Procedia Computer Science

Volume 66, 2015, Pages 546–555

YSC 2015. 4th International Young Scientists Conference on Computational Science

Hardware-Software Complex Prototyping for the Pulse Power Supply Control System of Tokamak T-15

Pavel Anistratov, Yuriy Golobokov and Vadim Pavlov

National Research Tomsk Polytechnic University, Lenina Avenue 30, 634050 Tomsk, Russia
pavel903@gmail.com

Abstract

A new hardware-software complex for digital control of a pulse power supply system was developed for the tokamak T-15 upgrade. Special controller software provides implementation of flexible digital control of semiconductor converters. For most functions it is required that the software should keep control loop timing accurately. The complex software runs under the Linux operating system. To turn it into a real-time system open-source Xenomai framework was used. To test the framework applicability for the tasks of power sources control a research of a test system executing the basic functions of a real system was conducted. The required cycle was accomplished by separation of important tasks into real-time threads while using previously developed program code and libraries, which were already tested in the real system for non-real-time task without significant changes.

Keywords: real-time control, tokamak, power supply control, control system

1 Introduction

Provision of reliable electricity supply for tokamak is a complex technical problem even for relatively small systems. It is related to a large quantity of stationary and pulse loads. Significant power and energy reserves are transferred in each pulse to a tokamak electromagnetic system and then returned to the power network. Also it is essential to generate complex form pulses in numerous interconnected control coils. It requires consistent control of multichannel semiconducting power transforming system operating in a deep adjustment mode.

To solve this problem a hardware-software complex for digital control of Tokamak pulse power supply system was developed. The package was used to control the thyristor rectifier system with total power of 60 megawatts, loaded on the toroidal field coil of the tokamak T-10 (Kurchatov Institute, Moscow), during a power test experiment. Conditions being created during operation of powerful electricity converters form a number of requirements limiting the possibility of application in a control system of the industrial base. The need of flexible configuration of control system equipment and embedded software according to the features of the transformation complex structure; accurate synchronization between the distributed

 Selection and peer-review under responsibility of the Scientific Programme Committee of YSC 2015

doi:10.1016/j.procs.2015.11.062

complex of field devices and power supply network; standard control algorithms as well as custom one refer to such requirements.

1.1 Hardware Structure of the Complex

The developed complex consists of intelligent functional modules, built on the basis of modern digital electronics components. The simplified structure of the complex is shown in Figure 1. The main component forming the core of the complex is the CompactPCI automated system controller under the single-board computer control. Also special and general purpose modules are used in the system. For communication with power transformers and adjoining systems two interface converters are used: the control and synchronization multiplexer (CSMUX), the multipurpose interface adapter (MPIA). Both provides electric-optic signals conversion, the CSMUX also allows to control 2 or 4 output groups of synchronized rectifiers.

