BOOLEAN REPRESENTATION CODE – AN EFFICIENT METHOD TO REPRESENT BOOLEAN FUNCTIONS

Vinícius N. Possani, Renato S. Souza, Julio S. Domingues Jr., Felipe S. Marques, Leomar S. da Rosa Jr.

Group of Architectures and Integrated Circuits – GACI
Technology Development Center - CDTec
Federal University of Pelotas – UFPel
Pelotas, RS, Brazil

{vnpossani, rsdsouza, jsdomingues, felipem, leomarjr}@inf.upel.edu.br

ABSTRACT

CAD (Computer Aided Design) tools are currently indispensable in the development of digital circuits due to the feasibility of adapting technology parameters. They are widely used in different design levels, from high-level synthesis to layout design, simulation, analysis and verification. This paper describes a quickly and secure method to generate Boolean Representation Code of logic functions to efficiently represent Boolean functions. To perform a case of study, the proposed method was applied in the Soptimizer to validate the optimizations performed by the tool. Experiments show a reduction in runtime up to 41.4% when comparing to the previously adopted strategy

Categories and Subject Descriptors

B.6.3 [Logic Design]: Design Aids – automatic synthesis, optimization.

General Terms

Algorithms, performance, design, experimentation, theory.

Keywords

Logic synthesis, switching theory, transistor networks.

1. INTRODUCTION

Electronic devices are increasingly present in our days, causing a great impact on society, due to the fact that they apply directly to different areas of knowledge. Thus, it has been noted the importance of advances in the development of digital circuits. Due to that it is possible to create new technologies. Consequently, great difficulties are eventually found due to adaptation of new technology parameters, as the complexity of designing a chip in a time short enough that the product is launched on the market. In this scenario, CAD (Computer Aided Design) tools have contributed to developers increase the efficiency and reduce the complexity in a project [1-6].

Following this trend, a tool that implements a graph-based method to generate transistor networks was proposed. This tool is called Soptimizer [7]. Basically, from a Boolean expression, it is obtained a graph, where each edge represents a transistor, and, in a posterior step, it is performed an optimization process by edges sharing, reaching a reduced network in terms of switches. However, due to the edges sharing process, it can be introduced new paths in the graph, which may change the logical behavior of the function. Therefore, it is necessary to ensure that these new paths do not change the logical behavior of the circuit that is represented by the graph.

Thus, this paper describes a method to generate a Boolean Representation Code (BRC) of a logic function. The proposed method is incorporated in Soptimizer tool to verify if the new paths in the graph are valid and have not changed the logical behavior of the circuit. Apart from that, by using the proposed method, the Soptimizer tool becomes able to perform some algebraic optimizations that are not possible when using the previous solution.

2. BOOLEAN REPRESENTATION CODE

The main idea of this method is to generate a BRC for a Boolean function. The first step consists in checking how many variables there are in the function. The proposed method represents the BRC by integers. So, to discover how many integers are necessary, it is computed 2^n, where n represents the number of variables existing in the input function. After that, the result is divided by 32. If needed to use more of one integer to represent the BRC, than it is used a structure of vector to store each integer. For example, in a case that a function has six variables, the result of calculation $2^6/32$ is equal to 2. So it is needed two integers to generate the basic BRC for each variable. A vector is used to guarantee that during the logical operations the comparisons are performed correctly, where each integer in the vector is compared with another integer in an equivalent position.

After verifying how many integers are necessary to create a BRC, the method generates the basic BRC, which are the BRC of each

✓ 1 variable:	✓ 2 variable:	✓ 3 variable:	✓ 4 variable:	✓ 5 variable:
var1 = 1	var1 = 5	var1 = 85	var1 = 21845	var1 = 1431655765
	var2 = 3	var2 = 51	var2 = 13107	var2 = 858993459
		var3 = 15	var3 = 3855	var3 = 252645135
			var4 = 255	var4 = 16711935
				var5 = 65535

Figure 1. Default values of basic BRC for each variable according to the number of variables present in the input function.

variable in the input function. If the function has no more than five variables, it is performed a naïve assigning process, where each variable receive a BRC as shown in Figure 1. The data present in Figure 1 were generated by concatenating bits, a similar process of mounting a truth table. Figure 2 exemplifies that when considering two variables.

