Classifying n-Input Boolean Functions

Vinícius P. Correia vincor@inf.ufrgs.br

André I. Reis andreis@inf.ufrgs.br

Instituto de Informática – UFRGS - Caixa Postal 15064
CEP 91501-970 – Porto Alegre – RS – Brasil

Abstract

This paper discusses the classification of n-input Boolean functions. The concept of P and NPN equivalence classes is used to classify n-input functions. This way, the set of n-input functions is classified according to three criteria: the number of functions, the number of P classes and the number of NPN classes. The meaning of these criteria is discussed through some affirmations relating them to practical aspects of Boolean function implementation.

Resumo

Este artigo discute a classificação de funções Booleanas de n entradas. O conceito de classes de equivalência P e NPN é usado na classificação de funções de n entradas. Deste modo, o conjunto das funções de n entradas é classificado de acordo com três critérios: o número de funções, o número de classes P e o número de classes NPN. O significado destes critérios é discutido através de algumas afirmações relacionando-os com aspectos práticos da implementação de funções Booleanas.

1 Introduction

The design of digital circuits involves a deep understanding of the concept of Boolean Functions. There are many operations usually applied in the digital circuit synthesis process. Many of these operations depend on the search space under consideration. This is the case of the matching phase performed during technology mapping [10], where a function (or only part of it) to be implemented is matched against cells from a library. Sometimes this matching is limited to cells with a maximum number of inputs. The goal of this paper is to classify n-input functions in order to have a precise idea about the search space of the whole set of n-input functions. The emphasis of the paper is on how to use some equivalence classes to reduce the search space, and on the meaning of these equivalence classes.
For a given number of input variables, there is a well-defined number of functions. This number is given by *2^(2^n)*, where *n* is the number of input variables. Each n variable function has 2^n possible minterms, resulting in a truth table with *2^n* lines. This is shown in figure 1 for the case of 2-input Boolean functions. The *2^n* possible minterms, or lines of the truth table, give the number of bits in each column of the truth table. This way, the output columns of the truth tables characterize a given Boolean function as a binary number of *2^n* bits. As there are *2^(2^n)* numbers of *2^n* bits, there are *2^(2^n)* possible different functions of n inputs. This is also shown in figure 1 for the case of 2-input functions.

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

Figure 1: All the 16 different 2-input functions.

Table 1 contains the information on the number of n-input functions for n varying from 2 to 4. It is possible to see that for the case of 4-input functions the search space is almost intractable if many operations need to be repeated.

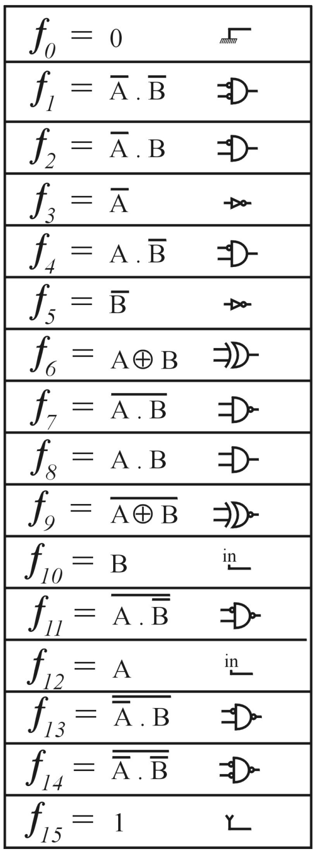
Table 1: Number of functions with 2, 3, 4 and n variables

Number of inputs	Number of functions
2	16
3	256
4	65536
n	2^(2^n)

The n-input functions can be classified into different classes (set of functions) in order to reduce the search space. The number of functions is the first number to be considered in the classification of the set of n-input functions. Other two numbers that could be used in the classification of n-input functions are the numbers of P and NPN equivalence classes. As it will be discussed later, P classes group functions that are equivalent under input permutation while NPN classes group functions that are equivalent under input negation/permutation as well as output negation.
The data structures used to verify function equivalence in this paper are the Binary Decision Diagrams (BDDs) [1]. BDDs are the most used form to represent Boolean functions in Electronic Design Automation. There are many specific kinds of BDDs, depending on its use. Reduced Ordered Binary Decision Diagrams (ROBDDs) are commonly used to compare Boolean functions using the ROBDD strong canonical form (unique representation) presented in [2]. This canonical form is necessary to verify the equivalence of the functions and group them into a common equivalence class. More details about BDDs can be found in [1][2][3][4].
This paper is organized as follows. The implementation of n-input functions is discussed in section 2. Sections 3 and 4 introduce the concepts of P classes, and NPN classes, respectively. Section 5 presents some results for the

classification of 2, 3 and 4-input functions. The concepts introduced in this paper are worked out in section 6, while conclusions are discussed in section 7.