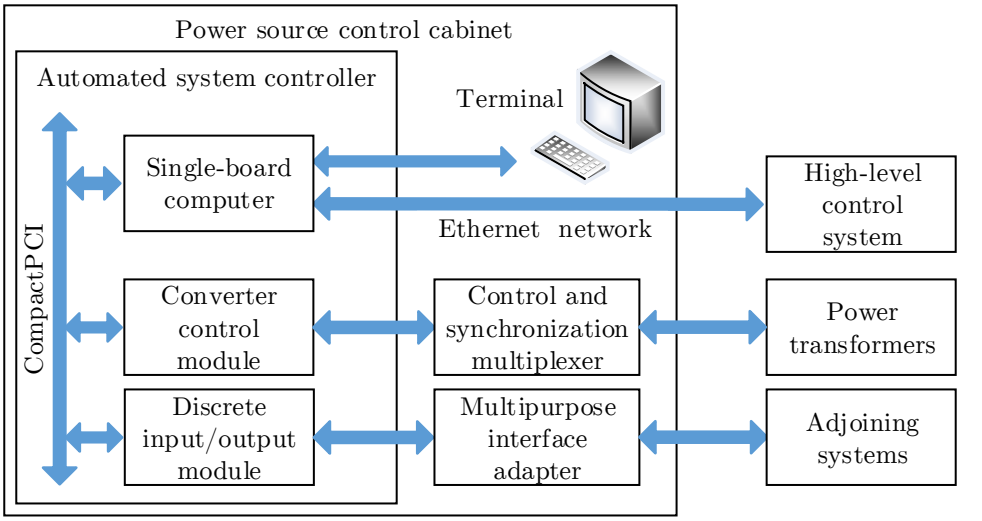

Figure 1: The complex structure

The converter control module (CCM) forms control time diagrams, synchronizes with power supply network and gets telemetry information. The discrete input/output module (DIOM) is an subsidiary module for interaction with adjoining systems. Tokamak T-10 control system uses the CCM and the DIOM based on a IC-102-DIO programmable discrete input/output module with different firmware. The module was developed by TomICS-Project company in PMC format and has 32 LVDS lines with flexible configuration based on Altera Cyclon 3 FPGA. Adlink cPCI-8602 PMC Slots Carrier Board is used to connect modules to CompactPCI bus. The TomICS IC-301-LS control and synchronization multiplexer has 24 channels and provides LVDS pulse signals group multiplexing and conversion to pulse optic signals by maximum time of 25 ns. The single-board computer cPCI-6615 has Intel Atom D525 1.8 GHz processor and 2 GB DDR3-800 memory.

For most functions, including currents and voltages regulation at the output of the converter, it is required for the complex software to bind to control loop accurately and keep time intervals, which is especially important for a PID-controller components calculation. The experimental test showed that without program system switching to a real-time mode its cycle could change in a wide range. Since the complex software already runs under Linux operating system the

open-source Xenomai framework was used to turn it into a real-time system. It allowed general-purpose system to perform real-time tasks.

The automated system controller is a CompactPCI crate intended for installation into electrotechnical 19″ cabinet. Special controller software provides implementation of flexible digital control of semiconductors and converters group with varying phasing, pulsing and binding to supply network and load. The complex has an ability to adapt to changes in the transformation complex structure, as well as changes of load parameters.

The complex carries out the following functions:

- Sets up device under programme on a digital interface;
- Controls parameters of load and protects electrotechnological equipment from emergency;
- Manages the power of electric energy converter scheme (thyristor rectifier, inverter voltage or other) for the purpose of current input/output into the load by a specified dependence;
- Regulates load parameters with voltage or current feedback loop;
- Operates autonomously for a given algorithm and operates under higher level control systems.

1.2 Test System Description

For simplification the test system was made on basis of a personal computer. Xenomai framework was used to provide real-time performance. It uses Adeos microkernel to provide a hardware abstraction level (HAL) between the hardware and an operating system. The microkernel is applied as a patch to the Linux kernel. Then it is necessary to compile the kernel and install it into the system in accordance with the official documentation [2]. Since real-time driver for the CCM and the DIOM are not developed yet, a parallel port of the PC was used instead as the faster and the simplest port to use.

Xenomai framework was installed on a computer with Intel 700 MHz Celeron processor and 512 MB of RAM with Debian 6.0.1 distributive with linux-3.18 kernel. Configuration with clearly weak characteristics is used to test the program in critical conditions, as well as provide work in existing systems with low performance.

1.3 Previous Results

Xenomai performance testing on such system was conducted earlier [1]. A signal generator was used to form impulses for the parallel port, which generated interrupts in the program. Interrupt handler configuration was made by Xenomai Native Skin [5]. For 1 kHz input signal frequency the mean deviation of registrated periods was 9.5 µs, the maximum deviation was 117 µs. In addition, deviation for 95 % of periods was less than 10 µs. For response time testing in severe conditions Stressful Application Test [6] program was used. The program generated a high load on the test computer subsystems, accessing to the hard drive and RAM, which gave full CPU load. This program was executed together with the user interrupt handling program. Under load the maximum deviation of the period was around 450 µs that was why the maximum reached input pulses frequency without interrupt loss was about 2 kHz.

2 Hardware-Software Prototype

In a new program, instead of Native Xenomai API, POSIX skin is used. It allows parallel execution of real-time tasks and non-real-time tasks, and also provides an information exchange mechanism between them. Real-Time Driver Model (RTDM) approach was used to implement interrupt handling, which presupposes the use of a special driver to control interrupts and allows synchronizing information with the user environment by ad hoc ioctl_rt() handler. This approach is also recommended by the user community [3].

To test the framework applicability for the tasks of power sources control a research of a test system executing the basic functions of a real system was conducted. The developed program implements a real-time thread for user interrupt handling and non-real-time threads to write information about events to a data base and a file log, as well to as provide information about the number of registrated interrupts over Modbus TCP protocol. For information exchange between real-time and non-real-time threads the Cross-Domain Datagram Protocol (XDDP) is used, which allows exchanging information between threads in a way that does not require the former to leave the real-time domain.

