Delay Budgeting for A Timing-Closure-Driven Design Method

Chien-Chu Kuo and Allen C.-H. Wu
Department of Computer Science, Tsing Hua University
Hsinchu, Taiwan, 30043, ROC

Abstract

In this paper, we present an RTL delay-budgeting approach for a timing-closure-driven design method. We formulate the delay-budgeting problem into the Lagrange-Multipliers-based slack distribution problem. We present two algorithms, namely the balanced slack distribution algorithm and the AT-based (Area-Time) slack distribution algorithm, to solve the problem. We also present a timing-closure-driven design flow by integrating commercial synthesis/layout tools with the proposed algorithms. We have demonstrated the viability of the proposed RTL delay-budgeting method. The results show that without an accurate AT-characteristic projection of modules the balanced slack distribution algorithm will be a good choice for delay budgeting at RTL.

1 Introduction

Over past decades, academia and industry have invested much effort in computer-aided-design research related to physical designs, including floorplanning, partitioning, placement, and routing. Several excellent reviews of physical design techniques were 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, follows 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, the complexity of chip designs has increased considerably in the past few years. Using a logic-level specification as the design entry may not be adequated to handling designs of such complexities. Consequently, more and more designers move their design entry to a higher abstraction, e.g., RTL, instead of logic level. Furthermore, as devices' geometries shrink, chip designers are facing a new set of design challenges, especially in electrical characteristics of circuits. This has led to a new research direction in design automation at synthesis and physical levels. Recently, several papers [4, 5, 6] have addressed the challenges and considerations in physical designs targeted to deep-submicron processes.

Figure 1 shows a typical RTL-based design flow consisting of two stages: (1) RTL/logic-level synthesis and (2) physical-level synthesis. In the first stage, the designer applies an RTL/logic-level synthesis procedure to convert the RTL design specification into a gate-level design. In the second stage, the layout designer applies a series of physical-level design tasks, including floorplanning, placement, and routing, to generate the final chip layout. In addition, two design iteration loops, *Loop B* and *Loop C*, may be applied to incrementally improve the design quality at each design stage.

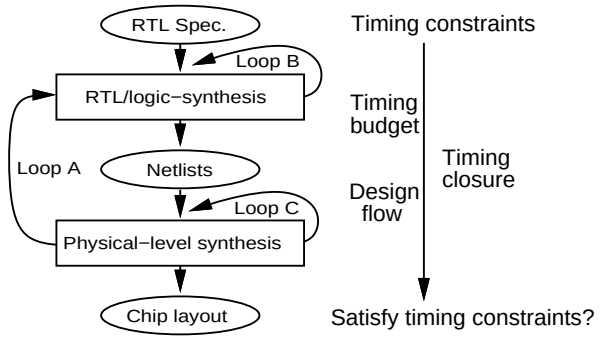

Figure 1: A typical RTL-based design flow.

Using such an RTL-based design flow, one of the crucial design considerations is the *timing closure* issue. Consider that an RTL-based design flow involves multi-level design tasks, including RTL/logic synthesis, floorplanning, placement, and routing. At each design level, the designer needs to determine the delay budget (i.e., the timing constraint) for the design. The ultimate goal is to achieve a timing-closure solution such that the final layout design satisfies the timing requirement. It's crucial to provide proper delay budgets to the design throughout the design process so that the design team can achieve the most cost-efficient chip layout while satisfying the timing requirement. If the delay budget is over-constrained, the cost may overrun. On the other hand, if the delay budget is under-constrained, the final chip layout may not meet the design requirement.

Without physical information, determining delay budget for a design at RTL is a non-trivial problem. It is also a very risky task because an inferior delay budget assignment may result in an infeasible design. One way to solve this problem is to close the synthesis and physical-design loop by exploiting the interaction between these two design stages, as *Loop A* shown in Figure 1. This motivates us to develop an RTL delay-budgeting method and a timing-closure-driven design flow.

In this paper, we present an RTL delay-budgeting approach and a timing-closure-driven design method. We formulate the delay-budgeting problem into the Lagrange-Multipliers-based slack distribution problem. We present two algorithms, namely the balanced slack distribution

†This work was supported in part by the National Science Council of R.O.C. under Grant NSC 89-2215-E-007-002 and NSC 89-2215-E-007-026.

