Device-Level Placement for Analog Layout: An Opportunity for Non-Slicing Topological Representations

Florin Balasa
University of Illinois at Chicago, Dept. of EECS, Chicago, IL 60607

Abstract– Layout design for analog circuits has historically been a time consuming, error-prone, manual task. Its complexity results not so much from the number of devices, as from the complex interactions among devices or with the operating environment, and also from continuous-valued performance specifications.

This paper addresses the problem of device-level placement for analog layout in a non-traditional way. Different from the traditional approaches – exploring a huge search space with a combinatorial optimization technique, where the cells are represented by means of absolute coordinates, being allowed to illegally overlap during their moves in the chip plane – this paper advocates the use of non-slicing topological representations, like (symmetric-feasible) sequence-pairs, ordered- and binary- trees. Extensive tests, processing industrial analog designs, have shown that using skillfully the symmetry constraints (very typical to analog circuits) to remodel the solution space of the encoding systems, the topological representation techniques can achieve a better computation speed than the traditional approach, while obtaining a similar high quality of the designs.

1 Introduction

In recent years, complete systems that before occupied separate chips are being integrated on a single chip. Examples of such *systems on a chip* (SoC's) include telecommunications IC's, wireless designs – as components in RF receivers and transmitters, and networking interfaces. Although most functions in such integrated systems are implemented with digital circuitry, the analog circuits needed at the interface between the electronic system and the real world are now being integrated on the same die for reasons of cost and performance.

Layout design for analog circuits has historically been a time consuming, error-prone, manual task. If layout for digital IC's is usually regarded as a difficult task mainly because of the scale of the problem (but also complex delay modeling, timing optimization), analog circuits and the analog portions of mixed-signal systems-on-chip are significantly smaller – usually up to 100 devices in a cell, and less than 20,000 devices in a complete subsystem [11]. The task complexity for analog layout results not so much from the sheer number of devices, as from the complex interactions among devices or with the operating environment, and also continuous-valued performance specifications. As most analog circuits are very distinct among themselves, today they are usually still designed and laid out by hand.

This paper addresses an essential problem of analog layout design – the device-level analog placement – in a non-traditional way, using recent results on topological representations for non-slicing floorplans. The traditional way of approaching the analog placement problem is to explore a huge search space with a combinatorial optimization method (for instance, simulated annealing) where the cells are represented by means of absolute coordinates and they can illegally overlap during their moves in the chip plane. However, the topological representations – the main focus of the research plan – are based on a different idea: to define an encoding system as a solution space, each code representing a feasible placement configuration.

While the classic absolute representation approach trades off a larger number of (annealing) moves for easier and quicker-to-build layout configurations which may not be always physically realizable, the idea of using *topological* representations is to trade off more complex (but physically correct!) layout constructions for a smaller number of moves. Our research results show that the skillful use of symmetry constraints (very typical to analog circuits) can remodel the encoding systems, diminishing significantly the solution space size and, therefore, making its exploration more effective in comparison with the classic approaches.

The paper is organized as follows: Section 2 discusses the slicing and non-slicing topological representations of placement configurations; then, Section 3 presents the device-level analog placement problem and discusses several tech-

niques to solve it based on different topological representations; finally, Section 4 gives an overview of the experimental results, and Section 5 presents the basic conclusions of this research.

2 Topological representations

The topological representations of floorplans got recently a renewed attention in the context of block placement. A brief historical overview will introduce many concepts used along this paper.

The block placement problem subject to nonoverlapping constraints, often called *packing*, was proven to be NP-hard even for rectangular blocks [8]. The nature of the problem entailed the use of combinatorial optimization methods (as simulated annealing, genetic algorithms) to explore the solution space of placement configurations.

A combinatorial optimization method can equally operate with two classes of representations of block configurations. The most straightforward and widely used is the *absolute* (or *flat*) representation, where the positions of the blocks are directly specified in terms of coordinates relative to an arbitrary system of axes in the chip plane [2]. However, the convergence of the exploration may be slow due to the huge size of the search space (which contains also infeasible placement configurations, since blocks are allowed to illegally overlap).