2.1 Software Implementation

The irqbench [4] test program is used as the basis for the developed program prototype. It is supplied together with Xenomai framework to test interrupt functioning.

The program implements RDTM approach by processing interrupts initially at a kernel layer with the possibility of further processing at kernel level, as well as at the user level. Thus the program consists of two modules. Irqbench.ko is responsible for interrupt handling at the kernel level. And irqloop program produces a kernel module configuration, user-level interrupt handling and execution of non-real-time threads, records statistical data in a text file for further analysis. The structure is shown in Figure 2.

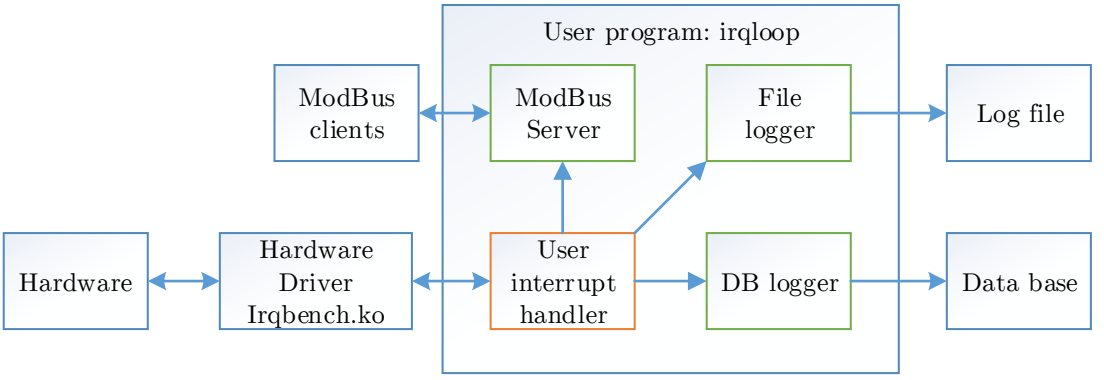

Figure 2: Software structure of the complex

Small changes were made in the driver module. A time variable (irqs_time) was added to a synchronization context of the driver to store monotonic time of an interrupt registration in nanoseconds which is returned to the user level task in response to the interrupt waiting command. Unused checks of the parallel port signals were disabled, because the signal generator was used instead of the second computer. The driver requires exclusive access to the parallel port; therefore for its work the system parport_pc module should be unloaded. The irqloop

program underwent significant changes to realize the user threads. Small changes were made to make compatible code for g++ compiler, since previously developed libraries for the user threads were written in C++, but the original code for irqloop was in C.

The program consist of four POSIX threads, one of which is for the real-time thread for the user interrupt handling, and three non-real-time threads perform information processing. The real-time thread sends to the driver the request interrupt command and expects an interrupt registration time (irqs_time) in response. When the response is received, the current time on the user lever is registrated in the statistic array for future delay calculation. After that the real-time thread transmits an interrupt counter to the non-real-time tasks. When a non-real-time thread receives a message, the current time is also registrated in the statistic array. The first non-real-time thread counts interrupts for the Modbus TCP server (the main function of ModbusTCPServer library is executed for testing period separately). The second one writes to a remote database an event message using the previously developed library (expcasdb library). The third thread writes information to a file log (expnewlog library).

Implementation of the threads with such functionality is caused by the need to verify the use of the libraries which are expected to be used in real system, but for now processing is simplified due to absence of real data and systems.

3 Testing

The series of tests where the signal generator was connected to the parallel port of the personal computer executing developed software were made. The time interval of the recorded interrupts was estimated by experimental data as 1000.1 µs. It was 0.01 % larger than the expected interval for the 1 kHz frequency, which testified to the difference in oscillator frequencies.

3.1 Real-Time Threads

The differences between experimental and estimated time registrations of interrupts were found using the experimental estimated interval. As the delay between the moment of an interrupt appearance and execution of the time reading function was not known, and as for analysis convenience, the time values were decreased by the differences mean value for the test (Figure 3). One standard deviation in the test was $\sigma = 1.28$ µs. The confidence interval for confidence probability $p = 0.95$ was ± 2.11 µs. The interval defined as the difference between the maximum and minimum values of deviations was 14.23 µs.