✓ 2 variable:

var1 = 0000 0000 0000 0101 = 5

var2 = 0000 0000 0000 0011 = 3

Figure 2. BRC when considering two variables.

When the function contains more than five variables, the process of generating basic BRC is modified. So, it is used a vector. For the first five variables, the same values are used for the five variables indicated in Figure 1. These values are written in all positions of the vector according to the corresponding variable. Then, for the first variable is assigned the value 1431655765 for all positions of the vector. This is done for all the next four variables, changing only the value of the assignment for each case. For the next variables it is performed a process in which it is concatenated the value of 0 and -1 at each position of vector. The number of concatenations of 0 and -1 required is indicated by 2^{n-5}. This process resembles the method of assembling a truth table, where each variable is represented by a sequence of bits in the columns of the table. Figure 3 shows the reason for use 0 and -1 during the basic BRC generation.

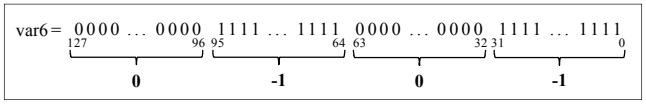

Figure 3. Splitting a bit sequence and associating to an equivalent integer.

This whole process of generating basic BRC for a given function that contains more than five variables is shown in Figure 4.

In a case where some variable of the function is negated, the process of basic BRC generation is the same. It will be assigned, for this negated variable, the value presented in Figure 1. The main difference is that bits that are 0 become 1, and those that are 1 become 0. In a case where the function has more than five variables, it is performed the same procedure of concatenation explained before. The only difference is the order of concatenation of the values in vector. Firstly, it is concatenation the value -1. After that, the value 0 is concatenated. Figure 5 shows a case of a random function that contains the third and seventh negated variables.

✓ 7 variable:

var1 =	1431655765	1431655765	1431655765	1431655765
var2 =	858993459	858993459	858993459	858993459
var3 =	252645135	252645135	252645135	252645135
var4 =	16711935	16711935	16711935	16711935
var5 =	65535	65535	65535	65535
var6 =	0	-1	0	-1
var7 =	0	0	-1	-1

Figure 4. Vectors with the values for each variable.

!var3 =	-252645136	-252645136	-252645136	-252645136
!var7 =	-1	-1	0	0

Figure 5. Vectors with the values for each negated variable.

After generating basic BRC, it is created the BRC for the entire function by using logical operations AND and OR. Expression 1 shows the function used as example.

$$A*C*E*F + A*B*F + A*B*!C + D*E*!G \qquad \text{(Exp. 1)}$$

First it is obtained the BRC of the products through bitwise AND operation of each integer value present in a position of the vector with the other integer value of the corresponding position in the next vector. After that, it is performed the bitwise OR operation between the BRC generated before. Figure 6 shows a BRC generation of a product through a bitwise AND operation performed between each integer of vectors.

After generating all BRC of the products, it is performed the bitwise OR operation between each position of vectors of each BRC of these products. This process is illustrated in Figure 7, which also shows the BRC of the function shown in Expression 1.

D =	16711935	16711935	16711935	16711935
	↓AND↓	↓AND↓	↓AND↓	↓AND↓
E =	65535	65535	65535	65535
	↓AND↓	↓AND↓	↓AND↓	↓AND↓
!G=	-1	-1	0	0
D*E*!G =	255	255	0	0

Figure 6. Generation of the BRC for the product D*E*!G.

A*C*E*F =	0	1285	0	1285
	↓OR↓	↓OR↓	↓OR↓	↓OR↓
A*B*F =	0	286331153	0	286331153
	↓OR↓	↓OR↓	↓OR↓	↓OR↓
A*B*!C =	269488144	269488144	269488144	269488144
	↓OR↓	↓OR↓	↓OR↓	↓OR↓
D*E*!G =	255	255	0	0
Exp.1 =	269488383	286332415	269488144	286332181

Figure 7. Generation of the BRC for Expression 1.

