# Configuration-Level Hardware/Software Partitioning for Real-Time Embedded Systems

Joseph G. D'Ambrosio

General Motors R&D Center
Bldg 1-6, Dept. 40
Warren, MI 48090-9055
jdambros@gmr.com

Xiaobo (Sharon) Hu*

Department of Electrical Engineering
Western Michigan University
Kalamazoo, MI 49008-5066
sharon.hu@wmich.edu

**Abstract**

*In this paper, we present an approach to hardware/software partitioning for real-time embedded systems. The abstraction level we have adopted is referred to as the configuration level, where hardware is modeled as resources with no detailed functionality and software is modeled as tasks utilizing the resources. Through configuration-level analysis, cost and performance tradeoffs can be studied early in the design process and a large design space can be explored. Feasibility factor is introduced to measure the possibility of a real-time system being feasible, and is used as both a constraint and an attribute during the optimization process. Optimal partitioning is achieved through the use of an existing computer-aided design tool.*

## 1 Introduction

Hardware/software codesign is a rapidly growing research area. A number of recent publications have addressed many issues in hardware/software codesign, such as co-specification, co-simulation, and co-synthesis. In this paper, we focus on hardware/software partitioning. Specifically, we consider the following problem: given a list of general-purpose processors and application-specific hardware circuits, and the specification of a system, find an implementation that satisfies the overall system requirements and is optimal in some sense. An implementation is a combination of hardware components, software components, and processors that execute the software components.

There are many possible abstraction levels that can be used to model hardware and software components [5, 7]. Most of the hardware/software partitioning approaches start at the behavioral level and proceed downward [2, 3, 12]. There are a few papers that extend hardware/software codesign to higher levels. For example, Kumar et al. [7] discuss higher abstraction levels, such as system and algorithmic. The level of abstraction that we have adopted for modeling hardware and software components is called configuration level [5], which is similar to the system level. At the configuration level, hardware is modeled as resources with no detailed functionality and software is modeled as tasks utilizing the resources. The goal of configuration-level design is to determine the overall system architecture. Typical decisions include which functions should be implemented in dedicated hardware circuits and which should be in software, how many processors should be used, and which software components should be executed by which processor.

We are particularly interested in hardware/software partitioning for real-time embedded systems. Such systems can be found in many applications, such as powertrain control of automobiles, navigation and landing control of aircraft, and networks and communications. The design of real-time embedded systems, which are generally composed of both hardware and software components, is quite challenging. In addition to the usual design criteria for embedded systems, such as reliability, maintainability and cost effectiveness, real-time embedded systems must provide timely services. That is, the functional behavior of these systems must be not only logically correct but also temporally correct. We will show that performing hardware/software partitioning at the configuration level allows us to efficiently analyze the timing behavior of a large number of implementations with respect to the system timing specifications, and thus to explore a wide range of design options.

In this paper, we present our approach to hard-

*Supported in part by the New Faculty Research Support Program grant from Western Michigan University.

ware/software partitioning for real-time embedded systems. We first discuss general issues related to partitioning at the configuration level and the tools we are using. In Section 3, we consider specific issues related to partitioning for real-time embedded systems. Then, in Section 4, we give a specific example to illustrate the partitioning process. Finally, we summarize our approach and point out future work.

## 2 Hardware/software partitioning at the configuration level

To select an optimal system configuration, a large design space must be explored during hardware/software partitioning. In this section, we discuss how modeling hardware/software components at the configuration level can help us achieve this goal. We also discuss briefly the design tools we are using.

### 2.1 Configuration-level partitioning

We propose to perform hardware/software partitioning at the configuration level. At the configuration level, hardware is modeled as resources and software as tasks utilizing the resources. In particular, software is partitioned into tasks and each task is represented by the amount of memory and estimated number of instructions required to execute it (resource requirements). The corresponding processor model specifies the available memory and time needed to execute a single instruction (available resources). Since an application-specific hardware component implements specific system functions, it is modeled as a resource that is used exclusively by the corresponding functions.

