Optimal Integer Delay Budget Assignment on Directed Acyclic Graphs

Elaheh Bozorgzadeh, *Member, IEEE,* Soheil Ghiasi, *Student Member, IEEE,* Atsushi Takahashi, *Member, IEEE,* and Majid Sarrafzadeh, *Fellow, IEEE*

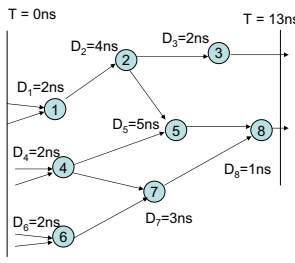
***Abstract*— Excess delay that each component of a design can tolerate under a given timing constraint is referred to as delay budget. Delay budgeting has been widely exploited to improve the design quality in VLSI CAD flow. The objective of the delay budgeting problem investigated in this paper is to maximize the total delay budget assigned to each node in a directed acyclic graph under a given timing constraint. Due to discreteness of the timing of the components in the libraries during design optimization flow, discrete solution for delay budgeting is essential. We present an optimal integer delay budgeting algorithm. We prove that the problem can be solved optimally in polynomial time. In addition, we look at different extensions of the delay budgeting problem, such as maximization of weighted summation of delay budgets assigned to the nodes with constraints on lower bound and upper bound on the delay budget allocated to each node. We prove that for both aforementioned extensions, our algorithm can produce an optimal integer solution in polynomial time. Our algorithm is generic and can be applied in different design tasks at different levels of abstraction. We applied our proposed optimal delay budgeting algorithm in library mapping during datapath synthesis on an 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 [4] 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.**

***Index Terms*— delay budgeting, timing constraint, core-based design implementations,integer programming.**

I. Introduction

Due to the complexity of system design and high uncertainty of timing issues and quality metrics, it is not effective to optimize performance intensively in earlier stages of VLSI CAD flow. Instead, the optimization should aim at ensuring correctness and convergence of the design. In order to abstract away the design complexity, each design is decomposed into a set of sub-designs.

The essential constraint during the design optimization flow is the timing constraint. Along with timing, there are other constraints such as size, power dissipation, etc. In order to manage the timing constraint, a percentage of the total delay in a complex design is dedicated to each sub-design. 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 delay allocated to 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*). In other words, delay budget assignment is the problem of assigning the upper bound on the delay (or latency) of all the sub-designs under the given timing constraint. Delay assignment is applied at all levels of abstraction in VLSI CAD flow.



Node	Delay (nsec)	Budget A	Budget B
1	2	-	-
2	4	-	1
3	2	5	4
4	2	5	-
5	5	-	-
6	2	5	-
7	3	2	7
8	1	-	-
Total Budget	--	17	12

Fig. 1. Delay budgeting problem on a directed acyclic graph.

There is usually a trade-off between timing and some other design metrics such as size, power consumption, throughput, etc. Hence, delay budgeting can be exploited through the whole CAD design flow to improve the other design metrics such as area, power consumption, etc. The more delay budget assigned to the design, the more flexibility would be given for further optimization on other design constraints. From another point of view, larger upper bound on delay relaxes the optimization, hence resulting faster compilation and design time.

Each design is represented by a directed acyclic graph (DAG), $G = (V, E)$. There is a delay associated with each node. Let the timing constraint be the maximum latency (or delay) at the output nodes. Delay or latency at the output is computed as the longest path delay in the graph from input to output. The delay along each path is the total delay associated with the nodes and/or edges along the path. 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 for the 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 nodes/edges is not the total budgets nodes/edges can tolerate.

Manuscript received August 23, 2003; revised October 2, 2003. This work was recommneded by Associate Editor John Lillis.

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

A. Takahashi is with the Department of Communications and Integrated Systems at Tokyo Institute of Technology, Tokyo 152-8552 Japan.

M. Sarrafzadeh and S. Ghiasi are with the Department of Computer Science at the University of California, Los Angeles, CA 90095 USA.

In Figure 1, two different methods of delay budgeting (A and B) are applied on a DAG. Columns "Budget A" and "Budget B" of the table correspond to excess delay (delay budget) assigned to each node under timing constraint ($13nsec$) in approach A and approach B. 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 12.

In this paper, we study the problem of the assignment of maximum total budget in a graph. The budgeting problem in a graph is well studied in theory and practice and is widely used in today's industry and research. Delay budgeting has several applications in design optimization as follows:

- Timing-driven placement and floor planning- Delay budgeting during placement and floor planning has been extensively studied by several researchers [7], [8], [9], [10]. In timing-driven placement, the goal is to optimize the path delays with fewer numbers of iterations. Delay budget is assigned to edges in the graph. Per-net delay bounds are considered in order to have a better distribution of delay budgets in the graph. In [6], [7], placement and re-budgeting are combined. The optimization problem of budgeting on the edges in a graph is formulated as a piece-wise linear objective function and solved using a modified graph-based Simplex algorithm ([1]).
- Gate/wire sizing and power optimization- Under timing constraint, gate sizing problem is to find a set of nodes/edges in the graph such that their physical size can be reduced by mapping to smaller cell instances with larger delays from a target library [17], [18]. In general, delay budgeting can be applied during *library mapping* stage. Delay budget at each node can be exploited to map the node to a smaller cell (or with a lower power consumption) with a larger delay [10].
- VLSI layout compaction- The main objective is to minimize the physical area of the layout. In addition, minimizing wirelength cannot be ignored during the optimization. Concept of budget in such problems is exploited to reduce wirelength [14]. An important constraint in analog IC design is the symmetry constraint in layout. With multiple symmetry constraints, layout compaction is solved using LP solver[15]. In [16], a graph-based simplex method is applied to improve the runtime of linear programming algorithm. LP formulation of compaction is similar to formulation of the delay budgeting problem. The space budget is assigned along the x-axis or y-axis to leave a sufficient space for wiring.
- Exploiting slack in high-level synthesis- There exists 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 [21], power minimization under timing constraint [19], [20], etc. In [21], the design entry is a pipelined datapath. In the problem formulation, there are a set of constraints regarding the number of registers and depth of pipeline stages, which are not considered in budgeting on directed acyclic graphs. All the proposed algorithms are heuristic sub-optimal algorithms.

There are heuristic algorithms in literature and industry to solve the delay budgeting problem such as MISA [3] and ZSA [4] algorithms. In maximum delay budgeting, the objective is to maximize the value of an expression, which is a function of budgets associated with the nodes/edges in a graph. The most popular and efficient algorithm for delay budgeting is zero-slack algorithm (ZSA)[4], [5]. The solution is not optimal and can be far away from optimal result. MISA algorithm proposed in [3] finds the total budget in the graph with a more sophisticated and intuitive technique using maximum independent set in the graph. 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 and need to be normalized. According to the following reasons optimal *integer* solution is preferred: First, the space/timing budget is mostly a discrete value especially at higher levels of abstraction. For example, delay on interconnect is discrete in grid-based global routing. At datapath level, latency of each component is 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 [12]. Secondly, 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. Hence, in the formulation of the delay budgeting problem, we assume the variables associated with the 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 since budgeting problem was first formulated by Wong, et. al. in [11]. 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 the 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 ($O(V^2)$). The preliminary version of this work is published in DAC'03. In this paper, we describe the detailed analysis of our delay budgeting algorithm. In addition, we look at different extensions of the delay budgeting problem, such as maximization of weighted summation of delay budgets assigned to the nodes and additional constraints on lower bound and upper bound on the delay budget allocated to each node. We prove that in both aforementioned extensions, our algorithm can produce an optimal integer solution in polynomial time.

We apply delay budgeting technique in library mapping stage. Mapping sub-designs to already existing pre-optimized and synthesized components is an unavoidable scheme to abstract away the complexity of the given design to be optimized. For faster compilation and exploiting the architectural features of FPGAs, FPGA vendors provide a relatively rich IP (*Intellectual Property*) library of arithmetic functions and application-specific operations such as MAC, FFT, and DCT in DSP domain. Along with this growth, design automation and synthesis flows need to be able to exploit the existing libraries with a better design planning. We apply our methodology and technique for timing budget management during library mapping at datapath level. The datapath of a given application at behavioral level is mapped to the components in a library customized for the target programmable architecture (e.g. FPGA libraries). We applied timing budgeting algorithm in selecting the components of the library and mapping to different components of the application such that the design complexity is reduced without violation of timing constraints. Using IP library of FPGAs, we show that the delay budgeting resolves the trade-off between latency of a datapath and area of hardware resources. Our empirical results show that delay budgeting yields a solution with smaller area and faster design time compared to the case in which no delay budgeting is applied. We compare our proposed optimal delay budgeting algorithm with ZSA, the well-known sub-optimal heuristic algorithm. 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 in library mapping.

