Estimating Implementation Bounds for Real Time DSP Application Specific Circuits

Jan M. Rabaey and Miodrag Potkonjak

Abstract—**This paper discusses techniques for estimating implementation bounds on computational resources and their role in the high-level synthesis process. Accurate estimations can be extremely useful in a multitude of synthesis operations, such as algorithm and architecture selection, design space search, module selection, transformations, allocation, assignment, and scheduling. Several techniques to efficiently estimate sharp minimum and maximum bounds on the resource requirements of a hardware implementation are discussed. The performance of the algorithms as well as their applications is analyzed using an extensive benchmark set. The proposed techniques have been implemented in the HYPER synthesis system.**

I. Introduction

AT NUMEROUS times in the design process of a real time application, important decisions have to be made that might dramatically affect the quality of the final solution. Unfortunately, most of these decisions are currently made on an ad hoc base, since evaluating the effect of a decision requires a complete run through the design process and is therefore extremely time consuming. The advent of high-level synthesis helps to alleviate this problem, as it allows for a much faster traversal of the design cycle. However, the majority of the high-level synthesis tasks, such as optimizing transformations, allocation, assignment, and scheduling have been proven to be at least NP-complete. To solve the problems in polynomial time, numerous heuristic as well as probabilistic solutions have been proposed. As a result, it is hard for a designer to determine the quality of a solution obtained with a particular synthesis environment. Furthermore, even though faster than the manual approach, running through the complete synthesis cycle still takes a substantial amount of computation time and is hence not effective for traversing the global design space.

The problem could be alleviated considerably if estimations of the implementation requirements of an application could be made fast and accurately. Currently, the implementation quality of an algorithm is usually expressed by the number of operations required during the execution of an algorithm [2]. Very often, a significant emphasis is placed on the number of multiplications. However, such measures are poor predictors of the ASIC implementation cost of the algorithm. For example, important parameters such as the distribution of the operators over the algorithm, the critical path, and the cost of memory and interconnect are neglected.

This paper presents a set of techniques to accurately predict the computational requirements of an algorithm, given the algorithmic flow graph and the maximum execution time. A technique called *discrete relaxation* is introduced to establish sharp minimum and maximum bounds on all hardware resources. The computed bounds can be used for a myriad of purposes in the design synthesis process.

1) The derived minimum and maximum bounds delimit the search space, thus speeding up the design synthesis search process.
2) The minimum bounds can serve as an initial solution for the above search process. We have experienced that this solution is often very close to the final solution.
3) Most synthesis tasks are optimization tasks that attempt to minimize a cost function, which is very often the implementation area. The accuracy of this cost function will directly influence the quality of the synthesis process. Accurate estimations can therefore help to boost synthesis performance by contributing to the accuracy of cost functions.
4) Estimations can help to establish the relative or absolute quality of a proposed solution. This information is extremely useful to direct the overall synthesis search process. For instance, in an iterative transformation environment, it is important to know how much a proposed transformation will influence the implementation cost, hence establishing a relative ordering of the candidate moves. It is also useful to know the maximum improvement to be expected from a transformation.
5) Estimation of the optimal solution can help to determine the absolute quality of a synthesis algorithm (such as a scheduler or a transformation). Since most algorithms are at least NP-complete, benchmarking (with all associated traps [23]) is the dominant approach at present.
6) Finally and most importantly, estimations can play a crucial role in the algorithm and architecture selection and partitioning processes, topics that are by and large unexplored territory at present.

The estimation techniques described in this paper were developed for the analysis of high-performance signal processing applications, where the implementation cost is dominated by the computation and storage resources and the cost of the

Manuscript received September 22, 1991; revised April 26, 1993. This work was sponsored in part by DARPA, the Semiconductor Research Consortium, and grants from MICRO, Harris Semiconductor, LSI Logic, Analog Devices, and Sony Corporation. This paper was recommended by Associate Editor R. Camposano.

J. M. Rabaey is with the Department of Electrical Engineering and Computer Science, University of California-Berkeley, Berkeley, CA 94720 USA.

M. Potkonjak was with the Department of Electrical Engineering, University of California-Berkely; he is now with C & C Research Laboratories, NEC USA, Inc., Princeton, NJ 008540 USA.

IEEE Log Number 921546.

controller is only of secondary importance. Therefore, this paper will concentrate on the former components.

After a brief discussion of the previous work in this area and a description of the global estimation framework, the proposed techniques for the estimation of maximum and minimum bounds will be discussed in detail. The effectiveness and efficiency as well as the application domain of the estimation techniques will be demonstrated with a number of examples.

II. Previous Work

While the idea of estimations is relatively new in the high level synthesis arena, the concept of resource requirement prediction has been around for quite some time in the areas of compiler design, performance modeling, operations research, and VLSI and digital signal processing. An overview of the state of the art in those areas will help to substantiate the presented results.

Until recently, the software compiler literature has ignored the topic of estimations and their use. This is easily explained by the fact that the designers of software compilers are as much concerned about compilation speed as about execution speed. Therefore, the most complex scheduling algorithms used are of a quadratic worst case complexity, which is as fast as any nontrivial estimation technique can achieve. With the introduction of vector computers [1], [28], very long instruction words [43], and super-scalar and super-pipeline architectures [24], [54], numerous studies have been published on the available parallelism in both general purpose and numerically intensive computations. The scope of those papers is quite different from the one addressed here: their main goal is to demonstrate that a particular class of algorithms has sufficient parallelism for a given architecture. This is measured by explicit scheduling of the algorithms on the target architecture.

Performance modeling, and benchmarking in particular, have recently received a lot of attention [18], [23]. The major concern here is to predict the average performance of a general purpose machine for a particular group of users. The most often-used techniques are simple statistical models, which are built either manually or with the help of statistical packages and use as input parameters the run-time results of a set of benchmark programs typical for a particular area (for instance linear algebra or databases).

The operations research and theoretical computer science efforts that most resemble the problem addressed here is in the framework of approximation algorithm development [19]. The goal is to develop algorithms for a particular NP-complete scheduling problem, which guarantee that the solution will be within ϵ percent of the unknown optimal solution. Although those algorithms have a polynomial complexity, their complexity order is most often high. The goal is once again different: explicit solution generation versus prediction.

The prediction of implementation requirements is of extreme importance in the area of digital signal processing and has therefore been discussed extensively [2]. The predominant measure used to express the quality on an algorithm is the number of operations, often with stress on the number of multiplications. Since the late seventies, the regularity of an algorithm has also become an important measure [31], [40].

In the VLSI arena, theoretical computer science has made some significant progress in the study of lower bounds on area and time of elementary circuits such as adders and multipliers [59]. Three fundamentally different techniques are used, providing complexity measures on A (area), $A \times T$ (area-time), and $A \times T^2$.

It seems that the use of prediction has predated CAD in the circuit design area. In the early 1960's, an unpublished study was performed at IBM by E. Rent on the structure of computer logic design. His formulation, now known as Rent's rule, was followed by many other studies, not only on the block-to-pin ratio, but also on other physical implementation parameters [17]. More recent studies achieved an excellent correlation between predicted and actual physical design characteristics [34], [46], [53].

The role of estimation has not received much attention in the architectural and high-level synthesis area. Early estimation efforts include a model developed by Davio *et al.* [10]. One of their assumptions was that all nodes (called universal logic elements) have identical delay, area, and functional characteristics, which greatly reduces the prediction complexity but severely limits the application range.

Significant and important work has been performed at the University of Southern California. Kurdahi [33] presented a technique to predict the number of required registers using a variant of the Dinic max-flow/min-cut algorithm, which was later refined by Mlinar [41]. Jain [21] used absolute min-bounds (discussed later in this paper) to drive the module selection process. Most recently, Kucukcakar [30] reported the successful usage of prediction tools in the partitioning phase of behavioral synthesis as well as several new estimation approaches [29].

Kurdahi also developed PLEST [33], [34]—a CAD tool for estimation of the area of standard cell VLSI chips. Several other researchers addressed the problem of area prediction, starting from the circuits schematic expressed in a hardware description language. CHAMP [58] also estimates the areas of standard cell blocks by using empirical formulas obtained by extensive experimentation. Chen and Bushnell [5] reported estimator modules in both standard cell and full custom layout methodologies. Zimmerman [63] employs the min-cut algorithm as a fast means to generate slicing trees used in area estimation for both standard cell and general cell designs. All four groups reported excellent prediction capabilities (within 10% of actual implementation).

Two approaches advocate the direct consideration of layout information in high-level synthesis. Weng and Parker propose integration of scheduling with floorplanning [61], and Knapp [25] proposes use of layout data to drive the choice of optimizing RTL transformations for iterative improvement of designs. Minimization of literal count is a common objective during logic synthesis optimization. Lightner and Wolf [37] showed that for standard cell design, literal count is indeed a good estimator of area before routing.

Several approaches on the border between high-level and lower-level design have also been reported. The Chippe expert

design system [3] analyzes interconnect in terms of performance, area, and power using a worst-case waterfilling model. The ELF system had a mechanism for the prediction of the wiring area, given an RTL level description [15]. Several authors, including [39], studied the area-delay performance tradeoff, yielding a model that could be used in a prediction tool. Wu, Chaiyakul, and Gajski [62] proposed layout area models starting from an RTL description for two types of datapath (bit-slice stack with abutment and bit-sliced macro standard cells) and control (random logic and PLA) layout models. Finally, Powell and Chau proposed an approximation technique for the early estimation of power consumption [50].

The Tron system [11] uses a hierarchical min-flow algorithm for estimating the upper bound on the number of required registers so that register cost can be taken into account during efficient list scheduling. Another high-level synthesis estimation technique was recently proposed by Hagerman [16]. Hagerman uses the concept of a distribution graph [44] to estimate the functional unit allocation used by a modified force directed scheduling, allocation, and mapping tool [8] and reports favorable results.

