International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 18 (2017) pp. 7449-7454

nMPRA-MT Hardware Rate Monotonic Scheduling

Ayman AbuBaker

Electrical and Computer Egnieering Department, Applied Science Private University, Jordan.

Orcid: 0000-0002-4553-8037

Abstract

Real time systems execute tasks that vary in their criticality in the same hardware plateform. Many processor architectures have been proposed to deal with such real time envoironments in order to increase the effeciency of the execution of mixed critical tasks. One of these arcithectures is Multi Pipeline Register Architecture-Multithreading (nMPRA-MT) which is a fine grained multithreaded architecture. nMPRA-MT offers a hardware design that supported for scheduling. In this paper, rate monotonic scheduling algorithm is proposed to be supported by Hardware Scheduling Engine (nHSE) to increase the effieciency of executing taskes with different levels of criticallity. The proposed dynamic scheduling algorithm can effecently increase the utilization of the processor that affect on task's processing time.

Keywords: Dynamic Scheduling, nMPRA-MT, nHSE, pipline Architcuture

INTRODUCTION

The increase of the usage of the embedded systems in the recent years shows their importance in technological and socioeconomic fields. These systems are existed in real time applications such as automotive and operate under real time constraints.

Real Time Systems (RTS) are critical systems that are subject to real time constraints that are usually called deadlines. RTS can be classified depending on their ability to respond in a predetermined time (deadline) into two major systems [1]: Hard RTS in which the effect of missing the deadline causes critical situation and Soft RTS, where systems can miss some of the deadlines. In Soft RTS, the missing of the deadline of some tasks does not cause critical situations but missing too many deadlines will degrade the performance.

The deadline of the tasks in real time systems is very important as a classifying factor. The tasks in real time applications can be categorized into three classes depending on their effect on the deadline:

1. Hard: Hard real-time task if missing the deadline cause disastrous situation.
2. Soft: Missing the deadline of soft real-time tasks affects only the performance of the system.
3. Firm: missing the deadline will invalidate the results so they cannot be used in the system but they do not introduce damage [1][4].

In order to decide which of the available tasks to be executed at a specific time, scheduler needed to be used. Scheduler is responsible of starting, suspending and resuming a task at a particular time. One of the implementations of the scheduler is by embedding it as a service provided by the kernel of the operating system. Real-Time Operation System (RTOS) is a popular OS for real time systems. It provides easy and simple solutions to design real-time applications but it originates overhead that might prevent some real-time systems from working efficiently. RTOS introduces overhead that may lead to missing some deadlines. In addition, the RTOS response for interrupts is unpredictable. Some problems have been identified in RTOS; one of them is a generated from interrupt service routines and as result of this problem, it is hard to calculate the Worst Case Execution Time (WCET). The miscalculation of WCET leads to missing deadlines which depends on the calculations of WCET [2].

One of the ways to solve the problems that might be faced in RTOS scheduler is to migrate scheduling and interrupt handling to hardware [13].

To increase the utilization of the processor a multithreaded pipelined special designed architecture - Multi Pipeline Register Architecture (MBRA) for real time systems is used. Speeding up the context switches and improving the scheduler time are the main advantages of this architecture. The hardware implementation of the scheduler gives almost two orders of magnitude of software scheduler [3].

In (MPRA) each task has its own pipeline registers and context switching from one task to another will not lead to an instruction discard. This property of MPRA results high throughput of the pipeline. In addition, MPRA has a Hardware Scheduling Engine (HSE) which is used to implement scheduling algorithms.

Scheduling algorithms allow real time system to execute different tasks with different priorities. The using of fine grained multithreading will preserve the spatial isolation [1].

