



Co-funded by the
Tempus Programme
of the European Union

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
Zaporizhzhya national technical university

REMOTE AND VIRTUAL TOOLS IN ENGINEERING

Textbook

General editorship Dr. Ing. Karsten Henke

Zaporizhzhya
DIKE POLE
2016

УДК 004.41

ББК 32.973

R-42

Authors: A.V. Parkhomenko, G.V. Tabunshchyk, M.O. Poliakov, O.M. Gladkova, T.I. Kapliencko, T. Y. Larionova.

Recommended for publication by the Academic Council of Zaporizhzhya National Technical University (protocol № 9 of February 22, 2016)

Reviewers:

HOMENYUK S.I. – PhD, Professor, Dean of the Faculty of Mathematics, Zaporizhzhya National University

TARASOV O.F. – PhD, Professor, Head of Department of Computer Information Technology, Donbass State Engineering Academy

TESLYUK V.M. – PhD, Professor, Professor of CAD Department, National University “Lviv Polytechnic”

R-42 Remote and virtual tools in engineering: textbook / general editorship Dr.Ing.Karsten Henke. – Zaporizhzhya: Dike Pole, 2016. – 250 pp.

ISBN 978-966-2752-93-9

The textbook highlights the basic issues of practical study of virtual, remotely controlled devices and distributed remote laboratories. The publication contains the possibilities of their application for design and quality evaluation of embedded systems, for development of control systems of complicated electrotechnical installations and complexes. The textbook can be useful for students and graduate students in the field of study: Information technologies, Electrical engineering as well for professionals in the field of Systems design.

УДК 004.41

Published with the support of the international project ICo-op «Industrial Cooperation and Creative Engineering Education based on Remote Engineering and Virtual Instrumentation» (530278-TEMPUS-1-2012-1-DE-TEMPUS-JPHES) according to the European Commission programme TEMPUS.

The content of this edition represents the opinion of the authors and the European Commission does not bear responsibility for the use of the information contained in the textbook.

ISBN 978-966-2752-93-9

©A.V.Parkhomenko, G.V.Tabunshchyk, M.O. Poliakov, O.M. Gladkova,
T.I. Kapliencko, T. Y. Larionova, 2016

CONTENTS

PREFACE	6
PART 1. TECHNOLOGIES AND SYSTEMS OF VIRTUAL AND REMOTE ENGINEERING	7
INTRODUCTION TO PART 1	7
1 New Approaches to Design and Production Activity Based on Virtual Engineering and Remote Experiment . . .	10
1.1. Technologies of Virtual Engineering.	10
1.2. Virtual and Remote Laboratories	12
1.3. Control Questions	25
1.4. Practical Exercises	25
2 Embedded Systems	26
2.1. Embedded Systems Market and Features	26
2.2. Embedded Systems Requirements Analysis and Creation of Project Documentation	28
2.3. The basic concept of embedded systems design . .	38
2.4. Approaches to Embedded Systems Hardware Realization	44
2.5. Approaches to Embedded Systems Software Realization	49
2.6. Control Questions	53
2.7. Practical Exercises.	53
3 Embedded Systems Design Using Virtual and Remote Tools	54
3.1. ES Integrated Development Environment.	54
3.2. ES Practical Design Using Remote Experiment .	58
3.3. Control Questions	93
3.4. Remote experiments	93
3.5. Lab Works	93
Conclusions.	94
Literature for part 1.	94

**PART 2. CONTROL SYSTEMS WITH
PROGRAMMABLE LOGIC CONTROLLERS 101**

INTRODUCTION TO PART 2. 101

4 Introduction to controller control systems.....102

4.1. Types and Properties of Controllers 102

4.2. Hardware of Industrial Controller.....106

4.3. Functional Organization of Controller 114

**5 Controller programming and formalization of design
solutions 122**

5.1. IEC61131 programming language overview ...122

**5.2. Software Structure of Industrial Automation
Systems 129**

**5.3. Ladder Diagram (LD) language:
construction and basic instruction 132**

5.4. Formalization of control tasks 136

5.5. FSM models of the control system behavior ...138

5.6. Tools for work with remote laboratory.....140

6 Typical control tasks 143

6.1. FSM modeling on LD language 143

6.2. Standard tasks of control.....151

6.3. Man Machine Interface Software 156

**6.4. Behavioral synthesis and scripting of
visualization problem.....158**

6.5. Self-evaluation in form of quiz, exercises 158

6.6. Self-evaluation in multiple-choice form.....160

6.7. Laboratory Work.....162

6.8. Practical Work.....163

Literature for part 2.....163

PART 3. QUALITY OF INFORMATIONAL SYSTEMS . . . 166

INTRODUCTION TO PART 3. 166

7 Basics of Embedded systems verification 167

7.1. Embedded Systems verification methods 167

7.2. Model for verification of the Embedded System .174

7.3. Individual tasks 177

7.4. Control questions 177

8 Embedded Systems testing 179

8.1. Terms and definitions 180

8.2. Hardware faults models 183

8.3. Testing of Embedded Core-Based System-on-Chips 185

8.4. Boundary scan. 187

8.5. Embedded systems software testing 191

8.6. Regression testing for Informational Systems with web-interface. 194

8.7. Individual tasks 202

8.8. Control questions 203

9 Teaching of Embedded systems testing based on GOLDI. 204

9.1. Analysis of RL GOLDi facilities for Quality of Informational System Study 204

9.2. RL GOLDi usage for model-based testing 209

9.3. RL GOLDi usage for functional testing of Embedded systems. 213

9.4. Individual tasks 220

9.5. Control questions 221

Literature for part 3. 221

ALPHABETICAL-SUBJECT INDEX. 224

ABOUT AUTHORS. 230

APPENDIX. THE REMOTE LAB CLOUD “GOLDI-LABS.NET” 233

PREFACE

One of the goals of the International Tempus project 530278-TEMPUS-1-2012-1-DE-TEMPUS-JPHES “Industrial Cooperation and Creative Engineering Education based on Remote Engineering and Virtual Instrumentation” [ICoop] (www.ico-op.eu) is modernization of engineering education for improvement of graduates professional level and strengthen of the university-enterprise collaboration.

Solving of project tasks base on GOLDi (Grid of Online Lab Devices, www.goldi-labs.net) usage. GOLDi hardware and software were developed by Integrated Communication Systems Group at the Ilmenau University of Technology, Germany and installed in Zaporizhzhya National Technical University (ZNTU), Ukraine.

Facilities of new ZNTU Laboratory of Remote Engineering provide permanent online access for students, teachers and specialists from industry; give possibilities to organize students’ self-study process more efficiently, to control students’ work and to broaden the ways of communication in research work with companies and enterprises.

Our partners from ZNTU recommended GOLDi usage for such specialties as Systems Software, Information technologies of design, Electrical and electronic apparatus. In the frameworks of new developed in ZNTU courses students deal with using of remote and virtual tools for embedded systems and intellectual controller systems design and verification.

During the first pilot teaching ZNTU master students together with specialists from Zaporizhzhya region enterprises had an opportunity to work with the Laboratory of Remote Engineering and to give a good practical experience.

This project gave ZNTU new possibilities to spread the ideas of creative engineering education and find new industrial partners for sustainable bidirectional cooperation in education and research. I hope that textbook in English will help to improve students foreign language knowledge and skills for their future successful career.

Dr. Ing. Karsten Henke
Coordinator of the ICo-op project,
Ilmenau University of Technology,
Germany

PART 1

TECHNOLOGIES AND SYSTEMS OF VIRTUAL AND REMOTE ENGINEERING

Anzhelika Parkhomenko, Olga Gladkova

INTRODUCTION TO PART 1

The rapid development of Internet technologies and their growing popularity very significant impact on the development of technology. Sharing virtual and remotely controlled devices and distributed remote laboratory is an urgent task today in connection with:

- increasing complexity of engineering problems that require a solution in the short term of the project;
- use in the design of more and more specialized expensive equipment and software;
- inaccessibility of high-tech equipment for SMEs (small and medium-sized enterprises);
- need for highly qualified personnel to manage modern equipment;
- the requirements of globalization and division of labor market.

Today Remote Laboratories are successfully developed and implemented worldwide. An observation is that the vast majority of already implemented experiments and installations with remote access are used as pure educational resources and that the possibilities for the professional researcher or designer are not fully explored.

On the other hand the Embedded Systems market is permanently evolving and requires the creation of more complex systems in an increasingly shorter period of time. Under these conditions, the development of systems «from scratch» is not effective. Therefore, one of the key areas to improve the efficiency of Embedded Systems design is the accumulation of technical solutions for reuse. This (reuse) of knowledge can be illustrated and trained in virtual and remote laboratories. The disunity of descriptions of already developed components (hardware

units, programs, algorithms implementations, etc.) hinders their reuse, but the application of virtual and remote experiments will allow the designer to obtain information about hardware and software platforms and components for making decisions on the Embedded Systems realization.

Course «TECHNOLOGIES AND SYSTEMS OF VIRTUAL AND REMOTE ENGINEERING» provides students in the fields of Information technologies, Electrical engineering with skills and knowledge required for design and production activity based on virtual engineering and remote experiment.

Main learning objectives are:

- exploring new approaches to design and production activity based on virtual engineering;
- study of embedded systems hardware and software features of realization;
- exploring stages and approaches to design of embedded systems;
- familiarizing with tools for managing requirements to embedded systems and creating project documentation (IBM Rational Requisite Pro);
- study of architecture and functionality of Integrated Development Environment for computer-aided design of embedded systems software and hardware;
- software-oriented design of embedded control systems using Atmel Studio and Arduino;
- hardware-oriented design of embedded control systems using Altium Designer and CREO;
- creating and testing embedded systems' virtual prototype using Proteus or Arduino simulators;
- embedded systems' design using lab GOLDI and RELDES remote experiments.

After completing this course the student will know the principles of design and implementation embedded systems and be able to elaborate Hardware and Software of embedded systems using technologies and systems of virtual engineering.

Main learning outcomes are:

- students will be able to implement new approaches to research, design and production activity based on virtual and remote engineering;

- students will be able to analyze structural and functional features of modern virtual and remote laboratories, as well as embedded systems Integrated Development Environment;
- students will have the skills of embedded systems virtual prototype elaboration and research;
- students will have the skills of remote experiment organizing and holding.

The first chapter presents new approaches to design and production activity based on virtual engineering and remote experiment. The analysis of the structural and functional features of some well-known remote laboratories is shown.

At the second chapter ES market research presented. Also the requirements analysis that must be considered during embedded systems design has been done. Peculiarities of the requirements structuring and developed model of requirements are described. Methodology of creating requirements, taking into account the processes of their definition and analysis proposes. The existing approaches to embedded systems hardware/software realization are described.

The third chapter presents issues of embedded systems design using virtual and remote tools:

- ES Integrated Development Environment;
- ES Practical Design Using Remote Experiment;
- REmote Laboratory RELDES (ZNTU, Ukraine) and Hybrid laboratory GOLDI (TU Ilmenau, Germany) using for ES design.

1 NEW APPROACHES TO DESIGN AND PRODUCTION ACTIVITY BASED ON VIRTUAL ENGINEERING AND REMOTE EXPERIMENT

1.1. Technologies of Virtual Engineering

Virtual engineering provides means for simulation of various types of engineering activities at all stages of development and production of products: design and modeling; simulation of machine processing; imitation of assembly; management of production lines; visual inspection and evaluation. Exploring of virtual prototype allows: to reduce the development time; to explore more alternative designs; do not create expensive physical prototypes; perform research of prototype, which is impossible in real conditions on the layout; do not implement physical experiments that require a substantial period of time.

Virtual manufacturing is an integrated working environment that can be classified as:

- design-oriented virtual production or simulation environment for the design of the product and assess its production possibilities.
- production-oriented virtual production or simulation environment for the planning of production processes and production;
- management-oriented virtual production or simulation environment for simulating the operation of industrial shops.

The main components of virtual manufacturing is a virtual design, digital simulation, virtual prototyping and virtual plant.

Virtual design provides the ability to create components, modify them, to control devices and to interact with virtual objects in a virtual environment. The designer can see a stereoscopic image of the virtual objects and hear realistic surround sound. These images and sounds occur when the hand of the designer moves the virtual hand and finger. Touching the virtual object is felt by the designer in the form of feedback. The designer gets the opportunity to effectively implement the project ideas, and to test the functional behavior of the design. This approach allows also to take into account the point of view of a potential user of the product because at early stages of design can be fully appreciated qualities such as availability and manageability. There is also the

opportunity to be considered complex and difficult to formalize the experience of experts in the assembly or manipulation of parts when designing. Thus, the virtual design system helps to understand a user's position, its interaction with objects and sequence of assembly.

Digital simulation aims to evaluate the performance of the products without the use of physical prototypes and validation of manufacturing processes. Using simulation, the user can predict a collision between the tool and the fixture or part, to check the path of movement of the CNC tool, probe coordinate measuring machine or robot hand. Visualization helps engineers to better understand the system as it allows to easily understand the idea of design and to check its performance. Currently for this purpose mainly used kinematic simulation of solids. Higher level simulation models of liquids, human beings and complex environments requires sophisticated modeling of physical effects, including the effects of dynamics, vibration, acoustics and deformation. However, complex simulation using virtual prototypes can verify the operating characteristics of the system faster and with less effort.

Virtual prototyping involves creation of designed product computer prototype, which geometry and physical behavior correspond to the real product. Virtual prototyping systems allow check the possibility of assembly in the existing production environment by visualizing the assembly process. Virtual prototype assembly allows identify structural failures, to work out the sequence and the optimal path of movement of the parts and, if necessary, make changes in the project. Advanced systems provide the opportunity for structural and functional analysis of the virtual prototype, as well as kinematic and dynamic simulation of its operation.

Virtual plant – a production system model that simulates the production sites, processes, storage systems, and automated factory equipment – machine tools, robots and conveyors. Constructed production model allows analyze labor costs, operating costs, processing costs and duration of the production cycle. This allows use virtual plant for production planning, production systems project evaluation and comparison of alternative methods of production. If using a virtual plant to simulate the entire supply chain, this will allow you to evaluate and optimize the entire process of resource management and production.

Thus, modern technologies of virtual engineering allow:

- evaluate the possibilities of different designs producing (including the assessment of the assembly quality or operational characteristics of design product);

- optimize the production process (using digital simulation);
- easy customize the product to the customer's requirements;
- effectively gain a broad knowledge base;
- provide the basis for collaborative development projects [48].

1.2. Virtual and Remote Laboratories

Virtual laboratory is a hardware-software complex, allowing to conduct a research without direct contact with the actual installation or its complete absence. In the first case we are dealing with the so-called laboratory installation with remote access, which includes real lab, hardware and software for plant control and digitization of the received data, and means of communication. In the second case, all processes are modeled using the computer.

So, Virtual laboratories refer to two types of hardware-software complexes:

- laboratory installation with remote access – we will call such a set of Remote laboratory (or laboratory with remote access);
- software which allows you to simulate laboratory experiments – Virtual laboratory (in the narrow sense).

The main advantages of virtual laboratories are the following:

- no need to purchase expensive equipment and reagents;
- ability of simulation of process, which realization is fundamentally impossible in the laboratory;
- the opportunity to observe what is happening in a different time scale;
- security when you are working, for example, with high stress or chemicals;
- the ability to quickly perform a series of experiments with different values of input parameters;
- save time and resources for presentation of results in electronic format;
- the possibility of using virtual laboratories for distance learning [66].

Today in the world experience of creation and operation of remote laboratories there is a clear trend «information aggregation» of existing Internet resources access to laboratory experiments. The need for this is due to two factors:

- too many already implemented remote access projects for experiments from different branches of science makes very difficult the task of identification and classification of required laboratories;
- financial-economic factor: the possibility of obtaining grants for projects laboratories remote access for a large number of groups of performers who then place links to their remote laboratories web resources on the same (main) website [58].

Below are some examples of virtual laboratories for industry and education.

Hybrid online laboratory GOLDI (Grid of Online Lab Devices Ilmenau) (<http://www.goldi-labs.net/>) – was developed at the Department of Integrated Communication Systems at the Ilmenau University of Technology. It provides a tool set, supporting all design steps for complex control tasks (e.g., in the fields of control engineering, robotics, tele-control engineering). Goal of the GOLDI system is to show state of the art ways and challenges of remote controlling and remote observation of real processes, dealing with the integrated and interactive usage of modern Internet and intranet technologies like HTML5, JavaScript, etc. It offers various features like visualization and animation, which allows to observe and to test all the properties of the design. In connection with formal design techniques, simulation and prototyping are used to establish a foundation for the development of a reliable system design.

To check the functionality of the whole design, some special simulation and validation features are included as integral part of the *GOLDI* system, such as:

- usage of simulation models of the physical system for visual prototyping,
- step by step and parallel execution of these prototypes,
- visualization of the simulation process with the tools also used for specification,
- features for test pattern generation and
- code generation for hardware and software synthesis.

GOLDI offers a web-based environment supporting the above mentioned features to generate and execute a design by using simulation models (Fig. 1.1). At any time the students have the chance to adjust their algorithms in case of faults. Therefore, they are able to achieve a fault free solution (a validated control algorithm) step by step.

This online lab is used for teaching practical lessons as well as giving hands-on experiences for the development of embedded electronics. This is done using remote lessons via the Internet and has the advantage that courses can be offered internationally world-wide. Additionally, even for local students, the lab offers extended opening hours (twenty-four-seven) when compared to a regular lab. Besides the advantages for students, this also reduces the costs for academic teaching and improves the quality by offering more practical training possibilities.

GOLDI offers three different specification and control mechanisms to handle the physical systems (electro-mechanical models, e.g. elevator, production cell, high rack warehouse) in the lab room.