By performing partitioning at the configuration level, we can study system architectural tradeoffs upfront. Our observations and previous design experience indicate that the decisions made at the configuration level are the primary determinant of the overall cost and performance of the final system. Automatic hardware/software partitioning at the behavioral level or lower [2, 3] can only examine a limited design space in order to produce any results in a reasonable amount of time. Configuration-level partitioning, however, makes tradeoff studies more efficient and hence allows significantly more design options to be examined.

Of course, because detailed implementations are not considered at the configuration level, the accuracy of results is an issue. However, the same problem also exists for lower-level partitioning; modeling at a lower level depends heavily on the detailed specification of the system to be designed, which may not be available at the time of devising system configurations, as pointed out in [5]. Many real-life design problems are based on previous implementations. As a result, reasonable modeling accuracy can be achieved by using past design data to develop software and hardware models. Furthermore, by applying a hierarchical refinement design methodology, we can treat the configuration-level partitioning results as an initial design, which will be refined by the behavioral and other lower-level designs.

We approach the hardware/software partitioning problem as follows. A system is specified by a set of time-critical functions and constraints associated with each function, such as timing. These function evaluations can be carried out either by executing software tasks on processors or by dedicated hardware circuits. There is a library of different processors and hardware components that can be used for such a purpose. Using available data or previous design knowledge, each processor and hardware component is characterized by several attributes, such as cost and power. Then, the partitioning problem becomes that of selecting hardware components and processors and assigning software tasks to processors such that the resultant system is optimized in terms of cost/performance, while satisfying all the given constraints.

The number of possible solutions for a given system can be enormous. To find a global optimal solution, we need efficient algorithms/tools to explore the design space. We have made some modifications to an existing tool, GOPS [4], to accomplish global optimization.

### 2.2 Global Optimal Part Selection (GOPS)

GOPS [4] is a configuration-design tool for synthesizing electronic systems. Given a set of functions to implement, a library of parts that implement the functions, a set of constraints that must be satisfied, and a set of attributes for evaluating solutions, GOPS finds the Pareto-optimal set of part-set solutions. Each part set in the Pareto-optimal set implements all functions and satisfies all constraints. For a part set $S$ to be included in the Pareto-optimal set, no other part set may exist with all attribute values better than or equal to that of $S$. Hence, the optimal design is guaranteed to be in the Pareto-optimal set.

GOPS enumerates the Pareto-optimal set by performing a multi-attribute branch-and-bound search of possible function/part assignments. Constraints are encoded in a constraint network [11], and after each

assignment in the search, the constraint network is checked for violations. If a violation is identified, the search algorithm removes the most recent assignment and tries another. Additional search-path pruning is possible by identifying other yet-to-be-made assignments that are either required or inferior [4].

A configuration-design tool must support both *one-to-many mappings* and *many-to-one mappings*. A one-to-many mapping describes how a function is implemented by a set of parts or subfunctions. A GOPS part may contain *required functions*, which are additional functions that must be implemented if the part is chosen. By defining parts with required functions, GOPS implements one-to-many mappings, which provides the means to hierarchically construct a design. A many-to-one mapping, which is necessary when a single component implements multiple functions, is also supported by GOPS.

Using the function-part concept, hardware/software partitioning becomes a problem of implementing system functions with hardware and software "parts." Several modifications to GOPS were required in order to correctly model and efficiently solve the hardware/software partitioning problem. In Section 4, we present an application example and discuss in detail the use of GOPS for partitioning.

## 3 Hardware/software partitioning in real-time embedded systems

The decision of whether functions of a real-time embedded system should be implemented in hardware or software is a typical tradeoff between cost and performance. There are many performance measurements associated with a real-time embedded system, such as meeting timing requirements, satisfying communication constraints, and being reliable. In our initial attempt, we have focused on timing requirements, since timely service is an integral part of the functionality of a real-time embedded system.

In a real-time system, the timing characteristics of each time-critical function is generally specified by a triplet: $(a, d, p)$, where $a$ is the activation time (when the function is ready to be evaluated), $d$ is the deadline (when the evaluation of the function should be completed), and $p$ is the period (the time interval at which the evaluation of the function should be repeated). If a function is to be implemented in hardware, satisfying timing constraints does not present a problem provided that the hardware circuit is designed according to the timing specification. On the other hand, a function may also be implemented as a software task executing on a processor. Since a processor generally needs to execute several tasks competing for its resource, it is not always guaranteed that the tasks will all finish on time. Therefore, one major effort in hardware/software partitioning for real-time systems is to guarantee that the resultant partition satisfies the timing requirements of every function.

