

# Optimal Integer Delay Budgeting on Directed Acyclic Graphs

E. Bozorgzadeh<sup>†</sup>S. Ghiasi<sup>†</sup>A. Takahashi<sup>‡</sup>M. Sarrafzadeh<sup>†</sup>

<sup>†</sup> Computer Science Department  
University of California, Los Angeles (UCLA)  
Los Angeles, CA 90095, USA  
e-mail: elib, soheil, majid@cs.ucla.edu

<sup>‡</sup> Department of Communications and Integrated Systems  
Tokyo Institute of Technology  
Tokyo 152-8552, Japan  
email:atsushi@lab.ss.titech.ac.jp

## ABSTRACT

Delay budget is an excess delay each component of a design can tolerate under a given timing constraint. Delay budgeting has been widely exploited to improve the design quality. We present an optimal integer delay budgeting algorithm. Due to numerical instability and discreteness of libraries of components during library mapping in design optimization flow, integer solution for delay budgeting is essential. We prove that integer budgeting problem - a 20-year old open problem in design optimization [7]- can be solved optimally in polynomial time. We applied optimal delay budgeting in mapping applications on FPGA platform using pre-optimized cores of FPGA libraries. For each application we go through synthesis and place and route stages in order to obtain accurate results. Our optimal algorithm outperforms ZSA algorithm [3] in terms of area by 10% on average for all applications. In some applications, optimal delay budgeting can speedup runtime of place\_and\_route up to 2 times.

## Categories and Subject Descriptors

B.5 [Hardware]: register-transfer-level implementation;  
B.6 [Hardware]: logic design; B.m [Hardware]: miscellaneous  
(design management); G.1 [Numerical Analysis]: optimization

## General Terms

Algorithms, Design.

## 1. INTRODUCTION

Due to design complexity, optimization techniques need to be applied in multiple stages starting from high level of abstraction down to gate level and physical design. In order to abstract the complexity, each design is decomposed into a set of sub-designs. The essential constraint during the design optimization flow is the timing constraint. The sub-designs along the critical paths are the most constrained components during the optimization process in CAD flow. However, timing constraint is loose on the other sub-designs. Hence the allowable delay allocated on each sub-design can be greater than

actual/intrinsic delay of the sub-design. This excess delay is referred to as *delay budget* (or *timing budget*). Delay budgeting has been exploited through the whole CAD flow to improve the design quality.

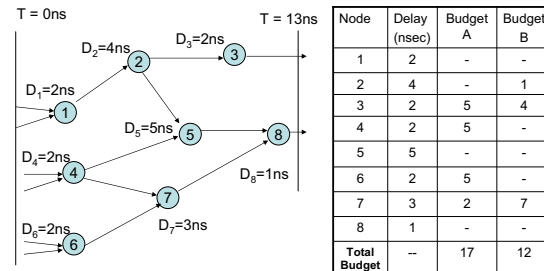


Figure 1: Delay Budgeting Problem in a DAG.

Each design is represented by a directed acyclic graph (DAG  $G = (V, E)$ ). There is a delay associated with each node. Under a given timing constraint, delay budget at each node is the extra delay the component can tolerate such that no timing constraint is violated. Similar definition can be applied to budget of an edge. Budget of each node/edge is related to timing slack of the node/edge. If there is any node or an edge with negative slack, timing constraint is violated. However, due to dependency between the nodes, the total timing slack of the node/edges is not the total budgets nodes/edges can tolerate. In Figure 1, two different delay budgetings ( $A$  and  $B$ ) are applied on a DAG. Budget column of the table corresponds to excess delay that can be allocated to each node under timing constraint ( $13ns$ ). After applying any of budgeting  $A$  or  $B$  on the graph, no other node can tolerate any excess delay. Total delay budget after budgeting  $A$  is 17 while the total delay budget after budgeting  $B$  is only 12.

Delay budgeting has many applications in design optimization as follows:

- Timing-driven placement and floorplanning [6, 4, 5]- In timing-driven placement, the goal is to optimize the path delays with lesser number of iterations. Per-net delay bounds in delay budgeting is applied. In [4], placement and re-budgeting are combined.
- Gate/wire sizing and power optimization [10]- Under timing constraint, gate sizing problem is to find a set of nodes/edges

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC 2003, June 2-6, 2003, Anaheim, California, USA.  
Copyright 2003 ACM 1-58113-688-9/03/0006 ...\$5.00.

in the graph such that their physical size can be reduced by mapping to smaller cell instances with larger delay from a target library. In general, delay budgeting can be applied during *library mapping* stage.

- VLSI layout compaction [7, 8, 9]- The main objective is to minimize the physical area of the layout. Concept of budget is exploited to reduce wirelength. With multiple symmetry constraints, layout compaction is solved using LP solver. LP formulation of compaction is similar to formulation of delay budgeting problem.

The most popular and efficient algorithm for delay budgeting is zero-slack algorithm [3]. The solution is not optimal and can be far away from optimal result. MISA algorithm proposed in [2] finds the total budget in the graph using maximum independent set. MISA algorithm finds a potential slack which correlates strongly with the total budget in the graph. However, both ZSA and MISA algorithms cannot solve the budgeting problem optimally.