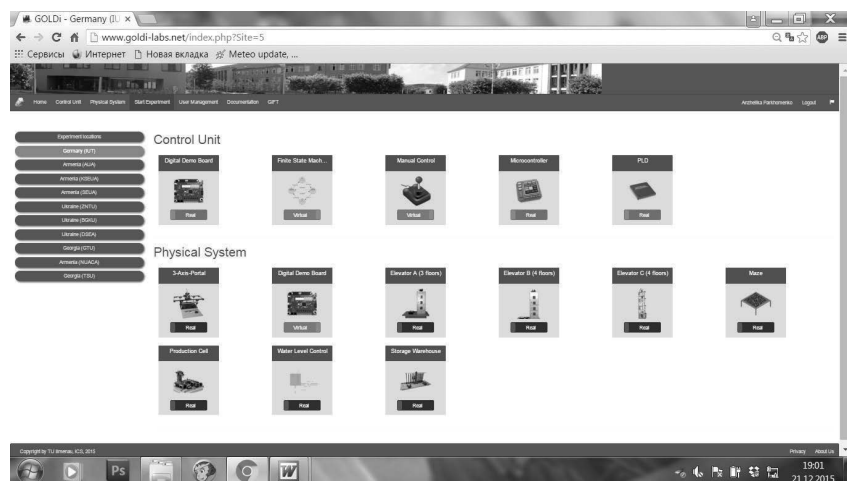


Figure 1.1 Example of GOLDI Web-Interface

WebLab-Deusto – a remote laboratory for technical curricula, provided by the Faculty of Engineering of the University of Deusto (<https://www.weblab.deusto.es/web/>). It can be downloaded and deployed to serve new remote experiments in different environments and operating systems (Fig. 1.2).

The general features of the system are a secure and scalable design, using SSL in the communication, checking experiment in the management layers, and tracking down the use of the system and the messages sent by students. The system also supports different authentication schemes such as a regular database, LDAP, supported external students OpenID. It is extensible enough to be integrated into a number of platforms

such as Facebook, .LRN, Joomla and Moodle, and to integrate external experiments such as VISIR.

Currently the experiment can be developed using two different approaches: managed and unmanaged experiments. WebLab-Deusto pools their resources for the development of such scenarios where students access hardware of experiments located at the University electronically.

Seven types of different experiments are offered by WebLab-Deusto: Xilinx Programmable Logic Devices; Microchip PIC Microcontrollers; Programmable Mobile Robot; Electronics & instrumentation; Logic games and custom C++ application connected to an oscilloscope through GPIB; Weblab-Submarine: a real aquarium is remotely maintained.

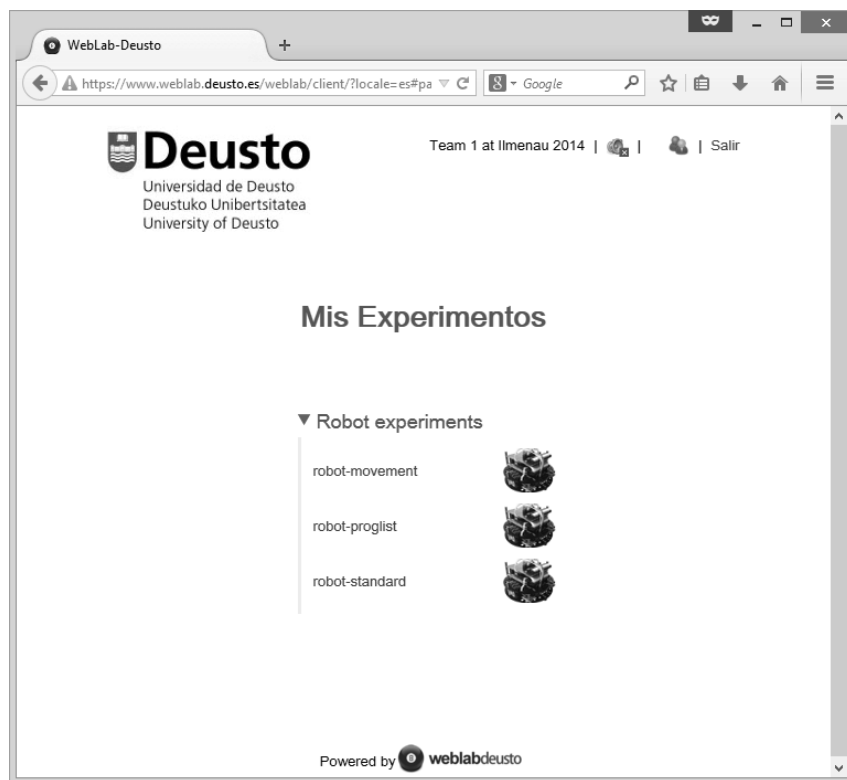


Figure 1.2 Example of Lab-Deusto Web-Interface

WEBENCH® Design Center – virtual laboratory of Texas Instruments (<http://www.ti.com/lstds/ti/analog/webench/overview.page>).

Design Tools WEBENCH represent effective software implemented algorithms and graphical user interfaces that provide fast work of applications for power management, developing drivers for powering LEDs, and applications that work with different sensors (measuring devices). These easy-to-use tools help you create, optimize and simulate designs that meet the unique technical requirements of development. This allows the user to compare the current cost on system level and supply chain before design (Fig.1.3).

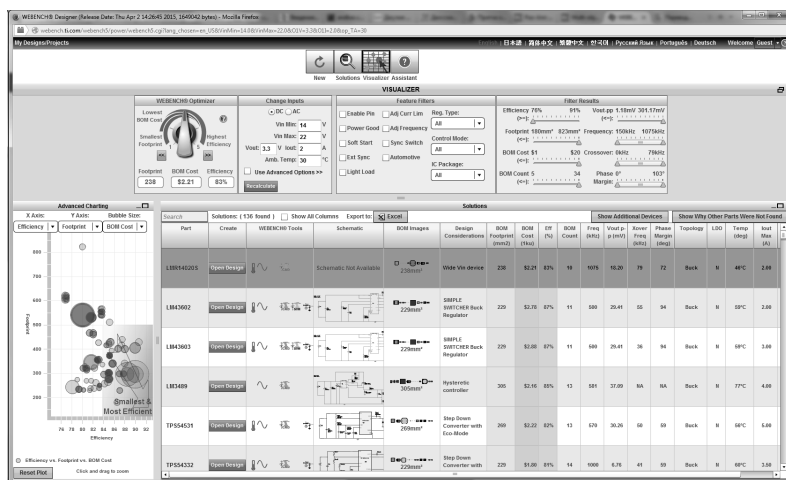


Figure 1.3 WEBENCH® Design Center lab interface example

Labshare – National Australian nonprofit remote control labs network (<http://www.labshare.edu.au/>). It consists of 5 technical universities. Experimental stands provide for 10 experiment (Fig.1.4).

- control of liquid in connected tanks;
- engineering geology (the study of rock samples and materials);
- hydropower plant, demonstrating the basic principle of converting kinetic energy of water flowing into electrical energy by rotating turbines (Peltona wheels) connected to an electric generator;
- iRobot installation to study the concept of remote control robot, precision sensors, localization (adaptation to space) and ordering scheme of movement;

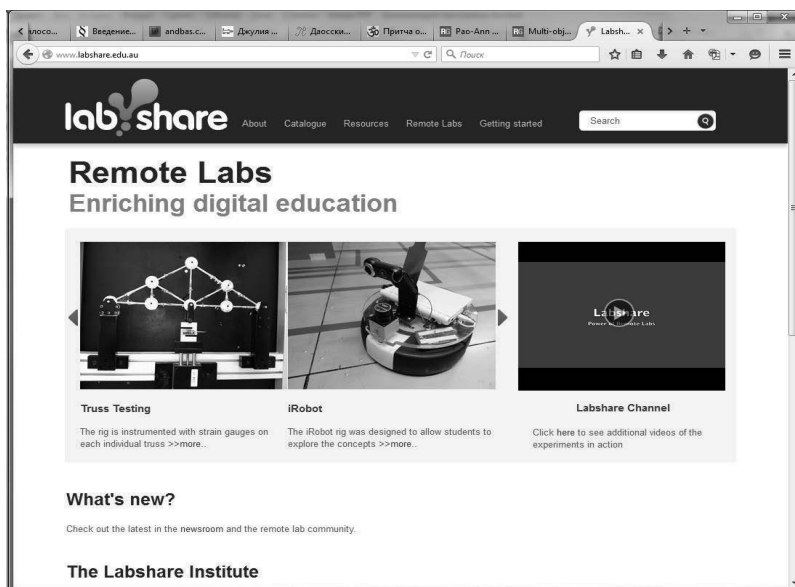


Figure 1.4 Labshare web interface example

- sample structural beams, to study the tensile and compression forces in structures such as concrete floor or wing aircraft;
- setting that allows you to connect real electronic components in the chain, which then can be tested using test equipment with real data;
- learning the basics of automation PLC (programmable logic controller) by industrial PLC programming using relay logic;
- advanced analysis of the behavior of the building during the earthquake;
- observing how supply source change and current magnetization effects generated capacity, voltage and current;
- turbulent and laminar currents difference inside the tubes.

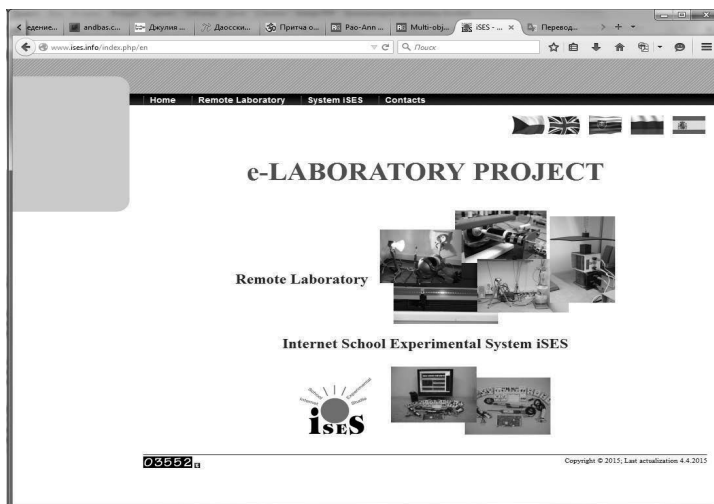


Figure 1.5 e-Laboratory Project web interface example

e-Laboratory Project – Czech Republic and Slovakia remote lab (<http://www.ises.info/index.php/en/>) (Fig. 1.5). Distant experiments: water level control; weather station; electromagnetic induction; natural and controlled fluctuations; solar energy transformation; micro objects diffraction; Heisenberg indeterminacy principle; photo effect; light polarization; radioactivity.

iLabs – the online laboratory, created for enrichment of laboratory practical works in engineering education. Remote lab is created at the Massachusetts Institute of Technology (Fig. 1.6). (<http://icampus.mit.edu/projects/project/?pname=iLabs>, <https://wikis.mit.edu/confluence/display/ILAB2/Home>)

iLab is dedicated to the proposition that online laboratories-real laboratories accessed through the Internet can enrich science and engineering education by greatly expanding the range of experiments that students are exposed to in the course of their education. Unlike conventional laboratories, *iLab* can be shared across a university or across the world. The *iLab* vision is to share expensive equipment and educational materials associated with lab experiments as broadly as possible within higher education and beyond, thus profoundly changing the economics of engineering education. *iLab* teams have created remote laboratories at MIT in microelectronics, chemical engineering, polymer crystallization, structural engineering, and signal processing as case

studies for understanding the complex requirements of operating remote lab experiments and scaling their use to large groups of students at MIT and around the world. One such laboratory, which allows students to take measurements of the current/voltage characteristics of transistors and other microelectronic devices, is now used every year in three different MIT courses.

It is also regularly used by hundreds of students in several countries on four different continents.

In addition, seven other iLabs were created to continue to promote shared access to high-quality laboratory equipment by educational institutions around the world. To date, iLabs have been used in credit-bearing assignments by over 4,500 students, including students in the U.S., China, Australia, Sweden, Italy, Greece, and SubSaharan Africa.

Remote-LABGymKT – Czech Republic remote lab provide access to 6 experiments in the field of physics (<http://remote-lab.fyzika.net/vzdalene-experimenty.php?lng=en#DERIL>):



Figure 1.6 iLabs lab web interface example

- metallic and semiconductors resistance dependency;
- define horizontal element of the Earth magnetic field using Helmholtz chokes;
- robotic-manipulator hand remote control;
- defining Volt-Ampere characteristics of light diodes (Fig. 1.7).

The purpose of the laboratory is not only to support the practical teaching of physics, but also gathering experience with the operation of remote online laboratories.

Remotely controlled laboratory – the remote experiment at Department of Experimental Physics, Palacky University in Olomouc (Czech Republic) is available free, without registration, 24 hours and 7 days a week. (http://ictphysics.upol.cz/remotelab/rlab_en.html) Remotely controlled experiments have become a widespread tool for teaching physics at the university level of education. Their aim is not the replacement of the real experiment by the remote experiment but finding new alternatives in teaching science. 5 experiments were implemented (Fig. 1.8):



Figure 1.7 Remote-LABGymKT lab web interface example

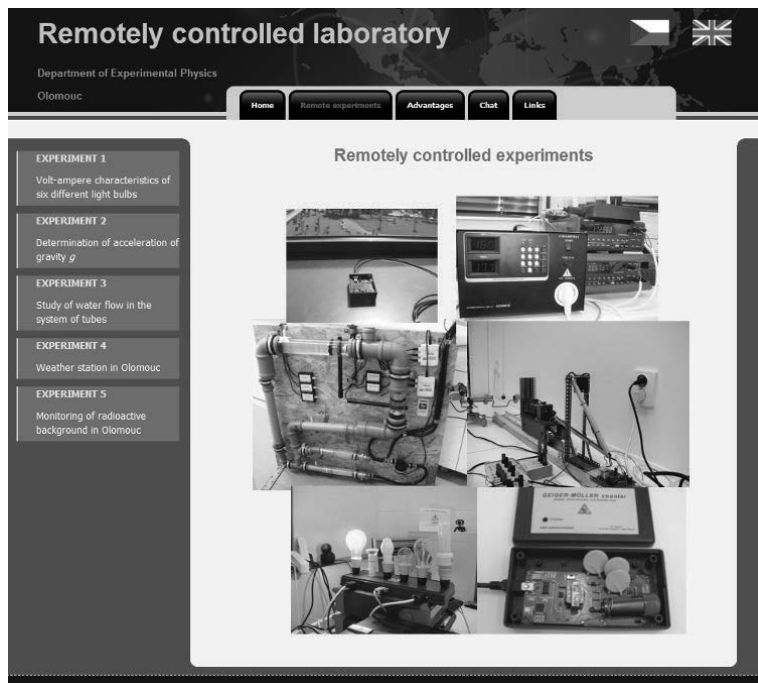


Figure 1.8 Remotely controlled laboratory web interface example

- Volt-Ampere characteristics of 6 different bulbs;
- measuring gravitation boost;
- observing water current within tubes network;
- weather station;
- background radiation monitoring.

OpenLabs – Blekinge Institute of Technology, Sweden. The lab provides resources needed to experiment in electronics (3 experiments) via the Internet. Supply basic equipment, such as oscilloscope, multimeter, function generator and power supply. With these and a number of electronic components you can build circuits on virtual breadboard. None of the measurements are simulated. The circuits that build will be formed and measured on, and the real measurement results will be displayed. (Fig. 1.9).

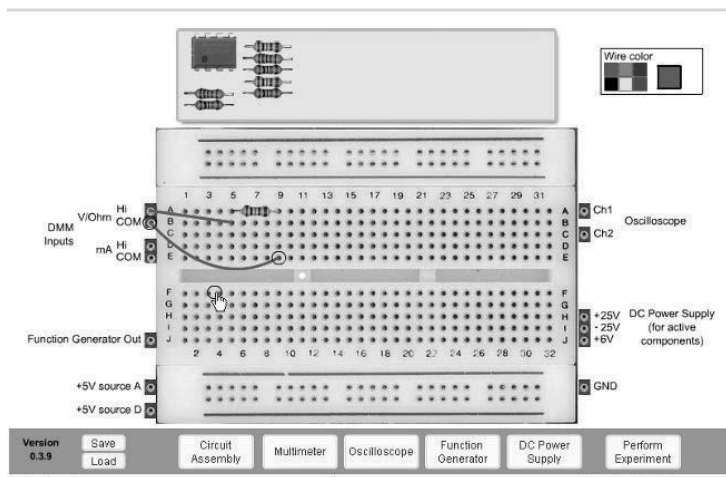


Figure 1.9 OpenLabs lab web interface example

Relle – Remote Experimentation Laboratory (<http://relle.ufsc.br/>), at the Federal University of Santa Catarina (UFSC), Brazil, which currently is part of a network of 12 universities (Rexnet) in five different countries. REXLab has as one of its objectives to meet the need for social appropriation of science and technology, popularizing scientific and technological knowledge by encouraging young people to become interested in scientific and technological careers. It is needed to pursue initiatives that integrate science education to the educational process by promoting the improvement renewing education in all levels, emphasizing actions and activities that enhance and encourage creativity, experimentation and interdisciplinarity.

Remote experimentation allows the remote access to real labs using the Internet. The experiments can be accessed anytime, from anywhere (Fig. 1.10).

DC Electric Panel – study of mixed, series and parallel association of resistors in DC networks.

AC Electric Panel – study of associations resistors in series, parallel and mixed in AC networks.

Heat conduction in metal bars – this lab studies the heat propagation model in metal bars.

Deployment Environment for Arduino – environment that allows code verification and upload, and controlling sensors and actuators in Arduino

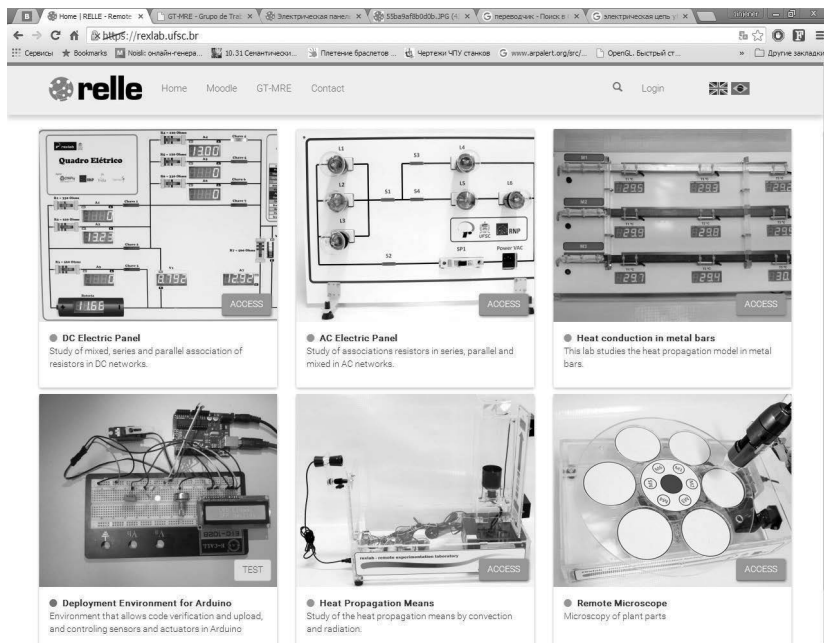


Figure 1.10 Relle lab web interface example

Heat Propagation Means – study of the heat propagation means by convection and radiation.

