Національний технічний університет України

«Київський політехнічний інститут»

Факультет інформатики і обчислювальної техніки

Кафедра обчислювальної техніки

**Реферат**

**з теорії проектування комп’ютерних систем**

**на тему: «Побудова цифрових автоматів в середовищі розробки Active-HDL»**

Виконав*:*

студент ІV курсу

групи ІО-12

Бута С. О.

залікова знижка №1205

Київ, 2014 р.

**Зміст**

1. Цифрові автомати
2. Середовище розробки Active-HDL
3. Побудова тригерів
4. Побудова автомату
5. Список використаної літератури
6. **Цифрові автомати**

Цифровий автомат (ЦА) – це система обробки дискретної інформації, яка має вхід і вихід, що служать для подачі (зняття) вхідних (вихідних) сигналів (слів) представлених в паралельній чи послідовній формі.

Вхідні і вихідні слова можуть змінюватися тільки в визначені моменти часу, що називаються тактовими. Інтервал часу між двома сусідніми тактовими моментами називають тактом. Тривалість тактів визначається генератором тактових імпульсів. ЦА повинен мати здатність запам’ятовувати на час тривалості такту (тактів) інформацію (чи тільки факти її надходження), що поступає на його входи. Типовий ЦА з пам’яттю це електронний чи механічний годинник. За 1 хвилину (60 сек.) на такий ЦА поступає 60 вхідних слів при частоті генератора тактових імпульсів (маятника) 60 с-1. Для збереження інформації, наприклад тільки секунд (вихідних слів) достатньо 2 елементи пам’яті (одиниці та десятки секунд).

Цифрові автомати мають скінчене число внутрішніх станів. Вихідне слово Zi на виході ЦА в конкретний момент часу, рис.4.1 визначається вхідним словом в цей же момент часу Хі і внутрішнім станом автомата Yі, який є наслідком дії на автомат вхідних сигналів в попередні моменти часу. ЦА мають пам’ять, тому їх можна ще представити як сукупність комбінаційних схем і елементів пам’яті, рис.4.1,б

ЦА під дією вхідного слова переходить з одного стану в інший, зберігаючи прийнятий стан на протязі часу, що дорівнює не менше 1го такту. ЦА і будуються на основі комбінаційних схем і схем з пам’яттю. Вихідне слово Zt ЦА визначається вхідним словом Xt в кожному конкретному такті, чи часі ti.

Будь-який ЦА складається з пристрою запам’ятовування (ЗП), операційного пристрою (ОП) і пристрою керування (ПК).

ОП – пристрій, який за допомогою комбінаційних схем проводить операції над вхідними словами (додавання, множення, зсув, переміщення). Для роботи ОП необхідні керуючі сигнали, що змусять виконати ОП деяку дію за один такт. Такі керуючі сигнали називаються командою, а власне керуючий пристрій повинен забезпечити в відповідності вибраному алгоритму деякої послідовності команд, тобто програми.

ОП можуть працювати з схемною логікою роботи, якщо воно розраховано на виконання тільки одної програми, або по принципу програмного керування, при якому для зміни порядку роботи ОП достатньо записати і відтворити з носія інформації нову програму роботи. Програма роботи повинна зберігатись в запам’ятовуючому пристрої (ЗП), який може будуватись з послідовним доступом, або адресним доступом при якому вказується в ЗП:

1) адреса перетворюючого символу;

2) вказівка на елементарну операцію, яку треба здійснити над цими символами;

3) адресу ЗП куди треба помістити результат;

4) адресу комірки ЗП, де зберігається наступна команда програми.

Пристрій керування (ПК) при програмному керуванні забезпечує використання принципу мікропрограмного керування, коли програма розбивається на окремі програмки, що виконується мікроопераціями за один або декілька мікротактів.

