Probabilistic Delay Budgeting for Soft Realtime Applications

Soheil Ghiasi Po-Kuan Huang

Department of Electrical and Computer Engineering
University of California, Davis, CA 95616
soheil, pohuang@ece.ucdavis.edu

Abstract

Unlike their hard realtime counterparts, soft realtime applications are only expected to guarantee their "expected delay" over input data space. This paradigm shift calls for customized statistical design techniques to replace the conventional pessimistic worst case analysis methodologies. Statistical design methods can provide a realistic assessment of design space, and improve the design quality by exploiting its stochastic behavior. We present a novel probabilistic time budgeting algorithm that translates the application expected delay constraint into its components delay constraints. Our algorithm which is based on mathematical properties of the problem, determines the optimal maximum weighted timing relaxation of an application under expected delay constraint. Experimental results on core-based synthesis of several multi-media applications on FPGAs show about 20% *and* 19% *average energy and area improvement, respectively.*

1. Introduction

Timing is traditionally treated as a hard constraint throughout the system design process. As a result, applications are often pessimistically analyzed for worst case scenarios and their slowest responses determine their performance. Although this is a necessity for hard realtime applications, their soft realtime counterparts can occasionally take longer than the deadline to finish some tasks [1]. They are often expected to guarantee an *expected delay* (or latency) rather than a worst case execution time over the input data space. Therefore, realistic statistical design techniques, as opposed to pessimistic worst case analysis, are required to automate the design process for such application domains.

In this paper, we present a probabilistic delay budgeting technique for soft realtime applications. Our technique, which is based on mathematical properties of our model, relaxes the timing constraints of different components of a design, while guaranteeing that the expected delay of the application does not violate a given constraint. We develop and employ an optimal incremental delay relaxation algorithm for each component of the design. The incremental technique is integrated into a higher level probabilistic algorithm that performs time budgeting for the entire design. Note that the output design might (and most probably will) have a larger delay than the constraint for infrequent input patterns. However, it is guaranteed that the expected delay over the entire input data space meets the given constraint.

We apply our algorithm to core-based synthesis of application CDFGs. The components with relaxed timing constraint are optimized to improve design energy and area. Experimental results on several multimedia applications show an average of about 20% and 19% energy and area improvement over not using our technique, respectively.

2. Background

2.1 Application and Execution Model

Figure 1 illustrates the application model, and its corresponding hardware realization that is used throughout this paper. We use the well-known control data flow graph (CDFG) structure to model a given application or its compute-intensive kernels, where nodes represent application basic blocks (tasks), and outgoing edges of a node denote control flow edges. Nodes of the graph take a given amount of time to finish their execution (called delay or latency). Edges of the graph model the dependency among tasks and are assumed to have zero delay. Note that our model can capture edge delays by inserting a dummy node on the edge whose delay is equal to the original edge delay. Therefore, our model is not restricted to ignore communication latency.

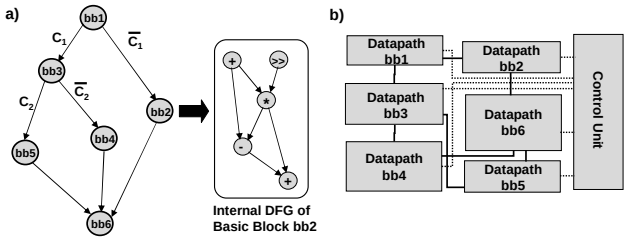


Fig. 1: Our synthesis scheme: a)An example CDFG: each node is a basic block containing internal data flow graph (DFG) b) An illustration of generated hardware.

Each basic block has an internal data flow graph (DFG), which is composed of a number of basic operations. The compute model *within* a node (DFG) differs from the CDFG in that all of the DFG paths are activated at runtime, and therefore, all of the paths have to meet the timing constraint of the basic block (Figure 1). The synthesized hardware is a pre-optimization design that has a separate datapath for each basic block. When the execution of a basic block is finished, depending on the computation result, it invokes another dependent basic block. Hence, depending on the input data, one and exactly one path in the graph is executed at runtime. Edges of the graph are annotated with the probability of taking that edge if its source node is executed. For example in Figure 1, *bb*1 upon completion might invoke *bb*2 or *bb*3 with probability $\overline{C_1}$ or C_1, respectively. The probabilities are profiled over the input data space and are assumed to be known a priori.

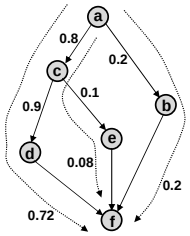
We assume that there is no backward edge in the CDFG. Loops and other type of backward edges are removed from the CDFG by preprocessing, which is performed by either putting loops into nodes and forming complex nodes, or completely unrolling the loops and removing them from the application. If neither complex node formation nor complete unrolling are possible, we focus on the body of the loop, which constitutes the compute-intensive kernel of the application.

