Real-time Scheduling in Heterogeneous Dual-core Architectures

Kwangsik Kim, Dohun Kim, Chanik Park
System Software Lab, POSTECH
{honesty,hunkim,cipark}@postech.ac.kr

Abstract

With high computational application, embedded systems are becoming more complex. To achieve high performance in the midst of increased complexity, dual-core SoC (System-on-Chip) is used. Of many Dual-core architectures, a general purpose CPU and DSP are most widely used. But only a few scheduling policies for this heterogeneous architectures exist to guarantee real-time character. This paper discusses scheduling policy for heterogeneous dual-core architectures. We explore the problem of previous scheduling policy [9] based on DPCP (Distributed Priority Ceiling Protocol) [4, 5, 6] and provide a solution of strict schedulability bound model.

1. Introduction

With high computational application, embedded systems are becoming more complex. To achieve high performance in the midst of increased complexity, the industry has mostly used heterogeneous systems including multiple programmable processor cores, specialized memories, and other components on a single chip. For example, automotive systems, network processing systems, consumer electronics, and mobile handhelds such as multimedia player, PDA, smart-phone use heterogeneous architectures. Particularly, most mobile handhelds are adapted to a heterogeneous dual-core architecture composed by an ARM processor and a DSP (Digital Signal Processing).

Figure 1 shows the dual-core architecture. For example, the OMAP processor, which is manufactured by TI (Texas Instruments) for mobile applications, includes two cores, ARM926 (ARM9 core) and TMS320C55X (DSP coprocessor) [7, 8]. The Freescale i.300-30 processor is also adapted to this architecture. It includes two cores, ARM11 and StarCore SC140 (DSP processor) [10]. Two cores on chip are used to process complex applications such as multimedia application, and high computational application.

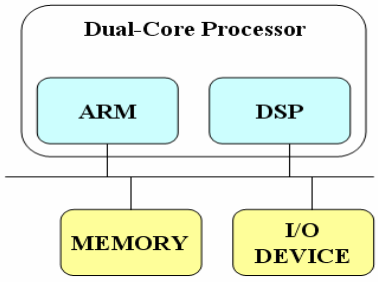

Figure 1. Overview of dual-core architecture

These architectures are mentioned because a multimedia stream can be easily executed using a DSP coprocessor. Reducing computational power needed on the main CPU, DSP specializes for signal processing. Moreover, DSP architectures are designed to have predictable execution times, so they offer a viable alternative to the implementation of fast general purpose multiprocessors [9].

This paper points out the problem of real-time scheduling in heterogeneous dual-core architectures. The important point handled in this work is to verify whether the use of a dedicated processor can still efficiently improve the performance of an embedded system to guarantee a real-time character.

* The authors would like to thank the Ministry of Education of Korea for its support towards the Electrical and Computer Engineering Division at POSTECH through the BK21 program. This research has also been supported in part by the grant number R01-2003-000-10739-0 from the basic research program of the Korea Science and Engineering Foundation, in part by the grant number IITA-2005-C1090-0501-0018 from the Ministry of Information and Communication, Korea, under the Information Technology Research Center support program supervised by the Institute of Information Technology Assessment , in part by the grant number F01-2005-000-10241-0 from international cooperative research program of the Korea Science and Engineering Foundation, and in part by the wearable computer platform project from ETRI.

Real-time scheduling in heterogeneous dual-core architectures did not handle an aspect of the real-time problem. Although many results have been derived for multiprocessor and distributed architectures, very few of them considered the peculiarities of our dual-core architecture composed by ARM core and DSP core. DSP core is different from main processor like ARM core. DSP is a coprocessor, as a DSP accelerator responding to requests of the CPU [1].

We considered that DSP scheduling can be viewed as DPCP - a special case of scheduling with shared resources in multiprocessor distributed systems [6]. The schedulability bound [11] of previous papers adapted to DPCP is too pessimistic and not sufficient to guarantee real-time character.

Our paper contributes two aspects:

- It guarantees more real-time character than the previous method.
- It proposes accurate the schedulability bound of a considered architecture.

In the following section, we will explain system model for a heterogeneous dual-core architecture. In Section 3, we will review a previous scheduling analysis approaches for a heterogeneous dual-core architectures. In Section 4, 5, and 6, we will show a problem of previous scheduling model with a real-time point of view and suggest a new model to improve the real-time character and the schedulability bound formula based on our approach. In Section 7, we will set the experimental environment for simulation and interpret the simulation results. In Section 8, we will state our conclusions and future work.

