

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторної роботи з дисципліни
«Методи синтезу енергоефективних систем на ПЛС»
за темою:
«Проектування схеми загорткового кодеру»
для магістрів
спеціальності 123 "Комп'ютерна інженерія"
усіх форм навчання

Методичні вказівки до виконання лабораторної роботи з дисципліни «Методи синтезу енергоефективних систем на ПЛІС» за темою «Проектування схеми загорткового кодеру» для магістрів спеціальності 123 "Комп'ютерна інженерія" усіх форм навчання / Укл. І.Я Зеленьова, Т.В. Голуб. – Запоріжжя: НУ«Запорізька політехніка», 2023. – 13 с.

Укладачі:

І.Я Зеленьова, доцент, к.т.н.
Т.В. Голуб, доцент, к.т.н.

Рецензент:

Грушко С.С., доцент, к.т.н.

Відповідальний за випуск:

Т.В. Голуб, доцент, к.т.н.

Затверджено
на засіданні кафедри КСМ
Протокол № 8 від 27.04.2023

Рекомендовано до видання
на засіданні НМК КНТ
Протокол № 10 від 24.05.2023

ЗМІСТ

1 Загальні відомості	4
2 Приклад згорткового кодування (2, 1, 4)	5
2.1 Розробка загорткового кодування.....	5
2.2 Реалізація та тестування схеми кодеру (2,1,4).....	6
3 Завдання до лабораторної роботи.....	12
4 Зміст звіту по лабораторній роботі.....	12
5 Контрольні питання	13
Перелік рекомендованої літератури	13

Мета роботи: опанувати основні принципи побудови згорткового коду, розробити та виконати тестування схеми згорткового кодеру з використанням VHDL та Quartus II.

1 ЗАГАЛЬНІ ВІДОМОСТІ

Згортковий код використовується для коректування помилок. Сенса у тому, що якщо при передачі у каналах зв'язку згорткового коду у ньому з'являються помилки, то при його декодуванні можна отримати вихідну коректну послідовність біт, яка була закодована. Це досягається завдяки надлишковості при кодуванні згорткового коду.

Згорткові коди представляються трьома параметрами (n , k , m):

n – кількість вихідних кодованих бітів.

k – кількість вхідних бітів що кодуються.

m – кількість регістрів (тригерів) пам'яті для зберігання попередніх вхідних бітів.

В загальному випадку ці параметри можуть бути обрані проектувальником.

Кодером згорткового коду по суті є зсувний регістр, виходи якого підключені до напівсуматорів. Кількість напівсуматорів, що потрібні для формування одного біту коду, визначається кількістю вихідних біт регістра зсуву, що потрібно сумувати для отримання вихідного біту коду. Один напівсуматор – два вихідних біти регістра зсуву на один вихідний біт коду, два напівсуматори – три вихідних біти регістра на один вихідний біт коду і т.д.

Співвідношення "біт даних (k)/кількість біт коду (n)" називається швидкістю кодування, зазвичай позначається R . Для двох біт коду на один біт даних $R=1/2$; відповідно, для трьох біт коду на один біт даних $R=1/3$ і т.д.

Довжина регістру зсуву позначається m і визначає кількість бітів даних, що впливають на формування бітів коду.

У даній лабораторній роботі, як приклад, буде розглянутий згортковий код (2,1,4).

При обраному способі кодування згорткового коду (2,1,4) кожний вхідний біт буде кодуватися у два вихідних. Кодування цих двох вихідних бітів виконується з урахуванням нового вхідного біта та попередніх трьох вхідних біт, які вже кодувалися (тобто потрібно 4 регістри). При цьому кожен з вихідних кодованих біт не обов'язково

кодується з урахуванням усіх трьох попередніх біт. У даному прикладі перший вихідний біт кодується сумуванням за модулем 2 (напівсуматор) усіх трьох збережених попередніх вхідних бітів та вхідного, а другий – сумуванням першого, третього та вхідного біту.

2 ПРИКЛАД ЗГОРТКОВОГО КОДУВАННЯ (2, 1, 4)

2.1 Розробка загорткового кодування

Оберемо вхідну послідовність біт для кодування – наприклад, 1011.

У вихідному стані в осередках зсувного регістру містяться нулі. Далі на вхід, в першу ліворуч, надходить перший інформаційний символ. Одночасно із записом символу до першого осередку її вміст переписується до другого, другого – до третього тощо. На суматорах утворюється сума за модулем два з розрідів вмісту комірок кодера.

