On-the-Fly Layout Generation for PTL Macrocells

Luca Macchiarulo [‡] Luca Benini [*] Enrico Macii [‡]

[*] Università di Bologna
Bologna, ITALY 40136

[‡] Politecnico di Torino
Torino, ITALY 10129

Abstract

Pass transistor logic (PTL) has been recently proposed as an alternative to standard MOS for aggressive circuit design. Even though PTL has been successful in a few handcrafted designs, its acceptance into mainstream digital design critically depends on the availability of tools for logic and physical synthesis and optimization. The automatic synthesis of pass transistor circuits starting from BDDs has been intensively studied in the past with promising results, but back-end tools for PTL cell generation are still missing. We describe an automatic layout generator that has been designed for seamless integration in a library-free PTL design flow. The generator exploits the distinctive characteristics of pass transistor networks produced by synthesis to achieve quality of results comparable with state-of-the art commercial cell generation tools in a fraction of the execution time.

1 Introduction

Pass-transistor logic (PTL) has been proposed as a viable alternative to standard fully-complementary MOS (FCMOS) logic for aggressive deep sub-micron design [1, 2]. In the last few years, several pass transistor implementations of digital functional units have been presented in the literature, showing performance, area and power advantages over FCMOS [2]. Pass-transistor sub-circuits are routinely used in high-performance arithmetic units [1]. On the other hand, usage of PTL is still confined to handcrafted solutions in highly constrained designs.

PTL has disadvantages with respect to standard FCMOS, as detailed in [3]. The robustness of FCMOS is still unrivaled, and its scalability is well-known. In addition, a key feature that makes FCMOS preferable to PTL is the availability of a well-tuned design methodology and support tools for characterization, simulation and optimization [3]. For this reason, the last few years have witnessed a flurry of research activity on tools for supporting PTL design [4]. A good synthesis tool for PTL would undoubtedly facilitate its acceptance outside the circle of transistor-level designers. Thus, most research on PTL tools has focused on logic synthesis. Approaches to PTL synthesis can be coarsely grouped into two classes, namely *library-based* [5, 6] and *library-free* [7, 8, 9].

Library-based PTL synthesis closely follows the traditional two-step paradigm that has gained wide acceptance in the synthesis of FCMOS circuits [10]: A technology independent logic synthesis phase is followed by library binding. The intermediate form used for technology-independent optimization is based on binary decision diagrams (BDDs) and their many variants [4], as they can be easily mapped onto two-way MUX networks that, in their turn, can be efficiently implemented in PTL. Probably, the most widely known library-based synthesis approach is LEAP, developed by Yano et al. [5].

Library-free PTL synthesis is still based on a two-step paradigm, but library binding is replaced by on-the-fly macrocell layout generation. In other words, a coarsely clustered BDD network is fed to a cell generator that directly produces an optimized layout for all the clusters. This approach has been developed to address the key design closure challenge arising in deep submicron technology because of the increased relative importance of wiring-related parasitics with respect to device-related parasitics. Wiring for a coarse-granularity hierarchical layout can be controlled and optimized more easily than that for a fine-granularity layout of tiny elementary cells. On the other hand, a fast and high quality on-the-fly macrocell generator is critical for the success of library-free PTL synthesis. Obviously, the back-end requirements for the two PTL synthesis paradigms are quite dissimilar. Library-based synthesis needs standard placement and routing tools for the fine-granularity mapped netlist and, optionally, a high-quality cell compactor for the library (which can also be designed by hand). An example of such a small-scale PTL cell compactor is ALPS [11], developed by Sasaki *et al.* to support LEAP. In contrast, the critical back-end requirement for the library-free approach is a fast, predictable, and good quality PTL macrocell layout generator. To the best of our knowledge, no such tool has been developed yet. Hence, the viability of library-free PTL synthesis remains to be proven.

This paper addresses the missing link in the library-free PTL synthesis flow. We have developed a PTL macrocell generator that takes a BDD as input and produces, as output, a complete and legal layout. The tool is tailored to on-the-fly generation of PTL structures; thus, it produces results that are competitive with state-of-the art general-purpose commercial cell generators and compactors in a fraction of the time. From the physical design stand-point, our tool adopts a 2-dimensional layout style [12], in which the height of the macrocell is optimized jointly with its internal micro-placement, while width optimization is based on a modified left-edge algorithm. Speed and quality of the results stem from the PTL-tailored floorplan of the macrocell, providing a good starting point to optimization.

