COPYRIGHT © 2011 IJCIT, ISSN 2078-5828 (PRINT), ISSN 2218-5224 (ONLINE), VOLUME 01, ISSUE 02, MANUSCRIPT CODE: 110103

The Standard of Embedded World by Studying of OSEK/VDX

Nahida Sultana Chowdhury, Muhammad Iqbal Hossain *and* Md.Tanvir Ahmed

Abstract—This paper will describe the concept of a real-time operating system, capable of multitasking, which can be used for motor vehicles. Embedded systems are characterised by stringent real-time requirements. Therefore the OSEK/VDX operating system offers the necessary functionality to support event driven control systems. This document also specifies the OSEK/VDX operating system - Application Program Interface. The specified operating system services constitute a basis to enable the integration of software modules made by various manufacturers. To be able to react to the specific features of the individual control units as determined by their performance and the requirements of a minimum consumption of resources

Index Terms— OSSEK/VDX, Embedded systems.

1 INTRODUCTION

IN recent years there has, in general, been a noticeable increase in the size and complexity of code in embedded system. Designers struggle to deal with these problems using traditional run-time structures (based on background loops and interrupt-driven processing) and seek better techniques. It is clear that one way forward is to use an operating system to support run-time operations. As a result now OSEK / VDX OS widefy used in embedded systems.

This paper is about embedded systems, OSEK / VDX Operating System introduction, specification, in short OSEK OS, which is an established standard in automotive industry. It focuses on the structure and architecture of OSEK OS, especially in the scheduling and resource management. Furthermore it will explain the importane role of OSEK to optimize the embedded systems. The OSEK OS intended to provide the necessary functional and non-functional requiremnets for embedded system software to execute.

2 EMBEDDED SYSTEMS

The word embedded reflects the fact that these systems are typically a fundamental part of a layer system. Typically, embedded systems also called embedded real time systems. An embedded system is a computer system designed to perform one or more dedicated functions often with real-time computing constraints. It is embedded as part of a complete device often including hardware and mechanical parts. By contrast, a general-purpose computer, such as a personal computer (PC), is designed to be flexible and to meet a wide range of end-user needs. Embedded systems control many devices in common use today.

Embedded systems are controlled by one or more main processing cores that are typically either microcontrollers or digital signal processors (DSP). The key characteristic, however, is being dedicated to handle a particular task, which may require very powerful processors. For example, air traffic control systems may usefully be viewed as embedded, even though they involve mainframe computers and dedicated regional and national networks between airports and radar sites (each radar probably includes one or more embedded systems of its own).

Since the embedded system is dedicated to specific tasks, design engineers can optimize it to reduce the size and cost of the product and increase the reliability and performance. Some embedded systems are mass-produced, benefiting from economies of scale.

Physically, embedded systems range from portable devices such as digital watches and MP3 players to large stationary installations like traffic lights, factory controllers or the systems controlling nuclear power plants. Complexity varies from low, with a single microcontroller chip, to very high with multiple units, peripherals and networks mounted inside a large chassis or enclosure.

In general, "embedded system" is not a strictly definable term, as most systems have some element of extensibility or programmability. For example, handheld computers share some elements with embedded systems such as the operating systems and microprocessors which power them, but they allow different applications to be loaded and peripherals to be connected. Moreover, even systems which don't expose programmability as a primary feature generally need to support software updates. On a continuum from "general purpose" to "embedded", large application systems will have subcomponents at most points even if the system as a whole is "designed to perform one or a few dedicated functions", and is thus appropriate to call "embedded".

- *Nahida Sultana Chowdhury is with the Department of Computer Science and Engineering, Kyungpook National University (wwwknu.ac.kr), Daegu, South Korea. E-mail: nahida_uap@yahoo.com.*
- *Muhammad Iqbal Hossain is with the Department of Computer Science and Engineering, Kyungpook National University (wwwknu.ac.kr), Daegu,South Korea. E-mail: milon_mi7@yahoo.com.*
- *MD. Tanvir Ahmed is with the Annanovas Company (www.annanovas.com), Dhanmondi, Bangladesh. E-mail: im_tanvir2007@yahoo.com.*