The techniques presented in this paper differ in both the employed techniques as well as the application areas. The idea of relaxation is introduced and applied extensively for the estimation of computational units, memory, and interconnect. Attention is paid to hierarchy in the representation, an area that has until now been largely unexplored. Furthermore, techniques to derive the best possible estimations for max-bounds are presented. The obtained results are analyzed and verified.

III. Global Framework

Before starting the discussion of the estimation algorithms, a number of assumptions and definitions have to be put forward. First of all, we will define our target goal more precisely. The goal is to develop sharp minimum and maximum bounds on the required quantity of execution units, registers, and interconnect in ASIC implementations of numerically intensive applications. In these types of applications, the datapath resources contribute a majority of the implementation costs. Numerically intensive algorithms are common in a wide range of important applications such as sonar, radar, speech recognition and synthesis, image and video processing, and sonographics. The developed techniques can be also used for estimating the datapath cost in arbitrary architectures in an application specific domain (for instance, control dominated ASIC's). In conjunction with estimation tools for controllers and random logic modules, they provide an overall prediction tool.

The techniques presented in this paper concentrate on determination of bounds on the *number* of execution units, registers, and interconnect. Since there is a direct relation between the number of execution units and registers and their respective areas, simple calculation gives the corresponding areas. There does not exist a direct relation, however, between the number and area of interconnect, since the later is dependent on the effects of lower level CAD tools and methodology. There exists, though, a good correlation between the two. By presenting the number of interconnect lines as the final result, we enable comparison with other high level synthesis estimation techniques. Also, it is important to note that this information can be obtained significantly faster than interconnect area, and can be used in conjunction with high-quality, lower-level prediction tools.

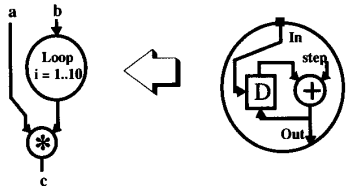

Fig. 1. Example of a Hierarchical Flow Graph. This graph is equivalent to the following computation: $c = a^*(b + 10^*\text{step})$. D represents a delay operation with initial value In.

We assume that the algorithm under study is represented as a Data-Control Flow Graph G (N, E, C), where the nodes N represent the flow graph operations with fixed latency delay, and the edges E and C represent the data and control dependencies between the operations, respectively. The control dependencies are used to express relations between operations, which are not imposed by the data precedence relations. The control dependencies are particularly useful for the expression of timing constraints, as well as for the implementation of side effects caused by memory assignments (both for background and foreground memories) [52].

We also assume that the graph G is a hierarchical graph: each vertex N of G can be an instance of a subgraph $G'(N', E', C')$. An example of such a hierarchical flow graph is shown in Fig. 1. The representation allows for the simple introduction of loops and block-conditionals in the flow graph. It is assumed that for each data dependent loop, either the maximum or average number of iterations, is known. Which one is chosen depends upon the application. Some signal processing applications (for instance audio processing) require that a fixed throughput rate is sustained, hence enforcing worst case design; others, such as speech recognition, only require an average rate. The value of those nondeterministic parameters can be estimated through algorithm profiling, based on simulation or sometimes using amortization techniques [9].

In order to make the estimations useful and accurate, a well-defined underlying hardware model is essential. The following restrictions are placed on the implementation:

1) All edges E represent variables, which will be stored in registers.
2) All leaf nodes N are purely combinational. The hardware unit of choice r (such as adder, multiplier or ALU) and its execution time t_{dr} (in number of clock cycles) is known *a priori*.

It is always possible to transform a noncomplying flow graph into the format described above by contracting nodes, connected by temporary edges (without storage), into a single, complex combinational node.

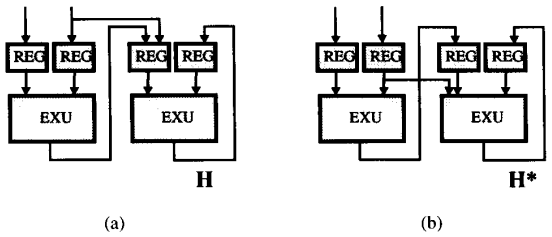

Fig. 2. Hardware Models. (a) INTERCONNECT-REGFILE-EXU. (b) INTERCONNECT-REGFILE-INTERCONNECT-EXU.

The paper will adopt the hardware model H, shown in Fig. 2(a). It is assumed that all registers are clustered in register files, connected to the inputs of the execution units. The techniques described below are, however, easily adapted to other models, such as the "register file-interconnect-exu-interconnect" model shown in Fig. 2(b) (called H^* in the rest of the text). Although the hardware model will not affect the execution unit bounds, it *will* have a severe effect on interconnect and register bounds (see Section IV.B).

The estimation process for real time applications can be defined within this framework:

Given a hierarchical flow graph $G(N, E, C)$, an underlying hardware model H and a maximum execution time $t_{\max}$, determine the minimum and the maximum bounds on the required hardware resources (execution units, registers and interconnect) such that the graph G can be executed within $t_{\max}$.

Flow graph transformations are often the most powerful high-level synthesis tool, in that they can drastically reduce implementation requirements. This power makes a comprehensive prediction of transformation effects an extremely difficult task. For several transformations (e.g. loop pipelining and commutativity), insightful prediction tools are developed in the HYPER high-level synthesis system [47]. In general, though, a different strategy for estimating effects of transformations is used. This is done for two reasons: the effect of a transformation greatly depends on the particular algorithm that executes the transformation, and run times of transformation algorithms are often short. HYPER first performs a transformation and then estimates its effects. (If the effects are counterproductive, the transformation is rejected.) Therefore, we assume that in all presented examples no transformations will be applied. In the experimental result section, half of the examples are obtained by applying one or more transformations on the initial flow graph.

For the sake of completeness, it should be mentioned that the techniques presented below are easily adapted to address the dual problem (given the hardware resources, estimate the bounds on the execution time). The rest of the paper will now proceed as follows: Section IV will describe techniques to estimate the max-bounds on the resources, while a selection of approaches to estimate min-bounds will be analyzed in Section V. The paper will conclude with a study of the applications of prediction techniques and future work. A number of examples will be used throughout the paper to demonstrate the effectiveness of the proposed techniques.

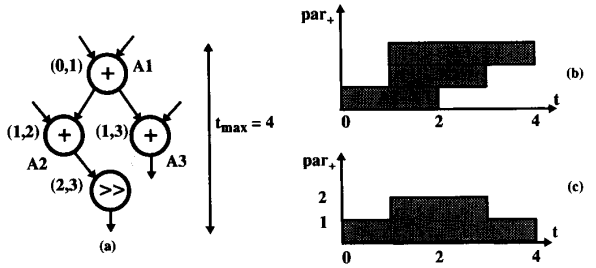

Fig. 3. Analysis of Max-Bound: example Flow graph (a), Parallelism Graphs before (b) and after (c) precedence collapsing. It is assumed that each operation takes one control step.

IV. Estimating the Maximum Bounds

It might be argued that maximum bounds on hardware resources are hardly interesting, since the synthesis process is in essence a minimization process. However, knowing the upper bounds helps to delineate the search space for the hardware allocation and transformation processes. Furthermore, it is in general advantageous to have the maximum bounds on the resources as high as possible, since they are a measure of the concurrency available in a particular instance of the flow graph (this is demonstrated in this paper as well as in [48]). Therefore, results of the max-bound estimation can act as a driver for the concurrency-improving transformations.

A. Max-Bounds on Execution Units

For the sake of clarity, we will first assume that the graph G does not contain any hierarchy. This constraint will be relaxed further in the section. The estimation process starts with a topological ordering and leveling of the graph with respect to the input and output nodes. As a result, the earliest (t^i_{asap}) and latest (t^i_{alap}) execution times are obtained for each node N_i. The length of the time slot available for the execution of N_i is called the *slack time* ($t^i_{\text{slack}} = t^i_{\text{alap}} - t^i_{\text{asap}}$). The length of the critical path is also determined during this process.

Based on this information, a set of parallelism plots par $_r(t)$ (with r = 1..R; R = total number of resource classes) can be constructed:

$$\text{par}_r(t) = \sum \text{all nodes } i \text{ executed on resourcer } r \quad (\text{with } (t^i_{\text{asap}} < t < t^i_{\text{alap}})) \tag{1}$$

Such a parallelism plot displays the potentially available concurrency over time. Hence, the max-bound on resource r Equals:

$$\max_r = \text{MAX}_{t=1}^{t_{\max}} (\text{par}_r(t)) \tag{2}$$

Equation 2 is, however, overly pessimistic, since it totally ignores the precedence relationships that might prevent nodes from being executed simultaneously, even though they have overlapping slack times. We will call this bound the *absolute max-bound*. The simple example of Fig. 3 will clarify the issue. Assume that all operations take 1 control step and that $t_{\max}$ equals 4 clock cycles. The earliest and latest execution times of each node are shown between brackets. par $_+(t)$ is

plotted in Fig. 3(b). This suggests that the maximum parallelism for adders equals 3. A close inspection of the plot of Fig. 3(a) clearly proves that this is not achievable, due to the precedence relations between the operations.

A more precise max-bound can be obtained by eliminating nodes with potential concurrency, but with precedence relationships between them (as demonstrated in Fig. 3(c) for the simple example). This task can be defined as a maximum independent set problem [23]:

Given:
For control step t and resource r, a set of nodes N_i with the following properties: N_i is executed on r and $t^i_{\text{asap}} \leq t \leq t^i_{\text{alap}}$, a set of precedence relations (E, C) between the nodes N_i.

Problem:
Determine the maximum potential concurrency at time t for resource r.

