A Timing-Driven Soft-Macro Resynthesis Method in Interaction with Chip Floorplanning

Hsiao-Pin Su[1,2], Allen C.-H. Wu[1] and Youn-Long Lin[1]
[1]Department of Computer Science, Tsing Hua University, Hsin-Chu, Taiwan, ROC
[2]Taiwan Semiconductor Manufacturing Company, Ltd., Hsin-Chu, Taiwan, ROC

Abstract

In this paper, we present a complete chip design method which incorporates a soft-macro resynthesis method in interaction with chip floorplanning for area and timing improvements. We develop a timing-driven design flow to exploit the interaction between HDL synthesis and physical design tasks. During each design iteration, we resynthesize soft macros with either a relaxed or a tightened timing constraint which is guided by the post-layout timing information. The goal is to produce area-efficient designs while satisfying the timing constraints. Experiments on a number of industrial designs have demonstrated that by effectively relaxing the timing constraint of the non-critical modules and tightening the timing constraint of the critical modules, a design can achieve 13% to 30% timing improvements with little to no increase in chip area.

1 Introduction

Over past decades, academia and industry have invested much effort in physical design related research, including floorplanning, partitioning, placement, and routing. Several excellent reviews of physical design techniques are given by [1, 2, 3]. By integrating various techniques, many design methods and software systems have been developed for chip designs. One of the most popular design methods uses schematics as the design entry, followed by floorplanning, placement, and routing to produce final chip layouts. This design method is very effective and efficient on small to medium-scaled designs. However, with the advent of deep-submicron technology, more and more devices can be packed into a very complex single chip. Due to the time-to-market pressure of designing complex chips and the maturity of synthesis tools, more and more integrated-circuit designers use an HDL-based synthesis approach to develop and manage large designs. Furthermore, as devices geometries shrink, a new set of design challenges, especially in electrical characteristics of circuits, are faced by integrated-circuit designers. This has led to a new research direction in design automation at synthesis and physical levels.

A typical HDL-based design flow involves multi-level design tasks. Over the years, much effort has been invested to improve the quality of design tasks at each design level. Few studies have been conducted to investigate the interaction between different design tasks. Pedram and Bhat [4] presented technology mapping techniques by considering net lengths for area and delay optimization. Liu et al. [5] presented a resynthesis technique that resynthesizes the most congested region of the chip to reduce routing area. Stenz et al. [6] proposed a timing-driven placement method in interaction with netlist transformations. The netlist transformation procedure is integrated into the placement process so that accurate delay models are available to guide the transformation process. Their results showed that delay reduction is achieved with almost no increase in chip area. Holt and Tyagi [7] proposed an integrated approach that incrementally develops a placement during the logic synthesis process for power minimization.

In this paper, we present a complete chip design method which incorporates a floorplanning-guided soft-macro resynthesis method for area and timing improvement. The main objective is to develop a timing-driven design flow by exploiting the interaction between HDL synthesis and floorplanning design tasks. Experiments on a number of industrial designs have been conducted to demonstrate the effectiveness of the proposed method.

2 Problem Description

Figure 1(a) shows a typical HDL-based chip design flow. It consists of five steps: (1) HDL synthesis, (2) floorplanning, (3) place and route, (4) back annotation, and (5) post-layout timing analysis. The inputs to the design flow is a mixed RTL and gate-level HDL description in Verilog or VHDL, and a timing constraint. In the first step, a synthesizer converts an HDL design description into a hierarchical gate-level netlist by performing HDL compilation and a series of RTL and logic synthesis tasks. In the second step, a floorplanning procedure is invoked to determine the location of each macro on the layout plane. In the third step, a placement-and-routing procedure is used

[†]This work was supported in part by the National Science Council of R.O.C. under Grant NSC 88-2215-E-007-030 and NSC 86-2215-E-007-024.

