A Unified Theory of Timing Budget Management

Soheil Ghiasi† Elaheh Bozorgzadeh‡ Siddharth Choudhuri‡ Majid Sarrafzadeh†

† Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095, USA
{soheil,majid}@cs.ucla.edu

‡ Computer Science Department
University of California, Irvine
Irvine, CA 92697, USA
{eli, sid}@ics.uci.edu

Abstract

This paper presents a theoretical framework that optimally solves many open problems in time budgeting. Our 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. We show that many of the time management problems can be transformed into a min-cost flow instance that can be optimally and efficiently solved through well-known combinatorial techniques. Experiments include mapping of several designs, which are implemented using parameterized CoreGen IP cores, on Xilinx FPGA devices. Different time budgeting policies have been applied during the mapping stage. Our time management techniques always improved the area requirement of the implemented testbenches compared to a widely-used path-based method. We also compared the maximum budgeting and fairness in delay budget assignments. Our experimental results show that an average improvement of 19% *in area can be achieved when fairness and maximum budgeting policies are combined, compared to pure maximum budgeting.*

1. 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 requires the system level constraints to be translated into component level constraints. 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 in order to slow down as many components as possible without violating the system's timing constraints. The slowed-down components can be further optimized to improve system's area, power dissipation, or other design quality metrics.

However, almost all of the previous research efforts employ sub-optima heuristics for addressing the reasonable formulations of the budgeting problem. In our previous work [11], we solved the problem of integral budget assignment optimally through LP relaxation. However, combinatorial methods are often preferred to LP-based approaches due to their numerical instability and slow runtimes.

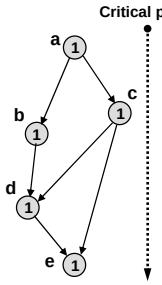
In this paper, we present a unified theoretical framework that can solve different well-known formulations of the budgeting problem through efficient combinatorial techniques. We model the given application as a directed acyclic graph (DAG) and assign timing budget values to the edges of the DAG. Many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

Timing budget management is applied to different design tasks such as gate and wire sizing, and library mapping. To experiment our theoretical results, we apply our technique during the library mapping at datapath level. We integrate the time budgeting and module selection into the synthesis flow for mapping applications onto 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.

Next section presents some applications and related works. Section 3 formally formulates the problem. Section 4 presents a combinatorial optimal algorithm for solving the formulated problem. The problem is then extended to other budgeting policies in Section 5. We also show that all of the extensions are amenable to our approach. In Section 6, we discuss some interesting incremental properties of the problem, which are useful for incremental budget assignment. Some experiments supporting our theoretical results are reported in Section 7. Section 8 concludes the paper.

1.1 Motivating Example

Figure 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 8 time units. Hence, delay budget assignment should not create any path that takes longer than 8 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.



	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: Different timing budget management policies lead to different budget assignments. The example shows the optimal solution for maximum, bounded and min-max (with different total budgets) delay budget assignment. Our technique can accommodate all these policies optimally.

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. First column of the table shown in Figure 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.

Second column of the table (bounded maximum budgeting) illustrates the node budgets when the cost function tries 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. Lower/upper bounds are useful for many application domains, due to non-linear relation of the utility function for each node to its delays.

Another popular policy is to distribute the budget values fairly (min-

imizing the maximum budget value), while still trying to maximize the total budget. The last two columns of the table (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 has trivial solutions if there is no constraint on the total budget.

The technique that we present in this paper unifies all of the aforementioned budget assignment policies (plus a few others including min-skew budget assignment) into one generic theoretical framework. We continue to present efficient combinatorial algorithms for solving all of the motivated problems.

2. Related Works

The concept of slack in a more general context has been addressed in the synthesis community, where negative slack for a node translates to the timing constraint violation. The budgeting problem on a graph (both temporal and spatial budgeting) has been studied in theory and practice and is widely used for many different applications.