Remote Microscope – microscopy of plant parts

Inclined Plane – study of second Newton's law and force decomposition
Thomson's Experiment – lab for determining the charge-to-mass ratio of the electron

Newton's Disc – Study of color composition

Conversion of Light Energy into Electrical Energy – the light energy reaches the solar cell and is converted into electricity by the photovoltaic effect.

Cobra Cora – collaborative use of a robotic arm to teach logics using a visual programming language

Optical Bench – Study of the behavior of light in the most common lenses and various formats.

The *NetLab* remote laboratory project (<http://netlab.unisa.edu.au/>) is being developed as part of the University of South Australia's policy and commitment to innovation and its support of the development of new technologies.

NetLab is an online remote laboratory, which is used by academic staff for teaching and demonstrations during lectures, and by students for conducting their experiments remotely on real laboratory equipment.

The application presents users with graphical user interfaces (GUIs) that look like actual laboratory instruments. They click the buttons and turn the knobs with their mouse, interacting with it as they would with the real device (Fig. 1.11).

What makes NetLab unique, is that when students interact with the GUIs, they are actually operating real instruments that are set up in a laboratory in some remote location, controlling them via the Internet. The output they see on the instrument's display panels is not a simulation, but is actual data being read from the real instruments, in real time.

As you can see in many examples below, most remote library around the world were created for educational purposes, they have not been investigated from the point of view of actual production labs.

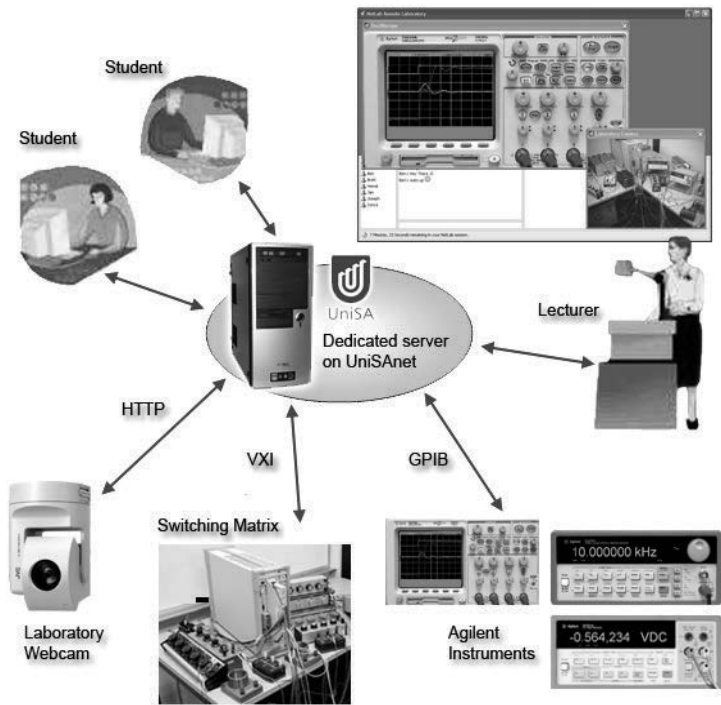


Figure 1.10 NetLab architecture

1.3. Control Questions

- Components of virtual manufacturing.
- Possibilities of virtual engineering.
- Types of Virtual laboratories.
- Advantages of virtual laboratories.

1.4. Practical Exercises

- Use the links, visit laboratories GOLDI and WebLab-Deusto for analysis of their structure and basic functionality. Prepare short presentation.
- Do a search on the Internet of Remote laboratories for embedded systems design. Prepare short presentation.

2 EMBEDDED SYSTEMS

2.1. Embedded Systems Market and Features

Embedded System (ES) – nowadays defined as a specialized computer system, which operates with physical objects and processes directly while solving the problem. ES devices can be defined as:

- microprocessor module with memory;
- boundary system (sensors, executive elements and output-input controllers in order to connect with a controlled object, human-machine interface devices if necessary);
- supply system;
- connecting constructive elements (shell, chassis);
- managing software (SW).
- main ES characteristics are:
- real time operation (almost always);
- various and complex exploitation conditions;
- independent operation (no supply limitations, no operator control);
- high functional safety and stability expectations;
- limited resources;
- critical use cases when human life is in question [57].

Embedded systems are a special class of systems, which can be referred both to automated and to the information-management systems, because the embedded system support (resolves) a series of automated tasks, and by definition is a specialized system that is built into the device, which it will control.

The world market of embedded systems is continuously growing (Fig. 2.1) [10,11,12]. Each year embedded systems are becoming more complex and require a higher level of reliability and durability.

According to the area of application the embedded systems market is segmented as follows: an automotive segment, telecommunications, medicine, industry, household appliances, military and aerospace machinery etc. One of the most difficult and time-consuming embedded system categories is the control system of different moving objects (air, water, land). Interesting for both children and adults is the class of

gaming objects, namely: unmanned aircraft, helicopters, a quad copter, remote-controlled cars and boats.

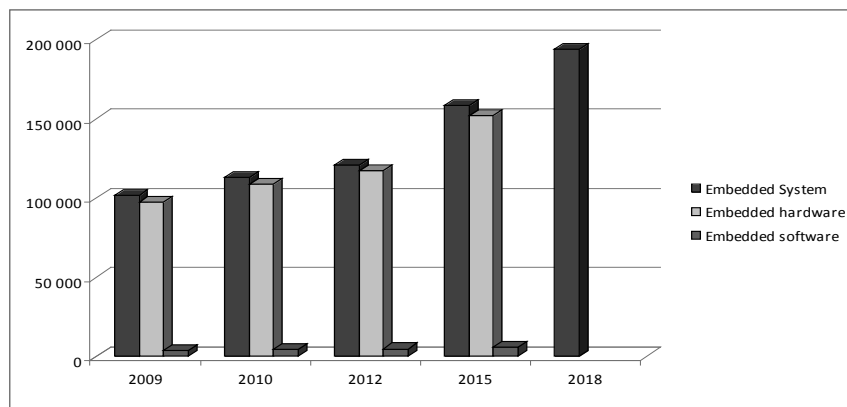


Figure 2.1 Embedded technologies market analysis

Embedded Systems are one of the most complex design objects for computer technology developers [56]. Even a cursory analysis of typical requirements and constraints that must be considered when creating the ES, confirms this:

- minimal own power (possibly self-powered);
- minimal own size and weight;
- toughness and rigidity of the design;
- thermal control;
- radiative and electromagnetic resistance (possibly working capacity in vacuum);
- guaranteed time between failures;
- term availability solutions on the market, etc. [40].

Requirements definition, analyze and correct distribution between hardware and software components of the architecture at the stage of high-level design is a critical success factor in the implementation of the ES.

However, as experts note in the near future the existing significant gap between the requirements of the ES and the efficiency of the hardware and software design, the required amounts of verification and testing devices will only increase [56]. Mainly, the problem lies in the insolvency of traditional approaches to the design of the ES amid of

modern requirements for such systems. One of perspective directions in terms of the creation of modern design methodologies ES, is a complex accounting of the requirements specification and constraints within the overall project.

A developer of an embedded system must implement its full life cycle (LC), beginning from the development of technical documentation and ending with complex testing with the product and may be production maintenance. According to the carried out research, today Ukraine follows the national standards of lifecycle management systems and software. But, the issue of project documentation creation, which contains requirements for the created system, as well as its structure and contents, remain open. Practical application of the existing international standards in Ukraine requires preliminary work on their adaptation and implementation.

2.2. Embedded Systems Requirements Analysis and Creation of Project Documentation

2.2.1 Analysis of the requirements types for the ES

Working with the requirements is very important when creating a project of embedded system and achieves the goals for its implementation. However, existing for today standards, as well as work in the creation and analysis of requirements describe only certain aspects of this problem and do not include all the features of embedded systems design.

The system design, according to ISO/IEC 14288:2004 [47], includes the following technical processes work with requirements:

- stakeholder requirements definition process;
- requirements analysis process.

Stakeholder requirements definition process result is a list of all stakeholders and their system requirements, which are the basis for further analysis and identification of functional and non-functional requirements of the system.

Requirements analysis process result is a complete transformation holder (stakeholder) requirements in the technical vision of the developed system, i.e. a well-defined classification system requirements.

Requirements analysis process subsequently affects the architectural design, as well as the means to implement it.

According to standard GOST 34.602-89 [43], term “System Requirements” means the combinations of:

- requirements for the system generally (business requirements, external interfaces);
- requirements for the functions (tasks), which performs system (functional requirements, quality attributes);
- requirements for the types of support (constraints).

The first type of requirements also includes description of the system structure, which includes a list of its components. For complicated projects including a plurality of components (subsystems), it is necessary to produce a placement of high-level requirements on these components [32].

The standard GOST R 41904-2002 [44] gives a definition of an embedded system – a collection of hardware and software components designed to perform a specific function or set of functions. Since the software (SW) development is an essential part of designing the ES, necessary to adopt an integrated approach based on the processes of the system and software engineering, and to determine both the system (high-level requirements to the project) and the program requirements (i.e., requirements, distributed between software components of the project).

The first step to the joint set of standards that describe systems and software life cycle processes is the international standard ISO/IEC 12207:2008 [42]. It establishes a strict connection between the system and applied it in software, interpreting software as part of the system, by allocating requirements to the software from the system requirements. The standard shows a difference between the analysis of system requirements and requirements analysis of the SW product. In the general case, the construction of the system architecture defines system requirements for the various components of the system. Analysis of the requirements to software predetermines the requirements for them, based on the system requirements, assigned to each program component.

Addition to the above processes work with the system requirements in the standard ISO/IEC 14288:2004 [47], also in the standard RNS ISO/IEC 12207-2010 [42] has special processes of software intended to create a specific program element of the system, in particular the software requirements analyzing process. During this process, the requirements are determined by the software elements of the system and their interfaces,

the impact of software requirements on the operating environment is installed, as well as compatibility and traceability between the software requirements and system requirements.

In GOST R 41904-2002 [44] highlights the fact that the input to the process of determining the requirements for the software are the system requirements, the description of the hardware interface and system architecture (if they are not included in the system requirements). A goal of the design process software consists to develop software architecture and requirements of the lower level on the basis of top-level requirements for software. It follows that the definition of requirements for software development and the development of the architecture of the software only after the hardware description of the project and its architecture.

Leffingwell [19] discussed in detail the requirements management process and methods for developing software applications. Leffingwell requirements model consists of 3 levels requirements:

- needs – users/stakeholders requirements;
- features – system provides services to meet the needs of stakeholders;
- software requirements – concretized software requirements.

Cluster “needs” relates to the field of the users problem (customers) understanding, for the further construction of the system that satisfies their needs. The following two clusters “function” and “software requirements” refer to solving the user problem.

As noted in SWEBOK [51], in a variety of software development methodologies, an approach based on the definition of requirements groups to the software are used. This approach typically comprises group (type, class) requirements, for example: system, program, functional, non-functional (e.g., attributes quality) etc.

A classic example of a high-level structuring requirement groups as requirements of the software product, is presented in the book K.E. Wiegers [32]. Requirements model proposed by Wiegers for designing software consists of three levels of requirements:

- business – why should developing a system;
- user – describe the users of the system, as well as their work with it;
- functional – determine the functionality (behavior) of a software system.
- Model envisages two types of requirements:
- functional – that the system must do;
- non-functional – what conditions must be met when working.

In this requirements model under the external interface understand “user interface”, i.e. user interaction with the software. For the development of the ES in the standard GOST R 41904-2002 [44] provides a more expanded interface definition – it is the interdependence between two or more objects that share and ensure information or exchange them. As such objects may be: software configuration element (SWCE) – is a set of software components. And hardware configuration element (HWCE) – is a set hardware. Accordingly, the interfaces may be the following variants: SWCE/SWCE, SWCE/HWCE, SWCE/user, HWCE/user or module HW/ module HW.

Thus, of the SW requirements model presented in [32, 19] can be considered as a basis for developing a requirements model, taking into account the specific features and aspects of the design of ES. However, they must undergo extensive revision to take into account other types of requirements and work processes with the system requirements presented in the above standards [42-44].

2.2.2 Structure and features of requirements model for the ES

Based on the performed analysis, a requirements model for the design of ES was proposed. Schematic representation of which is shown in Fig. 2.2. Like the requirements model by Wiegers and by Leffingwell, has 3 levels, however, in contrast to the model Wiegers, does not use the concepts of requirements functional type and functional level. Since the ES represents set of hardware and software, the proposed scheme shows the distribution of levels in the model, on the requirements to hardware and software components of the system.

In the proposed requirements model highlighted in the following levels:

- stakeholder requirements – include requirements all project stakeholders, such as customers and end users of the product. Can be represented by: business requirements, use cases, quality attributes;
- system – system requirements, comprising a combination of interconnected components (hardware, software, users, etc.). Can be represented by: system requirements (the whole system), constraints, interfaces, required operating conditions;

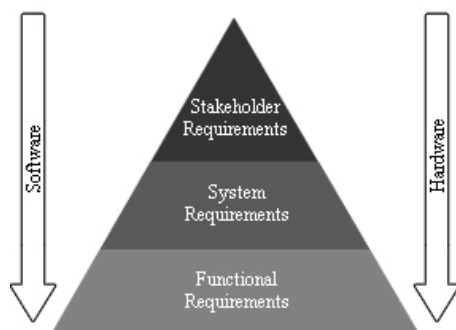


Figure 2.2 Requirements model for the ES

- functional – function (functional characteristics) distributed among software and hardware components of the system.

Table 2.1 shows the features of the proposed requirements model and the models Wiegiers and Leffingwell. Can emphasize such distinctions proposed model from Wiegiers:

- on the upper level are combined business requirements and user requirements;
- on the introduction of the second level are placed high-level requirements to developed embedded system and identifies the data requirements, interfaces, and system constraints;
- on the lower level, considering features of the ES, decomposes the functional requirements on the software and hardware components of the system.

Thus, requirements model for ES was developed based on requirements model of Vigers and Leffingwell. Within the model they implemented system global 3 requirements levels and 2 requirements types, as well as similar gradation for system soft- and hardware.

For requirements model which was developed provides requirements creation methodology for an embedded system (Fig.2.3), which includes a sequence of steps:

1. Identify business requirements: identification of the product concept which we want to create.
2. Identification of the users, combining them into a groups and fixing their requirements.

3. Definition of use cases (scenarios) of the product by describing the user's workflow.
4. Develop use-case: fixation of use cases and a thorough study of the sequence of interaction between the system and external actors (example of an effective way of formulating and documenting requirements of this type is shown below).
5. Emphasizing quality attributes: identification of high-quality system performance characteristics of concrete actions.
6. Identify: system requirements, including the whole environment, constraints, requirements to the operating conditions, interfaces.
7. Check the requirement specifications that have been developed for errors, completeness, unambiguousness, consistent and others. Desirable that in checking specifications involving all stakeholders.

Table 2.1 Distinctive features of requirements models

Proposed model	By Wiegers	By Leffingwell
3 levels of requirements: - stakeholder requirements; - system requirements; - functional requirements.	3 levels of requirements: - business requirements; - user requirements; - functional.	3 levels of requirements: - stakeholder needs; - features; - software requirements
2 types of functional requirements: - hardware; - software.	2 types of requirements: - functional; - nonfunctional.	
Interface: user, SW/ SW, SW/HW, SW/ user, module HW/ module HW	Interface: user, SW/ SW, SW/user	Additional category: user interface

After that follow the steps of determining the functional requirements for HW (8) and SW (9), which implies a more detailed identification and transformation of user requirements into a form useful for developers.

Fixation of functional software requirements in the specification of software requirements, may be performed by presenting the expected behavior of the system in the form of an «event-reaction» (example shown below). This form is convenient for the further development of test cases.

10. Checking developed requirements specifications.

11. Development of test cases: description of the states and transitions between states, as well as the conditions under which transitions from state to state.

12. Checking developed specifications to system requirements, functional requirements and use cases for errors, completeness, inconsistency, etc.

Creating requirements is an iterative process. The first three stages are usually carried out once (but they need to be reviewed and revised), the remaining steps are repeated for each new version of the product.

One of the important controls stages of the created requirements is traceability (relationship). To provide traceability, each requirement must be unique and identified to be able to refer to it. Best to manage traceability, using requirements management tool. Can be identified such as requirements management tools: IBM Rational Requisite Pro, Borland CaliberRM, DOORS. For Rational Requisite Pro, you can point out a number of advantages:

- support Word to create project documentation;
- availability of templates for the preparation of project documentation;
- easy navigator between Microsoft Word and powerful infrastructure of the database;
- integration with the tools of object-oriented modeling;
- the possibility of tracing the relationship of different levels requirements by Traceability matrix.

Traceability matrix created in IBM Rational Requisite Pro allows the modification of some requirements easily determine which requirements affect this change and what should be checked [35]. Traceability created requirements in accordance with the scheme requirements model in Requisite Pro is as follows (Fig. 2.4).

On the upper level there are Stakeholder Requirements (ID of requirements this type – STRQ), a and the System Requirements (SYST), made of stakeholder needs.

