Cost Optimization in ASIC Implementation of Periodic Hard-Real Time Systems using Behavioral Synthesis Techniques

Miodrag Potkonjak†
C&C Research Laboratories
NEC USA
Princeton, NJ 08540

Wayne Wolf
Dept. of EE
Princeton University
Princeton, NJ 08540

Abstract

Modern applications are often defined as sets of several computational tasks. This paper presents a synthesis algorithm for ASIC implementations which realize multiple computational tasks under hard real-time deadlines. The algorithm analyzes constraints imposed by task sharing as well as the traditional datapath synthesis criteria. In particular, we demonstrated an efficient technique to combine rate-monotonic scheduling, a widely used hard real-time systems scheduling discipline, with estimations and scheduling and allocation algorithms. Matching the number of bits in tasks assigned to the same processor was the most important factor in obtaining good designs. We have demonstrated the effectiveness of our algorithms on several multiple-task examples.

1 Introduction

Until now high level synthesis has concentrated on the synthesis of a single computational task [McF90]. In this paper we introduce the first high-level synthesis algorithm for the creation of **multi-task application-specific systems**. By using information provided by **hard-real time scheduling** methodologies and **behavioral synthesis** tools, we connect the synthesis process to operating systems methodologies and technologies and enable **efficient sharing of hardware by several tasks**.

We target hardware design problem for systems of processes with deadlines is specified as a set of periodic **tasks**. Each task is defined using control-data flow graph and the set of timing constraints. For each task three timing constraints are imposed: **period interval**, 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). As in much real-time scheduling work, we assume that the finish time for each task is the end of its period.

The synthesis goal is to partition the set of tasks in an arbitrary number of subsets so that for each subset can be implemented on one dedicated multifunctional ASIC. All timing constraints for all tasks must be satisfied. The partitioning is conducted in a such a way that the area of the ASIC is minimized. We developed a search strategy which partitions the tasks into groups. In our methodology, "partitioning" denotes that different subsets of tasks are implemented on different chips. Each partition is implemented by a single datapath/controller machine; tasks are executed one at a time on the datapath, with the highest-priority active task being executed. The search strategy is guided by a simple and fast estimation procedure which predicts the required hardware and corresponding time resources for each task.

In the next section we outline the terms and models used in our work. In Section 3 we define the targeted synthesis problem and introduce the rank-order-based optimization algorithm, while Section 4 presents experimental results. We have deferred the comparison of our work until Section 5, since our problem makes use of results from several different disciplines.

†Miodrag Potkonjak is now with UCLA CS Dept.

2 Preliminaries

2.1 Computational and Hardware Models

Our computational model for a single task is synchronous data flow [Lee87]. The model has two important ramifications. First, the proper speed metric for a synchronous data-flow task is the sampling period with which input data can be processed. At the system level the appropriate metrics of speed is how many tasks with a given periodicity can be accepted so that the constraints imposed by their sampling periods are satisfied. Second, the tasks are well behaved in the sense that one can impose an upper limit of the execution time of each tasks and tasks can be statically scheduled.

Each task is defined as a hierarchical data-control flow graph (CDFG). We assume it is possible to derive upper bounds on the execution of each task on the available hardware. We do not impose any restriction on the assumed hardware model. For implementation we used the Hyper high level synthesis system [Rab91] which targets dedicated register file model.

2.2 Rate Monotonic Scheduling

Rate-monotonic scheduling (RMS) theory addresses the problem of ensuring that independent periodic tasks are scheduled without violation of the associated timing constraints. Tasks are independent if their correct execution does not imply need for synchronization. Note that tasks can be independent even when they interchange data, so long as the interchange does not cause a task to block.

The real-time scheduling basis for our work is found in the following two theorems [Liu73, Sha90].

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

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

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 tasks, if each task meets its first deadline when all tasks are started at the same time, then the deadline will always be met for any combination of start times.

# of tasks	**1**	**2**	**3**	**4**	**5**
U(n)	1.0	0.828	0.779	0.756	0.743

Table 1. Efficiency bound U(n) for resource utilization under RMS as a function of number of tasks.