- Design timing closure- 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 re-allocated [4, 20]. Distribution of delay budget is applied to determine the wirelength under the given timing constraints.
- Timing-driven placement and floor planning- Delay budgeting during placement and floor planning has been extensively studied by several researchers [1, 3, 16]. In [16, 18], placement and net re-budgeting are combined. In [26], a novel technique for net-weighting algorithm is proposed for timing optimization in placement. Recently, in [6], a new technique for delay budgeting on sequential circuits is proposed.
- Gate/wire sizing and power optimization- Under a given timing constraint, budget management can be applied to find a set of nodes/edges in the netlist graph such that their physical size or power dissipation can be reduced by mapping to smaller, or power-efficient cell instances with larger delays from a target library [3, 13, 21].
- Exploiting slack in high-level synthesis- There are several related work in the area of high-level synthesis where timing slack of the nodes in the data flow graphs are considered for better optimization in area and power. Examples are the algorithms and techniques developed for area minimization in pipelined datapath [24] and power minimization under timing constraint [14, 27].
- Layout compaction- Space budgeting has been studied in layout compaction in the field of VLSI CAD [28, 15, 12]. The concept of budgeting was first proposed by C.K.Wong et al. [28], for spatial budgeting in layout compaction.

Almost all of the techniques employed in these works are suboptimal heuristics such as Zero Slack Algorithm [23] and MISA [2]. In our previous work [11], we solved the problem of integral delay budgeting through LP relaxation. Due to the LP numerical instability and slow runtime, combinatorial methods are often preferred.

In this paper, we present an optimal combinatorial method for solving the integral budgeting problem. 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 data-flow graphs [9, 10]. However, time budgeting problem is fairly different from their problem at hand. Moreover, to the best of our knowledge, this technique is quite novel in EDA community.

3. 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 modelled 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 5 in more details. 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 node budgeting problem which applies to many other areas such as application and architecture synthesis.

Given an application represented by a DAG $G(V,E)$, there is a non negative delay value d_{ij} and a non negative 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: 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, there is a virtual super output node (*SO*) that takes only all of the primary outputs of G as its fanins (Figure 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 paths from *SI* to *SO* take no longer than T.

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

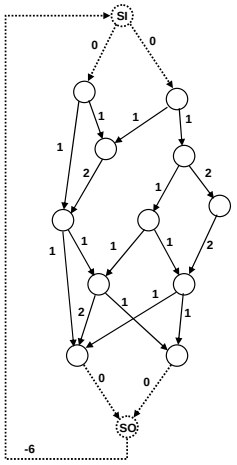


Fig. 2: A sample DAG with edge delay annotations. Nodes *SI* and *SO* and corresponding virtual edges are shown with dashed lines.

The problem can be stated as the following ILP formulation:

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

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

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

Directed acyclic graphs (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, data flow graphs

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.

3.1 Equivalent Formulations

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

Lemma 1. *In an optimal budget assignment, the delay of all paths 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 the equations 1-3 leads to the following set of equations.

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

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

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

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

Note that the number of constraints in equations 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 (Figure 2). The constraint $r_{SI} - r_{SO} \leq T$ can be represented as one of the regular edge constraints and can be safely removed as a separate constraint. The example shown in Figure 2 assumes that the timing constraint for the application is 6 time units.

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

$$\sum_{e_{ij} \in E} b_{ij} = \sum_{e_{ij} \in E} r_i - r_j - d_{ij} = \sum_{v_i \in V} r_i [out(v_i) - in(v_i)] - \sum_{e_{ij} \in E} d_{ij}$$

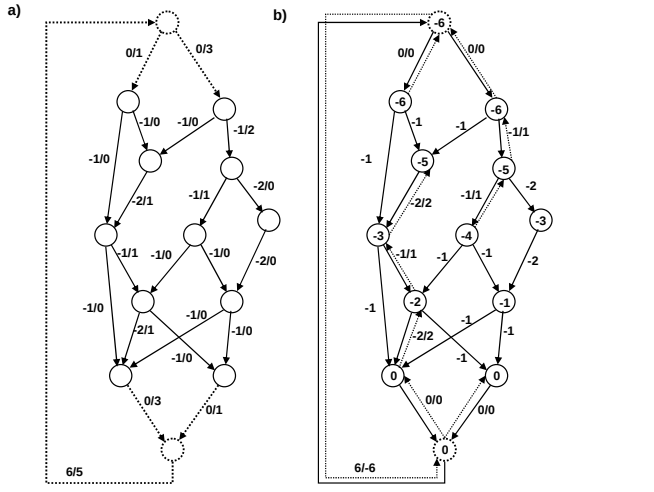


Fig. 3: a)The dual Min-cost flow problem/solution for example in Figure 2. Each edge is annotated with its cost and flow, respectively. Supply at each node i is $in(v_i) - out(v_i)$. b) The corresponding residual graph and edge costs. The number in each node shows the shortest path to SO (δ_i).

where $in(v_i)$ and $out(v_i)$ 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 $\rho_i = out(v_i) - in(v_i)$, equations 5-8 can be rephrased as:

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

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

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

the dual problem to the LP equations 9-11 is:

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

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

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

Interestingly, the equations 12-14 formulate a conventional min-cost flow problem on the DAG, where y_{ij} variables are the amount of flow along edges eij 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 [22, 9].