In this paper, we focus on theoretical study of integer delay budgeting problem on the nodes in a directed acyclic graph. Objective function in our delay budgeting problem is to maximize the total delay budget of the nodes under a given timing constraint. The general problem can be formulated as a linear programming problem. However, the solution can have fractional value. According to the following reasons optimal *integer* solution is preferred: First, the budget at each node is mostly used to map the sub-design to another component in a target library which inherently is discrete rather than continuous. For example, delay on interconnect is discrete in a grid-based routing methodology. In a datapath, delay of each component can be given in terms of number of clock cycles under a given frequency. Delay of gates can be scaled to integer values. In VLSI compaction, grid constraints require integer solution [8]. Secondly, due to numerical instability in representation of real numbers, linear programming solvers suffer from instability and difficulty in convergence. Therefore we assume the variables associated with budgets are all integer. ZSA and MISA algorithms can be modified to generate integer budgets, but with no guarantee on the optimality of the solutions.

The complexity of integer delay budgeting problem on DAGs has been an open problem for over a decade [7]. Applying rounding techniques to LP optimal solution of budgeting problem cannot preserve the optimality of the integer solution. In this paper, we propose our novel efficient graph-based transformation technique to produce optimal integer solution from optimal LP solution. We prove that integer budgeting problem can be solved optimally by transformation from LP relaxation solution to an integer solution in polynomial time.

We apply delay budgeting technique in mapping a given datapath on a FPGA platform. For faster compilation and exploiting the architectural features of FPGAs, FPGA vendors provide a relatively rich IP library. Using IP library of FPGAs, we show that the delay budgeting plays a trade-off between latency of a datapath and area of resources used by the application. We compare our proposed optimal delay budgeting algorithm with ZSA. The decrease in complexity of datapath improves the runtime of place and route stage, which is the most time-consuming stage in mapping an application on FPGA platforms. Our experimental results show the effectiveness of budgeting on IP-based application mapping.

The rest of the paper is organized as follows: In Section 2, the problem is formally defined. In Section 3, the budget re-assignment is proposed. Applying budget re-assignment on LP solution of budgeting problem is described in Section 4 and it is proven that the final solution is integer and optimal. In Section 5, the experimental results on trade-off between latency and area by budgeting technique in FPGA platform are presented. In Section 6, conclusions and some possible future directions are outlined.

## 2. LP FORMULATION OF DELAY BUDGETING PROBLEM IN A DAG

In a given directed acyclic graph  $G = (V, E)$ , associated with each node  $v_i$ , there is a delay variable  $d_i > 0$  and budget variable  $b_i$ .

edge  $e_{ij}$  is incident to node  $v_j$  and incident from node  $v_i$ . Edge  $e_{ij}$  is called an *outgoing* edge with respect to node  $v_i$  and an *incoming* edge with respect to node  $v_j$ .  $V_I(i)$  is the set of incoming edges to node  $v_i$ .  $V_O(i)$  is the set of outgoing edges from node  $v_i$ . Primary inputs (PIs) are the nodes with no incoming edges. Primary outputs (POs) are the node with no outgoing edges.

**arrival time of  $v_i$ :** If input to primary input of graph is ready at time 0, the output of node  $v_i$  is ready at  $a_i$  which can be calculated as  $a_i = \max_{v_j \in V_I(i)} a_j + (d_i + b_i)$ ,  $a_i = 0$  for  $v_i \in PI$ .

Arrival time at a primary output is maximum summation of budget and delay associated with each node along the path from primary input up to primary output. Arrival time at each primary output cannot exceed a fixed value,  $T$ . This is referred as *required time* at primary outputs. Although required time at primary outputs and arrival time at primary inputs can be different, for simplicity, we assume that arrival time at each primary input is zero and required time at primary outputs is  $T$ .

**Delay budgeting formulation:** On a directed acyclic graph  $G = (V, E)$  with delay  $d_i$  associated with each node  $v_i$  and required time  $T$ :

$$\text{Max} \quad \sum_{v_i \in V} b_i \quad (1)$$

$$a_j \geq a_i + b_j + d_j \quad \forall e_{ij} \in E \quad (2)$$

$$a_i \leq T \quad \forall v_i \in PO \quad (3)$$

$$a_i = 0 \quad \forall v_i \in PI \quad (4)$$

$$d_i, b_i, T \in \mathbb{Z}_+ \quad \forall v_i \in V. \quad (5)$$

General LP formulation of budgeting problem is

$$\text{max} \{ \sum_{i=1}^{|V|} x_i \mid Ax \leq b \}.$$

In area of linear programming theory, there has been a deep study on the linear programs that automatically have optimal integer solutions. In particular, it is the case for network flow problems. If matrix  $A$  is *totally unimodular*, the linear programming relaxation can solve the ILP, proposed by Heller and Tompkins [1]. We observe that the linear programming relaxation of integral delay budgeting for a given directed path holds the sufficient condition to give optimal integer solution, that is constraint matrix  $A$  is totally unimodular.

**THEOREM 1.** *The linear programming relaxation of integer budgeting problem gives optimal integer solution if the input graph is a directed path.*

The aforementioned sufficient condition does not necessarily hold for general directed acyclic graph rather than a directed path. In the following sections, we prove that the integral budgeting problem can be solved optimally in polynomial time, using the solution of the linear programming relaxation problem.

