Proceedings of the International Multiconference on
Computer Science and Information Technology, pp. 745 – 750

ISBN 978-83-60810-14-9
ISSN 1896-7094

Development of a Flight Control System for an Ultralight Airplane

Vilem Srovnal Jr, Jiri Kotzian
VSB – Technical University of Ostrava
Department of Measurement and Control, FEECS
17. listopadu 15, 708 33, Ostrava – Poruba,
Czech Republic
Email: {vilem.srovnal1, jiri.kotzian}@vsb.cz

***Abstract*—This paper presents development of the hardware and software for the low cost avionic system of ultralight airplanes. There are shown three levels of a hardware and software system design. As far as the software is concerned, we focused on changeover from non real-time operating system (embedded Linux) to the real-time embedded platform (QNX). We discussed the problems that led to operating system change. Various advantages and disadvantages of both operating systems are presented in this contribution. Concerning the hardware we concentrated on the development of the avionic control, monitoring and display modules that are a components of the dash board. The paper has been focusing on safety and reliability in the ultralight aviation and what can be improved or extended by the real-time operating system.**

I. Introduction

THIS paper shows dependence hardware and software reliability. The control and monitoring system for ultralight or sport airplane has to be stable in an extreme situation as well as in a daily routine. When one designs your own hardware and software architecture one has to keep in mind the hazard states that can occur in many cases. In the aviation industry the safety and reliability are very important goals. But the other aspect is a product price. It means that hardware for double or triple protection is too expensive for the customer so we have to presume that hardware will work reliably at all times. This is the same case of our system that is why we focus on software architecture development in this article. The embedded Linux as main operating system has been implemented in our embedded system. The basic problem are drivers that have to be certificated and be very safe as all operating system. On the Linux platform it is difficult to ensure that drivers do not corrupt the kernel because both run within the same address space as the OS kernel. Therefore we were looking for a new software architecture solution. As the ideal answer it resulted in QNX real-time platform with client-server architecture where drivers are independent on the kernel (microkernel) that only implements the core services, like threads, signals, message passing, synchronization, scheduling and timer services. There are other possible real-time operating system of course that are very often use in the avionic systems, for example VxWorks from Wind River or Integrity from Green Hills but the problem for the low cost product is license price of the RTOS. When the run-time license price of RTOS is over the client limits there is no way how to use them. Especially Integrity software is very powerful system for developing purpose on the different platforms and has a very nice debugging tools.

This work is supported by grant of the Grant Agency of the Czech Republic GA102/08/1429 - Safety and security of networked embedded system applications

II. RTOS Selection

The right choice of a software system architecture is important for following development of all stuff. Due to the lack of our experiences and that requirements on the system where growing up during developing period, it caused that we had to try three different architectures.

A. Embedded uClinux

The first platform was the embedded uClinux based on microprocessor ColdFire MCF5329 from Freescale. This solution had shown us that graphic hardware is not powerful enough as we supposed and the lack of MMU was the huge

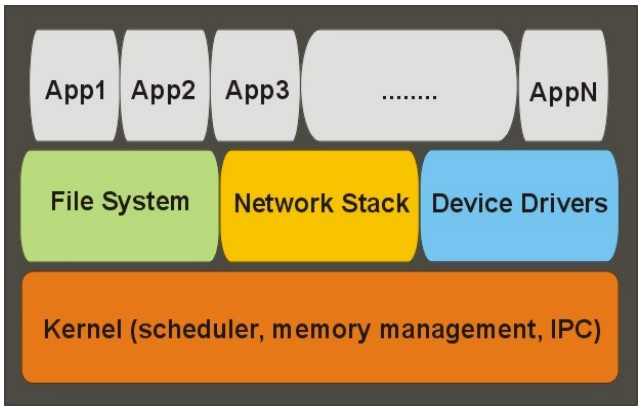

Fig 1. Architecture of uClinux - traditional for old RTOS

problem as well. [3] On the other hand the advantage of this solution is quick access to memory and integrated CAN hardware device on the developing board. The address space

is common for the kernel as well as for the user application which decreased safety and reliability of the entire control system but it is real-time in nature because there is no overhead of system calls, message passing, or copying of data. The uClinux architecture (flat addressing model) is shown on the Figure 1.

