A Methodology and Algorithms for the Design of Hard Real-Time Multitasking ASICs

MIODRAG POTKONJAK
University of California
and
WAYNE WOLF
Princeton University

Traditional high-level synthesis concentrates on the implementation of a single task (e.g. filter, linear controller, A/D converter). However, many applications—multifunctional embedded controllers, intelligent wireless end-points, and DSP and multimedia servers—are defined as sets of several computational tasks. This paper describes new techniques for the synthesis of ASIC implementations that realize multiple computational processes under hard real-time constraints. Our synthesis methodology establishes connections between two important computer engineering domains: operating systems and behavioral synthesis. Our hierarchical approach starts from an incompletely-specified preliminary solution and uses, interchangeably, operating system and behavioral synthesis techniques to derive increasingly more detailed and accurate design solutions. We have experimented with both optimal and heuristic algorithms to implement this methodology. The optimal algorithm uses several heuristics to speed up the average run time of an exhaustive branch-and-bound search. Force-directed optimization is the core of the heuristic synthesis method. Analysis of the proposed algorithms and the experiments shows that matching the number of bits and type of operations in tasks assigned to the same application-specific processor was the most important factor in obtaining area-efficient designs.

Categories and Subject Descriptors: B.7.1 [**Integrated Circuits**]: Types and Design Styles—*algorithms implemented in hardware*

General Terms: Algorithms, Design, Performance

W. Wolf was supported by the National Science Foundation under grant MIP-9424410.
Authors' addresses: M. Potkonjak, Dept. of Computer Science, University of California, Los Angeles, CA 90095; W. Wolf, Dept. of Electrical Engineering, Princeton University, Princeton, NJ 08544.

1. INTRODUCTION

1.1 Motivation

Traditionally, high-level synthesis has concentrated on the synthesis of machines that implement a single computational process. While early behavioral synthesis techniques concentrated on computations without control structures such as loops, conditionals, and subroutines [McFarland et al. 1990], more recently a number of high-level synthesis systems targeted synthesis of hierarchical computations (with loops and subroutines) and stressed the importance of the associated global optimization techniques across the lowest levels of the hierarchy [Walker and Camposano 1991].

The trend in high-level synthesis research is to consider higher levels of abstraction and larger chunks of a system design; this trend continues unabated, thanks to both application and implementation factors. The size of average embedded and DSP applications has been approximately doubling each year [Gibbs 1994]. While the number of ASIC designs slightly decreases each year, both their size and volume have increased rapidly. For example, the average size, in terms of number of gates, of integrated circuits has been doubling every three years, while the clock period has been steadily decreasing at a just slightly lower pace. For example, the state of the art general-purpose processor in 1971 (Intel 4004) had 2,300 transistors and had clock rate 0.1 MHz. In 1993, a latest-generation processor, the Intel Pentium, had 3 million transistors and a clock rate of 66 MHz [Minoli and Keinath 1994, p. 24]. These trends imply that, in each new generation of technology, higher levels of hardware sharing are required, feasible, and economically desirable. As we move toward systems-on-a-chip, a natural extension of behavioral synthesis is to the next level of the hierarchy of computation: this time to synthesis which conducts global optimization across boundaries of individual tasks.

The increase in design complexity requires not only quantitative changes in the effectiveness of synthesis algorithms, but also qualitatively new design approaches. This is well illustrated in several related computer engineering domains. For example, rapidly increasing memory needs did not result in just larger SRAM and DRAM modules and chips, but also in several levels of hierarchy (e.g., registers in a datapath, three levels of caches, main memory, disk, and tape) in modern memory design of both general-purpose and application-specific systems [Patterson and Hennessy 1990]. To adequately address qualitative new aspects of design, new design methodologies are needed. In this paper we introduce a methodology that considers both intra- and inter-hardware and timing requirements of several hard real-time tasks to produce globally optimized implementations.

Table I shows a set of avionics tasks used for the development of an avionics application specific computer [Ratner and Shapiro 1973; Bannister and Trivedi 1983]. Bannister and Trivedi [1983] proposed three different

Table I. An Example of Multitask Application-Specific Systems: Avionics Tasks Characteristics [Ratner and Shapiro 1973; Bannister and Trivedi 1983]

Task no.	Task description	Iterations per second	Instructions per iteration
1.	Attitude control	20	1,228
2.	Flutter control	250	138
3.	Gust control	240	58
4.	Autoland	160	342
5.	Autopilot	5	200
6.	Attitude detector	30	2,560
7.	Inertial navigation	25	1,350
8.	VOR/DME	5	770
9.	Omega	5	800
10.	Air data	5	200
11.	Signal processing	0.2	1,750
12.	Flight data	5	5,520
13.	Airspeed	16	549
14.	Graphics display	8	3,975
15.	Text display	10	1,900
16.	Collision avoidance	670	32
17.	Onboard communication	250	28
18.	Offboard communication	4	155
19.	Data integration	4	360
20.	Instrumentation	5	2,792
21.	System management	0.5	2,320
22.	Life support	0.5	2,320
23.	Engine control	33	3,597
24.	Executive	5	200

implementation solutions. The smallest implementation uses six processors, the largest eight processors. Several other segments of the automotive industry have similar needs for the synthesis of application-specific machines. The products in emerging and rapidly growing markets for multimedia and video games are also often multitask multiprocessor application-specific systems [Huang et al. 1996; Borel 1997; Brodersen 1997; Yasuda 1997].

1.2 The Synthesis Problem

We focus on exploring synthesis and optimization issues during behavioral synthesis of application-specific systems at the task hardware-sharing level. By using a combination of information provided by *hard real-time scheduling* methodologies and tools and by *behavioral synthesis* tools, we connect the synthesis process to operating systems methodologies and technologies and *enable efficient sharing of application-specific hardware by several tasks*.

In particular, we address common hardware design problems for systems of processes with deadlines specified as a set of periodic *tasks*. We use the term "process" in the conventional operating systems sense—a single instance of a program, representing a single flow of control and a set of input data values. However, our implementation of each process will be a

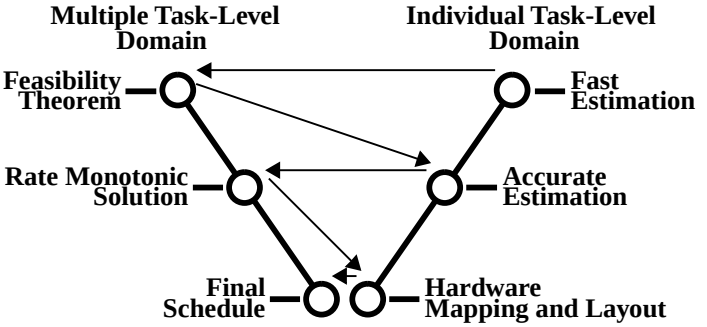

Fig. 1. V-Chart of the new multiresolution synthesis approach.

special-purpose ASIC with hardwired control logic and a custom datapath. Each process is defined by a control data-flow graph (CDFG).

A task is a set of processes: $\tau_i = \{P_{i1}, P_{i2}, \ldots\}$. Without loss of generality, we assume that there are no data, control, or timing dependencies among the tasks. If two tasks have dependencies, they are merged into a new atomic task with joint timing constraints.The set of all tasks that must be executed by the system is called the task set. For each task, three timing constraints are imposed: the *period*, the *start time* (the earliest time when all required data for one iteration is available), and the *finish time* (the latest time by which a task has to be completed, i.e., output data for a given iteration provided to the user). All the processes in a task share the identical periodic timing constraints. It is assumed in the rest of the paper that, as regularly done in hard real-time scheduling literature and practice, and often required in real-life embedded and communication designs, the finish time for each task is the end of its period. The system has to be designed so that all functionality and timing requirements are satisfied while the cost of implementation is minimized.

Processes from different tasks may be implemented in the ASIC on a single datapath. We call a set of processes implemented on a single datapath a process set. The synthesis goal is to partition the set of processes in an arbitrary number of process sets so that each can be implemented on one dedicated multifunctional ASIC design. All timing constraints for all tasks on all chips must be satisfied. The partitioning is conducted in such a way that the cost (sum of the areas of all resulting chips) is minimized.

1.3 Hierarchical, Iterative Refinement Synthesis Approach

We have developed a synthesis approach for hard real-time system design based on the strategy shown in Figure 1. The detailed, algorithmic description of all phases of the approach and complete explanation of Figure 1 are given in Sections 4, 5, and 6.

There are several reasons why partitioning of multiple task application-specific system is often the best—or even only—alternative. Sometimes

applications have very different characteristics and executing them on the same datapath inevitably leads to inefficient implementation. For example, if one has to simultaneously execute both a 64-bit graphics task and a 1-bit encryption task, executing the latter on wide datapath required by the former application is very inefficient. Also, experimental studies indicate that when two or more applications are merged on single datapath, the resulting processor may have significantly higher area requirements than the sum of areas of the individual implementations [Kim et al. 1997]. Finally, it is often the case that one cannot use a single processor for the realization of all tasks due to the intrinsic timing constraints. For example, if the sum of critical paths of the individual task is longer than the total available time, the only implementation option is a multiprocessor platform.

Several of the subproblems of the overall synthesis problem that must be solved during synthesis are NP-complete or of even higher computational complexity (see Section 3). In addition to this layered structure of computationally intractable problems, one has to simultaneously consider numerous constraints and the cost components of the solution for individual tasks as well as interaction between tasks due to hardware sharing.

