Generating New Benchmark Designs using a Multi-terminal Net Model

Dirk Stroobandt* Jo Depreitere† Jan Van Campenhout

Department of Electronics and Information Systems
University of Ghent
St.-Pietersnieuwstraat 41, B-9000 Gent, Belgium
phone: +32 9 264.34.01; fax: +32 9 264.35.94
E-mail: {dstr, jdp, jvc}@elis.rug.ac.be

Abstract

For the development and evaluation of CAD-tools for the layout, placement, and routing of digital designs and for the evaluation of new computer hardware, a huge amount of benchmark circuits is required. Observing the lack of enough real benchmark designs for use in evaluation tools, one could consider to actually generate such benchmarks. In that case, it is very important that those synthetic benchmarks have the same characteristics as real designs. This paper describes and evaluates a new benchmark generation procedure that produces benchmarks with characteristics, similar to those of real benchmark designs. It will be shown that, in this respect, our new technique outperforms an existing method, presented by Darnauer and Dai [1].

Keywords: Benchmark generation, Net degree distribution, Rent's rule.

1 Introduction

The production of VLSI and ULSI computer chips requires the layout (placement and routing) of the chip design on a carrier. With the advent of high level description languages such as VHDL, with the extensive use of component libraries, and with the standardization of production parameters, more and more steps in the design cycle are being automated. In the early days of chip design, manually designing a chip was still feasible. Nowadays, computer aided design (CAD) tools are indispensable to cope with the complexity and the limited time resources.

For the high demands put on system performances these days, CAD tools often lack flexibility. The helping hand of expert system designers is still needed for making important design decisions. Improving the existing CAD tools therefore remains necessary. New algorithms for partitioning, placement, and routing should be evaluated thoroughly and this entails the need for "good" evaluation tools. Crucial to this evaluation is the use of a large set of benchmarks. This set of benchmarks should consist of a sample of the designs for which the CAD tool is aimed. A thorough evaluation therefore requires a very large set of very different benchmark designs.

Also, for the evaluation of new computer architectures, it is important to be able to predict parameters such as interconnection lengths, clock speed, area occupancy etc. The accuracy of these estimates has to be evaluated to gain confidence in the predictions. Generally, "typical" benchmark circuits are implemented in the target architecture and the parameters measured and then compared to the predictions. An accurate evaluation thus requires, again, a large and reliable set of benchmark designs.

The set of benchmarks used in the research community today is fairly small. Moreover, these benchmarks are often too small to be useful for the complex tools we want to evaluate today. New sets of benchmarks are definitely needed for evaluating new tools that are developed in several research groups [2]. The lack of good benchmarks can be overcome if we are able to generate them fast and in a controllable manner. Ghosh et al. [3] suggest a benchmark generation technique by searching for 'mutants' of

*The author is Research Assistant with the Fund for Scientific Research of Flanders, Belgium.

†Supported by an Inter University Attraction Poles research program (IUAP IV-13) on Photonic Interconnects initiated by the Belgian government.

existing designs with the same 'signature'. A similar method, based on the 'cloning' of designs, has been presented by Hutton et al. [4]. Both methods still use existing benchmark designs as a starting point to generate (a lot of) designs from the same equivalence class. Circuits with other characteristics can only be generated by starting from an existing benchmark with those characteristics. The possibility of generating completely new benchmark circuits (with viable, but not necessarily existing characteristics) remains desirable.

The major problem of synthetic benchmark generation is the requirement that the obtained benchmarks are good representatives of real designs, i.e. circuits that could be the result of real design processes. It is commonly acknowledged that a design can be characterized by its interconnection complexity. The so called Rent exponent of the design [5] is a measure for this interconnection complexity. It results from Rent's rule, which is a relationship between the number of pins in a partitioned design and the number of blocks per partition. Real circuits have been shown to follow Rent's rule, so controlling the Rent exponent of theoretical benchmarks is a necessity. Another important parameter in designs is the average net degree. It has been observed that more than 75% of the nets in real designs are 2- and 3-terminal nets [6]. Moreover, Stroobandt and Kurdahi [7] showed that the distribution of net degrees follows a power law. Since this distribution can be observed in real designs, generated benchmarks should have the same net degree distribution.