## 3. BUDGETING ASSIGNMENT IN A DAG

In this section, we first define the maximal budgeting on a given directed graph  $G = (V, E)$  with required time  $T$  at primary outputs. Arrival time of any node cannot exceed  $T$ . Otherwise the dependency constraints in Equation 5 are not satisfied. Due to space constraints, lemmas and theorems are stated with no proof. Some basic definitions used in this section are as follow:

**Definitions:** required time at  $v_i$ ,  $r_i$ , is computed as  $\min_{v_j \in V_O(i)} (r_j - (d_j + b_j))$ .  $r_i = T$  for  $v_i \in PO$ .  $T$  is required time at primary outputs in graph  $G$ . slack at node  $v_i$  is  $s_i = r_i - a_i$ . *a-slack* of edge  $e_{ij}$ ,  $\epsilon_{a_{ij}}$ , is:  $(a_j - (d_j + b_j)) - a_i$ ,  $e_{ij} \in E$ . Similarly, *r-slack* of  $e_{ij}$ ,  $\epsilon_{r_{ij}}$  is:  $(r_j - (d_j + b_j)) - r_i$ ,  $e_{ij} \in E$ . Edge  $e_{ij}$  is said to be *critical* if the *a-slack* value and *r-slack* value associated with edge  $e_{ij}$  are zero. A path in a graph which includes only critical edges is called *critical path*.

The following lemma can be easily derived from the abovementioned definitions:

LEMMA 1. In a directed graph  $G$ , if  $e_{ij} \in E$  and  $s_i = s_j$ , then  $\epsilon_{a_{ij}} = \epsilon_{r_{ij}} = \epsilon_{ij}$ .

**Maximal Budgeting Graph  $(G, B_m)$ :**  $B_m$  is a feasible solution to budgeting problem on a directed acyclic graph  $G$ . Feasible solution  $B_m$  with associated objective value,  $|B_m|$ , is called maximal budgeting if no more budget can be given to any node while the budget of any other node does not decrease.

The maximum solution  $B^*$  is also a maximal solution. Maximal budgeting solution  $B_m$  can be obtained by applying different algorithms such as MISA algorithm [2] and ZSA algorithm [3].

LEMMA 2. In  $(G, B_m)$ , if the slack of each node is zero,  $B_m$  is a maximal budgeting.

Non-critical edges are referred to as  $\epsilon$ -edges. According to Lemma 1 the a-slack and r-slack of a  $\epsilon$ -edge in  $(G, B_m)$  are equal, that is  $\epsilon_{ij} = \epsilon_{a_{ij}} = \epsilon_{r_{ij}}, \forall e_{ij} \in (G, B_m)$ .

LEMMA 3. In a maximal budgeting  $(G, B_m)$ , each node (except PIs and POs) has at least one critical incoming edge and at least one critical outgoing edge.

Associated with solution  $B_m$ , critical graph  $G_T \subseteq G = (V, E)$  is the graph obtained from the graph  $G$  by deleting all non-critical edges in  $G$ .  $G_T = (V, E_T)$ ,  $E_T = E - \{e_{ij} | \epsilon_{ij} \neq 0\}$ .

In any budgeting on graph  $G$ , slack of each node and edge must be non-negative or in other words  $a_i \leq T$ . This is referred to as feasibility in graph. A graph with budgeting  $B$  is not feasible if slack of a node or an edge is negative.

We propose a budget re-assignment method on a given maximal budgeting.

**Feasible Budget Re-assignment on  $(G, B_m)$ :** In graph  $G$  with maximal budgeting solution  $B_m$ , the budgets of the nodes are changed such that the new budgeting  $B'_m$  is still a maximal budgeting  $(G, B'_m)$ . Budget re-assignment on graph  $G$  transforms the budgeting from solution  $B_m$  to  $B'_m$ . Feasible  $\beta$ -budget re-assignment on graph  $(G, B_m)$  is a feasible budget re-assignment in which the change of budget in each node is either  $\pm\beta$  or 0.

Theorem 2 presents two sufficient conditions for feasible  $\beta$ -budget re-assignment.

THEOREM 2. The re-assignment of budget of  $\{0, \pm\beta\}$  at each node in graph  $(G, B_m)$  is a feasible  $\beta$ -budget re-assignment if

- the total amount of change in the budget of the nodes along each critical path from PI to PO is zero (Figure 2), and
- for each  $\epsilon$ -edge  $e_{il}$ ,  $\epsilon_{il} \geq (k_i - k_j) \cdot \beta$ , where edge  $e_{jl}$  is critical.  $k_i\beta$  and  $k_j\beta$  are the amount of change in total budget along any critical path from PI to node  $v_i$  and  $v_j$ , respectively (Figure 2(b)).

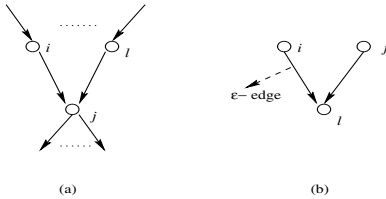


Figure 2: Two Sufficient Conditions for  $\beta$ -budget Re-assignment.

We show that the budget exchange between two sub-graphs under child-parent relation satisfies the conditions, hence it is a feasible  $\beta$ -budget re-assignment in graph  $(G, B_m)$ .