It follows that the original problem can be solved optimally in polynomial time. Section 4 presents an algorithm that can determine the value of b_{ij} for each edge after solving the dual min-cost flow problem.

4. Efficient Optimal Algorithm

As shown in Subsection 3.1, the dual of the original edge delay budgeting problem can be stated as a min-cost flow problem on a new graph called $G'(V,E)$. G' are G are the same 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 $-\rho_i = in(v_i) - out(v_i)$. 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 [22]. Figure 3 illustrates the dual min-cost flow problem and its solution for the graph shown in Figure 2.

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 non-zero 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 cycles in the residual graph G_y and 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 [25], that is applicable to graphs with negative edge costs. Figure 3 shows the residual graph (G_y) and δ_i variables for the example shown in Figure 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 2. *$r_i = -\delta_i$ is an optimal solution to the equations 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 [8], we only need to assign values to r_i variables such that the corresponding constraints become an equality for edges with non-zero flow. We now 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:

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 δ_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};

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

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

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

Edge delay budgets are now easily calculated by $b_{ij} = r_i - r_j - d_{ij}$. This process is depicted in Algorithm 1. Well-studied techniques including min-cost flow and shortest path calculation are only used as a black box.

For the graph shown in Figure 2, the amount of the budget assigned to each edge of the graph is illustrates in Figure 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 $b_{ij} = r_i - r_j - d_{ij}$.

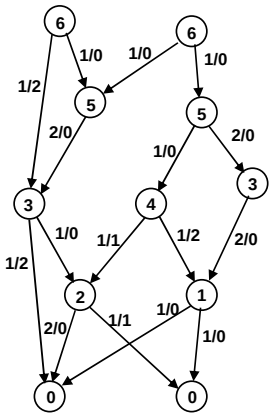


Fig. 4: Optimal budget assignment for the example in Figure 2. The numbers on each edge denote the delay and the assigned budget. Nodes are annotated with their delay to SO (r_i).

5. Extensions to Other Budgeting Policies

The timing budget assigned to each of the design components can be exploited to improve different design quality metrics such as area, power dissipation, predictability or cost. The objective function presented in Section 3 ($\sum_{e_{ij} \in E} b_{ij}$) 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 of 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 in order to optimize it for some 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 3.

5.1 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 non-negative w_{ij} variables denote the weight of edge e_{ij}. The weighted budget assignment problem can be formulated as:

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

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

$$r_{SI} - r_{SO} \leq T \tag{17}$$

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

And similar to what we did in the previous section:

$$\sum_{e_{ij} \in E} w_{ij} b_{ij} = \sum_{e_{ij} \in E} w_{ij}(r_i - r_j - d_{ij})$$

$$= \sum_{v_i \in V} r_i \Big(\sum_{v_j \in out(v_i)} w_{ij} - \sum_{k \in in(v_i)} w_{ki} \Big) - \sum_{e_{ij} \in E} w_{ij} d_{ij}$$

which turns into the equations 9 after defining $\rho_i = \sum_{v_j \in out(v_i)} w_{ij} - \sum_{k \in in(v_i)} w_{ki}$. Therefore, the algorithm described in Section 4 can handle the weighted version as well. The only required modification is the demand function (ρ_i) at each node for the dual min-cost flow problem.

An interesting special case of the weighted edge budgeting problem can solve the *node budgeting* problem, where each node has a delay value and we would like to assign budget values to nodes instead of edges. It is straight forward to see that a weighted node budgeting instance can be transformed into a weighted edge budgeting problem. The only required transformation is splitting each node into two other subnodes that are connected to each other by an edge with weight equal to the original node edge. All other regular edges should have zero weight. The technique is similar to the implementation of node capacities in a network flow instance that can handle only edge capacities. Note that some edges with zero weight might be assigned delay budgets in order to validate Lemma 1.