A first attempt to generate random benchmark circuits has been presented by Darnauer and Dai [1]. Their program, called `rmc`, generates large random circuits with a specified number of inputs, outputs, blocks, terminals per cell, and Rent exponent. However, the Rent exponent is treated as a target value that the program aims for and the generated benchmarks only follow Rent's rule on average. An even more serious drawback is to be found in the net degree distribution. This distribution is not 'controlled' by `rmc` and therefore it does not follow a power law. Generally, there is a shortage of 2-terminal nets and an excess of multi-terminal nets with high net degrees. The generated designs also have a very narrow terminals-per-block distribution. Finally, the method of Darnauer and Dai is restricted to generating circuits for FPGAs containing k-input LUTs (LookUp Tables) with only one output [1].

In this paper, we will present a new benchmark generation program `gnl` (Generate NetList) that does not have the deficiencies described above. Our program is based on a thorough examination of the constraints that apply to the possibilities of net connections [8] and on an accurate model for multi-terminal nets in real designs, presented in [7]. The program `gnl` generates benchmarks with a preset Rent exponent and, most importantly, a viable net degree distribution. Not only the number of gates, the number of primary in- and outputs, and the Rent exponent can be predefined, also the terminals-per-block distribution can be specified and a specific deviation from Rent's rule, commonly observed in real designs, is included.

Section 2 will give an overview of the critical parameters that characterize real designs, section 3 describes the basic procedure of generating benchmarks and section 4 will elaborate on the constraints we have to consider. The features of our benchmark generation method will be the subject of section 5 whereas section 6 will show that the designs generated by our generation program `gnl` strongly resemble real benchmark circuits. We will also compare our method to the program `rmc`, described by Darnauer and Dai in [1].

2 Critical parameters of real designs

Partitioning, placement, and routing, as well as parameter extraction (interconnection lengths, area usage, clock speed,...) mainly require what we call "partitioning information": number of blocks in the design, list of terminals per block and their nature (inputs or outputs), netlist, and interconnection structure. We thus need to characterize designs (synthetic or real) on the basis of this information. The most important parameter to capture is the interconnection complexity of the design. This interconnection complexity is reflected in the Rent exponent and the net degree distribution. For a clear understanding, we will start this section with an overview of the basic definitions used and we will continue with a discussion on the Rent exponent and the net degree distribution.

2.1 Definitions

A design can be represented by a set of interconnected blocks as in figure **??** (the blocks can be the representation of transistors, gates, or even whole designs). An interconnection between blocks is called a

net. A net that is connected to more than two blocks is called a *multi-terminal net*. We will assume that a net cannot have more than one connection to the same block.[1] Some of the nets are also connected to the outside of the design. These nets are called *external nets* (as opposed to the *internal nets* which only connect blocks within the design). In order to model these external nets properly, we introduce a new kind of block which we will call a *pin*. The other blocks will be called *logic blocks*. Every external net will be connected to exactly one pin. Note that the number of pins thus equals the number of external nets. The *net degree* of a (multi-terminal) net will be defined as the number of blocks (logic blocks and pins) the net is connected to. A *net degree distribution* is a collection of values, indicating, for each net degree n, how many nets have a net degree equalling n.

Partitioning a design means dividing this design into disjoint sub-designs (called *modules*), each containing a subset of the blocks (figure **??**). This partitioning is done using some kind of criterion. Generally, the criterion is to minimize the number of nets cut, i.e. the number of nets crossing the borders of modules in the partition. Nets that are cut by module boundaries are shared between two or more modules and are said to be external to the modules. Therefore, the net will be split into a number of sub-nets, one for each module that shares the net. A new pin will be assigned to each sub-net (if the net was already external to the design then the pin assigned to it can be reused for one of the sub-nets). Each module can then itself be seen as a design and can be partitioned further. A partitioning process where the modules themselves are recursively partitioned will be called a *hierarchical partitioning method*.