2 Implementation of n-input functions



$f_0 = 0$	
$f_1 = \overline{A} . \overline{B}$	
$f_2 = \overline{A} . B$	
$f_3 = \overline{A}$	
$f_4 = A . \overline{B}$	
$f_5 = \overline{B}$	
$f_6 = A \oplus B$	
$f_7 = \overline{A . B}$	
$f_8 = A . B$	
$f_9 = \overline{A \oplus B}$	
$f_{10} = B$	in
$f_{11} = \overline{A . \overline{B}}$	
$f_{12} = A$	in
$f_{13} = \overline{\overline{A} . B}$	
$f_{14} = \overline{\overline{A} . \overline{B}}$	
$f_{15} = 1$	

Figure 2: Implementations for all the 16 different 2-input functions.

The number of different n-input functions is *2^(2^n)*, as already demonstrated in section 1. Figure 2 shows the gates needed to implement each of the 2-input functions from figure 1. Note that many different functions are implemented using the same gates.

3 The concept of P class

The fact that many different 2-input functions may have the same gate-level implementations naturally introduces the concept of P equivalence. P equivalence between 2 functions is obtained when it is possible to achieve identical values for both truth table outputs by permuting the function inputs. Functions that are P equivalent can be grouped into P classes. For instance, the functions and are P equivalent. Figure 3 shows all the 12 different P classes of 2-input functions. It is important to note that despite the existence of 16 different 2-input functions, there are only 12 different 2-input P classes. The circuits used to implement each P class are also shown in figure 3. Four P classes are composed by 2 functions, while eight P classes are composed by only one function. The most important property of P equivalent functions is that they can always be implemented with the same circuit (or cell from a library). Therefore, it is possible to implement any of the 2-input functions with a single cell from a library composed of one gate implementation for each P class.

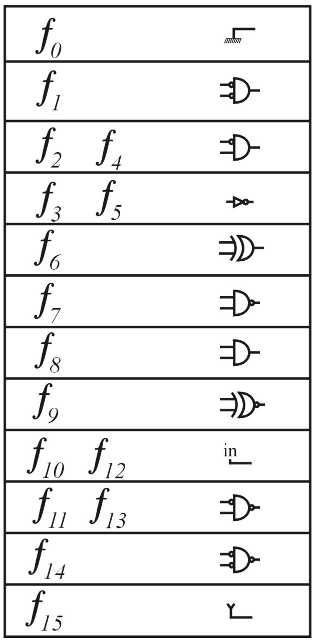


Figure 3: The 12 different 2-input P classes.

4 The concept of NPN class

From figure 3, it is possible to see that even P classes may have similar implementations. For instance, functions f_1, f_2, f_4, f_7, f_8, f_{11}, f_{13} and f_{14} have gate implementations based on a single *nand* gate plus some inverters. These functions may be grouped into a NPN equivalence class. NPN equivalence between 2 functions is obtained when it is possible to achieve identical values for both truth table outputs by permutation and/or negation of the function inputs and/or negation of the function output. Figure 4 shows all the 4 different NPN classes of 2-input functions, one NPN class per line. It is important to note that despite the existence of 16 different 2-input functions, there are only 4 different 2-input NPN classes. There are 2 NPN classes composed of 2 functions, one NPN class composed of 4 functions, and one NPN class composed of 8 functions. NPN equivalent functions can be implemented with the same circuit plus some inverters (used in the negation operations for the inputs and the output, if necessary). This way, it is possible to use a smaller library composed of one representative gate for each NPN class plus one inverter cell. This approach is specially useful when the cost of the inverter is very low.

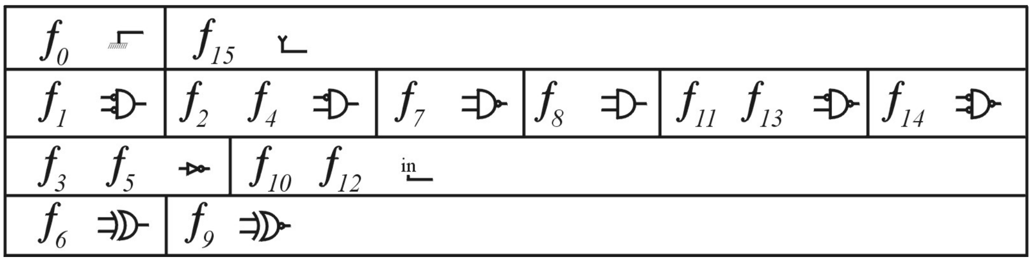


Figure 4: The 4 different 2-input NPN classes.

