

Online Monitoring of Mobile Robots Using Low Cost Microcontroller Probes

Kemal Köker, J. Eckert, R. German

University of Erlangen-Nuremberg/Department of Computerscience 7,
Computer Networks and Communication Systems, Erlangen, Germany
{koeker, eckert, german}@informatik.uni-erlangen.de

Abstract

We present a cost efficient microcontroller based measurement infrastructure to analyse the performance of embedded systems. Our architecture allows monitoring and evaluating the system performance of distributed embedded mobile systems. For this purpose we selected the RoboCup small size league scenario and provided soccer robots with dedicated Linux driven embedded systems. The robots' embedded system has capabilities to process communication via WLAN and execute strategy applications in a fast way. Our architecture easily allows evaluating and verifying the usability of embedded systems in a dynamic environment with respect to response times. In addition, example measurements and results for the system response with our infrastructure are presented.

Keywords: embedded system, IR sensor application, microcontroller probe, online monitoring

1 Introduction

Simulations are frequently used to test the performance and usability of embedded systems in different operating scenarios before deploying them to real tasks. The main reasons for this are the cost involved in the deployment and reliability analysis, as well as the difficulties in executing hundreds of trials using real physical systems. The analysis of the usability also includes a performance analysis of the operating systems which requires e.g. monitoring the response time for interrupts that are triggered by an input (e.g. pushing a button or getting new values of sensors). Therefore the system model for the simulation should be as accurate as possible for minimum divergence between the simulation and the real world considering system responses. A lot of tools for building system models require data for the input modelling. Often these data represent the response time or one-way delay for an input, whereby the monitoring is done by soft- and hardware. Software monitoring often results to additional system load up to 5 percent and more, whereby the accuracy depends of the systems' load. Therefore software based system analysis is not appropriate as a non intrusive monitoring method. The better choice for this purpose is hardware monitoring which usually requires a specialized device with a built-in clock to generate timestamps for signals observed at some system interfaces. This solution described in [1] provides precise timestamps but can only be applied to systems located in a fix place.

Because of the mobility of the soccer robots, we build a low cost measurement infrastructure to locally monitor and evaluate the performance for response for

events or external interrupts. Our architecture uses an own designed microcontroller circuit and is logging the response times for interrupts or system events. The architecture allows online monitoring of the entire robot system and also allows on-demand-transmitting the data to a host computer for further evaluation.

2 RoboCup Test Application

The selected indoor application for the test bed is a scenario of mobile soccer playing robots similar to the F180 RoboCup [2]. In a common RoboCup small size league system, a host computer acts as a global vision-system for object recognition and performs remote control of each player according to the results of the centralized strategy software. In this case the robot players are not autonomous and remotely controlled by the host computer.

2.1 Soccer Robot

The architecture of a soccer robot in the F-180 league usually consists of the mechanical and electronic part. The mechanical part covers the body and the actors. The electronic part consists of a microcontroller for the actors, a dedicated FM-module to receive action commands from the host, and the power supply with batteries. Usually this architecture disallows autonomous behaviour like path planning or collision avoidance because each action command is triggered from the host computer. We modified the system by splitting the robots in an upper layer with an embedded Linux system, and the lower layer covering the mechanical and electronic part for the actors and sensors.

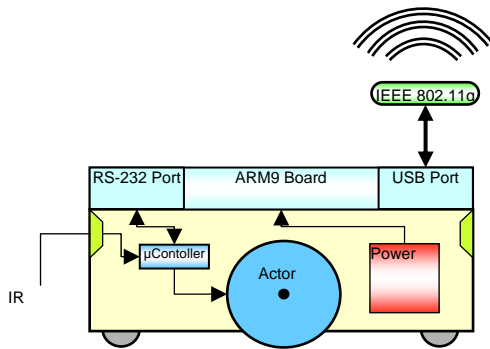


Fig. 1 Autonomous F-180 soccer robot

2.2 Embedded System

The embedded system of the upper layer is running at 180 MIPS and has capabilities for local execution of the strategy application. The communication with the host is done over WLAN (IEEE 802.11g). The object recognition is still handled by the host system and only coordinates of the objects are transmitted via WLAN broadcast to the robots. This modification shifts the autonomy from the host computer to each of the player. Figure 1 displays the schematic of our architecture for the soccer robots using the ARM9 embedded system.

