УДК 004.021 DOI: 10.31114/2078-7707-2022-2-26-30

# Генерация библиотек логических функций

А.А. Лялинский

Институт проблем проектирования в микроэлектронике РАН, г. Москва

zelyal@inbox.ru

***Аннотация* — Рассматриваются вопросы создания библиотек логических функций, используемых на этапах проектирования электронной аппаратуры с применением программируемых логических интегральных схем. Предложен эффективный алгоритм, позволяющий решать больший круг вопросов и получать библиотеки большего размера по сравнению с ранее предложенным алгоритмом.**

***Ключевые слова* — логическая функция, булева переменная, грамматика логического выражения, библиотека логических функций.**

## I. Введение

Библиотеки логических функций (БЛФ) используются в маршруте проектирования электронной аппаратуры с применением программируемых логических интегральных схем (ПЛИС) [1]. Несмотря на то, что в конкретной ПЛИС состав элементов БЛФ ограничен структурой ядра ПЛИС, не позволяющего реализовать любую логическую функцию, задача генерации максимально полного набора функций, имеющих различное синтаксическое описание, представляет определенный интерес как с научной, так и с практической точки зрения – то, что нельзя реализовать на имеющейся ПЛИС, возможно, пригодится для ее более продвинутой версии.

Автором ранее проводились работы по этому направлению [2, 3]. Исследование показало недостатки использованных в ранних работах подходов, а именно:

- каждая из переменных функции может быть использована в выражении только один раз;
- реализация программной части на языке PHP и расположение ее на веб-сервере не позволяет эффективно использовать компьютерные время и память.

В целом эти ограничения привели к тому, что в библиотеке отсутствовали некоторые достаточно часто используемые функции, например функция мультиплексора f=a&c|b&!c (здесь **a** и **b** – переключаемые переменные, **c** – вход управления). Выявленные недостатки привели к разработке новой, более совершенной версии алгоритма создания БЛФ, представленной в этой работе.

Далее в статье вначале рассмотрены базовые понятия, связанные с определением логической функции; описаны этапы работы предлагаемого алгоритма; представлены результаты генерации БЛФ для различных значений параметров задачи.

## II. Способы задания логической функции

Вначале рассмотрим определение понятий «логическая функция» и «таблица истинности».

В общем случае под *логической функцией* понимается отображение $B^n \to B$, где $B = \{0, 1\}$ [4, 5]. Элементы $B^n$ называются булевыми векторами. Логическая функция размерности **n** полностью определяется заданием своих значений на своей области определения, т.е. на всех булевых векторах размерности **n**. Число таких векторов равно $2^n$. Так как функция может принимать значение 0 или 1, то количество всех функций размерности n равно $2^{(2^n)}$.

*Таблица истинности* (ТИ) размерности **n** представляет собой реальную таблицу из n+1столбцов и $2^n$ строк. В ней для каждого варианта сочетаний значений аргументов определено соответствующее ему значение функции (см. пример в табл. 1 для случая n=3). В общем случае строки ТИ могут следовать в любом порядке, но для определенности далее будем использовать только порядок, примененный в табл. 1 – быстрее всего изменяется последний аргумент, самый медленный – первый.

Таблица 1

*Таблица истинности для n=3*

| $x_1$ | $x_2$ | $x_3$ | $f(x_1,x_2,x_3)$ |
|---|---|---|---|
| 0 | 0 | 0 | f(0,0,0) |
| 0 | 0 | 1 | … |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

В практическом плане использование логической функции в ее строгом определении на основе таблицы истинности неудобно, более общепринятым является определение ее в синтаксической форме. Для этого следует определиться с обозначениями аргументов и

операторов логической функции. Достаточно часто можно встретить следующую форму:

**a**, **b**, **c**, … - аргументы логической функции (будем также для них использовать понятие “независимые переменные»);

