УДК 004.021

Генерация больших наборов логических функций для систем автоматизации проектирования цифровых интегральных схем

А.А. Лялинский

Институт проблем проектирования в микроэлектронике РАН,

zelyal@inbox.ru

***Аннотация* — Важной составляющей процесса проектирования схем на ПЛИС (Программируемые Логические Интегральные Схемы) является наличие библиотеки с большим набором логических ячеек. Несмотря на то, что обычно она создается в ручном режиме опытным разработчиком библиотек, представляет интерес и автоматизация этого процесса. В данной работе рассматриваются аспекты, связанные с автоматизацией создания и оптимизации больших наборов уникальных логических функций, которые служат основой для последующей разработки библиотек. Предложена методика создания набора логических функций для заданного числа переменных и логических операций в формульном задании функций. Предложен критерий оценки задержки сигнала в логической ячейке, создаваемой на основе получаемых представлений функций. Разработан алгоритм оптимизации состава функций библиотеки.**

***Ключевые слова* — генерация библиотек комбинационных ячеек, вычисление логических функций, тождественность логических функций, оценка быстродействия логической ячейки, оптимизация состава библиотеки логических функций, системы автоматизации проектирования.**

I. Введение

В ряде случаев, например при разработке библиотеки комбинационных ячеек, встает задача получения максимально большого числа различных логических функций от заданного числа переменных и операторов в виде скобочного представления в базисе операторов И, ИЛИ, НЕ, ИСКЛЮЧАЮЩЕЕ ИЛИ. Иными словами, каждая ячейка должна реализовать логическую функцию, отличающуюся от функций других ячеек. Это позволяет, с одной стороны, устранить ненужное дублирование ячеек в библиотеке, и с другой – иметь библиотечные ячейки для максимально большого количества логических функций (реальные библиотеки проектирования интегральных схем имеют в своем составе ячейки с одинаковыми логическими функциями, но при этом они отличаются по потребляемой мощности и задержке).

Проблема оптимальности состава обычно решается в «ручном режиме» - разработчик библиотеки ячеек на основе личного опыта определяет набор необходимых функций, при этом вопрос оптимального набора функций и их полноты остается открытым.

Краеугольный камень в решении задачи автоматизированного подбора функций – это проблема определения их тождественности. Известны несколько подходов.

1. Преобразование каждой функции к канонической форме, например к **ДНФ** (Дизъюнктивной Нормальной Форме) или к **КНФ** (Конъюнктивной Нормальной Форме) [1, 2, 5, 6, 13]. В данном подходе исследуется скобочное представление функции в базисе логических операторов. Так называемая нормализация логической функции (приведение ее к ДНФ или КНФ) позволяет свести задачу проверки равенства двух ДНФ D_1, D_2 к выполнению условий $D_1 \& \overline{D_2} = 0$, $\overline{D_1} \& D_2 = 0$. Если эти равенства выполняются, то $D_1 = D_2$. Приведение к ДНФ формульного представления функции требует обеспечения применения соответствующих программ.
2. Использование **диаграмм двоичных решений** (BDD, Binary Decision Diagram) [14-16], в частности ROBDD (Reduced Ordered Binary Decision Diagram), являющейся каноническим представлением для конкретной функции и заданного порядка переменных.
3. **SAT-решатели** [17].
4. Методы **формальной верификации** [18].
5. Использование **таблиц истинности**.

Определим методы 1-4 как «аналитические». Для них характерно то, что ручное преобразование несколько десятков или сотен функций на основе данного подхода требует много времени и чревато ошибками. Автоматическое же требует использования специализированного программного обеспечения (ПО), что влечет определенные временные и финансовые затраты. Универсальность этих методов обуславливает их сравнительную неэффективность для функций небольших размеров, которые являются предметом рассмотрения данной статьи, так как библиотеки логических ячеек в подавляющем

большинстве реализованы на функциях малого размера.

