A Methodology and Algorithms for the Design of Hard Real-Time Multi-Tasking ASICs[1]

Miodrag Potkonjak
Dept. of Computer Science, University of California, Los Angeles, CA 90095

Wayne Wolf
Dept. of Electrical Engineering, Princeton University, Princeton, NJ 08544

ABSTRACT

Traditional high-level synthesis concentrates on the implementation of a single task (e.g. filter, linear controller, A/D converter). However, current and future applications—multifunctional embedded controllers, intelligent wireless end-points, and DSP and multimedia servers—are often defined as sets of several computational tasks. This paper describes new techniques for the synthesis of ASIC implementations which 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 systems and behavioral synthesis techniques to derive increasingly detailed and accurate design solution. We have experimented with both optimal and heuristic algorithms to implement this methodology. The optimal algorithm uses several heuristics to speed-up average run time of an exhaustive branch-and-bound search. Force-directed optimization, which simultaneously considers influence of several cost components, such as the number of bits in the word-length, number of execution units, registers, and interconnects in an uniform and non-greedy fashion, is the core of the heuristic synthesis method. Analysis of the proposed algorithms and the conducted 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.

1. Preliminary versions of portions of this work appeared in the *Proc. IEEE/ACM ICCAD*, November 1995 and will appear in *Proc. 1996 IEEE ICASSP*, June 1996.

1.0 Introduction

1.1 Motivation

Traditionally, high-level synthesis has concentrated on the synthesis of machines which implement a single computational process. While early behavioral synthesis techniques concentrated on computations without control structures such as loops, conditionals, and subroutines [McF90a], 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 [Wal91, DeM94].

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 [Gib94]. While the number of ASIC designs slightly decreases each year, both their size and volume have increased rapidly [Fuj94]. For example, the average size, in terms of number of gates, of the integrated circuits has been doubling every three years, while the clock period has been steadily decreasing at 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 [Min94, page 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's 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 resulted just in ever 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 [Pat90]. To adequately address qualitatively new aspects of designs, new design methodologies are needed. In this paper we introduce an methodology which considers both intra and inter hardware and timing requirements of several hard real-time atomic tasks to produce globally optimized implementation.

1.2 The Synthesis Problem

We focus our attention on exploring synthesis and optimization issues during behavioral synthesis of application-specific systems at the process 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 problem for system of processes with deadlines specified as a set of periodic **tasks**. We use the term process in the conventional operating systems meaning—a single instance of a program, representing a single flow of control and set of input data values. However, our implementation of each process will be as a special-purpose ASIC with hard-wired control logic and a custom datapath. Each process is defined using 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 which 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 are available) and the **finish time** (the latest time by which 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 it is 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. This type of set of tasks and associated timing constraints is common in many modern DSP, communications, control and multimedia applications, such as intelligent wireless end-points, DSP, video, and multimedia servers, and embedded car and airplane controllers. The system has to be design so that all functionality and timing requirements are satisfied while the cost of implementation is minimized.

In the implementation, processes from different tasks may be implemented 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 a 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 design hard-real time design based on the strategy shown in Fig. 1. The detailed, algorithmic description of all phases of the approach and complete explanation of Fig. 1 are given in sections 4, 5, and 6.

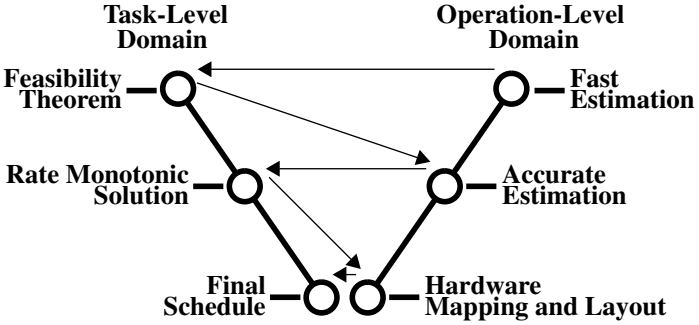

FIGURE 1. V-Chart of the new multiresolution synthesis approach.

Several of subproblems of the overall synthesis problem which must be solved during synthesis are NP-complete or of even higher computational complexity (see Section 3). This layered structure of computationally intractable problems is in addition such that 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 among tasks.