B. *Embedded Linux*

The embedded Linux architecture was chosen as the next platform for avionic system. This architecture is based on monolithic kernel that has distinction between the user and kernel space. [5] Any fault in the application will not cause system crash. The system has run on ARM processor PXA 270 from Intel and ARM processor PXA 320 from Marvell. The both processors are very powerful but there is a lack of floating point instructions set that is very important for graphic operations. The next problem that was mentioned above is incorrectly written driver or module that can cause the system to crash. [6] The embedded Linux architecture is shown on Figure 2.

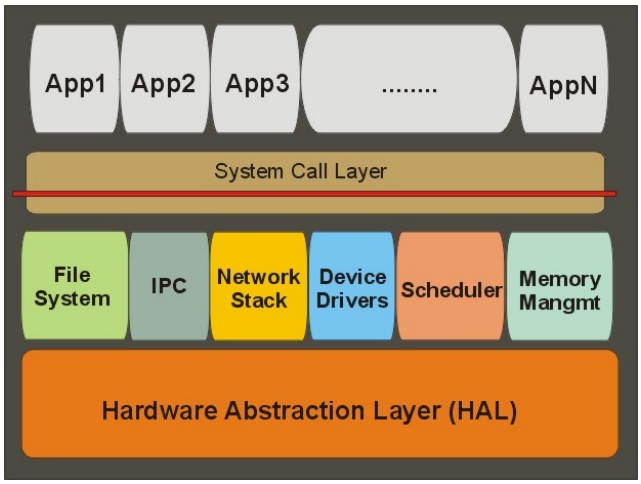

Fig 2. Architecture of embedded Linux

C. *QNX Neutrino real-time OS V 6.3*

The embedded Linux has worked without problems but there was demand on system to be high reliable and safe. The embedded Linux was sufficient for the monitoring of avionic values but for the control activity it was worse. We had to ensure time death line for the control process activity and graphic smooth step to the next frame on LCD display. There were requirements to isolate graphic process from control process but in monolithic kernel architecture the graphic driver could corrupt all system. [4] So the next logical step was to use microkernel to realize mentioned requirement. The basic principle of microkernel architecture is that each of kernel subsystem (network stack, file system, scheduler, etc.) has private address space similar to application. This way offers complete memory protection, not only for user applications, but also for OS components. This architecture provides maximum modularity and relies on robust message passing schema. The QNX microkernel architecture is shown on Figure 3. [11]

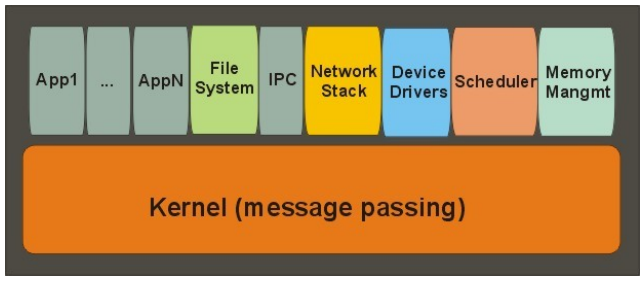

Fig 3. Architecture of QNX microkernel

Due to the lack of floating point instructions set we needed to use another processor that supports FPU and QNX RTOS. We decided to use ARM microprocessor iMX31 from Freescale but parallel solution stop on ARM processor PXA 270 with QNX RTOS. Unfortunately for the ARM microprocessor PXA 320 was not QNX BSP available.

III. Software Architecture

The reliability is main goal in the process control of the most industrial or others systems. As we presented in software architecture part, there are many reasons, why divide all control and monitoring system into the smaller software distributed units – processes.

The client-server architecture brings features that are able to guarantee more stability of the whole system. The failure of one part of the system (process) causes crash of any other one. The interprocess communication is based on robust message passing. The processes are separate according its functionality and hardware requirement. It means that each process uses one specific driver. Every individual process runs in independent address space.

There are five basic process groups. The first one is the motor control group that is responsible for everything what is connected with plane control. We measure avionic sensors inputs that are needed for autopilot control, maximum and minimum plane speed, etc.

The second group takes care of communication with other hardware parts. We are using CAN communication protocol and CAN driver where each hardware device is CAN node. [2] There is possible to use serial communication RS232 or USB as special process as well but for another purpose.

The third big group is SCADA system. The SCADA system or we can say graphic tasks that are responsible for monitoring and display avionic values in continuous time (it depend on processor time scheduling). Graphic process is very demanding on time of the processor so the priority level are low but have to ensure specific frame count per second.