The main conceptual and technical difficulty of the synthesis problem is related to the strong interdependence between the design optimization at the task-level and the optimization of individual tasks. Solving the task assignment problem requires accurate information about the hardware and timing requirements of the individual tasks. However, that timing information depends in turn on the partitioning (assignment) of the tasks. In order to break this circular dependency, we developed a new, multiresolution iterative refinement synthesis strategy. The method refines an initial solution by employing increasingly more accurate and CPU-intensive lower-bound estimations at the task level and for individual processes (tasks). The estimates at both levels are obtained by relaxing some of the constraints of the targeted problem, so that lower-bound estimates or partial solutions can be obtained very fast (low polynomial time). At the task level, estimates are generated by rate-monotonic scheduling [Liu and Layland 1973] providing mechanisms that guarantee that all the outlined timing constraints are simultaneously satisfied by examining the timing constraints for the individual processes, under the simplifying assumption of no time and hardware overhead for context switching between processes. For individual processes, timing and hardware requirements are estimated by the discrete-relaxation-based high-level synthesis tools, Hyper and Hyper-LP [Rabaey and Potkonjak 1994; Chandrakasan et al. 1995]. These tools consider one type of hardware resource at a time to obtain sharp lower bounds on the required timing and hardware resources for the individual processes. As soon as one of the estimates indicates that the proposed solution is unfeasible, the search strategy redirects further efforts.

Based on this methodology, we have developed both an optimal but worst-case exponential run-time optimization algorithm as well as a fast, heuristic synthesis program. In the former case, the search is organized

using a branch-and-bound methodology. The search is first conducted among the most promising design solutions. If an inevitable violation of timing constraints or inferiority of new solutions is detected, further search along this direction in the solution space is terminated. A new current best solution is accepted only after its area is obtained using the Hyper high-level synthesis system [Rabaey et al. 1991] and at the task level, a non-preemptive schedule is generated using the task-level scheduling algorithm described in Section 4.

Our heuristic algorithm, described in detail in Section 6, partitions the tasks into a number of groups. Note that in the rest of the paper, with the exception of the related work section, we use the word "partitioning" to denote that different subsets of tasks are implemented on different chips. Each process is itself treated as an atomic entity and is realized on a single chip that, in general, can also be used for realization of other processes. The search strategy is initially guided by a simple and fast estimation procedure that predicts the required hardware and corresponding time resources for each process assigned to each of the partitions. Once the proposed set of processes for one chip is available, the required composite hardware and individual time resources are estimated using more elaborate and accurate estimation tools. Next, estimates for the required time (number of control steps) of all processes targeted for implementation of one platform, are analyzed using hard real-time scheduling policies. In particular, the likelihood of successful scheduling for tasks is estimated using the critical zone theorem [Liu and Layland 1973; Lehoczky et al. 1989] (see Section 2). If the analysis implies a feasible solution, the designs are implemented by the proposed hardware using the Hyper high-level synthesis system. After this step, the realization for each process under the exact time requirements obtained during estimation is obtained. During implementation the available time for each process is prolonged by 5% if required to keep implementation smaller. If there is a change in timing requirements for any of the processes, overall timing constraints are checked for compliance with the real-time constraints using the hard real-time tools provided by the rate-monotonic scheduling. The final schedule is derived using the critical zone theorem. Matching the number of bits used by operations (tasks) that share the common hardware platform is critical to successful partitioning. The key technical optimization novelty is a multidimensional force-directed algorithm which rapidly searches the large solution space. The effectiveness of the approach is experimentally verified and the results presented and analyzed in Section 7.

1.4 Contributions

Our methodology and algorithms can be viewed from several perspectives. From a perspective of behavioral synthesis, this is the first effort, to the best of our knowledge, that considers and explores task-level concurrency and time-sharing in high-level synthesis. This consideration results in the formulation and solution of a novel synthesis process of selecting subsets from a set of several applications sharing common hardware resources.

From a real-time scheduling and system design viewpoint, this is also the first approach for optimizing cost of implementation in real-time systems. From design methodology and algorithmic optimization points of view, the four main contributions are as follows:

(1) a new multiresolution iterative refinement synthesis methodology that provides a clean and effective mechanism for simultaneously addressing all important technical details and establishing and maintaining global information about the targeted designs;

(2) establishment of the computational complexity of a class of synthesis problems;

(3) a novel optimal branch-and-bound approach for solving the optimization problem; and

(4) a simple and fast, yet effective, force-directed heuristic optimization approach that simultaneously considers several design metrics (e.g., word length, number of execution units of a particular type, number of registers).

Finally, this is the first project that builds connections between two computer engineering domains, real-time systems, and behavioral synthesis, by building connections from hard real-time system scheduling disciplines to high-level synthesis and, more importantly, to ASIC implementation technology. This connection also bridges the gap between emerging system-level synthesis and existing high-level synthesis techniques and tools.

1.5 Paper Organization

In the next section we outline background material, including selected computational and hardware models with key results provided by hard real-time process-level sequencing, and in particular, by rate-monotonic scheduling, research, and use of high-level synthesis and estimation techniques and tools. In Section 3 we explain and formulate the targeted synthesis problem and establish its computational complexity. Section 4 introduces the new two-domain multiresolution iterative refinement design methodology which is used in the two subsequent sections to develop an optimal, branch-and-bound worst-case exponential complexity optimization algorithm and a force-directed heuristic optimization algorithm. Section 7 presents experimental results. We have deferred the comparison of our work until Section 8, since our problem makes use of results from several different disciplines. We conclude the paper by outlining future research and development efforts and summarizing key results.

2. PRELIMINARIES

2.1 Computational and Hardware Models and Timing Constraints

Our task-level computational model assumes that each task is periodic, that the input/output sampling rate is specified, and that the deadline for

each task is the end of its sampling period. It also assumes a single thread of control for each of final implementations, and no preemption. The two final assumptions are postulated in order to avoid often difficult to model and high context-switching costs.

Our computational model for a single process is homogenous synchronous data flow [Lee and Messerschmitt 1987]. This model is widely used in many application domains (DSP, video and image processing, communications, control, information theory applications) and is also widely used for modeling individual tasks in hard real-time scheduling. These computational models assume a seminfinite stream of input data that arrives periodically. The design of the ASIC that executes these processes influences both the sampling period that can be supported for a process and the number of processes that can be supported. We assume that processes have bounded execution times. As a result, the processes can be statically scheduled at synthesis time. Static scheduling allows more aggressive and accurate optimization during synthesis. Both the generic synthesis approach and the optimal and heuristic algorithms can be easily adapted to numerous computational models which are suitable for static compilation, and, with some additional augmentation, to other computational models.

The computation performed by a process is defined as a hierarchical control data-flow graph. The CDFG represents the computation of a process as a flow graph, with nodes, data edges, and control edges. The nodes at the lowest level of hierarchy denote data operations, where the data edges represent data precedences. The control edges enforce an additional timing relationship among nodes. At higher levels of hierarchy, nodes encapsulate lower-level CDFG flow into procedures, loops, and if-then-else statements. The semantics underlying the syntax of the CDFG format, as we already stated, is of a synchronous data flow model of computation. We assume that it is possible to derive upper bounds on the execution of each process on the available hardware, for example, that the CDFG contains no indefinite loops. For the great majority of DSP, control, and other application areas this assumption is usually satisfied.

We do not impose any restriction on the assumed hardware model. Our synthesis experiments were performed with the Hyper high-level synthesis system from University of California at Berkeley [Rabaey et al. 1991] which targets a dedicated register file model. In this register-level hardware model all registers are grouped in relatively few register files. While each register file can send data to at most only one execution unit, there is no restriction on the number of register files to which any of the execution units can send data. While almost all industrial-strength designs use some version of register file models [Patterson and Hennessy 1990], the dedicated register file model is particularly popular in ASIC designs where it often provides an efficient way for significant reduction in dominating interconnect area at usually nominal increase in the area of registers. The Hyper behavioral synthesis system provides a mechanism for merging of dedicated interconnections into buses, which we invoked during our experimentations in order to further reduce the area of the designs.

Although in our implementation of the algorithms described in this paper we followed the dedicated register file hardware model, all our observations, algorithms, and software implementations can be directly used in conjunction with high-level synthesis systems that use a different hardware model. Our hardware cost model is complete in the sense that it accounts for all seven atomic register transfer (RT)-level components of a datapath: execution units, registers, background memory, interconnect, control logic, clock distribution wires, and the number of I/O pins.

2.2 Task-Level Scheduling and Estimation Techniques: Earliest Deadline First, Rate-Monotonic, and Deadline Monotonic Scheduling

This section introduces hard real-time scheduling techniques that we use for synthesis of hard real-time multitask ASICs. The backbone of all estimation techniques is the rate-monotonic scheduling algorithm [Liu and Layland 1973]. Real-time scheduling theory addresses the problem of ensuring that independent periodic processes are scheduled without violation of the associated timing constraints. Processes are independent if their correct execution does not imply a need for synchronization. Note that processes can be independent even when they interchange data between them or have a set of imposed control and timing constraints [Sha and Goodenough 1990].

Stankovic et al. [1995] provided an excellent survey of a number of real-time scheduling algorithms, their computational complexity, assumed timing and hardware models. We review a few results of direct importance to our synthesis methodology. One of the first proposed real-time scheduling algorithms is the earliest due-date, Jackson's rule [Jackson 1969]. Jackson's rule addresses the problem in which n independent processes, each with a specified processing time and due date, have to be scheduled on a single machine. It states that the optimal solution, if preemption is allowed, is to schedule the processes in non-decreasing due-date order. Unfortunately, neither Jackson's rule nor any other known algorithm can optimally solve the same problem if preemption is not allowed, since the associated optimization problem is NP-complete [Stankovic et al. 1995].