3. EXPERIMENTAL RESULTS

The proposed method was implemented in Java using NetBeans IDE 7.0 and was integrated into the Soptimizer tool to be validated and tested as a case of study. In order to evaluate the efficiency of the proposed method, it was used as benchmark all functions from the 4-input p-class logic functions set [8]. This set is composed by 3982 Boolean functions. Also, it was used 54 random logic functions with six input variables, called Random6. Apart from that, three functions were chosen for analysis. The XOR 4 was chosen because it is extremely used in several circuits such as adders and multipliers. Functions F5 and F13 [9, 10], were chosen because they contains a large number of variables if comparing to the 4-input p-class logic functions set.

Table 1 presents the results obtained in terms of runtime. The column "Without BRC" shows the results when Soptimizer tool uses the old version algorithm to compare functions equivalence. This algorithm consists in traversing the graph and obtaining all cubes that compose the function. In the sequence, each cube obtained from the graph is compared to the ones from the input expression. The column "With BRC" shows the runtime when the proposed method is used. The column “Reduction” reports the percentage of gain and loss in runtime. These tests were executed on a computer with an Intel Pentium Dual Core T2370 1.73GHz, 2GB of memory and Windows Seven Ultimate 64bit.

As can be seen in the results of Table 1, for the benchmarks p-class, Random6, XOR 4 and F13, the total runtime of the Soptimizer tool is smaller when using the proposed algorithm. However, for the benchmark F5, the obtained runtime was worst when using the proposed algorithm. The main reason for that is that the benchmark F5 contains large cubes, with few variables. In this situation the proposed algorithm presents a disadvantage if comparing to the old strategy usage by the Soptimizer tool. All the process to generate the BRC and compare them when necessary is more time consuming than just directly compare products stored in vector structures. Our method is able to deliver better results when there are several cubes to be checked in a SOP form.

Also, a study was conducted to evaluate the individual runtime for each logic functions that compose the 4-input p-class benchmark. This way, it was possible to perform a better analysis in which functions the runtime was reduced. Figure 8 shows a graphic that summarizes these results.

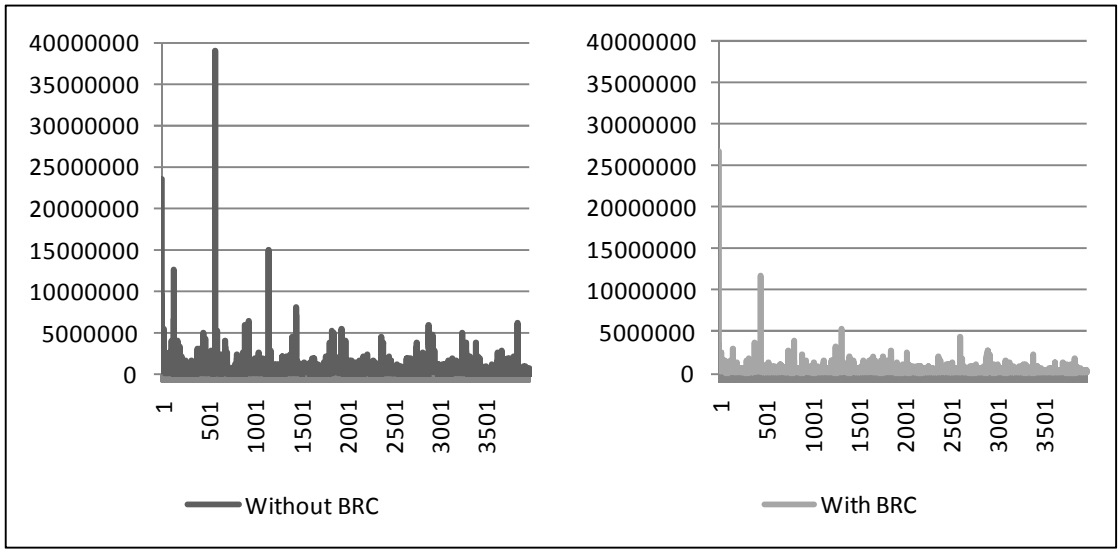

Figure 8. Runtime comparison when using 4-input p-class benchmark to generate transistor networks in the Soptimizer tool.