For maximum flexibility we used the ARM9 embedded system from [3] with an Atmel AT91RM9200 200 MHz [4] fan less CPU and an onboard SD card socket and modified the layout according to our requirements, whereby the embedded system has a debug port, 4 serial ports and an USB host/client port. This embedded board is meanwhile a widely spread architecture for common embedded systems using the Linux or Windows-CE operating system. The ARM9 board in our architecture is operated by Gentoo Linux [6] with kernel 2.6.18. The whole system is stored on the SD card using 700 MByte of storage space. This ARM9 board was selected because of the easy access via pointer to mapped event registers which are usually protected by the operating system. This benefit allows toggling between event registers in our architecture for online monitoring of the running system. In our example test run we mapped the register for the universal asynchronous receiver transmitter UART to one of the five serial ports to signal the occurrence of a system event. For this purpose and for any kind of system events, we can use each one of the free I/O pin of the B ports of the ARM9 CPU. Figure 2 shows an extract of the datasheet.

The lower layer including the electronic part is provided by an Atmel ATmega32 [7] microcontroller to drive the two DC motors by executing a linear algorithm for speed control or via odometry. This microcontroller unit is also intended to convert and collect analogy data from the infra red sensors. The lower circuit is connected to the RS-232 port of the ARM9 board which is enabled to switch the running

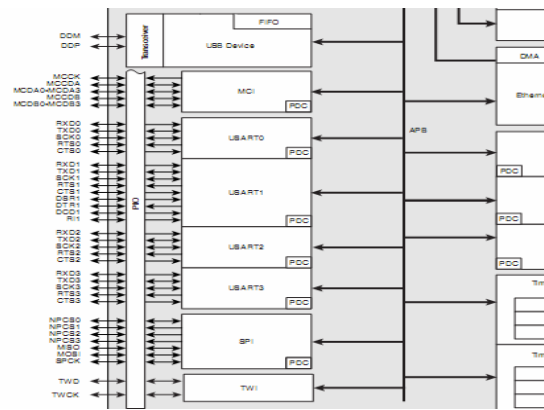


Figure 1. Extract of the block diagram - ARM9 CPU

mode of the microcontroller. This communication channel is also used to request and get data from the IR sensors.

3 Measurement infrastructure

Our measurements are based on the hardware methodology to avoid affectation of the system load like in [8]. The performance of the whole robot system is influenced by the response time of the lower electronic layer in various running mode with 5 difference states (motor off, motor on with velocity/position control, sensors on/off). The state of the microcontroller is thus similar to the system load of a common embedded device with an operating system and its scheduler. Figure 3 and figure 4 on the next side illustrates the infrastructure to monitor and measure response times for the communication using a dedicated ARM7 [9] microcontroller probe.

3.1 Probe Unit

The probe unit was designed and built considering an easy way, cost efficient, fast enough on board possibility to measure and record communication delays without additional system load like in software monitoring. For that reason and for rapid development, we selected an NXP LPC2138 [10] which is using an ARM7tdmi core. The probe unit monitors the communication between the lower layer with the ATmega32 with a 14.7456 MHz clock and the upper layer with the ARM9 CPU, whereby the I/O pins are clocked with about 60MHz. Therefore the measurement device has to run at least 60 MHz.

The crystal oscillator of the probe unit provides a frequency of 14.7456 MHz. Furthermore an intern PLL of the ARM7 microcontroller quadruplicates the frequency to 58.9824 MHz which is necessary for the monitoring process. The microcontroller has two timers that allow performing two measures simultaneously.

The device provides 32kB of SRAM to cache 2048 measure cycles and is connected to an RS-232 port of the ARM9 board to capture the measured data. Each measure cycle allows recording of three values of time of any event by setting a bit in the memory.