**&**, |, ^, **!** - логические операции «И», «ИЛИ», «исключающее ИЛИ», «отрицание».

В данной работе используется немного модифицированные по сравнению с приведенными выше множествами, а именно: операция отрицания «!» из состава операторов удалена, а множество аргументов расширено их формой с отрицанием:

**a**, **b**, **c**, **!a**, **!b**, **!c**, … - аргументы логической функции;

**&**, |, ^ - логические операции «И», «ИЛИ», «исключающее ИЛИ».

Это позволяет использовать достаточно простую форму задания логической функции, использующую **n** операторов и **n+1** переменных:

$$f = v_1[O_1v_2 \dots [O_nv_{n+1}]],$$

где:

- **v** – либо переменная (в том числе и с отрицанием), либо взятое в скобки логическое выражение «(expr)», либо его отрицание «!(expr)»;
- **O** – знак операции;
- **[]** – мета-скобки, показывающие необязательность синтаксической конструкции.

В завершение введем также понятие *«битовой маски функции»* (БМФ), под которой будем понимать последовательность битов, взятых из последнего столбца таблицы истинности функции (столбец результатов) в следующем порядке:

$$f(1,1,\dots,1) \dots f(0,0,\dots,0),$$

т.е. результат верхней строки в ТИ – это самый правый бит, самой нижней – самый левый. Битовая маска может быть представлена как в двоичной, так и в шестнадцатеричной форме. В последнем случае слева могут возникать незначащие нули.

## III. Постановка задачи

После определения исходных понятий можно сформулировать решаемую в данной работе задачу: *для заданного максимального числа аргументов **n** получить максимально большие наборы логических функций для всех $n_i = \{1, 2, \dots, n\}$.*

Несмотря на то, что некоторые логические функции из создаваемой библиотеки путем преобразований могут быть выражены через другие, более элементарные функции этой же библиотеки, исходное описание всех функций библиотеки может быть представлено только сочетанием независимых переменных, знаков операций и скобок.

## IV. Этапы решения задачи

Предложенный алгоритм генерации библиотеки логических функций состоит из следующих этапов:

1. Для заданного **i** из диапазона [1, n] создаются строки-шаблоны будущих логических выражений.
2. Для каждого шаблона определяется количество вариантов изменения составляющих его компонентов.
3. Формируются первичные строки логических функций.
4. Удаляются дубликаты.
5. Для каждой логической функции вычисляется БМФ.
6. Функции с одной и той же БМФ группируются в отдельные наборы.
7. В каждой из таких групп выделяется основная логическая функция, остальные считаются её алиасами (копиями).
8. Группы сортируются по какому-либо признаку, например по БМФ или по количеству использованных переменных.
9. Конец алгоритма.

Рассмотрим некоторые этапы подробнее.

### *A. Генерация шаблонов*

Для каждого **i** из множества {1, …, n} создаем первичный набор шаблонов. Каждый из шаблонов $\mathbf{p_i}$ имеет вид: $\mathbf{p_i} = v_1[O_1v_2 \dots]$, где **v** обозначает независимую переменную, а **O** – операцию. Например, для n=3 набор шаблонов имеет вид, приведенный в табл. 2.

Таблица 2

*Набор шаблонов для n=3*

| ***i*** | $\boldsymbol{p_i}$ |
|---|---|
| 0 | *v* |
| 1 | *vOv* |
| 2 | *vOvOv* |
| 3 | *vOvOvOv* |

Далее, учитывая что скобки имеют наивысший приоритет при синтаксическом разборе и способны добавить новую функциональность в выражение, из первичных строк шаблона создаем дополнительные формы со скобками (там, где это возможно). Например, строка «vOvOv» дает две дополнительных формы:

vOvOv → (vOv)Ov, vO(vOv).

Для достаточно длинных строк можно было бы вводить следующие, более высокие уровни скобок, но в данной работе это не делалось – практика показала,

что наборы большой размерности получаются и без этой эвристики.