To manage the execution of the tasks assigned to a processor, a schedule is needed to prioritize tasks that request execution simultaneously. A schedule is said to be feasible if the timing requirements of all tasks can be satisfied. Feasibility of real-time systems has been studied extensively (e.g., [9, 10]). However, most of the previous research makes certain assumptions regarding the values of $a$, $d$, or $p$ in order to derive closed-form formulas to predict feasibility. Adopting these assumptions can be quite costly for certain applications [5]. To better predict the feasibility of a general system, one may have to use an event-driven simulator, such as TASSIM [5]. However, the event-driven simulation approach can be rather time consuming, so directly incorporating TASSIM (or some other simulator) into the partitioning process is impractical.

To address the feasibility problem in the configuration-level design, we introduce a metric called *feasibility factor*, which indicates the possibility of a system being feasible. It can be used as both a constraint and an attribute. In this section, we describe the feasibility factor and how it is used in GOPS. We also introduce several other timing related attributes specifically for evaluating real-time embedded systems.

### 3.1 Feasibility factor of a real-time system

We first define some notation. Let $TR_P$ be the throughput rate of processor $P$ measured in millions of instructions per second (MIPS). For each time-critical function $F_i$, there is a corresponding software task $T_i$. Assume that $T_i$ requires $c_i$ instructions when executed on processor $P$. In addition, $T_i$ has an activation time $a_i$, deadline $d_i$, and period $p_i$ (all in microseconds). We define $TR_T$ as the minimum throughput requirement in order for processor $P$ to feasibly schedule all the tasks assigned to it. Clearly, if $TR_P \geq TR_T$, the tasks on processor $P$ can all be feasibly scheduled. Note that the value of $TR_T$ may vary depending on the particular scheduling algorithm used [10].

As pointed out earlier, determining the precise value of $TR_T$ for a general real-time system requires simulation, which can be very time consuming. We introduce two bounds for $TR_T$: an upper bound $TR_U$, and a lower bound $TR_L$. The bounds are defined such

that if $TR_U \leq TR_P$, all the tasks on processor $P$ are *definitely* schedulable, and if $TR_L > TR_P$, the tasks are *definitely not* schedulable. The actual throughput requirement $TR_T$, thus, satisfies $TR_L \leq TR_T \leq TR_U$.

We define the feasibility factor for processor $P$ as

$$\lambda_P \equiv \begin{cases} \frac{TR_P - TR_L}{TR_U - TR_L} & \text{if } TR_P - TR_L < TR_U - TR_L \\ 1 & \text{otherwise.} \end{cases}$$

It then follows that the task set on processor $P$ is feasible if $\lambda_P = 1$, and it is not feasible if $\lambda_P < 0$. For $0 \leq \lambda_P < 1$, the larger the value of $\lambda_P$, the greater the chance for the task set to be feasible. Hence, $\lambda_P$ indicates the possibility of processor $P$ being able to meet all the timing requirements of the tasks assigned to it. It will become clear in the next subsection how to use $\lambda_P$ in hardware/software partitioning.

In the rest of this subsection, equations for $TR_U$ and $TR_L$ are found based on the following assumptions: all overhead for context swapping, task scheduling, etc., is zero; tasks do not need to synchronize with one another; finally, a task can be instantly preempted. Let the total number of tasks on processor $P$ be $N$. The value of $TR_U$ can be derived based on the result in [10]. Assuming that the rate-monotonic scheduling algorithm is used [10], then

$$TR_U = [N(2^{\frac{1}{N}} - 1)]^{-1} \sum_{i=1}^{N} \frac{c_i}{d_i - a_i}. \quad (1)$$