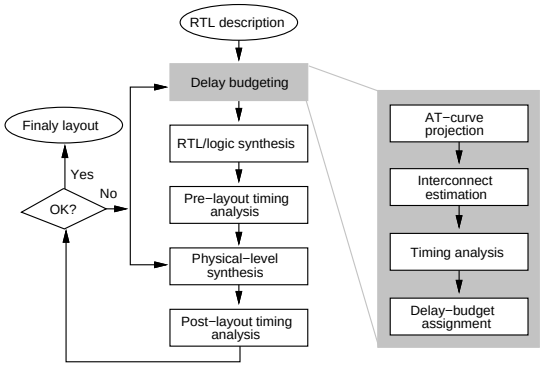

Figure 2: The proposed design flow.

algorithm and the AT-based slack distribution algorithm, to solve the delay-budgeting problem. We also present a timing-closure-driven design flow by integrating commercial synthesis/layout tools with our proposed algorithms. Experimental results are reported to demonstrate the effectiveness of the proposed algorithms and design flow.

2 Related Work

Hauge et al. [7] proposed the zero-slack algorithm which identifies a continuous path segment with the minimum non-zero slack. It then distributes the excessive delay uniformly among connections on the path segment. In [8], Nair et al. used the zero-slack algorithm to assign delay budgets to nets on long paths for placement. In [9], Youssef and Shragowitz improved the zero-slack algorithm for performance-driven layout. They presented a minimax approach to identify maximal delay bounds for nets and distribute slacks to nets according to their weights.

Sarrafzadeh et al. [10] proposed a delay budgeting method for the net-based timing-driven placement problem. They presented a formulation which performs delay upper-bounded budgeting and placement modification simultaneously. The experimental results showed that significant reduction on the longest path can be achieved using the proposed method. In [11], Sarrafzadeh et al. proposed a general approach to determine the upper bounds on net delays, called delay budgets, so that timing constraints on paths are satisfied. They formulated the problem as a convex programming problem and further modified it into an LP problem. The experimental results showed that the proposed algorithm outperforms previous approaches both in quality and efficiency.

3 The Proposed Design Flow

Figure 2 depicts the proposed design flow consisting of five steps: (1) delay budgeting, (2) RTL/logic synthesis, (3) pre-layout timing analysis, (4) physical-level synthesis, and (5) post-layout timing analysis. In this study, we focus on the delay-budget assignment problem. Our proposed method consists of four steps: (1) AT-curve (area-time characteristics) projection, (2) interconnect estimation, (3) timing analysis, and (4) delay-budget assignment. The inputs to the method include an RTL design specification described in HDLs and a timing constraint. In the first step, we perform an AT-curve projection procedure to estimate the AT (area-time) characteristics for each soft macro. In our approach, we use commercial tools to conduct a series of synthesis and physical design tasks

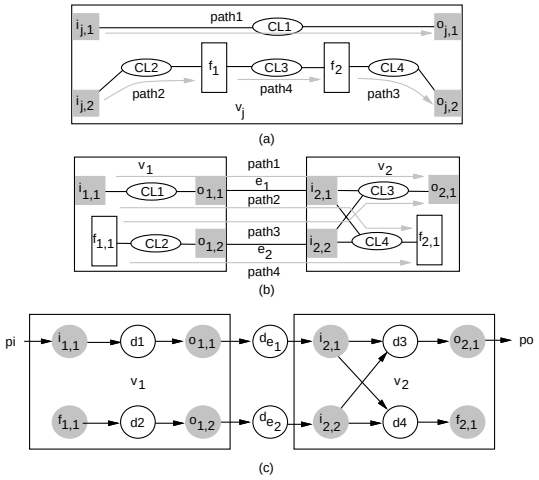

Figure 3: The signal paths: (a) in a module, (b) cross modules, (c) the signal-path directed graph.

to generate an AT-curve for each module. In the second step, we apply a floorplanning procedure to estimate interconnect delays between macros. In the third step, we perform timing analysis on the design to determine the critical paths and slack values of macros. Finally, we perform delay-budget assignment to determine timing constraints for each soft macro.