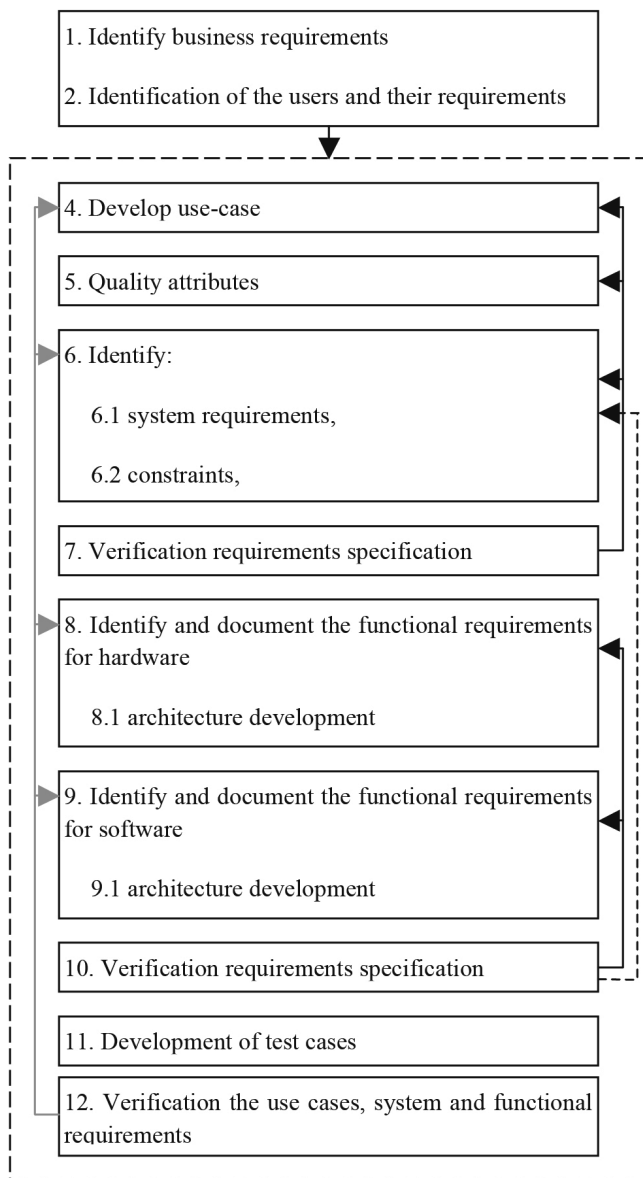


Figure 2.3 Methodology of requirements creating

Functional Requirements – it is functionality provided by the system, usually formulated by a business analyst, the appointment requirements – to meet the needs of the customer. Functional requirements are divided on the various subsystems of project on the basis the System Requirements: FEAT_HW – functional hardware requirements; FEAT_SW- functional software requirements.

Use Case (UC) – a description of the system behavior in terms of action sequences.

Supplementary Requirement (SUPL) – other requirements (usually non-functional), which can not be described Use Cases [21].

Implementation of moving platform ES control involved requirements model suggested as well as the very outlining requirements [22]. Examples below represent an interdependent requirements sequence related to a specific part of the system.

Example 1: User requirements

STRQ7: Altering a moving platform max allowed speed according to user category (kids, adults) and user experience level.

Example 2: Usage option.

UC3: Altering a moving platform speed.

Key parties: Operator.

Source environment: operator activated Master control station (MCS) and performed platform real-operation set up session.

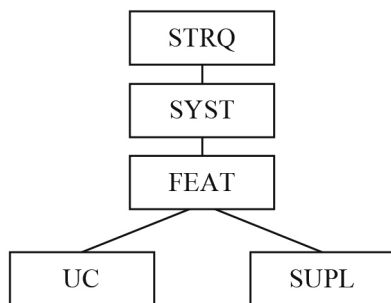


Figure 2.4 Traceability requirements in Requisite Pro

Original sequence:

1. Operator alters moving platform speed rate.
2. System alerts a moving platform about speed rate alternation.
3. A moving platform accepts the alert and processes it.

4. On a display one will see a message about speed alternation and an indication of a current speed rate.

Alternative sequence: does not exist.

Exceptions: alert was not sent and a system will display error.

Example 3: System requirements.

SYST4: System MUST alter (increase or decrease) max speed of a moving platform.

While switching between speed rates our system should display the speed rate we have switched to.

Examples 4: Limitations.

SUPL2: When two speed rates are on for moving platform max speed parameter:

- Kids mode – 6 km/hour.
- Normal mode – 20 km/hour.

In this particular case we are dividing the requirement into a few hardware and software functional requirements.

Example 5: Hardware requirements

FEAT_HW7: Moving platform engine should provide minimum 20 km/hour speed.

Software functional requirements will derive from hardware characteristics and existing requirements. Our actual experimented system is equipped with an engine which generates minimum speed of 60 km/hour, and so our software requirements are based on this fact and are depicted in the example below.

Example 6: Software requirements.

Event: Switching speed into a Kids mode from MCS.

Reaction: Upscale original speed by 10%. Speed change confirmation shows up on the screen as «Speed Mode: Kids» message.

Event: Switching speed into a Normal mode from MCS.

Reaction: Upscale original speed by 30%. Speed change confirmation shows up on the screen as «Speed Mode: Normal» message.

Find more examples of parties interested in implementation of a moving platform ES control requirements below:

- STRQ1: the object (platform) must be controlled remotely;
- STRQ3: create on-platform experience for the person in control of the platform;
- STRQ10: provide a system to trace and collect system states during operation.
- Such requirements later developed into a high lever system requirements:

- SYST3: data must be transmitted from control center to the object by means of airwaves. The source of airwaves should be no farther than 40m from the object;
- SYST4: by means of airwaves a video image must be transmitted from a camera, mounted on the moving platform, to video-glasses;
- SYST12: provide a tool to save system state data to a flash drive, which is a part of MCS (this exact requirement may be traced into a software functional requirement: a system must store MCS data in txt files with separators).

Technical documentation was created in order to support success of project development by means of IBM Rational RequisitePro requirements management system:

Glossary – terms used in the project;

- Vision – full system description, full system level features description (system requirements);
- Use Case Specific – use case, algorithm;
- Functional Specification – functional requirements;
- Supplementary Specification – functional requirements which are not related to use cases or nonfunctional requirements.

Based on working with requirements results via RequisitePro, we were able to produce full set of documentation to describe full range of requirements for moving platform ES control developed. System architecture was developed when full list of stakeholders requirements was put together. Requirements analysis for HW and SW and HW architecture is also based on that information.

Practical application of the proposed model and methodology allowing performing efficient requirements development and implementing successfully a creation of ES motion control platform project.

2.3. The basic concept of embedded systems design

Currently experts [1,31,33,56] represent and analyze various high-level embedded systems design methodologies :

- object-oriented design;
- hardware and software parallel design;
- platform-oriented design;

- aktor-oriented design;
- multi-design;
- aspect based design.

Parallel hardware and software design (Hardware-Software CoDesign) is regarded as one of the promising ES design approaches. This methodology has both advantages and shortcomings. On the one hand, the complexity of the embedded systems design process, on the other hand, a significant characteristics improvement of the end product compared with alternative versions of the design decisions.

Overall, the design process of embedded systems is shown on Fig.2.5. The process of designing hardware and software is distributed in a certain time approach (Fig. 2.6).

The current practice of distributed design shows that hardware solutions are usually accepted initially. Further, program superstructure performs on this basis. This in turn can lead to a high risk delay of the entire product design cycle [31].

After hardware and software components are implemented, they must be individually tested, integrated, and then tested one more time as the entire system. At hardware and software integration stage there are also problems because of the deadlines for system integration. Often there is a must to start work with an incomplete set of ready-made solutions.

As the market is constantly evolving and requires more complex systems in a shorter time, the development type «from scratch» is becoming ineffective. Therefore, in the embedded systems design it's popular to use ready-made solutions to accelerate product development and thus reduce entering market time.

Today, many manufacturers (Arduino, Texas Instruments, Parallax Inc., Netmedia, Microchip, Digilent, MBED, Raspberry PI, Cyclone etc.) offer ready-made hardware-software platforms, each of which has the form factor and functionality, which depends on the problem being solved. In this case, ES designer must know existing market platforms and be able to make a responsible decision on the project specific hardware and software platform. Unfortunately, information offered on the sites does not always make the right choice and implement the project optimally. Purchase SMEs huge number of different platform, despite low prices is still unacceptable [54].

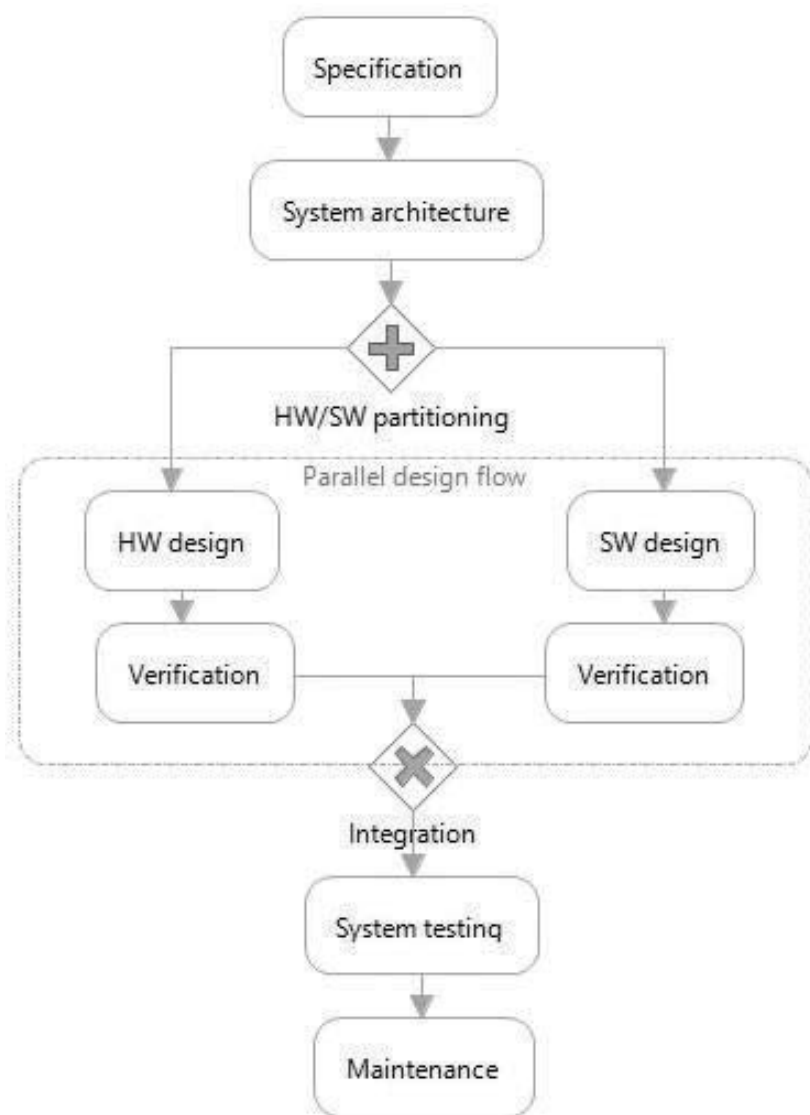


Figure 2.5 Classic embedded systems design process

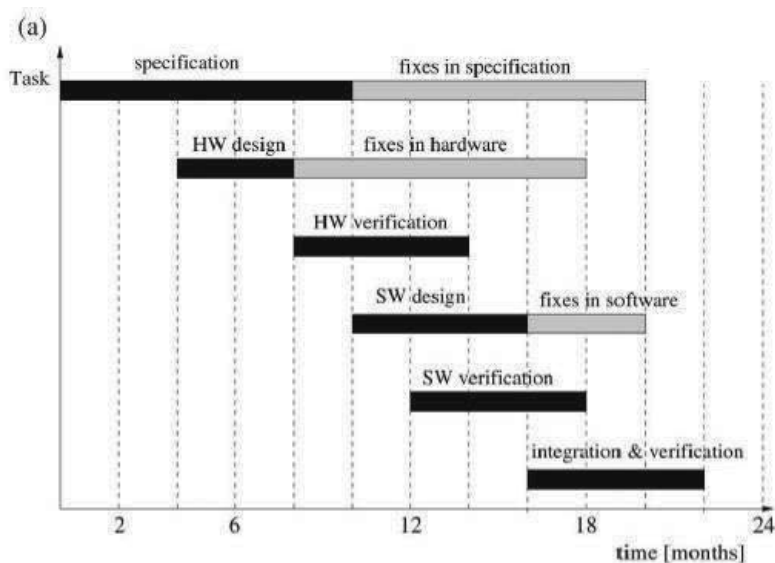


Figure 2.6 Time chart of classic embedded systems design process

With the rapid increase of the popularity of ready-made hardware and software platforms, some companies (eg. Autodesk) offer programs of virtual simulation of their work. For example, simulators for Arduino could be the following: 123D Circuits, Virtualbreadbord, Simulino, Virtronics, Simuino, Arduino Simulator and others. [4,13,28,29,37]. However, the real work cannot be replaced by simulation. In fact, instead of the actual physical process, simulator allows to use his mathematical model only. Many programs are not free, require time to study them, have limited functionality and incomplete element base.

There are also powerful ECAD (Electronic Computer-Aided Design) systems, such as Altium Designer, AutoCAD Electrical and others. However, the use of their full functions stack is not always necessary, and the cost is quite high.

Thus, one of the key areas to improve ES design efficiency, is the accumulation of technical solutions for reuse. The diversity and fragmentation descriptions of already developed components (hardware blocks, algorithms, software implementations, etc.) prevent their reuse, but the use of remote experiments allow designer getting information for decision making in order to implement ES.

Therefore, the creation and application of specialized remote laboratory for complex hardware-software design of the embedded systems is an actual task, because the problems existing in this area demand qualitatively new techniques, technologies and tools of ES design.

The use of remote lab amends design process of embedded systems (Fig. 2.7).

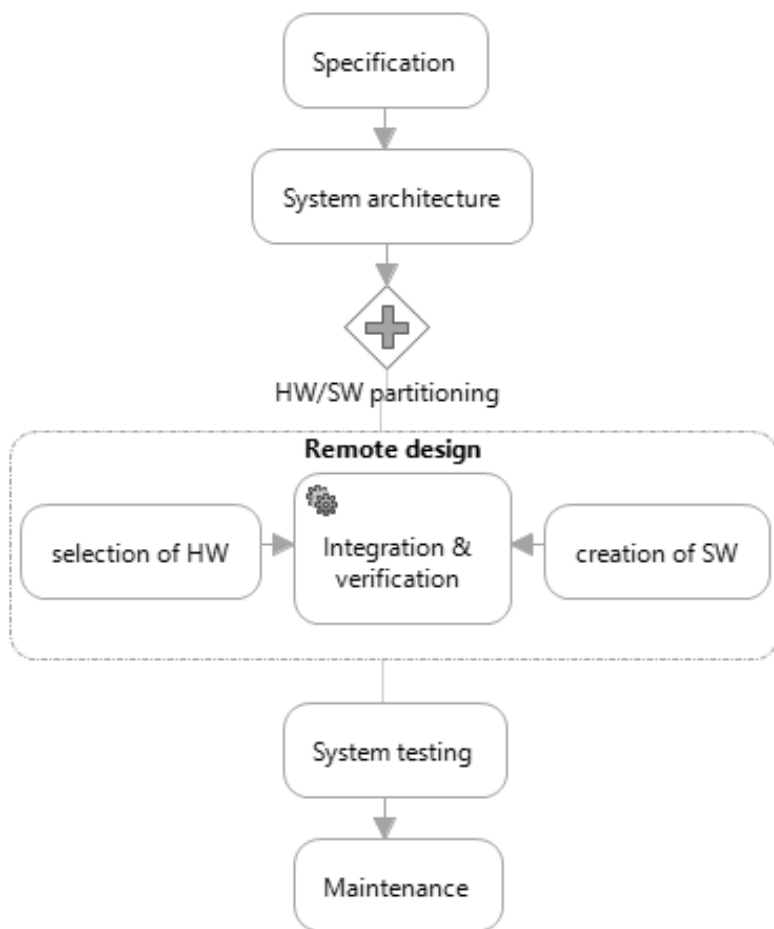


Figure 2.7 Embedded systems design process using remote experiment

With remote lab designer is able to:

- receive and analyze information of the specifications;
- handle selection of ready hardware solutions;
- perform design and software verification;
- perform the integration of ES hardware and software;
- test the compatibility and performance of designed system hardware and software components;
- observe the experiment on real hardware.

Remote labs give developers a chance to estimate platforms capabilities, make decisions about its integration, thus lowering a risks and development expenses [23,24]. Using capabilities of remote lab, designer can more optimally organize the project and realize it in a shorter time. A timing diagram of the project with using a remote lab can be changed, as shown on Fig. 2.8. Compared with the Time chart on Fig. 2.6, the time-to-market of products reduced and released time for research and development of innovative solutions.

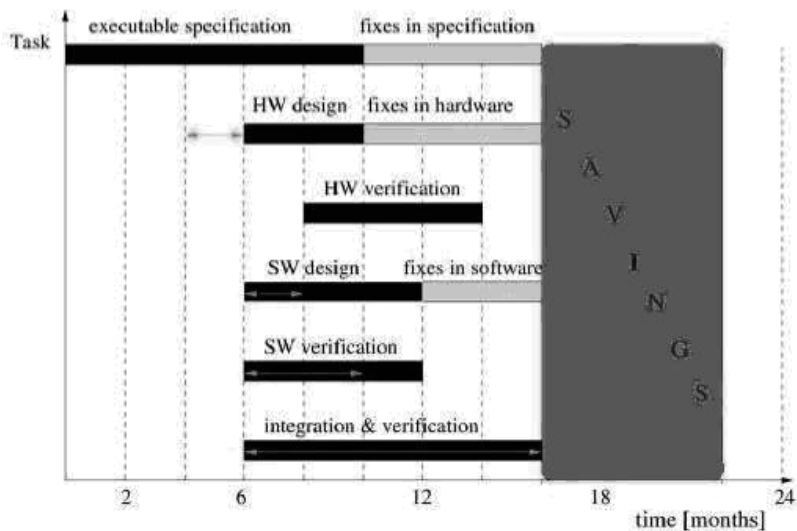


Figure 2.8 Time chart of embedded systems design process with remote experiment usage

2.4. Approaches to Embedded Systems Hardware Realization

Hardware part of current embedded systems can be created with the help of different technologies: a microcontroller (MCU), FPGA (Field-programmable gate array), ASIC (Application-specific integrated circuit), etc. (Fig. 2.9). All technologies for developing embedded applications have both strengths and weaknesses. In most cases, the choice depends on a complex combination of factors and none of these technologies will act as a perfect solution. Usually, the choice of implementation technology is conditioned by functional system and its requirements [57].

It should be noted that in order to popularize and promote their products the MCU developers created debugging boards (platforms). The most known today are: Arduino, FreeDuino, etc. They are equipped with all the necessary hardware and can be complemented with additional modules, which are targeted to expand the functionality of the base board.