The rest of the paper is organized as follows: In Section II, the problem is formally defined. In Section III, the budget re-assignment is proposed. Applying budget re-assignment on LP solution of budgeting problem is described in Section IV and it is proven that the final solution is integer and optimal. In Section V, two different extensions to the formulation of integer delay budgeting problem are presented on which our algorithm can be applied to produce optimal integer solution. In Section VI, the experimental results on trade-off between latency and area by budgeting technique in FPGA platform are presented. In Section VII, conclusions and some possible future directions are outlined.

II. LP Formulation of Delay Budgeting Problem

In a directed graph $G = (V, E)$, edge e_{ij} is incident to node v_j and incident from node v_i. $V_I(i)$ 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 nodes with no outgoing edges. Associated with each node v_i, there is a delay variable $d_i > 0$. Assume node v_i drives node v_j, i.e., there is an edge e_{ij} in graph G. If data or signal at the output of node v_i is ready at time t, the output of node v_j is ready at least at time $d_j + t$. Let b_j be the extra potential delay assigned to node v_j. Hence, the output at v_j will not be ready before $t + d_j + b_j$.

arrival time of v_i: Arrival time at node v_i is defined as the maximum total path delay among all the paths from PI nodes to node 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. a_i is computed as

$$a_i = \max_{v_j \in V_I(i)} a_j + (d_i + b_i). \tag{1}$$

Arrival time at a primary output is maximum summation of the delay budget and the intrinsic delay associated with each node along the path from primary input to primary output. Arrival time at each primary output cannot exceed a fixed value, T. This is referred to as *required time* at primary outputs. Although requited 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:

$$Max \quad \sum_{v_i \in V} b_i \tag{2}$$
$$a_j \geq a_i + b_j + d_j \quad \forall e_{ij} \in E \tag{3}$$
$$a_i \leq T \quad \forall v_i \in PO \tag{4}$$
$$a_i = 0 \quad \forall v_i \in PI \tag{5}$$
$$d_i, b_i, T \in Z_+ \quad \forall v_i \in V. \tag{6}$$

General LP formulation of budgeting problem is $max\{\sum_{i=1}^{|V|} x_i | A\mathbf{x} \leq \mathbf{b}\}$. Constraint matrix A corresponding to abovementioned LP formulation of budgeting problem is as follows. Variable x_j corresponds to a_j if $1 \leq j \leq n$ and corresponds to b_j if $n < j \leq 2n$. Each row index corresponds to an edge in graph G. Assume $A = (a_{ij})_{m \times n}$. For $1 \leq j \leq n$, $a_{ij} = 1$ if edge e_i is incident from node j and $a_{ij} = -1$ if edge i is incident to node j (e_{ij} is incoming edge to node v_i). Otherwise, it is set to zero. For $n < j \leq 2n$, $a_{ij} = 1$ is edge i is incident to node $j - n$.

In the area of linear programming theory, there has been a deep study on the linear programs that 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 [2].

Totally Unimodular Matrix (TU): A matrix A is totally unimodular (TU) if every square sub-matrix of A has determinant $+1$, -1, or 0 [2].

Lemma 1: If matrix A is TU, the linear programming relaxation can solve the ILP [2].

A set of sufficient conditions for matrix $A = (a_{ij})_{m \times n}$ to be totally unimodular is proposed by Heller and Tompkins [2] as follows:

Lemma 2: A matrix $A_{m \times n} = (a_{ij})_{m \times n}$ is TU if

- $a_{ij} \in \{-1, +1, 0\}$, $\forall a_{ij}, 1 \leq i \leq m, 1 \leq j \leq n$.
- Each column contains at most two non-zero coefficients, i.e. $\sum_{i=1}^{m} |a_{ij}| \leq 2$).
- There exists a partition (M_1, M_2) of the set M of rows such that each column j containing two nonzero coefficients satisfies $\sum_{i \in M_1} a_{ij} - \sum_{i \in M_2} a_{ij} = 0$ [2].

By total unimodularity (TU) of coefficient matrix every extreme point of LP relaxation is integral regardless of objective function.

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 other 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.

III. Delay Budget Re-assignment

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 6 are not satisfied. 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 $s_i = r_i - a_i$. The value of *a-slack* for edge e_{ij} is computed as $\epsilon_{a_{ij}} = (a_j - (d_j + b_j)) - a_i$, $e_{ij} \in E$. Similarly, *r-slack* of e_{ij}, is computed as $\epsilon_{r_{ij}} = (r_j - (d_j + b_j)) - r_i$, $e_{ij} \in E$. 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 that includes only critical edges is called *critical path.*In a directed graph G, if the slack of the nodes at the two ends of the edge e_{ij} are equal, i.e., $s_i = s_j$, then $\epsilon_{a_{ij}} = \epsilon_{r_{ij}}$. ϵ_{ij} is used to refer to this value as the slack of edge e_{ij}.

In any budgeting on graph G, slack of each node/edge must be non-negative. This is referred to as *feasibility* in graph. A graph with budgeting B is not feasible if the timing slack of any node/edge is negative.

Maximal Budgeting Graph (G, B_m): Let B_m be the set of delay budgets assigned to the nodes in graph G. B_m is a feasible solution to budgeting problem on a directed acyclic graph G. Feasible solution 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.

In graph G, if the slack of each node is zero, the corresponding budgeting B_m is a maximal budgeting. In a maximal budgeting, all the non-critical edges have the same a-slack and r-slack values. The term ϵ_{ij} is used to refer to the slack of non-critical edge e_{ij}.

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 [3] and ZSA algorithm [4].

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.

Proof: By the way of contradiction, we assume that there is no critical outgoing edge from v_i. v_i is not a primary output. There has to exist at least one outgoing edge from v_i. If there are k non-critical outgoing edges from node v_i then ϵ_{ij}, $j = 1, ..., k$ is not zero. The slack of each node is zero. Therefore, $min(\epsilon_{ij})$, $j = 1, .., k$ can be added to the budget of node v_i while all the arrival time and required time constraints for the whole graph are met. Hence, more delay budgets is obtained, and this contradicts the definition of maximal budgeting. Similar argument can be applied to prove that at least one incoming edge to each node must be critical. ■

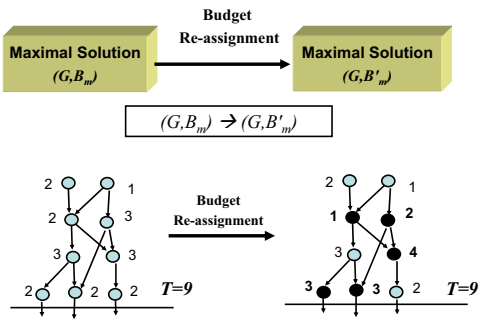


Fig. 2. Conservative budget re-assignment.

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

Feasible Budget Re-assignment on (G, B_m): In a 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 β-budget re-assignment on (G, B_m) is a feasible budget re-assignment in which the change of budget in each node is either $\pm\beta$ or 0.

In Figure 2, example of feasible budget re assignment on a DAG is shown. After feasible budget re-assignment, the budgeting is maximal and feasible.

Assume that in a re-assignment of budget of $\{\pm\beta, 0\}$ at each node in graph G, the total amount of change in the budget of the nodes along each critical path is zero. In this case, arrival time at each node v_i is changed by $k_i\beta$. Since budget of each node changes either β or $-\beta$, the change of budget along each critical path from PI to node v_i is multiple of β, say $k_i\beta$, $k_i \in Z$. Theorem 2 presents two sufficient conditions for feasible β-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 β-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, and
- for each ϵ-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.

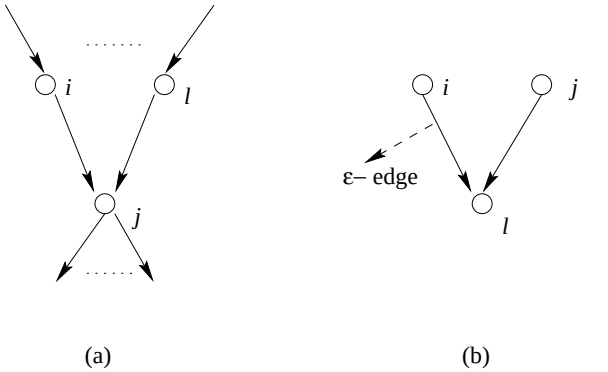


Fig. 3. Two sufficient conditions for feasible β-budget re-assignment.

