Improvements of the SwitchCraft Framework

Gabriel Ammes, Jeferson José Baqueta[1], André I. Reis[1,2], Renato P. Ribas[1,2]
[1]PPGC / [2]PGMicro / Institute of Informatics
UFRGS, Porto Alegre, Brazil
{gabriel.ammes, jeferson.josebaqueta, andreis, rpribas}@inf.ufrgs.br'

Abstract— **With the constant growth of the number of devices in a single chip, the use of CAD tools is essential to manufacture efficient and optimized integrated circuits (IC). For each step of the IC design flow, there is a set of CAD tools that may be applied to automate several tasks, such as the algebraic and Boolean factoring, technologic mapping, local and global routing, test and verification of failures, among others. In this context, we presented some improvements performed on the SwitchCraft Framework, a CAD tool used to generate transistors networks and logic gates for CMOS technology. Among the new features implemented are the inputs and output logic formats, And-inverter graph (AIG) and threshold logic format, besides the implementation of a transistor placement method, which may be used to decrease the congestion of a logic gate.**

Keywords—CAD tools, design of digital circuit, CMOS logic gate;

I. INTRODUCTION

Along of the years, more compacts and faster circuits have been manufactured due to high density of transistors in a single chip. However, to provide the high density of components, CAD tools more complex and optimized are needed. For instance, for the VLSI circuits, composed by thousands of transistors, specific CAD tools are used to evaluate and automate the design of such circuits [1] [2]. Therefore, due to number components used in this type of circuit, the manual synthesis is unfeasible.

In this sense, to keep up the technology advancement, it is essential that the CAD tools are flexible. This way, such tools may offer the needed support for the constants technology innovations. Currently, there is a great amount of academic and commercial specific CAD tools adopted to automate different task in relation to synthesis of integrated circuits [3][4][5][6].

The representaion of Boolean functions is one important task performed during the integrated circuit synthesis, since the such representation may impacts in several others parts of the IC design. In this context, the SwitchCraft framework [7] was proposed, offering a complet set of tools for switch network and logic gates generation. This way, we propose two new features that were implement in the SwitchCraft framework. The first is a expansion of the inputs and outputs logic formats supported by SwitchCraft, incluing the And-inverter graph (AIG) and threshold logic format. Another improvement performed was the implementation of a placement transistor method. In such method the Euler path algorithm is applied to obtain a compact CMOS logic gate. In this context, a quite simple evaluation metric adopted is wire congestion estimation. Since several possible paths can be generated through a input switch network.

The rest of this paper is organized as follow. Section II presents some logic structures and definitions. Section III presents the SwitchCraft framework and the improvements performed. Section IV presents the transistor placement method. Section V presents the obtained results, whereas the conclusions are presented in Section VI.

II. BACKGROUND AND DEFINITIONS

This section reviews some useful definitions regarding the logic structures used to represent Boolean functions adopted in this work.

A. Switch networks and CMOS gate

A logic switch is the most basic element of a switch network. Such logic switch contains one control terminal, gate (G), and two contact terminals, source (S) and drain (D). The control terminal determines if there is a connection between the contact terminals [8]. The two possible configuration for a logic switch, the direct and complementary switch are present in Fig.1(a) and Fig.1(b), respectively. Through an arrangement of logic switches a given Boolean function may be implemented. Usually, according to the kind of logic switch associations present in a switch network, such network can be classified in serie-parallel (SP) and non-serie-parallel (NSP) network. In Fig.1(c) a NSP switch network is shown, whereas in Fig1(d) a SP is presented.

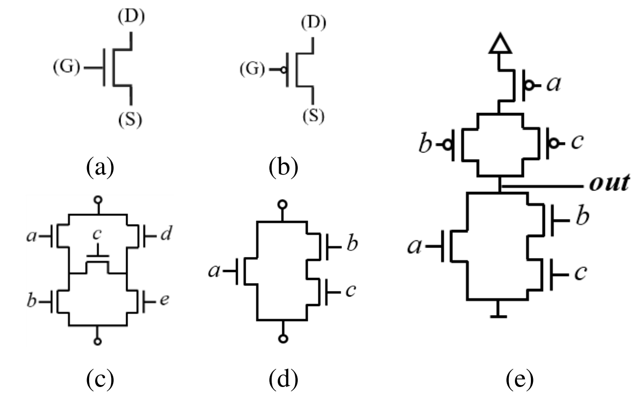

Fig. 1 switch networks and CMOS logic gate representation: (a) direct logic switch; (b) complementary logic switch; (c) NSP switch network; (d) SP switch network; (e) CMOS logic gate that implements the Boolean function $F = a+(b*c)$.