5.2 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. In such cases, it is desirable to have a/an lower/upper bound on the delay of each edge. Budget lower bounds are easy to implement, since they can be added to the edge delay at the first place.

Upper bounds, however, are not as easy as the lower bounds to handle. In this subsection, 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 3.

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

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

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

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

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

the dual problem to this problem is:

$$Min \sum_{e_{ij} \in E} u_{ij} z_{ij} - d_{ij} y_{ij} \quad (23)$$

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

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

Equations 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}. y_{ij} and z_{ij} are the amount of flow along edges e_{ij} and e_{ji}, respectively. Note that by definition $d_{ij} \leq u_{ij}$. 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 will be easily determined after knowing y_{ij} and z_{ij} values.

5.3 Min-Max Budget Distribution

Fair distribution of the available budget to the application components is another reasonable objective. *Fair* distribution can be quantified by minimizing the maximum budget assigned to the 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. Therefore, 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 that have at least %80 of the total budget of an optimal solution.

In any case, the problem has to be initially solved using the algorithm described in Section 3.1 to maximize the total budget. Assuming that minimizing the maximum assigned budget (min-max) is the objective, we can perform 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.

6. Potentials for Incremental Budgeting

In many practical situations, the delay of a component and hence, the problem instance might change during the design flow iterations. 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 re-execute the Algorithm 1 to find a new budget assignment for each local change. In such cases, it is often required to transform the current solution to a new feasible solution by performing local, rapid and incremental calculations.

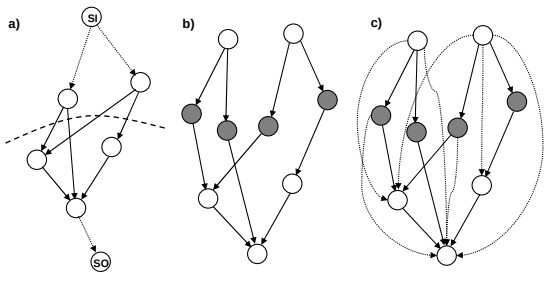


Fig. 5: a)A sample DAG and a sample cut are shown. b)The corresponding edge graph. The edges in the cut correspond to dark nodes. c) The transitive graph of the edge graph. The cut corresponds to the maximum independent set here.

In this section, we present some interesting properties of the problem formulated in Section 3. These properties can be further exploited to tackle other practical budgeting problems including incremental budget assignment. Due to the page limitation, some proofs or details of the proofs are eliminated.

Lemma 3. *Flow variables of the dual LP problem will not change with the increase of the timing constraint T.*

Proof: There is no negative cycle in G'. Therefore, all paths that do not include e_{oi} have costs not less than $-T$ (Figure 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_{oi}. Note that all such paths have negative cost.

However, edge e_{oi} is the only *backward* edge of the graph G' and is common among the rest of the flows. Hence, increasing its cost will increase the cost for all such paths and will not change the choice of the path for min-cost flow algorithms. It follows that the flow 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_{io} in the original G_y, the value of the path is decreased by the amount of the increase in T. For other nodes, δ_i variables remain intact. It follow 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.

Definitions: 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 (Figure 5). 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 there is an edge e^*_{ijk} between 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.

Definitions: Let $Gain = OPT(G,T)$ 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 4. *For a given instance of the edge budgeting problem with critical path equal to T:*

$Gain = OPT(G,T+\Delta_T) = OPT(G,T) + OPT(G_b,\Delta_T) = OPT(G,T) + \Delta_T \mid MIS(G_b^{t*}) \mid = OPT(G,T) + \Delta_T \mid MIS(G^{t*}) \mid$

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 [7] (Figure 5). 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 re-assignment and incremental budget management.

Lemma 5. *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 leads to another maximal feasible solution (one that does not violate the timing constraint and all paths take exactly T).*

7. Delay Budget Assignment During Library Mapping

Delay budget assignment is applied during different design tasks at various stages of VLSI CAD flow. We apply delay budgeting during

library mapping on a given datapath. We experimented with some applications from MediaBench [5]. We generated the dataflow graphs for these applications using SUIF compiler [19] and Machine-suif [17].

