ISSN 0002-306X. Изв. НАН РА и ГИУА. Сер. ТН. 2007. Т. LX, № 3.

УДК 519.95

АВТОМАТИЗАЦИЯ И СИСТЕМЫ УПРАВЛЕНИЯ

В.С. ЕГИАЗАРЯН, С.П. ПОГОСЯН

МИНИМАЛЬНОЕ ПРЕДСТАВЛЕНИЕ БУЛЕВЫХ ФУНКЦИЙ ОТ ЧЕТЫРЕХ ПЕРЕМЕННЫХ И ИХ ПРИМЕНЕНИЕ В АВТОМАТИЗАЦИИ ПРОЕКТИРОВАНИЯ ЦИФРОВЫХ СХЕМ

Приведены определения классов P и NPN эквивалентностей и возможности их применения с целью создания базисных элементов для комбинационных схем, имеющих два и более входа. Описан алгоритм для получения минимальных представлений всех булевых функций от четырех переменных для предварительно заданного базиса функций от двух переменных.

Ключевые слова: оптимальные сети, двоичные разрешающие диаграммы (BDD), синтез, булевы функции, минимальное представление.

Введение. Проектирование цифровых схем тесно связано с теорией булевых функций (далее - функции). Одним из таких применений является их замена на эквивалентные функции, имеющие более короткую запись, что приводит к реализации функции с помощью меньшего количества логических элементов. Такая операция обычно применяется в процессе синтеза цифровых схем.

Целью настоящей работы является классификация функций по классам эквивалентностей, что уменьшает область наблюдения, нахождение минимального представления для всех функций от четырех переменных и применение этих классов для комбинаторной оптимизации цифровых схем. При этом подразумевается минимизация количества базисных элементов в представлении для предварительно выбранного базиса функций от двух переменных.

В табл. 1 показаны все 16 функций от двух переменных.

Таблица 1

AB	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
00	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
01	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
10	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
11	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Количество функций возрастает экспоненциально от числа переменных, делая невозможным оптимальное управление инструментами оптимизации (для $N = 4$ уже имеем 65536 функций).

В проектировании цифровых схем обычно используют только реализации 5-6 функций из всех 16-и от двух переменных, что облегчает работу инструментов в процессе проектирования и минимизирует возможные конструкторские недоработки.

Этот же подход можно применить к функциям, зависящим более чем от двух переменных. При этом нужно найти такой базис функции (точнее, реализацию этих функций) от N переменных, чтобы все остальные функции могли быть представлены с помощью них (т.е. схема будет собрана только из элементов, реализующих эти функции).

Объединим все функции, которые реализуются с помощью одной и той же схемы, в одном классе эквивалентности. В частности, можно использовать понятия P (Permutation) и NPN (Negation/Permutation, output Negation) эквивалентности [1,2].

Реализации функций от двух переменных. В табл. 2 показана реализация всех 16 функций от двух переменных. Следует отметить, что некоторые функции здесь реализованы с использованием одного и того же логического элемента.

Таблица 2

Реализации функций		Функция	Название функции
$F_0 = 0$		0	Константа 0 (FALSE)
$F_1 = \overline{A} \wedge \overline{B}$		$A \downarrow B$	Стрелка Пирса ИЛИ-НЕ (NOR)
$F_2 = \overline{A} \wedge B$		$A \Leftarrow B$	Подавление A (INHIBIT)
$F_3 = \overline{A}$		$\overline{A}$	Инверсия A, НЕ A (NOT)
$F_4 = A \wedge \overline{B}$		$A \Rightarrow B$	Подавление B (INHIBIT)
$F_5 = \overline{B}$		$\overline{B}$	Инверсия B, НЕ B (NOT)
$F_6 = A \oplus B$		$A \oplus B$	Сложение по модулю 2 (XOR)
$F_7 = \overline{A \wedge B}$		A / B	Штрих Шеффера, И-НЕ (NAND)
$F_8 = A \wedge B$		$A \wedge B$	Конъюнкция – логическое умножение, И (AND)
$F_9 = \overline{A \oplus B}$		$A \sim B$	Эквивалентность (EQUIVALENCE)
$F_{10} = B$		B	Повторение B (IDENTITY)
$F_{11} = \overline{A \wedge \overline{B}}$		$A \rightarrow B$	Импликация из A в B (IMPLICATION)
$F_{12} = A$		A	Повторение A (IDENTITY)
$F_{13} = \overline{\overline{A} \wedge B}$		$A \leftarrow B$	Импликация из B в A (IMPLICATION)
$F_{14} = \overline{\overline{A} \wedge \overline{B}}$		$A \vee B$	Дизъюнкция – логическое сложение, ИЛИ (OR)
$F_{15} = 1$		1	Константа 1 (TRUE)