An alternative approach to the absolute representation is to define a set of codes – each code representing a placement configuration – as a solution space. An encoding system of feasible placement configurations is usually referred to as a *topological* representation, being an encryption of the positioning (topological) relations between any pair of modules. The first topological representation was derived from the *slicing* floorplan model where the blocks are organized in a set of slices which recursively bisect the layout horizontally and vertically. The direction and nesting of the slices is recorded in a (binary) slicing tree or, equivalently, in a normalized Polish expression (see, e.g., [2]).

The slicing representation limits the set of layout topologies. This can degrade layout density, especially when cells are very different in size, which is often the case in analog layout. Consequently, it was widely acknowledged that slicing placement is not a good choice for high-performance analog design. This is the reason why topological representations were in general disregarded, and the most effective analog placement tools existent so far employed simulated annealing – as optimization engine – operating with *absolute* placement representations [2, 6, 7].

More recently, several novel topological representations, not restricted to slicing floorplan topologies, have been proposed. Murata *et al.* suggested to encode the "left-right" and "up-down" positioning relations between cells using two sequences of cell permutations, named *sequence-pair* [8]. Nakatake *et al.* devised a meta-grid structure (called *bounded-sliceline grid*) without physical dimensions to define orthogonal relations between modules [9]. Guo *et al.* proposed the *O-tree* data structure to reduce the drawback effect of redundancies in the two previous representations [3].

The advent of non-slicing topological representations is likely to have an important impact in device-level placement for analog layout, as the superiority claim – undisputed until recently – of the absolute representation for analog layout becomes questionable.

Although Kahng expressed some concerns regarding topological representations in general – as *scalability*, or *"the packing obsession"* [4], these concerns are hardly justified in our context. Scalability to instances of hundreds of thousands of cells is a problem indeed, but only for digital circuits. Our results are very effective (see Section 4), even when processing real-life analog placement problems, where part of the blocks are "soft" capacitors or having a number of alternative realizations. Moreover, the optimization deals with a complex cost function with multiple objectives subject to "hard" and "soft" constraints.

The problem of analog placement is a perfect opportunity for non-slicing topological representations to prove their practical value.

3 Device-level analog placement

In order to automatically produce analog device-level layouts matching in density and performance the high-quality manual layouts, an analog placement tool must be provided with the capability of dealing with analog-specific features:

1) The ability to deal with topological constraints for symmetry and device matching.

In high-performance analog circuits it is often required that groups of devices are placed symmetrically with respect to one or several axes. Analog circuits use very often differential architectures based on electrically symmetric networks, and therefore, layout symmetry matches the induced parasitics in the two halves of a group of devices. Symmetry is also used to prevent unwanted oscillations by balancing thermal couplings in differential structures, or to reduce the thermal sensitivity of the circuit. In order to reduce systematically-induced mismatches – produced by dissimilar geometrical choices, matching groups of devices should be constrained to the same orientation and variant.

2) The ability to arrange devices such that critical structures are shared in common (*device merging* or *geometry sharing*) in order to reduce both layout density and induced parasitics.

3) The existence of a library of device generators and the ability to exploit their reshaping capability.

The device-matching constraints and the geometry sharing situations can be handled relatively easy when using topological encodings 1) by imposing constraints at the move-set level of the combinatorial optimization algorithm employed to explore the set of topological representations, 2) by introducing penalty terms in the cost function, and 3) by modifying the procedure which builds the placement configurations from the topological encodings.