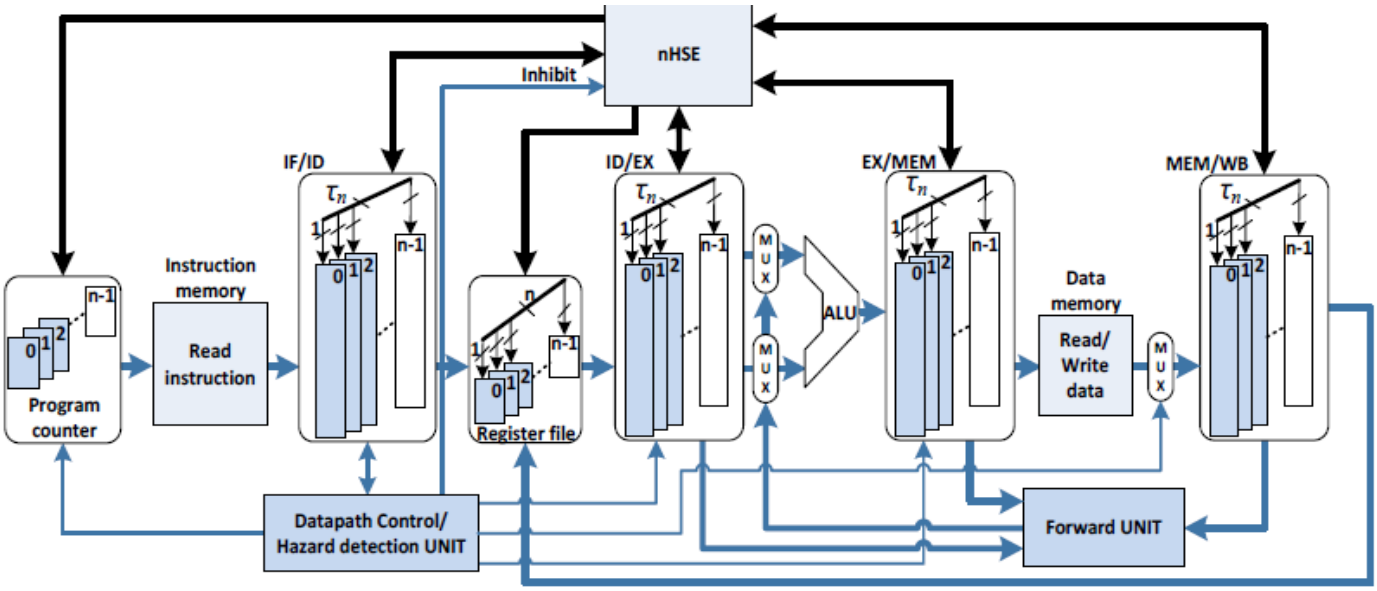

Figure 1: nMPRA-MT Architecture.

In this paper, the idea of using the HSE existed in the MPRA were investigated as in ref [5] to test the performance of the pipeline multithreaded processor. The hardware scheduler will deal with hard real time tasks and software real-time tasks by applying the Rate Monotonic Scheduler (RMS) in HSE of MPRA to increase the utilization of the pipeline of multithreaded processor. A multithreaded processor use the hardware support of the scheduler to allow multiple threads to share the pipeline taking into account preserving the spatial and temporal isolation.

This paper is organized as follows: section 2 presents the architecture of the multithreaded processor nMPRA-MT. In section 3, the fine grained multithreading background is presented. Scheduling is presented in section 4. The proposed dynamic scheduler is explained in section 5. Finally, in section 6, conclusion and future work are presented.

nMPRA-MT ARCHITECTURE

Multi Pipeline Register Architecture-Multithreading (nMPRA-MT) architecture consists of 5 stages pipeline Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory (MEM) and Register Write Back (WR). The architecture supports multi pipeline register in a way that each task has its own program counter (PC) which contains the address of the instruction fetched from memory of the task. Register file is also connected with the task and is built to allow very fast context switch. The instructions of MPRA are divided into the following groups: arithmetic, logic, conditional jump instructions, data transfer, unconditional jump instructions and special instructions [2].

The nMPRA-MT is a fine-grained multithreaded processor designed specifically for real time systems. The system is consists of 5 stages pipeline which allows 4 different instructions from different threads to be executed simultaneously.

The basic architecture of the nMPRA processor contains a hardware unit for scheduling that can be implemented to support static and dynamic scheduling of the tasks as shown in Figure 1. It allows fast context switching of register file through stack-free architecture. The stack has removed so the register to memory overhead is eliminated.

The MPRA architecture provides fine grained multithreading processor with temporal and spatial isolation between tasks.

HSE is heart of the architecture and it is part of the CPU and directly controlled with dedicated instructions. The function of this unit is to specify the address of the next task to be executed [2][3].

FINE-GRAINED MULTITHREADING

The main goal for find grained multithreading is allowing tasks from different hardware threads to interleave execution in the pipeline in order to increase the throughput and minimize the number of stalls. It involves fetching different instructions from different hardware threads in each clock cycle. Each thread is able to maintain its program counter and registers. The decision for which next instruction thread to be fetched is specified by the scheduler [1].

