Proceedings of IEEE Real-Time Systems Symposium (RTSS), pp. 201-211, Dec. 2002.

Embedded System Design Framework for Minimizing Code Size and Guaranteeing Real-Time Requirements [*]

Insik Shin, Insup Lee
Dept. of Computer & Information Science
University of Pennsylvania
Philadelphia, PA 19104 USA
{ishin,lee}@cis.upenn.edu

Sang Lyul Min
Schl. of Computer Science & Engineering
Seoul National University
Seoul, 151-742, KOREA
symin@dandelion.snu.ac.kr

Abstract

In addition to real-time requirements, the program code size is a critical design factor for real-time embedded systems. To take advantage of the code size vs. execution time tradeoff provided by reduced bit-width instructions, we propose a design framework that transforms the system constraints into task parameters guaranteeing a set of requirements. The goal of our design framework is to derive the temporal parameters and the code size parameter of each task in such a way that they collectively guarantee the system end-to-end timing requirements while the system code size is minimized. Our design framework is based on asynchronous periodic tasks with pre-period deadlines under EDF scheduling. For schedulability analysis, we present a new feasibility condition that can be more efficiently evaluated than existing ones. When the code size vs. execution time tradeoff can be safely approximated as linear functions, the minimization problem becomes a linear programming problem. However, when the tradeoff is given by a table of possible (code size, execution time) pairs, the problem becomes NP-hard. We provide three heuristic algorithms that can find sub-optimal solutions and evaluate their performance with simulation results.

1 Introduction

The program code size is one of the key factors that determine the manufacturing cost of an embedded system, especially when the embedded system is implemented as an SOC (System On a Chip). One code size reduction technique at the instruction set architecture (ISA) level is to use a subset of normal 32-bit instructions compressed into a 16-bit format as in ARM Thumb [6] and MIPS16 [13]. These 16-bit instructions are dynamically decompressed by hardware into 32-bit equivalent ones before execution. This approach can substantially reduce the program code size; however, it increases the number of instructions to be executed, and thus, increases the execution time of the program. For typical examples, the compressed code may require around 70% of the space of the original code, while executing 40% more instructions [3].

When an embedded system is used as a real-time system, there are also temporal requirements imposed on the system that must be met for correct operation. In such a real-time embedded system, the code size vs. execution time tradeoff resulting from the use of reduced bit-width ISA [7] gives rise to a challenging question: to minimize the total code size of all the tasks in the system while satisfying all the temporal requirements imposed on the system.

There has been much work on the design of real-time systems guaranteeing the system temporal requirements. In particular, Period Calibration Method (PCM) [5] is a design framework that transforms the system temporal requirements into the temporal parameters of tasks that collectively guarantee the system-level end-to-end timing requirements. The design framework proposed in this paper extends the PCM framework by considering the code size vs. execution time tradeoff of each task to come up with a solution that minimizes the total system code size while satisfying the system-level real-time requirements. The proposed framework assumes that the code size vs. execution time tradeoff for each task is given either by a table that lists possible (code size, execution time) pairs or by a linear tradeoff function that safely approximates the table. From this tradeoff relationship, the proposed framework formulates the optimization problem of minimizing the total system code size subject to the system-level real-time requirements. The optimization problem is a linear programming

[*]This research was supported in part by NSF CCR-9988409, NSF CCR-0086147, NSF CISE-9703220, ARO DAAD19-01-1-0473, DARPA MOBIES F33615-00-C-1707, ONR N00014-97-1-0505, and the Ministry of Science and Technology of the Republic of Korea under the National Research Laboratory program.

problem when the tradeoff is given as a linear function, or it becomes an NP-hard problem when the tradeoff is given as a tabular form. For the latter case, we describe three heuristic algorithms, each of which finds a sub-optimal solution to the optimization problem using different criteria in the solution process. In addition, for a set of asynchronous periodic tasks with relative deadlines less than or equal to their periods, we develop a new feasibility condition under EDF (Earliest Deadline First) scheduling [10] that can be more efficiently evaluated than existing ones.

The rest of this paper is organized as follows: Section 2 describes the system model and gives the problem description and the overview of our solution process. Section 3 briefly reviews PCM and the code size reduction technique. Section 4 presents a new feasibility condition. Section 5 formulates the optimization problem and presents heuristic algorithms. Section 6 illustrates how our design framework works using an example and evaluates the performance of the heuristic algorithms with simulation results. Finally, we conclude in Section 7 with discussion on future research.

2 System Model and Problem Description

2.1 System Model

We assume that an embedded system τ_{sys} is composed of a set of tasks, $\{\tau_1, \ldots, \tau_n\}$. Each task $\tau_i \in \tau_{sys}$ has the following task temporal parameters:

- *period T_i:* the fixed time interval between the arrival times of two consecutive requests of τ_i,
- *offset O_i:* the time instant relative to the start-of-its-period at which τ_i's execution becomes available ($O_i \geq 0$),
- *deadline D_i:* the time instant relative to the start-of-its-period by which τ_i's execution is required to be completed ($D_i \leq T_i$),
- *execution time e_i:* the time amount required to complete τ_i's execution in the worst case,
- *execution window W_i:* the time interval of $[O_i, D_i]$ during which τ_i's execution becomes available and needs to be completed. $|W_i| = D_i - O_i$.

A task τ_i is said to be synchronous if $O_i = 0$ or asynchronous if $O_i \geq 0$. A task τ_i is said to have a period deadline if $D_i = T_i$ or a pre-period deadline if $D_i \leq T_i$. Our design framework considers asynchronous tasks with pre-period deadlines. In this paper, when it comes to an execution time, it means the worst-case execution time. When a worst-case execution time has its value range due to the code size vs. execution time tradeoff possible at compile

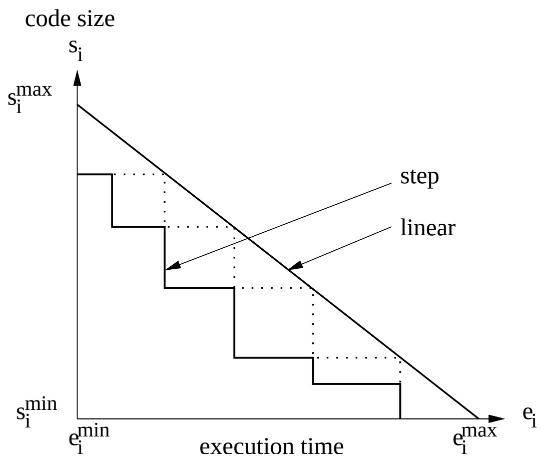


Figure 1. tradeoff functions between code size and execution time

time, the minimum execution time refers to the minimum value of the the worst-case execution time. In addition to the temporal parameters, each task τ_i has a task code size parameter as follows:

- *code size s_i:* the size of τ_i's executable code.