1. **Середовище розробки Active-HDL**

Програмний продукт Active-HDL компанії Aldec Inc. є сучасною системою проектування і моделювання проектів з підтримкою всіх сімейств програмованих інтегральних схем ПЛІС. Дана система проектування забезпечує гнучкість підходу до розробки проекту і надає високофункціональну службу підтримки для проектування складних багаторівневих FPGA проектів. Середовище проектування Active- HDL включає в себе інтерфейси з усіма провідними програмними продуктами, надаючи розробникам свободу у використанні тих систем проектування, які найбільш повно відповідають вимогам для кожного конкретного проекту. Тісна інтеграція всіх додатків забезпечує повний контроль над проектом, починаючи від специфікації на всьому шляху проектування, закінчуючи його фізичної реалізацією.

Active-HDL структурно побудована навколо свого блоку дороговказу проектування, який дозволяє користувачам легко отримувати доступ до засобів проектування і використовуваним бібліотекам проекту. Active-HDL містить відкомпілювалися і готові до використання бібліотеки виробників ПЛІС. Блок управління проектом Active-HDL дозволяє розробнику виконувати всі модифікації та операції над проектом з єдиного середовища проектування. Дана інтеграція дає найкращий контроль над процесом розробки, мінімізує час і усуває необхідність у запуску безлічі віконних інтерфейсів і процесів.

Щоб запустити керуючу оболонку пакету, досить клацнути на піктограмі кнопка. При установці даного програмного продукту ця іконка автоматично поміщається на робочому столі вашого комп'ютера.

На стартовій панелі Getting Started (рис. 1) ви можете вибрати із запропонованого списку і відкрити існуюче робочий простір (Open existing workspace) або створити нове (Create new workspace). Ми почнемо знайомство з створення нового робочого простору, а тому встановимо перемикач в положення Create new workspace.

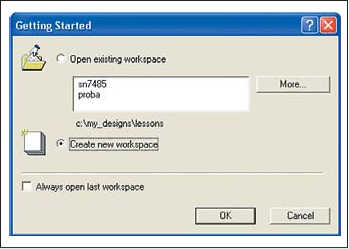


Рис. 1.

Поняття «робочий простір» є відносно новим, до нього ми ще повернемося, а поки задовольнимося простим поясненням. Це ще одна надбудова над сукупністю звичайних проектів, свого роду суперпроект або мультипроектного робоче середовище. Ви можете «тримати під рукою» відразу кілька проектів і мати швидкий доступ до їх сукупного ресурсу.

На другій панелі New Workspace (рис. 2) необхідно надрукувати ім'я робочого простору і вказати місце, де воно буде розташоване. За замовчуванням вам буде запропонована папка my\_designs, що знаходиться в корені того логічного диска, на якому встановлена система.



Рис. 2

Щоб не робити потім зайву роботу, переконайтеся, що прапорець Add New Design to Workspace.

На третьому панелі New Design Wizard (рис. 3) за замовчуванням пропонується режим Create an Empty Design with Design Flow («Створити порожній проект з використанням менеджера маршруту проектування»). Це найбільш розумний варіант, в чому ми пізніше переконаємося.



Рис. 3

На четвертой панели сохраняем все, как есть (установки по умолчанию), потому что синтез и реализация проекта не поддерживаются данным пакетом, если на вашем компьютере не установлены дополнительные программные средства. Во всяком случае, обсуждение этих вопросов не входит в планы первоначального знакомства с системой Active-HDL 7.1.

На пятой панели введем имя проекта, например Lesson\_1 (урок\_1). Обратите внимание, синхронно с вводом имени проекта заполняется и нижнее поле, где необходимо задать имя рабочей библиотеки. По умолчанию предполагается, что они будут иметь одинаковые имена, что представляется весьма разумным.

На последней панели мастера New Design Wizard вы должны проверить правильность введенных данных и, если нет нужды возвращаться к уже пройденным панелям, нажать на кнопку «Готово».

С этого момента собственно и начинается активная работа по созданию проекта. Она выполняется в интегрированной среде проектирования Active-HDL 7.1 (рис. 4). В строке заголовка видно это название, а правее в круглых скобках высвечивается имя загруженного рабочего пространства и проекта: Lessons, Lesson\_1. Далее следует название активной закладки — Design Flow Manager.