The feasibility theorem guarantees a sufficient condition for any distribution of start times that can arise when the rate monotonic scheduling policy is applied. As the number of tasks approaches infinity, the bound converges to 0.69 (ln 2). Table 1 gives the utilization bound for small number of tasks sharing one processing element. If the execution engine is fast enough that the run time of the periodic tasks satisfy the conditions of the theorem, a feasible schedule is guaranteed, regardless of the task start times.

We will illustrate the feasibility theorem using the following example. Suppose that four tasks are given with parameters as stated in Table 2. These tasks 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) as indicated by the feasibility theorem.

task	**duration (C)**	**period (T)**	**Utilization (U = C/T)**
task 1	40	160	0.25
task 2	15	75	0.2
task 3	30	150	0.2
task 4	10	100	0.1

Table 2. An example set of tasks.

The feasibility theorem is a basis for the synthesis scenario when the start times of tasks are not known *a priori*. It provides the mechanisms which guarantees feasible solution, once its conditions are satisfied. The bound presented in the theorem is a conservative one. For example, it has been proven that for randomly selected set of tasks the likely bound for a very large number of tasks is 0.88 [Leh89]. So, 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 tasks scheduled by the rate monotonic algorithm will always meet its deadlines, for all tasks phasings, if and only if

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

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

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

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

2.3 Area Prediction from a CDFG

We use estimators at three different levels of accuracy for predicting the area of a datapath. The fastest estimation is provided by the estimation tools from high level synthesis system Hyper [Rab91]. 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 task. For area estimation we use two techniques: fast and elaborated. Both techniques use the Hyper set of estimation tools [Rab91, Cha95]. The fast technique predicts the number of instances of hardware primitives at RT-level using min-bound technique. Consequently, it uses the Hyper-LP statistical model to estimate the area of implementation. The elaborated technique first conducts 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 first technique is about 20% less accurate than the second, but it is an order of magnitude faster.

3 The algorithm

The synthesis problem for rate monotonic scheduling-based high level synthesis of hard real time systems can be defined as follows:

Given: A set of M tasks described by their CDFGs and their hard real-time timing constraints (task period, start and finish time). The finish time is the end of the task period, and the start time can take an arbitrary value. It is assumed that all tasks must be implemented under the hard real-time constraints. *Goal:* Partition the set of tasks in N subsets (N <= M) so that for each subset can be implemented on one dedicated programmable chip. The cost of implementation (sum of areas of all chips) must be minimized. It can be shown that even in the cases when the computational complexity of the associated high level synthesis tasks are of polynomial complexity, the synthesis of minimal cost implementation of a set of real-time asks is NP-hard.

To describe our synthesis algorithm, we first present the search strategy, which is used to propose a partitioning of tasks in several groups, so that each group is implemented on one ASIC. We next explain how the critical zone theorem is used to obtain a task level schedule. Finally, we describe modifications to traditional high level synthesis tools for the new application domain.

The outline of the algorithm for the synthesis of set of tasks under hard-real time constraints is given in Figure 1. Our algorithm 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 processor elements (PE); it is easy to determine in this case if there is a feasible solution. During synthesis, each step moves the task from one PE to another. Since only two PEs are involved in a move, it is possible to determine if the reallocation is feasible. In general, testing an arbitrary configuration for feasibility is NP-complete.