Let T_{LCM} be the least common multiplier (LCM) of T_i's of all τ_i's $\in \tau_{sys}$. We consider T_{LCM} as the major period of the system.

Let $\tau_{i,j}$ denote the j-th job (execution unit) of τ_i ($j \geq 0$). Let $o_{i,j}, d_{i,j}$ and $W_{i,j}$ denote the offset, the deadline and the execution window of $\tau_{i,j}$, respectively. Note that $o_{i,j}$ and $d_{i,j}$ are time instants relative to the start-of-major-period such that $o_{i,j} = O_i + j \cdot T_i$ and $d_{i,j} = D_i + j \cdot T_i$. We define J_{LCM} as the set of jobs whose deadlines are earlier than or equal to T_{LCM}, i.e.,

$$J_{LCM} = \Big\{ \tau_{i,j} \quad | \quad d_{i,j} \leq T_{LCM} \Big\}.$$

Let $e_{i,j}$ and $s_{i,j}$ denote the execution time and code size of $\tau_{i,j}$, respectively. Since the execution time and the code size are determined at compile time, all $e_{i,j}$'s and $s_{i,j}$'s are equal to e_i and s_i, respectively.

In this paper, we assume that each task τ_i is given a tradeoff relationship between code size (s) and execution time (e) either by a discrete step function (i.e., a table of possible (s, e) pairs) or a linear function that safely approximates the step function. Figure 1 shows an example of a discrete step function that gives the code size vs. execution time tradeoff and a linear function that safely approximates the step function. The tradeoff function is denoted by

$$s = f_i(e), \qquad e_i^{min} \leq e \leq e_i^{max}. \tag{1}$$

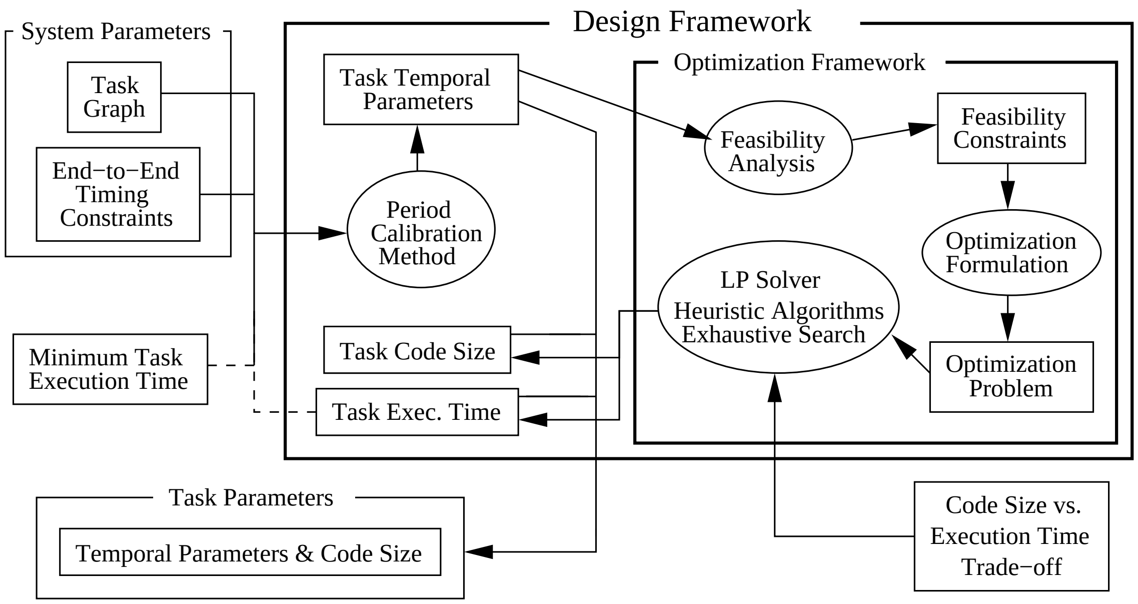


Figure 2. Overview of our design framework

For real-time embedded systems, we consider the following three classes of timing constraints presented in [5]:

- A *freshness constraint* bounds the time it takes for data to flow from input X to output Y (denoted as $F(Y|X)$).
- A *correlation constraint* bounds the maximum time-skew between several inputs $(X_1, \ldots, X_n)$ used to produce output Y (denoted as $C(Y|X_1, \ldots, X_n)$).
- A *separation constraint* bounds the jitter between consecutive values on a single output Y (denoted as $u(Y)$ and $l(Y)$).

We define the system code size (s_{sys}) as the sum of the code sizes of all the tasks in the system, i.e.,

$$s_{sys} = \sum_{\tau_i \in \tau_{sys}} s_i.$$

2.2 Problem Description

We consider a design framework for real-time embedded systems that transforms the system constraints into task parameters guaranteeing a set of requirements. The goal of our design framework is to derive the temporal parameters and the code size parameter of each task in such a way that they collectively guarantee the system end-to-end timing requirements and that the system code size is minimized. Figure 1 depicts the overview of our design framework. The input, output, and requirements of our design framework are given as follows:

Input:

- *System structure:* a task graph representing dataflow and task precedence.
- *System constraints:* the system end-to-end timing constraints
- tradeoff between the code size and the (worst case) execution time of each task.

Output:

- *Task parameters:* the temporal parameters (period, offset, deadline, and execution time) and the code size parameter of each task.

Requirements:

- *Correctness:* system behaviors that satisfy all the task parameters also satisfy the system constraints.
- *Feasibility:* task temporal parameters never demand a time interval during which the CPU utilization exceeds 100%.
- *Optimization:* the total code size should be minimized subject to the above two requirements.

When solving the problem of determining the temporal parameters and the code size of each task satisfying the system end-to-end timing constraints and optimizing the system code size, our design framework breaks the problem into two sub-problems: (1) transforming the system end-to-end timing constraints into task temporal parameters with the system structure and constraints and the execution time of each task and then (2) deriving the code size and execution time of each task with the optimization goal subject to the task temporal parameters. We iteratively solve these two sub-problems. We initially use the minimum (worst-case) execution time of each task to solve the first sub-problem. The solution to the second sub-problem determines the execution time of each task with the optimization goal, and then we go back to the first sub-problem with this new execution time of each task. We repeat this iterative procedure until the solution converges.