Alternatively, if the earliest deadline scheduling algorithm is used [10], then

$$TR_U = \sum_{i=1}^{N} \frac{c_i}{d_i - a_i}. \quad (2)$$

Since the average throughput requirement of task $T_i$ is $c_i/p_i$, a straightforward lower bound can be derived as

$$TR_{L_s} = \sum_{i=1}^{N} \frac{c_i}{p_i}. \quad (3)$$

However, this bound is quite loose for task sets with either $a_i \neq 0$ or $d_i \neq p_i$. In the following, we give two lemmas that will be used to obtain a tighter lower bound. In the two lemmas, $k_i$ and $h_i$ are the minimum number of times task $T_i$ needs to execute between $[a_i, d_n]$ and $[a_n, d_n]$, respectively. (The proofs are omitted due to the page limit.)

**Lemma 1** *For $n$ tasks that are arranged in the ascending order of their deadlines, processor $P$ cannot feasibly schedule them if*

$$\sum_{i=1}^{n} \frac{k_i \cdot c_i}{d_n - a_i} > TR_P,$$

*where*

$$k_i \equiv \begin{cases} \lceil (d_n - a_i)/p_i \rceil & \text{if } \lfloor (d_n - a_i)/p_i \rfloor \cdot p_i + d_i \leq d_n \\ \lfloor (d_n - a_i)/p_i \rfloor & \text{otherwise.} \end{cases}$$

**Lemma 2** *For $n$ tasks that are arranged in the ascending order of their deadlines, processor $P$ cannot feasibly schedule them if*

$$\sum_{i=1}^{n} \frac{h_i \cdot c_i}{d_n - a_n} > TR_P,$$

*where $k_i$ is the same as that defined in Lemma 1, and*

$$h_i \equiv \begin{cases} k_i - \lceil \frac{a_n - a_i}{p_i} \rceil & \text{if } a_i < a_n \\ k_i & \text{otherwise.} \end{cases}$$

The following theorem which defines the lower bound of a general real-time system can be proven based on the above two lemmas.

**Theorem 1** *Let the $N$ tasks on processor $P$ be arranged in the ascending order of their deadlines. Then, $TR_L$ can be calculated as follows:*

$$TR_L = \max_{n=1}^{N} \left\{ \sum_{i=1}^{n} \frac{k_i \cdot c_i}{d_n - a_i}, \quad \sum_{i=1}^{n} \frac{h_i \cdot c_i}{d_n - a_n} \right\}$$

*where $k_i$ and $h_i$ are as defined in Lemma 1 and 2.*

Given the throughput rate of a processor and the set of tasks to be executed, the feasibility factor can be easily calculated. It can then be used as both a constraint and an attribute in GOPS to control the optimization process. We will discuss this in the next subsection.

### 3.2 Real-time system attributes for design evaluation

The most important timing requirement of a real-time system is to make sure that all time-critical functions can be completed on or before their deadlines. To verify whether a configuration satisfies such timing constraints, GOPS can compare the upper bound of the total throughput requirement, $TR_U$, imposed by the tasks assigned to a processor with the throughput rate of the processor, $TR_P$. However, since $TR_U$ can be significantly larger than what is actually needed ($TR_T$), this approach tends to choose more costly hardware/software partitions.

We propose to use feasibility factor as both a constraint and an attribute in GOPS for handling the feasibility problem. In the constraint case, GOPS eliminates those solutions that result in a negative feasibility factor, since only solutions with $\lambda_P \geq 0$ are

possibly feasible. Furthermore, GOPS uses feasibility factor as one of several attributes to generate the Pareto-optimal set of hardware/software partitions. It has been pointed out earlier that a larger feasibility factor indicates a higher probability of the implementation being feasible. Hence, for a partition to be included in the Pareto-optimal set, one of its attributes, e.g., cost or feasibility factor, must be superior. Once GOPS completes, the solutions with $\lambda_P < 1$ in the Pareto-optimal set need to be checked by TASSIM to verify their schedulability. The optimal feasible design can then be identified.