Rate-monotonic scheduling addresses the problem of scheduling a set of periodic processes. It belongs to the class of static priority-driven preemptive approaches where no explicit scheduling is provided, but instead the process with the current highest priority among the available processes is always executed [Ramamritham and Stankovic 1994]. The rate-monotonic algorithm assigns each periodic process a single static priority level that is inversely proportional to its period. Therefore, processes with shorter periods get higher priority, regardless of their actual relative computational requirements. The algorithm assumes that finish times are exactly the end of corresponding time intervals in which the process has to be scheduled. Under this restriction, which is common in many applications, the rate-monotonic scheduling algorithm is optimal in the sense that if any static priority algorithm is capable of scheduling a particular set of periodic

Table II. Efficiency Bounds for Resource Utilization During Application of Rate-Monotonic Scheduling When a Number of Different Processes Are Executed on a Single Implementation Platform

No. of processes	1	2	3	4	5	6	7	8	9
Efficiency bound, U(n)	1.0	0.828	0.779	0.756	0.743	0.734	0.728	0.724	0.720

processes, the rate-monotonic scheduling algorithm will also provide a feasible schedule. Rate-monotonic scheduling has been generalized in several directions. For example, the deadline-monotonic algorithm [Leung and Whitehead 1982] is often the algorithm of choice when the tasks do not have deadline times identical to their periods.

While the final high-level synthesis steps required for synthesis of the ASIC must base performance tradeoffs on detailed models that violate some of the assumptions of the real-time scheduling techniques, real-time scheduling is a very important estimation tool. Real-time scheduling provides hard bounds on the feasible design space—if a set of processes cannot be scheduled when preemption is allowed, it will be impossible to schedule them under stricter conditions when preemption cannot be used. Rate-monotonic scheduling's requirement that all processes have fixed priority increases (though it does not guarantee) the likelihood that a feasible non-preemptive schedule can be generated. This observation is one of the starting points for our approach to synthesis of area-efficient hard real-time systems.

The task-level performance estimation techniques used in our methodology are based on the following two theorems related to a particular scheduling discipline—rate-monotonic scheduling [Liu and Layland 1973; Sha and Goodenough 1990], which we use to develop task-level timing estimation techniques.

FEASIBILITY THEOREM [LIU AND LAYLAND 1973]. *A set of n independent periodic processes scheduled by the rate-monotonic algorithm will always meet its deadlines, for all task phasings, if*

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq n(2^{1/n} - 1) = U(n)$$

where C_i and T_i are the execution time and period of task i respectively.

CRITICAL ZONE THEOREM [LIU AND LAYLAND 1973]. *For a set of independent periodic processes, if each task meets its first deadline when all processes are started at the same time, then the deadline will always be met for any combination of start times.*

The feasibility theorem provides a sufficient worst-case condition that guarantees a feasible schedule of a task set when the rate-monotonic scheduling policy is applied. As the number of processes approaches infinity, the bound converges to 0.69 (ln 2). Data presented in Table II illus-

Table III. Efficiency Bounds for Resource Utilization During Application of Rate-Monotonic Scheduling When a Number of Different Processes Are Executed on a Single Implementation Platform

Task	Task duration (C)	Task period (T)	Utilization (U = C/T)
task 1	40	160	0.25
task 2	30	60	0.2
task 3	30	150	0.2
task 4	10	100	0.1

trates the feasibility theorem when the number of processes varies from one to nine. Essentially, the feasibility theorem indicates that if one allocates enough hardware resources so that the run time of the periodic processes satisfies the condition of the theorem, the feasible schedule is guaranteed, if static priority is assigned to all processes and preemption is allowed.

We illustrate the feasibility theorem using the following example: Suppose that four processes are given with parameters as stated in Table III. Four processes are given and they can be scheduled on one processor, regardless of their start times, because the sum of utilization bound is 0.75 ($<$0.756). However, if task 4 is replaced with a task that has a duration of 30 and a task period of 200, there is no guarantee that the feasible schedule exists, because the sum of efficiency bounds is now 0.8 ($>$ 0.756). Note that this does not imply that the feasible schedule does not exist. The bound only indicates that the likelihood of generating a feasible schedule is relatively low.

The feasibility theorem is a basis for performance estimation. It provides a bound for the non-preemptive schedules by removing non-preemption constraints. It also provides the mechanisms that guarantee a feasible solution, once conditions are satisfied.

It is important to note that the bound presented in the theorem is a conservative one. For example, it has been proven that for a randomly selected set of processes the likely bound for a very large number of processes is 0.88 [Lehoczky et al. 1989]. To enable a higher probability of accurately predicting the realistic resource utilization, we multiplied each value in Table III during checking process by 1.14 (1/.88), except, of course, for the single processor case.

The critical zone theorem can be restated using an equivalent mathematical test that is often more suitable for implementation than the criteria outlined in the initial version of the theorem.

CRITICAL ZONE THEOREM, REVISED VERSION [LEHOCZKY ET AL. 1989]. *A set of independent periodic processes scheduled by the rate-monotonic will always meet its deadlines, for all processes phasings, if and only if*

$$\forall i, 1 \leq i \leq n,$$

$$min_{(k,\,l)\varepsilon R_i} \sum_{j=1}^{i} C_j \frac{1}{lT_k} \left\lceil \frac{lT_k}{T_j} \right\rceil \leq 1$$

where C_j and T_j are the execution time and period of task j respectively and

$$R_j = \{(k, l) \mid 1 \leq k \leq i, l = 1, \ldots, \lfloor T_i / T_k \rfloor\}.$$

Detailed explanations of the critical zone theorem and illustrations of its application are given elsewhere [Lehoczky et al. 1989; Sha and Rajkumar 1994]. We use this theorem in the second stage of our task-level domain to estimate the likelihood of generating a feasible schedule at the task level. Again, it provides a bound for non-preemptive schedules by removing non-preemption constraints. While it is computationally more expensive to apply the critical zone theorem than the feasibility theorem for this estimation task, it provides significantly sharper and more realistic task-level scheduling prediction.

As indicated in Figure 1, the feasibility theorem and the rate-monotonic scheduling solution are used for estimation of the feasibility of the multiple task-level scheduling.

Finally, for the realization of the final schedule of all processes assigned to each of the partitions, the least common multiple (LCM) theorem is used [Lawler and Martel 1981]. The theorem states that for a set of periodic processes, there exists a feasible non-preemptive schedule if and only if there exists a feasible schedule for the LCM of their periods.

2.3 Area Prediction from a CDFG

We also use three different levels of abstraction for predicting the area of implementation. Each level has higher computational requirements and is more accurate. These estimation techniques are based on algorithms and tools from the existing behavioral synthesis techniques.

It is well known that the area of numerically-intensive ASIC design is dominated by interconnect, control logic, and clock distribution requirements. This makes the prediction of area of the implementation from the CDFG level of abstraction very difficult. For area estimation, we use two techniques: fast and elaborated. Both techniques use the Hyper set of estimation tools [Rabaey and Potkonjak 1994; Chandrakasan et al. 1995]. The fast technique predicts the number of instances of hardware primitives at RT-level using min-bound technique. This is done very rapidly by transforming the NP-complete scheduling problem in a series of polynomially solvable relaxed scheduling problems which provide lower bounds for the number of execution units of each hardware type, the number of interconnect and registers.

Chandrakasan et al. [1995] developed a technique which uses the information provided by the Hyper estimation routines as input to a experimentally validated statistical model to more accurately estimate the area of implementation taking into account all intricacies of logic synthesis and

physical layout design steps. In our elaborated technique, we further augmented the accuracy of estimations by first conducting complete scheduling and then used the exact value for the RT level hardware components as input parameters in the Hyper-LP statistical model for total area. The experimentation indicated that the fast estimation technique is about 20% less accurate than the elaborated one, but it is an order of magnitude faster.

Hyper and Hyper-LP also provide relatively accurate and statistically validated estimations of the clock cycle time, by using a combination of analytical and statistical techniques [Rabaey et al. 1991; Chandrakasan et al. 1995].

3. PROBLEM FORMULATION AND COMPUTATIONAL COMPLEXITY

In this section we present the formulation of the area minimization of ASIC implementation of a set of hard real-time tasking problems and establish its computational complexity. First, we must define the synthesis problem.

Problem. Synthesis of multiple tasks hard-real time systems problem:

Given: A set of M processes described by their control data-flow graphs (CDFGs), their hard real-time timing constraints (task period, start, and finish or deadline time). The finish time is the end of the task period. All processes must be implemented to satisfy all hard real-time constraints.

Goal: Is there a partitioning of the set of processes into N subsets (NM) so that each subset can be implemented on one dedicated application-specific multifunction integrated circuit? Is there an implementation with the cost of implementation (sum of areas of all chips) less than C?

As stated in the previous section, two additional control and timing disciplines are enforced during the synthesis: a single thread of control for each of the partitions is assumed and no preemption is allowed.

We start the analysis of the computational complexity by considering a simple restricted version of the problem. It is interesting to note that even in the cases when the computational complexity of the associated high-level synthesis problem and task-level hard real-time scheduling are of polynomial complexity, the synthesis of minimal cost implementation of a set of real-time processes is still an NP-complete problem. This restricted problem has only one type of operation, only the execution unit area is considered as the implementation cost, all the processes have the same sampling period, and all the tasks have identical start times equal to the beginning of the sample period. The restricted problem asks for partitioning of all processes into two partitions, each containing one execution unit (or equivalently, from a silicon area point of view, one chip with only two execution units). This problem still captures the essence and the difficulty of the associated partitioning of the initial hard real-time multiple processes synthesis into several chips.

