Одеський національний університет імені І.І.Мечникова
(повне найменування вищого навчального закладу)

Факультет математики, фізики та інформаційних технологій
(повне найменування інституту, назва факультету (відділення))

Кафедра комп'ютерних систем та технологій
(повна назва кафедри (предметної, циклової комісії))

# Дипломна робота

на здобуття ступеня вищої освіти «бакалавр»
(освітньо-кваліфікаційний рівень)

Побудова елементів трійкових обчислювальних систем
(назва українською)

Construction of elements of ternary computing systems
(назва англійською)

Виконав: студент денної форми навчання
спеціальності 123 Комп'ютерна інженерія .
(шифр і назва напряму підготовки, спеціальності)
Буравчук Патрик Борисович
(прізвище, ім'я, по-батькові)

Керівник д.т.н., проф. Гунченко Ю. О.
(науковий ступінь, вчене звання, прізвище та ініціали, **підпис**)

Рецензент ст. викл.Мартинович Л. Я.
(науковий ступінь, вчене звання, прізвище та ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
№ 8 від «08» травня 2022 р.

Завідувач кафедри
_____ Ю.О. Гунченко
(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___
протокол № __ від «___» ____ 2022р.
Оцінка _____/_____/_____
(за національною шкалою, шкалою ECTS, бали)

Голова ЕК
_____ Н.Ф. Казакова
(підпис) (прізвище, ініціали)

Одеса - 2022

# АНОТАЦІЯ

У дипломній роботі розробляється тема «Побудова елементів трійкових обчислювальних систем».

Зараз найбільш розповсюджена двійкова логіка, проте вона має ряд недоліків, які можна усунути завдяки використанню трійкової логіки, серед яких збільшення діапазону представлення чисел, пришвидшення виконання операцій, зменшення кількість обладнання. Цим обумовлена актуальність теми.

Мета роботи – синтез одномісних логічних елементів трійкової логіки шляхом розробки метода та його програмної реалізації.

Предмет дослідження – арифметичні та логічні елементи трійкових обчислювальних систем.

Об’єкт дослідження – методологія та алгоритми побудови елементів трійкової логіки.

Для досягнення мети, в роботі поставлено та вирішено наступні взаємопов’язані задачі:

1) Аналіз трійкової логіки.
2) Дослідження існуючих засобів реалізації трійкової системи.
3) Реалізація логічних та арифметичних елементів і вузлів трійкових обчислювальних систем.
4) Порівняння розроблених реалізацій з існуючими аналогами та прототипами.
5) Програмна реалізація методу синтезу трійкових унарних операцій.

В роботі запропонований метод синтезу трійкових унарних функцій та його програмна реалізація.

# АННОТАЦИЯ

В дипломной работе разрабатывается тема «Построение элементов троичных вычислительных систем».

Сейчас наиболее распространенной является двоичная логика, но у нее есть ряд недостатков, которые можно устранить, применив троичную логику, среди которых увеличение диапазона представления чисел, ускорение выполнения операций, уменьшение количества оборудования. Этим обусловлена актуальность темы.

Цель работы – синтез одноместных логических элементов троичной логики путем розработки метода и его программной реализации.

Предмет исследования – арифметические и логические элементы троичных вычислительных систем.

Объект исследования – методология и алгоритмы построения элементов троичной логики.

Для достижения цели, поставлены и решены следующие взаимосвязанные задачи:

1) Анализ троичной логики.
2) Исследование существующих реализаций троичных систем.
3) Реализация логических и арифметических элементов и узлов троичных вычислительных систем.
4) Сравнение разработанных реализаций с существующими аналогами и прототипами.
5) Программная реализация метода синтеза троичных унарных операций.

В роботе предложен метод синтеза троичных унарных функций и его программная реализация.

# ABSTRACT

In the diploma work, the topic "Construction of elements of ternary computing systems" is being developed.

Now the most common is binary logic, but it has a number of disadvantages that can be eliminated by applying ternary logic, including an increase in the range of representation of numbers, acceleration of operations, and a decrease in the amount of equipment. This is due to the relevance of the topic.

The purpose of the work is the synthesis of single-place logical elements of ternary logic by developing a method and its software implementation.

The subject of research is the arithmetic and logical elements of ternary computing systems.

The object of research is the methodology and algorithms for constructing elements of ternary logic.

To achieve the goal, the following interrelated tasks were set and solved:

1) Analysis of ternary logic.

2) Research of existing implementations of ternary systems.

3) Implementation of logical and arithmetic elements and nodes of ternary computing systems.

4) Comparison of the developed implementations with existing analogues and prototypes.

5) Software implementation of the method of synthesis of ternary unary operations.

The robot proposes a method for the synthesis of ternary unary functions and its software implementation.

**ЗМІСТ**

стор.

ВСТУП ..... 7
1 ТЕОРИТИЧНІ ОСНОВИ ТРІЙКОВОЇ ЛОГІКИ ..... 9
1.1 Трійкова система числення ..... 10
1.2 Переваги та недоліки трійкової системи числення ..... 11
1.3 Ефективність тризначної логіки ..... 14
1.4 Одномісні операції в трійковій логіці ..... 18
1.5 Алгебраїчні властивості трійкової логіки ..... 19
1.6 Історія, сьогодення та майбутнє трійкової логіки ..... 20
1.7 Висновок ..... 23
2 ОГЛЯД АНАЛОГІВ СТРУКТУР ТРІЙКОВИХ ЕЛЕМЕНТІВ ..... 24
2.1 Пороговий елемент на магнітних елементах ..... 24
2.2 Пороговий елемент на біполярних транзисторах ..... 25
2.3 Пристрої на основі ПЕТЛ ..... 27
2.4 Трійкові елементи на КМОП-транзисторах ..... 28
2.5 Трійкові елементи на К-МОП-С транзисторах ..... 31
2.6 Висновок ..... 32
3 БАГАТОПОРОГОВИЙ ЕЛЕМЕНТ БАГАТОЗНАЧНОЇ ЛОГІКИ ТА ТРІЙКОВІ ЕЛЕМЕНТИ НА ЙОГО ОСНОВІ ..... 33
3.1 Чотирьохпорогова реалізація БПЕБЛ ..... 34
4 МЕТОД ПОБУДОВИ ТРІЙКОВИХ ОДНОМІСНИХ ФУНКЦІЙ ..... 38
4.1 Універсальний пристрій для реалізації трійкових одномісних функцій 38
4.2 Метод побудови трійкових унарних функцій ..... 40
4.3 Програмна реалізація методу побудови трійкових одномісних функцій ..... 43
4.4 Висновок ..... 46
ВИСНОВОК ..... 47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ..... 48

ДОДАТОК А ..... 54
Код програмної реалізації методу синтезу трійкових унарних функцій ..... 54
ДОДАТОК Б ..... 67
Синтезовані структури трійкових одномісних функцій ..... 67

# ВСТУП

Однією з найважливіших тем розмов про майбутнє нерідко стає питання технологій. Згідно з законом Мура, кількість транзисторів в процесорі подвоюється кожні півтора року.

Провідні компанії, що займаються напівпровідниковими технологіями та випуском напівпровідникових пристроїв, підійшли до дуже маленьких масштабів. Мінімальні розміри затвора транзистора – 14, 10, та навіть 7 і 5 нанометрів, використовуваного в сучасних серійно вироблюваних процесорах.

Мінімальний фізично можливий розмір затвора працездатного кремнієвого транзистора – п'ять нанометрів. Нижче цього значення відбувається явище "тунельний ефект" (електрони отримують можливість прорвати потенційний бар'єр p-n переходу і, грубо кажучи, почати вільно "гуляти" по сусідніх транзисторах процесора).

Довгий час вважалося, що це – крайня межа, після якого нарощування продуктивності обчислювальних машин знову стане приводити до зростання їх розмірів. І цю межу, якщо темпи мініатюризації збережуться, буде досягнуто вже приблизно через чотири-п’ять років.

Стратеги Intel ще у 2016 році прийшли до висновку, що одним з варіантів розв'язання проблеми може бути перехід від двійкової системи до систем з більшою значністю, в тому числі і до трійкової.

Однією з перешкод, що стримують розвиток та впровадження трійкової техніки, є незовсім вірне уявлення про незвичайність і важку досяжність трійкової логіки.

Насправді тризначна логіка є цілком коректною і відповідає дійсності, але й більш зручна і звична для людей, ніж двозначна логіка, адже кожна людина з нею стикається практично кожен день, приймаючи те чи інше рішення.

Зараз з'являються нові можливості напівпровідникових технологій на основі яких можна розробляти трійкові логічні та арифметичні елементи і шукати нові схемотехнічні підходи до практичної реалізації недвійкових комп'ютерів.

Тому актуальною практичною задачею є створення і реалізація загального підходу до трійкових вузлів та методів синтезу трійкових логічних та арифметичних елементів, так як досі немає стандартів в розробці і реалізації трійкових елементів, отже, немає чітко представленої елементної бази.

Предметною областю дослідження є логічні та арифметичні елементи і пристрої трійкових обчислювальних систем, а також функціональні вузли.

Мета роботи – синтез одномісних логічних елементів трійкової логіки шляхом розробки метода та його програмної реалізації.

Предмет дослідження – арифметичні та логічні елементи трійкових обчислювальних систем.

Об’єкт дослідження – методологія та алгоритми побудови елементів трійкової логіки.

Для досягнення мети, в роботі поставлено та вирішено наступні взаємопов’язані задачі:

1) Аналіз трійкової логіки.
2) Дослідження існуючих засобів реалізації трійкової системи.
3) Реалізація логічних та арифметичних елементів і вузлів трійкових обчислювальних систем.
4) Порівняння розроблених реалізацій з існуючими аналогами та прототипами.
5) Програмна реалізація методу синтезу трійкових унарних операцій.

# 1 ТЕОРИТИЧНІ ОСНОВИ ТРІЙКОВОЇ ЛОГІКИ

Сучасні обчислювальні системи, реалізація інформаційних технологій практично повністю базуються на двійковій логіці. Всі обчислення проводяться на рівні нулів і одиниць. Двійкова логіка комп'ютерів – природний наслідок фізичних особливостей напівпровідників. Одиниця і нуль – в даний час це основа всіх обчислювальних процесів в комп'ютерах і інших «розумних» пристроях.

Відповідно до закону Мура, кількість транзисторів в мікропроцесорах подвоюється кожні півтора року. Корпорація Intel і інші компанії, що займаються випуском напівпровідникових пристроїв, вже впритул підійшли до дуже маленьких масштабів; гіга- і навіть терагерцовими частотами комп'ютерних систем вже нікого особливо не здивуєш і не налякаєш, так само як і мега-, гіга- і терабайтовими сховищами даних.

Зараз галузь стоїть на складному роздоріжжі: поміщати все нові і нові виконавчі кремнієві транзистори на платах стає важче – і так масштаби вже нанометричні. У міру подальшого зменшення розмірів транзисторів забезпечено ускладнення технологічного процесу і безліч інших проблем, аж до квантової невизначеності.

Технологія інтегральних схем за рахунок великих вкладень і багаторічних зусиль доведена до досконалості. Наміри змінити її елементну базу не отримають підтримки – немає підстав проходити пройдене. Прорив в цій сфері можливий лише при появі кардинально нових ідей, що несуть в собі багатообіцяючі перспективи.

На допомогу приходить трійкова логіка.

Трійкова логіка (англ. Ternary logic) – логіка, що використовує три значення істини: «правда», «неправда», «незнаю». Вона була запропонована Яном Лукасевичем у 1920 році.

Як правило, для визначення станів «неправда» і «правда» використовують знаки «–» і «+», третьому стану відповідає значення «0», тобто використовується симетрична система числення. Вона більш зрозуміла і

близька для людського розуміння, але також великою популярністю користуються такі набори: $\{0, 1, 2\}$, $\{-1, 0, 1\}$, $\left\{0, \frac{1}{2}, 1\right\}$, $\{N, Z, P\}$ [1].

Трійковою логікою зацікавились задовго до появи перших комп'ютерів в зв'язку з чудовими властивостями симетричного коду чисел [2].

Практична доцільність трійкової техніки не очевидна. Ясно, що трійкова техніка рівноцінна двійковій техніці в тому сенсі, що все, здійсненне в одній з них, з тим чи іншим наближенням можна здійснити і в інший. Ясно також, що тризначні вентилі і елементи пам'яті повинні бути складніше і дорожче, ніж двозначні, а тризначна логіка явно складніше двозначної. Але з іншого боку, тризначні елементи пам'яті потужніші і операційні можливості тризначних вентилів багатші.

Отже, обробка даних в умовах троїчної техніки здійснюється при однаковій фізичній швидкодії елементів швидше, а структура троїчного пристрою, як правило, виявляється простіше, ніж структура функціонально рівноцінного двійкого пристрою.

Іншими словами, троїчна техніка характеризується в порівнянні з двійковою ускладненням елементів, завдяки якому можливо спрощення структур, що з них створюються, і збільшення швидкості обробки даних. Чудово, що трійкова техніка є єдиною недвійковою технікою, що не пов'язана з необхідністю посилення діючими в двійковій техніці допусків на параметри сигналів і характеристики елементів.

Збільшення значності з двох до трьох без посилення допусків досягається за рахунок недовикористовуваних двійковою технікою можливості розрізняти сигнал як по амплітуді, так і по полярності.

## 1.1 Трійкова система числення

Логіці, що оперує трьома значеннями, природним чином відповідає трійкова система числення – трійкова симетрична, якщо говорити точніше, найпростіша з симетричних систем. До цієї системи вперше звернувся Фібоначчі для вирішення своєї «задачі про ваги».

Трійкова система числення – позиційна система числення з цілочисловою основою, рівною 3. Відомо про дві системи: симетричну і несиметричну. У несиметричній системі, як правило використовують цифри {0,1,2}, а в симетричній використовуються симетричні значення, наприклад {−,0,+}, {−1,0,+1} [3].

Трійкова логіка для комп'ютерів, вводить свої одиниці виміру інформації: трит та трайтен (аналогічно двійковим біту та байту).

Трит (приблизно 1.585 біта) – це трійчастий розряд в трійчастий системі числення. Трайтен це мінімальна одиниця, що адресується в пам'яті троїчного комп'ютера. Один трайтен дорівнює шести тритам.

В цифровій електроніці «біт» реалізується двійковим тригером. Це мінімальний логічний елемент двійкового комп'ютера. Трит – потрійним тригером, який здатний, одночасно, оперувати відразу трьома значеннями, а не двома як у двійкового тригера.

Один трайтен здатний закодувати 729 значень, проти 256 одного байта. Він приймає значення з діапазону від -364 до 364, на відміну від байтового діапазону 0 – 255. Це дозволяє обробляти більше інформації за один такт процесора [1].

Якщо тактові частоти двійкової і трійкової шини однакові, то у потрійного комп'ютера, інформація буде передаватися в 2,8 рази швидше ніж у двійковій машині.

## 1.2 Переваги та недоліки трійкової системи числення

Питання про доцільність застосування трійкового коду в цифровій техніці виник разом з появою швидкодіючих автоматичних цифрових машин і протягом усього наступного періоду неодноразово розглядалося в ряді монографій з цифрових пристроїв [5].

Трійчастий код є найпростішим (щодо структури та фізичної реалізації) кодом, який допускає використання симетричного алфавіту, тобто може бути

симетричним кодом. Цим обумовлені його головні переваги перед іншими кодами стосовно цифрових машин.

З симетричним кодом пов'язано шість цінних властивостей:

1) Природність представлення негативних чисел;
2) Відсутність проблеми округлення: обнулення непотрібних молодших розрядів наближає число до найближчого «грубого».
3) Таблиця множення в цій системі, як зазначив О. Л. Коші, приблизно в чотири рази коротше [4].
4) Наявність позитивної та негативної цифр дозволяє безпосередньо представляти як позитивні, так і негативні числа. При цьому немає необхідності в спеціальному розряді знака і не треба вводити додатковий (або зворотний) код для виконання арифметичних операцій з негативними числами. Всі дії над числами, представленими в троїчній симетричній системі числення виконуються природно з урахуванням знаків чисел. Знак числа визначається знаком старшої значущої цифри числа: якщо вона позитивна, то і число позитивне, якщо негативна, то і число негативне. Щоб змінити знак, треба змінити знаки всіх його цифр.
5) При підсумовуванні великої кількості чисел значення для перенесення в наступний розряд зростає зі збільшенням кількості доданків не лінійно, а пропорційно квадратному кореню числа доданків.

Внаслідок зазначених властивостей в симетричному троїчному коді докорінно спрощується логіка і скорочується число необхідних варіантів арифметичних операцій, відкривається можливість оперування зі словами різної довжини, істотно спрощується структура операційних пристроїв, виходить значна економія часу, що витрачається на виконання операцій, а в ряді випадків також обладнання.

Основними плюсами трійкової логіки і системи числення є:

1) Трійкова СЧ дозволяє вміщати більший діапазон чисел в пам'яті трійкового комп'ютера, оскільки $3^n > 2^n$. Це прискорює послідовну передачу та обробку даних.
2) Для трійкової системи числення потрібно менше розрядів, ніж двійковій для представлення числа. Це означає, що трійкова система є економічнішою, в тому сенсі, що кількість обладнання, необхідне для реалізації p-ічного масиву, що володіє заданим числом різних значень, виявляється мінімальним при p = 3. Оцінка витрат обладнання проводиться при цьому в припущенні, що кількість обладнання в p-значному елементі пропорційно p. При такому припущенні відношення кількості обладнання, необхідного для реалізації p-ічного масиву, до кількості аналогічного обладнання, необхідного для реалізації двійкового масиву тієї ж значності, виражається функцією [6]: $f(p) = p / (2\log 2p)$, яка має мінімум при p = 2,718…, а для цілих p при p = 3: $f(3) = 0{,}946$. Таким чином, якщо елементи задовольняють зазначеному припущенню, то трійчастий код виявляється на 5,4% економніше двійкового.
3) З одночасним зменшенням розмірів елементів троїчної логіки і споживання енергії, технологічність їх виготовлення та швидкодії підвищується [7].
4) Проблема округлення зникає.
5) Є можливість передачі тризначного алфавіту по одному дроту без збільшення часу і без посилення допусків. Йдеться про передачу сигналами позитивної і негативної полярності. Реалізація цієї можливості при використанні троїчного коду замість двоїчного підвищує пропускну здатність каналів в 1,59 рази: при послідовній передачі по одному дроту в 1,59 рази скорочується витрачається час, при паралельній передачі в 1,59 рази зменшується число необхідних проводів[8].

Не зважаючи на переваги, трійкова логіка і система числення мають також і недоліки:

1) Відсутність недвійкових логічних елементів і перевірених схемотехнічних рішень, що стосуються устрою вузлів недвійкових комп'ютерів [7].
2) Електронні компоненти для побудови логіки, що використовують більше двох станів, вимагають більше матеріальних витрат на їх виробництво, досить складні в реалізації, і споживають більше електроенергії.
3) Немає стандартного підходу реалізації логічних елементів, отже – немає чітко представленої елементарної бази.
4) В умовах інтегральної технології можливість спрощення структури пристроїв за рахунок ускладнення елементів і економія кількості з'єднань між елементами можуть виявитися важливіше звичайної економії деталей [8].

Отже, очевидно, що недоліків трійкової системи числення менше ніж переваг. Тому, що трійкова логіка цілком включає в себе двійкову логіку, як центральна підмножина, тому трійкові комп'ютери можуть робити все, що роблять двійкові комп'ютери, плюс деякі додаткові можливості.

## 1.3 Ефективність тризначної логіки

Одним з бар'єрів, що стримують розвиток і поширення трійкової техніки, є невірне уявлення про незвичайність і важку досяжність тризначної логіки. Сучасна формальна логіка (як традиційна, так і математична) заснована на принципі двозначності. У числі її фундаментальних законів є закон виключеного третього: «Третього не дано», пояснюється зазвичай в тому сенсі, що правильна логіка нічого, крім «Так» і «Ні», допустити не може. Тризначна логіка при цьому асоціюється з інтуїцією, модальностями, мікросвітом і іншими речами, але тільки не з повсякденною дійсністю, яка за сформованим

протягом століть переконанням, нібито влаштована і функціонує за двійковими правилами.

Насправді тризначна логіка не лише є цілком коректною і відповідає дійсності, але навіть більш зручна і звична для людей, ніж двозначна логіка. Це підтверджують наступні приклади.

Перший приклад – ваги важелів (рис.1.1). Вони представляють собою характерний трійковий пристрій, трьом станам якого відповідають три можливих відношення: $A > B, A = B, A < B$. Для порівняння розглянемо також виконавчі ваги, які можуть приймати тільки два стани, відповідні, наприклад, відношенням $A > B, A \leq B$ (рис.1.2). Зрозуміло, що двійкові ваги менш зручні, ніж трійчасті. Тільки в разі $A > B$ результат зважування на них визначається відразу, а в інших двох випадках необхідно проводити повторне зважування, помінявши місцями A і B [9].

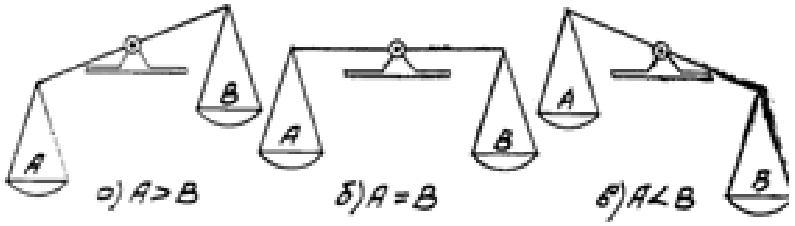



Рисунок 1.1 – Троїчні ваги

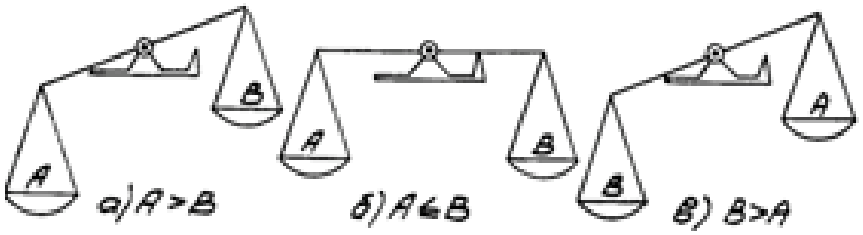



Рисунок 1.2 – Двоїчні ваги

Наступний приклад – розгалуження по знаку змінної $x$ (рис.1.3). Цей приклад демонструє принципову відмінність тризначної логіки від двозначної. Вона полягає в тому, що одне і теж може бути представлено в більш компактному вигляді в троїчній логіці, ніж в двійковій. У розглянутому прикладі трійкове розгалуження по знаку $x$ описане заданням єдиної тризначної операції $sign(x)$ і виконується за один крок, в той час як таке ж розгалуження, яке здійснюється за допомогою засобів двозначної логіки, пов'язане з необхідністю двох операцій і виконується, за два кроки. Такі алгоритми розгалуження часто використовуються в «Розумних будинках» для обробки сигналів з датчиків та відповідної реакції на них. Трійкова логіка в такому випадку допоможе значно пришвидшити цю реакцію [9, 45 – 46].

