ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 *2010. No. 8(104)*
ELEKTRONIKA IR ELEKTROTECHNIKA

AUTOMATION, ROBOTICS
T 125
AUTOMATIZAVIMAS, ROBOTECHNIKA

Real-time Acquisition of the Distributed Data by using an Intelligent System

N. C. Gaitan
Department of Computers, Faculty of Electrical Engineering and Computer Science,
„Ştefan Cel Mare" University of Suceava, Romania, str. Universitatii nr.13, e-mail: cristinag@eed.usv.ro

Introduction

This paper introduces the hardware and software architecture of an intelligent system, specific to real time acquisition of the distributed data.

The intelligent system is represented under the form of *an embedded* device, accomplished by means of a STR710FZ2 microcontroller and an RTX real time kernel.

A real time system signifies a system whose correctness does depend upon both a logical result of computing, and upon a moment when that result was produced. An equivalent definition is focusing over the behavior of the system, which might be analyzed outdoor. This fact might be described by means of the service concept: the service(s) provided by a system is (are) represented by its behavior noticed by the user(s) [1, 2], meaning by the specific environment. Basing upon the previous mention, a real time system service has to meet both the temporal and functional (logical) requirements; in this way, the system has to be seen as "erroneous". This fact has carried out the following definitions [3]:

1. The real time service signifies that service necessary to be provided in certain time ranges, decided by the environment.
2. A real time service represents a system, which provides at least one real time service. One might mention that this definition also emphasizes that different services (parts) of the same system can be a subject to various temporal necessities.

The complex applications need more computing resources, which means that they will be much easier to achieve if using some processors. The critical applications offer an increase for more rigorous necessities, which *might* be carried out only by solving adequately the mechanisms of tolerance to fault (error) [4, 5]. The tolerance to fault (error) needs the redundancy, which can be achieved by means of using additional processors.

A real time system has *severe time constraints*, when a time error might cause human, economic or ecological disasters. A real time system has *easy time constraints* when the time errors can be treated by a certain extension. A computerized real time system signifies that system whose behavior is established by the dynamics of an application. In this way, a real time application [6, 7] consists of two connected parts: a computerized system of real time controlling and the controlled process. Concerning the real time systems, the word task is the most used as unit, so as to represent the concurrent activities of the logical architecture [8, 9]. The physical parallelism of the hardware architecture and the logical parallelism of application requirements signify most often the basis for distributing an application towards competing tasks.

The hardware architecture of an intelligent master towards the acquisition of distributed data

The hardware architecture of an intelligent master has offered the necessary resources, so as to implement the real time application, also known as Intelligent Master. The microcontroller has to assure an increased computing efficiency [10], but of low price. For the time being, the ARM7 TDMI and Cortex M3 architectures offer enough resources for such applications. The estimated memory is of 128KB of flash memory and of minimum 64KB of SRAM, due to the high number of communication buffers.

In order to accomplish a history of inputs specific to objects dictionary and to set up the communication protocols and an acquisition cycle, the system was endowed with a SD or MMC non-volatile memory type.

Considering the time chart, a microcontroller with an embedded watch is chosen. For the PC communication, three possibilities were foreseen, such as:

- The communication with the serial port and RS232 line standard;
- The communication using USB port;
- The communication using an Ethernet connection of 10/100 Mb/s.