The proof uses the standard two-phase approach for establishing NP-completeness of an optimization problem [Garey and Johnson 1979]. The

first part of the proof, that the restricted problem indeed belongs to the NP-complete class, is straightforward—once the solution is provided, due to the simplicity of the timing constraints, it is sufficient to check that all processes which are assigned to one of the particular units can be scheduled by just adding their number of operations, which is proportional to their execution time. Note that any order of the tasks on both partitions is a valid task-level schedule.

For the second part of the proof we use Karp's polynomial transformations technique. We transform a known NP-complete problem, equal subset problem [Garey and Johnson 1979], into the above-described very restricted instance of our initial problem. The equal subset set problem can be informally defined as follows: a set of n numbers is given; we must divide the set into two subsets such that the sum of numbers in each of the subset is equal. We now construct an instance of our restricted synthesis problem which corresponds to an instance of the equal subset problem. For each number in the equal subset problem, we create a process which has only one type of operations; its duration, measured in required control steps, is equal to the corresponding number in the instance of the equal subset problem. The period for each process, measured in the number of control steps, is equal to half of the sum of all numbers in the equal subset problem. Now, if one can solve the restricted real-time scheduling problem using only two chips with one execution unit each, it would imply a solution to the equal subset problem. Note that the solution for our problem must have 100% utilization and that under the described set of timing constraints the rate-monotonic scheduling provides that type of solution. Therefore, the synthesis of multiple-task hard real-time systems is at least as hard as the equal subset problem. As a result, the problem is NP-complete.

4. SYNTHESIS AND OPTIMIZATION APPROACH

The problem formulation presented in the previous section is mainly influenced by constraints common to the specification and design of modern ASICs. We proved that the synthesis of minimal cost implementation of a set of real-time processes is an NP-complete problem by highly restricting the initial problem and proving that even the resulting subproblem is computationally intractable. We now turn our attention to illustrating why this synthesis problem is also challenging in a practical sense.

Solving the synthesis problem requires solving several interacting, computationally intractable subproblems. For example, scheduling of a single process on the minimum amount of resources is itself also an NP-complete problem [Graham et al. 1977]. Actually, even when the minimization of only one type of hardware resource, such as execution units, is attempted, the subproblem is still NP-complete [Garey and Johnson 1979]. Similarly, for circular-arc CDFGs (which depict lifetimes of the variables in the scheduled single process) register minimization is also NP-complete [Garey and Johnson 1979]. At the task level, the non-preemption constraint results

in an NP-complete optimization problem [Graham et al. 1977]. It is also well known that the optimization of one hardware component very often inevitably implies a need for additional resources of another type [Guerra et al. 1993]. This layering of computationally intractable subproblems does not influence overall worst-case asymptotic computational complexity. However, it makes the synthesis problem exceptionally challenging in practice because numerous contradictory effects along several hardware dimensions at two levels of granularity (process and task) must be taken into account.

We developed a new two-domain iterative refinement multiresolution synthesis strategy to help manage the complexity of our synthesis problem. Since the synthesis problem traverses several levels of abstraction, our iterative refinement methodology guides the search by applying successively more accurate and computationally expensive models. The approach is shown in Figure 1. The solution consists of the assignment of individual processes to the partitions. Depending on the used particular search strategy for partitioning, as described in the next two sections, the partitioning is either fully or only partially specified.

We start by considering each process separately in the single-process domain. Using estimation tools from the Hyper high-level synthesis system, we obtain area-time trade-off curves for three types of hardware resources: execution units, interconnect, and registers. Next, in the tasking domain, we use the feasibility theorem and the area-time trade-off curves to obtain a first estimate of the required hardware resources and feasibility of timing constraints for each partition. In particular, it is important to verify that the critical paths in all processes assigned to each of the partitions satisfy the feasibility theorem. As soon as one of the estimations indicates that the proposed solution is infeasible, the search backtracks to the previous best feasible solution.

The design is further refined by returning to the single-task domain and coming back to the tasking domain for new checking using the critical zone theorem. In the single process domain, using the augmented Hyper-LP estimations, more accurate estimates for all seven hardware components and complete implementations are obtained. Similarly as in the initial round of visiting the two domains, inferior and infeasible solutions are discarded.

Finally, in the last two visits to the two domains, the complete single-process and task-level schedules are obtained using the Hyper scheduler and our task-level scheduler. A new complete solution is accepted only after its area is obtained using the Hyper high-level synthesis system and at the task level non-preemptive schedule is generated and the quality of the result is superior to all other previously considered solutions.

Our task-level scheduler is built on top of the Hyper hierarchical scheduler, which performs global optimizations across several blocks. While the initial semantic interpretation of the blocks in Hyper is that they denote loops and subroutines, the new one, at the highest level of hierarchy, interprets each block as one iteration of an individual task. Two other

important additions to the initial hierarchical Hyper scheduler are also done. First, set the available time to LCM of the sampling periods of the processes assigned to the partition and replicate accordingly, if necessary, the number of iterations considered for each process. The second algorithmic addition orders all the considered iterations of all processes using the EDD rule. The additional hardware allocation or removal and the time distribution across the iteration are done automatically by the Hyper hierarchical scheduler (see Potkonjak and Rabaey [1992]).

This design methodology is a basis for an optimal worst-case exponential time branch-and-bound synthesis algorithm as well as a fast heuristic synthesis algorithm. These two algorithms are the subjects of the next two sections.

5. OPTIMAL BRANCH-AND-BOUND SYNTHESIS ALGORITHM

Developing a strictly optimum algorithm for the synthesis of multiple hard real-time processes is, in view of its layered structure of NP-complete subproblems, not a practical solution to ASIC synthesis. However, it is still very interesting and beneficial to try to solve the problem optimally under an additional set of reasonable assumptions. Reasonable assumptions are ones which constrain the problem so that it can be addressed within the limits of today's computational resources. All important aspects of the original problem are preserved so that the generated solution represents a high-quality solution for the initial problem. In this section, we will show one such optimal solution, based on the assumptions that our behavioral synthesis generates an optimal implementation for a single process and that our EDD-based task-level scheduler also produces optimal solutions. Our experimentation indicates that the assumptions are actually surprisingly often correct, since both schedulers often produce the solutions which are either equal or close to lower bounds [Rabaey and Potkonjak 1994; Ramamritham and Stankovic 1994]. Also, the approach would provide the optimum solution, if exponential run-time schedulers are used for synthesis (e.g., ILP-based schedulers [Walker and Camposano 1991]).

We now present an optimal branch-and-bound method augmented with several heuristic search space pruning strategies [Papadimitrou and Steiglitz 1982]. Figure 2 illustrates the approach and shows how the search space is enumerated. The algorithm first tries to cluster all the processes in two groups, then in three, and so on until it is proven that all solutions which use the larger number of groups are inferior to the current best solution. Each of these attempts is termed an epoch. Figures 2a and 2b show epochs which correspond to clustering into 2 and 3 groups, respectively.

For a given number of utilized chips, at each step of the optimal algorithm a decision is made as to which set to assign a particular process. To each process the task number is assigned. In order to speed up pruning, without loss of generality and optimality, the processes with larger single-process implementation areas are assigned smaller numbers. The underly-

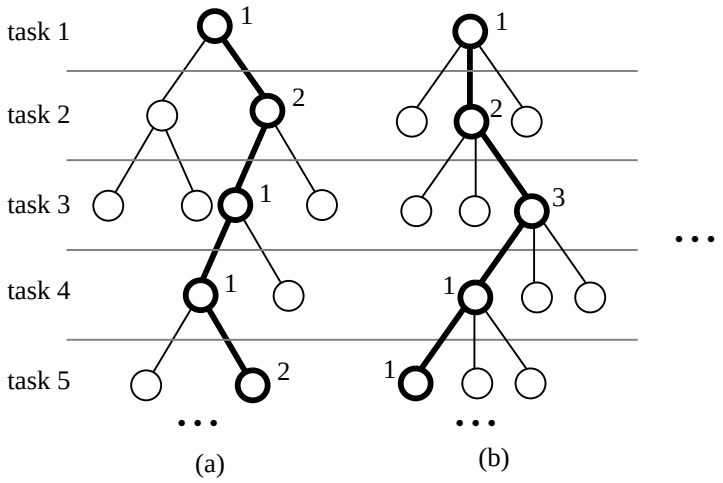

Fig. 2. Branch-and-bound-based optimal synthesis approach: representation of partial solutions (a) two-processor synthesis epoch and (b) three-processor synthesis epoch. For a detailed description of the branch-and-bound algorithm, see Section 5.

ing idea is to check whether a particular solution is infeasible as soon as possible and therefore speed up the search. Each node has k children, where k is the number of groups considered in a particular epoch. The children are marked with a number which indicates to which group the corresponding process is assigned from the left to the right.

The complete tree encodes, using paths from the root to any of leaves, all possible solutions for a given number of groups in the following way: A particular process (node) is assigned to the group indicated by the position of the edge that connects it to its parent. The root is, without loss of generality, assigned to group 1. For example, the bold path in Figure 2a indicates that processes 1, 3, and 4 are in set 1, and processes 2 and 5 in set 2. Similarly, the bold path in Figure 2a shows partial solutions where processes 1, 4, and 5 are assigned to set 1, process 2 to set 2, and process 3 to set 3. We compare the quality of solutions corresponding to all paths using a depth-first search. During the enumeration, all paths which correspond to partial solutions which either violate timing constraints in any of the groups or are inferior to the current best solution are terminated and further considered. The size of solution and feasibility are checked using the strategy provided by the V-chart—the algorithm first considers fast single-process and task estimations, and then more accurate estimation, and eventually the complete solution.