To handle the complexity of this problem, we developed a new, multiresolution iterative refinement synthesis strategy. The method refines an initial solution by employing increasingly accurate and increasingly CPU-intensive lower-bound estimations at the task-level and for individual processes. The estimates at both levels are obtained by relaxing some of constraints of the targeted problem, so that lower bound estimates or partial solution can be obtained very fast (low polynomial) time. At the task level, estimates are generated by rate-monotonic scheduling [Liu73] which provides mechanisms which guarantee that all the outlined timing constraints are simultaneously satisfied by examining the timing constraints for the individual processes, under simplifying assumption of no time and hardware overhead context switching between the processes. For the individual processes, timing and hardware requirements are estimated by the discrete-relaxation-based high-level synthesis tools Hyper and Hyper-LP [Rab94, Cha95]. These tools consider one type of hardware resource at the time to obtain sharp lower bounds on the required timing and hardware resources for the individual processes. As soon as one of estimates indicates that the proposed solution is infeasible, 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 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 a 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 [Rab91a] 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 in some number of groups. Note, that in the rest of the paper, with the exception of the related work section, we use word "partitioning" to denote that different subsets of tasks are implemented on different chips. Each process itself is treated as an atomic entity which is realized on only chip, which in general can be used for realization of some other processes as well. The search strategy is initially guided by a simple and fast estimation procedure which 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 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 for successful scheduling for tasks is estimated using the critical zone theorem [Liu73, Leh89] (see Section 2). If the analysis implies a feasible solution, the designs are implemented of 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 are obtained. During implementation it is allowed that 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) which share the common hardware platform is critical to the successful partitioning. The key technical optimization novelty is multidimensional force-directed algorithm which rapidly searchers the large solution space. The effectiveness of the approach is experimentally verified and the result 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, which considers and explores task-level concurrency and time-sharing in high-level synthesis. This consideration results in formulation and solution of a novel synthesis process of selecting subsets from a set of several applications which are 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 four main contributions are:

1. a new multiresolution iterative refinement synthesis methodology which provides a clean and effective mechanism for simultaneously address all important technical details and establish and maintain global information about the targeted designs;
2. establishment of the computational complexity of 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 which 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 which builds connections between two modern computer engineering domains, real-time systems and behavioral synthesis, by building connections from hard real-time systems scheduling disciplines to high-level synthesis and more importantly to ASIC implementation technology. This connection also bridges 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, key results provided by hard-real time process-level sequencing, and in particular rate-monotonic scheduling, research, and used 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 6 introduces the new two domain multiresolution iterative refinement design methodology which is used in two consequent section for development of optimal, branch-and-bound worst case exponential complexity optimization algorithm and force directed heuristic optimization algorithm. Section 7 presented experimental results respectively. 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.

2.0 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 cost.

Our computational model for a single process is homogenous synchronous data flow [Lee87]. 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 which periodically arrives. The design of the ASIC which executes these processes influences both the sampling period which can be supported for a process and the number of processes which 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 model 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 data-control 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 additional timing relationship among nodes. At higher level of hierarchy nodes encapsulate CDFG flow lower levels into procedures, loops, and if-then-else statements. The semantics underlying the syntax of the CDFG format, as we already stated, is one of 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, e.g., that the CDFG contains no indefinite loops. For great majority of DSP, control, and other application areas this assumption is usually satisfied.

We 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 [Rab91a] which targets 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 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 design use some version of register file models [Pat90], 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 implementation can be directly used in conjunction with high-level synthesis systems which use 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 subsection introduces hard-real time scheduling techniques which we use for synthesis of hard-real time multitask ASICs. The backbone of all estimations techniques is the rate-monotonic scheduling algorithm [Liu73]. 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 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 [Sha90].

Stankovic et al. [Sta95] provided an excellent survey of a number of real-time scheduling algorithms, their computational complexity, assumed timing and hardware models. We review a a few results of direct importance to our synthesis methodology. One of the first proposed real-time scheduling algorithm is earliest due-date Jackson's rule [Jac54]. Jackson's rule addresses the problem in which n independent processes, each with a specified processing time and due date, has to scheduled on a single machine. It states that the optimal solution to this problem, 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 [Sta95].

The rate-monotonic scheduling address 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 always the process with the current highest priority among the available processes is executed [Ram94]. The rate-monotonic algorithm assigns to each periodic process a single static priority level which 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 had 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 processes, the rate-monotonic scheduling algorithm will also provide a feasible schedule. rate-monotonic scheduling has been generalized in several direction. For example, the deadline-monotonic algorithm [Leu82] 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 which 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 can not be used. The constraint in rate-monotonic scheduling that all processes have fixed priority increases the likelihood (though it does not guarantee) 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. [Liu73, Sha90a].which we use to develop task-level timing estimation techniques.

Feasibility Theorem [Liu73]: A set of n independent periodic processes scheduled by the rate-monotonic algorithm will always meet its deadlines, for all task phasings, if

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

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

Critical Zone Theorem [Liu73]: 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.

# 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

Table 1: Efficiency bounds for resource utilization during application of rate-monotonic scheduling when different number of processes are executed on a single implementation platform

The feasibility theorem provides a sufficient (worst-case) condition which 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 1. illustrates the feasibility theorem when the number of processes varies from one to nine.

Essentially, what the feasibility theorem indicates that if one allocates enough hardware resources so that the run time of the periodic processes satisfy condition of the theorem, the feasible schedule is guaranteed, if static priority is assigned to all processes and preemption is allowed.

We will illustrate the feasibility theorem using the following example. Suppose that four processes are given with parameters as stated in Table 2. 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 which has duration 30 and task period 200, there is no guarantee that the feasible schedule exist, because the sum of efficiency bounds is now 0.8 (> 0.756). Note, that this not imply that the feasible schedule exist. The bound only indicates that 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 provides the mechanisms which guarantees feasible solution, once its conditions are satisfied. It is important to note that

task	task's duration (C)	task's 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

Table 2: Efficiency bounds for resource utilization during application of rate-monotonic scheduling when different number of processes are executed on a single implementation platform

the bound presented in the theorem is a conservative one. For example, it has been proven that for randomly selected set of processes the likely bound for a very large number of processes is 0.88 [Leh89]. To enable higher probability of accurately predicting the realistic resource utilization, we multiplied each value in Table 1 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 which is often more suitable for implementation than the criteria outlined in the initial version of the theorem.