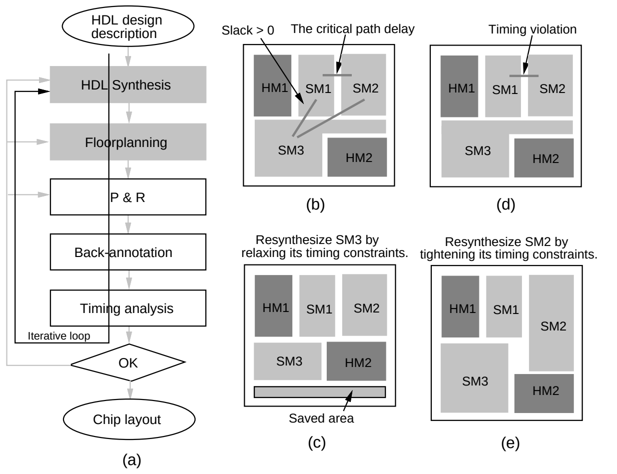

Figure 1: A typical HDL-based method for macro-based designs: (a) the design flow, (b) design with positive slack values, (c) design with timing-relaxed resynthesis, (d) design with timing violation, (e) design with timing-tightened resynthesis.

to perform detailed gate-level placement and routing. In the fourth step, the layout parasitic information is extracted. Finally, a post-layout timing analysis procedure is performed to determine the most critical paths and their delays. If the timing does not satisfy the design requirement, a refinement iteration will proceed until the timing requirement is satisfied. The refinement procedure can be applied at different design levels. This motivates us to investigate how to develop a complete chip design methodology by integrating multi-level design tasks and exploiting the interaction between them.

In this study, we focus on developing a complete chip design methodology which incorporates a soft-macro resynthesis method in interaction with chip floorplanning for area and timing improvements. The main objective of this research is to develop a timing-driven design flow by exploiting the interaction between HDL synthesis and physical design tasks, as depicted in Figure 1(a). Consider an example which consists of five macros, two hard macros and three soft macros. Initially, each soft macro is synthesized into a gate-level netlist. After the floorplanning, place-and-route, and post-layout timing analysis, there are two possible cases. Figure 1(b) shows the first case in which the design satisfies the timing constraint and the critical path occurs between soft macros $SM1$ and $SM2$. Consider that the slack between $SM3$ and $\{SM1,SM2\}$ is larger than zero. This indicates that we may have provided an excessive timing constraint to $SM3$ during the synthesis process. In this case, we can resynthesize $SM3$ with a relaxed timing constraint which usually results in a more area-efficient design, as depicted in Figure 1(c). Figure 1(d) shows the second case in which a timing violation occurs between $SM1$ and $SM2$. This indicates that we may have provided an under-estimated timing constraint to either $SM1$ or $SM2$ during the synthesis process. In this case, we may have to resynthesize $SM2$ with a tightened timing constraint which can produce a timing-violation free design but costs some area overhead, as depicted in Figure 1(e). The goal is to produce the most area-efficient design while satisfying the timing constraints.

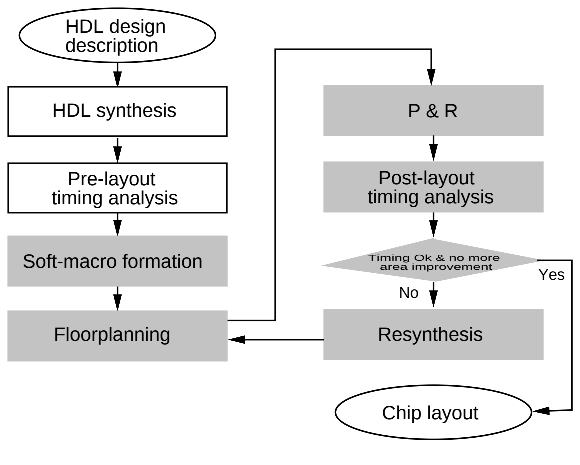

Figure 2: The proposed design flow.

3 The Proposed Method

3.1 Overview

Figure 2 depicts the proposed design flow which consists of seven steps: (1) HDL synthesis, (2) pre-layout timing analysis, (3) soft-macro formation, (4) floorplanning, (5) place and route, (6) post-layout timing analysis, and (7) resynthesis. The input to the design flow is an RTL design description in Verilog. In the first step, an HDL-based synthesizer converts the Verilog design description into a hierarchical gate-level netlist. In the second step, a timing analysis procedure is applied to perform pre-layout timing analysis of the design. A set of critical paths will be identified and used to guide the following macro-clustering, floorplanning, and placement-and-routing procedures. In the third step, the system groups soft macros connected to the same clock sources into the same cluster. Furthermore, it also groups small subcircuits to form large macros and decomposes extremely large macros into smaller ones. In the fourth step, we use a commercial floorplanner to perform macro floorplanning to determine the locations of macros. In the fifth step, we use a commercial tool to perform placement and routing tasks. In the sixth step, a post-layout timing analysis procedure is invoked to compute the final timing of the design. If there exits a timing violation or there is a chance for area reduction, a soft-macro resynthesis procedure is invoked. The system iterates four to the final step until all the timing constraints are satisfied and no more area improvement can be achieved.