In Figure 6, the synthesis flow of mapping an application on an FPGA device is illustrated. This is a core-based implementation in which the computational operations are directly mapped to the existing optimized cores in the libraries. We use Xilinx VirtexE FPGA device as the target platform. Xilinx CoreGen tool is used to generate parameterizable cores optimized for the target architecture. For a given functionality, a number of cores can be generated to realize it with different characteristics such as latency. Figure 7 demonstrates the trade-off between the latency and the area in a 16-bit multiplier generated by CoreGen and mapped on FPGA VirtexE.

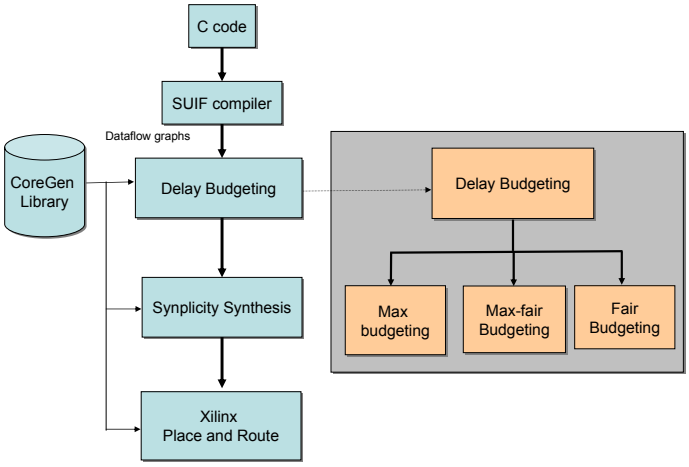


Fig. 6: Delay budgeting in core-based FPGA CAD flow.

From mediaBench applications, we selected the large dataflow graphs (DFGs) that could fit in the largest FPGA devices. The computations in the selected DFGs are multiplication, addition, subtraction, division, and shifting. We assume that all the data paths are 16-bit wide. Each computation is assigned to a resource generated from CoreGen tool based on delay budget allocated to the node.

The characteristics of the selected DFGs are outlined in Table 7. The latency column is the DFG latency plus additional slow-down for further delay budget injection. We apply different delay budgeting algorithms to allocate the delay budgets to the nodes (Figure 6). After library mapping and synthesis, the circuit is placed and routed on a FPGA device. We used the Synplify Pro 7.3 from Synplicity for synthesis and ISE 6.1 place and route tool provided by Xilinx.

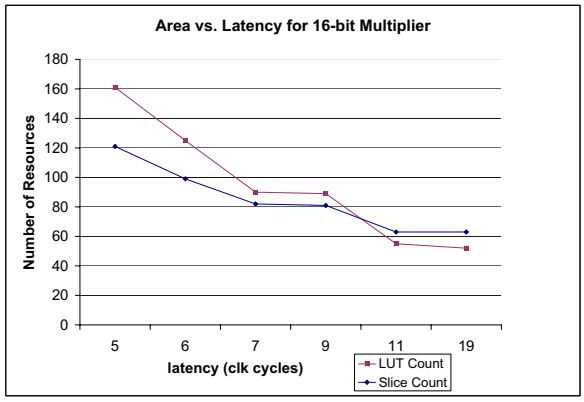


Fig. 7: Area vs. Latency for a 16-bit CoreGen Multiplier.

Figure 7 demonstrates almost linear trade-offs between the area and latency of the multiplier core in the library. Hence, maximizing the total budget can lead to further optimization in the area. In our previous work [11], we theoretically and empirically showed that optimal maximum budgeting outperforms ZSA [23] in reducing the design size. Maximum budgeting reduced the design size by more than 10% on average compared to ZSA algorithm. In this work, we apply the min-max (fair) budgeting using the algorithm described in Subsection 5.3.

The objective function for this method is to maximize the minimum delay budget assigned to the nodes. In this technique, the total budgeting is compromised with the upper bound on the delay budgets. In the first set of experiments, we allow at most 10% degradation in the maximum total budget. In the second set, we allow more fair distribution of delay budget among the nodes by allowing at most 25% of degradation in total delay budget.