*B. Формирование множества вариаций*

Для каждого из шаблонов $p_i$ множества [1, …, n] необходимо определить точное количество вариантов реализаций $v_i$ или $O_i$. Если для $O_i$ с этим нет проблем – мы заранее знаем состав разрешенных операторов (два: «**&**» и «|» или три: «**&**», «|» и «^»), то для $v_i$ это зависит от числа «v» в шаблоне. Так, для шаблона vOv каждое из «v» может принять 4 значения: a, b, !a, !b. Для шаблона vOvOv таких значений уже 6: a, b, c, !a, !b, !c.

Далее на основе полученного числа вариаций для каждого шаблона создается множество вариаций. Так, шаблон vOv имеет количество вариаций по позициям {4, 3, 4} (4 – количество переменных, 3 количество операторов). Для него множество вариаций представляется следующим образом:

| | | | | |
|---|---|---|---|---|
| 0,0,0 | 0,1,0 | | | 3,2,0 |
| 0,0,1 | 0,1,1 | | | 3,2,1 |
| 0,0,2 | 0,1,2 | … | … | 3,2,2 |
| 0,0,3 | 0,1,3 | | | 3,2,3 |

Здесь каждое число – это индекс (начиная с 0) переменной или операции. Всего мы имеем 4x3x4 → 48 вариантов.

Теперь, на основе индексов мы создаем реальные строки логических функций. Примеры перехода от индексов к реальным строкам: 0,0,1 → a&b; 1,1,0 → b|a (используется индексация переменных: a,b,c,… и операторов: &|^).

Одна из проблем предлагаемого алгоритма – это получение огромного числа вариантов функций, которые надо обработать. Это создает как временные проблемы (большое время работы алгоритма), так и проблемы с памятью – если не предпринимать специальных мер, то достаточно быстро мы упираемся в нехватку памяти ОЗУ. Поэтому по возможности еще на ранних этапах отсекаются заведомо ненужные варианты. Например, некоторые строки переименовываются так, чтобы переменные «возрастали». Строку «b&a» мы заменяем на равноценную ей «a&b».

Здесь же можно было бы убрать константные выражения типа «a&a&a», но реально затраты на просмотр и выявление таких конструкций не оправдывают получаемого эффекта.

*C. Вычисление битовой маски функции*

Для каждой полученной логической функции необходимо вычислить значения последнего столбца таблицы истинности (столбца результатов). К сожалению, в стандарте широко распространенного языке C [6] отсутствует аналог функции «eval», имеющейся в скриптовых языках (например, в PHP [7]), позволяющей вычислять логические функции «в одну строку». В связи с этим пришлось подготовить дополнительное программное обеспечение для вычисления значения логической функции.

Разработана грамматика описания логического выражения (рис. 1), в которой приоритет операций заложен в правила грамматики. Её не конца строгое описание приведено ниже, в нем EOS – это символ конца строки, [] – мета-скобки, показывающие необязательность заключенной в них конструкции. Для данной грамматики подготовлены парсер и вычислитель выражения. Переменные **a,b,c** получают значения 0 и 1 в соответствии с таблицей истинности; результат, полученный в вычислителе, формирует БМФ.

```
логическое_выражение  →  OR_выражение EOS
        OR_выражение  →  XOR_выражение [ | OR_выражение]
       XOR_выражение  →  AND_выражение [ ^ XOR_выражение]
       AND_выражение  →  NOT_выражение [ & AND_выражение]
       NOT_выражение  →   [ ! ] терм
                терм  →  имя
                      →  (OR_выражение)
                имя   →  a,b,c,…,0,1
```

**Рис. 1. Грамматика логического выражения**

*D. Обработка результатов вычисления выражения*

Полученные битовые маски позволяют однозначно определить тождественные функции, имеющие на одном и том же наборе исходных данных одинаковые результаты по таблице истинности. Мы опускаем здесь вопрос выявления симметричных функций, у которых результаты совпадают при перестановке строк в таблице истинности. Это требует определенных программных усилий, но в итоге имеет лишь небольшой результирующий эффект.