Critical Zone Theorem, revised version [Leh89]: A set of independent periodic processes scheduled by the rate-monotonic algorithm will always meet its deadlines, for all processes phasings, if and only if

$$\forall i \ , 1 \le i \le n,$$

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

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

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

Detailed explanation of the critical zone theorem and illustrations of its application are given elsewhere [Leh89, Sha94]. We use this theorem in the second stage of our estimations dimension for estimating 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.

Finally, or the realization of the final schedule of all processes assigned to each of partitions, the least common multiple (LCM) theorem is used [Law82]. The theorem states that for a set of periodic processes, there exist a feasible non-preemptive schedule if and only if there exist 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 area of numerically intensive ASIC design is dominated by interconnect, control logic, and clock distribution requirements. This makes 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 [Rab94, Cha95]. 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 types, the number of interconnect and registers.

Chandrakasan et al. [Cha95] developed a techniques 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 after that 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 [Rab91a, Cha95].

3.0 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 problem and establish its computational complexity. First, we must define the synthesis 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: Partition the set of processes in N subsets ($N \leq M$) so that for each subset can be implemented on one dedicated application-specific multifunction integrated circuit. The cost of implementation (sum of areas of all chips) must be minimized.

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.

Interestingly, it is remarkably simple to show 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 in still 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 task have the identical start time 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 silicon area point of view one chips 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 [Gar79]. The first part of the proof, that the restricted problem indeed belong the class of NP-complete problem, 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 particular units can be schedule by just adding their number of operations which is proportional to their execution time. Note that any order of the task on both partitions is a valid task-level schedule.

For the second part of the proof we will use the Karp's polynomial transformations technique. We will transform a known NP-complete problem, equal subset problem [Gar79], into just 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 will 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 and which 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 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, our problem is at least as hard as the equal subset problem. As a result, the problem is NP-complete.

4.0 Synthesis and Optimization Approach: V-chart of Multiresolution Refinement in two Levels of Granularity

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 NP-complete problem by highly restricting the initial problem and proving that even then 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 itself is also an NP-complete problem [Gra77]. Actually, even when the minimization of only one type of hardware resource, such as execution units, is attempted, the subproblem is still NP-complete [Gar79]. Similarly, for circular-arc CDFGs (which depict lifetimes of the variables in the scheduled single process) register minimization is also NP-complete [Gar79]. At the task level, the constraint that no preemption is allowed results in NP-complete optimization problem

[Gra77]. It is also well known that optimization of one hardware component very often inevitably implies a need for additional resources of another type [Gue93]. 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 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 search by applying successively more accurate and computationally expensive models. The approach is shown in Fig. 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 two next 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 partitions satisfy the feasibility theorem. As soon as one of estimation indicates that the proposed solution is infeasible, 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 now, 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 non-feasible solutions are discarded.

Finally, in the two last visits to the two domains, the complete single process and task-level scheduling 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 are 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 was 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 replicates accordingly, if necessary, the number of iterations considered for each process. The second algorithmic addition orders the all considered iterations of all processes using the EDD rule. The additional hardware allocation or removal and the time distribution across the iteration is done automatically by the Hyper hierarchical scheduler (see [Pot92]).

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

5.0 Optimal Branch-and-Bound Synthesis Algorithm

Developing a strictly optimum algorithm for the synthesis of multiple hard-real 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 additional set of reasonable assumptions. Reasonable assumptions are assumptions which constraint the problem just to be addressed within the limits of today's computational resources and preserve all important aspects of the original problem so that generated solution represents 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 generate an optimal implementation for a single process and that our EDD-based task-level scheduler produces also optimal solutions. Our experimentation indicates that the assumptions are actually surprisingly often correct, since both the schedulers often produce the solutions which are either equal or close to lower bounds [Rab94, Ram94] Also, the approach would provide the optimum solution, if exponential run time schedulers are used for synthesis (e.g. ILP-based schedulers [Wal91]).

We now present an optimal branch-and-bound method augmented with several heuristic search space pruning strategies [Pap82]. Figure 2 illustrates the approach. and shows how the search space is enumerated. The algorithm first tries to clusters all 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 attempt we term an epoch. Figure 2a and 2b shows epochs which correspond to clustering into 2 and 3 groups respectively.