In this paper, fine grained multithreading processors was utilized to increase the processing time. A multithreaded processor shares the pipeline between different hardware threads using the hardware support. The idea is to reach better utilization of resources by allowing multiple tasks to execute in the processor, one task from each thread. Moreover, the

thread scheduler must take into the consideration the temporal isolation of threads.

Spatial and temporal isolation are important to preserve the independent tasks from being affected by another tasks. Temporal isolation related to the timing behavior of the task while spatial isolation protects the state of the task. WCET plays the basic role in timing predictability of tasks especially when the task is hard real-time task [5].

XMOS requires four threads or more to be active to fully utilize the processor. So, if the number of the active tasks is varying then the temporal isolation is reduced [6]. Some fine grained processors interleave from different hardware threads in each cycle [7][8]. Also, PTRAM uses round robin scheduling between tasks to utilize the pipeline which consists of 5 stages but it needs four active threads to be active to guarantee isolation and to not lose cycles [9].

FlexPRET uses the classification of hard real time task and soft real time tasks to schedule the tasks from different threads. In each cycle one thread is going to scheduled either HRTT or SRTT. When there is no scheduled thread for the current cycle, the cycle is used by any available SRTT in a round robin technique [5].

By classifying each thread as either a hard real-time thread (HRTT) or a soft real-time thread (SRTT), FlexPRET provides hardware-based isolation to HRTTs while allowing SRTTs to efficiently utilize the processor. Each thread, either an HRTT or SRTT, can be guaranteed to be scheduled at certain clock cycles for isolation or throughput guarantees. If no thread is scheduled for a cycle or a scheduled thread has completed its task, that cycle is used by some SRTT in a round-robin fashion—efficiently utilizing the processor.

An example of pipeline of a multithreaded processor is listed in Table 1, where we have instructions interleaved from two threads in the pipeline.

Table 1: Fine grained multithreading example.

TID	ADD	INS	Cycle								
			1	2	3	4	5	6	7	8	9
0	0x00	BR	F	D	E	M	W				
1	0x30	LD		F	D	E	M	W			
0	0x04	ADD			F	D	E	M	W		
1	0x34	ADD				F	D	E	M	W	
0	0x0c	I					F	D	E	M	W

SCHEDULING

nHSE is a unit that offers static and dynamic scheduling. The output of the nHSE is an activation signal of the task to be performed as shown in Figure 2.

The rule of nHSE is that the interrupts of a higher importance task are allowed to interrupt the current executed task. The interrupts assigned to a lower priority task will postpone until the higher priority tasks are executed. In Figure 3, it was noticed that Service Request Register (SRR) specify which task is higher priority than the others. The Interrupts Source Task (IST) and the Hardware Activated Task (HAT) are ordered based on the descending order of the SRR.