However, dealing efficiently with symmetry constraints in the framework of topological representations has proven to be a problem of unexpected difficulty. In our first empirical tests using the sequence-pair representation [8], we were initially tempted to perform minor changes to the search space exploration: if the current encoding proved to be consistent with the symmetry constraints then the cost of the placement configuration would be evaluated and the annealing algorithm would operate normally; otherwise, the current encoding would be infeasible (in symmetry point of view) and, therefore, disregarded. Unfortunately, such a simple solution proved to be extremely ineffective, the computation times being disappointingly high (see Section 4). The main reason was revealed to be the huge number of encodings infeasible with respect to symmetry constraints, which was overwhelming versus the "symmetric-feasible" ones [1].

This experiment led to the following conclusion: a topological representation proves to be effective only if the analog placement tool is able to explore only those encodings which comply with the imposed set of symmetry constraints. This obviously better strategy encounters two hurdles though:

a) *how to recognize the encodings complying with the given symmetry constraints, without building the corresponding layout ?*

b) *how to efficiently restrict the exploration* (performed by a combinatorial optimization, like simulated annealing) *only to the subspace of these "symmetric-feasible" (S-F) encodings ?*

3.1 Using S-F sequence-pairs

In the case of the sequence-pair representation [8], the questions above have already received satisfactory answers [1].

Let (α, β) be the sequence-pair of a placement configuration containing a number of symmetry groups (each group composed of pairs of symmetric blocks, and self-symmetric blocks relative to a common vertical axis). Denoting by α_A^{-1} the position of block A in sequence α (as α can be viewed as a one-to-one mapping, α^{-1} is well-defined) and defining similarly β_A^{-1}, and also denoting by $sym(x)$ the block symmetric to x, the sequence-pair (α, β) is called *symmetric-feasible* (S-F) if for any distinct blocks x, y in any of the symmetry groups

$$(S) \qquad \alpha_x^{-1} < \alpha_y^{-1} \iff \beta_{sym(y)}^{-1} < \beta_{sym(x)}^{-1}$$

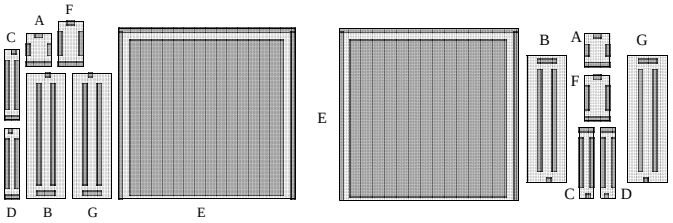

Figure 1: Block placement encoded by the sequence-pairs (a) ($CDAFBGE$, $DCBGAFE$) ; (b) ($EBAFCDG$, $EBCDFAG$)

Fig. 1 displays two placement configurations corresponding to the sequence-pairs ($CDAFBGE$, $DCBGAFE$) and ($EBAFCDG$, $EBCDFAG$), respectively. Assuming the existence of a symmetry group composed of the pairs of symmetric cells (C, D) and (B, G), and the self-symmetric cells A and F, the first encoding is infeasible and the corresponding placement does not verify the symmetry constraints; the second sequence-pair is symmetric-feasible – in the sense of condition (S), leading to a correct placement solution.

The exploration of the sequence-pair encodings using simulated annealing can be restricted to the symmetric-feasible codes as follows: it is sufficient to start the search from an initial sequence-pair which can be easily built to verify condition (S) (for instance, corresponding to a placement where the symmetric pairs in any symmetry group top one another) and, afterwards, to constrain the move-set of the optimizer such that the property (S) of symmetric-feasibility is preserved: for instance, if two cells from symmetric pairs are interchanged in sequence α, then their symmetric counterparts must be also interchanged in sequence β, etc.

According to our tests (see Section 4), the strategy described above has proven highly beneficial in terms of both computation time and quality of solutions. The basic reason is that, instead of exploring a search space of size $(n!)^2$ – the total number of sequence-pairs for n placeable cells (over 25 million for the illustrative example in Fig. 1), only a reduced part this space (having the size[1] $(n!)^2/(2p+s)!$, where p is the number of symmetric pairs and s is the number of self-symmetric cells) is explored, i.e. the sequence-pairs verifying condition (S) – which are only 35,280 for this simple example.