2.2 Rent's rule as a measure of interconnection complexity

Designs can be classified on the basis of their interconnection complexity. This *interconnection complexity* of a design is based on the notion that some designs have a totally different structure of interconnections than others. These differences have been experimentally observed by Rent and his observations led to the well-known Rent's rule [5]. This rule is a relationship between the average number of elementary blocks B in the modules of a partitioned design, and the average number of the module's external connections (pins) P:

$$P = T_b B^r, \tag{1}$$

where T_b is the average number of terminals per logic block, and r is called the *Rent exponent*. This exponent is a measure of the interconnection complexity of the design. Its value is bounded by 0 and 1, with increasing values for increasing interconnection complexity. Generally, r ranges from 0.47 for regular designs (such as Random Access Memories), up to 0.75 for complex designs (such as fast full custom VLSI designs) [9]. The validity of Rent's rule is a result of the fact that designers tend to build their designs hierarchically, imposing the same complexity at each level of hierarchy. This leads to the observed "self-similarity" of designs. Rent's rule can be observed in figure **??** for the ISCAS89 benchmark design 's953' [10]. The data from the benchmark are obtained using the 'ratiocut' partitioning program described in [11]. The deviation of Rent's rule from the data, observed for high values of P and B, is known as the second region in Rent's rule [5, 12]. At the highest levels of the hierarchical partitioning method, the number of pins is lower than predicted by Rent's rule due to the fact that designers have to deal with the pin limitation problem in today's chips.

2.3 The net degree distribution

Another important parameter in characterizing designs through their interconnection structure is the net degree of the nets. It has been observed earlier that more than 75% of the nets in real designs are 2- and 3-terminal nets [6]. A more elaborate study on multi-terminal nets revealed that the distribution of net degrees generally follows a power law [7].[2] This power law distribution results from an accurate model of the behaviour of multi-terminal nets during the partitioning process. The model has been validated with benchmark data from the ISCAS benchmark set (e.g. figure **??**).

In addition to the requirement that benchmarks should obey Rent's rule, the power law net degree distribution should be found as well. Any benchmark, generated by a program, should have a net degree distribution that follows the power law distribution in order to be a valid sample of real benchmark designs.

[1] Otherwise, the two (or more) connected terminals can be grouped and treated as one terminal.

[2] In fact, it converges to a power law distribution for very large designs and it follows a power law in the region of small net degrees.

3 Benchmark generation

Since it is the interconnection structure that characterizes designs, it is of the utmost importance that the connections between blocks in the design are chosen with great care. Therefore, the critical part of any benchmark generation program is the process of generating the netlist. We will call this process the *net generation process*. Note that the net generation process (where different net parts are connected) is the reverse of the partitioning process described in section 2 (where nets are cut instead of combined). The net generation process will be explained using figure **??**. Consider two modules `a` and `b` that are part of a certain partition of the benchmark design. We shall denote the number of logic blocks contained in those modules as B_a and B_b respectively. The module that is formed by combining modules `a` and `b`, the *cluster module* `c`, then contains $B_c = B_a + B_b$ logic blocks. The number of inputs and outputs of module `a` is denoted as I_a and O_a. For module `b` we have I_b and O_b and for the cluster module `c` I_c and O_c. Observability and controllability of every part of the design require the number of inputs and outputs to be at least 1 [13].