The remainder of this article is organized as follows. Section 2 provides an overview of the library-free PTL synthesis flow in which the on-the-fly macrocell generator of this paper must be plugged-in in order to complete the path to layout. Section 3 constitutes the core of our contribution, since it provides details on both the implementation style we have chosen for the PTL macrocells and on the tool that automatically performs layout generation for the macrocells. Section 4 collects the results of an extensive experimentation we have performed on the complete suite of the Iscas'85 combinational benchmarks [13]. Finally, Section 5 concludes the paper.

2 Design Flow for PTL Circuits

As mentioned in the introduction, the focus of this paper is on the implementation of part of the back-end (i.e., the on-the-fly PTL macrocell generator) of a fully-automated, library-free PTL synthesis flow. Nevertheless, in order to provide the reader with an outline of the complete flow, in this section we briefly summarize the key steps that, starting from a logic-level specification (i.e., a Boolean network) of the target design, must be taken to create an intermediate, library-independent description suitable to be fed to the macrocell generator. Figure 1 depicts the flow, in which we can identify five main phases:

- *Cell Creation*: Due to the technological limits imposed by series-connected pass transistors, as well as the scalability limitations of global BDDs, the original network needs to be decomposed into a set of macrocells of reasonable size. This step concerns the strategy to be followed for the decomposition of the network into manageable cells. The output of this phase consists of a set of decomposition points, each one with a multiple BDD representation to be used in the covering phase.

- *Cell Appraisal*: Each cell is optimized and characterized in terms of area and delay. At this stage, internal buffers are also allocated.

- *Covering*: The set of BDDs is used to build a cover of the entire network which is optimal with respect to some cost function. This is carried out by solving an implicit binate covering problem which uses a layout-driven cost function to accurately model the effect of cell area. The output of this phase consists of a set of BDDs, one for each decomposition point, that cover the original circuit. Each BDD has now a simple mapping to a PTL cell, since the technological constraint about the size and the length of a cell has been already considered in the creation phase.

- *Macrocell generation*: The layout for each PTL cell is generated using the algorithms of Section 3.

- *Placement and routing:* The logical structure of the netlist of macrocells, together with the PTL macrocells are passed to the place and route tool for the generation of the final circuit layout.

For the experiments described in Section 4, the input to the PTL macrocell generator (i.e., the covering of the initial Boolean network, specified as a set of BDDs) is obtained using the `waves` PTL synthesis tool [9].

We point out that estimates of the costs for each PTL cell that are used to drive the covering algorithm are obtained through models that relate size and shape of the BDD to area and delay of the cell layout. However, using such models may lead to inaccurate evaluations of the cost of some cells, thus causing the generation of a sub-optimal covering of the Boolean network. This problem may be overcome by bringing into the synthesis flow the macrocell generator; in other words, since macrocell generation is fast (see the results in Section 4), model evaluation could be effectively replaced by actual data extracted from the layout of the PTL cells produced by our generator.

Although applicable in principle, the viability of this option needs to be further investigated (for example, with respect to its scalability to industrial-size designs). Results of Section 4 were thus obtained using the models of [9], properly adapted to our technology and layout rules.

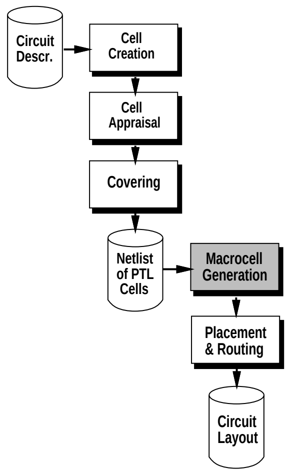

Figure 1: PTL synthesis flow.