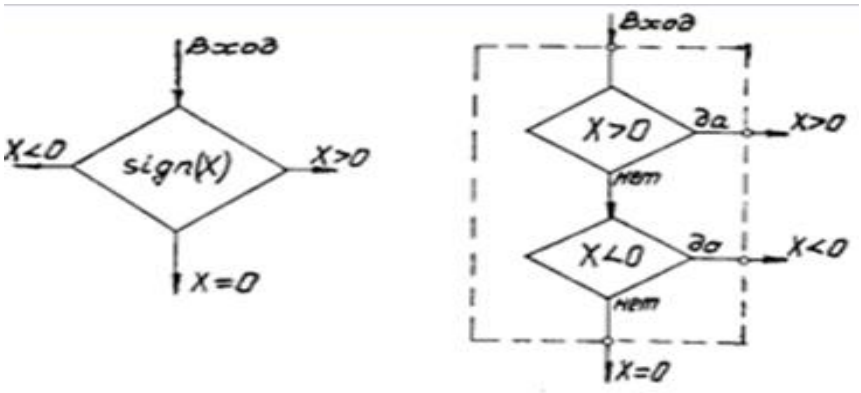



Рисунок 1.3 – Операція розгалуження по знаку: а) троїчна схема;
б) двійкова схема

Наведені приклади показують, що тризначна логіка не є щось протиприродне або незвичайне. Вона не тільки доступна для людей, але дозволяє міркувати більш просто і більш швидко в порівнянні з міркуваннями в умовах двозначної логіки. На практиці люди користуються, мабуть, переважно тризначною логікою [9].

З огляду на все вищесказане можна виділити основні властивості трійкової логіки, що визначають її ефективність та практичну цінність троїчного коду:

- має місце природне уявлення чисел зі знаком, тобто не потрібно користуватися штучними прийомами типу прямого, зворотного або додаткового коду;
- знак числа визначається знаком старшої ненульової цифри і не потрібно використовувати спеціальний знаковий біт, як в двійковій системі;
- просто проводиться порівняння чисел за величиною, при цьому не потрібно звертати увагу на знак числа;
- відповідно до цього команда розгалуження по знаку в троїчній машині займає в два рази менше часу, ніж в двоїчній;
- усічення довжини числа рівносильне правильному округленню - способи округлення, які використовуються в довічних машинах не забезпечують цього;
- трійчастий суматор здійснює віднімання при інвертуванні одного з доданків, звідки випливає, що трійчастий лічильник автоматично є реверсивним;
- в тривходовому троїчному суматорі перенесення в наступний розряд виникає в 8 ситуаціях з 27, а в двійковому суматорі – в 4 з 8;
- таблиці множення і ділення майже так само прості, як і в двійковій системі, множення на -1 інвертує множник;
- трирівневий сигнал більш стійкий до впливу перешкод в лініях передачі. Це означає що спеціальні методи надлишкового кодування троїчної інформації простіше, ніж двійкової.

## 1.4 Одномісні операції в трійковій логіці

Усього в трійковій логіці існує всього $3^{(3^n)}$ операцій для n аргументів. Тому можна вирахувати, що в даній логіці є $3^{(3^1)} = 27$ одномісних операцій [1]. У той час як в двійковій логіці $2^{(2^1)} = 4$ одномісних операцій.

Розглянемо основні одномісні операції в трійковій логіці.

Отже, перша – це інверсія. Трійкова інверсія – унарна операція, яка міняє місцями два з трьох логічних станів.

$NOT^-, NOT, NOT^+$ – оператори інверсії, що зберігають стан -, 0 і + відповідно, коли він відповідає типу оператора, або інвертують в значення, не рівне вихідному стану і не відповідає типу оператора інверсії, тобто в третій стан, що залишився. Оператори інверсії:

- $NOT^-$ - інверсія, що міняє місцями 0 і +1
- $NOT$ – інверсія, що міняє місцями -1 и +1.
- $NOT^+$ - інверсія, що міняє місцями -1 и 0.

Наступна одномісна операція – операція вибору.

$S^-$, $S$ і $S^+$ — оператори вибору. Таблиця істиності кожної з цих трьох операцій містить всюди "-", крім єдиного значення, яке нею можна вибрати.

Ще однією з найпоширеніших одномісних операцій в тріковій логіці є модифікація. Повна назва цих одномісних операцій: збільшення на одиницю по модулю три (INC) і зменшення на одиницю по модулю три (DEC). Збільшення на одиницю по модулю три – це циклічне додавання одиниці. У разі переповнення трита рахунок починається знову.

Наступні операції – граничне збільшення і зменшення.

↗, ↘— схожі з операторами модифікації, але при переповненні трита рахунок не починається знову, і значення так і залишається мінімальним або максимальним [1].

Існує також циклічне заперечення. Воно позначається (a)′. Якщо подивитись на його таблицю істинності, то видно, що це заперечення циклічно зсуває вхідні змінні.

В табл. 1.1 наведена таблиця істинності для всіх вищеописаних операцій.

Таблиця 1.1 – Таблиця істинності одномісних операцій

| a | $NOT^-$ | $NOT$ | $NOT^+$ | $S^-$ | $S$ | $S^+$ | $INC$ | $DEC$ | ↗ | ↘ | ′ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | - | + | 0 | + | - | - | 0 | + | 0 | - | 0 |
| 0 | + | 0 | - | - | + | - | + | - | + | - | + |
| + | 0 | - | + | - | - | + | - | 0 | + | 0 | - |

## 1.5 Алгебраїчні властивості трійкової логіки

Алгебраїчний підхід полягає в тому, щоб визначити над множиною {–, 0, +} двомісні операції диз'юнкцію і кон'юнкцію та одномісні (', S, інверсія) операції за допомогою законів, а решту властивостей вже виводити з них алгебраїчно.

Основні алгебраїчні властивості [12]:

1) Властивості констант:

$$a \wedge (+) = a$$
$$a \wedge (-) = -$$
$$a \vee (+) = +$$
$$a \wedge (-) = a$$
$$\overline{(-)} = (+)$$
$$\overline{(+)} = (-)$$

2) Для кон'юнкції і диз'юнкції в троїчній логіці зберігаються комутативні, асоціативні і дистрибутивні закони, закон ідемпотентності.

3) Закон подвійного заперечення (заперечення Лукасевича) і потрійного (циклічного) заперечення:

$$\bar{\bar{a}} = a$$
$$a''' = a$$

4) Буквальне визначення циклічного заперечення випливає з наступних властивостей:

$$(-)' = 0$$

$$(0)' = (+)$$

$$(+)' = -$$

5) Незмінність третього стану (0) при запереченні Лукасевича:

$$\bar{0} = 0$$

$$\overline{(a \wedge 0)} = \bar{a} \vee 0$$

6) Закон несумісності станів (аналог закону протиріччя в двійковій логіці):

$$S_a \wedge S_{a''} = (-)$$

$$S_{a'} \wedge S_{a''} = (-)$$

$$S_{a'} \wedge S_a = (-)$$

7) Закон виключеного четвертого (замість закону виключеного третього), він же закон повноти станів:

$$S_{a'}^{-} \vee S_a \vee S_a^{+} = (+), \textit{ або}$$

$$S_{a'} \vee S_a \vee S_{a''} = (+)$$

8) Трьохчленний закон Блейка-Порецкого:

$$a \vee S_{a'} \wedge b \vee S_a \wedge b = a \vee b, \textit{ або}$$

$$a \vee S_a^{-} \wedge b \vee S_a \wedge b = a \vee b$$

9) Закон тричленного склеювання:

$$a \wedge S_b^{-} \vee a \wedge S_b \vee a \wedge S_b^{+} = a, \textit{ або}$$

$$a \wedge S_{b'} \vee a \wedge S_b \vee a \wedge S_{b''} = a$$

10) Антиізотропність заперечення Лукасевича:

$$a \leq b \Longrightarrow \bar{a} \geq \bar{b}$$

## 1.6 Історія, сьогодення та майбутнє трійкової логіки

До кінця 1958 року була побудована перша трійкова машина, якій дали ім'я «Сетунь». «Сетунь» була відносно невелика для обчислювальних машин того покоління і займала площу 25-30 м2. Завдяки своїй витонченій архітектурі вона була здатна виконувати 2000-4500 операцій за секунду, мала оперативну пам'ять в 162 девятитритних осередки і запам'ятовуючим пристроєм на магнітному барабані ємністю 36-72 сторінки по 54 осередки кожна [13].

Технічні характеристики «Сетуні» [14]:

1) Тактова частота процесора – 200 кГц.
2) АЛП послідовний.
3) Числа, що опрацьовуються: з фіксованою комою; діапазони чисел, що можна позначити $3^{-16}$ <= | x | <1/2 $3^2$ і $3^{-7}$<= | x | <1/2 $3^2$.
4) Продуктивність – 4500 оп/сек.
5) ОЗП на феритових сердечниках – 162 дев'ятирозрядні осередки, час звернення 45 мкс.
6) ЗП – магнітний барабан ємністю 3888 дев'ятирозрядні осередки, швидкість обертання 6000 об/хв, час звернення 7,5 мс для обробки зони (групи з 54 дев'ятирозрядних осередків).
7) Споживана потужність – 2,5 кВт.
8) Пристрій введення: електромеханічний, 7 знаків/сек; фотоелектричний, 800 знаків в секунду, перфорована паперова п'ятипозиційна стрічка.
9) Пристрій виведення: телетайп, 7 знаків в секунду (одночасно виконує друк і перфорацію).

Після «Сетуні» було кілька експериментальних проектів, що здійснювалися ентузіастами (таких, наприклад, як американські Ternac і TCA2) [15], однак це були або дуже недосконалі машини, далекі від двійкових аналогів, або взагалі програмні емуляції на двійковому «залізі».

Основна причина полягає в тому, що використання в комп'ютерах трійчастих елементів поки не дає ніяких істотних переваг перед двійковими: випуск останніх налагоджений масово, вони простіше і дешевше за собівартістю. Навіть, якщо зараз був би побудований трійчастий комп'ютер, недорогий і подібний характеристиками з двійковим комп'ютером, він повинен бути повністю сумісний з ним. Уже розробники «Сетуні-70» зіткнулися з необхідністю забезпечити сумісність [13]: щоб обмінюватися інформацією з іншими університетськими машинами, довелося додати можливість читати з

перфострічок двійкові дані і при виведенні також конвертувати дані в двійковий формат.

Однак не можна сказати, що трійковий принцип в будуванні комп'ютерів – це безнадійний пережиток. В останнє десятиліття виникла необхідність в пошуку нових комп'ютерних технологій, і деякі з цих технологій лежать в області троїчності.

Одне з таких дослідницьких напрямків – пошук альтернативних способів збільшення продуктивності процесорів. Кожні 24 місяці число транзисторів в кристалі процесора збільшується приблизно вдвічі – ця тенденція відома як «закон Мура», і вічно тривати вона не може: масштаби елементів і зв'язків можна виміряти в нанометрах, і дуже скоро розробники зіткнуться з цілою низкою технічних складнощів.

Крім того, є і економічні міркування – чим менше, тим дорожче розробки і виробництво. І з якогось моменту виявиться дешевше пошукати альтернативні способи робити процесори могутнішими, ніж продовжувати гонку за нанометрами, – звернутися до технологій, від яких раніше відмовлялися як від нерентабельних. Перехід від однорідних кремнієвих структур до гетеро перехідних провідників, що складаються з шарів різних середовищ і здатні генерувати кілька рівнів сигналу замість звичних «так» і «ні», – це можливість підвищити інтенсивність обробки інформації без збільшення кількості елементів (і подальшого зменшення їх розмірів). При цьому від двозначної логіки доведеться перейти до багатозначних – тризначних, чотиризначних і т. д.

Інший напрямок, також націлений на збільшення продуктивності, – розробки в області асинхронних процесорів. Відомо, що забезпечення синхронності процесів в сучасних комп'ютерах неабияк ускладнює архітектуру і витрачає процесорні ресурси – до половини всіх транзисторів в чіпі працює на забезпечення цієї самої синхронності. Компанія Theseus Logic пропонує використовувати «розширену двійкову» (фактично – трійкову) логіку, де крім звичайних значень «істина» і «брехня» є окремий сигнал «NULL», який

використовується для самосинхронізації процесів [15]. У цьому ж напрямку працюють ще кілька дослідницьких груп.

Є і більш фантастичні напрямки, де виправдане використання тризначної логіки: оптичні та квантові комп'ютери.

## 1.7 Висновок

Отже з огляду на актуальність можна зробити висновок, що трійкова логіка є досить перспективним напрямком дослідження і розвитку комп’ютерних систем.

В роботі розглянуті переваги і недоліки трійкової логіки, її ефективність. Також досліджені трійкова симетрична система, що скорочує число необхідних варіантів арифметичних операцій, спрощує структуру пристроїв. Це дає значну економію часу, що витрачається на виконання операцій.

Також розглянуті окремі одномісні трійкові операції і їх алгебраїчні властивості. Розглянуті спроби побудови трійкових систем, а також дослідницькі напрямки використання систем з трьома значеннями істини.

# 2 ОГЛЯД АНАЛОГІВ СТРУКТУР ТРІЙКОВИХ ЕЛЕМЕНТІВ

На сьогоднішній день є декілька прикладів трійкових елементів, на базі яких можна реалізувати пристрої трійкових обчислювальних систем.

Реалізація трійкових пристроїв на основі порогової логіки є особливо економною і представляє шлях створення трійкових пристроїв, що здатні конкурувати з двійковими щодо кількості обладнання.

## 2.1 Пороговий елемент на магнітних елементах

Відомі порогові елементи трійкової логіки на магнітних елементах (ПЕТЛМ), на яких були побудовані ЕОМ «Сетунь» [16]. Характерною особливістю ПЕТЛМ є представлення трійкових значень -1, 0, +1 дискретними фіксованими струмами $–I_ф$, 0, $+I_ф$, виконання порогових функцій трійкової логіки шляхом алгебраїчного додавання струмів (ампервитків) у вхідних ланцюгах елементів, можливість розділяти трійчасті значення на їх двозначні компоненти, можливість складати трійчасті значення з їх двозначних компонентів. ПЕТЛМ виконані на основі електромагнітної техніки на магнітних сердечниках.

Феритова пам'ять Сетунь, відрізнялася від двійкових комп'ютерів, тому що кожна комірка пам'яті могла зберігати одне з трьох різних значень. Пам'ять представляла з себе матрицю феритових кілець. На кожному кільці було по три обмотки (рис.2.1). Це дозволяло записувати одне зі значень 0, 1, –1.Доступ до матриці послідовний, що значно знижувало швидкість читання/запису триту.

Недоліки порогового елементу трійкової логіки на магнітних елементах: побудований по застарілим технологіям на магнітних сердечниках, внаслідок чого має низьку швидкодію і не може бути реалізований засобами сучасних інтегральних напівпровідникових технологій.

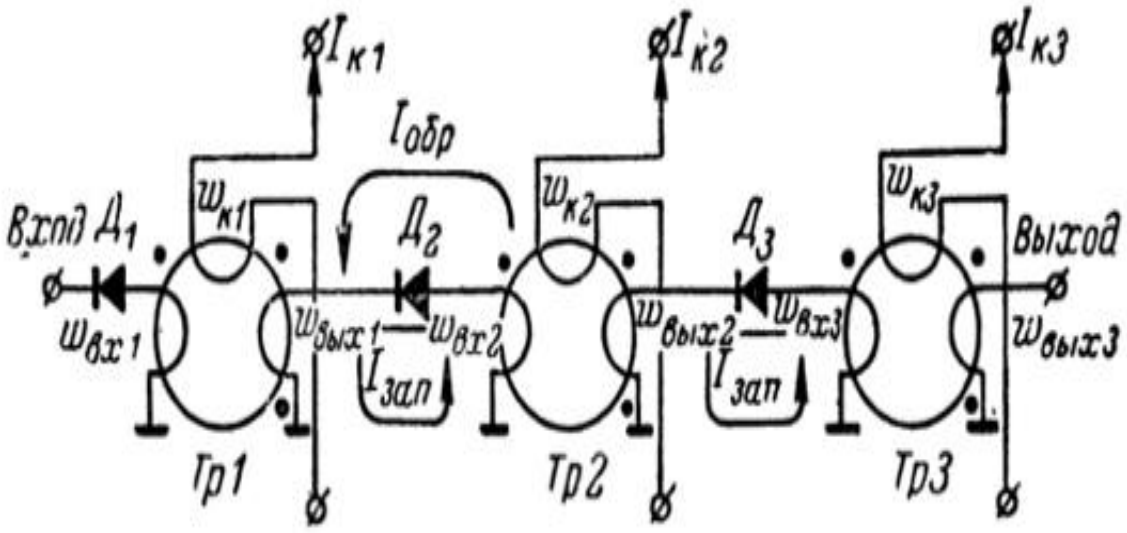



Рисунок 2.1 – Феррит-діодні логічні елементи

## 2.2 Пороговий елемент на біполярних транзисторах

Також відома реалізація трійчастого логічного елемента на біполярних комплементарних ненасичених транзисторах – порогового елемента трійкової логіки (ПЕТЛ) [17]. Елемент реалізується на основі двійкового ECL-елемента, схема якого доповнена її реплікою на комплементарних транзисторах.

ПЕТЛ (рис. 2.2) складається з блоку емітерних повторювачів (БЕП) і підключених до його виходів m блоків струмових перемикачів (СП.1…СП.m). БЕП реалізовано на двох повторювачах, відповідно на n-p-n і p-n-p транзисторах. Перший повторювач включений між спільною шиною і шиною живлення «–Е», другий – між шиною живлення «+Е» і спільною шиною. Кожний блок СП містить 2 перемикача струму.

Токи фіксованої величини $I_ф$ формуються двома джерелами струму, підключеними відповідно до шин живлення «+ Е» і «–Е».

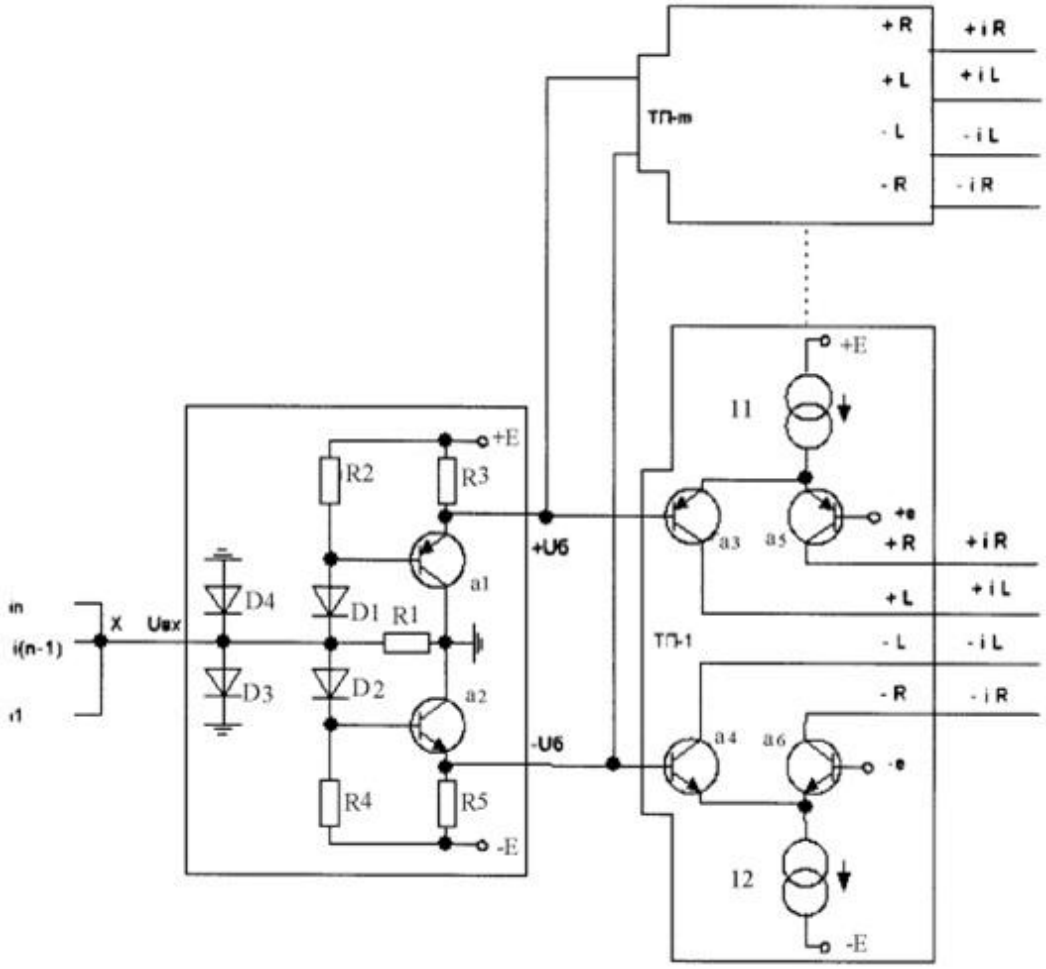



Рисунок 2.2 – Пороговий елемент трійкової логіки

На вхід X ПЕТЛ надходить n дискретних троїчних сигналів. Перетворення проводитися за допомогою функції $\mathrm{tersgn}(\mathrm{n}_{+1}, \mathrm{n}_{-1}, \mathrm{n}_0)$:

$$tersgn(n_{+1}, n_{-1}, n_0) = \begin{cases} +1, \text{якщо } (n_{+1} - n_{-1}) > 0 \\ 0, \text{якщо } (n_{+1} - n_{-1}) = 0 \\ -1, \text{якщо } (n_{+1} - n_{-1}) < 0 \end{cases}$$

де $\mathrm{n}_{+1}$ – число сигналів , поточні значення яких +1,

$\mathrm{n}_{-1}$ – число сигналів , поточні значення яких -1,

$\mathrm{n}_0$ – число сигналів , поточні значення яких 0.

Троїчна функція $tersgn(n_{+1}, n_{-1}, n_0)$ представлена на виходах елемента двома парами її двозначних компонент – дворозрядних двоїчних значень [+R, -R] і [+L, -L]. Відповідність значень компонент значенням функції $tersgn(n_{+1}, n_{-1}, n_0)$ показано в табл. 2.1 [17].

З'єднання виходів елемента один з одним і з виходами інших елементів і перетворення $tersgn(n_{+1}, n_{-1}, n_0)$ надає набір засобів, за допомогою яких можна здійснювати на запропонованих порогових елементах троїчної логіки різні логічні і технологічні функції.