2.1 CHARACTERISTICS

- Embedded systems are designed to do some specific task, rather than be a general-purpose computer for multiple tasks. Some also have real-time performance constraints that must be met, for reasons such as safety and usability; others may have low or no performance requirements, allowing the system hardware to be simplified to reduce costs.
- Embedded systems are not always standalone devices. Many embedded systems consist of small, computerized parts within a larger device that serves a more general purpose. For example, the Gibson Robot Guitar features an embedded system for tuning the strings, but the overall purpose of the Robot Guitar is, of course, to play music. [5] Similarly, an embedded system in an automobile provides a specific function as a subsystem of the car itself.
- The program instructions written for embedded systems are referred to as firmware, and are stored in read-only memory or Flash memory chips. They run with limited computer hardware resources: little memory, small or non-existent keyboard and/or screen.

3 OSEK/VDX

A consortium of mostly European automotive companies created the OSEK standard in 1993. The consortium's goal was to establish Open Systems and the corresponding Interfaces for Automotive Electronics / Vehicle. In 1994, OSEK merged with VDX, a similar initiative in the French automotive industry. The two standards bodies presented their harmonized OSEK/VDX standard in October 1995.

OSEK/VDX has gained broader recognition in the industry. Founding members are BMW, Delco Electronics, Motorola, Opel, Siemens, ST Microelectronics, Renault, United Technologies Automotive, etc.

3.1 Specification of osek/vdx

- OSEK Operating System (OS) defines a standard interface for a single-processor operating system and offers the necessary functionality to support event-driven control systems. The OSEK OS is the primary focus of this article. OSEK OS allows direct interfacing between application and the hardware.

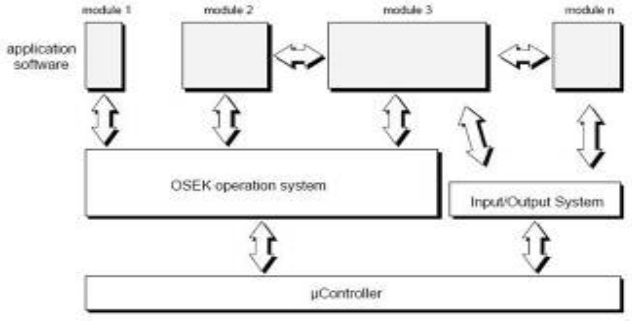

Figure 1: Software interfaces inside Eectronic Control Unit

- OSEK Communication (COM) defines a protocol for inter-task and inter-module communications among deeply embedded systems.

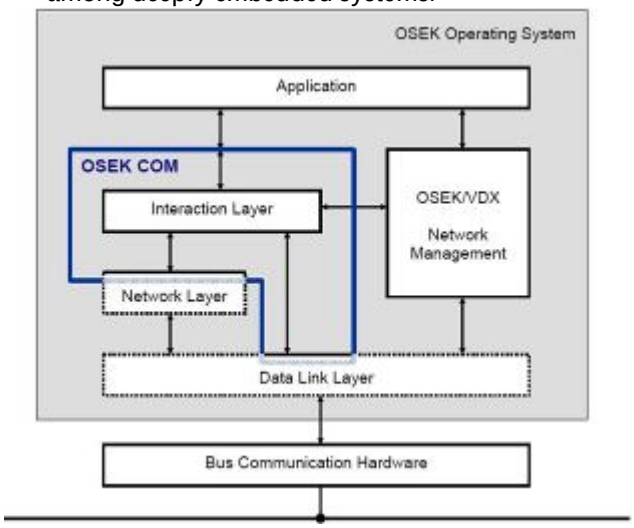

Figure 2: OSEK Layer Model with OSEK OS

- OSEK Network Manager (NM) defines protocols for managing networks during runtime. The NM provides standardized features, which ensure the functionality of inter-networking by standardized interfaces.
- OSEK Run-Time Interface (ORTI) enhances Interoperability and portability by defining a common interface for any microcontroller platform and any OSEK vendor. ORTI allows different development tool vendors to debug various OSEK implementations. In addition, an ASCII interface for the ORTI file makes extensions easy and manageable.
- OSEK Implementation Language (OIL) is the Configuration language that allows embedded designers to describe the complete OSEK system for system configuration and generation. OIL creates readable, archivable ASCII text files and is the key to enabling interoperability between different OSEK OS vendors.

