Develepment of Real Time Application Platform for Linux Kernel with Preempt-RT

V.Deepti
Dept. of ECM
K L UNIVERSITY
Guntur (Dt), India

K.Mahender
Scientist
Defence Research & Development
Organisation (DRDO), India

N.Venkatram
Professor
Dept. of ECM
K L UNIVERSITY, India

ABSTRACT
The On board computer (OBC) of an aerospace vehicle carries out control, guidance and navigation operations during the flight. It consists of a processor and IO board to execute the real time embedded software and to read input data from other subsystems & send commands to control system. In the specific system, the Power-PC 7410 based processor board is used and the IO board supports four UART, two Ethernet, four Arinc channels, two MIL-1553 nodes eight ADC's, eight DAC channels. It consists of 128MB SD-RAM, 8MB boot flash, and 64MB user flash. The OBC has to execute the control and guidance software which is bounded by hard real-time constraints. To achieve the hard real time constraints of the OBC, the Linux kernel is patched with the PREEMPT-RT and ported on to it. The customization of real time Linux kernel for PowerPC-7410 based hardware, and application development on the same platform using kernel primitives are presented in this paper. The software requires an extensive use of the kernel primitives like THREADS, TIMERS, SEMAPHORES, MUTEXES, SIGNALS, and PIPES etc. Application modules demonstrating the apt usage of the kernel primitives for real time application development have been created. These application modules will be used in the development of the Real-time-embedded flight software for the onboard computer of an aerospace vehicle.

Keywords
PREEMPT-RT, On Board Computer, MPC-7410, threads, semaphores, mutexes, pipes.

1. INTRODUCTION
The kernel is an intermediate layer between the hardware and the software. Its purpose is to pass application requests to the hardware and to act as a low-level driver to address the devices and components of the system. Generally Linux kernel is non real-time system. These systems have no hard guarantees and are able to utilize optimization strategies conflicting to real-time requirements. To make the Linux kernel as real-time operating systems the existing Linux kernel is patched with the Preempt-RT. The PREEMPT-RT having the following features [1]:

a. System time is a managed resource. Timing resources are managed with the highest possible level of precision.
b. Guaranteed worst-case scheduling jitter. If a task needs to be happening within a certain deviation, it is guaranteed to occur.
c. Guaranteed maximum interrupt response time. As with scheduling latency, interrupts are guaranteed to be acknowledged and handled within a certain window.
d. No real-time event is ever missed. This is important. Under no circumstances will a scheduled task not be run on time, an interrupt be missed or any other event the real-time code is interested in.
e. System response is load-independent. Execution of real-time tasks is guaranteed to fall within the worst case value range, regardless of the system load factor.

Usage of a patched Linux kernel with real time utilities is required for achieving the timing constraints specified for the application. Most of the avionic subsystems of any aerospace vehicle are bounded by hard real-time constraints and this necessitates the usage of an RTOS in the process of application development. The growth of Linux kernel in embedded software development is incredibly high, because of its convenient and efficiency in development. The kernel is designed to maximize the CPU throughput, instead of achieving the real-time responsiveness. There are various ways to utilize the Linux kernel for real-time applications. The PREEMPT-RT is one of the way in adding real-time capability to the Linux kernel. Here, in this application, the preempt-rt patched Linux kernel is used to develop real-time application for the onboard computer of an aerospace vehicle. As part of this work, set of real-time applications are developed to demonstrate the usage of the kernel primitives in the preemp-rt patched kernel environment; which will be used in the development of the real-time embedded software for the OBC.

2. REAL TIME SYSTEMS
Real-time systems can be defined as those systems that respond to external events in a timely fashion [2]. Responding to external events includes recognizing when an event occurs, performing the required processing as a result of the event, and outputting the necessary results within a given time constraint. Real-time systems can be classified into two types. First one is hard real-time systems. A hard real-time system is a real-time system that must meet its deadlines with a near-zero degree of flexibility. The deadlines must be met, or catastrophes occur. The cost of such catastrophe is extremely high and can involve human lives.

The differences between hard real-time systems and soft real-time systems are the degree of tolerance of missed deadlines. For hard real-time systems, the level of tolerance for a missed deadline is extremely small or zero tolerance. The weapon systems, aerospace vehicle's control and guidance systems are examples for hard real-time systems. Second one is soft real-time systems. A soft real-time system is a real-time system that must meet its deadlines but with a degree of flexibility. In a soft real-time system, a missed deadline does not result in system failure, but costs can rise in proportion to the delay, depending on the application. For soft real-time systems, the level of tolerance is non-zero. DVD player is example of soft real-time system.