The switch networks can be used to implement the CMOS logic gate. Basically, a gate is composed by two complementary planes, *pull-up* plan and *pull-down* plan. The *pull-up* plan is composed by P-MOS transistors, which are represented by complementary logic switches, and *pull-down* plan is composed by N-MOS transistors, which are represented by direct logic switches. In Fig.1(e) it is shown the CMOS logic gate that implements the Boolean function $F = a+(b*c)$.

B. Threshold logic

The threshold logic functions (TLF) are a subset of Boolean functions, which can be implemented by a structure called threshold logic gate (TLG). Such structure is composed by a set of inputs (x_i), where each input has a weight value (w_i), and a threshold value (T) associated to the function. The logic behavior of the TLF is defined by the sum of the inputs weights, as follows:

$$F = \begin{cases} 1, & \sum_{i=1}^{n} w_i x_i \geq T \\ 0, & otherwise \end{cases} \quad (1)$$

All threshold functions can be implemented through a single TLG. In Fig. 2 some TLG are shown, in this case the weights of the inputs and the Threshold value are defined as integers. For example, in Fig. 2 (a) the logic function AND3 is shown, this way, each input has its weight equals to 1 and the threshold value is 3. The process to identify if a given Boolean function is threshold, is called TLF identification [9]. If a given function is not TLF, more than one gate is necessary to implements such a function. The set of TLGs used to implement a non-TLF is called TLG network. In Fig. 2(d) is shown a TLG network corresponding to $F = (a*b)+(c*d)$.

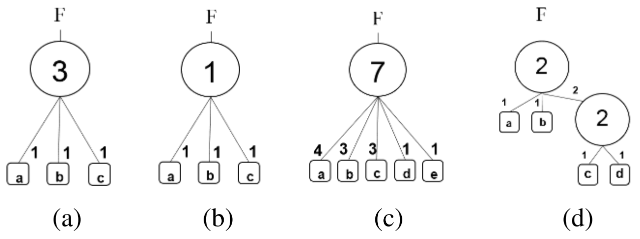

Fig. 2 Threshold logic function implementations; (a) 3-input OR; (b) 3-input AND; (c) $F = (a*b) + (a*c) + (b*c*d) + (b*d*e)$; and (d) $F = (a*b) + (c*d)$.

C. And Inverter Diagram (AIG)

An and-inverter graph (AIG) is directed acyclic graphs (DAG) where the nodes are represented by AND2 gates or primary inputs. The nodes of a AIG are conneted by edges, which has its polarity defined by inverters represented by bubbles on the edges. An exemple of AIG is presented in Fig.3, the Boolean function F = ($x1 + x2 + (x3 * x4)$).

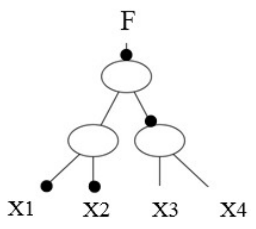

Fig. 3 AIG representation: such structure implements the Boolean function $F = (x1 + x2 + (x3 * x4))$

III. SWITCHCRAFT IMPROVEMENTS

The main goal of the SwitchCraft framework is to provide a set of algorithms and methods to help designers to generate transistors networks and logic gates, as well as to estimate and evaluate them [7]. Basically, such framework is divided in four main modules:

- *Data input module*: offer support to several inputs formats, such as Boolean expression, binary decision diagram (BDD), truth table, BLIF and Spice netlist. These formats are used as inputs to generate the switch networks and logic gates.
- *Transistor network generation*: responsible to generate transistor networks with different arrangements and electrical behaviors. Besides the switch networks generated through the inputs formats, this module can generate the complementary switch networks considering the input switch network.
- *Transistor network profile and estimation*: implements a set of estimation methods, considering the electrical behavior of switch networks. Among the estimations methods implemented are Elmore delay module, Dynamic power consumption, Static power consumption and Area estimation.
- *Transistor network view*: this module implements some options to visualize different switch networks and logic gates generated by framework.

