СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc451336866)

[1 РАЗРАБОТКА АЛГОРИТМА УМНОЖЕНИЯ 6](#_Toc451336867)

[2 РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ СУММАТОРА-УМНОЖИТЕЛЯ 9](#_Toc451336868)

[3 РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ СХЕМ ОСНОВНЫХ УЗЛОВ СУММАТОРА-УМНОЖИТЕЛЯ 10](#_Toc451336869)

[3.1 Логический синтез одноразрядного четверичного сумматора 10](#_Toc451336871)

[3.2 Логический синтез одноразрядного четверичного умножителя 14](#_Toc451336872)

[4 СИНТЕЗ КОМБИНАЦИОННЫХ СХЕМ НА ОСНОВЕ МУЛЬТИПЛЕКСОРА 27](#_Toc451336873)

[5 ОЦЕНКА РЕЗУЛЬТАТОВ РАЗРАБОТКИ 29](#_Toc451336874)

[ЗАКЛЮЧЕНИЕ 30](#_Toc451336875)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 31](#_Toc451336876)

[ПРИЛОЖЕНИЕ А 32](#_Toc451336877)

[ПРИЛОЖЕНИЕ Б 33](#_Toc451336878)

[ПРИЛОЖЕНИЕ В 34](#_Toc451336879)

[ПРИЛОЖЕНИЕ Г 35](#_Toc451336880)

[ПРИЛОЖЕНИЕ Д 36](#_Toc451336881)

[ПРИЛОЖЕНИЕ Е..................................................................................................37](#_Toc451336881)

# **введение**

Предмет «Арифметические и логические основы вычислительной техники» является основополагающим в вопросах организации ЭВМ, а, следовательно, и неотъемлемой частью подготовки специалиста в области информационных технологий.

Целью курсового проекта является ознакомление с принципами построения отдельных составных частей ЭВМ и их взаимосвязи, а также получение практических знаний о правильном оформлении .

В данном курсовом проекте выполнен синтез сумматора-умножителя первого типа для алгоритма А в прямых кодах на два разряда одновременно.

Для достижения цели необходимо:

Разработать алгоритм умножения и оценить погрешности вычислений.

Разработать структурную схему сумматора-умножителя первого типа с блоком сравнения порядков.

Разработать функциональные схемы основных узлов сумматора-умножителя в заданных логических базисах.

Разработать комбинационную схему на основе мультиплексора.

Рассчитать время умножения.

# **Разработка алгоритма умножения**

1. Перевод сомножителей из десятичной системы счисления в четверичную.

Мн = 84,19; Мт = 55,13;

**Множимое**

|  |  |  |  |
| --- | --- | --- | --- |
| \_84  84 | 4 | | 4 |
| \_21  20 | 4 |
| 0 | \_5  4 |
| 1 | | 1 |
| 1 | |

|  |
| --- |
| 0.19  \* 4 |
| 0.76 |
| \* 4 |
| 3.04 |

Мн4 = 1110,03.

В соответствии с кодировкой множимого:

Мн2/4 =00000011,1110.

**Множитель**

|  |
| --- |
| 0.13  \* 4 |
| 0.52  \* 4 |
| 2.08  \* 4 |
| 0.32 |

|  |  |  |  |
| --- | --- | --- | --- |
| \_55  52 | 4 | |  |
| \_13  12 | 4 |
| 3 | 3 |
| 1 | |
|  | |

Мт4 = 121.113

В соответствии с обычной весомозначной кодировкой множителя:

Мт2/4 = 011001.010111.

2. Запишем сомножители в форме с плавающей запятой в прямом коде:

Мн = 0.000000111110 Рмн = 0.0011 +104  – закодировано по заданию

Мт = 0.110111001000 Рмт = 0.0011 +034 – закодировано традиционно

3. Умножение двух чисел с плавающей запятой на два разряда множителя одновременно в прямых кодах. Это сводится к сложению порядков, формированию знака произведения, преобразованию разрядов множителя согласно алгоритму и перемножению мантисс сомножителей.

Порядок произведения будет следующим:

Рмн = 0.0011 +104

Рмт = 0.0011 +034

= 0.0010 +134

Результат закодирован в соответствии с заданием на кодировку множимого.

Знак произведения определяется суммой по модулю два знаков сомножителей, т.е:

Для умножения мантисс необходимо преобразовать множитель. При умножении чисел в прямых кодах диада 11() преобразуется в триаду 10. Преобразованный множитель имеет вид или

= 0101000100. Перемножение мантисс по алгоритму “А” приведено в таблице 1.1.

Таблица 1.1 - Перемножение мантисс

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Четверичная с/c** | | | **Двоично-четверичная с/с** | | | **Комментарий** |
| **1** | | | **2** | | | **3** |
| 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. | 0000000  0000000  0000000  0000000  0222012  0222012  0022201  0000000  0022201  0002220 | 0  0  0  20  20  120 | 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. | 11 11 11 11 11 11 11  11 11 11 11 11 11 11  11 11 11 11 11 11 11  11 11 11 11 11 11 11  11 01 01 01 11 00 01  11 01 01 01 11 00 01  11 11 01 01 01 11 00  11 11 11 11 11 11 11  11 11 01 01 01 11 00  11 11 11 01 01 01 11 | 11  11  01 11  01 11  00 01 11 |  |

*Продолжение таблицы 1.1*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **1** | | | **2** | | | **3** |
| 3.  3.  3.  0.  0.  0.  3.  3.  3.  0.  0. | 3222331  3231211  3323121  0222012  0211133  0021113  3222331  3310110  3331011  0111003  0102020 | 120  1120  1120  31120  31120  031120  031120 | 1.  1.  1.  0.  0.  0.  1.  1.  1.  0.  0. | 10 01 01 01 10 10 00  10 01 10 00 01 00 00  10 10 01 10 00 01 00  11 01 01 01 11 00 01  11 01 00 00 00 10 10  11 11 01 00 00 00 10  10 01 01 01 10 10 00  10 10 00 11 00 00 11  10 10 10 00 11 00 00  11 00 00 00 11 11 10  11 00 11 01 11 01 11 | 00 01 11  00 00 01 11  00 00 01 11  10 00 00 01 11  10 00 00 01 11  11 10 00 00 01 11  11 10 00 00 01 11 |  |

После окончания умножения необходимо оценить погрешность вычислений. Для этого полученное произведение приводится к нулевому порядку, а затем переводится в десятичную систему счисления:

;

.

Результат прямого перемножения операндов дает результат:

..

Абсолютная погрешность

– = 0.5588.

Относительная погрешность

Эта погрешность получена за счёт приближённого перевода из десятичной системы счисления в четверичную обоих сомножителей, а также за счёт округления полученного результата произведения.

**2. РАЗРАБОТКА СТРУКТУРНОЙ СХЕМЫ СУММАТОРА – УМНОЖИТЕЛЯ ПЕРВОГО ТИПА**

Структурная схема сумматора-умножителя первого типа для алгоритма умножения «А» приведена в приложении А.

Сразу после начала алгоритма обнуляются все регистры.