Таблиця 2.1 – Таблиця значень функції $tersgn(n_{+1}, n_{-1}, n_0)$

| $tersgn(n_{+1}, n_{-1}, n_0)$ | +R | – R | +L | – L |
|---|---|---|---|---|
| + | + | 0 | 0 | – |
| 0 | 0 | 0 | + | – |
| – | 0 | – | – | 0 |

Недолік ПЕТЛ, те що визначається мала кількість порогів, внаслідок чого рівні «++» = «0+» і «--» = «0-» не розрізняються.

Однак, схемні та структурні рішення, використані в них і перевірені практично, можуть знайти застосування в сучасній цифровій техніці.

## 2.3 Пристрої на основі ПЕТЛ

На базі ПЕТЛ реалізовані пристрої трійкової логіки (рис. 2.3), такі як формувач тризначних констант, трійковий повторювач, нециклічнийй інвертор, схема «АБО», .

Пристрій для формування констант (ФК) має 2 виходи: +1 і –1 . При будь-якому вхідному троїчному значенні на виходах постійно виставлені +1 та –1 (дискретні струми фіксованої величини $+I_ф$ і $–I_ф$). Призначення ФК – задавати в потрібних точках схеми постійне потрійне значення: +1 і –1. Таким чином здійснюють прив'язку необхідної точки пристрою до заданого логічного значення.

Двовходова схема «АБО» має два виходи – прямий і інверсний.

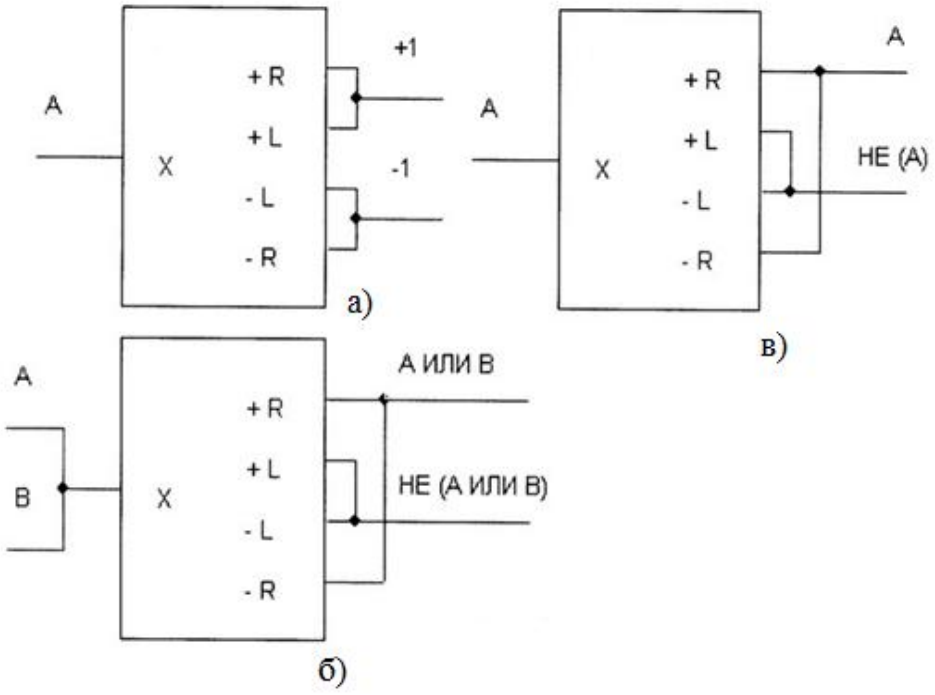



Рисунок 2.3 – Елементи, реалізовані на ПЕТЛ: а) Пристрій для формування констант; б) Повторювач і циклічний інвертор вхідної змінної; в) двовхідна схема «АБО»

## 2.4 Трійкові елементи на КМОП-транзисторах

Також були спроби створювати трійкові логічні елементи на традиційній елементній базі [21]. Але запропоновані схеми були малопривабливими, так як споживали енергію в статичному режимі.

Деякі джерела [7] пропонують використовувати лямбда-транзистор (λ-транзистор) для побудови трійкових логічних елементів (рис. 2.4). Він будується або на польових JFET транзисторах, або на польових транзисторах з вбудованим каналом. Проте λ-транзистор не здатний вирішити проблему енергоспоживання логічним елементом, так як при нульовій напрузі на базі (логічний 0), транзистор проводить досить значний струм.

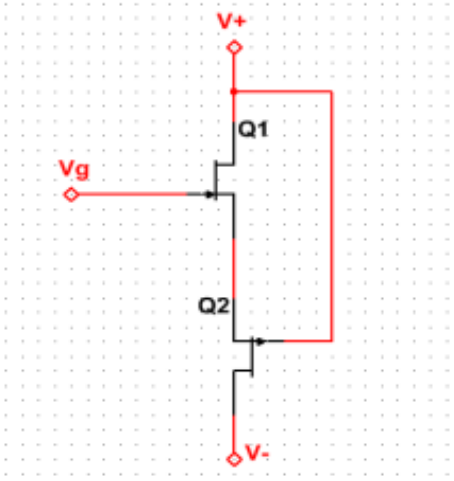



Рисунок 2.4 – Схема NLT λ-транзистора

Деяким розробникам вдалося поліпшити енергодинамічні характеристики КМОП-схем [22-27] – їх схеми були здатні працювати в симетричній трійковій системі.

Електрична схема базового одновходового елемента-повторювача (буфера) з 10 транзисторами представлена на рис.2.5 [28].

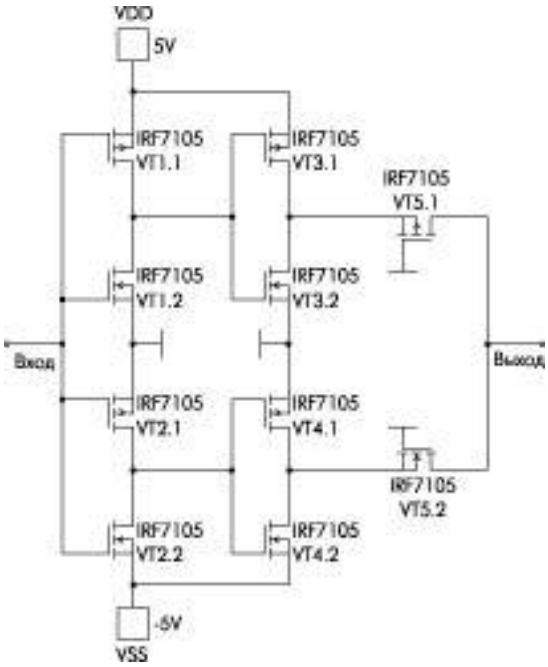



Рисунок 2.5 – Принципова електрична схема трійкового елемента на КМОП-транзисторах

Трійковий сигнал передається по одному дроту. Для кодування логічного нуля використовується нульова напруга щодо загальної точки («землі»), для кодування логічного «–1» – негативна напруга, а для логічної «+1» – позитивне. Таким чином, для живлення трійкових логічних елементів потрібне симетричне джерело живлення з трьома виходами.

Данний трійковий базовий елемент на КМОП-транзисторах, за енергодинамічними характеристиками і ступенем інтеграції перевершує існуючі аналоги і може бути реалізований на основі стандартної КМОП-технології. Проте, трійкові логічні елементи на КМОП-елементній базі приблизно в три рази поступаються по інтеграції і енергодінаміці двійковим аналогам.

На КМОП-транзисторах був реалізований трійковий інвертор [28]. Схема трійкового інвертора зображена на рис. 2.6.

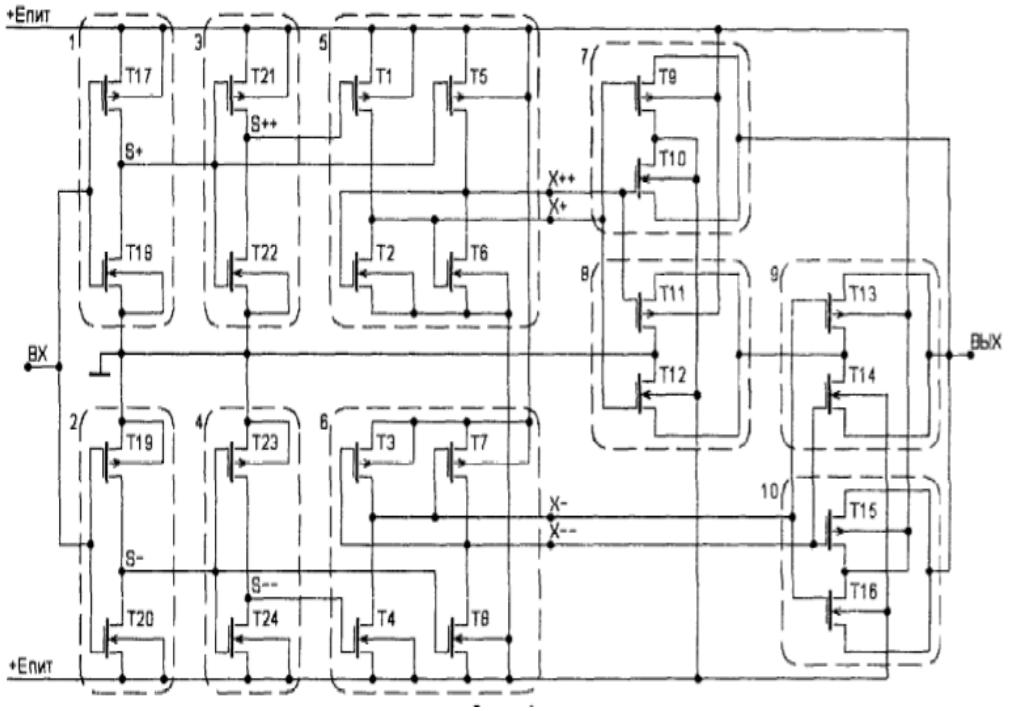



Рисунок 2.6 – Схема трійкового інвертора на КМОП-транзисторах

Значення трійкового коду «–1» і «+1» відповідають напруженням негативною і позитивною шин двополярного джерела живлення, а значення «0» – «землі». Входом трійкового інвертора є вхід схеми управління.

Схема управління формує сигнали для комутації виходу елемента з шинами джерела живлення згідно логічної функції інвертування. При заданому логічному стані на вході забезпечується замикання ключів так, що напруга на виході відповідає необхідному значенню трійкового коду.

За рахунок зменшення величини статичних струмів до значень струмів витоку МОП транзисторів здійснюється зниження статичної споживаної потужності в схемі трійкового КМОП інвертора.

## 2.5 Трійкові елементи на К-МОП-С транзисторах

На рис. 2.7 зображена схема трійкового елемента «НІ» [31].

Затвори МОП транзисторів з'єднані з вхідною шиною. На поверхні стоків і витоків n-МОП і p-МОП транзисторів розташовані відповідні електроди стоків, причому електрод витоку n-МОП транзистора підключений до загальної шині, а електрод витоку p-МОП транзистора підключений до шини живлення.

Дана електрична схема не дозволяє більше двох логічних рівнів сигналу (1 біта інформації).

Необхідно зазначити, що в даній схемі функціонально інтегруються області стоків та МОП конденсатори схеми.

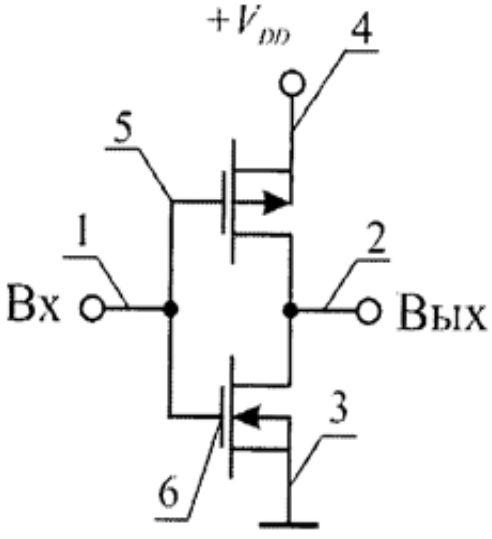



Рисунок 2.7 – Трійковий елемент «НІ» на К-МОП-С транзисторах

## 2.6 Висновок

Отже, усі розглянуті приклади реалізації трійкових елементів мають досить вагомі недоліки:

- порогові елементи на магнітних сердечниках використовують застарілу технологію, та не можуть бути реалізовані за допомогою сучасних інтегральних технологій;
- пороговий елемент на біполярних транзисторах визначає малу кількість порогів, в наслідок чого не розрізняє деякі рівні;
- пристрої на КМОП-транзисторах не мають стандартного підходу до реалізації трійкової логіки;

Таким чином усі ці елементи або не дозволяють повністю реалізувати трійкову логіку, або не мають загального підходу до її реалізації, або ускладнюють реалізацію трійкових пристроїв та їх структуру. Тому досить актуальною є  розробка стандартного підходу та методів синтезу трійкових елементів.

# 3 БАГАТОПОРОГОВИЙ ЕЛЕМЕНТ БАГАТОЗНАЧНОЇ ЛОГІКИ ТА ТРІЙКОВІ ЕЛЕМЕНТИ НА ЙОГО ОСНОВІ

Одним з рішень, направлених на подолання проблеми відсутності перевірених схемотехнічних рішень є використання багатопорогового елемента багатозначної логіки (БПЕБЛ) [35] для побудови інших трійкових елементів. Структурна схема БПЕБЛ зображена на рис. 3.1.

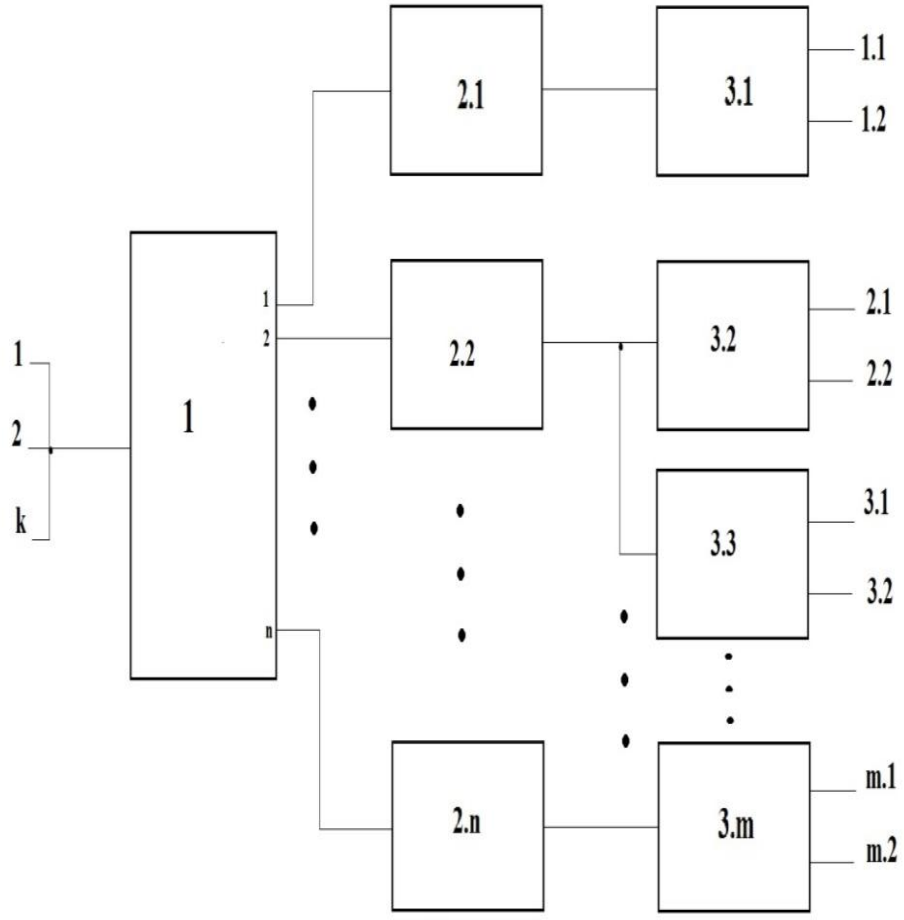



Рисунок 3.1 – Структурна схема БПЕБЛ

Перший блок – блок формування порогів (БФП), 2.1…2.n – емітерні повторювачі, 3.1…3.m – струмові перемикачі. На вхід блоку формування порогів поступають k дискретних струмових сигналів $I_j$ з попередніх елементів [36]. Вони можуть приймати одне з типових значень (наприклад,

для двійкової логіки таких значень буде два: $I_j = +1$, $I_j = 0$; для трійкової симетричної системи таких значень буде три: $I_j = +1$, $I_j = 0$, $I_j = –1$).

$k = k_{+1} + k_{-1} + k_0$,

де $k_{+1}$ – число сигналів, поточні значення яких +1,

$k_{-1}$ – число сигналів, поточні значення яких -1,

$k_0$ – число сигналів, поточні значення яких 0.

БФП в свою чергу формує n порогів . Його виходи подаються на входи емітерних повторювачів (ЕП) 2.1…2.n. Активні ЕП формують сигнали на підключених до них струмових перемикачів (СП) 3.1…3.m.

Особливості структури БПЕБЛ:

– Елемент оперує не з потенціальними, а зі струмовими значеннями сигналів, тому виходи БПЕБП можуть об’єднуватися у довільній кількості, але подаватися сигнал може тільки на вхід одного елемента.
– Можливість формування будь якої кількості порогів, які БПЕБЛ може розрізнити.

Від кількості порогів БФП залежить кількість рівнів вхідної змінної, які БПЕБП в змозі розрізнити й, відповідно, розрядність змінної або складність операцій, які можуть виконуватися.

Для того, щоб сформувати логіку функціонування даного елемента, потрібно об’єднати виходи СП у необхідній комбінації [37].

## 3.1 Чотирьохпорогова реалізація БПЕБЛ

Розглянемо приклад реалізації БПЕБЛ для трійової симетричної системи – його чотрьохпорогову версію [38].

Структурна схема чотирьохпорогового БПЕБЛ зображена на рис. 3.2 [37].

Електрична принципова схема реалізації БПЕБЛ з чотирма симетричними порогами показана на рис. 3.3.

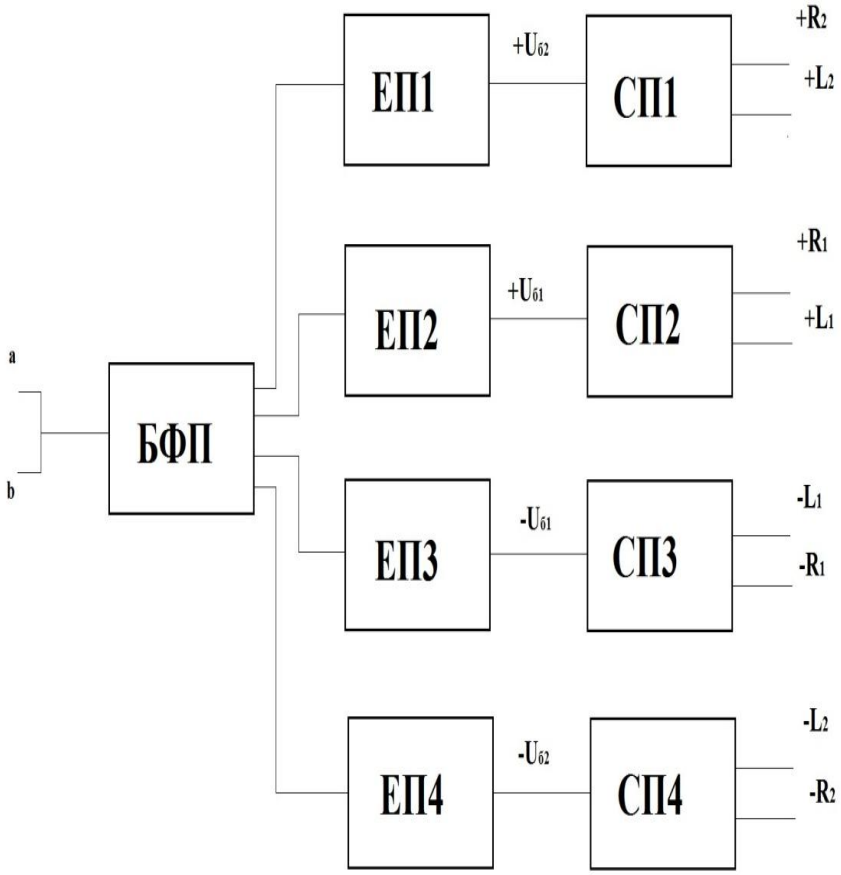



Рисунок 3.2 – Схема чотирьохпорогового БПЕБЛ

Значення напруг, що формуються на виходах ЕП і відповідно подаються на входи струмових перемикачів наведені у табл. 3.1.

В даній таблиці «1» – це активний сигнал (впливає на СП), «0» – неактивний сигнал (не впливає на СП) на виході відповідного ЕП [37].

Якщо сигнал на вході СП активний, на виході L відповідного СП формується струм, інакше – струм формується на виході R. СП у сукупності мають 8 виходів, комбінація яких може формувати необхідні логічні або арифметичні функції.

Виходи СП, в залежності від вхідних сигналів описуються функцією terlev відповідно значенням, наведеним у табл. 3.2.