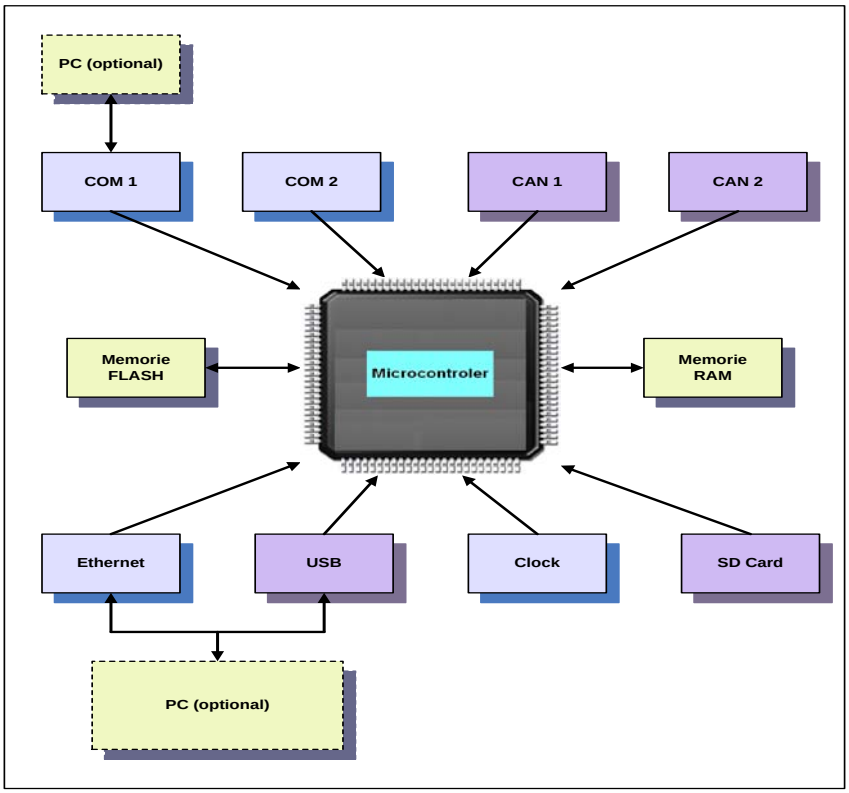

Fig. 1. The interface of digital systems towards the environment

In order to implement the protocols imposed by the defined requirements, the architecture was fitted with the following:

- Serial ports for ASCII, M-Bus and Modbus RTU/ASCII protocols and communication by radio modem (the RS232 and RS485 line standards will be used);
- Ethernet ports, which can ensure the Modbus TCP/IP protocol;
- CAN ports, which allow the implementation of one or more networks, based upon CANOpen protocol.

Taking into account the reliableness and wiring space aspects, it is preferred that resources should be integrated on the microcontroller's chip, as many as possible. This will represent an essential criterion for choosing the microcontroller. The challenge for real time standards consists of choosing between real time kernels, which are standardized by adopting the Unix standard interface, and the non-Unix real time kernels, modified in order to offer specific facilities to real time.

A set of application programming interfaces (API), which extend the Unix interface to real time, were proposed as Posix 1003.1b standard. These interfaces, which allow the portability of applications in real time requirements, are represented by:

- The functions of timer interface, so as to set up and read the internal timers of high resolution;
- The scheduling functions, which allow the taking over an setting up the scheduling parameters. Three politics are defined: SCHED_FIFO, meaning a preemptive scheduling, based upon priorities, SCHED_RR, meaning a preemptive scheduling, based upon quantum priorities (round-robin) and SCHED_OTHER, meaning a scheduling defined by implementation;
- The functions of files, which allow the creation and access to files with determinist performances;
- Efficient synchronization primitives, such as semaphores, and facilities to synchronous and asynchronous transmission of messages;
- Functions of asynchronous notification of events and signals in real time, placed in queues;
- Functions of locking a process memory and of facilitating the mapping of the shared memory;
- Efficient functions, which accomplish the synchronous and asynchronous I/O operations.

The real time operating system

A real time operating system should provide facilities, so as to accomplish three main essential requirements to real time applications[11][12][13]:

- Ensuring a response from the computers system;
- The promptitude of response, once this has been decided;
- The security of application code.

A real time operating system has to be able to take into consideration the periodical tasks, with periods and terms established, as well as the discontinuous tasks, of unknown occurrence data, but of fixed terms. These properties might be reached by an approach of levels, based upon the real time planning of tasks and of real time kernel.

The evaluation of a real time operating system has been mainly based upon the real time abilities, such as:

- The promptness of the answer given by the computer system;
- The predictability of execution times, specific to a call to the kernel;
- Disposing of scheduling politics;
- Providing the assistance, so as to debug the program in real time context, when the application runs in the field;
- Storing the performances for future case studies.

Two significant aspects will be developed:

- The promptness of an answer. The promptness of an answer specific to a real time kernel can be evaluated by two parameters, such as the delay of interruption and the delay of response. The delay of interruption signifies the delay occurred between the arrival of an event to the application and the moment when this event is stored in the computer's memory. The delay of response is that delay occurred between the arrival of an event to the application and the fastness by which the event is processed by that task.
- The predictability of the execution times, specific to a call to the kernel. A real time kernel includes a set of methods, in order to reduce the delays, meaning: the reentrance, the preemption, the scheduling of priorities and the succession of priority. As consequence, the execution time of each call to the kernel can be evaluated when this is executed for the task having the highest priority. This time represents the sum of the properly call and the delay given by the highest critical section of the kernel.