3.2 Using O-trees

The use of the O-tree representation [3] (see Fig. 2a) has also produced good results employing the algorithm proposed by Pang *et al.* [10] due mainly to the fact that the

[1] A more general formula is proven in [1].

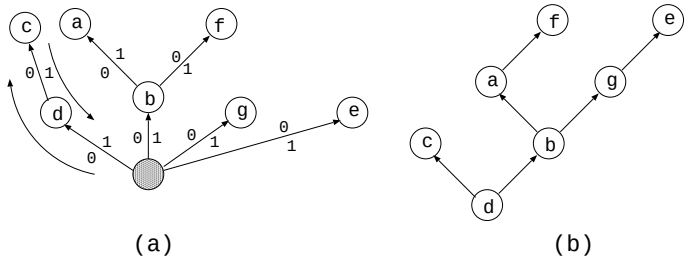

Figure 2: (a) O-tree representation, (b) binary tree representation of the block placement in Fig. 1a

number of O-trees (therefore, the search space) is always smaller than the number of sequence-pairs, the former representation exhibiting less redundancy than the latter. For the example in Fig. 1 ($n = 7$), the number of O-trees is [5] $\frac{1}{n+1}\begin{pmatrix} 2n \\ n \end{pmatrix} \cdot n! = 2,162,160$ – significantly smaller than the total number of sequence-pairs (over 25 million), but still larger than the total number of symmetric-feasible sequence-pairs[2] (i.e., 35,280 when $p = s = 2$).

The exploration of the O-trees is done with simulated annealing as well. After each new O-tree is generated, a verification routine – having the complexity $O(n^2)$ – attempts to detect as early as possible whether the current encoding is symmetric-feasible or not. Two constraint graphs – vertical G_v and horizontal G_h – are built taking into account both the positioning constraints derived from the O-tree and from the given symmetry constraints. The O-tree is *symmetric-feasible* if G_h is acyclic and G_v does not contain positive cycles [10]. Only in this case the move is considered for acceptance, according to the probabilistic hill-climbing of the annealer. Otherwise, the move is immediately rejected as the placement built from the O-tree cannot satisfy the symmetry constraints.

3.3 Using S-F binary trees

Binary trees can lead to a better performance than both the sequence-pair and the O-tree representations: while offering in general the same solution space size as the O-trees (due to a one-to-one mapping property [5]), in the presence of symmetry constraints the exploration can be restricted to a proper subset of binary trees – which were called *symmetric-feasible* (S-F) binary trees.

Let $((leftSubtree)root\ \ rightSubtree)$ and $(root(leftSubtree)\ \ rightSubtree)$ be the recursive inorder and, respectively, preorder representations of a binary tree. Let (γ, δ) be the pair of sequences[3] derived from the inorder and preorder representations by dropping the parentheses (used only to express the tree structure). Using the same notations as for sequence-pairs, the binary tree encoding is called *symmetric-feasible* (S-F) relative to a symmetry group G if for any distinct blocks x, y in G:

$$(S') \qquad \gamma_x^{-1} < \gamma_y^{-1} \iff \delta_{sym(y)}^{-1} < \delta_{sym(x)}^{-1}$$

The S-F binary trees have a desirable property: their number is smaller than both the S-F sequence-pairs and O-trees. Indeed, if all the blocks in the layout are organized in a symmetry group ($n = 2p$), the number of (labeled) S-F binary trees is [5] $\frac{1}{p+1}\begin{pmatrix} 2p \\ p \end{pmatrix} \cdot p! \cdot 2^p \leq (2p)!$ which is the number of symmetric-feasible sequence-pairs when $n = 2p$.