Basically, there are two types of connections possible between the modules `a` and `b`. The first type connects an output of one module with an input of the other module and does not leave the cluster module. We will therefore call these *internal connections*. They are represented by a dashed line in figure **??**. The other connections, the *external connections*, connect a pin (input or output) of one module with an input of the other module and leave the cluster module through a pin. We do not allow connections between two pins of the same module since this type of connection can be made within the module itself (at another hierarchical level). The pins of the modules `a` and `b` that are not connected through an internal or external connection are routed directly to a pin of the cluster module. Depending on whether the connections are driven from outputs of modules `a` or `b` or from inputs of the cluster module, we distinguish between the subtypes as described in table 1 (see also figure **??**).

Our basic procedure for benchmark generation is a bottom-up combination of logic blocks and can be described as follows:

1. All logic blocks in the design are generated and given a number of input and output terminals. The number of logic blocks and the number of inputs and outputs per logic block are specified by the user.
2. The logic blocks are paired and connections are made between their terminals. This results in a cluster of blocks with a number of input and output terminals.
3. The clusters themselves are recursively paired further with other clusters until all clusters are combined to one design.

Of course, the connections made in step 2 of the generation process have to satisfy certain constraints in order to lead to a feasible benchmark design. These constraints will be the subject of the next section.

4 The constraints that have to be satisfied

In step 2 of the generation process, the following equations must be satisfied (this can be obtained immediately from figure **??**)

$$O_a = V_{o,a} + S_{i,a} + S_{e,a} \tag{2}$$
$$O_b = V_{o,b} + S_{i,b} + S_{e,b} \tag{3}$$
$$I_a = V_{i,a} + S_{i,b} + S_{e,b} + S_{e,i} \tag{4}$$
$$I_b = V_{i,b} + S_{i,a} + S_{e,a} + S_{e,i} \tag{5}$$
$$O_c = V_{o,a} + V_{o,b} + S_{e,a} + S_{e,b} \tag{6}$$
$$I_c = V_{i,a} + V_{i,b} + S_{e,i} \tag{7}$$

The parameters I_a, I_b, O_a and O_b are known from earlier clustering. All other parameters must be positive and the number of cluster inputs and outputs must at least be equal to 1 for controllability and observability reasons [13]. This leads to a number of constraints. Furthermore, the designs must comply with the demand of a similar interconnection complexity as can be found in real designs. Therefore, the number of pins for the cluster module is defined by Rent's rule

$$I_c + O_c = P_c = T_b B_c^r, \tag{8}$$

with T_b the average number of terminals per logic block. For designs where the number of pins should be bounded, we can also introduce Rent's second region.

The next requirement we should meet is the power law net degree distribution. In [7], Stroobandt and Kurdahi showed that a ratio f of the number of internal connections to the total number of connections,

$$f = \frac{S_i}{S_i + S_e} \tag{9}$$

with

$$S_i = S_{i,a} + S_{i,b} \tag{10}$$
$$S_e = S_{e,a} + S_{e,b} + S_{e,i}, \tag{11}$$

that is a constant for all clusters, leads to a power law net degree distribution.

This *fraction* f ranges from 0 to 1. It is a parameter of the design (just as the Rent exponent) and is defined by (for details, see [8])

$$f = \frac{GT_o - O}{GT_i - I}, \tag{12}$$

with T_i and T_o the average number of inputs and outputs per logic block, I and O the total number of primary inputs and outputs of the design, and G the total number of logic blocks in the design.

The constraints lead to (see [8])

$$S_i = \frac{f}{1+f}\left(P_a + P_b - P_c\right), \tag{13}$$

with $P_a = I_a + O_a$ and $P_b = I_b + O_b$. The number of internal connections is thus dictated by the power law net degree distribution requirement through the fraction f.

The choice of P_c and S_i, as directed by Rent's rule and the power law net degree distribution requirement, leaves only three degrees of freedom (with constraints). Once $S_{i,a}$ is chosen, $S_{i,b}$ is fixed and the choice of both $S_{e,a}$ and $S_{e,b}$ leaves no choice for $S_{e,i}$.