For a given number of used 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. Without loss of generality and optimality, in order to speed-up pruning, the processes with larger single

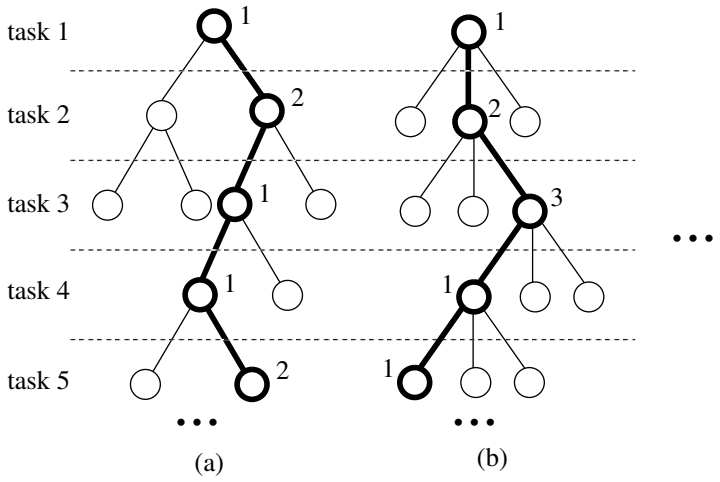

FIGURE 2. Branch and Bound-based Optimal Synthesis Approach: Representation of partial solutions (a) two processor synthesis epoch and (b) three processor synthesis epoch. For a detailed description of the branch-and-bound algorithm, see Section 5.

process implementation areas are assigned smaller numbers. The underlying idea is to check whether a particular solution 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 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 unmarked by the number which 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 a 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 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 and eventually the complete solution.

In order to speed-up the B&B 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 details on the results in Section 7.

6.0 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.

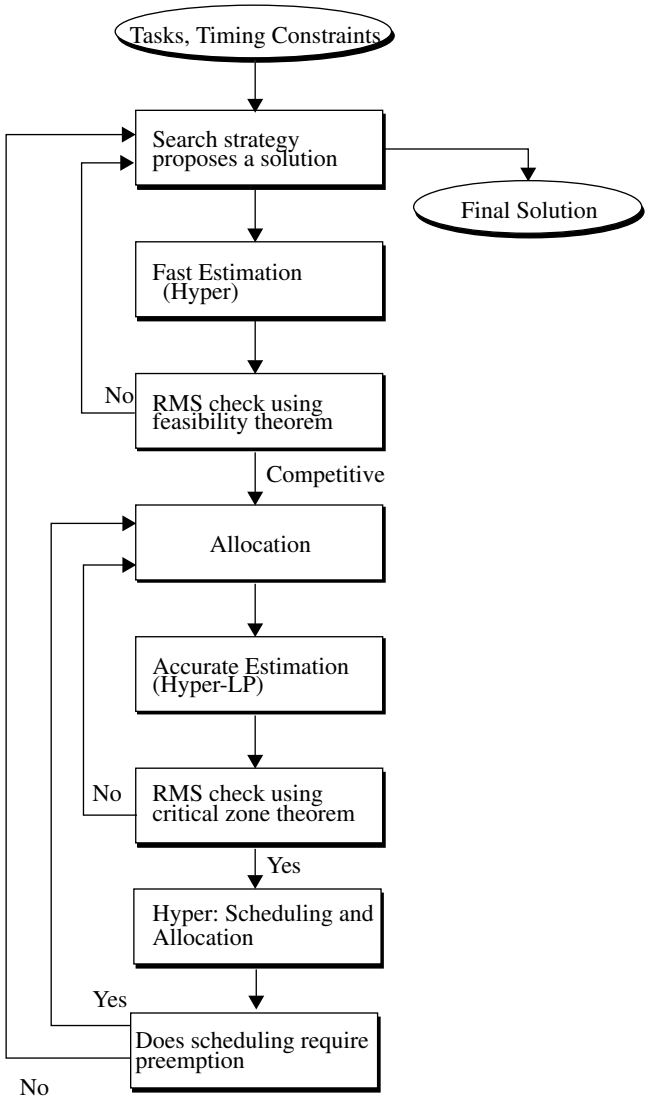

FIGURE 3. The design flow of heuristic synthesis algorithms for design of hard-real time multitasks applications. For detailed description of the corresponding optimization algorithm, see Section 6.

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 attempt to eliminate a need for some PEs. Since only two PEs are involved in a move, it is possible to determine if the reallocation is feasible.

```
RMS_synthesis() {
  find initial feasible solution with 1 process 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);
}
```

FIGURE 4. Outline of our synthesis algorithm

Our heuristic search algorithm iteratively applies two phases. The first phase selects a suitable expensive process and tries to merge that process' 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 a 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 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 the 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. *A similar number of registers.* The required number of registers is estimated using the Hyper estimation tools which calculate the maximum cutset of the dataflow graph [Rab94], as measured by the ratio of the estimated number of registers in the initial designs.

We tried several other criteria, e.g. 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 toward 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 have strong attraction forces. Since there are four independent components in each force, we termed this heuristic the multidimensional force-directed algorithm.

As we already mentioned earlier high-level synthesis tools and rate-monotonic critical zone scheduling algorithm and 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 very efficient method for predicting feasibility and chances for efficient area implementation of the final solution.

We modified the Hyper high-level synthesis tools to accommodate needs of try_to_merge (P1,P2) and update_solution() steps and produce area-efficient implementation. 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 [Rab91a], 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 non-preemptive 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.

7.0 Experimental Results

7.1 Description of Examples

Table 3 gives 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 tables 4 and 5. The examples are: GE Controller1 and GE Controller2 - two 4 and 5 state linear controllers; Honda Controller1 and Honda Controller2 - two linear mechanical controller each 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, 3 output linear controller for automotive motion control applications. We grouped these processes into tasks as shown in Table 4.

