізації тестованого пристрою (UUT) і процесів, що підтримують стимулятори, які застосовуються до UUT. При цьому створюється гібридна специфікація, в якій використовуються як структурні, так і поведінкові оператори.

Стимулятори для UUT описуються всередині архітектури тестового стенду, або можуть бути прочитані із зовнішнього файлу. Реакції UUT, з іншого боку, можуть спостерігатись як засобами симулятора у вигляді повідомлень симуляції (наприклад, часові діаграми, що спостерігаються на екрані), так і у вигляді файла, створеного операторами текстового вводу-виводу VHDL.

Тестовий стенд має характерні елементи:

- інтерфейс тестового стенду не має портів,
- реалізація компонента UUT - відповідність між тестовим стендом і UUT задається за допомогою реалізації компонента і структурної специфікації,
- стимулятори - це набір сигналів, що декларуються всередині архітектури тестового стенду і присвоюються портам UUT в його реалізації; стимулятори визначаються як часові діаграми в одному або більше поведінковому процесі.

2. Синтез VHDL-моделі спеціалізованого пристрою

Програмовані логічні інтегральні схеми (ПЛІС) дуже широко використовуються для створення цифрових систем різного призначення. Фірма Xilinx®, яка є провідним світовим виробником ПЛІС, надає розробникам широкий спектр кристалів з різною технологією виробництва, ступенем інтеграції, архітектурою, швидкодією, споживаною потужністю і напругою живлення, що випускаються в різних типах корпусів і в декількох варіантах виконання, включаючи промислове, військове і радіаційно-стійке.

Кристали, що випускаються фірмою Xilinx, повною мірою реалізують переваги ПЛІС у порівнянні з "жорсткою логікою":

- висока швидкодія;
- можливість перепрограмування безпосередньо в системі;
- високий ступінь інтеграції, що дозволяє розмістити цифровий пристрій в одному кристалі і тим самим знизити час і витрати на трасування і виробництво друкованих плат;
- скорочення часу циклу розробки і виробництва пристрою;
- наявність могутніх інструментів САПР, що дозволяють усунути можливі помилки в процесі проектування пристрою;
- порівняно низька вартість (у перерахуванні на один логічний вентиль);
- можливість наступної реалізації проектів ПЛІС для серійного виробництва у виді замовлених НВІС, що дозволяє значно знизити їх собівартість.

Етапи проектування цифрових пристроїв на базі ПЛІС Xilinx

У процесі створення цифрових пристроїв на базі ПЛІС Xilinx можна виділити наступні етапи:

- створення нового проекту (вибір сімейства і типу ПЛІС, а також засобів синтезу);
- підготовка опису проектованого пристрою в схемотехнічній, алгоритмічній або текстовій формі;
- синтез пристрою;
- функціональне моделювання;
- трасування проекту в кристал;
- часове моделювання;
- програмування ПЛІС (завантаження проекту в кристал).

Вихідна інформація про проектований пристрій може бути представлена у вигляді принципових схем, описів мовою HDL, діаграм станів і бібліотек користувача. У процесі синтезу на підставі

вихідних модулів проекту формується список кіл, що далі використовується в якості вихідних даних засобами трасування. Функціональне моделювання пристрою виконується без врахування реальних значень затримок проходження сигналів і дозволяє проконтролювати відповідність вихідних сигналів алгоритмам роботи проектованого пристрою. На етапі трасування проекту в кристал виконується розподіл виконуваних функцій у конфігуровані логічні блоки CLB (Configurable Logic Block) або макрокомірки Macrocell, в залежності від використовуваного сімейства ПЛІС, і формування необхідних зв'язків у кристалі.

В процесі трасування проекту в кристал також визначаються реальні значення затримок поширення сигналів, що необхідні для повного (часового) моделювання пристрою. Основним результатом етапу трасування є формування файлу, в якому міститься інформація про конфігурацію ПЛІС, що реалізує проектований пристрій. Завершенням процесу розробки цифрового пристрою є завантаження конфігураційних даних в кристал за допомогою відповідних програм і завантажувального кабелю.