Proof: We prove that after the budget re-assignment of $\{\pm\beta, 0\}$, (G, B'_m) is a feasible maximal budgeting. a_j is arrival time at node v_j before budget re-assignment. By induction, it can be shown that arrival time at v_j is $a_j + k_j\beta$ after budget re-assignment. $k_j\beta$ is total budget re-assignment along the critical paths from PI to node v_j. As shown in Figure 3(a), edges e_{ij} and e_{lj} are both critical in graph (G, B_m). After budget re-assignment, arrival time at node v_i is $a_i + k_i\beta$. Arrival time at node v_l is $a_l + k_l\beta$. Since along the critical path from PI to PO through edge e_{lj} total change in budget is zero, the amount of change in budget along any critical path from node v_j until PO is $-k_l\beta$. Since the critical path from PI through nodes v_i and v_j until PO is critical, total budget along this path is zero, that is $k_i\beta \pm \beta - k_l\beta \mp \beta = 0$. Hence $k_l = k_i$. Non-critical edges in the graph need to be considered. There is a slack of ϵ associated with each non-critical edge. In Figure 3(b), edge e_{il} is a non-critical edge and $\epsilon_{il} = a_j - a_i$. After budget re-assignment edge e_{jl} has to remain critical. Based on the second condition in the Theorem $\epsilon_{il} \geq (k_i - k_j) \cdot \beta$, arrival time a_i cannot become greater than arrival time a_j. Hence, edge e_{jl} remains critical.

Now assume node v_i is a primary output. Arrival time at node v_i is $a_i + k_i\beta$. $k_i\beta$ is the total change of budget along the critical paths from PIs to node PO. Due to first condition k_i is zero. Hence, arrival time at node v_i does not change after budget re-assignment, i.e. feasibility is satisfied ($a_i \leq T$).

The arrival time at node v_j is $a_j + k_j\beta$. Similar argument can be applied to show that the amount of change in the required time at node v_j is $r_j - k'_j\beta$, where $k'_j\beta$ is the amount of change in the total budget along the critical paths from primary outputs to node j. Slack of node v_i after budget change is $r_i - a_i - k'_i\beta - k_i\beta$. We have $s_i = r_i - a_i = 0$ since B_m is a maximal budgeting. $k_i\beta + k'_i\beta$ is equivalent to total change of budget along the critical path through node v_j, which is zero. Hence slack of node v_j after budget change is still zero. Therefore, we have a feasible maximal budgeting. ■

According to Lemma 2, if the budget of β is re-assigned among the nodes under the aforementioned conditions, another maximal solution on graph G is obtained. We show that the budget exchange between two sub-graphs under child-parent relation satisfies the sufficient conditions, hence a feasible β-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 $v_1 \sim_p v_2 ... \sim_p v_n$, 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 $v_1 \sim_c v_2 ... \sim_c v_n$, 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 $p(v_i) \sim_p^* p(v_j)$.

Proof: If $v_i \sim_c^* v_j$, there exists node v_l such that $v_i \sim_c v_l$ and $v_j \sim_c^* v_l$. Hence, nodes v_i and v_l share a parent, i.e., $p(v_i) \sim_p p(v_l)$. According to transitive property in child and parent relation, it can be shown that $p(v_i) \sim_p^* p(v_j)$. ■

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

Proof: Nodes $v_i \sim_p v_j$. Let v_k be the child node of nodes v_i and v_j. Since v_i is a parent of node v_k, $a_k = a_i + b_k + d_k$. Similarly a_k is equal to $a_j + b_k + d_k$. Hence, $a_i = a_j$. If v_i and v_j do not share a common child, due to transitive property in equivalent parent relation, $v_i \sim_p v_k \sim_p ... v_l \sim_p v_j$, arrival time at v_i is equal to arrival time at v_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 4. 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 $\forall v_i \in V, d_i > 0$.

Proof: Since $v_i \sim_p^* v_j$, by Lemma 5, $a_i = a_j$. If there is a critical path between v_i and v_j, then a_i cannot be equal to a_j according to definition of arrival time and critical edges with assumption of $d_i > 0$. Hence there cannot exist any critical path between any two nodes in the parent set. ■

Lemma 7: If nodes $v_i \sim_c^* v_j$, there is no directed critical path between v_i and v_j if $\forall v_i \in V, d_i > 0$.

Proof: Assume that there is a critical path $P_{ij} = \{v_i, ...v_k, v_j\}$ connecting nodes v_i and v_j. Since v_k is $p(v_j)$, v_k belongs to the parent set according to Lemma 6. Hence $p(v_i) \sim_p^* v_k$. That is $a_k = a_{p(v_i)}$. However, since there is path between v_i and v_k and delay of each node is non-zero, $a_{p(v_i)} < a_{v_i} < a_{v_k}$. Therefore, by the way of contradiction, the path P_{ij} cannot be critical. ■

Lemma 8: In a parent-child set (S_p, S_c), S_p and S_c do not intersect if $\forall v_i \in V, d_i > 0$.

Proof: Assume that the two sets intersect at node v_k. Since node v_k is in set S_p, node v_k has at least a child in set S_c, say node v_l. Therefore, there is an edge from node v_k to node v_l both belonging to S_c. This contradicts Lemma 7. ■

Let β-budget exchange in parent-child set (S_p, S_c) be decreasing the budget of the nodes in S_p by β and increasing

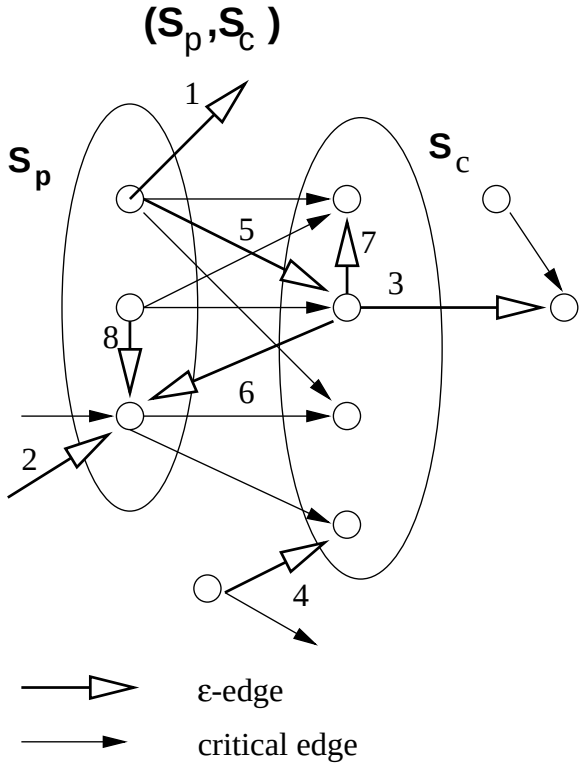


Fig. 4. ϵ-edges with respect to parent-child set (S_p, S_c).

budget of nodes in S_c by β, $\beta > 0$.

Lemma 9: In a given (S_p, S_c) in (G, B_m), if $\beta \leq min(\epsilon_{ij})$, where ϵ_{ij} is the slack of the non-critical ϵ-edge with $v_j \in S_c$ and $v_i \notin (S_p, S_p)$ (incoming ϵ-edges to S_c), the β-budget exchange is a feasible β-budget re-assignment in (G, B_m).

Proof: We show that the sufficient conditions in Theorem 2 are satisfied during budget exchange between a parent-child set. Since there is no critical path between any two nodes in S_c or S_p, the critical paths in $S_c \cup S_p$ consist of two nodes, one in parent set and the other in child set. At each parent node v_i, the amount of change in arrival time is $-\beta$. At each child node v_j, the amount of change in arrival time is zero. Hence, along each critical path in this subgraph, the total amount of change in budget is zero. In addition, there is no change in budget or arrival time at any other nodes outside the parent-child set in graph G. Therefore the first sufficient condition in Theorem 2 is satisfied. The ϵ-edges can be categorized based on where the two ends of the edges are located. Figure 4 shows all such possible edges with respect to a given parent-child set. At each child node v_j, the amount of change in arrival time is zero. Therefore, arrival time at a child node and hence the criticality of the edges connecting the child nodes to the rest of the graph remains unchanged. Similarly, the criticality of incoming edges to parent nodes are unchanged after budget exchange, i.e., ϵ-edges 3 and 2 remains non-critical. The inequality $\epsilon_1 \geq \beta$ is satisfied as well. For ϵ-edge 4, the inequality $\epsilon_4 \geq \beta$ is held since $\beta \geq \epsilon$ for all incoming ϵ-edges to child set. There cannot exist any ϵ-edges between two parent nodes, two child nodes, or between a child and a parent node. Hence the second sufficient condition in Theorem 2 is satisfied. ■

Similarly, the budget can be increased by β in parent set and reduced by β in child set. This is called $(-\beta)$-*budget exchange* in (S_p, S_c). Lemma 9 can be adjusted to be applied for $(-\beta)$-budget exchange on parent-child set as well. In this paper, we apply β-budget exchange on a given parent-child set. In the next section, we apply β-budget re-assignment on LP solution which is a maximal budgeting on G in order to obtain integer solution.