3 On-the-Fly PTL Cell Generator

The PTL synthesis flow described in Section 2 has proven to be effective in generating compact decompositions, in terms of a cost function which is supposed to be related to the real implementation of the cells. However, this relation is meaningful only if two conditions are satisfied: *(i)* The logic function is realized with a technology that maps almost exactly the computational properties of the BDDs used in the optimization phase (that is, all nodes in a BDD are mapped onto multiplexers); *(ii)* The layout of the generated cells maintains some properties of the area function employed. This second condition is particularly stringent: Even if the schematic design of the cells (i.e., connectivity of transistors) resembles the structure of the BDD, the physical design, as it is performed by traditional synthesis tools (e.g., LAS by Cadence [15]), may jeopardize the connection with the cost function. In fact, in this case, the cost of the cell (in terms of area) is only loosely connected to any logical cost that might be extracted from the mapped BDDs. The consequence is that the optimization process tries to minimize an incorrect function; in addition, it is impossible to take into account the very special features of the implementations obtained at the end of the design flow. These considerations call for a physical design phase that properly accounts for all the characteristics of the data structure, so that a closer cost to the final implementation might be known at a very early stage. In this contribution, we offer a promising solution to these problems by exploiting some of the properties we identified in the functions generated by `waves`.

3.1 PTL Cell Features

Experiments run on standard benchmarks (see [9] for the details) provided with a wealth of observations on the typical results produced by `waves` (and, likely, by any BDD-minimizing decomposition engine). In particular, although many of the generated functions were atypical for a library-based technology mapping phase, as they depended on a fairly large number of variables (the limit was set to 9), all of them had an interesting property: The total number of nodes per level was extremely small. In fact, only 6 out of all 2439 generated BDDs had more than 2 nodes per level. Another important parameter associated to a BDD level is the number of cuts: Define them as the number of different edges that connect a node with index greater than i with nodes labeled with variables equal to or smaller than i. Two edges are considered distinct if and only if their sons are different. From the electrical point of view, the cuts represent the number of different signals that are needed from a level above to compute the value of the function. It is true that their number is relatively small on the average, less than 4 for most cases. This simplifies the routing task and has other positive consequences (i.e., buffer reduction). These properties (that are likely to be shared by all decompositions obtained through a similar process) can be conveniently exploited in the physical design phase of the macrocells.

3.2 General Placement and Microcells

The implementation style we used for the various functions is based on the previous observations. Let us consider first the netlist of the circuits: We chose an implementation that uses minimal multiplexers, realized with pairs of NMOS transistors (as in LEAP [5]). The complemented signal is generated by a single inverter feeding all the MUXes driven by the same variable; as the number of MUXes per level is relatively small, it is possible to use a minimum size inverter without loosing much in performance. On the other hand, it is absolutely impossible to avoid the introduction of re-powering inverters to break long paths of pass transistors; therefore, we added a group of inverters every three levels of nodes (the number of 3 is often considered as a good compromise between loss in area and gain in performance). An interesting observation is that if we uniformly introduce inverters every three BDD levels (buffering all the signals passing through) the logic functionality of the realized cell will be preserved, provided that all the leaves are re-assigned accordingly, as Figure 2c shows. Due to the limited number of cuts experimentally observed, the loss in area due to buffer insertion is relatively small, proportional to the number of cuts in the levels where the buffers are placed. An output buffer (not shown in Figures 2 and 3) is then added after the topmost MUX.

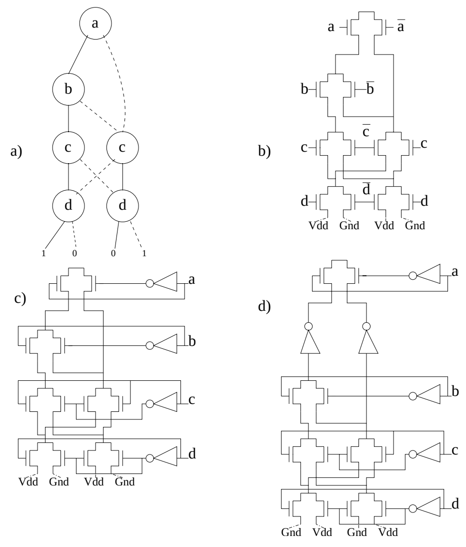

Figure 2: Mapping of a BDD onto PTL logic.