3. RT-PREEMPT PATCH
The standard Linux kernel only meets soft real-time requirements: it provides basic POSIX operations for user space time handling but has no guarantees for hard timing

deadlines. With Ingo Molnar's Real-time Preemption patch and Thomas Gleixner's generic clock event layer with high resolution support, the kernel gains hard real-time capabilities [3]. The RT-Preempt patch converts Linux into a fully preemptible kernel.

3.1 Purpose of Preempt-Rt

The reasons for the design of PREEMPT-RT can be understood by examining the working of the standard Linux kernel. The kernel uses scheduling algorithms and assigns priority to each task for providing good average performances or throughput. Thus the kernel has the ability to suspend any user level task, once that task has outrun the time slice allotted to it by the CPU. These scheduling algorithms along with device drivers, uninterruptible system calls, and the use of interrupt disabling and virtual memory operations are sources of unpredictability. A real-time kernel should be able to guarantee the timing requirements of the processes under it. The PREEMPT-RT kernel accomplishes real-time performances by removing such sources of unpredictability as discussed above. We can consider the PREEMPT-RT kernel as sitting between the standard Linux kernel and the hardware. The user can achieve correct timing for the processes by deciding on the scheduling algorithms, priorities, frequency of execution etc. The PREEMPT-RT kernel assigns lowest priority to the standard Linux kernel. Thus the user task will be executed in real-time.

The actual real-time performance is obtained by intercepting all hardware interrupts. Only for those interrupts that are related to the PREEMPT-RT, the appropriate interrupt service routine is run. All other interrupts are held and passed to the Linux kernel as software interrupts when the PREEMPT-RT kernel is idle and then the standard Linux kernel runs. The PREEMPT-RT executive is itself non preemptible. Real-time tasks are privileged, and they do not use virtual memory. Real-time tasks are written as special Linux modules that can be dynamically loaded into memory. The initialization code for real-time tasks initializes the real-time task structure and informs PREEMPT-RT kernel of its deadline, period, and release time constraints.

3.2 Differences between Preemptive Kernel and Non Preemptive Kernel

Asynchronous events are handled by Interrupt Service Routine (ISRs) [4]. An ISR can make a higher priority task ready to run, but the ISR always returns to the interrupted task. The new higher priority task will gain control of the CPU only when the current task gives up the CPU. Non pre-emptive kernel uses the non pre-emptive scheduling. Non-Pre-emptive scheduling is a process enters the state of running; the state of that process is not deleted from the scheduler until it finishes its service time. Advantages of non pre-emptive scheduling are simple and robust. Disadvantages of non pre-emptive scheduling are they are not very responsive and a higher priority task that has been made ready to run may have to wait a long time to run, because the current task must give up the CPU when it is ready. (Figure 1) shows execution of a task in a non pre-emptive kernel.

(1) A low priority task is executing but gets interrupted.
(2) If interrupts are enabled, the CPU vectors (i.e. jumps) to the Interrupt Service Routine (ISR).
(3) The ISR handles the event and makes a higher priority task ready-to-run.
(4) Upon completion of the ISR, a Return from Interrupt instruction is executed and the CPU returns to the interrupted task.
(5) The task code resumes at the instruction following the interrupted instruction.
(6) When the task code completes, it calls a service provided by the kernel to relinquish the CPU to another task.
(7) The new higher priority task then executes to handle the event signalled by the ISR.