The real time kernel

RTX real time kernel has allowed a flexible scheduling of the system's resources, as the CPU and the memory, thus offering some communication means between the tasks[14–16].

The programs written for the RTX real time kernel use standard C builders, and they are not compiled by means of RealView Compilation Tools, provided by MDK-ARM Development Kit.

In addition to C language, one might declare very easily the functions of the tasks, and also the writing of real time programs, which only need the including of a special header file into the program and the connection with the RTX library.

RTX provides the functionality, as basic point of starting and stopping the competing tasks (processes). This also offers additional functions for the communications among processes. Some communication functions can be used, in order to synchronize the different tasks, of managing the common resources (as the peripherals and memory parts), and also of transmitting complete messages among tasks.

The basic functions might be used, so as to start the real time executive, to start and to stop the tasks, or to pass the control from one task to another (round-robin scheduling). Certain execution priorities might also be assigned towards the tasks. When more than one task exists in READY list, the RTX kernel uses the execution priorities, so that the next task should be executed (scheduling by suspension).

The RTX kernel is fitted with drivers specific to communication peripherals, as follows:

- CAN communication;
- Serial communication;
- USB communication.

The software architecture of an intelligent master

The proposed software architecture is structured in tasks, as illustrated in Fig. 2.

The proposed tasks are grouped in three categories:

- Tasks for communications;
- Tasks for implementing certain protocols;
- Tasks for managing the objects dictionary, the communication with the PC and working with history and configuration files.

The CANOpen communication task uses two types of objects:

- PDO (Process Data Object) – objects specific to real time data exchange, at the level of processes;
- SDO (Service Data Object) – objects specific to configuration and service, at the level of local nodes.

For PDO objects, some communication methods are used, of type *producer-consumer* and *master-slave*, while for SDO objects, *the client-server* model is used.

CAN communication task implements a driver for the CAN communication, driver which hides the particularities of CAN controller on the microcontroller[17, 18]. This receives all network messages and will transmit them to the task which implements the CANOpen Master protocol, if there are messages managed by the network, or will send them to the task which manages the objects dictionary, if there are messages of SDO or PDO types.

The serial communication task implements a driver for serial communication. The serial driver uses two round buffers, one of transmission and one of reception. The transmission and reception of messages will be accomplished depending upon the serial communication protocol, active at that moment:

- The MODBUS protocol, with RTU and ASCII transmission modes;
- The M-Bus protocol, used for energetic consumption supervision;
- The ASCII protocols.

The structure of the acquisition cycle, executed by the communication tasks, excepting CANOpen, is illustrated in Fig. 3.

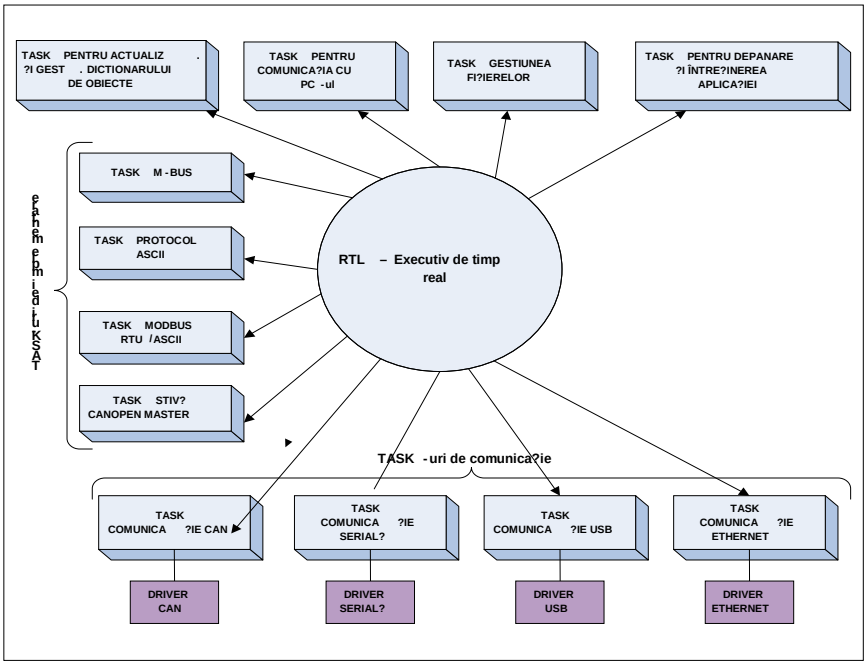

Fig. 2. The main diagram of an intelligent system towards real time acquisition of the distributed data