## V. Оценка разработанной программы

В таблице 3 приведены результаты работы созданного программного обеспечения. В качестве параметров использовались максимально разрешенное в выражении количество переменных и операторов. Показатели качества работы – количество полученных функций и потраченное CPU-время. Степень полноты библиотеки вычисляется как процент от $\mathbf{2^{2^n}}$ - максимальное количество позиций (строк) в таблице истинности для **n** переменных. Для простоты две вырожденные константные функции: f(x) = 0 и f(x)=1 также включены в это число. Так как начиная с nv=4

максимальное количество функций стремительно растет и степень полноты становится слишком малой, то для этих значений параметра данный показатель не вычислялся.

Таблица 3

*Размер БЛФ (количество логических функций) для различных параметров логического выражения*

| ***Максимальное количество различных переменных*** | ***Количество использованных переменных*** | ***Максимальное количество операторов в выражении*** | | | | | |
|---|---|---|---|---|---|---|---|
| | | **1** | **2** | **3** | **4** | **5** | **6** |
| **2 (макс. #функций=16)** | **1** | 2 | 2 | 2 | повтор данных столбца 3 | | |
| | **2** | 3 | 5 | 7 | | | |
| | ***в сумме*** | 5 | 7 | 9 | | | |
| | ***степень полноты БЛФ*** | 31% | 44% | 56% | | | |
| | ***время CPU, сек*** | < 1 | < 1 | < 1 | | | |
| | | | | | | | |
| **3 (макс. #функций=256)** | **1** | 2 | 2 | 2 | 2 | 2 | повтор данных столбца 5 |
| | **2** | 3 | 5 | 7 | 7 | 7 | |
| | **3** | | 11 | 30 | 58 | 78 | |
| | ***в сумме*** | 5 | 18 | 39 | 67 | 87 | |
| | ***степень полноты БЛФ*** | 2% | 7% | 15% | 26% | 34% | |
| | ***время CPU, сек*** | < 1 | < 1 | < 1 | 2 | 48 | |
| | | | | | | | |
| 4 **(макс. #функций=65536)** | **1** | 2 | 2 | 2 | 2 | 2 | |
| | **2** | 3 | 5 | 7 | 7 | 7 | |
| | **3** | | 11 | 26 | 53 | 71 | |
| | ***4*** | | | 48 | 247 | 671 | |
| | ***в сумме*** | 5 | 18 | 83 | 309 | 751 | |
| | ***время CPU, сек*** | < 1 | < 1 | < 1 | 9 | 273 | |
| | | | | | | | |
| 5 **(макс. #функций=*4.29*10^10^*)** | **1** | 2 | 2 | 2 | 2 | 2 | |
| | **2** | 3 | 5 | 7 | 7 | 7 | |
| | **3** | | 11 | 26 | 53 | 71 | |
| | ***4*** | | | 48 | 231 | 626 | |
| | ***5*** | | | | 166 | 1243 | |
| | ***в сумме*** | 5 | 18 | 83 | 459 | 1949 | |
| | ***время CPU, сек*** | < 1 | < 1 | < 1 | 29 | 1356 | |
| | | | | | | | |

Некоторые позиции в правой колонке таблицы не вычислялись, так как для большого количества переменных и операторов размер множества проверяемых вариантов выражений стремительно растет (уходя в сотни миллионов и миллиарды), что значительно увеличивает время расчета. В то же время практического значения эти сочетания не имеют.

Степень полноты в табл. 3 оценивалась по отдельности для каждого количества переменных. Реально для заданного **n** строки таблицы истинности заполняются выражениями с количеством переменных от 1 до **n** (например, первые две строки – это простейшие функции f=a и f=!a). Поэтому в табл. 4 приведены более точные оценки степени полноты для различных **n** при условии заполнения ТИ функциями с различным **n**.

