Experimental Effectiveness Analysis of EDF-eCos for Real-Time computing in Small Unmanned Helicopter OFP

Nodir Kodirov*, Doo-Hyun Kim†, Jun-Yeong Kim*, Chang-Joo Moon‡

*School of Computer Science and Engineering
Konkuk University, Seoul, Korea
e-mail: {knodir, plaa}@konkuk.ac.kr

†School of Internet and Multimedia (corresponding author)
Konkuk University, Seoul, Korea
e-mail: doohyun@konkuk.ac.kr

‡School of Aerospace Engineering
Konkuk University, Seoul, Korea
e-mail: cjmoon@konkuk.ac.kr

Abstract

In this paper we mainly address task scheduling in embedded systems, which has a direct relationship with the system efficiency and an indirect relationship with its execution stability. Timeliness execution of application will guarantee system stability. Our practical application is running on Embedded Configurable OS (eCos) Real-Time OS with x86-architecture based embedded board. Need for and suitability of the Earliest Deadline First (EDF) scheduling algorithm for our practical application will be illustrated. EDF implementation approach will be explained fully, accompanied with source code skeletons. Enhanced kernel performance based on less re-scheduling needed, higher CPU utilization allowance and finer timeliness qualities will be demonstrated. Lastly, applicability of our implementation and results for embedded application carrying Real-Time computing of Small Unmanned Helicopter's Operational Flight Program will be illustrated.

1 Introduction

Our motivation to do this research got an impetus from two fronts. The first is suitable deadline based EDF characteristics with periodic nature of the OFP threads, which enables us to have higher CPU utilization allowance. The second motive promising EDF features, to have an improved performance and timeliness. Basis for the second motivation came from the theoretical framework congested during the last four decades on real-time scheduling algorithms, especially those researches targeted to evaluate trade-offs between dynamic and static scheduling algorithms [3-5]. Via precise deadline keep feature of the EDF, we will indirectly guarantee timeliness execution of the OFP.

Operational Flight Program (OFP) is our practical application written on C programming language. It is a flight control program of the small UAV, which is utilized at the disaster response and recovery phase of a disaster management system [6]. UAV is supposed to fly to the remote and/or hazardous places to take a moving and still pictures of the "hot-spot" location, together with capability of having up to 20 kg payload (which could be the first aid box).

Research done by Giorgio C. Buttazzo in 2005 provides clear picture of the EDF and Rate Monotonic (RM) implementation and performance evaluation from several aspects [4]. Although implementation approach may differ for a various Real-Time Operating Systems (RTOS), but in general, EDF is expected to have a higher CPU utilization and efficient scheduling. Based on these assumptions, we have implemented EDF scheduling in the eCos kernel and initially tested it with OFP motivated prototype application. Prototype application consists of concurrently running producer-consumer threads, where target is to keep execution deadline. Next, we run our practical OFP application with EDF kernel and had an enhanced kernel performance, which will be illustrated on respective diagrams.

Rest of the paper is organized as follows. We will start by explaining computational characteristics of UAV OFP, where our implementation is targeted to be utilized. Next, we provide problem definition where a new approach is needed to enhance system performance and timeliness. Before explaining our approach in detail, we will explore eCos kernel and its schedulers, which will serve as a base for all other coming sections. Next, we will explain our eCos-EDF implementation in detail, providing source code skeletons for critical points. Also, we will introduce scheduler feasibility support we have integrated into the

ISBN 978-979-16955-1-0

kernel space. In the penultimate section we will demonstrate performance improvements introduced by the EDF, first in the case of prototype application, followed by for practical eCos-EDF-OFP application. Next, we will provide practical results, where OFP will run on top of enhanced kernel. Lastly, we will conclude our work, sharing future plans.

2 OFP characteristics and software design

In this section, we discuss about functional and non-functional requirement of the OFP for small-scale rotorcraft. We also illustrate design of the OFP core modules in respective UML diagrams. In the first half, we illustrate Sensor and Controller modules, and UML Collaboration Diagram of OFP threads. Next, we will depict UML State Chart Diagram to get more insight view on OFP execution scenario.

