A Unified Theory of Timing Budget Management

Soheil Ghiasi, *Member, IEEE*, Elaheh Bozorgzadeh, *Member, IEEE*, Po-Kuan Huang, *Student Member, IEEE*, Roozbeh Jafari, *Student Member, IEEE*, and Majid Sarrafzadeh, *Fellow, IEEE*

***Abstract*—This paper presents a theoretical framework that solves optimally and in polynomial time many open problems in time budgeting. The approach unifies a large class of existing time-management paradigms. Examples include time budgeting for maximizing total weighted delay relaxation, minimizing the maximum relaxation, and min-skew time budget distribution. The authors develop a combinatorial framework through which we prove that many of the time-management problems can be transformed into a min-cost flow problem instance. The methodology is applied to intellectual-property-based datapath synthesis targeting field-programmable gate arrays. The synthesis flow maps the input operations to parameterized library modules during which different time budgeting policies have been applied. The techniques always improve the area requirement of the implemented test benches and consistently outperform a widely used competitor. The experiments verify that combining fairness and maximization objectives improves the results further as compared with pure maximum budgeting. The combined fairness and maximization objective improves the area by 25.8% and 28.7% in slice and LUT counts, respectively.**

***Index Terms*—Implementation selection, slack distribution optimal algorithm, time budgeting.**

I. Introduction

With tremendous growth in the complexity of today's systems, traditional design techniques are no longer capable of handling the design issues. One approach to tackle this problem is to design the system in a modular and hierarchical fashion, which in turn calls for methodologies to transform system-level constraints to component-level constraints. Such techniques enable the application of divide-and-conquer-based approaches to address the system issues. This task is generally referred to as "budget management."

The problem of budget management has been studied for several design constraints including timing and area. Particularly, time budgeting is performed to assign timing constraints to subcomponents of a design. This intuitively translates to relaxing the timing constraints for as many components as possible without violating the system's timing constraints. The components with relaxed timing constraints can be further optimized to improve system's area, power dissipation, or other design quality metrics.

Manuscript received March 15, 2005; revised August 22, 2005. This paper was recommended by Associate Editor M. D. F. Wong.

S. Ghiasi and P.-K. Huang are with the Department of Electrical and Computer Engineering, University of California, Davis, CA 95616 USA (e-mail: soheil.ece.us.davis.edu).

E. Bozorgzadeh is with the Department of Computer Science, University of California, Irvine, CA 92697 USA.

R. Jafari and M. Sarrafzadeh are with the Department of Computer Science, University of California, Los Angeles, CA 90095 USA.

Digital Object Identifier 10.1109/TCAD.2006.873901

However, almost all of the previous research efforts employ suboptimal heuristics for addressing the reasonable formulations of the budgeting problem. In our previous work [12], [13], we optimally solved the problem of integral budget assignment through linear programming (LP) relaxation and subsequent optimal delay budget reassignment. Combinatorial methods, however, are often preferred to LP-based approaches due to their numerical instability and slow runtimes.

In this paper, we present a unified theoretical framework that solves different well-known formulations of the budgeting problem through efficient combinatorial techniques. We model the given application as a directed acyclic graph (DAG) and assign timing budget values to the edges of the DAG. We show that many other common budgeting models such as node budgeting or hybrid edge/node budgeting are special cases of our generic model. Moreover, our method optimally solves the problems of maximum, weighted, bounded, fair, and min-skew budgeting. It also provides some guidelines for incremental budget reassignment, which is useful for many practical applications.

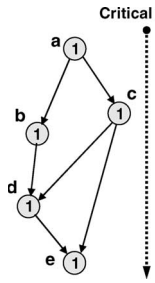
To experiment our theoretical results, we apply our technique to library mapping during datapath synthesis. We integrate the time budgeting and module selection into the synthesis flow for mapping applications onto field-programmable gate array (FPGA) devices. Efficient time budgeting allows us to choose the proper modules from the library to obtain further quality improvements. Our results highlight that along with the total weighted summation of delay budget, its distribution throughout the design can significantly impact the design quality.

II. Background

A. Application and Delay Model

A given application can be represented as a DAG $G = (V, E)$, where V is a set of vertices and E is a set of directed edges. A set of sources (or primary inputs) $I \subset V$ is a set of vertices without incoming edges, and a set of sinks (or primary outputs) $O \subset V$ is a set of vertices without outgoing edges. Given $|I|$ signals each starting from a source at the time zero, we consider their propagation toward O along directed edges and vertices in G.

Traditional formulations assume that each vertex $v \in V$ is associated with a delay $d(v)$, which represents the time it takes for a signal to pass through v, and that there is no delay on edges. On the other hand, some formulations assume that each edge $e \in E$ is associated with a delay $d(e)$ and nodes have zero delay. In this section, we adopt the node delay model for explaining the definitions and the example. However, this paper



	Maximum Budget	Bounded Maximum Budget	Min-Max (within 25% of max)	Min-Max (within 40% of max)
a	0	1 (≥1)	1	1
b	4	3 (≤3)	2	1
c	4	3 (≤3)	2	1
d	0	0	1	1
e	0	0	0	1
Total Budget	8	7	6	5

Fig. 1. Example of delay budget assignment for maximum, bounded, and min−max (with different total budgets bounds) objectives.

deals with the more generic edge delay model. Note that edge delay model can be utilized to model the node delay model as a special case.

The latest time of signals to arrive at the output of any vertex $v \in V - I$ is given recursively by $a(v) = \max_{u \in \mathrm{FI}(v)}(a(u) + d(v))$, where $\mathrm{FI}(v)$ is a set of immediate predecessors (or fan-ins) of v, and $a(v)$ is the arrival time at v. The application is often required to meet a given timing constraint T, which means that $a(v) \leq T \; \forall v \in O$. Intuitively, the problem of timing budget assignment is to allocate extra timing delay $b(v)$ to $v \in V$ such that the application timing constraint is met and some objective function is optimized.

B. Motivating Example

Fig. 1 illustrates an example of different delay budget assignment policies in action. All of the nodes in the example have unit intrinsic delay. Therefore, the critical path of the graph has length 4. Delay budgets are assigned to the nodes, and we assume that the timing constraint for the application is eight time units. Hence, delay budget assignment should not create any path that takes longer than eight units of time. For each cost function (policy), an optimal solution is depicted in the table. The delay budget assigned to each node is shown in each cell of the table. Note that the delay budget should be added to the unit intrinsic delay of the node to calculate its actual latency.

An intuitive budget assignment policy tries to maximize the total delay budget assigned to the graph, assuming that larger total budget correlates to larger improvements in the utility function. The first column of the table shown in Fig. 1 (maximum budgeting) represents the result of applying this cost function. A useful extension of this policy considers different weights for the nodes and tries to maximize the total weighted budget assignment.

The second column of the table (bounded maximum budgeting) illustrates the node budgets when the cost function aims to maximize the total budget while maintaining some lower/upper bounds on the amount of delay budget assigned to nodes. In this example, node a has a lower bound of 1, and nodes b and c have an upper bound of 3 on their delay budgets. Bounds are useful in many applications due to nonlinear or semidiscrete relation of the utility function to node delays. Another popular policy is to distribute the budget values fairly (minimizing the maximum budget value) while still trying to maximize the total budget. The last two columns of the table (i.e., min−max) represent the result of applying this policy to the sample graph, where 25% and 40% deviation from maximum budget is allowed for each case. Note that minimizing the maximum budget value leads to trivial solutions if there is no constraint on the total budget.

III. Related Work

The idea of resource budgeting (both temporal and spatial budgeting) has been widely used in many different applications. For example, during design optimization flow, timing budget is allocated to each node under a given timing constraint and optimization is applied. If timing constraint is not met, the delay budget is reallocated [5], [24]. Distribution of delay budget is applied to determine the wire length under the given timing constraints. Delay budgeting has also been utilized to slow down the noncritical functional units or gates through supply or threshold voltage adjustment. A significant amount of savings in dynamic or leakage power has been reported using such techniques [2], [32]. Another example of application is timing-driven placement and floor planning for which delay budgeting has been extensively studied by several researchers [1], [4], [20]. In [20] and [22], placement and net rebudgeting are combined. In [34], a novel technique for net weighting algorithm is proposed for timing optimization in placement. Recently, in [7], a new technique for delay budgeting on sequential circuits is proposed.