2. System model

Figure 1 shows two cores on a chip. Two cores share a common bus and can use memory and peripherals. We assume that the cost of communication between two cores is negligible. This assumption is possible because two processors are supposed to be built on the same chip, and both processors can share the memory address space of architecture.

We also assume that all the jobs executing on a DSP are invoked using a remote procedure call (RPC) mechanism and are executed in a non-preemptive way. Such RPCs will be called DSP activities. To simplify the analysis we assume that on CPU, the DSP activities are scheduled one at a time. Hence, it is the real-time kernel on CPU's responsibility to avoid issuing a DSP request while the DSP is active.

The real-time task model considered in this work is illustrated in Figure 2: each task τ_i is a stream of instances, or jobs; each job $J_{i,j}$ arrives at time $r_{i,j}$, executes for C_i units of time on the master CPU, and may require a DSP activity for C_i^{DSP} units of time.

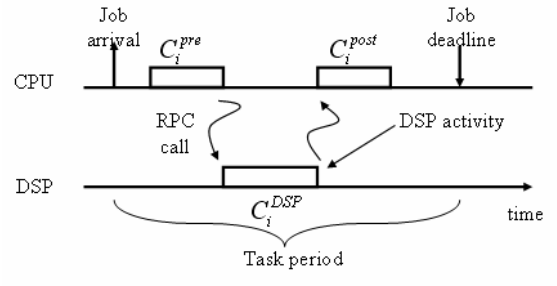

Figure 2. Structure of a DSP task.

We assume that each job performs at most one DSP request, after C_i^{pre} units of time, and then executes for other C_i^{post} units, such as $C_i^{pre} + C_i^{post} = C_i$. Job arrivals can be periodic (if $r_{i,j} - r_{i,j-1} = T_i$, being T_i the task's period) or sporadic (if $r_{i,j} - r_{i,j-1} \geq MIT_i$, MIT_i being the minimum inter-arrival time). If a task requires a DSP activity, it is denoted as a DSP task. Otherwise it is denoted as a regular task. Note that a regular task is equivalent to a DSP task with $C_i^{DSP} = 0$. Whenever a fixed priority scheduling algorithm is used, P_i denotes the priority of task τ_i [9].

3. Related Work

The basic idea of this approach is to re-arrange the scheduling and guarantee algorithms in order to account the DSP time $C_i^{DSP} = 0$ only to tasks that use the DSP (without influencing the schedule and the guarantee of the regular tasks). This can be achieved by modeling the DSP activity as a blocking time: when a DSP task τ_i requests a DSP activity, it blocks for a time B_i waiting for its completion (since we are using an RPC protocol). Moreover, the scheduler has to be modified in such a way that the B_i factor affects only τ_i in the scheduling analysis.

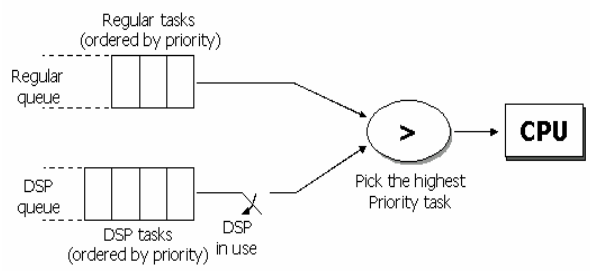

Figure 3. Previous scheduling approach. When the DSP is active, the scheduler selects tasks from the regular queue only [9].

A DSP task, however, can also be delayed by other tasks. In particular, a DSP task τ_i can be delayed by a lower priority job that is already using the DSP, and by those higher priority jobs that may interfere with τ_i before it is scheduled. Hence, the blocking factor B_i of a DSP task can be computed as the sum of three terms:

$$B_i = C_i^{DSP} + B_i^{lp} + B_i^{hp}$$

Where B_i^{lp} denotes the blocking caused by the lower priority task and B_i^{hp} denotes the blocking due to the interference of higher priority tasks. B_i^{lp} and B_i^{hp} are considered with the worst-case scenario. As also done in [4, 5], the two blocking terms can be computed as follows:

$$B_i^{lp} = \max_{P_j<P_i}\{C_j^{DSP}\}$$

$$B_i^{hp} = \begin{cases} \sum_{k=1}^{i-1} \left\lceil \frac{T_i}{T_k} \right\rceil C_k^{DSP}, & \text{for a DSP task} \\ 0, & \text{for a regular task} \end{cases}$$

