Hardware-Aware Rate Monotonic Scheduling Algorithm for Embedded Multimedia Systems

Jaebeom Park and Joonhyuk Yoo

Many embedded multimedia systems employ special hardware blocks to co-process with the main processor. Even though an efficient handling of such hardware blocks is critical on the overall performance of real-time multimedia systems, traditional real-time scheduling techniques cannot afford to guarantee a high quality of multimedia playbacks with neither delay nor jerking. This paper presents a hardware-aware rate monotonic scheduling (HA-RMS) algorithm to manage hardware tasks efficiently and handle special hardware blocks in the embedded multimedia system. The HA-RMS prioritizes the hardware tasks over software tasks not only to increase the hardware utilization of the system but also to reduce the output jitter of multimedia applications, which results in reducing the overall response time.

Keywords: Hardware-awareness, real-time scheduling, embedded multimedia systems.

I. Introduction

An embedded multimedia system must guarantee the specific rate and timing requirements to deliver or stream continuous media data such as audio and video [1], [2]. In general, multimedia systems have the following characteristics [3]-[7]. First, a typical requirement of continuous media is that data must be delivered to a client by a certain deadline because data arriving after the presentation time is unusable. Multimedia systems thus require hard real-time scheduling to ensure that a critical task should be serviced within a guaranteed period of time. Second, multimedia applications are sensitive to timing delays during playback. Once a continuous media file is delivered to a client, delivery must continue at a certain rate during playback of the media. When missing the deadline due to the output jitter, the quality of multimedia applications severely degrades and the listener or viewer will be subjected to long pauses or lost frames of video during the presentation due to the jitter. Thus, minimizing these timing delays provides a certain quality of service (QoS) guarantee.

Multimedia data generally requires huge computational power because compression, decoding, and rendering may require significant CPU processing [4], [6]-[8]. Most of embedded multimedia systems are currently employing special hardware blocks to co-process with the main processor to support multimedia applications of high quality and compensate for its demanding computational power [8]. For example, Qualcomm's MSM chipsets equip two DSPs inside their chipsets to support multimedia applications, and TI's OMAP chipsets also embed one or more DSPs. In addition, other chipsets such as SAMSUNG's S3C6400 integrate hardware CODEC blocks to encode and decode video streams.

Manuscript received Feb. 22, 2010; revised June 8, 2010; accepted July 15, 2010.
This work was supported by the Daegu University Research Grant, 2009.
Jaebeom Park (email: jbm.park@samsung.com) is with the Optics and Imaging Division, Samsung Techwin Co. Ltd., Seoul, Rep. of Korea.
Joonhyuk Yoo (correspondence author, email: joonhyuk@daegu.ac.kr) is with the College of Information and Communication Engineering, Daegu University, Gyeongsan, Rep. of Korea.
doi:10.4218/etrij.10.1510.0027

In these systems, the performance of the special hardware block determines the performance of the system and the quality of multimedia applications. Thus, the system needs to handle the hardware block efficiently without delays and jerking to support the high quality of multimedia applications. Since the hardware blocks are controlled by tasks (threads or processes), in terms of software, efficient handling of the tasks is crucial on the overall performance of the system and quality of multimedia applications.

This paper presents an efficient real-time scheduling algorithm, hardware-aware rate monotonic scheduling (HA-RMS), to manage the tasks which control special hardware blocks on the multimedia system. The proposed HA-RMS algorithm reduces the response time and output jitter of the tasks which control the hardware block. So, the utilization of the hardware block is increased, and the performance of multimedia applications will be greatly enhanced.

Multimedia applications are required to play high-quality contents maintaining consistent quality during playback. A multimedia system must process a huge amount of computations to support high quality contents. In some systems, the main processor processes all computation to execute multimedia applications. However, most multimedia systems adopt a heterogeneous architecture to distribute computation power to special hardware blocks (DSPs, coprocessors, or special-purpose hardware IPs). In the heterogeneous multimedia system, the performance of the special hardware block determines the performance of the multimedia system because the special hardware block usually computes the heaviest part of computation. Therefore, the system needs to control the special hardware blocks efficiently to increase the performance of multimedia applications.