2.2 Motivating Example

Figure 2 illustrates an example application, its execution traces, and several examples of delay budget assignment under expected delay constraint. In this example, task (basic block) *c* upon completion might invoke task *d* or *e* with probability 0.9 or 0.1, respectively. The dashed paths demonstrate the possible execution traces of the application. Each trace is annotated with the probability of its activation over the input data space. Note that the number of execution traces can grow exponentially with the number of edges in the graph.

To motivate the idea of time budgeting under *expected delay* constraint, let us assume that the delay of the tasks are as shown in the "original delay" column of the table, and the application's expected delay constraint is 5. Originally, the expected delay of the application is 4.52 over the input data space. We can slow down some tasks to optimize

[1] The argument pertains to system/application level design as opposed to gate/layout level, where critical paths determine the clock period



node	visiting prob.	original delay	budgeting #1	budgeting #2	budgeting #3
a	1	1	1	1	1
b	0.2	1	1+2	1	1+1
c	0.8	1	1	1	1
d	0.72	2	2	2	2
e	0.08	1	1	1+6	1+3
f	1	1	1	1	1
exp. delay		0.72*5+0.08 *4+0.2*3 =4.52	0.72*5+0.08* 4+0.2*5 =4.92	0.72*5+0.08* 10+0.2*3 =5.00	0.72*5+0.08* 7+0.2*4 =4.96

Fig. 2: Probabilistic timing budget assignment under expected delay constraint

them further, and improve the quality of the synthesized design. The quality metric (power dissipation, area, accuracy, cost, etc) greatly depends on the application domain. The table illustrates three delay budget assignment examples that would all meet the expected delay constraint.

This paper provides detailed formulation, property analysis, and efficient algorithms to address the delay budget management under expected delay constraint. We develop and utilize an optimal incremental delay budgeting algorithm for each node (basic block), which is used to maximize the utility function for the entire application. Note that after delay budget assignment, infrequent execution traces might take longer than the expected delay. This makes the problem drastically different from conventional pessimistic analysis, where all paths must meet the timing constraint.

3. Related Work

The concepts of slack and timing relaxation have been extensively studied in the synthesis community. Time budgeting on directed acyclic graph, while different in principle, relates to the conventional slack idea [7]. However, our problem at hand is different due to the soft realtime constraints, and the probabilistic nature of the required analysis. The budgeting problem on a graph (both temporal and spatial budgeting) has been studied for many different applications. Timing-driven placement and floor planning is one such example, during which the issue of delay budgeting has been addressed by several researchers [1, 3, 12, 14, 5]. Moreover, delay budgeting has been utilized to perform gate and wire sizing for power optimization. Under a given timing constraint, budget management can be applied to find a set of nodes/edges in the netlist graph such that their physical size or power dissipation can be reduced by mapping to smaller, or power-efficient cell instances with larger delays from a target library [3, 9, 16]. High-level synthesis is another application, in which timing slack of the operations is utilized for optimization in area and power. Examples are the algorithms and techniques developed for area minimization in pipelined datapath [18] and power minimization under timing constraint [11, 22].

The techniques employed in above papers are sub optimal heuristics driven from Zero Slack Algorithm [17] and MISA [2]. In our previous result [19], we solved various non-probabilistic formulations of the problem through a unified theoretical framework. In this paper, we extend our previous work to incrementally solve the problem of delay budget assignment for each basic block of the CDFG. Furthermore, we develop a probabilistic analysis framework that considers the stochastic behavior of the application over the input data space.

4. Target Application Domain

We focus on the class of soft realtime applications that perform intensive computations and demand hardware realizations for realtime performance. Examples include multimedia applications such as video encoding/decoding, and image compression. Such applications are often characterized by their *intensive, periodic, heavily input data dependent (content dependent), and loss tolerant* behavior.

While intensity and periodicity impose timing constraints on the designs that target such applications, content-sensitivity and loss-tolerance allow occasional violations of the timing constraint for infrequent input data. For example, a video decoder can occasionally skip a frame without affecting the user experience. For such application domains, guaranteeing an expected delay for each period of execution is as good as maintaining a hard timing constraint. In practice, the former is often preferred due to the potentials of further optimizations under softer timing constraints.

In this paper, we target the aforementioned class of applications and hence, assume that the application demands a guarantee on the expected delay. Furthermore, we assume that the input data, or a representative subset of it, is available at the design time. This is a reasonable assumption that allows us to profile the probability of each execution trace in the application CDFG.