Moreover, the idea has been applied to gate and wire sizing under timing constraints, where budget management is utilized to find a set of nodes/edges in the netlist graph whose physical size or power dissipation can be reduced by mapping to smaller or power-efficient cell instances with larger delays from a target library [4], [16], [25]. High-level synthesis is another sample application in which timing slack of the nodes in the data flow graphs (DFGs) is considered for better optimization in area and power. Examples are the algorithms and techniques developed for area minimization in pipelined data path [28] and power minimization under timing constraint [18], [35]. Similarly, layout compaction has benefited from space budgeting [14], [19], [36]. The concept of budgeting was first proposed by Liao and Wong [36] for spatial budgeting in layout compaction. Finally, software optimization constitutes another application domain in which timing slack has been utilized to improve power efficiency or computation quality. Examples include intraprogram dynamic voltage scaling [15] and accuracy improvement in tracking systems [31] via timing relaxation for noncritical software blocks.

The techniques employed in previous works are suboptimal heuristics such as zero slack algorithm (ZSA) [27] and maximum independent set algorithm (MISA) [3]. In our previous work [12], we solved the problem of integral delay budgeting through LP relaxation. However, LP-based techniques are prone to numerical instability and slow runtimes, hence, combinatorial methods are preferred. In this paper, we present an optimal combinatorial method for solving the integral budgeting problem [30]. Furthermore, we utilize our method to solve the budgeting problem under many other objective functions, including weighted, bounded, and fair budgeting. Moreover, we present potentials for tackling the incremental budgeting

problem. Our method is similar to the approach that Boros *et al.* have taken for the problem of balancing the DFGs [10], [11]. However, the time-budgeting problem is fairly different from the problem at hand. Moreover, to the best of our knowledge, this technique is quite novel in the electronic design automation (EDA) community.

IV. Problem Statement

Intuitively, the problem of timing budget management can be stated in the following way: Given an application with distinct constituting blocks, what is the maximum tolerable slow down of individual blocks without violating the timing constraints of the application? The slowed-down blocks can be further optimized to improve any of the generic design quality metrics depending on the application domain.

Although constituting blocks are often modeled as nodes in a DAG, the problem of delay budgeting for nodes is a special case of the more general edge budgeting case. This fact will be discussed in Section VI in more detail. Therefore, we focus on the edge-budgeting problem. We assume that delay values and delay budgets are assigned to each edge of the DAG, and nodes do not impose any delay on application paths. Edge budgeting has direct application in many areas such as routing and network optimization. Moreover, its special cases can solve the node budgeting problem that applies to many other areas such as application and architecture synthesis.

Given an application represented by a DAG $G(V, E)$, there is a nonnegative delay value d_{ij} and a nonnegative delay budget variable b_{ij} associated to each edge $e_{ij} \in E$. There is a given value T that specifies the timing constraint of the application. Timing constraint implies that all paths from any primary input to any primary output must take no longer than T. The delay of each path is calculated according to the following definition.

Definition 1: The delay of a path p from node s to node t is equal to $\sum_{e_{ij} \in p}(d_{ij} + b_{ij})$. We may use the terms delay of the path, cost of the path, and the distance between nodes s and t interchangeably.

For simplicity, we add a virtual super input node SI to G that is only connected to all of the primary inputs. Similarly, we add a virtual super output node SO that takes only all of the primary outputs of G as its fan-ins (Fig. 2). All of the edges connecting SI or SO to any other node in G have zero delay. We still use V and E to represent the set of nodes and edges after adding SI and SO to G. The problem at hand can be formally formulated as maximizing $\sum_{e_{ij} \in E} b_{ij}$ such that all the paths from SI to SO take no longer than T.

The problem can be stated as the following ILP formulation:

$$\text{Max} \sum_{e_{ij} \in E} b_{ij} \tag{1}$$

$$\sum (d_{ij} + b_{ij}) \leq T \quad \forall \text{SI} \rightarrow \text{SO paths} \tag{2}$$

$$b_{ij}, d_{ij}, T \in Z_+ \quad \forall e_{ij} \in E. \tag{3}$$

We assume that the problem input and output variables are nonnegative integers. This is particularly useful for application domains in which only discrete integer values are meaningful. Examples include high-level synthesis, which deals with discrete clock cycles, or grid routing, which can only handle discrete values for addressing grid coordinates.

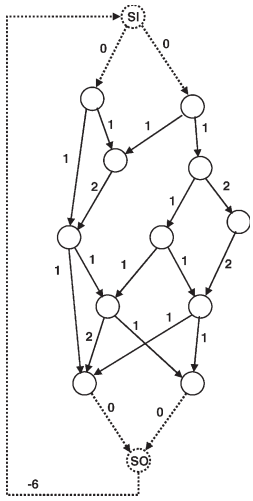


Fig. 2. Sample DAG with edge delay annotations. Nodes SI and SO and corresponding virtual edges are shown with dashed lines.

Relaxing the integral constraints to allow the variables to be any nonnegative number would form an LP formulation. LP problems are known to be solvable in polynomial time and, in that sense, are easier to solve. Moreover, in our previous work, we showed that the LP formulation of the budgeting problem has an optimal integer solution [13], which implies that the solution with integral constraint is optimal for the nonintegral version of the problem as well. Consequently, we assume that the variables should be nonnegative integer. Note that the problem becomes NP-hard if only arbitrarily discrete (as opposed to consecutive) integers are allowed for each variable [29].

DAGs are usually utilized to model the applications at different levels of abstraction. Examples include task graphs modeling a high-level computation at the task level, DFGs representing applications at the architectural level, and netlists modeling a gate-level combinational circuit. The problem and techniques presented in this paper are valid on any DAG. Therefore, they are quite generic and applicable at different levels of abstraction.

A. Reformulations Enabled by Problem Properties

We use a series of problem reformulations to transform our problem into one whose dual LP formulation has a known combinatorial solution. Our approach is inspired by the technique that Boros *et al.* developed for a graph balancing problem [10], [11]. We have adjusted and extended their results to the problem at hand (delay budget assignment).

The following lemma is a crucial observation that allows us to reformulate the problem.

Lemma 1: In an optimal budget assignment, the delay of any path from SI to SO is T.

It follows that for a given graph G, the cost of a path from node i to SO does not depend on the choice of the path and is

only a function of i. Let r_i be a variable assigned to each node i that represents its distance to SO. Therefore

$$r_i - r_j - d_{ij} = b_{ij} \quad \forall e_{ij} \in E. \tag{4}$$

Substituting this equation into (1)–(3) leads to the following set of equations:

$$\text{Max} \sum_{e_{ij} \in E} b_{ij} \tag{5}$$

$$r_i = r_j + d_{ij} + b_{ij} \quad \forall e_{ij} \in E \tag{6}$$

$$r_{\text{SI}} - r_{\text{SO}} \leq T \tag{7}$$

$$r_i, b_{ij} \in Z_+ \quad \forall v_i \in V \text{ and } e_{ij} \in E. \tag{8}$$

Note that the number of constraints in (1)–(3) can grow exponentially with respect to the number of nodes in the graph. However, formulation (5) has polynomial number of constraints with respect to the problem size. Moreover, we can assume that there is a virtual edge e_{oi} from SO to SI with $-T$ delay (Fig. 2). The constraint $r_{\text{SI}} - r_{\text{SO}} \leq T$ can be represented as one of the regular edge constraints and can be safely removed as a separate constraint. The example shown in Fig. 2 assumes that the timing constraint for the application is six time units.