5 Results

In order to classify n-input functions into P and NPN classes, a tool that constructs minimal implementations of Boolean functions in the ROBDD strong canonical form was implemented. By performing input permutations, the set of P classes is derived. By performing permutation/negation at the inputs and negation at outputs the set of NPN classes is derived. As the equivalence-checking tool is based on ROBDDs, we used the number of nodes in the final ROBDD implementation as a parameter of the relative complexity of functions and classes. Results are shown in tables 2, 3 and 4, respectively, for 2-input, 3-input and 4-input functions.

Table 2: Number of 2-input functions, P classes and NPN classes according to their ROBDD size.

Nodes	BDDs	P	NPN
0	2	2	1
1	4	2	1
2	8	6	1
3	2	2	1
Total	16	12	4

Table 3: Number of 3-input functions, P classes and NPN classes according to their ROBDD size.

Nodes	BDDs	P	NPN
0	2	2	1
1	6	2	1
2	24	6	1
3	62	26	4
4	88	30	4
5	74	14	3
Total	256	80	14

Table 4: Number of 4-input functions, P classes and NPN classes according to their ROBDD size.

Nodes	BDDs	P	NPN
0	2	2	1
1	8	2	1
2	48	6	1
3	236	26	4
4	960	204	14
5	3248	710	38
6	8928	1342	70
7	17666	1272	68
8	23280	420	25
9	11160	0	0
Total	65536	3984	222

The results presented in table 2 confirm the number of 2-input functions, P classes and NPN classes discussed earlier in figures 1 to 4. Table 4 shows the number of different 4-input functions, P and NPN classes, according to the size of its ROBDD representation. First column shows the BDD sizes considering the number of non-terminal nodes. BDD sizes vary from 0 to 9, and this is compatible with the upper bound for BDD size derived in [4]. All the BDD sizes in table 1 were obtained with the same variable ordering [3]. In the case of P and NPN classes, the smaller BDD size for the functions belonging to the class is considered.

Table 5: Number of functions, P classes, and NPN classes according to the number of inputs.

# of inputs	Functions	P classes	NPN classes
1	4	4	2
2	16	12	4
3	256	80	14
4	65536	3984	222

6 Working out the concepts

In order to understand the criteria that were used for the classification of n-input functions, some related affirmations will be discussed. As the examples involves the concept of Universal Logic Gates (ULGs), we will begin by explaining this concept.

6.1 The concept of ULG

An Universal Logic Gate (ULG) is a programmable logic gate that can implement several different logic functions, according to its configuration. The term *Universal* was introduced because this kind of gate normally can implement all the n-input functions for a given n. This way an *Universal* logic gate could implement all the *universe* of n-input functions. However, this is not always true. For instance, if an ULG is able to implement all the n-input functions, it is not necessarily able to implement all the (n+1)-input functions. This way an ULG can only implement a limited number of functions, P classes or NPN classes. A classic example of implementation of an ULG as a binary decision tree is shown in figure 5. In this binary decision tree, a variable is evaluated at each level. If the variable value is 0, the left arc is activated. Otherwise, the value is 1 and the right arc is activated. This way, if *abc=101* then the output value is the minterm m_5. The ULG works similarly for all the other minterm values, represented as squares in figure 5. These minterms are stored in flip-flops, therefore they are programmable to be equal to 0 or 1. Decision nodes, represented in figure 5 as circles, are implemented with multiplexors.

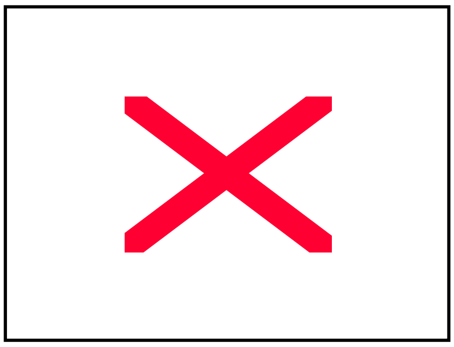


Figure 5: ULG viewed as a binary decision tree.

6.2 How many cells in the library?

Based on the ULG implementation presented in figure 5, it is possible to state some misleading conclusions which unadvertised people could accept as true.

> **Fallacy:** This ULG is able to implement all the 256 3-input functions, so it is as powerful as a cell library composed of 256 cells.

This is not true, because several different 3-input functions may be implemented with the same circuits. These functions are grouped in P classes. Therefore, it is necessary only 80 (the number of 3-input P classes) cells in a library able to implement any of the 3-input functions with only one cell.

6.3 How to measure ULGs?

The question raised in the previous subsection leads to another question: Which number should be used to measure the efficacy of an ULG? There are 3 criteria: the number of functions, P classes or NPN classes it is able to implement. The number of functions is not a good criterion because many functions have the same implementation. Therefore, the number of P classes was used. What about to use the number of NPN classes?

> **Pitfall:** My ULG is able to implement 180 out of the 222 NPN classes of 4-input variables.