2.1 Sensor and controller modules

UAV system consists of several external devices, such as sensor, actuator and wireless communication module. The OFP should accept every sensor data and calculate control signal. UAV system has AHRS and GPS for control and guidance. Wireless communication device is used interact with GCS. Also, OFP accepts control signal feedback to make evaluation and analysis of controller behavior. Figure 1 illustrates UML Collaboration Diagram of OFP core modules. There are four reader modules to accept input data from sensors (`Current_AHRS_Reader`, `Current_GPS_Reader`, `Current_Ctrl_Reader`, `Current_Asyn_Reader`). Each reader module accepts data from serial communication port and stores it into the `Glob_AHRS_Sens`, `Glob_Ctrl_Pack`, and `Packet Adt` shared storage modules. `Do_Guid_Ctrl` reads shared storage data and calculates proper control signal. `Do_Guid_Ctrl` module contains main core logic code for guidance and control of UAV, such as autopilot, forwarding, way pointing and etc. Four reader modules and one controller module compete for an access to the shared storage. In other words, the shared storage module access should be done as critical section.

2.2 OFP periodical execution

Reader modules and `Do_Guid_Ctrl` module are designed based on multi tasking. As external devices send messages at pre-defined frequency, `Do_Guid_Ctrl` module also should execute with an exact periodicity. To keep these qualities OFP must have an accurate timeliness. In order to keep stability of flight system, periodicity of the `Do_Guid_Ctrl` module must be guaranteed. If not, OFP control signal will not be generated on time, and aircraft may lose stability. This is the reason why aircraft control software should be designed on real time system.

The first thread interacting with Helicopter is Control logic thread, marked as the `Do_Guid_Ctrl` (Guidance and Control thread) in the Figure 1. It is OFP's core thread running on 50Hz frequency. Its task is to calculate control signal based on input sensor data and control algorithm, and send them to the UAV `Ctrl` (Control) module. The second one is `Do_Ser_Monitor` (Serial monitor thread) for polling asynchronous I/O ports. The other four threads are to read sensor data and store it into the global storage.

