CYCLE TIME OPTIMIZATION BY TIMING DRIVEN PLACEMENT WITH SIMULTANEOUS NETLIST TRANSFORMATIONS

Hendrik Hartje[*], *Ingmar Neumann*[‡], *Dominik Stoffel*[‡], *Wolfgang Kunz*[‡]

[*]University of Potsdam
Fault Tolerant Computing Group
14415 Potsdam, Germany

[‡]University of Frankfurt
Electronic Design Automation Group
60054 Frankfurt am Main, Germany

ABSTRACT

We present new concepts to integrate logic synthesis and physical design. Our methodology uses general Boolean transformations as known from technology-independent synthesis, and a recursive bi-partitioning placement algorithm. In each partitioning step, the precision of the layout data increases. This allows effective guidance of the logic synthesis operations for cycle time optimization. An additional advantage of our approach is that no complicated layout corrections are needed when the netlist is changed.

1. INTRODUCTION

Cycle time optimization has become one of the most important issues for the design of highly integrated circuits. A lot of performance optimization techniques exist at all stages of the design flow. However, during the last couple of years it has become apparent that the exchange of technical data between the different design levels is insufficient in conventional design flows. Optimization algorithms often lack important information and therefore fail to exploit the full optimization potential of the circuit.

During logic synthesis the interconnect delay is only approximated by rough net models. Layout data is not available at this stage. However, with the advent of deep submicron technologies the appropriate consideration of geometrical circuit data to estimate interconnect delay has become key in circuit design. This is why performance optimization at the logic synthesis stage cannot be successful for deep submicron circuits. Errors in the approximation could result in a design far off from an optimal one. In order to better exploit the global optimization potential of a circuit it is unavoidable to improve the interaction between the logical and physical design stages.

Simple feedback loops between the stages of logic synthesis and layout generation are not sufficient. They do not guarantee convergence and tend to be very time-consuming. Only a more intricate combination of suitable algorithms for logic synthesis and layout generation can lead to effective solutions.

In this paper, we propose a new technique to directly integrate general Boolean netlist transformations into a timing-driven placement algorithm. By close interaction between logic synthesis and placement we obtain accurate data for the wire delays to select netlist transformations. Since logic transformations are performed *while* the placement is generated and not after its completion we can alter the logic circuit structure without additional layout correction steps.

2. PREVIOUS WORK

Recently, several approaches to improve the interaction between logic synthesis and physical design have been published [2, 9, 11, 14–16].

We can roughly divide these approaches into two classes. The first class of approaches is anchored in logic synthesis. No placement has yet been generated. The interconnect delay is estimated based on the netlist structure [16]. This approach permits full flexibility of the logic transformations and can exploit the complete spectrum of logic synthesis techniques. However, without any layout data it is not easy to get good approximations for the interconnect delays.

The second class of approaches starts with a physical design so that an accurate post-layout delay model is obtained. It is attempted to optimize the circuit by a restricted set of logical netlist transformations. These transformations can be applied before the routing process [15]. In this case the wire lengths are estimated from the placement. The other possibility is to apply the transformations in a routed design with all wire lengths known [9].

Performing logical transformations in a completed physical design is a delicate issue. Most approaches only use *local* transformations to keep tight control on the changes in the netlist. Transformations are often restricted to buffer insertion [14] or resynthesis of small parts of the circuit. The method described in [3] uses local decomposition and remapping as netlist transformation. Some approaches also use layout transformations like gate sizing [4] or wire sizing [5] that do not change the netlist. It is also possible to combine optimizing transformations with technology mapping [11, 13].

Jiang et al. [9] proposed a method for post-layout performance improvement. They start with a complete circuit layout. For the optimization algorithm they employ a redundancy addition and removal technique in combination with an engineering change order (ECO) layout tool.

In [15] Stenz et al. use global netlist transformations (signal substitutions) together with an iterative placement algorithm. They also start with a netlist after placement. For performance optimization they propose a two-phase algorithm. The first phase restructures the netlist by signal substitutions. In the second phase they legalize the perturbed placement by an iterative placement improvement algorithm.