Our design framework uses an existing technique called Period Calibration Method (PCM) [5] to solve the first sub-problem and we provide a new optimization framework to solve the second sub-problem.

PCM is a design methodology that transforms the system end-to-end timing constraints into the temporal parameters of each task when the system task graph, the system constraint, and the worst case execution times of tasks are given. Having PCM as the first sub-problem solver, our design framework can use the same asynchronous task graph and the end-to-end timing constraints as those in [5].

With PCM serving as the front-end technique, our design optimization framework works as back-end. The goal of this back-end is to determine the code size vs. execution time tradeoff of each task to minimize the system code size while guaranteeing a feasible schedule. The input to the back-end is the code size vs. execution time tradeoff of each task as well as the temporal parameters of each task that PCM determined as solutions to the first sub-problem.

To guarantee a feasible schedule with the temporal parameters of each task, our design framework generates feasibility constraints according to a feasibility condition. With the feasibility constraints, the back-end formulates the optimization problem of minimizing the system code size. Depending on the format by which the code size vs. execution time tradeoff is given, the back-end solves the optimization problem with a different tool. If the tradeoff relationship is given as a linear function, the back-end uses an existing linear programming solver to find the optimal solution. If the tradeoff is given in a tabular form, the back-end uses heuristic algorithms to find sub-optimal solutions. For this purpose, we give three heuristic algorithms that use different criteria in the solution process. With the solution given by the back-end, our design framework finds the temporal parameters and the code size parameter of each task that satisfy the three requirements of correctness, feasibility, and optimization.

3 Overview of Existing Techniques

3.1 Period Calibration Method (PCM) Overview

Period calibration method (PCM) [5] provides a design methodology that transforms the system end-to-end timing requirements into task temporal parameters. As input, PCM takes a task graph and a set of system end-to-end timing constraints of the embedded system: the task graph represents dataflow, task precedence, and end-to-end timing constraints on freshness from input to output, input correlation and allowable output separation. Solving the constraints generated from the system timing requirements, PCM first derives the periods of tasks minimizing the CPU utilization and then derives the offset and deadline of each task subject to the periods locally maximizing the schedulability of each task. The static priorities of tasks are assigned considering the task precedence given in the task graph.

There has been work on applying PCM to the design of real-time embedded systems including an avionics control system [12] and a computerized numerical control system [11]. While reporting some weaknesses of PCM such as infeasible solutions [12] and the absence of overload handling [11], these papers concluded that PCM provides a useful methodology for the design of real-time embedded systems.

PCM derives task temporal parameters for static priority-based scheduling. However, our design framework uses the earliest deadline first (EDF) scheduling to improve system utilization and to enable efficient feasibility analysis. This requires a change in task temporal parameters generated by PCM so that when tasks are scheduled by EDF, the end-to-end timing requirements are still satisfied. In the original PCM, the task precedence is preserved by static priorities. If τ_i precedes τ_j, τ_i has a higher priority than τ_j. In our EDF-based scheduling, we assign deadlines in such a way that if τ_i precedes τ_j, D_i is smaller than D_j by one.

3.2 Code Size Reduction Technique Overview

For many embedded systems, program code size is a critical design factor. We present a brief overview of a compiler technique for code size reduction that works for a processor capable of executing reduced bit-width instructions. A very good example of such a processor is ARM microprocessors with a 32-bit instruction set (IS) for normal modes and a 16-bit reduced bit-width IS for Thumb modes [6]. A reduction in code size comes from encoding a subset of the 32-bit normal mode IS into the 16-bit Thumb mode IS. At the execution time, a decompression engine converts a Thumb-mode instruction into an equivalent normal-mode instruction during the decode stage. The Thumb IS can access only 8 general purpose registers (out of 16 general purpose registers in the normal mode) and can encode only a small immediate value. These limitations increase the number of instructions executed and, thus, increases the program execution time. For typical programs, by using this technique the code size can be reduced by around 30%, while the number of instructions executed increases by about 40% [3].

The reduced bit-width ISA allows a program that contains both 32-bit normal-mode instructions and 16-bit reduced bit-width instructions where the mode change between the two can be performed by executing a single mode-change instruction. This capability allows for a tradeoff between code size and execution time when compiling a program. For example, by progressively transforming program units such as functions or basic blocks in the normal mode into the equivalent ones in the reduced bit-width mode while adding patch-up code to maintain the correct semantics, we can obtain a table that gives possible (code size, execution time) pairs. The order by which the transformation is performed considers both reduction in code size and increase in the program execution time, i.e., it favors program units that give large reduction in code size with only a small increase in the program execution time.

4 Feasibility Analysis under EDF

There has been much work on feasibility analysis techniques under EDF scheduling. The following important results have been proved for independent periodic tasks on one processor. Liu and Layland [10] showed that any set of n synchronous periodic tasks with period deadlines with processor utilization $U = \sum_{i=1}^{n} \frac{e_i}{T_i}$ is feasible if and only if $U \leq 1$. Coffman [2] also showed that any set of n asynchronous periodic tasks with period deadlines is feasible if and only if $U \leq 1$. The problem of deciding whether a set of asynchronous periodic tasks with pre-period deadlines has been proved to be NP-hard by Leung and Merrill [9]. Further, Baruah et al. showed that this problem is NP-hard in the strong sense [1].

Definition 4.1 *Let $\eta_i(t_1, t_2)$ denote the number of jobs of task τ_i that must be completely scheduled in $[t_1, t_2)$. It is computed as follows:*

$$\eta_i(t_1, t_2) = \max\left\{0, \left\lfloor \frac{t_2 - O_i - D_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - O_i}{T_i} \right\rceil + 1 \right\}.$$

Lemma 1 (Baruah et al. [1]) *A set of n asynchronous periodic tasks with pre-period deadlines is feasible if and only if*

$$\sum_{i=1}^{n} \eta_i(t_1, t_2) \cdot e_i \leq t_2 - t_1,$$

for all $0 \leq t_1 < t_2 \leq O_{max} + 2T_{LCM}$, where O_{max} is the maximum among all O_i's.

Lemma 1, by Baruah et al., gives a necessary and sufficient feasibility condition for asynchronous periodic tasks with pre-period deadlines. We provide an alternative necessary and sufficient condition that can be more efficiently evaluated than the one in Lemma 1.