4 Delay-Budgeting at RTL

4.1 Preliminaries

We use a connected hypergraph to represent a hierarchical design. Let $G = (V, E)$ represent a hierarchical netlist, where $V = \{v_i \mid i = 1..n\}$ denotes a set of modules and $E = \{e_{ij} \mid v_i, v_j \in V\}$ a set of interconnections. Let $PI = \{pi_i \mid i = 1..p\}$ and $PO = \{po_i \mid i = 1..q\}$ denote a set of primary inputs and outputs, respectively. Each module contains a set of inputs $I_{v_j} = \{i_{j,k} \mid i_{j,k} \in v_j, k = 1..r\}$, a set of outputs $O_{v_j} = \{o_{j,l} \mid o_{j,l} \in v_j, l = 1..m\}$, and a set of latches $F_{v_j} = \{f_{j,s} \mid f_{j,s} \in v_j, s = 1..t\}$.

Figure 3(a) illustrates the four possible signal paths in a module v_j: (1) $path < i_{j,k} \rightarrow o_{j,l} > (path1)$, (2) $path < i_{j,k} \rightarrow f_{j,s} > (path2)$, (3) $path < f_{j,s} \rightarrow o_{j,l} > (path3)$, and (4) $path < f_{j,s1} \rightarrow f_{j,s2} > (path4)$. In addition, Figure 3(b) illustrates the four possible cross-module signal paths between two interconnected modules v_1 and v_2 : (1) $path < i_{1,k} \rightarrow o_{2,l} > (path1)$, (2) $path < i_{1,k} \rightarrow f_{2,s} > (path2)$, (3) $path < f_{1,s} \rightarrow o_{2,l} > (path3)$, and (4) $path < f_{1,s} \rightarrow f_{2,s} > (path4)$.

We construct a directed graph $G_{pd} = (V_d, E_d)$ to represent cross-module signal paths, where $V_d = \{V_{CL}, V_T\}$, V_{CL} denotes a set of delay nodes and V_T a set of input/output terminal nodes. Each delay node associates with a delay value which is contributed by the path delay of a combinational circuit or wiring delay. Each input/output terminal node associates with a zero delay value. E_d denotes a set of edges which represent signal flows between nodes. Figure 3(c) depicts a signal-path directed graph which is derived from the design in Figure 3(b).

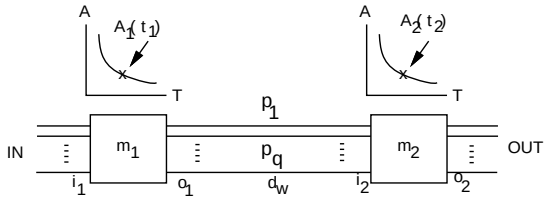

Figure 4: An RTL delay-budgeting example.

4.2 Slack Computation

The slack $s(v)$ of a node v [7] is defined as:

$$s(v) = t_r(v) - t_a(v),$$

$$t_a(v) = |v| + max\{t_a(x) \mid x \in \pi^+(v)\},$$

$$t_r(v) = min\{(t_r(x) - |x|) \mid x \in \pi^-(v)\},$$

where $t_a(v)$ and $t_r(v)$ denote arrival time and required time of v, $|v|$ and $|x|$ the delays of nodes v and x, and $\pi^-(v)$ and $\pi^+(v)$ the fan_out and fan_in set of node v, respectively. The slack value of each node represents the delay tolerability for the node without violating the timing constraint.

4.3 Problem Description

The delay-budgeting problem is to determine the delay budget for each module in the design. The delay budget of a module is used as the timing constraint for the module in the synthesis process. The goal is to produce the minimum-cost design while the most critical-path delay in the design satisfies the given timing constraint.

We formulate the delay-budgeting problem into a slack distribution problem. Let's first use the following example to explain our considerations. As shown in Figure 4, there are $p_1..p_q$ signal paths traveling from IN to OUT via modules m_1 to m_2. There are two main considerations when we determine the delay budgets for the two modules. The first one is the inter-module wire delay (d_w) between the two modules. In today's deep-submicron technology, especially when the modules become larger, the inter-module wire delay may become the dominant factor to the most critical-path delay. Hence, it's necessary to take the inter-module wire delay into consideration in the delay-budgeting process.