This value can be used in the classical admission test for fixed priority systems [2, 3, and 6], that is:

$$\forall i = 1,\ldots,n$$

$$\sum_{P_j \geq P_i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq U_{RM}(i) = i(2^{1/i} - 1)$$

4. Problem definition

When executing a DSP task, a hole within each job is generated in the schedule of CPU. Such holes are created because the DSP task executes a DSP activity on another processor. When the DSP task executes a DSP activity, another DSP task cannot execute in previous model case due to the scheduling policy of a DSP task queue. The DSP task with C_i^{pre} will execute during a hole time since a hole time is an available time in view of CPU. However, the previous model does not support the view of C_i^{pre}.

In figure 4, there are two DSP tasks: one task τ_1 with a period $T_1 = 4$ units of time, $C_1^{pre} = 2, C_1^{post} = 0$, and $C_i^{DSP} = 2$ and another task τ_2 with a period $T_2 = 25$ units of time, $C_2^{pre} = 2, C_2^{post} = 0$, and $C_2^{DSP} = 3$. If τ_1 is added to the system, the Rate Monotonic algorithm fail to generate a feasible schedule. The reason is that if the DSP part of τ_2 is active, the task τ_1 cannot execute C_1^{pre} time by scheduling policy of previous model. If the previous model cares a view of C_i^{pre}, the example in figure 4 generates a feasible schedule.

Another problem is that the previous model is not proper from a priority view point. In a priority view point, high priority must execute first. In the previous model, a regular task with low priority can execute than a DSP task with high priority.

Also, if all tasks are DSP tasks, the tasks cannot use a hole time – available CPU time. It means that the system utilization will increase and the schedulability bound will increase.

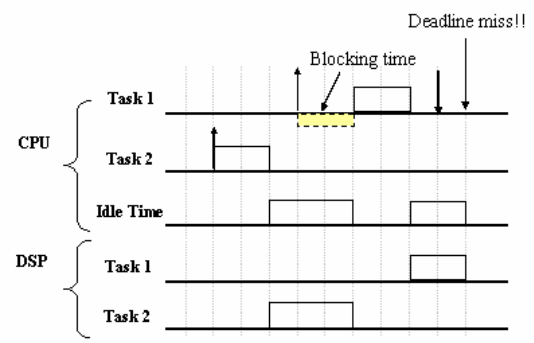

Figure 4. A task set that cannot be feasibly scheduled by RM: task 1 misses all its deadlines.

5. Our approach

Firstly, we continue the assumptions of the previous system model. As we mentioned in Section 4, the previous model (figure 3) has some problems. For solution, we do not separate two queues – a regular task queue and a DSP task queue. Rather, we change the system model as shown in figure 5. Our system

model has a queue combined with DSP tasks and regular tasks and schedule by task priority.

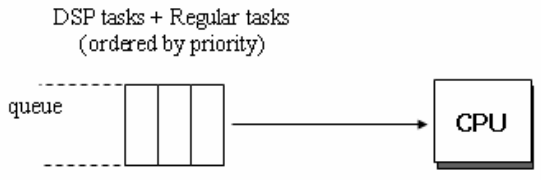

Figure 5. Our scheduling approach.

Figure 6 shows when the previous model changes to our model. There are two DSP tasks: one task τ_1 with a period $T_1 = 9$ units of time, $C_1^{pre} = 1, C_1^{post} = 0$, and $C_i^{DSP} = 2$ and another task τ_2 with a period $T_2 = 9$ units of time, $C_2^{pre} = 2, C_2^{post} = 0$, and $C_2^{DSP} = 1$. In the previous model, If τ_1 and τ_2 is released at the same time, the task τ_2 will finish after 9 units due to the rule of the previous model - when the DSP task executes a DSP activity, another DSP task blocks in the previous model case. The task τ_2 has a block time as 4 units. The task τ_2 will respond slowly. However, in our model, If τ_1 and τ_2 is released at the same time, the task τ_2 will finish after 4 units because our approach can use a hole time. The task τ_2 has no block time. Thus, the task τ_2 will respond fast