Utilizing (4), we can eliminate b_{ij} variables from the objective function by substituting $b_{ij} = r_i - r_j - d_{ij}$. Note that the nonnegativity constraint of b_{ij} transforms to $\forall e_{ij} \in E$: $r_i - r_j \geq d_{ij}$. It follows that:

$$\begin{aligned} \sum_{e_{ij} \in E} b_{ij} &= \sum_{e_{ij} \in E} r_i - r_j - d_{ij} \\ &= \sum_{v_i \in V} r_i \left[\text{out}(v_i) - \text{in}(v_i)\right] - \sum_{e_{ij} \in E} d_{ij} \end{aligned}$$

where $\text{in}(v_i)$ and $\text{out}(v_i)$ are the in-degree and out-degree of vertex v_i, respectively. Note that the term $\sum_{e_{ij} \in E} d_{ij}$ is constant and can be eliminated from the objective function. Defining $\rho_i = \text{out}(v_i) - \text{in}(v_i)$, (5)–(8) can be rephrased as

$$\text{Max} \sum_{i \in V} \rho_i r_i \tag{9}$$

$$r_j - r_i \leq -d_{ij} \quad \forall e_{ij} \in E \tag{10}$$

$$r_i \in Z_+ \quad \forall v_i \in V \tag{11}$$

and the dual problem to the LP (9)–(11) is

$$\text{Min} \sum_{e_{ij} \in E} -d_{ij} y_{ij} \tag{12}$$

$$\sum_{e_{ki} \in E} y_{ki} - \sum_{e_{ij} \in E} y_{ij} = \rho_i \quad \forall v_i \in V \tag{13}$$

$$y_{ij} \in Z_+ \quad \forall e_{ij} \in E. \tag{14}$$

Interestingly, (12)–(14) formulate a conventional min-cost flow problem on the DAG, where y_{ij} variables are the amount of flow along edges e_{ij} with cost $-d_{ij}$, and ρ_i is the amount of demand at node i. Equivalently, $-\rho_i$ can be interpreted as the amount of flow supply at that node. Note that $\sum_{i \in V} \rho_i = 0$ is satisfied as required in the min-cost flow problem [10], [26].

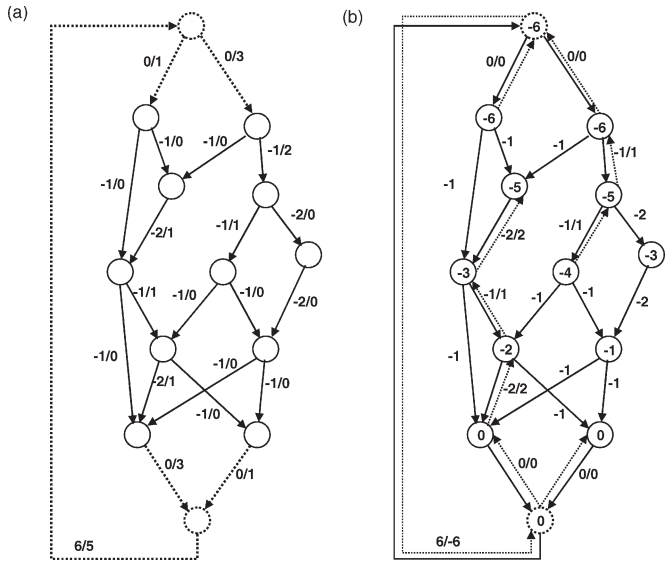


Fig. 3. (a) Dual min-cost flow problem/solution for example in Fig. 2. (b) Corresponding residual graph and edge costs.

It follows that the original problem can be solved optimally in polynomial time. Section V presents an algorithm that can determine the value of b_{ij} for each edge after solving the dual min-cost flow problem. Our solution is similar to that of [10] and [11].

V. Efficient Optimal Algorithm

The dual of the original edge delay budgeting problem can be stated as a min-cost flow problem on a new graph called $G'(V, E)$ (Section IV-A). G' and G are identical in terms of nodes and edges. However, the cost of edge e_{ij} is $-d_{ij}$ in G', and the amount of flow supply at node v_i is $-\rho_i = \text{in}(v_i) - \text{out}(v_i)$. The flow supply has to be satisfied at each node by a feasible flow solution. Note that the cost of e_{oi} is T. Hence, there is no negative cycle in the graph, and the dual problem can be solved by any of the well-known min-cost flow algorithms [26]. Fig. 3 illustrates the dual min-cost flow problem and its solution for the graph shown in Fig. 2. In Fig. 3(a), edges are annotated with their cost and flow, respectively. Supply at node i is $\text{in}(v_i) - \text{out}(v_i)$. In Fig. 3(b), nodes are annotated with their shortest path to SO (i.e., δ_i).

Once y_{ij} variables (the amount of flow along edge e_{ij}) are figured out, we can construct the residual graph $G_y(V, E')$ from $G'(V, E)$. For any edge e_{ij} in G' with nonzero flow along it, there are two edges e_{ij} and e_{ji} in the residual graph. The cost of each backward edge e_{ji} is d_{ij}, which is equal to the complement of the forward edge cost.

Let δ_i be the shortest distance of node i to SO in the residual graph G_y. There is no negative cost cycle in the residual graph G_y, hence, δ_i variables are well defined. δ_i variables can be determined by utilizing any well-known shortest path algorithm, such as Bellman–Ford algorithm [33], which is applicable to graphs with negative edge costs. Fig. 3 shows the

residual graph G_y and δ_i variables for the example shown in Fig. 3.

Variables r_i and b_{ij} of the primal problem can be easily calculated by substituting $r_i = -\delta_i$ and $b_{ij} = r_i - r_j - d_{ij}$. The following theorem proves that this simple equation determines the primal variables correctly.

Theorem 1: $r_i = -\delta_i$ is an optimal solution to the (9), where δ_i is the shortest path of node i to SO in the residual graph G_y.

Proof: Corresponding to each flow variable y_{ij} on edge e_{ij}, there is a constraint for that particular edge in the primal problem. According to the complementary slackness condition [9], we only need to assign values to r_i variables such that the corresponding constraints become an equality for edges with nonzero flow. We show that $r_i = -\delta_i$ satisfies this condition. Suppose that $y_{ij} > 0$ for an arbitrary edge e_{ij} in G'. Therefore, both edges e_{ij} and e_{ji} exist in G_y. Note that the cost of edges e_{ij} and e_{ji} are $-d_{ij}$ and d_{ij} in G_y, respectively. According to the shortest path definition, both of the following equations hold:

$$\delta_i \leq \delta_j - d_{ij}$$
$$\delta_j \leq \delta_i + d_{ij}.$$

Consequently, $r_i - r_j = \delta_j - \delta_i = d_{ij}$, which is exactly what complementary slackness condition implies. ■

Algorithm 1 Optimal assignment of budget values to edges of a DAG

Input: $G(V, E), d_{ij}, T$
Output: b_{ij}
Create $G'(V, E')$ with appropriate edge costs and node supplies;
Solve min-cost flow on G' and determine y_{ij};
Create the residual graph G_y;
for all $v_i \in V$ **do**
  Let $\delta_i =$ length of the shortest path from v_i to SO in G_y;
  Let $r_i = -\delta_i$;
end for
for all $e_{ij} \in E$ **do**
  Let $b_{ij} = r_i - r_j - d_{ij}$;
end for
Return b_{ij}.

Edge delay budgets are now easily calculated by $b_{ij} = r_i - r_j - d_{ij}$. This process is depicted in Algorithm 1. According to this algorithm, we first create the graph G', whose nodes are the same as graph G and whose edges have the cost equal to $-d_{ij}$. Furthermore, node v_i in G' will have a flow supply equal to $\text{in}(v_i) - \text{out}(v_i)$. Then, the min-cost flow on G' will be solved, and the residual graph G_y will be created. Note that for those edges in G' that have a nonzero flow going through them, there will be edges with reverse direction and cost in G_y. Finally, the complement of the shortest path of each node v_i to SO in the residual graph forms the variable r_i. Budget vector b is readily given once r_i variables are determined. Well-known techniques including min-cost flow and shortest path calculation are only used as black boxes here.

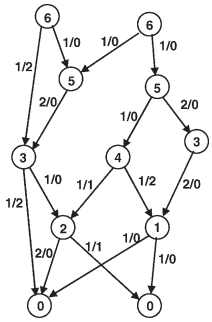


Fig. 4. Optimal budget assignment for example in Fig. 2. Numbers on each edge denote delay and assigned budget. Nodes are annotated with their delay to SO (r_i).

The time complexity of Algorithm 1 is determined by the time complexity of the min-cost flow step, which functions on the augmented graph G'. Assuming that the input graph G has n nodes, G' would also have $O(n)$ nodes. Therefore, the min-cost flow instance can be solved in $O(m \cdot \log(n) \cdot (m + \log(n)))$ via enhanced capacity scaling algorithm [26], where m is the number of edges in G'. For practical computer-aided design (CAD) problems, G is a sparse graph, and the degree of nodes is bound by a small constant. Hence, the number of edges is $O(n)$, and the time complexity reduces to $O(n^2 \cdot \log(n))$, which is quite affordable for many problem instances. In Section VIII, we will report our observations regarding the runtime of our algorithm on practical benchmarks, which advocate the practicality of our approach for CAD problems.