*Режим работы «Сумма»* (Mul/sum – «1»). В начале есть необходимость выравнивания порядков. Для этого в блоке сравнения порядков они вычитаются и, в зависимости от знака суммы и функции сравнения ее с нулем, в регистр множимого поступает мантисса, которая будет сдвигаться. Формируется ее дополнительных код и попадает в аккумулятор. В регистр множимого заносится вторая мантисса.

Далее выполняется выравнивание порядков посредством некоторого количества сдвигов мантиссы в аккумуляторе, равном разности порядков.

По завершению сложения проверяется переполнение. Если переполнение было выявлено, происходит обратный сдвиг содержимого аккумулятора, а порядок результата уменьшается на единицу.

При необходимости выравнивания порядков в регистре-аккумуляторе может выполняться сдвиг мантиссы первого слагаемого.

Если устройство работает как *умножитель* (Mul/sum – «0»), то в порядок результата заносится сумма порядков множимого и множителя в дополнительном коде. Параллельно с этим их мантиссы заносятся в соответствующие регистры.

Далее пошагово выполняется сама операция умножения: сдвиг мантиссы и преобразование разрядов множителя, формирования дополнительного кода частичного произведения и занесение его в регистр аккумулятор, после разряды последнего сдвигаются, а регистр множителя служит его «продолжением». Операция продолжается, пока разряды множителя не иссякнут.

**3. РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ СХЕМ ОСНОВНЫХ УЗЛОВ СУММАТОРА – УМНОЖИТЕЛЯ**

# **Логический синтез одноразрядного четверичного сумматора**

ОЧС – это комбинационное устройство с 5 двоичными входами (2 разряда из одного слагаемого, 2 разряда другого и вход переноса) и 3 двоичными выходами.

Принцип работы ОЧС представлен с помощью таблицы истинности (таблица 1.4).

Разряды обоих слагаемых закодированы: 0-11; 1-00; 2-01; 3-10. В таблице 3.n выделено 16 безразличных наборов, т.к. на входы ОЧС со старших выходов ОЧУ не могут прийти «2» и «3».

Таблица 3.1 – Таблица истинности ОЧС

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **a1** | **a2** | **b1** | **b2** | **p** | **П** | **S1** | **S2** | **Пример операции в четверичной с/с** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1+1+0=02 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1+1+1=03 |
| 0 | 0 | 0 | 1 | 0 | x | x | x | 1+2+0=03 |
| 0 | 0 | 0 | 1 | 1 | x | x | x | 1+2+1=10 |
| 0 | 0 | 1 | 0 | 0 | x | x | x | 1+3+0=10 |
| 0 | 0 | 1 | 0 | 1 | x | x | x | 1+3+1=11 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1+0+0=01 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1+0+1=02 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2+1+0=03 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 2+1+1=10 |
| 0 | 1 | 0 | 1 | 0 | x | x | x | 2+2+0=10 |
| 0 | 1 | 0 | 1 | 1 | x | x | x | 2+2+1=11 |
| 0 | 1 | 1 | 0 | 0 | x | x | x | 2+3+0=11 |
| 0 | 1 | 1 | 0 | 1 | x | x | x | 2+3+1=12 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 2+0+0=02 |
| 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 2+0+1=03 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 3+1+0=10 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 3+1+1=10 |
| 1 | 0 | 0 | 1 | 0 | x | x | x | 3+2+0=11 |
| 1 | 0 | 0 | 1 | 1 | x | x | x | 3+2+1=12 |
| 1 | 0 | 1 | 0 | 0 | x | x | x | 3+3+0=12 |
| 1 | 0 | 1 | 0 | 1 | x | x | x | 3+3+1=13 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 3+0+0=03 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 3+0+1=10 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0+1+0=01 |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0+1+1=02 |
| 1 | 1 | 0 | 1 | 0 | x | x | x | 0+2+0=02 |
| 1 | 1 | 0 | 1 | 1 | x | x | x | 0+2+1=03 |
| 1 | 1 | 1 | 0 | 0 | x | x | x | 0+3+0=03 |
| 1 | 1 | 1 | 0 | 1 | x | x | x | 0+3+1=10 |
| 2 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0+0+0=00 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0+0+1=01 |

Минимизацию переключательных функций(ПФ) проведём с помощью карт Вейча. На картах символом «x» отмечены наборы, на которых функция может принимать произвольное значение (безразличные наборы).

Для функции *S1* заполненная карта приведена на рисунке 3.1.

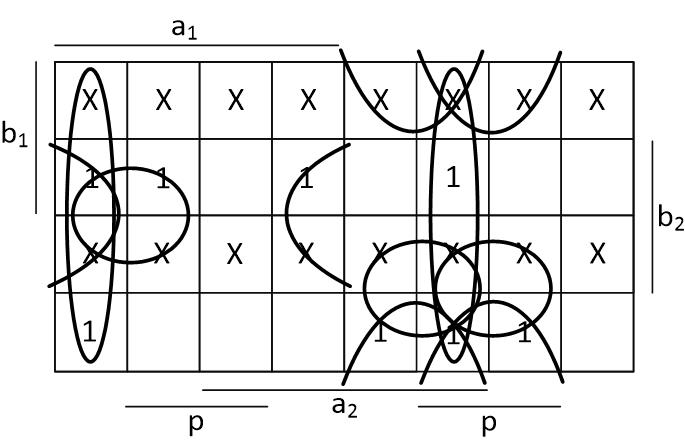


Рисунок 3.1 – Минимизация функции *S1* при помощи карты Вейча

Исходя из результатов минимизации *S1* могут быть получены следующие тупиковые формы:

*,*

*,*

*,*

*.*

Для реализации выхода *S1* на функциональной схеме будем использовать тупиковую форму . Для этого приведем ее к базису, заданному по условию.

.

Эффективность минимизации для всех переключательных функций будем оценивать, как отношение цен по Квайну до и после минимизации.

Тогда эффективность минимизации переключательной функции *S1*:

Функциональная схема выхода *S1* представлена в приложении Б.

Для функции *S2* заполненная карта приведена на рисунке 3.2.

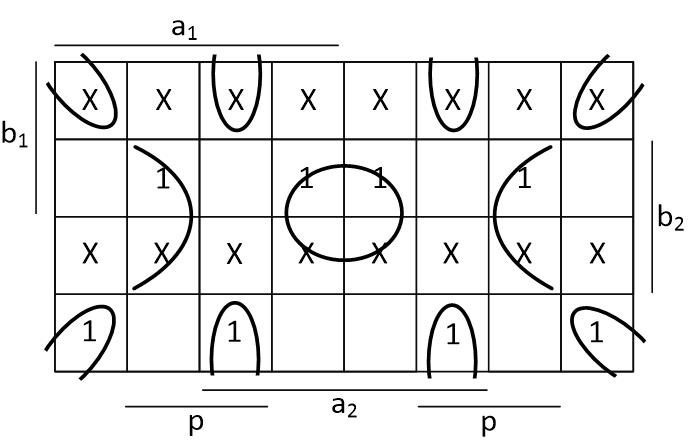


Рисунок 3.2 – Минимизация функции *S2* при помощи карты Вейча

Следовательно:

*.*

Эффективность минимизации переключательной функции *S2*:

Функциональная схема выхода *S2* представлена в приложении Б.