В табличном подходе (метод 5) анализируются только результаты моделирования на всем множестве двоичных векторов, что достаточно легко можно автоматизировать, так как генерируемые логические функции зависят от небольшого числа переменных. Исходя из этого автором был сделан выбор в пользу применения таблиц истинности, позволяющих при небольшом объеме программного кода выполнять быстрое сравнение функций малой размерности.

Статья построена следующим образом. В главах II-III описана методика сравнения функций, базирующаяся на «табличном» сравнении. В главе IV дано описание разработанного алгоритма генерации логических функций. Совместное применение методик генерации и сравнения функций позволило разработать алгоритмы наполнения библиотеки, результаты этого процесса даны в главе V. В главах VI-VII описана методика выделения из полученной на предыдущих шагах библиотеки оптимального (по критерию минимального количества) набора функций. В главе VIII дано описание веб-сайта http://www.ippm.ru/funclib, который представляет собой программную реализацию предложенных алгоритмов.

II. «Табличное» сравнение логических функций

Прежде всего для характеризации логической функции необходимо получить вектор значений функции на всех возможных наборах значений ее переменных (далее обозначаемый как INP в параграфе IV). При представлении функций таблицами истинности данный вектор является столбцом значений функции. Для удобства этот вектор будем представлять строкой. Примеры векторов INP для функции *f(a,b)* приведены в табл. 1, для функции *f(a,b,c)* - в табл. 2. Длина m вектора INP для функции, зависящей от n переменных, равна 2^n. «s_i» в этих таблицах – просто присвоенное данному вектору обозначение.

Таблица 1

Набор входных векторов INP для n=2

n=2					INP
a:	0	0	1	1	s_1
b:	0	1	0	1	s_2

Таблица 2

Набор входных векторов для n=3

n=3									INP
a:	0	0	0	0	1	1	1	1	s_1
b:	0	0	1	1	0	0	1	1	s_2
c:	0	1	0	1	0	1	0	1	s_3

Так как мы проверяем чисто комбинационные (не имеющие памяти) ячейки, то порядок состояний может быть любым. Для определённости зафиксируем показанный выше порядок состояний как стандартный.

Далее отметим, что функции можно разделить на два множества – симметричные (например, a&b) и несимметричные (a&!b). Для симметричной функции значение (0,1) функции зависит только от числа единиц в наборе значений переменных, значение симметричной функции не изменяется при перестановке (переименовании) ее переменных:

a:	0 0 1 1		a:	0 1 0 1
b:	0 1 0 1	или	b:	0 0 1 1
a&b:	0 0 0 1		a&b:	0 0 0 1

Выходной вектор в обоих случае один и тот же: 0 0 0 1.

Для несимметричной функции это не так:

a:	0 0 1 1	a:	0 1 0 1
b:	0 1 0 1	b:	0 0 1 1
a&!b:	0 0 1 0	a&!b:	0 1 0 0

Выходные векторы различаются: 0 0 1 0 и 0 1 0 0.

III. Сравнение несимметричных функций

Для каждой функции предлагается построить ее «портрет» и использовать его для дальнейшего анализа библиотеки функций.

«Портрет» функции строится следующим образом. Пусть дана логическая функция f, зависящая от n переменных:

$$f = f(x_1, x_2, \ldots, x_n).$$

Обозначим INP = $\{s_1, s_2, \ldots, s_m\}$ – набор входных тестов, позволяющий проверить значения функции **f** на всех возможных наборах значений аргументов (см. самый правый столбец табл. 1 и 2). Каждое s_i из INP имеет уникальный набор 0 и 1 по отношению к другим компонентам INP. Создадим размещение P из различных INP_j, при этом каждое INP_j создается из различных сочетаний s_i. Например:

$INP_1 = \{s_1, s_2, \ldots, s_m\}$,
$INP_2 = \{s_2, s_1, \ldots, s_m\}$ и т.д.

Проще говоря, все INP_i состоят из одинакового набора s_j, но отличаются порядком их следования в каждом из INP.