IV. Integer Solution to Delay 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 β in β-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 $IS_n =< v_1, v_2, ..., v_n >$ along a critical path in (G, B^*) is called an *integral sequence* if $a_1, a_n \in Z_+$ and $a_2, ..., a_{n-1} \notin Z$.

Lemma 10: The total budget of the nodes along any integral sequence in (G, B_m) is integer if $d_i, T \in Z$.

Proof: Since the arrival time of the nodes at the two ends of an integral sequence IS_n is integer, $\sum_{j \in IS_n} (d_j + b_j) \in Z_+$. Since each d_j is integer, $\sum_{j \in IS_n} b_j \in Z_+$. ■

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

Based on Lemma 10, each node with fractional budget belongs to an integral sequence. Hence, within an integral sequence, it is sufficient to re-assign the fractional budgets only on the nodes in an integral sequence. On the other hand, in graph G, there are several integral sequences connected to each other. 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 a given graph (G, B^*). The nodes in graph G_f are a subset of nodes in graph G that have non-integer (fractional) budgets. 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 non-critical ϵ-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 ϵ-edges along the path. Among all different paths between the two nodes, the minimum of total ϵ value of the ϵ-edges along each path is the ϵ value of the ϵ-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

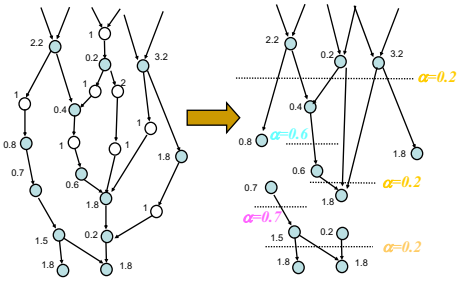


Fig. 5. (G, B^*_{LP}) and corresponding Fractional Adjacency Graph G_f.

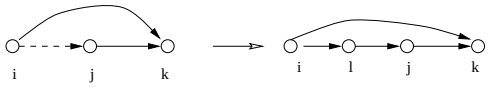


Fig. 6. In graph G_f, nodes v_i and v_j share a child (node v_k) while there is a directed path from v_i to v_j.

G with fractional budget, both belonging to same integral sequence. Figure 5 demonstrates a budgeted DAG (G, B^*_{LP}) and the corresponding fractional adjacency graph.

β-budget re-assignment is applied on graph G_f such that the budget of all the nodes become integer. Only fractional value of budgets needs to be re-assigned in order to obtain integer solution. Hence β 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 11: 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., $a_i - [a_i] = a_j - [a_j]$.

Proof: Assume $v_i \sim_p v_j$. Let v_k be the child node of both nodes v_i and v_j. Arrival time at node v_k is equal to fractional value of summation of fractional value of arrival time at node v_i and fractional value of budget at node v_k. Budget of the nodes along the critical path from v_i to v_k in graph G are all integer. Similarly, arrival time at node v_k is equal to fractional value of summation of fractional value of arrival time at node v_j and fractional value of budget at node v_k. Hence, $a_i - [a_i] = a_j - [a_j]$. If v_i and v_j do not share a child node, due to transitivity in parent relation, we still have $a_i - [a_i] = a_j - [a_j]$. ■

Lemma 12: 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.

Proof: Assume there is a path between node v_i and v_j in graph G. Let node v_k be the child node of nodes v_i and v_j. There are two paths from node v_i to v_k, one is the direct edge e_{ik} and the other is the path $(v_i, v_j)(v_j, v_k)$. See Figure 6. The fractional value at the node v_k from the first path is $a_i - [a_i]$ and from the other path is $a_i - [a_i] - a_j + [a_j]$. According to Lemma 11, these two values need to be equal. This is possible iff $a_j - [a_j] = 0$ which contradicts that $e_{jk} \in E(G_f)$. Therefore there has to exit at least one node say v_l on the path from v_i to v_j such that $a_i - [a_i] + a_l - [a_l] = 0$. Similarly if the two nodes v_i and v_j do not have a same child, we can prove that the total fractional value on the path from v_i to v_j including v_j needs to be integral, i.e. there has to exist at least one node on the path between v_i and v_j. Figure 7 shows such a case. ■

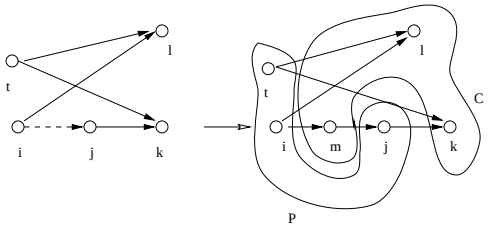


Fig. 7. In graph G_f, $v_i \sim^*_p v_j$ while there is a directed path from v_i to v_j.

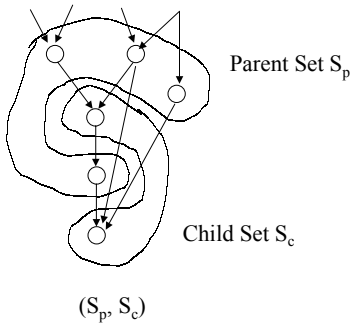


Fig. 8. Parent-Child Set (S_p,S_c) in graph G_f of graph G.

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 8, a parent-child set in G_f is shown.

According to Lemma 12, Lemma lemma:int is derived.

Lemma 13: Set $S_p(v_i)$ and $S_c(v_j)$ do not intersect ($e_{ij} \in E(G_f)$).

Proof: Assume that the two sets intersect, i.e., there is a node v_k belonging to both sets. Since node v_k is in set $S_c(v_j)$, it has at least one parent, say $v_l \in S_p(v_i)$. Therefore there is an edge from node v_l to node v_k). On the other hand, $v_k \in S_p(v_i)$. That is there is a direct edge between two nodes $v_k, v_l \in S_p(v_i)$. This contradicts Lemma 12. ■

On a given parent-child set in graph G_f, we apply β-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 sequence. There are ϵ-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.

Lemma 14: β-budget exchange on a parent-child set in

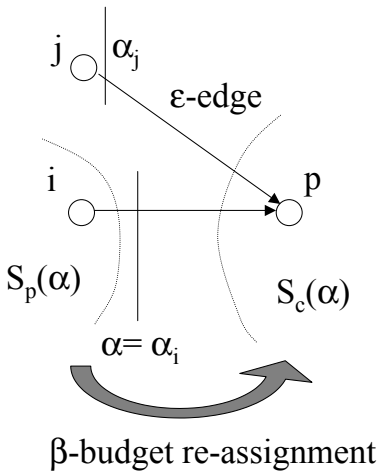


Fig. 9. ϵ-edge incident to a child node in $(S_p, S_c)_\alpha$.

graph G_f is a feasible β-budget re-assignment if $\beta \leq min(\epsilon_{ij}, \alpha)$, where ϵ_{ij} is ϵ-edge. e_{ij} is an incoming edge to child set. α_i is the fractional value at parent nodes.