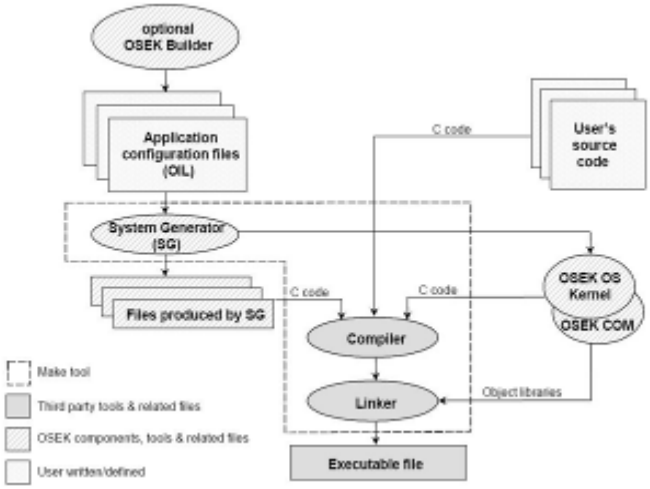

Figure 3: Example of development process for applications

- Time Triggered OS (OSEKtime) is an emerging standard that extends the OSEK OS specification to allow systems designers to sequence applications and communications (via FTCom) on local and remote controllers as if they were running on the same controller.
- Fault Tolerant Communication (FTCom) usually represents part of an OSEKtime implementation and defines a standard interface for fault tolerant communication.

All these specifications tend to move forward independently of one another.

3.2 Task Management of osek/vdx

A task is a framework for a specific functionality within complex control software, which should be executed according to its timing requirements.

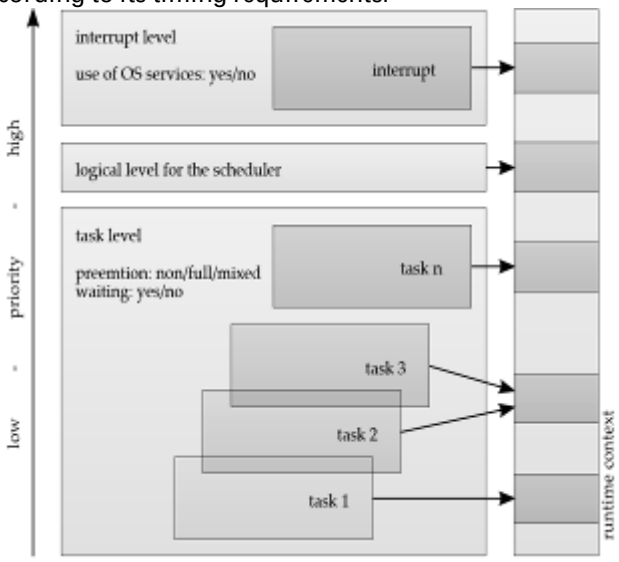

Figure 4: Processing levels inside OSEK OS

The OSEK operating system provides a task switching mechanism), including a mechanism which is active when no other system or application functionality is active. This mechanism is called idle-mechanism. Two different task concepts are provided by the OSEK operating system:

- Basic tasks
- Extended tasks

3.2.1 Basic and Extended Task Models

OSEK OS distinguishes between two task types: Basic Tasks and Extended Tasks. The main difference is that extended tasks are allowed to wait for events whereas basic tasks are not. A task has to change between different states, because the processor can only execute one instruction of one task at a time. While one task is executed, others are competing for the processor, and the operating system is responsible for saving and restoring a task's context whenever state transitions are necessary. A basic task can only enter three states, shown in figure 5. A task is