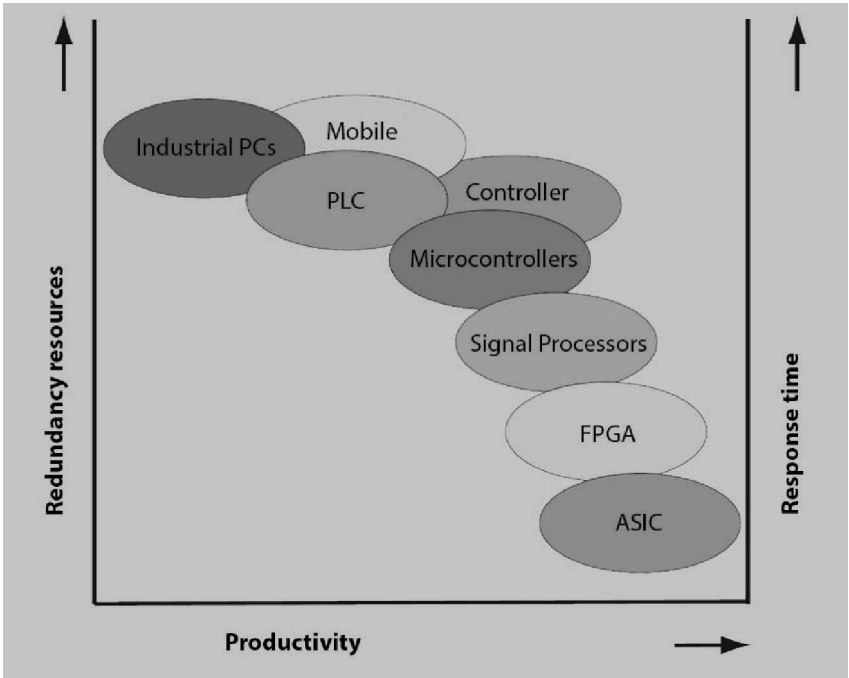


Figure 2.9 Embedded systems implementation techniques



Arduino



Freeduino



MBED



Raspberry Pi

Altera
Cyclone

Figure 2.10 Ready platforms examples

Arduino – soft-hardware trademark of building simple automated and robotic systems tools, for pro users mainly. Free software shell (Integrated Development Environment – IDE) for creating and compiling programs and hardware programming composes a software part of such tools. However, hardware part of it is represented as mounted printed cards, which are supplied by both: official distributors and side representatives. Open source architecture of Arduino provides a possibility of changing and supplementing their product line. Arduino can be used for either building independent automated objects or connecting to an existing PC software through usual wire and wireless interfaces.

Arduino concept does not cover case or mounting structural elements. It is up to a developer to decide on cards installation method and technical security choices. Side vendors provide automated robotic mechanics manufactured specifically for Arduino cards systems.

Arduino product line mostly uses Atmel AVR ATmega328, ATmega168, ATmega2560, ATmega32U4 16 and 8 MHz clocking microcontrollers. Less recent items operate with ATmega8, ATmega1280 etc. There is also ARM Cortex M3 AT91SAM3X8E 84 MHz clocking processor cards. They do not require a programator due to a loader (BootLoader) being weaved first. However, most cards are using male end for in-system programming [5].

Freeduino is a join-production open source project, targeted mostly to a replication and producing of Arduino-compatible hardware platforms. *Freeduino Eagle SCH, BRD and Gerber* format models provides users with an opportunity of creating functionally, electrically and physically 100% Arduino compatible cards.

While Arduino is a protected brand, *Freeduino* has a free license and so has no use case limitation.

There are also Arduino analogues among other existing platforms, such as: *Mega 2560, UNO, Duemilanove, Nano, Diecimila, Diecimila*. Aside from that there are also extensional cards, so cold shields.

Their technical characteristics are almost identical to those of Arduino cards, their assemblage varies from a manufacturer to a manufacturer and *Freeduino* designers. Of course the most important difference from those Arduino products is a significantly lower price.

Freeduino is applicable at almost every electronics field, any system related to algorithmic system control where a reaction to outside events is required.

One can easily design a control system for any electrical devices, using *Freeduino*: light chasers, signalization, stepping motor, LED display, it may also be used together with level, movement, weight sensors in order to measure physical units etc. [14,46].

Raspberry Pi is a compact monocard computer. It has analogue and digital outputs for video signals well as USB, which can be used to connect a keyboard and other components. As a multi-operated system it is designed for Python and Scratch among others. It can easily handle a work load of a regular PC. *Raspberry Pi* also has network interface and general outputs, which can be used to connect it with a wide range of system components.

It was design as an educational device, in order to introduce users to a deeper knowledge of the world of computer science. It supports several Linux distributives, as well as RISC OS for RISC computers.

It actually can be used in so many ways, e.g. low capacity dedicated home server. Another example would be an expansion card for smartphones, used for an infrared imager. And don't forget about a "smart house" solution, which is very popular today [52,59,61,63].

Altera Cyclone® – is an FPGA designed as a low price product. It is a perfect solution for a wide range of low price and low priority problems due to the embed memory, external memory interface and s clock rate *Cyclone* control cards.

Cyclone is a low price alternative for integrations specific integrated circuit (ASIC), which are very popular in embed systems of nowadays. Low price of a Cyclone brought FPGA to massive production. FPGA is a design basis for digital devices of different complexity. They are cases which require big amount of input and output points (over 1000 of outputs), digital signal processing, digital audio and video devices, high speed data transmission, cryptography, ASIC designing and prototyping, breaks to connect systems with different algorithms and source voltage, neurochips development, design of quantum calculations etc. [53].

Mbed is a platform for building intelligent devices. It is based on 32-bit ARM Cortex-M architecture. Its task is to provide high capacity and high speed prototyping and production. It's mostly for Internet connected devices.

Mbed development kit is a strictly online one. Program code is created and compiled in IDE within AJAX-component Internet. It's integrated with online tools – network community support forms, code and documentation exchange.

Mbed goes with a set of high-level object oriented libraries, which connect o most of input-output interface units on a card. Mbed controllers are meant to control online operating devices. You can think of a “smart” housekeeping devices and “smart house” as an example of such devices [20].

Usually, the choice of technologies of embedded systems realization is conditioned by functional and system requirements for it. During realization of moving objects control system (Fig. 2.11) the use a microcontroller offers several advantages: compactness, low cost and minimal power consumption, which is important for battery-powered devices such as moving objects. Moreover, according to specialists, the MCU-based systems are more flexible because the control program can be repeatedly processed without changes in hardware [7,9,36,39,41,49].

Today there is a classic approach to creating microcontroller systems (MCUS), shown on Fig. 2.12 which attracts the vast amount of developers [60]. This approach is time-consuming, requires a lot of time on system planning and a high level knowledge of circuits-, electrical-engineering, programming MCU, etc. (study of the datasheets and features of the used device, creating circuit board project, program debugging).

A new approach based on the debugging platform by using ready-made building blocks makes a design stage much easier. Moreover, specialized software, which is supplied with the board, high-level programming language and lots of standard libraries, simplifies the writing and debugging programs [25].

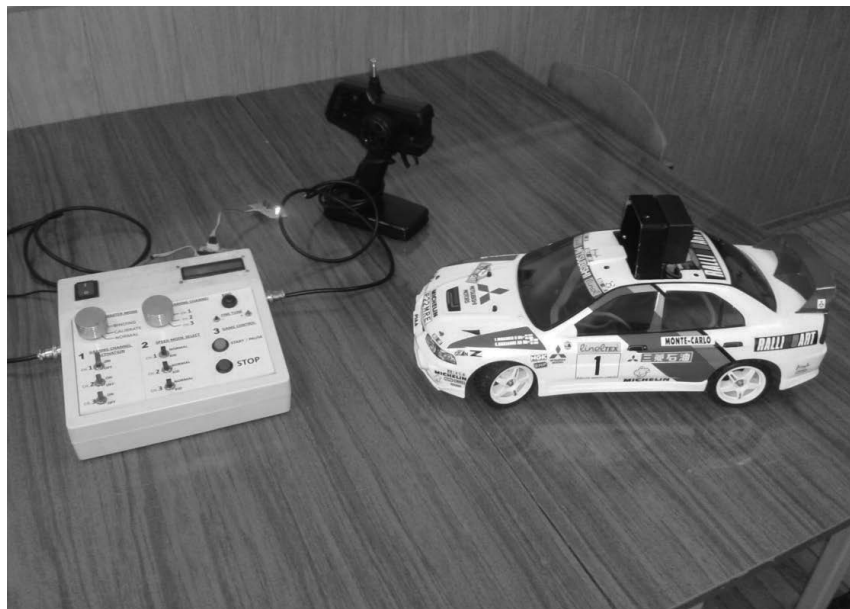


Figure 2.11 Moving object control system

It must be noticed that bringing such platforms to a broad market has set the task of universal access and work with these devices. Template principle caused problems while using such platforms. Thus, for the creation of complex systems it is necessary to combine several platforms, which leads to significant cost, size and weight and is unsuitable for embedded devices. On the other hand, it makes no sense to use a ready platform for the organization of simple devices (for example, traffic light), because it will cost more and full functionality of the chosen platform will not be used.

Thus, sometimes a lot easier to use a few MCU with independently developed board so that each module performs a specific function. This method requires more systems planning time and knowledge, but lacks the disadvantages of using universal platforms and more, has advantages such as: lower power consumption, the relative ease of debugging and longer service life, which is achieved by reducing the number of used components and connectors.

So, a range of factors must be considered to make a correct choice of approach and the most important of them are functional and design requirements.

2.5. Approaches to Embedded Systems Software Realization

So, success of product promotion on the market depends on the time spent on development, including SW development time, which depends on the approaches that are used in the developing.

Look at Raspberry Pi as an example. It's using Linux family OS or RISC OS, which requires scripts as a software and/or other binary executed files, depends on an OS. When talking about a design of microcontrollers systems the software is represented as a set of such microcontroller computer commands using assembler language or C. Developers suggest using library kits and development environments which simplify development process for SW-HW microcontrollers based systems. FPGA based systems have two development levels: hardware description language (e.g. VHDL), designed microprocessor system language (e.g. C).

The analysis of used languages for embedded systems programming shows that in most cases this is a family of C languages. Approximately a third part of embedded systems developers must still use Assembler language for a small part of low-level procedures on some devices. The result of this analysis is shown in Fig. 2.13.

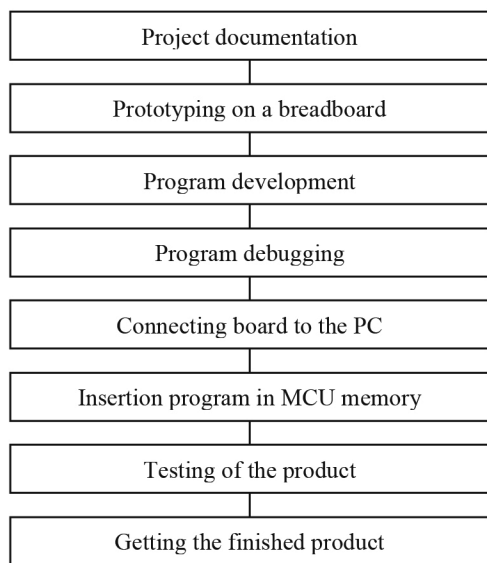


Figure 2.12 Stages of ES design

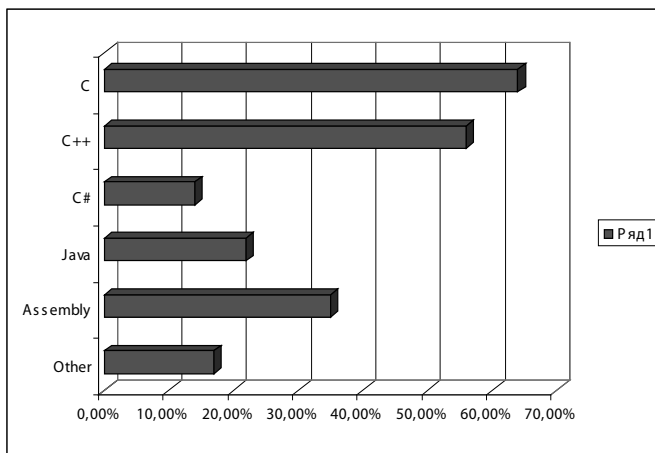


Figure 2.13 Languages of ES programming

The software part of an embedded system can be realized using a real-time operating systems (Windows CE, QNX, OnTime RTOS, VxWorks) or by organizing the program cycle (using of the approach «supercycle» or the one with a switch-case). If the system is very simple and has short reaction time, it is easier to make it according to the pattern «supercycle». If the system is complex, combining many different actions and reactions, which are also critical to a time, then it is more appropriate to use the pattern «switch-case» or real-time operating systems (RTOS).

Most of ES, are manufactured today, are real-time systems. This imposes additional constraints on the software development for ES.

After analyzing the definitions of «real-time system», which are given in different works, we can summarize that the real-time system must provide: to handle many simultaneously external events; timely execution of data processing; predictable behavior of the system to external input; limited reaction time to input signals.

As a result, search for the designing methods of software for ES, the two existing approaches have been identified. One of them – is the use of RTOS, the second – the organization of the program using the IDE [45,65].

Analysis of the work on this topic showed, that the RTOS have advantages and disadvantages [38,50].

RTOS – is a scheduler asynchronous parallel task in «real time». The order in which task are performed is determined the Scheduler. It

alone ensures that the high-priority task was performed first of all, and the low-priority – only when there is time. Without the RTOS you must implement their own timers and delay necessary for the implementation of the program with lots of finite automata, which is a laborious process.

Real-time operating systems have a modular structure and the kernel (main core), which does not depend on the characteristics of the microcontroller. This allows transferring them the various MCU. When writing a program in Asembler, it is difficult to do, which is related with a different command system and presence of MCUs of various functional blocks. Another advantage of RTOS: if you suddenly make a mistake in the code, the «fall» only the task in which the error is present (it is also possible that all the tasks with a lower priority, too), providing overload protection.

However, use of the operating system does not solve all problems in software development. First, the use of RTOS relatively limited powerful microcontrollers. Since the operating system for their work requires microcontroller resources: memory and system time, which is very small (minimum requires MCU for work RTOS: 512 bytes of RAM and 4-16 KB of program memory). And instead of that put them on the tasks, necessary to give the Scheduler. Secondly, the RTOS is used where it is necessary to organize the execution and interaction of multiple program streams. Application development, consisting from the set asynchronous streams running often with different priorities, requires the programmer to use sophisticated means of interaction flows. Plus, the need for additional knowledge and skills to the organization is not a very simple task: mutexes, semaphores, priorities, etc.

Thus RTOS is a necessary tool for creating complex multi-tasking real-time systems. Helps simplify and reduce time SW development. But provided opportunities allocating MCU additional resources for the work of RTOS, and experience with the OS.

When using the IDE, solution of designing SW reduces to choosing variant of the organization the program cycle: «supercycle»; «switch-technology».

The program is modeled on the «supercycle» looks like an endless loop in which MCU consistently doing what it should do.

In the ES, which has an outer periphery, MCU can operate it in two ways: to interrogate or using interrupt. If the first case, we will periodically interrogate whether the data, until they are received. In the second case, MCU configure so as to generate an interrupt whenever new data. The peculiarity of work interrupts, such that an interrupt handler (code

that is called directly at the time when the interrupt occurs) should be as short.

If, for example, need to organize software overload protection that you need to start work as soon as possible after the event of an overload (otherwise it will burn), the total cycle will not allow it to do, because all the events in it are executed in order. And guaranteed response time to an event can not be less than the execution of one loop iteration. And if somewhere in this cycle will be an error – then the whole system stops working.

This structure is convenient to perform a very small number of tasks. If necessary, the system interrogate dozen sensors, responsive to a number of critical events, process commands is sequential overall cycle will not allow it to do, because for each event has its limitations in time and frequency response interrogate or control.

The second option software implementation – «switch-technology» (basic design – automaton). Its development was by professor Shalyto A.A., in 1991.

Switch-technology is also called «automata-based programming». It submission program as a transition graph of a finite automaton based. Switch-technology is called from the C language switch statement, which is the main structure. In this case, the organization of tasks, though not simple, but reduces to determining where what function to run. After completing a task, the program returns to the supercycle.

Use of switch-technology in some cases eliminates the need for an RTOS (however does not exclude) for complex and multi-tasking systems. Firstly, the interaction between automata simpler than between threads in RTOS. Secondly, the use of automata makes the behavior of the program is deterministic, and the interaction flows in RTOS requires additional (and considerable) efforts to resolve conflicts.

The first advantage of switch-technology: program built on the proposed technology, it is easy to document. It is a set of finite automata, which can be expressed in graphic form with constraints and internal structures automata. Second advantage: the ability to reuse code. Automaton in the proposed programming style is a kind of “brick”, a standalone unit program. His connections with the rest of automata minimized and standardized. It can be developed separately and then used in various SW projects. From this it follows the third advantage: program easily modified, since the number of links between automata minimal [65].

So, if the system is very simple and has short reaction time, it is easier to make it according to the pattern “supercycle”. If the system is

complex, combining many different actions and reactions, which are also critical to a time, then it is more appropriate to use the pattern “switch-technology” or RTOS.

Each of these approaches has its advantages and disadvantages, and the choice of a particular approach depends on the objectives of the project, and the skills of the developer.

2.6. Control Questions

- Areas of embedded systems application.
- Requirements types for the ES.
- Structure and features of requirements model for the ES.
- Methodology of requirements creating.
- Approaches to Embedded Systems Hardware Realization.
- Approaches to Embedded Systems Software Realization.

2.7. Practical Exercises

- Using one of requirements management tools (IBM Rational Requisite Pro, Borland CaliberRM, DOORS) perform the development of requirements specification for moving objects embedded control systems.

3 EMBEDDED SYSTEMS DESIGN USING VIRTUAL AND REMOTE TOOLS

3.1. ES Integrated Development Environment

An important part in ES designing is the modeling and debugging of future electronic device using ECAD-system. In this area today there are many solutions: Altium Designer, AutoCAD Electrical, Proteus VSM, etc.

Altium Designer – integrated system design of electronic devices based on printed circuit boards, which allows developers to create projects, from concept and VHDL-description of the FPGA, to carry out simulation derived schemes and VHDL-code, prepare files for production, and the concept of Live Design so-called living design, allows us to complete the project on-board debugging NanoBoard. Monitoring the integrity of the project allows you to track changes in the parts of the project and synchronize them. Using the fee debugging NanoBoard possible to debug FPGA projects at the stage of creating the circuit diagram [2,3,55,62,64] (Fig. 3.1).