The main disadvantage of all these methods starting with a complete layout description is that it is difficult to exploit the full Boolean optimization potential. If the circuit is changed greatly, the layout also changes significantly and convergence cannot be guaranteed. Additionally, after each transformation a legalization step is necessary to correct the layout. For this legalization step ECO-algorithms [9] or other placement improvement algorithms [15] are used.

0-7803-6685-9/01/$10.00©2001 IEEE

3. THE PROPOSED APPROACH

To avoid the disadvantages of the described approaches, we propose a new method to merge logic synthesis and physical design. Instead of performing logic transformations before or after placement we perform them *during* placement generation. We integrate netlist transformations in a recursive partitioning-based placement algorithm. All layout information is used at the moment it becomes available. During the first iterations of the recursive partitioning process our approach resembles the first class of methods described in the previous section. Only a global placement exists and the wire length estimations are still very rough. On the other hand, there is almost unlimited freedom to apply Boolean circuit transformations. As the procedure continues the circuit partitions become smaller and the performance estimation becomes more accurate. Boolean transformations are used incrementally to make corrections according to the refined timing model. In this way we attempt to combine the advantages of the approaches described in the previous section while avoiding some of their disadvantages.

In our previous work [12], we have already experimented with this paradigm and obtained promising results for speed optimization. However, in [12] *cell replication* was the only logical transformation being considered. Recently, in [6] it has been shown that a large variety of local timing optimizations such as cloning, remapping, gate sizing, buffer insertion and clock tree optimization can also be integrated into such a framework. Our goal is to demonstrate that general Boolean transformations as originally developed for technology-independent logic synthesis can also be performed during placement. We show that not only concepts for local timing correction are suitable for this framework but that it is beneficial to merge a general Boolean optimization phase into placement generation.

3.1. Outline

The overall algorithm is shown in Fig. 1. The algorithm starts with the netlist and a description of the standard cell library. In each level of the algorithm two steps are performed.

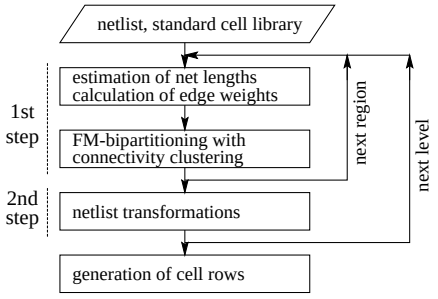


Figure 1: The overall placement algorithm

In the first step all regions containing more than one cell are bi-partitioned into two child regions using the well-known Fiduccia-Matthysis-algorithm [7]. As the child regions become smaller in each level the locations of the cells are determined with increasing precision and better estimations of the wire lengths can be obtained. The bi-partitioning algorithm is described in more detail in section 3.2.

In the second step, appropriate netlist transformations are performed based on the current estimation of wire lengths. These transformations are described in section 3.3.

As long as there exist regions containing more than one cell the algorithm proceeds to the next level. Finally the cells are arranged in rows. This changes the locations of the cells only marginally.

3.2. Partitioning-Based Timing-Driven Placement

In this section we briefly describe the timing-driven placement algorithm and the delay cost function which we use in our approach.

The placement algorithm starts with a netlist generated by a logic synthesis tool and the chip area that is available for placement. For the placement process we use a recursive bi-partitioning algorithm [1]. The algorithm iteratively bi-partitions so-called circuit *regions*. A region is a set of cells together with the chip area allocated for placing the cells. In each recursion *level* of the algorithm all current regions are bi-partitioned into two child regions [7].

Additionally we use a connectivity clustering algorithm [8] for improving both run time and quality of result. Before each bi-partitioning step, the algorithm selects cells to be merged into clusters according to their connectivity.

With every recursion level, the sizes of the regions decrease, so that in each step the individual cell positions can be determined with increasing accuracy. Recursive partitioning is performed until every region consists of a single cell. For the final placement the cells are arranged in rows, which changes the locations of the cells only slightly.

For partitioning, the circuit is represented as a weighted hypergraph. The weights of the hyper-edges represent the circuit delay and are calculated as follows:

Immediately before each bi-partitioning step, wire lengths are estimated based on the current cell locations. The position of a cell within a region is approximated by the region center point. As the region sizes decrease with each recursion level, the approximation becomes better. Using the approximate cell positions, wire lengths are estimated for calculating the wire capacitances.