Proof: In Figure 8, a set of parent-child set is shown in a given graph G_f. In a budget exchange on the set, there is an alternative $\pm\beta$ budget exchange along each critical edge in graph G. $k_i\beta$ corresponds to total change of budget along the critical paths from PI to node v_i. At each child node v_i, the corresponding k_i is zero. At each parent node v_i, the corresponding $k_i = -1$ when budget in parent set is decreased by β. Hence the first sufficient condition in Theorem 2 is satisfied. We prove that as long as $\beta \leq \epsilon$, the budget exchange is a feasible β-budget re-assignment on a given graph G. There are 8 possible type of ϵ-edges with respect to (S_p, S_c). At each edge, we check the inequality defined in Theorem 2 after budget exchange. ϵ-edges 2 and 3 will not change since the arrival time at the child set and incoming edges to parent set are not affected by budget exchange. At ϵ-edge 1, the inequality $\epsilon \geq (-1-0)\beta$ is $True$ for any $\beta > 0$. At ϵ-edge 5, the inequality $\epsilon \geq (-1-(-1))\beta = 0$ is $True$ for any $\beta > 0$. At ϵ-edge 6, the slack will not change since the arrival time at child node does not change. At ϵ-edge 8, the inequality $\epsilon \geq (-1-(0))\beta$ is $True$ for any $\beta > 0$. At ϵ-edges 7 and 4, the arrival time at the nodes incident from ϵ-edges do not change. $\epsilon \geq (0-(-1))\beta$ is satisfied as well. Therefore both sufficient conditions in Theorem 2 are satisfied. ■

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

In Figure 9, a ϵ-edge incident to a child node is shown. Let α_i and α_i be the fractional value of arrival time at nodes v_i and v_j, respectively. In β-budget re-assignment, if $\beta = \alpha_i$, for $\epsilon \geq 1$, $\epsilon > \beta$ is True. Assume $\epsilon < 1$. The value of ϵ is computed as follows:

$$\epsilon_{jp} = \begin{cases} \alpha_i - \alpha_j & \text{if } \alpha_i > \alpha_j \\ 1 + \alpha_i - \alpha_j & \text{if } \alpha_i < \alpha_j \end{cases} \quad (7)$$

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

Lemma 15: Let (S_p, S_c) be a parent-child set with α_p, the fractional value at the arrival time at the parent nodes. Assume that α_p is the smallest fractional value of arrival time at all the nodes in graph G_f. β-budget exchange of $\beta = \alpha_p$ from parent nodes to child nodes is a feasible budget re-assignment.

Proof: In order to be able to re-assign budget of β from parent nodes, each parent node must have at least budget of β, i.e. $\forall v_i \in S_p, b_i \geq \beta$. Assume that there is a node $v_j \in S_p$ such that $b_j \leq \beta$, hence $b_j \leq \alpha_p$. In this case, arrival time at parent of node v_j is $\alpha_p - b_j < \alpha_p$ and this contradicts the fact that fractional value of arrival time at no other nodes other than parent nodes can be as small as α_p. Hence each $b_i \geq \beta$. Next, consider ϵ-edges connected to (S_p, S_c). According to Lemma 14, only two types of ϵ-edges, ϵ-edges 4 and 7 as shown in Figure 4 are under the condition that ϵ value of such edges have to be larger than β for edges with $\epsilon < 1$. According to Equation 7, since parent set has the smallest fractional value ($\alpha_i < \alpha_j$), $\beta < \epsilon$. This ends the proof that the budget re-assignment is feasible. ■

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.

An important fact is that after budget re-assignment, the parent nodes do not have any outgoing edges in updated graph G_f. Hence, the corresponding nodes cannot become parent nodes anymore. Therefore, we have the following lemma:

Lemma 16: 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 β-budget exchanges, 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 β-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 β-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 $O(|V|)$ iterations.

If graph $G_f = \emptyset$, the budget of all the nodes in graph G are integer. Hence, Theorem 3 shows that a maximal integer

Algorithm: Integer Delay Budgeting
Input: (G,B*$_{LP}$)
Output: (G, B*$_{Z}$)

begin
1. Construct G_f from *(G,B*)*
2. **while** $G_f \neq \emptyset$
3. Start from a parent-child set with $\alpha_p = \alpha_{min}$
4. Apply budget re-assignment, $\beta = \alpha_p$
5. Update G_f and *(G,B$_Z$)*

end

Fig. 10. Pseudo-code for delay budget re-assignment on (G, B^*_{LP}).

solution can be obtained from LP solution using β-budget exchange on graph G_f. The following lemma proves that during budget re-assignment optimality is preserved.

Lemma 17: In graph G_f corresponding to (G, B^*), $|S_p(v_i)| = |S_c(v_j)|$ if $e_{ij} \in G_f$.

Proof: Assume that there are more number of nodes in one of the sets, say $S_c(v_j)$. After budget re-assignment of minimum budget, say f_{min}, the total budget changes by $|S_c(v_j)| \cdot f_{min} - |S_p(v_i)| \cdot f_{min}$. This contradicts the optimality of budget in (G, B^*). ■

Theorem 4: In any feasible β-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. The pseudo-code of budget re-assignment to produce optimal integer solution is shown in Figure 10.

Each parent-child set construction takes $O(|E|)$, 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 equivalent integer solution in polynomial time $(O(|V|^2)$ with same objective value.

V. Extension of Integer Delay Budgeting Problem

We proved that the maximum integer delay budgeting problem as formulated in Section II, is polynomially solvable. In this formulation, there are two major simplifications. First, the objective function is simply a summation of the delay budgets on the nodes. This means that the objective is independent on the type of the operation on each node. Depending on the type and complexity of the operation at each node, the extra budget can have a different impact. We extend the problem to maximization of weighted summation of delay budgets assigned to the nodes. We assume that based on the complexity and type of operation, a non-negative weight is given for each node. This value determines the rate of relaxation on the structure of the component and/or synthesis effort on the component for each extra delay budget assigned to the node. The second important simplification in the original problem is that the delay budget assigned to a node can be unlimited. However, in reality, the delay budget can be exploited within a certain range and beyond that range, it is more beneficial to assign the remaining budget to other nodes in the graph. Both extensions are still integer linear programming problems. The formulation of the extended integer delay budgeting problem is:

$$Max \quad \sum_{v_i \in V} w_i b_i \tag{8}$$

$$a_j \geq a_i + b_j + d_j \quad \forall e_{ij} \in E \tag{9}$$

$$a_i \leq T \quad \forall v_i \in PO \tag{10}$$

$$a_i = 0 \quad \forall v_i \in PI \tag{11}$$

$$b_i \leq u_i \quad \forall v_i \in V \tag{12}$$

$$b_i \in Z \tag{13}$$

The following propositions prove that in both aforementioned extensions of integer delay budgeting problem, the optimal integer solution can be obtained using our algorithm.

Lemma 18: For each parent-child set (S_p, S_c) in fractional Adjacency graph (G_f) corresponding to (G, B^*), the condition $\sum_{v_i \in S_p} w_i = \sum_{v_j \in S_c} w_j$ is satisfied.

Proof: Assume that the condition does not hold in (G, B^*). Assume $\sum_{v_j \in S_c} w_j > \sum_{v_i \in S_p} w_i$. After β-budget re-assignment of minimum budget, say f_{min}, the total budget changes by $f_{min} \cdot \sum_{v_j \in S_c} w_j - f_{min} \cdot \sum_{v_i \in S_p} w_i$. Hence, total value of objective increases and this contradicts the optimality of budget in (G, B^*). ■

Based on this lemma, applying budget re-assignment iteratively on G_f does not change the value of objective. Hence, the integer solution is optimal.

Lemma 19: In budget re-assignment algorithm on G_f, the budget of each node never exceeds the corresponding upper bound on the budget at each node.

Proof: The order based on which the parent-child set is constructed and budget re-assignment is applied, depends on the fractional value of arrival time at the nodes. Due to this ordering, when budget of a node is increased by β in the child set, it is guaranteed that the total budget at the node cannot exceed the upper bound on this node. When budget re-assignment is applied on a parent-child set, the fractional value of arrival time at each child node is either zero or greater than the fractional value of arrival time at parent set (α_p). After budget re-assignment of β, the summation of fractional value of the budget at each node and the increase in budget (β) is at most 1. Hence, the total budget at each child node never exceeds its upper bound. ■

The lower bound on the budget can be added to the original delay of each node and if the initial solution remains feasible under a given timing constraint, we apply the integer delay budgeting algorithm to assign the extra delay budget to the nodes. In this paper, we formulated the delay budgeting

problem as an integer linear programming problem. The main assumption in this problem and its aforementioned extensions is that the objective value increases linearly with any unit of delay budget assigned to each node. However, in several applications, the gain is obtained only if the budget is a certain value. We refer to this problem as *discrete budgeting* problem that can be formulated as a mixed $0-1$ integer linear programming problem as follows:

$$Max \quad \sum_{v_i \in V} w_i x_i \tag{14}$$

$$a_j \geq a_i + b_j x_j + d_j \quad \forall e_{ij} \in E \tag{15}$$

$$a_i \leq T \quad \forall v_i \in PO \tag{16}$$

$$a_i = 0 \quad \forall v_i \in PI \tag{17}$$

$$x_i \in \{0, 1\} \tag{18}$$

In [22], it is proved that this problem is an NP-hard problem and an approximation algorithm on a rooted tree has been proposed. This is out of the scope of this paper and our experiments in this paper are only based on regular integer delay budgeting. In the next section, we show that the application on which our technique is applied, the regular unit change in the budget has almost linear correlation with the objective function. Hence, delay budgeting as formulated and solved in this paper can be applied to solve the delay budget assignment.