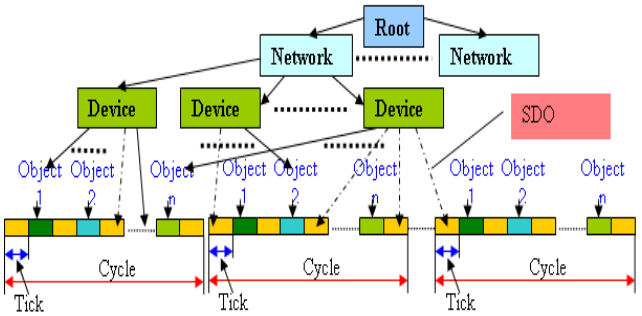

Fig. 3. The structure of the acquisition cycle, executed by the communication tasks, excepting CANOpen

The USB communication task implements a driver for the USB communication. This driver carries out the communication with the PC, by means of USB device.

The Ethernet communication task implements a driver for the Ethernet communication. The driver accomplishes the communication with the PC, basing upon TCP/IP protocol (stack), where efficiencies are measured by taking into account the data flow, when high blocks of data are transmitted.

CANOpen disposes of a network management (NMT), in order to monitor and control all the devices connected to the network. Each CANOpen device implements a state machine, which might be controlled by the NMT Master, through the help of the bus. This “states machine” represents the central part of the NMT Slave functioning. The NMT Master is the only device able to influence other states machines of the network, and it is allowed to only one NMT Master on network. One might notice that an available NMT Master is part of the network resources and is able to coexist with a NMT Slave, on the same physical device.

In contradistinction to CANOpen protocol, the MODBUS RTU/ASCII protocols, the ASCII protocol and M-BUS protocol do not specify separately the behavior of the Master device. As consequence, these tasks were conceived only for the configuration and managing the tasks that accomplish a properly communication. The assignments of these tasks are the following:

- Defining the structure of the acquisition cycle, thus emphasizing the quanta for PDOs and SDOs, respectively;
- Defining the size of the acquisition quantum and their numbering;
- Attaching to each PDO quantum a specific protocol command, thus indicating if a response is waited or not for that command;
- Defining, if such be the case, certain series of successive quanta for multiple commands.

These tasks are accomplished by means of two ways:

- According to a structure stored into a flash memory or under the form of a file stored either into a flash memory or external device (SD Card, MMC Card, etc.);
- Upon basis of certain commands received from the host computer, at the stage of initializing the intelligent Master. The protocol will not be operational in the situation when the work configuration is not available (local or of the host computer).

The tasks can meet supervision functions, such as:

- Counting the messages properly transmitted, and received, respectively;
- Counting the messages with errors at the transmission and reception;
- Counting the timeouts;
- Metering the functioning time, maintaining the records of inputs and outputs of stations on the network.

These functions can generate LOG files, locally stored or can send command messages to the host

computer.

These tasks, similar to the task corresponding to CANOpen protocol, will not interpret the EDS files, since this task needs increased resources and might be easily to implement on the host computer.

Taking into consideration the Intelligent Master, the Dictionary of objects consists of messages of variable length, specific to each protocol. In order to keep these messages, two round buffers or proper size are used, one for the transmission towards the host computer and one towards the reception. The operations over the reception buffer are accomplished by:

- The task specific to the communication with the PC, which submits the messages into the dictionary (buffer);
- The task specific to updating and managing the Dictionary of objects, which takes over the data messages and send them to serial communication tasks, either CAN or Ethernet;
- The control messages, which are sent towards the implementation tasks.

The task used for the communication with the Pc manages two round buffers, one for the transmission and one for the reception, whose size depends upon the communication type, chosen among the RS232 serial communication, USB or Ethernet.