**Parent/Child Relation:** In a directed graph  $G$ , edge  $e_{ij} \in E$  and  $e_{ij}$  is critical. Node  $v_j$  is child of node  $v_i$ .  $c(v_i)$  is used to refer to as a child of node  $v_i$ . Node  $v_i$  is said to be the parent of node  $v_j$ .  $p(v_j)$  is used to refer to as a parent of node  $v_j$ . If  $v_i$  and  $v_j$  have common child,  $v_i \sim_p v_j$ . If  $v_1 \sim_p v_2 \dots \sim_p v_n$ , then  $v_1 \sim_p^* v_n$ .  $\sim_p^*$  is an equivalent relation, called parent relation. If  $v_i$  and  $v_j$  have common parent,  $v_i \sim_c v_j$ . If  $v_1 \sim_c v_2 \dots \sim_c v_n$ , then  $v_1 \sim_c^* v_n$ . Similar to parent relation,  $\sim_c^*$ , called child relation, is an equivalent relation.

LEMMA 4.  $v_i \sim_c^* v_j$ , iff  $p(v_i) \sim_p^* p(v_j)$ .

LEMMA 5. In  $(G, B_m)$ , if  $v_i \sim_p^* v_j$ , arrival time at nodes  $v_i$  and  $v_j$  are equal;  $a_i = a_j$ .

According to Lemma 3, each node is incident to/from a critical edge. Consider node  $v_i$  in graph  $G = (V, E)$ . Let  $S_p(v_i) = \{v_j | v_i \sim_p^* v_j\}$  be a parent set. Let  $v_l$  be a child node of  $v_i$ .  $S_c(v_l) = \{v_j | v_j \sim_c^* v_l\}$ . According to Lemma 4, sets  $S_p(v_j)$  and  $S_c(v_l)$  are a pair of sets such that all the child nodes of the nodes in  $S_p$  are in  $S_c$ . Similarly, all the parent nodes of the nodes in set  $S_c$  are in  $S_p$ . The sets  $S_p(v_i)$  and  $S_c(v_l)$  are called parent-child set  $(S_p, S_c)$  associated with node  $v_i$ . Parent-child set  $(S_p, S_c)$  is shown in Figure 3. The followings are the propositions regarding the parent-child set in  $(G, B_m)$ .

LEMMA 6. If nodes  $v_i \sim_p^* v_j$ , there is no directed critical path between  $v_i$  and  $v_j$  if  $\forall v_i \in V, d_i > 0$ . Similarly, if nodes  $v_i \sim_c^* v_j$ , there is no directed critical path between  $v_i$  and  $v_j$  if  $\forall v_i \in V, d_i > 0$ .

LEMMA 7. In a parent-child set  $(S_p, S_c)$ ,  $S_p$  and  $S_c$  do not intersect if  $\forall v_i \in V, d_i > 0$ .

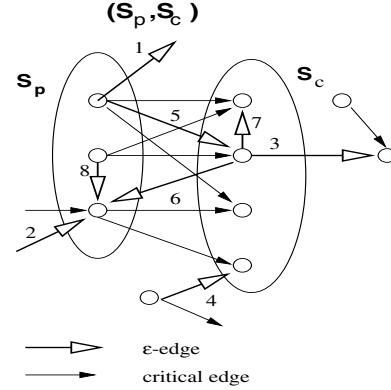


Figure 3:  $\epsilon$ -edges with respect to Parent-Child Set  $(S_p, S_c)$ .

Let  $\beta$ -budget exchange in parent-child set  $(S_p, S_c)$  be decreasing the budget of the nodes in  $S_p$  by  $\beta$  and increasing budget of nodes in  $S_c$  by  $\beta$ ,  $\beta > 0$ .

LEMMA 8. In a given  $(S_p, S_c)$  in  $(G, B_m)$ , if  $\beta \leq \min(\epsilon_{ij})$ , where  $e_{ij}$  is an  $\epsilon$ -edge with  $v_j \in S_c$  and  $v_i \notin (S_p, S_p)$  (incoming  $\epsilon$ -edges to  $S_c$ ), the  $\beta$ -budget exchange is a feasible  $\beta$ -budget re-assignment in  $(G, B_m)$ .

## 4. INTEGER SOLUTION OF LP BUDGETING PROBLEM

$(G, B^*)$  is the optimal solution to linear programming relaxation of integer budgeting problem.  $B^*$  is also a maximal budgeting. Hence, budget re-assignment is applicable to  $(G, B^*)$ . In addition, since  $B^*$  is the optimal solution,  $B_m \leq B^*$  for any maximal budgeting  $B_m$ . We define  $\beta$  in  $\beta$ -budget re-assignment on graph  $(G, B^*)$  such that the budget of all the nodes become integer. We show that during this transformation from optimal solution to integer solution  $(B^*)$ , the objective value of new solution is equal to  $|B^*|$ .

**Integral sequence:** A sequence of nodes  $IS_n = \langle v_1, v_2, \dots, v_n \rangle$  along a critical path in  $(G, B^*)$  is called Integral Sequence if  $a_1, a_n \in \mathbb{Z}_+$  and  $a_2, \dots, a_{n-1} \notin \mathbb{Z}$ .

