Enhancing Schedulability of Hard Real-Time Systems through Codesign

Youngsoo Shin Kiyoung Choi
School of Electrical Engineering
Seoul National University, Seoul 151-742, Korea
Phone: +82-2-880-5457, fax: +82-2-887-6575
e-mail: ysshin@poppy.snu.ac.kr, kchoi@azalea.snu.ac.kr

Abstract

This paper deals with the problem of hardware-software codesign of hard real-time systems. For a given task set, we perform an exact schedulability test to determine whether the task set is schedulable or not. When there is a task that cannot meet the deadline, we compute the amount of time by which the deadline is missed. Then we determine which tasks should reduce their execution time to compensate that amount of time deviation. The reduction of execution time is achieved by implementing parts of the tasks with hardware. With this approach, we can systematically design a hard real-time system which is infeasible with all software implementation. Preliminary experimental results are given to demonstrate the effectiveness of our approach.

I. Introduction

In recent years, hardware-software codesign has become a major concern for a design of embedded systems. Embedded systems are special purpose systems composed of reprogrammable components(microprocessor, microcontroller, DSP, etc.) and a dedicated application-specific hardware components. As the complexity of this kind of systems increases, a systematic design approach receives a lot of attention.

Most embedded systems are described as a set of processes or tasks which communicate with each other through shared medium or using message passing. Usually timing or resource constraints are specified together with the system description. Depending on the characteristics of timing constraints, a system can be referred to as a hard real-time or a soft real-time system. In a hard real-time system, all tasks should complete their computation within specified deadlines. In usual approaches to a real-time system design, if there exists no feasible schedule for a given task set, redesign processes are repeated through modification of deadlines or periods of some tasks or through tuning of tasks. In designing complex systems, however, these engineering approaches are hard to employ or even are not acceptable. Therefore, we need a systematic approach which will replace the ad hoc approaches.

In this paper, we take an approach in which the execution time of tasks can be reduced by moving some code fragments to hardware components. In other words, when the given task set do not satisfy schedulability condition, we reduce the execution time of some tasks by employing a coprocessor which is a hardware implementation of the code fragments of the tasks. In this approach, the essential problem is a decision about which tasks and which part of the tasks should be implemented with hardware components. This problem can be simplified by dividing it into three subproblems as follows.

1. Which tasks should be entirely implemented with hardware.
2. Which tasks should be partially implemented with hardware and how big is the hardware portion in each of the tasks.
3. Within each of the tasks determined as in 2 above, which portion should be implemented with hardware.

The first two decisions must be made in such a way to minimize hardware resources. When resource constraints are given, decision should be made to meet these constraints. The last subproblem is similar to the circuit partitioning problem. The circuit partitioning problem can be modeled as the graph partitioning problem which is known to be NP-hard[1]. In contrast to the circuit partitioning which is homogeneous, the partitioning in our case is heterogeneous in that some one part is implemented in software and the other part is implemented in hardware. Therefore, the latter problem is considered to be harder to solve. There have been many researches about this problem[2], [3], [4], but it is beyond the scope of this paper.

We tackle the first two subproblems using the exact schedulability test of the rate monotonic scheduling algorithm[5]. The rate monotonic algorithm in which a task with shorter period or with higher execution rate is assigned a higher priority is proved to be an optimal fixed priority algorithm[6]. Among various scheduling algorithms, the rate monotonic algorithm is widely used and of great practical importance[5], [7]. One of the reasons is that there exist schedulability tests[5], [6], [8] for a given task set.