5. Formalizations and Problem Statement

A given application can be represented as a directed acyclic graph $G = (V, E)$, where V is a set of vertexes and E is a set of directed edges. Each vertex $v \in V$ represents a task (basic block) of the application that takes d_v units of time to perform its computation. When v_i finishes its computation, it invokes exactly one of its successor (dependent) nodes depending on the computation results. Edge $e_{ij} \in E$ models the data dependency between v_i and v_j, denoting that v_j can start its computation only when v_i is finished and e_{ij} is selected to continue the execution trace of the application. There is a probability p_{ij} associated with each $e_{ij} \in E$ that corresponds to the probability of taking e_{ij} when node v_i finishes its computation, over the input data space. Similarly, for each node v_i, its probability p_i, is defined as the probability of the execution trace visiting node i. It follows that for every vertex $\sum_{j \in fanout(i)} p_{ij} = 1$.

A source node $s \in V$, is a vertex without incoming edges, and a sink node $t \in V$, is a vertex without outgoing edges. We assume that there is exactly one source and exactly one sink node in G. A CDFG with more than one source (sink) node can be transformed to comply with this constraint by adding a dummy source (sink) with zero delay, and connecting it to all application source nodes (all application sink nodes to it). An execution trace (or simply a trace) of the application is a directed path from s to t in G. Let XT be the set of execution traces of G. Note that $|XT|$ can grow exponentially with respect to $|E|$. A trace can be represented by the edges that appear in the path from s to t. We use the notion $e_{si} \rightarrow e_{ij} \rightarrow \ldots \rightarrow e_{ut}$ to represent those edges.

The probability of an execution trace $x = e_{si} \rightarrow e_{ij} \rightarrow \ldots \rightarrow e_{ut}$ or P_x is defined as the probability of the application taking x at runtime, over the input data space. Hence, if $x = e_{si} \rightarrow e_{ij} \rightarrow \ldots \rightarrow e_{ut}$ then $P_x = p_{si} p_{ij} \ldots p_{ut}$. The runtime or delay of trace x or D_x, is defined as the application's runtime when taking trace x and is equal to $D_x = d_s + d_i + d_j + \ldots + d_u + d_t$. By definition: $\sum_{x \in XT} P_x = 1$ and $d_{exp}(G) = \sum_{x \in XT} P_x D_x$. A partial execution trace refers to a subset of an execution trace that connects two nodes that lie on the trace. Let x_{ij} denote a partial execution trace that starts from node i and finishes at node j. We use XT_{ij} to refer to the set of all possible x_{ij} (all possible partial execution traces that start at i and finish at j). We define $P_{x_{ij}}$ and P_{ij} as the probability of the execution of a particular partial trace x_{ij}, or one of traces in XT_{ij}, respectively. By definition, $P_{st} = 1$. Note that we use the lower case variables d and p to refer to delay of a node, and probability of a/an node/edge, respectively. The upper case variables D and P denote the delay and probability of a (partial) trace. Considering the notion of partial execution trace, we have:

$$\forall v_i \in V \qquad p_i = \sum_{x_{si} \in XT_{si}} P_{x_{si}} \tag{1}$$

$$\forall v_i \in V \qquad d_{exp}(i) = \sum_{x_{si} \in XT_{si}} P_{x_{si}} . D_{x_{si}} \tag{2}$$

Any node i induces a subgraph on G that is composed of nodes and edges that can be visited by some x_{si}. We extend the notion of expected delay to such induced subgraphs. That is, $d_{exp}(G_i)$ is the expected delay of the subgraph, induced by treating node i as the sink node. Let us assume that each node $v_i \in V$ utilizes a unit of slow down in its delay with weight w_i. The problem of timing budget management for soft realtime applications can be formally stated as:

Given $G(V, E)$, vectors p, d, and w, and an expected delay constraint, D_{max}, the objective is to determine an integral delay relaxation budget b_i for each v_i that maximizes $\sum w_i . b_i$ (Note that the delay of node v_i after assigning the delay budget would be $d_i + b_i$) Subject to expected delay constraint for the application:

$$d_{exp}(G) = \sum_{\forall x \in XT} P_x . D'_x \leq D_{max}$$

Where D'_x is the updated delay of trace x after slowing down the nodes by vector b. In practice, the application, its timing constraint, and the library elements are the only available inputs. Hence, the remaining input parameters, i.e., vectors p, d and w, have to be automatically determined. Vector p is determined by profiling the application over input data space. In section 7 we prove properties by which, vectors d and w are determined.