In order to speed-up the branch-and-bound approach, the initial solution is generated using the heuristic algorithm presented in the next section. Even then, relatively small instances of the problem require very long run times (several hours on a modern workstation). We give more detailed results in Section 7.

```
RMS_synthesis() {
  nd initial feasible solution with 1 pr  ocess per PE;
  repeat {
    repeat { /* merge expensive processes */
      P1 = select_most_expensive();
      P2 = select_target(P1);
      /* try to move P2's processes onto P1 */
      if (try_to_merge(P1,P2))
        update_solution(); /* successful move */
    } until (no expensive choices);
    repeat { /* merge processes with slack */
      P1 = select_slack();
      P2 = select_target(P1);
      /* try to move P2's processes onto P1 */
      if (try_to_merge(P1,P2))
        update_solution(); /* successful move */
    } until (no slack choices);
  } until (no improvement);
}
```

Fig. 4. Outline of our synthesis algorithm.

6. FAST HEURISTIC SYNTHESIS ALGORITHM

This section describes the heuristic algorithm for behavioral synthesis of multiple-process, hard real-time systems. We will first present the search strategy used to propose a solution (partitioning of processes in several groups, so that each group is implemented on one ASIC). Next, we briefly explain how the critical zone theorem is used to obtain a task-level schedule. Finally, we conclude with a description of our modifications of traditional high-level synthesis tools for the new application domain.

The outline of our heuristic synthesis procedure is given in Figure 3. It starts from an initial feasible solution and iteratively refines the solution to reduce cost while maintaining feasibility. The initial feasible solution is found by allocating each process to a separate partition, so that each process is on a separate processing element (PE). It is easy to determine in this case if there is a feasible solution since the execution times of the processes are independent. During synthesis, each step moves the processes from one PE to another in order to eliminate a need for one of the PEs. Since only two PEs are involved in a move, it is possible to determine if the reallocation is feasible.

Our heuristic search algorithm iteratively applies two phases. The first phase selects a suitable expensive process and tries to merge that process'

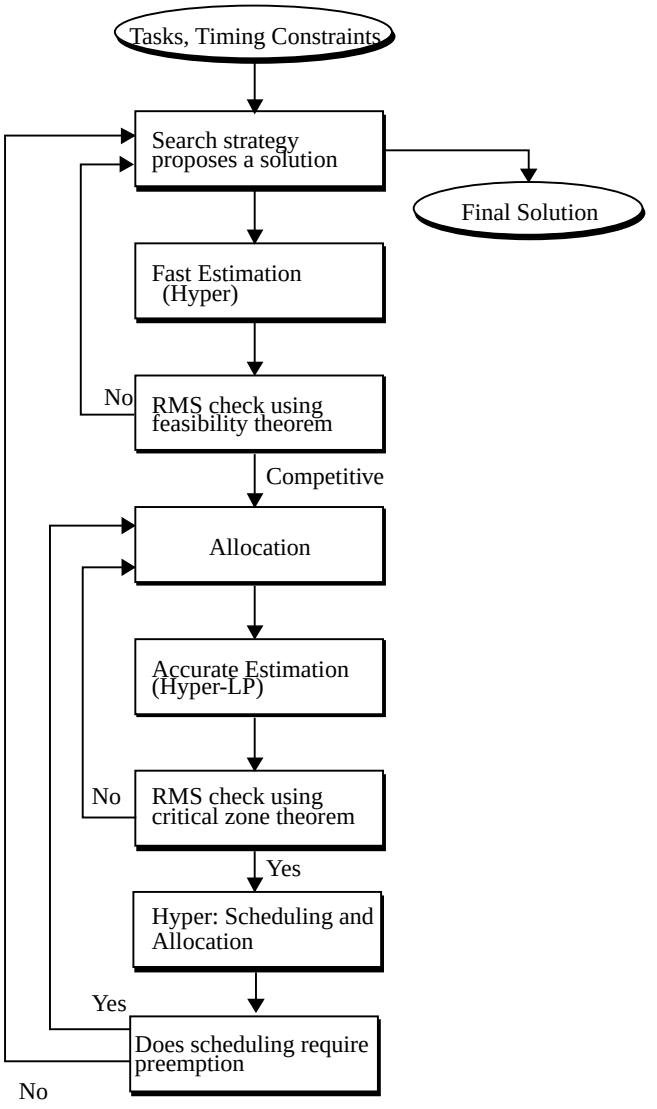

Fig. 3. The design flow of heuristic synthesis algorithms for design of hard real-time multitask applications. For a detailed description of the corresponding optimization algorithm, see Section 6.

PE with another PE. The cost of the process is measured as the area of its current ASIC implementation. This type of move tries to choose small, inexpensive processes to share the expensive process's PE in such a way that the implementation resources are efficiently time-shared. The second phase tries to merge a process which has a large slack with respect to its deadline when implemented on their current hardware resources. Processes with large slacks are more likely to be able to be executed on another PE, so this move helps reduce cost.

The function select_target(P1) uses several criteria to determine which PE is the best candidate for possible sharing with P1. We have found that four criteria are important in selecting a target PE which is to share a given process. The criteria in descending order of importance are:

(1) *Similar bit widths for operations.* Hardware is wasted if two processes operate on data of different bit widths. Also, longer bit width implies longer cycle time. If neither process has multiplies, the similarity is quantified using the ratio of the number of bits in the process; otherwise, the similarity measure is the square of the ratio, so that both increases in the hardware size and the clock cycle time are taken into account.

(2) *Similar types of functional units required.* The similarity is quantified as the sum of ratios of the estimated number of execution units for all types of resources.

(3) *The sources and sinks of data transfers.* It is important to match processes which have similar communication patterns. The Hyper estimation tools are used to obtain this information and the measure is quantified in the same way as for functional units.

(4) *Similar number of registers.* The required number of registers is estimated using the Hyper estimation tools which calculate the maximum cutset of the data-flow graph [Rabaey and Potkonjak 1994], as measured by the ratio of the estimated number of registers in the initial designs.

We tried several other criteria, for example, the sum of the current ratios of time the processes use the current set of allocated hardware resources, but preliminary experiments showed that they do not contribute positively to the quality of the final solution. The final objective function is weighted sum of the four components. Note that the objective function in some sense models through its four components how much two given PEs attract each other, and tries first to make matches with strong attraction forces. Since there are four independent components in each force, we termed this heuristic the multidimensional force-directed algorithm.

As mentioned earlier, the high-level synthesis tools, the rate-monotonic critical zone scheduling algorithm, and the feasibility theorem are used in try_to_merge(P1,P2) and update_solution() steps. This process is done in two steps: first using the fast estimation procedure and if this test is satisfied using the elaborated task-level estimation procedure.

The application of the critical zone theorem for rate-monotonic scheduling, which is done in the try_to_merge(P1,P2) and update_solution() steps, is a very efficient method for predicting feasibility and the chances for efficient area implementation of the final solution.

We modified the Hyper high-level synthesis tools to accommodate the needs of try_to_merge(P1,P2) and update_solution() steps in producing area-efficient implementations. The number of bits for all processes is set

to the number of bits in the highest word length requirements among individual processes, so that all allocated hardware resources can be reused when any process is scheduled. All processes are treated as subroutines during scheduling, and Hyper automatically allocates to all iterations of all processes in the provably optimal fashion [Rabaey et al. 1991], the available time among the individual processes. If required, additional hardware resources are allocated automatically by the Hyper hierarchical scheduler after merging. However, in this case the decision to update the current solution is postponed until at least three other proposed solutions are examined. The solution is updated only after a feasible task-level nonpreemptive schedule is obtained. This technique does not guarantee optimality, but the final achieved results are most often fully satisfactory, as indicated by the experimental results presented in the next section.

7. EXPERIMENTAL RESULTS

7.1 Description of Examples

Table IV gives the description of 16 processes used to construct eleven different set of tasks. For each process the number of operations, the word length, and the initial area when each process is implemented on a separate chip is given. We used these processes to construct the task sets used for the experiments as shown in Table V. The examples are: GE Controller1 and GE Controller2—two 4 and 5 state linear controllers; Honda Controller1 and Honda Controller2—two linear mechanical controllers with 5 states; Wavelet filter—QMB sub-band filter used in filter bank speech recognition; Low Pass Filter, BandPass Filter, High Pass Filter and Band Stop Filter—four audio filters (pass filters are cascade and parallel IIR structures, two other filters are direct form and transpose form FIR structures); 8X8 DCT–2 dimensional Lee's algorithm version 8 point of discrete cosine transform; DAC–4 cascade NEC digital-to-analog converter for audio applications; modem and adaptive modem—signal processing components of two low-speed low-cost communication modems; LMS audio formatter and Echo-Canceller—two NEC communication designs and Large Controller–11 states, 3 input, and 3 output linear controller for automotive motion control applications. We grouped these processes into tasks as shown in Table V.

7.2 Results and Analysis

Table V also presents the solutions produced by our multidimensional force-directed-based high-level synthesis algorithm. The second column in Table V gives the set of considered designs, the third column specifies the obtained solution, and the fourth column gives area in square millimeters for each design which implements the corresponding set of tasks from the third column. The final column gives the ratio of the sum of all initial designs corresponding to the considered set of tasks and the sum of area after the application of our optimization algorithm. The average and the

Table IV. Individual Characteristics of Processes in Experiments

Process no.	Process	No. of operations	No. of bits	Initial cost [mm2]
1.	GE controller1	48	8	10.47
2.	GE controller2	108	20	38.88
3.	Honda controller1	97	16	27.28
4.	Honda controller2	67	16	23.63
5.	Wavelet filter	31	12	15.10
6.	Low pass filter	32	10	13.65
7.	Band pass filter	38	11	17.82
8.	High pass filter	42	14	18.24
9.	Band stop filter	30	13	16.37
10.	8X8 DCT	46	24	27.06
11.	DAC	354	16	26.62
12.	Modem	227	20	31.78
13.	Adaptive modem	200	20	35.52
14.	Large controller	324	32	66.82
15.	LMS audio formatter	464	32	73.26
16.	Echo-canceller	212	32	64.57

median area reductions are by factors 2.02 and 1.98, respectively, clearly indicating the advantages of combining several processes on one ASIC implementation.