7.2 Results and Analysis

Table 4 also presents the solutions produced by our multidimensional force-directed based high-level synthesis algorithm. The second column in Table 4 gives the set of considered designs, the third column specifies the obtained solution, and the fourth column gives area in mm^2 for each design which implements the corresponding set of tasks from the third column. The final column gives the ratio of 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 median area reductions are by factors 2.02 and 1.98 respectively, clearly indicating the advantages of combining several processes on one ASIC implementation. The comparison between the two sets

Process #	Process	number of operations	number of bits	Initial Cost [mm^2]
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.	BandPass Filter	38	11	17.82
8.	High Pass Filter	42	14	18.24
9.	BandStop 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

Table 3: Individual Characteristics of Processes considered during experimentations

set of tasks	Processes in the set	Final Solution	Final Cost [mm^2]	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** (343.43/1 73.60)
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)

Table 4: Experimental result - Area of implementation for ASIC hard-real time designs using force-directed heuristics. For detailed explanation of the results see Section 7.

results indicates more than 11% improvement. 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 the similar type of operations. Neither of those two components can not be ignored, if the high quality solution is targeted. It is interesting to nota that in many cases, although new solution did not require any additional execution units and no new interconnects compared to the large design of the current designs, 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 has often 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 expressed.

8.0 Related Work

The directly related work can be mainly traced along the following five lines of research: high-level synthesis techniques; system level synthesis; scheduling 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 instructions sets (ASIP) and application-specific programmable processors (ASSP). An excellent summary and review of the early work on high-level synthesis is given in [McF90a]. More recently, several additional comprehensive surveys have been conducted, including exceptionally comprehensive one presented by Walker and Camposano [Wal91].

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 in both cases, regardless of differences in the semantic meaning of partitioning in two design scenarios, differences in considered level of abstraction, and different nature of timing constraints, the similar underlying principle is common in both cases. In both cases the goal is to identify subsets of computations which are well suited for hardware sharing after ASIC

implementation.

High-level synthesis partitioning techniques were pioneered by McFarland [McF83], Rajan and Thomas [Raj85], and Camposano and Brayton [Cam87]. All of them considered similarity of functions, common data carriers, and low level parallelism within single-stage clustering procedures. Consequently, Lagnese and Thomas [Lag89], generalized this work by considering multi-stage clustering and reported 20% reduction in the number of wiring tracks on a benchmark example.

Gupta and De Micheli [Gup90] compared the effectiveness of simulated annealing and heuristic approaches for partitioning of programmable and hardwired designs [Gup90]. Gebotys applied an ILP-based technique to obtain optimal solution for partitioning several popular high-level synthesis benchmarks [Geb92]. high-level synthesis group at the University of Southern California proposed several partitioning techniques for both high-level synthesis [Kuc91] and system level synthesis [Pra91, and at University of California, Irvine [Vah92, Rao92]

Recently synthesis of application-specific programmable processors (ASPP) [Bre90, Gue93] and application-specific instruction sets processors (ASIP) [Leu94, Pau94, Goo95] received a great deal of attention in CAD community. While both ASPPs and ASIPs and the technique proposed in this paper target implementation of several processes on the same processor, the similarity between 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 the ASPP or ASIP design will be eventually used to realize only one of several (or many) applications, hard real-time rate-monotonic scheduling-based ASIC design enable that several processes 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 functional and RT-level was first and most clearly pointed out pointed out by McFarland and Kowalski [McF90b]. Rabaey and Potkonjak [Rab91b] and Sharma and Jain [Sha93] 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 the time. Chaudhury and Walker developed similar techniques which derive lower bounds considering several types of operations simultaneously [Cha94].

Most high-level synthesis layout estimators are at least partially constructive and therefore timing consuming [Kur93]. Recently, Chandrakasan et al. [Cha95] 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 attentions [Wol94]. The most relevant system research subdomain is hardware-software partitioning [Bar94,Ern93, Gaj94, Gup93, Ism94]. The goal of software partitioning is to identify parts of computations which should be implemented on programmable and ASIC platform so that an overall optimization function is maximized. However, again as in the case of high-level synthesis partitioning the same set of key difference exist between hardware-software partitioning and the rate-monotonic scheduling-based high-level synthesis.

Hard-real time scheduling efforts has 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 [Liu73]. Consequently, rate-monotonic scheduling has been extensively analyzed and generalized, mainly by researchers at Carnegie-Mellon university [Leh89, Sha90a, Sha94]. The most notable practical application of real-time scheduling approaches, an 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 [IEE91, IEE93, Sha90b], and use of the generalized rate-monotonic scheduling techniques in several major advance-technology projects such as Space Station Program and the European Space Agency on-board operating system [Sha94]. 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 [Nag92, Ram94, Ste95] and ATM switch scheduling [Swa93] further emphasis 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 widely studied topic [Alm94].

Scheduling under a complex set of timing constraints was studied by Ku and De Micheli [Ku92], Cho et al [Cho94], and by Yen and Wolf [Yen96].