Етапи функціонального і часового моделювання не є обов'язковими. Однак зневажати цими етапами не рекомендується, тому що високоефективні засоби моделювання пакетів САПР Xilinx дозволяють знайти більшість можливих помилок і тим самим значно скоротити загальний час розробки пристрою. При виявленні помилок на кожному з етапів (наприклад, логічних помилок на етапі функціонального моделювання або при одержанні незадовільних результатів часового моделювання) варто повернутися на стадію розробки вихідних описів проекту, внести необхідні зміни і повторити наступні етапи.

Основні характеристики пакету WebPACK ISE

Програмні засоби WebPACK ISE являють собою систему наскрізного проектування, що реалізує всі етапи створення цифрового пристрою на базі ПЛІС, включаючи програмування кристалу:

- розробка проекту,
- синтез,
- моделювання,
- трасування,
- завантаження в кристал.

САПР WebPACK ISE призначена для проектування цифрових пристроїв на базі ПЛІС виробництва Xilinx, що відносяться як до сімейств CPLD: XC9500, XC9500XL, XC9500XV, CoolRunner2, CoolRunner XPLA3, так і FPGA: Spartan™-II, Spartan™-IIE, Virtex™-E, Virtex-II та більш новіших.

Відмінні риси пакету:

- підтримка різних методів опису проектованих пристроїв (графічних і текстових);
- можливість використання проектів, підготовлених в інших системах проектування, у тому числі в середовищі пакета Altera MAX+PlusII™;
- наявність схемотехнічного редактора, укомплектованого набором великих бібліотек;
- інтелектуальні засоби створення HDL-описів, що формують шаблони на підставі інформації, наданої користувачем, для мов опису апаратури VHDL, Verilog і ABEL HDL;
- високоефективні засоби синтезу HDL-проектів, що підтримують мови VHDL, Verilog і ABEL HDL, з можливістю оптимізації;
- розвинуті засоби верифікації проекту, що дозволяють скоротити повний час розробки пристрою за рахунок виявлення можливих помилок на більш ранніх стадіях проектування і скорочення тривалості і кількості можливих ітерацій;

- автоматичні засоби трасування проекту в кристали різних сімейств ПЛІС Xilinx з врахуванням оптимізації проекту по різних параметрах;
- засоби програмування кристалів сімейств ПЛІС Xilinx, виконаних за різною технологією (CPLD і FPGA), що підтримують кілька типів завантажувальних кабелів JTAG-інтерфейсу;
- зручний для розробника користувацький інтерфейс і наявність у кожному модулі пакету довідкової системи, що скорочують час освоєння САПР;
- наявність інтегрованого з пакетом САПР набору інструментів і утиліт інших фірм, які надають додаткові зручності в процесі проектування, що включає утиліту генерації тестових сигналів HDL Bencher™, програму моделювання ModelSim XE Starter™ і редактор діаграм станів StateCAD™.

Користувацький інтерфейс пакета WebPACK ISE

Керуюча оболонка пакета WebPACK ISE Навігатор проекту (Project Navigator™) надає користувачеві зручний інтерфейс для роботи з проектом і керування всіма процесами проектування і програмування ПЛІС. Запуск усіх необхідних програмних модулів пакета здійснюється безпосередньо в середовищі Навігатора проекту. Основне вікно Навігатора проекту крім стандартних елементів керування (основного меню і оперативної панелі керування) містить чотири вбудованих вікна:

- вікно вихідних модулів (файлів) проекту (Sources in Project);
- вікно необхідних процедур (процесів) для обраного вихідного модуля проекту (Processes for Current Source);
- вікно консольних повідомлень програмних модулів (Console);
- вікно редактора текстових HDL-описів проекту.

У вікні вихідних модулів (файлів) проекту відображається ієрархічна структура, що складається з модулів (файлів), в яких міститься опис проектованого пристрою в графічній або текстовій формі, а також опис тестових впливів, що використовуються в процесі моделювання. Кожний тип модуля має відповідне графічне позначення - піктограму.