6. Tractable Expected Delay Calculation

The expected delay of an application is $d_{exp}(G) = \sum_{x \in XT} P_x D_x$, by definition. However, the number of execution traces of an application can grow exponentially with respect to the problem size (number of nodes or edges in the graph). Therefore, the complexity of definition-based approach to calculating the expected delay can be prohibitive. In this section, we prove some interesting properties of the problem by which, we can rapidly relate the expected delay of the application to parameters that are easy to calculate from problem inputs. To do so, we break the execution traces over fanins of t. Figure 3 assists in visualizing the following equations:

$$d_{exp}(G) = \sum_{x \in XT} P_x D_x = \sum_{x_{si} \in XT_{si}} \sum_{e_{it}} [(P_{x_{si}}.p_{it})(D_{x_{si}} + d_t)]$$

$$= d_t . \sum_{x_{si} \in XT_{si}} \sum_{e_{it}} (P_{x_{si}}.p_{it}) + \sum_{x_{si} \in XT_{si}} \sum_{e_{it}} (P_{x_{si}}.p_{it}.D_{x_{si}})$$

$$= d_t.P_{st} + \sum_{e_{it}} \sum_{x_{si} \in XT_{si}} (P_{x_{si}}.p_{it}.D_{x_{si}}) = d_t.P_{st} + \sum_{e_{it}} p_{it} . \sum_{x_{si} \in XT_{si}} (P_{x_{si}}.D_{x_{si}})$$

$$= d_t.P_{st} + \sum_{e_{it}} p_{it}.d_{exp}(G_i) \quad (3)$$

Equation 3 presents a recursive method to calculate the expected delay of G based on the partial expected delay at fanins of t. We can reuse equation 3 for subgraphs induced by fanins of t, to plug in the expanded forms for $d_{exp}(G_i)$ and eliminate the recursion. A non-recursive solution is favorable due to its practicality and improved complexity, especially if it results in a simple fast method to calculate the expected delay.

When expanding equation 3, each edge is accounted for exactly once in reverse topological order. For each edge e_{ij}, the source node i, contributes to $d_{exp}(G)$ with the term $p_{ij}.d_{exp}(G_i)$. Interestingly, the addition of all such terms for fanouts of a node, is equal to $d_{exp}(G_i)$ (note that $\sum_{j \in fanout(i)} p_{ij} = 1$). We can reuse equation 3 to expand $d_{exp}(G_i)$, which would contribute to $d_{exp}(G)$ with the term $d_i.P_{si}$. However, P_{si} is the probability of the execution trace going through node i over the input data space, which is equivalent to p_i, by definition. Therefore:

Theorem 1. *The expected delay of an application under execution model explained in Section 5 can be calculated using the following equation in $O(E)$. Note that each edge has to be traversed once to determine node visiting probabilities.*

$$d_{exp}(G) = \sum_{v_i \in V} d_i.p_i$$

Detailed proof is omitted for brevity, however, an outline of the proof is sketched in the aforementioned arguments. Theorem 1 provides a very intuitive expression for the expected delay: The contribution of a node to the application's expected delay, is its intrinsic delay times the likelihood of that node being visited during application runtime (no matter what trace is taken as long as the node is executed). This can be easily verified on small examples and special cases such as paths, or symmetric graphs. In addition, it raises the point that infrequently visited nodes can be assigned large delay budgets with little effect on application's expected delay. This property is further analyzed in subsequent sections.

7. Timing Budget Management for Basic Blocks

The problem of delay budget management for CDFGs has two stages. First, the extra delay budget (timing constraint relaxation) has to be assigned to each node of the graph (basic block). Subsequently, it has to be distributed onto functional units inside the basic block to improve design utility. In this section, we discuss the problem of intra basic block delay budgeting, and its connection with the problem formulated in Section 5. We leverage the existing techniques for basic block level time budgeting, and present properties of the problem that allow us to quickly and efficiently determine the appropriate weights for each basic block (vector w in Section 5).

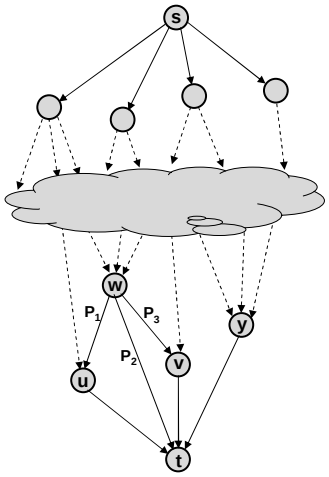


Fig. 3: Nodes are visited once per each outgoing edge during expansion of the recursive equation 3. Node w contributes $(p_1 + p_2 + p_3).d_{exp}(G_w) = d_{exp}(G_w)$ to the application's expected delay.

The execution model for the data flow graph (DFG) of each basic block is different from the CDFG model we presented for soft realtime applications (Figure 1). Specifically, all of the execution traces of a DFG are activated at runtime and therefore, all of them have to meet the timing constraint. Existing techniques offer polynomial-time algorithms to maximize the total delay budget assigned to operations of a DFG under delay constraints [19, 7]. The most efficient existing technique, converts the problem into a weighted edge budgeting instance, and injects delay units onto *selected* edges, until all of the edges become critical.