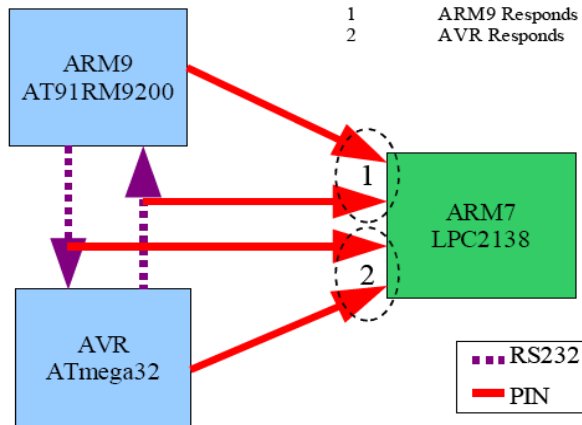


Fig. 3 Measuring the response time of the communication channel

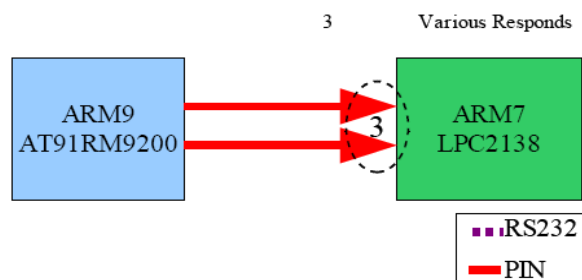


Fig. 4 Measuring delay of various applications

The bit is set by one programming line and the extra load for the measurement conditions does not carry authority. The probe unit is fully integrated in the design of the ARM9 board and the robot in our test bed. Redesigning the probe board also allows connecting the unit to a normal personal computer off the shelf. For this purpose one free serial port and two free output pins are required. Alternatively it is also possible to use the output pins of the parallel port if there are no more output pins left.

All in all, the microcontroller based probe unit is capable to measure nearly everything with a sampling frequency of almost 60 MHz. For each measure cycle 3 timestamps can be saved and a total of 2048 measurement values can be cached on board. The production coast are below 20US\$ and the complete unit is an open source platform.

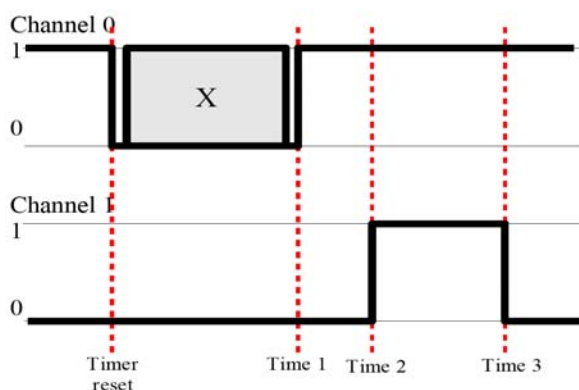


Fig. 5 Signal course of the measurement

3.2 Monitoring and Measurement

Principally the unit was designed to monitor the RS 232 communication between the upper and lower layer and the delay for the response time. Furthermore it is also possible to measure the throughput or the delays e.g. in the TCP/IP stack or any other application. Figure 3 and figure 4 illustrates the two possible methods of the measurement process.

Each measurement monitors the signal level of two pins. The start condition of the first pin (channel 0) has a high level and the second pin (channel 1) has a low level. As soon as a falling edge is detected on channel 0 the internal timer of the microcontroller is reset and started. The last rising edge on channel 0 is loading the current time stamp of the timer into a register.

Therefore channel 0 is able to measure the transmission delay of e.g. a serial string over an UART or something different. Only the first falling and the last rising edge are to be considered as the initial signal for the recording.

The rising edge on channel 1 indicates to load the current time stamp of the internal timer into another register of the microcontroller. The next falling edge on channel 1 again indicates to load the actual time stamp into a third register and immediately stops the timer. After all recording the time stamps Time 1, Time 2, Time 3 can be saved into the SRAM. Figure 5 illustrates the signal course of the measurement.

It is also possible in the monitoring process that Time 2 event occurs earlier then the Time 1 event. But Time 3 always stops the measurement and is the last monitored edge change. The duration of the recording depends on the resolution of the timer speed. At the maximum speed the timer is increased every 17 ns. In this mode the longest possible period for the measurement for Time 1 is about 9 seconds from timer reset; the measurement duration for Time 2 and Time 3 is about 72 seconds. All measurement periods can be enlarged, by decreasing the speed of the timer.

3.3 Data Storage

All monitored data is cached in an array in the SRAM of the microcontroller. The data structure for the recording is designed as the following struct variable.

```
struct MEASURE_Times
{
    unsigned int mode:3;
    unsigned int t0:29;
    unsigned int t1:32;
    unsigned int t2:32;
};
```

The above variable allows saving three time stamps and the mode of the recording (response of ARM9, AVR or various responses). After the storage space of the SRAM is full, the data in the cache can be depleted by ARM9 system of the upper layer. The data transmission of the probe unit to the ARM9 is similar to the FTP protocol and is implemented as a library. The control flow is human readable. In case of data transmission the protocol is entering the binary mode. The probe unit can also be reprogrammed for changes or extension of the functionality. For this purpose the ARM7 probe unit is provided with an on board boot loader, which enables the ARM9 system to flash the microcontroller probe directly with a new firmware without any additional hardware.