Other timing-related attributes may also be included for evaluating the performance of real-time embedded systems. An important property of an embedded system is its expandability. To limit costs, much of the hardware and software of an embedded system must be reusable through several design cycles and accommodate increasingly demanding functionality over the life of the design. Therefore, a designer may be willing to tradeoff cost for expandability in a particular design. To model the expandability of a real-time system, we introduce an attribute called critical excess MIPS, $\Delta_c$. It is defined as $\Delta_c \equiv TR_P - TR_L$. Clearly, the value of $\Delta_c$ is an estimate of the amount of peak execution power that a processor has after meeting the timing constraints of the current task specifications. A larger $\Delta_c$ will allow the current tasks to be expanded, i.e., to increase their execution requirements, and still be feasible. It may also allow the system to handle new time-critical tasks.

In addition to processing time-critical functions, a real-time embedded system may also need to handle certain non-time-critical functions. Compared to time-critical functions, these functions do not have hard deadlines, but may require a fast *average* response time. When executing both time-critical and non-time-critical software tasks, a scheduling algorithm such as the one in [8] can be used. From the view of hardware/software partitioning, the existence of non-time-critical tasks can be treated as extra demand on a processor's average throughput. Hence, we introduce another attribute, called average excess MIPS, $\Delta_a$. It is defined as $\Delta_a \equiv TR_P - \sum_{i=1}^{N} c_i/p_i$, where $c_i$ and $p_i$ are parameters of time-critical tasks. The average excess MIPS tells the designer the exact amount of execution power that the processor has for processing non-time-critical tasks. Depending on the particular system, the designer may be willing to tradeoff cost with average excess MIPS in order to be able to effectively handle more non-time-critical tasks.

In a single processor system, the above three attributes can simply be calculated based on the processor selected and tasks assigned to it. The process becomes more complicated for multiple-processor systems, since each processor has its own feasibility factor, critical excess MIPS, and average excess MIPS. For feasibility factors, the minimum one among all processors indicates the possibility of the overall system being feasible. Thus, we only need to use the minimum feasibility factor as the system feasibility attribute (denoted by $\lambda$) in the optimization process. For both excess-MIPS attributes, the summation of the excess MIPS of each processor is of particular interest since it is an indication of the overall system's capability.

| Name | $i$ | $d_i$ | $p_i$ | $a_i$ |
|---|---|---|---|---|
| DigitalFilter1 (DF1) | 1 | 46.00 | 104.17 | 0.00 |
| DigitalFilter2 (DF2) | 2 | 10000.00 | 10000.00 | 9895.83 |
| DecodeSPUB (DSB) | 3 | 83.00 | 208.33 | 0.00 |
| DecodeSPUA (DSA) | 4 | 138.00 | 208.33 | 83.00 |
| ReadCAM (RC) | 5 | 416.67 | 10000.00 | 0.00 |
| ServiceRoutine (SR) | 6 | 208.33 | 416.67 | 0.00 |
| FuelCalc (FC) | 7 | 1333.33 | 2500.00 | 833.33 |
| SparkCalc (SC) | 8 | 2500.00 | 2500.00 | 1666.67 |
| ReadMAP (RM) | 9 | 312.50 | 416.67 | 0.00 |
| CPU | - | - | - | - |

Table 1: Primary set of functions to be implemented. Deadline ($d_i$), period ($p_i$), and activation time ($a_i$) are all in units of microseconds.

## 4 Application example

In this section we provide an application example: nine time-critical functions must be implemented in a single processor system, and the system will be evaluated based on component cost, feasibility factor, and critical excess MIPS. For simplicity, we have assumed that communication between hardware components in the resultant system is carried out through a zero-delay shared memory area. We will discuss this simplification in the next section.

### 4.1 Problem formulation

Function and part specifications for GOPS are summarized in Table 1-3. Table 1 describes the functions to be implemented, where the data for the time-critical functions are specified by control-systems engineers [5]. Note that in addition to the time-critical functions, function CPU is also included, which indicates that only implementations containing a single processor need to be examined.

| Name | Function Implemented | Instructions Executed | RAM Req'd | ROM Req'd |
|---|---|---|---|---|
| DF1-S | DF1 | 64 | 100 | 100 |
| DF2-S | DF2 | 32 | 100 | 100 |
| DSB-S | DSB | 30 | 200 | 300 |
| DSA-S | DSA | 30 | 200 | 300 |
| RC-S | RC | 30 | 100 | 100 |
| SR-S | SR | 20 | 200 | 200 |
| FC-S | FC | 480 | 500 | 400 |
| SC-S | SC | 100 | 400 | 300 |
| RM-S | RM | 40 | 100 | 100 |

Table 2: Software parts to implement functions. The required RAM and ROM are measured in bytes.