The features implemented in this work are inserted in the *Data input* and *Transistor network profile and estimation* modules. In this context, the *Data input* module consists of an independent tool called Logic2Logic. This tool is used to converted logic formats representations, among them are truth table, BDD, BLIF and others [10]. This tool is integrated to the SwitchCraft, expanding the inputs formats used as base for the switch networks and logic gate generations. This tool, also offer a visual representation for each input logic format supported.

Basically, the Logic2Logic converts any input format to a truth table representation, which is converted for any output format defined by user. In this work, we presented two new input and output logic formats implemented in the Logic2Logic tool. The AIG format has a textual and visual representation, whereas the threshold format has only the textual representation. The transistor placement method implemented is described with more detail hereafter.

IV. TRANSISTOR PLACEMENT METHOD

The Euler path is a convencional algorithm of placement used to reduce the number of diffusion gap during the layout generation. The goal of algorithm is find the same uninterrupted path of transistors, called Euler path, in the *pull-up* plan and in the *pull-down* plan of a logic gate [6]. On the other hand, when it is not possible to find one single Euler path, the algorithm returns two or more sub-Euler paths, which are seperated by a diffusion gap. However, the diffusion gaps increase the number of contacts used to route the circuit, and consequently the congestion of wire.

In order to reduce the number of difussion gap, in cases where there is not a single Euler path, one pre-analysis can be applied. Such analysis consist of one reordering of the serial associations of transistors or sub-networks. Since the Boolean function represented by a given switch network is not changed by reordering of series associations. Besides, in some cases, such reordering can defines the occorence of an Euler path. An example is presented in Fig.4, where the Euler path is found after the reodering of the switch network. In Fig.4(a) it is possible to seen the switch network before the reordering, whereas Fig.4(b) presents the topology obtained after the reordering, which contens a single Euler.

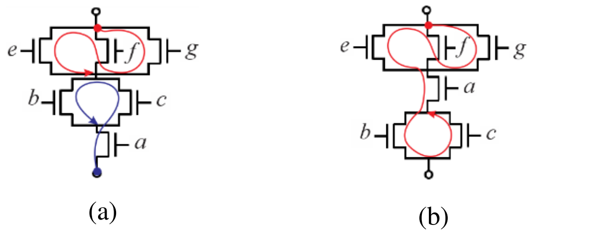

Fig. 4 Reordering of serial associations of switch: (a) switch network before the reordering and (b) switch network after one.

In the transistor placement method a SP switch network and its dual network are used as inputs. In the current implementation, our method cannot be applied in the NSP switch networks. In order to identify all combinations between the serial associations, the algorithm interchange the serial associations of each switch network. Each new combination represents a new switch network. In sequence, the Euler path algorithm is applied for all switch networks generated. A matching checker is used to verify if a given Euler path, found in a SP switch network, can be also found in its dual network. In positive case, theses switch networks are stored in a list, which is used in the next step of the method to find the logic gate that presents the smaller wire congestion. Otherwise, gaps are inserted and the results networks are submitted to the permutation step again, as it is presented in the Fig.5.

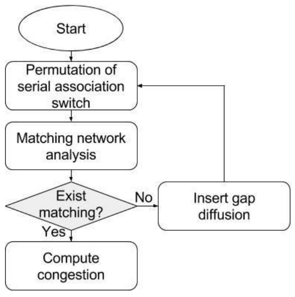
Start
Permutation of serial association switch
Matching network analysis
Exist matching?
No
Insert gap diffusion
Yes
Compute congestion

Fig. 5 flowchart that describes the propose method.

In the method implemented each switch network is represented by a multi-graph. A multi-graph G consists in a triple of nodes (vertices) set V(G), an edges set and the relation between edges and vertices [7]. In this work, each edge connects two distinct vertices. The nodes of the switch network are mapped as V(G), whereas, the transistors (or switches) are equivalent to the edges. Moreover, an *up-graph* is used to represents a given switch network and its dual network is represented by a *down-graph*.

The first step of our method interchange the serial association in both graphs, *up-graph* and *down-graph*. In this context, edges or sub-graphs in serial associations are interchanged. For each *up-graph* permutation, the Euler path algorithm is applied. All Euler paths found are stored and used as input for the next step of the method. Whether, through a given Euler path obtained from *up-graph*, it is possible to travel on *down-graph* visiting all its edges, such Euler path may be also found in the *down-graph*. This situation defines the matching between networks.