More importantly, the comparison with the optimal branch-and-bound method indicates that for several 8-process examples the optimal solution is generated. The run time for all examples is less than 10 minutes on SUN Sparcstation-5, including the complete synthesis of the final solutions.

We analyzed the resulting designs to determine factors which lead to good implementations. The best solutions tend to group together processes which require a similar number of bits in their word length as well as a similar type of operations. Neither of these two components can be ignored if the high quality solution is targeted. It is interesting to note that in many cases, although a new solution did not require any additional execution units and no new interconnects compared to the largest design of the designs which implement the individual tasks, the area of design increased significantly. The increase in the number of registers was mainly due to a need to store constants in the registers in the ROM register files for both designs which are combined. Since in the fixed point designs the area of a register is almost half of the area of an adder and for short word length a significant part of the area of a multiplier, this aspect often has a significant impact.

Finally, note that in modern implementation technologies pin count is often more important than chip area. In all examples, the number of pins in final designs was equal to the word length used in design. Therefore, the reductions in the number of pins was even more significant.

8. RELATED WORK

Directly related work can mainly be traced along the following five lines of research: high-level synthesis techniques; system-level synthesis; schedul-

Table V. Experimental Result - Area of Implementation for ASIC Hard Real–Time Designs Using Force-Directed Heuristics. For a Detailed Explanation of the Results see Section 7

Set of tasks	Processes in the set	Final solution	Final cost [mm2]	Improvement initial/final
set 1	{1,2,3,4,5,6,7,8}	{1,5,6}{2,3,4,7,8}	18.92+63.27	2.01 (165.07/82.09)
set 2	{5,6,7,8,9,10,11,12)	{5,6,7,8,9}{10,11,12}	52.77+46.38	1.68 (166.64/99.15)
set 3	{9,10,11,12,13,14, 15,16}	{9,11,12,13} {10,14,15,16}	41.52+99.22	2.43 (342.00/140.74)
set 4	{13,14,15,16,1,2,3,4}	{1,3,4}(2,13} {14,15,16}	32.28+42.52+98.80	1.98 (343.43/173.60)
set 5	{1,3,5,7,9,11,13,15}	{1,5,7,9}{3,11,13,15}	21.95+99.42	1.83 (222.44/121.37)
set 6	{2,4,6,8,10,12,14,16}	{2,12}{4,6,8} {10,14,16}	29.81+48.09+92.47	1.67 (284.63/170.37)
set 7	{1,2,3,4,5,6,7,8,9,10, 11,12}	{1,5,6,7}{2,3,4,8,11} {10,12}	26.91+74.07+37.55	1.93 (266.90/121.43)
set 8	{5,6,7,8,9,10,11,12, 13,14,15,16}	{5,6,7,8,9,11} {10,11,12,13}{14,15,16}	52.77+54.09+98.80	1.98 (406.81/205.66)
set 9	{9,10,11,12,13,14, 15,16,1,2,3,4}	{1,3,4}{2,11,12,13} {10,14,15,16}	32.28+45.92+99.22	2.49 (442.26/177.42)
set 10	{13,14,15,16,1,2,3, 4,5,6,7,8}	{1,5,6}{2,3,4,7,8}{13} {14,15,16}	18.92+63.27+35.52 +98.80	1.87 (405.24/216.51)
set 11	{1,2,3,4,5,6,7,8,9,10, 11,12,13,14,15,16}	{1,3,4,5,6,7,8,9,10} {2,11,12,13}{14,15,16}	63.36+49.98+98.80	2.39 (507.07/220.79)

ing and operating systems in hard real-time systems; compilation techniques for parallel and distributed systems; and design methodology and algorithmic techniques issues.

Within high-level synthesis several subdomains are to some extent related to the research described in this paper, including partitioning, estimations and area prediction, and design of application-specific instruction sets (ASIP) and application-specific programmable processors (ASPP). An excellent summary and review of the early work on high-level synthesis is given in McFarland et al. [1990]. More recently, several additional comprehensive surveys have been conducted, including an exceptionally comprehensive one presented by Walker and Camposano [1991].

Although there is a sharp conceptual difference between traditional high-level synthesis partitioning techniques, which target partitioning of a single computational task across several chips, and task-level partitioning (process assignment), it is still instructive to review the former. This is so because regardless of differences in the semantic meaning of partitioning in two design scenarios, differences in considered levels of abstraction, and the different nature of timing constraints, there is an underlying principle common to both. In both cases the goal is to identify the subsets of computations that are well suited for hardware sharing after ASIC implementation.

High-level synthesis partitioning techniques were pioneered by McFarland et al. [1983], Rajan and Thomas [1985], and Camposano and Brayton [1987]. All of them considered similarity of functions, common data carriers, and low-level parallelism within single-stage clustering procedures.

Consequently, Lagnese and Thomas [1989] generalized this work by considering multistage clustering and reported a 20% reduction in the number of wiring tracks on a benchmark example.

Gupta and De Micheli [1990] compared the effectiveness of simulated annealing and heuristic approaches for the partitioning of programmable and hardwired designs. Gebotys [1992] applied an ILP-based technique to obtain the optimal solution for partitioning several popular high-level synthesis benchmarks. A high-level synthesis group at the University of Southern California proposed several partitioning techniques for both high-level synthesis [Kucukcakar and Parker 1991] and system-level synthesis [Prakash and Parker 1991].

Recently synthesis of application-specific programmable processors (ASPP) [Breternitz and Shen 1990; Guerra et al. 1993] and application-specific instruction sets processors (ASIP) [Leupers et al. 1994; Paulin et al. 1994; Goosens et al. 1995] received a great deal of attention in the CAD community. While both ASPPs and ASIPs and the technique proposed in this paper target implementation of several processes on the same processor, similarities between the two domains is very limited due to the different operational modes of the processors and the different nature of timing constraints. For example, while both ASIP and ASPP designs assume that their design will eventually be used to realize only one of several (or many) applications, the hard real-time rate-monotonic scheduling-based ASIC design enables several processes to share the same hardware during their execution.

The run time effectiveness of the new approach is directly based on the exploration of high-level synthesis estimation techniques. The importance of precise area prediction from both the functional and RT-level of the design process was first and most clearly pointed out by McFarland and Kowalski [1990]. Rabaey and Potkonjak [1991] and Sharma and Jain [1993] developed relaxation-based techniques for calculating sharp lower bounds for the number of execution units, registers, and interconnect. Those techniques consider one type of resource at a time. Chaudhury and Walker [1994] developed similar techniques which derive lower bounds considering several types of operations simultaneously.

Most high-level synthesis layout estimators are at least partially constructive and therefore time-consuming [Kurdahi and Ramachandran 1993]. Recently, Chandrakasan et al. [1995] developed accurate linear piece-wise statistical models for predicting area of interconnect, control logic, clock circuitry, and the complete layout from the behavioral description of the design.

System-level synthesis and, in particular, hardware-software codesign also recently received a great deal of attention [Wolf 1994]. The most relevant system research subdomain is hardware-software partitioning [Barros et al. 1994; Ernst et al. 1993; Gajski et al. 1994; Gupta and De Micheli 1993; Ismail et al. 1994]. The goal of software partitioning is to identify parts of computations which should be implemented on programmable and ASIC platforms so that an overall optimization function is

maximized. However, again as in the case of high-level synthesis partitioning the same set of key differences exist between hardware-software partitioning and the rate-monotonic scheduling-based high-level synthesis.

Hard real-time scheduling efforts have been an important topic of research for three decades. The early work on scheduling of a set of periodic tasks with strict timing constraints on periodicity, arrival, and required time of each task, culminated in a classic rate-monotonic scheduling algorithm [Liu and Layland 1973]. Consequently, rate-monotonic scheduling has been extensively analyzed and generalized, mainly by researchers at Carnegie-Mellon University [Lehoczky et al. 1989; Sha and Goodenough 1990; 1994]. The most notable practical application of real-time scheduling approaches, and, in particular, rate-monotonic and generalized rate-monotonic scheduling algorithms, include the inclusion of rate-monotonic scheduling as the scheduling policy for the IEEE POSIX real-time operating system standard and IEEE Futurebus+ standards [IEEE 1991; 1993; Sha et al. 1990], and the use of the generalized rate-monotonic scheduling techniques in several major advance-technology projects such as the Space Station Program and the European Space Agency on-board operating system [Sha et al. 1994]. The strong endorsement of several research and development groups of the earliest-deadline-first and rate-monotonic scheduling as most suitable resource allocation policies for continuous media servers [Nagarajan and Vogt 1992; Ramamritham and Stankovic 1994; Steinmetz 1995] and ATM switch scheduling further emphasizes importance of this hard real-time scheduling approach.

The topic of partitioning of single or multiple tasks during compilation for parallel and distributed programmable platforms has been a widely studied [Almasi and Gottlieb 1994].

Scheduling under a complex set of timing constraints was studied by Ku and De Micheli [1992], Cho et al. [1994], and Yen and Wolf [1996].

Before concluding our survey of related research efforts with design methodology and algorithmic issues, we refer to three recent publications in the CAD literature which address use of the rate-monotonic scheduling algorithms. Hu et al. [1994] compared several hard real-time scheduling policies during hardware-software partitioning of the controller for an automotive powertrain module, but did not develop a synthesis approach or conduct any synthesis optimization. In the 1995 EDAC conference paper, Atlenbernd [1995] generalized deadline-monotonic scheduling policy along several lines, but also did not make a connection to the synthesis process. Recently, Yen and Wolf [1995] addressed synthesis of distributed embedded systems with both hard and soft deadlines. While all hard deadlines must be satisfied, the optimization goal is to satisfy as many soft deadlines as possible.