All constraints can be reduced to necessary conditions on the relation between T_i, T_o, f, and r (see [8]). If these conditions are met, a solution is guaranteed that obeys Rent's rule (on all hierarchical levels) and that converges to a power law net degree distribution for large designs. The important thing here is that this guarantee can be given before we even try to generate the design. This way, we can prevent loss of time due to trying to generate a design and then seeing that at a certain level some constraint is violated. In [8], we also present an extension to the conditions in the presence of Rent's second region.

5 Our new benchmark generation method

5.1 Basic procedure

We have developed a program called `gnl` – Generate NetList – that generates benchmark designs based on the net generation process described in section 3, taking in account the constraints as set in section 4. The main input parameters to be specified by the user are G, r, T_i, T_o, I, O, and B_b (the number of logic blocks that defines the boundary for the second region in Rent's rule).[3]

These input parameters are checked against the constraint that the fraction f lies within the range $[0, 1]$, based on equation 12. If the constraint is violated, the possible range for the number of in- and outputs is returned to the user.

The program then follows the following basic steps

1. All logic blocks are generated and assigned T_i inputs and T_o outputs.[4] Each logic block forms a cluster containing only one block.

2. The clusters are pairwise combined to a new cluster and the number of pins in the new cluster is set according to Rent's rule (first or second region respectively). The benchmark will thus follow Rent's rule by construction.

[3] Alternatively, a Rent exponent for the second region can be given.
[4] An extension is described in subsection 5.2.

3. The number of internal connections to be used is set according to equation 13.

4. The number of connections for all possible types ($S_{i,a}$, $S_{i,b}$, $S_{e,a}$, $S_{e,b}$, and $S_{e,i}$) is chosen randomly out of all possible choices that satisfy the constraints.

5. The net generation process is repeated from step 2 until all clusters are combined into one design.

The choice of P_c in step 2 is checked against the constraints. Since all necessary conditions on the input parameters are checked at the start of the generation process, these conditions can only pose problems due to the rounding of real numbers to integers.

Our benchmark generation program `gnl` outputs two files: one with a netlist of the final generated benchmark and one with the obtained net degree distribution.

5.2 Extending the class of benchmark designs that can be generated

The pairwise combination of clusters can be performed level by level by first combining all logic blocks into clusters of size two, then combining these clusters to clusters of size four, and so on. This produces regular designs in the sense that a partitioning program such as 'ratiocut' [11] will partition the design almost exactly into the clusters we generated (at least for big enough clusters). A parameter to be set by the user can make the program `gnl` choose the next two clusters for combination randomly without constraints to their relative sizes. This produces designs that are less regular but still obey Rent's rule well. The combination of modules with different sizes produces no problems whatsoever for the constraints since these constraints are deducted for an arbitrary number of logic blocks per module.

As stated before, Rent's second region can be easily imposed. It is sufficient to split the procedure in two parts. The first part remains the same as before. In Rent's second region, the net generating process is to be seen as a new generating process starting from modules corresponding with the clusters found from the previous clustering in the normal region. But, this time, a new Rent exponent is used (and thus a different further path for the P versus B curve).

One problem with the benchmark generation procedure described above is that all logic blocks are given the average number of terminals. This is one of the major shortcomings of the program `rmc` developed by Darnauer and Dai [1] too. However, since we generate the benchmarks by a bottom-up approach (as opposed to the top-down approach used in [1]), we can easily solve this problem. In fact, `gnl` enables the user to fully define the distribution of the number of terminals per logic block.

With the extensions of taking into account the net degree distribution, the inclusion of Rent's second region, and the possibility to let the user fully define the terminals-per-block distribution, our benchmark generation program does not have the shortcomings found in `rmc`.

6 Results and comparison

6.1 Properties of the designs generated by `gnl`

The deduction of all constraints guarantees that our benchmark generation program will find a solution that follows both Rent's rule and the power law net degree distribution.