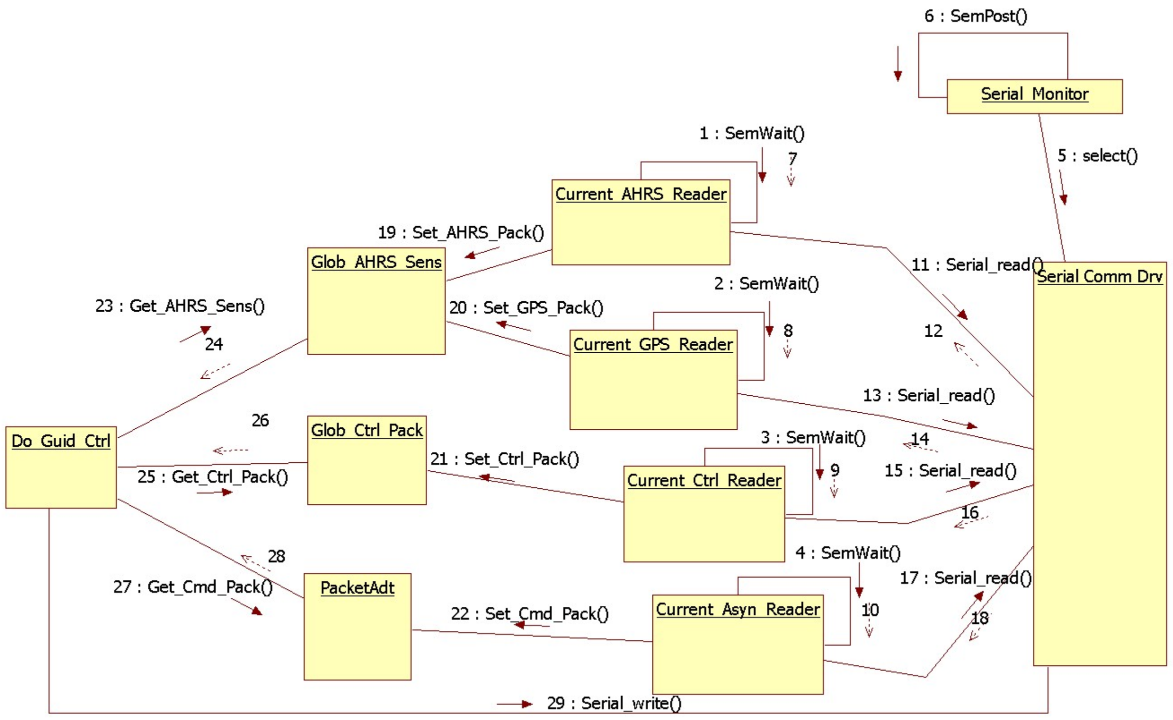

Figure 1: UML Collaboration Diagram of OFP core modules

ISBN 978-979-16955-1-0

These four reader threads manage Control Signal, Navigation, GPS, and Command message tasks. The `Current_Ctrl_Reader` (Control reader) thread receives control signal, which is fed back from the OFP. `Current_AHRS_Reader` thread receives AHRS sensor data sent at 100Hz frequency. Received data include current angle (roll, pitch, yaw), three-axis acceleration and angular velocity. Reading data from `Sens_AHRS_Pack`, `Current_AHRS_Reader` analyzes and stores it into the Global data (`Glob_AHRS_Sens`). The `Current_GPS_Reader` (GPS reader) thread's role is to receive GPS sensor data sent on 10Hz frequency and save it into the Global data. `Current_GPS_Reader` received data includes location values (longitudinal, latitudinal and altitudinal), and velocity of three-axis. Finally, `Current_Asyn_Reader` (Asynchronous Data Reader thread) receives command data including waypoint, operational mode and UAV information, which are sent from Ground Control System (GCS). All received data by `Current_Asyn_Reader` are saved into the Global storage. Global data storage content will be used to calculate control signal for proper navigation and stability of the UAV.

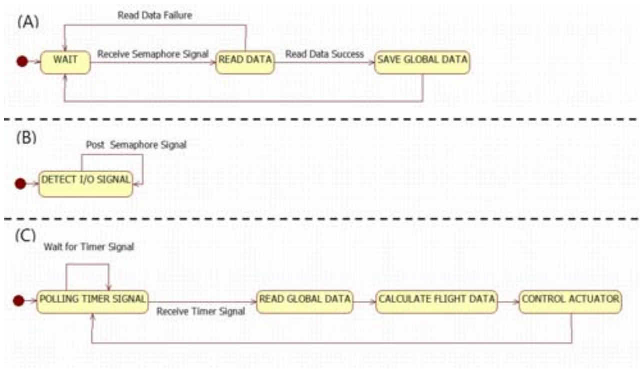

Figure 2: OFP thread UML State Chart Diagram

Figure 2 (A) illustrates execution scenario of the four reader (`Current_AHRS_Reader`, `Current_GPS_Reader`, `Current_Ctrl_Reader`, `Current_Asyn_Reader`) threads. When reader task receives signal from `Serial_Monitor` module, it starts reading data from I/O port and saves packets into the shared buffer. Figure 2 (B) shows state change scenario of `Serial_Monitor` module. It keeps polling serial communication I/O ports and signal reader modules once data is arrived. Figure 2 (C) is for `Do_Guid_Ctrl` module. Firstly, `Do_Guid_Ctrl` waits signal from timer, and once signal is arrived, it starts reading shared buffer and calculate control signal. When operation is completed, `Do_Guid_Ctrl` waits for a signal again.