7.1 Incremental Time Budgeting for Basic Blocks

The timing constraint for each basic block cannot be less than its critical path. The weight assigned to each basic block (vector w in Section 5) determines its potential for utilizing additional units of relaxation in its timing constraint. Ideally, we would like the weight to be an easy-to-calculate function of DFG's structure. In this subsection, we use an existing optimal method for edge budgeting on DFGs to determine the weight vector, w, for basic blocks.

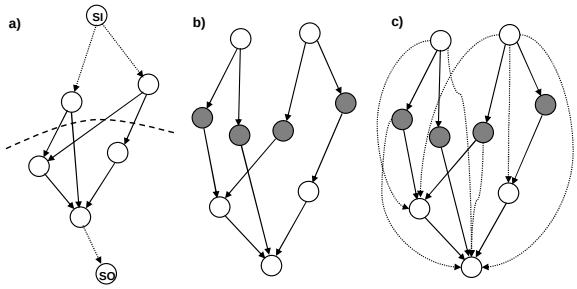


Fig. 4: a)A sample DFG and a sample cut are shown. b)The corresponding edge graph. The edges in the cut correspond to dark nodes. c) The transitive closure of the edge graph. The cut corresponds to the maximum independent set here.

Formally, a data flow graph $H(V,E)$ is a directed acyclic graph. Assume that two dummy nodes called super input (*SI*) and super output (*SO*) are connected to the primary inputs and primary outputs of H to make it single input/output. We can state the following [19]:

Definitions: A subset of edges of H, is called a *cut* if and only if every *SI* to *SO* path contains exactly one edge of the set (Figure 4).

Graph $H^*(V^*,E^*)$ is called the intersection graph (or edge graph) of $H(V,E)$, if there is a node $v^*_{ij} \in V^*$ for every $e_{ij} \in E$; and there is an edge e^*_{ijk} between v^*_{ij} and v^*_{jk}.

Lemma 2. *A cut in H corresponds to an* independent set *in transitive closure of H^* (H^{*t}). In transitive closure, if there is an edge from node v_x to v_y and from v_y to v_z, there is also an edge from v_x to v_z. We use H^t to represent the transitive closure of H. Similarly, a weighted cut in H corresponds to a* weighted independent set *of the transitive closure of H^* (H^{*t}). The cut with the maximum weight in H (weighted max cut) corresponds to the maximum weighted independent set (MWIS) in H^{*t}.*

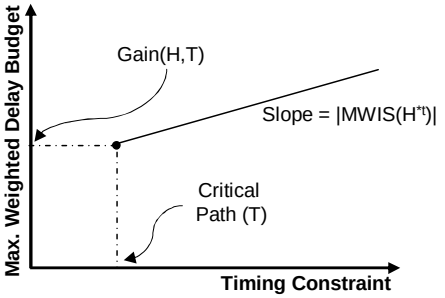


Fig. 5: Maximum total slow down of basic block operations

Note that although finding the maximum (weighted) independent set of an arbitrary graph is known to be NP-complete, it can be solved in polynomial time for transitive graphs [6].

Definitions: Let $Gain = OPT(H,T)$ denote the maximum amount of weighted delay budget that can be added to the edges of data flow graph H under timing constraint T. Let Graph H_b be the new data flow graph that is formed by adding the delay budgets to the edges of H. Hence, H_b has the same structure as H, however the delay of its edges are different.

There is a polynomial combinatorial algorithm for determining $Gain = OPT(H,T)$ and H_b [19], however we are not concerned with the algorithm details and treat it as a black box here.

Lemma 3. *For a given instance of the weighted edge budgeting problem with critical path equal to T:*
$Gain = OPT(H,T+\Delta_T) = OPT(H,T) + OPT(H_b,\Delta_T) = OPT(H,T) + \Delta_T \mid MWIS(H_b^{*t}) \mid = OPT(H,T) + \Delta_T \mid MWIS(H^{*t}) \mid$

Lemma 3 states that if the timing constraint T is increased by Δ_T, we do not need to recalculate the solution from scratch. It provides an optimal incremental method to extend the solution under timing constraint T, to another solution under timing constraint $T+\Delta_T$. To extend the solution, more specifically, the budget of the edges that form a weighted-max-cut (a cut with the maximum weight) will be increased by Δ_T. Such edges correspond to the maximum weighted independent set (MWIS) in the transitive intersection graph of the problem instance (Figure 4). Therefore, incremental calculation of the edge delay budgets for various values of timing constraint can be performed quite rapidly.

Caution has to be taken when applying delay budget to MWIS of a problem instance. Although MWIS can be used to augment an existing solution under timing constraint T to $T+\Delta_T$, it *cannot* correctly solve the instance for T. Previous work has discussed and investigated this issue [2].