For the graph shown in Fig. 2, the amount of the budget assigned to each edge of the graph is illustrated in Fig. 4 utilizing Algorithm 1. Note that the number on each node is its distance to SO and does not depend on the choice of path (Lemma 1). The amount of budget on edge e_{ij} is readily given by $b_{ij} = r_i - r_j - d_{ij}$.

VI. Extensions to Other Budgeting Policies

The timing budget assigned to each of the design components can be exploited to improve its quality. The quality metric depends on the application domain, among which area, power dissipation, predictability, and cost are only a few examples. The objective function presented in Section IV (i.e., $\sum_{e_{ij} \in E} b_{ij}$) assumes that a unit of delay budget assigned to any component will lead to the same amount of savings in the particular design quality metric of interest. This is not the case, however, in many practical situations.

For example, the amount of utility improvement per unit budget for a particular component might be twice as much as another component. Moreover, a component might be able to utilize only a limited amount of extra delay budget. Similarly, designers might need a minimum amount of delay budget assigned to some component to be able to optimize it for a particular design metric.

In this section, we study the problem of delay budget assignment under various cost functions. We will show that many natural budget distribution policies are simple extensions of what we presented in Section IV.

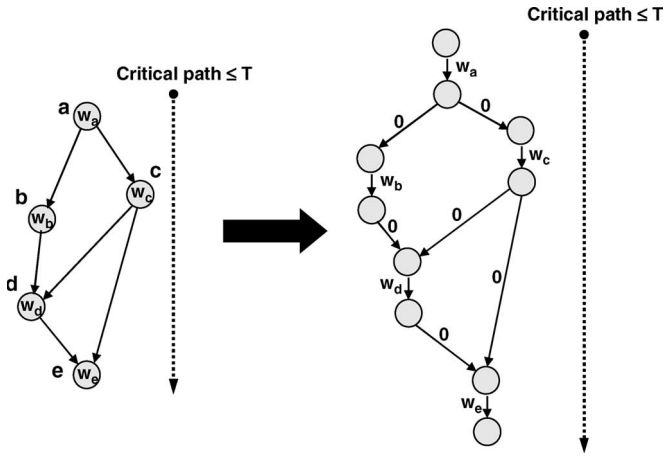


Fig. 5. Weighted node budgeting problem is transformed to weighted edge budgeting instance.

A. Weighted Budget Distribution

A unit delay budget can lead to different amounts of savings in the utility metric depending on the component that the budget is assigned to. In such cases, the budget assigned to different components of the application contributes to the cost function with different weights. Let nonnegative w_{ij} variables denote the weight of edge e_{ij}. The weighted budget assignment problem can be formulated as

$$\text{Max} \quad \sum_{e_{ij}\in E} w_{ij}\cdot b_{ij} \tag{15}$$

$$r_i = r_j + d_{ij} + b_{ij} \quad \forall e_{ij}\in E \tag{16}$$

$$r_{\text{SI}} - r_{\text{SO}} \le T \tag{17}$$

$$r_i, b_{ij} \in Z_+ \quad \forall v_i \in V \text{ and } e_{ij}\in E \tag{18}$$

and similar to what we did in Section IV-A, i.e.,

$$\begin{aligned}\sum_{e_{ij}\in E} w_{ij}\cdot b_{ij} &= \sum_{e_{ij}\in E} w_{ij}\cdot(r_i - r_j - d_{ij})\\ &= \sum_{v_i\in V} r_i\cdot\left(\sum_{v_j\in \text{out}(v_i)} w_{ij} - \sum_{k\in \text{in}(v_i)} w_{ki}\right)\\ &\quad - \sum_{e_{ij}\in E} w_{ij}\cdot d_{ij}.\end{aligned}$$

Note that the second term (i.e., $\sum_{e_{ij}\in E} w_{ij}\cdot d_{ij}$) is constant for a given problem instance. It follows that (15)–(18) can be transformed to (9)–(11) by defining $\rho_i = \sum_{v_j\in \text{out}(v_i)} w_{ij} - \sum_{k\in \text{in}(v_i)} w_{ki}$. Therefore, the algorithm described in Section V can handle the weighted version as well. The only required modification is to update the demand function ρ_i at nodes for the dual min-cost flow instance.

An interesting special case of the weighted edge budgeting problem solves the "node budgeting" problem where each node has a delay value, and we would like to assign budget values to nodes instead of edges. Fig. 5 shows an example for transforming a weighted node budgeting instance into a weighted edge budgeting problem. The only required transformation is to split each node into two other nodes that are connected by an edge with weight equal to the original node weight. All of the other regular edges will have zero weight. Note that in the final solution, some edges with zero weight might be assigned delay budgets to validate Lemma 1.

B. Bounded Budget Distribution

The delay budget assigned to each component can be exploited to some specific extent. Extra budget assigned to a component would potentially be wasted, i.e., it will not lead to any utility improvement. Similarly, a component might require a minimum amount of timing budget assigned to it to reflect any improvement in its utility function. In such cases, it is desirable to have a lower and/or upper bound on the delay of each edge. Budget lower bounds are easy to implement because they can be added to the edge delay at the first place. In other words, it is straightforward to create a problem instance that already satisfies the lower-bound requirement or find out that such a budget assignment solution is infeasible.

Upper bounds, however, are not as easy as the lower bounds to handle. In this section, we address the problem of maximum budget assignment under upper-bound constraints on edges. We show that this problem boils down to solving the min-cost flow on a modified network and is similar to what we presented in Section IV.

Assume that there is an upper bound u_{ij} for the delay of edge e_{ij}. Therefore, (9)–(11) change into

$$\text{Max}\sum_{i\in V}\rho_i r_i \tag{19}$$

$$r_j - r_i \le -d_{ij} \quad \forall e_{ij}\in E \tag{20}$$

$$r_i - r_j \le u_{ij} \quad \forall e_{ij}\in E \tag{21}$$

$$r_i \in Z_+ \quad \forall v_i \in V \tag{22}$$

whose dual problem is

$$\text{Min}\sum_{e_{ij}\in E} u_{ij}z_{ij} - d_{ij}y_{ij} \tag{23}$$

$$\sum_{e_{ki}\in E}(y_{ki}-z_{ki}) - \sum_{e_{ij}\in E}(y_{ij}-z_{ij}) = \rho_i \quad \forall v_i\in V \tag{24}$$

$$y_{ij}, z_{ij}\in Z_+ \quad \forall e_{ij}\in E. \tag{25}$$

Equations (23)–(25) formulate a min-cost flow problem on a network that is built by the following rule: For every edge e_{ij} with cost $-d_{ij}$, there is a reverse edge e_{ji} with cost u_{ij} (Fig. 6). y_{ij} and z_{ij} are the amount of flow along edges e_{ij} and e_{ji}, respectively. Note that by definition, $d_{ij}\le u_{ij}$. Hence, there is no negative cycle introduced into the network. Again, this problem can be solved using standard min-cost flow techniques. The solution to the primal problem can be determined after knowing y_{ij} and z_{ij} values. Fig. 6 illustrates an example of transforming a bounded edge budgeting problem to a conventional unbounded instance. In this example, nodes

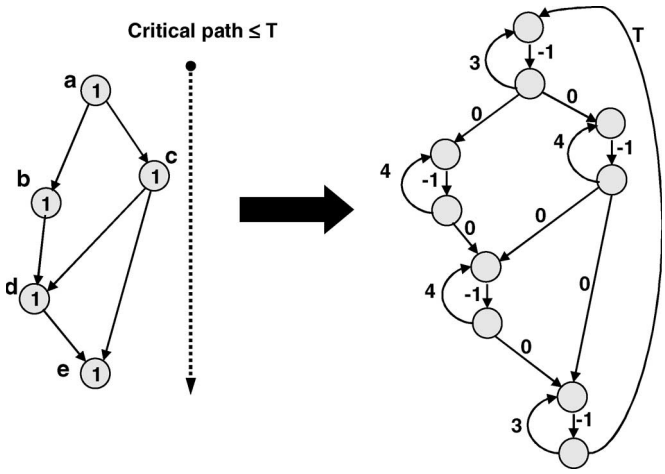


Fig. 6. Bounded edge budgeting problem is transformed to conventional unbounded instance.

a and e are assumed to have the upper bound of 3 on their delay. The rest of the nodes can have at most four units of delay.