The fourth group is IO device control for other purpose. We use GPIO driver for pins states monitoring. Each pin has special function where the most of them are used as button, trimer or switch. We use GPIO-event handling mechanism for pin state tracking.

The fifth group and the last process group is audio device group that is responsible for voice transmitting into the central communication system. There is audio system for warning and critical errors as well. The audio system can be used for other purpose it is up to client additional extension. The

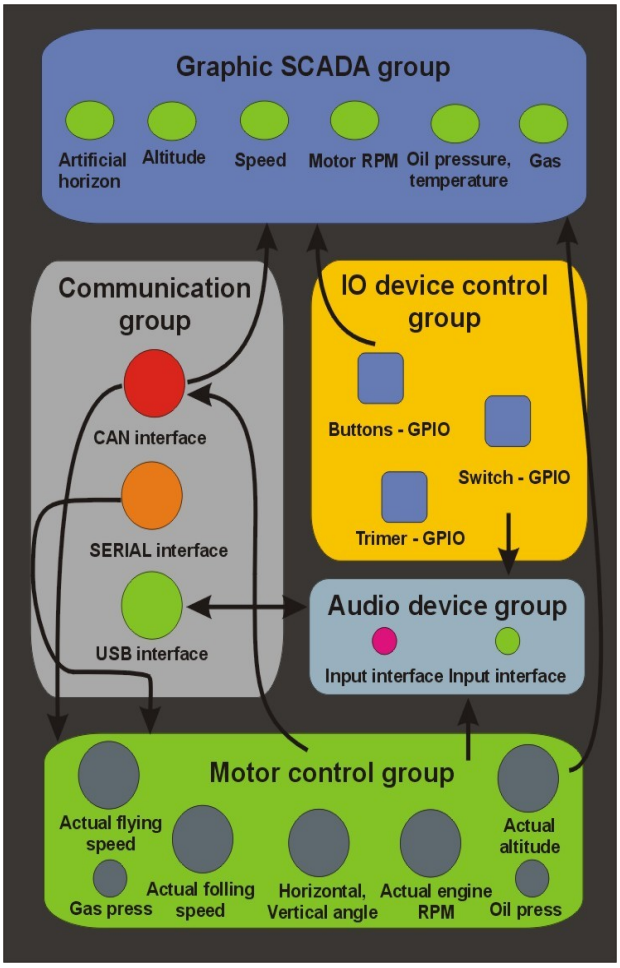

Fig 4. Architecture of process control

following schema on the Figure 4 shows process structure. The communication group allows data for other process groups from sensors and actuators. The motor control group uses communication group for engine values control. The communication unit has to be one of the most reliable part of the control and monitoring system.

IV. Critical System Control

The basic critical processes of the aviation control have to meet the following criteria. The plane take-off speed as well as landing speed has to meet the required characteristics. These values are measured with sensors which are build-in the hardware board. The measurements are completed by the pressure difference against the surrounding environment. All data which have been gained from the sensors are used for the rotation engine process. The measurement of the stalling speed is a very important to prevent the uncontrolled plane fall /accident/ and jet-engine failure which is closely connected with this event. The measurement of the vertical and horizontal plane heeling is the next critical process. Exceeding the allowed plane heeling can result in stability loss of the airplane. To make the jet-engine work sufficiently the pressure in the gas and oil tank must be sustained. Other important flying data are information about a plane position. The position is shown through GPS system. All above mentioned processes must run in the independent address space so that the safety of the system is sustained.

The driver design is very important for communication and stable control of the application. The critical data goes through CAN interface which gives or receives information for engine control and fly control. The CAN driver is connection between module control node (present in next chapter) and motor control process group. The next sophisticated driver which is used for SCADA system is framebuffer driver. This driver has to be very efficient and quick. The visualization system was extended by OpenGL ES 2D driver [12]

V. Hardware Architecture for Avionic Systems

The avionic system is distributed into the independent modules that measure specific values on mechanical parts of the airplane. The hardware solution is a configurable according type of airplane. The highest layer is graphic user module that represents received data on the LCD display. The sense of monitoring system is offer customers same facilities as have pilots in the professional aircrafts and make aviation more easier using low cost embedded electronic system.