VI. Application

Delay budgeting algorithm is very generic and can be applied in different design tasks at different stages of CAD flow such as gate sizing in logic synthesis, timing optimization in placement, and library mapping in datapath level. In this section, we apply integer 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. The results show that early management of timing budget on IPs can lead to a faster compilation in physical implementation level.

A. Delay Budgeting and Core-based Compilation Flow

IP components are pre-designed and pre-verified blocks realizing a particular functionality. Designers cannot spend a lot of time regenerating most of standard function for future designs. Designers try to leverage the existing designs of components and use it in the current/future development of new applications. Mapping sub-designs to already existing pre-optimized and synthesized components is an unavoidable scheme in design automation flow of today?s complex designs. Especially in programmable systems such as FPGAs, design and market of soft IPs are growing rapidly, hence providing a rich library of various functional components. In a programmable system, realizations of IPs are basically the predefined program bits for a subset of chip that corresponds to the functionality of the IPs. Since there is no fabrication cost, IPs are more cost-effective to be generated. There is lots of effort and research both in academia and industry to come up with customized functional units for FPGAs. The CoreGen [23] tool provided by Xilinx [24] is a library of parameterizable functional cores for Xilinx FPGA devices. Also, due to highly constrained and finely grained architecture of FPGAs, efficient implementation of functional units are more challenging and complicated compared to ASIC IP designs. The new generations of FPGAs are getting more and more irregular. There are special architectural features integrated into the device such as carry chain, etc. The synthesis tool cannot exploit such features efficiently at the gate level logic optimization.

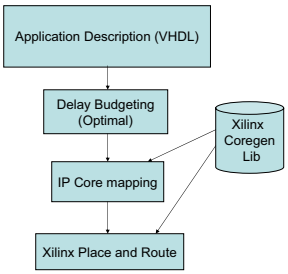


Fig. 11. Mapping an application on a FPGA device using library cores.

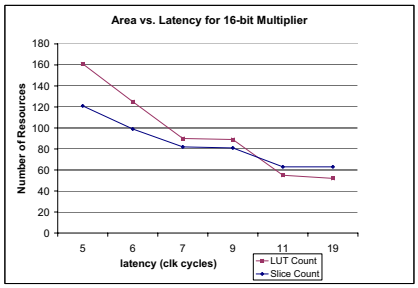


Fig. 12. Area vs. latency for a 16-bit Xilinx CoreGen multiplier.

In Figure 11, CAD flow of IP-based (or *core-based*) mapping of 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. 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 12 demonstrates a trade-off between the latency and the area of a CoreGen 16-bit multiplier mapped on FPGA VirtexE, Xilinx. *Slices* are the logic blocks in VirtexE FPGA series which consist of registers, LUTs (lookup tables), and other specific features.

Trade-off between delay and area is one of the most common relation observed in many library cores. Area and size of a design is an additive design metrics. Area of a

design is roughly computed by summation over the size of its components. However, during synthesis and optimization, if the design goes under a lot of optimizations across the boundary between the components, the area of the whole design cannot be estimated as the summation over the size of individual components. Such boundary merging often occurs during logic-level synthesis. In datapath level, design is defined as a control flow graph. A data flow graph is a directed acyclic graph. Latency at the output is defined as the delay of the longest path in the graph. Each basic element of a design is an operation which can take multiple cycles to execute. Hence they are mapped to registered cores in the library. Since there are registers between the cores, not much boundary merging and logic-like optimization can be applied. Hence, we can roughly estimate the area of the design as the summation over the size of individual cores the operations map to. Therefore, the area can be defined as a linear function of the size of each component.

Due to several cores each operation can be mapped to and dependency between the operations in a data flow graph under the given latency (as a timing constraint), it is not efficient and almost impractical to manually (arbitrarily or ad hoc method) choose the cores from the library. Instead, we need to develop a systematic and algorithmic technique for library mapping. That is where our delay budgeting can be a good guidance in library mapping for a set of DSP applications in mapping onto programmable devices.

B. Experimental Setup

We start from a DAG representation of an application. Each node corresponds to a computation in data path. Benchmark in our experiments is a set of some standard DSP benchmarks. The types of the computations are multiplier, adder, subtracter, division, and shifter. We assume all the datapaths are 16-bit wide. As shown in Figures 12, there is almost a linear relation between latency of the cores in CoreGen and their corresponding sized in terms of the number of CLBs and LUTs. Hence, based on the delay assigned to each component in the dataflow graph, it can be mapped to the core which gives a smaller size.

Delay of each computation is defined by a delay budgeting algorithm under the given latency at the output. 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. After library mapping and synthesis, the whole circuit is placed and routed on a FPGA device. We used *ISE 4.1* place_and_route tool provided by $Xilinx^{TM}$. We conducted two sets of experiments. Once we applied no delay budgeting algorithm and we mapped the components to the best latency cores. In the second set, we applied delay budgeting algorithms once our optimal delay budgeting and once a heuristic budgeting (ZSA like) to distribute the delay budget in the graph.

C. Experimental Results

The original latency and other characteristics of the benchmarks are given in Table I. The latency is the minimum latency of the data flow graphs with the fastest core in the library plus one more clock cycle in order to have sufficient delay budget in the applications. The benchmarks are the typical benchmarks used in high level synthesis experiments and research.

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 I
BENCHMARK INFORMATION AND CORE-BASED IMPLEMENTATION RESULTS.

Table II compares the implemented designs in terms of area when different delay assignment is used before library mapping. In this table, the first set of results correspond to original designs with no delay budgeting applied to them. Hence, each operation is mapped to the core in the library with the best delay. Gate count is one of the area metric reported by Xilinx mapping tool which corresponds to equivalent gate area. Gate count reflect the logic area of the design. On the other hand, the number of slices and number of LUTs are the other area metrics which are real physical area of design on FPGA devices. Due to optimization techniques during mapping and merging the sub-designs, the design can get more compact. Hence, we look at both metrics for area to understand the correlation between the delay budgeting and area. In this table, it is observed that in all the benchmarks, the area resulted from optimal delay budgeting algorithm is smaller than the area resulted from ZSA algorithm. Also, comparing the results when no budgeting is applied, we observe that delay budgeting is a useful technique to reduce the redundant complexity in the designs. In this table, the amount of total delay budget inserted into the graphs are also reported. The results show that solution by ZSA can be far away from optimal solution in some of the benchmarks such as *FDCT*. However, in some cases such as *DIFFEQ*, the solution is close to optimal.

In Table III, we compare the implemented designs in terms of other design metrics. The physical size of an implemented design on FPGA device is defined based on the number of LUTs and slices. We can observe similar behavior in terms of number of LUTs and slices as the gate count for area was reported in Table II in some of the benchmarks. However, in *ARF*, the number of slices does not vary much when ZSA is replaced by optimal algorithm for budgeting. In all benchmarks, the number of slices decreases when delay budgeting is applied and also optimal delay budgeting can minimize the number of slices further than ZSA algorithm. Another two design metrics evaluated in this experiments are the total compilation runtime from design entry until the end of place_and_route and maximum clock frequency reported at the end of design flow. Timing is analyzed after place_and_route. It is interesting to see that compilation flow can speed up due to decrease in the complexity of the computations in the application with delay budgeting. In small applications, this speed up is not very significant. In benchmarks *ARF* and *DCT*, the speed up is quite significant (up to two times). On average,

Benchmark	Runtime Area	Budgeting		
		No-Budget	ZSA	Opt
Diffeq	area(gates)	19,526	18,612	17,698
	Budget	-	2	3
ARF	area(gates)	47,028	43,596	43,380
	Budget	-	32	38
FDCT	area(gates)	49,722	46,380	43,860
	Budget	-	14	20
EWF	area(gates)	28794	27,990	26,502
	Budget	-	2	6
DCT	area(gates)	32,425	27,991	27,031
	Budget	-	24	27
Average	area(gates)	35499	32913.8	31694.2
	Budget	-	14.8	18.8

TABLE II

AREA (GATE COUNTS) AND TOTAL DELAY BUDGET ASSIGNED TO EACH BENCHMARK.

the optimal algorithm outperforms ZSA by 15% and 4% in terms of total delay budget and gate count, respectively. On average, applying proposed optimal delay budgeting during library mapping can reduce the gate count by 12% compared to the case where no budgeting is applied.