Понятие класса P. *Определение*. Функции f_1 и f_2 от переменных $A_1,...,A_N$ называются P *эквивалентными*, если существует перестановка $\omega:\{1,...,N\}\to\{1,...,N\}$ такая, что

$$f_1(A_1,...,A_N)=f_2(A_{\omega(1)},...,A_{\omega(N)})$$

при всех возможных значениях $A_i, i=1,...,N$, т.е. две функции P эквивалентны, когда возможно получить идентичные таблицы истинности, переставляя переменные одной из функций.

Например, функции F_2 и F_4 в табл. 2 P эквивалентны. Очевидно, что P эквивалентность разбивает множество булевых функций на классы эквивалентности.

В табл. 3 приведены все 12 различных классов P эквивалентных функций от двух переменных. Также приведены схемы, реализующие функции из соответствующего класса. Важным свойством P эквивалентных функций является то, что они могут всегда реализоваться, используя один и тот же логический элемент.

Таблица 3

F_0	F_1	F_2, F_4	F_3, F_5	F_6	F_7	F_8	F_9	F_{10}, F_{12}	F_{11}, F_{13}	F_{14}	F_{15}
								вход			

Понятие класса NPN. В табл. 3 показано, что некоторые классы P эквивалентности также могут иметь схожие реализации. Например, функции $F_1, F_2, F_4, F_7, F_8, F_{11}, F_{13}$ и F_{14} имеют реализации на уровне логических элементов, основанные на единственном логическом элементе И-НЕ (NAND) плюс несколько инверторов.

Определение. Функции f_1 и f_2 от переменных $A_1,...,A_N$ называются *NPN эквивалентными*, если существуют перестановка $\omega:\{1,...,N\}\to\{1,...,N\}$ и набор $\{\sigma,\sigma_1,...,\sigma_N\}, \sigma,\sigma_i\in\{0,1\}$ такие, что

$$f_1(A_1,...,A_N)=f_2{}^{\sigma}(A^{\sigma_1}{}_{\omega(1)},...,A^{\sigma_N}{}_{\omega(N)})$$

при всех возможных значениях $A_i, i=1,...,N$, где

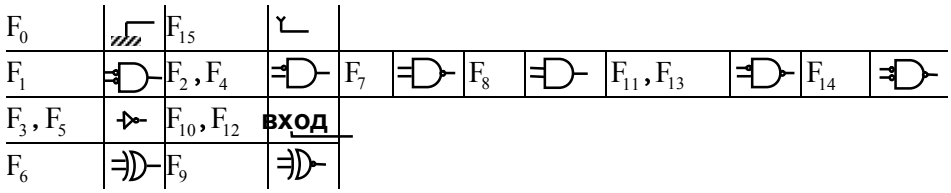
$$x^{\sigma}\equiv\begin{cases}x, \sigma=1,\\ \overline{x}, \sigma=0,\end{cases}$$

т.е. две функции NPN эквивалентны, если можно получить идентичные таблицы истинности для этих функций, переставляя и/или инвертируя переменные одной из функций и/или инвертируя значение одной из функций.