In the following sections, we will describe the soft-macro formation (step 3) and the soft-macro floorplanning and resynthesis loop (steps 4-7) in details.

3.2 Soft-Macro Formation

There are two main considerations in soft-macro formation. First, in many of today's applications, such as multimedia chips, designs usually have multiple clock sources with different rates. It is beneficial to group soft macros associated with the same clock source into the same cluster. Second, using an HDL-based synthesis method, the synthesized subcircuit of each leaf module is naturally a closely-connected cluster. However, a design may also contain extremely large modules containing tens of thousands of gates. This is undesirable because a large cluster is too rigid for macro placement and may often result in poor placement results. Furthermore, a design may also contain a large number of small subcircuits. This is also undesirable because a large number of macros will increase the computational complexity of the macro-cell placement process.

The soft-macro formation procedure consists of three steps: (1) clock-based clustering, (2) large-macro decomposition, and (3) small-macro clustering. In our approach, we first use a commercial synthesis system to convert a Verilog design description into a hierarchical gate-level netlist. We then construct an HDL-based structural tree to represent the structural hierarchy of the Verilog design description. In an HDL structural tree, the root node represents the top design, and each intermediate node represents a module construct. Each leaf node represents a circuit block generated from a leaf module.

After constructing the HDL structural tree of a design, we first groups the macros connected to the same clock source into the same cluster. Then we determine the large-macro candidates which need to be decomposed into smaller ones. The selection of large-macro candidates is based on the size of the macros. We define the threshold value M_{th} of a large-macro candidate as:

$$M_{th} = k * S_{avg}, \tag{1}$$

where S_{avg} is the average macro size $\frac{\#Total\ cells}{\#Macros}$, $\#Total\ cells$ and $\#Macros$ are the total number of cells and the number of soft macros in the design, respectively, and k is a user-defined threshold parameter for controlling the size of the large-macro. If a macro is larger than M_{th}, then it is selected as a large-macro candidate. For each large macro, we use the FM partitioning method [8] to recursively decompose large macros into smaller clusters.

Finally, we use a clustering algorithm [9] to group small macros into large ones based on the size constraint, and the criticality and connectivity between macros. Let $G = \{V, E\}$ be the connected graph where V is the set of macro nodes and E the set of edges. An edge e_{ij} exists if there exists at least a signal flow between macros v_i and v_j. A weight is associated with each edge indicating the number of connections between two corresponding macros. We define the connectivity $Conn_{ij}$, the criticality $Crit_{ij}$, and the closeness C_{ij} of two macros, v_i and v_j, as below.

$$Conn_{ij} = \begin{cases} \left(\frac{w_i+w_j}{w_i+w_j-2w_{ij}}\right) \times \left(\frac{s_i+s_j}{s_{th}}\right), & \frac{s_i+s_j}{s_{th}} \le 1; \\ 0, & \frac{s_i+s_j}{s_{th}} > 1, \end{cases} \tag{2}$$

$$Crit_{ij} = \begin{cases} 1, & \textbf{if } Crit\_Path(v_i \Leftrightarrow v_j) \textbf{ and } \frac{s_i+s_j}{s_{th}} \le 1; \\ 0, & \textbf{else}, \end{cases} \tag{3}$$

$$C_{ij} = \alpha\ Conn_{ij} + \beta\ Crit_{ij}, \tag{4}$$

where w_i denotes the total connection weight of v_i, w_{ij} the total connection weight between v_i and v_j, s_i the size of v_i, s_{th} the upper bound on the size of a cluster set by the user, $Crit\_Path(v_i \Leftrightarrow v_j)$ denotes that there is a critical path traveling across v_i and v_j, and α and β are two coefficients set by the user. In order to eliminate small macros and prevent the formation of large clusters, the user can set the upper bound on the size of a cluster. When the size of a new macro formed by merging two macros is larger than the upper bound, the closeness value between these two macros is 0.

