Library–Less Synthesis for Static CMOS Combinational Logic Circuits

S. Gavrilov, A. Glebov
Research Institute for VLSI CAD Systems, Russian Academy of Sciences, Moscow Russia.
S. Pullela, S. C. Moore, A. Dharchoudhury, R. Panda, G. Vijayan, D. T. Blaauw
Unified Design Systems Laboratory, Motorola, Austin.

Abstract

Traditional synthesis techniques optimize CMOS circuits in two phases i) logic minimization and ii) library mapping phase. Typically, the structures and the sizes of the gates in the library are chosen to yield a good synthesis results over many blocks or even for an entire chip. Consequently, this approach precludes an optimal design of individual blocks which may need custom structures. In this paper we present a new transistor level technique that optimizes CMOS circuits both structurally and size-wise. Our technique is independent of a library and hence can explore a design space much larger than that possible due to gate level optimization. Results demonstrate a significant improvement in circuit performance of our resynthesized circuits.

1. Introduction

CMOS circuit optimization can be performed at various levels of design description [1]. At the lower end of the design process, synthesis tools typically perform optimization at the gate level and operate in two phases. The technology-independent phase performs manipulation of boolean functions, which can be represented by logic equations or BDD networks of simple logic gates [2-5]. The technology-dependent phase subsequently maps the boolean functions into a target library—typically consisting of a few hundred logic gates [4, 6] designed and fixed apriori, to yield desirable area-performance tradeoffs across multiple blocks and sometimes across multiple chips. Consequently, the circuit structures are forced to stay within the predetermined library precluding an optimal solution that effectively utilizes the total design space of transistor structures. Custom design circuits to some extent alleviate this problem, however, there are no known techniques for automatic synthesis of custom logic.

It is apparent that the finer the granularity of the logic structures available for optimization, the greater is the quality of solution obtained in terms of area, power, and performance. In this regard, optimization of logic circuits at the transistor level, as opposed to gate level yields a better solution, since a much larger design space is explored. In addition, a transistor level approach would permit the use of significantly more accurate models for power, area, and delay, thereby increasing the scope of agreement with simulation results. However, state-of-the-art transistor level optimization techniques are limited to simple reordering of the transistor chains [6] and to transistor sizing.

In this paper, we propose a new approach which operates on a synthesized circuit and performs a transistor level logic optimization by transistor level restructuring. Consequently, we extend the design space beyond the traditional limits set by cell libraries. In addition, we also consider sizing of the individual transistors during structural optimization This ensures that the performance of a given structure is optimal at the given area of interest. We refer to our technique as transistor level resynthesis or simply resynthesis. The output of resynthesis is an area-delay trade-off curve shown in Figure 1. Each point on this curve represents a solution which is structurally as well size-wise optimized for circuit delay. While the smooth continuous parts represent transistor level circuit sizing with a fixed topology, the discontinuities result due to the timing improvements from restructuring.

2. Overview of Resynthesis

As one can observe, the number of transistors is extremely large if the whole circuit must be considered for sizing and restructuring simultaneously. Therefore, resynthesis breaks the circuit into smaller segments of logic which we refer to as *windows*. Typically, each window contains up to 50 transistors or approximately 10-20 gates. For each window, a large number of functionally equivalent, transistor level structures are generated using a simulated annealing method. These windows differ both in how the overall logic is implemented in terms of individual gates as well as the structure of the gate. Each window is also sized with a fast approximate sizing algorithm to match the size of the original window. We refer to this process as local resynthesis and it is described in more detail in section 4.2. Internally, local resynthesis evaluates all new windows relative to the original timing constraints of the window. The best window is inserted back in the circuit at minimum size. The overall circuit is then evaluated for relative merit after sizing the circuit back to the original area. (the dotted curves in Figure 1 show the resizing with new window). The best window in terms of timing is preserved after every iteration while

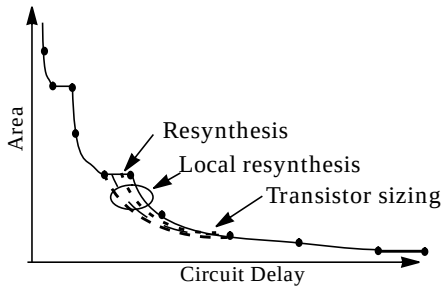

Figure 1. Resynthesis: Area delay trade-off curve.