Our algorithm iteratively applies two phases. The first phase selects an expensive process and tries to merge that process' PE with another PE. This type of move tries to choose small, inexpensive processes to share the expensive process's PE. The second phase tries to merge a process which has a large slack in meeting its deadline. Processes with large slacks are more likely to be able to be executed on another PE, so this move helps reduce cost.

```
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 1. Outline of our synthesis algorithm

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

1. **The bit width of the operations in the process.** Hardware is wasted if two processes operate on data of different bit widths. Also, longer bit width implies longer cycle time. Matching a task to a PE with function units of the proper bit width reduces the clock period.

2. **The similarity of the type of functional units required by the processes.** If two tasks require the same type of execution units, it is more likely that their combined realization will be smaller than the sum of the individual implementations. In particular, it is important to match tasks which require hardware and time expensive units such as multipliers.

3. **The sources and sinks of data transfers.** Similarly, it is important to match tasks which have similar communication patterns. For example, if two processes both send the output of a multiplier to an adder, they have more in common than a process which sends its multiplier output to an adder and another which sends the multiplier output to a subtracter.

4. **A similar number of registers.** This can be estimated in the general case by finding the maximum cutset of the dataflow graph. Hyper provides facilities for estimation of all three mentioned types of hardware resources (execution units, interconnect, and registers) from a CDFG.

5. **The sum of their current ratios of time they use the current set of allocated hardware resources.** Two processes whose have high ratios will contend for the existing resources, requiring addition of extra resources.

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(). This process is done in two steps: first using the fast estimation procedure and if this test is satisfied using the elaborated 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 based on two key results. The key idea of the corresponding scheduling algorithm is to always schedule first among tasks which are available for scheduling task with the earliest deadline. Rate-monotonic analysis guarantees the optimality of the produced schedule[Liu73].

High level synthesis tools are modified in the following way to accommodate needs of try_to_merge (P1,P2) and update_solution() steps. The number of bits for all tasks is set to the number of bits in the highest word length requirements among individual tasks. All processes are treated as subroutines during scheduling, and Hyper optimally allocates processes [Rab91], the available time among the individual tasks. If required, the hierarchical scheduler allocates additional hardware resources after merging of two tasks. However, in this case the decision to update the current solution is postponed until at least three other proposed solutions are examined. After scheduling, the required timing requirements for each individual task are recorded and their combination is checked using the rate monotonic scheduling theorems. The solution is updated only after a feasible task-level rate-monotonic schedule is obtained. During scheduling at the task level we do not allow preemption of tasks. This is done in order to eliminate the need for an expensive model of context switching among tasks. Although, in this way the optimality of the rate monotonic scheduling can not be guaranteed anymore, the final achieved results are most often fully satisfactory. Both our experiments and previously published theoretical and experimental results [Sha90] indicate that preemption during rate monotonic scheduling is rarely required and used, even when allowed. More importantly, we provide a mechanism for rescheduling of all tasks until a need for preemption is not eliminated.

4 Experimental Results

Table 3 gives descriptions of 16 tasks used to construct five different task sets, including the number of operations, the word length, and the initial area when each task is implemented on a separate chip. We used these tasks to construct the task sets used for the experiments. Table 4 presents the solutions produced by our rank-order based high level synthesis algorithm. Both the average and the median area reductions are by factors slightly larger than two, clearly indicating advantage of combining several tasks on one ASIC implementation.

The more detailed analysis of obtained solutions indicates the following interesting facts. The best solutions tend to group tasks which require the similar number of bits in their word length. 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. This can be mainly attributed to increase in the required numbers of registers in background ROM memory. The increase in the number of registers was mainly due to a need to store constants 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 a significant overall impact.

Finally note that in the modern implementation technologies pin count is an important cost criterion. In all examples, the number of pins in final designs was equal to the word-length used in design.

5 Related Work

The directly related work can be mainly traced along the following three lines of research: high level synthesis; system level synthesis; and scheduling in hard real-time systems.