The second one is how to assign a cost-efficient delay budget to each module. As we mentioned earlier, the delay budget assigned to each module will be used as the timing constraint when we synthesize the module into a gate-level design. Hence, the AT (area-time) characteristics of modules may directly affect the final synthesized results when assigning different delay budgets to the modules. For example, assume that we have a uniform AT characteristic for all paths from all inputs to all outputs in module m_1 as well as in module m_2. We also assume that m_2 has a much steeper AT characteristic compared to that of m_1, which means that tightening the delay budget of m_2 will result in a large increase in area cost. In this case, we prefer to assign a relaxing delay budget to m_2 and a tightened delay budget to m_1 in order to achieve a more cost-efficient synthesized result. This motivates us to investigate the effect of taking into account AT characteristics of modules in the delay-budgeting process.

In the following sections, we first describe the Lagrange Multipliers formulation for the delay-budgeting problem. Then, we present two algorithms, the balanced slack distribution algorithm and the AT-based slack distribution algorithm.

4.4 Slack Distribution based on Lagrange Multipliers

Given AT characteristics of all modules, we can obtain an optimal delay-budgeting solution for each module by applying the Lagrange Multipliers method. Let $A_i(t_i)$ represent the AT-curve function of module i, n the number of modules, and T the timing constraint. The objective function of the delay-budgeting problem can be formulated as below:

$$Minimize \sum_{i=1}^{n} A_i(t_i)$$

$$subject\ to \sum_{i=1}^{n} t_i \leq T\ for\ all\ signal\ paths.$$

To determine the maximum or minimum value of the above objective function, we need to find all values of $t_1, t_2, ..., t_n$ for each signal path and λ such that:

$$\bigtriangledown(\sum_{i=1}^{n} A_i(t_i)) = \lambda \bigtriangledown (\sum_{i=1}^{n} t_i)\ and\ \sum_{i=1}^{n} t_i \leq T$$

Using the above Lagrange Multipliers formulation, we can determine delay budgets for all paths simultaneously. In this case, the Lagrange Multipliers equation is: $\bigtriangledown(\sum A_i(t_i)) = \lambda_1 \bigtriangledown (\sum t_i)_1 + ... + \lambda_n \bigtriangledown (\sum t_i)_n$, where $(\sum A_i)$ represents the total containing nodes and $(\sum t_i)_i$ is the i-th time violation path constraint. If all of $\bigtriangledown A_i(t_i)$ are linear equations, these functions can be solved in $O((n+p)^2)$ time using the LU-decomposition method, where n and p are the number of nodes and paths, respectively. However, due to it's complexity and strict constraints, it is only feasible to solve small designs with a specific design characteristic.

The other problem is that the Lagrange Multipliers method requires an accurate AT characteristic for each module in order to achieve a feasible design solution. To accomplish that, we need an accurate estimation method to predict the AT characteristic for each module or to run the actual synthesis and physical design tasks for AT design space exploration of all modules. However, it is very difficult to develop an accurate AT-curve estimation method. Furthermore, it is a very time-consuming task if we have to generate AT characteristics for all modules by running a complete synthesis process.

In our approach, we propose two heuristics derived from the Lagrange Multipliers method. Both heuristics determine delay budgets for the critical paths one at a time. The first heuristic, the balanced slack distribution algorithm, does not consider the AT characteristics of modules but the second heuristic, the AT-based slack distribution algorithm, does.

4.5 The Balanced Slack Distribution Algorithm

In this section we present the balanced slack distribution algorithm. We first use the Lagrange-Multipliers-based method to resolve negative slack problem and then

apply the zero-slack algorithm [7] to distribute slacks to all nodes in a balance way. The main idea of the zero-slack algorithm is that if all nodes have the same property, distributing slacks uniformly over all nodes often achieves an optimal solution. In our approach, we assume that all the modules have the same AT-curve characteristic. Under this assumption, our propose algorithm does not require AT-curve information of each module in the delay-budget assignment process.