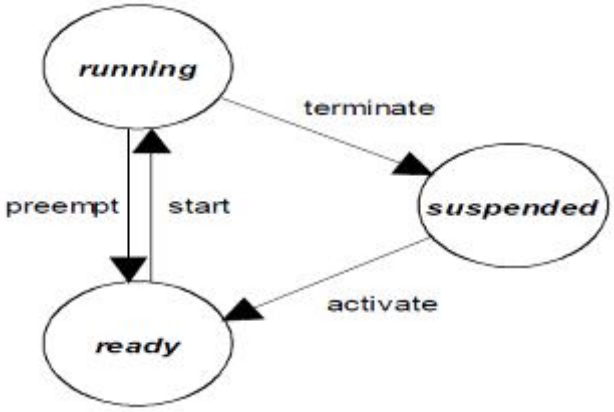

Figure 5: task states of basic tasks

in the suspended state when it was not activated or has terminated. This state exists because tasks cannot be created during execution of the system and must be defined statically during system design.Tasks can only be terminated when they are in the running state, meaning they can only terminate themselves which was a decision of OSEK design, made to keep the task state model and implementation as simple as possible. A task usually ends its execution with a call to the TerminateTask system service, but there is another function, ChainTask, which terminates the task and activates another one. The ChainTask service can be used to easily produce an asynchronous execution of various tasks. The extended task model has the additional waiting state (figure 6).

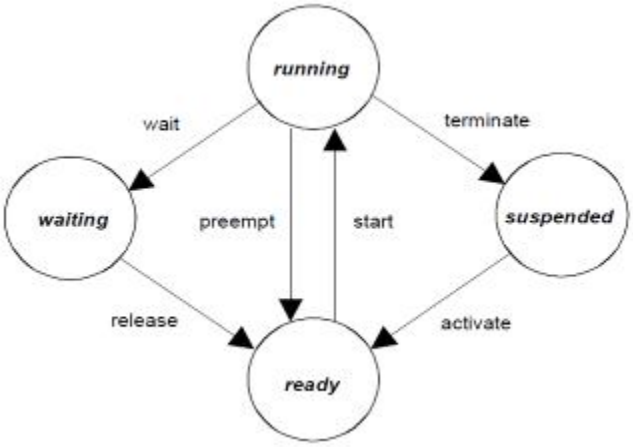

Figure 6: task states of extended tasks

3.3 Conformance Classes

Any OSEK OS conformant implementation must be compatible with one or more of the four Conformance Classes. They exist to allow better scaling of the system through partial implementation along defined lines. Conformance classes (figure 7) are determined by three attributes:

- Support of extended tasks or basic tasks only
- Multiple requesting for activation of a basic task
- Count of tasks per priority

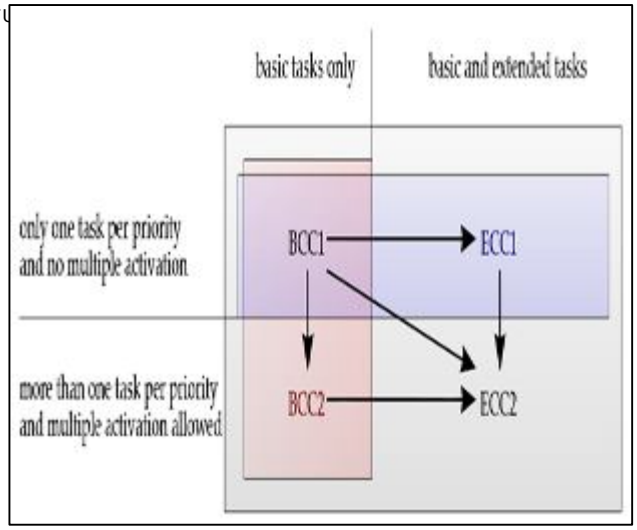

Figure 7: Relation and update path of conformance classes

4 TASK SCHEDULING POLICY

4.1 Full Preemptive Scheduling

It means that a task which is presently running may be re-scheduled at any instruction by the occurrence of trigger conditions pre-set by the operating system. It will put the running task into the ready state, as soon as a higher-priority task has got ready. But The task context is saved so that the preempted task can be continued at the location where it was preempted. Its latency time is independent of the run time of lower priority tasks.

In this Figure-8 task T2 with the lower priority does not delay the scheduling of task T1 with higher priority.