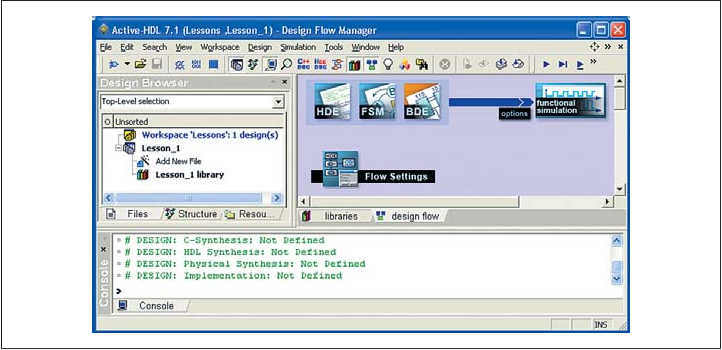


Рис. 4.

1. **Побудова тригерів**

Для прикладу будемо будувати JK тригер. Створимо однойменний файл в нашому проекті з розширенням .vhdl.

Далі опишемо роботу тригера:

-------------------------------------------------------------------------------

--

-- Title : JK

-- Design : TPCS

-- Author : Serhiy Buta

-- Company : KPI

--

-------------------------------------------------------------------------------

--

-- File : JK.vhd

-- Generated : Fri Dec 19 07:49:56 2014

-- From : interface description file

-- By : Itf2Vhdl ver. 1.22

--

-------------------------------------------------------------------------------

--

-- Description :

--

-------------------------------------------------------------------------------

--{{ Section below this comment is automatically maintained

-- and may be overwritten

--{entity {JK} architecture {JK}}

library IEEE;

use IEEE.STD\_LOGIC\_1164.all;

entity JK is

port(

J : in BIT;

C : in BIT;

K : in BIT;

Q : out BIT;

nQ : out BIT

);

end JK;

--}} End of automatically maintained section

architecture JK of JK is

begin

-- enter your statements here --

process (C, J, K)

variable state : BIT := '0';

begin

if(C'event and C = '1') then

if(J = '0' and K = '1') then

state := '0';

elsif ( J = '1' and K = '0' ) then

state := '1';

elsif ( J = '1' and K = '1' ) then

state := not(state);

end if;

end if;

Q <= state;

nQ <= not(state);

end process;

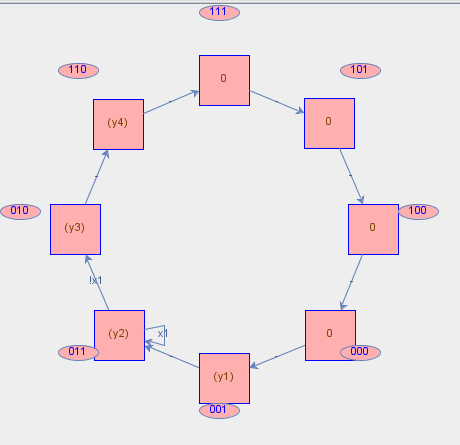
end JK;

Скомпілюємо. Переконаємося, що не виникло помилок. Це й усе. Тепер переконаємося, що він працює вірно. Для цього побудуємо довільний цифровий автомат.

1. **Цифрові автомати**

Для прикладу візьмемо наступний алгоритм:

B (y1) (y2) x1^2^4 (y3) (y4) E

Закодований автомат виглядає наступним чином:

Проведемо мінімізацію. Отримали наступні функції:

y1 = !(!( ( !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) v ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) )

y2 = !(!( ( !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) v ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) )

y3 = !(!( ( !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) v ( Qt2 Qt2 Qt2 ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) )

y4 = !(!( ( !( Qt0 v Qt0 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) v ( Qt0 Qt2 Qt2 ) ) !( ( !( Qt0 v Qt0 ) !( Qt0 v Qt0 ) !( Qt0 v Qt0 ) ) v ( !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) ) !( ( Qt2 Qt2 Qt2 ) v ( Qt2 Qt2 Qt2 ) ) )