Next, the *arrival time* and the *required time* are calculated for each signal using a static timing analysis. The arrival time for all primary inputs is set to 0. The *maximum path delay* is the largest arrival time among the primary outputs. It is used as the required time for every primary output. The *slack* of a signal is the difference between its required time and its arrival time. From the slack of a signal, an upper bound for the length of the corresponding wire can be calculated.

The ratio between the maximally allowable wire length and the minimally achievable wire length determines the edge weight used in the cost function of the min-cut bi-partitioning algorithm.

3.3. Cycle Time Optimization by Netlist Transformations

In order to be able to integrate general Boolean optimization into a placement algorithm, the logic transformations to be used must fulfill some important requirements.

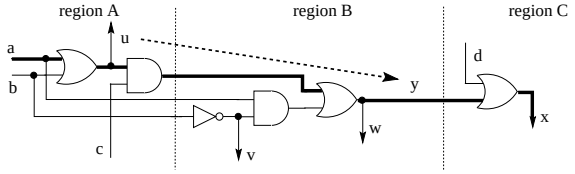
1. The optimization framework should not be restricted to technology-independent circuit descriptions, but should facilitate optimization of *mapped* netlists.
2. The transformations should make maximum use of the existing optimization potential, i.e., they should not be restricted to local netlist transformations.
3. Although the scope of the optimizations must not be local, the optimization process as a whole should be decomposable into a series of individual optimizing operations that each affect only a limited number of gates.

In order to meet these requirements, we use so-called *implicant-based* circuit transformations [10]. Each transformation consists of the following two steps:

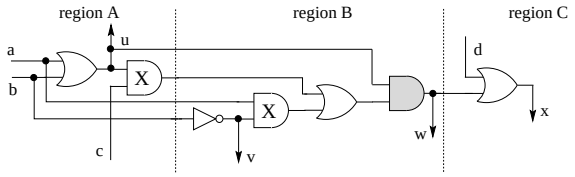
1. Calculation and insertion of implicants for a network function using an AND/OR reasoning technique called *recursive learning* [10].
2. Identification and removal of redundancy using ATPG.

With these two steps, the structure of the circuitry implementing an internal network function is modified, however, the logic function is not changed. Each transformation only affects a limited number of gates. However, as can be proved [10], arbitrary circuit transformations including the conventional synthesis techniques such as functional decomposition, kerneling, transduction, can be described using this two-step methodology. Implicant-based network transformations have already been applied very effectively in technology-independent multi-level logic optimization.

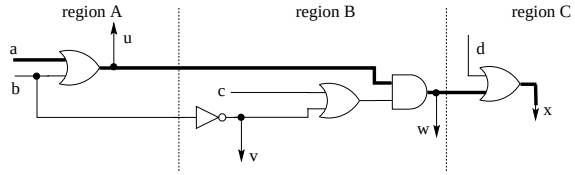
Fig. 2 shows an example how the delay of the critical path (shown as bold lines) can be reduced by implicant-based network transformations.



(a) Original circuit with implication



(b) Adding the implicant



(c) Circuit after redundancy removal

Figure 2: Example circuit

Fig. 2(a) shows the original circuit. In the first step recursive learning finds the implication $(u = 0) \rightarrow (y = 0)$. The implicant f is added to the function at y using an additional AND gate (Fig. 2(b)). The logic function of the circuit does not change by this modification.

In the second step, two gates are identified as redundant using ATPG and removed (labeled 'X' in Fig. 2(b)). The resulting circuit is shown in Fig. 2(c). As can be seen, this transformation has made the critical path significantly shorter.

In our approach, these implicant-based logic circuit transformations are tightly integrated into the placement algorithm (see Fig. 1). Fig. 3 shows the flowchart for the netlist transformation algorithm.