Since required time  $T$  and delay of each node is an integer:

LEMMA 9. *The total budget of the nodes along any integral sequence in  $(G, B_m)$  is integer.*

COROLLARY 1. *The total budgeting on any critical path from PI (Primary Input) to PO (Primary Output) is integral.*

Based on Lemma 9, each node with fractional budget belongs to an integral sequence. Hence, within an integral sequence, it is sufficient to re-assign the fractional budget only on the nodes in an integral sequence. On the other hand, in graph  $G$ , there are several integral sequences connected to each other. Therefore in re-assigning the budget between the nodes, the required conditions in Theorem 2 have to be satisfied in all those sequences. Hence, the goal is to apply budget re-assignment of the fractional budgets on the nodes in graph in  $(G, B^*)$  to obtain integer solution. Since the budget re-assignment needs to be applied between the nodes with fractional budget, we reduce the graph  $(G, B^*)$  to graph  $G_f$ , the fractional adjacency graph defined as follows:

**Fractional Adjacency Graph :** Graph  $G_f$  is the fractional adjacency graph corresponding to given graph  $(G, B^*)$ . The nodes in graph  $G_f$  are a subset of nodes in graph  $G$  that have non-integer (fractional) budget. A *critical edge* between two nodes in graph  $G_f$  represents the existence of a directed critical path between two nodes in graph  $G$  such that there is no fractional budget along the path and arrival time of each node along the path is not integer. There is a  $\epsilon$ -edge between two nodes  $v_i$  and  $v_j$ , if there is no critical path between the two nodes but at least a path with  $\epsilon$ -edges along the path. Among all different paths between the two nodes, the minimum of total  $\epsilon$  value of the  $\epsilon$ -edges along each path is the  $\epsilon$  value of the  $\epsilon$ -edge in graph  $G_f$ .

Two adjacent nodes  $v_i$  and  $v_j$  in graph  $G_f$  represents the two immediate nodes on a directed critical path in graph  $G$  with fractional budget, both belonging to same integral sequence.

$\beta$ -budget re-assignment is applied on graph  $G_f$  such that the budget of all the nodes become integer. Only fractional value of budgets need to be re-assigned in order to obtain integer solution. Hence  $\beta$  is a fractional value less than unit. As described in previous section, feasible budget-reassignment can be applied on a parent-child set on graph  $G$ . Similar argument can be applied to graph  $G_f$  as follows:

LEMMA 10. *In graph  $G_f$ , if node  $v_i \sim_p^* v_j$ , the fractional values of arrival time at both nodes are equal, i.e.,  $a_i - [a_i] = a_j - [a_j]$ .*

LEMMA 11. *If nodes  $v_i \sim_p^* v_j$  in graph  $G_f$  and there is a directed critical path between nodes  $v_i$  and  $v_j$  in graph  $G$ , there has to exist at least one node on the path between the nodes  $v_i$  and  $v_j$  in graph  $G$ .*

The set  $S_p(v_i) = \{j | v_j \sim_p v_i\}$  is the set of nodes in graph  $G_f$  such that each node shares at least a common child with another node in  $S_c(v_i)$ . The set  $S_c(v_i) = \{j | v_j \sim_c v_i\}$  is the set of nodes in which each node in the set shares a parent at least with one another node in the set. In Figure 4, a parent-child set in  $G_f$  is shown.

According to Lemma 11, Lemma 12 is derived.

LEMMA 12.  *$S_p(v_i)$  and  $S_c(v_j)$  do not intersect ( $e_{ij} \in E(G_f)$ ).*

On a given parent-child set in graph  $G_f$ , we apply  $\beta$ -budget exchange. If fractional budget in graph  $G_f$  are re-assigned by budget re-assignment on parent-child set, the fractional budget is removed from each parent node and re-assigned to one of its successor in the graph. Hence, the fractional budgets are re-assigned from PIs to POs, in one direction within an integral sequences. There are  $\epsilon$ -edges in a given graph  $G_f$ . In order to have a feasible budget re-assignment on parent-child set, we show that the sufficient conditions outlined in Theorem 2 are satisfied in a given graph  $G_f$  as well.

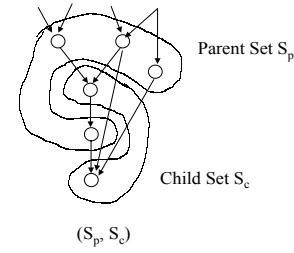


Figure 4: Parent-Child Set  $(S_p, S_c)$  in graph  $G_f$  of graph  $G$ .

LEMMA 13.  *$\beta$ -budget exchange on a parent-child set in graph  $G_f$  is a feasible  $\beta$ -budget re-assignment if  $\beta \leq \min(\epsilon_{ij}, \alpha)$ , where  $\epsilon_{ij}$  is  $\epsilon$ -edge.  $\epsilon_{ij}$  is an incoming edge to child set.  $\alpha_i$  is the fractional value at parent nodes.*

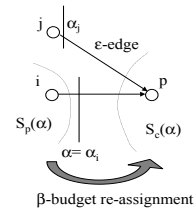


Figure 5:  $\epsilon$ -edge incident to a child node in  $(S_p, S_c)_\alpha$ .

If  $\beta$  is less than the fractional value of budget in parent nodes, after budget re-assignment, arrival time at parent node is reduced by  $\beta$ . Hence, if  $\beta$  is equal to fractional value of the arrival time, arrival time at all parent nodes become integer. On the other hand,  $\beta$  need to be at most as large as the minimum available budget in parent nodes.