«Портрет» функции – это набор пар B_i: $\{INP_i, out_i\}$, где out_i - выходной вектор, соответствующий результатам исполнения входного теста INP_i для некоторой функции **f**. Совокупность B_i составляет «портрет» функции. Например, «портрет» функции (!a|!b)&c:

function: (!a|!b)&c

Совокупность входных векторов					Выходной вектор out_i	Портрет функции
INP_1:	s_1	s_2	s_3	→	11110100	B_1
INP_2:	s_1	s_3	s_2	→	11110010	B_2
INP_3:	s_2	s_1	s_3	→	11011100	B_3
INP_4:	s_2	s_3	s_1	→	11001110	B_4
INP_5:	s_3	s_1	s_2	→	10111010	B_5
INP_6:	s_3	s_2	s_1	→	10101110	B_6

или то же самое в битовом представлении:

00001111 00110011 01010101 --> 11110100
00001111 01010101 00110011 --> 11110010
00110011 00001111 01010101 --> 11011100
00110011 01010101 00001111 --> 11001110
01010101 00001111 00110011 --> 10111010
01010101 00110011 00001111 --> 10101110

Анализ пар B_i выполняется в два этапа:

- на **первом** этапе отсеиваются пары, у которых выходной вектор out_i состоит из одних 0 или одних 1 – такие «константные» выходные векторы являются результатом автоматической генерации функции и не представляют практического интереса.
- на **втором** этапе векторы out_i попарно сравниваются между собой. Совпадение векторов говорит о том, что какие-то INP_i и INP_j имеют одинаковый выходной вектор и общее количество выходных векторов, хранимых для данной функции, можно уменьшить.

После построения набора выходных векторов задача выделения уникальных функций становится достаточно простой: разбиваем все множество созданных функций по количеству имеющихся в них независимых переменных и внутри каждого полученного подмножества сравниваем между собой наборы выходных векторов. Их совпадение говорит о том, что из этих двух функций можно оставить одну.

IV. Генерация логических функций

Для создания собственно образа логической функции (т.е. уникальной последовательности логических операторов и переменных) был применен следующий алгоритм.

Алгоритм создания образа логической функции

1. Пусть “n” – число независимых переменных, входящих в состав создаваемой логической функции.
2. “op” – число логических операторов (не обязательно уникальных), участвующих в представлении логической функции. Ограничение: op ≥ n-1.

Замечание: при интерактивном запуске данного алгоритма “n” и “op” могут задаваться пользователем. При пакетном выполнении, например, при генерации библиотеки, эти параметры являются переменными цикла при переборе в заданном диапазоне значений.

3. Зададим множество логических переменных: VAR: {a, b, c, d, …}.
4. Зададим множество логических операторов: OP: {“**&**”, “|”, “^”} (последнее - исключающее ИЛИ).
5. fImage = ‘’ *(очистим рабочий образ функции).*
6. for (i = 0; i < op+1; i++)
   - 6.1. Выберем случайным образом логическую операцию “logical_op” из множества OP.
   - 6.2. Если i > 0, то fImage += “ logical _op” *(т.е., если образ функции не пуст, то добавляем справа очередной оператор).*
   - 6.3. Выберем случайным образом переменную ‘v’ из множества VAR.
   - 6.4. Выберем случайным образом – стоит ли добавлять инверсию перед ‘v’*(иными словами, случайным образом решаем, какую переменную добавлять: v или !v).*
   - 6.5. Добавить *v или !v в* fImage справа.
7. Конец алгоритма.

В связи со случайным характером, заложенным в данный алгоритм, выход из него не тривиален. Могут быть использованы следующие критерии:

- выход после генерации заданного количества функций;
- выход после определенного количества пустых итераций, когда не удается получить нового образа функции.

V. Генерация библиотеки логических функций

Описанный выше алгоритм был реализован на языке PHP [12]. Потребовалось написание двух модулей – основного (размером 280 строк) для реализации собственно алгоритма и вспомогательного (размером 370 строк) для вычисления логических выражений. Скорость работы алгоритма зависит от заданного «n» и составляет от секунд до десятков секунд.