Solution:
Construct a directed graph $F(N, R)$, with the N_i as nodes. An edge R_{ij} is provided between two nodes N_i and N_j when there exists a precedence relation ($\in (E, C)$) between the two nodes. The maximum concurrency equals the *maximum independent* set of F.

Proof:
Two nodes N_i and N_j can be executed simultaneously at time t when no precedence relationship exists between them or, in other words, when no edge exists between them in the graph F. This is exactly the definition of an independent set: two nodes of a graph F form an independent set, when F contains no edge between those two nodes. The maximum possible concurrency then obviously equals the maximum set of nodes without precedence relations or, equivalently, the maximum independent set.

The maximum independent set problem is known to be NP-complete [23]. Fortunately, the graph F exhibits a property that turns the problem into a polynomial one. It is known that the maximum independent set problem for a class of graphs, called *comparability graphs* [14], can be solved in polynomial time ($O(N^3)$) using a minimum-flow algorithm. A comparability graph $F(N, R)$ is defined as a graph with the following property: if $R_{ij} \in R$ and $R_{jk} \in R$, then also $R_{ik} \in R$. This is clearly valid for the graph F, defined in the max-bound problem: when a precedence relation is present between nodes N_i and N_j as well as between nodes N_j and N_k, then node N_i must also precede N_k. Efficient algorithms to solve the maximum independent set problem for comparability graphs have been published in [14, p. 135]. It should be noticed also that the sizes of the graphs F are small, since they only consider the nodes, alive at time t and executing on resource r. The independent set problem has to be executed for every time t and for every resource (thus $t_{\max} \times R$ times) to obtain the improved parallelism graphs.

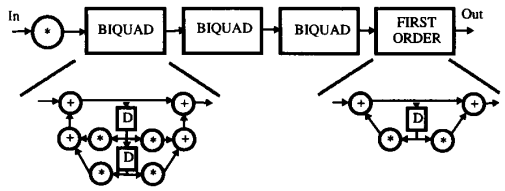

Fig. 4. Seventh-order biquadratic IIR filter: Signal flow graph.

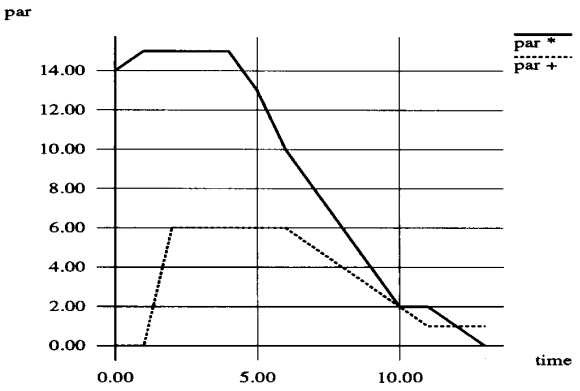

Fig. 5. Parallelism plots for 7th-order IIR filter ($t_{\max} = 14$, $t^* = 2, t+ = 1$).

The max-bound algorithm has been applied on the example of a seventh order biquadratic IIR filter, shown in Fig. 4. The results are plotted in Fig. 5 for both adder/subtracters and multipliers[1] (for $t_{\max} = 14$). From the parallelism plots, it can be deduced that the max-bounds on multipliers and adders equal 15 and 6, respectively. More important than the bounds themselves (which are of little practical value besides the delineation of the search space) is the information gained by studying the structure of the parallelism graphs. In the example, note that almost all parallelism is available in the initial clock cycles. This will definitely result in a poor implementation with low resource utilization towards the end of the algorithm. This demonstrates that the distribution of the plots can serve as a measure to drive resource-utilization improving transformations, such as retiming and pipelining [48].

The above techniques can be easily extended to cover hierarchical graphs as well. The main problem in dealing with hierarchy is that the available time is only known for the uppermost level and not for the sub-graphs. Fortunately, there is little or no dependency between the available time and the max-bound (since the dependencies remain identical). We therefore opted for the following approach: For each subgraph, the max-bounds of the resources are estimated using the subgraph's critical path as the available time. The max-bounds for the hierarchical graph are then obtained by taking the maximum over all sub-graphs for all resources.

[1] In a real implementation of such a filter, multiplications are replaced by add/shifts. We have opted here for using parallel multipliers (with a duration of 2 clock-cycles) to simplify the example.

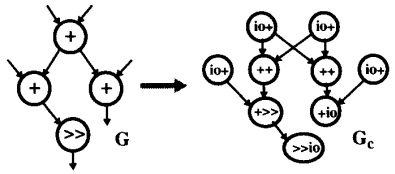

Fig. 6. Derivation of connectivity graph from computation graph. The nodes in G_c are represented by the source and destination resources (e.g, ioAE+).

B. Max-Bounds on Connectivity and Registers

Similar techniques can be used to estimate max-bounds on connectivity. The algorithms are executed on the connectivity graph $G_c(N_c, E_c)$. Every node N_c in G_c corresponds to an edge $E(N_i, N_j)$ in G and represents a hardware connection between resource r_i and r_j. For our hardware model H, the slack time of N_c is equivalent to the slack time of the source node N_i, since the output bus is directly connected to the source and should therefore be reserved for the duration of the computation. An edge E_c is defined in G_c when there exists a precedence between two interconnections. These precedences can be derived directly from the computation graph G. G_c is also a comparability graph [14]. Note that no assumptions are made about module binding. An example of how to derive the connectivity graph from the computation graph is shown in Fig. 6. It should be stressed that the construction of G_c strongly depends upon the hardware model H used. Furthermore, one should be aware that although there exist a strong correlation between the number of busses in the architecture and the interconnect area in the physical implementation, no precise cost can be attributed to a bus. This is in contrast to execution units and registers, which have a very precise implementation cost. Finally, it should be noticed from Fig. 6 that the interconnect estimation also allows us to estimate the number of input-output ports needed. This is of crucial importance in real time applications, which are often input-output (and hence IO-pin) hungry (for example, see [57]). This measure can for instance be used when considering chip partitioning.

Establishing a sharp max-bound on registers is nontrivial and strongly dependent upon the selected hardware model. In the hardware model H, the register count is closely correlated to the EXU assignment: in the worst case, every operation can be assigned to a different EXU and hence every variable has its own register. Therefore, the absolute max-bound on the number of registers used is identical to the number of edges in the graph, each of them multiplied by their fan-out. Parallelism plots can also be derived for registers, using exactly the same techniques as described above. We use the graph G_r, which is actually identical in structure to the interconnect graph G_c for the hardware model H. Each node now corresponds to a variable. The slack time of the node is set to the maximum lifetime of the variable, which equals $t_{\text{alap}}^{\text{dest}} - t_{\text{asap}}^{\text{source}}$, with *source* the source node of the variable and *dest* the destination node.

In the hardware model H^* from Fig. 2, the broadcasting factor is greater than 1, potentially saving some registers at the expense of extra interconnect. Since the register assignment is decoupled from the EXU-assignment, accurate bounds can be predicted using the maximum independent set techniques on the comparability graph for all edges.

V. Estimating the Minimum Bounds

From a design point of view, far more interesting information is represented by the minimum bounds: accurate lower bounds allow us to estimate the absolute minimum area needed for the implementation of a given computational graph. Lower bounds can also serve as an initial seed for allocation and design space search processes, normally resulting in faster convergence. Finally, a good lower bound allows us to judge the quality of solutions produced by heuristic or statistical tools for NP-complete problems such as scheduling or module selection.

Unfortunately, no exact lower bounds for the design synthesis problem have been established, and deriving those might prove to be an NP-complete problem in itself[2]. In order to be useful, the estimation process should be very efficient (the complexity of the estimation routines should not exceed $O(N^2)$). This precludes the utilization of complex estimation algorithms. Instead, we opted for a technique called **discrete relaxation**, which turns the estimation problem into a tractable one by relaxing some of the constraints imposed by the original problem. As will be demonstrated below, this results in extremely efficient estimation, while still delivering very sharp bounds. One set of constraints on which we can relax are the precedence relations. Most of our attention will be devoted to this class of relaxations. As was done in the chapter on maximum bounds, we will first concentrate on the estimation for execution units and later extend the approach to registers and interconnect. At the end of the chapter, we will discuss several other relaxation approaches.

A. Min-Bounds on Execution Units—Leaf Graphs

As stated in Section II, the majority of the complexity estimation approaches in the high-level synthesis arena are based on the *absolute min-bound*. This min-bound is in essence Equivalent to the traditional approach of a designer, when he measures the complexity of an algorithm by counting the number of multiplies (as the parallel multiplier is the most expensive execution unit) and dividing them by the available time. Such a measure is a poor predictor of the ASIC implementation cost of an algorithm. It assumes that the flow graph contains enough parallelism to support 100% utilization of the resource. Furthermore, important elements contributing to the implementation cost, such as input, output, interconnect, and foreground and background memory are ignored. This approach therefore generally results in a poor estimation. Within our framework, the absolute min-bound for an execution unit $r(r = 1 \cdot R)$ is defined in

$$\min_r^{\text{abs}} = \left\lceil \frac{\eta_r \times t_{\text{dr}}}{t_{\max}} \right\rceil, \tag{3}$$

[2]The authors are not aware of any proof of this statement.

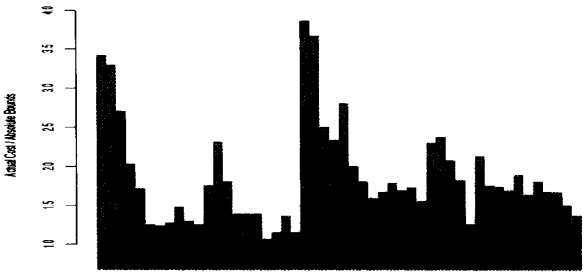

Fig. 7. Ratio between actual cost and absolute min-bound (for 50 examples).