Corollary 4. *The maximum total delay budget that can be added to the edges of a DFG H under timing constraint, is a linear function of the timing constraint. Namely:*
$OPT(H,T') = OPT(H,T) + (T'-T) \mid MWIS(H^{*t}) \mid \qquad T' \geq T$
In other words, the gain vs. timing constraint graph, is a line with slope of $\mid MWIS(H^{*t}) \mid$. Figure 5 visualizes the relation between $OPT(H,T')$ and T'.

Corollary 5. *For problem presented in Section 5, the minimum delay of each node, d_i, is equal to the critical path of the corresponding basic block. The weight of each basic block, w_i, is equal to $\mid MWIS(H^{*t}) \mid$, where H represents the data flow graph of the basic block. The cardinality of the maximum weighted independent set of transitive closure of H, or $\mid MWIS(H^{*t}) \mid$, can be determined using existing results [6].*

In summary, the linear increase of the maximum total weighted delay budget with the timing relaxation for each basic block, provides a fast, accurate method to determine the delay vector, d, and the weight vector, w, for the original problem (delay budgeting for soft realtime applications). Once the original problem is solved and timing constraints for basic blocks are determined ($d_i + b_i$), existing basic block level techniques can be utilized to assign the delay budget to operations.

8. Timing Budget Management for CDFGs

In this section, we reformulate the CDFG time budgeting problem and present our solution. Note that vectors d, w, p (node probabilities) can be determined using polynomial efficient algorithms from problem structure and characteristics of the operations in the library (Sections 6 and 7). The problem of timing budget management for CDFGs presented in Section 5 is equivalent to the following problem:

Given $G(V,E)$, a library of functional units for implementing operations in G, vector p for edges, and an expected delay constraint D_{max}, and determined (using aforementioned techniques) vectors d, w and p for nodes; The objective is to determine an integer delay budget b_i for each v_i that maximizes $\sum w_i.b_i$ subject to meeting the expected delay constraint of the application. The constraint is $d_{exp}(G) = \sum_{v_i} p_i.d'_i = \sum_{v_i} p_i.(d_i + b_i) \leq D_{max}$, or equivalently, $\sum_{v_i} p_i.b_i \leq D_{max} - \sum_{v_i} p_i.d_i = D^*_{max}$, where p_i is the probability of node v_i being executed over input data space (on any trace), and $d' = d + b$ is the updated delay of nodes (basic blocks) after slowing them down by vector b.

Interestingly, the problem is transformed into maximizing a linear expression of budget variables, under the constraint of another linear expression of budget variables. For arbitrary G, D_{max}, and edge p vector, the coefficients of the linear expressions (w and node p vectors) are arbitrary. Therefore, the problem is equivalent to the general integer knapsack problem, which is proved to be NP-complete [8]. Hence, standard dynamic programming solutions with pseudo polynomial complexity (with respect to timing constraint), and strongly polynomial ε-approximation algorithms are both applicable to the problem at hand [8, 21]. For real life CAD problems with practical graph size, and timing constraints, exact pseudo polynomial algorithms reflect reasonable performance. For example, integer linear programming (ILP) only took 0.06 second on an ordinary PC to solve our largest problem instance. Section 9 discusses the details of our experience in practice.

After solving the problem and determining the vector b, each basic block is assigned a local timing constraint, namely, basic block i is assigned the timing constraint $d_i + b_i$. Then, the existing DFG-based timing budget management algorithm [19] can be applied to all of the basic blocks to assign delay budgets to the operations in each DFG. The delay budget of an operation allows optimization of the operation, or smart selection of the operation from the given library.

9. Experimental Results

We utilize our delay budgeting technique during core-based synthesis of application CDFGs. Figure 6 illustrates our synthesis flow for mapping the applications to an FPGA device. We implemented the aforementioned probabilistic time budgeting algorithm to evaluate its impact on data-path energy dissipation and area. We compare its effectiveness against max-budgeting, a pessimistic optimal competitor (optimal under hard realtime constraint), and not performing delay budgeting. These algorithms correspond to the three application to analysis paths in Figure 6. We extract the application control data flow graphs (CDFG) from MediaBench [4] test suite using SUIF compiler [15] and Machine-suif [13]. To locate the computation kernel of each application, we apply call graph profiling on MediaBench to estimate the time spent in each sub-program of the application. Based on the the call graph profiling results and the CDFG structure of each subprogram, we choose ten functions as testbenches of our experiments. The testbenches are profiled

to compute the edge probabilities required to perform probabilistic time budgeting. Note that the MediaBench is comprised of multi-media applications, which cope with our intensity, periodicity, loss-tolerance, and content-sensitivity assumptions quite well (Section 4). The characteristics of benchmarks are shown in Table 1.