Lemma 2 *For each $j \geq 0$, $\eta_i(o_{i,j+1}, t_2) = \eta_i(t_1, t_2)$ for all $o_{i,j} < t_1 \leq o_{i,j+1}$ and for a fixed $t_2 > t_1$. For each $j \geq 0$, $\eta_i(t_1, d_{i,j}) = \eta_i(t_1, t_2)$ for all $d_{i,j} \leq t_2 < d_{i,j+1}$ and for a fixed $t_1 < t_2$.*

Proof. From the definition of $o_{i,j}$ and $d_{i,j}$, $o_{i,j}$ is $O_i + jT_i$ and $d_{i,j}$ is $O_i + jT_i + D_i$. From the definition of η_i, t_1 is contributed to computing $\lceil \frac{t_1 - O_i}{T_i} \rceil$ and t_2 is contributed to computing $\lfloor \frac{t_2 - O_i - D_i}{T_i} \rfloor$. We can easily see that when t_1 is $o_{i,j+1}$, $\frac{t_1 - O_i}{T_i}$ is exactly $j+1$ and $\lceil \frac{t_1 - O_i}{T_i} \rceil$ is $j+1$ as long as $o_{i,j} < t_1 \leq o_{i,j+1}$. We can also see that when t_2 is $d_{i,j}$, $\frac{t_2 - O_i - D_i}{T_i}$ is exactly j and $\lfloor \frac{t_2 - O_i - D_i}{T_i} \rfloor$ is j as long as $d_{i,j} \leq t_2 < d_{i,j+1}$. □

Let TT_{ALL} denote a set of possible (t_1, t_2) values for Lemma 1, that is, $\{(t_1, t_2) : \forall t_1, t_2 : 0 \leq t_1 < t_2 \leq O_{max} + 2T_{LCM}\}$. Let O_{ALL} denote a set of offsets of all tasks between 0 and $O_{max} + 2T_{LCM}$, that is, $\{o_{i,j} : \tau_i \in \tau_{sys}, o_{i,j} < O_{max} + 2T_{LCM}\}$. Let D_{ALL} denote a set of deadlines of all tasks between 0 and $O_{max} + 2T_{LCM}$, that is, $\{d_{i,j} : \tau_i \in \tau_{sys}, d_{i,j} \leq O_{max} + 2T_{LCM}\}$. Let OD_{ALL} denote a set of possible pair of (offset, deadline) between 0 and $O_{max} + 2T_{LCM}$, that is, $\{(o, d) : o \in O_{ALL}, d \in D_{ALL}, o < d\}$. We can see that $OD_{ALL} \subseteq TT_{ALL}$, and if there is any time $t, 0 < t < O_{max} + 2T_{LCM}$, such that t is not included in both O_{ALL} and D_{ALL}, then $OD_{ALL} \subset TT_{ALL}$.

Theorem 1 *A set of n asynchronous periodic tasks with pre-period deadlines is feasible if and only if*

$$\sum_{i=1}^{n} \eta_i(t_1, t_2) \cdot e_i \leq t_2 - t_1,$$

for all $(t_1, t_2) \in OD_{ALL}$.

Proof. We prove this theorem by showing that this feasibility condition (referred as FC_2) is equivalent to the feasibility condition (referred as FC_1) given in Lemma 1 in terms of feasibility analysis. While sharing the same set of inequalities, the two conditions differ in the domains of (t_1, t_2) values: TT_{ALL} for FC_1 and OD_{ALL} for FC_2. In what follows, we show that the two conditions yield the same feasibility results with the two different domains.

First, we show that if FC_1 finds a task set feasible, then FC_2 also finds the task set feasible. This is obvious from $OD_{ALL} \subseteq TT_{ALL}$.

Second, we show that if FC_1 finds a task set infeasible, then FC_2 also finds the task set infeasible. To prove this, we will show that if there is $(t_1, t_2) \in TT_{ALL}$ that makes the inequality false, then there is $(t_1^*, t_2^*) \in OD_{ALL}$ that makes the same inequality false. Suppose the inequality is false with $(t_1, t_2) \in TT_{ALL}$. We can transform (t_1, t_2) to (t_1^*, t_2^*) such that t_1^* is $\min\{o_i : o_i \in O_{ALL}, o_i \geq t_1\}$ and t_2^* is $\max\{d_i : d_i \in D_{ALL}, d_i \leq t_2\}$. From its definition, $(t_1^*, t_2^*) \in OD_{ALL}$. We can derive that for each task τ_i, there is j such that $o_{i,j} < t_1 \leq t_1^* = o_{i,j+1}$ and there is m such that $d_{i,m} = t_2^* \leq t_2 < d_{i,m+1}$. From Lemma 2, we can see that for each τ_i, $\eta_i(t_1^*, t_2) = \eta_i(t_1, t_2)$ and $\eta_i(t_1, t_2^*) = \eta_i(t_1, t_2)$. By obtaining t_1^* and t_2^* in turn, we can eventually get $\eta_i(t_1, t_2) = \eta_i(t_1^*, t_2^*)$ for all τ_i and

$$\sum_{i=1}^{n} \eta_i(t_1, t_2) = \sum_{i=1}^{n} \eta_i(t_1^*, t_2^*).$$

Since $t_1 \leq t_1^*$ and $t_2^* \leq t_2$,

$$\sum_{i=1}^{n} \eta_i(t_1, t_2) > t_2 - t_1 \rightarrow \sum_{i=1}^{n} \eta_i(t_1^*, t_2^*) > t_2^* - t_1^*.$$

Third, we show that if FC_2 finds a task set feasible, then FC_1 also finds the task set feasible. As we described in the last paragraph, we can transform each $(t_1, t_2) \in TT_{ALL}$ to $(t_1^*, t_2^*) \in OD_{ALL}$ satisfying

$$\sum_{i=1}^{n} \eta_i(t_1, t_2) = \sum_{i=1}^{n} \eta_i(t_1^*, t_2^*).$$

Since $t_1 \leq t_1^*$ and $t_2^* \leq t_2$,

$$\sum_{i=1}^{n} \eta_i(t_1^*, t_2^*) \leq t_2^* - t_1^* \rightarrow \sum_{i=1}^{n} \eta_i(t_1, t_2) \leq t_2 - t_1.$$

Fourth, we show that if FC_2 finds a task set infeasible, then FC_1 also finds the task set infeasible. This is obvious from $OD_{ALL} \subseteq TT_{ALL}$. □

From $OD_{ALL} \subseteq TT_{ALL}$. we can see that the feasibility condition given in Theorem 1 can be more efficiently evaluated than the one given in Lemma 1.