In addition to supporting high-quality contents, maintaining consistent quality of playback is important in terms of the quality of multimedia applications. To reduce audio/video jerking and delays of playback, the system needs to reduce output jitters [9] and increase responsiveness to tasks. Therefore, an efficient scheduling of multimedia tasks is very important to guarantee QoS in multimedia applications because of real-time characteristics [1], [4], [6], [7]. Especially in the heterogeneous multimedia systems, we need to solve real-time issues of tasks which control the special hardware blocks to obtain the best quality of multimedia applications because the performance of special hardware blocks determines the performance of multimedia applications.

In this paper, we present an efficient scheduling algorithm, the HA-RMS algorithm, to schedule the tasks that control hardware blocks on the heterogeneous multimedia systems. Applying the HA-RMS algorithm to the real-time multimedia applications, we show that the HA-RMS algorithm improves responsiveness and reduces output jitters of the tasks. As a result, we will prove that the HA-RMS algorithm enhances the performance of multimedia applications by reducing audio/video jerking and delays of audio/video playback.

Section II presents the background and motivation. Section III describes system environments and task-models. Section IV introduces the proposed HA-RMS algorithm and compares it with the traditional RMS algorithm by employing some scenario examples. Evaluation and analysis of the results are shown in section V, and section VI concludes this paper.

II. Motivation

Nowadays, most multimedia system-on-chips (SoCs) are equipped with low-speed CPUs to reduce power consumption and unit price [8]. In these types of multimedia SoCs, they embed special hardware blocks inside the chipsets to supplement the lack of processing power due to the low-speed CPU. Hardware blocks on multimedia SoCs proceed certain computations which require heavy computation power and repeat the same computations frequently. For example, multimedia SoCs usually embed hardware blocks which decode and render video/audio streams. Meanwhile, these hardware blocks are controlled by software tasks (threads), and tasks dealing with hardware blocks have the following typical characteristics in comparison to traditional software-oriented tasks:

Characteristic 1 (C1). Hardware tasks usually finish their job in a short time. Since most of their work is setting up a few hardware registers, they finish their job in a moment and release the processor resource back to other tasks soon. This is a concept close to the I/O-bound processes [10]. Therefore, their job execution time is much shorter than for one of the software tasks.

Characteristic 2 (C2). Hardware tasks depend on the completion of the earlier requests. Since the hardware block is a resource which executes a single process at a time, tasks of controlling hardware blocks should be protected by MUTEX or be handled similarly. Moreover, its slow responsiveness results in poor utilization of the requested hardware block.

Because of the C2, RMS scheduling is not a suitable algorithm to schedule tasks which control hardware blocks. Although not being dependent on a certain request, initiation or completion of other tasks, they depend on the completion of the hardware blocks operation. This fact implies that the tasks controlling hardware blocks should be scheduled within a proper time without delay. Otherwise, utilization of the hardware block will fall down. A detailed example for this

Table 1. Typical example of tasks in embedded SoC platform.

Task	Priority	Job execution time	Task type
A	High	30 ms	Software task
B	Middle	1 ms	Hardware task
C	Low	1 ms	Hardware task

issue will be shown in a later section.

In case of the multimedia SoCs with special hardware blocks, some tasks are run by software only with CPU processing, that is, they occupy CPU in a long time while consuming most of processing power and other tasks control hardware blocks in a short time in order to increase system performance. Being preempted by software-tasks, hardware tasks wait for a long time until software-tasks are finished. In contrast, software-tasks do not have to wait long even though preempted by many hardware-tasks because hardware-tasks are scheduled just for a short time (C2). For example, suppose that there are three tasks.

In Table 1, if task B and C, hardware-tasks, are preempted by task A, each of tasks B and C is delayed by 30 ms per each request of task A. However, task A would be delayed only 1 ms per each request of task B and C after letting the priority of task A be the lowest. This observation provides us a suitable way to apply RMS algorithm with small modifications, taking both C1 and C2 characteristics of hardware and software tasks into consideration.