3.3 Soft-Macro Resynthesis in Interaction with Floorplanning

After forming soft-macro clusters, a timing-driven floorplanning procedure [10] is invoked to determine the relative location of each macro (hard and soft macros) on the layout plane. We then use a commercial tool to perform placement and routing design tasks. Subsequently, we back-annotate RC parasitic values of the layout and perform post-layout timing analysis. Finally, we determine whether some soft macros can be resynthesized to achieve timing and/or area improvements.

The key issues for the resynthesis process are twofold. First, how to determine which soft macro should be resynthesized. Second, if a soft macro needs to be resynthesized, to what extent can its timing constraint be relaxed or tightened. Our resynthesis procedure consists of twp steps: (1) slack computation and (2) soft-macro resynthesis candidate selection.

In the first step, we start by back-annotating the delay information for each I/O port of soft macros. The delay information is extracted from a post-layout timing report. We then compute the slack value for each inter-macro signal path. Finally, we assign a slack value for each I/O port of soft macros. This value is computed using the following formula: $Slack(p_i) = MIN\{Slack(e(p_i, p_j)), p_i \in SM_k\ and\ p_j \in SM_l\}$, where $e(p_i, p_j)$ denotes the interconnection between ports p_i and p_j, and SM_k and SM_l denote two soft macros.

The slack of an I/O port is defined as the minimum slack value of all the signal paths associated with this I/O port.

In the second step, we use two cost functions to determine which soft macro should be resynthesized next so that maximal area and/or timing improvement can be achieved. The cost functions $POS(SM_i)$ and $NEG(SM_i)$ are defined as the sum of the positive and negative slack values of all the I/O ports in a soft macro:

$$POS(SM_i) = \sum Slack(p_j), for\ all\ Slack(p_j) > 0. \tag{5}$$

$$NEG(SM_i) = \sum Slack(p_j), for\ all\ Slack(p_j) < 0. \tag{6}$$

If there exists a negative slack value associated with any soft macro, a timing violation occurs. In this case, we select the soft macro with the highest $NEG(SM_i)$ as the candidate for resynthesis because it should be the most critical one. If all timing satisfies the timing constraint, we select the soft macro with the highest $POS(SM_i)$ as the candidate for resynthesis because resynthesizing it with relaxed timing constraints should result in a maximal area reduction. After selecting a candidate, we use a commercial synthesis tool to resynthesize the soft macro by specifying the I/O-ports' timing constraints according to their slack values. Subsequently, we invoke a floorplanning procedure to adjust the chip floorplan by preserving the original relative locations of all soft and hard macros.

The proposed timing-driven soft-macro resynthesis method ($TDSR$) is described below:

```
**Procedure_TDSR**($D_{hdl}$,$T_{const}$)
**begin**
  $D_{gate}$ ← HDL_Synthesis($D_{hdl}$);
  Pre_layout_timing_analysis($D_{gate}$);
  $S_{tree}$ ← Structural_tree_construction($D_{gate}$);
  Soft_macro_formation($S_{tree}$);
  Floorplanning($S_{tree}$,$T_{const}$);
  Place_route($D_{gate}$,$T_{const}$);
  RC_extraction($D_{gate}$);
  Post_layout_timing_analysis($D_{gate}$);
  Slack_computation($S_{tree}$,$D_{gate}$);
  **while** (timing constraint is violated or more area
        can be reduced)
  **begin**
    $POS(SM_i)$_and_$NEG(SM_i)$_computation($S_{tree}$);
    $SM_i$ ← Soft_macro_candidate_selection($S_{tree}$);
    $S_{tree}$ ← HDL_Synthesis($SM_i$);
    Floorplanning($S_{tree}$,$T_{const}$);
    Place_route($D_{gate}$,$T_{const}$);
    RC_extraction($D_{gate}$);
    Post_layout_timing_analysis($D_{gate}$);
    Slack_computation($S_{tree}$,$D_{gate}$);
  **end_of_while**
**end_of_procedure**
```