C. Min−Max Budget Distribution

Fair distribution of the available budget to application components is another useful objective. Fair distribution can be quantified as minimizing the maximum budget assigned to graph edges or minimizing the budget skew (the difference between the maximum and the minimum allotted budgets).

The problem of fair budgeting would have trivial ineffective solutions that minimize the maximum assigned budget or budget skew if fairness was the only objective. For example, the zero budget assignment minimizes the maximum assigned budget. Similarly, one might increment the budget on each edge as long as the timing constraints are not violated, which leads to minimizing the budget skew. In practice, fair budget distribution has to be considered as a supplementary objective to maximizing the total budget. For example, one might look for fair budget assignments among the solutions whose total budget is at least 80% of the total budget of an optimal maximum budget solution (with no fairness constraint).

For both fair and min-skew budgeting, the problem has to be initially solved using the algorithm described in Section IV-A to determine the maximum budget that can be assigned to the application nodes. Assuming that minimizing the maximum assigned budget (min−max) is the objective, we can perform a binary search on the budget upper bounds to choose the best solution that has a reasonable total budget distribution. It follows that the algorithm complexity is only $O(\log(T))$ times more than that of the Algorithm 1. Similarly, we can search on edge lower and upper bounds at the same time to minimize the budget skew (the difference between the maximum and minimum assigned budget). However, this can increase the time complexity by a factor of $O(T^2)$ in the worst case.

VII. Potentials for Incremental Budgeting

In many practical situations, the delay of one or only a few components might change during the design flow iterations.

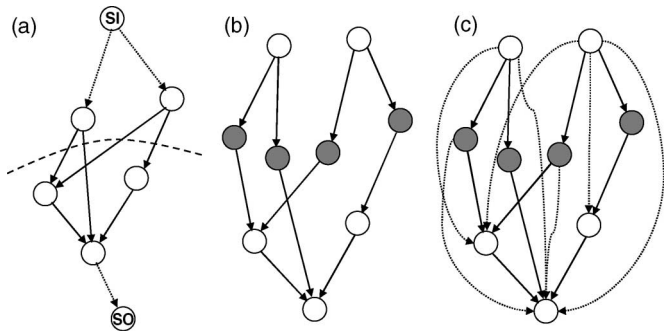


Fig. 7. (a) Sample DAG and sample cut. (b) Corresponding edge graph. Edges in cut correspond to dark nodes. (c) Transitive graph of edge graph.

Examples include, but are not limited to, library binding and physical design. Such problem instances can have millions of nodes in their representing graphs. Therefore, it is often impractical to reexecute Algorithm 1 to find a new budget assignment for each local change and its corresponding problem instance. In such cases, it is often required to transform the current solution to a new feasible solution by performing local, rapid, and incremental calculations.

In this section, we present some interesting properties of the problem formulated in Section IV. These properties can be further exploited to tackle other practical budgeting problems including incremental budget assignment.

Lemma 2: Given a solution for the dual LP problem of a budgeting instance, the flow variables do not change with the increase of the timing constraint T.

Proof: There is no negative cycle in G'. Therefore, all of the paths that do not include $e_{\rm oi}$ have costs not less than $-T$ (Fig. 3). The min-cost flow algorithms work based on the augmenting path idea. In each iteration, a path with minimum cost from a source to a sink is found, and the flow along that path is added to the solution. Therefore, increasing T has no effect on the flows that run along paths that do not contain $e_{\rm oi}$. Note that all such paths have negative cost.

However, edge $e_{\rm oi}$ is the only "backward" edge of the graph G' and is common among the rest of the flows. Increasing its cost will increase the cost for all such paths; however, it will not provide a change in the choice of the path for min-cost flow algorithm. It follows that the flow solution and the corresponding variables will not change by increasing T. ■

Moreover, it is easy to update the shortest path variables δ_i in the new residual graph. If the shortest path from a node to SO passes through $e_{\rm io}$ in the original G_y, the value of the path is decreased by the amount of increase in T. For other nodes, δ_i variables remain intact. It follows that if the timing constraint T is increased by Δ_T, the budget of a certain set of edges will be increased by Δ_T. In other words, we start from the original budget assignment that has been carried out under timing constraint T. Δ_T is then added to the budget of a particular set of edges.

Definition 2: In graph $G(V, E)$, a subset of edges is called a "cut" if and only if every SI to SO path contains exactly one edge of the set (Fig. 7). Graph $G^*(V^*, E^*)$ is called the intersection graph (or edge graph) of $G(V, E)$ if there is a node $v^*_{ij} \in V^*$ for every $e_{ij} \in E$ and if there is an edge e^*_{ijk} between

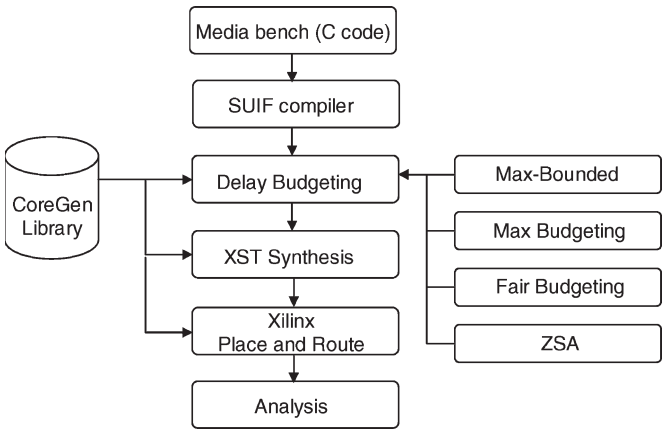


Fig. 8. Delay budgeting in core-based FPGA CAD flow.

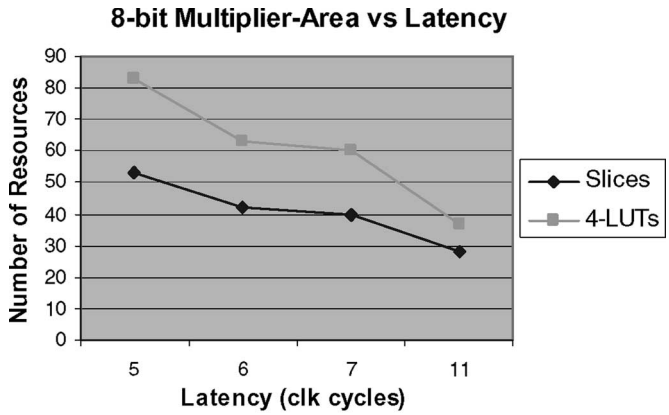


Fig. 9. Area characteristic of Xilinx CoreGen sequential multiplier cores.

TABLE I
CHARACTERISTICS OF BENCHMARK DFGS

	DFG Name	Application	# of nodes	Latency
dfg0	Invert-matrix	mesa	56	17
dfg1	gl-rotation-matrix	mesa	63	16
dfg2	matmul	mesa	85	19
dfg3	jpeg-idct-float	jpeg	85	24
dfg4	gl-translatef	mesa	66	18
dfg5	jpeg-idct-float	jpeg	93	27
dfg6	invert-matrix-general	mesa	141	23
dfg7	gl-frustum	mesa	51	15
dfg8	invert-matrix	mesa	144	28
dfg9	jpeg-idct-float	jpeg	24	11

v_{ij}^* and v_{jk}^*. Note that a cut in G corresponds to an "independent set" of the transitive graph of G^* (G^{t*}). In the transitive graph, if there is an edge from node v_x to v_y and from v_y to v_z, there is also an edge from v_x to v_z. We use G^t to represent the transitive graph of G.

Definition 3: Let $\text{Gain} = \text{OPT}(G, T)$ denote the maximum amount of delay budget that can be added to graph G under timing constraint T. Let Graph G_b be the new graph that is formed by adding the delay budgets to the edges of G.