Таблица 4

*Заполнение таблицы истинности*

| ***n*** | ***количество функций*** |
|---|---|
| 2 | *9 (из 16)* |
| 3 | *87 (из 256)* |
| 4 | *751 (из 65536)* |
| 5 | *1949 (из более 4 млрд.)* |

## *VI.* Выводы

Разработана методика генерации логических выражений и построения на их основе библиотеки логических функций. По сравнению с предыдущей работой новая методика обладает большей универсальностью, что позволяет создавать библиотеки логических функций с большим наполнением.

ЛИТЕРАТУРА

[1] Угрюмов Е. П. Программируемые логические матрицы, программируемая матричная логика, базовые матричные кристаллы / Цифровая схемотехника. Учеб. пособие для вузов. Изд.2, БХВ-Петербург, 2004. С. 357.

[2] Лялинский А.А. Генерация больших наборов логических функций для систем автоматизации проектирования цифровых интегральных схем // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. № 1. С. 9-15.

[3] Лялинский А.А. ПЛИС-ориентированная библиотека логических функций // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 1. С. 15-19. doi:10.31114/2078-7707-2020-1-15-19.

[4] Алексеев В. Б., Поспелов А.Д. Дискретная математика — М.: Изд. отд. фак. Вычислит. математики и кибернетики МГУ им. М. В. Ломоносова, 2002. — 44 с.

[5] Игошин В. И. Математическая логика и теория алгоритмов. — 2-е изд., стереотип.. — М.: Издательский центр «Академия», 2008. — 448 с.

[6] Керниган Б., Ритчи Д. Язык программирования Си = The C programming language. — 2-е изд. — М.: Вильямс, 2007. — С. 304. — ISBN 0-13-110362-8.

[7] Котеров Д.В., Симдянов И.В.. PHP 7. — СПб.: «БХВ-Петербург», 2017. — С. 1088. — ISBN 978-5-9775-3725-4.


# Generation of Logical Function Libraries


A.A. Lyalinsky

Institute for Design Problems in Microelectronics of RAS, zelyal@inbox.ru



***Abstract* — The issues of creating libraries of logical functions used at the design stages of electronic equipment using programmable logic integrated circuits are considered. An efficient algorithm is proposed that allows solving a larger range of issues and obtaining libraries of a larger size compared to the previously proposed algorithm.**



***Keywords* — logical function, Boolean variable, grammar of logical expression, library of logical functions.**



REFERENCES

[1] Ugrjumov E. P. Programmiruemye logicheskie matricy, programmiruemaja matrichnaja logika, bazovye matrichnye kristally / Cifrovaja shemotehnika. Ucheb. posobie dlja vuzov. Izd.2, BHV-Peterburg, 2004. S. 357.

[2] Lyalinsky A.A. Generation of large sets of logical functions for digital integrated circuits CAD systems // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2016. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2016. Part 1. P. 9-15.

[3] Lyalinsky A.A. Characterization of FPGA-based Digital Libraries // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 1. P. 15-19. doi:10.31114/2078-7707-2020-1-15-19.

[4] Alekseev V. B., Pospelov A.D. Diskretnaja matematika — M.: Izd. otd. fak. Vychislit. matematiki i kibernetiki MGU im. M. V. Lomonosova, 2002. — 44 s.

[5] Igoshin V. I. Matematicheskaja logika i teorija algoritmov. — 2-e izd., stereotip.. — M.: Izdatel'skij centr «Akademija», 2008. — 448 s.

[6] Brian W. Kernighan, Dennis M. Ritchie.The C Programming Language, Second Edition. Prentice Hall, Inc., 1988. ISBN 0-13-110362-8.

[7] Koterov D.V., Simdjanov I.V.. PHP 7. — SPb.: «BHV-Peterburg», 2017. — S. 1088. — ISBN 978-5-9775-3725-4.