Закодуємо перший біт – 1. Оскільки в регістрах попередньо були всі нулі, то перший біт формується таким чином: $0+0+0+1=1$, а другий формується так: $0+0+1=1$. Тепер в регістрах вже 001 (вхідний біт зберігся у пам'ять).

Закодуємо другий біт – 0. Оскільки в регістрах вже 001, то перший біт формується таким чином: $0+0+1+0=1$, а другий біт $0+1+0=1$. Тепер в регістрах вже 010.

Закодуємо третій біт – 1. Оскільки в регістрах вже 010, то перший біт формується таким чином: $0+1+0+1=0$, а другий біт $0+0+1=1$. Тепер в регістрах вже 101.

Закодуємо четвертий біт – 1. Оскільки в регістрах вже 101, то перший біт формується таким чином: $1+0+1+1=1$, а другий біт $1+1+1=1$. Тепер в регістрах вже 011.

В **результаті** отримуємо кодовану послідовність – 11 11 01 11.

На основі аналізу вищенаведених правил кодування можна сформувати таблицю дійсності для кодування даних (табл. 2.1).

Таблиця 2.1 - Таблиця дійсності для кодування даних

Input bit	Input State			Output Bits		Output State		
I ₁	s ₁	s ₂	s ₃	O ₁	O ₂	s ₁	s ₂	s ₃
0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	0	0
0	0	0	1	1	1	0	0	0
1	0	0	1	0	0	1	0	0
0	0	1	0	1	0	0	0	1
1	0	1	0	0	1	1	0	1
0	0	1	1	0	1	0	0	1
1	0	1	1	1	0	1	0	1
0	1	0	0	1	1	0	1	0
1	1	0	0	0	0	1	1	0
0	1	0	1	0	0	0	1	0
1	1	0	1	1	1	1	1	0
0	1	1	0	0	1	0	1	1
1	1	1	0	1	0	1	1	1
0	1	1	1	1	0	0	1	1
1	1	1	1	0	1	1	1	1

Таблиця дійсності необхідна для визначення закону функціонування та подальшої реалізації схеми кодера. На основі цієї таблиці формуються функції вихідних сигналів пристрою.

2.2 Реалізація та тестування схеми кодера (2,1,4).

З вищенаведених міркувань можна зробити висновок, що для реалізації кодера нам потрібні лише регістр зсуву для зберігання попередніх трьох кодованих біт, а також напівсуматори для формування вихідних біт.

Опис схеми кодера мовою VHDL виконаємо безпосередньо у пакеті Quartus II. Моделювання та тестування схеми виконаємо на мікросхемах фірми Altera\Intel.

Отже, опишемо напівсуматор, який складає два вхідних біти (Лістинг 2.1).

Лістинг 2.1 – Опис **напівсуматора**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity h_adder is
    port (a,b:in std_logic;
          s: out std_logic);
end h_adder;
architecture fh1 of h_adder is
begin
    s<=NOT(a XOR (NOT b));
end fh1;
```

Опишемо D-триггер, який використовується у регістрі зсуву (Лістинг 2.2).

Лістинг 2.2 – Реалізація **D-триггера**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity DTrigger is
    Port ( d : in std_logic;
          clk : in std_logic;
          q : out std_logic);
end DTrigger;
architecture Behavioral of DTrigger is
    signal q_temp:std_logic;
begin
    process(clk)
    begin
        if(clk'event and clk='1')then
            q_temp<=d;
        end if;
    end process;
    q<=q_temp;
end Behavioral;
```

Опишемо регістр зсуву мовою VHDL (Лістинг 2.3).