In order to check that our benchmark generation method produces designs with properties comparable to those of real benchmarks, we generate synthetic benchmarks, based on the parameters of real benchmarks and compare the results. In figure **??**, the Rent behaviour is compared for the ISCAS85 benchmark 'c3540nr' [14] and its generated counterpart. The distribution of terminals per logic block has been chosen exactly as in the real benchmark. The figure shows the Rent behaviour of the original benchmark circuit after partitioning with '`ratiocut`', the Rent behaviour of the synthetic benchmark after partitioning in the modules as generated, and the Rent behaviour of the synthetic benchmark after partitioning with '`ratiocut`'. Figure **??**(b) clearly shows that the generated benchmark follows Rent's rule perfectly (not taking the discretization effect into account).[5] A partitioning of the generated design by '`ratiocut`' still gives an acceptable result (figure **??**(c)) but the partitioning program did not find the optimal solution which results in a behaviour of seemingly higher complexity. This is probably also the case in real designs. The scaling behaviour clearly remains visible, a sign of the fact that the Rent-behaviour of the generated

[5] Only at the lowest levels, there is some small deviation from the desired Rent behaviour due to the choice of an excessive number of terminals for some logic blocks. More details can be found in [8].

design is visible in all parts of the design and is not a mere consequence of an imposed behaviour in some discrete points.

The net degree distribution of the original benchmark circuit 'c3540nr' is compared to that of the generated benchmark design in figure **??**. Both distributions follow the same path but the variations are larger in the real design. Especially the good resemblance for two- and three-terminal nets is important because more than 75% of all nets in a general design has net degree 2 or 3 [6].

6.2 Comparison between `gnl` and `rmc`

Next, we compare our benchmark generation method with `rmc`, the method of Darnauer and Dai [1]. As we already stated in the introduction, `rmc` does not guarantee that the generated design perfectly follows the Rent behaviour. The Rent exponent provided by the user is used to choose the number of pins as good as possible, but there is no guarantee that this number can be actually reached. This can be seen in figure **??** for a benchmark design generated on the basis of the parameters of the benchmark 'industry3' [15]. In this figure, the Rent behaviour is shown for a partitioning in the modules as they were generated by the programs `gnl` and `rmc` respectively. The deviation from the imposed (normal) Rent behaviour in `rmc` (all be it a small one) is the result of an excess of degrees of freedom. The generation programs have to choose what sort of connection they are going to use at certain moments. With `rmc`, choices that exclude the desired solution are possible. Because we derived all boundary conditions and all ranges for the input parameters that lead to a viable solution, `gnl` does not have the same problem.[6]

Rent's second region can also be fully defined in `gnl`. The user can fix the Rent exponent of the second region or the boundary between the two regions. The program `rmc` tries to reach the normal Rent region as fast as possible, regardless of the input parameters and only dependent on the random choices that are made to layout the nets. This can also be clearly seen in figure **??**.

With respect to the Rent behaviour, both methods are doing well, if we do not consider Rent's second region. The major difference between both methods is to be found in the net degree distribution. For designs with only a few terminals per logic block, the net degree distributions generated by `gnl` and `rmc` are still comparable. However, when dealing with designs with a higher average number of terminals per logic block (the benchmark design 'industry3' has 4.4 terminals per logic block on average), only `gnl` still produces a viable net degree distribution. This can be clearly seen in figure **??**. The benchmark design generated by `gnl` has a power law net degree distribution (especially in the important region: at low net degrees) and follows the real net degree distribution very well. The benchmark design generated by `rmc` creates too many nets with higher net degrees and, much more important, a shortage of two-terminal nets. The three-terminal nets even outnumber the two-terminal nets.

Another difference between both benchmark generation methods is the fact that, in `gnl`, the user can impose the distribution of terminals per logic block. The choice of this distribution is free because our method generates designs in a bottom-up fashion, i.e. by starting from the individual logic blocks and clustering these together. The method presented by Darnauer and Dai is based on a partitioning and thus follows the reverse path, from one cluster down to the individual logic blocks. The number of terminals per logic block then depends on the choices for connections made at higher levels. In practice, `rmc` seems to always find the same number of terminals for all logic blocks (only the two integer values closest to the average) [1]. Moreover, `rmc` has been designed to consider only logic blocks with exactly one output. Real designs can also contain blocks with multiple outputs, especially if we want to generate designs at another level than the gate level. As a matter of fact, we explicitly use blocks with multiple outputs to easily handle Rent's second region.