3 Problem definition

eCos (Embedded Configurable OS) in Open Source RTOS available under GNU General Public License [8]. It is mainly known for its high configurability and low memory footprint in target image file [2]. In our development we use the latest stable eCos 3.0 releases, which includes two completed and one incomplete thread scheduler. Schedulers with full support are so called Bitmap and Multi-Level Queue, and there is one more ongoing Lottery scheduler implementation by eCos developers [8].

OFP can successfully run on top of the abovementioned scheduling algorithms. But, OFP running on top of the abovementioned algorithms can have less efficient performance, which results to the high CPU consumption. In order to have efficient CPU utilization, we opted to have more efficient kernel to lessen CPU consumption; thus having more CPU time for an additional computation to meet an exact timeliness requirements. Also, as it was shown in the OFP computational characteristics, similarity between EDF scheduling properties and OFP's periodically executing threads, could allow us optimally schedule OFP threads. Based on these assumptions we expect EDF to perform better than its alternatives such as RM (Rate Monotonic) scheduler to provide improved OFP performance and timeliness, given periodical nature of the OFP and respective EDF scheduling properties.

4 EDF Scheduling and Implementation Approach

EDF scheduling is well-known and one of the widely academically researched real-time scheduling algorithm. It is believed that EDF has the highest CPU utilization (up to 100%) among its alternatives [1, 9]. Main policy of EDF is that threads are dynamically assigned a priority depending on their remained time units before their deadline. Thread with the earliest deadline is supposed to be the highest priority and take a CPU ownership in the next scheduling point.

The EDF requires each thread to have its deadline value represented in RTOS specific time units. In the case of eCos, it is clock tick generated by the hardware (HW) clock interrupt. With Hardware Abstraction Layer (HAL) eCos provides 100 Hz tick frequency for all platforms. Application programmer is able to have a custom clock frequency via system supplied resolution set mechanism. Basically, eCos clock tick count is used as a primary synchronization unit for the thread alarms, delays and suspends. In order to keep easy interoperability with other kernel primitives, we designed EDF fundamental data structure to be in target specific clock ticks:

```
struct cyg_sched_edf_info_t {
  cyg_tick_count deadline_tick_cnt;
  cyg_tick_count wcet_tick_cnt;
  cyg_tick_count period_tick_cnt;
}
```

This C structure with three properties was added as a kernel data structure, where `cyg_tick_count` is wrapper type for `long int` in C language. We added the same structure

ISBN 978-979-16955-1-0

to the thread control block (TCB) of eCos, while each thread will have its own deadline, WCET (worst case execution time) and period as it is required by the EDF.

Application programmer is supposed to provide accurate values for the thread's {`deadline`, `WCET`, `period`} triple in milliseconds during its creation time. All these three are brought into kernel's `cyg_thread_create()` method and put into the TCB as a thread's property. In eCos, all of the threads are in the *suspended* state once they created. Thus, application programmer needs to explicitly call `cyg_thread_resume()` function to make them ready. Based on this *split mechanism*, we made additional `edf_prep()` function for the application programmer make a call between creation and resuming. This call-in-the-middle is the very right place to execute two major objectives. The first is to interpret user provided millisecond values into the platform specific clock tick units. Thus, we will achieve platform-independent timing mechanism.

In computation systems, especially Real-time embedded computing systems, schedule feasibility is considered to be one of the major research topics [12-15]. There is no general approach to make feasibility test for all systems, for example it will be different for fixed-priority and dynamic-priority scheduling environment [11]. We have done simple mechanism to make schedule feasibility test. As we described in the previous section, user is responsible to supply thread specific information during thread creation. `wcet_tick_cnt` and `period_tick_cnt` are the last two properties provided by the application programmer and we use them to make schedulability decision. If application fails to pass the test, kernel rejects it from running after prompting with an appropriate message about non-feasible schedule.