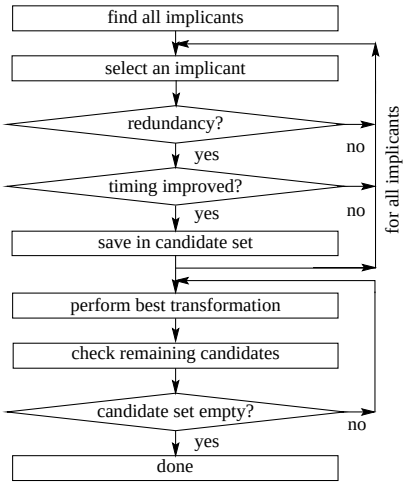


Figure 3: The netlist transformation algorithm

First, for every signal in the circuit a set of implicants is calculated and saved. Then, for each implicant, a circuit transformation following the above-mentioned two-step methodology is performed. By inserting the implicant, additional redundancy may be introduced which can occur anywhere in the circuit, not only in the immediate vicinity of the implicant. All redundant connections and gates are removed by ATPG-based redundancy elimination. It is this step by which a delay improvement may be achieved. If redundancy elimination removes gates or inputs to gates on the critical path, the cycle time of the circuit may be improved. For this reason, only transformations which yield redundancy are considered further. They are called transformation *candidates*.

Inserting an implicant either results in adding a single wire or in adding a new cell with additional wires. In the former case, the current placement is not changed. In the latter case, the new cell is assigned to a region such that the corresponding wires have minimal length.

In redundancy removal, wires as well as cells may be removed. Again, removal of a wire does not change the placement. However, if one or more cells have to be removed, they are also deleted from their partitions.

After the insertion or removal of cells, the cell area allocated for a partition does no longer correspond to the sum of the cell sizes in the partition. Therefore, after a transformation, the new region sizes have to be calculated. Note that this is the only layout correction step needed in our algorithm.

For each transformation candidate, a static timing analysis is performed and the maximum path delay is calculated. If a transformation reduces the maximum path delay, it is saved in the *candidate set*. From all transformation candidates, the one which yields the greatest improvement in cycle time is performed and removed from the set.

After a transformation has been performed, other transformations in the candidate set may become invalid, because the corresponding implications no longer exist or the corresponding gates have been removed from the circuit. Also, the cycle time may have changed. Therefore, after a transformation, all remaining candidates are checked for validity and are removed if invalid. Then, using static timing analysis, the cycle time improvement is recalculated for the valid candidates. Transformation candidates yielding no improvement are also removed from the set.

This loop is iterated until no more candidates are left that improve the circuit performance. The placement algorithm then continues with the next recursion level. Note that in each recursion level, new performance-improving transformation candidates may be found, because cell positions have changed resulting in new estimations for wire capacitances and path delays.

4. EXPERIMENTAL RESULTS

To evaluate our new approach, we performed the following three experiments with combinational ISCAS'85 benchmark circuits. For physical design we used a $0.18\mu m$ CMOS library, which contains all standard gates. Complex gates are not used in our current implementation, but our method can be easily extended to use all kinds of gates. The results of the experiments are given in table 1.

In the first experiment we generated a placement from the original netlist. We used the timing driven bi-partitioning algorithm described in section 3.2 with a static netlist. This experiment is denoted by EX1. In the second experiment we performed the implicant-based logical netlist transformations of section 3.3 to optimize the cycle time *before* the placement. Because in this logic optimization phase no layout data exist, the algorithm does not use any information about the net delay. After the logic optimization we performed timing driven placement of the optimized netlist. This experiment is called EX2 in the table. In the last experiment we used our new method with logic transformations during the timing driven placement generation. This experiment is labeled EX3.

All delay results are calculated at the end of the placement process. The cycle time is calculated from the estimated net lengths based on the final placement. Routing algorithms are not considered in this paper.