In Figure 5, an  $\epsilon$ -edge incident to a child node is shown. Let  $\alpha_i$  and  $\alpha_j$  be the fractional value of arrival time at nodes  $v_i$  and  $v_j$ , respectively. In  $\beta$ -budget re-assignment, if  $\beta = \alpha_i$ , for  $\epsilon \geq 1$ ,  $\epsilon > \beta$  is True. Assume  $\epsilon < 1$ . The value of  $\epsilon$  is computed as follows:

$$\epsilon_{jp} = \begin{cases} \alpha_i - \alpha_j & \text{if } \alpha_i > \alpha_j \\ 1 + \alpha_i - \alpha_j & \text{if } \alpha_i < \alpha_j \end{cases} \quad (6)$$

When  $\alpha_i < \alpha_j$ ,  $\epsilon > \alpha_i$ . Since  $\beta = \alpha_i$ ,  $\epsilon > \beta$ . Hence the inequality of Theorem 2 is held. Hence  $\beta$  value in  $\beta$ -budget re-assignment can be computed independent of  $\epsilon$ -edges incident to child set as follows:

LEMMA 14. *Let  $(S_p, S_c)$  be a parent-child set with  $\alpha_p$ , the fractional value at the arrival time at the parent nodes. Assume that  $\alpha_p$  is the smallest fractional value of arrival time at all the nodes in graph  $G_f$ .  $\beta$ -budget exchange of  $\beta = \alpha_p$  from parent nodes to child nodes is a feasible budget re-assignment.*

After budget re-assignment on parent-child set  $(S_p, S_c)$ , arrival time at each parent node becomes integer with  $\beta = \alpha_p$ . If budget of any node in parent set or child set becomes integer, the node is removed from  $G_f$ . In this budget re-assignment, an integer budget of any node in graph  $G$  never becomes fractional. Hence no node is added to graph  $G_f$  after budget re-assignment. Since arrival time at a parent node becomes integer, all the edges connecting the parent nodes to the child nodes are removed from graph  $G_f$ . Similarly no edge is added to graph  $G_f$  after budget re-assignment.

Assume that generating the parent-child sets and applying budget re-assignment on the parent-child sets in graph  $G_f$  continues. An important fact is that after budget re-assignment, the parent nodes do not have any outgoing edges in graph  $G_f$ . Hence, the corresponding nodes cannot become parent nodes anymore. Therefore we have the Lemma as stated below:

LEMMA 15. Each node in graph  $G_f$  can only be once in a parent set during sequential parent-child budget re-assignment.

Note that after each  $\beta$ -budget exchange, the outgoing edges of parent nodes are removed. No more outgoing edges are added to parent nodes in  $G_f$  since arrival time at parent nodes are integer. On the other hand, integer budget of a node never becomes fractional after any  $\beta$ -budget exchange. Since each node can only once appear in a parent set, the number of parent-child which can be generated followed by budget re-assignment on each set is  $O(|V|)$ , where  $V$  is set of nodes in graph  $G$ .

THEOREM 3. Sequentially generating parent-child set followed by  $\beta$ -budget re-assignment in the order of increasing fractional value of arrival time at parent nodes of the parent-sets with  $\beta = \alpha_p$ ,  $G_f = \emptyset$  in  $O(|V|)$ .

If graph  $G_f = \emptyset$ , the budget of all the nodes in graph  $G$  are integer. Hence, Theorem 3 shows that a maximal integer solution can be obtained from LP solution using  $\beta$ -budget exchange on graph  $G_f$ . The following lemma proves that during budget re-assignment optimality is preserved.

LEMMA 16. In graph  $G_f$  corresponding to  $(G, B^*)$ ,  $|S_p(v_i)| = |S_c(v_j)|$  if  $e_{ij} \in G_f$ .

THEOREM 4. In any feasible  $\beta$ -budget re-assignment on parent-child set  $(S_p, S_c)$  in graph  $(G, B^*)$ , the total budget does not change.

Hence after applying the budget re-assignment on  $(G, B^*)$ , the solution is still optimum.

Each parent-child set construction takes  $O(|E|)$  and budget re-assignment takes  $O(|E|)$ . Updating graph  $G_f$  takes  $O(|E|)$ . This repeats  $O(|V|)$  times. However, by amortized analysis we see that the complexity of  $O(|E|)$  during the process applies to a set of edges during the current iteration and then those edges are removed from graph  $G_f$  before the next budget re-assignment. Hence the total complexity is  $O(|E|) = O(|V|^2)$ . The result is transformation from solution  $B^*$  to a new solution  $(G, (B^*))$  in which integer budget is assigned to each node while objective value does not change, i.e.,  $|B^*|$ .

THEOREM 5. The solution to linear programming relaxation problem of integer delay budgeting problem on graph  $G = (V, E)$  can be transformed to optimal integer solution in polynomial time  $O(|V|^2)$ .

## 5. APPLICATION

In this section, we apply delay budgeting in mapping datapath of an application on FPGA platform. Delay budgeting is exploited in library mapping. First we describe the experimental setup and then we present some experimental results applied to some DSP benchmarks.

### 5.1 Experimental Setup

In Figure 6, CAD flow of IP-based (or *core-based*) mapping an application on a FPGA is illustrated. Xilinx Coregen tool generates and delivers parameterizable cores optimized for target architecture. The parameters include data width, registered output, number of pipeline stages, etc. Core layout is specified up front. Cores are delivered with optimally floorplanned layouts. Also, performance of cores are independent of FPGA device size. Hence, more predictable results can be obtained during front-end optimization. Since CoreGen cores are pre-optimized, they are considered as black boxes during the synthesis. Hence, synthesis is ignored in core-based design. In a rich core library, there can exist several cores realizing same functionality with different implementation and latency (in terms of clock cycle). Figure 7 demonstrates a trade-off between latency and area of a core-Gen 16-bit multiplier with target FPGA VirtexE, Xilinx. *Slices* are the logic blocks in VirtexE FPGA series.