5 Optimization Formulation and Solutions

5.1 Optimization Formulation

Once the PCM generates the temporal parameters (periods, offsets and deadlines) of tasks and feasibility constraints are generated with the feasibility condition shown in Theorem 1 based on the temporal parameters of tasks, the optimization framework, i.e., the back-end technique, tries to minimize the system code size while satisfying all the feasibility constraints. This optimization uses the code size vs. execution time tradeoff relationship of each task for this purpose. More formally, this optimization is formulated as follows:

- Objective: the objective function is as follows:

$$\text{Minimize} \sum_{\tau_i \in \tau_{sys}} s_i. \qquad (2)$$

- Constraints: the feasibility constraints are from Theorem 1 as follows:

$$\sum_{i=1}^{n} \eta_i(t_1, t_2) \cdot e_i \leq t_2 - t_1, \text{ for all } (t1, t2) \in OD_{ALL}.$$

Depending on the tradeoff function of each task, this optimization problem can be solved by linear programming or heuristic approaches.

Linear programming (LP) problem. If all the tradeoff functions are linear, this optimization problem becomes an LP problem with the objective of minimizing $\sum_{\tau_i \in \tau_{sys}} f_i(e_i)$. Thus, it can be solved by an existing LP solver.

NP-hard problem. If the tradeoff functions are given as discrete step functions, the problem of finding the optimal solution becomes intractable.

Theorem 2 *When the code size vs. execution time tradeoff is given as a tabular form, the problem of finding the minimum system code size satisfying the real-time requirements is NP-hard.*

Proof. The proof is via a polynomial-time reduction from the subset sum problem that is known to be NP-complete [4]. Let a set of positive integers $A = \{a_1, \ldots, a_n\}$ and k represent an instance of the subset sum problem. Assume that for each i, $1 \leq i \leq n$, $a_i \geq 1$ and $\sum_{i=1}^{n} a_i = M$. For reduction, for each i, $1 \leq i \leq n$, we first construct the tradeoff for τ_i in the form of (s_i, e_i) as $(\epsilon, a_i - \epsilon)$ and $(a_i - \epsilon, \epsilon)$ such that $\epsilon < \frac{1}{n}$ and then construct the temporal parameters of τ_i such that $O_i = 0$, $D_i = k$, and $T_i = 2k$. In any schedule, $\sum_{i=1}^{n}(e_i + s_i) = M$. In any feasible schedule, $\sum_{i=1}^{n} e_i \leq k$. Then, we know that in any feasible schedule, $\sum_{i=1}^{n} s_i \geq M - k$. Considering ϵ, we know that the minimum system code size is between $M - k$ and $M - k + 1$. The minimum system code size is achieved if and only if there is a subset A' of A such that the sum of the elements of A' is k. □

Given the difficulty of this optimization problem when the tradeoff is given as discrete functions, a natural approach is to develop heuristic algorithms that find sub-optimal solutions.

5.2 Heuristic Algorithms

We present heuristic solutions to the optimization problem that can be used when tradeoff functions are given as discrete step functions. Basically, we consider a solution that gradually reduces the system code size by increasing the execution time of a task. We give three greedy algorithms that differ only in the sequence of tasks they consider in increasing the execution time. In all the three algorithms, the iterative procedure continues until there is no task eligible to increase its execution time. Before explaining the three algorithms individually, we define a few terms.

When the execution time of a task τ_i increases, let δ_i denote the amount of execution time by which e_i increases. Since a solution to the optimization problem should satisfy the feasibility constraints, each e_i can be increased as long as it does not violate any of the feasibility constraints. With the feasibility constraints on e_i, we can derive the upper bound δ_i^{max} of δ_i as follows:

$$\delta_i^{max} = \min\Big(\forall(t_1,t_2) \in OD(\tau_i) : t_2 - t_1 - \sum_{i=1}^{n} \eta_i(t_1,t_2)\Big),$$

where $OD(\tau_i)$ is a subset of OD_{ALL} that includes at least one $W_{i,j}, j \le 0$.

We define a reduction ratio (ρ_i) for a task τ_i as a ratio of the amount of code size reduction to the amount of execution time increase. When e_i increases by δ_i, its reduction ratio ρ_i is given by

$$\rho_i = \frac{f_i(e_i) - f_i(e_i+\delta_i)}{\delta_i}, \quad \delta_i \ge 0.$$

With the range of δ_i that is $0 \le \delta_i \le \delta_i^{max}$, we can define the best reduction ratio ρ_i^* as follows:

$$\rho_i^* = \max(\rho_i), \quad \text{where } 0 \le \delta_i \le \delta_i^{max}.$$

Let δ_i^* denote the value of δ_i that gives ρ_i^*.

Highest Best Reduction-Ratio First (HBRF). This algorithm favors a task with a higher best reduction ratio. In each iterative step, the HBRF algorithm calculates the best reduction ratio ρ_i^* for each task τ_i and increases e_m by δ_m^* such that ρ_m^* is the highest among all ρ_i^*.

Longest Period First (LPF). This algorithm favors a task with a longer period. Given a fixed time interval, a task with a longer period has a smaller number of jobs than a task with a shorter period. Thus, in general, increasing the execution time of a task with the longest period has the least impact on the schedulability of the task set. The LPF algorithm captures this and, in each iterative step, this algorithm increases e_m of τ_m by δ_m^{max} where τ_m is the task with the longest period with $\delta_m^{max} \ge 0$. If there are multiple tasks that have the same longest periods, the algorithm chooses task τ_m with the highest ρ_m^* among them and increases e_m by δ_m^*.

Highest Best Weighted-Reduction-Ratio First (HBWF). This algorithm combines the HBRF algorithm and the LPF algorithm in a sense that it favors a task with a higher best reduction ratio and a lower increase in utilization. The HBWF algorithm defines a weight of a task as $\frac{T_i}{T_{LCM}}$ and determines a weighted reduction ratio as follows:

$$w\rho_i = \frac{T_i}{T_{LCM}} \cdot \frac{f_i(e_i) - f_i(e_i+\delta_i)}{\delta_i}, \quad \delta_i \ge 0.$$

This algorithm determines the best weighted reduction ratio $w\rho_i^*$ for each τ_i in the same way as the HBRF algorithm does ρ_i^* and increases e_m by δ_m^* where $w\rho_m^*$ is the highest among all $w\rho_i^*$.