Вікно необхідних процедур (процесів) показує маршрут обробки виділеного вихідного модуля в процесі проектування пристрою. Таким чином, у даному вікні докладно відображаються всі етапи процесу розробки і програмування ПЛІС, що робить останній "прозорим" для користувача САПР. Послідовність і зміст етапів визначається типом вихідного модуля і сімейством ПЛІС. Навігатор проекту автоматично показує у вікні процедур (процесів) структуру процесу проектування, що відповідає обраному сімейству ПЛІС, виключаючи тим самим можливі помилки в послідовності дій розробника. У цьому ж вікні вказується інформація про додаткові інструменти, що можуть бути використані на кожному етапі.

Вікно консольних повідомлень призначене для виводу інформації програмних модулів пакету, що працюють в консольному режимі. Ряд програмних модулів пакета WebPACK ISE, як, наприклад, програми трансляції, синтезу, автоматичного трасування, є консольними аплікаціями, тобто не створюють власних вікон. Для того, щоб інформація про хід виконання цих програм була доступна розробникові безпосередньо в процесі роботи з проектом, вона відображається у вікні консольних повідомлень Навігатора проекту.

Вікно інтегрованого текстового редактора стає активним, якщо для проектованого пристрою або використовуваних бібліотек обраний спосіб опису мовою HDL.

Структура проекту в САПР WebPACK ISE

Проектом у САПР WebPACK ISE називається сукупність модулів (файлів), що містять інформацію, необхідну для виконання всіх етапів процесу розробки цифрового пристрою на базі ПЛІС Xilinx. У його структурі можна виділити наступні групи модулів:

- вихідні описи проектованого пристрою в графічній або текстовій формі;
- документація, що супроводжує проект;
- проміжні результати, які використовуються в якості вихідних даних для наступних кроків проектування;
- звіти про виконання основних етапів проектування;
- описи тестових впливів, необхідних для моделювання пристрою;
- остаточні результати проектування, які використовуються для конфігурування ПЛІС.

Всі модулі проекту розташовуються в одній директорії, назва якої збігається з назвою проекту. Відпочатку проект представлений тільки заголовком і модулем, у якому вказуються параметри проекту. Пізніше до нього додають модулі опису проектованого пристрою, а після виконання кожного етапу процесу розробки пристрою - результати, отримані на цьому етапі, і відповідний звіт. Крім того, розробник може включити в проект необхідну текстову документацію.

Коротка методика роботи з проектом в середовищі пакету WebPACK ISE

Створення нового проекту ініціюється командою FILE/New Project основного меню. При виборі цього пункту меню відкривається діалогова панель, у якій розробник повинен вказати ім'я і розташування проекту на диску, а також вибрати сімейство ПЛІС, тип кристала і засіб синтезу пристрою. Після введення зазначених даних у вікні вихідних модулів проекту з'явиться піктограма основного модуля з вказанням типу кристала й інструментів синтезу.

Для вводу опису проекту варто вибрати пункт New Source основного меню або натиснути відповідну кнопку на панелі інструментів. Далі відкривається список можливих типів вихідних модулів: схемотехнічне представлення, опис модулів, бібліотек і тестових впливів мовою HDL, діаграма станів, модулі документації. Розробник повинен вибрати тип нового вихідного модуля і вказати ім'я файлу для його наступного збереження. У залежності від типу створюваного вихідного модуля відкривається вікно схемотехнічного редактори Shematic Editor, редактора діаграм станів StateCad, генератора тестів HDL Bencher або активізується вікно текстового редактора.

У випадку успішного завершення створення вихідного модуля він автоматично додається до проекту і відображається у вікні вихідних модулів у вигляді відповідної піктограми. Розробник повинен по черзі створити всі необхідні модулі опису пристрою, після чого перейти до наступного етапу проектування.

При виборі способу опису проектованого пристрою рекомендується використовувати мову опису VHDL як найбільш ефективний і перспективний метод. Для розробників, що використовують САПР інших фірм, надана можливість вводу вихідних даних проекту у вигляді списку з'єднань Netlist.