The inputs to the system include an HDL design description (D_{hdl}) and timing constraints (T_{const}). Let D_{gate} and S_{tree} denote the gate-level design and structural tree. Initially, the system performs HDL synthesis, pre-layout timing synthesis, structural-tree construction, soft-macro formation, floorplanning, place-and-route, and post-layout timing analysis to produce an initial chip layout. During the resynthesis iteration, the system first computes the slack value for each soft macro I/O port, and then computes for each soft macro. Following, the system selects one soft-macro candidate which contributes the most in timing or area improvement. After resynthesizing the soft macro, the system performs floorplanning adjustment, followed by RC parasitic extraction and post-layout timing analysis. Finally, if there is improvement, then the resynthesis iteration continues. Otherwise, the system stops and reports the final chip layout.

4 Experiments

We have tested the proposed method on three industrial designs. All three designs are described as hierarchical, mixed RTL and gate-level netlists in Verilog. Table 1 shows the characteristics of the designs in which $Nets$, IOs, HMs, $SMs(Before/After)$, $SMs(Cells/Gates)$, and $Total\ Gates$ denote the number of nets, IO pins, hard macros, soft macros before and after performing clustering, cells/gates of soft macros, and total gate-count of the design. In the three designs, $ind1$ contains three clock sources, and $ind2$ and $ind3$ contain two clock sources. In all experiments, we set the threshold values $k = 2$ for large-macro decomposition and $S_{th} = 0.1\ S_{avg}$ for small-macro clustering.

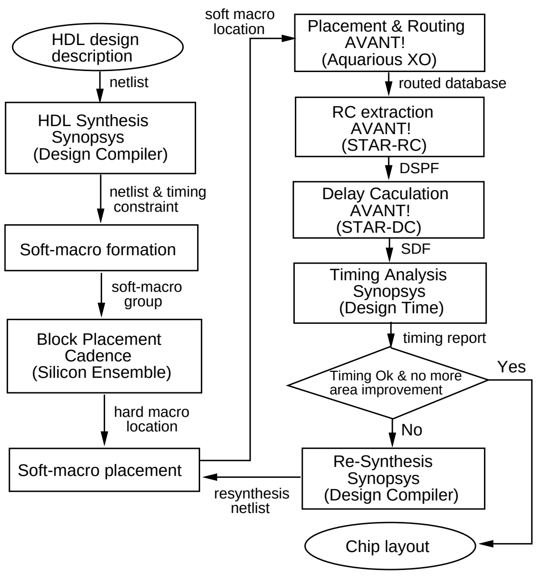
HDL design description
netlist
HDL Synthesis Synopsys (Design Compiler)
netlist & timing constraint
Soft-macro formation
soft-macro group
Block Placement Cadence (Silicon Ensemble)
hard macro location
Soft-macro placement
soft macro location
Placement & Routing AVANT! (Aquarious XO)
routed database
RC extraction AVANT! (STAR-RC)
DSPF
Delay Caculation AVANT! (STAR-DC)
SDF
Timing Analysis Synopsys (Design Time)
timing report
Timing Ok & no more area improvement
Yes
No
Re-Synthesis Synopsys (Design Compiler)
resynthesis netlist
Chip layout

Figure 3: The experimental flow.

Figure 3 shows the experimental flow. In the first step, we used Synopsys' *Design Compiler* [11] to convert the input Verilog design description into a hierarchical gate-level netlist and then performs timing analysis to report the 200 most critical paths. In the second step, we used our proposed soft-macro formation method as a pre-processing step to generate soft-macro clusters for the floorplanner. In the third step, we used Cadence's *Silicon Ensemble* [12] to perform chip floorplanning and determine hard-macros' locations. In the fourth step, we used the performance-driven soft-macro placement algorithm [10] to perform soft-macro placement. In the fifth step, we used AVANT!'s *Aquarious XO* [13] to perform detailed placement and routing. In the sixth step, the AVANT!'s $STAR \Leftrightarrow RC$ [14] was used to extract layout parasitic information. In the seventh step, we used AVANT!'s $STAR \Leftrightarrow DC$ tool [15] to perform delay calculations and generate an SDF file. In the eighth step, we used Synopsys' *Design Time* [11] to perform post-layout timing analysis. Finally, we applied the proposed soft-macro resynthesis iteration to incrementally improve the area and timing of the layout. During each resynthesis iteration, we first used Synopsys's *Design Compiler* to perform logic resynthesis by supplying a relaxed or a tightened timing constraint to the soft macros. We then applied an ECO function supported by AVANT!'s *Aquarious XO* to perform placement and routing. For all experiments, we provided the floorplanner (the third

Table 1: Characteristics of the benchmarking designs.