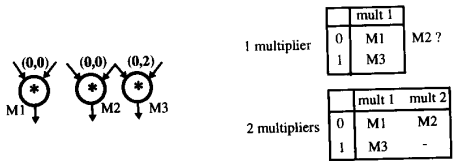

Fig. 8. Determination of relaxed min-bound for the example in Fig. 8.

with η_r the number of nodes to be executed on resource r and t_{dr} the number of clock cycles it takes to execute one operation on r.

We have evaluated the performance of the absolute min-bound using a class of 50 examples (all of them without hierarchy). The list of the examples includes the 7th-order IIR filter, the standard 5th-order wave digital filter, an 11th-order FIR filter, a 19th-order CORDIC rotation, and an 8-point discrete cosine transform. For all those examples, we constructed different alternatives structures (and hence different graph properties) by applying transformations such as pipelining and retiming. Furthermore, we considered multiple ratios of $t_{\max}$/critical path for each example. This ratio (which will be called the *stress ratio* from now on) influences the performance of the estimation algorithms in an important way. We have compared the results of the estimation with the results obtained after going through the complete synthesis process[3]. The results are plotted in Fig. 7. The cost factor used in the evaluations is equivalent to the total area of the execution units. The maximum error observed between actual and estimated cost equals 386.3%, while the average and median errors equal 72.1% and 86.6%, respectively.

An improvement on the absolute lower bound can be found by observing from the parallelism graphs (obtained in the previous section) that for some clock cycles not enough parallelism is available to sustain 100% utilization. This results in a more precise lower bound:

$$\min_r^{\mathrm{adj}} = \left\lceil \frac{\eta_r \times t_{\mathrm{dr}} + \text{Unused Time}}{t_{\max}} \right\rceil. \tag{4}$$

Unused Time in the above equation is actually a function of $\min_r^{\mathrm{adj}}$, as the resource utilization is clearly dependent upon the number of resources available. Equation (4), therefore, has to be solved iteratively:

1) Derive the Parallelism graph for r, using the techniques discussed in the previous section. Set the initial value of $\min_r^{\mathrm{adj}}$ to the absolute min-bound (3).
2) Compute Unused Time given $\min_r^{\mathrm{adj}}$.
3) Recompute $\min_r^{\mathrm{adj}}$ using (4).
4) If $\min_r^{\mathrm{adj}}$ changed with respect to the previous iteration, go to 2, or stop.

[3]To make the comparision between the different algorithms fair, we have used identical scheduling and allocation routines for all examples.

Even though this approach presents a significant improvement over the absolute bound, it still might be off significantly. This discrepancy is mainly caused by the fact that nodes with a large slack (or a lot of freedom to move) are falsifying the parallelism graphs by giving a too-optimistic point of a view on the available concurrency. This is demonstrated with the simple example of Fig. 8. Using (4), one would get the impression that one multiplier is sufficient, as the *Unused Time* on the Parallelism Graph for 1 multiplier is 0. A close inspection of the flow graph reveals that 2 multipliers are needed: both multiplications M1 and M2 have to be executed in the first time slot. The discrepancy between estimations and actual results is caused by the large slack of node M3, which gives the impression that at least one multiplication can be executed in every clock cycle.

A more advanced approach is therefore necessary. Our proposed approach is based on the principle of relaxation: while the general scheduling problem is NP-complete [13, pp. 236–244], some simplified scheduling problems have been proven to be of polynomial complexity. By removing some constraints of the original problem, we can turn the estimation problem into a polynomial one. In other words, we trade off accuracy for speed. In this section, we will relax on the precedence constraints. For the lower bound estimation problem for EXU r, let us temporarily consider the precedences only indirectly (through the ASAP and ALAP times of the nodes) and ignore the direct formulation. This translates the estimation problem into the following format:

Relaxed Estimation Problem: Given a computational problem, consisting of identical tasks with integer ASAP and ALAP times and a known duration t_{dr}, determine the minimum number of resources r needed to complete the task within the available time $t_{\max}$.

This problem cannot be solved directly, but can be defined as an iterative version of its dual formulation:

Dual Relaxed Estimation Problem: Given a computational problem consisting of η_r identical tasks with integer ASAP and ALAP times and a known duration t_{dr} and the number of available resources r, determine the minimum execution time $t_{\min}$.

Given a solution for the dual problem, the original relaxed estimation problem can then be solved with the following

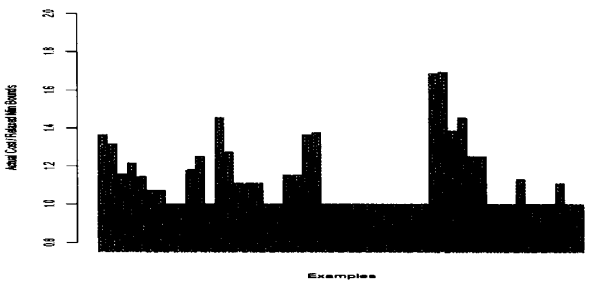

Fig. 9. Ratio between Actual Cost and Relaxed Min-Bounds (for 50 examples).

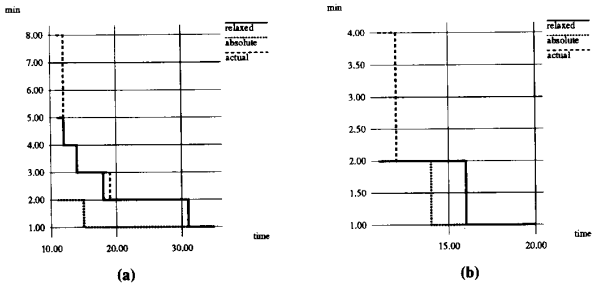

Fig. 10. 7th-order IIR filter: Minbound-time tradeoffs for multiplications (a) and add/substract (b) (relaxed, absolute, and actual).

simple iteration:

Relaxed Estimation Problem (iterative version):

1) *Set* $\min_r$ *to the absolute lower bound* $\min_r^{\text{abs}}$.
2) *Given* $\min_r$, *determine the minimum execution time* $t_{\min}$ *(dual relaxed estimation problem).*
3) *If* $t_{\min} > t_{\max}$ *or if no solution, increment* $\min_r$ *and go to 2, else* $\min_r$ *is the relaxed min-bound.*

This procedure is illustrated with the aid of the simple example of Fig. 8. The absolute min-bound on the number of multiplications for this example is equal to 1. Consider now the relaxed problem shown in Fig. 9. It is obvious that no solution can be found for the scheduling of the problem on 1 multiplier. When 2 multipliers are available, the problem can be solved in 2 clock-cycles ($< t_{\max} = 3$). Hence the $\min_*^{\text{rel}} = 2$.

The dual estimation problem as defined above is well known in the scheduling literature. From [38], the following analysis can be derived.

1) When the duration t_{dr} of all tasks (or operations) is Equal to 1 clock cycle, then the minimum execution time can be determined exactly using a *slack driven list scheduling* (also known as the earliest deadline scheduling algorithm). The complexity of this algorithm is $O(N \log N)$, with N the number of tasks.
2) When the duration of the tasks is larger than 1, the problem becomes more complex. It can be transformed into a scheduling problem with unit task duration time (by dividing all times with the length of the task t_{dr}), but in this case the new ASAP and ALAP times are no longer integers. Finding the exact solution for this problem requires backtracking during the list-scheduling and is known as the *earliest deadline with barriers scheduling* algorithm [55]. The complexity of this algorithm is $O(N^3 \log N)$. Since this exceeds our goal of using only algorithms with maximally quadratic complexity, some further relaxation is needed. This can be achieved by turning the ASAP and ALAP times into integer numbers. The ASAP times are rounded to the nearest lower integer number, while the ALAP times are rounded to the next higher integer. The resulting problem can once again be solved with the earliest deadline scheduling problem. It is easily seen that the integer relaxation can only lower $t_{\min}$, hence preserving the min-bound nature of the obtained solution.

The performance of the relaxed estimation routines is analyzed using the same benchmark set as was used for the absolute lower bound. The results are plotted in Fig. 9. The maximum error has been reduced to 67%, while the average and median errors equal 13.7% and 7%, respectively. The largest errors occur when the ratio approaches 1. This can be partially explained by the fact that for ratios near unity, the relaxation on the precedence relations introduces an oversimplification. Another reason could be that in the fixed amount of run time, the probabilistic scheduler performs better on examples that are not overly constrained.

The proposed iterative algorithm for solving the min-bound problem can also be used to generate the minbound-time plots for each execution unit, which determine for each possible time $t_{\max}$ the minimum number of units needed. These plots are extremely useful when studying the area-time trade-offs for a particular algorithm (this will be discussed in more detail in the application section). They will also be used extensively when performing estimations for hierarchical graphs.

Generating the Minbound-Time Graph for resource r:

1) it Set $\min_r$ to 1.
2) it Solve $t_{\min}$ using the dual relaxed estimation algorithm.
3) When $t_{\min} > t_{\text{critical path}}$, increase min_r and go to 2, else stop.

The minbound-time graphs for multipliers and adders for the 7th-order IIR filter are plotted in Fig. 10. To demonstrate the excellent performance of the relaxed estimation algorithms, the area-time plot obtained using the absolute min-bound estimation technique as well as the actual cost-time plot (obtained after allocation, assignment and scheduling) are also included. As can be noticed from the plots, the only major discrepancy between actual cost and the relaxed min-bound occurs when $t_{\max} = t_{\text{critical path}} = 11$.