With the unmodified RMS algorithm, if the period of a hardware-task is longer than that of a software-task which executes for a long time in a single request, the hardware task should be preempted by the software-task [2], [11]. As a result, the response time of hardware-task becomes worse due to the long period of preemption. However, dealing with hardware-tasks separately, we can make hardware-tasks preempt software-tasks, and it will provide pretty short response time for the hardware-task. Although the response time of software-task becomes longer in the result of modification, the amount of delay will be very small because hardware-tasks do not preempt software-tasks for a long time.

III. System Model

Since the HA-RMS algorithm originates from the RMS in terms of the software platform, it uses the same scheduling environments as in the original paper on RMS [11]. In [11]-[14], a scheduling algorithm is based on the priority of each task, and is preemptive. That is, if a task has a higher priority than that of the task currently running, the current task must be immediately interrupted by the task with higher priority. Since most of commercial real-time operating systems and even high-level operating systems provide priority-based preemptive schedulers, this algorithm would not be seriously dependent on the lower software platform, that is, less dependent on O/S and kernel.

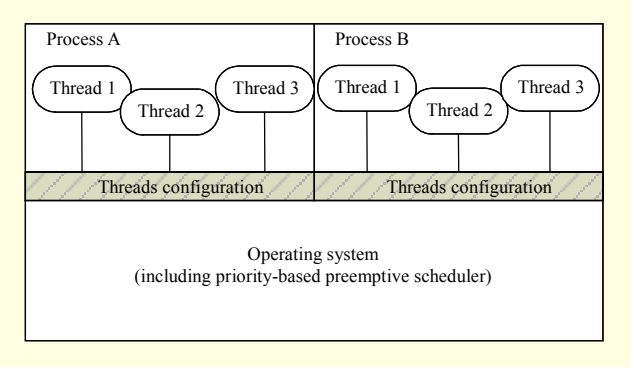


Fig. 1. Location of HA-RMS algorithm on software platform.

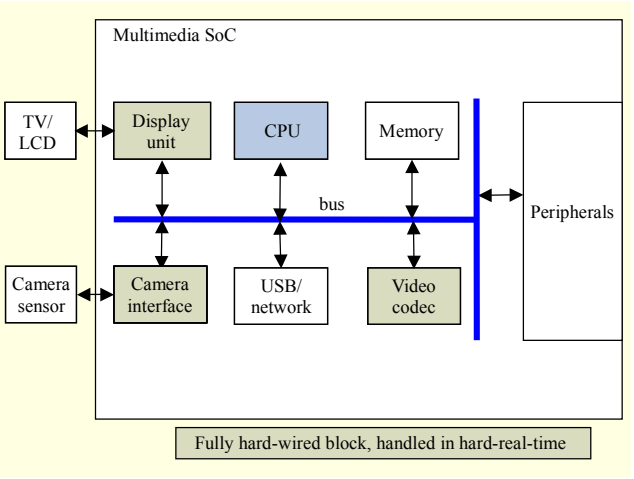


Fig. 2. Hardware platform of embedded multimedia SoC.

The location of the HA-RMS algorithm on the software platform is shown in Fig. 1. This algorithm is located in the threads configuration layer and runs when a process configures priorities of its own threads.

The HA-RMS algorithm is mainly designed for embedded multimedia applications which operate on embedded multimedia SoCs where various multimedia contents like MPEG video are run. This paper, therefore, focuses on multimedia SoCs which embed special hardware blocks to operate complex and repeated computations. However, any other hardware systems which are equipped with a special hardware block controlled by software would become good candidates for the hardware platform to adopt the HA-RMS algorithm.

Figure 2 shows a block diagram of an embedded multimedia SoC. Basic characteristics of tasks to be scheduled by the

Table 2. Simple task model applied to compare RMS with HA-RMS.

Task	Period	Priority by RMS	Execution time	Hardware use
A	Shortest	1	> 1 ms	No
B	Middle	2	< 100 µs	Yes
C	Longest	3	< 100 µs	No

HA-RMS algorithm are the same as those described in the original paper on RMS [11]. Tasks should be periodic with a fixed-priority, and the deadlines of tasks should be the same as the request interval of each task. In addition to the characteristics above, this paper provides a few task models to clearly show the advantage of HA-RMS over RMS:

Model 1 (M1). A software-task which consumes huge computation power with the shortest request intervals. One example is an audio decoder task.