Table 1. Total runtime obtained by Soptimizer tool with and without using the BRC proposed method.

Benchmark	Number functions	Number variable	Without BRC	With BRC	Reduction
p-class	**3.982**	**4**	2193 ms	1599 ms	27,1%
Random6	**54**	**6**	189 ms	156 ms	17,5%
XOR 4	**1**	**4**	57 ms	47 ms	17,6%
F5	**1**	**8**	78 ms	100 ms	-28,3%
F13	**1**	**10**	546 ms	320 ms	41,4%

4. CONCLUSIONS AND FUTURE WORK

This paper presented a method to generate a Boolean Representation Code which can efficiently represent Boolean functions. At a first moment, the proposed method was integrated into the Soptimizer tool to validate the optimization process of transistor networks.

The method was validated using several Boolean functions with different number of input variables.

The results demonstrated that the algorithm can minimize the total runtime when incorporated in a CAD tool. In the case of study, it was possible to achieve an average gain of 27.1% in runtime when considering 4-input p-class benchmark.

As future work, more tests will be performed considering different benchmarks. Also, it is intended to incorporate the proposed method into other Boolean evaluation algorithms developed by the group, especially in those related to technology mapping.

5. ACKNOWLEDGMENTS

Research partially supported by Brazilian funding agencies CNPq and FAPERGS, under grant 11/2053-9 (Pronem).

6. REFERENCES

[1] Da Rosa Junior, L. S.; Marques, F. S.; Cardoso, T. M. G.; Ribas, R. P.; Sapatnekar, S.; Reis, A. I. *Fast Disjoint Transistor Networks from BDDs*. In: 19th ACM Symposium on Integrated Circuits and Systems Design, 2006, p. 137-142.

[2] Callegaro, V.; Marques, F. S.; Klock, C. E.; Da Rosa Junior, L. S.; Ribas, R. P.; Reis, A. I. *SwitchCraft: a framework for transistor network design*. In: 23rd Symposium on Integrated Circuits and System Design, 2010, p. 49-53.

[3] Das, S.; Chandrakasan, A.; Reif, R. *Three-dimensional integrated circuits: performance, design methodology, and CAD tools*. In: IEEE Computer Society Annual Symposium on VLSI, 2003, p. 13-18.

[4] Keutzer, K.; Vanbekbergen, P. *The impact of CAD on the design of low power digital circuits*. In: IEEE Symposium on Low Power Electronics, 1994, p. 42-45.

[5] Da Rosa Junior, L. S.; Marques, F. S.; Schneider, F.; Ribas, R. P.; Reis, A. I. *A Comparative Study of CMOS Gates with Minimum Transistor Stacks*. In: 20th ACM Symposium on Integrated Circuits and Systems Design, 2007, p. 93-98.

[6] Da Rosa Junior, L. S.; Schneider, F.; Ribas, R. P.; Reis, A. I. *Switch Level Optimization of Digital CMOS Gate Networks*. In: 10th IEEE International Symposium on Quality Electronic Design, 2009, p. 324-329.

[7] Possani, V. N.; Souza, R. S.; Domingues Jr., J. S.; Agostini, L. V.; Marques, F. S.; Da Rosa Junior, L. S. *Optimizing Transistor Networks Using a Graph-Based Technique*. In: Journal of Analog Integrated Circuits and Signal Processing, Springer, 2012.

[8] Ledur, M.; Marranghello F.; Da Rosa Junior, L. S.; Reis, A. I.; Ribas, R. P. *Set of Digital Cells According to Logic Equivalences*. In: VII Student Forum on Microelectronics, 2007.

[9] J. Zhu et al. *On the Optimization of MOS Circuits*. In: IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications, 1993, p. 412-422.

[10] D. Kagaris et al. *A Methodology for Transistor-Efficient Supergate Design*. In: IEEE Transactions on Very Large Scale Integration Systems, 2007, p. 488-492.