B. Min-Bounds on Execution Units—Hierarchical Graphs

The situation becomes somewhat more complex when hierarchical graphs are considered. The problem here is that $t_{\max}$ is only defined for the overall problem. The distribution of the time over the sub-graphs is unknown and is actually an optimization problem of its own. Without loss of generality, we will assume that at every hierarchy level, the nodes of

the graph are either all hierarchy-nodes or all leaf-nodes. This can be achieved by clustering leaf-nodes into sub-graphs, such that all precedences are preserved. We also assume that only one sub-graph can be executed at a time (single thread of control). This is the most common case in both ASIC design practice and in high-level synthesis. While the use of multiple threads of control has high potential for some computational areas, it usually has as a consequence a controller of high complexity, which can have a negative impact on both the area and clock-speed of the ASIC design. Obviously, we can derive the bounds for multiple threads of control by flattening the flow graph. Since the algorithms for bounds run essentially in linear time (flow graphs are almost always sparse), sizable examples can be analyzed using this approach. If the task is to derive min-bounds while preserving hierarchy, it can be achieved by a dynamic programming algorithm. The run time complexity of this algorithm would be high, however, and the implementation complex. Therefore, for the reasons mentioned above, we will concentrate on single thread of control designs. For computations that have conditional branches, the min and max bounds for a particular control step are given by the maximum min and max bounds over all branches in that control step.

One method for performing the hierarchical estimation uses the minbound-time plots, derived in Section V.A. For an arbitrary hierarchical graph $G(N, E)$, let each node N represents a sub-graph $G'(N', E')$. Assume that for each node N, the minbound-time plots of its sub-graph G' are known as well as the maximum[4] number of iterations on the node.

The minbound-time plot for a resource r and for graph G (called $\mathrm{MB}_r(t)$) can now be constructed from the minbound-time plots of the sub-graphs ($\mathrm{MB}_r^i(t)$, with $i = 1 \cdots N$) in the following way:

Procedure Estimate Hierarchy (for resource r):

1) Set η_r, the number of resources of type r, to 1.
2) Compute $t^r_{\min}$ for graph G using (Eq 5).

$$t^r_{\min} = \sum_{i=1}^{N} (\mathrm{MB}_r^{i^{-1}}(\eta_r) \times \mathrm{iter}^i) \tag{5}$$

with iter^i the number of iterations of node i.

3) If $t^r_{\min} > t_{\text{critical path}}$, increment η_r and go to 2, else stop.

This procedure is illustrated for the simple example of Fig. 11. In the case of conditional graphs (if-then-else functions), we take the results of the worst case sub-graph. Procedure *Estimate Hierarchy* is repeated for every hierarchy level in the graph in a bottom-up fashion until the top level is reached. At that level, the available time $t_{\max}$ is known and the actual min-bound $t^r_{\min}$ can be determined. The minbound-time plots for a 19th-order CORDIC algorithm (containing one hierarchy level) are plotted in Fig. 12. The cost plotted here is the sum of the estimated minimum costs of the adder/subtracters (unit cost 3), shifters (cost 4), comparators (cost 3) and multiplexers (cost 1)[5]. A large discrepancy between absolute and relaxed min-bound can once again be seen. Note also that the actual cost tends to change every 19 cycles in large steps (which is equivalent to the number of iterations of the loop). This phenomenon is not present in such a clear form in the relaxed estimation cost.

[4] Average number when looking at the average throughput.

[5] These cost ratios are obtained from the actual data-path library of the LARGER-IV system [54].

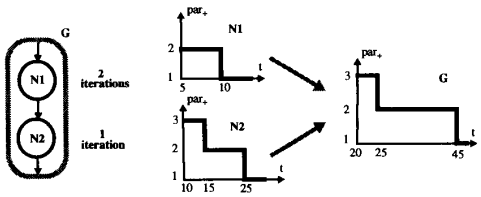

Fig. 11. Construction of hierarchical minibound-time graphs—simple example.

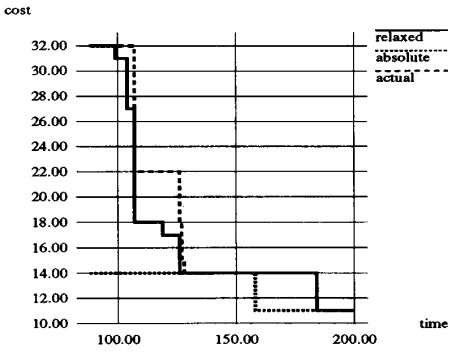

Fig. 12. Estimated cost of a 19th-order CORDIC algorithm (relaxed, absolute, and actual).

The procedure *Estimate Hierarchy* in fact introduced another form of relaxation: the actual time allotted to each sub-graph differs from resource to resource and is determined by (5). In reality, this is clearly not the case. This relaxation might therefore be overly optimistic, as it picks the optimal solution for each resource independent of the other resources. This will hurt especially when the *stress ratio* approaches 1. A more accurate solution can be obtained by considering the resources in a combined fashion. For instance, when considering two resources $r1$ and $r2$ simultaneously, (5) is reformulated as follows:

$$t^{r12}_{\min} = \sum_{i=1}^{N} \mathrm{MAX}((\mathrm{MB}_{r1}^{i^{-1}}(\eta_{r1}), \mathrm{MB}_{r2}^{i^{-1}}(\eta_{r2}))) \times \mathrm{iter}^i \tag{6}$$

This approach turns the hierarchical estimation problem from an R times 1-dimensional problem into an R-dimensional one. This might sound bad, but it isn't in reality. First of all, R is normally rather small (more than 6 different resources is rare). Secondly, only a couple of resources are critical and need more than one instance, which reduces the estimation space in an important way.

C. Min-Bounds on Interconnect and Registers

All the above described techniques can be applied in an identical fashion for the estimation of the lower bounds of interconnect. The only difference is that the routines are

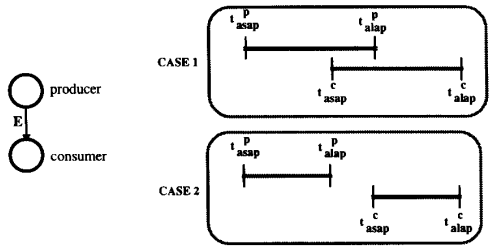

Fig. 13. Minimum variable lifetime of variable E: Possible scenarios.

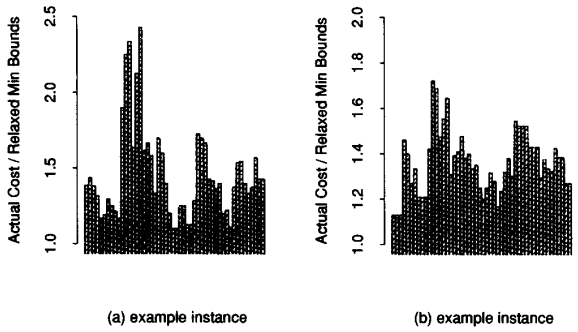

Fig. 14. Ratio between actual cost and min-bound for interconnect (a) and registers (b) (for 50 examples).

applied on the interconnect graph G_c instead of on the computation graph G (Fig. 6). As described above, the slack of an interconnect node is identical to the slack of the source computation node: in the hardware model H, a bus is directly connected to the output of the execution unit and is thus reserved for the duration of the computation.

Estimating the minimum bounds on the number of registers requires a somewhat different approach. In order to find the absolute minimum on the register count, we have to assume that no broadcasting occurs and that each variable is alive for the minimum possible amount of time. The minimum lifetime now depends upon the slack time of both the producer and consumer nodes. As illustrated in Fig. 13, two scenarios are possible:

1) The slack periods of the producer and consumer nodes overlap (or $t^p_{\text{alap}}\ t^c_{\text{asap}}$). In that case, the variable E can be consumed immediately after generation. The minimum lifetime is therefore t^c_{dr}. The interval where the variable can be alive stretches from t^c_{asap} to $\max(t^p_{\text{alap}}, t^c_{\text{alap}})$.
2) The slack periods of producer and consumer nodes do not overlap ($t^p_{\text{alap}} < t^c_{\text{asap}}$). Here, the variable E has to be minimally alive for a longer period, namely from t^p_{alap} to $t^c_{\text{asap}} + t^c_{\text{dr}}$. This is also the lifetime interval.

Estimating a lower bound on the number of registers associated with an execution unit r (hardware model H), can now be solved in terms of the following relaxed scheduling problem: Given a number of resources η_{reg} (the number of registers available) and two classes of tasks: all tasks of the first class have a fixed, integer duration (t_{dr}) and have integer ASAP and ALAP times; the tasks of the second class have a variable, integer duration but have a fixed scheduling time, determine the minimum time to execute the tasks on the given resources.

The scheduling of tasks with varying duration is in general NP-complete (in fact, strongly NP-complete) [55]. The fact however, that the tasks with varying duration are fixed in time, saves us here. The following modification to the earliest deadline algorithm can be used to approximate the solution:

1) Divide all times by the largest t_{dr} (as defined in the beginning of this subsection) and round all ASAP and ALAP times to respectively the next lower and upper integers. This is identical to the approach taken for execution units with duration larger than 1. This translates the problem into a scheduling problem with integer ASAP and ALAP times and task durations of 1.
2) Reserve the slots, needed by the tasks with variable duration (scenario 1), but fixed scheduling time.
3) Schedule the remaining tasks using the earliest deadline algorithm. For each time-slot, the number of available resources is Equal to η_{reg} minus the number of reserved slots. The complexity of the algorithm is still $O(N\ \log\ N)$.

The obtained bound is clearly on the pessimistic side, since the actual solution will contain broadcasting. It can also be observed that the actual lifetime of the variables will approach the average value between min and max, rather than the minimum value. A simple explanation of this is that in a typical schedule, decreasing the lifetime of one variable actually increases the lifetime of others. We are studying other techniques (statistical techniques, among others; see Section VII) to get more precise register bounds.

Finally, its should be mentioned that a similar approach can be used to estimate the register count for the hardware model $H*$. Since this model does not support broadcasting, the estimated register count will be closer to the actual solution.