Эти функции могут быть сгруппированы в класс эквивалентностей NPN. В табл. 4 показаны все 4 различных NPN класса для функций от двух

переменных, в каждой строке - по одному классу. Важно заметить, что, несмотря на существование 16 различных функций от двух переменных, есть только 4 различных класса NPN от двух переменных: 2 NPN класса, содержащих по две функции, один NPN класс, содержащий четыре функции, и один NPN класс, содержащий восемь функций. NPN эквивалентные функции могут быть реализованы единственной схемой плюс несколько инверторов (используемых для инвертирования входов и выходов - в случае необходимости).

Таблица 4



В табл. 5 приведены количественные характеристики соответственно P и NPN эквивалентностей для булевых функций, зависящих от 1, 2, 3 или 4 переменных [1,2].

Таблица 5

Кол. переменных	Кол. функций	Кол. P классов	Кол. NPN классов
1	4	4	2
2	16	12	4
3	256	80	14
4	65536	3984	222

Из 222 различных NPN классов 208 содержат функции точно от четырех переменных, 10 – точно от трех переменных, 2 – от двух переменных, 1 – от одной переменной и 1 – от 0 переменных (константы).

Функции от трех или менее переменных рассматриваются как вырожденные функции от четырех переменных.

Для случая четырех переменных каждый NPN класс содержит не более чем 768 элементов:

16 (количество всех возможных инверсий четырех переменных)

x 24 (4! - количество всех возможных перестановок четырех переменных)

x 2 (инвертирована функция или нет)

= 768

Так как различные комбинации инверсии и перестановок часто могут порождать одну и ту же самую функцию, то количество элементов NPN класса меньше 768.

Алгоритм получения минимальных представлений. Известно несколько схожих алгоритмов создания классов NPN эквивалентностей. Однако все они используют представление функции с помощью BDD (Binary

Decision Diagrams) [3-6]. Предлагаемый алгоритм имеет простую реализацию и не требует дополнительных усилий создания BDD.

Так как на практике в базис включаются функции от двух переменных (AND, NAND, OR, NOR, XOR), для нахождения P и NPN классов эквивалентностей для функции от четырех переменных важно найти минимальные представления функций от четырех переменных в этом базисе. Нахождение минимального представления функций широко используется в моделировании цифровых схем с помощью оттранслированной программы, где скорость моделирования прямо пропорциональна числу логических элементов схемы.

Так как количество всех таких функций равно $2^{2^4} = 65536$, то можем кодировать функции с помощью 16-битных чисел (например, первый бит будет показывать значение функции на совокупности (0, 0, 0, 0), второй бит – на совокупности (0, 0, 0, 1) и т.д. и, наконец, шестнадцатый бит – на совокупности (1, 1, 1, 1)). Например, функция И-НЕ(A, B) имеет код 30583.

Алгоритм. Алгоритм основан на рекурсивном подходе. Так как функция всегда должна иметь вид $F(f_1, f_2)$, где F - функция от двух переменных, а f_1, f_2 - функции от четырех переменных (могут быть и вырожденными), то будем искать функции подобного вида. Алгоритм для нахождения представлений длины L + 1 будет подбирать все уже найденные пары f_1, f_2 функции, длины которых в сумме равны L, и, перебирая поочередно все базисные функции F, построит следующий возможный вариант - $F(f_1, f_2)$. Так как функции f_1 и f_2 являются независимыми аргументами для функции F, то, очевидно, с целью получения минимального представления для функции $F(f_1, f_2)$ необходимо использовать минимальные представления для функций f_1 и f_2.

Имея очередной возможный вариант для представления функции, вычисляется ее код, и если этот код не был получен прежде, то он является одним из минимальных представлений для функции $F(f_1, f_2)$.

Заметим, что минимальное представление функции не является единственным.

Ниже на метаязыке приводятся основные фрагменты реализации алгоритма.

Получение следующего возможного представления для функции

Добавить примитивные функции ‘a’, ‘b’, ‘c’, ‘d’ к функциям длины 1.