4 Example Measurements

We first monitored and measured the delay and the execution time of the AVR under different loads for external requests. Our performance evaluation for the system focuses on the delay between the end of the command string transition and the start of the execution of a valid command (Time 2 - Time 1). Execution time defines the time between start of the execution of a valid command string and the end of execution (Timer 3 - Timer 2).

The overall delay for the system response $d_{response}$ is according to equation in [5] composed of the hardware propagation delay for an incoming command signal d_{hwi} , the interrupt latency d_{lat} , a delay to process or execute the command $d_{execution}$, a delay to generate the response $d_{generate}$ and the hardware propagation delay for the outgoing response signal d_{hwo} .

Equation 1 characterizes the overall delay, where the sum of $d_{execution}$ and $d_{generate}$ and is also given as d_{ISR} as the delay for the interrupt service routine. While the other delays remain more or less constant and can be compensated for, d_{lat} depends on the system state at the time of the command reception.

$$d_{response} = d_{hwi} + d_{lat} + d_{ISR} + d_{hwo} \quad (1)$$

The duration for the first test run differs for various running mode of the AVR system. The microcontroller system enables accessing the actors in 3 different states (motor off, motor on with velocity/position control) and the sensors in 2 states (sensors on/off). Depending on the running mode of the microcontroller, the execution time for requests from outside of the AVR system should vary and in the full functional mode (e.g. motor on with position control and sensors on) the time should lead to an extended execution time compared with an idle state of the microcontroller.

4.1 Measurement and Monitoring

We started our test run using the probe unit and an oscilloscope to simultaneously monitor the



Fig. 6 Screenshot of oscilloscope - delay for execution 1.96ms

communication. Figure 6 display the screenshot of the oscilloscope with the upper signal course as a request UART command from the upper ARM9 layer and the lower signal course as the response of the AVR microcontroller. Although the system was idle we monitored fluctuations for the response time. Accounting the oscilloscopes recordings, we observed response times in the range of about 4.6 – 5.6 ms when the AVR system is idle.

In the second test run, we monitored the reverse direction and measured the delay and execution time of the ARM9 Linux system under various system loads. The variation are different CPU load (e.g. by compiling a kernel) and interrupt requests (e.g. by generating traffic over the USB port). To measure and visualize the fluctuation of the delay, we used the cumulative sampling function of the oscilloscope. In this case, the sampling is recorded continuously and additionally overdrawn over the previous signal course. The frequencies for the values are now represented by coloured pixels, in which a red colour symbolizes a higher and the violet colours a lower appearance of the recorded value. Figure 7 shows the example of the cumulative sample for the load state of the ARM9 system.

4.2 Statistical Evaluation

To evaluate the system performance based on statistical data, we took a closer look to the quartiles of the execution times. We used our lab construction



Fig. 7 Cumulative recording for the response time – ARM:load

to run data collection for 40,000 request/responses for different load states of the ARM9 and AVR system. The test duration lasts about 10 minutes for each direction and each system state (load or idle).

Table 1 gives a brief overview of the collected data of the execution time and the quartile distance. The quartile distance QD is defined as the difference of the first and third quartile. Normally this distance should cover 50% of all values and is used as the mass of the distribution.

This mass is usually robust against runaways and is usable for statistical evaluations. All data were recorded using the microcontroller probe for different system states of the AVR microcontroller (μC) system for the actors and different system states of the ARM9 (A9).

Comparing the quartiles of the AVR and ARM9 system it is remarkable, that the 25% quartile of the ARM9 system in the idle state is reached at 1730.432 μs in comparison to the load state, where the 25% quartile is reached at 1365.289 μs . This behaviour can be justified with the scheduler of the Linux operating system, whereby the scheduler shifts the mass centre of the response times to higher value and strews the results to a wider range in the load state (Figure 9).

Table 1: Quartiles of execution time (in μs)

Quartiles		25%	50%	75%	100%	Mean	QD
AVR μC	Idle	39.419	40.571	215.895	1229.096	214.928	176.476
	Load	773.468	1220.042	1665.683	2488.539	1218.634	892.215
ARM9	Idle	1730.432	1816.220	1839.125	9682.905	1688.620	108.693
	Load	1365.289	2042.932	3127.865	20122.562	3011.153	1762.576

Figure 8 on the right side shows the histogram plot for the idle and load state of the AVR microcontroller. The upper plot displays the idle state whereby the AVR system still has a minimum load because of the permanently running odometry algorithm. The scheduler of the AVR has no further tasks to schedule so the mean is located at 214.98 μs (Table 1).