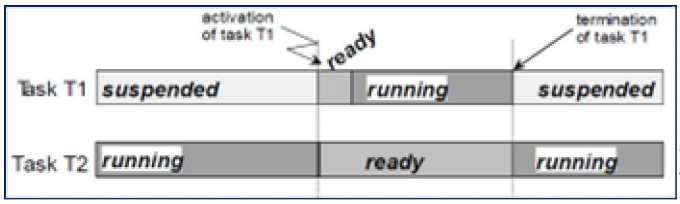

Figure 8: Full preemptive scheduling

4.2 Non Preemptive Scheduling

Here a running task with lower priority delays the start of a task with higher priority up to the next point of rescheduling.

In this Figure-9 task T2 with the lower priority delays task T1 with higher priority up to the next point of rescheduling (in this case termination of task T2).

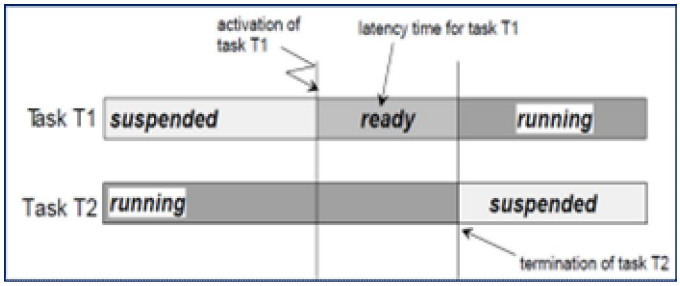

Figure 9: Non preemptive scheduling

4.3 Mixed Preemptive Scheduling

The scheduling policy is called mixed preemptive scheduling if tasks with full preemptive and tasks with non-preemptive policy are both existent on the same system. The scheduler is invoked depending on the policy of the currently running task which means if the task is non-preemptable, non-preemptive scheduling is performed and if the task uses full preemptive scheduling policy, rescheduling can occur on any specified event.

This policy is a compromise between non and full preemptive scheduling and was introduced to fulfil the usual needs of control applications which consist of only a few parallel tasks with long execution times for which a full preemptable system would be best and many short tasks with defined execution times for which a non-preempting system would be more efficient.

5 RESOURCE MANAGEMENT– PRIORITY CELLING

There are various problems to be solved by the resource management. It has to coordinate concurrent access of tasks with different priorities to shared resources. A resource can be everything that must or could be managed by the functionality of the resource management which includes management entities itself (scheduler), program sequences, memory areas and hardware objects. There are several constraints to be ensured during the distribution of resources:

- Two tasks cannot occupy the same resource at the same time.
- Priority inversion cannot occur.
- No deadlock is possible when using these resources.
- Request of a resource by a task does not result in a blocked state.

5.1 Priority Celling

The Priority Ceiling Protocol resolves the problems of priority inheritance as well as deadlocks and fulfils the requirements of resource management and operates as follows.

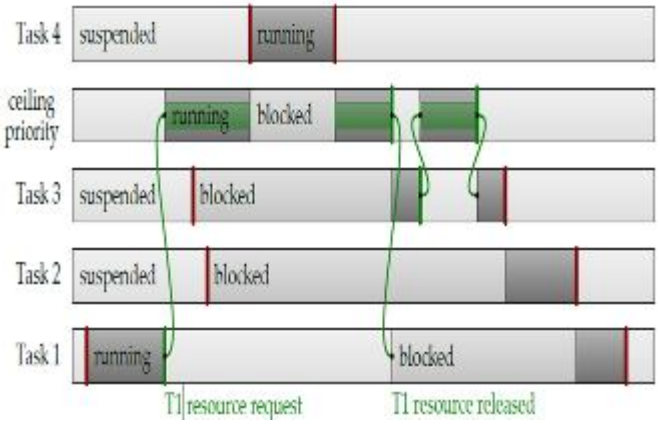

Figure 10: Example of priority ceiling

Every OSEK OS application is a statically scaled and specified system which includes tasks with static priorities and all re-

sources. These definitions are extended by the priority ceiling protocol by the means of assigning an additional ceiling priority to every resource. The ceiling priority of a resource is set according to the highest priority of every task requesting the resource. The priority must be at least the numerical value of the highest task requesting this resource but it should not be larger than priorities of tasks never requesting the resource.