Designs	Nets	IOs	HMs	SMs(Before/After)	SMs(Cells/Gates)	Total Gates
Ind1	15,373	83	13	157/22	15,086/38,240	75,000
Ind2	27,404	155	8	150/28	42,030/75,361	95,000
Ind3	53,344	73	31	292/50	45,378/124,180	230,000

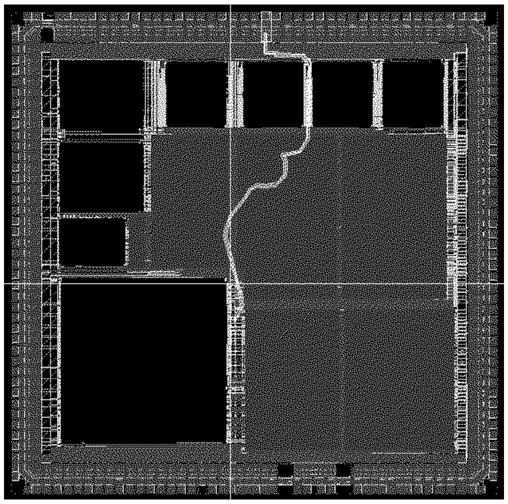
Figure 4: The initial critical path of *ind*2 using the $0.5\mu m$ library.

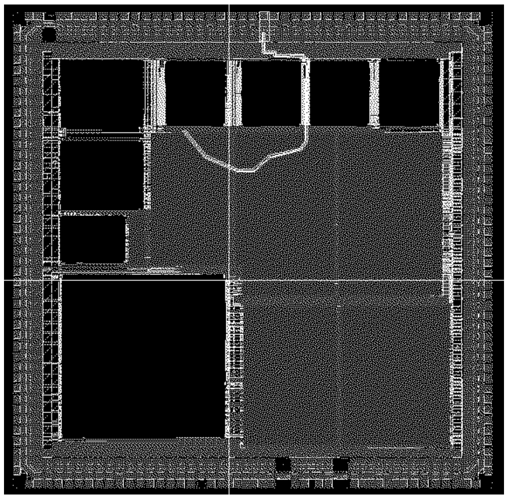
Figure 5: The new critical path of *ind*2 using the $0.5\mu m$ library after two resynthesis iterations.

step) and the placer-and-router (the fifth step) with the most critical 200 paths (generated in the first step) as the timing constraints.

We have conducted two sets of experiments. In the first experiment, we used the TSMC $0.5\mu m$ cell library [16]. In the second experiment, we used the TSMC $0.25\mu m$ cell library [17]. Note that *ind*1 and *ind*3 contain a PLL module. Unfortunately, the $0.25\mu m$-based PLL module is not available and we could not perform the experiment on the both designs using the TSMC $0.25\mu m$ cell library. Hence, in this paper, we only report the result of *ind*2 using the TSMC $0.25\mu m$ cell library.

Table 2 shows the area-delay comparisons of *ind*1 using the $0.5\mu m$ library, in which IO denotes the number of IO pins, $\#HM$ the number of hard macros, $\#SM(B/A)$ the number of soft macros before and after applying the soft macro clustering, $Gate_{SM}$ the total gate count of soft macros, $Gate_{Tot}$ the total gate count, $Area$ the chip area, $Delay$ the worst path delay, T_{resyn} the resynthesis run times in hours, and T_{eco} the ECO run times. The results show that by resynthesizing some soft macros, the timing was improved up to 20% with almost no area penalty. Table 3 shows the area-delay comparisons of *ind*2 using the $0.5\mu m$ library. We obtained the same result as that of *ind*1, in which the timing was improved up to 13% with almost no area penalty. Table 4 shows the area-delay comparisons of *ind*3 using the $0.5\mu m$ library. The results show that the timing was improved up to 11% with almost no area penalty. Figure 4 shows the critical path before resynthesis (*Iteration* 1 in Table 3). After two resynthesis iterations, the new critical path is shown in Figure 5 (*Iteration* 3 in Table 3). Table 5 shows the area-delay comparisons of *ind*2 using the $0.25\mu m$ library. The results show that the timing was improved up to 30% with 11% area penalty.