The results are outlined in Table 2. We compare different budgeting algorithms from two different perspectives: the objective value and the number of nodes with non-zero delay budgets (fair budgeting). Also, the impact of different techniques on total area is demonstrated. There are four different implementations for each DFG. In the first set called *no_budgeting* no delay budget assignment is applied and in the rest, different algorithms are applied for delay budget assignment.

DFGs	Application	description	# of nodes	Latency
dfg0	DSP	AR-lattice-filter	18	20
dfg1	mesa	matrix rotation	28	17
dfg2	mesa	invert matrix	101	20
dfg3	jpeg	idct	58	23

Table 1: Characteristics of DFG benchmarks.

The topology and connectivity in the applications affect the distribution of delay budgets in the graph. If most of the paths in the graph are critical, there is not much timing slack to be exploited. We assume that the timing constraint at the output of each application is its minimum possible latency plus some excess delay budget ($\Delta T = 4$). Therefore, more timing slack is injected to the graph before applying different delay budgeting algorithms.

The results on the area of the implemented designs (both LUT and slice count) show that the combination of fair and maximum budgeting can greatly reduce the design size. For most of the benchmarks, in the case of fair budgeting within 10% degradation in total budget the best results are obtained. Hence, the total injected budget, as well as the distribution of delay budgets both have a significant impact on the design quality. In *DFG0* and *DFG1* the total budget does not change but fair distribution outperforms the other max budgeting with no fairness. For *DFG2*, the fair budgeting with more degradation in total budget leads to better results. One reason is that the delay constraints for a larger number of nodes are relaxed (better distribution).

The last two columns compare the quality of different delay budgeting techniques. The column *Variation in quality metric* illustrates the percentage of variation in the quality metric (area) among different budgeting policies. The last column compares the best value of quality metric vs. the value where no budgeting technique is applied. The values in the last two columns are computed as the percentage of the difference between the best and the worst value in the set over the best: $\frac{worst-best}{best} \times 100$. On average, the area of designs after delay budgeting varies 18.08% in terms of LUT count (and 9.72% in terms of slice count) from the best depending on the algorithms used for budgeting. Applying delay budgeting improves the area by 53.66% (in terms of LUT count) and 35.83% (in terms of slice count) on average, compared to no budgeting.

8. Conclusion

We presented a theoretical framework that unifies a large class of existing time-management paradigms. Our model can optimally handle many time budgeting policies including maximizing total budget, weighted, bounded and fair budgeting. In addition, our technique is applicable to time management for edges, nodes and hybrid combination of these two elements.

We have performed experimental results on mapping some applications onto Xilinx FPGAs. We have generated the applications' datapath components using CoreGen IP cores. We have 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.

Benchmark	Design Metric	Delay Budgeting Algorithms				Variation among Budgeting policies(%)	Delay Budgeting vs. no Budgeting (%)
		No-budget	Max	Max-fair (10%)	Max-fair (25%)		
DFG0	LUT count	1790	1342	1158	1478	27.63	35.30
	Slice count	1232	1008	920	1104	8.73	33.91
	Total budget	-	66	64	53	20.7	-
	Relaxed nodes	-	8	8	8	0	-
DFG1	LUT count	3856	2365	2168	2608	20.20	43.77
	Slice count	2672	1930	1837	2090	13.77	31.72
	Total budget	-	66	64	53	24.52	-
	Relaxed nodes	-	13	18	18	38.46	-
DFG2	LUT count	8658	5149	5102	4837	6.40	78.99
	Slice count	6249	4525	4496	4355	3.90	43.49
	Total budget	-	138	122	109	26.60	-
	Relaxed nodes	-	28	38	40	42.85	-
DFG3	LUT count	3168	2022	2075	2388	18.10	56.60
	Slice count	2224	1657	1686	1864	12.49	34.21
	Total budget	-	42	39	32	31.25	-
	Relaxed nodes	-	11	12	12	9.09	-
Average	LUT count	4368.00	2719.5	2625.75	2827.75	18.08	53.66
	Slice count	3094.25	2280.00	2234.75	2353.25	9.72	35.83
	Total budget	-	78	72.25	61.75	25.76	-
	Relaxed nodes	-	15	19	19.5	22.6	-

Table 2: The quality of different delay budget assignment techniques.

9. References

[1] A. Kahng, S.Mantik, and I.L. Markov. "Min-Max Placement for Large-Scale Timing Optimization". In *ACM International Symposium on Physical Design*, pages 143–148, 2002.