Lemma 3: For a given instance of the edge budgeting problem with critical path equal to T

$$\begin{aligned}\text{Gain} &= \text{OPT}(G, T + \Delta_T) \\ &= \text{OPT}(G, T) + \text{OPT}(G_b, \Delta_T) \\ &= \text{OPT}(G, T) + \Delta_T \left|\text{MIS}\left(G_b^{t*}\right)\right| \\ &= \text{OPT}(G, T) + \Delta_T \left|\text{MIS}\left(G^{t*}\right)\right|\end{aligned}$$

i.e., if the timing constraint T is increased by Δ_T, the budget of the edges that form a max-cut (a cut with the maximum cardinality) will be increased by Δ_T. Such edges correspond to the maximum independent set in the transitive intersection graph of the problem instance and can be found using existing methods (Fig. 7) [8]. Therefore, incremental calculation of the budget assignment and edge delay budgets for various values of T can be performed optimally.

The following lemma assists in performing budget reassignment and incremental budget management.

Lemma 4: Let G_b denote the graph after budget assignment. Let c_1 and c_2 represent two cuts in G_b, where all edges in c_1 are assigned at least δ units of budget. Decreasing the budget of all of the edges in c_1 by δ and increasing the budget of all of the edges in c_2 by δ lead to another maximal feasible solution (one that does not violate the timing constraint and all paths take exactly T).

VIII. DELAY BUDGET ASSIGNMENT DURING LIBRARY MAPPING

A. Experimental Setup

We applied delay budgeting during library mapping for a given data path. Fig. 8 illustrates our synthesis flow for mapping an application to an FPGA device. The application DFGs are extracted from MediaBench [6] test suite using SUIF compiler [23] and Machine-suif [21]. We selected ten of the largest DFGs as our benchmarks. The characteristics of the benchmarks are shown in Table I. For experimentation purposes, we assumed that each of the operands is 8-bits wide. We also assumed that the timing constraint for each application is equal to its critical path latency plus four extra cycles. The extra cycles are allowed to better separate different budgeting policies and highlight their differences.

We used Xilinx CoreGen [17] to generate parameterized hardware modules (cores) with different latencies. Xilinx synthesis (XST) and placement and routing tools [17] have been used in our experimental flow to implement the designs and measure their area requirement and timing. We also recorded the runtime for solving the budgeting instance and placement and routing of the synthesized design to the FPGA. Our FPGA target platform is Xilinx VirtexE device XCV3200V with FG1156 package and speed grade -8. The Xilinx Integrated Software Environment (Xilinx ISE) version 6.3 was used for the experiments.

The major operations in the selected application DFGs are addition, subtraction, multiplication, division, and shifting. We synthesized the CoreGen library modules corresponding to the aforementioned operations to characterize their area variations with respect to latency. Fig. 9 demonstrates the area characteristics of the CoreGen sequential multiplier core. The CoreGen

shifting, division, addition, and subtraction cores implement latency variation by inserting registers and pipelining the operation. Therefore, slower implementations of shifting, division, addition, and subtraction consume more area. Consequently, in our experiment, we assigned the timing budget only to multipliers of the application DFG.

The library cores generated by CoreGen do not provide a continuous range of latencies to choose from. CoreGen sequential multipliers can have the latency of 5, 6, 7, or 11 cycles. The points corresponding to available latencies are illustrated in Fig. 9. The figure also demonstrates the tradeoff between area and latency of the multipliers in the library. The discontinuity in core latency motivates the bounded budget assignment to limit the amount of timing budget that a library core can accept. For example, more than two units of delay budget on the fastest multiplier core is likely to be "wasted" because the next choice of multiplier core requires four more delay units (multiplier with 11 cycle latency).

We have implemented seven different budget assignment policies to compare their impact on datapath area. They include maximum budgeting, maximum budgeting with upper bounds, three variations of fair budgeting (minimizing the maximum budget with a lower-bound constraint on the total budget), weighted budgeting, and ZSA [27]. ZSA is widely used in industrial tools and is considered to be the major competitor for our algorithms.

For bounded budgeting, we impose the upper bound of two units of delay for assigning delay budgets to each multiplier core. Fair budgeting, described in Section VI-C, minimizes the maximum budget assigned to the nodes while meeting the lower-bound constraint on the total budget assigned to the DFG. In our experiment, fair budgeting has been implemented to minimize the maximum budget while allowing 10%, 25%, or 40% degradation in total budget as compared with maximum budgeting.

B. Experimental Results

The results of our experiments are summarized in Table II. For each benchmark, area requirement in terms of both lookup tables and slices, design timing in nanoseconds, and tool runtime (synthesis, placement and routing tools) in seconds are reported. In addition, the total delay relaxation and the number of nodes whose delay is relaxed are also reported in the table. The last two columns of the table compare the improvement of our best budgeting technique (max, bounded, and fair) with ZSA and no budgeting. The optimization runtime to solve the generated budgeting problem instance according to our formulation was less than 0.3 s for our benchmarks and, therefore, was neglected in comparison to large physical design time. The runtimes are recorded on a PC running Linux on a Pentium processor at 2.8 GHz with 512-MB memory.

Table II shows no single budgeting algorithm that generates the minimum area results for all ten benchmarks; the fair distribution with 75% lower bound on total budget produces the best results on the average. This highlights the fact that distribution of delay budget along with maximization of total budget leads to the best results. Conventional methods often try to maximize the total budget and neglect the effect of budget distribution. Our techniques improve the LUT count by 28.7% and 13.3% as compared to not using budgeting or using the ZSA, respectively. The improvements are 25.8% and 11.5% in terms of slice count.

Note that some DFGs have many unrelated operations that can be slowed down as much as the application timing constraint. For example, DFG9 has many unrelated multiplication operations. For these cases, bounded and fair budgeting do not perform as well as the maximum budgeting because they do not assign maximum latency to unrelated tasks. In the case of DFG9, ZSA performs as well as the maximum budgeting technique and better than the bounded or fair budgeting. This is due to the fact that ZSA algorithm distributes delay budgets similar to maximum budgeting for unrelated operations.

For benchmarks that are dominated with unrelated operations, maximum budgeting performs similar to or better than other policies due to its maximization objective and distribution biproduct for unrelated operations. Because it is easy to detect and handle such situations, we decided not to differentiate between unrelated/related operations and did not process these benchmarks differently. Our algorithms can be further optimized if they are tuned to adopt the appropriate budget assignment policy according to DFG topology.

Table II illustrates the timing and tool runtime associated with each design. The variations in timing numbers are small among different budgeting algorithms, ZSA, and no budgeting. Therefore, the small difference in timing can be neglected and attributed to noise. However, the benchmarks that are processed by some budget assignment algorithm (either ZSA or one of our algorithms) slow down the synthesis process by about 82% on average as compared with original designs that do not go through component delay relaxation. The additional tool runtime overhead can be attributed to the fact that relaxed operations diversify the type of operations existing in the design, and each of them requires additional time for physical design. On the other hand, there is only one type of core in the original designs, and the physical synthesizer can reuse the placement and/or routing information for each core. Note that our budgeting techniques incur no additional cost when compared with ZSA, in terms of required tool runtime.

Our methodology provides a framework to find the proper budget assignment policy that is suitable for a particular application and objective function. After exploration of the budget assignment policies, the suggested policy constantly outperforms ZSA, which is widely used in today's industrial tools. The "Budgeting vs. ZSA" column in Table II shows the improvement of the best budgeting policy, found through our methodology, over ZSA.