The annealing algorithm can be adapted to explore only the subspace of S-F binary trees rather than the whole space of binary trees: (1) the initial binary tree must be symmetric-feasible. If (a_i, b_i) are p pairs of symmetric blocks, such a binary tree is, e.g., $(a_1 \cdots a_p\ b_p \cdots b_1 \cdots)$, using the preorder traversal representation; (2) the moves during the exploration (changes of the binary tree codes) are chosen such that property (S') continues to hold.

3.4 Using the absolute representation

A complementary placement tool using the absolute representation (briefly presented in Section 2) has been implemented as well. The tool employs *virtual* symmetry axes [7], having mobile positions, to model multiple symmetry groups.

4 Experimental results

A placement tool for analog layout has been implemented and embedded in a retargetable object symbolic environment for layout design. The tool can use alternative optimization algorithms based both on the absolute representation (similar to other traditional approaches), and on different topological representations. In order to ensure a correct comparative evaluation, the simulated annealing schedule is identical for all the placement algorithms. The results presented below have been obtained on an HP 9000/777 workstation.

Fig. 3 displays two placement solutions – one obtained using S-F sequence-pairs, the other using S-F binary trees – of a 110-cell analog circuit having six symmetry groups. Fig. 4 shows other two examples, the first obtained using the absolute representation, the second – based on O-trees. Note that in all these examples, the (usually large) capacitors are soft cells, which shapes are optimized and regenerated at the end of the placement.

[2] However, when the asymmetric part of the circuit is larger, the number of O-trees can become smaller than the number of symmetric-feasible sequence-pairs (e.g., if in Fig. 1 there is only one symmetric pair of cells: $p = 1$ and $s = 0$).

[3] Usually (γ, δ) is not equal to the sequence-pair representation (α, β). There are sequences α and β which do not correspond to the inorder and preorder traversals of any binary tree: such a sequence-pair is, for instance, (CAB, ABC).

Table 1 displays the results obtained for several analog blocks, components of a digital spread spectrum transceiver used in cordless telephone applications and wireless modems. The conclusions derived from these experiments are the following:

1) the use symmetric-feasible sequence-pairs (as described in section 3.1) is highly beneficial (both in terms of CPU time and solution quality) in comparison with the "classic" sequence-pair representation where time is spent investigating codes which are infeasible in symmetry point of view;

2) the results for the "classic" sequence-pairs on tests exhibiting symmetry are much worse than those obtained using the traditional absolute representation; however, in general, the use of *S-F* sequence-pairs has yielded better results: therefore, topological representations should be used with care in the presence of symmetry constraints;

3) using S-F sequence-pairs is more effective in terms of speed than using O-trees in the examples exhibiting more symmetry, but less effective otherwise;

4) if the asymmetric part of circuit is predominant, the CPU times when using S-F binary- and O -trees are similar; otherwise, using S-F binary trees is the most effective in terms of speed, as the solution space is the smallest.

In general, all the algorithms produced high quality placement solutions. However, the optimization techniques operating with the absolute and tree representations have proven difficult to tune. On the other hand, the algorithm using S-F sequence-pairs exhibits a clearly higher robustness.

5 Conclusions

This paper has addressed the problem of device-level analog placement in a non-traditional way, using very recent results on topological representations for non-slicing floorplans. Using skillfully the symmetry constraints to remodel and diminish the size of the solution space, the topological representation techniques have proven to be more effective in terms of computation speed than traditional approaches based on the absolute representation.

References

[1] F. Balasa, K. Lampaert, "Symmetry within the sequence-pair representation in the context of placement for analog design," *IEEE Trans. on CAD of IC's and Syst.*, Vol. 17, No. 7, July 2000.

[2] J. Cohn, D. Garrod, R. Rutenbar, L. Carley, *Analog Device-Level Automation*, Kluwer Academic Publishers, 1994.

[3] P.-N. Guo, C.-K. Cheng, T. Yoshimura, "An O-tree representation of non-slicing floorplan and its applications," *Proc. 36th ACM/IEEE Design Automation Conf.*, pp. 268-273, June 1999.