The user interrupt handler is executed with the real-time thread priority. The average delay between a driver interrupt registration and call of the user interrupt handler (it includes time for basic driver interrupt handling, the time to transfer information to the user the level and time to call the user interrupt handler) was 22.77 µs, standard deviation: $\sigma = 2.29$ µs, the confidence interval for confidence probability: $p = 0.95$ was ± 3.78 µs.

3.2 Non-Real-Time Threads

To characterize the information transmission between the real-time thread and non-real-time threads is more difficult as the last are planned together with the other tasks of the system, which sometimes can cause an increase in delays. The average delay of the data transfer for the interrupt counting thread was 124 µs with maximum of 395 µs.

For the data base thread the delay in most cases was as the amount of less than 0.5 ms, but there were periodic bursts to the value of 3 ms. It was caused by the fact that the recording

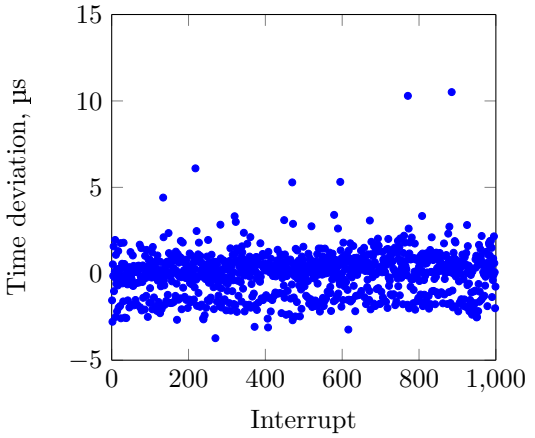

Figure 3: Deviation of average time delay from the moment of interrupt event and registration by the driver

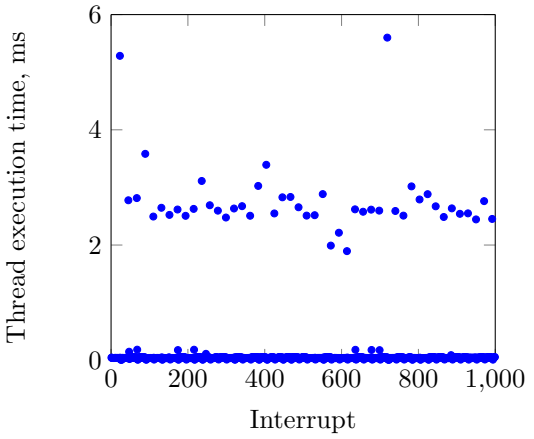

Figure 4: Database thread execution time

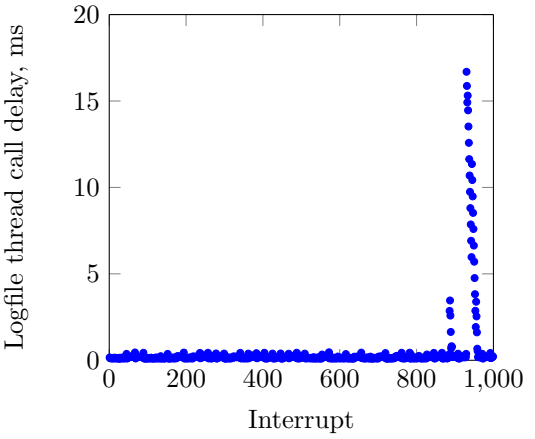

Figure 5: Logfile thread call

to the database was made by packages, so the run time varied from around 0.05 ms at writing to the memory case and 5 ms during the actual data transfer case (Figure 4). In other words, it can explain arising delays of the data base thread call. The file log thread also had delays (Figure 5). By the end of the experiment the transmission delay sharply increased to the maximum value of 17 ms, it might be caused by memory allocation features of the used library, since new memory was added on the fly. For the first 800 interrupts the average transmission delay was 0.16 ms.