If there is a support for files, the task used for files management will manage the following types of files:

- Files that include the configuration of the acquisition cycle;
- Files that include the logs;
- Files that accomplish a history of data messages on the Dictionary of objects (selectively for each protocol or without selection).

The task used for debugging and maintenance of the application will carry out the following basic functions:

- Manages the counting indicators used by the other tasks, accomplishing log type files, or transmitting control messages towards the task of communication with the host computer;
- Takes decisions as concerns the stopping or starting of a protocol, depending upon the errors number;
- Manages the messages of spy type, useful to application debugging.

Conclusions

A main objective of this paper consists in emphasizing the structure of an Intelligent Master device, which is real time operational and depending upon variants, offers a subset or a complete set of facilities, meaning:

- Will be external to the host computer;
- Will allow the connection to host computer, by using RS232, USB or Ethernet interfaces;
- Will implement the following network protocols: MODBUS TCP/IP Master and Slave, MODBUS RTU and ASCII MASTER, CANOpen MASTER, M-Bus MASTER or ASCII MASTER;
- Will allow the connection of radio modems: XTREAM of 2,4 GHz, GSM or Sony – Ericson modem;
- Will implement the data logger function (will create and store the history);
- Will implement: the TCP/IP stack and sockets, SLIP and PPP protocols, protocols for WEB sites (HTTP), protocols used to e-mails, protocols used to files transferring or the protocols used for the radio modems;
- Will implement the communication component for the connection with the OPC DA 2.05 or OPC XML-DA servers.

The Intelligent Masters can be:

- Local, next to the host computer, or
- Placed away, if they implement the TCP/IP stack and other protocols at the application level (HTTP, WEB servers, MODBUS TCP/IP, FTP or e-mail protocols, etc.).

Acknowledgements

This paper was supported by the project "Computer system for controlling and checking the authenticity of product - ATPROD" - Contract no. 12082/2008 , project co-funded by 2007-2013 PNCDI Program.

References

1. **Dertouzos M. L., Mok A. K.** Multiprocessor on–line scheduling of hard–real–time tasks // IEEE Transactions on Software Engineering, 1989. – No. 12(15). – P. 1497–1506.
2. **Gaitan A. M., Popa V., Gaitan V. G., Hrebenciuc F. A.** Approach on Applications of Random High Data Flows Concerning the Architecture of Computer Networks // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 7(103). – P. 61–66.
3. **Eidukas D.** Modeling of Level of Defects in Electronics Systems // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 3(99). – P. 13–16.
4. **Lin Xu, Yang Han, Jun-min Pan, Chen Chen, Gang Yao, Li–Dan Zhou.** Selective Compensation Strategies for the 3-Phase Cascaded Multilevel Active Power Filter using ANF-based Sequence Decoupling Scheme // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 2(98). – P. 15–20.
5. **Topaloglu N., Gürdal O.** A Highly Interactive PC based Simulator Tool for Teaching Microprocessor Architecture and Assembly Language Programming // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 2(98). – P. 53–58.
6. **Gaitan V. G., Ungurean I., Gaitan N. C., Popa V.** Keeping industrial systems and communication up–to–date using interoperable communicating components and electronic data sheet // IEEE Eurocon 2009. – Saint Petersburg, Rusia, 2009. – P. 389–396.
7. **Gaitan V., Popa V., Gaitan N. C., Danila M. G.** MCPI Application – A scalable Human – Computer Interaction (HCI) // Proceedings of ED–MEDIA 2008 World Conference on Educational Multimedia, Hypermedia & Telecommunications. – Vienna, Austria, 2008. – P. 1522–1527.