5 EDF-eCos performance improvements

As we mentioned above, EDF scheduler is expected to be more efficient, thus resulting to better kernel characteristics in several points. First, we have tested eCos-EDF kernel with our prototype application, which has shared computational characteristics with OFP. Having successful performance on prototype, then we tested our implementation for real UAV control application – OFP. Both of the experimental results will be illustrated in subsequent sections of this chapter.

5.1 Experimental results on prototype application

As it was mentioned in the Chapter 3, we were expecting EDF to perform better in two points. They are less re-scheduling need and preemption introduced by an application. When we tested eCos with the EDF and MLQ scheduler, we had similar properties when number of threads is less and CPU utilization is low as it is illustrated in the Figure 3.

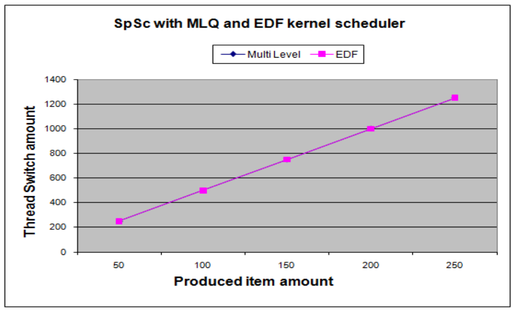

Figure 3: EDF and MLQ in Single producer and Single consumer

Figure 3 illustrates EDF and MLQ kernel performance with Single producer and Single consumer (SpSc) application. For all tests (if not explicitly stated otherwise) X axis represents the number of items produced by each producer thread. In the Y, we count number of thread switches introduced by an application. In this figure overlapped lines mean SpSc had the same performance in both kernels.

As we increase the number of threads (having higher CPU utilization), we start to notice the difference between two schedulers as it is depicted in the Figure 4. In this test we have five producers and one consumer thread. Each of producer thread's execution `period` is equal to 250 Millisecond (ms) and worst case execution time (`WCET`) is 30 ms. For consumer threads period is equal to 200 ms and WCET 20 ms.

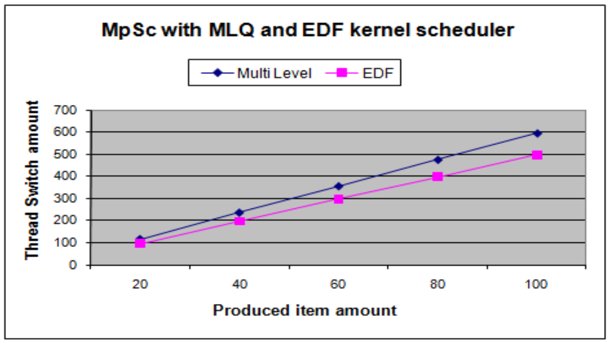

Figure 4: EDF and MLQ in Multiple producers and Single consumer

In this test application all of the producer and consumer threads meet their deadlines. We made this conclusion based on produced and consumed item values. If producer failed to produce specified item value or consumer didn't consume all of the items correctly, it shows deadline miss of some of the system threads. However, we didn't have such evidence during runtime. It confirms EDF kernel to provide fine timeliness for prototype application, made of several scenarios of producers and consumers.

As we continue to increase the number producer and consumer threads, we will have far different performance in two kernel schedulers.

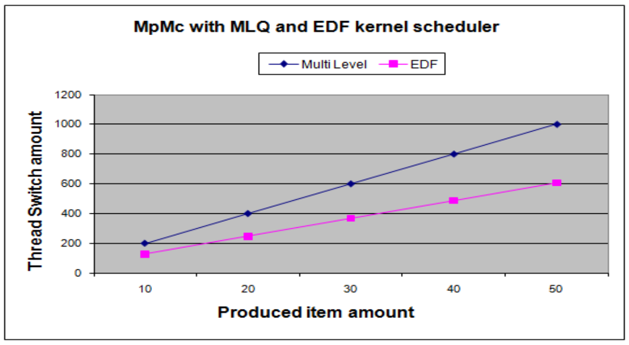

Figure 5: EDF and MLQ in Multiple producer and Multiple consumers

As it can be seen from the Figure 5, when we run five producers and consumer threads concurrently, we will have almost doubly improved kernel performance in terms of thread switch amount when produced and consumed item amount is equal to 50.