The performances of the estimation routines for interconnect and registers were analyzed using the benchmark set. The results are plotted in Fig. 14(a) for interconnect and Fig. 14(b) for registers. The maximum discrepancy for interconnect is 142%, while the average and median discrepancy are 46% and 39%. The maximum discrepancy for registers is 72%, while both the average and median discrepancy are 39%. The largest discrepancy for interconnect occurs again when the stress ratio is close to 1. The explanation is the same as in the case for execution units. The discrepancy for the number of registers is much less dependent on the stress ratio.

Although the discrepancy is still relatively small, it is not as impressive as in the case of execution units. The higher discrepancy between the min-bounds and the actual cost for interconnect and registers is partially due to the fact that the used scheduler pays more attention to execution units than to the two other hardware components, because the execution elements have a higher implementation cost. We have already stated that the register optimization is also the more difficult problem.

D. Other Relaxation Approaches

The number of different relaxations that can be introduced to simplify the high-level synthesis scheduling problems is almost unlimited. A large number of scheduling problems are indeed known to be of a polynomial complexity. Of course, the most interesting ones are those that offer a good compromise between accuracy and run time. We will briefly discuss three other approaches: relaxing on the constraints that operations should be scheduled on integer times, ignoring all edges that violate certain graph properties, and ignoring the fact that the operation execution should be nonpreemptive.

The first approach (and also the most promising one) is based on the fact that the scheduling, resource allocation, and assignment processes can be formulated as integer programs [45]. Although integer program solvers take exponential time in the worst case (and are hence unapplicable for problems of large size), linear programs can be solved in polynomial time. Turning an integer program into a linear one corresponds to relaxing on the integer starting time of the operations. Although such a solution cannot be used in an actual design, it can be employed to provide an accurate lower bound on the execution time of a program, given the hardware resources (the dual estimation problem). It is important to note that in general, relaxation of integer constraints in integer programs may sometimes result in a solution with a cost value that is dramatically different from the real optima [42, Ch. II.1.1, in particular the example on p. 327] and therefore produce bounds that are too optimistic.

The second approach is based on a result shown by Hu [20], that the as soon as possible scheduling algorithm produces the optimal solution when the computational graph has an in-forest or out-forest structure. A graph can relaxed into this particular format by removing all precedence edges that violate this constraint. The quality of the prediction depends upon the number of edges deleted and even upon the choice of the edges (since many forest structures can be derived for a single problem).

The third approach is derived from a max-flow based algorithm for the optimal scheduling of J jobs on M machines, which was developed by Federgruen and Groenevelt [12]. They assume that tasks can be scheduled in discontinuous intervals, which is, of course, not the case in high-level synthesis. It is possible, however, that the solution of this relaxed problem may prove to generate accurate lower bounds.

Finally, it is interesting to note that sometimes the addition of extra constraints might result in a more tractable problem formulation. Leung [36] succeeded to optimally schedule a graph for which the execution times of the operations are restricted to at most k values. His algorithm, which uses a sophisticated dynamic programming approach, has a worst case run time of $O(n^{2(k-1)})$, where n is the number of nodes in the graph. Although it is an impressive algorithmic result, its actual application is limited to cases where k is small. Since this is often the case when the available time approaches the critical path, this technique can be used to produce sharper bounds for the estimation instances with large stress ratios.

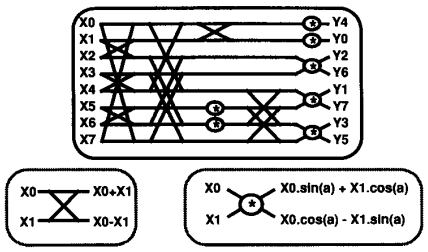

Fig. 15. Eight-point discrete cosine transform—computational graph.

VI. Applications of Estimations in High-Level Synthesis and in the Synthesis System

The proposed techniques have been implemented and incorporated into the HYPER synthesis system, which is targeted at the synthesis of high-performance, data-path-intensive real time applications [6], [52]. Within HYPER, the complexity estimation routines are used in numerous places, as will be demonstrated in this section. Complexity estimation, however, has a scope which rises beyond the confines of HYPER. Other applications for which the presented techniques could be useful are the areas of algorithm and design style selection, as well system partitioning. A number of those applications will be discussed with the aid of a simple example, an 8-point Discrete Cosine Transform (DCT). The DCT is used in virtually all video and image compression systems, such as those present in HDTV and tele-conferencing. One form of the DCT is shown in Fig. 15 [60]. The following paragraphs will discuss the applications of estimation in a top-down fashion.

A. Algorithm and Architecture Selection

Often, various computational algorithms are available to perform the same function. The optimality of an algorithm depends upon the required throughput rate, the available input-output and memory bandwidth, and the operation library available. The results of the estimation process can help to differentiate between different algorithms over a range of implementation constraints. For instance, while an FFT might require less multiplications than a DFT, the DFT can be advantageous when spectral information is only required for a limited frequency band or when memory is in short supply.

As another case consider the DCT example. An important part of the DCT is a rotation of the input data, requiring complex multiplications. It is well known [2] that the number of multiplications required for a complex multiplication with a constant can be reduced by a reorganization of the computation. In Fig. 16, it is shown that the reorganized computational graph reduces the number of multiplications by 1 at the expense of an extra adder and an increased critical path. In order to compare the two approaches, we have used the proposed estimation techniques to compute the implementation cost over the range of throughput speeds. For this and the following examples, the cost is computed as the sum of the minimum execution unit cost plus the minimal register cost. The properties of the module library used are shown in Table I. The results of the estimation process are shown

TABLE I
SAMPLE RESOURCE UTILIZATION TABLE

Block	Critical Path	Cycles	IO - >+	+ - > *	+	*
Subgraph$_1$	c_1	t_1	1	2	2	3
Subgraph$_2$	c_2	t_2	0	4	4	1
Subgraph$_3$	c_3	t_3	2	0	3	0
Total	$c=\Sigma c_i$	$t=\Sigma t_i$	2	4	4	3

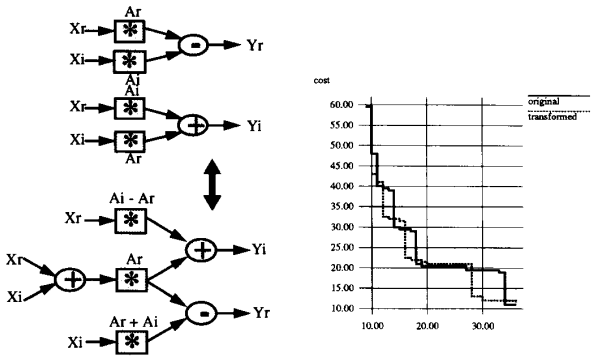

Fig. 16. DCT: Cost comparision for traditional versus reconfigured complex multiplictions.

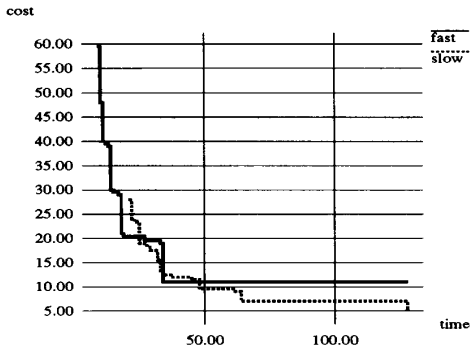

Fig. 17. DCT: Parallel versus serial multiplier.

TABLE II
SPEED AND COST PROPERTIES OF SIMPLE MODULE LIBRARY

	Speed	Cost
Add/Subtract	1	1
Parallel Multiplier	2	8
Serial Multiplier	8	2
Register	–	0.5

in Fig. 16. It is interesting to notice that neither algorithm is clearly better. Although the reorganized flow graph reduces the number of (expensive) parallel multiplications by 3, the increased critical path offsets this gain for certain throughputs. These observations are extremely hard to come by using just flow graph inspection.

Similarly, the estimation results can help to select between architectural styles, such as general purpose programmable versus custom programmable or hard-wired, time-shared interconnect versus a dedicated interconnect network or bit-serial versus bit-parallel. Different architectures can be compared by modifying both the hardware model and the available module set.

B. Module Selection

Accurate estimation can help to improve the quality of the module selection process, which selects between different alternative versions of an execution unit present in the hardware library. Traditional module selection tools [21] use the absolute min-bound to estimate the cost of a given selection. Chu [7] has demonstrated that the relaxed estimation can result in more optimal selections.

The detailed description of how HYPER uses the presented estimation techniques for the module selection problem is presented in [7]. We will limit our presentation here to illustration of how accurate estimations can help in the module selection process. Fig. 17 shows the effect on the implementation cost of the DCT for the cases where a slower parallel-serial multiplier is used and where a fully parallel multiplier is used. It can be observed that the serial-multiplier solution only becomes dominant for very large values of the available time. For very small times, the parallel solution is obviously the only choice. In the intermediate zone, both solutions are comparable, which is somewhat predictable, since both multipliers have identical Area-Speed products.

C. Transformations

Optimizing transformations are an essential component of any synthesis environment. Pipelining, retiming, arithmetic laws, and loop unrolling are typical examples of transformations that are often applied to improve the implementation quality. The main questions arising in a transformation environment are what transformation to apply when and what improvement can be expected? Once again, estimations can help substantially to resolve these questions. As an example of how estimations can aid the transformation process, consider the *retiming for resource utilization* transformation [48]. This transformation, based on a probabilistic iterative improvement approach, uses estimations to both determine the cost-function and hence the next move, as well as the optimal result and its distance from it. A detailed description of how several transformations for throughput, area, and power optimization are using estimations in HYPER can be found in [47], [4].