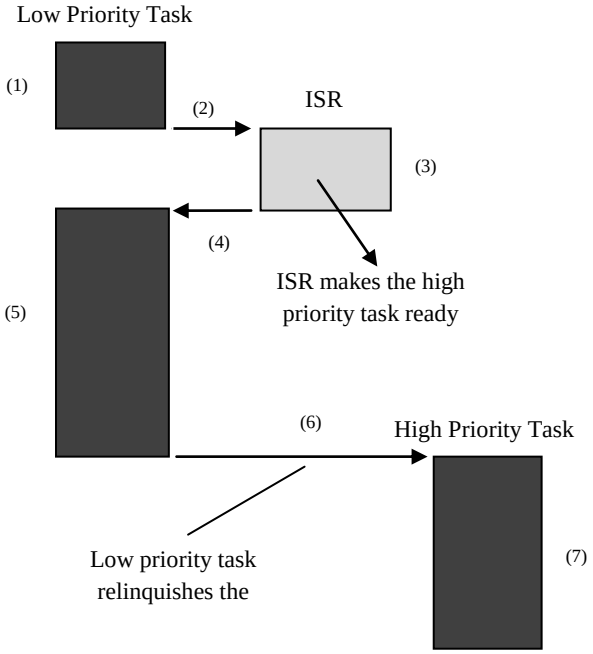

Fig 1: Execution of a task in a non pre-emptive kernel

In pre-emptive kernel the highest priority task ready to run is always given control of the CPU. If an ISR makes a higher priority task ready, when the ISR completes, the interrupted task is suspended and the new higher priority task is resumed. Preemptive kernel uses the preemptive scheduling. The pre-emptive scheduling is prioritized [9], [10]. The benefit of a pre-emptive kernel is the system is more responsive and the execution of a task is deterministic. A high-priority task gain control of the CPU instantly when it is ready (if no resource-locking is done). ISR might not return to the interrupted task it might return a high priority task which is ready. Concurrency among tasks exists. As a result, synchronization mechanisms [semaphores] must be adopted to prevent from corrupting shared resources. (Figure 2) shows execution of a task in a preemptive kernel.

(1) A low priority task is executing but interrupted.
(2) If interrupts are enabled, the CPU vectors (jumps) to the Interrupt Service Routine (ISR).
(3) The ISR handles the event and makes a higher priority task ready to run. Upon completion of the ISR, a service provided by the kernel is invoked.
(4) After completion of the ISR, high priority task is executed.
(5) This function knows that a more important task has been made ready to run, and thus, instead of returning to the interrupted task, the kernel performs a context switch and executes the code of the more important task. When the more important task is done, another function that the kernel provides is called to put the task to sleep waiting for the event to occur.

(6) After completion of high priority task the kernel goes to the low priority task
(7) The kernel then sees that a lower priority task needs to execute, and another context switch is done to resume execution of the interrupted task.

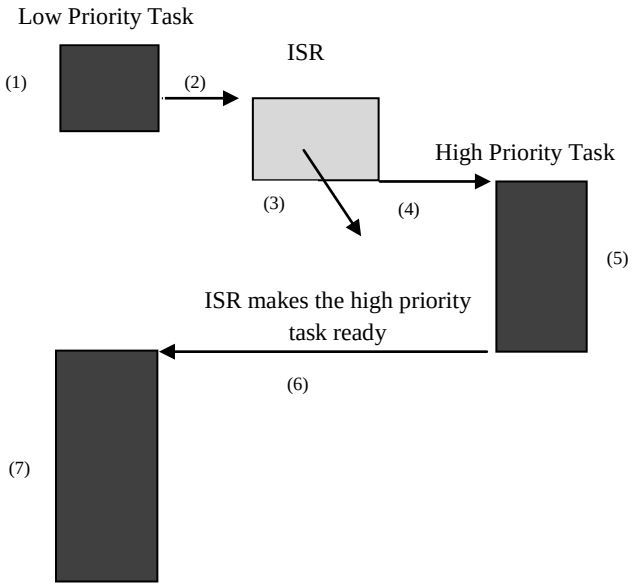

Fig 2: Execution of a task in a pre-emptive kernel

4. ON BOARD COMPUTER

The On board computer (OBC) of an aerospace vehicle carries out control, guidance and navigation operations during the flight. The unit has an 8 Mbytes boot flash on 8 Bit wide data bus and a 64Mbytes user flash on minimum 64-bit wide data bus. The IO structure of the OBC is shown in (Figure 3). It consists of 8 ports they are OJ1, OJ2, OJ3, OJ4, OJ5, OJ6, OJ7 and OJ8 [5]. OBC consists of a MPC 7410 processor, SDRAM, NVSRAM, Serial communication channels (RS-422, RS-232), Ethernet channel, MIL-STD-1553, ADC and DAC. A Programmable Watch Dog Timer (WDT) is implemented in Logic core with range of 0 to 5.6 seconds. The Watch Dog unit consists of a 16-bit timer. The counter is loaded by two 8-bit data. An interrupt is generated from this unit at the end of the count operation. The system clock is used as a clock input to the watchdog timer with a pre-scalar of 10 KHz. Hence the maximum time for which the watch dog timer can be programmed using the system clock of 10 KHz is 5.6 seconds.