3.3 Buffer Prediction

For data exchange between the threads buffers with defined size should be allocated in advance. They serve for a temporary data storage until they are read out by the corresponding non-real-time thread. As the size of buffers is finite and predefined, to avoid data loss it is required to

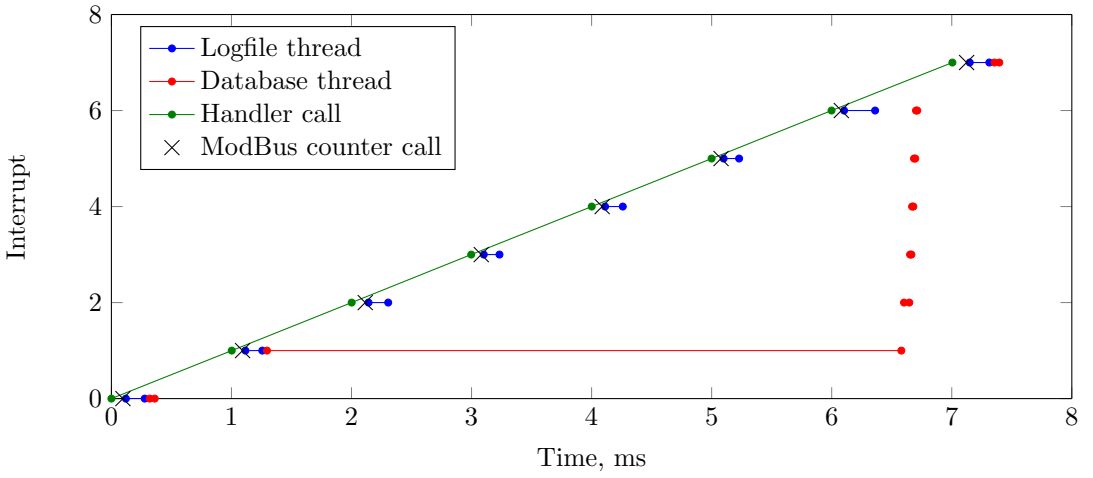

Figure 6: Time diagram of thread executions and calls

determine necessary amount of allocated memory in advance. On the basis of the conducted experiment an attempt to predict the required size of buffers was carried out. The graph shows time intervals of thread execution of the data base thread and log file thread. In the most cases, as it can be seen from the graph, the threads are executed in short time close to the point of an interrupt appearance, but at moments of actual writing to a remote database the bigger execution times (red rectangles) appear. Let us will examine a zoomed fragment close to the actual writing to a remote database (Figure 6).

In most cases data processing finishes until the next interrupt, but at times of actual writing to a remote database the delay occurs, which causes the buffer accumulation. Let us estimate the time required for data processing when the data base thread writes only to a local storage, without actual writing to a remote database. According to the statistic array, an interrupt handler call is followed by execution of the interrupt counting thread, then by the log files thread and by database thread in the last. Thus the average data processing time was estimated as an interval between the time of the user interrupt call and the database thread processing completion (without actual writing to a remote database). Figure 7 shows the estimated data processing time intervals for the whole experimental set. In most cases the processing takes less than half a millisecond, and visually the data can be divided into two groups by the value of 0.5 ms: 85 % of the intervals belong to the first group with an average value of 0.25 ms and the upper limit around 0.4 ms. An average value of the second group is 2 ms with the maximum of 5.7 ms. Thus the maximum number of information packages in the buffer query by Equation 1 was estimated.

$$d_{max} = \frac{5.7}{1.0 - 0.4} = 9.5 \approx 10 \tag{1}$$

Figure 8 a shows a queue in the buffer for database thread. In 88 % of the cases the queue consists just of only one element. The maximum queue in the experiment had 6 elements. Figure 8 b shows a queue in the buffer for log file thread. It had only one element for the major part of the experiment, but a queue up to 18 elements close to the end occurred. It relates to the increase of information transmission delays between real-time thread and file log thread that was shown earlier.