Increasing the load of the AVR system significantly changes the shape of the histogram for the response time. When the AVR is running in access mode to the sensors and actors at the same time, the scheduler has to handle higher priority tasks before responding to the request. Sensor values, UART commands (receiving, sending) are declared as non priority tasks. Therefore the response time of the AVR system in the load state is spread over the bandwidth (lower part of figure 8) and can rise up to about 2500 μs which should be considered for the system design of the robots' electronic layer. The histogram plot for the ARM9 system indicates also a typical response when the system is in idle state (upper part of figure 9). In analogy to the AVR system, the ARM9 system is may in idle state, but there are still running system tasks of the Linux operating system which leads to a mean of 1688.620 μs for the response time and to a very low QD of 108.693 μs .

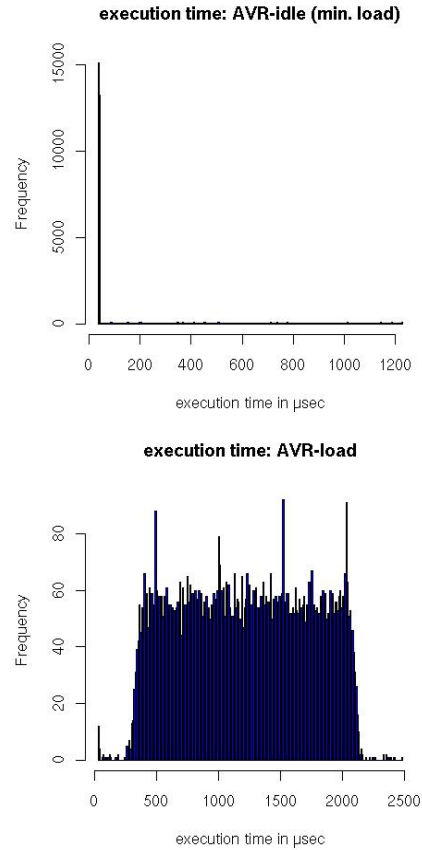


Fig. 8 Histograms for the execution time of the AVR μC

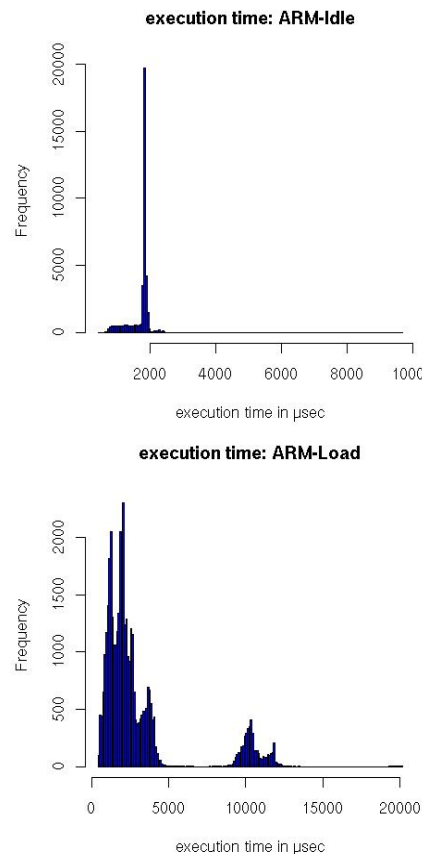


Fig. 9 Histograms for the execution time of the ARM9 CPU

The lower part of figure 9 shows the histogram plot for the load state of the ARM9 system. In this state a while loop was executed to print a message on the standard output and simultaneously a ping flood from an external host was conducted. The ping flood usually overwhelms the receiver with ICMP echo request packets and causes a huge amount of interrupts. Both actions together increase the system load and therefore the response time for the serial port is scattered over the range from 0 up to 20,000 μ s. The mean for the system state for this scenario is negligible lower located than the 75% quartile at 3127.865 μ s.