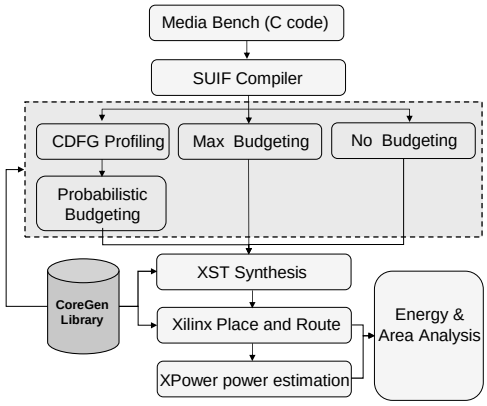


Fig. 6: Experimental flow to compare delay budgeting schemes

For experimentation purpose, we assumed that all of the operands are 8-bit wide. Furthermore, we assumed that the timing constraint for each application is equal to its critical path latency. For probabilistic budgeting, timing is treated as a soft constraint, and the application's expected latency is guaranteed not to exceed the timing constraint. However, for max budgeting, which performs worst case analysis, the timing constraint is met for all of the input samples. We used Xilinx CoreGen [10] to generate parameterized hardware modules (cores) with different latencies. Xilinx synthesis (XST) and placement and routing tools [10] are used in our flow to implement the designs and measure their area requirement. Xilnx XPower tool is utilized for power estimation. Our target platform is XCV3200V from Xilinx VirtexE FPGA series.

CDFG	Application	# of Basic Blocks	# of ALU Operations	Critical Path (cycles)
wrjpeg	jpeg	37	49	48
decode-mcu	jpeg	37	66	52
process-opt	mpeg2dec	47	59	50
mpeg-dec-1	mpeg2dec	26	47	51
mpeg-dec-2	mpeg2dec	36	69	56
reference-idct	mpeg2dec	10	29	33
readparmfile	mpeg2enc	84	112	82
smooth-color	mesa	36	61	54
comp-row	mesa	17	39	34
comp-noise	rasta	72	192	168

Table 1: Characteristics of benchmark control data flow graphs

We characterized the area variations of the CoreGen library modules with respect to their latency. Figure 7 demonstrates the area characteristic of the CoreGen sequential multiplier cores. In our experiment, we assigned the timing budget only to multipliers of the application CDFG, because CoreGen shifting, addition and subtraction cores implement latency variation by inserting registers and pipelining the operation, which increases their area. We also synthesize multiplexers to implement functional unit input sharing [20]. The multiplexers are not targeted for delay budget assignment in our experiments.

For further efficiency of the budgeting policies, we impose the upper bound of two times the critical path on the delay relaxation of each basic block. This prevents the assignment of large delay relaxation to a few basic blocks, and provides a rather fair distribution of delay budget over them. Multipliers are the only candidates for intra basic block delay budgeting. Therefore, they are assigned the weight 1, while every other operation has the weight 0 for determining the maximum weighted independent set (MWIS) of basic blocks.

Our experiment results are summarized in Table 2. For each benchmark, the design area in LUT and slice count, the energy dissipation in nanojoule, total delay budget ($\sum b_i$) and the number of basic blocks

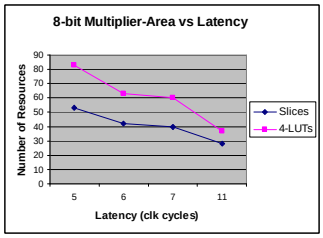


Fig. 7: Area characteristics of library multiplier cores

with delay relaxation ($b_i \neq 0$) are reported. The applications timing constraint are equal to their corresponding critical path latency. Our results highlight that total delay budget correlates well with design utility (energy and area in our study). It also shows that probabilistic budgeting always outperforms the other two competitors. On Average, we improve the energy consumption by 20.3% and 13.8%, and the area requirement (LUT count) by 18.9% and 10.4% compared to not using budgeting or using the maximum budgeting, respectively.

Unlike maximum budgeting (optimal under hard timing constraint), probabilistic budgeting assigns the delay budget to basic blocks based on the characteristics of control flow. The associated cost is occasional timing violation for infrequent input data patterns, which would sporadically hinder the realtime quality. Assuming this cost is tolerable for soft realtime applications, probabilistic budget assignment approach consistently outperforms the other two competitors.

Similar to other delay budgeting algorithms, the topology and connectivity of the applications CDFG affect the effectiveness of probabilistic budgeting. However, unlike other competitors the control flow behavior of the application can either provide additional opportunity, or limit the performance of probabilistic budgeting. For example, there is a small difference between the area associated with the two algorithms for *reference-idct*. This is mainly attributed to the fact that control flow structure visits all of the basic blocks with multiplier operations in most of the execution traces.

10. References

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

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

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

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

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

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

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

[8] M. Garey and D. Johnson. *Computers and Intractability, A guide to the theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

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

[10] Xilinx Inc. *"Xilinx documentations and online manuals"*. http://www.xilinx.com.