The proposed balanced slack distribution algorithm is listed below:

Algorithm 1 *Balanced_Slack_Distrbution*(G_{pd}, T)
begin
  $S_{assign} = \phi$; $S_{crit} = \phi$; $S_{temp} = \phi$;
  while(Timing_Violation_free(G_{pd}, T) = **False**)
  begin
    Slack_Computation(G_{pd});
    $\{P_{crit}, S_{crit}\} \leftarrow$ Critical_Path_Extraction(G_{pd}, T);
    $S_{temp} = S_{crit} - S_{assign}$;
    for$(v_i \in S_{temp})$
    begin
      $\triangle v_i = \frac{[(Delay\ of\ critical\ path) - T] * |v_i|}{\sum_{v_i \in S_{temp}} v_i}$;
      $S_{assign} \leftarrow S_{assign} + |v_i|$;
    end_for
  end_while
  Zero_Slack_Algorithm(G_{pd}, T);
end_Algorithm 1

Let S_{assign}, P_{crit}, S_{crit}, and S_{temp} denote the set of nodes with reassigned slacks, the set of critical paths, the set of critical nodes, and the set of nodes for slack reassignments. The inputs to the algorithm include a path-delay directed graph and a timing constraint. Procedure *Timing_Violation_free* evaluates whether the design is timing-violation free. If yes, it returns *True* and invokes the *Zero_Slack_Algorithm* to perform delay-budget assignment. Otherwise, it returns *False* and enters the *while* loop to resolve the negative slack problem. The algorithm first invokes procedure *Slack_Computation* to compute the slack value of nodes. It then invokes procedure *Critical_Path_Extraction* to extract critical paths and the nodes on the critical paths (in our implementation we extract 1,000 most critical paths that will be discussed in the experimental section). For each node on the critical path which has not been reassigned with a new slack value, we will use a balanced-distribution scheme to reassign the node with a new slack value. After reassigned slacks to the nodes on the critical path, Procedure *Timing_Violation_free* will be invoked to evaluate whether the design is timing-violation free. If not, the algorithm executes the *while* loop repeatedly until all the timing violations are resolved.

The following theorem proves that *Algorithm* 1 can distribute slacks to all nodes in a balance way. We define that the node v_i is more *critical* than node v_j if the value of v_i is greater than v_j's and the reduce rate of node $v_i = \triangle|v_i|/|v_i|$.

Theorem 1 : For a given directed weighted graph $G_{pd} = (V_{pd}, E_{pd})$ and the timing constraint T, the balance slack distribution algorithm guarantees that the reduce rate of v_i is greater than v_j's if and only if v_i is more *critical* than v_j. *Proof* : [12].

Theorem 1 guarantees that; (1) a more critical node has a greater reduce rate compared to a less critical one and (2) two nodes have the same reduce rate if they are on the same critical path. After resolving all of timing violated nodes, the algorithm invokes the Zero Slack Algorithm to distribute the positive slack to nodes until all nodes have a zero slack value.

4.6 The AT-Based Slack Distribution Algorithm

The balanced slack distribution algorithm assumes that all modules have the same AT-curve characteristic. In this section, we present the AT-based slack distribution algorithm that takes into account the AT-curve property of each module for delay-budget assignment.

We define the cost of node v_i as $c_i = (-\triangle A/\triangle T)$. The greater c_i, the higher cost is required to reduce one unit time in v_i. Before we present the AT-based slack distribution algorithm, we need to consider the following two factors:

Consideration 1: The node with a large c_i value has a low priority to share the negative slack.

Consideration 2: The node with a large weight of $|v_i|$ has a high priority to be assigned with a tighter resynthesis timing constraint.

Consideration 1 tries to reduce negative slacks by adding less area penalty to the design. Consideration 2 is made from the property of synthesis techniques. Intuitively, if $|v_i|$ has a large value, $\triangle|v_i|$ will be large due to the property of the zero-slack algorithm. For a large $\triangle|v_i|$ value, it means that it has more room to explore the design space of module v_i during the resynthesis process. Hence, v_i has a high priority to be resynthesized with a tighter constraint.