Model 2 (M2). A hardware-task with a short job execution time and long hardware working period. An example is a task which controls the video decoder hardware.

Model 3 (M3). A software-task with a short job execution time and the longest request intervals. An example is a data read/write task.

According to the above classification, a simple task model is described in Table 2 and will be considered to analyze both RMS and HA-RMS algorithms in the following section.

IV. Hardware-Aware Rate Monotonic Scheduling

The RMS algorithm configures the priority of each task in the order of request latencies. The shorter request interval a task has, the higher priority it has. Figure 3 shows an example scenario of the RMS algorithm with three tasks which are modeled as examples of software-tasks and hardware tasks.

Task A has characteristics of the M1 with the shortest request latency. Task B has characteristics of the M2 and its request interval is longer than the task A and shorter than the task C. Dotted lines in the figure stand for durations of hardware working. Task C has characteristics of the M3 with the longest request interval.

In Fig. 3, task A, having the highest priority, executes as soon as the requests come. However, the responsiveness of task B is awful. First, due to lower priority, task B is preempted by task A which has a significantly long processing period. Furthermore, there is another important reason that causes the poor responsiveness of task B. The reason is that the period of hardware working (dotted-line in the figure) by the earlier request makes task B wait for the finish of the hardware job even

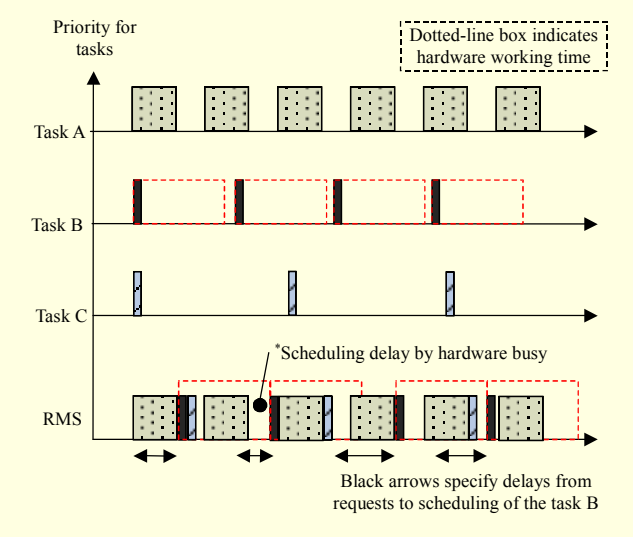


Fig. 3. Scheduling scenario of RMS algorithm.

though the processor is available for task B to run on CPU. We define this phenomenon as *hardware illusion* on RMS. This causes several problems for the original RMS algorithm [1], [2], [13], [14].

The processor utilization proposed in the original RMS paper [11] becomes worse. In a critical situation, scheduling could be not-feasible even though RMS tells its feasible. The response time of a hardware-task is not guaranteed in a certain time. This will make unexpected output jitters [9] happen on the system. The utilization of the hardware block becomes poor due to bad responsiveness of hardware-tasks which are preempted by software-tasks with long execution times.

The HA-RMS algorithm solves problems by exploiting characteristics of the hardware/software modules embedded in the multimedia SoC platform. Because hardware-tasks just set up a few hardware registers with neither complex computation nor heavy processing, they only occupy the processor resource for a very short time during each hardware request. This implies that, in terms of responsiveness, software-tasks are not significantly affected by hardware tasks which preempt software-tasks for the very short time. Furthermore, to increase utilization of the special hardware blocks, hardware-tasks which control the hardware blocks should have good responsiveness.