The real-time embedded control system is designed with a modular structure. [4] This structure supports a flexible configuration. In terms of user requirements, the control system can be configured in different sizes and options. [8] The s everal modules with different options were designed. All modules are connected to an industrial bus – so each module is the bus node. Except the GPS module that is connected directly to the main control module.

This architecture supports a future expansion. In terms of user requirements it is possible to design new modules. The new module node will be connected to the bus. The new module can work to satisfy a user after upgrading of the firmware in the main control mode. This way it is possible to connect a maximal 30 modules - nodes. The block diagram of a desk control and monitoring system with today's full configuration of prototype is shown on the Figure 5.

The monitoring and control modules are connected together by using an industrial bus [2]. This bus has to be highly reliable and have enough speed. Depending on these two main requirements a CAN bus was selected. The main reason is that the CAN has an extremely low probability of non-detected error. The versatility of the CAN system has proven itself useful in other applications, including industrial automation as well, anywhere that a network is needed to allow controllers to communicate. A CAN bus is given the international standard ISO11898 which uses the first two layers of ISO/OSI model (CAN-CIA 2005). The CAN is a multi-master protocol. When a CAN message is transmitted over the network all nodes connected to the network will receive the message. [7] Any node can begin transmitting at any time the bus is free of traffic and all nodes will listen to the message. Each node may employ a filtering scheme that allows it to process only relevant information. Each message has either an 11 bit identifier or 29 bit identifier which will define which node will receive the message, error checking bits, 8 bytes of data and priority information. If two nodes try transmitting a message at the same time, the node with

the higher identifier (lower priority) loses control of the bus as soon as another node exerts a dominant bit on the bus. It ensures that the message with the highest priority will be transmitted on the first try.

Designed modules are the following:

- Main control module
- User interface (LCD display) module
- Motor measuring values module
- Advanced avionic data module
- Black-Box module

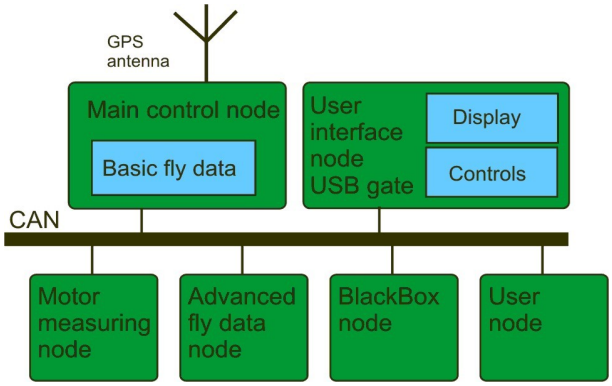

Fig 5. Block diagram of the control and monitoring system

A. Main control module

The main control node serves as master for all other nodes. (Figure 6) [1] The requesting values from other nodes are compared with given limits and stored in the local memory. The main module decides what will be displayed and sends information to the user interface module by the CAN bus. The main control module contains a real time clock and data flash memory for storing measured values and statistics. For a measuring of the basic values the main module is equipped by sensors.

B. Engine monitoring and control module

The engine monitoring and control module supports the complete monitoring and control of all necessary engine data. It collects all possible temperatures, pressures and the diagnostics of the engine. The basic motion control values and critical avionic data are already connected to the main control module.

C. Avionic monitoring module

The advanced avionic data are supported by this module, for example a magnetic compass. This module only increase basic monitoring and control data that are supported in the main control module. It is up to client requirement.

D. Black-Box module

The black-box module controls all traffic on the CAN bus. It reads data from CAN messages and stores data in the local memory. The black-box module is equipped with its own RTC timer and stores time together with the CAN data.

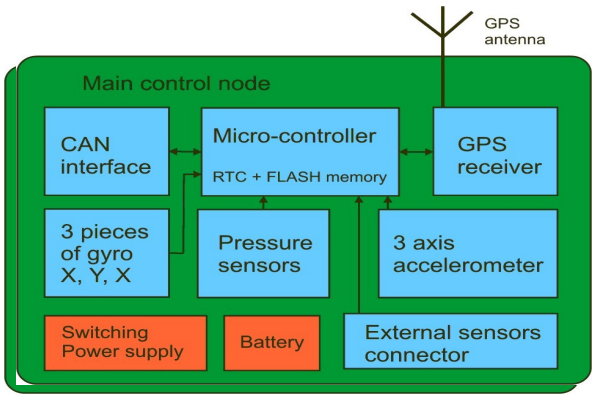

Fig 6. Main control node block diagram

There is no other connection to this module with such high reliability. [10]