В табл. 3 показаны результаты по количеству генерируемых функций в зависимости от числа переменных и операторов. Пустые клетки в таблице обозначают либо недопустимое сочетание параметров, либо не представляющие интереса вычисления. Число уникальных функций для количества переменных больше или равного 4 не является исчерпывающим – работа алгоритма в этом случае ограничивается предельным временем исполнения скрипта на веб-сервере. Для получения полного значения необходимо выполнять скрипт генерации многократно или реализовать данный алгоритм на локальной машине на языке C++. Отметим, что библиотеки с таким

количеством уникальных функций вряд ли представляют практический интерес.

Таблица 3

Генерация логических функций

число операций	1	2	3	4	5	6
количество независимых переменных: 2						
# уникальных функций	4	6	8	8		
всего создано функций	6	12	18	18		
количество независимых переменных: 3						
# уникальных функций		20	64	74	78	
всего создано функций		64	800	4736	10671	
количество независимых переменных: 4						
# уникальных функций			146	456	636	802
всего создано функций			862	3272	4269	3646

Из таблицы видно, что превышение числа операций над числом переменных более чем на единицу на малом числе переменных мало что дает, так как количество уникальных функций при этом почти не увеличивается. Рост общего количества функций (не уникальных) при увеличении числа операций связан с появлением в представлении функции константных термов, то есть часть представления функции после операций приведения всегда равна 0 или 1 и не представляет интереса.

VI. Эффективность получаемых функций

Введем понятие *глубины логического выражения*, под которым будем понимать максимальное количество операций, которым будет подвергнут сигнал в некоторой абстрактной логической ячейке, реализующей соответствующую логическую функцию. Примеры даны на рис. 1.

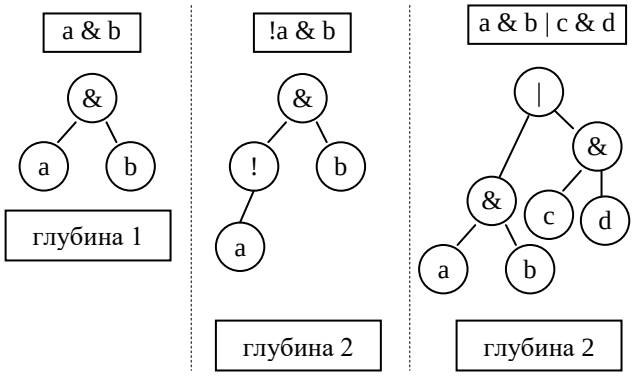

Рис. 1. Глубина логического выражения

С определенной долей точности глубина логического выражения может быть соотнесена с задержкой прохождения сигнала через логическую ячейку и в этом смысле помогает выбрать функцию с нужной задержкой среди *алиасов* (функций, имеющих различное представление, но выполняющих одну и ту же логическую функцию).

VII. Разложение произвольной логической функции

На основе алгоритма генерации набора логических функций довольно легко может быть решена «обратная» задача – как разложить произвольно заданную функцию на более простые логические компоненты? Предполагается, что исходная функция имеет довольно большую размерность, а компоненты, на которые ее надо разложить – небольшие библиотечные функции. Предлагается следующее решение.

Постановка задачи:

Пусть:

- $f = f_1(x_1, x_2, \ldots, x_n)$ – исходная функция, которую надо реализовать функциями меньшей размерности. **n** – количество независимых переменных данной функции.
- $g_i = g(x_1, x_2, \ldots, x_{mi})$ - библиотечные функции, на которые надо разложить **f**. Размерность m_i каждой из функций g_i существенно меньше n, обычно $1 \le m_i \le 4$.
- m – максимальное число переменных библиотечной функции.

Требуется:

найти такую функцию f_2 и набор библиотечных функций g_i, что $f = f_2(g_1(x), g_2(x), \ldots, g_k(x))$ давало бы результаты, идентичные функции f_1 на полном наборе значений входных переменных.