All the time-critical functions have associated software parts, which are described in Table 2. Each software part is characterized by the number of instructions executed and the amount of RAM and ROM required to implement the part. Since the total amount of RAM and ROM needed for the system depends on the hardware/software partition, RAM and ROM are required functions (see Section 2.2) of the software parts: only after a software part is chosen for a part set do the amounts of RAM and ROM needed by the part become functions to implement. For this example, we assume the software characterization given in Table 2 is valid for every processor. A task's required memory and instructions generally vary from one instruction set to the next, and in such cases this assumption is not valid. The general case can be supported by defining software parts for each instruction set, and defining constraints so that parts can only be assigned to appropriate processors.

Although all functions may be implemented in software, functions one through four may also be implemented in hardware. Table 3 lists some of the *hardware parts* available for this system. The parts include: microcontrollers (MC), processors (P), application-specific components (ASIC), standard peripherals (PIO), RAM, and ROM. For example, three derivatives of microcontroller four (MC4a-H, MC4b-H, MC4c-H) are available, and MC4a-H has 2K bytes of RAM, fourteen timing channels (TC), and custom circuits to implement functions one through four. Note that most of the parts in the table are multifunction parts. In addition to the parts given in Table 3, fifteen other parts were available, but were not contained in any of the Pareto-optimal solutions.

In addition to software and hardware parts, transformation parts, which describe how a function can be implemented by another function, are also required. For instance, a timing channel, which is a standard input/output device provided by most microcontrollers, is a function that can be used to implement functions one or two (DF1 or DF2). In order for GOPS to implement DF1 or DF2 with a timing channel, two transformation parts, DF1toTC and DF2toTC, were included in the part library. The two parts have TC as a required function and incur zero cost.

| Name | Functions Implemented | Cost | MIPS |
|---|---|---|---|
| MC1-H | CPU, RAM(2K), ROM(2K), DF1,DF2,DSB,DSA | 3.50 | 1.30 |
| MC2-H | CPU, RAM(2K), ROM(2K), TC(32) | 3.25 | 1.50 |
| MC3a-H | CPU, RAM(4K), TC(16) | 5.25 | 2.50 |
| MC3b-H | CPU, RAM(4K), DF1,DF2, DSB,DSA | 6.25 | 2.50 |
| MC4a-H | CPU, RAM(2K), DF1, DF2, DSB, DSA, TC(14) | 3.75 | 1.70 |
| MC4b-H | CPU, RAM(2K), DF1, DF2, DSB, DSA, TC(14) | 3.25 | 1.35 |
| MC4c-H | CPU, RAM(2K),TC(16) | 2.50 | 1.70 |
| P1-H | CPU, RAM(2K), ROM(2K) | 2.00 | 1.43 |
| P2-H | CPU | 13.00 | 13.50 |
| ASIC1-H | DF1,DF2,DSB,DSA | 2.50 | - |
| PIO1-H | TC(16) | 1.00 | - |
| RAM1-H | RAM(2K) | 2.00 | - |
| ROM1-H | ROM(2K) | 1.00 | - |

Table 3: Hardware parts to implement functions.

As stated above, functions one through four can be directly implemented in hardware. This is accomplished using application-specific VLSI designs. If a hardware design for a function does not exist yet, but implementation of such a design may be desirable, then an estimate of the design's cost and performance is used to define a proposed hardware part, which is considered by GOPS during the configuration-design process.

The only constraint used for this problem is that the feasibility factor of a part set must be greater than or equal to zero. For most real problems, additional constraints would be required to guarantee interoperability of the hardware parts chosen to implement the design. For example, connection of an Intel processor to a Motorola peripheral device may not be possible, because of different bus interface models. However, additional constraints will reduce the difficulty of solving the example problem using GOPS, since there will be fewer feasible solutions.