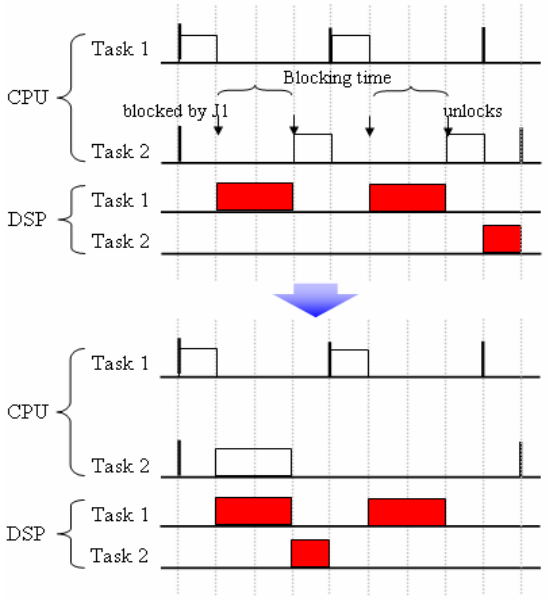

Figure 6. Example of our scheduling approach.

6. Schedulability test

According to the previous approach, the schedulability of the task set is guaranteed by the following test:

$$\forall i = 1,\ldots,n$$

$$\sum_{P_j \geq P_i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq U_{RM}(i) = i(2^{1/i} - 1),$$

where the blocking factor B_i of a DSP task can be computed as the sum of three terms:

$$B_i = C_i^{DSP} + B_i^{lp} + B_i^{hp},$$

where B_i^{lp} denotes the blocking caused by the lower priority task and B_i^{hp} denotes the blocking due to the interference of higher priority tasks. The worst case time of C_i^{DSP} and B_i^{lp} can't reduce anymore. But the worst case time of B_i^{hp} will reduce.

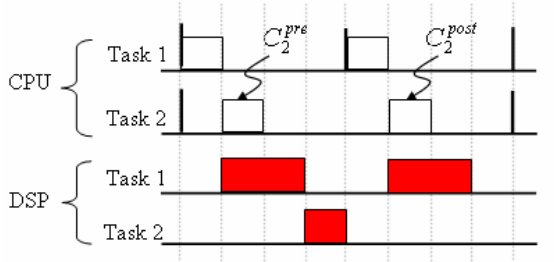

Figure 7. When the task τ_1 is active in DSP, the task τ_2 execute for C_2^{pre} units.

B_i^{hp} is the sum of DSP activation time in higher tasks. In figure 7, C_2^{DSP} is blocked by C_1^{DSP} which executed first. In the view of C_2^{pre} and C_2^{post}, C_2^{post} time is affected by C_2^{DSP} because C_2^{post} will be able to execute after C_2^{DSP} time executes, but C_2^{pre} time is not affected by C_2^{DSP} because C_2^{pre} will have to execute before C_2^{DSP} time executes. C_2^{pre} is independent of C_2^{DSP}, but C_2^{post} is dependent of C_2^{DSP}. The blocking factor B_i^{hp} is generated from C_i^{DSP}. C_i^{pre} will execute first before C_i^{DSP} executes. Thus, we can subtract C_i^{pre} time of tasks from B_i^{hp} which is generated by C_i^{DSP} because C_i^{pre} can use B_i^{hp} as available CPU time. The blocking factors B_i in a DSP task can be computed as follows:

$$B_i^{hp} = \begin{cases} \sum_{k=1}^{i-1} \left\lceil \frac{T_i}{T_k} \right\rceil C_k^{DSP} - \sum_{k=2}^{i} \left\lfloor \frac{T_i}{T_j} \right\rfloor C_j^{PRE}, & \text{for a DSP task} \\ 0, \quad \text{if } B_i^{hp} < 0 \text{ or for a regular task} \end{cases}$$

7. Simulation results

7.1. Simulation environment

We have evaluated the performance of the proposed scheduling algorithm presented in Section 5 by simulation of many task sets. For each task sets we simulated the blocking time of each task and we checked the schedulability using both our model and previous model.