Виділивши підготовлений модуль опису проекту у вікні вихідних модулів, розробник одержує у вікні процесів поетапну структуру наступних процедур проектування. Якщо перед назвою етапу вказаний знак ⊞, то цей етап містить у собі кілька процедур. Щоби побачити структуру такого етапу в розгорнутому вигляді, користувач повинен помістити курсор миші на зображення значка ⊞ і клацнути лівою кнопкою. Перш ніж перейти безпосередньо до етапу синтезу, розробник може скористатися при необхідності Утилітами вводу проекту (Design Entry Utilities), наприклад, Редактором часових і топологічних обмежень (Constraints Editor), інструментами підготовки тестів

для моделювання HDL-проектів (HDL Bencher Tools). Перед активізацією того або іншого процесу варто вказати його параметри. Для цього необхідно виділити рядок з назвою процесу і вибрати в основному меню команду редагування властивостей процесу Process/Properties або натиснути відповідну кнопку на панелі інструментів. На екран виводиться діалогова панель, що містить список параметрів, доступних користувачеві. Опції процесів можуть бути розбиті на групи, що представлені на окремих вкладках діалогової панелі. Зміст списку параметрів визначається виконуваним процесом і сімейством ПЛІС, на базі якого реалізується проект. Розробник може залишити значення параметрів, пропоновані за замовчуванням, або при необхідності встановити необхідні значення.

Щоби виконати процедуру (активізувати процес), слід вибрати команду Process/Run основного меню або просто двічі клацнути лівою кнопкою миші на назві відповідної процедури у вікні процесів. У випадку успішного виконання процедури у вікні консольних повідомлень після назви процесу відображається рядок:

```
Done: completed successfully.
```

Крім того, у вікні процесів перед назвою виконаної процедури з'являється піктограма у вигляді символу  , щовідповідає успішному завершенню процесу. При виявленні помилок у вікні консольних повідомлень виводиться рядок із інформацією про код помилки, модуль і рядок в модулі.

У випадку, якщо процедура виконана без помилок, але є попередження; вона позначається піктограмою у вигляді символу  .

Після повідомлень про помилки відображається рядок, що вказує на невдале завершення процесу, і відповідний код:

```
Done: failed with exit code: 0001.
```

У вікні процесів невдале завершення процедури позначається піктограмою у вигляді символу  .

Крім консольних повідомлень і піктограм у вікні процесів після виконання процедур на кожному етапі створюється звіт (Report), який містить докладну інформацію про хід і результати виконання процесу. Рекомендується аналізувати звіти для кожного етапу проектування не тільки у випадку виявлення помилок, але і при успішному виконанні процедур.

Етапи синтезу і трасування виконуються в пакеті WebPACK ISE автоматично. Розробнику необхідно тільки визначити параметри цих процесів. Моделювання пристрою здійснюється в середовищі програми ModelSim XE Starter з використанням тестів, сформованих за допомогою програми HDL Bencher. Керування процесом моделювання може здійснюватися за допомогою як елементів керування ModelSim XE Starter (основного меню і кнопок швидкого доступу), так і командного файлу, підготовленого раніше. Після успішного завершення етапу часового моделювання можна приступати безпосередньо до програмування кристалу.

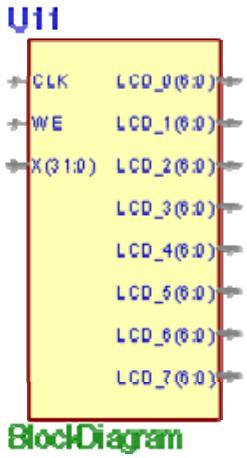
Постановка задачі

В лабораторній роботі проводиться

1. Проектування, симуляція та верифікація спеціалізованого пристрою.
2. Синтез, функціональне моделювання, трасування, та часова симуляція спеціалізованого пристрою для реалізації його в ПЛІС фірми Xilinx.