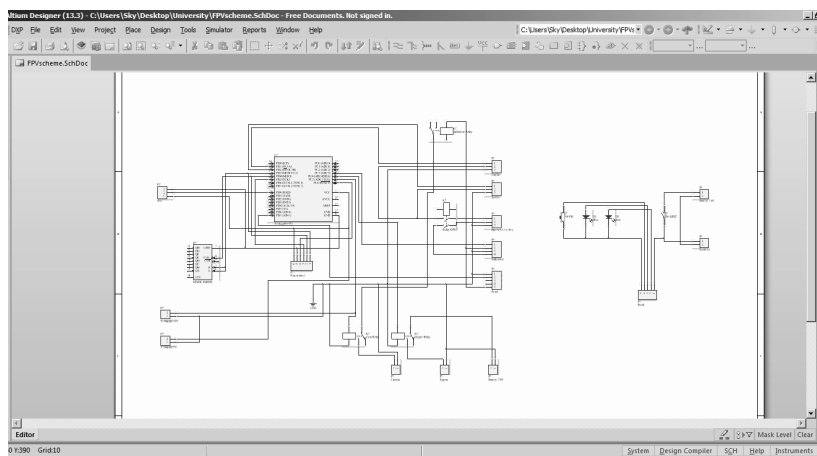


Figure 3.1 Altium Designer project example

PROTEUS VSM – simulator of microcontroller systems and electronic circuits. Supports MC: PIC, 8051, Atmel, HC11, ARM7/LPC2000 and others. Allow reliably simulate and debug fairly sophisticated devices

that could contain a few MCU at the same time and even different families in one device. Proteus Pack consists of two parts: ISIS – program of synthesis and simulation of electronic circuits and ARES – PCB design software [26] (Fig.3.2).

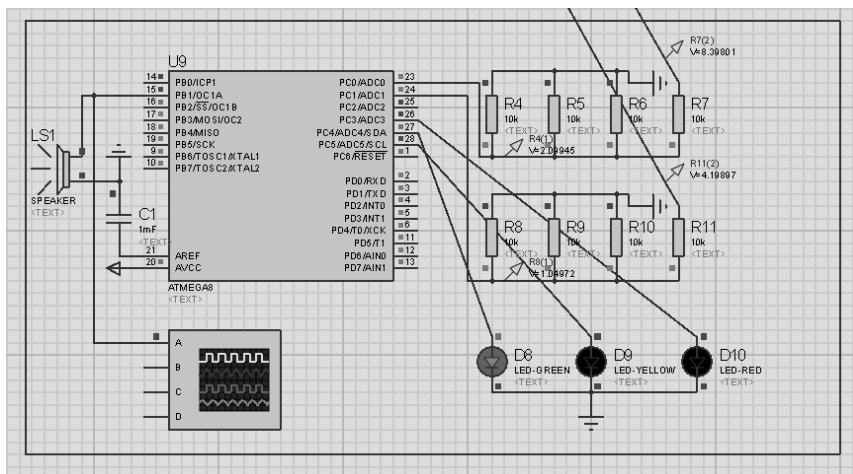


Figure 3.2 Proteus project example

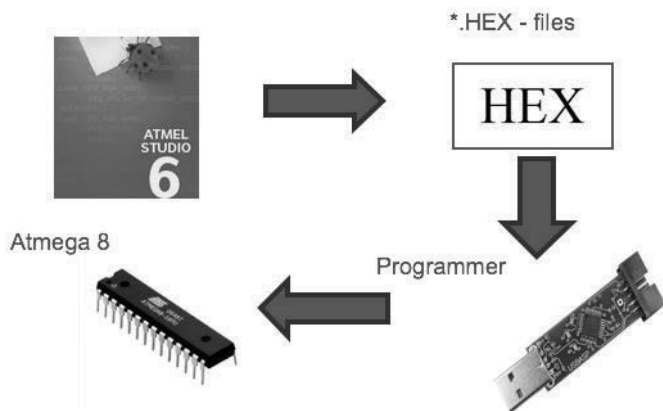


Figure 3.3 Main stages of project

Modeling of programmed devices: MCUs, microprocessors, DSP and others, is carried out by loading in the program *.hex file. A hex-file is created with special tools (e.g., AVR Studio, WinAVR, Atmel Studio). After developing models of electronic modules and applications for the MC, is carried out modeling of the system as a whole, by combining in a single Proteus project all system modules and upload *.hex-files. It reveals the mistakes that have been made or program, or into the scheme, as well as track and adjust the algorithm of the MC. Then, hex-files upload in microcontrollers' program memory and are tested on device prototype (Fig. 3.3).

Atmel® Studio is the integrated development platform (IDP) for developing and debugging 8- and 32-bit applications from the Atmel company. The Atmel Studio IDP gives you a seamless and easy-to-use environment to write, build and debug your applications written in C/C++ or assembly code. Contains the assembler and the simulator that allows to trace implementation of the program. Atmel Studio is comprised by the project manager, the editor of an output code, the instruments of virtual simulation and in-circuit debugging [6] (Fig. 3.4).

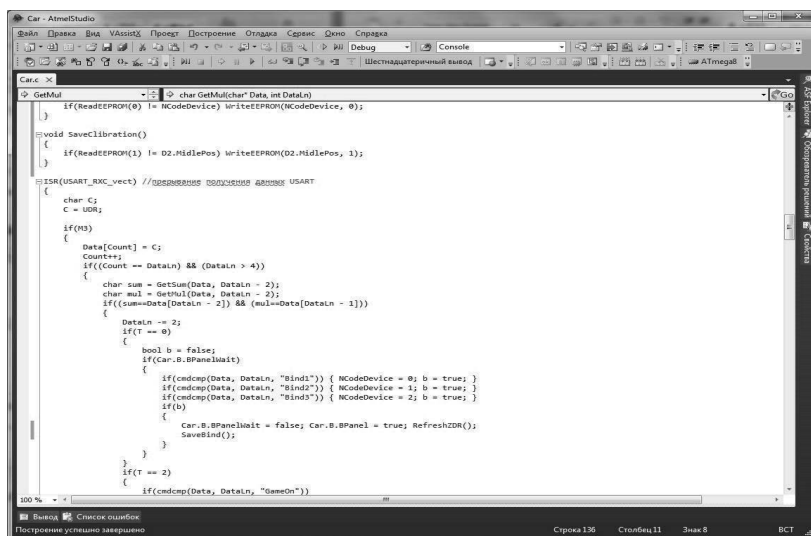


Figure 3.4 Atmel Studio working environment

Arduino – a convenient rapid development platform of electronic devices for beginners and professionals. The platform is very popular worldwide for its convenience and ease of programming language, as

well as an open architecture and software code. The device is programmed via USB without programmer. The microcontroller on the board is programmed using the language of Arduino (based on language Wiring) and IDE Arduino (based on medium Processing). Devices projects based on Arduino, can work independently, or as to interact with the software on your computer (Flash, Processing, MaxMSP). The boards can be assembled by the user or purchased ready-made. The software is available for free download [5].

Having ready models of electronic modules and programs for MCU we conducted the modeling of the whole system, combining all the system modules in a Proteus project and downloading *.hex-files. It helped to detect errors that were made either in the program or in the same scheme, and track and adjust the algorithm of the MCU. Then, hex-files were loaded into program memory of MCU and tested on the prototype.

The virtual prototype of the mobile platform can be designed and researched among any MCAD-systems. PTC Creo is a scalable, interoperable software for computer aided product design. It helps teams create, analyze, view and leverage product designs via 2D CAD, 3D CAD, parametric & direct modeling [8,30,55] (Fig. 3.5).



Figure 3.5 Creo project example

3.2. ES Practical Design Using Remote Experiment

3.2.1 The specialized remote laboratory RELDES

Today, in the field of embedded systems (ES) design is very popular use of ready hardware and software platforms allowing to accelerate development of products so to reduce the time to market.

The specialized remote laboratory RELDES (REmote Laboratory for Design of Embedded Systems) is focused on the help to the developer in a fast and effective choice of standard design decisions on the basis of ready hardware-software platforms.

Laboratory provides remote access to Arduino based experimental equipment. Arduino is the powerful microcontroller with large volume of memory that allows you to create complex electronic devices. Besides, the hardware-computing Arduino platform is one of the most popular and simple in studying. In comparison with other similar platforms, Arduino has a number of advantages:

- the project was developed and develops as the project with an open code which works as the network project / community, allowing participants to exchange experience and ready applied practices, further accelerating the process of development and debugging;
- the low cost of the microcontroller and expansions to it;
- the simplicity and cross-platform of the programming environment (OS Windows, Macintosh OSX and Linux: 32/64bit).

The laboratory allows users: to extract and analyze information about specifications of ready hardware and software components for their reuse, perform selection of hardware platforms, development of program code, tested for compatibility and working capacity of hardware and software parts of the projectable system. Remote testing of components allow quickly assess the feasibility of their application and decide on their integration into the project, thereby reducing the risks and the time expenses for design.

The set of experiments depends on specifics of the projected ES. Today, as standard decisions for control system of mobile objects design, experiments on the basis of servomotors, liquid crystal display, LED traffic light and sonic distance sensor are offered. The proposed set of experiments may be expanded with other hardware-software platforms and tasks related to various types of embedded system solutions.

The stand with experiments connects to the server of laboratory via the serial interface. The computer with Linux Ubuntu OS acts as the server of remote laboratory. The server provides access to experiment programming and video stream of the laboratory. Video stream is implemented with use of “ffmpeg”. The server processes web client requests and carries out the following actions depending on the obtained data:

- compilation of received initial code;
- returning the compilation results;
- uploading binaries to the microcontroller (provided that the controller is free);
- client queuing (if experiment is occupied).

Client-server communication is carried out via web-interface implemented with HTML+CSS+JS. The service is running on the Apache server with use of the relational MySQL database. Besides web browser no additional software is required to access the remote laboratory (Fig. 3.6, 3.7, 3.8).

The laboratory provides users with information on software and hardware solutions for further implementation; allows them to obtain skills in embedded systems programming. The scenario of use of laboratory provides both the use of ready templates of programs for bench devices and the development of programs by users from templates.

Benefits of our laboratory:

- cross-platform (use any device and operation system);
- accessibility (use our hardware instead of spending money);
- configurability (write your own code or use code templates).

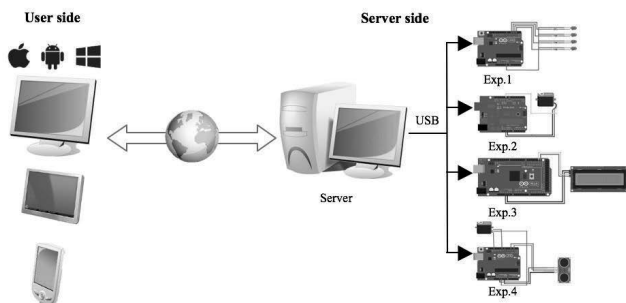


Figure 3.6 RELDES remote lab architecture

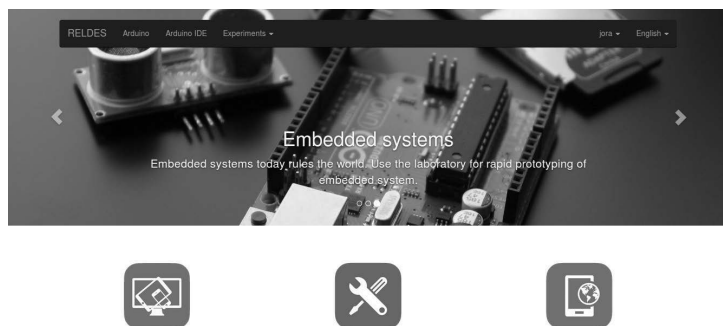


Figure 3.7 REDES lab general interface

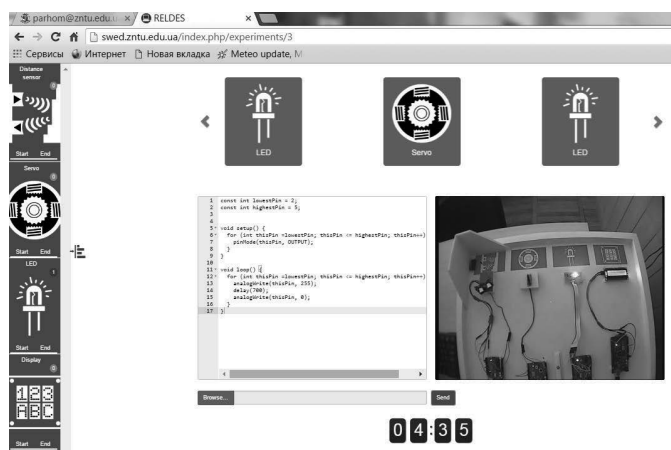


Figure 3.8 REDES interface

Practical application of the developed laboratory will allow the designer to make the right design decision on expediency of application of this or that ready platform, and also possible options of development of the project [23,24,27,54].

3.2.2 Description of laboratory REDES set of remote experiments

Experiment 1. Traffic Lights.

This experiment allows users to control set of light-emitting diodes (LED) with Arduino board. LEDs are connected to digital pins of the board. Resistors are used to restrict the current flowing through the

diodes. Diodes have common ground. (Fig. 3.9, 3.10). Scenarios of experiment: Sequential lights, Running lights, Brightness change.

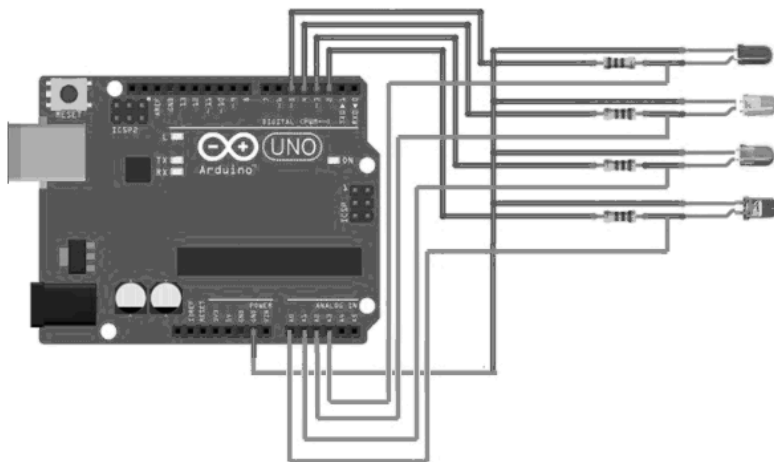


Figure 3.9 Model of the experiment Traffic Lights

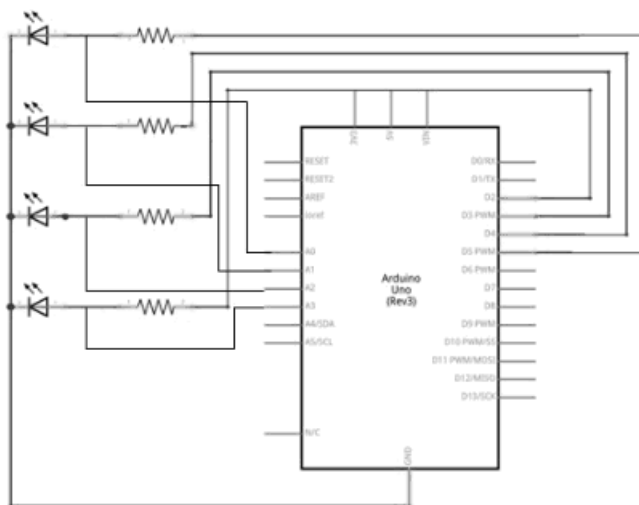


Figure 3.10 Traffic light project implementation scheme

Examples of experiment realization describes below.

The code consists of *setup* and *loop* functions. *Setup* is called only once when board powers on. In this case in the function pins from 2 to 5 are set up in output mode. Function *pinMode* is used for that. Its first argument is pin number and second is pin mode. *Loop* function is called continuously during all the working time of the board. In this function all the work is done. To control diodes' brightness *analogWrite* call is used. Its arguments are *pin number* and *voltage value*. Voltage range is 0-5V. Voltage value is not used directly as parameter of *analogWrite*, value ranged from 0 to 255 is used instead. Each value in this range is corresponded to voltage from 0 to 5V. It is done by so called pulse width modulation (PWM). To make it possible for user to see diodes brightness change function *delay* is called. It stops board's work for amount of time (in ms) stated as argument of the function.

Scenario 1.1. This example displays sequential activation and deactivation of every diode with 700ms delay.

```
const int lowestPin = 2;
const int highestPin = 5;
void setup() {
  for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
    pinMode(thisPin, OUTPUT);
  }
}
void loop() {
  for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
    analogWrite(thisPin, 255);
    delay(700);
    analogWrite(thisPin, 0);
  }
}
```

Scenario 1.2. This variation of a loop function performs in-turns activation of first cycle diodes and their in-turns deactivation in a second cycle. Cycles happen in turns as well.

```
const int lowestPin = 2;
const int highestPin = 5;
void setup() {
  for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
```

```

pinMode(thisPin, OUTPUT);
}
}
void loop() {
  for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
    analogWrite(thisPin, 255);
    delay(300);
  }
  for (int thisPin =lowestPin; thisPin <= highestPin;
thisPin++) {
    analogWrite(thisPin, 0);
    delay(300);
  }
}

```

Scenario 1.3. Brightness level is changed gradually for each diode in its own turn, going from zero to top and backwards. Brightness is changed in nested iterations.

```

const int lowestPin = 2;
const int highestPin = 5;
void setup() {
  for (int thisPin =lowestPin; thisPin <= highestPin
; thisPin++) {
    pinMode(thisPin, OUTPUT);
  }
}
void loop() {
  for (int thisPin =lowestPin; thisPin <= highestPin
; thisPin++) {
    for (int brightness = 0; brightness < 255; brightne
ss++) {
      analogWrite(thisPin, brightness);
      delay(2);
    }
    for (int brightness = 255; brightness >= 0; brightn
ess--) {
      analogWrite(thisPin, brightness);
      delay(2);
    }
    delay(100);
  }
}

```

Experiment 2. Servomotor.

In this experiment user is allowed to control servo. It is controlled through digital pin. Servo got 3 pins: vcc, ground and control pin (Fig. 3.11, 3.12). Scenarios of experiment: Two position turns, Smooth turns. Examples of experiment realization describes below.