In the HA-RMS algorithm, to improve the responsiveness of hardware-tasks and to utilize hardware blocks efficiently, hardware-tasks with short job execution time are grouped and dealt specially with higher priorities than those of software-tasks. For example, they are grouped as scheduling classes similar to what is implemented in the Linux kernel [10]. The most recent Linux kernel not only supports two scheduling classes, RT and CFS, but also adds a new scheduling class if

```
//Step 1
for 'i' from '0' to 'the # of tasks –1' by '1' incremental
        if 'Task[i] is a hardware-task'
                register to the hardware-tasks group
        else
                register to the software-tasks group
//Step 2
Set_Priority_by_RMS(software-tasks group, 0)
//Step 3 to 5
temp = Get_Highest_Priority_in_Group(software-tasks group)
Set_Priority_by_RMS(hardware-tasks group, temp)
//functions
Set_Priority_by_RMS(group, the lowest priority)
        setting priorities of tasks in 'group' with 'the
        lowest priority' value

Get_Highest_Priority_in_Group(group)
        return the highest priority value among tasks in
        'group'
```

Fig. 4. Pseudo-code of HA-RMS algorithm.

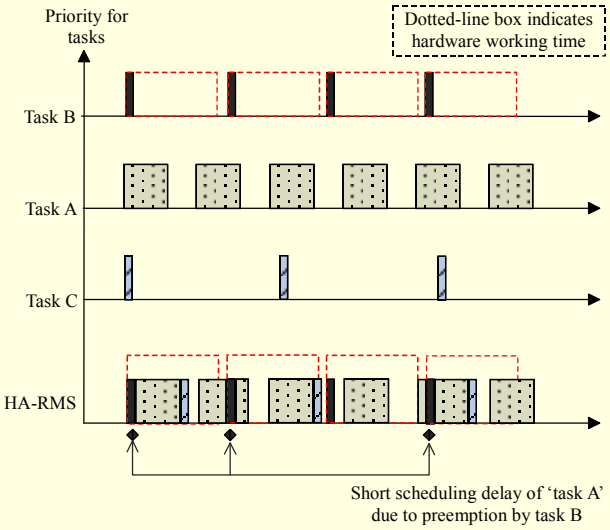


Fig. 5. Scheduling scenario of HA-RMS algorithm.

needed. In a similar manner, hardware-tasks and software-tasks can be classified as two different scheduling classes, where a class of higher preference can fully preempt another class. The proposed HA-RMS algorithm is described by the following five steps:

Step 1. Separate tasks into two groups: software-tasks and hardware-tasks.

Step 2. Set up priorities of software-tasks with the unmodified RMS algorithm.

Step 3. Make the order of priorities of hardware-tasks following the unmodified RMS algorithm.

Step 4. Configure the priority of the lowest-priority task in the hardware-tasks group to be higher than the highest-priority task in the software-tasks group.

Step 5. Configure priorities of hardware-tasks in the order decided on the step 3.

Figure 4 shows a pseudo-code for the implemented algorithm. Regardless of the request latency of each task, all tasks in the hardware tasks group have higher priorities than those of tasks in a software-tasks group. Members in a group are then ordered by request latencies of each task. Tasks in the software-tasks group could be preempted by tasks in the hardware-tasks group, hence scheduling of the software task preempted by hardware task would be deferred. However, any delay due to the preemption should not be significantly long because job execution time of the hardware-task is characterized by a short execution time by its definition. Figure 5 applies the HA-RMS algorithm to the same scheduling scenario to compare it with the RMS algorithm.

The time complexity of the proposed HA-RMS algorithm remains the same as the original RMS. However, the priority setting of HA-RMS is a bit different. Task B has higher priority than that of task A because task B is a hardware-task, even though task A has shorter request time. As a result, the responsiveness of task B is drastically increased. Moreover there is no scheduling delay in this example. In other words, it reduces the output jitter of hardware block, so that it increases the hardware utilization [15]. Even though there is a bit delay of task A preempted by task B, this overhead incurred by HA-RMS with preference on HW tasks over SW tasks, however, has little impact because job execution time of task B is very short.