Before concluding our survey of the related research efforts with design methodology and algorithmic issues, we refer to three recent publications in the CAD literature which addressed use of the rate-monotonic scheduling algorithms. Hu et al. [Hu94] compared several hard real time scheduling policies during hardware-software partitioning of the controller for automotive powertrain module, but did develop synthesis approach or conduct any synthesis optimization. In the 1995 EDAC conference paper Atlenbernd [Atl95] generalized deadline monotonic scheduling policy along several lines, but also did not make connection to the synthesis process. Recently Yen and Wolf addressed synthesis of distributed embedded systems with both hard

and soft deadlines [Yen95]. While all hard deadline must be satisfied, the optimization goal is to satisfy as many as possible soft deadlines.

It appears that van Cleemput was first to formally discuss the hierarchical design process along three different dimensions as three concurrent tasks: behavioral, structural, and physical design [Cle79]. Gajski and Kun used those three dimensions as three dimensions of Y-chart of the design process along which design refinement is conducted [Gaj83]. 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 represent single process domains, the refinement goes from the functional to geometric representation, while along 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 [Sou81], who used it as a key optimization step in the epitaxial growth algorithm for constructive placement. Paulin and Knight [Pau89] developed a conceptually very similar approach for force-directed scheduling, which due to its clear intuitive foundations have been very popular in behavioral synthesis research [McF90, Wal91].

9.0 Future Research and Conclusion

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

1. creation of an impetus for comprehensive treatment of interaction between synthesis process, an in particular high-level synthesis, and requirements imposed by operating systems; and
2. study of synthesis techniques for exploration of hardware sharing at the process level.

The major goals form a basis for several extensions. Two conceptually most 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 alterative optimization goals include power, testability assuming different testing strategies, and fault-tolerance overhead minimization. Important alternative implementation platforms include variety of programmable platforms (e.g. microprocessors, microcontrollers, DSP and video processors), ASIPs and ASPPs, and 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 level 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 synthesis process

Also, it will be interesting to address impact of other high-level synthesis tasks, such transformations and template matching, on both task-level hardware sharing and interaction between scheduling policies of operating systems.

We will conclude the outline of future research directions by pointing out that numerous research avenues are open also if one considers in addition to the static non-preemptive scheduling schemes other frequently used operating system policies, such as priority ceiling, round-robin, first-come first-serve [Sha94].

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 novel iterative refinement two domain optimization approach. The key algorithmic novelty is an introduction of 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, that synthesis of multiple-process systems will become increasingly important.

Acknowledgments

W. Wolf was supported by the National Science Foundation under grant MIP-9424410.

10.0 References

[Alm94] G.S. Almasi, A. Gottlieb, "*Highly Parallel Computing"*, The Benjamin/Cummings, Redwood City, CA.

[Atl95] P. Atlenbernd, "Deadline-Monotonic Software Scheduling for the Co-Synthesis of Parallel Hard Real-Time Systems", pp. 190-195, *EDAC-95*, 1995.

[Bar94] E. Barros, W. Rosenstiel, and X. Xiong, "A method for partitioning UNITY language in hardware and software," in *Proceedings, EuroDAC '94*, pp. 220-225, IEEE Computer Society Press, 1994.

[Bre90] M. Breternitz, J.P. Shen, "Architecture Synthesis of High performance Application-Specific Processors, *27th DAC,* pp. 542-447, 1990.

[Cam87] R. Camposano, R.K. Brayton, "Partitioning Before Logic Synthesis", *ICCAD'87*, pp. 324-326, 1987.

[Cha94] S. Chaudhury, R.A. Walker, "Computing Lower Bounds on Functional Units before Scheduling", *7th high-level synthesis Symposium,* pp. 36-41, 1994.

[Cha95] A.P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, R. Brodersen, "Optimizing Power Using Transformations", I*EEE Transactions on CAD*,Vol. 14, No. 1, pp. 13-32, January 1995.

[Cle79] W.M. van Cleemput, "Hierarchical Design for VLSI: Problems and Advantages", Caltech Conference on VLSI, pp. 259-274, 1979.

[Cho94] P. Chou, E.A. Walkup, G. Borrielo: "Scheduling for Reactive Real-Time Systems", *IEEE MICRO*, Vol. 14, No. 4, pp. 37-47, 1994.

[Ern93] R. Ernst, J. Henkel, and Th. Benner, "Hardware-software co-synthesis for microcontrollers," *IEEE Design & Test of Computers*, Vol. 10, No. 4, pp. 64-75, December 1993.

[Fuj94] H. Fujinami, NEC Corporation, Personal Communication, November 1994.

[Gaj83] D.D. Gajski, R.H. Kun, "Guest Editor's Introduction: New VLSI Tools", *IEEE Computer*, Vol. 16, No. 12, pp. 11-14, 1993.

[Gaj94] D. D. Gajski, F. Vahid, and S. Narayan, "A System-Design Methodology: Executable Specification Refinement," in *Proceedings, Euro-DAC '94*, pp. 458-463, IEEE Computer Society Press, 1994.

[Gar79] M.R. Garey, D.S. Johnson, *"Computers and Intractability: A Guide to the Theory of NP-Completeness"*, W.H.Freeman and company, New York, 1979.

[Geb92] C.H. Gebotys, "Optimal Synthesis of Multichip Architecture", *ICCAD'92*, pp. 238-241, 1992.

[Gib94] W.W. Gibbs, "Software's Chronic Crisis", *Scientific American*, pp. 86-95, September 1994.