The HYPER strategy for the ordering of transformations involves using the Resource Utilization Table as the global view of the quality of the current flow graph and as a guideline of which transformations to apply next. The table is obtained from the estimation tool and lists the bounds on the resources over time. Since the flow graph is hierarchical, the table is also constructed in a hierarchical fashion: a subgraph is represented at the next higher hierarchy level by its global min and max bounds. A sample table is shown in Table II.

The overall design space search using transformations and estimations can now proceed as follows: First, the critical path is analyzed to check if the graph meets the performance requirements. If not, performance transformations (which target critical path reduction) are applied. After each transformation,

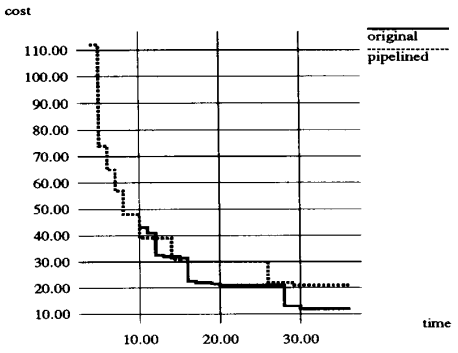

Fig. 18. Pipelined versus nonPipelined DCT cost.

the resulting flow graph is simulated and checked to assure the word-length requirements are met. Next, the resource allocation process is initiated [47]. The task of this process is to come up with a minimum hardware configuration that will meet the performance constraints.

The overall design space search process is executed under the control of the search mechanism (which can be directly influenced by the user or works using probabilistic search mechanism). The search mechanism forms the core of the system. Its task is to determine where to pipeline, which transformations to apply on which subgraph, and what resources to provide. The search is driven by information from the resource utilization table, as obtained by estimation routines. Most often, subgraphs that dictate values for various resources are targeted using transformations that relax demand on those resources. For example, if we want to reduce the number of multipliers used in the final implementation, it is obvious that we have to address the Subgraph 1, either by allocating more time for its execution or by invoking transformations that have the potential to reduce the high demand for multipliers. For a more detailed description of the transformation framework based on estimation, see [51].

The DCT example is used again to illustrate how estimations can be employed to select which transformations to apply in a given situation. We compared the original version (Fig. 15) with a pipelined version, where three pipeline stages were introduced to increase the throughput. The estimated cost of pipelined and nonpipelined versions is compared in Fig. 18. The results demonstrate that the pipelined approach achieves its goal of increasing the throughput, but also that over-pipelining hurts: at lower speeds, adding pipeline stages only adds extra registers.

D. Allocation, Assignment, and Scheduling

The derived minimum and maximum bounds delimit the design search space, thus speeding up the hardware allocation process. The minimum bounds can serve as an initial solution for the search. We have experienced that this solution is often very close to the final solution [49]. Information such as the distribution of the parallelism plots or the distance between the absolute and relaxed min-bounds can be used as a metrics to select an optimizing transformation [51].

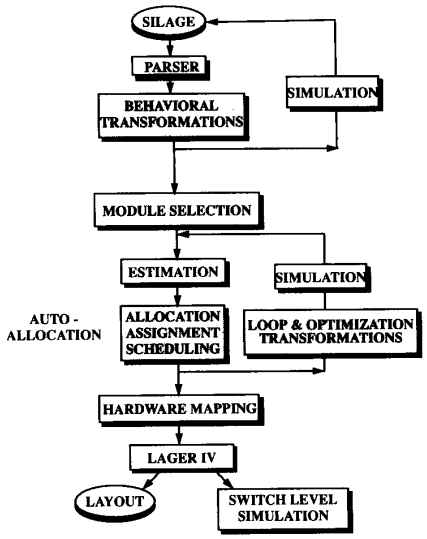

Fig. 19. The use of estimations in the standard design flow of HYPER.

When all automatic techniques fail, the direct feedback that the estimation information provides the designer can help to guide him or her through an interactive search process. Fig. 19 shows a screen dump of the HYPER user interface. The bounds are printed in the main window. The plots on the left are the parallelism plots for the execution units, registers, and interconnect, respectively. The example shown here is the 7th-order IIR filter discussed previously.

Estimation can also help to improve the quality of assignment and scheduling algorithms. For instance, the HYPER scheduler uses a relaxed scheduling as a heuristic to select a candidate node in a list scheduling process [49].

E. Synthesis Algorithm Validation

Most of the problems in high-level synthesis (such as allocation, assignment, and scheduling) have been proven to be at least NP-complete. As a result, it is hard to judge the quality of a proposed synthesis algorithm. Typically, benchmarks examples are used as a means to establish the performance of a technique. However, algorithms can be over-tuned to one particular benchmark ("the fifth order elliptical filter" syndrome). Furthermore, comparing algorithms against a particular benchmark establishes a relative measure, but it does not result in an absolute idea of the quality of the solution. Short of actually determining the optimal solution (which is virtually impossible for complex applications), the best approach is to estimate that optimal solution as accurately as possible. The proposed min-bound estimation techniques present a means to achieve this goal.

F. The Use of Estimations in the HYPER High-Level Synthesis System

As we have already outlined, estimations are an essential part of several HYPER high-level synthesis tools: allocation, assignment and scheduling, transformations, and module and

clock selection. Although the design flow in HYPER is not fixed and can be directly influenced by the designer, a predefined (standard) design flow is also available. The HYPER design flow is shown in Fig. 19. The shaded tasks directly use estimations of min and max bounds. Estimations routines are also used explicitly for providing feedback about design characteristics to the designer.

VII. Future Work

As demonstrated with the benchmark examples, the relaxed estimation techniques tend to be rather accurate. The average error is approximately 10%. Sometimes, the predicted bounds are too optimistic and estimation errors of up to 67% can be observed. This is typically the case when the available time approaches the critical path. Partly, this is a consEquence of graph properties becoming very constrained, making the relaxation approach overly simplistic. Getting more precise information by using deterministic approaches of greater complexity would, however, defeat the goal of the estimation approach, which is to get *accurate* results *fast*. This eliminates all approaches with a complexity larger than $O(N^2)$.

One way to get more precise results without extra computational complexity is to use a statistical approach. The results of the deterministic estimation process can be used as parameters in a statistical model, obtained through the analysis of a large number of examples, which span the complete design space. This approach does not result in bounds, but in an estimation of the location of the actual solution. Early experiments (over the same benchmark set as used above) indicate that an average error of 4.6% can be obtained with the maximum error equal to 18.7%. An extended research effort in this direction is currently under way.

We are also studying the use of the relaxation approach for the estimation of background memory bounds. These results will be extremely useful in the memory module selection, memory partitioning, and transformation for memory optimization processes.

Finally, the reader might have observed that the majority of the estimation techniques presented focus on the computational part of the implementation. This is justifiable for high-performance, real time applications where memory and data paths dominate the chip area. This is not the case, however, for control-dominated applications. Some results in this area have already been reported in [41].

VIII. Conclusion

A library of techniques to efficiently and accurately estimate the minimum and maximum bounds on the implementation cost of an application specific circuit have been presented. All algorithms have a complexity not larger than quadratic and the observed results on our benchmark set display an average error of approximately 10%.

We have demonstrated the application of the techniques in a variety of design synthesis areas such as design space exploration, transformation selection, and synthesis algorithm evaluation. It is the authors' belief that the major impact of estimation techniques will be in the higher levels of the synthesis process, such as algorithm and design style selection as well as design partitioning. It is our conviction that estimation will be one of the essential components in the system designer's tool-box.

Acknowledgment

The authors would like to thank Lisa Guerra for her helpful comments.

References

[1] U. Banerjee *et al.*, "Time and Parallel Processor Bounds for Fortran-like Loops," *IEEE Trans. Comput.*, vol. 28, no. 12, pp. 660–670, 1979.
[2] R. E. Blahut, *Fast Algorithms for Digital Signal Processing*. Reading, MA: Addison-Wesley Publishing Co., 1985.
[3] F. Brewer and D. D. Gajski, "Chippe: A System for Constraint driven Behavioral Synthesis," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 7, pp. 681–695, 1990.
[4] A. Chandrakasan, M. Potkonjak, J. Rabaey, and R. Brodersen, "HYPER-LP: A Design System for Power Minimization Using Architectural Transformations," in*Proc. IEEE Int'l Conf. on Computer-Aided Design*, Santa Clara, Nov. 1992, pp. 304–308.
[5] X. Chen and M. L. Bushnell, "A Module Area Estimator for VLSI Layout," in*Proc. 25th Design Automation Conf.*, Anaheim, June 1988, pp. 54–59.
[6] C. Chu *et al.*, "HYPER: An Interactive Synthesis Environment for High Performance Real Time Applications," in *Proc. IEEE Int'l Conf. on Computer Design*, Nov. 1989, Boston.
[7] C. Chu, "Hardware Mapping and Module Selection in the HYPER Synthesis System," Ph.D. Thesis, University of California, Berkeley, May 1992.
[8] R. J. Cloutier and D. E. Thomas, "The Combination of Scheduling, Allocation, and Mapping in a Single Algorithm," in *Proc. 27th ACM/IEEE Design Automation Conf.*, June 1990, Orlando, FL, pp. 71–76.
[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. Cambridge, MA: MIT Press; New York: McGraw-Hill, 1990.
[10] M. Davio, J.-P. Deschamps, and A. Thayse, *Digital Systems with Algorithm Implementation*, John Wiley & Sons, 1983.
[11] F. Depuydt, G. Goossens and H. De Man: "Clustering Technique for Register Optimization during Scheduling Preprocessing," *Proc. IEEE International Conference on Computer-Aided Design*, Santa Clara, pp. 281–284, November 1991.
[12] A. Federgruen and H. Gronewelt, "Preemptive scheduling on uniform machines by ordinary flow techniques," *Management Sci.*, vol. 32, pp. 341–349.
[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W. H. Freeman and Co., 1979.
[14] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*. New York: Academic Press, 1980.
[15] E. Gyrczyc, "Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Description," Ph.D. Thesis, Carleton Univ., 1984.
[16] J. W. Hagerman, "A Fast and Accurate Technique for Function Unit Allocation Estimation," Tech. Rep. CMUCAD-91-28, Carnegie Mellon Univ., April 1991.
[17] D. L. Hanson, "Interconnection Analysis," in *Physical Design Automation of Electronic Systems*, B. T. Preas and M. J. Lorenzetti, Eds., pp. 31–64, 1988.
[18] P. Hiedelberger and S. Lavenberg, "Computer Performance Evaluation Methodology," *IEEE Trans. Comput.*, vol. 33, no. 12, pp. 1195–1220, 1984.
[19] D. S. Hochbaum and D. B. Shmoys, "Using Dual Approximation Algorithms for Scheduling Problems: Theoretical & Practical Research," *J.ACM*, vol. 34, no. 1, pp. 144–162, 1987.
[20] T. C. Hu, "Parallel Sequencing and Assembly Line Problem," *Operations Research.*, vol. 9, pp. 840–844, 1961.
[21] R. Jain *et al.*, "Module Selection for Pipelined Synthesis," in *Proc. 25th ACM/IEEE Design Automation Conf.*, June 1988, Anaheim, CA, pp. 542–547.
[22] R. Jain, "High-Level Area-Delay Prediction with Application to Behavioral Synthesis," Tech. Rep. 89–23, Univ. of Southern California, 1989.