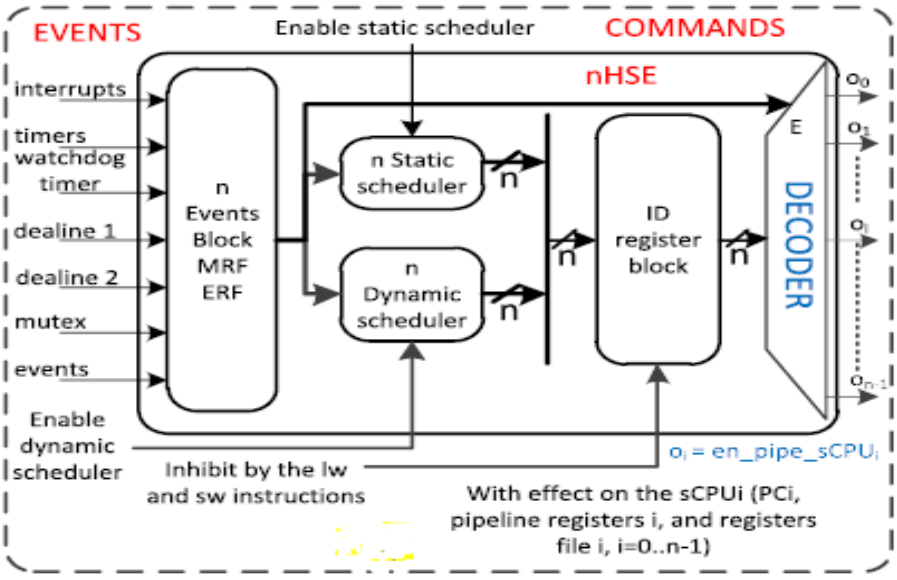

Figure 2: nHSE Structure

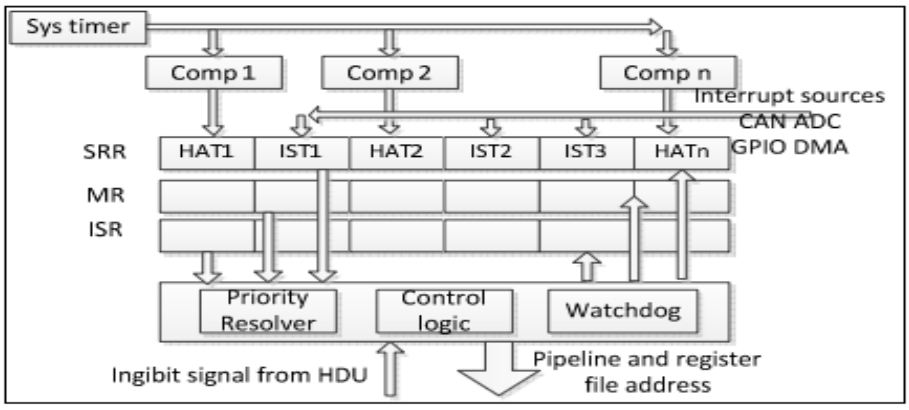

Figure 3: HSE Organization

In nHSE there is a register called special state register that has the state of each thread scheduled by nHSE. IDLE, SLEEP and RUN state for each thread. In addition ID priority register assigned for each thread. Each thread has its own state and ID registers where thread with ID equal 0 το has the highest priority and the thread with ID n-1 id the lowest thread priority. The blue threads in the Figure1 are in RUN state while the blank ones are in IDLE or SLEEP state [2].

PROPOSED HARDWARE RATE MONOTONIC SCHEDULER

The scheduling algorithm in nHSE is kept simple and left to the user to organize and arranged the tasks in the SRR. Many systems built the scheduling on the priorities [10]. In this paper a new dynamic scheduling is proposed. The implementation of the proposed dynamic scheduling is based on priorities for nHSE of nMPRA architecture [12]. Moreover, Monotonic Rate algorithm is used in this novel approach.

The hardware scheduler will have set of threads that are represented by the thread class. The scheduler has set of priority queues and the class of scheduling agent will be implemented in any scheduling algorithm. In our case it will be Rate Monotonic Scheduling (RMS).

Rate Monotonic Scheduling

Rate Monotonic Scheduling (RMS) is useful because if the response time is enough then it guarantees that the threads will finish execution without missing the deadlines [11].

The RMS scheduling works by assigning each thread a priority based on its interval. The thread with the smallest interval gets the highest priority and the thread with the longest interval gets the lowest priority. The threads are then run similar to a prioritized preempting. This means, any task that can run runs, and if a task runs but a task with a higher priority is available, the higher one runs instead.

A scheduler that is aware of rate monotonic scheduling would be provided with process timing parameters (period of execution) when the process is created and compute a suitable priority for the process as shown in Figure 4 [5].

Thread	Capacity	Period
T1	1	4
T2	2	5
T3	2	10

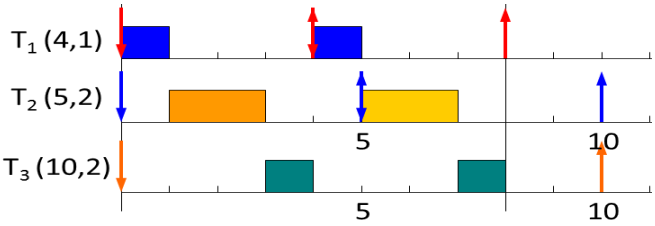

Figure 4: Proposed Hardware Rate Monotonic Scheduler

Scheduler Structure

In this work, the threads are going to be prioritized according to their interval in priority queue. The priority queue is divided into two sub queues. The first one is holding the HRTT and the second one is keeping the SRTT. Both queues are prioritized according to RMS. When a SRTT is required we take the thread in the top of the SRTT queue and when a HRTT thread is to be started running we take the one in the top of the HRTT queue.