V. Experimental Evaluation

An embedded media player application is employed to evaluate the proposed HA-RMS over the CL5500, a multimedia SoC developed by CoreLogic Co., Ltd. [16], which is used for mobile handset devices. The CL5500 SoC chipset is equipped with the ARM9 processor, which normally runs with a 133 MHz clock frequency (up to 540 MHz maximum frequency). Since the computation power of ARM CPU is not enough to run high-resolution video decoding and rendering, it embeds various special hardware blocks: video decoders (JPEG, MPEG2, MPEG4, H.264, and VC1), video renderer (Draw YUV422 image on display device), and other peripherals (I2C, DMA, GPIO, and others).

On the CL5500 chipset, the NEOS real-time operating system [17] is installed to provide multithread scheduling environments. The NEOS is fit for evaluating RMS and HA-RMS algorithms because it provides priority-based preemptive thread scheduling. Using the CL5500 hardware and NEOS operating system, an embedded media player is programmed

with five tasks (threads):

Task 1 (T1). The video decoder task controls the video decoder hardware block to decode a frame of MPEG4 video stream [18]. Request of decoding a single video frame arises every 33 ms.

Task 2 (T2). The video renderer task draws a frame of YUV422 image to LCD display device with controlling the video rendering hardware block. Request interval of rendering is the same as that of video decoding: 33 ms.

Task 3 (T3). The audio decoder task decodes one frame of AAC audio stream [19]. It is fully run by the CPU processing. Hence, it consumes the largest amount of computation power among all tasks. One frame of AAC audio stream with 48 kHz sample-rate contains an amount of 24 ms of PCM data.

Task 4 (T4). The audio render task pushes PCM data into a certain hardware buffer. It does not control any hardware block: it is a software-task. Request interval of rendering is the same as that of audio decoding: 24 ms.

Task 5 (T5). A roll of a data read task is reading video and audio streams, which need to be decoded, using the file system service. Reading every 40 ms is enough to feed data into video decoder task and audio decoder task on this media player.

With the traditional RMS, task priorities are determined by the order of request latency. T3 and T4, therefore, have the highest priorities. Strictly speaking, T3 and T4 should have the same priority, however, for convenience, T3 is set as the highest priority task and T4 as the second one. This modification does not distort the simulation result, but helps us to examine the job execution periods more explicitly. In the same manner, T1 has the third-highest priority and T2 the fourth. Finally, T5 will be configured as the lowest priority task.

When applying the proposed HA-RMS to the given multimedia application, a new task priority is assigned to each task. T1 and T2 belong to the hardware-task group because T1 and T2 control special hardware blocks, and the rest of tasks become members of the software-tasks group. T1 and T2 in the hardware-tasks group have the same request intervals, however, for convenience, T1 is set to have a higher priority than T2. Similarly, T3 is set as the highest one and T4 as the second one, T5 then becomes the lowest priority task in the software-tasks group. Finally, the lowest priority task T2 in the hardware group should have higher priority than that of the highest priority task T3 in the software-tasks group according to HA-RMS.

While processing the multimedia application up to 500 requests of T1, its average response time, defined by the average delay time between a request and its actual processing,

Table 3. Multimedia tasks evaluated in the experiment.

Task	Task type	Request interval	Execution time	Priority by HA-RMS
T1	HW	33 ms	50 μs	1
T2	HW	33 ms	50 μs	2
T3	SW	24 ms	13,000 μs	3
T4	SW	24 ms	30 μs	4
T5	SW	40 ms	1,000 μs	5

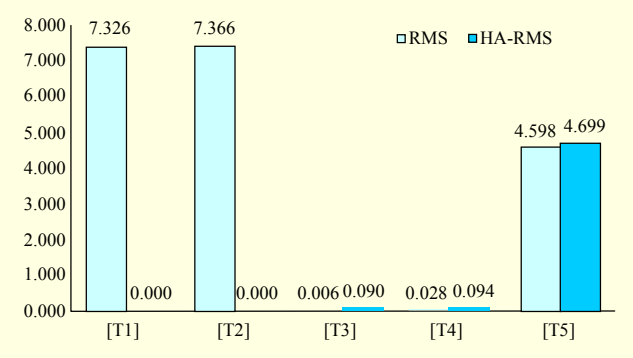


Fig. 6. Average response time.