Для функции Пзаполненная карта приведена на рисунке 3.3.

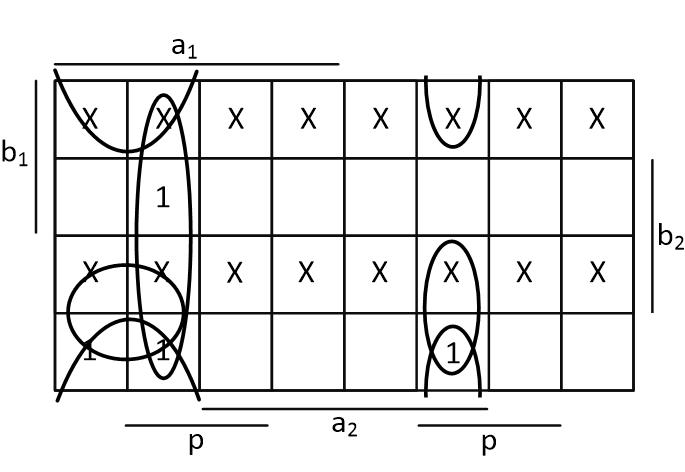


Рисунок 3.3 – Минимизация функции Ппри помощи карты Вейча

Исходя из результатов минимизации П могут быть получены следующие тупиковые формы:

*,*

*,*

*,*

*.*

Для реализации выхода П на функциональной схеме будем использовать тупиковую форму . Для этого приведем ее к базису, заданному по условию.

*.*

Эффективность минимизации переключательной функции П:

Функциональная схема выхода Ппредставлена в приложении Б.

# **3.2 Логический синтез одноразрядного четверичного умножителя**

ОЧУ – это комбинационное устройство с 5 двоичными входами (2 разряда из регистра Мн, два разряда из регистра Мт и управляющий вход h) и 4 двоичными выходами.

Принцип работы ОЧУ представлен с помощью таблицы истинности (таблица 3.2).

Разряды множимого закодированы: 0-11; 1-00; 2-01; 3-10.

Разряды множителя закодированы: 0-00; 1-01; 2-10; 3-11.

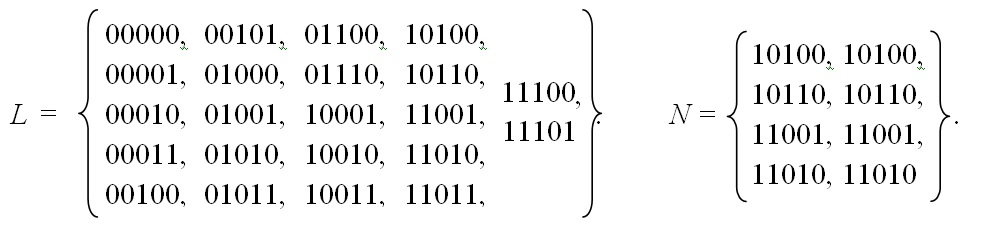
Управляющий вход h определяет тип операции: «0» – умножение закодированных цифр, поступивших на информационные входы; «1» – вывод на выходы без изменения значения разрядов, поступивших из регистра множимого.

В таблице 3.2 выделено 8 безразличных наборов, т.к. на входы ОЧУ из разрядов множителя не может прийти код «11».

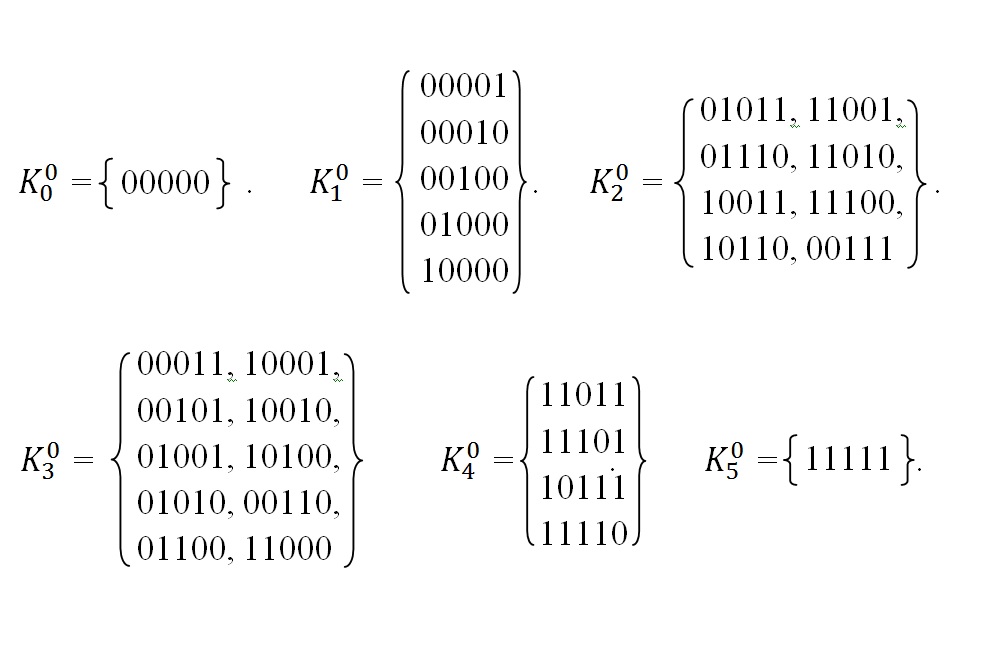
Таблица 3.2 – Таблица истинности ОЧУ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **X1** | **X2** | **Y1** | **Y2** | **h** | **P1** | **P2** | **P3** | **P4** | **Пример операции в четверичной с/с** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1\*0 = 00 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | Выход - код «01» |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1\*1= 01 |
| 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | Выход - код «01» |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1\*2 = 02 |
| 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | Выход - код «01» |
| 0 | 0 | 1 | 1 | 0 | x | x | x | x | 1\*3 = 03 |
| 0 | 0 | 1 | 1 | 1 | x | x | x | x | Выход - код «01» |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 2\*0 = 00 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | Выход - код «02» |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 2\*1 = 02 |
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | Выход - код «02» |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 2\*2 = 10 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | Выход - код «02» |
| 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 2\*3 = 12 |
| 0 | 1 | 1 | 1 | 1 | x | x | x | x | Выход - код «02» |
| 1 | 0 | 0 | 0 | 0 | x | x | x | x | 3\*0 = 00 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | Выход - код «03» |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 3\*1 = 03 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | Выход - код «03» |
| 1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 3\*2 = 12 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | Выход - код «03» |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 3\*3 = 21 |
| 1 | 0 | 1 | 1 | 1 | x | x | x | x | Выход - код «03» |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| 1 | 1 | 0 | 0 | 0 | x | x | x | x | 0\*0 = 00 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | Выход - код «00» |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0\*1 = 00 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | Выход - код «00» |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0\*2 = 00 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | Выход - код «00» |
| 1 | 1 | 1 | 1 | 0 | x | x | x | x | 0\*3 = 00 |
| 1 | 1 | 1 | 1 | 1 | x | x | x | x | Выход - код «00» |