The starting point for layout generation is a netlist containing three elementary microcells: *(i)* A MUX, consisting of a pair of NMOS transistors (*node cell*); *(ii)* An input inverter (*complement generator*): *(iii)* A re-powering inverter (*buffer*). The problem becomes to decide how to place these microcells and route them according to their connectivity in such a way that area occupation gets reduced and (possibly) fast area estimation is enabled. This task is similar to that of traditional cell generators, with the added constraint (which might turn out to be an advantage) of working on a specific implementation rather than a generic arrangement of transistors.

The key idea is to take into account the "natural" arrangement given by the BDD structure when setting up the floorplanning of the macrocell. All BDD nodes corresponding to a given variable are placed at the same y coordinate and fed by the same two signals, one of them being the external input, the other the locally generated complement. Thus, the BDD layout grows "taller" as the number of input variables increases. The BDD grows "wider" (i.e., it grows in the x dimension) as the number of nodes dependent on a variable increases. More precisely, the width of the layout is set by the maximum number of nodes at any level of the BDD. Experimental data have shown that, for functions resulting from the optimization process, the width (i.e., the number of nodes per level) of the BDDs is small and relatively uniform. Hence, the floorplanning style described above is very effective on typical BDDs generated by the synthesizer.

We have chosen to constrain the maximum number of BDD nodes per physical level, and to fold all levels with a large number of nodes, if any, into extra levels in the vertical

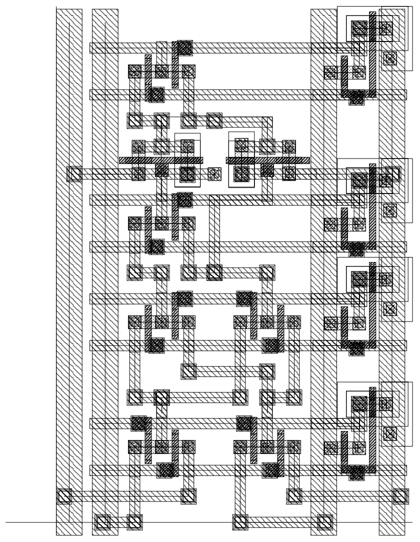

Figure 3: Layout generated for the example of Figure 2.

direction. Hence, one logical level in the BDDs (i.e., all the nodes controlled by the same variable) always results in one or more *physical levels* (i.e., horizontal slices) of the layout. The resulting structures are “taller”, but their width is tightly controlled. With this strategy, we obtain a simple rectangular layout, with uniform width (the only variability is due to routing, as discussed in the next sections). An example of an actual layout is given in Figure 3. The big advantage of this placement strategy is that the final dimensions of the cells are determined by factors, such as the support variable number, the node count and the number of cuts per level, that are fairly easy to compute from the information of the mapped BDD.
The macrocells created by the generator must be placed and connected in standard-cell style. From the preceding discussion, it turns out to be much easier to control the horizontal dimension variability (caused solely by the different dimensions of the vertical channel) rather than the cell height (depending on the number of support variables, buffer levels, horizontal routing and, in general, widely different from cell to cell). However, in a layout with a standard cell style, the variability in the y direction is paid in terms of wasted area, because the height of a row is determined by the maximum height of any macrocell in the row. This macro layout style emphasizes the need for keeping macrocell width under strict control. Therefore, the cells are rotated by 90 degrees, and placed in rows (see Figure 4).

3.3 Placement Optimization and Vertical Routing

Based on the statistics summarized in Section 3.1, two nodes per level are chosen. A larger number, in fact, would result in a big waste of silicon area, as the great majority of BDD levels had one or two nodes, while one single node would create more problems in the routing phase. The

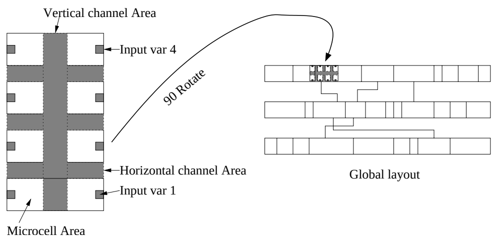

Figure 4: Layout abstract of a sample cell and the proposed placement and routing style.