The overall flow of our approach is as follows. First, the system is specified as a set of tasks, timing attributes/constraints, and hardware cost of each task. There have been many researches for estimating the execution time of a task[9], [10] and computing the detailed timing information(period, deadline, etc.) from the timing constraints at the higher abstraction level[11]. We perform an exact schedulability test[5] for the task set. If there exists a task which violates the schedulability condition, we compute the percentage of the task's execution time to be cut off in order to satisfy the schedulability condition. Then parts of the task are implemented with hardware such that the execution time is reduced. All these processes are performed in such a way to minimize the hardware cost of the resultant implementation. The partitioner also finds hardware modules which are shared among different parts in the tasks. Hardware sharing incurs block-

0-7803-3583-X/97 $10.00

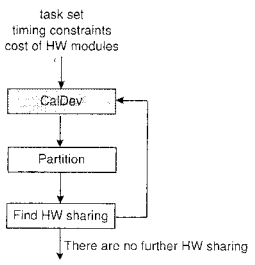

Fig. 1. Conceptual design flow.

ing or mutually exclusive relation for tasks thereby lowering schedulability. Using this information on hardware sharing, we repeat the above mentioned procedures until all tasks meet their deadlines. All these processes are illustrated in Fig. 1. Automatic partitioning is currently under development and is beyond the scope of this paper.

The rest of the paper is organized as follows. In the next section, we describe basic assumption and its underlying effects on the design of hardware-software mixed systems. We propose and describe a new algorithm for achieving schedulability using schedulability test in section III. In section IV, we consider the effect of hardware resource sharing and task blocking. We show experimental results in section V and draw conclusions in section VI.

II. Basic Assumption

To simplify our problem we make assumptions as follows.

- Hardware components have bounded delay. This is enforced by the property of predictability of real-time systems.
- Communication overhead between hardware and software is determined only by the number and the type of data to be transferred.

The first assumption can be relaxed in non real-time or in soft real-time applications and in that case handshaking protocol can be used to synchronize a communication between hardware and software. However, in hard real-time systems which enforce predictability of system behavior, bounded delay of hardware components is a necessary condition. The property of bounded delay of hardware components can be effectively used in two respects. First, handshaking is not necessary because the completion time of hardware execution can be known in advance(off-line). Therefore, the overhead for synchronization can be removed thereby enhancing the system performance. Second, a measure of parallelism between hardware and software can be determined in advance and used for execution overlap between hardware and software. In this case, the execution overlap can be efficiently realized with static scheduling only in contrast to the realization with mixed static and dynamic scheduling in[12], where unbounded delay is allowed.

The second assumption is based on the fact that in most embedded systems, reprogrammable components are usually a bus-master and therefore bus arbitration is not needed. Communication latency is dependent only on the size of data to be transferred and on the bandwidth of the bus between the reprogrammable and the hardware components. Data which can not be packed into the bus bit width should be properly divided into packets and an appropriate communication protocol should be used between the hardware and the software[13].

III. Algorithm for Achieving Schedulability

Consider a set of n periodic tasks $\tau_1, \tau_2, \ldots, \tau_n$, and a pair (T_i, C_i) for each of the tasks, where T_i is the period of task τ_i and C_i is the execution time of the task and τ_i's are assumed to be sorted in the ascending order of the period. We define mrc_i as the maximum value by which we can reduce the execution time of task τ_i. When there are no resource constraints, this value is achieved when τ_i is implemented entirely with hardware. mrc_i can be computed using the following equation.

$$\begin{aligned} mrc_i &= C_i - [input\_communication\_overhead + \\ &\quad critical\_path\_length + \\ &\quad output\_communication\_overhead] \end{aligned} \tag{1}$$

where $critical\_path\_length$ is the latency obtained with hardware after scheduling[1] and allocation.

As a first step, we perform an exact schedulability test[5] for a given task set, which tests on the sets of scheduling points defined by the following equation.

$$S_i = \{kT_j | j = 1, \ldots, i; k = 1, \ldots, \lfloor \frac{T_i}{T_j} \rfloor\} \tag{2}$$

We define $S_{i,j}$ as the jth scheduling point of task τ_i when elements of S_i are sorted in the ascending order. A task τ_i is schedulable if it satisfies the following equation.

$$\min_{\{t \in S_i\}} \frac{\sum_{j=1}^{i} C_j \cdot \lceil \frac{t}{T_j} \rceil}{t} \leq 1 \tag{3}$$

We can find out tasks which cannot meet their deadlines in a critical instant[6] by the above equation. For each task which does not satisfy equation (3), we compute the time deviation by which the task misses its deadline. We define $\Delta c_{i,j}$ as the time deviation of task τ_i at the jth scheduling point. It is given by the following equation.