0-89791-993-9/97 $10.00

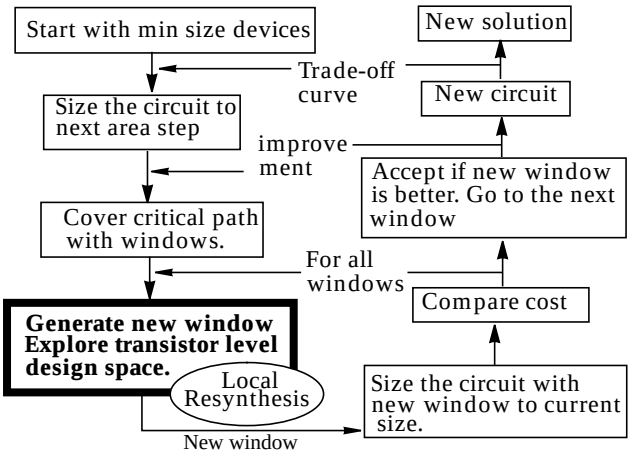

Figure 2. Resynthesis Flow.

the rest are discarded. For efficiency only the critical path in the circuit is covered with windows and each window is evaluated for improvement in timing. This step is repeated till no further improvement is possible at this size. Subsequently the whole circuit is sized to a predetermined percentage, and the above procedure is repeated. Figure 2 shows the basic steps involved in resynthesis.

As one can observe, to operate on an equivalent design space, a library based synthesis approach will need an enormously large number of cells in the library, in order to cover the functionality, power, and delay characteristics provided by the transistor level design space. The layout for the resynthesized windows is generated using a proprietary layout tool [12].

The local resynthesis step indicated in Figure 2 forms the core of the structural optimization technique and is discussed in Section 4. Section 3 briefly describes circuit and logic models used in the remainder of the paper. In section 5 we analyze the design space covered by local resynthesis and demonstrate that this is much larger compared to a typical cell-based synthesis. Section 6 shows some experimental results.

3. CMOS Logic and Circuit Models

3.1 Logic Representation

The SP-BDD [8] representation is an efficient means of capturing the logic functionality and the structure of static CMOS gates whose pull-up and pull-down structures are complementary series-parallel networks. In the remainder of this paper, we limit our focus to series-parallel CMOS structures. Figure 3 shows an example of CMOS gate and its gate-BDD. Each vertex of the gate-BDD corresponds to a gate input and a pair of transistors (one in pull-up and one in pull-down). A combinational circuit composed of CMOS gates may be represented as an ordered network of gate-BDDs.

3.2 Power and delay model

Since the SP-BDD captures the structural information of the gate, the delay and the power can be calculated directly from the SP-BDD representation. The SP-BDD representation allows the use of accurate transistor level models at gate level efficiency. The transistor level power model used in resynthesis is described in [9]. The delay of the gate is estimated using a proprietary semi-analytical piece-wise quadratic approximation of the MOS characteristics [11] for circuit timing during global resynthesis. During the local resynthesis however, since we need to evaluate only the relative merit of the windows, we use the well known Elmore delay model to evaluate the gate delay.

4. Resynthesis

4.1 Global Resynthesis

The overall global resynthesis flow is shown in Figure 2. As described inSection 2, global resynthesis is based on covering the entire circuit with a sequence of windows and performing detailed local resynthesis on each window. When comparing the costs of different structures for the same window, we consider timing, area and power appropriately depending upon the optimization objective. Timing is performed using static timing analysis techniques.Windows are sequentially selected to minimize timing recalculations. The circuit is covered in topological order from primary inputs to primary outputs. Consequently, to calculate old and new time slack correctly, we need to recalculate arrival / required times before and after every local resynthesis only for those gates that fall in the cone of influence of the window.

4.2 Local Resynthesis

Local resynthesis is an optimization procedure that minimizes the cost which is a function of area, delay, power, or any convex combination of these variables for a given window. In addition, user specified constraints such as a limit on the maximum stack depth, maximum number of transistors per stage or maximum fan out can all be considered during local resynthesis by assigning appropriate penalties when a violation of these constraints occurs. Local resynthesis is a sequence of the following basic steps:

- A. DeMorgan transform changes a gate to a complementary one and adds or deletes inverters on its inputs and output.
- B. Reordering changes places of two SP-networks connected in series in either pull-up or pull-down network of a gate.
- C. Decomposition step decomposes a complex gate into two subsequent gates with an inverter between them.