Для длины L от 2 до 50 // 50 - максимальная предполагаемая длина функций
Для функции F2 из множества базиса
Для всех длин J от 1 до [L/2]
    Взять все функции P длины J

```
        Взять все функции Q длины (L-J)
            Функция := F2 + P + Q;
            Если не имеется Код (Функция), то
                увеличить количество найденных функций;
                добавить функцию во множество функций с длинами (L+1);
                проверить, если нашли 65536 функций, то выйти
        Конец // Q
        Конец // P
Конец // J
Конец // F2
Конец // L
```

Получение кода функции (если дана функция)

```
Code := 0
Для бита K от 0 до 15
        Если вычисленное значение функции на K равно 1, то
                в K-й бит Code написать 1 (Code |= (1 << K)).
Конец // K
```

Вычисление функции (если даны K и функция)

Каждой функции, переменной и константам ‘0’, ‘1’ присваивается некий символ для хранения в стеке, где производится вычисление.

```
Берем пустой стек “символов”.
Для индекса I от 0 до размера функции
        “символ” := I-й символ функции
        Если “символ” это функция, то
                в стек добавить эту функцию
        Иначе // это “символ” переменное ‘a’, ‘b’, ‘c’ или ‘d’
                V := бит K для переменного “символ”
                Пока стек не пуст
                        T := верхний “символ” стека
                        Если это значение ‘0’ или ‘1’, то
                                удаляем верхний элемент стека
                                O:= верхний “символ” стека
                                удаляем верхний элемент стека
                                V:= Значение базисной функции символом O, на (T, V);
                        Иначе
                                Выход из цикла;
                Конец // пока стек не пуст
                Добавляется значение V в стек
        Конец // иначе
Конец // I
        Возвращается единственно оставшийся элемент (‘0’ или ‘1’) из стека.
```

Результаты. Результаты были получены для трех различных базисов функций от двух переменных. Первым рассматривался базис из всех 16 функций. Программа реализована на машине Intel P4 мощностью 1,8 *ГГц* с оперативной памятью 768 *Мб* под операционной системой RedHat Linux 7.2.

Результаты получены после работы программы примерно через 8 *мин.* 25 *с.* Последний код, который был найден, имел следующее представление длиной 15 (7 функций и 8 переменных):

$$32407 \rightarrow F_6(F_8(F_6(a,d),F_{14}(b,c)),F_8(F_{13}(a,d),F_7(b,c))).$$

В табл. 6 приведена частота появления функций в представлениях.

Таблица 6

F_0	F_1	F_2	F_4	F_6	F_7	F_8	F_9	F_{11}	F_{13}	F_{14}	F_{15}
1	69074	17835	13806	95229	55925	9274	23908	25012	18008	13204	1

Функции F_3, F_5 (инверсия) и F_{10}, F_{12} (повторение) вообще не встречаются в представлениях. Если эти функции ($F_3\ F_5\ F_{10}\ F_{12}$ + $F_0\ F_{15}$) не включены в базис, то результаты будут получены примерно через 4 *мин.* 50 *с.*

Далее рассматривался базис из шести функций - F_1(NOR), F_4(INHIBIT), F_6(XOR), F_7(NAND), F_8(AND) и F_{14}(OR) (табл. 7). Результаты получены после работы программы примерно за 2 *мин.* 38 *с.*

Таблица 7

F_1	F_4	F_6	F_7	F_8	F_{14}
32999	51733	121326	37459	49521	51586

И, наконец, рассматривался базис из пяти функций - F_1(NOR), F_6(XOR), F_7(NAND), F_8(AND) и F_{14}(OR) (табл. 8) (тот же самый базис, что на прежнем примере, но без функции F_4(INHIBIT)). Результаты были получены после работы программы примерно за 3 *мин.* 40 *с.* Самое длинное представление уже имеет длину 17 (8 функций и 9 переменных).

Таблица 8

F_1	F_6	F_7	F_8	F_{14}
37908	138987	55477	71692	59982

Применение. Полученные результаты можно применить для оптимизации цифровых схем, что соответственно приводит к ускорению

процесса моделирования. Для этого нужно выделить все подсхемы, имеющие четыре входа, заменить их на новую, имеющую минимальное представление.

В табл. 9 приведены результаты оптимизации на эталонных схемах.