$$\Delta c_{i,j} = \sum_{k=1}^{i} C_k \lceil \frac{S_{i,j}}{T_k} \rceil - S_{i,j} \tag{4}$$

For each $\Delta c_{i,j}$, we compute the time d_{ijk}, $k = 1, \ldots, i$, by which the execution time of each task τ_k must be reduced in order to make τ_i schedulable. It is computed by the following equation.

$$d_{ijk} = \frac{\Delta c_{i,j}}{\lceil \frac{S_{i,j}}{T_k} \rceil} \tag{5}$$

[1]This should not be confused with the scheduling of real time tasks. Scheduling in this phrase means assigning a control step to each operation in hardware implementation and is one of the phases performed in a high-level synthesis.

It can be easily shown that if the execution time of each task τ_k is reduced by the amount given in equation (5), then τ_i can be made schedulable. From this computation, we can compute the minimum required time by which the execution time of τ_k must be reduced in order to make all tasks schedulable as follows.

$$\mathcal{D}_k = \max_i \left[\min_{j,k} d_{ijk} \right] \tag{6}$$

For any k, if we reduce the execution time of τ_k by $\mathcal{D}_k$, then all tasks become schedulable. However, if $\mathcal{D}_k$ is larger than mrc_k, the maximum value we can take off from the execution time of task τ_k, then it is impossible to achieve our goal by only reducing the execution time of task τ_k. We solve this problem through iteration. First, we try to reduce the execution time of τ_1 by $\mathcal{D}_1$. If $\mathcal{D}_1$ is larger than mrc_1, then we reduce the execution time of τ_1 by mrc_1. In that case, we iterate the above steps with $\tau_2, \tau_3, \ldots$ until all tasks become schedulable. We start from τ_1 because it is more effective than starting from any other task. Note that $\mathcal{D}_m$ is always smaller than or equal to $\mathcal{D}_n$, provided that m is smaller than n. In the $(k+1)$th iteration, $\Delta c_{i,j}$ computed in the kth iteration should be updated. Note that the execution time of $\tau_1, \ldots, \tau_k$ have been modified during the first k iterations. This computation is performed incrementally using the following equation.

$$\Delta c_{i,j}^{k+1} = \Delta c_{i,j}^{k} - mrc_k \cdot \lceil \frac{S_{i,j}}{T_k} \rceil \tag{7}$$

The following example helps clarify the above mentioned procedures. Consider the case of three tasks:

τ_1: C_1=4, T_1=10, mrc_1=2
τ_2: C_2=9, T_2=16, mrc_2=5
τ_3: C_3=7, T_3=25, mrc_3=3

Then

$S_1 = \{T_1\}$
$S_2 = \{T_1, T_2\}$
$S_3 = \{T_1, T_2, 2T_1, T_3\}$

We perform schedulability test for all three tasks at each scheduling points as follows.

$$\begin{array}{lll}
\tau_1: & C_1 \le T_1 & \\
\tau_2: & C_1 + C_2 > T_1, & \Delta c_{2,1} = 3 \\
 & 2C_1 + C_2 > T_2, & \Delta c_{2,2} = 1 \\
\tau_3: & C_1 + C_2 + C_3 > T_1, & \Delta c_{3,1} = 10 \\
 & 2C_1 + C_2 + C_3 > T_2, & \Delta c_{3,2} = 8 \\
 & 2C_1 + 2C_2 + C_3 > 2T_1, & \Delta c_{3,3} = 13 \\
 & 3C_1 + 2C_2 + C_3 > T_3, & \Delta c_{3,4} = 12
\end{array}$$

Then it is found that task τ_1 is schedulable, but τ_2 and τ_3 are not. For each $\Delta c_{i,j}$, we compute the value of d_{ijk}. Fig. 2 shows these values in a tabular form. Fig. 2 (a), which is the result of the first iteration, shows the value of d_{ijk} for each combination of i, j, and k. From these tables, we can conclude that τ_2 can be made schedulable if the execution time of τ_1 is reduced by 0.5 or the execution time of τ_2 is reduced by 1. τ_3 can be made schedulable if C_1 is reduced by 4, C_2 is reduced by 6, or C_3 is reduced