1. Створення VHDL-моделі спеціалізованого пристрою

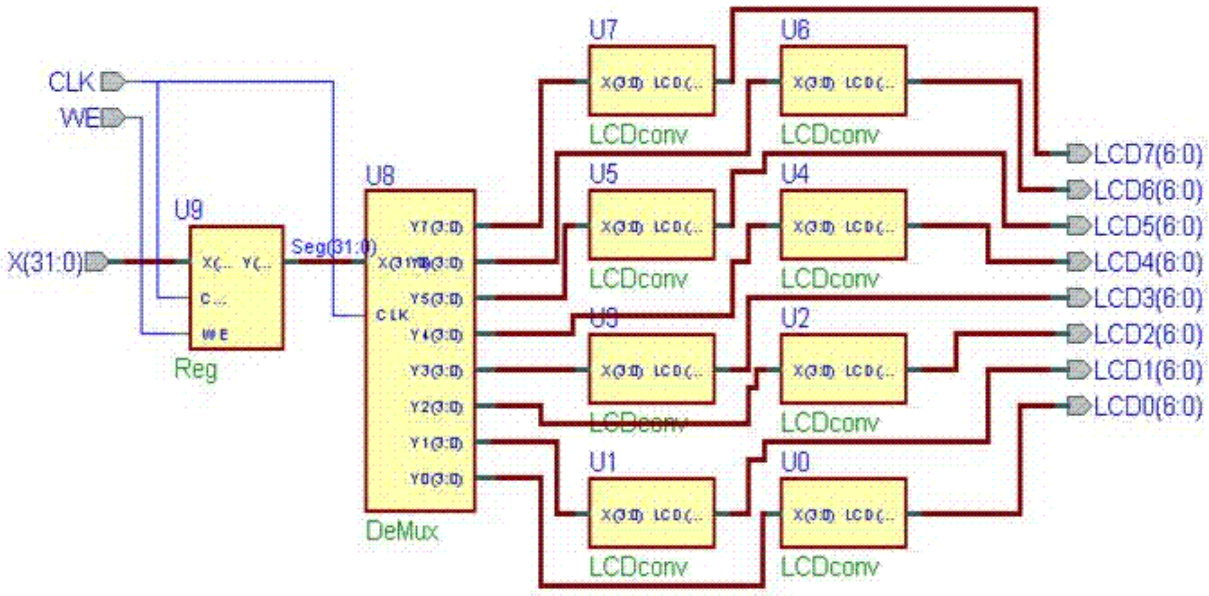
Спеціалізований пристрій (СП) має наступний інтерфейс:



Призначення портів СП:

- *CLK* - вхід тактової частоти,
- *WE* – дозвіл запису,
- *X(0..31)* – вхідна шина даних,
- *LCD0(0..6) – LCD7(0..7)* – вихідні шина даних.

Принцип дії проектованого СП повинен відповідати наступній функціональній схемі:



Алгоритм роботи СП:

1. Вхід *CLK* використовується для подавання тактової частоти, за якою здійснюється синхронізація роботи всіх внутрішніх блоків СП.
2. Вхід *WE* дозволяє запис по шині даних *X(0..31)* :

при WE=1 за додатнім фронтом *CLK* відбувається запис вхідного аргументу в регістр Reg;
при WE=0 за додатнім фронтом *CLK* відбувається читання аргументу з регістру Reg.

3. Після запису нового аргументу в СП, за додатнім фронтом *CLK* відбувається розподіл вхідного аргументу між виходами *Y0(0..3) – Y7(0..3);*

2. Синтез VHDL-моделі спеціалізованого пристрою

Реалізація проекту в ПЛІС - це складний ітераційний процес, що складається з послідовних етапів. У випадку виникнення помилок на деякому етапі необхідно повернутись до попередніх ітерацій для їх виправлення. Лише успішне завершення всіх етапів проектування приводить до отримання закінченої реалізації проекту в ПЛІС.

Весь цикл проектування виконується в пакеті WebPACK ISE. Для ознайомлення з методикою використання пакету WebPACK ISE можна скористатись [1], де на прикладі простого проекту докладно розглянуто всі етапи проектування.

Реалізація СП в рамках даної роботи виконується в ПЛІС FPGA сімейства Spartan-II. Ознайомитись із загальною інформацією про це сімейство ПЛІС можна в [2], з його функціональним описом - в [3], а з електричними та часовими характеристиками - в [4]. В [5] описано призначення виводів кристалів різних типів, що відносяться до цього сімейства, в тому числі виводів живлення, керування, інтерфейсу із конфігураційним ПЗП, інтерфейсу JTAG та виводів вводу-виводу.