[2] C. Chen, E. Bozorgzadeh, A. Srivastava and M. Sarrafzadeh. "Budget Management with Applications". *Algorithmica*, 34(3):261–275, July 2002.

[3] C. Chen, X. Yang, M. Sarrafzadeh. "Potential Slack: An Effective Metric of Combinational Circuit Performance". In *ACM/IEEE International Conference on Computer-Aided Design*, pages 198–201, 2000.

[4] C. Kuo and A. C.H. Wu. "Delay Budgeting for a Timing-Closure-Design Method". In *ACM/IEEE International Conference on Computer-Aided Design*, pages 202–207, 2000.

[5] C. Lee, M. Potkonjak, and W.H. Mangione-Smith. "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems". In *International Symposium on Microarchitecture*, 1997.

[6] C. Yeh and M. Marek-Sadowska. "Delay Budgeting in Sequential Circuit with Application on FPGA Placement". In *ACM/IEEE Design Automation Conference*, 2003.

[7] D. Kagaris and S. Tragoudas. "Maximum Independent Sets on Transitive Graphs and Their Applications in Testing and CAD". In *IEEE/ACM International Conference on Computer-Aided Design*, pages 736–740, 1997.

[8] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[9] E. Boros, P. Hammer and R. Shamir. "A Polynomial Algorithm for Balancing Acyclic Data Flow Graphs". *IEEE Trans. Computers*, 41(11):1380–1385, 1992.

[10] E. Boros, P. Hammer, M. Hartmann and R. Shamir. "Balancing Problems in Acyclic Networks". *Discrete Applied Math.*, 49(1-3):77–93, 1994.

[11] E. Bozorgzadeh, S. Ghiasi, A. Takahashi and M. Sarrafzadeh. "Optimal Integer Delay Budgeting on Directed Acyclic Graphs". In *Design Automation Conference*, June 2003.

[12] 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 *Conference on European Design Automation*, November 1992.

[13] H.R. Lin and T. Hwang. "Power Reduction by Gate Sizing with Path-Oriented Slack Calculation". In *IEEE Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 7–12, 1995.

[14] J. Luo and N. Jha. "Battery-Aware Static Scheduling for Distributed Real-Time Embedded Systems". In *IEEE/ACM Design Automation Conference*, 2001.

[15] J.F. Lee and D.T. Tang. "VLSI Layout Compaction with Grid and Mixed Constraints". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5), September 1987.

[16] M. Sarrafzadeh, D. Knol and G.E. Tellez. "Unification of Budgeting and Placement". In *ACM/IEEE Design Automation Conference*, June 1997.

[17] M.D. Smith, and G. Holloway. "An introduction to machine SUIF and its portable libraries for analysis and optimization". Technical report, Division of Engineering and Applied Sciences, Harvard University, 2002.

[18] M.Sarrafzadeh, D.A. Knol and G.E. Tellez. "A Delay Budgeting Algorithm Ensuring Maximum Flexibility in Placement". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1332–1341, 1997.

[19] 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*, 29(12):84–89, 1996.

[20] O. Coudert. "Timing and Design Closure in Physical Design Flows". In *IEEE International Symposium on Quality Electronic Design*, 2002.

[21] P. Girard, C. Landrault, S. Pravossoudovitch and D. Severac. "A Gate Resizing Technique for High Reduction in Power Consumption". In *International Symposium on Low Power Electronics and Design*, pages 281–286, 1997.

[22] T. Magnanti R. Ahuja and J. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993.

[23] R. Nair, C. Berman, P. Hauge and E. Yoffa. "Generation of Performance Constraints for Layout". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8:860–874, August 1989.

[24] S. Bakshi and D. Gajski. "Component Selection for High-Performance Pipelines". *IEEE Transactions on Very Large Scale Integrated Systems*, 4(2):181–194, 1996.

[25] R. Rivest T. Cormen, C. Leiserson. *An introduction to algorithms*. MIT Press, 1990.

[26] T. Kong. "A Novel Net Weighting Algorithm for Timing-Driven Placement". In *ACM/IEEE International Conference on Computer-Aided Design*, 2002.

[27] 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 *ACM/IEEE International Symposium on Microarchitecture*, 2001.

[28] Y. Liao and C. K. Wong. "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints". *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(2), April 1983.