The scheduling algorithm depends on classifying the threads into HRTT or SRTT. The architecture of the scheduler is shown in Figure 5.

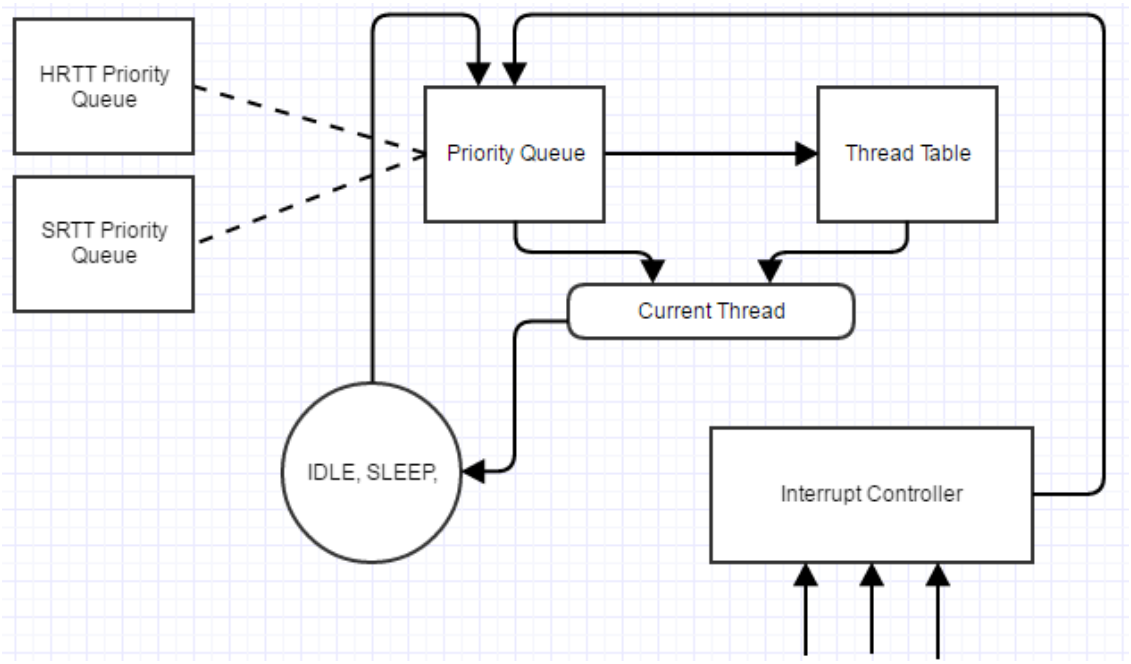

Figure 5: Architecture of the scheduler.

The main components of the scheduler are:

- Priority Queue: This queue consists of two related Queues HRTT which consists of threads that are hard real time and SRTT for soft real time threads.
- The Thread Table
- The interrupt controller.

Priority Queue

The priority ques is a sorted list that contains the active threads. The queue entry that is shown below, the C field contains the capacity and P is the period in the case of rate monotonic scheduler. The list inside the PQ is sorted by the period. The shortest comes first.

ID	C	P

The hardware realization consists of an array of similar registers and a number of comparators which ensure that elements with a large priority value are moved backwards in the queue while small elements are kept in the front as shown in Figure 6.

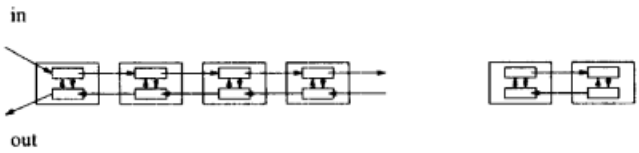

Figure 6: Priority Queue

Thread Table

It is a lockup table indexed by the ID. Each entry of the table has the following fields: PRI, PERIOD, WCET, TYPE, and STATUS. PRI holds the priority of the thread's PERIOD and WCET. While the TYPE is an indicator of the thread's type which either be HRTT or SRTT type. STATUS will be one of three cases which are SLEEP, IDLE and RUN. Every time the scheduler fetch a thread the information is brought from the thread table which shown in Table 2.

Table 2: Thread Table

PRI	PERIOD	WCET	TYPE	STATUS

Interrupt Controller

The main rule of nHSE is that the interrupts of a higher importance task are allowed to interrupt the current executed task. The interrupts assigned to a lower priority task will postpone until the higher priority tasks are executed. So, the assigned priority of each task will be according to its period where the shorter period has a higher priority in processing.