Для ПЛІС FPGA фірми Xilinx необхідно забезпечити конфігурування кристалу, тобто завантаження бітової послідовності, отриманої за допомогою програмного забезпечення проектування, у внутрішню конфігураційну пам'ять ПЛІС. Spartan-II може завантажуватись як побітно (ведучий/підпорядкований послідовні режими та JTAG), так і побайтно (підпорядкований паралельний режим). Конфігураційні дані при виключеному живленні повинні зберігатись у зовнішньому пристрої статичної пам'яті. В цій роботі використовуються послідовні ПЗП Xilinx серії XC1700. Вибір необхідного ПЗП можна здійснити за [6].

Порядок виконання роботи:

I. Створення VHDL-моделі спеціалізованого пристрою
   1. Спроектувати на мові VHDL спеціалізований обчислювальний пристрій у відповідності з наведеним вище алгоритмом.
      - VHDL-опис СП повинен складатись з одного інтерфейсу та з однієї поведінкової архітектури. Окремі вузли описуються за допомогою окремих процесів.
      - При проектуванні СП
        - не дозволяється використовувати сигнали інших типів, крім **std_logic** та **std_logic_vector**;
        - не дозволяється використовувати процедури і функції;
        - не рекомендується використовувати змінні;
        - не рекомендується використовувати оператор **wait**.
   2. Створити тестовий стенд для перевірки розробленого проекту, при цьому
      - тестовий стенд повинен бути створений вручну;
      - інтерфейс тестового стенду повинен мати ім'я "*testbench*";
      - СП в тестовий стенд повинен бути включений як компонент.
   3. Провести симулювання та верифікацію розробленого проекту за допомогою підготовленого тестового стенду.

II. Синтез VHDL-моделі спеціалізованого пристрою
1. Створити новий проект в пакеті WebPACK ISE:
   - задати ім'я проекту та директорію розміщення;
   - в якості цільового кристалу вибрати ПЛІС Xilinx сімейства Spartan-II (Spartan2) *xc2s15-5vq100*;
   - вибрати інструменти синтезу XST VHDL.
2. За допомогою команди меню *Project -> Add source...* додати до проекту
   - готовий VHDL-опис СП як *VHDL Module*;
   - опис тестового стенду як *VHDL Test Bench*, при цьому
     - інтерфейс тестового стенду повинен мати ім'я "*testbench*";
     - СП в тестовий стенд повинен бути включений як компонент.
3. Провести функціональну симуляцію тестового стенду (*Process Window -> ModelSim Simulator -> Simulate Behavioral VHDL Model*) і перевірити коректний результат роботи СП.
4. Ознайомитись з механізмом призначення номерів виводів кристалу портам СП (*Process Window -> Design Entity Utility -> User Constraints -> Edit Implementation Constraints*).
5. Провести синтез проекту (*Process Window -> Synthesize*) і ознайомитись із звітом про синтез (*Synthesis Report*).
6. Провести трасування проекту (*Process Window -> Implement Design*) і ознайомитись із звітами про трасування (*Translation Report*, *Map Report*, *Place & Route Report* і*Pad Report*).
7. Провести часову симуляцію тестового стенду (*Process Window -> ModelSim Simulator -> Simulate Post-Place & Route VHDL Model*) і перевірити результат роботи схеми СП, реалізованої в кристалі ПЛІС.
8. Згенерувати бітову послідовність для конфігурування ПЛІС (*Process Window -> Generate Programming File*) і ознайомитись із звітом про генерацію бітової послідовності (*Programming File Generation Report*).
9. За [5] для використаного кристалу ПЛІС визначити виводи кіл живлення, кіл керування та кіл конфігурування.
10. За [6] для використаного кристалу ПЛІС вибрати послідовний ПЗП конфігураційної пам'яті.

Зміст звіту

1. Назва та мета виконання лабораторної роботи.
2. Опис функціонування розробленого проекту з відповідними VHDL-описами.
3. Вміст файлів згенерованого тестового стенду.
4. Часові діаграми симуляції та верифікації проекту за допомогою тестового стенду.
5. Скоректований VHDL-опис СП, отриманий в результаті реалізації проекту в ПЛІС.
6. Скоректований VHDL-опис тестового стенду для СП із зазначенням вхідних даних та очікуваного результату симуляції.
7. Часові діаграми функціональної симуляції проекту.
8. Часові діаграми часової симуляції проекту.
9. Висновки.