J0 = !(!( ( !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) v ( Qt2 Qt2 Qt2 ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) )

K0 = !(!( ( !( Qt0 v Qt0 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) v ( Qt0 Qt2 Qt2 ) ) !( ( !( Qt0 v Qt0 ) !( Qt0 v Qt0 ) !( Qt0 v Qt0 ) ) v ( Qt2 Qt2 Qt2 ) ) !( ( Qt1 Qt1 Qt1 ) v ( Qt1 Qt1 Qt1 ) ) )

J1 = !(!( ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) v ( Qt1 Qt1 Qt1 ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) !( ( Qt0 Qt0 Qt0 ) v ( Qt0 Qt0 Qt0 ) ) )

K1 = !(!( ( !( Qt0 v Qt0 ) !( Qt0 v Qt0 ) !( Qt0 v Qt0 ) ) v ( !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) ) !( ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) v ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) ) !( ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) v ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) ) )

J2 = !(!( ( !( Qt0 v Qt0 ) Qt1 Qt1 ) v ( Qt0 Qt2 Qt2 ) ) !( ( Qt0 !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) v ( Qt2 Qt2 Qt2 ) ) !( ( Qt0 !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) v ( Qt2 Qt2 Qt2 ) ) )

K2 = !(!( ( ( !( Qt0 v Qt0 ) Qt1 Qt2 ) x1 x1 ) v ( !( Qt0 v Qt0 ) !( Qt1 v Qt1 ) !( Qt1 v Qt1 ) ) ) !( ( Qt0 Qt1 Qt1 ) v ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) ) !( ( Qt0 Qt1 Qt1 ) v ( !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) !( Qt2 v Qt2 ) ) ) )

Мінімізуємо їх:

y1 = !(!Qt1 v !Qt2 v Qt0 )

y2 = !()

y3 = !(!Qt1 v Qt2 v Qt0 )

y4 = !(!Qt0 !Qt2 v Qt0 Qt2 v !Qt0 v !Qt1 v Qt2 )

J0 = !(!Qt1 v Qt2 v Qt0 )

K0 = !(!Qt0 !Qt2 v Qt0 Qt2 v !Qt0 v Qt2 v Qt1 )

J1 = !(!Qt2 v Qt1 v Qt0 )

K1 = !(!Qt0 v !Qt1 v !Qt2 )

J2 = !(!Qt0 Qt1 v Qt0 Qt2 v Qt0 !Qt1 v Qt2 )

K2 = !(!Qt0 Qt1 Qt2 x1 v !Qt0 !Qt1 v Qt0 Qt1 v !Qt2 )

Створимо файл, у якому проведемо моделювання цих функцій:

library IEEE;

use IEEE.std\_logic\_1164.all;

entity test\_entity is

port(

CLK : in BIT;

x1 : in BIT;

y1 : out BIT;

y2 : out BIT;

y3 : out BIT;

y4 : out BIT

);

end test\_entity;

architecture test\_arch of test\_entity is

component JK

port (

C : in BIT;

J : in BIT;

K : in BIT;

Q : out BIT;

nQ : out BIT

);

end component;

signal Q0 : BIT;

signal nQ0 : BIT;

signal Q1 : BIT;

signal nQ1 : BIT;

signal Q2 : BIT;

signal nQ2 : BIT;

signal J0 : BIT;

signal K0 : BIT;

signal J1 : BIT;

signal K1 : BIT;

signal J2 : BIT;

signal K2 : BIT;

signal J3 : BIT;

signal K3 : BIT;

begin

T0 : JK

port map(

C => CLK,

J => J0,

K => K0,

Q => Q0,

nQ => nQ0

);

T1 : JK

port map(

C => CLK,

J => J1,

K => K1,

Q => Q1,

nQ => nQ1

);

T2 : JK

port map(

C => CLK,

J => J2,

K => K2,

Q => Q2,

nQ => nQ2

);

--J0 <= Q2 or '0' or '0';