The AT-based slack distribution algorithm is listed below:

Algorithm 2 *AT_Based_Slack_Distrbution*(G_{pd}, T)
begin
  $P_{crit} = \phi$; $S_{crit} = \phi$
  Slack_Computation(G_{pd});
  $\triangle v_i = 0$, for all v_i;
  $\{P_{crit}, S_{crit}\} \leftarrow$ Critical_Path_Extraction(G_{pd}, T);
  while$(P_{crit} \neq \phi)$
  begin
    for$(v_i \in S_{crit})$
    begin
      c_i = Cost_Computation$(|v_i|)$;
      $g_i = (\frac{1}{c_i})^{\alpha} * (|v_i|)^{(1-\alpha)}$;
    end_for
    for$(p_j \in P_{crit})$
    begin
      for$((v_i \in p_j) \cap (v_i \in S_{crit})))$;
      begin
        $\triangle v'_i = \frac{[(Delay\ of\ critical\ path) - T] * g_i}{\sum_{v_k \in p_j} g_k}$;
        if $\triangle v'_i > \triangle v_i$ **then** $\triangle v_i = \triangle v'_i$;
      end_for
    end_for
    Update(v_i);
  end_while
  Zero_Slack_Algorithm(G_{pd}, T);
end_Algorithm 2

Table 1: Characteristics of the benchmarking designs.

Designs	# Mods(Seq./Comb.)	# Inter-Nets	# IOs
SP	12(8/4)	799	152
ELLIPTIC	10(1/9)	1311	213
Controller	9(7/2)	387	162

Let S_{crit} and P_{crit} denote a set of critical nodes and critical paths, respectively. The inputs to the algorithm include a path-delay directed graph and a timing constraint. The algorithm first invokes procedure *Slack_Computation* to compute the slack value of nodes. It then invokes procedure *Critical_Path_Extraction* to extract the critical paths. If the design is timing-violation free (i.e., $P_{crit} = \phi$), the algorithm invokes the *Zero_Slack_Algorithm* procedure. Otherwise, it enters the *while* loop to resolve the negative slack problem. In the *while* loop, the algorithm first invokes procedure *Cost_Computation* to compute the cost of nodes, follows by computing the *gain* g_i of resynthesizing module v_i. The *gain* g_i of module v_i is defined as a function of c_i and $|v_i|$ such that g_i is proportional to $|v_i|$ and inversely proportional to c_i, where α is a constant set by the user to express his/her preference of one term over the other's. After that, the algorithm performs negative-slack distribution on the critical paths one at a time. Since a node may be located at several paths simultaneously, we choose the one with the largest slack value as the node's slack value. (i.e., $if\ \triangle v'_i > \triangle v_i\ then\ \triangle v_i = \triangle v'_i$). Procedure *Update* updates nodes' delay values (i.e., $|v_i| = |v_i| - \triangle v_i$). The algorithm executes the *while* loop repeatedly until all the timing violations are resolved. Finally, it invokes the *Zero_Slack_Algorithm* procedure to perform delay-budget assignment.

5 Experiments

We have implemented the proposed algorithms and the design method in the C and Perl programming languages. We have tested our proposed algorithms on three benchmarks, as shown in Table 1. The first design is a 64-bit simple processor (*SP*). The second design is a 64-bit elliptic filter. The third example is the controller of the P6 processor [13]. All three designs are described as hierarchical RTL netlists in Verilog. Table 1 shows the characteristics of the benchmarks in which $\#\ Mods(Seq./Comb.)$, $\#\ Inter-Nets$, and $\#\ IOs$ denote the number of modules(sequential modules/combinational modules), inter-module nets, and input/output pins. In all experiments, we used a 0.6μm cell library.

We have conducted two sets of experiments to examine the proposed two algorithms, as depicted in Figure 5. In the first experiment *Exp*1, we tested the balanced slack distribution algorithm *Algorithm 1*. In the first step, we used Synopsys' *Design Compiler* [14] to synthesize each leaf module into a gate-level netlist (*compile -ungroup_all -map_effort medium*) and then performed timing analysis to report the 100 most critical paths. In the second step, we used the *TimberWolf Macro Cell Placement Program* [15] to perform macro placement. We set the aspect-ratio of all modules ranging 0.33 to 3 and used the 100 most critical paths as the constraint. After performing the macro placement procedure, TimberWolf reported a floorplan result with module locations on the layout plane and their aspect ratios. We then estimated the inter-module wire length using the Manhattan distance model. Based on