Таблица 9

Название схемы	Кол. элементов до оптимизации	Кол. элементов после оптимизации	Процент оптимизации, %
c6288	6073	5128	15,56
c7552	6811	5053	25,81
s5378	4986	3861	22,56
s13207	12733	8567	32,72
b14	21583	20576	4,67
b15_1	27727	25432	8,28
b17_1	84254	78957	6,29
b20	42985	40817	5,04%

СПИСОК ЛИТЕРАТУРЫ

1. **Culliney J. N., Young M. H., Nakagawa T., Muroga S.** Results of the Synthesis of Optimal Networks of AND and OR Gates for Four-Variable Switching Functions // IEEE Transactions on Computers. – January, 1979.- Vol. C-27, ¹ 1.- P. 76-85.
2. **Baugh Charles R., Chandersekaran C. S., Swee Richard S., Muroga Saburo.** Optimal Networks of NOR-OR Gates for Functions of Three Variables // IEEE Transactions on Computers. – February, 1972.- Vol. C-21, ¹ 2.- P. 153-160.
3. **Bryant R.E.** Graph-based algorithms for Boolean function manipulation // IEEE Transactions on Computers. – August, 1986.- Vol. C-35, ¹ 8.- P. 677-691.
4. **Brace K.S., Ruddel R.L., Bryant R.E.** Efficient implementation of a BDD package // Proceedings of 27th DAC. 1990.- P. 272-277.
5. **Friedman Steven J., Supowit Kenneth J.** Finding the Optimal Variable Ordering for Binary Decision Diagrams // IEEE Transactions on Computers.- May, 1990.- Vol. 39, ¹ 5.- P. 710-713.
6. **Liaw Heh-Tyan, Lin Chen-Shang.** On the OBDD-Representation of General Boolean Functions // IEEE Transactions on Computers.- June 1992.- Vol. 41, ¹ 6.- P. 661-664.

Российско-Армянский (Славянский) государственный университет. Материал поступил в редакцию 07.04.2007.

Վ. Ս. ԵՂԻԱԶԱՐՅԱՆ, Ս. Պ. ՊՈՂՈՍՅԱՆ

ՉՈՐՍ ՓՈՓՈԽԱԿԱՆԻՑ ԲՈՒԼՅԱՆ ՖՈՒՆԿՑԻԱՆԵՐԻ ԿԱՐՃԱԳՈՒՅՆ ՆԿԱՐԱԳՐՈՒԹՅՈՒՆԸ ԵՎ ԴՐԱՆՑ ԿԻՐԱՌՈՒԹՅՈՒՆԸ ԹՎԱՅԻՆ ՍԽԵՄԱՆԵՐԻ ՆԱԽԱԳԾՄԱՆ ԱՎՏՈՄԱՏԱՑՄԱՆ ՄԵՋ

Նկարագրված են բուլյան ֆունկցիաների համարժեքության P և NPN դասերը և դրանց կիրառման հնարավորությունները համակցված սխեմաների երկու և ավելի մուտքերով բազիսային տարրերի ստեղծման համար: Նկարագրված է ալգորիթմ՝ չորս փոփոխականի բոլոր բուլյան ֆունկցիաների կարճագույն նկարագրությունները ստանալու համար՝ նախապես տրված երկու փոփոխականի բուլյան ֆունկցիաների բազիսում:

Առանցքային բառեր. լավարկված ցանցեր, երկուական վերլուծող դիագրամներ (BDD), սինթեզ, բուլյան ֆունկցիաներ, կարճագույն նկարագրություն:

V.S. YEGHIAZARYAN, S.P. POGHOSYAN

MINIMAL REPRESENTATION OF BOOLEAN FUNCTIONS OF FOUR VARIABLES AND THEIR APPLICATION IN DESIGN AUTOMATION OF DIGITAL CIRCUITS

P and NPN equivalence classes of Boolean functions and possibility of their application in creation of basis of elements with two or more inputs for combinational circuits are described. An algorithm for finding minimal representation of all four variable Boolean functions, from two variable ones is given.

Keywords: optimal networks, Binary Decision Diagrams (BDD), synthesis, Boolean functions, minimal representation.