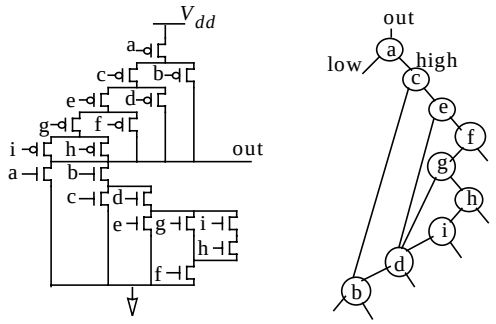

Figure 3. CMOS gate and corresponding gate-BDD.

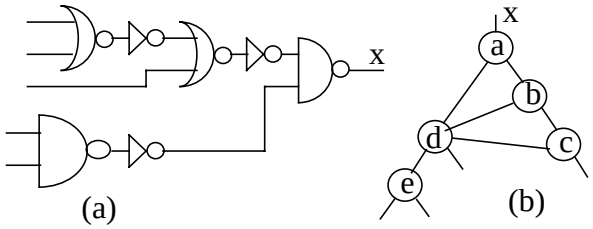

Figure 5. (a) Window state and (b). merged gate-BDD.

- D. Merging converts two subsequent gates with an inverter between them into one complex gate. If there is no inverter, then preliminary DeMorgan transform may be done for one of the two gates. The result of merging may be a gate with redundant structure. Therefore merging is followed by gate-BDD minimization.

Step A allows us to explore every possible structure of a window that can be reached by DeMorgan transforms on the given structure. This set of window structures obtained by closure under DeMorgan is referred to as a *superstate.* In addition, there are also some auxiliary steps:

- deleting double gates: deletes one of two identical gates with pairwise common inputs, this step is performed whenever possible;
- splitting a gate: inverse of deleting double gate, this step precedes merging when necessary;
- deleting buffers: deletes two subsequent inverters, this step is performed every time when possible for reducing power;
- inserting buffer: inverse of deleting buffer, this step is performed when it is necessary to reduce delay.

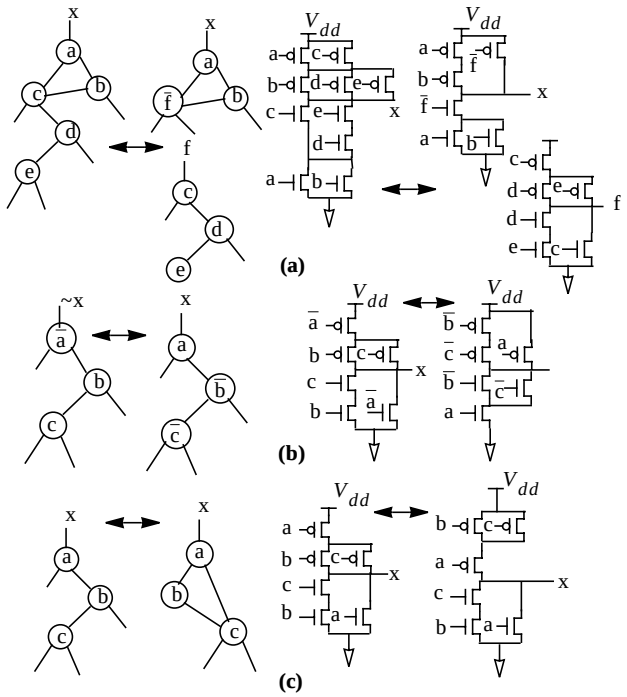

Figure 4. Examples of BDD transformations: (a) Merging and Decomposition–forward arrow indicates decomposition and the reverse arrow indicates merging. (b) DeMorgan transforms. (c) Reordering. Note that the re-ordering shown in (c) cannot be achieved by simple input swapping.

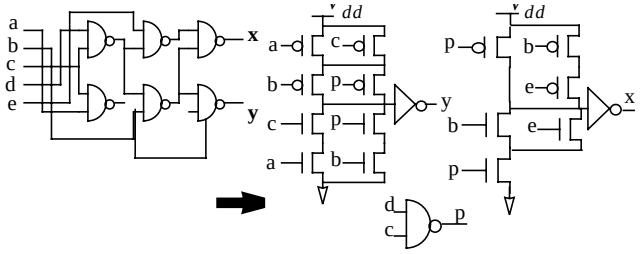

Figure 6. Initial circuit (c17) with 24 transistors and result of local resynthesis with 22 transistors.