IX. Conclusion

We presented a theoretical framework that unifies a large class of existing time-management paradigms. We developed a totally combinatorial framework through which we optimally solved several time budgeting problems, including maximizing total budget, weighted, bounded, and fair budgeting. In addition, our technique is applicable to time management for edges,

TABLE II
AREA, RUNTIME, TIMING, AND BUDGET DISTRIBUTION COMPARISON AMONG DIFFERENT DELAY BUDGETING ALGORITHMS

Benchmark	Design Metric	Delay Budgeting Algorithms							Budgeting vs. ZSA (%)	Budgeting vs. No-budget (%)
		No-budget	Max	Bounded	Fair (90%)	Fair (75%)	Fair (60%)	ZSA		
DFG0	LUT count	2698	2146	2008	2146	2075	2080	2369	15.24	25.57
	Slice count	1694	1385	1304	1385	1344	1345	1512	13.76	23.02
	Timing	10.98	11.21	11.13	11.21	11.19	11.20	11.20	0.60	-1.38
	Run time	231	423	421	421	420	421	415	-1.20	-81.82
	Total budget	0	119	60	107	91	72	86	24.42	-
	Relaxed nodes	0	21	30	21	28	27	21	42.86	-
DFG1	LUT count	3348	2267	2566	2267	2267	2635	2931	22.65	32.29
	Slice count	2351	1759	1909	1759	1759	1951	2115	16.83	25.18
	Timing	11.36	12.85	12.76	12.85	12.85	12.78	12.63	-1.02	-12.30
	Run time	233	444	443	444	443	444	441	-0.45	-90.13
	Total budget	0	223	68	207	169	134	126	76.98	-
	Relaxed nodes	0	28	34	28	28	28	28	21.43	-
DFG2	LUT count	3680	2821	2752	2829	2784	2760	3181	13.49	25.22
	Slice count	2332	1860	1808	1864	1828	1812	2035	11.15	22.47
	Timing	10.63	10.96	10.96	10.96	10.95	10.79	10.81	0.24	-1.46
	Run time	236	409	410	409	411	410	407	-0.49	-73.31
	Total budget	0	159	80	143	120	95	123	29.27	-
	Relaxed nodes	0	24	40	24	40	40	26	53.85	-
DFG3	LUT count	2330	1597	1824	1597	1597	1735	1827	12.59	31.46
	Slice count	1418	1017	1132	1017	1017	1089	1141	10.87	28.28
	Timing	10.77	11.02	11.04	11.02	11.02	11.03	11.03	0.04	-2.40
	Run time	235	426	424	425	426	424	424	0.00	-80.43
	Total budget	0	152	44	137	114	91	88	72.73	-
	Relaxed nodes	0	19	22	19	19	19	20	10.00	-
DFG4	LUT count	2928	2284	2192	2284	2284	2388	2664	17.72	25.14
	Slice count	1836	1480	1420	1480	1480	1532	1688	15.88	22.66
	Timing	11.33	10.86	10.76	10.86	10.86	10.92	10.86	0.93	5.00
	Run time	233	481	483	481	481	480	480	0.00	-106.01
	Total budget	0	144	64	132	108	86	87	65.52	-
	Relaxed nodes	0	20	32	20	20	24	21	52.38	-
DFG5	LUT count	3495	2509	2670	2509	2469	2463	2742	10.18	29.53
	Slice count	2173	1632	1707	1632	1610	1606	1760	8.75	26.09
	Timing	10.66	10.93	10.91	10.93	10.93	10.90	10.91	0.11	-2.26
	Run time	237	443	441	442	444	445	441	0.00	-86.08
	Total budget	0	287	71	258	215	172	156	83.97	-
	Relaxed nodes	0	27	36	27	29	29	28	28.57	-
DFG6	LUT count	6844	5188	5292	5128	5056	5648	6169	18.04	26.13
	Slice count	4304	3392	3428	3359	3318	3628	3924	15.44	22.91
	Timing	11.13	11.48	11.52	11.45	11.45	11.53	11.43	-0.15	-2.81
	Run time	250	416	415	416	416	415	411	-0.97	-66.00
	Total budget	0	370	132	339	281	236	218	69.72	-
	Relaxed nodes	0	48	68	51	54	52	51	33.33	-
DFG7	LUT count	2512	1528	2135	1526	1480	1810	1827	18.99	41.08
	Slice count	2078	1295	1687	1250	1224	1424	1470	16.73	41.10
	Timing	10.33	10.82	10.67	10.89	10.77	10.82	10.78	1.03	-3.30
	Run time	229	418	416	419	419	417	417	0.24	-81.66
	Total budget	0	205	50	193	155	123	126	62.70	-
	Relaxed nodes	0	22	25	22	24	24	22	13.64	-
DFG8	LUT count	7325	5210	5605	5210	4934	4701	5379	12.60	35.82
	Slice count	4688	3481	3688	3487	3334	3257	3643	10.60	30.52
	Timing	12.18	12.88	12.74	12.88	12.77	12.63	12.67	0.28	-3.72
	Run time	250	425	423	424	424	425	424	0.24	-69.20
	Total budget	0	710	148	645	533	426	322	120.50	-
	Relaxed nodes	0	48	77	48	57	65	61	26.23	-
DFG9	LUT count	1017	649	833	833	833	833	649	0	36.18
	Slice count	635	435	531	531	531	531	435	0	31.50
	Timing	11.38	10.92	10.83	10.83	10.83	10.83	10.66	-1.61	4.76
	Run time	207	398	392	392	392	392	395	0.76	-89.37
	Total budget	0	48	16	16	16	16	48	0	-
	Relaxed nodes	0	8	8	8	8	8	8	0	-
Average	LUT count	3617.7	2619.9	2787.7	2632.9	2577.9	2705.3	2973.8	13.31	28.74
	Slice count	2350.9	1773.6	1861.4	1776.4	1744.5	1817.5	1972.3	11.55	25.79
	Timing	11.07	11.39	11.33	11.39	11.36	11.34	11.30	-0.31	-2.34
	Run time	234.1	428.3	426.8	427.3	427.6	427.3	425.5	-0.31	-82.32
	Total budget	0	241.7	73.3	217.7	180.2	145.1	138	75.14	-
	Relaxed nodes	0	26.5	37.2	26.8	30.7	31.6	28.6	30.07	-

nodes, and hybrid combination of these two elements in a graph representation of the applications. We performed experimental results on mapping some applications onto Xilinx FPGAs. We generated the applications' datapath components using CoreGen intellectual property cores. We compared different time budgeting policies in terms of the design area under equal timing constraints. Experimental results exhibit significant savings in design quality (area in our experiments) and advocate our theoretical results. Future works include extension of the idea to discrete and incremental time budgeting.

REFERENCES

[1] A. Kahng, S. Mantik, and I. L. Markov, "Min-Max placement for large-scale timing optimization," in *Proc. ACM Int. Symp. Phys. Design*, 2002, pp. 143–148.

[2] A. Srivastava, "Simultaneous Vt selection and assignment for leakage optimization," in *Proc. Int. Symp. Low Power Electron. Design*, 2003, pp. 146–151.

[3] C. Chen, E. Bozorgzadeh, A. Srivastava, and M. Sarrafzadeh, "Budget management with applications," *Algorithmica*, vol. 34, no. 3, pp. 261–275, Jul. 2002.

[4] C. Chen, X. Yang, and M. Sarrafzadeh, "Potential slack: An effective metric of combinational circuit performance," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 198–201.

[5] C. Kuo and A. C. H. Wu, "Delay budgeting for a timing-closure-design method," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, 2000, pp. 202–207.

[6] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: A tool for evaluating and synthesizing multimedia and communications systems," in *Proc. Int. Symp. Microarchitecture*, 1997, pp. 330–335.

[7] C. Yeh and M. Marek-Sadowska, "Delay budgeting in sequential circuit with application on FPGA placement," in *Proc. ACM/IEEE Design Automation Conf.*, 2003, pp. 202–207.

[8] D. Kagaris and S. Tragoudas, "Maximum independent sets on transitive graphs and their applications in testing and CAD," in *Proc. Int. Conf. Computer-Aided Design*, 1997, pp. 736–740.

[9] G. B. Dantzig, *Linear Programming and Extensions*. Princeton, NJ: Princeton Univ. Press, 1963.

[10] E. Boros, P. Hammer, and R. Shamir, "A polynomial algorithm for balancing acyclic data flow graphs," *IEEE Trans. Comput.*, vol. 41, no. 11, pp. 1380–1385, Nov. 1992.

[11] E. Boros, P. Hammer, M. Hartmann, and R. Shamir, "Balancing problems in acyclic networks," *Discr. Appl. Math.*, vol. 49, no. 1–3, pp. 77–93, Mar. 1994.

[12] E. Bozorgzadeh, S. Ghiasi, A. Takahashi, and M. Sarrafzadeh, "Optimal integer delay budgeting on directed acyclic graphs," in *Proc. Design Automation Conf.*, Jun. 2003, pp. 920–925.

[13] ——, "Optimal integer delay budget assignment on directed acyclic graphs," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 23, no. 8, pp. 1184–1199, Aug. 2004.