## 4.2 Solving the problem

As GOPS executes, it enumerates all Pareto-optimal solutions. First GOPS finds an initial imple-

mentation for the functions given in Table 1. We have modified GOPS so that once the initial implementation is found, it implements any required functions before investigating other solutions for the functions in Table 1. This allows the modified GOPS to immediately identify a complete solution, and as a result, to quickly enable branch-and-bound checks to minimize the amount of search required. In this application, the required functions may include various amounts of RAM, ROM, and timing channels.

Table 4 gives the twelve Pareto-optimal solutions for this problem. As an example, consider solution one. The first two parts in the part set are transformation parts, indicating that both DF1 and DF2 are implemented as timing channels. The next two parts, DSB-S and DSA-S, are software parts, indicating that DSB and DSA are implemented as software. Note that all part set solutions implement RM, SC, FC, SR, and RC as software. The next part, P1-H, is a processor providing enough RAM, ROM, and MIPS to support the software parts. The last part is PIO1-H, a peripheral I/O device that contains sixteen timing channels. Therefore, DF1 and DF2 are implemented as timing channels on PIO1-H.

The first five solutions in Table 4 have feasibility factors less than one, and thus, may or may not be feasible. Checking these five solutions with TASSIM indicates that all but solution one are feasible. After removing solution one from consideration, it may be possible to directly identify the best design using only the attribute information in the table, since the Pareto-optimal set is small and the attribute values of the solutions are well distributed. In other situations, to obtain the most promising solutions for certain given criteria, detailed analysis is required.

## 5 Summary and discussion

In this paper, we motivated the need for hardware/software partitioning at the configuration level. Investigation of specific issues related to the design of real-time embedded systems identified several attributes for evaluating such systems, including feasibility factor and critical excess MIPS. The feasibility factor provides an effective means to identify infeasible, potentially feasible, and feasible designs without having to perform time-consuming simulations. The feasibility factor can be used as a constraint for eliminating infeasible partitions and as an attribute for efficiently evaluating real-time systems modeled at the configuration level. Finally, we presented the formulation and solution of an example problem.

One limitation of our work to date is the assumption that all communication is performed through a zero-delay shared memory area. Although our approach is suitable for creating single-processor, tightly-coupled dual-processor, and loosely-coupled dual-processor systems, the communication delays between functions can be significant in determining the performance and feasibility of these systems. However, extending our communication model does not present any major difficulties, and we are currently working on this.

Another limitation relates to design complexity. For example, in dual-processor systems, partitioning functions between processors must be considered in addition to hardware/software partitioning and hardware-component selection. Our experience with such systems indicates that generating the Pareto-optimal set of solutions is not practical in a significant number of cases. Furthermore, the Pareto-optimal set also grows as more and more attributes are introduced to evaluate system performance (e.g., average excess MIPS, excess memory, power and area). In these cases, straightforward enumeration of all solutions in the Pareto-optimal set is prohibitive.

Our approach to solving this problem is to view the design process as a decision problem [1, 6, 13]. Designer preferences are specified by evaluating a random sample of design alternatives. In specifying the preferences, the designer is forced to make tradeoffs among the attributes used to evaluate the designs. The preferences effectively create an objective function that is used during the design process. Instead of generating the entire Pareto-optimal set of solutions, a small subset, which contains preferred solutions, is produced.

For instance, a designer can tradeoff his/her willingness to spend time evaluating solutions against the cost/performance of the design. The feasibility factor can be used as an indicator of the amount of evaluation and iteration required by the design process. If cost/performance is valued over design time, then GOPS tends to select those designs with low feasibility factors ($\lambda < 1$). Such designs must be evaluated using TASSIM to identify the feasible ones. It is also possible that none of the designs found will be feasible, and GOPS iterations will be required. In contrast, if design time is valued over cost/performance, THEN the set of designs identified by GOPS will most likely all be feasible, thus minimizing TASSIM evaluations and GOPS iterations.