Task sets were generated using random parameters with uniform distribution with the following characteristics:

- The number of tasks was selected as a random variable from 2 to 50
- Task periods were generated from 10 to 1000.
- Task worst-case execution times ($C_i^{'} = C_i + C_i^{DSP}$) were selected in such a way that total utilization varied from 0.01 to 0.99
- DSP tasks were generated to be 80% (in the average) of the total number of tasks.
- C_i^{DSP} was generated to be a random variable with uniform distribution in the range of 10% to 80% of the $C_i^{'}$.
- C_i^{pre} was generated to be a random variable with uniform distribution in the range of 0% to 100% of the C_i .

7.2. Simulation results

In figure 8 and 9, when varying the total utilization factor and the number of task set, schedulability results of the previous model and our model are demonstrated. As total utilization factor increases, the schedulable solution decreases after a point of 0.4. As the number of tasks increases, there is almost no schedulable solution.

In figure 10, our model is compared with the previous model. The value of z-axis is the difference of schedulable solution between our model and the previous model. We note that the advantage of our model with respect to previous model is its sensitivity for task sets with total utilization in the range from 0.6 to 0.8. As the number of tasks increases, the difference between our model and the previous model gradually increases at 0.7 of total utilization. To better evaluate the enhancement achieved with our model, figure 11 also reports the schedulability results of the two models and their difference as a function of the utilization factor for task sets composed by 30 tasks.

The last experiment shows the schedulability results of our model when varying the total utilization factor and the rate of C_i^{pre} in C_i for task sets composed by 30 tasks. We proposed the schedulability bound test using the new blocking factor B_i^{hp} related with C_i^{pre}. When the total utilization is 0.4, there is almost no schedulable solution by the rate of C_i^{pre}. However, when the total utilization is 0.99, there is much in the schedulable solution by the rate of C_i^{pre}. In conclusion, as total utilization increases, the difference of schedulable solution between 0% of C_i^{pre} and 100% of C_i^{pre} increases.

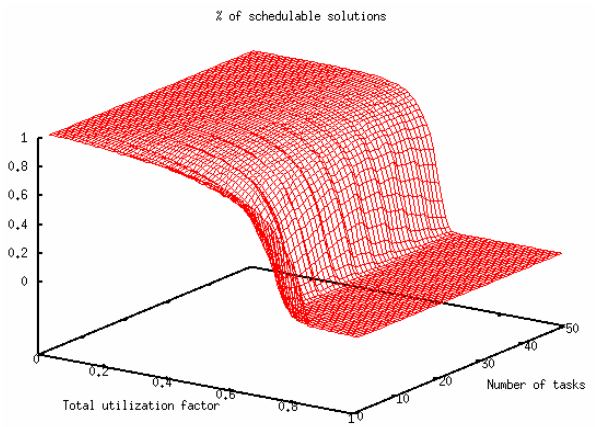

Figure 8. Schedulability results of the previous model when varying the total utilization factor and the number of task set.

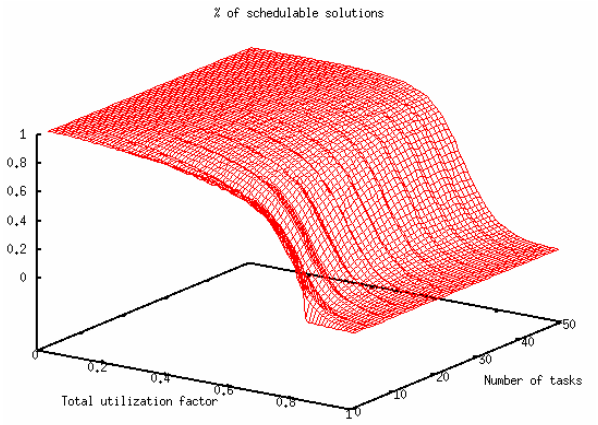

Figure 9. Schedulability results of our model when varying the total utilization factor and the number of task set.

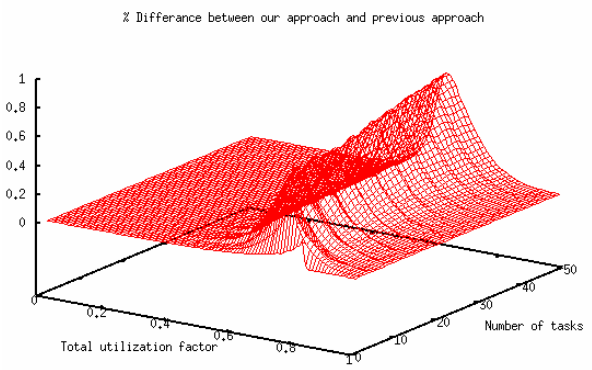

Figure 10. Difference between the two models.