--K0 <= not((Q2 or nQ1 or '0') and (Q2 or not(x1) or '0') and '1');

--J1 <= not((nQ2 or Q0 or '0') and '1' and '1');

--K1 <= not((nQ0 or '0' or '0') and (Q2 or x1 or '0') and '1');

--J2 <= nQ1 or '0' or '0';

--K2 <= not((nQ1 or nQ0 or '0') and '1' and '1');

--y1 <= not(((Q2 or Q1 or Q0) or x1 or '0') and '1' and '1');

--y2 <= not((Q2 or nQ1 or Q0) and '1' and '1');

J0 <= not(not(Q1) or Q2 or Q0);

K0 <= not((not(Q0) and not(Q2)) or (Q0 and Q2) or not(Q0) or Q2 or Q1);

J1 <= not(not(Q2) or Q1 or Q0);

K1 <= not(not(Q0) or not(Q1) or not(Q2));

J2 <= not((not(Q0) and Q1) or (Q0 and Q2) or (Q0 and not(Q1)) or Q2);

K2 <= not((not(Q0) and Q1 and Q2 and x1) or (not(Q0) and not(Q1)) or (Q0 and Q1) or not(Q2));

y1 <= not(not(Q1) or not(Q2) or Q0);

y2 <= '1';

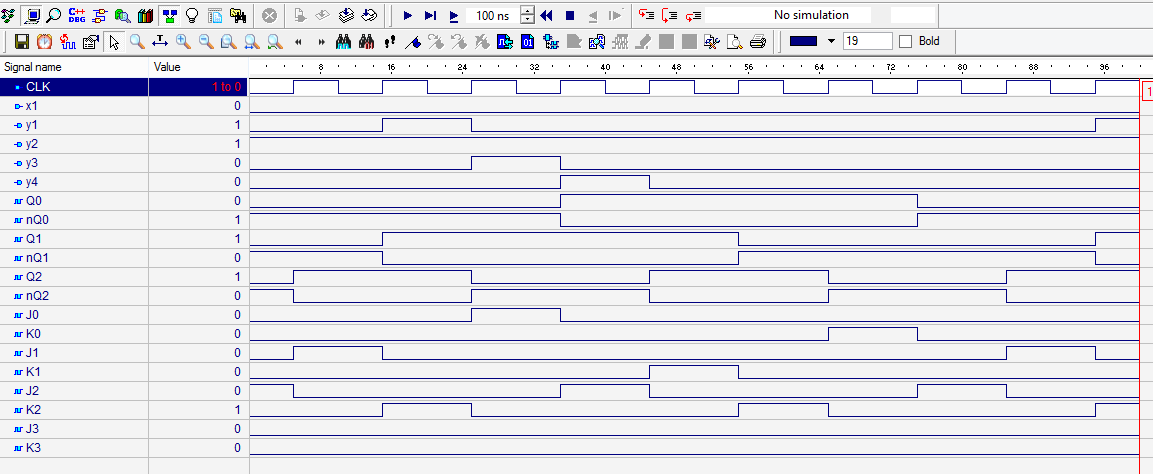
y3 <= not(not(Q1) or Q2 or Q0);

y4 <= not((not(Q0) and not(Q2)) or (Q0 and Q2) or not(Q0) or Q2 or not(Q1));

end test\_arch;

Скомпілюємо. Далі в Design Browser виберемо наш проект, ввімкнемо симуляцію і додамо усі наші сигнали на графік. Встановимо сигнал годинника з частотою 100 МГц.

Натиснемо кнопку F5. Отримаємо наступний результат:



Вітаю, ви навчилися моделювати цифрові автомати в середовищі Active-HDL.

1. **Список використаної літератури**
2. <http://www.kit-e.ru/articles/circuit/2009_03_134.php>
3. <https://www.aldec.com/en/solutions/fpga_design/documentation>
4. <https://www.aldec.com/en/products/fpga_simulation/active-hdl>
5. <https://ru.wikipedia.org/wiki/Active-HDL>
6. <http://www.eurointech.ru/active>