Решение:

Шаг 1. Построить направленный граф для исходной функции (см. рис.1). Переменные составляют терминальные (нижние) узлы графа, остальные (нетерминальные) узлы – это результаты операций. Ребра графа создаются от операнда (переменная или результат операции) к результату операции. Таким образом, все ребра направлены снизу вверх.

Шаг 2. Выбрать терминальный узел из ветви графа, имеющей наибольшее расстояние до вершины графа. Продвинуться на одно ребро вверх к «отцу» этого узла и образовать промежуточную функцию **g** из данного узла, его «отца» и другого «сына» этого «отца» (если это бинарная операция, а не отрицание).

Шаг 3. Продвигаться от узла «отца» вверх и присоединять новые узлы к **g** до тех пор, пока количество независимых переменных в **g** не достигнет **m**. Сохранить полученную функцию **g** как новую библиотечную.

Заменить в исходном графе все узлы и ребра, вошедшие в **g**, одним новым нетерминальным узлом.

Шаг 4. Повторять шаги 2 и 3 до тех пор, пока в графе не останется одно ребро.

Шаг 5. Построить библиотеку функций из набора функций $\mathbf{g_i}$, удалив функции с повторяющимися значениями.

VIII. Веб-интерфейс

Для удобства пользования данной системой был разработан веб-интерфейс [7, 8, 9, 10]. Программное обеспечение размещено на веб-сервере. Доступ к программе возможен с веб-страницы по адресу http://www.ippm.ru/funclib/. На рис. 2 показан интерфейс страницы генерации библиотеки функций.

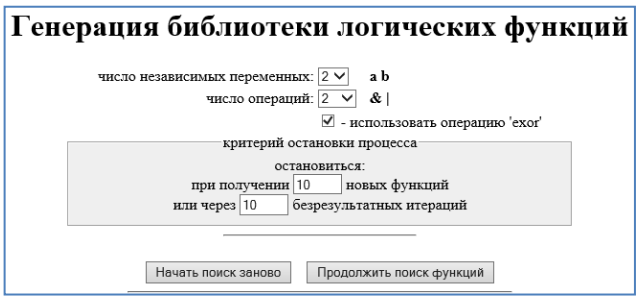

Рис. 2. Задание исходных данных для генерации функций

Пользователь имеет возможность указать число независимых переменных логической функции, число операций в функции, возможность использования операции «Исключающее ИЛИ».

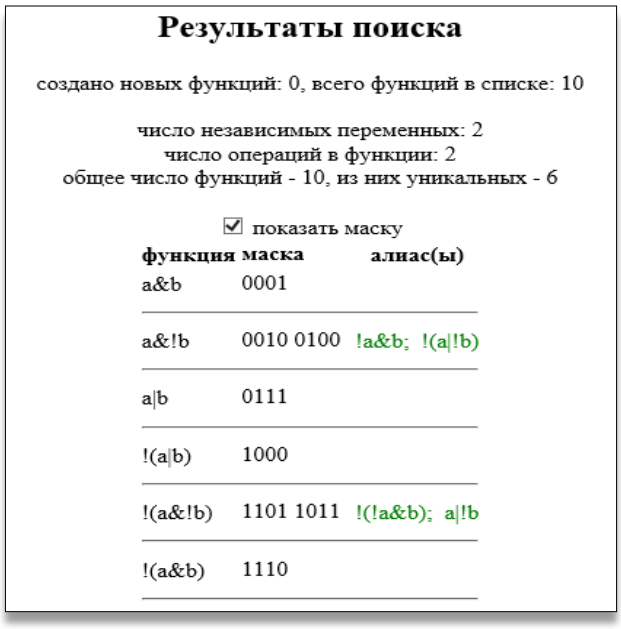

Рис. 3. Результаты генерации списка функций для nvar=2 и nop=2

Учитывая, что на сервере можно выполнять задачу не более заранее оговоренного времени, то внутри программы ведется контроль длительности ее исполнения. При достижении лимита времени происходит останов и сохранение полученных результатов. После этого возможен повторный расчет с предыдущей точки.