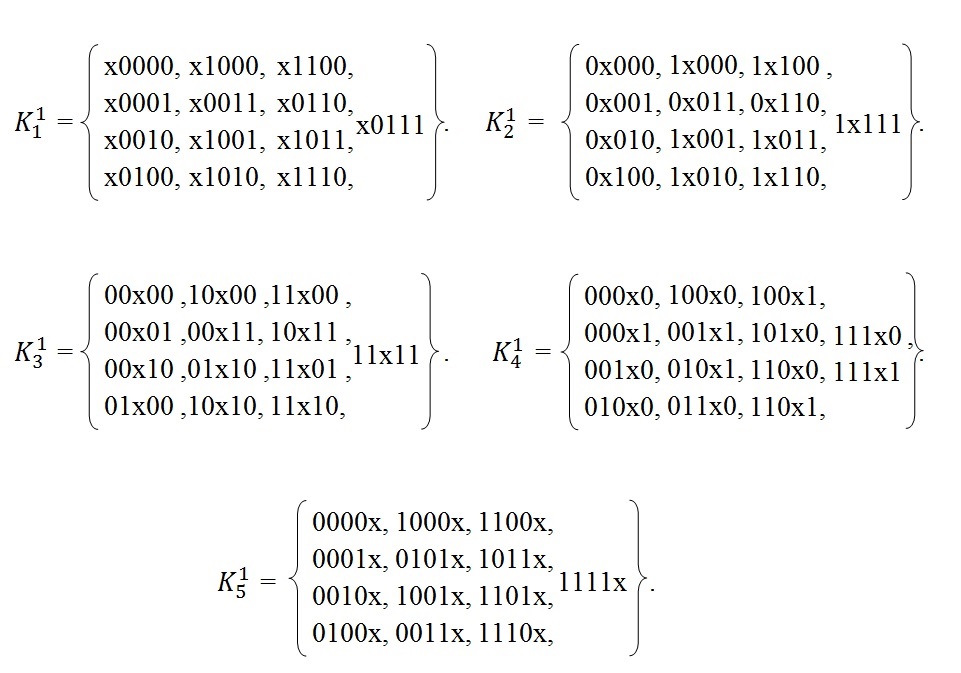
Минимизацию переключательной функции *Р3* проведем при помощи алгоритма Квайна - Мак-Класки. Для начала определим множества единичных единичных кубов (*L*) и множество безразличных наборов (*N*).



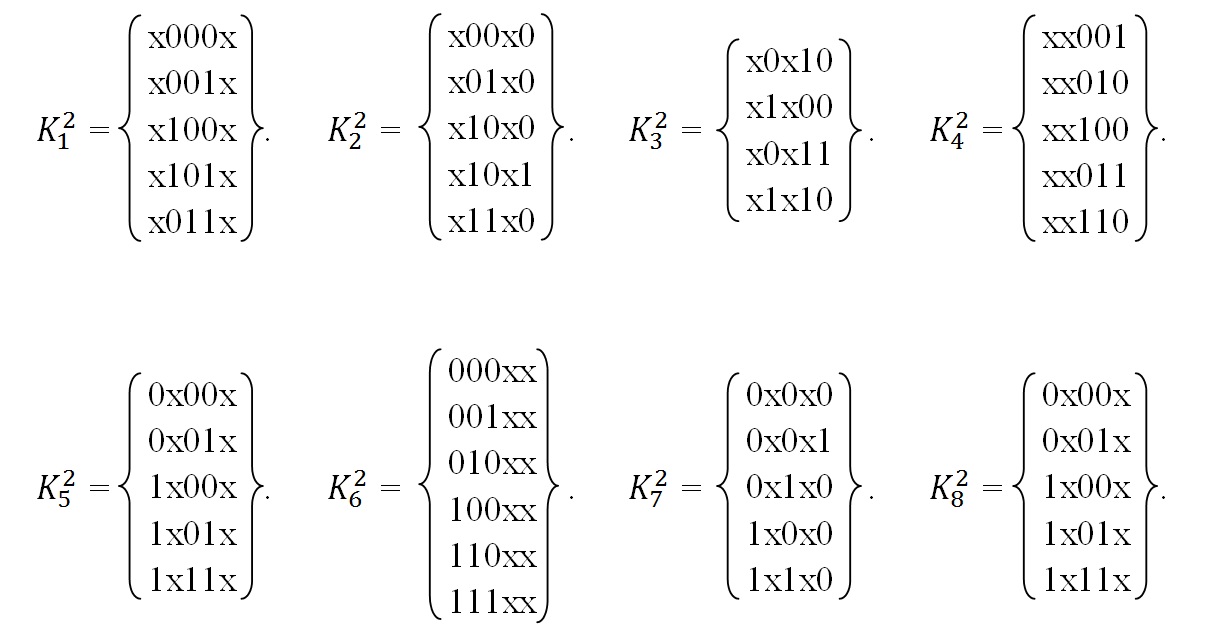
Выполним разбиение множеств на группы, содержащие одинаковое количество «1».



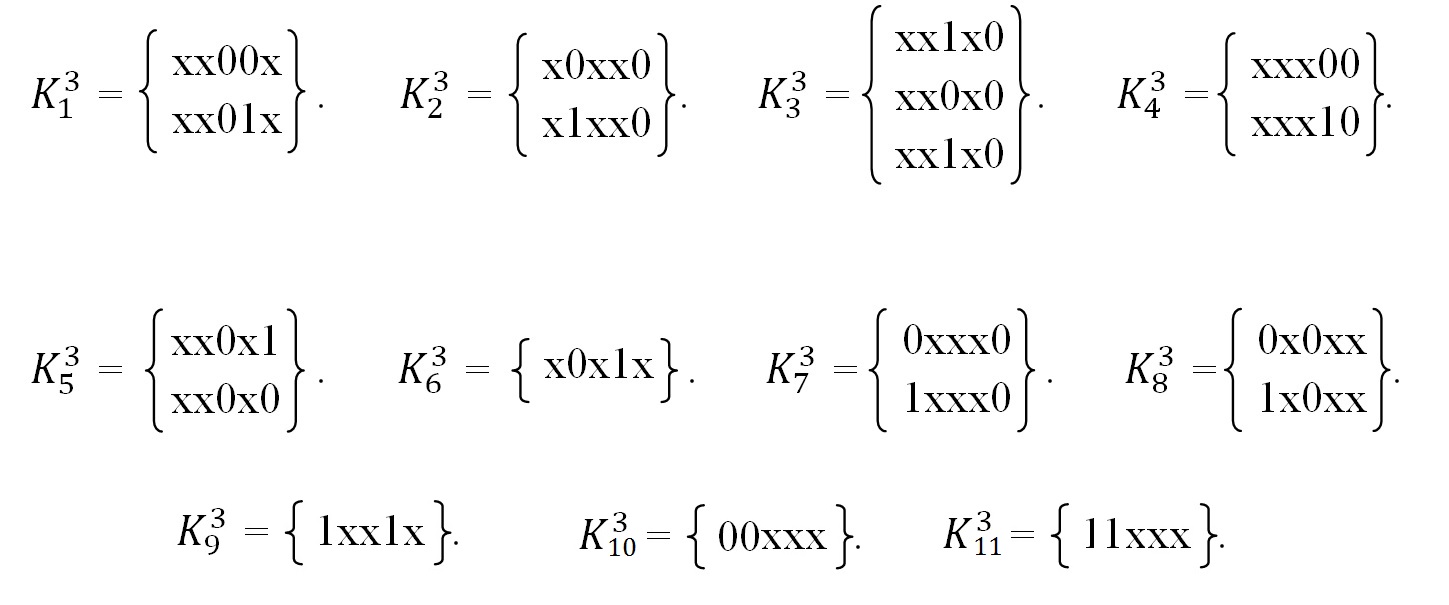
Теперь попарно склеим соседние (*Ki*  и *Ki+1*) множества.