A closer look to the histogram shows a buckle for the range from 5000 μ s to 9000 μ s. During the increase of the system load state, the CPU is allocated first by the scheduler to the higher prior tasks like the while loop or the interrupt service routine for the access to the Ethernet. Thereafter the lower priority tasks are ready to be served by the scheduler. In case of simultaneously execution request of the higher and the lower prior tasks, the lower priority task (e.g. evaluating of the library for response) is displaced by the scheduler. The CPU is reallocated for the lower prior task, after the allocation of the CPU for the higher priority task is released. This is illustrated by the hump at the range from 9,000 μ s to 12,000 μ s. The screenshot of the oscilloscope in figure 6 also verifies the histogram plot for the load state. In figure 5 the blue coloured area for the response indicates the two hump on the right part of the histogram plot.

5 Conclusion and Future Work

We presented a possibility to monitor and evaluate the system performance of distributed embedded mobile systems. For this purpose we introduced a cost efficient, fast enough, on board microcontroller based small circuit for less than 20US\$. Based on the hardware methodology and using our circuit, we are able to monitor e.g. the serial communication between two systems. Our solution allows to record four different events using the on board timers of the microcontroller. The internal SRAM of the microcontroller enables to record 32kB to cache 2048 measurement cycles. We realised multiple test runs in different systems states and collected data for 40,000 test cycles for statistically analysis. The histogram plots of the data clarified the impact of system load to the response time and are especially to be considered for the design of embedded systems. Using the modified ARM9-board, we are also able to access to all data via internet and an SSH connection from a host computer.

Although the probe unit allows a simply way to monitor and record events, it takes multiple runs to gather enough data for statistically analysis. Each time, the storage space of the microcontroller is exceeded, all data has to be transferred to the ARM9 system before the next measurement cycle can run.

The data transmission is done via the serial bus which is a very time consuming part of the progress. To avoid the disadvantage in the system design, it is considerable to modify the 24kB of SRAM from a cache to a ring buffer system and install a SD card on the unit as data storage. Thus it is possible to increase the storage space up to 1GByte for extended measurements. An additional idea is to implement a file system (FAT32) for the SD card access to allow for reading the data from any computer rather than transmitting data over slow RS-232 serial port.

The probe unit also allows further experiments e.g. to measure the time period of internal programs. As an example, it is possible to measure the time it takes to generate a timestamp to estimate the overhead of software based instrumentation of a system. In this case, Time 1 will be the consumed time for generating the time stamp according to the explanation of figure 4. The difference of Time 2 – Time 1 represents the time consumption for the execution of the main application, and Time 3 – Time 2 the time period to generate the time stamp for the end of the measurement. Another scenario for further experiments could be the local scan of distributed robots to complete an environment map. For this purpose, at first each robot has to collect data of the IR sensors for a long scan period and store it on its local SD card. As soon as another robot is in range, each robot can interchange the piece of map to assemble the entire map. Further more, each data of the robots also can be accessed trough a SSH connection.

6 References

- [1] Kemal Köker, Richard Membarth, Reinhard German, "Performance analyses of embedded real-time operating systems using high-precision counters", Proceedings of The 3rd International Conference on Autonomous Robots and Agents (ICARA), New Zealand, pp 485-490, (2006)
- [2] RoboCup Small Size League, <http://www.roboocup.org/02.html>, visited on 14/01/2007
- [3] Ascending Technologies, <http://www.asctec.de/>, visited 01/06/2007
- [4] ARM920 Processor family <http://www.arm.com>, visited 15/01/2007
- [5] Hielscher, Kai-Steffen Jens ; German, Reinhard, "A Low-Cost Infrastructure for High Precision High Volume Performance Measurements of Web Clusters", Proc. 13th Conf. on Computer Performance Evaluations, Modelling Techniques and Tools (TOOLS 2003 Urbana, IL, USA September 2-5, 2003)
- [6] Gentoo Linux, special flavour for automatically optimizing and customizing of Linux, <http://www.gentoo-linux.org>, visited 13/01/2007
- [7] ATMEL ATmega32, AVR 8-Bit RISC Microcontroller, http://www.atmel.com/dyn/products/product_card.asp?part_id=2014, visited 15/01/2007
- [8] Thomas Elste, „Untersuchungen zum Aufbau eines echtzeitfähigen Embedded-Linux-Moduls“, TU Ilmenau, Institut für Theoretische und Technische Informatik, Diplomarbeit, 2005
- [9] ARM720T, ARM7EJ-S and ARM7TDMI Processor family, <http://www.arm.com/products/CPU/families/ARM7Family.html>, visited 17/01/2007
- [10] NXP LPC2138, Single-chip 16/32-bit microcontroller, <http://www.nxp.com/pip/LPC2132FBD64.html>, visited 15/01/2007