Лістинг 2.3 – Реалізація **регістра зсуву**

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ShiftReg is
    Port( a : in std_logic; -- вхідний біт
          clk : in std_logic; -- по clk тригери змінюють стан
          b1: out std_logic; -- виходи регістра (тригерів)
          b2: out std_logic;
          b3: out std_logic;
          b4: out std_logic);
end ShiftReg;
architecture gen_shift of ShiftReg is
    signal z:std_logic_vector(0 to 3);
    component DTrigger -- Створення компонента D-тригера
        port (d,clk:in std_logic;
              q:out std_logic);
    end component;
    begin
        DTrigger0: DTrigger port map(a,clk,z(0)); -- по clk значення a
        записується до z(0)
        DTrigger1: DTrigger port map(z(0),clk,z(1)); -- і т. д.
        DTrigger2: DTrigger port map(z(1), clk, z(2));
        DTrigger3: DTrigger port map(z(2), clk, z(3));
        b1<=z(0);
        b2<=z(1);
        b3<=z(2);
        b4<=z(3);
    end gen_shift;
```

Тепер залишається тільки скласти кодер, як зображено на рисунку 2.1. Для спрощення процесу поєднання функціональних блоків будемо використовувати схемний редактор. Для цього створимо Block Diagram File та додамо на робочу область створені VHDL файли.

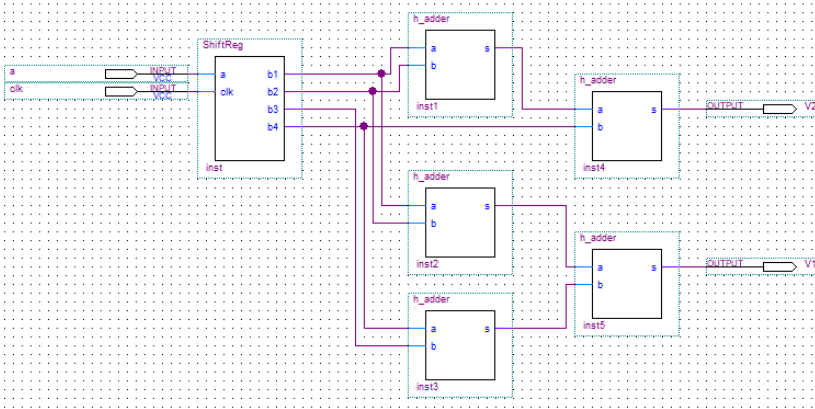


Рисунок 2.1 – Функціональна схема кодеру

Як можна побачити на рисунку 1, для обчислення суми чотирьох бітів регістра, щоб отримати перший вихідний біт, потрібно три напівсуматори. Для складання трьох бітів регістра, щоб отримати другий вихідний біт, достатньо двох напівсуматорів.

Щоб протестувати створену схему, створимо Vector Waveform File та додамо у нього сигнали схеми. Для завдання значення вхідного сигналу clk виділимо його, натиснемо Overwrite Clock, введемо дані, як зображено рисунку 2.2, та натиснемо ОК.

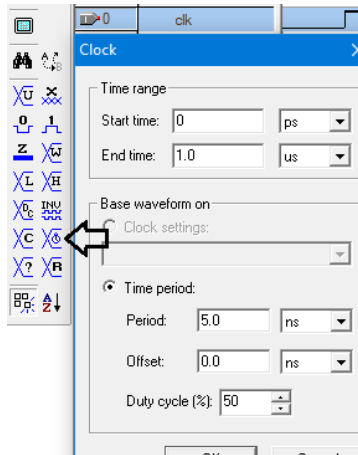


Рисунок 2.2 – Завдання значення вхідного сигналу clk

Для завдання значення вхідного сигналу **a** виділимо його, натиснемо Arbitrary Value, введемо дані, як зображено рисунку 2.3, та натиснемо OK.

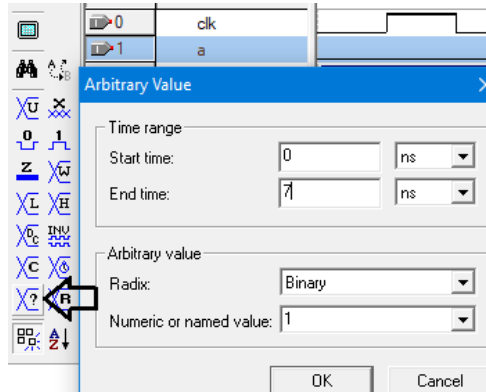


Рисунок 2.3 – Завдання значення вхідного сигналу **a**

Натиснемо Arbitrary Value ще раз та введемо дані, як зображено рисунку 2.4, та натиснемо OK.

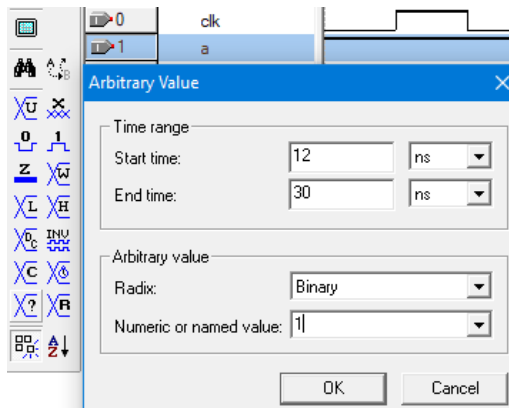


Рисунок 2.4 – Завдання значення вхідного сигналу **a**

На рисунку 2.5 наведена часова діаграма роботи кодеру, отримана в результаті тестування схеми.

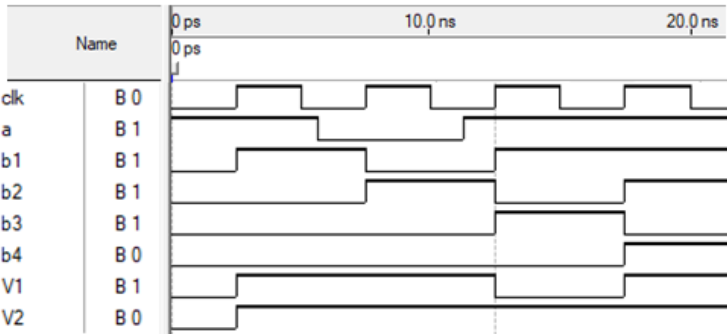


Рисунок 2.5 - Часова діаграма роботи кодера

Як можна побачити, діаграма показує результати згідно з таблицею дійсності. Послідовність виконання кроків кодування для зручності перевірки наведено на рисунку 2.6.

Input Bit		Input State			Output Bits	
	I ₁	s ₁	s ₂	s ₃	O ₁	O ₂
1 →	0	0	0	0	0	0
	1	0	0	0	1	1
	0	0	0	1	1	1
	1	0	0	1	0	0