На первом этапе алгоритма простых импликант не выявлено. Теперь проведем склеивание внутри полученных множеств.



На втором этапе простых импликант также не выявлено. Проводим склеивание внутри полученных множеств.

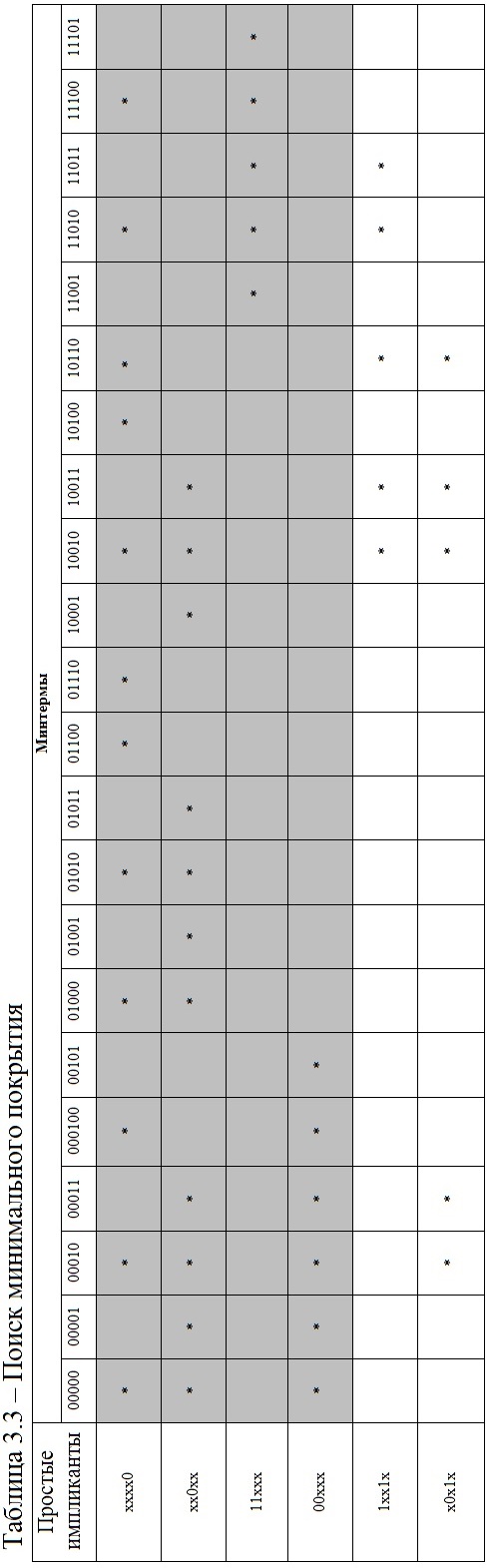


Выявлены следующие простые импликанты: 1xx1x, x0x1x, 00xxx, 11xxx. Продолжим склеивание в полученных множествах.

##### = { xххx0 }. = { хх0хх }.

Таким образом, множество простых импликант имеет вид : 1xx1x, x0x1x, 00xxx, 11xxx, xххx0, хх0хх.

Теперь построим импликантную таблицу (таблица 3.3) и найдем минимальное покрытие.

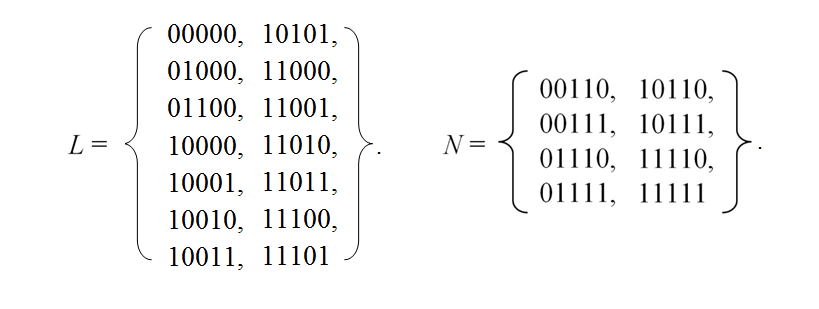


Следовательно:

Эффективность минимизации переключательной функции *P1*:

Т.к. таблица истинности для выхода *P2* полностью повторяет выход *P1* , то он будет иметь такую же минимальную функцию. Функциональная схема выходов *P1* и *P2* представлена в приложении В.

Минимизацию переключательной функции *Р3* проведем при помощи алгоритма извлечения (метода Рота). Для этого определим множество единичных кубов (*L*) и множество безразличных наборов (*N*)



Предварительное склеивание безразличных наборов (*N*) для упрощения алгоритма минимизации приведено на рисунке 3.4.

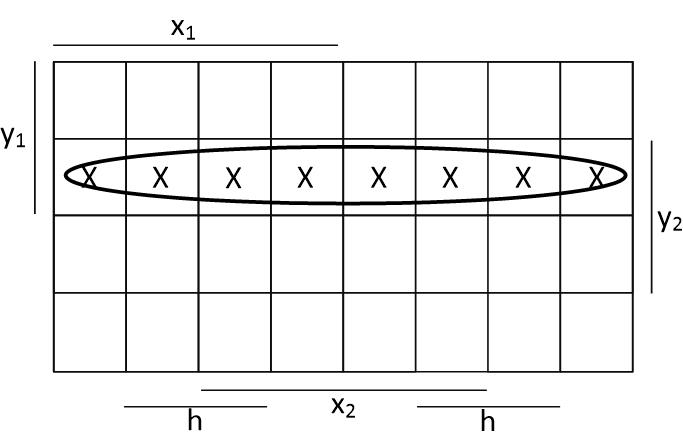
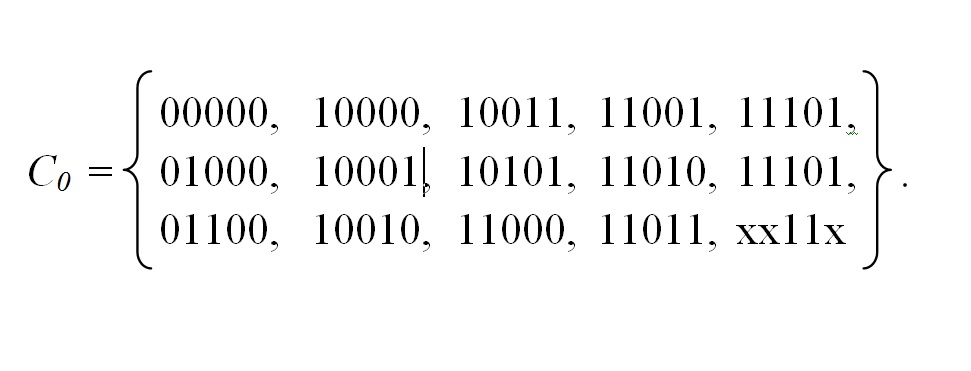


Рисунок 3.4 – Cклеивание множества *N* при помощи карты Вейча

Следовательно:

*N* = xx11x .