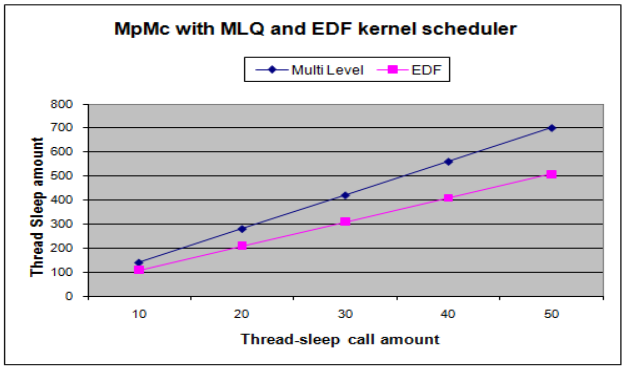

Figure 6: Thread-sleep amount on EDF and MLQ with Multiple producers and Multiple consumers

The less preemption in kernel means the less thread sleep and wake calls also. Figure 6 illustrates the amount of thread sleep calls for the same application as in the Figure 5. Increase in the number of threads introduces more preemption in MLQ scheduler. But, as EDF keeps execution of the current thread until it has the shortest deadline value, it causes to the less preemption, hence resulting to the less thread sleep calls.

5.2 Experimental results for OFP

In order to run OFP with the eCos-EDF kernel, we did minor modifications to the application code. Modifications were done only in two points. The first one was to supply appropriate {`deadline`, `WCET`, `period`} values for the each thread properties and other was to call `edf_prep()` function between thread creation and its resume (as it was mentioned in the Chapter 4).

For real OFP we made slight modification to our EDF implementation, calling it as a Hybrid-EDF. In Hybrid-EDF application programmer can create threads with EDF properties or without EDF properties, where latest one will be executed based on CPU availability. As OFP has several concurrent threads with specific deadline and without, we run our OFP on top of Hybrid-EDF kernel. OFP with Hybrid EDF had slight improvements as it was expected (Figure 7). This time in horizontal axis represents OFP running time in minutes (not amount of items produced by the producer).

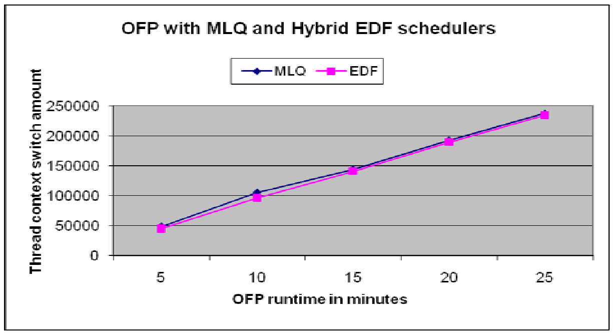

Figure 7: OFP with MLQ and Hybrid-EDF

We measured an amount of the thread context switches for each five minutes, ending execution in exactly 25th minute. Vertical axis represents amount of thread context switches during specified time units. As it can be seen from the figure above, performance improvements are around 2%, not as much dramatic as we have expected. However, as it is the result from our initial implementation, we are working/expecting performance to be better with more accurate implementation of our eCos Hybrid-EDF. We will make further enhancements and illustrate them in future works.

6 Conclusions and future work

In this work we addressed extension of eCos kernel with efficient EDF scheduler. We started from introduction of execution characteristics and software design of our practical application – UAV OFP. OFP sensor and controllers, periodical execution nature and timely criticalness of UAV control module were explained in the second chapter. OFP explanation was accompanied with necessary UML diagrams for thread collaboration and state diagrams. Next, based on our practical application requirements, we defined problem itself and stated the reasons for our enhancement approaches.