We start from a DAG representation of an application. Therefore in this experiment, resource sharing is not applied. Each node corresponds to a computation in data path. This assumption is reasonable

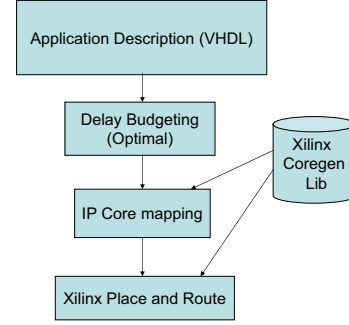


Figure 6: Mapping an Application on FPGA Using IP Library.

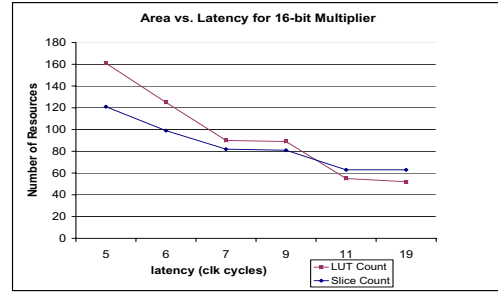


Figure 7: Area vs. Latency for a 16-bit CoreGen Multiplier.

for pipelined datapath or loops of datapaths. Benchmark in our experiments is a set of some standard DSP benchmarks. The type of computations are multiplier, adder, subtracter and shifter. We assume all the datapaths are 16-bit wide.

Each computation is assigned to a resource generated from CoreGen tool based on delay budget allocated to the node. We apply a delay budgeting algorithm to allocate the delay budget at each node. Then the whole circuit is placed and routed on a FPGA device. We used ISE 4.1 place\_and\_route tool provided by Xilinx<sup>TM</sup>. The target device is VirtexE 300 under clock frequency of 75 MHz.

Among different computations in the applications, Coregen has a relatively complete library (See Figure 7). Hence we applied delay budgeting only among the nodes that correspond to computation type *multiply*. We conducted two sets of experiments. Once we applied our optimal delay budgeting and once we applied a heuristic budgeting (ZSA like) to distribute the latency in graph.

### 5.2 Experimental Results

The original latency and other characteristics of the benchmarks are given in Table 1.

Benchmark	Nodes	Latency	Slices	LUTs
DiffEq	10	18	780	1030
ARF	28	20	1982	2476
FDCT	42	14	2044	1734
EWf	34	25	1138	1472
DCT	33	14	1618	1338

Table 1: Benchmark Information and Core-based Implementation Results.

Table 2 summarizes the implementation results of applying delay budgets to the applications. Latency of each application is the original latency reported in Table 1 plus the excess latency ( $\Delta T$ ) applied to the circuit. The excess latency is distributed in graph using delay budgeting algorithm. We use both exact (our optimal method) and heuristic (ZSA like) methods. Area (number of used slices of FPGA device) and place-and-route runtime and total budget are reported.

Benchmark	Runtime Area	$\Delta T=0$	$\Delta T=1$ clk cycle			$\Delta T=2$ clk cycle			$\Delta T=4$ clk cycle			$\Delta T=6$ clk cycle		
			Heuristic	Optimal	Imp	Heuristic	Optimal	Imp	Heuristic	Optimal	Imp	Heuristic	Optimal	Imp
Diffeq	area(slices)	780	740	700	5.7%	708	652	8%	708	652	8%	672	582	15.5%
	PAR(sec)	15	10	10	1	11	9	1.2	14	9	1.56	11	7	1.6
	Budget(clk cyc)	-	2	3	50%	4	6	50%	8	12	50%	12	18	50%
ARF	area(slices)	1982	1806	1803	0.2%	1670	1665	0.3%	1662	1553	6.6%	1518	1290	15%
	PAR(sec)	45	42	29	1.5	43	26	1.65	42	25	1.68	39	20	1.95
	Budget(clk cyc)	-	32	38	19%	36	48	33%	44	68	54%	52	88	69%
Fdct	area(slices)	2044	1867	1734	7.1%	1728	1491	14%	1718	1474	14.2%	1574	1222	22.3%
	PAR(sec)	48	39	39	1	38	36	1.05	36	36	1	43	21	2.04
	Budget(clk cyc)	-	14	20	43%	22	34	54%	38	62	63%	54	90	66.7%
Ewf	area(slices)	1138	1094	1016	7.2%	1058	982	7.2%	1054	942	10.6%	1018	906	11%
	PAR(sec)	24	21	18	1.67	19	17	1.11	20	17	1.17	19	15	1.27
	Budget(clk cyc)	-	2	6	200%	4	10	150%	8	18	125%	12	26	116%
Dct	area(slices)	1338	1091	1032	5.4%	1038	996	4%	1031	990	4%	977	918	6%
	PAR(sec)	38	19	18	1	20	15	1.33	19	14	1.36	18	13	1.4
	Budget(clk cyc)	-	24	27	12.5%	30	34	13.3%	42	48	14%	54	62	8%
Average	area(slices)	1456	1327.6	1257	5.4%	1240	1157.2	7%	1234.6	1122.2	10%	1151.8	983.6	14.6%
	PAR(sec)	34	26.2	22.8	1.15	26.2	20.6	1.27	26.2	20.2	1.29	26	15.2	1.7
	Budget(clk cyc)	-	14.8	18.8	27%	19.2	26.4	37.5%	28	41.6	49%	36.8	56.8	54%