Сформируем множество *C0* = *L* ⋃ *N* :



Первым этапом алгоритма Рота является нахождение множества простых импликант.

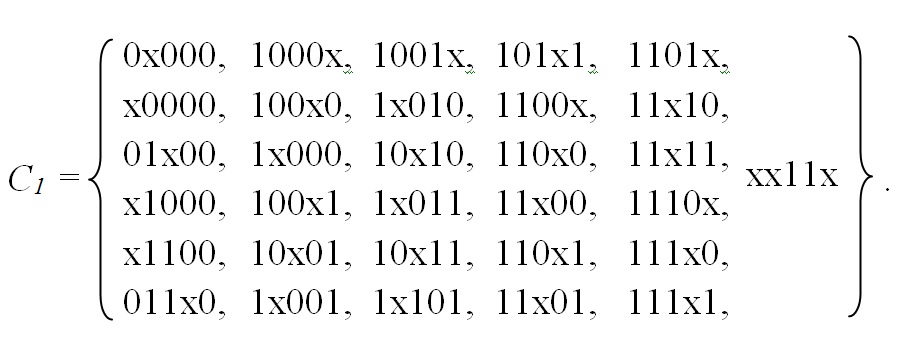
Для реализации этого этапа будем использовать операцию умножения (\*) над множествами *C0*, *C1* и т. д., пока в результате операции будут образовываться новые кубы большей размерности.

Первый шаг умножения (*C0*\* *C0*) приведён в таблице 3.4.

Таблица 3.4 – Поиск простых импликант (*C0* \* *C0*)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *C0\*C0* | 00000 | 01000 | 01100 | 10000 | 10001 | 10010 | 10011 | 10101 | 11000 | 11001 | 11010 | 11011 | 11100 | 11101 | xx11x |
| 00000 | - |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 01000 | 0y000 | - |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 01100 |  | 01y00 | - |  |  |  |  |  |  |  |  |  |  |  |  |
| 10000 | y0000 |  |  | - |  |  |  |  |  |  |  |  |  |  |  |
| 10001 |  |  |  | 1000y | - |  |  |  |  |  |  |  |  |  |  |
| 10010 |  |  |  | 100y0 |  | - |  |  |  |  |  |  |  |  |  |
| 10011 |  |  |  |  | 100y1 | 1001y | - |  |  |  |  |  |  |  |  |
| 10101 |  |  |  |  | 10y01 |  |  | - |  |  |  |  |  |  |  |
| 11000 |  | y1000 |  | 1y000 |  |  |  |  | - |  |  |  |  |  |  |
| 11001 |  |  |  |  | 1y001 |  |  |  | 1100y | - |  |  |  |  |  |
| 11010 |  |  |  |  |  | 1y010 |  |  | 110y0 |  | - |  |  |  |  |
| 11011 |  |  |  |  |  |  | 1y011 |  |  | 110y1 | 1101y | - |  |  |  |
| 11100 |  |  | y1100 |  |  |  |  |  | 11y00 |  |  |  | - |  |  |
| 11101 |  |  |  |  |  |  |  | 1y101 |  | 11y01 |  |  | 1110y | - |  |
| xx11x |  |  | 011y0 |  |  | 10y10 | 10y11 | 101y1 |  |  | 11y10 | 11y11 | 111y0 | 111y1 | - |

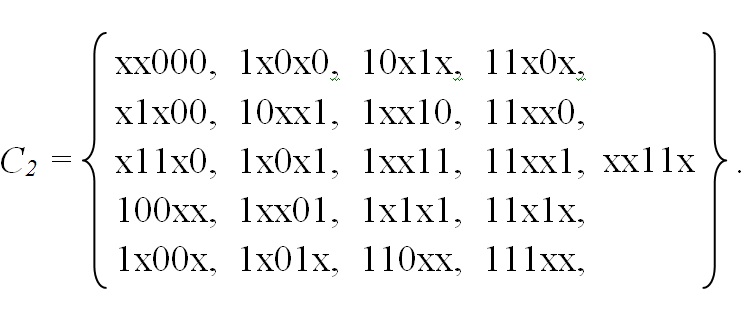
В результате этой операции сформируется новое множество кубов *C1*:



Множество *Z0* кубов, не участвовавших в образовании новых кубов, пустое.

В приложении Д приведён следующий шаг поиска простых импликант с помощью операции *C1* \* *C1*.

В результате образовалось множество *C2* кубов второй размерности:



Множество *Z1* кубов, не участвовавших в образовании новых кубов, пустое.

В таблице 3.5 приведен следующий шаг поиска простых импликант – операция *C2 \* C2*.

В результате этой операции сформируется новое множество кубов *C3*:

##### *C3* = { 1x0xx, 1xxx, 1xx1x, 11xxx, xx11x }.

Множество *Z2* кубов, не участвовавших в образовании новых кубов, имеет вид:

##### *Z2* = { xx000, x1x00, x11x0 }.

В таблице 3.6 приведен следующий шаг поиска простых импликант – операция *C3 \* C3*.