This way, the matching between *up-graph* and the *down-graph* is used to define the wire congestion value for a specific gate configuration. The congestion value is computed considering the canal routing approach [11]. Thus, the number of tracks and contacts used in the routing of each network defines the congestion value. However, depending of the Euler path adopted some tracks may be shared, decreasing the congestion value.

If a matching between the switch networks is not possible, it is applied a method to insert fake edges in one or both graphs. The number of fake edges inserted is minimum, since the Eulerian algorithm is applied. Thus, the fake edges are added in the graph until only two vertices in the graph take odd degree. Each new fake edge represents a gap diffusion in the transistor placement. After the insertion of fake edges, the modified switch networks return to the permutation of series step again.

V. RESULT AND IMPLEMENTAIONS

A. *AIG format*

The textual format used to represent the AIG implementation was the AIGER format [12]. In this format, an integer number, multiple of two, represents each input, output and AND2 gate from AIG structure. The inverter gate does not have an explicit representation. By this reason, the polarity of a given signal (input, output or AND2 gate output) is defined by number that represents such signal. Therefore, whether this number is an even number the polarity of the signal is positive. Otherwise, the polarity is negative. An example of this format is shown in Fig.6, where the Boolean function $F = (x1+x2+(x3*x4))$ is described. It is important to notice that the first line of the textual AIG format represents its header, where the name of AIG is defined, followed by number of components, inputs, latches, outputs and AND gates. A graphic representation for such Boolean function is presented in Fig.3, which was obtained through AIG viewer implemented.

```
#header
    AIG_name 7 4 0 1 3
#Declaration of inputs:
    2                # x1
    4                # x2
    6                # x3
    8                # x4
#Declaration of outputs:
    11               # !((! x1*! x2)*!( x3* x4))
#Declaration of AND gates:
    10 12 15         # (! x1*! x2)*!( x3* x4)
    12 3 5           # ! x1*! x2
    14 6 8           # x3* x4
```

Fig. 6 Textual AIG format description

B. *Threshold format*

The textual format used to represent a threshold logic gate (TLG), consists in a sequence of inputs variables and its respective weight values, besides the threshold value. Such format is represented as in the follow, considering the threshold function $F = (x1+x2+(x3*x4))$:

$$\mathrm{T[x1(2),x2(2),x3(1),x4(1);2]} \qquad (2)$$

Depending of the situation, the threshold logic format can be used as input or output. This way, the convectional logic formats supported by Logic2Logic can be converted for threshold logic format, as well as, a given threshold logic function can be converted to conventional formats supported by Logic2Logic. As we adopted the threshold identifier proposed in [9], the threshold logic converter implemented herein, cannot be used to non-threshold logic functions, since such functions results in a TLG network, as discussed in Section 2.

C. *Transistor placement method*

In order to evaluate the method proposed in the Section 3, we used a specific set of Boolean functions obtained from the library Genlib [13]. In total we selected one hundred Boolean functions, from five up to ten input variables. A logic gate has been built for each logic function, and hence, its respective *pull-graph* and *down-graph*. Two wire congestion estimations have been performed, one for the logic gate disregarding to serial reordering, and other regarding the serial reordering. The wire congestion for a logic gate is estimated through the number of contacts (#C) and the number of tracks to routing (#T). In this context, the wire congestion value is computed by product between the #C and #T, as it is shown in Table 1. Also a percentage value is presented in Table 1, such value represents the improvement in ration to the wire congestion (T*C) when the reordered network is compered with the original network.