Generally Power supply is connected to the OJ1 port; here 28v of power supply is given to the OBC. OJ2 and OJ3 are connected to the input output (IO) interface socket and pin type respectively. 1553 channel 1 BNC connector, 1553 channel 2 BNC connector, 1553 channel 1's redundant BNC connector, and 1553 channel 2's redundant BNC connector are connected to the ports OJ4, OJ5, OJ6 and OJ7 respectively. 1553 is a bus it is used to transmit and receive the data. Sometimes the signal strength is low at that time we lose the data. To prevent the loss of data 1553 redundant BNC connector is used to transmit and receive the data. Ethernet channel is connected to the OJ8 port.

The MPC7410 is a PowerPC reduced instruction set computing (RISC) microprocessor [6]. The 7410 processor operates at the core frequency of 400MHz with ALTIVEC support and 2MB L2 cache operating at 200MHz. The Unit has an onboard 128MB SDRAM with ECC provision. The SDRAM is a high-speed CMOS, dynamic random access memory using 5 chips containing 268,435,456 bits. Each chip is internally configured as a quad-bank DRAM with a synchronous interface. Each of the chip's 67,108,864-bit banks is organized as 16 bits in 8192 rows by 512 columns. Read and write accesses to the SDRAM are burst oriented; accesses start at a selected location and continue for a programmed number of locations in a programmed sequence. Ethernet controller selection is based on IEEE 802.3 complaint 10/100 Base TX interfaces and temperature range., Enhanced 10/100 Mbps PCI Ethernet Controller with integrated PHY interface 82551IT, which converts PCI interface to serial differential format and then given to the Magnetics HX1188**.** These are the only Ethernet interface ICs available in the industrial grade. The Unit has three RS422 Asynchronous channels supporting up to 1Mbps and one RS232 channel both with isolated receivers.

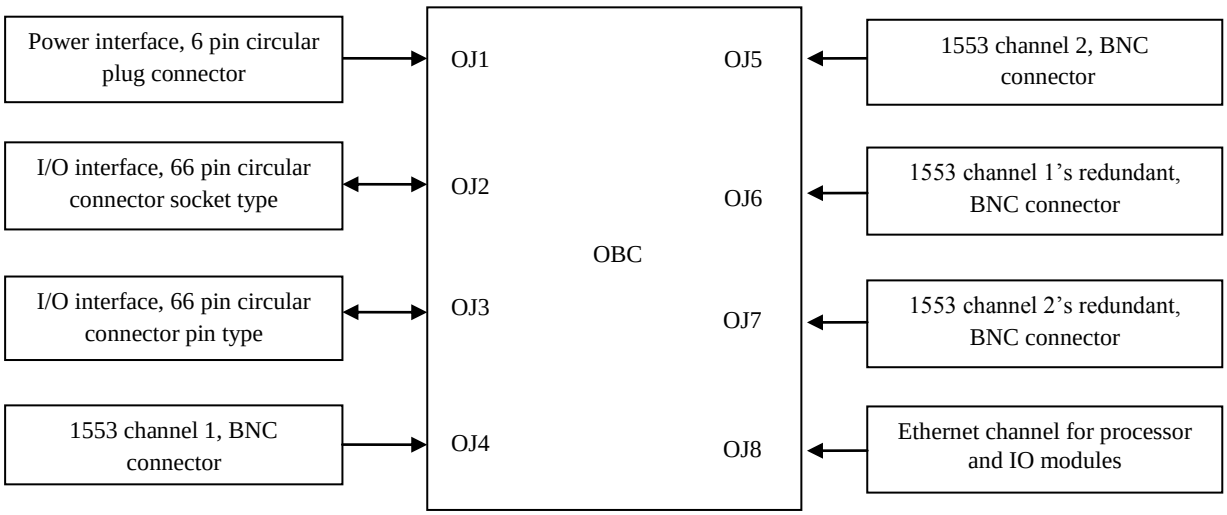

Fig 3: OBC-IO Structure

4.1 Installation steps on host system

1. Put a fresh copy of the Linux kernel in the /usr/src
   - cd /usr/src
   - tar –xvf linux-3.10.10.tar.bz2
   - cd linux-3.10.10
2. Patch the kernel with the PREEMPT-RT patch
   - bzcat ../patch-3.10.10-rt7.patch.bz2 | patch –p1
3. Now configure the Linux kernel
   - make menuconfig or make xconfig or make config

Then separate kernel configuration window is opened. In that we have to do some changes before installing the PREEMPT-RT. Go to processor type and features in that go to preemption model. In that select fully preemptible kernel (RT).