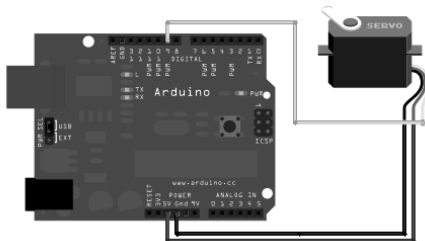


Figure 3.11 Model of the experiment Servomotor

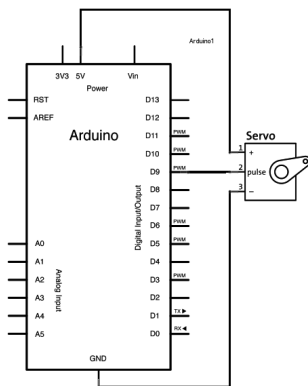


Figure 3.12 Servomotor experiment implementation scheme

Program is using *Servo.h* library, which contains *Servo* class implementation. It is used to communicate with motor. Servo object is created on the beginning of the code. The code consists of *setup* and *loop* functions. *Setup* is called only once when board powers on. In this function servo is attached to pin 9 by calling *attach* metod of *Servo* object. *Loop* function is called continuously during all the working time of the board. In this function all the work is done. To set position of the servo method *write* is used. Angle is passed to the method as parameter. Function *delay* is halt board's work for desirable time in milliseconds. It

is necessary to give the servo time to perform turn before next command is given. Not all servos got turning range of 0-180 degrees. It may vary depending on model.

Scenario 2.1. Two positions revolving. Servomotor receives two positions in turns. There is 2 sec delay between turns. The delay is meant to allow motor to finish the action.

```
#include <Servo.h>
  Servo myservo;
int pos = 0;
void setup()
{
  myservo.attach(9);
}
void loop()
{
  if(0 == pos) pos = 120;
else pos = 0;
myservo.write(pos);
delay(2000);
}
```

Scenario 2.2. Slow revolving. Motor revolving is performed slowly. Motor cycle is triggered to change its position by one degree unit every 15ms in order to achieve it.

```
#include <Servo.h>
  Servo myservo;
int pos = 0;
void setup()
{
  myservo.attach(9);
}
void loop()
{
  for(pos = 0; pos < 180; pos += 1)
  {
    myservo.write(pos);
    delay(15);
  }
  for(pos = 180; pos>=1; pos--=1)
  {
    myservo.write(pos);
    delay(15);
  }
}
```

Experiment 3. LCD display

In this experiment user is allowed to control liquid crystal display. Display has integrated controller and not accessed by Arduino directly. Communication is done via I2C serial protocol (Fig. 3.13, 3.14). Scenarios of experiment: Static text, Scrolling text.

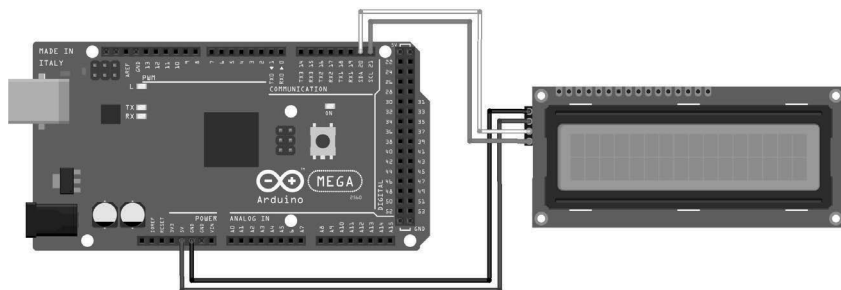


Figure 3.13 Model of the experiment LCD display

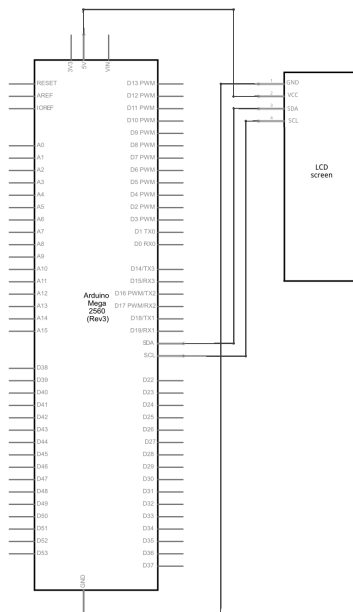
Examples of experiment realization describes below.

LiquidCrystal_I2C.h and **Wire.h** libraries are included in the program. **Wire.h** implements I2C protocol and **LiquidCrystal_I2C.h** implements communication with LCD via I2C. **LiquidCrystal_I2C** class object is created on the beginning of the code with parameters specific to given LCD controller. Changing those is not recommended.

The code consists of **setup** and **loop** functions. **Setup** is called only once when board powers on. In this case in the function LCD parameters are set in **begin** call. Its parameters are numbers of rows and columns. **Backlight()/noBacklight()** function is called to enable/disable LCD's backlight.

Loop function is called continuously during all the work time of the board. Cursor position may is set by calling **setCursor** (column, row) function. **Clear()** function is called to remove all the symbols from LCD. **Delay(millisecons)** function is called to delay activity of the board for stated time in milliseconds. Function **print(string)** is called to print symbols starting from cursor position.

Scenario 3.1. Static text. «Hello, world!» is displayed in the first row. In a second the second row is populated with «I am display» message. Display is cleared in a second after the second message shows up. Actions happen in cycle.



from that new position.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
LiquidCrystal_I2C lcd(0x27, 2, 1, 0, 4, 5, 6, 7, 3,
POSITIVE);
int col_offset = 0;
int row_offset = 0;
void setup() {
  lcd.begin(16,2);
  lcd.backlight();
}
void loop() {
  lcd.setCursor(col_offset, row_offset);
  lcd.print(«Running text»);
  delay(100);
  lcd.clear();
  col_offset = (col_offset + 1) % 16;
  if(15 == col_offset) row_offset = (row_offset + 1)
% 2;
}
```

Experiment 4. Sonic distance sensor

In this experiment user works with sonic distance sensor. Sensor has 4 pins: vcc, ground, trigger and signal. (Fig. 3.15, 3.16). Scenarios of experiment: Distance measurement, Distance sensor with servo. Examples of experiment realization describes below.

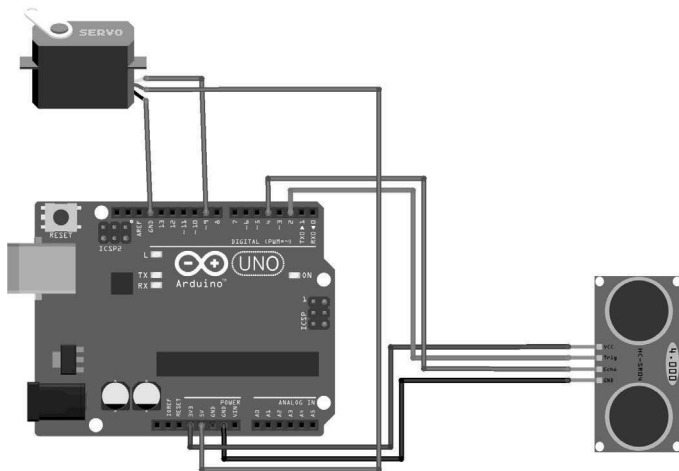


Figure 3.15 Model of the experiment Ultrasonic sensor distance

User receives data from sensor through the serial interface. **Serial** class is used for that. The program code consists of functions **setup** and **loop**. **Setup** is called only once when board powers on. In this function serial interface is initialized with baud rate of 9600 and sensor pins modes are set with **pinMode** call. Variables with sensor pins are declared before **setup** function. **Loop** function is called continuously during all the work time of the board. In this function all the work is done. Distance sensor has two modes: measuring and sending data. To set it in measuring mode pulse with 2 microseconds length must be sent to the sensor (trig pin). It is done by calling **digitalWrite** function. Its parameters are pin number and pin value. To form the pulse function called multiple times with LOW and HIGH values. Delays between calls are made with **delayMicroseconds** function. To read date from the sensor **pulsIn** is called. This function's arguments are pin number and expected value (HIGH or LOW). Returned value will represent travel time of sonic signal measured by the sensor. To convert that value to distance functions **microsecondsToCentimeters** and **microsecondsToInches** are used. Results are returned to user by **print** method of serial interface. Sequential interface and **print** function are used in order to display data to a user.

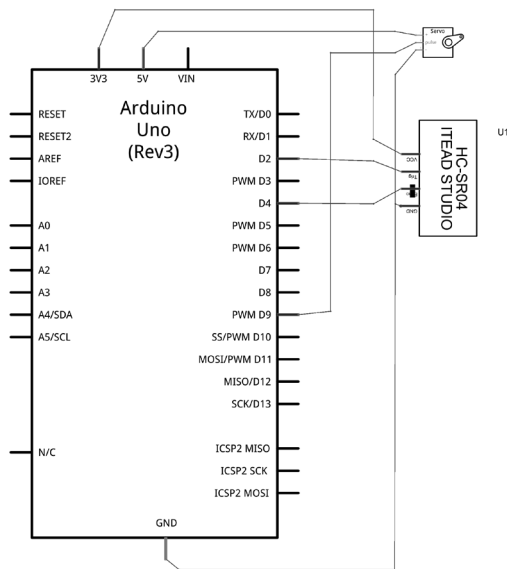


Figure 3.16 Sonic distance center experiment implementation scheme

Scenario 4.1. Measuring distance. Distance sensor can be revolved by a servomotor. It will measure a distance between different objects. There are partition walls on a stand, located on a various distance from distance sensor. We suggest a user revolves the motor while getting data from a distance sensor.

We are using **write** call to change motor position, rotation angle is a parameter here. In this particular case rotation data is stored in an array and it's transferred to a motor one by one. We are using 10ms impulse to trigger distance sensor. In this particular case angle values are stored in an array and are sent to a motor one by one. We are using **DigitalWrite** call in order to store data (0 and 1). We are using **pulseIn** call in order to read sensor measurements (sonic signal travel time). Sequential interface and **print** function are used in order to display data to a user.

```
const int trigPin = 2;
const int sigPin = 4;
void setup() {
  Serial.begin(9600);
  pinMode(trigPin, OUTPUT);
  pinMode(sigPin, INPUT);
}
void loop()
{
  long duration, inches, cm;
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(5);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(sigPin, HIGH);
  inches = microsecondsToInches(duration);
  cm = microsecondsToCentimeters(duration);
  Serial.print(inches);
  Serial.print(<in, <);
  Serial.print(cm);
  Serial.print(<<cm>>);
  Serial.println();
  delay(100);
}
long microsecondsToInches(long microseconds)
{
  return microseconds / 74 / 2;
```

```

}
long microsecondsToCentimeters(long microseconds)
{
    return microseconds / 29 / 2;
}

```

Scenario 4.2. Servomotor distance sensor.

In case of second scenario, sensor is attached to servo. It allows user to turn the sensor manually and measure distance to different objects. Servo position is set by write method of *Servo* object. Angle values are stored in the array. After servo position is set command to measure distance is called. To do so logical 1 pulse with duration of 10 microseconds is sent to the sensor with *digitalWrite* call. To read the measurements *pulseIn* is called. Results are sent to the user via serial interface by calling *print* method.