general floorplan of the cells is very regular and uniform: Each level has one or two nodes routed through a vertical channel connecting different levels and various horizontal channels connecting the levels among each other and to the vertical channel. Power supply and ground tracks run parallel to the vertical channel, while inputs are fed from the sides of the structure, parallel to the horizontal channels.
The placement of the nodes can be divided into two separate phases: *Global placement* and *detailed placement*. For global placement, we intend the choice of the relative ordering of the variables inside the BDD, that fixes the relative positions of nodes driven by different input signals. The variable order is automatically determined by the reordering tool available inside the BDD package. Detailed placement consists of the choice of the physical level and the position inside the level that a particular node has to occupy; for example, if a level has four nodes (called 1,2 3 and 4), a choice has to be taken whether to place nodes 1 and 2 in the first physical level assigned to that variable or to the second, as shown in Figure 5.

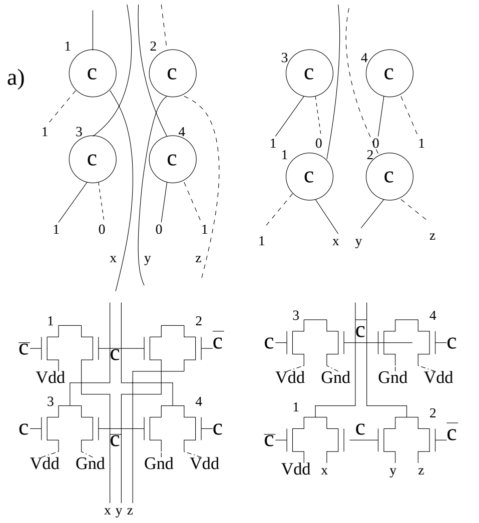

Figure 5: Effects of detailed placement on vert. routing.

The choice has no consequence on the area occupied by the microcells, but it can influence the quality of the routing. For example, in the case of two signals coming from previous levels that need to be fed to the second physical level, more space in the vertical channel is required. The posi-

tion (left or right) in the physical level affects the quality of horizontal routing.
As we want to build cells to be used with a standard-cell style (after a 90 degree rotation, as explained in Section 3.2), we would like to obtain cells that are as uniform as possible in their x dimension. Therefore, we gave precedence to the minimization of the vertical channel. In order to do so, for each variable, each possible assignment is tried and the one minimizing the number of vertical tracks is used. The total number of vertical tracks is a global property of the cell, as it traverses the entire length of the structure. When the physical level is decided, the position inside the level (left or right) is assigned through a heuristics that tries to simplify the horizontal routing. When all variables are considered in this phase, we can effectively route the vertical channel and compact it.

3.4 Horizontal Routing

After the vertical routing is fixed, it is still necessary to bring all the signals to the inputs of the multiplexers, and to connect their outputs either to the nodes of the following level or to the vertical tracks. Conversely from the vertical case, horizontal routing is local to a pair of levels, even if its quality is also dependent on how the vertical tracks have been assigned. We solved the problem with a constrained version of the left edge routing algorithm [14], that takes into account some limits on track assignment that arise from the interaction between horizontal and vertical routing to prevent wrong shorts. This phase completes the physical design of the cell.
In conclusion, the area of the final cell can be easily determined. The height (e.g., the width of the layout of Figure 3) is a constant (width of two nodes plus one complement generator) plus the width of the vertical routing channel. The width is equal to the sum of the heights of all the levels, the buffers and the sum of the horizontal routing tracks.
We observe that, except for the routing information, all other contributions to the area are known in advance by considering the BDD structure. Since a simple upper limit to the routing space can be determined before effectively doing the routing, a very fast way of estimating the size of a cell is available for usage as a replacement of the models developed in [9].