4. Compile the Linux kernel and modules
   - make
   - make modules
   - make modules_install
   - make install

4.2 Installation steps on OBC

Install Ubuntu 11.04 kernel version 3.10.10 in the host system Check whether the following services are installed:

- DHCP (Dynamic Host Configuration Protocol)
- NFS (Network File System)
- TFTP (Trivial File Transfer Protocol)
- XINETD (Extended Internet/Network Deamon)

If the above services are not installed and configured, copy the rpm package from the location shown below for installation.

Insert Ubuntu 11.04 (CD) CD ROM

bash$ mount /mnt/cdrom
bash$ cd /mnt/cdrom/RPMS
bash$ cp tftp-server-0.39-1.i386.rpm /usr/tmp
bash$ cp dhcp-3.0.1-11.i386.rpm /usr/tmp

This will copy all the required rpm files into a temporary directory.

Installing the RPM

bash$ cd /usr/tmp
bash$ rpm -ivh tftp-server-0.39-1.i386.rpm
bash$ rpm -ivh dhcp-3.0.1-11.i386.rpm
bash$ rpm -ivh ckermit-8.0.209-9.i386.rpm

Starting services

bash$ service xinetd restart
bash$ service nfs restart
bash$ service dhcpd restart

4.2.1 ELDK Installation

The Embedded Linux Development Kit (ELDK) includes the GNU cross development tools, such as the compilers, binutils, gdb, etc., and a number of pre-built target tools and libraries necessary to provide required functionality for the target system. To edit the script file need to press the Insert key or "i" to start updating and press Esc+:wq for save and exit from the editor.

a. Step 1: Insert the CD-ROM containing PPC – ELDK Installation Software.
b. Step 2: Mount the CDROM using the following command bash$ mount /dev/cdrom /mnt/cdrom
c. Step 3: Create a new directory where the ELDK should be installed, say bash$ mkdir /eldk
d. Step 4: Run the installation utility included on the distribution to install into that specified directory bash$ /mnt/cdrom/install -d /eldk PPC_74xx. If the ELDK installation is successful; it will shows an success message Done.
e. Step 5: After the installation utility completes, edit the following files.
   vi /etc/profile include the following two lines at the end of the script CROSS_COMPILE=PPC_74xx-
   PATH=$PATH:/eldk/usr/bin:/eldk/bin
   vi /root/.bash_profile include the following two lines at the end
   export CROSS_COMPILE=PPC_74xx-
   PATH=$PATH:/eldk/usr/bin:/eldk/bin

Note: The above command assumes that ELDK is installed in the location /eldk

f. Step 6: Create the device nodes for an MPC74xx based system, use the following commands
   bash$ cd /eldk/PPC_74xx/dev
   bash$ /mnt/cdrom /ELDK_MAKEDEV

4.3 Installing U-Boot on PPC7410

First of all, it is required to connect to the correct serial port to the target board, (typically ST40 Linux's /dev/ttyAS0) from a terminal emulator (running on a host system) with the following communications parameters: baud=115200, data=8, parity=none, Flow Control=none. In another window on the host system, then use sh4-Linux-gdb to download and run U-Boot on the target system, over the JTAG debug link. For example, for executing on a MPC74XX board (MPC7410), then the following would be appropriate.

host% sh4-Linux-gdb
Output: GNU gdb STMicroelectronics/Linux Base 6.5-33

GDB is free software, covered by the GNU General Public License, and you are welcome to change it and/or distribute copies of it under certain conditions.

This GDB was configured as "--host=i686-pc-Linux-gnu --target=sh4-Linux".

5. TESTING OF THE KERNEL

Latency is the delay between event occur and the response time. The term latency, when used in the context of the RT Kernel, is the time interval between the occurrence of an event and the time when that event is "handled" (typically "handled" means running some thread as a result of the event). "Cyclictest" tool is used to measure the latency [7]. Cyclictest measures the amount of time that passes between when a timer expires and when the thread which set the timer actually runs. It does this by taking a time snapshot just prior to waiting for a specific time interval (t1), then taking another time snapshot after the timer finishes (t2), then comparing the theoretical wakeup time with the actual wakeup time (t2 -(t1 + sleep_time)). This value is the latency for that timer wakeup.