If a task occupies a resource the task's priority is temporarily raised to the ceiling priority of the resource and thus the task cannot be preempted by another task requesting this resource. The task shall not terminate, wait for an event or exclusively call the scheduler while accessing a resource.

If a task releases a resource, its priority is lowered to the highest ceiling priority of the remaining owned resources or to its initially assigned task priority. OSEK OS assumes a LIFO order of resource demands and matching releases. This simplifies the determination of the resulting task priority equal to the one the task had before it occupied the resource.

Resulting from the priority ceiling protocol every request can be fulfilled instantly because the requesting task could not be selected by the scheduler if the resource is occupied.

High priority tasks can be blocked by low priority tasks due to priority ceiling. The maximum time the task can possibly be blocked is the maximum time of resource occupation by a low priority task of a resource with greater or equal ceiling priority than the task in question.

6 OTHER FEATURES OF OSEK/VDX

6.1 Counters and Alarms

Counters are OSEK OS implementation specific objects, such as timers or sensors, of the hardware represented as an integer value. It is the responsibility of the implementation how counters are manipulated. At least one counter, which is derived from a software or hardware timer, is present in every system.

An alarm is an action which is executed when a counter reaches a specific value. These actions can be task activation, execution of a callback routine on interrupt level or setting of an event whereas the underlying value of the counter can be specified as either absolute or relative. Alarms are specified during system creation and are assigned to one counter and one task or callback. One counter can activate more than one alarm whereas an alarm can be recurrent, too. OSEK OS provides services to manage the alarms which means setting and cancelling them as well as checking their state.

7 THE ROLE OF OSEK/VDX IN EMBEDDED SYSTEMS

7.1 Benefits From Standardization Of OS

Standardization of a real-time OS should on the one hand be targeted to improve the efficiency of the embedded software development process. On the other hand a specification which is driven by the customers' needs should lead to OS implementations which fulfill the industry's requirements. The major advantages using a standardized OS are:

- Common OS concepts from different suppliers on different platforms.
- Reliable management of the applications' real-time behavior in complex ECUs.
- Identical OS-API when working on different processors.
- Common abstraction of the underlying hardware.
- Reusage of a widely used software component.
- Scalable implementation with adequate performance.
- Concentration on the core business, i.e. application development.

7.2 Requirements to an Embedded system OS

Automotive applications introduce very stringent requirements to the embedded software and thus to the underlying OS. High production volumes of automotive ECUs introduce an enormous cost pressure to the manufacturers. Hence a very economical usage of memory resources and CPU performance has to be achieved. On the other hand powertrain applications can be classified as hard real-time applications since a system failure might lead to severe damages of the car and the passengers. Timing requirements are also very hard because powertrain applications have to track fast physical processes for instance at high crankshaft revolutions.

Together all these arguments lead to the demand of a very efficient operating system implementation. Very fast response times to asynchronous events have to be combined with the capability to manage task activations with periods in the range of 1kHz and more. Nevertheless, the overall CPU-load introduced by the operating system should not exceed 5% in total. Memory consumption for code, RAM- and ROM-data should be highly optimized to reduce the overall system costs.

Modern luxury vehicles contain up to 70 ECUs linked together over typically two or three CANbuses'. The applications range fiom rather simple functionalities like controlling motors for window openers to complex real-time software like engine management systems or gearbox control. Smaller ECUs might provide just 2 Kbytes of RAM whereas larger ECUs offer up to 64 Kbytes Therefore a general OS concept has to be scalable in order to be adapted to each application's needs.

7.3 How Does Osek Meet These Requirements

First of all OSEK addresses the need of an economical memory usage by statically configuring the operating system. Features like dynamic creation or instanciation of OS-objects, e.g. tasks, alarms or resources are not supported. All OS-objects are defined in a centralized file which is compiled and linked together with the application.

Secondly, an OSEK-OS does not support memory management functionality at runtime. The complete memory is allocated statically at system configuration time. Operating system objects do not share common memory areas2. Ths restricton does not allow for the most efficient usage of RAM but greatly increases the system stability and reduces the runtime overhead of the OS. Furtheron the OS is just working explicitly and not periodically, i.e. scheduling just takes place when calling the dedicated OS-sewices like ActivateTask, TerminateTask.. . Hence the runtime requirements of the OS are predictable by the application which is a very important fact for the design process of ECU-software.