Maximum clock frequency is another design metric reflecting the timing characteristic of the implemented design. Note that the latency of each design in all the three sets of experiments is the same in terms of the number of clock cycles. Hence, faster the clock, in shorter time the results is available at the output. By reducing the complexity of the design, the optimization during place_and_route can get more relaxed and better performance can be obtained. However, if timing is affected by the most critical components in design significantly, the delay budget on non-critical paths may not be helpful for timing optimization. Although the computational components with longer latency may operate with faster clock, they require more number of registers and during place_and_route, register allocation can lead to reduction in the clock frequency. Timing analysis depends on many other factors. However, in benchmark *DCT*, the clock frequency increases by larger delay budgeting algorithm. The topology and connectivity in the applications affect the distribution of the delay budget in the graph. If most of the paths in the graph are critical paths, there is not much timing slack in the graph in order to be able to compare different delay budget distribution and its effect on component selection and library mapping. On average, the clock frequency with no delay budgeting is 77.35 MHz. After applying optimal delay budgeting, the average clock frequency decreases to 75.84 MHz. However, In benchmarks *DCT* and *FDCT* the clock frequency increases. By applying ZSA delay budgeting technique, the resulting clock frequency is 77.65 MHz which is close to original implementation. Applying optimal budget management, the compilation runtime is 23 seconds on average, while it is 26.8 seconds with ZSA delay budgeting and 31.8 seconds when no delay budget management is applied. The runtime of delay budget management added to the compilation flow is very negligible compared to runtime of the place_and_route stage. Applying optimal delay budget management, area of the implemented design in terms of the number of LUTs and slices improves by 16% and 18%, respectively, compared to when no delay budgeting is applied. Based on the reported results, optimal delay budgeting algorithm always decrease the size of design in any of the three design metrics on average.

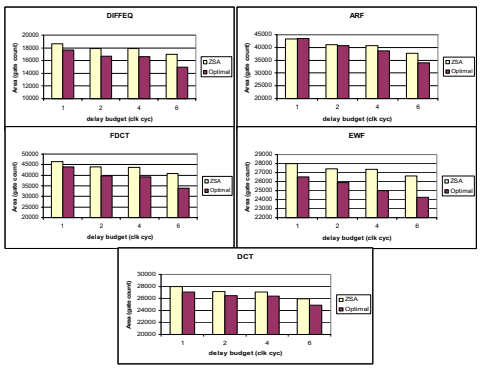


Fig. 13. Area (gate count) vs. delay budget with heuristic delay budgeting (ZSA) and optimal delay budgeting.

In the second set of experiments, we assume that the timing constraint at the output of each application is the original latency reported in Table I plus the excess latency (ΔT) applied to the design. Therefore, depending on ΔT, more timing slack is injected to the graph before applying different delay budgeting algorithms. Figure 13 demonstrate the size of implemented designs in terms of the number of gates in both cases when ZSA and optimal delay budgeting algorithms are used for delay assignment to the computations in the applications. Axis x is the increase in the latency of the output ΔT. It is shown as *delay budget* in the figure. y-axis is the area of the design after synthesis in terms of equivalent gate count. In all the benchmarks, with increasing value of ΔT the optimal algorithm outperforms ZSA more significantly. However, the trade-off is in the latency of the output (increased by Δ_T). If ΔT is a very large value, there may not exist cores in the library with the large delay budget assigned to the components in the design. Hence, the area and other design metric cannot improve in parallel with increase in total budget.

Figure 14 shows the change in clock frequency when

Benchmark	Runtime Area	Budgeting		
		No-Budget	ZSA	Opt
Diffeq	area(slices)	780	740	700
	area (LUTs)	1030	958	886
	runtime(sec)	13	12	11
	clk frequency(MHz)	81.2	78.42	70
ARF	area(slices)	1982	1806	1803
	area (LUTs)	2476	2188	2224
	runtime(sec)	43	40	29
	clk frequency(MHz)	72.58	77.45	70
FDCT	area(slices)	2044	1867	1734
	area(LUTs)	2572	2335	2155
	runtime(sec)	43	42	39
	clk frequency(MHz)	75.13	75.36	80
EWF	area(slices)	1138	1094	1016
	area(LUTs)	1472	1418	1291
	runtime(sec)	22	20	18
	clk frequency(MHz)	81	80	80.2
DCT	area(slices)	1338	1091	1032
	area(LUTs)	1618	1300	1221
	runtime(sec)	38	20	18
	clk frequency(MHz)	76.86	77	79
Average	area(slices)	1456.4	1319.6	1257
	area(LUTs)	1833.6	1639.8	1555.4
	runtime(sec)	31.8	26.8	23
	clk frequency(MHz)	77.35	77.65	75.84

TABLE III

AREA (NUMBER OF SLICES AND LUTS) AND COMPILATION RUNTIME AND CLOCK FREQUENCY FOR WHEN DIFFERENT DELAY BUDGETING IS APPLIED.

ΔT increases and compares the clock frequency in both cases when ZSA and optimal delay budgeting algorithms are applied. As described before, clock frequency is not an additive function of components as area and size of design is. Hence, increasing the delay budget can have different impact on the clock frequency. For example, for the benchmark *DCT* and *FDCT* the clock frequency resulted by optimal delay budgeting is greater the the clock frequency resulted from ZSA delay budget management. However, in benchmark *EWF* the clock frequency resulted from optimal budgeting (82 MHz) is slightly greater than the one resulted by ZSA budgeting (80 MHz). In large ΔT, the difference of clock frequency resulted from ZSA and optimal delay budgeting algorithm gets larger and larger. Hence, the impact of optimal delay budgeting is more visible.

Table IV shows the area in terms of number of LUTs and slices in FPGAs and compilation runtime for different excess delay (ΔT) of 2, 4, and 6 clk cycles. The larger ΔT, the more delay budget is distributed. Although budget increases significantly by ΔT, the improvement in area is not as significant as budget. 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. 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. Although speedup in PAR runtime were not significant in smaller applications, due to lesser complexity and smaller structure, the effect on runtime for place and route for larger applications.

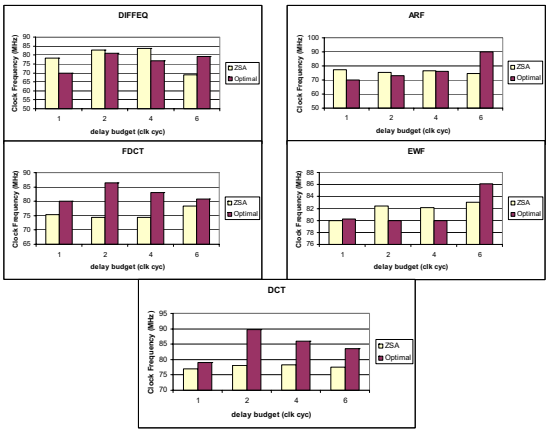


Fig. 14. Clock frequency (MHz) vs. delay budget with heuristic delay budgeting (ZSA) and optimal delay budgeting.

As a result, delay budgeting gives the flexibility of mapping the applications to components in the target library with simpler structure and smaller area. Developing complete libraries facilitates the design CAD tool to exploit the existing delay budget to improve design quality.