Reflection RMS on the Structure

The rate monotonic algorithm is going to be applied on both hardware real time tasks and software real time tasks. nMPRA processor schedules HRTT as a first in first out algorithm while it uses round robin technique for SRTT. In algorithm 1, RMS scheduling process were used.

Algorithm 1 : Scheduler

```
1: for each available task t
2:     Pri = ComputeRMS(t)
3:     if task is hardware task
4:         Type=HRTT
5:         HRTT_PQ.enqueue (t)
6:     else
7:         Type=SRTT
8:         SRTT_PQ.enqueue(t)
9:     end if
10:    Add record to thread table (ThreadID,Pri,Period,WCET,Type,Status)
```

Algorithm 1 fills the thread table and the priority queues of hard real time tasks and the soft real time task. If the processor needs new instruction it will get the next instruction from the highest priority thread in the HRTT queue and if it needs instruction from the soft threads it will get it from the SRTT queue and according to the highest RMS priority.

CONCLUSION AND FUTURE WORK

This paper presents a hardware implementation of one of the most popular real time scheduling algorithms which is Rate Monotonic Scheduling algorithm. The implementation is reflected on a fine-grained multithreaded processor nMPRA-MT. The algorithm takes into account not just the hard real time threads but also the soft real time threads to improve the overall performance. The proposed algorithm show a good performance in increasing the task's processing time.

The same procedure can be applied to different scheduling techniques such as Earliest Deadline First (EDF). Another enhancement could be modifying the scheduling unit in nMPRA in a way that it offers different scheduling techniques depending on the application.

ACKNOWLEDGMENT

The author is grateful to the Applied Science Private University, Amman, Jordan, for the full financial support.

REFERENCES

[1] Gaitan, N. C., Zagan, I., & Gaitan, V. G. (2015). Predictable CPU Architecture Designed for Small Real-Time Application-Concept and Theory of Operation. International Journal of Advanced Computer Science and Applications, 6 (4), 47-52.

[2] Gaitan, V. G., Gaitan, N. C., & Ungurean, I. (2015). CPU architecture based on a hardware scheduler and independent pipeline registers. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 23(9), 1661-1674.

[3] Dodiu, E., & Gaitan, V. G. (2012, May). Custom designed CPU architecture based on a hardware scheduler and independent pipeline registers—Concept and theory of operation. In Electro/Information Technology (EIT), 2012 IEEE International Conference on (pp. 1-5). IEEE.

[4] Zimmer, M., Broman, D., Shaver, C., & Lee, E. A. (2014, April). FlexPRET: A processor platform for mixed-criticality systems. In 2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS) (pp. 101-110). IEEE.

[5] Kuacharoen, P., Shalan, M., & Mooney, V. J. (2003, June). A Configurable Hardware Scheduler for Real-Time Systems. In Engineering of Reconfigurable Systems and Algorithms (pp. 95-101).

[6] May, D., & Muller, H. (2009). XMOS XS1 Instruction Set Architecture. XMOS XS, CPI Antony Rowe, Chippenham.

[7] Thornton, J. E. (1970). Design of a computer—the control data 6600.

[8] Kongetira, P., Aingaran, K., & Olukotun, K. (2005). Niagara: A 32-way multithreaded sparc processor. IEEE micro, 25(2), 21-29.

[9] Liu, I., Reineke, J., Broman, D., Zimmer, M., & Lee, E. A. (2012, September). A PRET microarchitecture implementation with repeatable timing and competitive performance. In Computer Design (ICCD), 2012 IEEE 30th International Conference on (pp. 87-93). IEEE.

[10] Moon, S. W., Rexford, J., & Shin, K. G. (2000). Scalable hardware priority queue architectures for high-speed packet switches. IEEE Transactions on Computers, 49(11), 1215-1227.

[11] Park, J., & Yoo, J. (2010). Hardware-aware rate monotonic scheduling algorithm for embedded multimedia systems. ETRI journal, 32(5), 657-664.

[12] Jamal, H., & Khan, Z. A. (2008). Hardware IP for scheduling of periodic tasks in multiprocessor systems. WSEAS Trans. Comput. Res, 3(3), 131-134.

[13] Daleby, A., and K. Ingström. "Technical Reference Manual for RTU Operating System Accelerator." Västerås, Sweden (2002).