With brief look at eCos RTOS features, we moved to the detailed explanation of approaches, including source code skeletons. We also briefly introduced feasibility check we have integrated to the eCos-EDF kernel. Experimental results were presented on two categories. First, we illustrated our eCos kernel performance for OFP-motivated producer-consumer prototype application. Next, implementation results were illustrated for real OFP. As we have mentioned above, further enhancements are needed to improve eCos-Hybrid-EDF implementation. Future works will contain more exact data on this topic.

ISBN 978-979-16955-1-0

Acknowledgments

This research was supported by the MKE (The Ministry of Knowledge Economy), Korea, under the ITRC (Information Technology Research Center) support program supervised by the NIPA (National IT Industry Promotion Agency) (NIPA-2010-C1090-1031-0003).

References

[1] Modern Operating Systems, by Andrew S. Tanenbaum, Pearson Education, 2nd edition, pp. 132-152, pp.~ 473-475, 2001.

[2] Embedded software development with eCos, Anthony J. Massa, Prentice Hall, 2002.

[3] Liu, C. L.; Layland, J, *Scheduling algorithms for multiprogramming in a hard real-time environment*, Journal of the ACM, Vol 20, No 1, pp.~46-61, 1973.

[4] Giorgio C. Buttazzo, *Rate Monotonic vs. EDF: Judgment Day*, Real-Time Systems, Vol 29, No 1, pp.~ 5-26, 2005.

[5] Jansen, Pierre G. and Mullender, Sape J. and Havinga, Paul J.M. and Scholten, Hans, *Lightweight EDF Scheduling with Deadline Inheritance*, Internal Report, University of Twente Publications, 2003.

[6] Doo-Hyun Kim, Nodir Kodirov, Chun-Hyon Chang, Jung-Guk Kim, *HELISCOPE Project: Research Goal and Survey on Related Technologies*, ISORC 2009– 12th IEEE International Symposium on Object / Component / service-oriented Real-time distributed Computing, pp.~112-118. Tokyo, Japan, March 17-20, 2009.

[7] Nodir Kodirov, Doo-Hyun Kim, *A Design Strategy for Enhancing eCos with EDF for Disaster Response System*, International conference on IT Promotion in Asia 2009 in conjunction with International Summit on Information and Communication Technologies, pp.~212-216. Tashkent, Uzbekistan, September 21-25, 2009.

[8] eCos home page - http://ecos.sourceware.org/

[9] Albert Cheng, Real-Time Systems: Scheduling, Analysis, and Verification. John Wiley & Sons. ISBN: 0471-184063, pp. ~45-55, 2002.

[10] Alan Burns, Andy J. Wellings and Fengxiang Zhang, *Combining EDF and FP Scheduling Analysis and Implementation in Ada 2005*, Lecture Notes In Computer Science; Vol. 5570, Springer-Verlag, ISBN:978-3-642-01923-4, pp.~ 119-133, 2009.

[11] Robert I. Davis and Alan B, *A Survey of Hard Real-Time Scheduling Algorithms and Schedulability Analysis Techniques for Multiprocessor Systems*, Technical Report available at http://www.cs.york.ac.uk/ftpdir/reports/2009/YCS/443/YCS-2009-443.pdf

[12] Y-H Chao, S-S Lin, K-J Lin, *Schedulability issues for EDZL scheduling on real-time multiprocessor systems*, Information Processing Letters, Volume 107, Issue 5, pp.~ 158-164, 2008

[13] S.K. Baruah., A. Burns, *Sustainable Scheduling Analysis*, In Proceedings of the IEEE Real-Time Systems Symposium, pp. ~159-168, 2006.

[14] T.P. Baker, S.K. Baruah, *Schedulability Analysis of global EDF*, Real-Time Systems, 38: pp.~223-235, 2008.

[15] M. Bertogna, M. Cirinei, G. Lipari, *Improved schedulability analysis of EDF on multiprocessor platforms*, In Proceedings of the 17th Euromicro Conference on Real-Time Systems, pp.~209-218, 2005.