To investigate each superstate for optimality, we use a simulated annealing procedure with an oscillating schedule [10] consisting of several iterations interposed with fully random steps and greedy steps. Every annealing step contains randomly (with proper weights) selected basic resynthesis steps (reordering, decomposition or merging) followed by a full superstate investigation and a fast sizing of the optimal structure. The best visited state during resynthesis is taken as the result of local resynthesis. The local resynthesis algorithm can be described by the following pseudo-code. The variables Dim, N_INF, and N_ZERO are calculated specifically for each window as explained in Section 5.

```
calculate Dim;
estimate N_INF and N_ZERO;
for (iter=0; iter<MAX_ITER; iter++) {
    for (i=0; i<N_INF+N_ZERO; i++) {
        select basic step type;
        select basic step object;
        make basic step;
        investigate superstate;
        fast sizing;
        if (i>=N_INF) {
            calculate cost;
            if (cost is increased)
                decline step result;
        }
    }
}
    take best solution;
```

An example of the result of local resynthesis is shown in Figure 6.

5. Design space for local resynthesis

In this section, we provide an overview of the design space explored during local resynthesis. The design space is much larger than that obtainable using library based synthesis. First we define several useful notions.

Definition 1. *A **window** is a connected subcircuit under local resynthesis, that is cut out from some larger circuit.*

Definition 2. *A **Superstate** is the set of window states that can be obtained from a single state by all possible DeMorgan transforms* (combined with inserting / deleting buffers).

Definition 3. *A **Merged gate-BDD** is a gate-BDD resulting from merging all structures in a window.* (For an arbitrary window we have a merged gate-BDD for every window output.) Figure 5 shows an example of a window state and the merged gate-BDD for this window. The gate-BDD is a single SP-BDD that represents the function performed by the entire window

and can represent all possible circuits that are obtainable from this circuit using DeMorgan transforms.

Definition 4. *An **SP-fragment** is either a single gate-BDD vertex, or a maximal series connection of two or more SP-fragments, or a maximal parallel connection of two or more SP-fragments*.For example in Figure 5b, the single vertices (a), (b), (c) are SP-fragments and the entire gate-BDD is also an SP-fragment. The series connection of vertices (a,b) is not an SP-fragment because this series structure is not maximal.

Definition 5. *A **SP-chain** is a series or parallel connection of two or more SP-fragments*. A gate-BDD with more than one vertex consists of a hierarchy of SP-chains. For example, the gate-BDD in Figure 5 contains two SP-chains–((a,b,c),(d),(e)) - 1st (top) level of hierarchy, and ((a),(b),(c)) - 2nd level of hierarchy.

Statement 1. The number of SP-chains in a gate-BDD is equal to the number of SP-fragments with length more than one, where the length of an SP-chain is the number of vertices on the chain.

Definition 6. An *SP-interval of a SP-chain* is one or more neighboring SP-fragments of the same SP-chain (but not the whole chain). For example in Figure 5b, (a), (a,b), (a,b,c,d), (d,e) are SP-intervals, while (c,d) is not a SP-interval.

Definition 7. *A **Marking of a SP-chain** is an assignment of a mark to each SP-interval (containing at least one vertex), satisfying the condition*:

- if A, B are SP-intervals with the same mark, then either A contains B, or B contains A, or A and B are non-intersecting.

Definition 8. A *Marking on gate-BDD* is a set of markings of all its SP-chains.

Statement 2. There is a bi-jection between all possible markings on a merged gate-BDD and all possible window superstates that can be obtained from the merged gate-BDD by decomposition steps.

The decomposition step of a gate-BDD discussed in section 4.2 is performed by replacing any SP-interval of the gate-BDD by a single new vertex. This removed SP-interval is a separate gate-BDD and connects the two gate-BDD's through an inverter. Thus the original gate-BDD is decomposed into three gate-BDD's (an inverter and two other gate-BDD's). This decomposition process can be repeated to obtain other structures from the original gate-BDD. For example, the circuit in the window state shown in Figure 5a can be obtained by first decomposing the SP-intervals (a,b,c) and (d,e) from the gate-BDD, and then further decomposing the sub SP-interval (a,b) from (a,b,c).

Statement 2 above indicates that the number of structures reachable from a window state structure is equal to the number of ways of marking a merged gate-BDD. Clearly, the number of markings is exponential resulting in an exponential solution space.

Again, we can shown that the reordering step in section 4.2 yields an exponential solution space. For a merged gate-BDD, for every SP-chain we can take all possible permutations of its SP-fragments. It is easy to show that permutations in any two SP-chains commute. It is also easy to show that for arbitrary window superstate there is an injection from the set of possible reordering to the set of reordering on a merged gate-BDD. Thus, a large number of new structures are also explored through reordering.