[4] A.B. Kahng, "Classical floorplanning harmful?," *Proc. Intn'l Symp. on Physical Design*, pp. 207-213, San Diego CA, April 2000.

[5] D.E. Knuth, *The Art of Computer Programming* (3rd edition), Addison Weslay Longman, 1997.

[6] K. Lampaert, G. Gielen, W. Sansen, "A performance-driven placement tool for analog integrated circuits," *IEEE J. of Solid-State Circuits*, Vol. SC-30, No. 7, pp. 773-780, July 1995.

[7] E. Malavasi, E. Charbon, E. Felt, A. Sangiovanni-Vincentelli, "Automation of IC layout with analog constraints," *IEEE Trans. on CAD of IC's and Syst.*, Vol. 15, pp. 923-942, Aug. 1996.

[8] H. Murata, K. Fujiyoshi, S. Nakatake, Y. Kajitani, "VLSI module placement based on rectangle-packing by the sequence-pair," *IEEE Trans. on CAD of IC's and Syst.*, Vol. 15, No. 12, pp. 1518-1524, Dec. 1996.

[9] S. Nakatake, K. Fujiyoshi, H. Murata, Y. Kajitani, "Module packing based on the BSG-structure and IC layout applications," *IEEE Trans. on CAD of IC's and Syst.*, Vol. 17, pp. 519-530, June 1998.

[10] Y.-X. Pang, F. Balasa, K. Lampaert, C.-K. Cheng, "Block placement with symmetry constraints based on the O-tree non-slicing representation," *Proc. 37th ACM/IEEE Design Automation Conf.*, pp. 464-467, Los Angeles CA, June 2000.

[11] R. Rutenbar, J. Cohn, "Layout tools for analog ICs and mixed-signal SoCs: a survey," *Proc. Intn'l Symp. on Physical Design*, pp. 76-83, April 2000.

Design (# sym.groups)	Nr. cells	Sym. constr.	Absolute		Seq-pair		S-F Seq-pair		O-tree	S-F Bin-tree
			Time	Area	Time	Area	Time	Area	Time	Time
bias crt. generator	85	-	41	45.1	32	44.0	32	44.0	25	25
lpf2_b25b (1)	52	12	29	153.0	26	198.2	12	137.7	12	11
modbias_2p4g (5)	87	30	67	65.9	99	84.7	41	56.9	43	33
lnamixbias_2p4g (6)	110	42	112	91.9	157	116.5	76	95.4	90	62
frequency divider (5)	116	46	92	49.6	131	72.4	68	51.4	84	57

Table 1: Placement results (Time [min] , Area [$10^3 \times \mu m^2$])