It appears that van Cleemput [1979] was the first to formally discuss the hierarchical design process along three different dimensions as three concurrent tasks: behavioral, structural, and physical design. Gajski and Kun [1983] used those three dimensions as three dimensions of Y-chart of the design process along which design refinement is conducted. They

classified several design methodologies depending on the order in which refinement is conducted. Our V-chart is very different because in one of our two dimensions, which represents single-process domains, the refinement goes from the functional to geometric representation, while in the other, which represents multiple tasks, the optimization is conducted solely at the behavioral level.

Finally, force-directed heuristics have been successfully used in several CAD domains for solving a variety of computationally intractable problems. In the CAD literature they were pioneered by Soukup [1981], who used it as a key optimization step in the epitaxial growth algorithm for constructive placement. Paulin and Knight [1989] developed a conceptually very similar approach for force-directed scheduling, which, due to its clear intuitive foundations, has been very popular in behavioral synthesis research [McFarland et al. 1990; Walker and Camposano 1991].

9. FUTURE RESEARCH AND CONCLUSION

Two major goals of the research presented in this paper are:

—creation of an impetus for comprehensive treatment of interaction between synthesis process—in particular, high-level synthesis—and requirements imposed by operating systems; and

—study of synthesis techniques for exploration of hardware sharing at the process level.

The major goals form a basis for several extensions. Two conceptually straightforward, but practically very important directions are to address task-level hardware sharing and interaction of high-level synthesis and real-time operating systems when a variety of design goals are targeted and when alternative implementation platforms are considered. The most attractive alternative optimization goals include power, testability assuming different testing strategies, and fault-tolerance overhead minimization. Important alternative implementation platforms include a variety of programmable platforms (e.g., microprocessors, microcontrollers, DSP and video processors), ASIPs and ASSPs, and a combination of programmable off-the-shelf platforms and ASIP, ASSP, and full-custom ASIC implementations. Another important direction is to address behavioral synthesis of hard real-time systems when other underlying computational models and different sets of timing constraints are used.

Recently, it has been demonstrated that by addressing higher levels of abstraction during synthesis, such as algorithm design and selection and algorithm and architecture matching, exceptionally high improvements in several design metrics are achievable. The influence of real-time operating systems are considered in conjunction with this phase of the synthesis process.

Also, it will be interesting to address the impact of other high-level synthesis tasks, such as transformations and template matching, on both

task-level hardware sharing and interaction between scheduling policies of operating systems.

We will conclude the outline of directions for future research by pointing out that numerous research avenues are open if one also considers, in addition to the static non-preemptive scheduling schemes, other frequently used operating system policies, such as priority ceiling, round-robin, and first-come first-serve [Sha and Rajkumar 1994].

In summary, we introduced the first synthesis approach for optimization at the task-sharing level using high-level synthesis techniques. The key novelty of this work is our method for addressing synthesis of multiple tasks using integration of operating systems and high-level synthesis techniques through a novel iterative refinement two-domain optimization approach. The key algorithmic novelty is an introduction of a simple, but effective multidimensional force-directed heuristic synthesis algorithm. The key new high-level synthesis phenomenon associated with task-level hardware sharing is the importance of matching the number of bits for the processes selected to be implemented on the same platform. We demonstrated the effectiveness of the approach and the optimization algorithms on several examples. We believe that, as designers move toward systems-on-a-chip, synthesis of multiple-process systems will become increasingly important.

REFERENCES

ALMASI, G. S. AND GOTTLIEB, A. 1994. *Highly Parallel Computing. 2nd ed*. Benjamin-Cummings Publ. Co., Inc., Redwood City, CA.

ATLENBERND, P. 1995. Deadline-monotonic software scheduling for the co-synthesis of parallel hard real-time systems. In *Proceedings of the European Conference on Design Automation* (EDAC-95). IEEE Computer Society Press, Los Alamitos, CA, 190–195.

BANNISTER, J. A. AND TRIVEDI, K. S. 1983. Task allocation in fault-tolerant distributed systems. *Acta Inf. 20*, 3, 261–283.

BOREL, J. 1997. Technologies for multimedia systems on a chip. In *Proceedings of the IEEE International Conference on Solid-State Circuits*. IEEE Computer Society Press, Los Alamitos, CA, 18–21.

BRETERNITZ, M. AND SHEN, J. P. 1990. Architecture synthesis of high-performance application-specific processors. In *Proceedings of the ACM/IEEE Conference on Design Automation* (DAC '90, Orlando, FL, June 24-28), R. C. Smith, Ed. ACM Press, New York, NY, 542–548.

BRODERSEN, R. W. 1997. The network computer and its future. In *Proceedings of the IEEE International Conference on Solid-State Circuits*. IEEE Computer Society Press, Los Alamitos, CA, 32–36.

CAMPOSANO, R. AND BRAYTON, R. 1987. Partitioning before logic synthesis. In *Proceedings of the International Conference on Computer-Aided Design* (ICCAD). 324–326.

CHAUDHURI, S. AND WALKER, R. A. 1994. Computing lower bounds on functional units before scheduling. In *Proceedings of the Seventh International Symposium on High-Level Synthesis* (ISSS '94, Niagara-on-the-Lake, Ont., Canada, May 18–20, 1994), P. G. Paulin, Ed. IEEE Computer Society Press, Los Alamitos, CA, 36–41.

CHANDRAKASAN, A. P., POTKONJAK, M., MEHRA, R., RABAEY, J., AND BRODERSEN, R. 1995. Optimizing power using transformations. *IEEE Trans. CAD 14* (Jan.), 13–32.

CHOU, P., WALKUP, E. A., AND BORRIELO, G. 1994. Scheduling for reactive real-time systems. *IEEE Micro 14*, 4 (1994), 37–47.

ERNST, R., HENKEL, J., AND BENNER, T. 1993. Hardware-software co-synthesis for microcontrollers. *IEEE Des. Test 10*, 4 (Dec. 1993), 64–75.

GAJSKI, D. D. AND KUN, R. H. 1993. Guest editor's introduction: New VLSI tools. *IEEE Computer 16*, 12, 11–14.

GAJSKI, D. D., VAHID, F., AND NARAYAN, S. 1994. A system-resign methodology: Executable specification refinement. In *Proceedings of the Conference on EURO-DAC'94*. 458–463.

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY.

GEBOTYS, C. H. 1992. Optimal synthesis of multichip architectures. In *Proceedings of the 1992 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '92, Santa Clara, CA, Nov. 8–12), L. Trevillyan, Ed. IEEE Computer Society Press, Los Alamitos, CA, 238–241.

GIBBS, W. W. 1994. Software's chronic crisis. *Sci. Am.* (Sept. 1994), 86–95.

GOOSSENS, G., LANNEER, D., PAUWELS, M., DEPUYDT, F., SCHOOFS, K., KIFLI, A., CORNERO, M., PETRONI, P., CATTHOOR, F., AND DE MAN, H. 1995. Integration of medium-throughput signal processing algorithms on flexible instruction-set architectures. *J. VLSI Signal Process. 9*, 1/2 (Jan. 1995), 49–65.

GRAHAM, R. L., LAWLER, E. L., LENSTRA, J. K., AND RINNOY KAN, A. H. G. 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete Math. 5*, 287–326.

GUERRA, L., POTKONJAK, M., AND RABAEY, J. 1993. High level synthesis for reconfigurable datapath structures. In *Proceedings of the International Conference on Computer-Aided Design* (ICCAD '93, Santa Clara, CA, Nov. 7–11, 1993), M. Lightner and J. A. G. Jess, Eds. IEEE Computer Society Press, Los Alamitos, CA, 26–29.

GUPTA, R. AND DEMICHELI, G. 1990. Partitioning of functional models of synchronous digital systems. In *Proceedings of the International Conference on Computer-Aided Design* (ICCAD'90). IEEE Computer Society Press, Los Alamitos, CA, 216–219.

GUPTA, R. K. AND DE MICHELI, G. 1993. Hardware-software cosynthesis for digital systems. *IEEE Des. Test 10*, 3 (Sept. 1993), 29–41.

HENNESSY, J. L. AND PATTERSON, D. A. 1990. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA.

HU, X., D'AMBROSIO, J. G., AND TANG, D-L. 1994. Codesign of architecture for automotive powertrain modules. *IEEE Micro 14*, 4, 17–25.

HUANG, C-H., YEN, J-Y., AND OUHYOUNG, M. 1996. The design of a low cost motion chair for video games and MPEG video playback. *IEEE Trans. Consumer Electron. 42*, 4, 991–997.

IEEE. 1991. Real-Time extensions to POSIX. 1003.4. IEEE Standards Office, New York, NY.

IEEE. 1993. Futurebus+ Recommended Practice. IEEE Standards Office, New York, NY.

ISMAIL, T. B., O'BRIEN, K., AND JERRAYA, A. 1994. Interactive system-level partitioning with PARTIF. In *Proceedings of the European Conference on Design Automation* (EURO-DAC '94, Grenoble, France, Sept. 19–23, 1994), J. Mermet, Ed. IEEE Computer Society Press, Los Alamitos, CA.

JACKSON, J. R. 1969. Scheduling a Production Line to Minimize Maximum Tardiness. Res. Rep. 43. Management Science Project. University of California at Los Angeles, Los Angeles, CA.