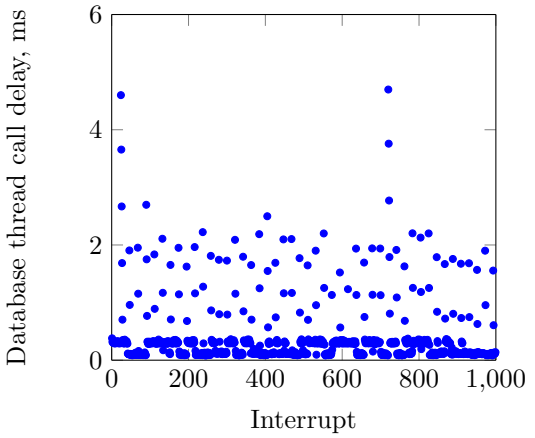

Figure 7: Execution time of full interrupt routing

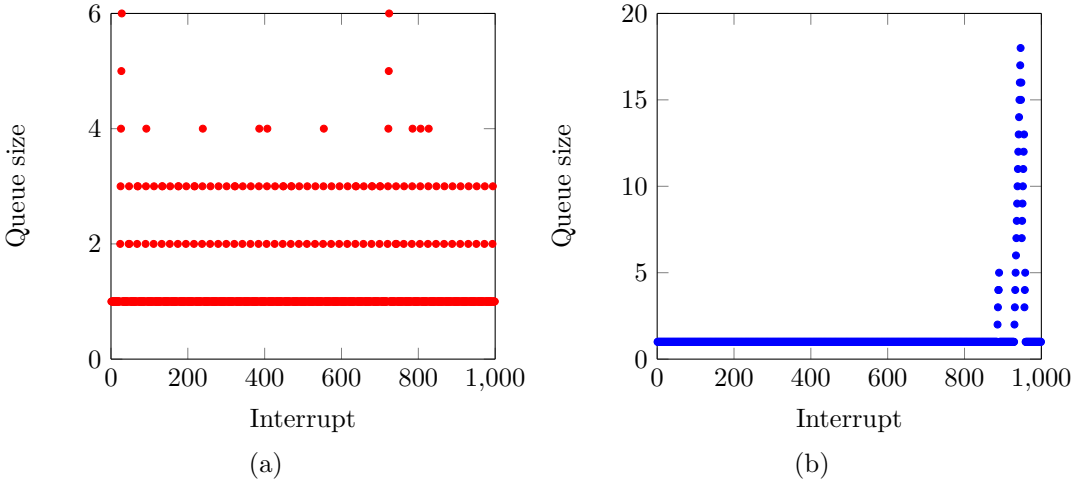

Figure 8: Buffer queues: (a) database buffer queue, (b) logfile buffer queue

3.4 The Series of 1 kHz Experiments

The 1 kHz frequency experiment was repeated several times to check the maximum buffer queue. In the second experiment the maximum queue of the database thread was seven, the value was close to the previous experiment. But unlike the previous experiment the maximum log file thread queue had 21 elements (Figure 9 a). On the time diagram of the experiment part (Figure 9 b) it can be seen that the large delays of log file thread execution occur, but it does not effect execution of other threads like the database thread or the interrupt counting thread. The third start of the experiment did not reveal significant deviations in maximum buffer queues, which had 4 elements for the database thread and 5 elements for the log file thread.

One more experiment was made also with 1 kHz frequency, but for one hundred thousand interrupts. The maximum queue of the database thread had 341 elements with local peaks

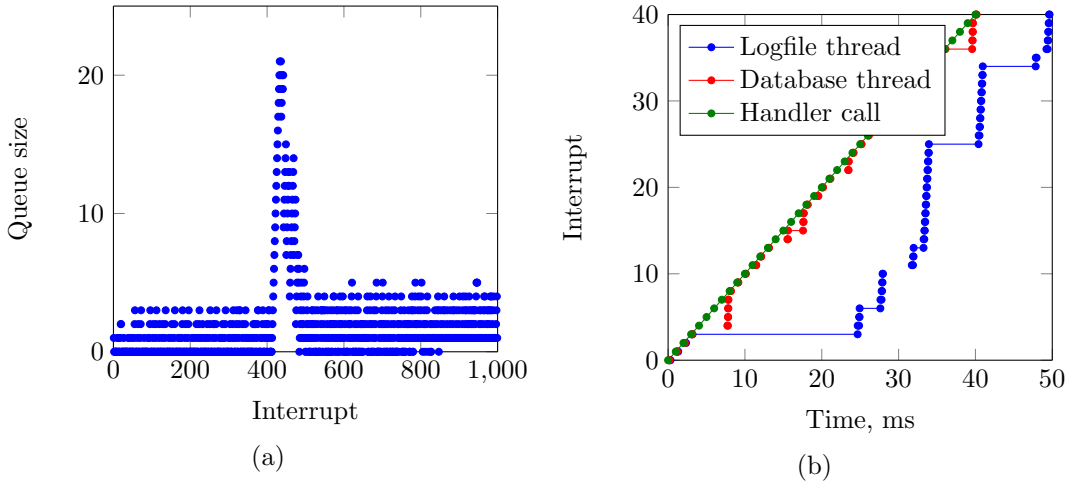

Figure 9: Second experiment: (a) shows logfile thread queue, (b) shows time diagram of thread executions and calls