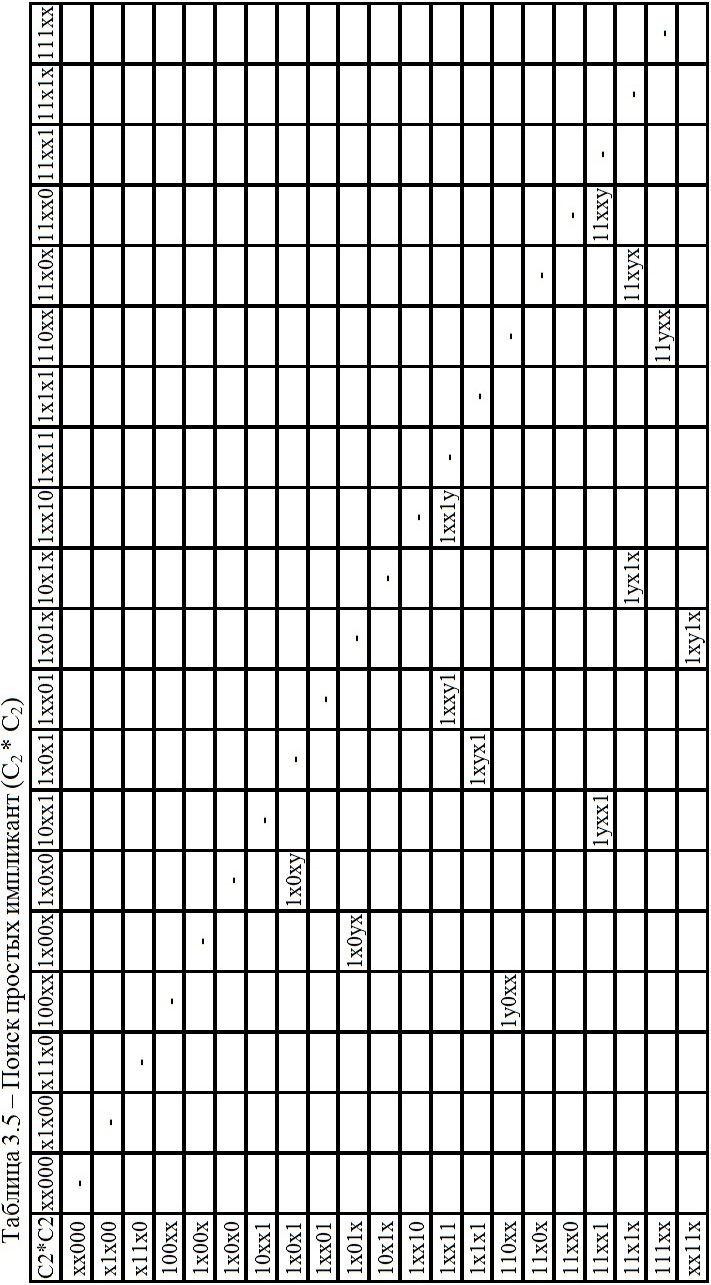


Таблица 3.6 – Поиск простых импликант (*C3 \* C3*)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| C3\*C3 | 1x0xx | 1xxx1 | 1xx1x | 11xxx |
| 1x0xx | - |  |  |  |
| 1xxx1 |  | - |  |  |
| 1xx1x |  |  | - |  |
| 11xxx |  |  |  | - |
| xx11x |  |  |  |  |

Новых кубов (четвёртой размерности) не образовалось.

Множество *Z3* кубов, не участвовавших в образовании новых кубов, имеет вид:

##### *Z3* = { 1x0xx, 1xxx1, 1xx1x, 11xxx, xx11x }.

Итоговое множество простых импликант имеет вид:

##### Z = Z2 ∪ Z3 =  { xx000, x1x00, x11x0, 1x0xx, 1xxx1, 1xx1x, 11xxx, xx11x }.

Следующий этап – поиск *L*-экстремалей на множестве простых импликант (таблица 3.7). Для этого используется операция # (решётчатое вычитание).

Таблица 3.7 – Поиск *L*-экстремалей

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *z*#(*Z-z*) | xx000 | x1x00 | x11x0 | 1x0xx | 1xxx1 | 1xx1x | 11xxx | xx11x |
| xx000 | - | x1100 | x11x0 | 1x01x 1x0x1 | 1xxx1 | 1xx1x | 111xx 11x1x 11xx1 | xx11x |
| x1x00 | x0000 | - | x1110 | 1x01x 1x0x1 | 1xxx1 | 1xx1x | 1111x 111x1 11x1x 11xx1 | xx11x |
| x11x0 | x0000 | Ø | - | 1x01x 1x0x1 | 1xxx1 | 10x1x 1x01x 1xx11 | 11111 111x1 1101x 11x11 11xx1 | x011x xx111 |
| 1x0xx | 00000 | Ø | x1110 | - | 1x1x1 | 1011x 1x111 | 11111 111x1 11111 111x1 | x011x xx111 |
| 1xxx1 | 00000 | Ø | x1110 | 1x010 | - | 10110 | Ø | 0011x x0110 0x111 |
| 1xx1x | 00000 | Ø | 01110 | Ø | 1x101 | - | Ø | 0011x 00110 0x111 |
| 11xxx | 00000 | Ø | 01110 | Ø | 10101 | 10110 | - | 0011x 00110 0x111 |
| xx11x | 00000 | Ø | Ø | Ø | 10101 | Ø | Ø | - |

В результате операции образовано множество (*Е*) *L*-экстремалей:

##### *E* = { xx000, 1xxx1, 11xxx }.

Теперь необходимо проверить, нет ли среди *L*-экстремалей таких, что стали *L*-экстремалями за счет безразличных наборов (таблица 3.8).

Таблица 3.8 – Проверка *L*-экстремалей

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *z*#*(Z-z*) ∩ *L* | 00000 | 01000 | 01100 | 10000 | 10001 | 10010 | 10011 | 10101 | 11000 | 11001 | 11010 | 11011 | 11100 | 11101 |
| 00000 | 00000 | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø |
| 10101 | Ø | Ø | Ø | Ø | Ø | Ø | Ø | 10101 | Ø | Ø | Ø | Ø | Ø | Ø |
| 0011x | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø |
| 00110 | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø |
| 0x111 | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø |

По результатам таблицы 3.8 *L*-экстремалями, не связанными с безразличными наборами, стали кубы хх000 и 1ххх1 (остаток от вычитания из них всех остальных простых импликант – 00000 и 10101 соответственно – относятся к множеству единичных наборов *L* исходного задания функции). Эти кубы обязательно должны войти в минимальное покрытие.

Теперь проанализируем, есть ли среди исходных единичных кубов множества (*L*) те, что не покрыты *L*-экстремалями (таблица 3.9).

Таблица 3.9 – Проверка *L*-экстремалей

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *L#E* | 00000 | 01000 | 01100 | 10000 | 10001 | 10010 | 10011 | 10101 | 11000 | 11001 | 11010 | 11011 | 11100 | 11101 |
| xx000 | Ø | Ø | 01100 | Ø | 10001 | 10010 | 10011 | 10101 | Ø | 11001 | 11010 | 11011 | 11100 | 11101 |
| 1xxx1 | Ø | Ø | 01100 | Ø | Ø | 10010 | Ø | Ø | Ø | Ø | 11010 | Ø | 11100 | Ø |

Множество кубов, не покрытых *L*-экстремалями, имеет вид:

*L’* = { 01100, 10010, 11010, 11100 }.

Чтобы их покрыть, воспользуемся множеством простых импликант, не являющихся *L*-экстремалями (таблица 3.10).

Таблица 3.10 – Покрытие оставшихся кубов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *L’∩ Ž* | 01100 | 10010 | 11010 | 11100 |
| x1x00 | 01100 | Ø | Ø | 11100 |
| x11x0 | 01100 | Ø | Ø | 11100 |
| 1x0xx | Ø | 10010 | 11010 | Ø |
| 1xx1x | Ø | 10010 | 11010 | Ø |
| 11xxx | Ø | Ø | 11010 | 11100 |
| xx11x | Ø | Ø | Ø | Ø |

Исходя из результатов минимизации *P3* могут быть получены следующие тупиковые формы:

*,*

*,*

*,*

*.*

Для реализации выхода *P3* на функциональной схеме будем использовать тупиковую форму . Для этого приведем ее к базису, заданному по условию.

*,*

Эффективность минимизации переключательной функции *P3*:

Функциональная схема выхода *P3* представлена в приложении В.

Минимизацию переключательной функции *P4* проведем при помощи карты Вейча

Для функции *P4* заполненная карта приведена на рисунке 3.5.