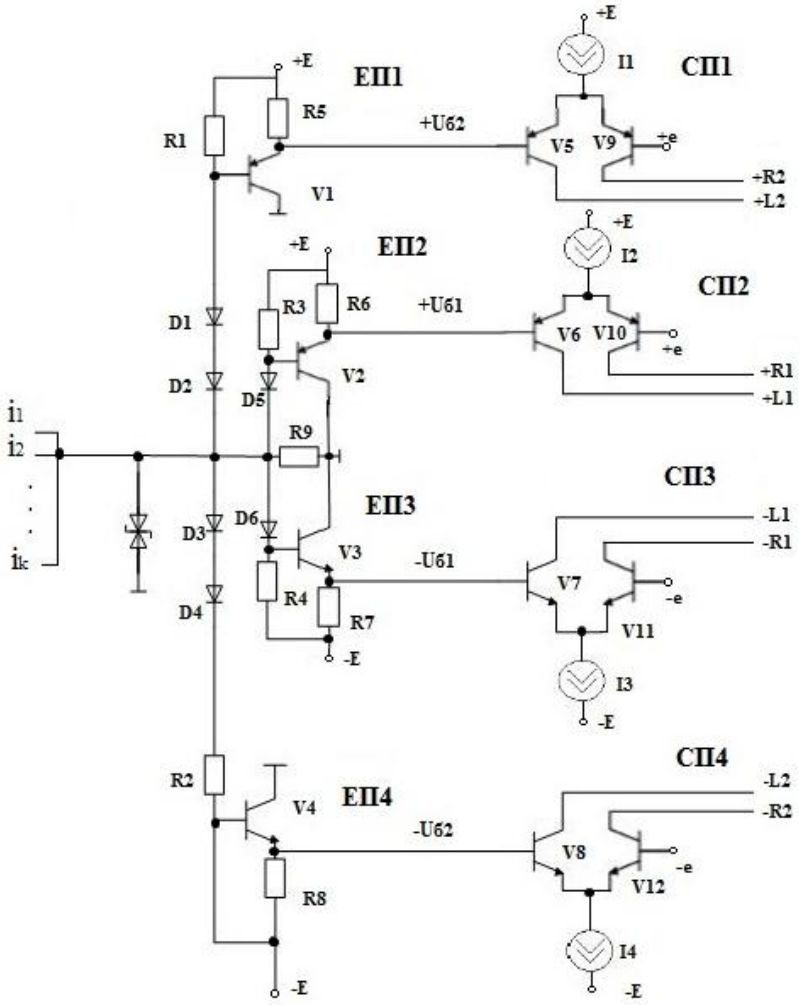



Рисунок 3.3 – Електрична принципова схема чотирьохпорогового БПЕБЛ

Таблиця 3.1 – Значення напруг на виходах ЕП1 – ЕП4

| Сума вхідних струмів | ЕП1($+U_{б2}$) | ЕП2 ($+U_{б1}$) | ЕП3 ($-U_{б1}$) | ЕП4 ($-U_{б2}$) |
|---|---|---|---|---|
| – – | 1 | 1 | 0 | 0 |
| – | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| + | 0 | 0 | 1 | 0 |
| + + | 0 | 0 | 1 | 1 |

Таблиця 3.2 – Значення функції terlev

| Сума вхідних струмів | Вихідні сигнали струмових перемикачів | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | СП1 | | СП2 | | СП3 | | СП4 | |
| terlev | $+R_2$ | $+L_2$ | $+R_1$ | $+L_1$ | –R1 | –L1 | $–R_2$ | $–L_2$ |
| – – | 0 | + | 0 | + | – | 0 | – | 0 |
| – | + | 0 | 0 | + | – | 0 | – | 0 |
| 0 | + | 0 | + | 0 | – | 0 | – | 0 |
| + | + | 0 | + | 0 | 0 | – | – | 0 |
| + + | + | 0 | + | 0 | 0 | – | 0 | – |

БПЕБЛ побудований шляхом модифікацій і доповнення ПЕТЛ, що описаний у попередньому розділі. Наведена реалізація БПЕБЛ для трійкової симетричної логіки, у порівнянні з ПЕТЛ, має 4 пороги вхідних сигналів (у прототипі – 2), розрізняє 5 рівнів (у прототипі – 3), має 8 вихідних сигналів (у прототипі – 4), що у сукупності дозволяє будувати більше різноманітних логічних і арифметичних пристроїв зі спрощеною реалізацією.

# 4 МЕТОД ПОБУДОВИ ТРІЙКОВИХ ОДНОМІСНИХ ФУНКЦІЙ

## 4.1 Універсальний пристрій для реалізації трійкових одномісних функцій

На основі багатопорогового елемента багатозначної логіки, побудований універсальний пристрій (УП) [42], за допомогою якого можна побудувати усі 27 трійкових унарних операцій. Його структура наведена на рис. 4.1.

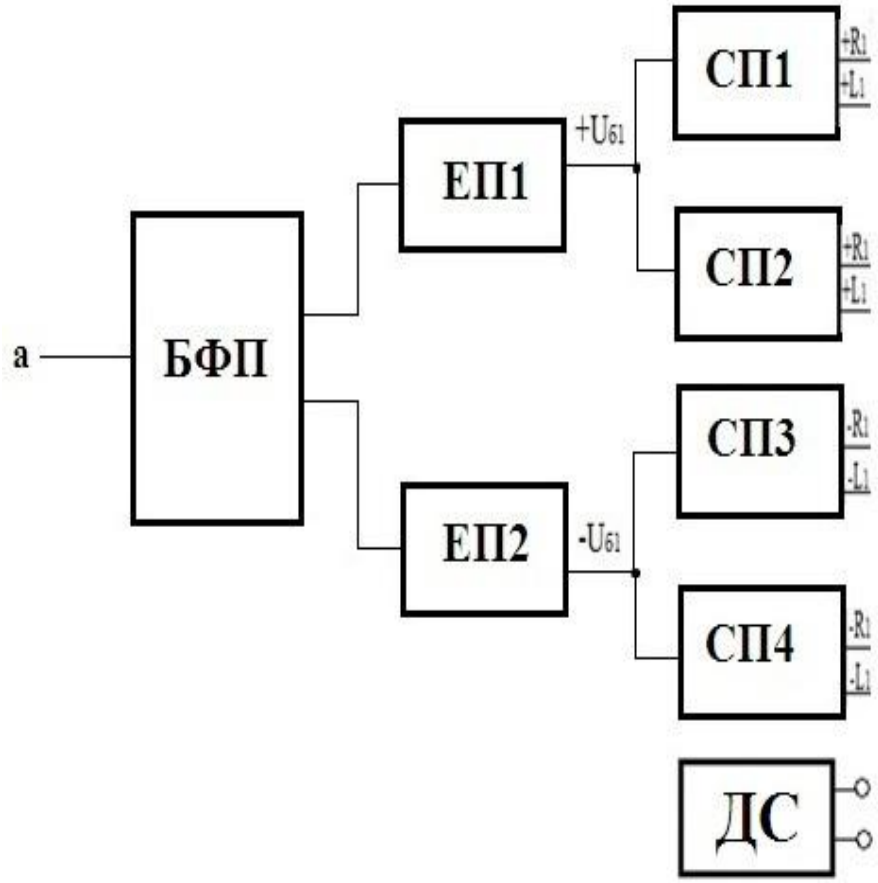



Рисунок 4.1 – Структурна схема універсального пристрою

УП складається з блоку формування порогів, двох емітерних повторювачів і чотирьох струмових перемикачів, а також джерела струму, що призначене для зсуву вихідного сигналу на «+1» або «–1».

УП має один вхід для трійкової змінної (+1, 0, –1) та 8 вихідних сигналів [42]. В залежності від поєднання виходів УП та константи, що формується джерелом струму (при необхідності) можливо реалізувати будь-яку трійкову одномісну функцію.

На вхід БФП 1 поступає трійкова змінна а (фіг.2). БФП формує два симетричні пороги. Виходи БФП подаються на емітерні повторювачі ЕП1, ЕП2, на виході яких формуються напруги, в залежності від вхідного струму (табл. 4.1).

Таблиця 4.1 – Напруги на виходах ЕП1-ЕП2

| Вхідний струм | ЕП1(+ $Uб_1$) | ЕП2(–$Uб_1$) |
|---|---|---|
| – | 1 | 0 |
| 0 | 0 | 0 |
| + | 0 | 1 |

Значення виходів струмових перемикачів в залежності від суми вхідних струмів наведені у табл. 4.2.

Таблиця 4.2 – Значення вихідних

| Сума вхідних струмів | Вихідні сигнали струмових перемикачів | | | |
|---|---|---|---|---|
| terlev | СП1, СП2 | | СП3, СП4 | |
| | $+R_1$ | $+L_1$ | $–R_1$ | $–L_1$ |
| – | 0 | + | – | 0 |
| 0 | + | 0 | – | 0 |
| + | + | 0 | 0 | – |

Поєднуючи виходи СП1 – СП4 та константу, сформовану джерелом струму, можна отримати необхідну трійкову унарну функцію.

## 4.2 Метод побудови трійкових унарних функцій

В ході дослідження, на основі універсального пристрою, запропонований метод, що дозволяє побудувати всі трійкові одномісні функції [43 – 44].

Для використання методу необхідно користуватися лише табл.4.2, вхідними значеннями і відповідними їм вихідними значеннями функції, що будується.

Метод побудови одномісних трійкових функцій на базі УП:

1. Обираємо набір (№ в таблиці істинності) функції, який будемо реалізовувати першим:
   - Це не нульове значення («+» або «–»), якого в функції менше.
   - Якщо ненульових значень однакова кількість, то обираємо те, яке відповідає ненульовому вхідному аргументу («+» або «–»).
   - Якщо є і те, і інше, то обираємо будь-яке (наприклад, відповідне вх. «–»).
2. Обираємо RL-функцію, яка реалізує значення з п.1, незалежно від отриманих значень при інших аргументах.
3. Обираємо набір (№ в таблиці істинності) функції, який будемо реалізовувати другим:
   - Це не нульове значення («+» або «–»).
   - Якщо два ненульових значень, то обираємо те, яке відповідає ненульовому вхідному аргументу («+» або «–»).
   - Інакше, обираємо те, що змінилось при виконанні п.2.
4. Обираємо RL-функцію (або її відсутність), яка реалізує значення з п.3 (можливо, їх буде дві, для компенсації дій функції з п.2).

   Якщо функція з п.4 змінює значення п.1:
   - на нульове, то необхідно додати RL-функцію з п.2.
   - інакше, необхідно додати RL-функцію з протилежним значенням у вибраному в п.1 номері функції з п.4.

5. Обираємо RL-функцію (або її відсутність), яка реалізує (або відповідним чином змінює) значення, що залишилось.

Якщо функція з п.5 змінює значення п.1:

- на нульове, то необхідно додати RL-функцію з п.2.
- інакше, необхідно додати RL-функцію з протилежним значенням у вибраному в п.1 номері функції з п.5.

Якщо функція з п.5 змінює значення п.3:

- на нульове, то необхідно додати RL-функцію з п.4.
- інакше, необхідно додати RL-функцію з протилежним значенням у вибраному в п.3 номері функції з п.5.

6. Оптимізація. Якщо в отриманому наборі функцій є такі, які в сумі дають константу («+1» чи «–1») на всіх наборах вхідних аргументів, то їх можна замінити на константу «+» або «–».

В результаті роботи методу, отримуємо таблицю, що демонструє, які виходи струмових перемикачів треба об’єднати для того, щоб отримати необхідну функцію.

Розглянемо приклад побудови функції («–1», «0», «–1»):

1) Згідно методу, першим номером для реалізації буде вихідна «–1», що відповідає вхідній «–1».
2) Обираємо RL-функцію. В даному випадку, це буде «$–R_1$», що відповідає значенням «–1», «–1», «0». Додаємо цю функцію в таблицю результатів (табл. 4.3).

Таблиця 4.3 – Таблиця результатів після другого пункту

| a | $–R_1$ | ∑ |
|---|---|---|
| –1 | –1 | –1 |
| 0 | –1 | –1 |
| +1 | 0 | 0 |

3) Другий номер – вихідна «–1», що відповідає вхідній «+1».

4) RL-функція, що реалізує обраний номер – «$–L_1$» («0», «0», «–1»). Додаємо її до таблиці результатів (табл. 4.4).

Таблиця 4.4 – Таблиця результатів після четвертого пункту

| a | $–R_1$ | $–L_1$ | ∑ |
|---|---|---|---|
| –1 | –1 | 0 | –1 |
| 0 | –1 | 0 | –1 |
| +1 | 0 | –1 | –1 |

5) Залишився вихідний «0», що відповідає вхідному «0». Функція, що відповідним чином змінює треті обраний набір – «$+R_1$» («0», «+1», «+1»). Так як обрана RL-функція змінила значення з п.3 на «0», то необхідно ще додати функцію з п.4, тобто «$–L_1$» («0», «0», «–1»). В результаті отримуємо табл. 4.5.

Таблиця 4.5 – Таблиця результатів після п’ятого пункту

| a | $–R_1$ | $–L_1$ | $+R_1$ | $–L_1$ | ∑ |
|---|---|---|---|---|---|
| –1 | –1 | 0 | 0 | 0 | –1 |
| 0 | –1 | 0 | +1 | 0 | 0 |
| +1 | 0 | –1 | +1 | –1 | –1 |

6) В табл. 4.5 є набори, що в сумі дають константу «–1» на всіх наборах вхідних аргументів. Це $(–L_1)+(–R_1)$. Згідно методу, ці RL-функції можна замінити на константу «–1», що формуватиметься джерелом струму (табл. 4.6).

Таблиця 4.6 – Таблиця результатів після оптимізації

| a | $+R_1$ | $–L_1$ | Const(–1) | ∑ |
|---|---|---|---|---|
| –1 | 0 | 0 | –1 | –1 |
| 0 | +1 | 0 | –1 | 0 |
| +1 | +1 | –1 | –1 | –1 |

З огляду на отриману таблицю, можна одразу побачити, які виходи струмових перемикачів необхідно об’єднати та чи потрібно задіяти джерело струму, для того щоб синтезувати обрану функцію.

В результаті можна побудувати структурну схему функції (рис. 4.2).

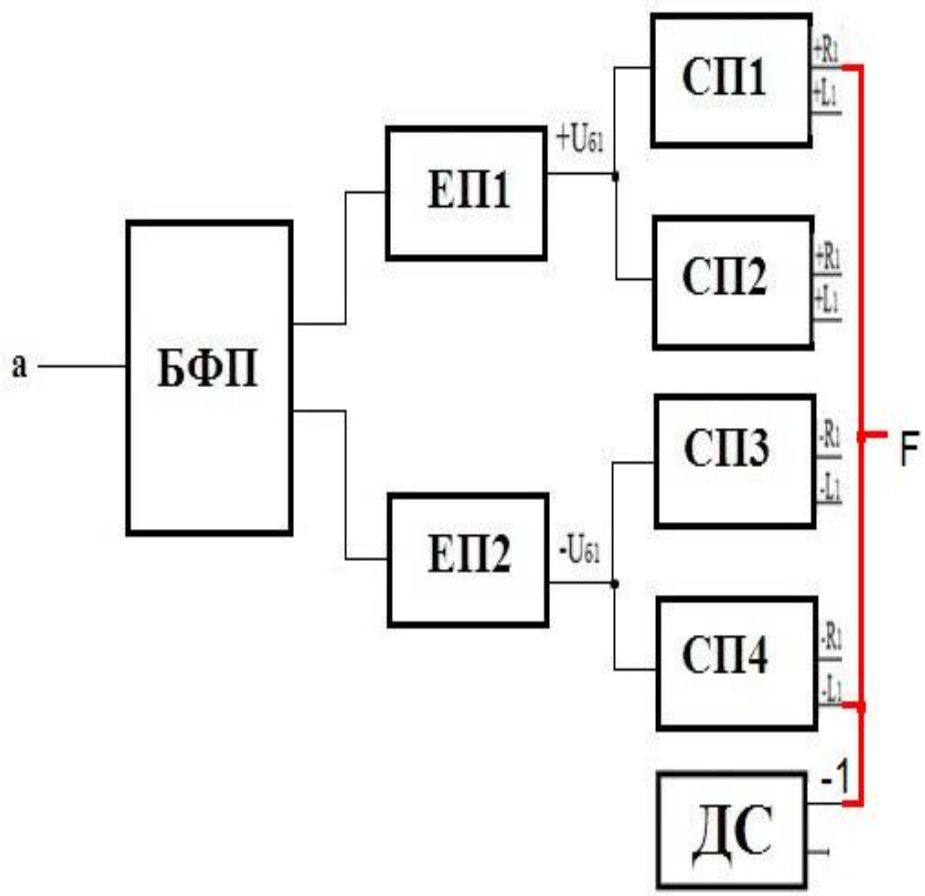



Рисунок 4.2 – Структурна схема функції –1,0,+1

4.3 Програмна реалізація методу побудови трійкових одномісних функцій

На основі запропонованого методу було розроблене програмне забезпечення, що дозволяє синтезувати всі трійкові унарні функції. Програмний код наведений у додатку А.

Головне вікно програми зображене на рис. 4.3.

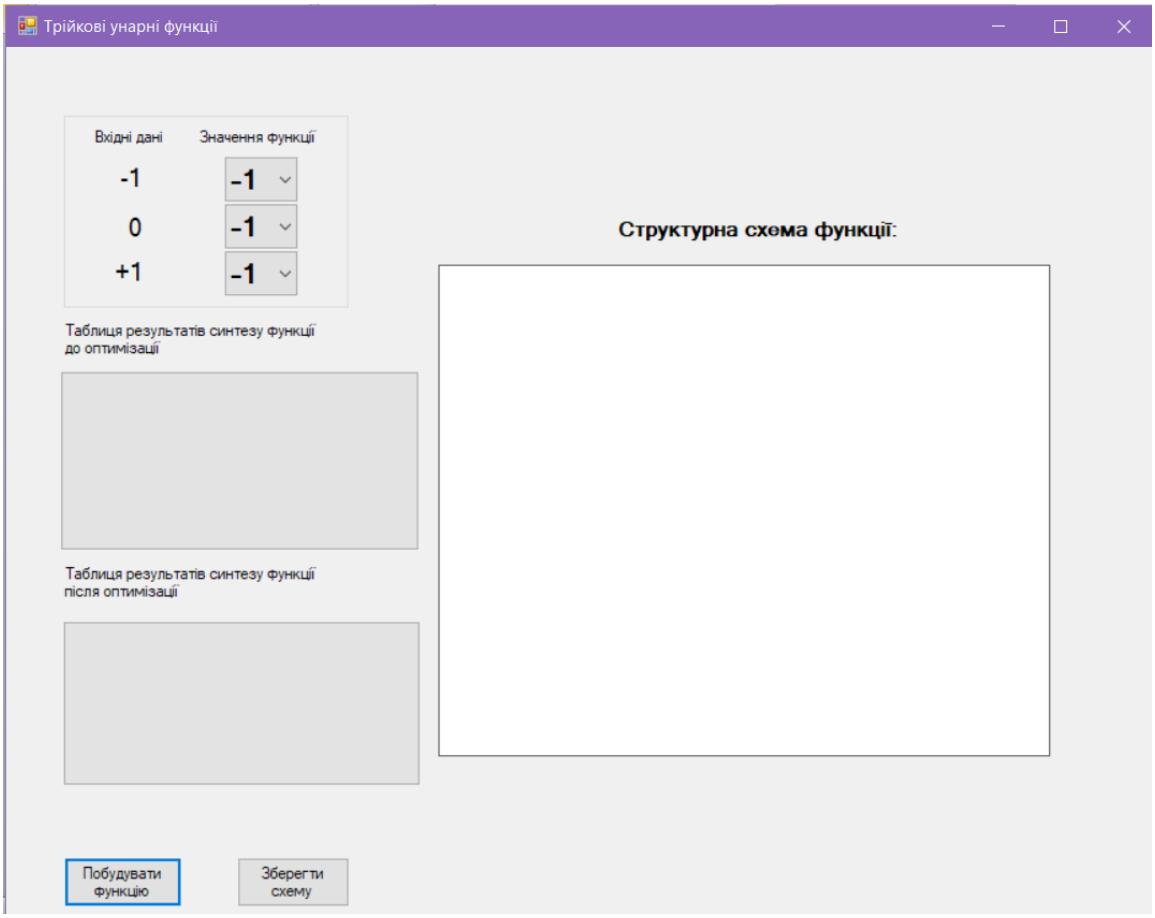



Рисунок 4.3 – Головне вікно програми синтезу трійкових унарних функцій

Для початку необхідно обрати вихідні значення функції відповідним їм вхідним аргументам, наприклад для функції 0, –1, –1 (рис. 4.4).

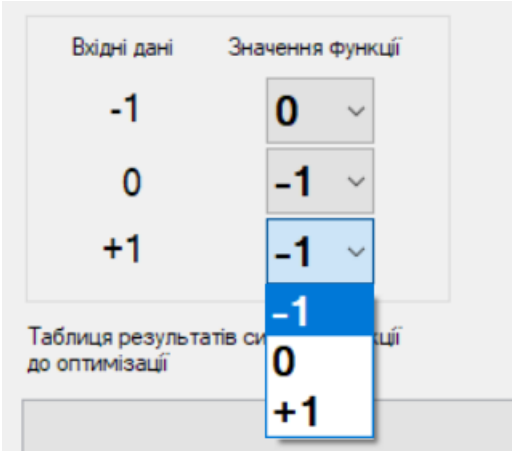



Рисунок 4.4 – Вибір значень функції 0, –1, –1 для синтезу

Далі необхідно натиснути кнопку «Побудувати функцію». В результаті отримуємо таблицю синтезу функції до оптимізації (рис. 4.5) та після оптимізації (рис. 4.6). RL-функції записані у порядку їх додавання до таблиці результатів синтезу.

Таблиця результатів синтезу функції до оптимізації

| | | -L1 | -R1 | +L1 | Σ |
|---|---|---|---|---|---|
| a | | | | | |
| - | | 0 | -1 | 1 | 0 |
| 0 | | 0 | -1 | 0 | -1 |
| + | | -1 | 0 | 0 | -1 |

Рисунок 4.5 – Таблиця синтезу функції до оптимізації

Таблиця результатів синтезу функції після оптимізації

| | | +L1 | Const(-1) | Σ | |
|---|---|---|---|---|---|
| a | | | | | |
| -1 | | 1 | -1 | 0 | |
| 0 | | 0 | -1 | -1 | |
| +1 | | 0 | -1 | -1 | |

Рисунок 4.6 – Таблиця синтезу функції після оптимізації

Також результатом роботи програми є структурна схема функції, що будується, в нашому випадку 0, –1, –1 (рис. 4.7). Програма дає можливість зберегти синтезовану структуру у вигляді картинки.

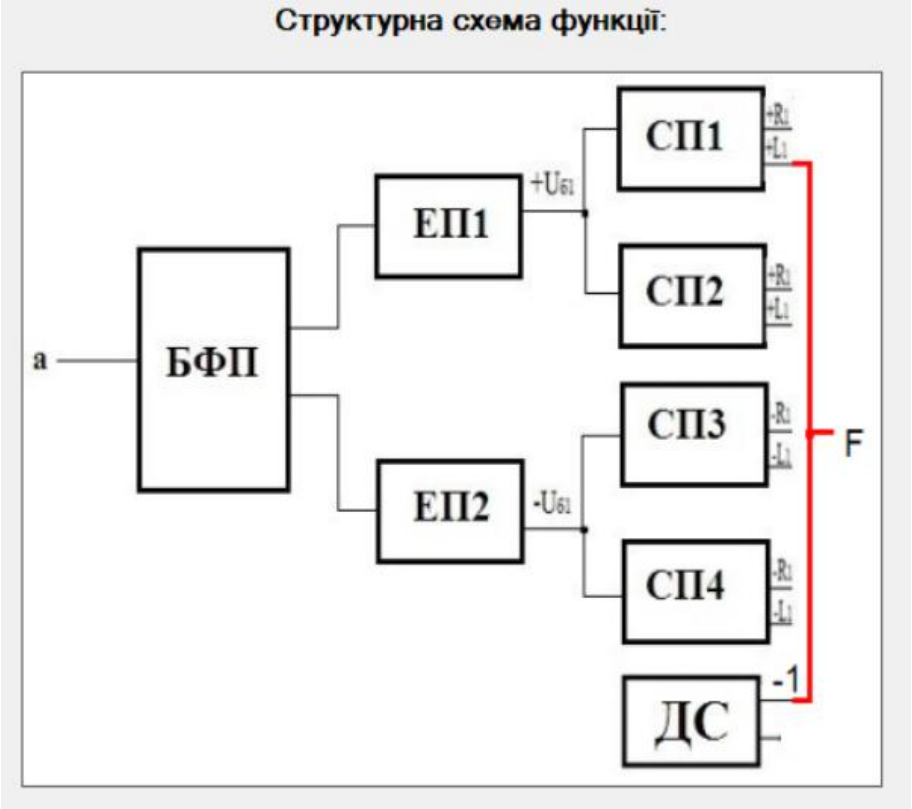



Рисунок 4.7 – Синтезована структурна схема функції 0, –1, –1

Усі синтезовані структури двадцяти семи трійкових унарних функцій наведені у додатку Б.