If the tradeoff relationship for each task τ_i is given by a table (i.e., a step function), one of the entries in the table gives the ρ_i^*. Assuming that the average number of entries is h, we can inspect one task in $O(h)$. Therefore, the time complexity of each iterative step for the HBRF and HBWF

HBRF:

```
do {
    for all $\tau_i \in \tau_{sys}$,
        calculate $\rho_i^*$ for $\delta_i$,  $0 \le \delta_i \le \delta_i^{max}$, such that
            $\rho_i^* = \max\Big(\frac{f_i(e_i) - f_i(e_i+\delta_i)}{\delta_i}\Big)$.
    determine $\tau_m$ such that $\rho_m^* \ge \rho_i^*$ for all $\tau_i \in \tau_{sys}$.
    $e_m = e_m + \delta_m^*$.
} until ($\rho_m^* = 0$).
```

LPF:

```
$Q_{sys} = \tau_{sys}$.
while ($Q_{sys}$ is not empty) {
    determine $\tau_m$ such that $T_m \ge T_i$ for all $\tau_i \in Q_{sys}$.
    $e_m = e_m + \delta_m^{max}$.
    exclude $\tau_m$ from $Q_{sys}$.
}
```

HBRF:

```
do {
    for all $\tau_i \in \tau_{sys}$,
        calculate $w\rho_i^*$ for $\delta_i$,  $0 \le \delta_i \le \delta_i^{max}$, such that
            $w\rho_i^* = \max\Big(\frac{T_i}{T_{LCM}} \cdot \frac{f_i(e_i) - f_i(e_i+\delta_i)}{\delta_i}\Big)$.
    determine $\tau_m$ such that $w\rho_m^* \ge w\rho_i^*$ for all $\tau_i \in \tau_{sys}$.
    $e_m = e_m + \delta_m^*$.
} until ($w\rho_m^* = 0$).
```

Figure 3. Heuristic algorithms

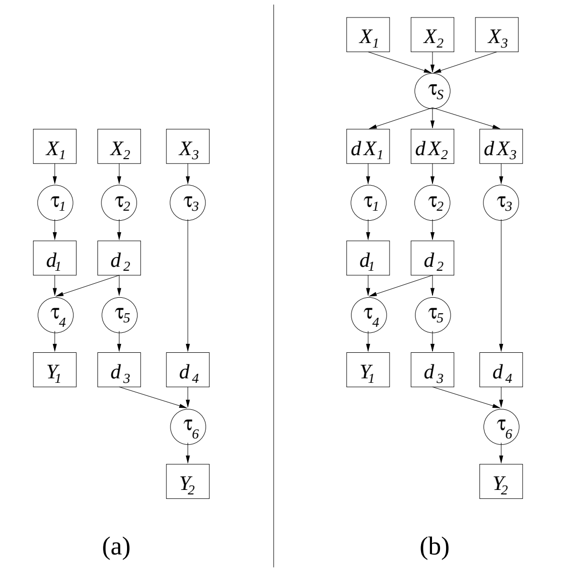


Figure 4. Examples of asynchronous task graph: (a) original task graph and (b) transformed task graph

Freshness	$F(Y_1\|X_1) = 30, F(Y_1\|X_2) = 30$ $F(Y_2\|X_2) = 20, F(Y_2\|X_3) = 15$
Correlation	$C(Y_1\|X_1, X_2) = 3$ $C(Y_2\|X_2, X_3) = 4$
Separation	$l(Y_1) = 21, u(Y_1) = 32$ $l(Y_2) = 29, u(Y_2) = 41$

Table 1. The system end-to-end timing constraints of the example

algorithms is $O(n \cdot h)$ where n is the number of tasks. Since the LPF algorithm increases the execution time of the task τ_m with the longest period by δ_m^{max}, the time complexity of each iterative step for the LPF algorithm is $O(n)$.

6 Illustration and Evaluation

6.1 Design Framework Illustration

In this section, we illustrate how our design framework works with the same example given in the PCM paper [5]. The system structure of the example is given by the task graph shown in Figure 4(a) and the system end-to-end timing constraints are given in Table 1. For the illustration purposes, the tradeoff function of each task is given by both the linear tradeoff function shown in Figure 5(a) and a table of possible (code size, execution time) pairs shown in Figure 5(b). Taking as its input the task graph, the end-to-end timing constraints, and the minimum execution time of each task, PCM transforms the task graph of Figure 4(a) into the task graph shown in Figure 4(b) and derives the temporal parameters of each task. Among various solution candidates, PCM chooses one shown in the upper part (Solution I) of Table 2 since it provides the minimum utilization with the minimum execution time of each task shown in Table 3. Our design framework modifies the deadline of each task in such a way that the task precedences are maintained under EDF scheduling. When the execution time of each task is initially its minimum value, the system code size is 15.05 Kbytes shown in Table 3.

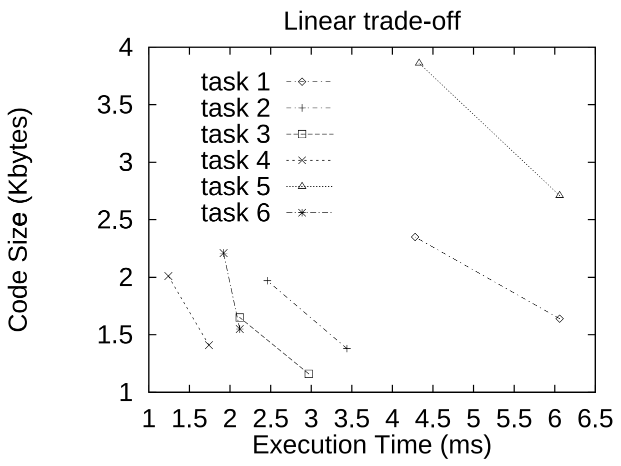
Solutions		τ_s	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
	T_i	13	26	13	39	26	39	39
Solution I	O_i	0	0	0	0	21	0	13
	D_i	3	21	12	13	26	13	15
	T_i	15	30	15	30	30	30	30
Solution II	O_i	0	0	0	0	21	0	13
	D_i	3	21	12	13	26	13	15

Table 2. The solutions of PCM for the example

	τ_s	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_{sys}
s_i	1.0	2.35	1.97	1.65	2.01	3.86	2.21	15.05
e_i	1.0	4.28	2.46	2.12	1.24	4.33	1.92	