[Goo95] G. Goossens, et. al., "Integration of Medium-Throughput Signal Processing Algorithms on Flexible Instruction-Stes Architectures", *Journal of VLSI Signal Processing,* Vol. 9, No. 1-2, pp. 49-65, 1995.

[Gra77] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, "Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey", Annals of Discrete Mathematics, Vol. 5, pp. 287-326, 1979.

[Gue93] L. Guerra, M. Potkonjak, J. Rabaey, "High-level synthesis for Reconfigurable Datapath Structures", *ICCAD93 International Conference on Computer-Aided Design*, pp. 26-29, November 1993.

[Gup90] R. Gupta, G. De Micheli, "Partitioning of Functional Models of Synchronous Digital Systems, *ICCAD90 International Conference on Computer-Aided Design*, pp. 216-219, 1990.

[Gup93] R. K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design & Test of Computers*, 10(3), September, 1993, pp. 29-41.

[Hu94] X. Hu, J.G. D'Ambrosio, B.T. Murray, D-L. Tang: "Codesign of Architecture for Automotive Powertrain Modules", *IEEE MICRO*, Vol. 14, No. 4, pp. 17-25, 1994.

[IEE91] IEEE, *"Real-Time extensions to POSIX"*, IEEE Std. P. 1003.4, IEEE, New York, NY, 1991.

[IEE93] IEEE, *"Futurebus+ Recommended Practice"*, IEEE Std. 896.3, IEEE, New York, NY, 1993.

[Ism94] T. B. Ismail, K. O'Brien, and A. Jerraya, "Interactive System-Level Partitioning with PARTIF," n *Proceedings, Euro-DAC '94*, IEEE Computer Society Press, 1994, pp. 464-468.

[Jac54] J.R. Jackson, "Scheduling a Production Line to Minimize Maximum Tardiness", Research Report 43, Management Science Research Project, University of California, Los Angeles, CA.

[Ku92] D.C. Ku, G. De Micheli: "Relative Scheduling Under Timing Constraints: Algorithms for High-Level synthesis of Digital Circuits", *IEEE Trans. on Computer-Aided Design,* No. 6, Vol. 11, pp. 696-717, 1992.

[Kuc91] K. Kucukcakar, A.C. Parker, "CHOP: A Constraint-Driven System-Level Partitioner", *28th DAC,* pp. 514-519, 1991.

[Kur93] F. Kurdahi, C. Ramachandran, "Evaluating Layout Area Tradeoffs for High Level Applications", *IEEE Transactions on VLSI Systems,* Vol. 1, No. 1, pp. 46-55, 1993.

[Lag89] E.D. Lagnese, D.E. Thomas, "Architectural Partitioning for System Level Design", *26th DAC,* pp. 62-67, 1989.

[Law81] E.L. Lawler, C.U. Martel, "Scheduling periodically occurring tasks on multiple processors", Information Processing Letters, Vol. 12, No. 1, pp. 9-12, 1982.

[Lee87] E.A. Lee and D.G. Messerschmitt, "Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing." *IEEE Transactions on Computers*, Vol. 36, No. 1, pp. 24–35, 1987.

[Leh89] J.P. Lehoczky, L. Sha, Y. Ding, "The Rate Monotonic Scheduling Algorithms - Exact Characterization and Average Case Behavior" *IEEE Real-Time System Symp.*, pp. 181-191, 1986.

[Leu82] J. Leung, J. Whitehead, "On the complexity of Fixed Priority Scheduling of Periodic, Real-Time Tasks, Performance Evaluation, Vol. 2, No. 4, pp. 237-250, 1982.

[Leu94] R. Leupers, W. Schednk, P. Marwedel, "Retargetable Assembly Code Generation by Bootstrapping", *Internation Symposium on high-level synthesis,* pp 88-93, 1994.

[Liu73] C.L. Liu, J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard Real Time Environment", *Journal of ACM*, Vol. 20, No. 1, pp. 46-61, 1973.

[McF83] M.C. McFarland, "Computer-Aided Partitioning of Behavioral Hardware Descriptions", *20th DAC*, pp. 472-480, 1983.

[McF90a] M.C. McFarland, A.C. Parker, R. Camposano: "The High-Level Synthesis of Digital Systems", *Proceedings of the IEEE*, Vol. 78, No. 2, pp. 301-317, February 1990.

[McF90b] M.C. McFarland, T.J. Kowalski: "Incorporating Bottom-Up Design into Hardware Synthesis", *IEEE Trans. on CAD*, Vol. 9. No. 9, pp. 938-950, September 1990.

[Min94] D. Minoli, R. Keinath, "*Distributed Multimedia Through Broadband Communications Services*", Artech House, Boston, MA 1994.

[Nag92] R. Nagarajan, C. Vogt, "Guaranteed Performance of Multimedia Traffic over the Token Ring", Tech. Report No. 439210, IBM-ENC, Heidelberg, Germany, 1992.

[Pap82] C.H. Papadimitrou, K. Steiglitz, "Combinatorial Optimization: Algorithms and Complexity", Prentice-Hall, Englewood Cliffs, NJ, 1982.