The design space for local resynthesis in terms of superstates can be represented by a Cartesian product of ***Dim=m+n*** one-dimensional subspaces, where ***m*** is the number of reordering subspaces and ***n*** is the number of merging / decomposition subspaces. For the example of Figure 5 we have ***Dim=2+2=4***. Finally, it should be noted that we use the following simple motivation for the simulated annealing procedure under use. Supposing that the design space consists of identical ***Dim***-dimensional cubes with local minimum of cost function inside each cube. By maximizing the expected number of cubes of finite length visited during annealing, we get the oscillating schedule as the optimal temperature schedule, with the following parameters for the local resynthesis algorithm:

N_INF = Kinf * sqrt(Dim), ***N_ZERO = Kzero * Dim***, where

Kinf, ***Kzero*** are empirical constants (nearly equal to 1–2).

6. Results

A prototype version of the resynthesis algorithm described above was implemented on a UNIX workstation. In this version the timing control and also some additional constraints (maximal gate size, maximal pull-up / pull-down path length, maximal fanout, etc.) can be switched on / off.

Figure 7 shows the area delay trade-off curve generated by resynthesis on a logic circuit consisting of about 120 gates (~ 500 transistors). The graph shows the point that corresponds to a solution generated by a commercial synthesis tool with settings to yield the maximum performance using a well designed fine grain library. The library consisted of a total of 150 cells with 4-5 strengths for each type of logic gate. Curve 1 in the graph is the area vs. delay trade-off of the same circuit generated using an optimal sensitivity based sizing algorithm with very accurate delay models. Typically on a critical path our delay model incurs no more 5% error on a gate-by-gate basis. Note that this sizing method produced solutions with about 15% improvement in timing for the same area or a 25% improvement in area for the same timing over the cell-based solution. This is not surprising since the sensitivity based method has the freedom to vary the sizes of every transistor in the circuit.

In the same graph, curve 2 demonstrates the gains due to resynthesis. Resynthesis is run on the same cell-based solution as before. Note that there is a significant improve-

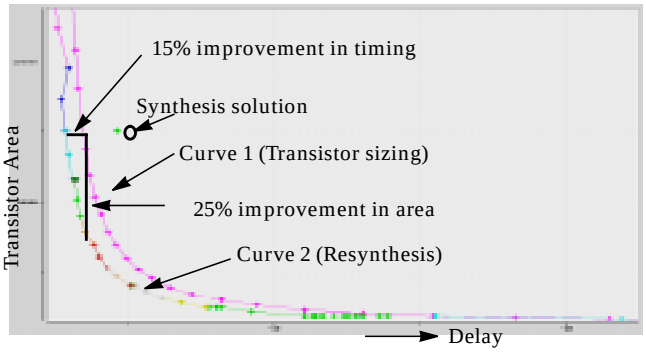

Figure 7. Results of resynthesis on a proprietary high performance circuit.

ment in area / performance of the resynthesized solutions even over the custom sized circuit. A 15% improvement is observed in timing at the same area while a 25% improvement is achieved in area at the same timing over transistor level sizing. Overall our technique improved the timing by 20% at the same transistor area and the area by about 37% for the same timing. We must also mention that there was about 20% reduction in the total number of transistors at the minimum area point. As we would expect once the circuit was sized up the original area point, total gain in transistor count fell to 5% due to addition of buffers for drive strength and implementation using smaller gates. The run time for this example was approximately 1hr for generating the entire curve on a Sparc–station 20 with 64 Mb of RAM.

Figure 8 shows the same results for a decoder circuit. This circuit has about 600 gates (2500 transistors) original synthesis was performed with a 400 cell library of which include a combination of nand gates, nor gates, and an extensive set of aoi's (and-or-invert) and oai's. Resynthesis yielded about 20% improvement in timing at the same area point and a 35% improvement in area at the same time point for this example over a cell-based design.This example took about 8hrs for the complete curve on the same machine.

Table 1 demonstrates the trade-off of performance and power obtained on 6 circuits. In this experiment we compare the inherent power–delay trade-off between a circuit obtained by library based synthesis and the corresponding circuit resynthesized for power. We select one solution from the area–delay trade-off space, and resize it to obtain the power–delay trade-off information for several delay constraints. Each of the sub columns show the different delay and power trade-offs for that circuit. In addition the rows 5 and 6 show the reduction in the number of transistors for each circuit. Observe that for the same delay, the resynthesized circuit consumes much less power than the original circuit from logic synthesis. Runtimes shown in the last column for generating the local curves were also reasonable as shown in the last row.