The results show that benchmark designs, generated by `gnl`, based on the generation method described above, follow Rent's rule by construction. The method also guarantees a monotonic decreasing net degree distribution that, at least for designs that are large enough, approximately converges to a power law, as can be seen in real designs. Especially this last property is a huge improvement of the existing method, presented by Darnauer and Dai [1] which is, as far as we know, the only other method able to generate benchmark designs from scratch, without having to use existing benchmarks. In this respect, both `gnl` and `rmc` differ from other generation methods that search for 'mutants' or 'clones' of existing benchmarks.

In our generation method, the degrees of freedom are very large without losing a guaranteed similar behaviour as in real designs. A very interesting property of our generation method is the ability to vary

[6] The only deviation from the imposed Rent behaviour is situated in the range of the small modules and is due to too large a difference in the number of terminals of different logic blocks. This can not occur in `rmc` since the distribution of the number of terminals per logic block always is very small (and not adjustable by the user).

important parameters, such as the Rent exponent r and the fraction f, in a controlled manner. This way, we can examine the dependence of CAD algorithms or new computer architectures on only one parameter, keeping the other parameters constant. This can yield very interesting results and would allow us to draw well-founded conclusions from experimental results.

7 Conclusion

The possibility of having huge amounts of benchmark data is mandatory for extensively evaluating CAD estimation tools (for interconnection length, area occupancy, clock frequency, ...) and new computer architectures. The advent of benchmark generation processes can help the research community to make a leap to better CAD tools resulting in designs with higher performance.

We developed a generation procedure that generates synthetic benchmarks that strongly resemble real designs. This is obtained by carefully observing the constraints that have to be satisfied and the inherent interconnection structure of designs. The interconnection structure of both real designs and our synthetic benchmarks obey Rent's rule and a power law net degree distribution. Since these are the most important parameters that characterize designs, we are able to produce benchmarks that are viable representatives for real benchmark circuits.

References

[1] J. Darnauer and W.W. Dai. A method for generating random circuits and its application to routability measurement. In *Proc. 1996 ACM/SIGDA Intl. Symp. Field Programmable Gate Arrays*, pages 66–72, February 1996.

[2] Computer-Aided Design Benchmarking Laboratory. http://www.cbl.ncsu.edu/benchmarks/.

[3] D. Ghosh, N. Kapur, J. Harlow III, and F. Brglez. Synthesis of wiring signature-invariant equivalence class circuit mutants and applications to benchmarking. In *Proceedings of the Design, Automation end Test in Europe Conference*, pages 656–663. IEEE Computer Society, February 1998.

[4] M. Hutton, J. Rose, and D. Corneil. Generation of synthetic sequential benchmark circuits. In *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, pages 149–155, 1997.

[5] B. S. Landman and R. L. Russo. On a pin versus block relationship for partitions of logic graphs. *IEEE Trans. on Comput.*, vol. C–20: pages 1469–1479, 1971.

[6] E. S. Kuh and T. Ohtsuki. Recent advances in VLSI layout. *Proc. of the IEEE*, vol. 78: pages 237–263, 1990.

[7] D. Stroobandt and F. J. Kurdahi. On the characterization of multi-point nets in electronic designs. In M. A. Bayoumi and G. Jullien, editors, *Proc. of the 8th Great Lakes Symposium on VLSI*, pages 344–350. IEEE Computer Society Press, February 1998.

[8] D. Stroobandt. Generating new benchmark designs for evaluation of CAD tools and new computer architectures. Technical report, University of Ghent, April 1998. ELIS Tech. rep. DG 98-05.