[23] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for experimental design, measurement, simulation, and modeling*. New York: John Wiley & Sons, 1991.
[24] N. Jouppi and D. Wall, "Available Instruction-Level Parallelism for Super-Scalar and Super-Pipelined Machines," in *Proc. 3rd Int'l Conf. Architectural Support for Programming Languages and Operating Sys.*, Boston, MA, May 1989, pp. 272–28.
[25] D. W. Knapp, "Feedback-Driven Datapath Optimization in Fasolt," *Proc. IEEE Int'l Conf. on Computer-Aided Design*, Nov. 1990, Santa Clara, CA, pp. 300–303.
[26] D. W. Knapp and A. C. Parker, "The ADAM Design Planning Engine," *IEEE Trans. Computer-Aided Design*, pp. 829–846, 1991.
[27] D. W. Knapp, "Fasolt: A Program for Feedback-Driven Data-Path Optimization," *IEEE Trans. Computer-Aided Design*, vol. 11, no. 6, pp. 677–695, 1992.
[28] D. Kuck, Y. Muraoka, and S. Chen, "On the Number of Operations Simultaneously Executable in Fortranlike Programs and their Resulting Speed-up," *IEEE Trans. Comput.*, vol. 21, no. 12, pp. 1293–1310, 1972.
[29] K. Kucukcakar and A. C. Parke, "BAD: Behavioral Area-Delay Predictor," Tech. Rep. 90–31, Univ. of Southern California, 1990.
[30] K. Kucukcakar and A. C. Parker, "CHOP: A Constraint-Driven System-Level Partitioner," in*Proc. 28th ACM/IEEE Design Automation Conf.*, June 1991, San Francisco, CA, pp. 514–519.
[31] S. Y. Kung, *VLSI Array Processors*. Englewood Cliffs, NJ: Prentice-Hall, 1988.
[32] F. J. Kurdahi and A. C. Parker, "PLEST: A Program for Area Estimation of VLSI Integrated Circuits," in *Proc. 23rd ACM/IEEE Design Automation Conf.*, June 1986, Las Vegas, NV, pp. 467–473.
[33] F. J. Kurdahi, "Area Estimation of VLSI Circuits," Ph.D. Thesis, Univ. of Southern California, 1987.
[34] F. Kurdahi and A. Parker, "Technique for Area Estimation of VLSI Layouts," *IEEE Trans. Computer-Aided Design*, vol. 9, no. 9, pp. 938–950, 1990.
[35] F. J. Kurdahi and C. Ramachandran, "LAST: Layout Area and Shape Function EsTimator," in *Proc. 1st European Design Automation Conf.*, Feb. 1991, pp. 351–355.
[36] J. Y-T. Leung, "On Scheduling Independent Task with Restricted Execution Times," *Operations Research*, pp. 163–171, 1982.
[37] M. Lightner and W. Wolf, "Experiments in logic optimization" in *Proc. IEEE Int'l Conf. on Computer-Aided Design*, Nov. 1988, Santa Clara, CA, pp. 286–289.
[38] M. C. McFarland, "Reevaluating the design space for register-transfer hardware synthesis," in *Proc. IEEE Int'l Conf. on Computer-Aided Design*, Nov. 1987, Santa Clara, CA, pp. 262–265.
[39] M. C. McFarland and T. J. Kowalski, "Incorporating Bottom-Up Design into Hardware Synthesis," *IEEE Trans. Computer-Aided Design*, 1990, pp. 938–950, vol. 9. no. 9.
[40] C. Mead and L. Carver, *Introduction to VLSI Systems*. Reading, MA: Addison-Wesley, 1980.
[41] M. J. Mlinar, "Control Path/Data Path Tradeoffs in VLSI Design," Tech. Rep. 91–16, Univ. of Southern California, 1991.
[42] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*. New York: John Wiley & Sons, 1988.
[43] A. Nicolau and J. Fischer, "Measuring the Parallelism Available for Very Long Instruction Word Architecture," *IEEE Trans. Comput.*, vol. 33, no. 11, pp. 968–976, 1984.
[44] P. G. Paulin and J. P. Knight: "Force-Directed Scheduling for the Behavioral Synthesis of ASIC," *IEEE Trans. Computer-Aided Design*, vol. 8. no. 6, pp. 661–679, 1989.
[45] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[46] M. Pedram and B. Preas, "Accurate Prediction of Physical Design Characteristic for Random Logic," in *Proc. IEEE ICCD Conf.*, 1989, Boston, MA, pp. 100–108.
[47] M. Potkonjak, "Algorithms for High Level Synthesis Resource Utilization Based Approach," Ph.D. Thesis, Univ. of California at Berkeley, 1991 .
[48] M. Potkonjak and J. Rabaey, "Optimizing the Resource Utilization Using Transformations," in *Proc. IEEE Int'l Conf. on Computer-Aided Design*, Nov. 1991, Santa Clara, CA, pp. 88–91.
[49] M. Potkonjak and J. Rabaey, "Scheduling Algorithms for Hierarchical Data Control Flow Graphs," to be published in the Special Issue on "Fundamental Methods in CAD," *Int'l J. Circuit Theory and Appl.*
[50] S. R. Powell and P. M. Chau, "Estimating Power Dissipation of VLSI Signal Processing Chips: The PFA Technique," in *VLSI Signal Processing IV*. New York: IEEE Press, H. S. Moscovitz, K. Yao, and R. Jain, Eds., pp. 250–259, 1990.
[51] J. Rabaey and M. Potkonjak, "Resource Driven Synthesis in the HYPER System," in *Proc. IEEE Int'l Symposium on Circuits and Systems 1990*, May 1990, New Orleans, LA, pp. 2592–2595.
[52] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast Prototyping of Data Path Intensive Architectures," *IEEE Design & Test Mag.*, pp. 40–51, June 1991.
[53] S. Sastry and A. C. Parker, "Stochastic Models for Wireability Analysis of Gate Arrays," *IEEE Trans. Computer-Aided Design*, , vol. 5, no. 1, pp. 52–65, 1985.
[54] C. Shung *et al.*, "An Integrated CAD System for Algorithmic Specific IC Design," *IEEE J. Computer Aided Design*, vol. 10, no. 4, pp. 447–463, April 1991.
[55] B. Simons, "On Scheduling with Release Times and Deadlines," in *Deterministic and Stochastic Scheduling*. M. Demster, Ed., Dordrecht: D. Reidel, pp. 75–88, 1981.
[56] M. Smith, M. Johnson, and M. Horowitz, "Limits on Multiple Instruction Issues," in *Proc. 3rd International Conference on Architectural Support for Programming Languages and Operating Systems, Boston*, pp. 290–302, May 1989.
[57] A. Stoelzle *et al.*: "A Flexible VLSI 60,000 Word Real Time Continuous Speech Recognition System," in *Proc. IEEE Workshop on VLSI Signal Processing*, Nov. 1990, pp. 247–284.
[58] K. Ueda, H. Kitazawa, and I. Harada, "CHAMP: Chip Floor Plan for Hierarchical VLSI Layout," *IEEE Trans. Computer-Aided Design*, , vol. 4, no. 1, pp. 12–22, 1985.
[59] J. Ullman, *Computational Aspects of VLSI*. Rockville, MD: Computer Science Press, 1984.
[60] M. Vetterli and A. Lichtenberg, "A Discrete Fourier-Cosine Transform Chip," *IEEE J. Selected Areas Commun.*, vol. 4, no. 1, pp. 49–61, Jan. 1986.
[61] J. Weng and A. C. Parker, "3D Scheduling" in *Proc. 28th ACM/IEEE Design Automation Conf.*, San Francisco, CA, pp. 668–673, June 1991.
[62] A. C-H. Wu, V. Chaiyakul, and D. D. Gajski, "Layout-Area Models for High-Level Synthesis," in *Proc. IEEE Int'l Conf. Computer-Aided Design*, Nov. 1991., Santa Clara, CA, pp. 34–37.
[63] G. Zimmerman, "A New Area and Shape Function Estimation Technique for VLSI Layouts," in *Proc. 25th ACM/IEEE Design Automation Conf.*, Anaheim, CA, pp. 60–65, June 1988.

Jan M. Rabaey, photograph and biography not available at the time of publication.

Miodrag Potkonjak, photograph and biography not available at the time of publication.