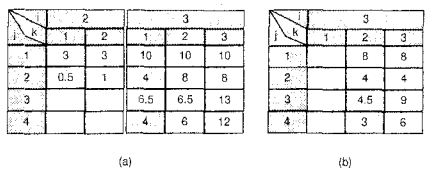

Fig. 2. Deviation time (a)the first iteration (b)the second iteration.

```
Calculate_deviation_time() {
    schedulability_test();
    for (k = 1, 2, ..., n) {
        for (all tasks τ_i wich are not schedulable)
            compute Δc_{i,j} at S_{i,j};
        D_k = max_i [min_{j,k} d_{ijk}];
        if (D_k > mrc_k) {
            Δc_{i,j} = Δc_{i,j} − mrc_k ⌈S_{i,j}/T_k⌉;
            implement whole τ_k with hardware;
        }
        else {
            implement (mrc_k − D_k)/C_k of τ_k with hardware;
            exit loop;
        }
    }
}
```

Fig. 3. A pseudo code for calculation of deviation time.

by 8. Applying equation (6), we obtain $\mathcal{D}_1 = 4$, $\mathcal{D}_2 = 6$, $\mathcal{D}_3 = 8$. First, we try to subtract $\mathcal{D}_1$ from the execution time of τ_1. However, it fails because mrc_1 is lower than $\mathcal{D}_1$. In the second iteration, $\Delta c_{i,j}$ is re-computed using the equation (7), which gives the following results.

$$\Delta c_{2,1} = 1 \quad \Delta c_{2,2} = -3$$

$$\Delta c_{3,1} = 8 \quad \Delta c_{3,2} = 4 \quad \Delta c_{3,3} = 9 \quad \Delta c_{3,4} = 6$$

Negative value of $\Delta c_{2,2}$ indicates that τ_2 is schedulable. For each $\Delta c_{i,j}$, we repeat the computation of equation (5) resulting in the table in Fig. 2 (b). The table for τ_2 is not shown because τ_2 is now schedulable. From the table, we see that $\mathcal{D}_2$ is 3 which is lower than mrc_2. Therefore, in order to satisfy the schedulability of all three tasks, τ_1 is entirely implemented in hardware and the execution time of τ_2 must be reduced by 33%. The pseudo code of the above mentioned procedures is shown in Fig. 3.

IV. Resource Sharing and Task Blocking

When tasks are implemented with hardware, there is a trade-off between the hardware cost and the overall system performance. To reduce the hardware cost, as many components as possible should be shared among tasks, but this causes mutually exclusive relations between tasks and unnecessary blocking time due to priority inversion.

In this case, priority ceiling protocol[14] can be used to solve the problem. When we use this protocol, the schedulability test given in equation (3) is modified as follows.

TABLE I

TIMING ATTRIBUTES OF TASKS OF CNC

	task						
	smpl	xref	yref	dist	stts	xctrl	yctrl
T_i	540	540	540	540	540	1620	1080
C_i	39	51	51	54	216	195	195

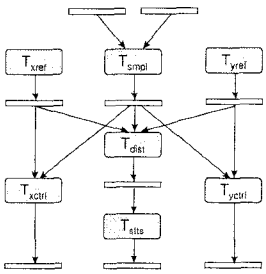

Fig. 4. A task graph of CNC controller.

$$\min_{\{t \in S_i\}} \frac{\sum_{j=1}^{i} C_j \cdot \lceil \frac{t}{T_j} \rceil + B_i}{t} \leq 1 \qquad (8)$$

where B_i indicates the worst case blocking time. When a hardware module is shared among different tasks, B_i is equal to the critical path length of that module.