## 4.4 Висновок

Отже, на основі багатопорогового елемента багатозначної логіки, запропонована структура універсального пристрою для побудови трійкових унарних функцій.

Для УП, вперше запропоновано метод синтезу, що дозволяє побудувати будь-яку трійкову одномісну функцію та його програмна реалізація. В роботі отримано всі структурні схеми для можливих двадцяти семи трійкових одномісних функцій.

# ВИСНОВОК

Отже, в роботі проаналізована трійкова логіка, досліджені існуючі аналоги трійкових елементів та способи їх реалізації. Усі вони мають свої недоліки:

- пороговий елемент трійкової логіки на магнітних сердечниках побудований по застарілим технологіям, внаслідок чого має низьку швидкодію;
- пороговий елемент на біполярних транзисторах визначає малу кількість порогів, внаслідок чого деякі рівні («++» = «0+» і «--» = «0-») не розрізняються;
- трійкові логічні елементи на КМОП-елементній базі приблизно в три рази поступаються по інтеграції і енергодінаміці двійковим аналогам та не мають загального підходу реалізації.

В результаті проведених досліджень та аналізу за основу в роботі взята структура багатопорогогового елемента багатозначної логіки та розглянута його двопорогова та чотирьохпорогова реалізація, що пропонується використовувати для трійкової логіки.

В роботі на основі БПЕТЛ запропановано метод побудови трійкових унарних функцій, універсальний пристрій для реалізації трійкових унарних (одномісних) функцій, та реалізовано інформаційну систему (програму), що дозволило синтезувати можливі структури.

Порівняння синтезованих елементів з існуючими аналогами і прототипами показало, що запропоновані структури простіші та потребують меншої кількості елементів, їх можно будувати та аналізувати з єдиних позицій та подходів.

Таким чином, обсяг результатів, нові структури та метод синтезу дозволяють зробити висновок, що поставлені задачі вирішені та мета роботи досягнута.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1) Википедия – свободная энциклопедия, «Троичная логика», [Електронний ресурс] – Режим доступу: https://neerc.ifmoru/wiki/index.php?title=%D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%B0#.D0.9E.D0.B4.D0.BD.D0.BE.D0.BC.D0.B5.D1.81.D1.82.D0.BD.D1.8B.D0.B5_.D0.BE.D0.BF.D0.B5.D1.80.D0.B0.D1.86.D0.B8.D0.B8
2) Shannonc. E.A Symmetrical notation for numbers. – "The American Mathematical Monthly", 1950, 57, N 2, p, 90 – 93
3) Википедия – свободная энциклопедия, «Троичная система счисления», [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/%D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0
4) Гашков С. Б. § 11. Д. И. Менделеев и троичная система // Системы счисления и их применение. — М.: МЦНМО, 2004.
5) Выдержки из научного отчета № 24-ВТ (378) Брусенцова Н.П. «Использование троичного кода и трехзначной логики в цифровых машинах», Москва – 1969 г.
6) Карцев А. А. Арифметические устройства электронных цифрових машин. М., Физматгиз, 1958.
7) Кушнеров А., Троичная цифровая техника. Ретроспектива и современность. Университет им. Бен-Гуриона Беэр-Шева, Израиль, 28.10.05
8) Брусенцов Н.П. Об использовании троичного кода и трехзначной логики в цифровых машинах, Москва – 1969 г.

9) Брусенцов Н.П., «Заметки о троичной цифровой технике» [Електронний ресурс] – Режим доступу: http://www.computer-museum.ru/histussr/12-1.htm
10) Википедия – свободная энциклопедия, «Троичные функции», [Електронний ресурс] – Режим доступу: https ://ru.wikipedia.org /wiki/ %D0%A2%D1%80%D0%BE%D0%B8%D1%87%D0%BD%D1%8B%D0%B5_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8
11) Википедия – свободная энциклопедия, «Модальная логика», [Електронний ресурс] – Режим доступу: https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BB%D0%BE%D0%B3%D0%B8%D0%BA%D0%
12) Трёхзначная логика. - Жизнь сквозь решето Сети, Електронний ресурс] – Режим доступу: http://arvi.livejournal.com/144849.html
13) Криницкий Н. А., Миронов Г. А., Фролов Г. Д., Программирование, под ред. М. Р. Шура-Бура, Государственное издательство физико-математической литературы, Москва, 1963 (Глава 10 Программно-управляемая машина Сетунь).
14) Википедия – свободная энциклопедия, «Сетунь(комп'ютер)», [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org /wiki/%D0%A1%D0%B5%D1%82%D1%83%D0%BD%D1%8C(%D0%BA%D0%BE%D0%BC%D0%BF'%D1%8E%D1%82%D0%B5%D1%80)
15) Троичный компьютер: Да, нет, может быть: Логика // «Популярная механика». [Електронний ресурс] – Режим доступу: http://www.popmech.ru/technologies/11918-troichnyy-kompyuter-da-net-mozhet-byt-logika/
16) Брусенцов Н.П. Пороговая реализация трехзначной логики электромагнитными средствами. // Вычислительная техника и вопросы кибернетики. Вып.9. - М.: Изд-во Моск. ун-та, 1972. С.3-35.

17) Пат. 2394366 Россия, МПК (2006.01) H03K19/00. Пороговый элемент троичной логики и элементы на его основе. Опубл. 10.07.2010.
18) Пат. 2461122 Россия, МПК (2006.01) H03K19/00. Узел троичной схемотехники и дешифраторы-переключатели на его основе. Опубл. 10.09.2012
19) Пат. 2510129 Россия, МПК (2006.01) H03K19/00. Троичный Д-триггер (варианты). Опубл. 20.03.2014
20) Пат. 2015130589 Россия, МПК (2006.01) H03K19/00. Троичный реверсивный регистр сдвига. Опубл. 27.05.2016
21) Пат. 1563821 Великобритания. Ternary logic circuits with CMOS Devices (Троичные логические схемы с КМОП-приборами) / Hussein Talaat Mouftah.Заявл. 19.03.1976, опубл. 02.04.1980.
22) А.с. 1707757 СССР. Троичный дизъюнктор на МОП-транзисторах / Кушниренко А.Н.. Заявл. 27.07.1987, опубл. 23.01.1992.
23) Пат. 19832101 Германия. Realisierung Ternärer Grundschaltungen in CMOS Technologie (Реализация троичных базовых схем в КМОП-технологии), Josef von Stackelberg. Заявл. 17.07.1998, опубл. 27.01.2000
24) Пат. 2005080257 Япония. Схема КМОП-драйвера, а также схема КМОП-инвертора / Хидэки Фукуда. Заявл. 04.09.2003, опубл. 24.03.2005.
25) Пат. 2278469 Р.Ф. Логическое устройство "ИЛИ" / Попов Н.Д., Лукашенко В.А. Заявл. 01.11.2004, опубл. 20.06.2006.
26) Пат. 2287895 Р.Ф. Логическое устройство "Отрицание" (варианты) / Попов Н.Д., Лукашенко В.А. Заявл. 01.11.2004, опубл. 20.06.2006.
27) Пат. 2281605 Российская Федерация. Логическое устройство "И" / Попов Н.Д., Лукашенко В.А. Заявл. 01.11.2004, опубл. 10.08.2006.
28) П.Ившин, С.Леготин, В.Мурашёв. Базовые троичные логические элементы. Снижение энергопотребления. [Електронний ресурс] – Режим доступу: http://www.electronics.ru/journal/2010/4

29) Пат. 2373639 C1 Российская Федерация. Троичный инвертор на КМОП транзисторах / Морозов Д.В., Пилипко М.М., Коротков А.С. Заявл. 23.04.2008, опубл. 20.11.2009.

30) Пат. 2 468 510 C1 Российская Федерация. ТРОИЧНЫЙ К-МОП-С ЛОГИЧЕСКИЙ ЭЛЕМЕНТ «ИЛИ-НЕ» / Мурашев В.Н., Забеднов П.В., Ившин П.А., Баранов А.Н., Леготин С.А. Заявл. 16.09.2011, опубл. 27.11.2012.

31) Пат. 2 481 701 C2 Российская Федерация. ТРОИЧНЫЙ К-МОП-С ЛОГИЧЕСКИЙ ЭЛЕМЕНТ «НЕ»/ Мурашев В.Н., Забеднов П.В. Заявл. 21.07.2011, опубл. 27.01.2013.

32) Ji-Zhong Shen, Mao-Qun Yao, Xie-Xiong Chen, Hua-Hua Chen Ternary BiCMOS circuit structure and its design at switch level // Proc. 5th Int. Conf. ASIC. - Oct V P

33) Троичный триггер ("flip-flap-flop") [Електронний ресурс] – Режим доступу: http://www.goldenmuseum.com/1411FlipFlap_rus.html

34) Свободная энциклопедия, «Троичный триггер» [Електронний ресурс] – Режим доступу: http://dic.Academic.ru/dic.nsf/ru wiki/451943

35) Левчук В.В., Малахов В.П., Гунченко Ю.О. Багатопороговий пристрій для реалізації логічно-арифметичних елементів багатозначних систем числення // Тези доповідей XIII Міжнародної науково-практичної конференції "Військова освіта і наука: сьогодення та майбутнє" , Київ, 24 листопада 2017/ за заг. редакцією І.В. Толока. – К. : ВІКНУ, 2017. – 51 с

36) Левчук В.В., Гунченко Ю.О., Ємельянов П.С., ЕЛЕМЕНТИ НА ОСНОВІ ТРІЙКОВОЇ ЛОГІКИ //Тези доповідей чотирнадцятої всеукраїнської конференції студентів і молодих науковців «Інформатика, інформаційні системи та технології», Одеса, 14 квітня 2017р. – Одеса 2017. – с.112-114.

37) Пат. UA 118735 Україна, МПК (2017.01) H03K19/00. Багатопорговий елемент багатозначної логіки / Гунченко Ю.О. Заявка 23.03.2017, опубл. 28.08.2017.

38) Левчук В.В., Гунченко Ю.О., Ємельянов П.С., Чотирьох пороговий елемент трійкової логіки // II Всеукраїнська науково-практична конференція студентів, аспірантів та молодих вчених «ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В МОДЕЛЮВАННІ », Миколаїв, 23-24 березня 2017р. – Миколаїв 2017. – с.140.

39) Пат. UA 122996 ТРІЙКОВИЙ ПІВСУМАТОР НА ОСНОВІ БАГАТОПОРОГОВОГО ЕЛЕМЕНТА БАГАТОЗНАЧНОЇ ЛОГІКИ / Гунченко Ю.О., Лєнков С.В., Малахов Є.В., Шворов С.А., Устимчук В.В., Лукін В.Є., Межуєв В.І., Лєнков Є.С, Левчук В.В., Заяв. 16.06.2017, опубл. 12.02.2018.

40) Levchuk V., GUNCHENKO S., TERNARY HALF-ADDER // 55 Confrence of Stundent's Scientific Cirles, Krakow, 10 may 2018. – c. 264.

41) Гунченко Ю., Левчук В., Кузніченко С., Олейник О., Концепція Побудови Логічних і Арифметичних Пристроїв для Багатозначних Логік // Міжнародна науково-практична конференція "Інформаційні технології та комп'ютерне моделювання", Івано-Франківськ, 14-15 травня 2018 р. – Івано-Франківськ 2018. – с.216.

42) Левчук В.В., Гунченко Ю.О., Ємельянов П.С., ПРИСТРІЙ ДЛЯ ПОБУДОВИ ТРІЙКОВИХ УНАРНИХ ФУНКЦІЙ // II Всеукраїнська науково-практична конференція «ПРИКЛАДНА ГЕОМЕТРІЯ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ в моделюванні об'єктів, явищ і процесів», Миколаїв, 18-20 жовтня 2017р. – Миколаїв 2017. – с.102

43) Левчук В.В., Гунченко Ю.О., МЕТОД ПОБУДОВИ ТРІЙКОВИХ ОДНОМІСНИХ ФУНКЦІЙ //Тези доповідей п'ятнадцятої всеукраїнської конференції студентів і молодих науковців «Інформатика, інформаційні системи та технології», Одеса, 27 квітня 2018р. – Одеса 2018. – с.16-18.

44) Levchuk V., Emelyanov P., METHOD OF BUILDING TERNARY LOGIC ELEMENTS // 54 Confrence of Stundent's Scientific Cirles, Krakow, 11 may 2017. – c. 259.

# ДОДАТОК А

Код програмної реалізації методу синтезу трійкових унарних функцій

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public bool flag = false;
        public string[] word = { "+R1", "+L1", "-R1", "-L1" };
        public string[] words = new string[9];
        public string[] Owords = new string[9];
        public int n = 0;
        public int value = 0;
        bool[] two = { false, false, false };
        bool[] three = { false, false, false };
        public int[,] terlev = { { 0,1,-1,0},
                                 { 1,0,-1,0},
                                 {1,0,0,-1 }};
        public int[,] newt = new int[3, 9];
        public int[,] Onewt = new int[3, 9];
        public int[] mas = { 200, 200, 200 };
        public int len = 0;
        public int[] newy = new int[9];
        public int[] Oy = new int[9];
        public int kx = 460;
        Graphics g;
        public  int[,]  ky={{30,  118},{49,  137},{195,  281},{
215,301}};
        public void Draw()
        {
            var bitmap = new Bitmap("YY.jpg");
            picture.Image = bitmap;
            g = Graphics.FromImage(bitmap);
            Pen p = new Pen(Color.Red, 3);
            int c = 0;
            int mid = 0;
            int nx = kx + 10;
            int min = 400;
            int max = 0;
            for (int i = 0; i < Oy.Length; i++)
            {
                if (Oy[i] != 0)
```

```
                    {
                        if (Oy[i] < min)
                        {
                            min = Oy[i];
                        }
                        if (Oy[i] > max)
                        {
                            max = Oy[i];
                        }
                        g.DrawLine(p, kx, Oy[i], nx + 2, Oy[i]);
                        c++;
                    }
                }
                g.DrawLine(p, nx, min, nx, max);
                mid = Math.Abs((max - min)) / 2;
                g.DrawLine(p, nx, min + mid, nx + 15, mid + min);
                if (c > 1)
                    g.FillRectangle(Brushes.Red, nx - 2, mid + min
- 1, 5, 5);
                Font drawFont = new Font("Arial", 12);
                SolidBrush         drawBrush         =         new
SolidBrush(Color.Black);
                g.DrawString("F",  drawFont,  drawBrush,  nx  +  18,
mid + min - 5);
                for (int i = 0; i < c - 1; i++)
                {
                    if (Oy[i] != min && Oy[i] != max)
                        g.FillRectangle(Brushes.Red, nx - 2, Oy[i]
- 1, 5, 5);
                }
                if (value != 0)
                {
                    g.DrawString(value.ToString(),       drawFont,
drawBrush, 445, 320);
                }
            }

            public void Opt()
            {
                bool opt = false;

                int x = -1, y = -1;
                int newj = 0, wi = 0;
                for (int i = 0; i < words.Length; i++)
                {
                    if       (words[i]       ==       "+R1"       &&
words.Contains("+L1"))
                    {
                        x = i;
                        y = Array.IndexOf(words, "+L1");
                        value = 1;
                        opt = true;
                    }
                    else
```

```
                    if (words[i] == "-R1" && words.Contains("-
L1"))
                    {
                        x = i;
                        y = Array.IndexOf(words, "-L1");
                        value = -1;
                        opt = true;
                    }
                }
                if (opt)
                {
                    for (int i = 0; i < len; i++)
                    {
                        if (i != x && i != y)
                        {
                            Owords[wi] = words[i];
                            Oy[wi] = newy[i];
                            wi++;
                        }
                    }
                    for (int i = 0; i < 3; i++)
                    {
                        for (int j = 0; j < len; j++)
                        {
                            if (j != x && j != y)
                            {
                                Onewt[i, newj] = newt[i, j];
                                newj++;
                            }

                        }
                        newj = 0;
                    }
                    if (value == 1)
                    {
                        Owords[len - 2] = "Const(1)";
                        Onewt[0, len - 2] = 1;
                        Onewt[1, len - 2] = 1;
                        Onewt[2, len - 2] = 1;

                    }
                    if (value == -1)
                    {
                        Owords[len - 2] = "Const(-1)";
                        Onewt[0, len - 2] = -1;
                        Onewt[1, len - 2] = -1;
                        Onewt[2, len - 2] = -1;

                    }
                    Oy[len - 2] = 343;

                }
                else
                {
```

```
            for (int i = 0; i < words.Length; i++)
            {
                Owords[i] = words[i];
                Oy[i] = newy[i];
            }
            for (int i = 0; i < 3; i++)
                for (int j = 0; j < words.Length; j++)
                    Onewt[i, j] = newt[i, j];
        }
    }
    public int RL(int p, int x)
    {
        for (int i = 0; i < 4; i++)
        {
            if (mas[p] <= 5)
            {
                if (terlev[p, i] + mas[p] == x)
                    return i;
                if (2 * terlev[p, i] + mas[p] == x)
                {
                    two[p] = true;
                    return (i);
                }
                if (3 * terlev[p, i] + mas[p] == x)
                {
                    three[p] = true;
                    return (i);
                }
            }
            else
            {
                if (terlev[p, i] == x)
                    return i;
            }
        }
        return 5;
    }
    public int InvRL(int num, int k)
    {
        for (int i = 0; i < 4; i++)
        {
            if (0 - terlev[num, k] == terlev[num, i])
                return i;
        }
        return -4;

    }
    public void add(int k)
    {
        newt[0, len] = terlev[0, k];
        newt[1, len] = terlev[1, k];
        newt[2, len] = terlev[2, k];
        words[n] = word[k];
```

```
                if (newy.Contains(ky[k, 0]) &&
!newy.Contains(ky[k, 1]))
                {
                    newy[n] = ky[k, 1];
                }
                else newy[n] = ky[k, 0];
                n++;
                int r = 0;

                if (mas[0] > 5)
                {
                    r = mas[0];
                    mas[0] += newt[0, len] - r;
                }
                else
                    mas[0] += newt[0, len];
                if (mas[1] > 5)
                {
                    r = mas[1];
                    mas[1] += newt[1, len] - r;
                }
                else
                    mas[1] += newt[1, len];
                if (mas[2] > 5)
                {
                    r = mas[2];
                    mas[2] += newt[2, len] - r;
                }
                else
                    mas[2] += newt[2, len];
                len++;

            }
            public void Table(DataGridView d, string[] w, int[,]
m)
            {

                int kol = 0;
                d.ColumnCount = len + 2;
                d.RowCount = 4;


                d.Rows[0].HeaderCell.Value = "a";
                d.Rows[1].HeaderCell.Value = "-1";
                d.Rows[2].HeaderCell.Value = "0";
                d.Rows[3].HeaderCell.Value = "+1";

                for (int i = 0; i < w.Length; i++)
                {
                    if (w[i] != null)
                    {

                        d.Columns[i + 1].HeaderText = w[i];
                        kol++;
```

```
                    }
                }

                for (int i = 0; i < 3; i++)
                {
                    for (int j = 0; j < kol; j++)
                    {
                        d.Rows[i + 1].Cells[j + 1].Value = m[i,
j].ToString();
                    }
                }
                d.Columns[kol + 1].HeaderText = "Σ";
                for (int i = 0; i < 3; i++)
                {
                    d.Rows[i + 1].Cells[kol + 1].Value =
mas[i].ToString();
                }

            }
            public Form1()
            {
                InitializeComponent();
            }

            private void Form1_Load(object sender, EventArgs e)
            {

                v1.SelectedIndex = 0;
                v2.SelectedIndex = 0;
                v3.SelectedIndex = 0;
            }
            public void Clear()
            {
                dataGridView1.Rows.Clear();
                dataGridView1.Columns.Clear();
                for (int i = 0; i < words.Length; i++)
                {
                    Owords[i] = null;
                    words[i] = null;
                }
                n = 0;
                Onewt = new int[3, 9];
                newt = new int[3, 9];
                mas = new int[3];
                len = 0;
                value = 0;
                two[0] = false;
                two[1] = false;
                two[2] = false;
                three[0] = false;
                three[1] = false;
                three[2] = false;
                mas[0] = 200;
                mas[1] = 200;
```

```
                mas[2] = 200;
                newy = new int[9];

                g.FillRectangle(Brushes.White,       407,       0,
picture.Width, picture.Height);
                Oy = new int[9];
            }

            private void button1_Click(object sender, EventArgs e)
            {
                if (flag)
                {
                    flag = false;
                    Clear();
                }
                int a1 = Convert.ToInt32(v1.SelectedItem);
                int a2 = Convert.ToInt32(v2.SelectedItem);
                int a3 = Convert.ToInt32(v3.SelectedItem);
                int S = a1 + a2 + a3;
                int f = 0, f1 = 0, f2 = 0;
                int k = -2, k1 = -2, k2 = -2;
                int num = -2;
                int num1 = -2;
                int num2 = -2;
                int[] arr = new int[3];
                // int[] arra = {a1}
                arr[0] = a1;
                arr[1] = a2;
                arr[2] = a3;
                //1.   Обираємо набір (№ в таблиці істинності)
функції, який будемо реалізовувати першим:
                //Це не нульове значення («+» або «-»), якого в
функції менше.
                //Якщо ненульових значень однакова кількість, то
обираємо те, яке відповідає ненульовому вхідному аргументу(«+» або
«-»).
                //Якщо є і те, і інше, то обираємо будь -
яке(наприклад, відповідне вх. «-»).

                if (S < 0)
                {
                    if (a1 > 0)
                    { f = a1; num = 0; }
                    else
                    if (a3 > 0) { f = a3; num = 2; }

                    else
                        if (a2 > 0)
                    { f = a2; num = 1; }
                    if (a1 != 0)
                    {
                        f = a1; num = 0;
                    }
                    else if (a3 != 0)
```