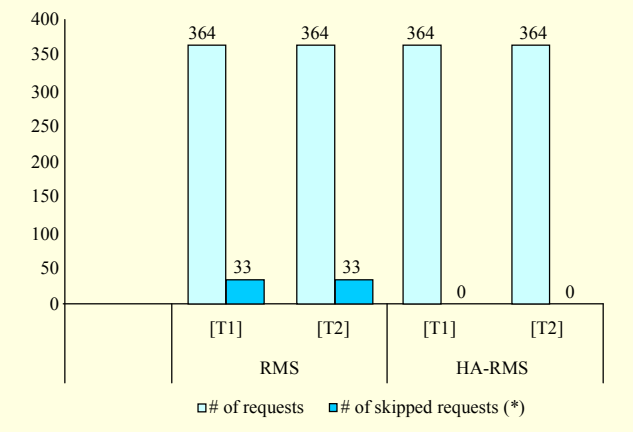


Fig. 7. Number of missed deadlines.

is measured as a performance metric of responsiveness for each task. In addition, whether some requests of the hardware-tasks are ignored or not is checked, and the number of occurrences is counted. In cases where the hardware is still processing the precedent request when a new request comes and the task is scheduled on CPU, the new request should be ignored and the job by the new request is skipped. This count stands for the number of happenings of hardware illusion on RMS, and represents that the system has output jitters and is operating with poor hardware utilization.

As shown in Fig. 6, when applying the HA-RMS algorithm,

the average response time of hardware tasks T1 and T2 is drastically down as an amount of over 7.300 ms compared with that of the RMS algorithm. Nevertheless, the results tell us that its impact on the responsiveness of software tasks T3, T4, and T5 are very small. T3 is delayed as 0.084 ms, 0.066 ms for T4, and 0.101 ms for T5. Thus, the above results prove that the HA-RMS algorithm greatly enhances responsiveness of hardware tasks with little impact on that of software tasks.

To evaluate a real-time QoS for the multimedia application, the number of deadline misses, called deadline miss rate, is measured as shown in Fig. 7. When applying RMS, there were 33 missed-request cases, that is, 33 hardware-illusion events for 364 requests because the precedent request was still processing at the moment that a new request arrived. However, these phenomena never happen when applying HA-RMS. It also shows that the proposed HA-RMS algorithm can provide much better hardware utilization than that of the RMS algorithm because the HA-RMS algorithm drastically reduces the hardware illusion problem.

VI. Conclusion

An embedded multimedia system usually adopts special hardware blocks to fulfill some multimedia data processing tasks to increase performance of the system and the quality of multimedia applications. Traditional RM scheduling has no way to support the employed hardware blocks and prioritize hardware tasks over software tasks, resulting in a poor utilization factor. Therefore, this paper presents a new hardware-aware RM scheduling of embedded multimedia tasks by differentiating hardware tasks to increase the quality of real-time audio/video applications and provide better hardware utilization. The experimental results show that the HA-RMS greatly enhances the responsiveness of hardware tasks with little impact on that of software tasks and reduces the output jitter drastically.

As future work, we plan to study the impact of HA-RMS along with the current trend of multi-core architectures. The proposed HA-RMS could be adapted to multi-core systems by exploiting its hardware awareness and grouping tasks of each core to increase utilization of special hardware blocks and reduce their response time.

Acknowledgments

We would like to thank Chuck Yoo at Korea University. He generously contributed his insightful feedback on early draft of this paper. We owe our special thanks to anonymous reviewers for their valuable comments and suggestions.

References

[1] M.V.P. Rao and K. Shet, "A Research in Real Time Scheduling Policy for Embedded System Domain," *CLEI Electron. J.*, vol. 12, no. 2, Aug. 2009.

[2] L. Sha et al., "Real Time Scheduling Theory: A Historical Perspective," *Real-Time Systems*, vol. 18 no. 2, 2004, pp. 46-61.

[3] G. Buttazzo, "Rate Monotonic vs. EDF: Judgement Day," *Real-Time Syst.*, vol. 29, no. 1, 2005 pp. 5-26.