	EX1	EX2		EX3	
	no opt.	conventional		proposed	
Circuit	delay [ns]	delay [ns]	impr. [%]	delay [ns]	impr. [%]
C432	4.443	4.639	-4.4	3.756	15.5
C499	4.060	3.687	9.2	3.689	9.1
C880	3.339	3.553	-6.4	3.167	5.2
C1355	3.648	3.498	4.1	3.296	9.6
C1908	5.059	5.317	-5.1	4.490	11.2
C2670	6.018	3.855	35.9	3.700	38.5
C3540	6.953	6.612	4.9	5.739	17.4
C6288	16.131	15.420	4.4	15.242	5.5
Avg.			5.3		14.0

Table 1: Experimental results

The results show that our new approach can reduce the cycle time for all benchmark circuits. On the average, a reduction of 14% is achieved. With the second experiment we demonstrated that the cycle time improvement is much smaller if we perform the Boolean transformations *before* the layout process not using any layout data. For some circuits the cycle time even increases in this experiment because wrong decisions are made during logic synthesis without the correct delay data from the layout.

5. CONCLUSIONS

In this paper, we presented a new approach to integrate general Boolean transformations into a placement algorithm. With our placement framework we show that network transformations originally developed for technology-independent logic synthesis can also be performed for optimization during placement. By this interaction between logic synthesis and placement we can use the layout data at the moment it becomes available.

Compared to the performance-driven placement of the original netlist, we can reduce the cycle time of the benchmark circuits by 14% on the average.

In our experiments, the same logic transformations without any layout data could not improve the circuit cycle time considerably. This shows that a close interaction between logic synthesis and layout generation is indeed necessary.

6. REFERENCES

[1] M. A. Breuer. A class of min-cut placement algorithms. In *Design Automation Conference (DAC)*, pages 284–290, 1977.

[2] S.-C. Chang, K.-T. Cheng, N.-S. Woo, and M. Marek-Sadowska. Postlayout logic restructuring using alternative wires. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(6):587–596, June 1997.

[3] C. Changfan, Y.-C. Hsu, and F.-S. Tsai. Timing optimization on routed designs with incremental placement and routing characterization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(2):188–196, February 2000.

[4] W. Chen, C.-T. Hsieh, and M. Pedram. Simultaneous gate sizing and placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(2):206–214, February 2000.

[5] C.C.N. Chu and D.F. Wong. A new approach to simultaneous buffer insertion and wire sizing. In *International Conference on Computer Aided Design (ICCAD-97)*, pages 614–622, November 1997.

[6] W. Donath, P. Kudva, L. Stok, P. Villarrubia, L. Reddy, A. Sullivan, and K. Chakraborty. Transformational placement and synthesis. In *Design Automation and Test in Europe (DATE-2000)*, pages 194–201, 2000.

[7] C.M. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference (DAC)*, pages 175–181, July 1982.

[8] S. Hauck and G. Borriello. An evaluation of bipartitioning techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 16(8):849–866, August 1997.

[9] Y.-M. Jiang, A. Krstic, K.-T. Cheng, and M. Marek-Sadowska. Postlayout logic restructuring for performance optimization. In *34th Design Automation Conference*, pages 662–665, Anaheim, CA, USA, June 1997.

[10] W. Kunz and D. Stoffel. *Reasoning in Boolean Networks.* Kluwer Academic Publishers, 1997.

[11] A. Lu, H. Eisenmann, G. Stenz, and F. M. Johannes. Combining technology mapping with post-placement resynthesis for performance optimization. In *IEEE International Conference on Computer Design (ICCD)*, 1998.

[12] I. Neumann, D. Stoffel, H. Hartje, and W. Kunz. Cell replication and redundancy elimination during placement for cycle time optimization. In *International Conference on Computer Aided Design (ICCAD)*, pages 25–30, San Jose, November 1999.

[13] M. Pedram and N. Bhat. Layout driven technology mapping. In *28th Design Automation Conference*, pages 99–105, 1991.

[14] K. Sato and M. Kawarabayashi. Post-layout optimization for deep submicron design. In *33rd Design Automation Conference (DAC)*, pages 740–745, Las Vegas, NV, USA, June 1996.

[15] G. Stenz, B. M. Riess, B. Rohfleisch, and F. M. Johannes. Performance optimization by interacting netlist transformations and placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(3):350–358, March 2000.

[16] H. Vaishnav and M. Pedram. Logic extraction based on normalized netlengths. In *IEEE International Conference on Computer Design (ICCD)*, pages 658–663, October 1995.