This affirmation is correct only if the ULG provides a mean to implement input and output negation. This way, the ULG can implement complete NPN classes. If the input/output negation is not allowed, the ULG will be able to implement some NPN classes only partially. Some functions of the NPN class can be implemented while others cannot. This way, it makes no sense to say that this NPN class can be implemented neither to say that it cannot. The best number to be used to classify ULGs is the number of P classes that it can implement, because input permutation of a cell is always allowed. If input/output negation is available at no cost, it makes sense to use NPN classes for classification. The origin of this pitfall is that the first ULGs allowed input/output negation, thus it was possible to classify them by using NPN classes. Afterwards, new ULGs that did not allow input/output negation were proposed, leading to this pitfall.

6.4 The cost of an ULG

At this point an attentive reader could ask the following question:

> **Disturbing question:** Why use ULGs if every Boolean function can be implemented by using only 2-input nand cells?

The use of ULGs offers programmability at later levels of implementation. For instance, there are circuits based on ULGs that can be quickly customized by using the higher metal levels. FPGAs based on RAM can be programmed more than once. The cost of an ULG is usually higher than an implementation using several cells from a library, but this cost is compensated by the programmability. When using cells from a library, it doesn't matter how many cells are used to implement each logic function, only the final cost matters. For ULG based logic, it is interesting to have more powerful ULGs that are able to implement a larger number of functions, while maintaining the cost of the ULGs as low as possible.

6.5 When inverters have no cost?

As the criterion used (P or NPN classes) in the classification depends on the cost of input/output inversion, it is interesting to discuss when inverters have no cost. In the case of ULGs, it is possible to invert input variables that are selection variables of multiplexors: it is possible to invert these variables by exchanging the multiplexed functions. The negation of the outputs can be easily done when the minterms are stored in flip-flops. However, even for multiplexor and flip-flop based ULGs, the inversion operation may not be available for all the inputs/outputs due to hardwired connections. This way, it is necessary to be very careful when using NPN classes to measure the efficacy of ULGs.

7 Conclusion

This paper discussed the classification of n-input functions considering the number of functions, P classes and NPN classes. The number of P classes is the best criteria, because each P class corresponds to a single implementation, and input permutation is always allowed. A BDD based tool that groups functions into P and NPN classes was developed in order to enumerate the number of P and NPN classes of n-input functions. Future work will include an investigation of minimal implementations for each P class considering several kinds of logic, like static CMOS logic [5], pass transistor logic[6] [7], and the ones used in [8] [9].

8 References

[1] R.E.Bryant. "Graph-based algorithms for Boolean function manipulation", *IEEE Transactions on Computers*, vol. C-35, n° 8, pp. 677-691, August 1986.
[2] K.S.Brace, R.L.Ruddel, R.E.Bryant. "Efficient implementation of a BDD package". Proc. of 27th DAC, pp. 272-277, 1990.
[3] Steven J. Friedman and Kenneth J. Supowit, "Finding the Optimal Variable Ordering for Binary Decision Diagrams", IEEE Transactions on Computers, Vol. 39, No. 5,pp. 710-713, May 1990.
[4] Heh-Tyan Liaw and Chen-Shang Lin, "On the OBDD-Representation of General Boolean Functions", IEEE Transactions on Computers, Vol. 41,No. 6, pp. 661-664, June 1992.
[5] T. Ibaraki, S. Muroga, "Synthesis of Networks with a Minimun Number of Negative Gates", IEEE Transactions on Computers, Vol. C-20, No. 1,pp. 49-58, January 1971.
[6] P. Buch, A. Narayan, A. R. Newton, A. Sangiovanni-Vincentelli, "On Synthesizing Pass Transistor Networks", IWLS '97, pp. 101-108.
[7] V. Bertacco, S. Minato, P. Verplaetse, L. Benini, G. de Micheli, "Decision Diagrams and Pass Transistor Logic Synthesis", IWLS '97, pp. 109-113, May 1997.
[8] Charles R. Baugh, C. S. Chandersekaran, Richard S. Swee and Saburo Muroga, "Optimal Networks of NOR-OR Gates for Functions of Three Variables", IEEE Transactions on Computers, Vol. C-21, No. 2,pp. 153-160, February 1972.
[9] J. N. Culliney, M. H. Young, T. Nakagawa and S. Muroga, "Results of the Synthesis of Optimal Networks of AND and OR Gates for Four-Variable Switching Functions", IEEE Transactions on Computers, Vol. C-27, No. 1,pp. 76-85, January 1979.
[10] E. Detjens, G.Gannot, R.Rudell, A.L.Sangiovanni-Vinccentelli, A.Wang. "Technology mapping in MIS" *ICCAD*, 1987, pp. 116-119.