8. **Schwan K., Zhou H.** Dinamic scheduling of hard real–time tasks and real–time threads // IEEE Transactions on Software Engineering, 1992. – No. 8(18). – P. 736–748.
9. **Zhao W., Ramamritham K., Stankovic J. A.** Preemptive scheduling under time and resource constraints // IEEE Transactions on Computers, 1987. – No. 8(36). – P. 949–960.
10. **Pahtma R., Preden J., Agar R., Pikk P.** Utilization of Received Signal Strength Indication by Embedded Nodes // Electronics and Electrical Engineering. – Kaunas: Technologija, 2009. – No. 5(93). – P. 39–42.
11. **Stankovic J. A.** Misconceptions about Real–Time Computing: A Serious Problem for Next–Generation Systems // IEEE Computer, 1988. – No. 10(21). – P. 10–19.
12. **Stankovic J. A., Ramamritham K.** IEEE Tutorial: Hard Real–Time Systems // IEEE Computer Society Press, Washington, USA, 1988.
13. **Wellings A.** Editorial: real–time software // lEEE Software Engineering Journal (Special Issue on Real–Time Software), 1991. – No. 3(6). – P. 66–67.
14. **Stewart D. B., Khosla P. K.** Real–time scheduling of sensor–based control systems // In Eighth IEEE Workshop on Real–Time Operating Systems and Software, 1991.
15. **Gaitan V. G., Popa V., Turcu C., Gaitan N. C., Uugurean I.** The Uniform Engineering of Distributed Control Systems Using the OPC Specification // Advances in Electrical and Computer Engineering, 2008. – Vol. 8. – No. 2. – P. 71–77.
16. **Gaitan V. G., Popa V., Ungurean I., Gaitan N. C.** The Integration Of Real Device Capabilities In Distributed Applications Based On OPC Technology // 12th WSEAS International Conference on Computers. – Heraklion, Creta Grecia, 2008. – P. 48–153.
17. **Riid A., Preden J., Pahtma R., Serg R., Lints T.** Automatic Code Generation for Embedded Systems from High–Level Models // E Electronics and Electrical Engineering. – Kaunas: Technologija, 2009. – No. 7(95). – P. 33–36.
18. **Brazaitis A., Guseinoviene E.** Control of Activators of Mechatronic Devices by Real Time Information Transfer // Electronics and Electrical Engineering. – Kaunas: Technologija, 2009. – No. 2(90). – P. 89–94.

Received 2010 01 30

N. C. Gaitan. Real-time Acquisition of the Distributed Data by using an Intelligent System // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 8(104). – P. 13–18.

For real time applications, the specific time requirements represent the main constraints, and their controlling is an essential factor in order to establish the quality of the accomplished services. The time constraints might be found within more application areas, such as the automatization of industrial equipments, the embedded systems, the control of vehicles, monitoring the nuclear devices, supervising the scientific experiments, robotics, conditioning of audio and video multimedia data flows, monitoring the surgical interventions, as well as supervising the stock exchange operations. Within this paper, the hardware and software architecture of an intelligent system for real time acquisitions of the distributed data is presented. This intelligent system is represented by means of an embedded device, performed with a STR710FZ2 microcontroller and a RTX real time kernel. Ill. 3, bibl. 18 (in English; abstracts in English, Russian and Lithuanian).

Н. С. Гайтан. Сбор распространённых данных, которые используют интеллектуальную систему в реальном времени // Электроника и электротехника. – Каунас: Технология, 2010. – № 8(104). – С. 13–18.

У программ реального времени существуют конкретные требования, зависящие от времени. Это и является одной из главных проблем. Контроль данных программ – это главный фактор определения качества предоставляемых услуг. Ограничение времени встречается в довольно многих областях применения программ, к примеру в областях автоматизации промышленности (встраиваемые системы, контроль транспортных средств, мониторизация ядерных объектов, контроль научных экспериментов, роботика, потоковое видиео и аудио, мониторизация хирургических операций, а также мониторинг фондовых рынков). В данной статье представлены обе архитектуры: аппаратная и на уровне программ интеллектуальной системы сбора распространённых данных в режиме реального времени. Эта система представляется как „embedded“ устройство сделанная с помощью микроконтроллера STR710FZ2 и ядра реального времени RTX. Ил. 3, библ. 18 (на английском языке; рефераты на английском, русском и литовском яз.).

N. C. Gaitan. Realaus laiko intelektualiųjų sistemų paskirstytųjų duomenų surinkimas // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2010. – Nr. 8(104). – P. 13–18.

Įrodoma, kad šiandieninėse sistemose (transportas, medicina, atomistika, intelektualiosios mokslinės sistemos ir kt.) daugiausia dėmesio skiriama programų duomenų kontrolei. Siūlomos dvi programų architektūros: plačiai naudojama paprasta ir realaus laiko intelektuali. Plačiai aprašoma intelektualioji architektūra, sukurta mikrovaldiklio STR710FZ2 pagrindu realiu laiku. Il. 3, bibl. 18 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).

DOI: 10.5755/j02.eie.9199