KIM, K., KARRI, R., AND POTKONJAK, M. 1997. Synthesis of application specific programmable processors. In *Proceedings of the 34th Annual Conference on Design Automation* (DAC '97, Anaheim, CA, June 9–13, 1997), E. J. Yoffa, G. De Micheli, and J. M. Rabaey, Eds. ACM Press, New York, NY, 353–358.

KU, D. AND DE MICHELI, G. 1990. Relative scheduling under timing constraints. In *Proceedings of the ACM/IEEE Conference on Design Automation* (DAC '90, Orlando, FL, June 24-28), R. C. Smith, Ed. ACM Press, New York, NY, 59–64.

KÜKÇAKAR, K. AND PARKER, A. C. 1991. CHOP: A constraint-driven system-level partitioner. In *Proceedings of the 28th ACM/IEEE Conference on Design Automation* (DAC '91, San Francisco, CA, June 17–21), A. R. Newton, Ed. ACM Press, New York, NY, 514–519.

KURDAHI, F. AND RAMACHANDRAN, C. 1993. Evaluating layout area tradeoffs for high level applications. *IEEE Trans. Very Large Scale Integr. Syst. 1*, 1, 46–55.

LAGNESE, E. D. AND THOMAS, D. E. 1989. Architectural partitioning for system level design. In *Proceedings of the 26th ACM/IEEE Conference on Design Automation* (DAC '89, Las Vegas, NV, June 25–29, 1989), D. E. Thomas, Ed. ACM Press, New York, NY, 62–67.

LAWLER, E. L. AND MARTEL, C. U. 1982. Scheduling periodically occurring tasks on multiple processors. *Inf. Process. Lett. 12*, 1, 9–12.

LEE, E. A. AND MESSERSCHMITT, D. G. 1987. Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Comput. C-36*, 1 (Jan. 1987), 24–35.

LEHOCZKY, J. P., SHA, L., AND DING, Y. 1986. The rate monotonic scheduling algorithms - Exact characterization and average case behavior. In *Proceedings of the IEEE Symposium on Real-Time Systems*. IEEE Press, Piscataway, NJ, 181–191.

LEUNG, J. AND WHITEHEAD, J. 1982. On the complexity of fixed priority scheduling of periodic, real-time tasks. *Perform. Eval. 2*, 4, 237–250.

LEUPERS, R., SCHENK, W., AND MARWEDEL, P. 1994. Retargetable assembly code generation by bootstrapping. In *Proceedings of the Seventh International Symposium on High-Level Synthesis* (ISSS '94, Niagara-on-the-Lake, Ont., Canada, May 18–20, 1994), P. G. Paulin, Ed. IEEE Computer Society Press, Los Alamitos, CA, 88–93.

LIU, C. L. AND LAYLAND, J. W. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20*, 1 (Jan. 1973), 46–61.

MCFARLAND, M. C. 1983. Computer-aided partitioning of behavioral hardware descriptions. In *Proceedings of the 20th Conference on Design Automation*. 472–480.

MCFARLAND, M. C. AND PARKER, A. C. 1990. The high-level synthesis of digital systems. *IEEE Computer 78*, 2 (Feb. 1990), 301–317.

MCFARLAND, M. AND KOWALSKI, T. 1990. Incorporating bottom-up design into hardware synthesis. *IEEE Trans. CAD 9*, 9 (Sept. 1990).

MINOLI, D. AND KEINATH, R. 1994. *Distributed Multimedia Through Broadband Communications Services*. Artech House, Inc., Norwood, MA.

NAGARAJAN, R. AND VOGT, C. 1992. Guaranteed performance of multimedia traffic over the token ring. IBM Corp., Riverton, NJ.

PAPADIMITRIOU, C. H. AND STEIGLITZ, K. 1982. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ.

PAULIN, P. G. AND KNIGHT, J. P. 1989. Force-directed scheduling for the behavioral synthesis of ASICs. *IEEE Trans. CAD 8*, 6 (June 1989), 661–679.

PAULIN, P. G., LIEM, C., MAY, T. C., AND SUTARWALA, S. 1994. CodeSyn: A retargetable code synthesis system (abstract). In *Proceedings of the Seventh International Symposium on High-Level Synthesis* (ISSS '94, Niagara-on-the-Lake, Ont., Canada, May 18–20, 1994), P. G. Paulin, Ed. IEEE Computer Society Press, Los Alamitos, CA, 94.

POTKONJAK, M. AND RABAEY, J. 1992. Scheduling algorithms for hierarchical data control flow graphs. *Int. J. Circuits Theor. Appl. 20*, 3, 217–234.

POTKONJAK, M. AND RABAEY, J. 1994. Algorithm selection: a quantitative computation-intensive optimization approach. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD '94, San Jose, CA, Nov. 6–10, 1994), J. A. G. Jess and R. Rudell, Eds. IEEE Computer Society Press, Los Alamitos, CA, 90–95.

POTKONJAK, M. AND WOLF, W. 1995. Cost optimization in ASIC implementation of periodic hard-real time systems using behavioral synthesis techniques. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 446–451.

PRAKASH, S. AND PARKER, A. C. 1991. Synthesis of application-specific multiprocessor architectures. In *Proceedings of the 28th ACM/IEEE Conference on Design Automation* (DAC '91, San Francisco, CA, June 17–21), A. R. Newton, Ed. ACM Press, New York, NY, 8–13.

RABAEY, J., CHU, C., HOANG, P., AND POTKONJAK, M. 1991. Fast prototyping of data path intensive architecture. *IEEE Des. Test 8*, 2, 40–51.

RABAEY, J. AND POTKONJAK, M. 1991. Complexity estimations for real time application specific circuits. In *Proceedings of the 17th European Conference on Solid-State Circuits* (Milan, Italy, Sept.). 201–204.

RABAEY, J. M. AND POTKONJAK, M. 1994. Estimating implementation bounds for real time DSP application specific circuits. *IEEE Trans. CAD 13*, 6 (June), 669–683.

RAJAN, J. V. AND THOMAS, D. E. 1985. Synthesis by delayed binding decisions. In *Proceedings of the 22nd Conference on Design Automation* (DAC). 367–373.

RAMAMRITHAM, K. AND STANKOVIC, J. 1994. Scheduling algorithms and operating system support for real-time systems. *Proc. IEEE 82*, 1 (Jan.), 55–67.

RATNER, R. S., SHAPIRO, E. B., ZEIDLER, H. M., WAHLSTROM, S. E., CLARK, C. B., AND GOLDBERG, J. 1973. Design of a fault tolerant airborne digital computer.

SCHWARTZ, D. B. 1993. ATM scheduling with queuing delay predictions. *SIGCOMM Comput. Commun. Rev. 23*, 4 (Oct. 1993), 205–211.

SHA, L. AND GOODENOUGH, J. B. 1990. Real-time scheduling theory and Ada. *IEEE Computer 23*, 4 (Apr. 1990), 53–62.

SHA, L., RAJKUMAR, R., AND LEHOCZKY, J. 1990. Real time scheduling support in Futurebus. In *Proceedings of the 11th IEEE Symposium on Real-Time*. IEEE Computer Society Press, Los Alamitos, CA, 331–340.

SHA, L., RAJKUMAR, R., AND SATHAYE, S. 1994. Generalized rate-monotonic scheduling theory: A framework for developing real-time systems. *Proc. IEEE 82*, 1 (Jan.), 68–82.

SHARMA, A. AND JAIN, R. 1993. Estimating architectural resources and performance for high-level synthesis applications. *IEEE Trans. Very Large Scale Integr. Syst. 1*, 2 (June), 175–190.

SOUKUP, J. 1981. Circuits layout. *Proc. IEEE 69*, 10 (Oct.), 1281–1304.

STANKOVIC, J. A., SPURI, M., NATALE, M. D., AND BUTTAZZO, G. C. 1995. Implications of classical scheduling results for real-time systems. *IEEE Computer 28*, 6, 16–25.

STEINMETZ, R. 1995. Analyzing the multimedia operating systems. *IEEE Multimedia 2*, 1, 68–84.

SUBRAHMANYAM, P. A. 1997. Embedded networked systems for an interactive world. *IEEE Computer 30*, 2, 92–93.

VAN CLEEMPUT, W. M. 1979. Hierarchical design for VLSI: problems and advantages. In *Proceedings of the Caltech Conference on VLSI*. 259–274.

WALKER, R. AND CAMPOSANO, R. 1991. *A Survey of High-Level Synthesis Systems*. Kluwer Academic Publishers, Hingham, MA.

WOLF, W. 1994. Hardware-software co-design of embedded systems. *Proc. IEEE 82*, 7 (July 1994), 967–989.

XIONG, X., BARROS, E., AND ROSENSTIEL, W. 1994. A method for partitioning UNITY language in hardware and software. In *Proceedings of the European Conference on Design Automation* (EURO-DAC '94, Grenoble, France, Sept. 19–23, 1994), J. Mermet, Ed. IEEE Computer Society Press, Los Alamitos, CA, 220–225.

YASUDA, H. 1997. Multimedia impact on devices in the 21st century. In *Proceedings of the IEEE International Conference on Solid-State Circuits*. IEEE Computer Society Press, Los Alamitos, CA, 28–31.

YEN, T.-Y. AND WOLF, W. 1995. Communication synthesis for distributed embedded systems. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-Aided Design* (ICCAD-95, San Jose, CA, Nov. 5–9), R. Rudell, Ed. IEEE Computer Society Press, Los Alamitos, CA, 288–294.

YEN, T. Y. AND WOLF, W. 1996. An efficient algorithm for scheduling in complex control. *IEEE Trans. Very Large Scale Integr. Syst. 4*, 1.

Received: January 1996; revised: October 1996; accepted: December 1997