Результаты выполнения задачи выдаются в виде таблицы (см. рис. 3). Исходные данные для примера в этом рисунке – число уникальных переменных: 2 и число операций – также 2 (отрицание в данном случае не входит в число операций). Уникальных функций в данном случае 6, остальные 4 функции повторяют выходную маску первых уникальных функций.

Вторая страница разработанного сайта позволяет оптимизировать состав только что полученной или отдельно заданной библиотеки функций. Во входном интерфейсе (рис. 4) указывается источник списка функций и максимальная глубина базовых логических функций.

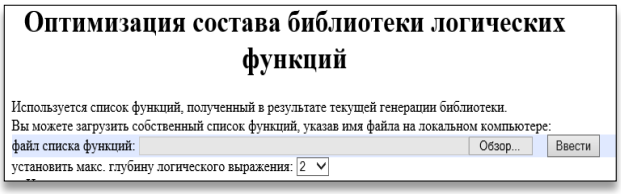

Рис. 4. Задание исходных данных для оптимизации состава библиотеки

Результаты выдаются в табличном виде (рис. 5).

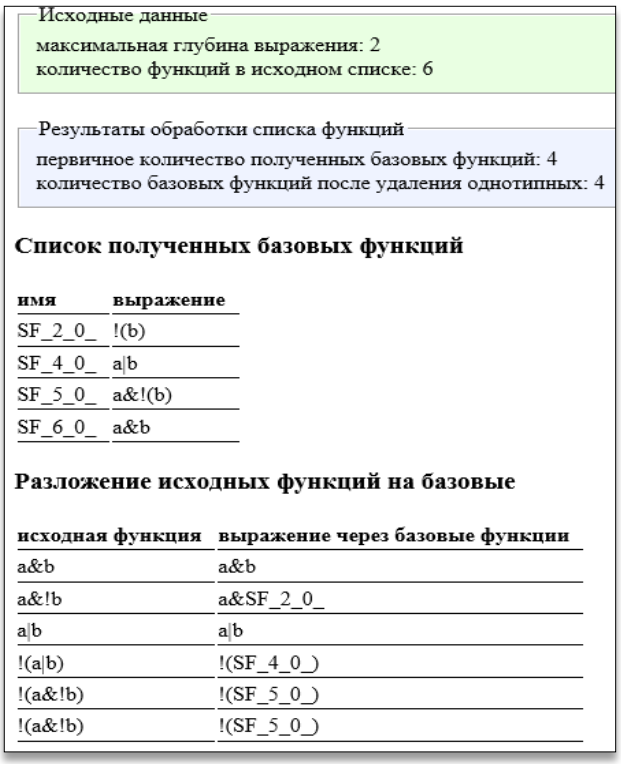

Рис. 5. Результаты оптимизации состава библиотеки

В дополнение к данным табл. 1 в табл. 4 приведены результаты разложения этих списков по базовым функциям и последующей оптимизации.

Таблица 4

Оптимизация состава библиотек логических функций

#перем.	#опер.	#сгенери-рованных функций до разложен.	после разложения	
			#базовых функций (до оптимизации)	#базовых функций (после оптимиз.)
2	1	4	3	3
2	>= 2	6	4	4
3	3	20	31	12
3	3	64	120	26
3	4	74	166	30
3	5	78	204	32
4	3	146	271	40
4	4	456	1064	63
4	5	636	1651	89
4	6	802	2396	105

Сопоставление количества базовых функций до и после оптимизации показывает ее важность, так как размер списка сокращается от 3 до 20 раз.

IX. Выводы

Предложен эффективный алгоритм сравнения произвольных логических функций, не требующий анализа скобочного представления функции в базисе операторов. На основе данного алгоритма разработан и апробирован алгоритм построения библиотеки логических функций любой размерности.

Предложен алгоритм разложения произвольной логической функции на базовые подфункции заданной глубины. На основе данного алгоритма реализована методика оптимизации состава библиотеки логических функций.

Благодарность

Автор выражает признательность профессору Объединенного института проблем информатики НАН Белоруссии д.т.н. Петру Николаевичу Бибило за советы и помощь в подготовке данной статьи.