We have also compared the average delays contributed by gates and interconnects using $0.5\mu m$ and $0.25\ \mu m$ technologies. Table 6 shows the average gate and interconnect delay comparisons of the most critical paths of *ind*2. The results show that using the $0.5\mu m$ technology the average gate's intrinsic delay and interconnect delay are $0.171ns$ and $0.277ns$, respectively. In addition, using the $0.25\mu m$ technology, the average gate's intrinsic delay and interconnect delay are $0.107ns$ and $0.325ns$, respectively. From the results, we observed that the average interconnect-delay vs. gate-delay ratios of the $0.5\mu m$ and $0.25\mu m$ technologies are 1.62 and 3.04. This indicates that interconnect delays play an important role in deep-submicron technologies.

From the experiments, the following observations can be made. When using the $0.5\mu m$ library for designs *ind*1, *ind*2, and *ind*3 our proposed method can improve timing from 11% to 20% with almost no area penalty. This demonstrates that by effectively relaxing the timing constraints of non-critical modules and tightening the timing constraints of the critical modules we can achieve significant timing improvements with little to no increases in chip area. When using the $0.25\mu m$ library, our method can improve timing by 8%, 22%, and 30% with 2%, 5%, and 11% increase in chip area, respectively. We found that the $0.25\mu m$ library supports a large set of components with a wide range driven capability. This feature provides more design alternatives during the synthesis process.

Table 2: The area-delay comparisons of *ind*1 using the 0.5μm library

Iter	IO	#HM	#SM(B/A)	$Gate_{SM}$	$Gate_{Tot}$	Area(μm^2)	Delay(ns)	T_{resyn}(hr)	T_{eco}(hr)
1	83	13	157/22	38,240	75,000	25,250,625	18.35	6	4
2	83	13	157/22	38,279	75,039	25,251,118	15.87	5	4
3	83	13	157/22	38,260	75,020	25,252,218	13.91	4	3

Table 3: The area-delay comparisons of *ind*2 using the 0.5μm library

Iter	IO	#HM	#SM(B/A)	$Gate_{SM}$	$Gate_{Tot}$	Area(μm^2)	Delay(ns)	T_{resyn}(hr)	T_{eco}(hr)
1	155	8	150/28	75,361	95,000	27,957,500	38.25	9	7
2	155	8	150/28	75,501	95,140	28,037,599	33.71	8	5
3	155	8	150/28	75,533	95,172	28,039,765	33.30	6	4

Table 4: The area-delay comparisons of *ind*3 using the 0.5μm library

Iter	IO	#HM	#SM(B/A)	$Gate_{SM}$	$Gate_{Tot}$	Area(μm^2)	Delay(ns)	T_{resyn}(hr)	T_{eco}(hr)
1	73	31	292/50	124,180	230,000	52,560,000	19.83	13	9
2	73	31	292/50	124,479	230,299	52,563,260	18.95	8	7
3	73	31	292/50	124,611	230,431	52,565,788	17.64	7	6

Table 5: The area-delay comparisons of *ind*2 using the 0.25μm library

Iter	IO	#HM	#SM(B/A)	$Gate_{SM}$	$Gate_{Tot}$	Area(μm^2)	Delay(ns)	T_{resyn}(hr)	T_{eco}(hr)
1	155	8	150/28	75,361	95,000	6,250,000	27.68	10	7
2	155	8	150/28	75,610	95,249	6,388,250	25.67	9	5
3	155	8	150/28	76,922	96,561	6,566,400	21.67	9	5
4	155	8	150/28	78,169	97,808	7,022,500	19.32	8	4

Table 6: The comparisons of the average gate and interconnect delays of *ind*2 using the 0.5μm and 0.25μm libraries

Lib.	Gate Delay(ns)[A]	Interconnect Delay(ns)[B]	[B]/[A]
0.5μm	0.171	0.277	1.62
0.25μm	0.107	0.325	3.04

The experiments were conducted on an HP-C180 workstation with 750Mb main memory. Tables 3-6 show the run times for the resynthesis and P&R ECO iteration. For example, in the first iteration of the *ind*1 design (Table 2), it took an average of 6 hours and 4 hours to run the synthesis and P&R ECO tasks.