3 →	0	0	1	0	1	0
	1	0	1	0	0	1
	0	0	1	1	0	1
	1	0	1	1	1	0
2 →	0	1	0	0	1	1
	1	1	0	0	0	0
	0	1	0	1	0	0
	1	1	0	1	1	1
4 →	0	1	1	0	0	1
	1	1	1	0	1	0
	0	1	1	1	1	0
	1	1	1	1	0	1

Рисунок 2.6 - Послідовність виконання кроків кодування

Стан зсувного регістру залежить тільки від інформаційних символів, що передаються. При цьому скільки б інформаційних символів не передавалося, кількість станів зсувного регістру обмежена і становить $2k$, де k - довжина зсувного регістру.

Послідовність станів теж не є випадковою. Кожний наступний стан виходить із попереднього шляхом зсуву праворуч, тобто «успадкоує» від попереднього стану три біти. Четвертий біт приймає значення «1» або «0», залежно від чергового вхідного біта даних.

3 ЗАВДАННЯ ДО ЛАБОРАТОРНОЇ РОБОТИ

3.1 Обрати самостійно та обґрунтувати в звіті параметри згорткового кодера (n , k , m):

n – кількість вихідних кодованих бітів.

k – кількість вхідних бітів що кодуються.

m – кількість регістрів (тригерів) пам'яті для зберігання попередніх вхідних бітів.

3.2 Виконати синтез та тестування схеми кодера, зробити висновок про коректність роботи схеми. Навести в звіті всі етапи синтезу.

3.3 Проаналізувати звіт з імплементації та навести в звіті скриншот топології схеми кодера на кристалі мікросхеми, обраної для моделювання.

3.4 Дати відповіді на два контрольних питання (на вибір).

4 ЗМІСТ ЗВІТУ ПО ЛАБОРАТОРНІЙ РОБОТІ

Зміст звіту має містити:

- опис всіх етапів розробки кодеру;
- фрагмент розробленої схеми кодеру;
- результати моделювання програмного опису розробленої схеми на VHDL;
- результати тестування кодеру на стенді;
- відповіді на контрольні питання.

Результати тестування отриманої схеми привести в звіті у вигляді часових діаграм, таблиці розпінування та фото стенду з результатами. Зробити висновки по роботі.

5 КОНТРОЛЬНІ ПИТАННЯ

1. Що таке швидкість кодування для згорткового коду?
2. Які параметри є основними для визначення згорткового коду?
3. Від чого залежить стан зсувного регістру?
4. Для чого використовують згорткові коди?
5. Чому згортковий код дозволяє коректувати помилки?

ПЕРЕЛІК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Signal processing & simulation newsletter [Електронний ресурс]. : complextoreal.com, 2018. - Режим доступу: <http://complextoreal.com/wp-content/uploads/2018/09/convolutional-codes.pdf>.
2. С. А. Іванець, Ю. О. Зубань, В. В. Казимир, В. В. Литвинов, Проектування комп'ютерних систем на основі мікросхем програмованої логіки [Текст]: монографія. / С. А. Іванець, Ю. О. Зубань, В. В. Казимир, В. В. Литвинов.; Сумський державний університет, Чернігівський державний технологічний університет. – Суми, Чернігів: СДУ, ЧДТУ, 2013. – 313 с.
3. Convolutional code [Електронний ресурс]. : wikipedia.org, 2022. - Режим доступу: https://en.wikipedia.org/wiki/Convolutional_code