```
            {
                f = a3; num = 2;
            }
            else
            {
                f = a2; num = 1;
            }
        }
        else
        if (S > 0)
        {
            if (a1 < 0)
            { f = a1; num = 0; }
            else
            if (a3 < 0) { f = a3; num = 2; }

            else
             if (a2 < 0)
            { f = a2; num = 1; }
            else if (a1 != 0)
            {
                f = a1; num = 0;
            }
            else if (a3 != 0)
            {
                f = a3; num = 2;
            }
            else
            {
                f = a2; num = 1;
            }
        }
        else
        {
            if (a1 != 0)
            {
                f = a1; num = 0;
            }
            else if (a3 != 0)
            {
                f = a3; num = 2;
            }
            else
            {
                f = a2; num = 1;
            }
         }

/*****************************************************************
*******************
                 * 2.  Обираємо RL-функцію, яка реалізує значення
з п.1, незалежно від отриманих значень при інших аргументах.
```

```
**********************************************************************
**/
                if (num != -2)
                {
                    k = RL(num, f);
                }
                else MessageBox.Show("Error num");
                arr[num] = -2;

                if (k != 5)
                {
                    add(k);
                }
                /*3.   Обираємо набір (№ в таблиці істинності)
функції, який будемо реалізовувати другим:
     Це не нульове значення («+» або «-»), якого в функції
менше.
     Якщо два ненульових значень, то обираємо те, яке
відповідає ненульовому вхідному аргументу («+» або «-»).
     Інакше, обираємо те, що змінилось при виконанні п.2.
     */
                if (mas[0] == a1 && mas[1] == a2 && mas[2] == a3)
                {
                    Table(dataGridView2, words, newt);
                    Opt();
                    Table(dataGridView1, Owords, Onewt);
                    flag = true;
                    Draw();
                }
                else
                {

                    if (arr[0] == a1 && a1 != 0)
                    {
                        f1 = a1;
                        num1 = 0;
                    }
                    else
                        if (arr[2] == a3 && a3 != 0)
                    { f1 = a3; num1 = 2; }

                    else if (arr[1] == a2 && a2 != 0)
                    {
                        f1 = a2; num1 = 1;
                    }
                    else if (mas[0] != a1)
                    { f1 = a1; num1 = 0; }
                    else if (mas[1] != a2)
                    { f1 = a2; num1 = 1; }
                    else if (mas[2] != a3)
                    { f1 = a3; num1 = 2; }
```

```
                    /*4.    Обираємо    RL-функцію    (або    її
відсутність), яка реалізує значення з п.3 (можливо, їх буде дві,
для компенсації дій функції з п.2).
        Якщо функція з п.4 змінює значення п.1:
        □ на нульове, то необхідно додати RL-функцію з п.2.
        □ інакше,  необхідно  додати  RL-функцію  з  протилежним
значенням у вибраному в п.1 номері функції з п.4.
        */

                    if (num1 != -2)
                    {
                        k1 = RL(num1, f1);
                    }
                    else MessageBox.Show("Error num1");
                    if (k1 != 5)
                    {
                        if (three[num1])
                        {
                            add(k1);
                            add(k1);
                            add(k1);
                        }
                        else if (two[num1])
                        {
                            add(k1);
                            add(k1);
                        }
                        else add(k1);


                        arr[num1] = -2;
                        if (mas[num] != f)
                            if (mas[num] == 0)
                            {
                                add(k);
                            }
                            else
                            {
                                int inv = 0;
                                inv = InvRL(num, k1);
                                if (two[num1])
                                {
                                    add(inv);
                                    two[num1] = false;
                                }
                                if (three[num1])
                                {
                                    add(inv);
                                    add(inv);
                                    three[num1] = false;
                                }

                                add(inv);
                            }
```

```
                    }

/*************************************************************
*********************
                      5.    Обираємо    RL-функцію    (або    її
відсутність),  яка  реалізує  (або  відповідним  чином  змінює)
значення, що залишилось.

*************************************************************
***************/

                    if (mas[0] == a1 && mas[1] == a2 && mas[2] ==
a3)
                    {
                        Table(dataGridView2, words, newt);
                        Opt();
                        Table(dataGridView1, Owords, Onewt);
                        flag = true;
                        Draw();
                    }
                    else
                    {
                        for (int i = 0; i < 3; i++)
                        {
                            if (arr[i] != -2)
                            {
                                num2 = i;
                                f2 = arr[i];
                            }
                        }
                        if (num2 != -2)
                        {
                            k2 = RL(num2, f2);
                        }
                        else MessageBox.Show("Error num2");
                        if (k2 != 5)
                        {
                            if (three[num2])
                            {
                                add(k2);
                                add(k2);
                                add(k2);
                            }
                            else if (two[num2])
                            {
                                add(k2);
                                add(k2);
                            }
                            else add(k2);

                            /* Якщо функція з п.5 змінює значення
п.1:
     □    на нульове, то необхідно додати RL-функцію з п.2.
```

```
     інакше, необхідно додати RL-функцію з протилежним
значенням у вибраному в п.1 номері функції з п.5.
     */
                              if (f != mas[num])
                              {
                                  if (mas[num] == 0)
                                  {
                                      add(k);
                                  }
                                  else
                                  {
                                      int inv = 0;
                                      inv = InvRL(num, k2);
                                      if (two[num2])
                                      {
                                          add(inv);
                                          two[num2] = false;
                                      }
                                      if (three[num2])
                                      {
                                          add(inv);
                                          add(inv);
                                          three[num2] = false;
                                      }
                                      add(inv);
                                  }
                              }
                              /*Якщо функція з п.5 змінює значення
п.3:
     на нульове, то необхідно додати RL-функцію з п.4.
     інакше, необхідно додати RL-функцію з протилежним
значенням у вибраному в п.3 номері функції з п.5.
     */
                              if (f1 != mas[num1])
                              {
                                  if (mas[num1] == 0)
                                  {
                                      add(k1);
                                  }
                                  else
                                  {
                                      int inv = 0;
                                      inv = InvRL(num1, k2);
                                      if (two[num2])
                                      {
                                          add(inv);
                                          two[num2] = false;
                                      }
                                      if (three[num2])
                                      {
                                          add(inv);
                                          add(inv);
                                          three[num2] = false;
                                      }
```

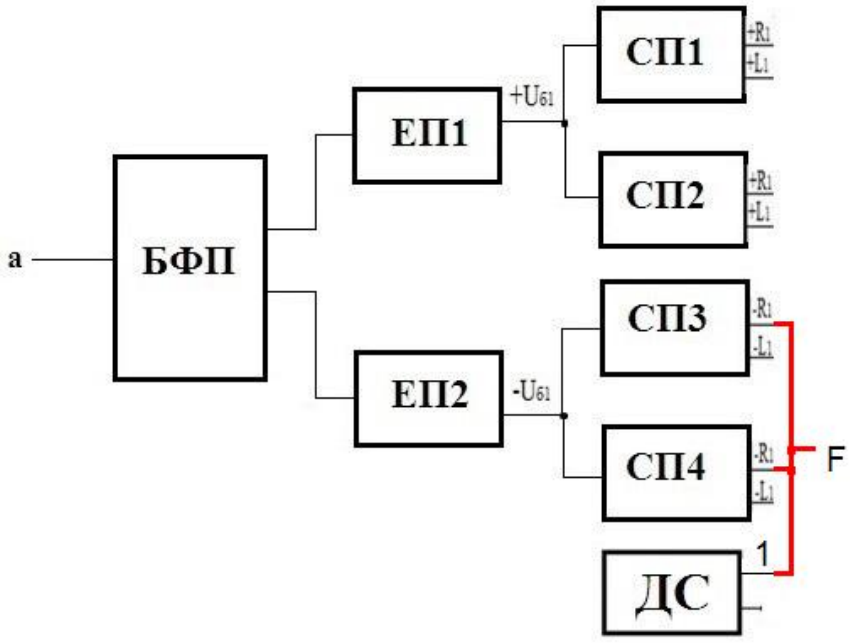
```
                                add(inv);
                            }
                        }
                    }
                    if (mas[0] == a1 && mas[1] == a2 && mas[2]
== a3)
                    {
                        Table(dataGridView2, words, newt);
                        Opt();
                        Table(dataGridView1, Owords, Onewt);
                        flag = true;
                        Draw();
                    }
                }
            }
        }