Литература

[1] Игошин В.И. Математическая логика и теория алгоритмов. 2-е изд., стереотип. М.: Издательский центр «Академия», 2008. 448 с. ISBN 978-5-7695-4593-1.

[2] Crama Y., Hammer P. L. Boolean Functions. Cambridge University Press. 2011.

[3] Hazewinkel Michiel, ed. Boolean function. Encyclopedia of Mathematics. Springer. 2001. ISBN 978-1-55608-010-4.

[4] Марченков С.С. Замкнутые классы булевых функций. М.: Физматлит, 2000.

[5] Поздняков С.Н., Рыбин С.В. Дискретная математика. 2008. С. 303.

[6] Ю.И. Галушкина, А.Н. Марьямов: Конспект лекций по дискретной математике - 2-е изд., испр. М.: Айрис-пресс, 2008. 176 с. (Высшее образование)

[7] URL: http://www.ippm.ru/index.php?page=webcad (дата обращения: 04.03.2016).

[8] Лялинский А.А. Автоматизированное формирование тестов при характеризации цифровых ячеек с использованием веб-доступа // Проблемы разработки перспективных микро- и наноэлектронных систем - 2012. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН, 2012. С. 95-100.

[9] Лялинский А.А. Формирование высокоуровневых моделей цифровых ячеек с использованием веб-доступа // Проблемы разработки перспективных микро- и наноэлектронных систем - 2012. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН, 2012. С. 101-106.

[10] Соловьев А.Н., Саблин А.В., Лялинский А.А. Разработка среды моделирования инерциальных навигационных систем // Проблемы разработки перспективных микро- и наноэлектронных систем - 2014. Сборник трудов / под общ. ред. академика РАН А.Л. Стемпковского. М.: ИППМ РАН, 2014. Часть I. С. 31-36.

[11] Лялинский А.А. Особенности построения прикладных программ с веб-доступом. Информатика. Январь-март 2013. Минск. №1(37). С. 76-83.

[12] URL: http://php.net/manual/ru/index.php (дата обращения: 21.04.2016).

[13] Закревский А.Д., Поттосин Ю.В., Черемисинова Л.Д. Логические основы проектирования дискретных устройств. М.: Физматлит, 2007. 589 с.

[14] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, C-35(8):677-691, 1986.

[15] Binary Decision Diagrams: Theory and Implementation. — Springer, 1998. — ISBN 978-1-4419-5047-5.

[16] C. Y. Lee. "Representation of Switching Circuits by Binary-Decision Programs". Bell Systems Technical Journal, 38:985–999, 1959.

[17] Vizel, Y.; Weissenbacher, G.; Malik, S. (2015). "Boolean Satisfiability Solvers and Their Applications in Model Checking". Proceedings of the IEEE 103 (11). doi:10.1109/JPROC.2015.2455034.

[18] Sanghavi, Alok (21 May 2010). "What is formal verification?". EE Times_Asia.

Generation of large sets of logical functions for digital integrated circuits CAD systems

A.A. Lyalinsky

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

***Keywords* — generation of libraries of combinational cells, calculation of logical functions, identity of logical functions, evaluation of performance of logical cell, optimization of library logical functions, automation system design.**

ABSTRACT

In some cases, for example, when developing library of combinational cells, there is the problem of obtaining the largest possible number of logical functions, each of which would be unique. This allows, on one hand, to eliminate unnecessary duplication of cells in the library, and on the other hand, to have library cells for the largest possible number of logical functions.

The optimality problem is usually solved in "manual mode" - the developer of the library cells defines the set of functions based on personal experience, the question of the optimal set of functions and their completeness is still open.

The difficulty for automated selection of functions is the problem of determining the identity of logical functions. Known methods are based on the transformation of each function and bringing it to a canonical form, such as DNF - Disjunctive Normal Form. In this case, manual converting of a few tens or hundreds of functions is time-consuming and error-prone manual processing. Besides, the size of logical expression is increased. In this paper, a method for generating libraries of logic functions based on the automated comparison of truth tables of logic functions is presented. In addition, another method is proposed for optimization of the composition of the library for the minimum criterion of available functions.