4 Experimental Results

To assess the feasibility of the approach described above, we applied it to the generation of a number of macrocells, as obtained by running `waves` on a set of standard benchmarks. Our goal was that of quantifying the efficiency of macrocell layout generation alone; therefore, we set up a comparison methodology that was able to characterize our generator in terms of two design variables, namely cell area and generation time. The rationale behind this is that a useful tool will need to be competitive with traditional flows in terms of areas and, at the same time, extremely fast. We therefore imported two layouts per cell: One with our tool, the other with the commercial automatic synthesizer LAS, a state-of-the-art generic automatic layout generator [15]. The two cells had to be generated starting exactly from the same netlists, since we wanted to analyze the performance of the layout tool alone. The BDDs representing the functions obtained through decomposition are converted into an electrical netlist of PMOS and NMOS transistors. These netlists are imported into the Cadence environment, and LAS is run on them with its default options to obtain a legal layout in a 0.25 micron technology. On the other hand, the same netlist is mapped onto a layout by our tool, then imported into Cadence to verify DRC compliancy w.r.t. the same technology. Both our cells and LASs' have the same VDD and GND track height, laid out in metal 1, while inputs and outputs are routed in metal 2 tracks and accessed on both upper and lower sides. Generation time of the cells (stripped of the time necessary to read/write the I/O files) is calculated and compared to the Cadence generation time (only the cell generation and compaction times were used, stripped of the I/O and database creation times). The comparison (see Table 1) shows that our tool is at least 1 order of magnitude faster than LAS, the reason being that it has to experiment on a small number of cases to obtain a good placement and routing, while LAS uses a more complicated algorithm working on the entire connectivity graph of the netlist. Time comparisons are even more favorable when we consider that LAS compaction phase is more expensive than layout generation, giving a difference of up to 400 times in CPU time (Table 1). It is important to note, however, that the generated cells are relatively small, and the absolute times needed for their production is negligible if considered at the end of a design flow. The time is no longer negligible if the generation (or estimation) has to be used at an early stage, within an optimization loop.
Also from the point of view of area our tool produces a much more compact realization w.r.t. the uncompacted view. Clearly, this comparison is unfair to LAS as the uncompacted view is typically used as an intermediate step of a flow that leads to the compacted layout. The compaction phase, which is relevant in terms of computation time, manages in greatly reducing the cells' dimensions, giving results that match closely those obtained with our tool. Even after compaction, however, the overall dimension of the cells is bigger than in our case, with percentage differences ranging from 0 to 15 (see Table 2).
The improvement becomes even more dramatic if we consider that the generated cells are intended to be placed and routed with a standard cell style, so that cell height variation is material in determining an area increase (all cells have to be matched in terms of their widths, and therefore the maximum x dimension must be considered). In this case (see Table 3), our cells are better with a percentage going from 13 to 49.
These results led us to conclude that our tool provides a good solution to the problem of fast generation of cells that are needed for a PTL synthesis flow as the one suggested in [9]. Aim of the comparison was to prove the speed and effectiveness of our generator in creating PTL macrocells.

Circuit	CPU Time (seconds)		
	Unc.	C.	Our
C432	6.22	29.52	0.076
C499	4.32	44.24	0.1
C880	12.73	63.58	0.22
C1355	4.17	44.27	0.12
C1908	10.93	81.29	0.27
C2670	16.13	104.39	0.29
C3540	37.07	212.38	0.6
C5315	41.19	223.84	0.49
C6288	48.89	329.36	0.75
C7552	45.68	341.48	0.93
Total	227.33	1375.35	3.85

Table 1: Time for cell generation.

Circuit	Area(μm^2)			Impr. (%)
	Unc.	C.	Our	
C432	42017	11177	10725	4
C499	69338	17196	15137	11
C880	95214	24200	24312	0
C1355	69338	17196	15137	11
C1908	96545	24254	21114	15
C2670	139417	36539	32797	11
C3540	291644	76347	74002	3
C5315	326456	84974	81602	4
C6288	562067	146554	139932	5
C7552	423265	106093	99785	6
Total	2115301	544530	514633	6

Table 2: Area for cell generation.

Circuit	Area (μm^2)			Impr. (%)
	Unc.	C.	Our	
C432	53305	13625	11562	18
C499	74507	19331	15725	23
C880	115134	29464	25990	13
C1355	74507	19331	15725	23
C1908	106303	29442	23845	23
C2670	160555	43740	35928	22
C3540	483738	133113	86324	54
C5315	426322	120527	92451	30
C6288	772055	221124	149235	48
C7552	1075190	276288	129814	113
Total	3341616	905985	586599	54

Table 3: Area (Standard Cell) for cell generations

It is obvious that the tool cannot provide efficient solutions to the case of arbitrary cell generation, for which LAS is certainly best targeted.