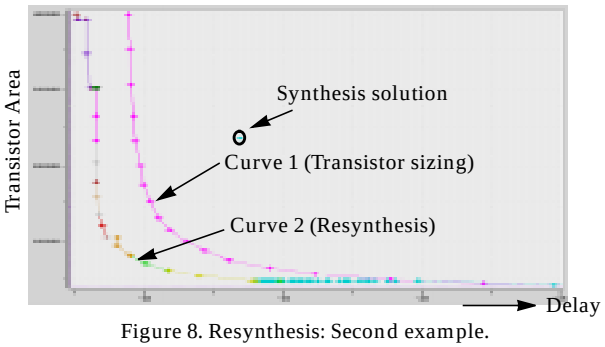

Figure 8. Resynthesis: Second example.

7. Conclusion

We have presented a new method to synthesize custom logic circuits. This technique could successfully improve the area-delay characteristics of the synthesized circuits. An analysis of the structural design space explored by this method is presented. Results show that transistor level resynthesis can yield performance improvements up to 20% with smaller transistor count and better power characteristics. In addition, results indicate that the overall delay-area characteristics of the circuits synthesized at transistor level are superior compared to circuits synthesized at gate level.

References

[1] S.Devadas, S.Malik. "A Survey of Optimization Techniques Targeting Low Power VLSI Circuits," DAC-95, p.242.

[2] S.Iman, M.Pedram. "Logic Extraction and Factorization for Low Power," DAC-95, p.248.

[3] R.E.Bryant. "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Trans on Computers, 1986, v.35, n.8, p.677.

[4] C.Y.Tsui, M.Pedram, A.M.Despain. "Technology Decomposition and Mapping Targeting Low Power Dissipation," DAC-93, p.68.

[5] J.P.Fishburn. "A Depth-Decreasing Heuristic for Combinational Logic; or How to Convert a Ripple-Carry Adder into a Carry-Look ahead Adder or Anything In-Between," DAC-90, p.361.

[6] V.Tiwari, P.Ashar, S.Malik. "Technology Mapping for Low Power," DAC-93, p.74.

[7] B.S.Carlson, S.J.Lee. "Delay Optimization of Digital CMOS VLSI Circuits by Transistor Reordering," IEEE Trans. on CAD, 1995, v.14, n.10, p.1183.

[8] A.L.Glebov, D.Blaauw, L.G.Jones. "Transistor Reordering for Low Power CMOS Gates Using SP-BDD Representation," Intern. Symp. on Low Power Design, 1995, p.161.

[9] S.Gavrilov, A.Glebov, S.Rusakov, D.Blaauw, L.Jones, G.Vijayan. "Fast Power Loss Calculation for Digital Static CMOS Circuits," European Design & Test Conf., 1997, p.411.

[10] K.D.Boese, A.B.Kahng, C.W.A.Tsao. "Best-So-Far vs. Where-You-Are: New Perspectives on Simulated Annealing for CAD," Euro-DAC'93, p.78.

[11] A. Dharchoudhury, S. M. Kang, K. H. Kim, and S. H. Lee, "Fast and Accurate Timing Simulation with Regionwise Quadratic Models for MOS," Int'l Conf on Computer Aided Design, Nov 1994, p. 190.

[12] Mohan Guruswamy, Robert L. Maziasz, Daniel Dulitz, Srilata Raman, Venkat Chiluvuri, Andrea Fernandez, and Larry G. Jones, "CELLERITY: A Fully Automatic Layout Synthesis System for Standard Cell Libraries," Design Automation Conf, Jun 1997, p. 327.

Table 1: Results of Resynthesis

Ckt	c1355			c1908			cla			cla1			cnt_0			cnt_1		
Constraint(ns)	27.46	21.99	17.77	32.34	24.86	23.13	22.97	14.70	10.21	24.57	15.73	10.7	24.01	15.37	11.86	23.76	15.15	11.49
P(nW) Before	592	629	698	640	654	717	367	378	471	358	370	494	85	93	138	81	88	137
P(nW) After	438	463	502	437	455	526	349	362	446	343	355	486	80	88	125	73	83	117
# trans (before)	2308			3482			1008			956			352			372		
# trans (after)	1904			1690			820			838			298			294		
Run Time (s)	500			639			181			198			95			71		