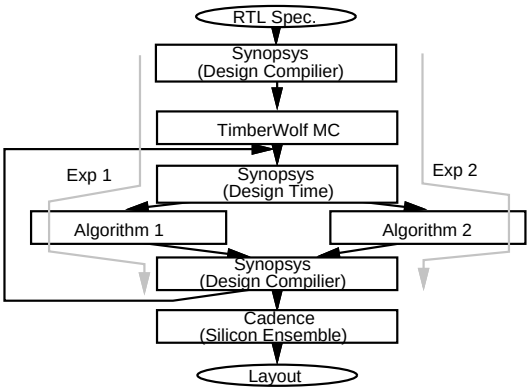

Figure 5: The experimental procedure.

the estimated inter-module wire lengths, we adjusted the *Wire Load Model* in *Design Compiler*. We then used *Design Time* to perform timing analysis and reported the 1000 most critical paths that will be used as the inputs to the *Algorithm 1*. After performing delay-budgeting assignment, we invoked *Design Compiler* to resynthesize the design with the new delay budgets. The iteration continued until all timing violations were resolved. Finally, we used Cadence's *Silicon Ensemble* [16] to perform placement and routing.

In the second experiment *Exp*2, we tested the AT-based slack distribution algorithm *Algorithm 2*. The experimental design flow is same as the first experimental design flow except the first step. In the first step of the second experiment, we used *Design Compiler* to project the AT-curve of each module. We first synthesized each module with two options: (1) minimizing area (*set_max_area 0*) and (2) minimizing delay (*set_max_delay 0 -tp all_outputs()*), which resulted in two design alternatives (A_a, T_a) and (A_t, T_t), respectively. Then, we projected the AT-curve of the module as $c_i = (A_t - A_a)/(T_a - T_t)$. In this experiment, we set $p = 1000$ and $\alpha = 0.5$.

Table 2 shows the results in which T_{req}, T_{final}, *A_cost*, *CPU*, and *Iter.* represent the timing requirement, the final timing, gate count, chip area, run time, and design iterations, respectively. The gate count results (*A_cost*) were reported from Synopsys's *Design Compiler*, while the final timing results (T_{final}) were reported from Synopsys's *Design Time*. The results show that it requires 2-6 design iterations to achieve a timing-closure solution (i.e., the final timing of the design satisfies the given timing constraint). Figures 6, and 7 show the AT-curve of design iterations for the *Controller* design. All experiments were running on a SUN Ultra Enterprise 150 workstation with 512 MB memory.

The results show that for the *SP* and the *Controller* designs, the balance slack distribution method achieved more area-efficient designs with shorter run times compared to that generated by the AT-based slack distribution method. The results also show that for the *Elliptic* filter design, the AT-based slack distribution method achieved a more area-efficient design with a shorter run time compared to that generated by the balance slack distribution method. The reason is that when applying the AT-based slack distribution method, the crucial condition to achieve an area-efficient design is that we have

Table 2: The comparisons between the two algorithms

Designs	The balance slack distribution algorithm					
	$T_{req}(ns)$	$T_{final}(ns)$	A_cost(gates)	Area(μm^2)	CPU(Mins.)	Iter.
SP	50	49.9	20,050	5,819,474	495	4
Elliptic	55	54.6	28,468	8,173,075	62.6	2
Controller	20	19.8	29,071	8,328,912	651	5
Designs	**The AT-based slack distribution algorithm**					
SP	50	49.9	20,568	5,927,447	527	5
Elliptic	55	54.8	27,809	7,989,245	54	2
Controller	20	19.9	29,385	8,418,394	863	6

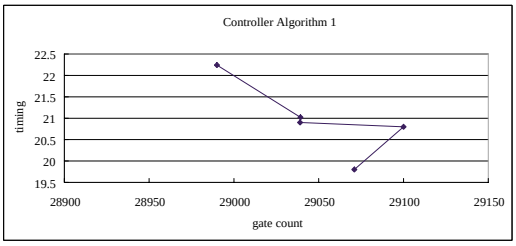

Figure 6: AT-curve during design iterations using the balance slack distribution algorithm (*Controller*).