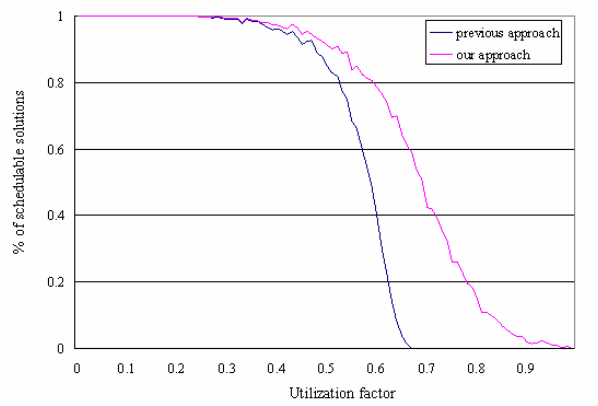

Figure 11. Schedulability results of the two models and their difference as a function of the utilization factor for task sets composed by 30 tasks.

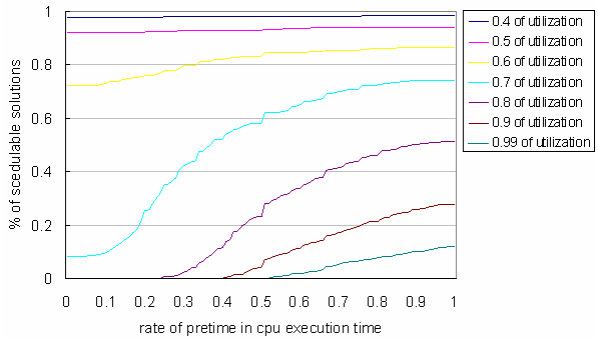

Figure 12. Schedulability results of our model when varying the total utilization factor and the rate of C_i^{pre} in C_i for task sets composed by 30 tasks.

8. Conclusion

We have explained the problem of the previous scheduling model in a heterogeneous multiprocessor composed by a general purpose CPU and a DSP coprocessor. To overcome the problem of the previous scheduling mode, we have proposed the new scheduling model using only a queue combined with regular tasks and DSP tasks ordered by priority. By analyzing the schedulability test, a method for computing blocking times considered by B_i^{hp} has been proposed, which has been showed to be more effective than the previous schedulability test model.

For future work, we plan to make more accurate the schedulability bound and analysis for dynamic priority assignments as well.

9. References

[1] R. Baumgartl and H. Hartig. Dsps as flexible multimedia accelerators. In Second European DSP Education and Research Conference (EDRC'98), Paris, September 1998.

[2] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In Proceedings of the IEEE Real-Time Systems Symposium, December 1989.

[3] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In Proceedings of the IEEE Real-Time Systems Symposium, December 1989.

[4] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real-time synchronization protocols for multiprocessors. In In Proceedings of the 1988 Real Time System Symposium, 1988.

[5] S. Saewong and R. R. Rajkumar. Cooperative scheduling of multiple resources. In Proceedings of the IEEE Real-Time Systems Symposium, December 1999.

[6] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. IEEE transaction on computers, 39(9), September 1990.

[7] OMAP1710 processor architecture and features: http://focus.ti.com/general/docs/wtbu/wtbuproductcontent.tsp?templateId=6123&navigationId=11991&path=templatedata/cm/product/data/omap_1710

[8] ARM9 Core Family http://www.arm.com/products/CPUs/families/ARM9EFamily.html

[9] Paolo Gai, Luca Abeni, Giorgio Buttazzo. Multiprocessor DSP Scheduling in System-on-a-chip Architectures. Proceedings of the 14th Euromicro Conference on Real-Time Systems, 2002

[10] i.300-30 processor architecture and features: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=i.300-30&nodeId=01J4Fsm6cyDbFf

[11] Wei Kuan Shih, J. W. S. Liu and C. L. Liu Modified Rate-Monotonic Algorithm for Scheduling Periodic Jobs with Deferred Deadlines. IEEE Transactions on Software Engineering, 19(12):1171-1179, December 1993.