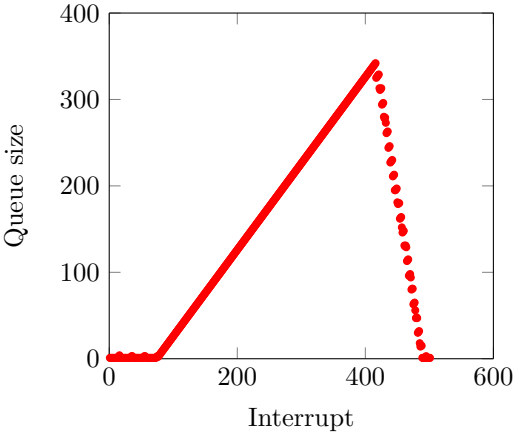

Figure 10: Third experiment: database buffer queue

up to 53 elements. On the magnified region it can be seen that for about 300 ms no database thread call occurs, which causes fast buffer queue grow (Figure 10). The similar situation is for the log file thread, where accumulation of the queue up to 665 elements is observed. It is also affected by no thread call for a long time, which causes accumulation of a queue in a buffer.

3.5 Results of Buffer Prediction

The emergence of the delays up to 0.7 ms makes it difficult to predict necessary size of the buffers for threads data exchange. Presumably the delays caused by other tasks of the operating system, elimination of such effects requires analysis and disabling of the operating system tasks that may cause a significant impact, or allocation of sufficient memory for buffers to store most data of the experiment.

Therefore on the basis of the testing results it can be concluded that the driver level task and the user interrupt handler are working as real-time tasks in predictable time and in the narrow time interval. Interrupts get registered at the kernel level at almost the same time, the confidence interval is just 2.11 µs, and information transfer to the user environment occurs within 23 ± 4 µs. During execution of normal flows sometimes delays occur, but during the whole experiment with 1 kHz frequency is carried out without losses.

4 Conclusion

Based on the test results the choice of such software architecture turned justified. The hardware-software complex was created for testing purposes in which the signal generator connects to the parallel port of the computer which runs the developed program under Linux/Xenomai system. The required control cycle for final system was 3.3 ms, but the prototype complex was tested even with requirement to the control cycle (1 ms). The control cycle in the experiment was maintained with the confidence interval of 2.11 µs. The mean time for interrupt information transmission to the user environment was 23 ± 4 µs. Sometimes information transfer from real-time thread to non-real-time threads was delayed because the latter were executed with the same priority as other system tasks. Consideration of this factor is important to allocate necessary amount of the memory for exchange buffers.

The results were considered successful because the required cycle was accomplished by separation of important tasks into real-time threads while using previously developed program code and libraries which were already tested in the real system for non-real-time task without significant changes. Implementation of a real-time driver for the programmable discrete input/output module (TomICS IC-102-DIO) to test the system with the project's hardware is planned in near future.

References

[1] Pavel Anistratov and Yuriy Golobokov. Analysis of possibilies of creating of a distributed control system based on real-time OC Linux/Xenomai. *XX International Conference of Students and Young Scientists: modern techniques and technology, National Research Tomsk Polytechnic University*, pages 139–140, 2014. `http://portal.tpu.ru/files/conferences/ctt/proceedings/2014/vol2_2014.pdf`.

[2] Gilles Chanteperdrix. Building debian packages. `http://xenomai.org/2014/06/building-debian-packages/`, last viewed July 2015, 2014.

[3] Philippe Gerum and Various Contributors. Xenomai-help ext. interrupt with posix skin from user space. `http://www.xenomai.org/pipermail/xenomai/2011-January/022523.html`, last viewed July 2015, 2011.

[4] Jan Kiszka. Irqbench(1) manual page. `http://www.xenomai.org/documentation/xenomai-head/html/irqbench/`, last viewed July 2015, 2014.

[5] Harco Kuppens. Real-time linux (xenomai) exercise 9: Interrupt service routines. `http://www.cs.ru.nl/lab/xenomai/exercises/ex09/Exercise-9.html`, last viewed July 2015, 2011.

[6] Nick Sandres and Various Contributors. Stressful application test. `https://code.google.com/p/stressapptest/`, last viewed July 2015, 2014.