We have tested this approach for a few cases, and the results are quite encouraging. We plan to continue investigating this decision theoretic approach in the future.

| Number | Part Set | Cost | Feasibility Factor ($\lambda$) | Critical Excess MIPS ($\Delta_c$) |
|---|---|---|---|---|
| 1 | DF1toTC, DF2toTC, DSB-S, DSA-S, P1-H, PIO1-H | 3.00 | 0.013 | 0.011 |
| 2 | DF1toTC, DF2toTC, DSB-S, DSA-S, MC2-H | 3.25 | 0.094 | 0.081 |
| 3 | MC1-H | 3.50 | 0.706 | 0.183 |
| 4 | DF1toTC, DF2toTC, DSB-S, DSA-S, MC4c-H, ROM1-H | 3.50 | 0.325 | 0.281 |
| 5 | MC4b-H, ROM1-H | 4.25 | 0.899 | 0.233 |
| 6 | P1-H, ASIC1-H | 4.50 | 1.000 | 0.313 |
| 7 | MC4a-H, ROM1-H | 4.75 | 1.000 | 0.583 |
| 8 | DF1toTC, DF2toTC, DSB-S, DSA-S, MC3a-H, ROM1-H | 6.25 | 1.000 | 1.081 |
| 9 | MC3b-H, ROM1-H | 7.25 | 1.000 | 1.383 |
| 10 | DF1-S, DF2-S, DSB-S, DSA-S, P2-H, RAM1-H, ROM1-H | 16.00 | 1.000 | 11.460 |
| 11 | DF1toTC, DF2toTC, DSB-S, DSA-S, P2-H, PIO1-H, RAM1-H, ROM1-H | 17.00 | 1.000 | 12.080 |
| 12 | P2-H, ASIC1-H, RAM1-H, ROM1-H | 18.50 | 1.000 | 12.380 |

Table 4: Pareto-optimal set of solution. Note that all parts sets also include: RM-S,SC-S, FC-S, SR-S, RC-S.

## References


[1] W.P Birmingham, J.G. D'Ambrosio, T. Darr, and E. Durfee, "Coordinating Decision Making in Large Organizations," University of Michigan Technical Report, CSE-TR-208-94, April 1994.

[2] R. Ernst, J. Henkel, and Th. Benner, "Hardware-software cosynthesis for microcontrollers," *IEEE Design & Test of Computers*, vol.10, no.4, December 1993, pp.64–75.

[3] G.K. Gupta and G. De Micheli, "Hardware-software cosynthesis for digital systems," *IEEE Design & Test of Computers*, vol.10, no.3, September 1993, pp.29–40.

[4] M.S. Haworth and W.P. Birmingham, "Towards optimal system-level design," *Proceedings of 30th Design Automation Conference*, June 1993, pp.434–438.

[5] X. Hu, J.G. D'Ambrosio, B.T. Murray, and D-L. Tang, "Analysis in hardware/software codesign: an automotive case study," Accepted for publication in *IEEE MICRO*, August 1994.

[6] R.L. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, John Wiley & Sons, New York, 1976.

[7] S. Kumar, J.H. Aylor, B.W. Johnson, and W.A. Wulf, "Exploring hardware/software abstractions & alternatives for codesign," *The 2nd International Workshop on Hardware/Software Codesign*, Cambridge, Massachusetts, October 1993.

[8] J. P. Lehoczky and S. Ramos-Thuel, "An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems," *Proceedings of Read-Time Systems Symposium*, December 1992, pp.110–123.

[9] J. Y-T Leung and J. Whitehead, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Performance Evaluation*, vol.2, 1982, pp.237–250.

[10] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard real-time environment," *Journal of the Association for Computing Machinery*, vol.20, no.1, 1973, pp.46–61.

[11] A.K. Mackworth, "Constraint satisfaction," In S.C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, New York, 1987.

[12] D.E. Thomas, J.K. Adams, and H.Schmitt, "A model and methodology for hardware-software codesign," *IEEE Design & Test of Computers*, vol.10, no.3, September 1993, pp.6–15.

[13] D.L. Thurston, "A Formal method for subjective design evaluation with multiple attributes," *Research in Engineering Design*, vol.3, 1991, pp.105–122.