[11] J. Luo and N. Jha. "Battery-Aware Static Scheduling for

Benchmark	Design Metric	Delay Budgeting Algorithms			Probabilistic vs. Max-Budgeting Improvement %	Probabilistic vs. No-Budgeting Improvement %
		No-Bud.	Prob.	Max		
wrjpeg	LUT count	1051	882	959	7.33	16.08
	Slice count	633	546	583	5.85	13.74
	Energy	876	735	840	11.99	16.10
	Total budget	0	16	12	33.33	-
	Relaxed nodes	0	4	2	100.00	-
decode-mcu	LUT count	1820	1528	1724	7.80	13.08
	Slice count	1574	1458	1520	3.94	7.37
	Energy	1450	1210	1370	10.72	16.80
	Total budget	0	19	14	35.71	-
	Relaxed nodes	0	6	3	100.00	-
process-opt	LUT count	1206	935	1022	7.21	22.47
	Slice count	730	625	587	5.21	19.59
	Energy	1080	909	1020	10.65	15.40
	Total budget	0	25	22	13.64	-
	Relaxed nodes	0	7	5	40.00	-
mpeg-dec-1	LUT count	1034	878	957	7.64	15.09
	Slice count	624	562	599	5.93	9.94
	Energy	954	768	883	12.25	18.72
	Total budget	0	18	13	38.46	-
	Relaxed nodes	0	7	3	133.33	-
mpeg-dec-2	LUT count	1423	1192	1276	5.90	16.23
	Slice count	911	844	875	3.40	7.35
	Energy	1290	1070	1250	14.11	17.48
	Total budget	0	22	16	37.50	-
	Relaxed nodes	0	8	4	100.00	-
reference-idct	LUT count	887	816	847	3.49	8.00
	Slice count	770	736	752	2.08	4.42
	Energy	417	355	393	9.19	14.96
	Total budget	0	13	12	8.33	-
	Relaxed nodes	0	5	2	150.00	-
readparmfile	LUT count	2655	2150	2413	9.91	19.02
	Slice count	1889	1610	1736	6.67	14.77
	Energy	2800	2430	2720	10.29	13.45
	Total budget	0	22	25	-12.00	-
	Relaxed nodes	0	8	5	60.00	-
smooth-color	LUT count	1591	1322	1428	6.66	16.91
	Slice count	1064	964	1013	4.61	9.40
	Energy	789	679	762	10.55	13.97
	Total budget	0	15	17	-11.76	-
	Relaxed nodes	0	7	4	75.00	-
comp-row	LUT count	988	872	913	4.15	11.74
	Slice count	593	552	572	3.37	6.91
	Energy	455	373	428	12.29	18.12
	Total budget	0	10	12	-16.67	-
	Relaxed nodes	0	4	2	100.00	-
comp-noise	LUT count	5228	3870	4820	18.17	25.98
	Slice count	3662	2890	3425	14.61	21.08
	Energy	10000	7510	9180	16.65	25.19
	Total budget	0	106	94	12.77	-
	Relaxed nodes	0	19	26	36.84	-
Average	LUT count	1788.3	1449.9	1635.9	10.40	18.92
	Slice count	1245	1074.9	1170	7.64	13.66
	Energy	1730	1380	1620	13.81	20.33
	Total budget	0	26.6	23.7	12.24	-
	Relaxed nodes	0	8.2	4.9	67.35	-

Table 2: The quality comparison among delay budgeting algorithms

Distributed Real-Time Embedded Systems". In *IEEE/ACM Design Automation Conference*, 2001.

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

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

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

[15] M.W. Hall, J.M. Anderson, S.P. Amarasinghe, B.R. Murphy, L. Shih-Wei, E. Bugnion, and M.S. Lam. "Maximizing Multiprocessor Performance with the SUIF Compiler". *Computer*, 29(12):84–89, 1996.

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

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

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

[19] S. Ghiasi, E. Bozorgzadeh, S. Choudhury, M. Sarrafzadeh. "A Unified Theory of Timing Budget Management". In *IEEE/ACM International Conference on Computer-Aided Design*, pages 653–659, 2004.

[20] S. Ogrenci Memik, G. Memik, R. Jafari, E. Kursun. "Global resource sharing for synthesis of control data flow graphs on FPGAs". In *IEEE/ACM Design Automation Conference*, pages 604–609, 2003.

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

[22] W. Zhang, N. Vijaykrishnan, M. Kandemir, M.J. Irwin, D. Duarte, and Y.Tsai. "Exploiting VLIW Schedule Slacks for Dynamic and Leakage Energy Reduction". In *ACM/IEEE International Symposium on Microarchitecture*, 2001.