Within high level synthesis several subdomains are related to the research described in this paper, including partitioning, estimations and area prediction, and design of application-specific instructions processors (ASIPs) and application-specific programmable processors (ASSPs).

High-level synthesis partitioning techniques were pioneered by McFarland [McF83] and Camposano and Brayton [Cam87]. 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.

Recently, synthesis of ASPPs [Gue93] and ASIPs [Leu94, Goo95] received a great deal of attention in CAD community. While both ASPPs/ASIPs and the technique proposed in this paper target implementation of several tasks on the same processor, the similarity between two domains is very limited. For example, while both ASIP and ASPP designs assume that the final design will be eventually used to real-

Table 3. Individual Characteristics of Tasks considered during experimentations

Task #	**Task**	**# operations**	**# bits**	**Initial area (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 4. Experimental result - Area of implementation for ASIC hard-real time designs.

set of tasks	**Tasks in the set**	**Final solution**	**Final area (mm^2)**	**Improvement initial/final**
set 1	{1,2,3,4,5,6,7,8}	{1,5,6,7} {2,3,4,8}	26.91+61.98	1.86
set 2	{9,10,11,12,13,14,15,16}	{9,10,11,12,13} {15} {14,15, 16}	60.70+ 98.80	2.47
set	{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	2.14
set 4.	{5,6,7,8, 9,10,11,12,13,14,15,16}	{5,6,7,8,9,11} {10,12,13} {14, 15,16}	52.77 +54.09 + 98.80	1.98
set 5	{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

ize only one of several (or many) applications, hard real-time rate monotonic scheduling-based ASIC design allows several tasks to share the same hardware during their execution.

Hardware-software codesign has received a great deal of attention recently [Wol94]. The most relevant system research subdomain is hardware-software partitioning [Bar94, Ern93, Gup93, Vah92]. These algorithms try to identify parts of computations which should be implemented on programmable and ASIC platform so that an overall optimization function is maximized. However, they do not address use and influence of hard real-time operating scheduling constrains and operating systems principles to optimize the implementation of their ASICs.

Hard-real time scheduling efforts are more than three decades long. 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, the rate-monotonic scheduling has been extensively analyzed and generalized in several directions, mainly by researchers at Carnegie-Mellon University [Leh89, Sha90].

The most notable practical application of real-time scheduling approaches, an in particular rate monotonic, include the inclusion of rate monotonic scheduling as the scheduling policy for the IEEE POSIX real-time operating system standard and IEEE Futurebus+ standards [IEE93], 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.

Finally, Hu et al. [Hu94] compared several hard real time scheduling policies during hardware-software partitioning of the controller for automotive powertrain module, but did not develop a synthesis algorithm.

6 Conclusions

This paper has introduced synthesis of hardware implementations of multi-task ASICs. Our methodology integrates techniques from operating systems—namely, rate-monotonic scheduling—and high level synthesis.We use a steepest-descent rank-order-based algorithm to optimize the design of the shared datapath system. Matching the number of bits for the tasks selected to be implemented on the same platform is critical to cost optimization. ASICs which can execute multiple tasks are important tools for real-time synthesis. We believe that this synthesis methodology will be useful in the co-synthesis of hard real-time systems.

Acknowledgments

This work was supported in part by the National Science Foundation under grant MIP-9424410.

7 References

[Bar94] E. Barros, W. Rosenstiel, and X. Xiong, “A method for partitioning UNITY language in hardware and software,” *EuroDAC ‘94*, pp. 220-225, 1994.

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

[Cha95] A.P. Chandrakasan, et. al.“Optimizing Power Using Transformations”, Vol. 14, No. 1, pp. 13-32, I*EEE Transactions on CAD,* 1995.

[Ern93] R. Ernst, J. Henkel, and Th. Benner, “Hardware-software co-synthesis for microcontrollers,” *IEEE Design & Test of Computers*, 10(4), 1993.

[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.

[Gue93] L. Guerra, M. Potkonjak, J. Rabaey, “High Level Synthesis for Reconfigurable Datapath Structures”, *ICCAD93*, pp. 26-29, 1993.

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

[Hu94] X. Hu, et al, “Codesign of Architecture for Automotive Powertrain Modules”, *IEEE MICRO,* Vol. 14, No. 4, pp. 17-25, 1994.

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

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

[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.

[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.

[McF90] 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, 1990.

[Pot94] M. Potkonjak, J. Rabaey, “Algorithm Selection: A Quantitative Computation Intensive Optimization Approach”, *ICCAD94*, pp. 90-95, 1994.

[Rab91] J. Rabaey, et al.“Fast Prototyping of Datapath-Intensive Architectures”, *IEEE Design and Test of Computers*, Vol. 8, No. 2, pp. 40-51, 1991.

[Sha90] L. Sha, J.B. Goodenough: “Real-Time Scheduling Theory and Ada”, *IEEE Computer,* Vol. 23, No. 4, pp. 53-62, 1990.

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

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