[14] E. Felt, E. Charbon, E. Malavasi, and A. Sangiovanni-Vincentelli, "An efficient methodology for symbolic compaction of analog IC's with multiple symmetry constraints," in *Proc. Conf. Eur. Design Automation*, Nov. 1992, pp. 148–153.

[15] F. Xie, M. Martonosi, and S. Malik, "Intraprogram dynamic voltage scaling: Bounding opportunities with analytic modeling," *ACM Trans. Archit. Code Optimization*, vol. 1, no. 3, pp. 323–367, Sep. 2004.

[16] H. R. Lin and T. Hwang, "Power reduction by gate sizing with path-oriented slack calculation," in *Proc. IEEE ASPDAC*, 1995, pp. 7–12.

[17] Xilinx Inc., (2006). *Xilinx Documentations and Online Manuals*. [Online]. Available: http://www.xilinx.com

[18] J. Luo and N. Jha, "Battery-aware static scheduling for distributed real-time embedded systems," in *Proc. IEEE/ACM Design Automation Conf.*, 2001, pp. 444–449.

[19] J. F. Lee and D. T. Tang, "VLSI layout compaction with grid and mixed constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-6, no. 5, pp. 903–910, Sep. 1987.

[20] M. Sarrafzadeh, D. Knol, and G. E. Tellez, "Unification of budgeting and placement," in *Proc. ACM/IEEE Design Automation Conf.*, Jun. 1997, pp. 758–761.

[21] M. D. Smith and G. Holloway. (2005). "An introduction to machine SUIF and its portable libraries for analysis and optimization," Division Eng. Appl. Sci., Harvard Univ. Cambridge, MA. [Online]. Available: http://www.eecs.harvard.edu/hube/software/nci/overview.html

[22] M. Sarrafzadeh, D. A. Knol, and G. E. Tellez, "A delay budgeting algorithm ensuring maximum flexibility in placement," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 16, no. 11, pp. 1332–1341, Nov. 1997.

[23] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, L. Shih-Wei, E. Bugnion, and M. S. Lam, "Maximizing multiprocessor performance with the SUIF compiler," *Computer*, vol. 29, no. 12, pp. 84–89, Dec. 1996.

[24] O. Coudert, "Timing and design closure in physical design flows," in *Proc. IEEE Int. Symp. Quality Electron. Des.*, 2002, pp. 511–516.

[25] P. Girard, C. Landrault, S. Pravossoudovitch, and D. Severac, "A gate resizing technique for high reduction in power consumption," in *Proc. Int. Symp. Low Power Electron. Design*, 1997, pp. 281–286.

[26] T. Magnanti, R. Ahuja, and J. Orlin, *Network Flows: Theory, Algorithms, and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1993.

[27] R. Nair, C. Berman, P. Hauge, and E. Yoffa, "Generation of performance constraints for layout," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 8, no. 8, pp. 860–874, Aug. 1989.

[28] S. Bakshi and D. Gajski, "Component selection for high-performance pipelines," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 2, pp. 181–194, Jun. 1996.

[29] S. Ghiasi, "Efficient implementation selection via time budgeting: Complexity analysis and leakage optimization case study," in *Proc. Int. Conf. Computer Design*, 2005, pp. 127–129.

[30] S. Ghiasi, E. Bozorgzadeh, S. Choudhury, and M. Sarrafzadeh, "A unified theory of timing budget management," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2004, pp. 653–659.

[31] S. Ghiasi, K. Nguyen, E. Bozorgzadeh, and M. Sarrafzadeh, "On computation and resource management in networked embedded systems," in *Proc. Int. Conf. Parallel Distrib. Computer Syst.*, 2003, pp. 445–451.

[32] S. Raje and M. Sarrafzadeh, "Scheduling with multiple voltages," *Integration*, vol. 23, no. 1, pp. 37–59, Oct. 1997.

[33] R. Rivest, T. Cormen, and C. Leiserson, *An Introduction to Algorithms*. Cambridge, MA: MIT Press, 1990.

[34] T. Kong, "A novel net weighting algorithm for timing-driven placement," in *Proc. ACM/IEEE Int. Conf. Computer-Aided Design*, 2002, pp. 172–176.

[35] W. Zhang, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, D. Duarte, and Y. Tsai, "Exploiting VLIW schedule slacks for dynamic and leakage energy reduction," in *Proc. ACM/IEEE Int. Symp. Microarchitecture*, 2001, pp. 102–113.

[36] Y. Liao and C. K. Wong, "An algorithm to compact a VLSI symbolic layout with mixed constraints," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. CAD-2, no. 2, pp. 62–69, Apr. 1983.

Soheil Ghiasi (S'98–M'05) received the B.S. degree in computer engineering from Sharif University of Technology, Tehran, Iran, in 1998, and the M.S. and Ph.D. degrees in computer science from the University of California, Los Angeles (UCLA), in 2002 and 2004, respectively.

Currently, he is an Assistant Professor with the Department of Electrical and Computer Engineering, University of California, Davis. His research interests include different aspects of embedded and reconfigurable system design.

Dr. Ghiasi received the Harry M. Showman prize from UCLA College of Engineering in 2004.

Elaheh Bozorgzadeh (S'01–M'03) received the B.S. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 1998, the M.S. degree in computer engineering from Northwestern University, Evanston, IL, in 2000, and the Ph.D. degree in computer science from the University of California, Los Angeles, in 2003.

She is currently an Assistant Professor with the Department of Computer Science, University of California, Irvine. Her research interests include computer-aided design for field-programmable gate arrays, reconfigurable computing, and design automation for embedded systems. She has authored three book chapters and more than 30 journal and conference papers.

Dr. Bozorgzadeh is a member of the Association for Computing Machinery.

Po-Kuan Huang (S'05) received the B.S. degree in electrical engineering from National Cheng Kung University, Taiwan, R.O.C., in 2002. He is currently working toward the Ph.D. degree in computer engineering at the Department of Electrical and Computer Engineering, University of California, Davis.

His research interests include the areas of embedded system design and design automation for electronic system.

Roozbeh Jafari (S'99) received the B.Sc. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2000, the M.Sc. degree in electrical engineering from the State University of New York, Buffalo, in 2002, and the M.S. degree in computer science from the University of California, Los Angeles (UCLA), in 2004. He is currently working toward the Ph.D. degree in computer science from UCLA.

He was with IBM, Endicott, NY, where he worked on the development of IBM TestBench tool designed for very large scale integration testing. His research is primarily in the area of networked embedded system design and reconfigurable computing with emphasis in medical/biological applications and their algorithm design.

Majid Sarrafzadeh (S'87–M'87–SM'91–F'96) received the B.S., M.S., and Ph.D. degrees from the University of Illinois, Urbana-Champaign, in 1982, 1984, and 1987, respectively, all in electrical and computer engineering.

He joined Northwestern University as an Assistant Professor, in 1987. In 2000, he joined the Department of Computer Science, University of California, Los Angeles (UCLA). He has collaborated with many industries in the past 15 years including IBM, Motorola, and many CAD industries. He is the architect of the physical design subsystem of Monterey Design Systems' main product. He is a cofounder of Hier Design, Inc. He has published approximately 250 papers and is the coeditor of the book *Algorithmic Aspects of VLSI Layout* (World Scientific, 1994) and the coauthor of the book *An Introduction to VLSI Physical Design* (McGraw-Hill, 1996). His recent research interests include the area of embedded and reconfigurable computing, very large scale integration (VLSI) computer-aided design (CAD), and design and analysis of algorithms.

Dr. Sarrafzadeh is a Fellow of the IEEE for his contribution to theory and practice of VLSI design. He has served on the technical program committee of numerous conferences in the area of VLSI design and CAD, including ICCAD, DAC, EDAC, ISPD, FPGA, and DesignCon. He has served as Committee Chairs of a number of these conferences. He is on the executive committee/steering committee of several conferences such as ICCAD, ISPD, and ISQED. He is the Program Committee Chair of ICCAD 2004. He is an Associate Editor of the *ACM Transaction on Design Automation* and an Associate Editor of the IEEE TRANSACTION ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS. He received an NSF Engineering Initiation Award, two Distinguished Paper Awards in ICCAD, and the Best Paper Award in DAC.