5 Conclusions

Although several researchers in the logic and physical synthesis domains have addressed the problem of automatically generating netlists using PTL technology instead of FCMOS, a complete top-to-bottom flow is nowadays still missing. Most of the investigation has been done in the front-end of the flow, wherein a decomposition of a given logic function can be obtained using sophisticated and powerful tools. Also the technology for placement and routing can be easily found in both the academia and the EDA market. On the other hand, algorithms and tools that enable the automatic generation of the layouts for all the PTL macrocells in which the original design is decomposed constitute the missing ring of the chain.

In this paper, we have addressed the problem of automatically generating the layout for logic functions represented as BDDs. The target is using the generator on-the-fly; therefore, besides the usual capabilities required to any layout generator (i.e., ability of generating compact macrocells), the tool must guarantee short execution times.

We have benchmarked the performance of the PTL macrocell generator on a set of standard examples, i.e, the Iscas'85 circuits. The results we have obtained are very promising, since the size of the macrocells created by our tool are, on average, as compact as those generated using a commercial layout generator. Layout generation time, however, is much shorter.

References

[1] V. Oklobdzija, "Differential and Pass-Transistor CMOS Logic for High Performance Systems," *Microelectronics Journal*, Vol. 29, No. 10, pp. 679-688, 1998.

[2] T. Kuroda, T. Sakurai, "Low-Power Circuit Design Techniques for Multimedia CMOS VLSIs," *Electronics and Communications in Japan, Part 3*, Vol. 81, No. 9, p. 67-74, 1998.

[3] R. Zimmermann, W. Fichtner,i "Low-Power Logic Styles: CMOS versus Pass-Transistor Logic," *IEEE Journal of Solid-State Circuits*, Vol. 32, No. 7, pp. 1079, 1997.

[4] K. Taki, "A Survey for Pass-Transistor Logic Technologies," *ASPDAC-98*, pp. 223-225, 1998.

[5] K. Yano, Y. Sasaki, K. Rikino, K. Seki, "Top-Down Pass-Transistor Logic," *IEEE Journal of Solid-State Circuits*, Vol. 31, No. 6, pp. 792-803, 1996.

[6] R. Chaudhry, T.-H. Liu, A. Aziz, J. Burns, "Area-Oriented Synthesis for Pass-Transistor Logic," *ICCD-98*, pp. 160-167, 1998.

[7] V. Bertacco, S. Minato, P. Verplaetse, L. Benini, G. De Micheli, "Decision Diagrams and Pass Transistor Logic Synthesis," *IWLS-97*, Paper 3.1, 1997.

[8] P. Buch, A. Narayan, A. R. Newton, A. L. Sangiovanni-Vincentelli, "Logic Synthesis for Large Pass Transistor Networks," *ICCAD-97*, pp. 663-670, 1997.

[9] F. Ferrandi, A. Macii, E. Macii, M. Poncino, R. Scarsi, F. Somenzi, "Symbolic Algorithms for Layout-Oriented Synthesis of Pass Transistor Logic Circuits", *ICCAD-98*, pp. 235-241, 1998.

[10] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw Hill, 1994.

[11] Y. Sasaki, K. Rikino, K. Yano, "ALPS: An Automatic Layouter for Pass-Transistor Cell Synthesis," *ASPDAC-98*, pp. 227-232, 1998.

[12] M.A. Riepe, K.A. Sakallah, "Transistor level Micro-Placement and Routing for Two-Dimensional Digital VLSI Cell Synthesis," *ISPD-99*, pp. 74-81, 1999.

[13] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *ISCAS-85*, pp. 785-794, 1985.

[14] M. Sarrafzadeh, C.-K. Wong, *An introduction to VLSI physical design*, McGraw Hill, 1996.

[15] S. Chow, H. Chang, J. Lam, Y. Liao, "The Layout Synthesizer: An Automatic Block Generation System," *CICC92*, pp. 11.1.1-11.1.4. 1992.

[16] S. I. Minato, *Binary Decision Diagrams and Applications for VLSI CAD*, Kluwer, 1996.