        private void button2_Click(object sender, EventArgs e)
        {
            Bitmap bmpSave = (Bitmap)picture.Image;
            SaveFileDialog sfd = new SaveFileDialog();
            sfd.DefaultExt = "bmp";
            sfd.Filter = "Image files (*.bmp)|*.bmp|All files
(*.*)|*.*";
            if (sfd.ShowDialog() == DialogResult.OK)
            {
                g.Save();
                (picture.Image as Bitmap).Save(sfd.FileName,
System.Drawing.Imaging.ImageFormat.Bmp);
            }
        }
    }
}
```

# ДОДАТОК Б

Синтезовані структури трійкових одномісних функцій

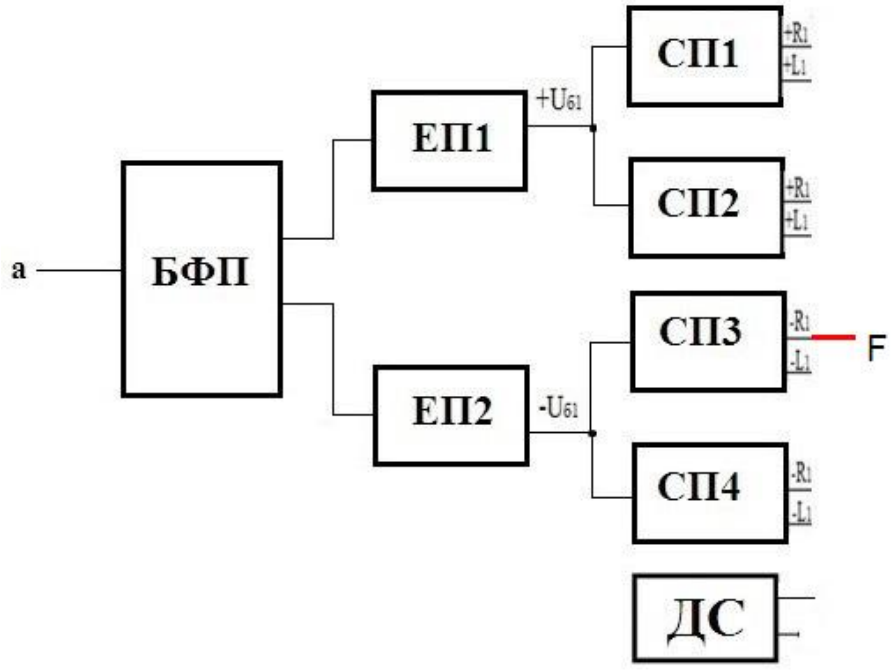
| X | Y |
|---|---|
| –1 | –1 |
| 0 | –1 |
| +1 | –1 |

Рисунок Б.1 – Структурна схема функції (–1, –1, –1)



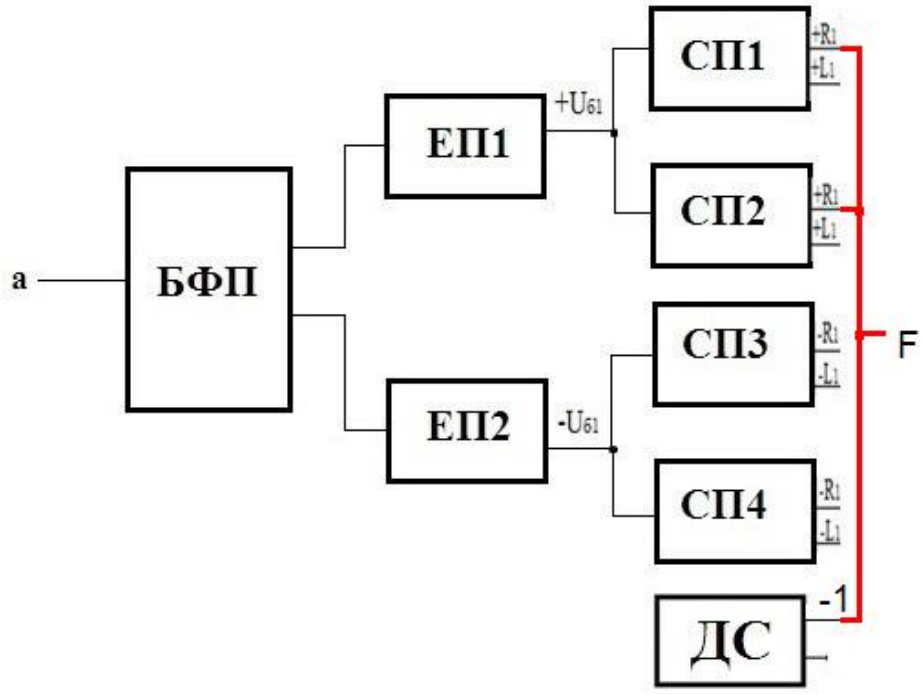

| X | Y |
|---|---|
| –1 | –1 |
| 0 | –1 |
| +1 | +1 |

Рисунок Б.2 – Структурна схема функції (–1, –1, +1)



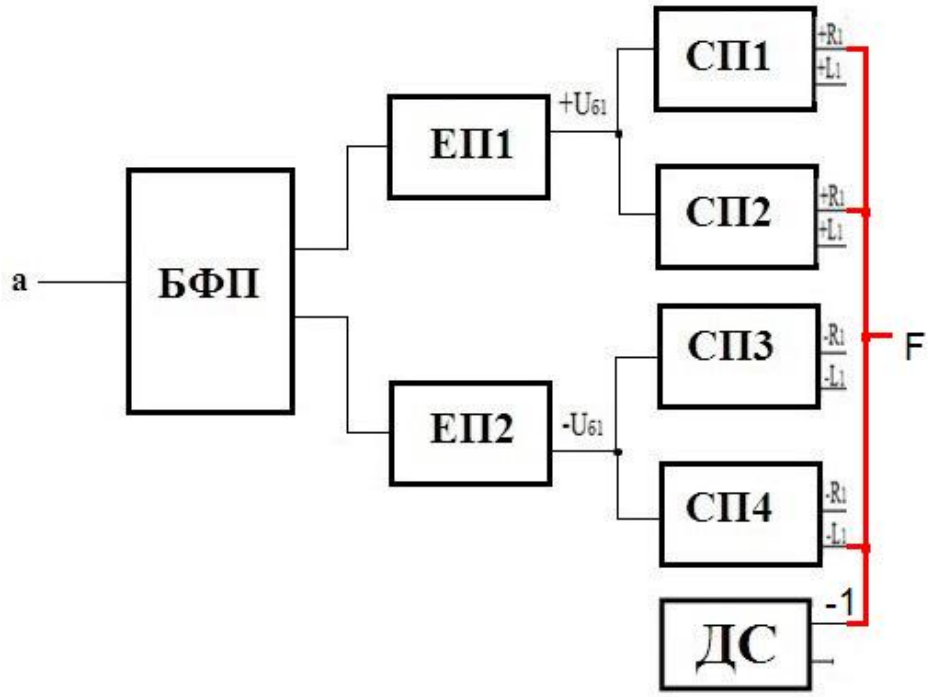

| X | Y |
|---|---|
| –1 | –1 |
| 0 | –1 |
| +1 | 0 |

Рисунок Б.3 – Структурна схема функції (–1, –1, 0)



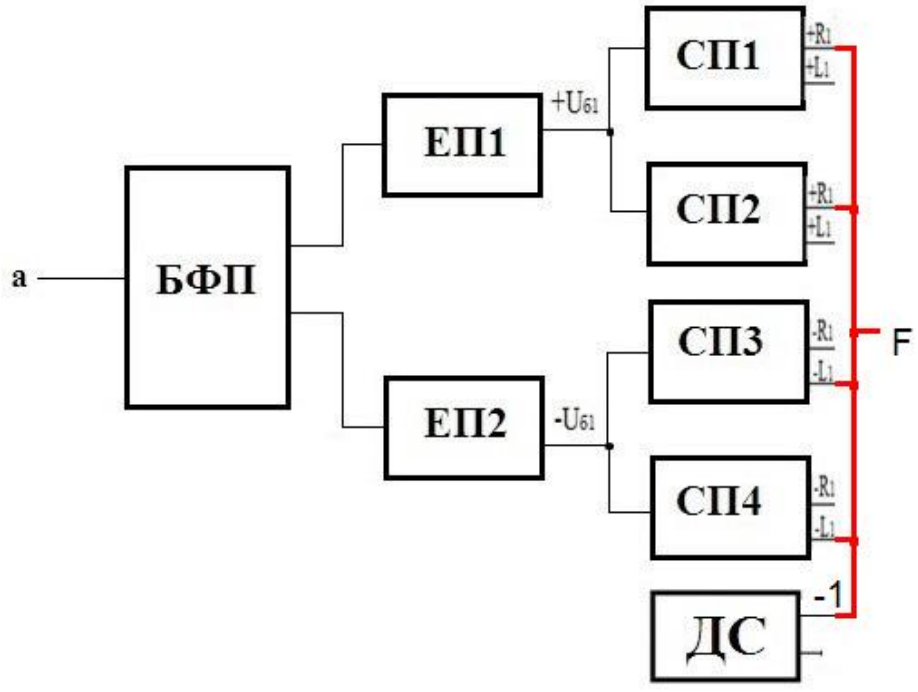

| X | Y |
|---|---|
| –1 | –1 |
| 0 | +1 |
| +1 | +1 |

Рисунок Б.4 – Структурна схема функції (–1, +1, +1)



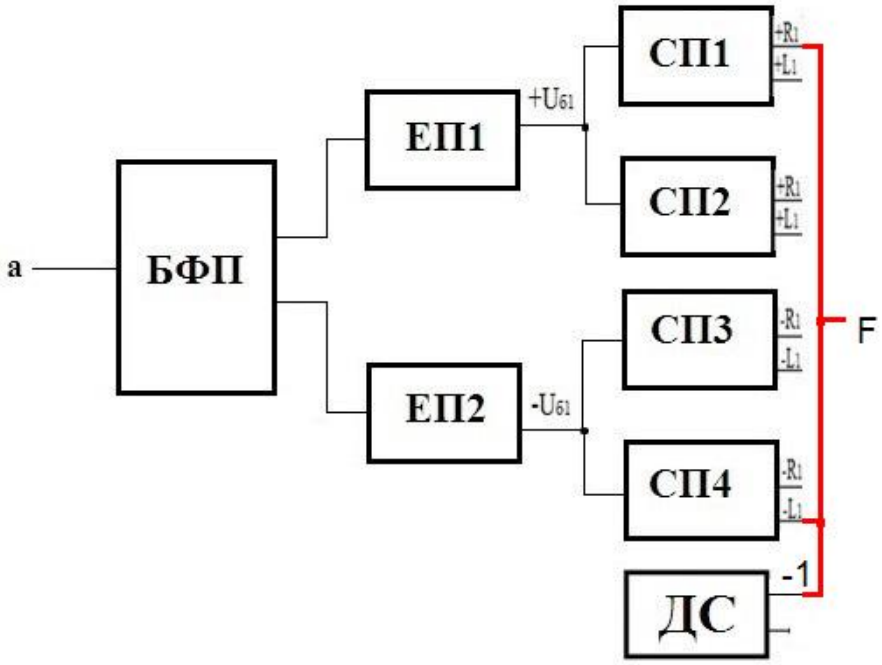

| X | Y |
|---|---|
| –1 | –1 |
| 0 | +1 |
| +1 | 0 |

Рисунок Б.5 – Структурна схема функції (–1, +1, 0)



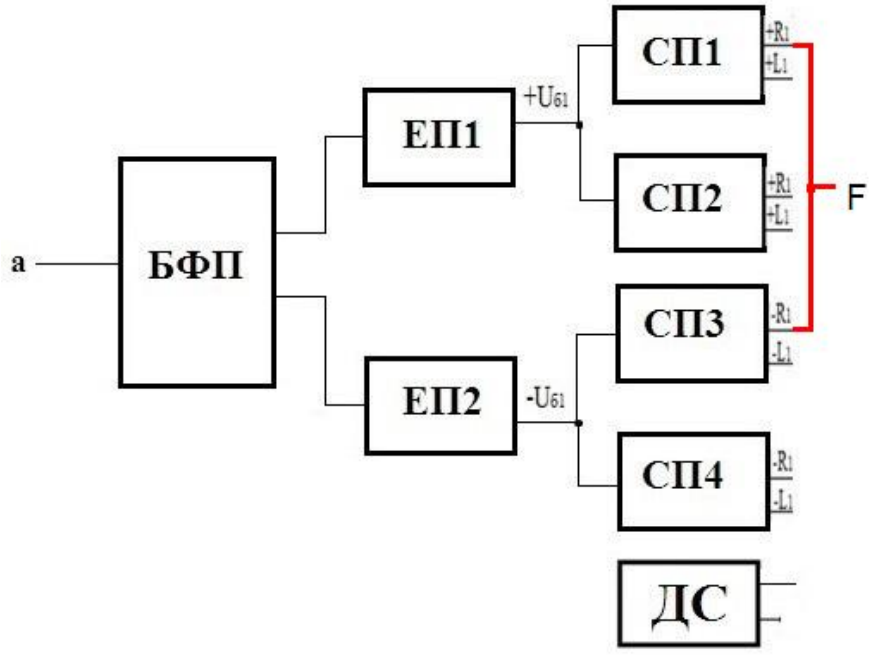

| X | Y |
|---|---|
| –1 | –1 |
| 0 | +1 |
| +1 | –1 |

Рисунок Б.6 – Структурна схема функції (–1, +1, –1)



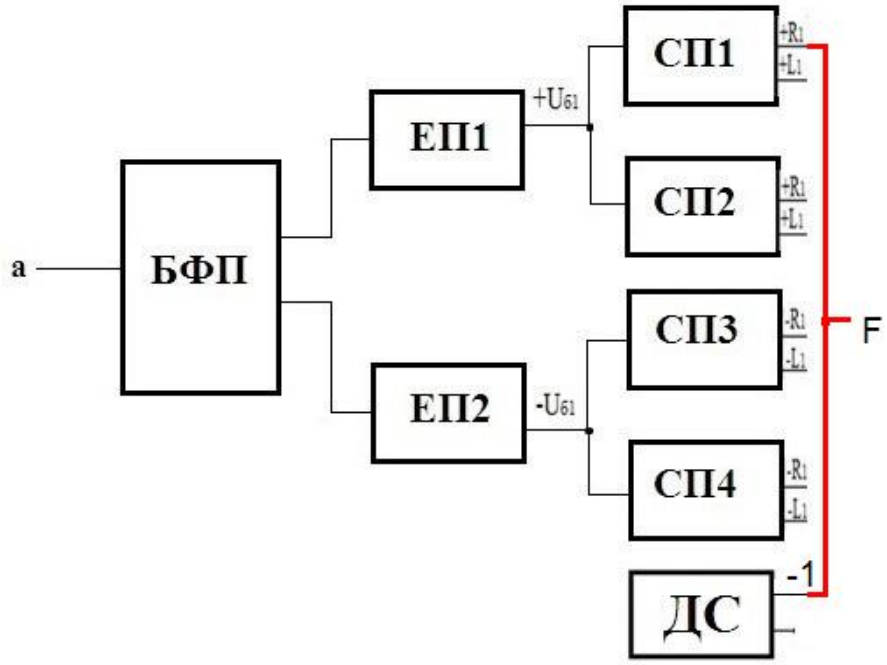

| X | Y |
|---|---|
| –1 | –1 |
| 0 | 0 |
| +1 | –1 |

Рисунок Б.7 – Структурна схема функції (–1, 0, –1)



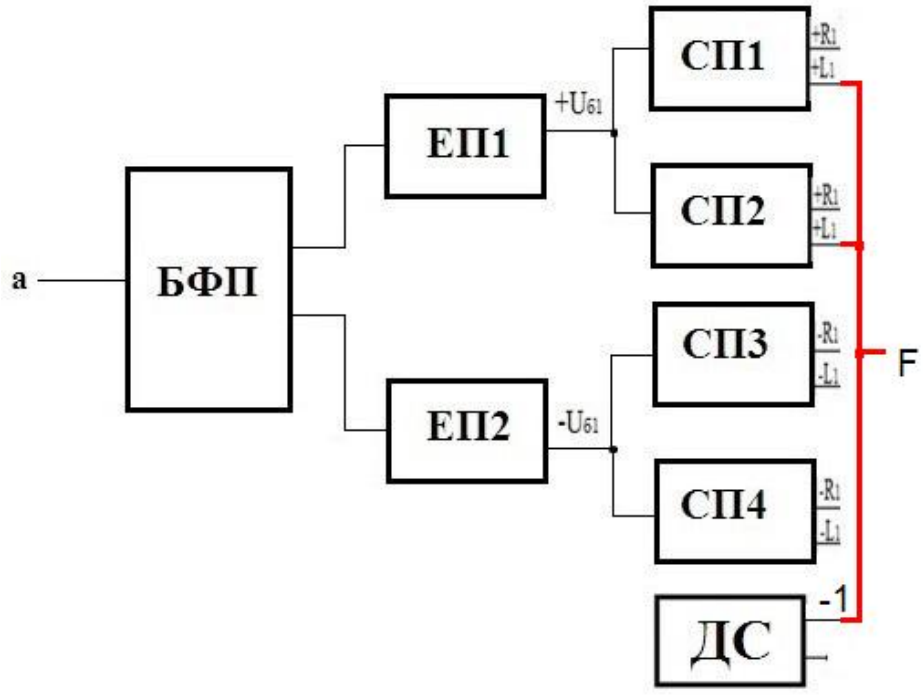

| X | Y |
|---|---|
| –1 | –1 |
| 0 | 0 |
| +1 | +1 |

Рисунок Б.8 – Структурна схема функції (–1, 0, +1)



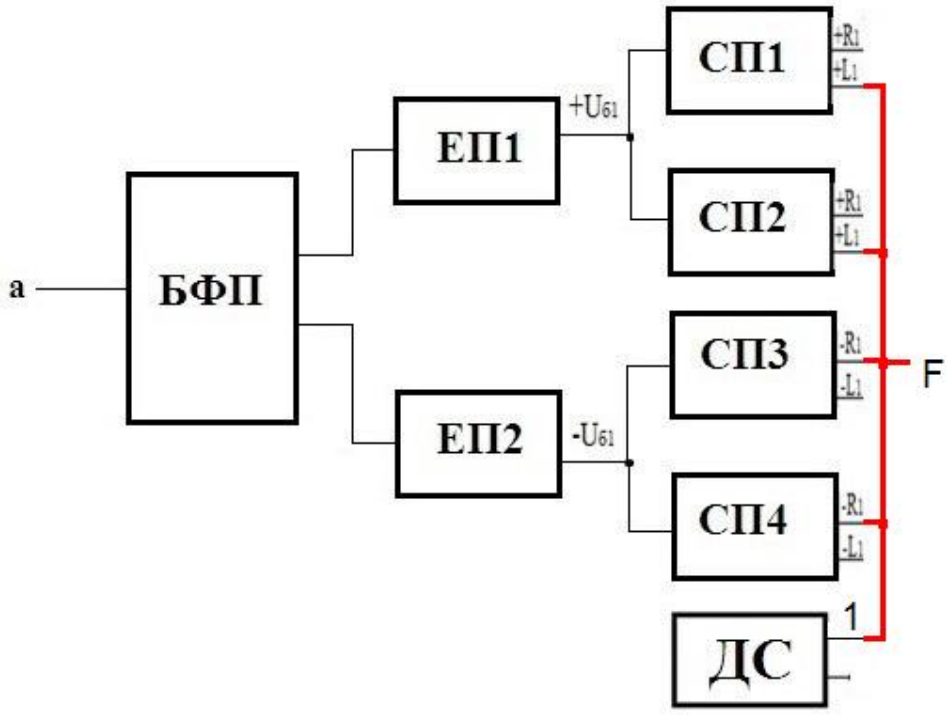

| X | Y |
|---|---|
| –1 | –1 |
| 0 | 0 |
| +1 | 0 |

Рисунок Б.9 – Структурна схема функції (–1, 0, 0)



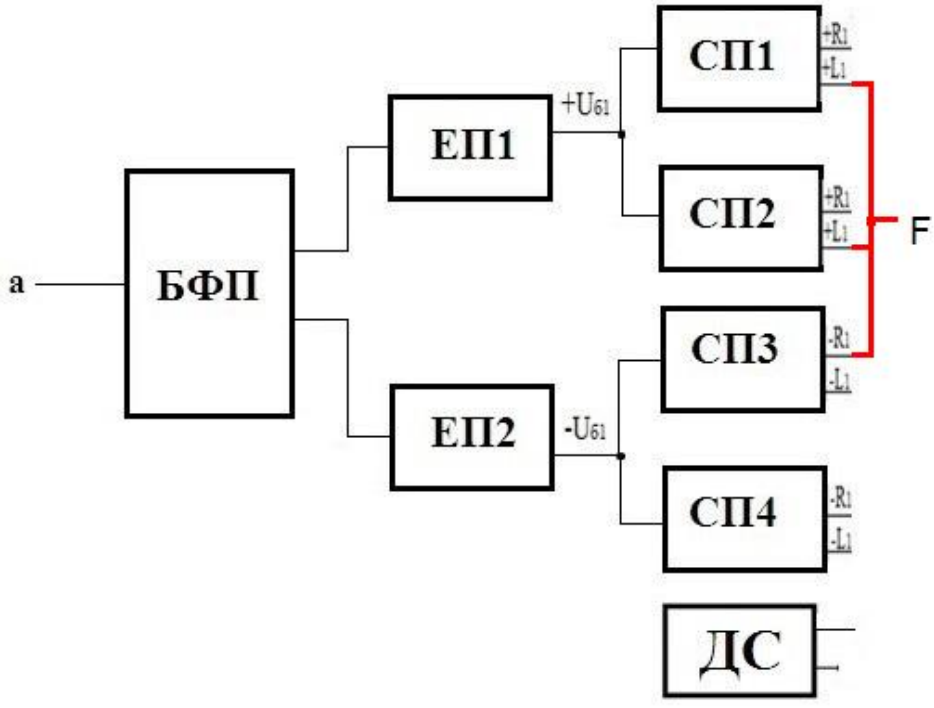

| X | Y |
|---|---|
| –1 | +1 |
| 0 | –1 |
| +1 | –1 |

Рисунок Б.10 – Структурна схема функції (+1, –1, –1)



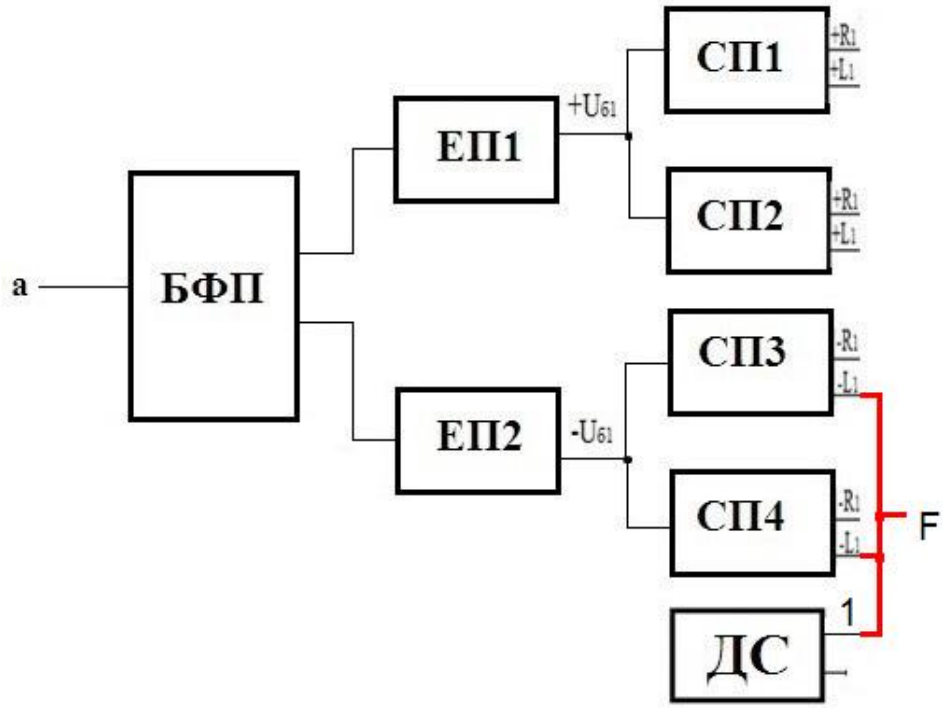

| X | Y |
|---|---|
| –1 | +1 |
| 0 | –1 |
| +1 | +1 |

Рисунок Б.11 – Структурна схема функції (+1, –1, +1)



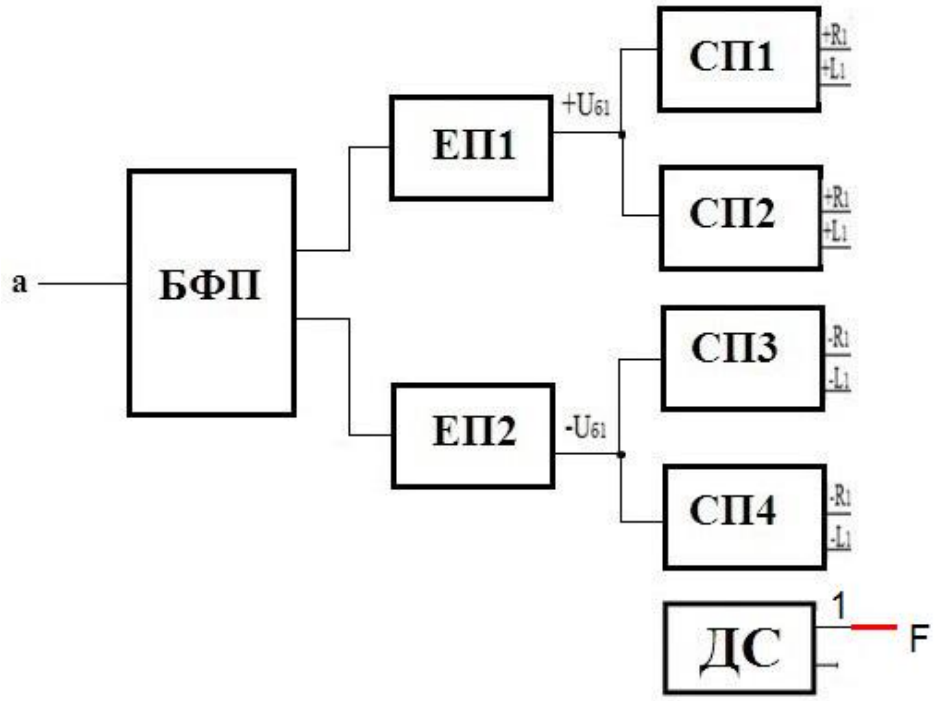