Table 3. Maximum code size and minimum execution time of each task

The solution of PCM that is given by a set of task temporal parameters now becomes the input to our optimization framework. It first generates the feasibility constraints given by Theorem 1 from the set of task temporal parameters. With these feasibility constraints, the optimization problem formulated in Section 5.1 can be solved by an LP solver, an exhaustive search, or heuristic algorithms depending on the type of the code size vs. execution time tradeoff of each task. If the tradeoff is given as a linear function shown in Figure 5(a), the optimization problem is formulated as a linear programming (LP) problem and its solution can be obtained by an LP solver. Table 4 shows the solution given by the back-end in its upper part (Solution I). This solution by the LP solver reduces the system code size to 12.57 Kbytes. For the tradeoff relationship of a task given in a tabular form as shown in Figure 5(b), an exhaustive search was performed to find the optimal solution over all possible execution time values and the three heuristic algorithms were used to find sub-optimal solutions with a reduced time complexity. The upper part (Solution I) of Table 5 shows the so-



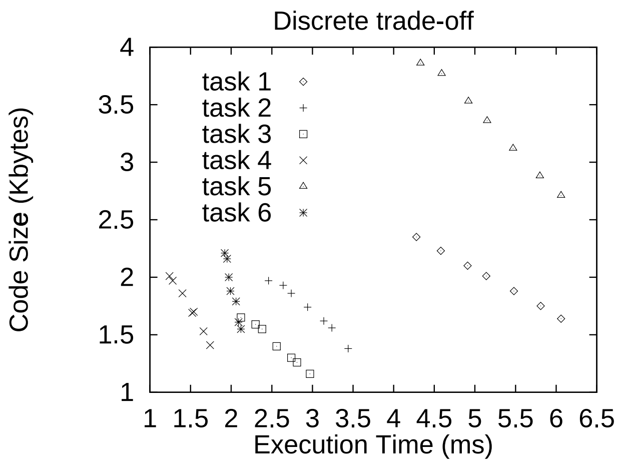


Figure 5. Code size vs. execution time tradeoff for the example:(a) linear and (b) discrete.

Solutions	Parameters	τ_s	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_{sys}
Solution I	Code Size	1.00	1.62	1.98	1.66	1.41	2.96	1.94	12.57
	Execution Time	1.00	6.06	2.46	2.12	1.74	5.68	2.00	
Solution II	Code Size	1.00	2.04	1.40	1.44	1.41	2.70	1.94	11.93
	Execution Time	1.00	5.00	3.43	2.51	1.74	6.06	2.00	

Table 4. The solutions by the back-end for the example with linear tradeoff

Solutions	Algorithms	s_s	s_1	s_2	s_3	s_4	s_5	s_6	s_{sys}
Solution I	OPT	1.00	1.64	1.97	1.59	1.41	3.12	1.88	12.61
	HBRF/LPF/HBWF	1.00	1.64	1.97	1.65	1.97	2.71	1.88	12.82
Solution II	OPT	1.00	2.10	1.38	1.30	1.41	2.88	1.88	11.95
	HBRF/LPF/HBWF	1.00	1.64	1.97	1.55	1.69	2.71	1.88	12.44

Table 5. The solutions by the back-end for the example with discrete tradeoff

lution by an exhaustive search (OPT) and those of the three heuristic algorithms (HBRF/LPF/HBWF). The system code size is reduced to 12.61 Kbytes by an exhaustive search and to 12.82 Kbytes by all the three heuristic algorithms. While the three heuristic algorithms (HBRF/LPF/HBWF) do not always yield the same solution as pointed out in Section 6.2, all the heuristic algorithms provide the same solution in this particular example.

The new execution time of each task given by the solution to the optimization problem becomes the input to PCM. With this new execution time of each task, the utilization is now 0.8174 with PCM Solution I and 0.8173 with PCM Solution II shown in Table 2 after the second path through PCM. The solution process of our optimization framework continues with this new solution by PCM. The lower parts (Solution II) of Table 4 and Table 5 show the new solutions obtained by the back-end with PCM Solution II. When linear tradeoff functions are used, the system code size is reduced to 11.93 Kbytes. When the tradeoff is given in a tabular form, the system code size is reduced to 11.95 Kbytes by the exhaustive search and to 12.44 Kbytes by all the three heuristic algorithms.

PCM takes as its input the new solution (Solution II) by the back-end. PCM gives the same solution as PCM Solution I (with the minimum utilization of 0.8710) and thus the iterative procedure terminates. Among all the solutions of our optimization framework obtained through the iterative procedure, the final solution is the one that minimizes the system code size while satisfying all the feasibility constraints.

6.2 Heuristic Algorithm Evaluation

We evaluate the performance of the three heuristic algorithms against the exhaustive search algorithm through simulations with various task sets. The parameters of each task

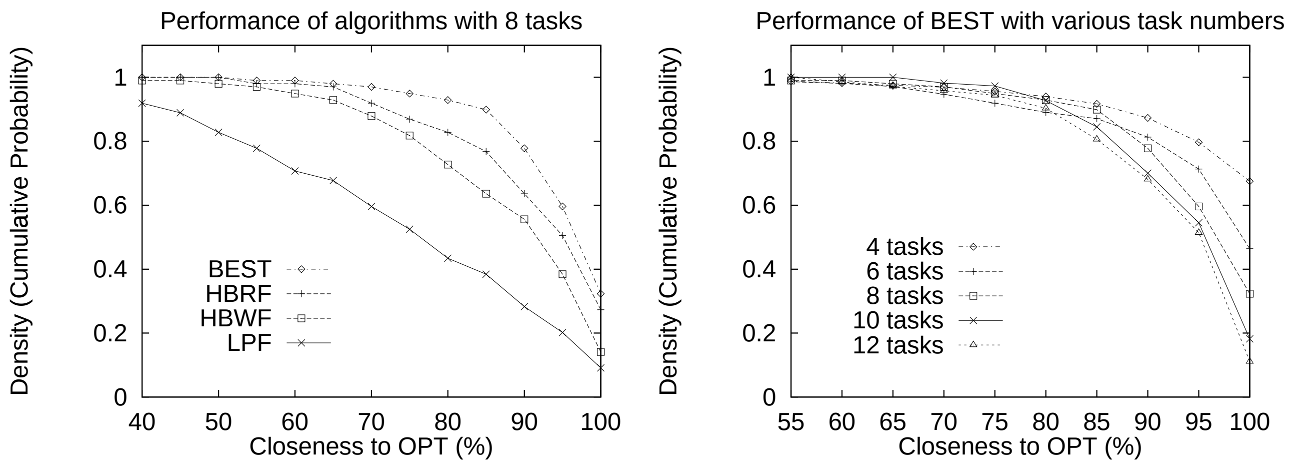


Figure 6. Performance evaluation: (a) performance of the three heuristic algorithms (HBRF/LPF/HBWF) and the best performing one (BEST) with 8 tasks and (b) performance of BEST with various numbers of tasks.