[Pat90] D.A. Patterson, J.L. Hennessy, *Computer Architecture: A Quantitative Approach,* Morgan Kaufmann Publishers, San Mateo, CA, 1990.

[Pau89] P.G. Paulin, J.P. Knight, "Force-directed scheduling for the behavioral synthesis of ASICS", *IEEE Transactions on CAD,* Vol. 8, No. 6, pp. 661-679, June 1989.

[Pau94] P.G. Paulin, C. Liem, T.C. May, S. Sutarwala, "CodeSyn: A Retargetable Code Synthesis System", *Internation Symposium on high-level synthesis*, page 94, 1994.

[}pt92] M. Potkonjak, J. Rabaey, "Scheduling Algorithms for Hierarchical Data Control Flow Graphs", *International Journal of Circuits Theory and Applications*, Vol. 20, No. 3, pp. 217-234, 1992.

[Pot94] M. Potkonjak, J. Rabaey, "Algorithm Selection: A Quantitative Computation-Intensive Optimization Approach", *ICCAD94 International Conference on Computer-Aided Design*, pp. 90-95, November 1994.

[Pot95] M. Potkonjak, W.H. Wolf, "Cost Optimization in ASIC implementation of Periodic Hard-Real Time Systems using Behavioral Synthesis Techniques", *ICCAD95 International Conference on Computer-Aided Design*, pp. 446-451, 1995.

[Pra91] S. Prakash, A.C. Parker, "Synthesis of Application-Specific Multiprocessor Architectures", *28th DAC,* pp. 8-13, 1991.

[Rab91a] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures", *IEEE Design and Test of Computers*, Vol. 8, No. 2, pp. 40-51, June 1991.

[Rab91b] J. Rabaey, M. Potkonjak, "Complexity Estimations for Real Rime Application Specific Circuits", *17th European Solid-State Circuits Conference*, Milan, Italy, pp. 201-204, September 1991.

[Rab94] J. Rabaey, M. Potkonjak, "Estimating Implementation Bounds for Real Time Application Specific Circuits", *IEEE Transaction on CAD*, Vol. 13, No. 6, pp. 669-683, June 1994.

[Raj85] J.V. Rajan, D.E. Thomas, "Synthesis by Delayed Binding Decisions", *22nd DAC,* pp. 367-373, 1985.

[Ram94] K. Ramamritham, J.A. Stankovic: "Scheduling Algorithms and Operating System Support for Real-Time Systems", Proc. of the IEEE, Vol. 82, No. 1, pp. 55-67, January 1994.

[Rao92] D.S. Rao, F.J. Kurdahi, "Partitiong by Regularity Extraction", *29th DAC,* pp. 235-238, 1992.

[Sha90a] L. Sha, J.B. Goodenough: "Real-Time Scheduling Theory and Ada", *IEEE Computer*, Vol. 23, No. 4, pp. 53-62, April 1990.

[Sha90b] L. Sha, R. Rajkumar, J. Lehoczky, "Real Time Scheduling support in Futurebus+", *IEEE 11th Real-Time Systems Symposium,* pp. 331-340, 1990.

[Sha94] L. Sha, R. Rajkumar, S.S. Sathaye: "Generalized Rate-Monotonic Scheduling Theory: A Framework for Developing Real-Time Systems", *Proc. of the IEEE*, Vol. 82, No. 1, pp. 68-82, January 1994.

[Sha93] A. Sharma, R. Jain, "Estimating Architectural Resource and Performance for high-level synthesis Applications", *IEEE Trans. on VLSI*, Vol. 1, No. 2, pp. 175-190, 1993.

[Sou81] J. Soukup, "Circuits Layout", *Proc. of the IEEE*, Vol. 69, No. 10, pp. 1281-1304, October 1981.

[Sta95] J.A. Stankovic, M. Spuri, M. Di Natale, G.C. Buttazzo, "Implications of Classical Scheduling Results for Real-Time Systems, IEEE Computer, Vol. 28, No. 6, pp. 16-25, June 1995.

[Ste95] R. Steinmetz, "Analyzing the Multimedia Operating Systems", *IEEE Multimedia,* Vol. 2, No. 1, pp. 68-84, 1995.

[Swa93] D.B. Schwartz, "ATM Scheduling with Queueing Delay Predictions", *SIGCOMM'93 Communications Architectures, Protocols and Applications,* pp. 205-211, 1993.

[Vah92] F. Vahid, D.D. Gajski, "Specification Partitioning for System Design", *29th DAC,* pp. 219-224, 1992.

[Wal91] R.A. Walker, R. Camposano: "*A Survey of high-level synthesis systems*", Kluwer Academic, Norwell, MA, 1991.

[Wol94] W.H. Wolf: "Hardware-Software Co-Design of Embedded Systems", *Proceedings of the IEEE,* Vol. 82, No. 7, pp. 967-989, July 1994.

[Yen95] T.-Y. Yen, W. Wolf, "Communication Synthesis for Distributed Embedded Systems", *ICCAD95 International Conference on Computer-Aided Design*, pp. 288-294, November 1995.

[Yen96] T. Y. Yen and W. Wolf, "An efficient algorithm for scheduling in complex control," *IEEE Transactions on VLSI Systems*, 4(1), February 1996.