In order to support scalability for the different automotive

applications, the OSEK specification defines four conformance classes according to the task structure of the application. Two different implementations of the OS (Standard- and Extended-Status) are defined to distinguish between the development and production phase of ECU-software projects.

After all the OSEK OS specification is designed to solely fulfill the core requirements of automotive embedded applications. These are:

- Scheduling of concurrent tasks.
- Providing a protected access to shared resources.
- An efficient management of time dnven tasks.
- Handling of asynchronous events and interrupts.
- A configurable error handling mechanism.

8 IMPLEMENTATION

[Source wikipedia: http://en.wikipedia.org/wiki/OSEK]

- TOPPERS-OSEK is a free, open source implementation (GPL license)
- openOSEK is a free, open source implementation (LGPL license)
- Trampoline is another open source implementation (LGPL license)
- PICOS18
- nxtOSEK is an open source implementation for the Mindstorms NXT robots
- OpenSEK is an OpenSource OSEK-VDX Implementation, still in pre release.
- ERIKA Enterprise is an implementation of the OSEK OS (BCC1, BCC2, ECC1, ECC2), OIL, ORTI specifications, provided with an Eclipse plugin and support for Microchip dsPIC, AVR, Nios II, ARM7. (License: GPL linking exception)

9 CONCLUSION

After all, OSEK OS is very flexible, which makes it hard to ensure realtime operation, but with some restrictions of the full OSEK/VDX operating system functionality it is possible to use it as a basis for realtime applications. Non-critical applications can still use the extended functionality of the system, such as events, but those should be background tasks and run with lowest priority so that they do not interfere with the critical operations.

As summary it can be said that the OSEWVDX project is an excellent example of an straightforward and successful co-operation within the automotive industry of different countries. The elaborated specifications of the application independent areas Operating System, Communication and Network Management seem to be a sufficient basis for the development of portable and reusable application software. This is emphasised by the world-wide interest within the automotive community. Nevertheless, lot of work remains to be done to achieve an acknowledged industrial standard. With active co-operation of all interested companies under the described lean organisational structure the aimed objectives can be achieved. Thus, the use of OSEK/VDX production cars will become reality in near future.

REFERENCES

[1] http://www.osek-vdx.org

[2] OSEK/VDX Communication Version 2.1 revision 1, OSEK Group, June, 1998.

[3] OSEK/VDX Operating System Specification 2.0, OSEK Group, June, 1997.OSEK/VDX.

[4] John D. OSEK/VDX history and structure. IEE Seminar 1998; 523:2/1-2/13.

[5] http://www.microsoft.com/windowsembedded

[6] Michael Barr. "Embedded Systems Glossary". Netrino Technical Library. http://www.netrino.com/Embedded-Systems/Glossary . Retrieved 2007-04-21.

[7] Embedded Linux - http://www.linuxdevices.com

Nahida Sultana Chowdhury is currently studying at Kyungpook National University, South Korea. She has been awarded the Korean Government Scholarship Graduate Program (2009-2012) by National Institute for International Education (NIIED) for her academic excellence. She has obtained the B.Sc. degree in Computer Science and Engineering from the University of Asia Pacific, Bangladesh, in April, 2008 (Vice Chancellor's Gold Medalist). From 2008 to 2009, she was a Teaching Assistant in the University of Asia Pacific (UAP), Bangladesh.

Muhammad Iqbal Hossain is studying in Master's in Computer Science and Engineering in Kyungpook National University, South Korea. He has been awarded the Korean Government Scholarship on 2009. He has completed his Bachelor degree in Computer Science and Engineering from The University of Asia Pacific 2008. He was a teaching assistant in The University of Asia pacific. His research interests are on embedded system, Artificial Engineering, Software engineering etc.

Md Tanvir Ahmed is serving as a software developer in Annanovas. He has completed his Bachelor degree in Computer Science and Engineering from The University of Asia Pacific 2008. His research interests are in Software engineering, Algorithm, Artificial intelligence, Database system etc.