E. Development kits

There were three development kits used during the whole evolution. The first development kit that has been used comes from the world-wide known company – Freescale. This development kit is based on 32.bit microprocessors ColdFire family. This microprocessor runs on frequency 240 MHz and integrates display controller and other useful interfaces. [9] The second development kit that has been chosen comes from the Toradex company and integrates ARM microprocessor Intel PXA270 (Marvell PXA320) runs on frequency 520MHz respective for PXA320 on frequency 806MHz. There is an integrated display driver and CAN driver as well as other interfaces. [10] The third development kit that has been tested comes from company Logic PD. There is Freescale ARM microprocessor i.MX31 as SOM module integrated. This microprocessor runs on frequency 532MHz and integrates display controller and floating point unit as well.

F. Prototype hardware development solution

The prototyping is the long process of improvements and cost lot of money and time. We have focused on finishing development on the ARM processors (Intel, Marvell and Freescale). The prototype board is designed for SOM module with processor PXA270 and PXA320. We are planning to design a new prototype board for SOM module based on processor iMX31. There is possibility to use other SOM modules for our prototype board as PXA300 or PXA255 as well.

These SOM module are intended for graphic processing of avionic data. The measurement and control module is based on 16.bit processor HCS12 from Freescale. There is not any RTOS integrated within. A lot of main control functions have been moved to SOM module (PXA270, iMX31) with RTOS QNX. The prototype implements interfaces as CAN, RS232, USB, GPIO. The standard interfaces like SERIAL and USB are supported by QNX BSP package but CAN and GPIO drivers have been developed by our software team to ensure drivers stability and reliability. We have chosen standard CAN controller SJA1000 produced by the Philips company. The GPIO driver has been implemented using older

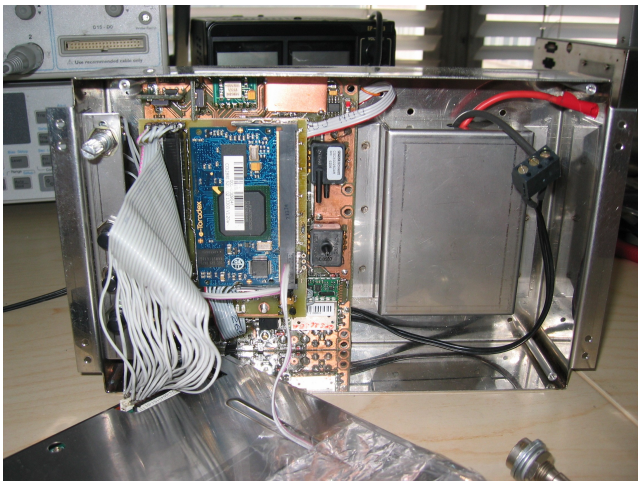

Fig 7. Prototype hardware realization

version where only address mapping had to be modified. The avionic system has three level of power supply. The first is standard supply from airplane battery that is recharged during a flight. When, due to some reasons or troubles this source is out of order, there are two batteries backup system. At the moment only one is embedded inside the box. The power supply for this prototype solution is 12V DC. The same power supply is used for LCD TFT display as well. The LCD TFT display has external backlight unit for better graphics recognition. The most energy from the power conception is consumed by the backlight module and LCD TFT display nearly more then 80% by the whole system. An extra extension can be reconfigured by the jumpers on the prototype board. Following Figure 7 and Figure 8 shows prototype realization.

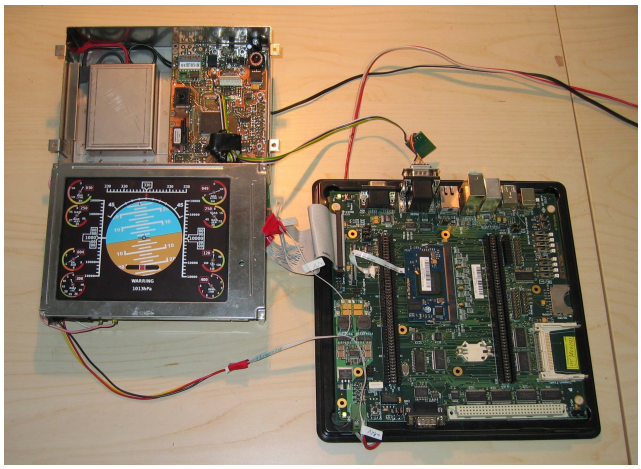

Fig 8. Prototype hardware realization with development kit

VI. Graphic System Design

The graphic system design for the embedded system is usually quite difficult to meet optimal parameters for the mobile devices. The prototype solution that we proposed is embedded in the global avionic system and runs continuously during the flight. There is a big problem how to ensure low power consumption when you have to supply backlight module and LCD display. We can control backlight using software settings but this is not a cellphone where it is possible to switch off display when not used. But there are special cases when the backlight is switch off. For example when main power supply has to be replaced by battery and system controls backlight directly.