VII. CONCLUSION

General delay budgeting can be solved using linear programming solver. However, due to numerical instability and

Benchmark	Runtime Area	ΔT=2 clk cycle		ΔT=4 clk cycle		ΔT=6 clk cycle	
		ZSA	Opt	ZSA	Opt	ZSA	Opt
Diffeq	area(slices)	708	652	708	652	672	582
	area(LUTs)	888	781	886	778	818	641
	runtime(sec)	11	9	14	9	11	7
	Budget	4	6	8	12	12	18
ARF	area(slices)	1670	1665	1662	1553	1518	1290
	area(LUTs)	1982	1908	1900	1833	1739	1381
	runtime(sec)	43	26	42	25	39	20
	Budget	36	48	44	68	52	88
FDCT	area(slices)	1728	1491	1718	1474	1574	1222
	area(LUTs)	2106	1752	2076	1698	1877	1349
	runtime(sec)	38	36	36	36	43	21
	Budget	22	34	38	62	54	90
EWF	area(slices)	1058	982	1054	942	1018	906
	area(LUTs)	1366	1232	1360	1164	1314	1114
	runtime(sec)	19	17	20	17	19	15
	Budget	4	10	8	18	12	26
DCT	area(slices)	1038	996	1031	990	977	918
	area(LUTs)	1222	1169	1213	1161	1144	1065
	runtime(sec)	20	15	19	14	18	13
	Budget	30	34	42	48	54	62
Average	area(slices)	1240	1157.2	1234.6	1122.2	1151.8	983.6
	area(LUTs)	1512.8	1368.4	1487	1326.8	1378.4	1110
	runtime(sec)	26.2	20.6	26.2	20.2	26	15.2
	Budget	19.2	26.4	28	41.6	36.8	56.8

TABLE IV

AREA (#SLICES-#LUTS), TOTAL BUDGET, AND RUNTIME VS. DELAY BUDGET (CLK CYC).

discrete behavior of libraries of components, integer solution is required. In this paper, using optimal solution to LP relaxation of budgeting problem, we transform the solution to optimal integer solution. For this purpose, we introduce budget re-assignment in a directed acyclic graph. 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. In this paper, we describe the detailed analysis of our delay budgeting algorithm. In addition, we look at different extensions of the delay budgeting problem, such as maximization of weighted summation of delay budgets assigned to the nodes and additional constraints on lower bound and upper bound on the delay budget allocated to each node. We prove that in both aforementioned extensions, our algorithm can produce an optimal integer solution in polynomial time.

We applied our budgeting technique in mapping of applications on FPGA device. We applied timing budgeting algorithm in selecting the components of library and mapping to different components of the application such that the design complexity is reduced without violation of timing constraints. Using IP library of different computations, delay budget is exploited to improve the area and hence, to speedup the runtime of place-and-route. Our experimental results show the effectiveness of budgeting on IP-based application mapping. Our optimal algorithm outperforms ZSA algorithm [4] in terms of area and compilation runtime significantly.

Our polynomial algorithm is applied to a general optimal LP solution. Developing a polynomial time graph-based algorithm for integer delay budgeting is the current problem we are working on. Discrete budgeting is another challenging problem that needs to be studied and intuitive heuristic algorithms need to be developed for variations of this problem. Other future directions are delay budgeting problem in pipelined datapaths and resource-shared datapaths in IP-based design implementation.

REFERENCES

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

[2] L. A. Wolsey. *Integer Programming*. New York, NY, Wiley-Interscience Publisher, John Willey & Sons Inc., pp. 37-52, 1998.

[3] C. Chen, E. Bozorgzadeh, A. Srivastava, and Majid Sarrafzadeh. "Budget Management with Applications". In *Algorithmica*, vol 34, No. 3, pp. 261-275, July 2002.

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

[5] . H. Youssef, E. Shragowitz. "Timing Constraints for Correct Performance",
In the proceedings of *ACM/IEEE International Conference on Computer-Aided Design*, 1990.

[6] 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.

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

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

[9] C. Kuo and A. C.-H Wu. "Delay Budgeting for a Timing-Closure-Design Method", In the proceedings of *International Conference on Computer-Aided Design*, pp. 202 207, 2000.
[10] C. Chen, X. Yang, M. Sarrafzadeh. "Potential Slack: An Effective Metric of Combinational Circuit Performance. In the proceedings of *ACM/IEEE International Conference on Computer-Aided Design*, pp. 198-201, 2000.
[11] Y. Liao and C. K. Wong. "An Algorithm to Compact a VLSI Symbolic Layout with Mixed Constraints". In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* , Vol. 2, No. 2, April. 1983.
[12] J. F. Lee and D. T. Tang. "VLSI layout compaction with grid and mixed constraints". In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 6, No. 5, Sep. 1987.
[13] 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 the proceedings of *Conference on European Design Automation*, November 1992.
[14] W. L. Schiele. "Improved Compaction by Minimized Length of Wires", In the proceedings of *20th ACM/IEEE Design Automation Conference*, pp. 121-127, 1983.
[15] R. Okuda, T. Sato, H. Onodera, K. Tamaru. "An efficient Algorithm for Layout Compaction Problem with Symmetry Constraints", In the proceedings of *International Conference on Computer-Aided Design*, pp. 148-153, November 1989.
[16] S. L. Lin and J. Allen. "MinPlex- A Compactor that Minimizes the Rounding Rectangle and Individual Rectangles in a Layout". In the proceedings of *ACM/IEEE Design Automation Conference*, pp. 123-130, 1986.
[17] P. Girard, C. Landrault, S. Pravossoudovitch, and D. Severac. " A Gate Resizing Technique for High Reduction in Power Consumption. In the proceedings of *International Symposium on Low Power Electronics and Design*, pp. 281 286, 1997.
[18] H. R. Lin and T. Hwang. "Power Reduction by Gate Sizing with Path-Oriented Slack Calculation". In the proceedings of '*IEEE ASP-DAC*, pp. 7 12, 1995.
[19] 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 the proceedings of *IEEE International Symposium on Microarchitecture*, 2001.
[20] J. Luo and N. Jha. "Battery-Aware Static Scheduling for Distributed Real-Time Embedded Systems". In the proceedings of *IEEE/ACM Design Automation Conference*, 2001.
[21] S. Bakshi and D. Gajski. "Component Selection for High-Performance Pipelines". In *IEEE Transactions on Very Large Scale Integrated Systems*, pp. 181-194, Vol. 4, No. 2, 1996.
[22] S. Ghiasi, K. Nguyen, E. Bozorgzadeh, and M. Sarrafzadeh. "On Computation and Resource Management in an FPGA-based Computing Environment". a poster presentation in *ACM International Symposium on Field-Programmable Gate Arrays (FPGA)*, February 2003.
[23] Xilinx IP Center. http://www.xilinx.com/products/design_resources/.
[24] Xilinx Inc. http://www.xilinx.com

Elaheh Bozorgzadeh received the B.S. degree in Electrical Engineering from Sharif University of Technology, Iran in 1998, M.S. degree in Computer Engineering from Northwestern University in 2000, and Ph.D. degree in Computer Science from the University of California, Los Angeles, in 2003.

She is currently as assistant professor in the Department of Computer Science at the University of California, Irvine. Her research interest includes VLSI CAD, design automation for embedded systems, and reconfigurable computing. She is a member of ACM and IEEE.

Soheil Ghiasi received his B.S. from Sharif University of Technology, Tehran, Iran in 1998, and his M.S. from the University of California, Los Angeles, in 2002. Currently, he is a PhD candidate in the Department of Computer Science at UCLA. His research interests include different aspects of embedded and reconfigurable system design.

Atsushi Takahashi received his B.E., M.E., and D.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1989, 1991, and 1996, respectively. He had been with the Tokyo Institute of Technology as a research associate from 1991 to 1997 and has been an associate professor since 1997. He visited University of California, Los Angeles, U.S.A., as a visiting scholar from 2001 to 2002. He is currently with Department of Communications and Integrated Systems, Graduate School of Science and Engineering, Tokyo Institute of Technology. His research interests are in VLSI layout design and combinational algorithms. He is a member of IEEE, IEICE, and IPSJ.

Majid Sarrafzadeh (M'87, SM'92,F'96) received his B.S., M.S. and Ph.D. in 1982, 1984, and 1987 respectively from the University of Illinois at Urbana-Champaign in Electrical and Computer Engineering. He joined Northwestern University as an Assistant Professor in 1987. In 2000, he joined the Computer Science Department at University of California at Los Angeles (UCLA). His recent research interests lie in the area of Embedded and Reconfigurable Computing, VLSI CAD, and design and analysis of algorithms. Dr. Sarrafzadeh is a Fellow of IEEE for his contribution to "Theory and Practice of VLSI Design". He received an NSF Engineering Initiation award, two distinguished paper awards in ICCAD, and the best paper award in DAC. 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.

Professor Sarrafzadeh has published approximately 250 papers, is a co-editor of the book "Algorithmic Aspects of VLSI Layout" (1994 by World Scientific), and co-author of the books "An Introduction to VLSI Physical Design" (1996 by McGraw Hill) and "Modern Placement Techniques" (2003, Kluwer) Dr. Sarrafzadeh is on the editorial board of the VLSI Design Journal, an Associate Editor of ACM Transaction on Design Automation (TODAES) and an Associate Editor of IEEE Transactions on Computer-Aided Design (TCAD).