V. Experimental Results

In this experiment, we have selected a CNC(computerized numerical control) machine controller which is an example of embedded digital controller. The CNC machine is an automatic machining tool which is used to produce user-designed workpieces[15]. A task graph for the controller is shown in Fig. 4. It consists of a set of tasks as shown in Fig. 4 by shaded boxes and a set of buffers together with precedence relations between them. We applied the period calibration method proposed in [11] to the task graph to calculate task-specific attributes(period and deadline) from end-to-end timing constraints. The results of period calibration is shown in Table I together with the execution time of each task. Utilization of the resultant task set is computed as 1.06 which implies that the system is infeasible with all software implementation. We applied our algorithm to this example where we assumed mrc_i to be 70% of C_i for each task τ_i. The result shows that task *smpl* should be implemented totally in hardware and 30% of *xref* should be implemented in hardware in order to meet the condition for the schedulability of all tasks.

VI. Conclusions

In this paper, we have proposed the systematic approach for hardware-software codesign of hard real-time systems. In this approach, we perform schedulability test to find out whether a given task set is schedulable or not. When there are tasks which cannot meet their deadlines at each critical instant, we compute the time deviation by which a deadline is missed. The amount of time deviation is used to determine which tasks should reduce their execution time and what is the amount of each reduction. The reduction in execution time is obtained by implementing parts of tasks with hardware. With this approach, we can design a hard real-time system which is infeasible with software only implementation.

We are exploring the way to extend our work to a more general hard real-time system design. This includes the case where a deadline is not equal to a period and the case where sporadic and/or aperiodic tasks exist. We are also working on extending our work so that hardware cost is considered. Minimum hardware cost can be achieved by avoiding hardware implementation of tasks if the hardware implementation is very expensive.

References

[1] M. Garey and S. Johnson, *Computers and Intractability: a Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[2] K. A. Olukotun, R. Helaihel, J. Levitt, and R. Ramirez, "A software-hardware cosynthesis approach to digital system simulation," *IEEE Micro*, pp. 48–58, Mar. 1994.

[3] A. Kalavade, *System Level Codesign of Mixed Hardware-Software Systems*, Ph.D. thesis, University of California, Berkeley, Sept. 1995.

[4] R. K. Gupta, *Co-Synthesis of Hardware and Software for Digital Embedded Systems*, Ph.D. thesis, Stanford University, Dec. 1993.

[5] J. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: exact characterization and average case behavior," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1989, pp. 166–171.

[6] 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, Jan. 1973.

[7] C. Locke, "Software architecture for hard real-time applications: Cyclic executive vs. fixed priority executives," *The Journal of Real-Time Systems*, vol. 4, no. 1, pp. 37–53, Mar. 1992.

[8] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard real-time scheduling: the deadline-monotonic approach," in *Proc. IEEE Workshop on Real-Time Operating Systems and Software*, May 1991, pp. 133–137.

[9] S. Lim, Y. Bae, G. Jang, B. Rhee, S. Min, C. Park, H. Shin, K. Park, and C. Kim, "An accurate worst case timing analysis for RISC processors," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1994, pp. 97–108.

[10] Y. S. Li, S. Malik, and A. Wolfe, "Performance estimation of embedded software with instruction cache modeling," in *Proc. Int. Conf. on Computer Aided Design*, Nov. 1995, pp. 380–387.

[11] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing end-to-end timing constraints by calibrating intermediate processes," in *Proc. IEEE Real-Time Systems Symposium*, Dec. 1995.

[12] Y. Shin and K. Choi, "Software synthesis through task decomposition by dependency analysis," to appear in Proc. Int. Conf. on Computer Aided Design, Nov. 1996.

[13] K. Kim, Y. Kim, Y. Shin, and K. Choi, "An integrated hardware-software cosimulation environment with automated interface generation," in *Proc. 7th IEEE Int. Workshop on Rapid Systems Prototyping*, June 1996, pp. 66–71.

[14] L. Sha, R. Rajkumar, and J. P. Lehoczky, "Priority inheritance protocols: An approach to real-time synchronization," *IEEE Tr. on Computers*, vol. 39, no. 9, pp. 1175–1185, Sept. 1990.

[15] N. Kim, M. Ryu, S. Hong, M. Saksena, C. Choi, and H. Shin, "Visual assessment of a real-time system design: a case study on a CNC controller," to appear in Proc. IEEE Real-Time Systems Symposium, Dec. 1996.