# An Effective Method for Generating Boolean Signatures

**Renato S. de Souza, Vinícius N. Possani, Julio S. Domingues Jr,**
**Felipe S. Marques, Leomar S. da Rosa Jr.**
{rsdsouza, vnpossani, jsdomingues, felipem, leomarjr}@inf.ufpel.edu.br

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

## Abstract

*CAD (Computer Aided Design) tools are currently indispensable in the development of digital circuits due to the feasibility of adapting technology parameters. This paper proposes a quickly and secure method based in signature of logic functions to represent Boolean functions efficiently. To perform a case of study, the proposed method was used in the Soptimizer tool to validate the optimizations performed by the tool. Experiments show a reduction in runtime of 41.4% when comparing to the previously adopted strategy.*

## 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 context, CAD (Computer Aided Design) tools have contributed to developers increase the efficiency and reduce the complexity in a project [1, 2].

In this context a tool that implements a graph-based method to generate transistor networks was proposed. This tool is called Soptimizer [3]. 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 sharing of edges, reaching a reduced network in terms of switches. Due to the shares of edges performed during the optimization 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 proposes a method to generate logic functions signatures which can efficiently represent Boolean functions. The proposed method is incorporate in Soptimizer tool to verify, from quickly to a secure way, 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. Method for Generating Boolean Signatures

The main idea of this method is to generate a signature for a Boolean function. The first step consists in checking how many variables there are in the function. The proposed method represents the signatures 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 signature, 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 signatures for each variable. A vector is used to guarantee that 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 signature, the method generated the basic signatures, which are the signatures of each 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 signature as shown in Fig.1. The data present in Fig.1 were generated by the concatenation of bits, a similar process of mounting a truth table. Fig.2 exemplifies that when considering two variables.

| ✓ 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 |

Fig. 1 – Default values of basic signatures for each variable according to the number of variables present in the input function.

✓ 2 variable:

var1 = 0000 0000 0000 0101 = 5
var2 = 0000 0000 0000 0011 = 3

Fig. 2 – Signatures when considering two variables.

When the function contains more than five variables, the process of generating basic signatures is modified. So, it is used a vector. For the first five variables, the same values are used for five variables indicated in Fig.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 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.

This whole process of generating basic signatures for a given function that contains more than five variables is shown in Fig.3.

✓ 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 |

Fig. 3 – Vectors with the values for each variable.

In a case where some variable of the function is negated, the process of basic signature generation is the same. It will be assigned, for this negated variable, the value presented in Fig.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. Fig.4 shows a case of a random function that contains the third and seventh negated variables.

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

Fig. 4 – Vectors with the values for each negated variable.

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

A*C*E*F + A*B*F + A*B*!C + D*E*!G (Exp. 1)

First it is obtained the signatures of the products through of bitwise AND operation of each integer value present in a position of the vector with the other integer value of the corresponding position of the next vector. After that, it is performed the bitwise OR operation between the signatures generated before. Fig.5 shows a signature generation of a product through a bitwise AND operation performed between each integer of vectors.

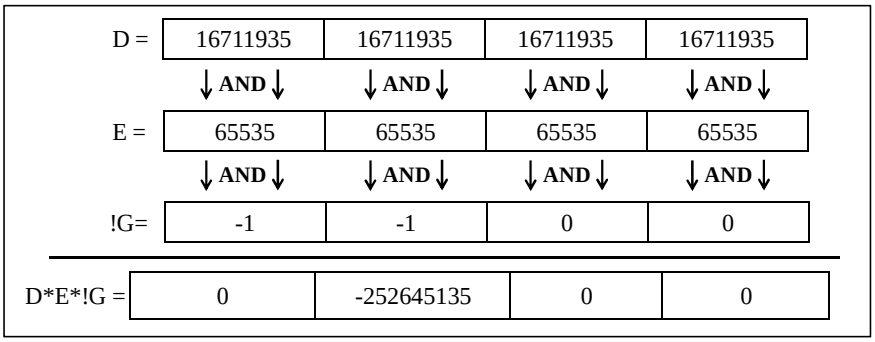


Fig. 5 – Generation of the signature for the product D*E*!G.

After generating all signatures of the products, it is performed the bitwise OR operation between each position of vectors of each signature of these products. This process is illustrated in Fig.6, which also shows the signature of the function shown in Exp.1.

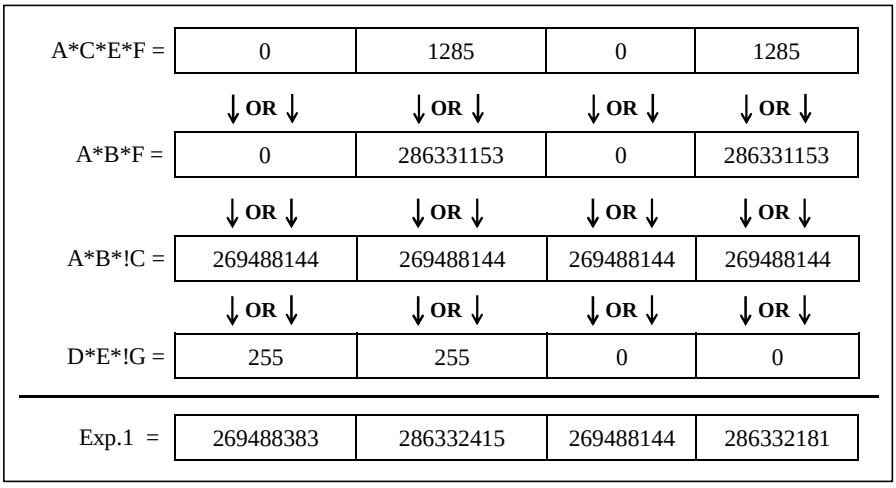


Fig. 6 – Generation of the signature for Exp.1.

# 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 [4]. 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 an extremely used in several circuits such as adders and multipliers. Functions F5 and F13 [5, 6], 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 signature" 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 signature" shows the runtime when the proposed method is used. These tests were executed on a computer with an Intel Pentium Dual Core T2370 1.73GHz, 2GB of memory and Windows Seven Ultimate 64bit.

Tab.1 – Total run time in ms tool Soptimizer with and without the signature method.

| Benchmark | Number functions | Number variable | Without signature | With signature | 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% |

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 signatures 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.

# 4. Conclusions and Future Works

This paper presented a method to generate logic functions signatures which can efficiently represent Boolean functions. At a first moment, the signature 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.

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 . Acknowledgment

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

# 6 . References

[1] Da Rosa Junior, L. S. Automatic Generation and Evaluation of Transistor Networks in Different Logic Styles. PhD Thesis PGMicro/UFRGS, Porto Alegre, Brazil. (2008), 147 p.

[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, São Paulo. Proceedings of the 23rd Symposium on Integrated Circuits and System Design. New York : ACM, 2010. p. 49-53.

[3] Possani, V. N. ; Timm, E. F. ; Agostini, L. V. ; Da Rosa Junior, L. S. . Transistor networks design using a graph-based approach. In: 10th Microelectronics Students Forum, 2010, São Paulo. 10th Microelectronics Students Forum, 2010.

[4] 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, Rio de Janeiro. VII Student Forum on Microelectronics CDROM. Porto Alegre : SBC, 2007.

[5] J. Zhu et al. On the Optimization of MOS Circuits. IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications. (1993), 412-422.

[6] D. Kagaris et al. A Methodology for Transistor-Efficient Supergate Design. IEEE Transactions On Very Large Scale Integration (VLSI) Systems. (2007), 488-492.