TABLE I. LOGIC GATES AND WIRE CONGESTION ESTIMATION

Boolean function	**Original gate**			**Reordered gate**			%
	#T	#C	T*C	#T	#C	T*C	
(a*(b+c*(d+e)))	5	16	80	4	12	48	60
(a*(b+c*(d+e*f)))	6	18	108	4	14	56	52
(a*(b+c*(d+e*(f+g))))	7	20	140	4	16	64	46
(a*(b+c*(d*e+f*(g+h))))	7	22	154	6	18	108	70
(a*(b+c*(d+e+f)))	5	14	70	4	14	56	80
(a*(b+(c+d*(e+f))*(g+h+i)))	7	24	168	5	20	100	60
(a*(b+(c*d+e*(f+g))*(h+i+j)))	7	26	182	6	22	132	73
(a*(b+c*d*(e+f)))	5	18	90	4	14	56	62
(a*(b+c*d*(e+f+g)))	5	16	80	4	16	64	80
(a*(b+c*(d+e)*(f+g+h)))	5	22	110	5	18	90	82
(a*(b*c+d*(e+f)))	5	18	90	5	14	70	78
(a*(b*c+d*(e+f*g)))	7	20	140	5	16	80	57
(a*(b*c+d*e*(f+g)))	5	20	100	5	16	80	80
(a*(b*c+d*(e+f)*(g+h)))	6	22	132	5	18	90	68
(a*(b*c+d*(e+f)*(g+h+i)))	6	24	144	5	20	100	69
(a*(b*c+d*(e+f+g)*(h+i+j)))	6	22	132	5	22	110	83
(a*(b*(c+d)+e*f*g))	5	20	100	4	16	64	64

As it is possible to notice, due to limited space, only some Boolean functions are presented in Table 1. However, for the functions presented in Table 1, the wire congestion estimation, for the logic gate built after the reordering, presents better results than original logic gate. In some cases, when the number of networks generated using method presented in this paper is higher than 3000 networks, the runtime is infeasible.

The most significant results, in Table 1, are obtained in cases where the original logic gates did not present gap diffusion. In this cases, after the serial reordering, the gap diffusions are avoided due to Euler path occurrence, impacting in the reduction of the number of contacts and of tracks used in the routing. Inclusive, logic gates that initially presented Euler path may be improved, because through the serial reordering, a small and better Euler path can be found to decreasing the number of tracks used in the gate routing.

CONCLUSIONS

The features implemented in this work extend the set of tools originally available by SwitchCraft framework, offering two new input and outputs formats, besides a transistor placement method. As a future work, we intend implements a method for dual-switch network generation, due to the great importance of such method in the generation of CMOS logic gates.

REFERENCES

[1] G. Suto, " Rule agnostic routing by using design fabrics," Design Automation Conference (DAC), pp. 471 - 475, 3-7 June 2012.

[2] Ning, P., Wang, F., and Ngo, K. D, "Automatic layout design for power module," IEEE Transactions on Power Electronics, vol. 28, nº 1, pp. 481 - 487, 2013.

[3] Martins, R. P., Lourenco, N., and Horta, N., " LAYGEN II—automatic layout generation of analog integrated circuits," IEEE Transactions Computer-Aided Design of Integrated Circuits and Systems, vol. 32, nº 11, pp. 1641-1654, 2013.

[4] McGeer, P. C., Sanghavi, J. V., Brayton, R. K., and Sangiovanni-Vicentelli, A, "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions," Very Large Scale Integration (VLSI) Systems, vol. 1, nº 4, pp. 432-440, 1993.

[5] Brayton, R., & Mishchenko, A., ABC: An academic industrial-strength verification tool, Berlin Heidelberg: Springer , 2010.

[6] Ziesemer, A., Reis, R., Moreira, M. T., Arendt, M. E., and Calazans, N. L., "Automatic layout synthesis with ASTRAN applied to asynchronous cells," Circuits and Systems (LASCAS), pp. 1-4, 2014.

[7] Callegaro, V., Marques, F. D. S., and Klock, C. E, "SwitchCraft: a framework for transistor network design," 23rd symposium on Integrated circuits and system design, pp. 49-53, 2010.

[8] Possani, V. N., et al., "Graph-Based Transistor Network Generation Method for Supergate Design," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, nº 2, pp. 692 - 705, 2015.

[9] Neutzling, A., Martins, M. G., Ribas, R. P., and Reis, A. I.. Neutzling, "A constructive approach for threshold logic circuit synthesis," Circuits and Systems (ISCAS), pp. 385 - 388, 2014.

[10] Logic2Logic. http://www.inf.ufrgs.br/logics/downloads. Last acessed in 04/06/2016.

[11] S. H. Gerez "Algorithms for VLSI design automation", New York, Wiley.

[12] AIGER format. http://fmv.jku.at/aiger/. Last accessed in 04/06/2016.

[13] Genlib. https://embedded.eecs.berkeley.edu/pubs/downloads. Last accessed in 04/06/2016.