| X | Y |
|---|---|
| –1 | +1 |
| 0 | –1 |
| +1 | 0 |

Рисунок Б.12 – Структурна схема функції (+1, –1, 0)



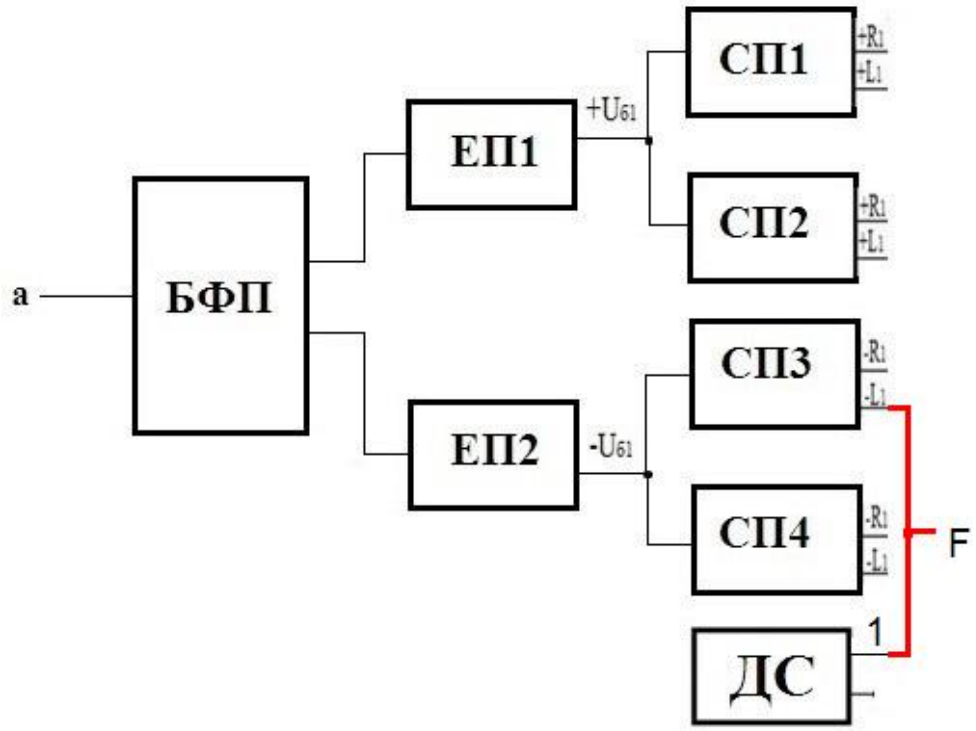

| X | Y |
|---|---|
| –1 | +1 |
| 0 | +1 |
| +1 | –1 |

Рисунок Б.13 – Структурна схема функції (+1, +1, –1)



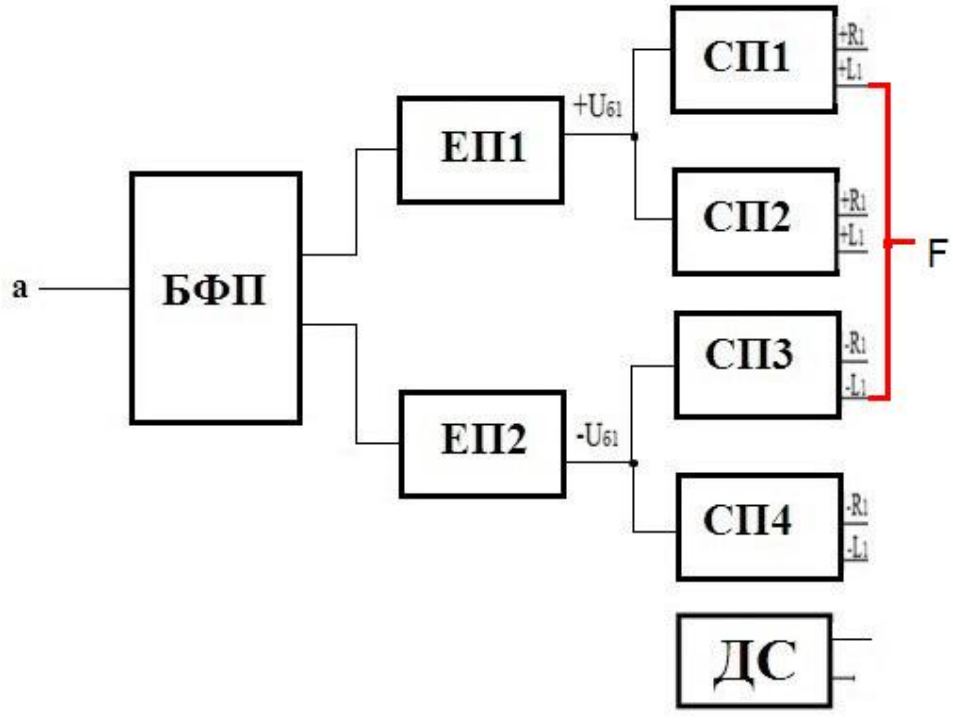

| X | Y |
|---|---|
| –1 | +1 |
| 0 | +1 |
| +1 | +1 |

Рисунок Б.14 – Структурна схема функції (+1, +1, +1)



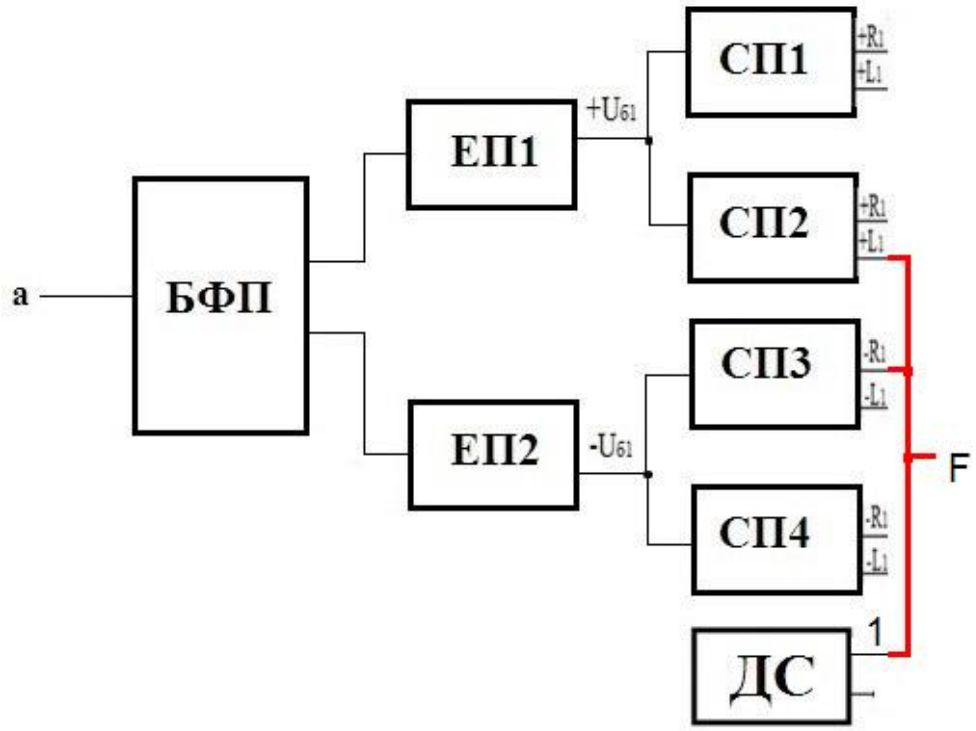

| X | Y |
|---|---|
| –1 | +1 |
| 0 | +1 |
| +1 | 0 |

Рисунок Б.15 – Структурна схема функції (+1, +1, 0)



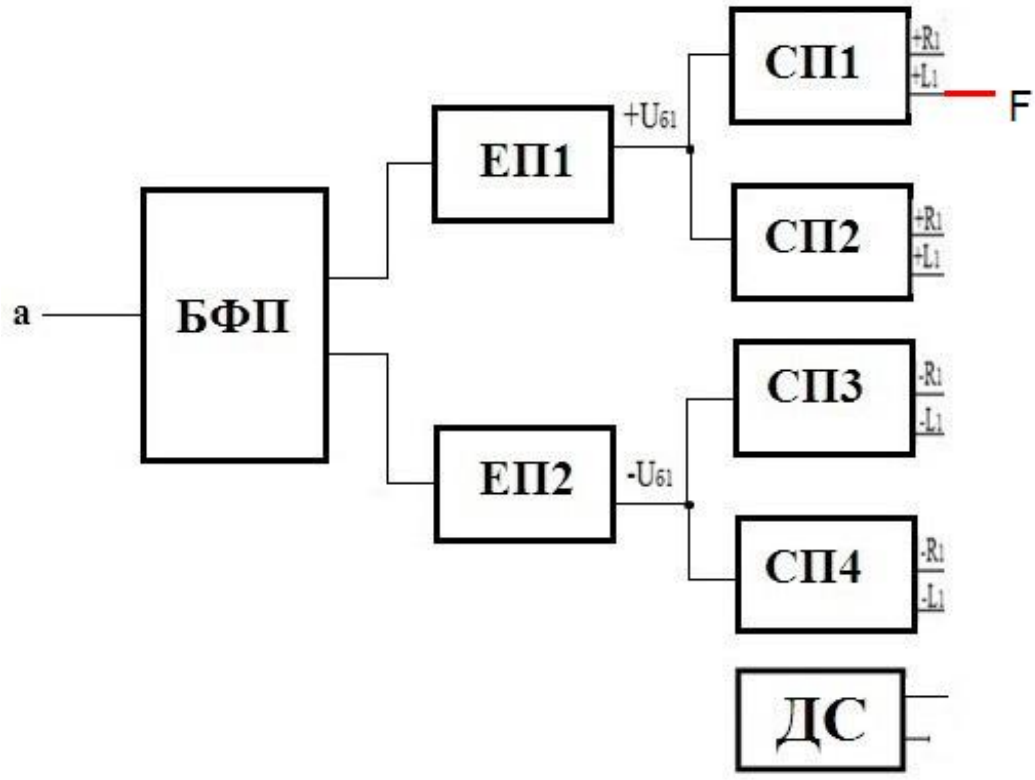

| X | Y |
|---|---|
| –1 | +1 |
| 0 | 0 |
| +1 | –1 |

Рисунок Б.16 – Структурна схема функції (+1, 0, –1)



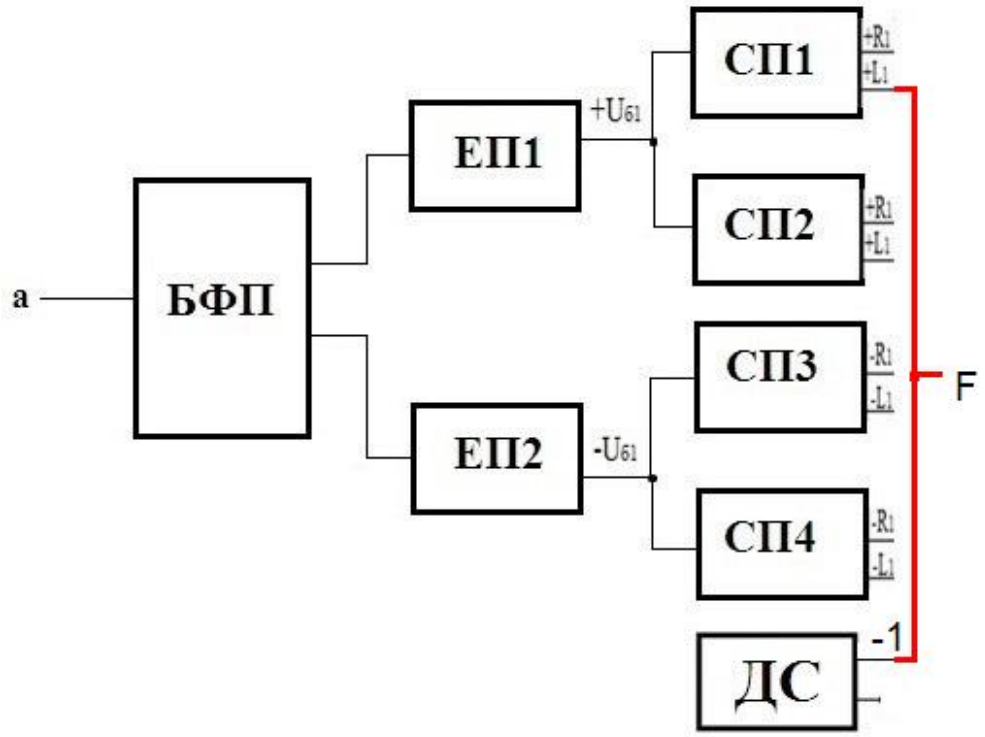

| X | Y |
|---|---|
| –1 | +1 |
| 0 | 0 |
| +1 | +1 |

Рисунок Б.17 – Структурна схема функції (+1, 0, +1)



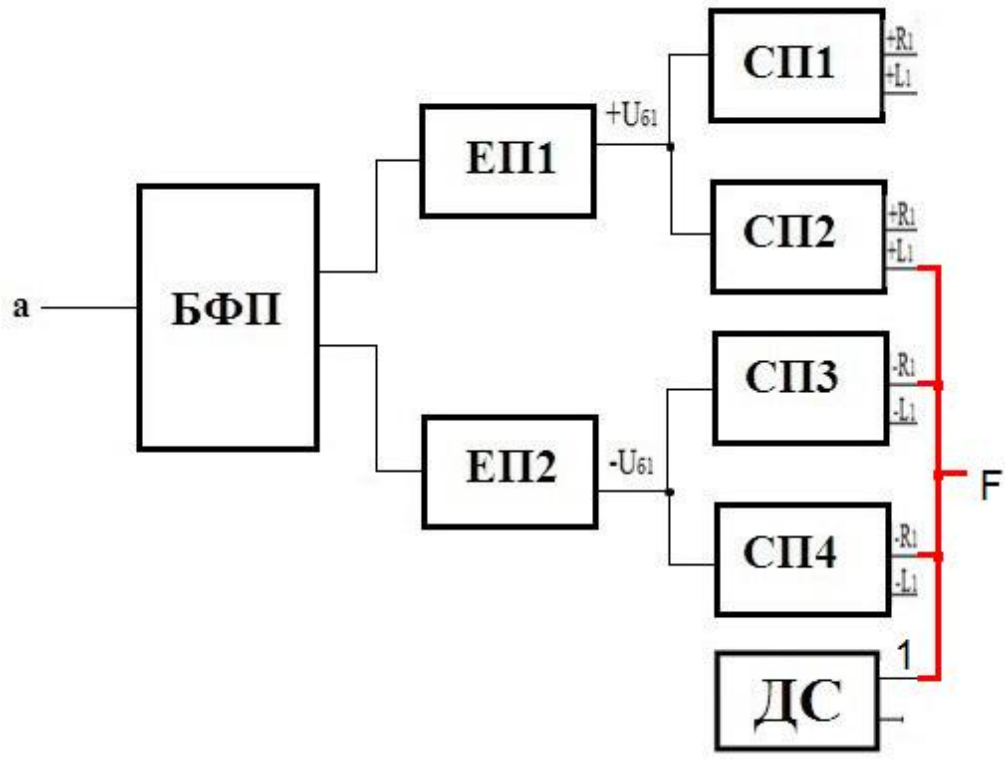

| X | Y |
|---|---|
| –1 | +1 |
| 0 | 0 |
| +1 | 0 |

Рисунок Б.18 – Структурна схема функції (+1, 0, 0)



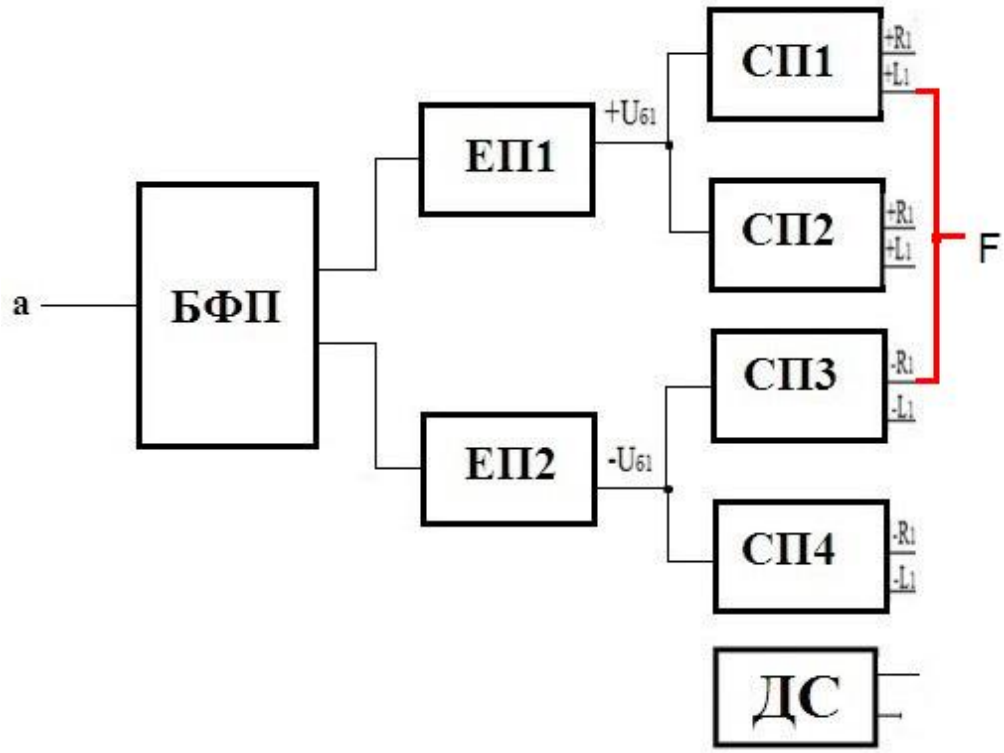

| X | Y |
|---|---|
| –1 | 0 |
| 0 | –1 |
| +1 | –1 |

Рисунок Б.19 – Структурна схема функції (0, –1, –1)



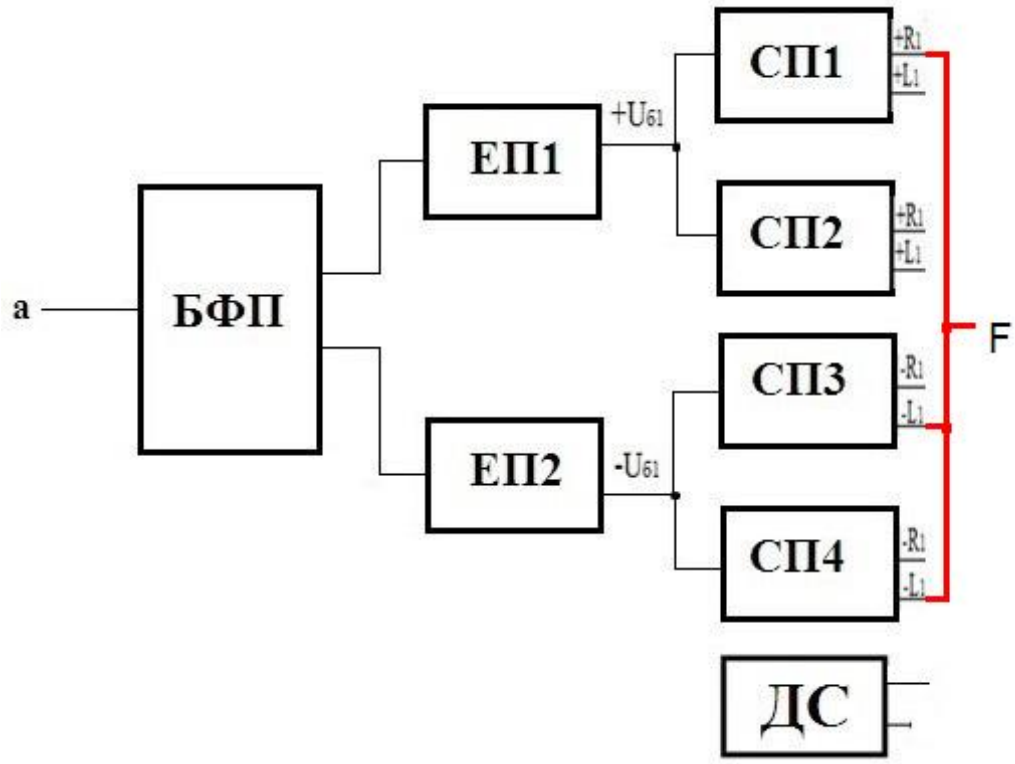

| X | Y |
|---|---|
| –1 | 0 |
| 0 | –1 |
| +1 | +1 |

Рисунок Б.20 – Структурна схема функції (0, –1, +1)



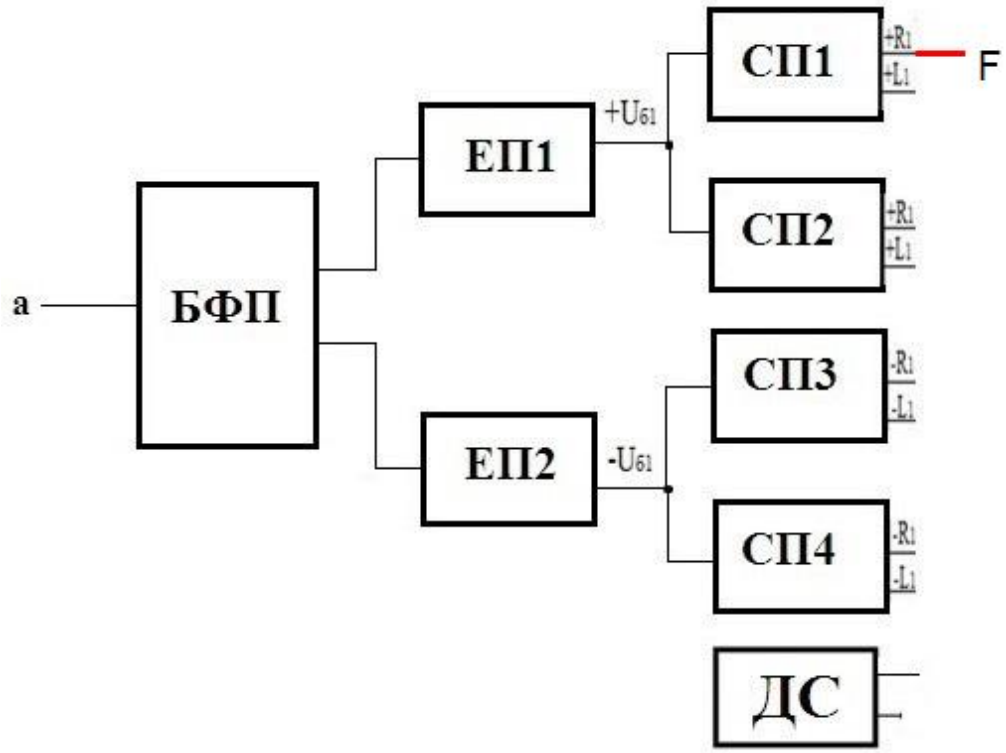

| X | Y |
|---|---|
| –1 | 0 |
| 0 | –1 |
| +1 | 0 |

Рисунок Б.21 – Структурна схема функції (0, –1, 0)



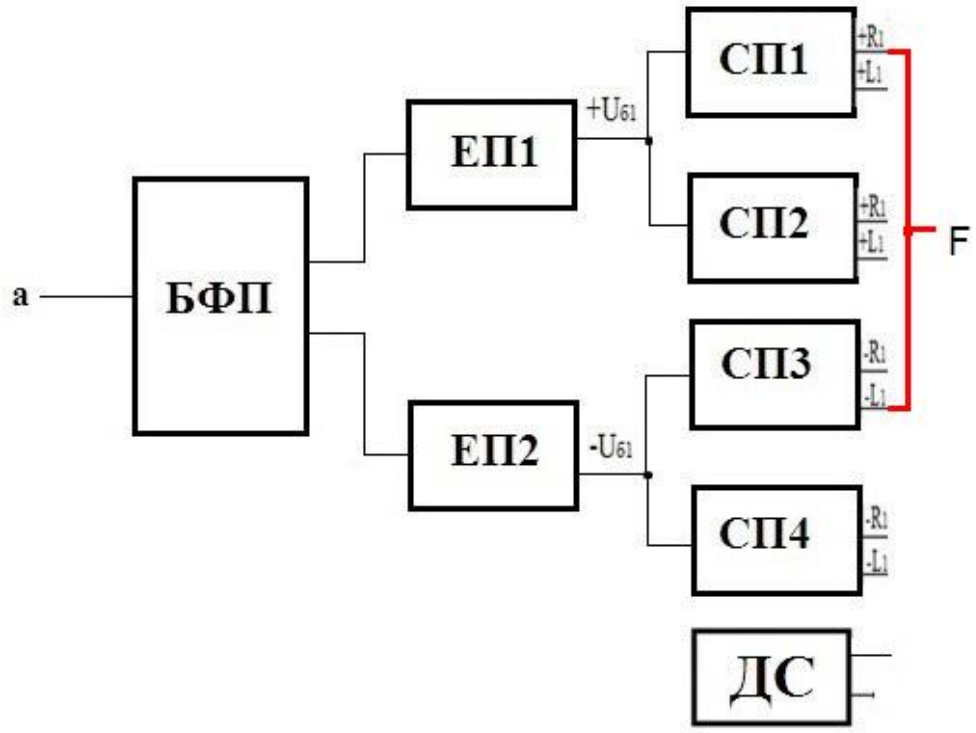

| X | Y |
|---|---|
| –1 | 0 |
| 0 | +1 |
| +1 | –1 |

Рисунок Б.22 – Структурна схема функції (0, +1, –1)



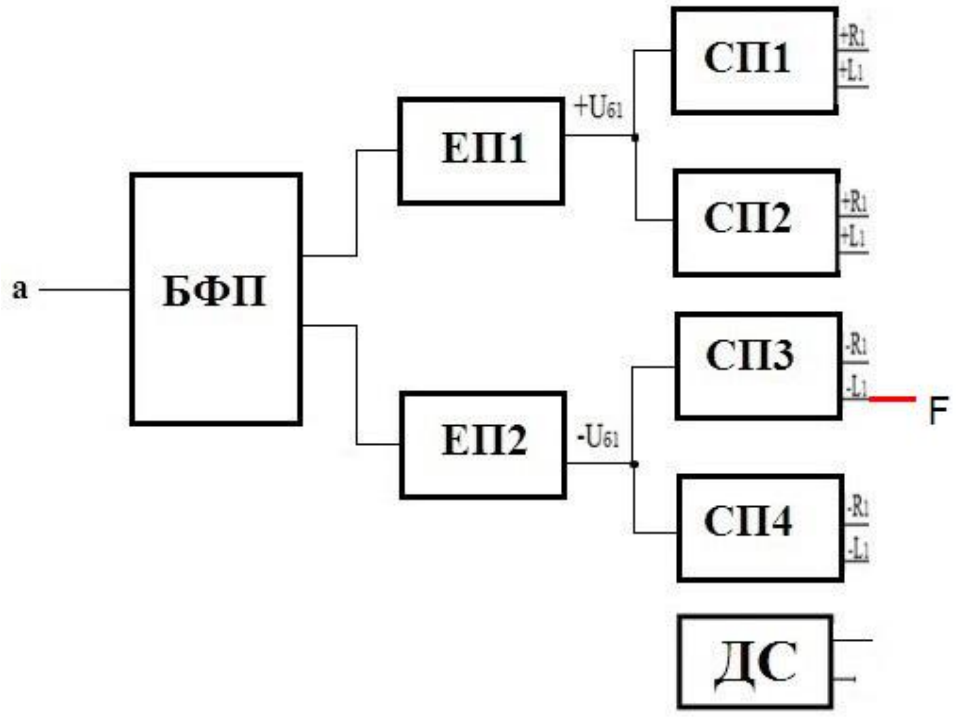

| X | Y |
|---|---|
| –1 | 0 |
| 0 | +1 |
| +1 | +1 |

Рисунок Б.23 – Структурна схема функції (0, +1, +1)



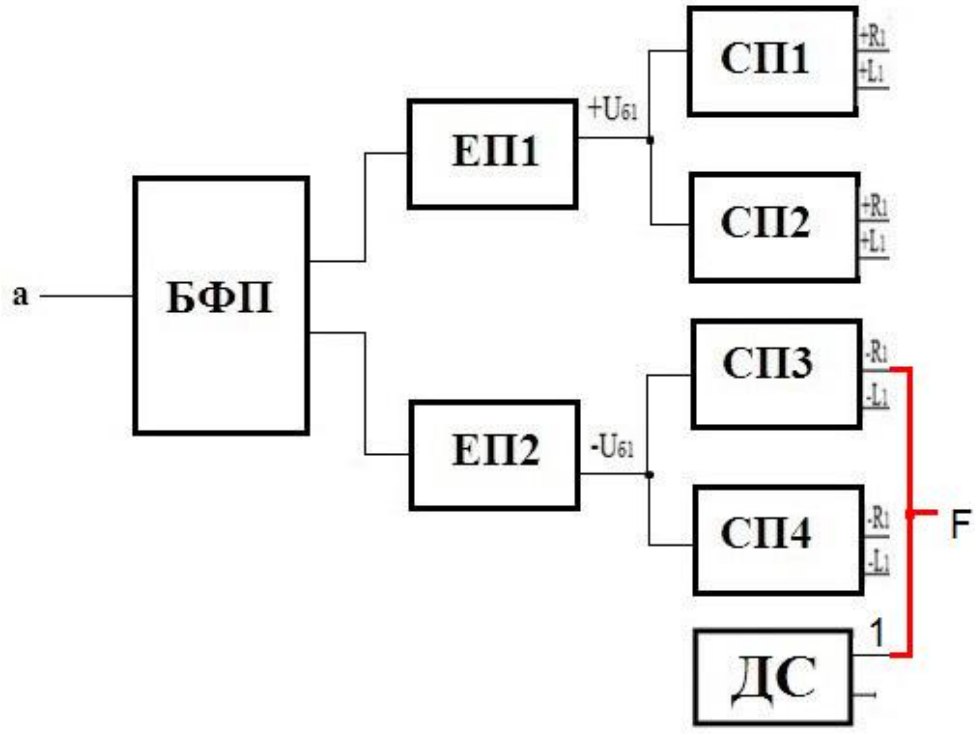

| X | Y |
|---|---|
| –1 | 0 |
| 0 | +1 |
| +1 | 0 |

Рисунок Б.24 – Структурна схема функції (0, +1, 0)



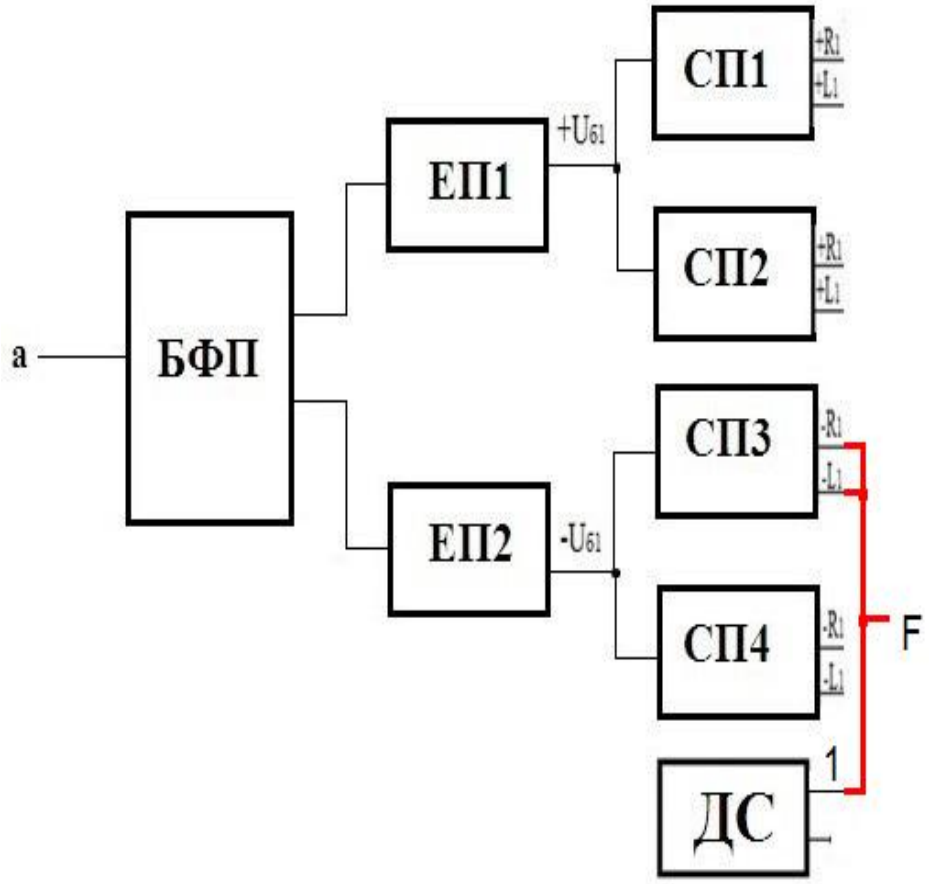

| X | Y |
|---|---|
| –1 | 0 |
| 0 | 0 |
| +1 | –1 |

Рисунок Б.25 – Структурна схема функції (0, 0, –1)




| X | Y |
|---|---|
| –1 | 0 |
| 0 | 0 |
| +1 | +1 |

Рисунок Б.26 – Структурна схема функції (0, 0, +1)




| X | Y |
|---|---|
| –1 | 0 |
| 0 | 0 |
| +1 | 0 |

Рисунок Б.27 – Структурна схема функції (0, 0, 0)