are randomly chosen during the simulations. The period of each task is randomly chosen to be one among 10, 20, 25, 50, and 100 ms. The discrete tradeoff between code size vs. execution time of each task is randomly determined to be 5 pair of values such that the tradeoff values are monotonically decreasing and the execution time of each task is smaller than its period. The offset and deadline of each task are also randomly determined such that its execution window is greater than its execution time. Simulations were performed more than 100 times with 4, 6, 8, 10 and 12 tasks, respectively. As the performance measure, we define the closeness of a non-optimal solution of a heuristic algorithm to the optimal solution by the exhaustive search algorithm as follows:

$$\text{Closeness } = \frac{S_{sys}(init) - S_{sys}(alg)}{S_{sys}(init) - S_{sys}(opt)},$$

where $S_{sys}(init)$ is the initial system code size, $S_{sys}(alg)$ is the system code size determined by the heuristic algorithm, and $S_{sys}(opt)$ is the optimal system code size.

Figure 6(a) plots the performance of the three algorithms and their best-case performance (BEST) with 8 tasks. We can see in Figure 6(a) that at least one of the three algorithms finds the optimal solution for about 32% of simulation cases. With the simulation results, we can state with 90% confidence that the mean of the best-case performance with 8 tasks in the real-world is between 75% and 100%. Among the three algorithms, HBRF exclusively yields the BEST solutions for 47% of simulation cases, HBWF does so for 18%, and LPF does so for 12%, respectively. For the remaining 23% of simulate cases, two or more algorithms provide the BEST solutions together. Figure 6(b) plots the best-case performance (BEST) of the three heuristic algorithms against 4, 6, 8, 10 and 12 tasks, respectively. The best-case solution is the optimal solution for about 68% with 4 tasks, 46% with 6 tasks, 32% with 8 tasks, 18% with 10 tasks, and 11% with 12 tasks of simulation cases.

7 Conclusion

With the reduced bit-width instruction sets of recent microprocessors, it is possible to take advantage of the code size vs. execution time tradeoff of a task during the design of embedded systems. This paper describes a design framework for real-time embedded systems that provides solutions to the optimization problem of minimizing the system code size subject to guaranteeing the system's real-time requirements under EDF scheduling.

Our design framework decomposes the optimization problem into two sub-problems: 1) deriving task temporal parameters that guarantees the system real-time requirements and 2) deriving the code size and execution time of each task while minimizing the system code size subject to feasibility constraints. Our framework iteratively solves the two sub-problems using solutions from each other until the solutions converge. This iterative approach inherently finds a locally optimal solution. However, it may be possible to develop an integrated approach that solves the optimization problem without breaking it into two sub-problems. Such an integrated approach increases the complexity of finding solutions, but may deliver solutions that are close to the globally optimal solution. We plan to develop such an approach and evaluate the complexity and effectiveness.

Our current design framework considers the issue of minimizing the total code size while guaranteeing the real-time requirements. In the design of real-time embedded systems, energy consumption is another critical design factor. With the Dynamic Voltage Scaling (DVS) technique that allows each job to execute at various CPU clock speeds, there is a possible tradeoff between energy consumption and execution time. Our future direction is to extend our design framework so that it can allow the embedded system designer to evaluate tradeoffs among code size, execution time, and energy consumption.

Acknowledgements

We would like to express our gratitude to Sudipto Guha for his help in proving the NP-hardness of the optimization problem and to Sheayun Lee for providing us with an example of tradeoff data. In addition, we also thank to Sampath Kannan and Minsoo Ryu for helpful discussions.

References

[1] S. Baruah, R. Howell, and L. Rosier, *Algorithms and Complexity Concerning the Preemptive Scheduling of Periodic, Real-Time Tasks on One Processor*, Real-Time Systems 2, pp. 301-324, 1990.

[2] E.G. Coffman, Jr., *Introduction to Deterministic Scheduling Theory*, E.G. Coffman, Jr., Ed., Computer and Job-Shop Scheduling Theory, Wiley, NY, 1976.

[3] S. Furber, *ARM System Architecure*, Addison Wisley, New York, NY, 1996.

[4] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to Theory of NP-Completeness*, W.H. Freeman and Company, 1979.

[5] R. Gerber, S. Hong, and M. Saksena, *Guaranteeing Real-Time Requirements with Resource-Based Calibration of Periodic Processes*, IEEE Transactions on Software Engineering, Vol. 21, No. 7, pp. 579-592, July 1995.

[6] L. Goudge and S. Segars, Thumb: Reducing the Cost of 32-bit RISC Performance in Portable and Consumer applications, In Proceedings of the 1996 COMPCON, Sept. 1996.

[7] A. Halambi, A. Shrivastava, P. Biswas, N. Dutt, and A. Nicolau, *An Efficient Compiler Technique for Code Size Reduction using Reduced Bit-width ISAs*, Design Automation and Test in Europe, March 2002.

[8] J.P. Lehoczky, *Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines*, In Proceedings of the 11th IEEE Real-Time Systems Symposium, pp. 201-209, Lake Buena Vista, FL, December 1990.

[9] J.Y.-T. Leung and M.L. Merrill, *A Note on Preemptive Scheduling of Periodic, Real-Time Tasks*, Information Processing Letters 11(3), 1980.

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

[11] N. Kim, M, Ryu, S. Hong, and H. Shin, *Experimental Assessment of the Period Calibration Method: A Case Study*, Real-Time Systems, pp. 1-26, 1999.

[12] U. Nilsson, S. Streiffert, and A. Törne, *Detailed Design of Avionics Control Software*, In Proceedings of the 19th IEEE Real-Time Systems Symposium, pp. 82-91, Madrid, Spain, December 1998.

[13] D. Sweetman, *See MIPS Run*, Morgan Kaufmann, San Francisco, CA, 1999.