**Table 2: Area (#slices-logic blocks), total Budget, and Runtime of Place-and-Route (sec) vs. delay budget (clk cyc). Imp column compares Optimal over heuristic. It indicates the percentage of improvement for area and budget and the ratio of runtime for PAR runtime.**

The first column shows the place\_and\_route (PAR) runtime, area of slices when no delay budget is applied. The next columns show the area and PAR runtime for different excess delay to required time ( $\Delta T$ ) of 1, 2, 4, and 6 clk cycles. The *Imp* column shows the percentage of improvement in area in different delay budgeting computed as  $\frac{Area(Heu) - Area(opt)}{Area(Heu)} \times 100$ . Similarly Imp is computed for total budget. The *Imp* column computes the improvement in runtime as ratio of  $\frac{PAR\_runtime(Heu)}{PAR\_runtime(opt)}$ .

The results show the average improvement in area for 5.2%, 7%, 10% and 14.6% in terms of number of slices when optimal algorithm is used for budgeting compared to area resulted by heuristic delay budgeting for different  $\Delta T$ . The larger  $\Delta T$ , the more delay budget is distributed. Although budget increases significantly by  $\Delta T$ , the improvement in area is not as significant as budget. One reason is that there does not necessarily exist another component in the target library for large delay budget. For example in *Fdct*, there are some multipliers on non-critical path with large delay budget which is not exploited in library mapping. Although the area of applications by optimal delay budgeting is always smaller than the area resulted by heuristic method by 10% on average, runtime of place\_and\_route in some application does not speed up. One reason is that some of applications such as *Fdct* are I/O bounded. A main portion of place and route is dedicated to I/O placement and routing. In other benchmark such as *ARF* the runtime of place\_and\_route gets almost two times faster. On average for excess delay budgeting of 6 cycles, the runtime of place\_and\_route gets faster by factor of 1.7. Note that we only applied budgeting to multipliers. Also the size of benchmarks are relatively small. Although speedup in PAR runtime were not significant in these applications, due to lesser complexity and smaller structure, the effect on runtime for place and route can be more visible when these applications are integrated into larger systems and mapped on large FPGAs.

As a result, delay budgeting gives the opportunity of mapping the applications to components in the target library with simpler structure and smaller area. Comparing the result of the first column when no budget is applied with the results of the next columns demonstrates this fact. However, the current libraries are not rich enough and do not contain different components with different latencies for same functionality. Developing complete libraries facilitates the design CAD tool to exploit the existing delay budget to improve design quality.

## 6. CONCLUSIONS

General delay budgeting can be solved using linear programming solver. Due to numerical instability and discrete behavior of libraries of components, integer solution is required. Complexity of integer

budgeting has been an open problem for the last decade. In this paper, using optimal solution to LP relaxation of budgeting problem, we transform the solution to optimal integer solution. We re-assign the fractional value of budget associated with the nodes in the graph such that budget of each node becomes integer. We prove that during this transformation ( $O(|V|^2)$ ), objective value from optimal LP solution does not change. Hence an optimal integer solution is obtained in polynomial time. We applied our budgeting technique in mapping applications on FPGA device. Using IP library of different computations, delay budget is exploited to improve the area, hence to speedup the runtime of place-and-route. Our optimal algorithm outperforms ZSA algorithm [3] in terms of area and design time significantly.

## 7. REFERENCES

- [1] L. A. Wolsey. *Integer Programming*. New York, NY, Wiley-Interscience Publisher, John Wiley & Sons Inc., pg. 37-52, 1998.
- [2] C. Chen, E. Bozorgzadeh, A. Srivastava, and Majid Sarrafzadeh. "Budget Management with Applications". In *Algorithmica*, vol 34, No. 3, pp. 261-275, July 2002.
- [3] R. Nair, C. L. Berman, P. S. Hauge, E. J. Yoffa. "Generation of Performance Constraints for Layout In *IEEE Transactions on Computer-Aided Design*, Vol. 8, No. 8, pp. 860-874, August 1989.
- [4] M. Sarrafzadeh, D. A. Knol, G.E. Tellez. "A Delay Budgeting Algorithm Ensuring Maximum Flexibility in Placement In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 16, No. 11, pp. 1332-1341, Nov. 1997.
- [5] C. Kuo and A. C.-H. Wu. "Delay Budgeting for a Timing-Closure-Design Method", In *International Conference on Computer-Aided Design*, pp. 202-207, 2000.
- [6] C. Chen, X. Yang, M. Sarrafzadeh. "Potential Slack: An Effective Metric of Combinational Circuit Performance. In Proc. of ACM/IEEE International Conference on Computer-Aided Design, pp. 198-201, 2000.
- [7] Y. Liao and C. K. Wong. "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints". In *Proceedings of IEEE Transactions on CAD*, Vol. 2, No. 2, April. 1983.
- [8] J. F. Lee and D. T. Tang. "VLSI layout compaction with grid and mixed constraints". In *Proceedings of IEEE Transactions on CAD*, Vol. 6, No. 5, Sep. 1987.
- [9] 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 *Proceedings of Conference on European Design Automation*, November 1992.
- [10] P. Girard, C. Landrault, S. Pravossoudovitch, and D. Severac. "A Gate Resizing Technique for High Reduction in Power Consumption. In Proc. of *International Symposium on Low Power Electronics and Design*, pp. 281-286, 1997.