The PXA processors 270 and 320 lack of floating point unit and there is only floating point emulation that is very slow. The graphic operations base on emulation floating point are five times slower then on a fixed math primer. The fixed mathematics operations have been based on 16.bits integer scale what is sufficient for a precision of our application. The graphics rendering process takes a lot of processor time. So we decided to integrate graphic acceleration chip 2700G5 by the Intel company. The graphic acceleration has increased the whole graphic process more then 100 times. The new processor PXA320 has integrated 2D graphics accelerator inside but the stable driver is not available yet. On the other hand the processor iMX31 by the Freescale company support 3D acceleration using their integrated chip inside and the driver is available for Linux and for QNX as well . The graphic library that has been used is OpenGL for embedded systems. There were only 2D graphic operations use in our system application. This OpenGL commands have to meet the avionic standards according OpenGL SC (Safety Critical for Avionics). The special example of 2D object rotation based on OpenGL is the gyro-horizon.[12]

VI. Conclusion

The main goal of this paper is to show development and realization of the low cost avionic control and monitoring system for ultralight planes. We discussed safety and reliability of the control processes. In the step by step procedure we described software development process as well as hardware development process. The whole system was presented as distributed control system that consists of a few modules interconnected using CAN bus interface. We presented makeover from common operating system Linux to real-time operating system QNX.

References

[1] Arnold K.: *Embedded Controller Hardware Design.* LLH Technology Publishing, USA, 2001, ISBN 1-878707-52-3.

[2] Sridhar T.: *Design Embedded Communications Software* . CMP Books, San Francisco, USA, 2003, ISBN 1-57820-125-X.

[3] Raghavan P., Lad A., Neelakandan S.: *Embedded Linux System Design and Development* . Auerbach Publication, New York, USA, 2006, ISBN 0-8493-4058-6.

[4] Li Q., Yao C.: *Real-Time Concepts for Embedded Systems* , CMP Books, San Francisco, USA, 2003, ISBN 1-57820-124-1.

[5] Hollabaugh, Craig.: *Embedded Linux,* Pearson Education, Indianapolis, USA, 2002, ISBN 0-672-32226-9.

[6] Yaghmour, Karim.: *Building embedded Linux systems,* O'Really & Associtates, Sebastopol, 2003, ISBN 0-596-00222-X.

[7] Kotzian J., Srovnal V.: *Can Based Distributed Control System Modelling Using UML.* In: Proceeding International Conference IEEE ICIT 2003, Maribor, Slovenia, ISBN 0-7803-7853-9, p.1012-1017.

[8] Kotzian J., Srovnal V.: *Development of Embedded Control System for Mobile Objects Using UML. In :* Programmable Devices and Systems 2004-IFAC Workshop, Krakow, Poland, IFAC WS 2004 0008 PL, ISBN 83-908409-8-7, p.293-298.

[9] Srovnal, V., Kotzian, J.: *FLYSYS–Flight Embedded Control System for Ultra-Light Airplane* . In proceeding 4th IFAC Symposium on Mechatronic Systems 2006, Duesseldorf, Germany, 2006, IFAC, p.998-1001.

[10] Kotzian J., Srovnal V. Jr,: *Distributed embedded system for ultralight airplane monitoring* , ICINCO 2007, Intelligent Control Systems and Optimization, Anger, France, 2007, ISBN 978-972-8865-82-5, p. 448-451.

[11] ONX Software Systems International Corporation: *QNX Neutrino RTOS–System Architecture* , Release V6.3 or later, Kanata, Ontario, Canada, 2007.

[12] Astle D., Durnil D.: *OpenGL ES Game Development*, Thomson Course Technology, Boston, MA, USA, 2006, ISBN 1-59200-370-2.