ACKNOWLEDGEMENTS

The author is grateful to Professor of United Institute of Informatics problems of NAS of Belarus, doctor of technical Sciences Peter N. Bibilo for advice and assistance in the preparation of this article.

REFERENCES

[1] Igoshin V. I. Matematicheskaja logika i teorija algoritmov. 2-e izd., stereotip. Moscow, Izdatel'skij centr «Akademija», 2008. 448 p. ISBN 978-5-7695-4593-1 (in Russian).

[2] Crama Y., Hammer P. L. Boolean Functions. Cambridge University Press. 2011.

[3] Hazewinkel Michiel, ed. Boolean function. Encyclopedia of Mathematics. Springer. 2001. ISBN 978-1-55608-010-4.

[4] Marchenkov S.S. Zamknutye klassy bulevyh funkcij. Moscow, Fizmatlit, 2000 (in Russian).

[5] Pozdnjakov S.N., Rybin S.V. Diskretnaja matematika. 2008. P. 303 (in Russian).

[6] Ju.I. Galushkina, A.N. Mar'jamov: Konspekt lekcij po diskretnoj matematike. 2-e izd., ispr. Moscow, Ajris-press, 2008. 176 p. (Vysshee obrazovanie) (in Russian).

[7] URL: http://www.ippm.ru/index.php?page=webcad (access date: 04.03.2016).

[8] Lyalinsky A.A. Avtomatizirovannoe formirovanie testov pri harakterizacii cifrovyh jacheek s ispol'zovaniem veb-dostupa // Problemy razrabotki perspektivnyh mikro- i nanojelektronnyh sistem. 2012. Sbornik trudov / pod obshh. red. akademika RAN A.L. Stempkovskogo. Moscow, IPPM RAN, 2012. pp. 95-100 (in Russian).

[9] Lyalinsky A.A. Formirovanie vysokourovnevyh modelej cifrovyh jacheek s ispol'zovaniem veb-dostupa // Problemy razrabotki perspektivnyh mikro- i nanojelektronnyh sistem - 2012. Sbornik trudov / pod obshh. red. akademika RAN A.L. Stempkovskogo. Moscow, IPPM RAN, 2012. pp. 101-106 (in Russian).

[10] Solov'ev A.N., Sablin A.V., Lyalinsky A.A. Razrabotka sredy modelirovanija inercial'nyh navigacionnyh sistem // Problemy razrabotki perspektivnyh mikro- i nanojelektronnyh sistem. 2014. Sbornik trudov / pod obshh. red. akademika RAN A.L. Stempkovskogo. Moscow, IPPM RAN, 2014. Chast' I. pp. 31-36 (in Russian).

[11] Lyalinsky A.A. Osobennosti postroenija prikladnyh programm s veb-dostupom. Informatika. No. 1(37). Janvar'-mart 2013. Minsk. pp.76-83 (in Russian).

[12] URL: http://php.net/manual/ru/index.php (access date: 21.04.2016).

[13] Zakrevskij A.D., Pottosin Ju.V., Cheremisinova L.D. Logicheskie osnovy proektirovanija diskretnyh ustrojstv. – Moscow, Fizmatlit, 2007. 589 p. (in Russian).

[14] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, C-35(8):677-691, 1986.

[15] Binary Decision Diagrams: Theory and Implementation. — Springer, 1998. ISBN 978-1-4419-5047-5.

[16] C. Y. Lee. "Representation of Switching Circuits by Binary-Decision Programs". Bell Systems Technical Journal, 38:985–999, 1959.

[17] Vizel, Y.; Weissenbacher, G.; Malik, S. (2015). "Boolean Satisfiability Solvers and Their Applications in Model Checking". Proceedings of the IEEE 103 (11). doi:10.1109/JPROC.2015.2455034.

[18] Sanghavi, Alok (21 May 2010). "What is formal verification?". EE Times_Asia.