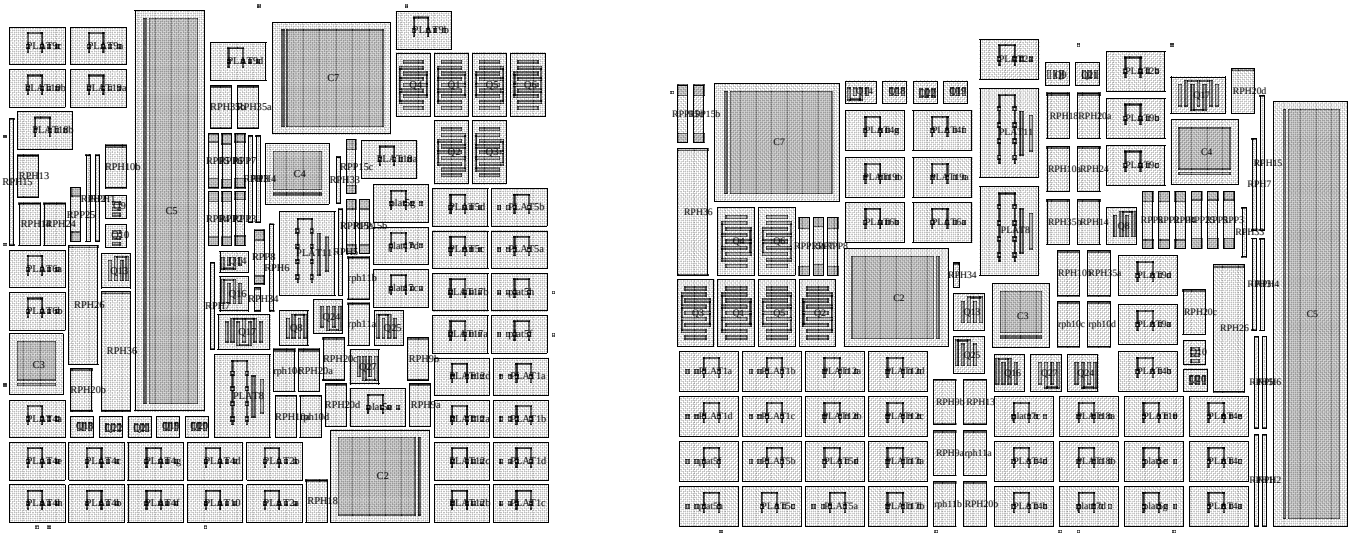

Figure 3: Placement of *lnamixbias_2p4g* using (a) S-F sequence-pairs, and (b) S-F binary trees

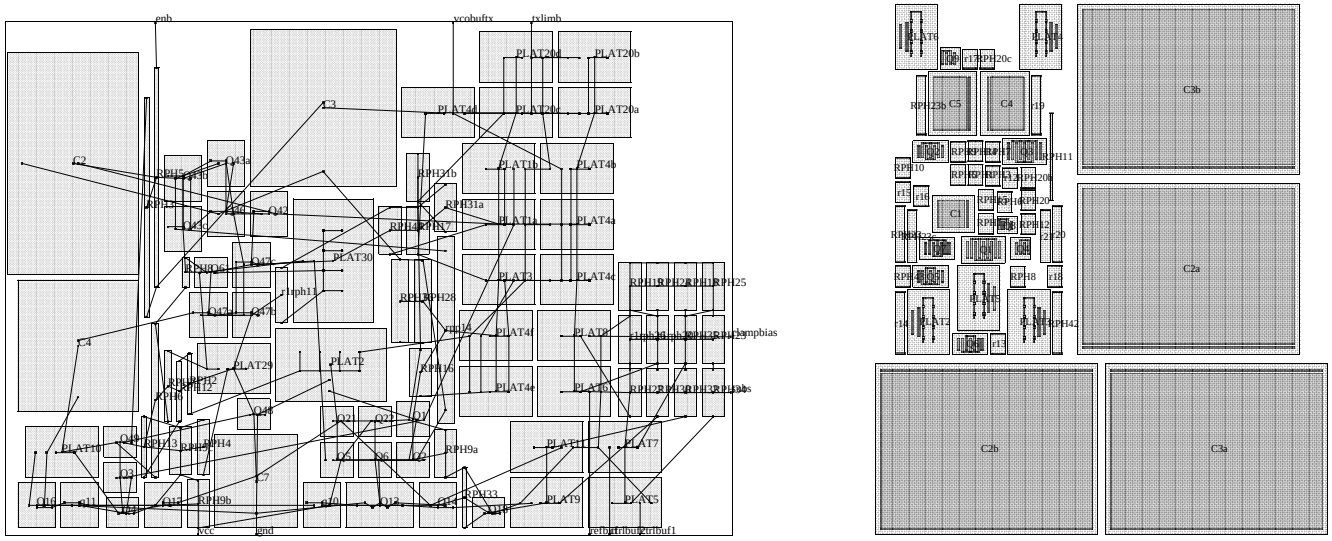

Figure 4: (a) Placement of *modbias_2p4g* using absolute representation, and of (b) *lpf2_b25b* using O-trees