The execution of Cyclictest can be divided into three phases [8]. During the initialization phase, the program creates a configurable number of threads (according to the specified parameters), which are then admitted as real-time tasks. The processor affinity mask is also set, which enables migration to be restricted. After that, each thread starts a periodic (i.e., cyclic) execution phase, during which Cyclictest executes the main measurement loop. Iteration (i.e., one test cycle) starts when the thread's associated one-shot timer expires. Once the thread resumes, a sample of scheduling latency is recorded as the difference between the current time and the instant when the timer should have fired. The timer is then rearmed to start

a new iteration and the thread suspends. After a configurable duration, the thread is demoted to best effort status and exits, and the recorded scheduling latency samples are written to disk. Here is output from a typical Cyclictest run:

To compile the Cyclictest the command is

```
sudo apt-get install build-essential libnuma-dev
make
```

Run tests

```
sudo ./cyclictest -a -t -n -p99
```

On a non-real-time system,

```
T: 0(4398) P: 99 I:1000 C: 96279 Min:2  Avg:12  Max: 23587
T: 1(4399) P: 99 I:1500 C: 64186 Min 2  Avg:8   Max:    33
T: 2(4400) P: 99 I:2000 C: 48139 Min:2  Avg:9   Max:   215
T: 3(4401) P:99  I:2500 C:38511  Min:2  Avg:11  Max: 22824
```

The rightmost column contains the most important result, i.e. the worst-case latency of 23.587 milliseconds. On a real-time-enabled system, the result may look like

```
T: 0 (1779)  P:99  I:1000 C: 9480  Min:3  Avg: 11  Max:   35
T: 1 (1780)  P:99  I:1500 C: 6320  Min:3  Avg: 11  Max:   39
T: 2 (1781)  P:99  I:2000 C: 4740  Min:3  Avg: 11  Max:   25
T: 3 (1782 ) P:99  I:2500 C: 3792  Min:3  Avg: 12  Max:   26
```

And, thus, indicate an apparent short-term worst-case latency of 35 microseconds. The execution of Cyclictest can be divided into three phases. During the initialization phase, the program creates a configurable number of threads (according to the specified parameters), which are then admitted as real-time tasks. The processor affinity mask is also set, which enables migration to be restricted. After that, each thread starts a periodic (i.e., cyclic) execution phase, during which Cyclictest executes the main measurement loop. Iteration (i.e., one test cycle) starts when the thread's associated one-shot timer expires. Once the thread resumes, a sample of scheduling latency is recorded as the difference between the current time and the instant when the timer should have fired. The timer is then rearmed to start a new iteration and the thread suspends. After a configurable duration, the thread is demoted to best effort status and exits, and the recorded scheduling latency samples are written to disk

6. CONCLUSION

The Onboard computer of an aerospace vehicle is a real time embedded computer bounded by hard real time constraints. The delays in the software execution have a direct impact on the system performance. For an accurate execution of the control and guidance software on the OBC, the application has to be built on top of a real time embedded operating system. Even though the Linux kernel has become an excellent choice for embedded software development, it lacks real time functionality. It was designed to optimize the system throughput but not for achieving real time responsiveness. The Linux kernel can be used for real time application development, by incorporating the Preempt RT-Patch. The process of patching Preempt-RT with the Linux kernel has been demonstrated. The real time application development process for preempt-RT patched kernel is given. The test result of a sample real time application has been illustrated.

REFERENCES

[1] FSM labs "Real-time programming in RTLinux", December, 2002.

[2] Qingli and Caroline Yao, "Real-time concepts for Embedded Systems", pages 11-17, 2003.

[3] RT-PREEMPT HOWTO-RTwiki, https://rt.wiki.kernel.org/index.php/RT_PREEMPT_HOWTO.

[4] Wolfgang Mauerer "Professional Linux kernel architecture", pages 847-878, 2008.

[5] Compact versatile OBC, VOL.1, version 2, pages 55-66.

[6] MPC 7410 RISC Microprocessor hardware specifications, Free scale semiconductor, pages 2-7, 2007.

[7] Real-time Linux wiki. Cyclictest-RTwiki, https://rt.wiki.kernel.org/index.php/cyclictest.

[8] Felipe Cerqueira, B. Brandenburg, "A comparison of scheduling latency in Linux Preempt-RT, and LITUMS", pages 1-9, 2010.

[9] P. McKenney, A real-time preemption overview, 2005, http://lwn.net/Articles/146861/.

[10] J. Y. Leung and M.L Merrill, "A note on preemptive scheduling of periodic, real time tasks", Inform. Processing Lett., vol.11 no. 3, pp. 115-118, Nov. 1980Sannella