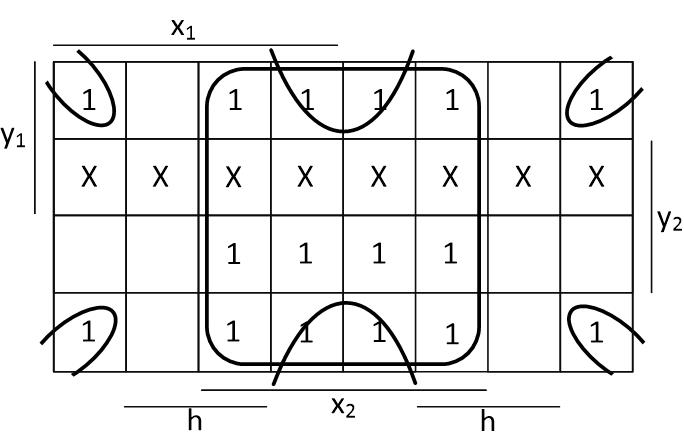


Рисунок 3.5 – Минимизация функции *P4* при помощи карты Вейча

Следовательно:

Эффективность минимизации переключательной функции *P4*:

Функциональная схема выхода *P4* представлена в приложении В.

**4. СИНТЕЗ КОМБИНАЦИОННЫХ СХЕМ НА ОСНОВЕ МУЛЬТИПЛЕКСОРА**

Мультиплексор – это логическая схема, имеющая n информационных входов, m управляющих входов и один выход. При этом должно выполняться условие n = .

Принцип работы мультиплексора состоит в следующем:

На выход мультиплексора может быть пропущен без изменений любой (один) логический сигнал, поступающий на один из информационных входов.

Порядковый номер информационного входа, значение которого в данный момент должно быть передано на выход, определяется двоичным кодом, поданным на управляющие входы.

Переключательные функции пяти переменных можно просто реализовать на мультиплексоре «один из восьми». Такой мультиплексор имеет три управляющих сигналов и, следовательно, два в степени три, т.е. восемь информационных. Реализация ОЧС потребует три ПФ.

Синтез логических схем для ПФ ОЧC приведён в таблице 4.1.

Таблица 4.1 – Таблица истинности для синтеза ПФ

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **N** | ***a1*** | ***a2*** | ***b1*** | ***b2*** | ***p*** | **П** | **Выход** | **S1** | **Выход** | **S2** | **Выход** |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | «0» | 0 |  | 1 |  |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | x | x | x |
| 0 | 0 | 0 | 0 | 1 | 1 | x | x | x |
| 1 | 0 | 0 | 1 | 0 | 0 | x | «0» | x | «0» | x |  |
| 1 | 0 | 0 | 1 | 0 | 1 | x | x | x |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 |  | 1 | «1» | 0 |  |
| 2 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 2 | 0 | 1 | 0 | 1 | 0 | x | x | x |
| 2 | 0 | 1 | 0 | 1 | 1 | x | x | x |
| 3 | 0 | 1 | 1 | 0 | 0 | x | «0» | x |  | x |  |
| 3 | 0 | 1 | 1 | 0 | 1 | x | x | x |
| 3 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |
| **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
| 4 | 1 | 0 | 0 | 0 | 0 | 1 | «1» | 1 |  | 1 |  |
| 4 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 4 | 1 | 0 | 0 | 1 | 0 | x | x | x |
| 4 | 1 | 0 | 0 | 1 | 1 | x | x | x |
| 5 | 1 | 0 | 1 | 0 | 0 | x |  | x | «1» | x |  |
| 5 | 1 | 0 | 1 | 0 | 1 | x | x | x |
| 5 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 5 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 6 | 1 | 1 | 0 | 0 | 0 | 0 | «0» | 0 | «0» | 0 |  |
| 6 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 6 | 1 | 1 | 0 | 1 | 0 | x | x | x |
| 6 | 1 | 1 | 0 | 1 | 1 | x | x | x |
| 7 | 1 | 1 | 1 | 0 | 0 | x | «0» | x |  | x |  |
| 7 | 1 | 1 | 1 | 0 | 1 | x | x | x |
| 7 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 7 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

Схема ОЧС на основе мультиплексоров представлена в приложении Г.

**5. ОЦЕНКА РЕЗУЛЬТАТОВ РАЗРАБОТКИ**

Время умножения на n разрядов равно:

,

где – время умножения на один разряд.

Время умножения на один разряд состоит из: времени сдвига (), времени работы преобразователя множителя (), времени формирования дополнительного кода (), времени работы ОЧУ (), времени работы цепочки из n ОЧС () и времени срабатывания аккумулятора (). Время работы ОЧУ и ОЧС равны, соответственно,

,

где – время работы одного логического элемента.

Таким образом

.

.

# **заключение**

Цель данного курсового проекта была достигнута. Все знания, полученные по дисциплине «Арифметические и логические основы вычислительной техники», были успешно применены на практике в ходе курсового проектирования. В частности, в ходе построения сумматора-умножителя.

Была произведена минимизация булевых функций различными способами (картами Карно (Вейча) и алгоритмом извлечения (Рота)). В качестве главного достоинства карт Карно (Вейча) можно выделить простоту и минимальные затраты времени. Однако применение данного способа для функций многих переменных будет затруднительно. Алгоритмом извлечения же можно минимизировать функции от любого числа переменных. Данный алгоритм является машинно-ориентированным и легко программируемым, но как минус можно отметить его сложность и громоздкость при ручном исполнении.

При построении функциональных схем были использованы различные логические базисы, что позволило отточить навыки булевой алгебры, а частности, использование правила де Моргана.

Использование такого устройства как мультиплексор для построения комбинационной схемы значительно упростило ее и позволило свести к минимуму использование логических элементов в схеме.

Для определения частоты тактового импульса было рассчитано время умножения на один разряд.

Таким образом, все задачи выполнены. Но на этом разработка данного устройства не завершена. В будущем необходимо предусмотреть округление результата сложения двух чисел и спроектировать устройство управления.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

# 

1. Луцик, Ю. А. Учебное пособие по курсу «Арифметические и логические основы вычислительной техники» / Ю. А. Луцик, И. В. Лукьянова, М. П. Ожигина. – Минск : МРТИ, 2001. – 77 с.

2. Луцик, Ю. А. Арифметические и логические основы вычислительной техники : метод. пособие / Ю. А. Луцик, И. В. Лукьянова. – Минск : МРТИ, 2004. – 35 с.

3. Искра, Н. А. Арифметические и логические основы вычислительной техники : пособие / Н. А. Искра, И. В. Лукьянова, Ю. А. Луцик. – Минск : БГУИР, 2016. – 75 с.

# **Приложение А**

*(обязательное)*

Сумматор-умножитель первого типа. Схема электрическая структурная

# **Приложение Б**

*(обязательное)*

Одноразрядный четверичный сумматор. Схема электрическая функциональная

# **Приложение В**

*(обязательное)*

Одноразрядный четверичный умножитель. Схема электрическая функциональная

# **Приложение Г**

*(обязательное)*

Одноразрядный четверичный сумматор. Реализация на мультиплексорах. Схема электрическая функциональная

# **Приложение Д**

*(обязательное)*

Поиск простых импликант *C1 \* C1*

# **Приложение Е**

*(обязательное)*

Сумматор-умножитель. Ведомость документов