[4] A. Silberschatz, P. Galvin, and G. Gagne, *Operating System Concepts*, 8th ed., NJ, USA: John Wiley and Sons, 2009.

[5] R. Steinmetz, "Analyzing the Multimedia Operating System," *IEEE Multimedia*, vol. 2, no. 1 Mar. 1995, pp. 68-84.

[6] B. Ahn et al., "A Real Time Scheduling Method for Embedded Multimedia Applications," *Proc. Int. Conf. Pervasive Syst. Computing*, June 2006, pp. 104-107.

[7] J. Nieh and M.S. Lam, "A SMART Scheduler for Multimedia Applications," *ACM Trans. Comput. Syst.*, vol. 21 no. 2, May 2003, pp. 117-163.

[8] W. Kim, J. Chang, and H. Cho, "Pipelined Scheduling of Functional HW/SW Modules for Platform-Based SoC Design," *ETRI J.*, vol. 27, no. 5, Oct. 2005, pp. 533-538.

[9] S. Baruah et al., "Scheduling Periodic Task Systems to Minimize Output Jitter," *Proc. 6th Int. Conf. Real-Time Computing Syst. Appl.*, Dec. 1999, pp. 62-69.

[10] R. Love, *Linux Kernel Development*, 2nd ed., Indiana, USA: Novell, 2005.

[11] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *J. ACM*, vol. 20, no. 1, Jan. 1973, pp. 46-61.

[12] Y. Manabe and S. Aoyagi, "A Feasibility Decision Algorithm for Rate Monotonic and Deadline Monotonic Scheduling," *Real-Time Syst.*, vol. 14, no. 2, Mar. 1998, pp. 171-181.

[13] E. Bini and G. Buttazzo, "Schedulability Analysis of Periodic Fixed Priority Systems," *IEEE Trans. Comput.*, vol. 53, no. 11, Nov. 2004, pp. 1462-1473.

[14] C. Yaashuwanth and R. Ramesh, "A New Scheduling Algorithm for Real Time Tasks," *Int. J. Comput. Sci. Inf. Security*, vol. 6, no. 2, 2009, pp. 61-66.

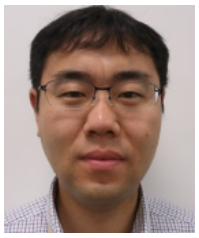
[15] F.M. Proctor and W.P. Shackledford, "Real-Time Operating System Timing Jitter and Its Impact on Motor Control," *Proc. Int. Conf. Sensors and Controls for Intelligent Manufacturing*, Oct. 2001, pp. 10-16.

[16] CoreLogic Co. Ltd. Available: http://www.corelogic.co.kr

[17] NEOS Real-Time Operating System. Available: http://www.mdstec.com/main/english/?no=254

[18] ISO/IEC Standard, "Information Technology: Coding of Audio-Visual Objects—part2: Visual," ISO/IEC 14496-2, Jun. 2004.

[19] ISO/IEC Standard, "Information Technology: Coding of Audio-Visual Objects-Part3: Audio," ISO/IEC 14496-3, Sept. 2009.



Jaebeom Park is a research engineer of the Department of Security Solution in Samsung Techwin Co., Ltd. and is currently developing system software for ISP and SoC applications. He received the BS from the POSTECH, Korea, in 2003, and the MS from the Georgia Institute of Technology, in 2009, respectively. He has a 7-year research career on embedded system software, device driver, and middleware on mobile multimedia systems. His research interests include real-time scheduling in RTOS on embedded SoC, balancing real-time tasks and user tasks in Linux, and building common device drivers for RTOS and Linux.

Joonhyuk Yoo is a faculty member of the Department of Computer and Communication Engineering at Daegu University, South Korea. He received his BS and MS in electronic and electrical engineering from the POSTECH, and MS and PhD degrees in computer engineering from the University of Maryland at College Park, USA. Prior to joining Daegu University, he was a research faculty member of the Department of Embedded Software at Korea University and was an embedded system engineer at Samsung Electronics Co., Ltd. His research interests are in embedded software, computer architecture, and cyber physical systems.