5 Conclusions

In this paper, we have presented a complete chip design method which incorporates a soft-macro resynthesis method in interaction with chip floorplanning for area and timing improvements. We have conducted a series of experiments on three industrial designs. The results have demonstrated that by effectively relaxing the timing constraints of non-critical modules and tightening the timing constraints of critical modules we can achieve significant timing improvements with very little to no area penalty.

In this study, we have shown that an integrated synthesis, floorplanning, placement, and routing design flow allows designers to perform design resynthesis and ECO-based placement-and-routing guided by accurate layout timing information. This method is very effective for timing improvement with very little to no increase in chip area. One drawback for such a design flow is that it is an extremely time-consuming task. It takes close to 1 full-day to run one resynthesis iteration. Shortening the iteration time will be a key factor in improving the design exploration process. One possible approach is to move the iteration loop to a higher-level, such as floorplanning level. In order to make this happen, a more accurate delay and area estimation method is required. Another important issue is how to determine the initial timing budget for each module before synthesis. Good initial time-budgeting should shorten the number of resynthesis iterations and thus speed up the entire design process.

References

[1] B. T. Preas and M. J. Lorenzetti, *Physical Design Automation of VLSI Systems*, Benjamin Cummings, Menlo Park, CA., 1988.

[2] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, 2nd ed., Kluwer Academic Publishers, 1995.

[3] C.J. Alpert and A. B. Kahng, "Recent Direction in Netlist Partitioning: A Survey," *INTEGRATION: the VLSI Journal*, N19, pp. 1-81, 1995.

[4] M. Pedram and N. Bhat, "Layout Driven Technology Mapping," *Proc. of the 28th Design Automation Conference*, pp. 99-105, 1991.

[5] S. Liu, K. Pan, M. Pedram, and A. M. Despain, "Alleviating Routing Congestion by Combing Logic Resynthesis and Linear Placement," *Proc. of European Conference on Design Automation*, pp. 578-582, 1993.

[6] G. Stenz, B. M. Riess, B. Rohfleisch, F. M. Johannes, "Timing Driven Placement in Interaction with Netlist Transformations," *Proc. of Int. Symp. on Physical Design*, pp. 36-41, 1997.

[7] G. Holt and A. Tyagi, "Minimizing Interconnect Energy Through Integrated Low-Power Placement and Combinational Logic Synthesis," *Proc. of Int. Symp. on Physical Design*, pp. 48-53, 1997.

[8] C. M. Fiduccia and R. M. Mattheyses, "A Linear Time Heuristic for Improving Network Partitions," *Proc. of the 19th Design Automation Conference*, pp. 175-181, 1982.

[9] D. M. Schuler and E. G. Ulrich, "Clustering and linear placement," *Proc. of the 9th Design Automation Conference*, pp.412-419, 1972.

[10] H.-P. Su, A. C.-H. Wu, Y.-L. Lin, "Performance-Driven Soft-Macro Clustering and Placement by Preserving HDL Design Hierarchy," *Proc. of Int. Symp. on Physical Design*, pp. 12-17, 1998.

[11] "HDL Compiler for Verilog Reference Manual Version 3.4b", Synopsys, 1996.

[12] "Silicon Ensemble Reference Manual Version 5.0", Cadence, 1996.

[13] "Aquarious XO Reference Manual Version 2.1.2", AVANT!, 1998.

[14] "STAR-RC Reference Manual Version 2.2", AVANT!, 1997.

[15] "STAR-DC Reference Manual Version 2.1.2", AVANT!, 1996.

[16] "TSMC ASIC Data Book TCB670", Taiwan Semiconductor Manufacturing Company, Ltd. 1997

[17] "TSMC DSD Data Book ACB872", Taiwan Semiconductor Manufacturing Company, Ltd. 1998