[9] R. L. Russo. On the tradeoff between logic performance and circuit-to-pin ratio for LSI. *IEEE Trans. Comput.*, vol. C–21: pages 147–153, 1972.

[10] F. Brglez, D. Bryan, and K. Kozminski. ISCAS'89 benchmarks, May 1989. Distributed on tape to participants of the Special Session on Sequential Test Generation, Intl. Symp. on Circuits and Systems; partially characterized in F. Brglez, D. Bryan, K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits", Proc. IEEE Intl. Symp. on Circuits and Systems, pages 1929–1934.

[11] Y.-C. Wei and C.-K. Cheng. Ratio cut partitioning for hierarchical designs. *IEEE Trans. Comput.-Aided Des., Integrated Circuits & Syst.*, vol. 10 (no. 7): pages 911–921, July 1991.

[12] H. Van Marck, D. Stroobandt, and J. Van Campenhout. Towards an extension of Rent's rule for describing local variations in interconnection complexity. In S. Bai, J. Fan, and X. Li, editors, *Proc. 4th Intl. Conf. for Young Computer Scientists*, pages 136–141. Peking University Press, 1995.

[13] D. Stroobandt and J. Van Campenhout. Hierarchical test generation with built-in fault diagnosis. In M. E. Kavanaugh, editor, *Proc. 5th Asian Test Symp.*, pages 22–28. IEEE Computer Society Press, November 1996.

[14] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran, June 1985. Distributed on tape to participants of the Special Session on ATPG and Fault Simulation, Intl. Symp. on Circuits and Systems; partially characterized in F. Brglez, P. Pownall, R. Hum, "Accelerated ATPG and Fault Grading via Testability Analysis", Proc. IEEE Intl. Symp. on Circuits and Systems, pages 695–698.

[15] C. Alpert. http://vlsicad.cs.ucla.edu/~cheese/benchmarks.html.

Type of connection	Number
Internal connections driven by module `a`	$S_{i,a}$
Internal connections driven by module `b`	$S_{i,b}$
External connections driven by module `a`	$S_{e,a}$
External connections driven by module `b`	$S_{e,b}$
External connections driven by cluster inputs	$S_{e,i}$
Cluster inputs routed through inputs of module `a`	$V_{i,a}$
Cluster inputs routed through inputs of module `b`	$V_{i,b}$
Outputs of module `a` routed through cluster outputs	$V_{o,a}$
Outputs of module `b` routed through cluster outputs	$V_{o,b}$

Table 1: The different types of connections in the net generation process and their number.

Dirk Stroobandt was born in Gent, Belgium, in 1972. He graduated in 1994 as electro-technical engineer with option in information science at the University of Ghent, Belgium and received a PhD Degree from the same University in 1998. Since October 1994 he is research assistant with the Fund for Scientific Research – Flanders. His research interests include the modelling of three-dimensional opto-electronic systems, estimating interconnection lengths in computer systems, and characterization of parameters for designs. Dirk Stroobandt is a member of K.VIV. and of the IEEE.

Jo Depreitere was born in Brugge, Belgium, on January 5, 1969. He received a degree in electronic engineering from the University of Ghent in 1992. Today, he is working towards a doctoral degree at the same university. His topics of interest are the architectural aspects of opto-electronic multi-FPGAs.

Jan Van Campenhout was born in Vilvoorde, Belgium, on August 9, 1949. He received a Degree in Electro-Mechanical Engineering from the University of Ghent, in 1972; and the MSEE and PhD Degrees from Stanford University, in 1975 and 1978, respectively. Prof. Van Campenhout teaches courses in computer architecture, electronics, and digital design at the Faculty of Applied Sciences of the University of Ghent, Belgium. His current research interests include the study and implementation of various forms of parallelism in computer systems, and their application in programming language support, computer graphics and robotics. He is a Member of Sigma Xi, K.VIV., and IEEE.