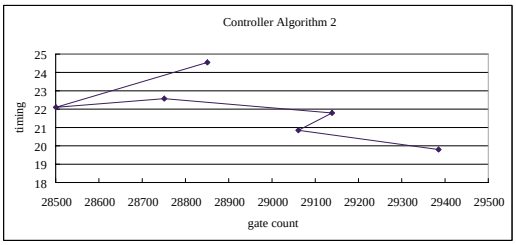

Figure 7: AT-curve during design iterations using the AT-based slack distribution algorithm (*Controller*).

to provide it with an accurate AT-curve for each module. In our implementation, we used the proposed AT-curve projection formula $c_i = (A_t - A_a)/(T_a - T_t)$ to project the AT-curve of each module. From the experiments, we observed that the proposed AT-curve projection formula contained much accurate AT-curve estimations for combinational modules than sequential modules. When a design contains mainly combinational modules, e.g., the *Elliptic* filter design (8 out of 9 modules contain combinational circuits), our proposed AT-curve projection formula can predict a fairly accurate AT-curve for each module. This leads to an area-efficient design with a shorter run time when applying the AT-based slack distribution method. On the other hand, when a design contains mostly sequential modules, e.g., the *SP* and *Controller* designs, our proposed AT-curve projection formula may not be able to predict accurate AT-curves for the modules. In this case, the AT-based slack distribution method may not be as effective as the balance slack distribution method.

6 Conclusions

In this paper, we have presented two delay-budgeting algorithms and a timing-closure-driven design method for delay-budgeting at RTL. Experimental results have demonstrated that using the proposed design method, timing-closure solutions for the three benchmarking designs can be achieved in 2-6 design iterations.

Without an accurate AT-curve projection, the AT-based slack distribution algorithm may not be as effective as the balanced slack distribution algorithm. In order to effectively utilizing the AT-based slack distribution algorithm for delay-budgeting, an accurate AT-curve projection method is required.

In this study, we have only tested our proposed methods on medium-scaled designs with a single clock. We have also not taken false paths into consideration. Future studies include testing on large designs, developing an accurate AT-curve projection method and a delay-budgeting method by considering multi-cycle clock and false paths.

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] E. S. Kuh, "Physical Design: Reminiscing and Looking Ahead," *Proc. of Int. Symp. on Physical Design*, pp. 206, 1997.

[5] R. Composano, "The Quarter Micro Challenge: Integrating Physical and Logic Design," *Proc. of Int. Symp. on Physical Design*, pp. 211, 1997.

[6] K. Keutzer, A. R. Newton, and N. Shenoy, "The future of Logic Synthesis and Physical Design in Deep-Submicron Process Geometries," *Proc. of Int. Symp. on Physical Design*, pp. 218-223, 1997.

[7] P. S. Hauge, R. Nair, and E. J. Yoffa, "Circuit Placement for Predictable Performance," *Proc. of Int. Conf. Computer-Aided Design*, pp. 88-91, 1987.

[8] R. Nair, C. L. Berman, P. S. Hauge, and E. J. Yoffa, "Generation of Performance Constraints for Layout," *Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 8, pp. 860-874, August, 1989.

[9] H. Youssef and E. Shragowitz, "Timing Constraints for Correct Performance," *Proc. of Int. Conf. Computer-Aided Design*, pp. 24-27, 1990.

[10] M. Sarrafzadeh, D. Knol, and G. Tellez, "Unification of Budgeting and Placement," *Proc. of 34th Design Automation Conf.*, pp. 758-761, 1997.

[11] M. Sarrafzadeh, D. Knol, and G. Tellez, "A Delay Budgeting Algorithm Algorithm Ensuring Maximum Flexibility in Placement," *Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 11, pp. 1332-1341, Nov. 1997.

[12] C. C. Kuo, "Delay Budgeting for A Timing-Closure-Driven Design Method", Master Thesis, Dept. of Computer Science, Tsing Hua Univ., Hinchu, Taiwan, ROC, June, 1998.

[13] "HDP Implementation Document", Technical Report, Dept. of Computer Science, Tsing Hua Univ., Hsinchu, Taiwan, ROC., June, 1997.

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

[15] C. Sechen, "TimberWolf6.0: Mixed macro/Standard cell floor planning, placement and routing package," user's manual, Yale University, Sep. 1991.

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