```

#include <Servo.h>
Servo myservo;
int positions[] = { 0, 30, 60, 90, 120, 150, 180 };
int pos_num = 7;
int pos = 0;
const int trigPin = 2;
const int echoPin = 4;
void setup()
{
    Serial.begin(9600);
    myservo.attach(9);
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
}
void loop()
{
    myservo.write( positions[pos] );
    pos++;
    if( pos == pos_num ) pos = 0;
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    Serial.print( pulseIn(echoPin, HIGH) / 29 / 2 );
    Serial.print(«cm»);
    Serial.println();
    delay(500);
}

```

See Fig. 3.17 for RELDES remote lab development team.



Figure 3.17 RELDES remote lab development team.

See Fig. 3.18 for RELDES lab use case diagram. Any user who wants to access experiment results must be registered. Registered users will be able to upload their own code into Arduino card microcontrollers using a web client. They may also use different programs, suggested by conducted experiments. Lab server compiles the source code and uploads it to Arduino card microcontrollers. Shall a server find any errors, it will compile and send back compilation results. Thus, a user can create and edit a programming code right there in the client interface. They don't even need to use their own development kit.

RELDES API provides access to the lab server infrastructure and allows a seamless reuse of the lab functionality. The API includes functions for the next HTTP requests (*id* – experiments number from 1 to 4 and *username* – user login):

- GET/.../expStatus/*id* – checks on the status of experiment and returns result: busy(1) or free (0).
- POST/.../queueUp/*id/username* – checks on the status of experiment and puts the user with the given *username* in the experiment queue. Returns the result about adding to the queue.
- GET/.../getQueue/ *id/username* – gets the waiting time specified user #*UserName* for a given experiment #*id*
- GET/.../getExpQueue/*id* – gets a list of all queues on the experiment #*id*

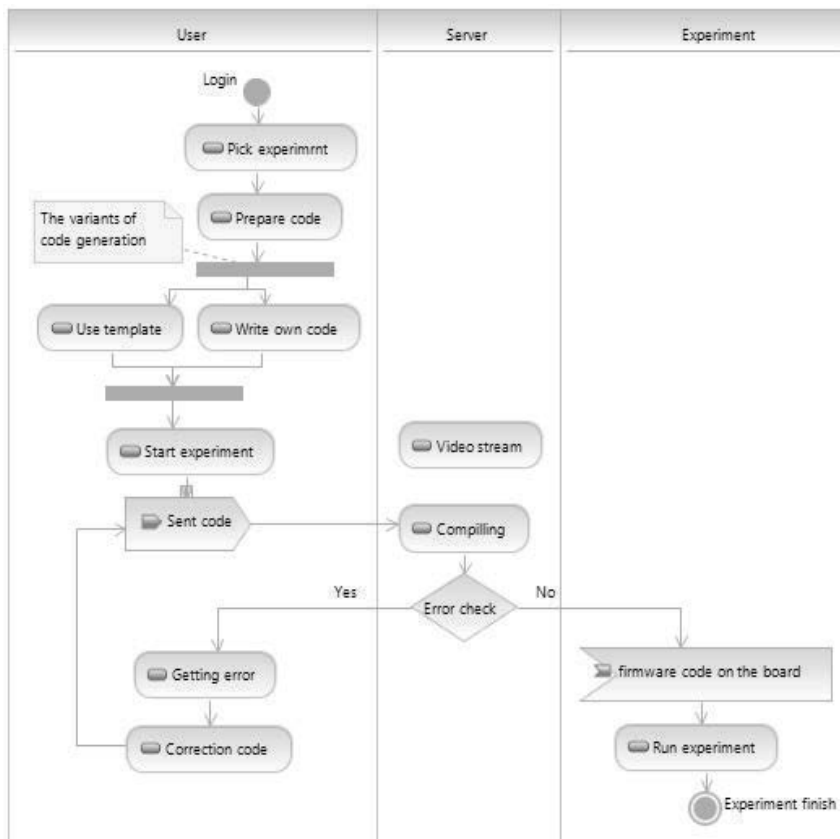


Figure 3.18 REDES lab use case diagram.

- GET/.../getAllQueue – gets a list of all queues on experiments
- POST/.../compile/id/ – checks the validity of the program code for Arduino board.
 - POST/.../upload/id/username/ – uploads the hex file after compiling if current user has taken the experiment.

Any registered developers can use the API for their own purposes. It may be use for creation of mobile applications, new clients or services.

3.2.3 Rapid Prototyping Board of laboratory GOLDI

View of the Rapid Prototyping (RP) board [15-18,34] is shown in Fig. 3.19. Operation Modes:

- 1) Web-based *rapid prototyping* of digital systems:
 - design (text based, PCB, automaton graph);
 - program the CPLD via the remote lab infrastructure;
 - test the design via RIA and Web-based RP board.
- 2) Web-based *validation* of digital systems:
 - identify the function of a given design (black box);
 - find malfunctions of a well-known design;
 - by manipulating the inputs or upload the truth table or test vectors;
 - CPLD not programmable by students.

Main Components of Rapid Prototyping Board are: MAX V – 5M1270Z (CPLD from Altera); Input buttons (8 slide switches, 8 push buttons, 2 rotary hex encoder); LED outputs (4 7-segment displays, 1 LED bar display); Other components (10 MHz crystal oscillator, Frequency synthesizer, Piezoelectric oscillator, Incremental encoder, UART (via USB), 25-pin SUB-D connector).

Interaction scheme RP board with the RL infrastructure is shown in Fig.3.20:

- communication with the remote lab infrastructure via an Interconnection FPGA;
- all inputs of the board are removed from the PCB;
- replaced by a direct connection to the outputs of the „Interconnection“ FPGA;
- sets inputs according user's input via the user interface;
- reads outputs directly;
- „Interconnection“ FPGA is connected to the BPU within the remote lab infrastructure [16].

Web-based User Interface realization as HTML5 RIA (Rich Internet Application) is shown in Fig. 3.21.

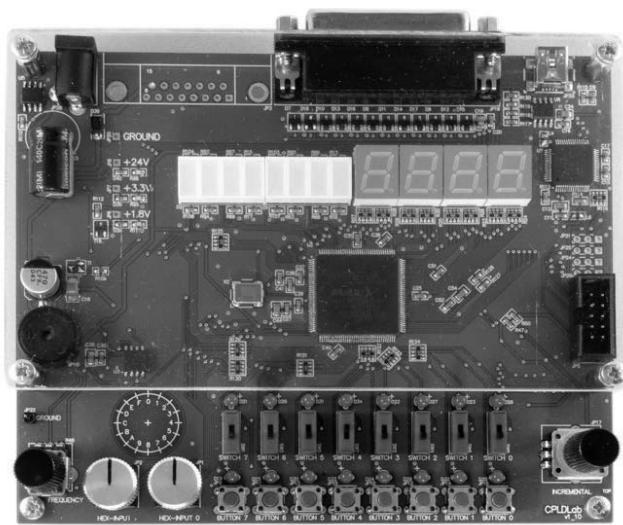


Figure 3.19 Rapid Prototyping Board

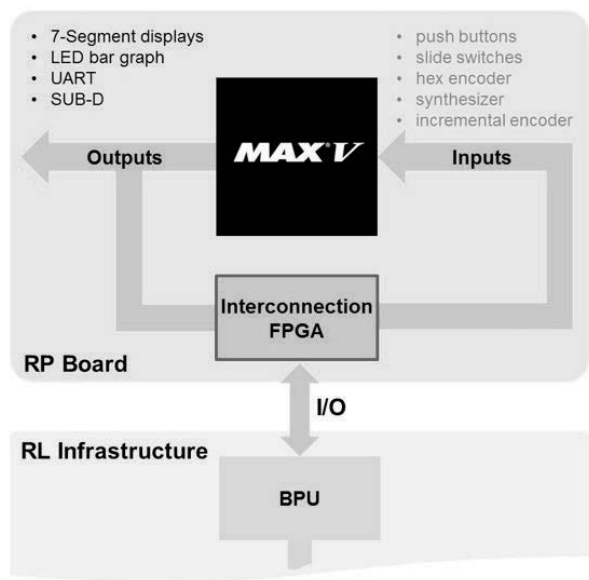


Figure 3.20 Interaction scheme of RP board with the laboratory

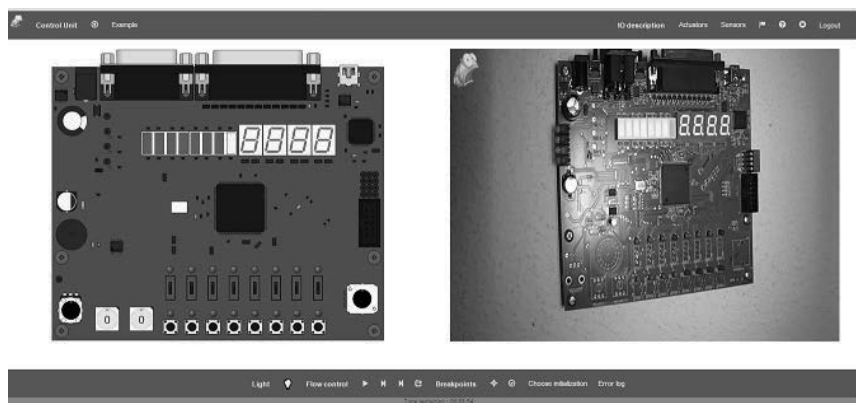


Figure 3.21 Web-based User Interface – RIA

Main interface components are:

- upload the synthesized CPLD code of the design to program the CPLD automatically in the remote lab;
- visual model of the RP Board, to manipulate all the inputs virtually and watch the outputs of the RP board;
- webcam observation in the remote lab to watch the results of the user's action directly.

Design Flow for Digital Systems is shown in Fig. 3.22 [16]. The diagram shows the basic blocks:

- common design tools: implement the control tasks e.g. with Altera's IDE; students can use „self-made“ IP core libraries;
- upload the synthesized design to the RP board;
- programming the stand-alone RP board;
- web-based programming of the RP board via the remote lab infrastructure.

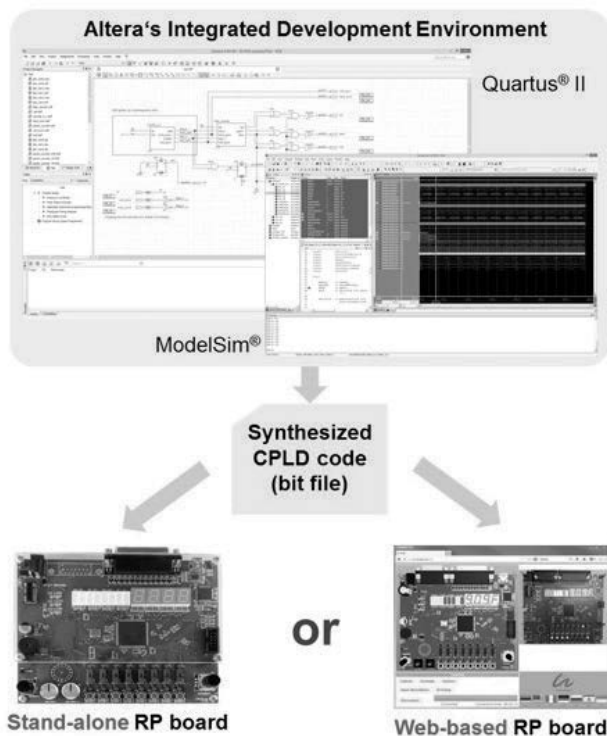


Figure 3.22 Design Flow for Digital Systems

Text based design methods enter the design by means of logical equations, truth tables or hardware description languages (AHDL, VHDL or Verilog) (Fig. 3.23).

Graphically based design methods enter the design by using block diagrams and/or schematic diagrams (Fig. 3.24).

Integrated FSM editor enter the design by using automata graphs (FSM) (Fig. 3.25).

Simulation of the design by using different simulation tools (e.g. Qsim or ModelSim) (Fig. 3.26) [16].

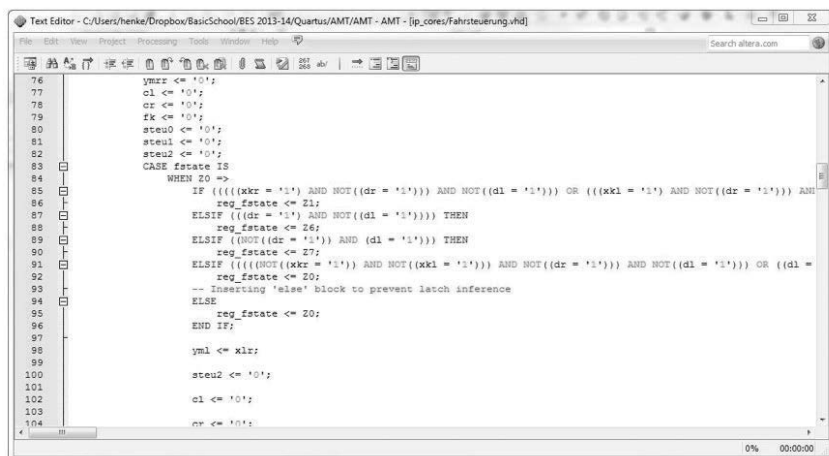


Figure 3.23 Text based design method

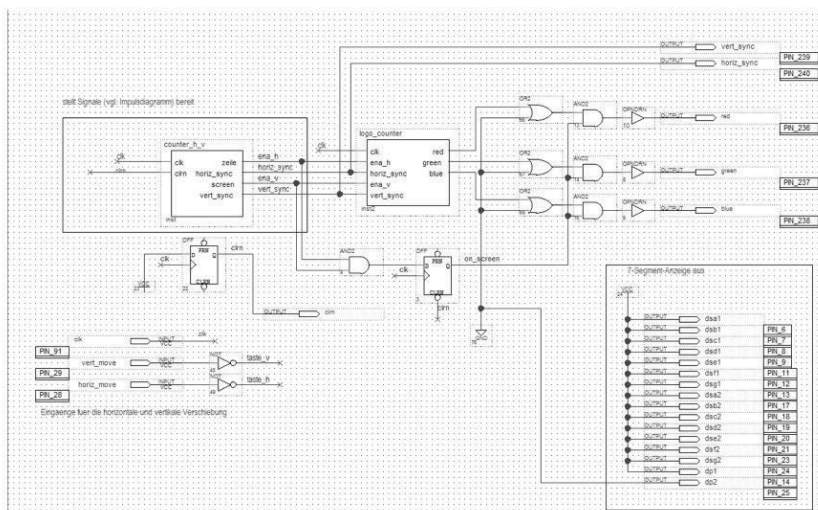


Figure 3.24 Graphically based design method

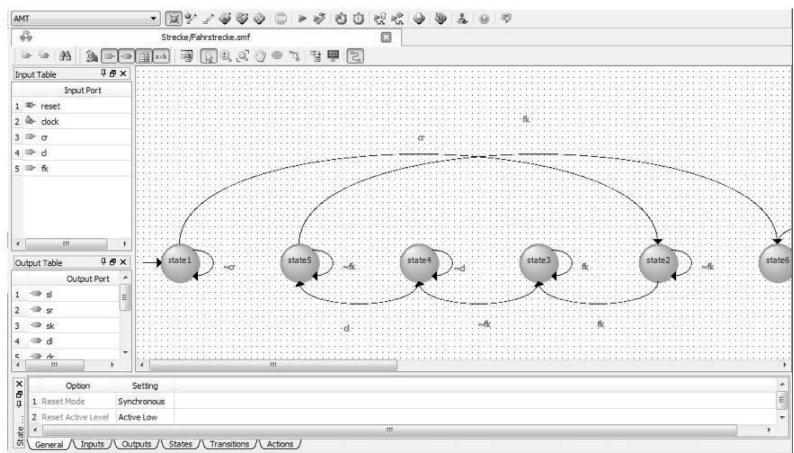


Figure 3.25 FSM based design method

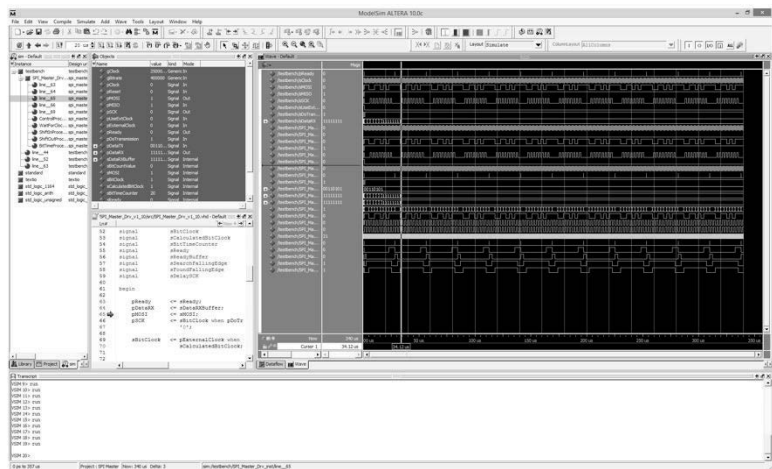


Figure 3.26 Simulation of the design

3.2.4 GOLDI experiment Using of the 7-segment Displays

Next, we'll look at the example of work the 7-segment display of the Rapid Prototype Board using the system Quartus II from Altera.

The MAX V CPLD (Complex Programmable Logic Device) RP Board has four 7-segment displays. Applying a low logic level to a segment causes it to light up, and applying a high logic level turns it off. Each segment in a display is identified by an index from 0 to 7, with the positions given in Fig. 3.27. Table 3.1 shows the assignments of CPLD pins to the first 7-segment display.

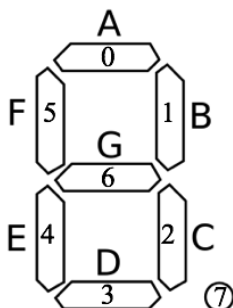


Figure 3.27 Position and index of each segment in a 7-segment display

Using this display we are displaying the numbers from 0 to 9. To display 1 by using the 7 segment display, we have to do output B and C higher. In the binary code the 1 is represented like a '0001'. So, '0001' is needed to be converted in '1001111' using BCD 7-segment decoder. Table 3.2 presents the truth table of the BCD 7-segment decoder.

As the binary inputs we'll use the 4 board's toggle switches.

The board provides 8 toggle switches (sliders). These switches are intended for use as level-sensitive data inputs to a circuit. Each switch is connected directly to a pin on the Rapid Prototype Board. When a switch is in the DOWN position it provides a low logic level (0 volts) to the CPLD, and when the switch is in the UP position it provides a high logic level. A list of the pin names on the Rapid Prototype Board that are connected to the toggle switches is given in Table 3.3.

Start the Quartus II. Select **File->New Project Wizard...** and «Next». Now you are asked to name your project and working directory. Here we will name the project «7_Seg_Decoder». Click twice on «Next» (Fig. 3.28).

Table 3.1 – Pin assignment for the first 7-segment display

Signal Name	CPLD pin number	Description
Hex0[7]	PIN_84	Output
Hex0[6]	PIN_88	Output
Hex0[5]	PIN_89	Output
Hex0[4]	PIN_91	Output
Hex0[3]	PIN_93	Output
Hex0[2]	PIN_85	Output
Hex0[1]	PIN_86	Output
Hex0[0]	PIN_87	Output

Table 3.2 – Truth table

Binary Inputs				Decoder Outputs							Display Outputs
X3	X2	X1	X0	a	b	c	d	e	f	g	
0	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	1	0	0	1	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0	2
0	0	1	1	0	0	0	0	1	1	0	3
0	1	0	0	1	0	0	1	1	0	0	4
0	1	0	1	0	1	0	0	1	0	0	5
0	1	1	0	0	1	0	0	0	0	0	6
0	1	1	1	0	0	0	1	1	1	1	7
1	0	0	0	0	0	0	0	0	0	0	8
1	0	0	1	0	0	0	0	1	0	0	9
1	0	1	0	0	0	0	1	0	0	0	A
1	0	1	1	1	1	0	0	0	0	0	b
1	1	0	0	0	1	1	0	0	0	1	C
1	1	0	1	1	0	0	0	0	1	0	d
1	1	1	0	0	1	1	0	0	0	0	E
1	1	1	1	0	1	1	1	0	0	0	F

Table 3.3 – Pin assignment for the toggle switches

Signal Name	CPLD pin number	Description
Switch[7]	PIN_55	Input
Switch[6]	PIN_58	Input
Switch[5]	PIN_60	Input
Switch[4]	PIN_62	Input
Switch[3]	PIN_66	Input
Switch[2]	PIN_68	Input
Switch[1]	PIN_70	Input
Switch[0]	PIN_72	Input

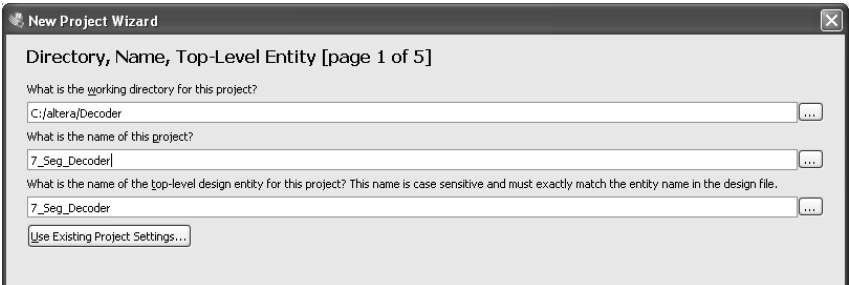


Figure 3.28 Window for input the project name

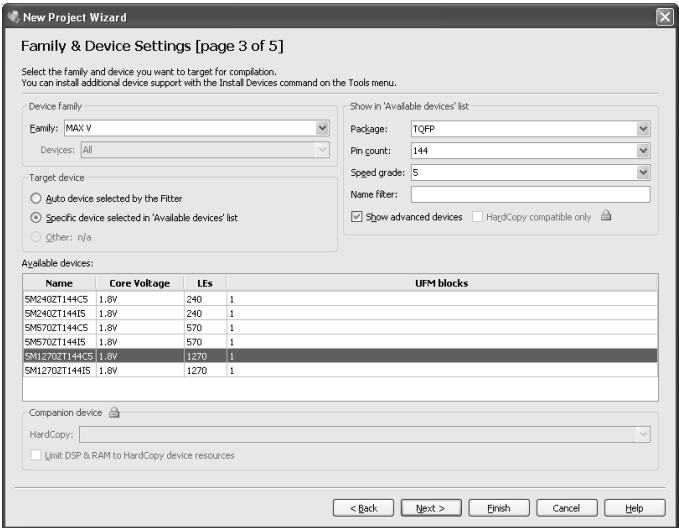


Figure 3.29 Window for selection the CPLD

On the following page (Fig. 3.29) you select the CPLD you want to use. We need here the MAX V 5M1270ZT144C5. But this selection can later be changed (**Assignments->Device**). Afterwards you select Next».

In the next window (Fig. 3.30) you chose the ModelSim-Altera like Simulation EDA tool. Afterwards you select «Next» and «Finish». For creating new design files select **File->New**. In the list you chose the “Block Diagram/Schematic File” design files (Fig. 3.31).

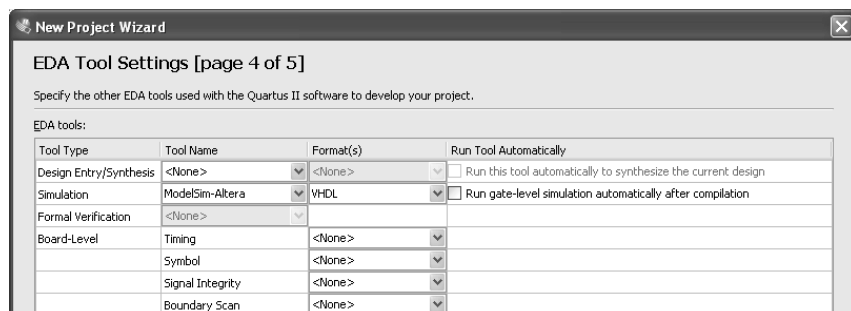


Figure 3.30 Window for select the Tool name

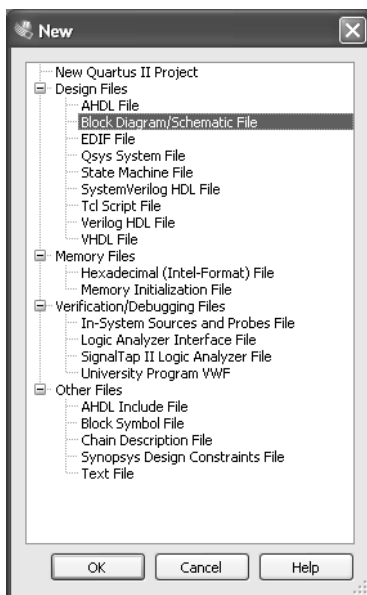


Figure 3.31 Window for creating new design files

Figure 1 displays seven 6x5 binary matrices, labeled (a) through (g), arranged in a 3x3 grid. Each matrix contains binary values (0 or 1) and is structured with a diagonal line in the top-left 3x3 submatrix. The diagonal elements are labeled X0, X1, and X3 X2. The remaining elements are binary values (0 or 1).

The matrices are as follows:

- (a)

X0	0	1	1	0
X1	0	0	1	1
X3 X2	0	0	0	0
0	0	0	1	0
0	1	1	0	0
1	1	0	1	0
1	0	0	0	1
- (b)

X0	0	1	1	0
X1	0	0	1	1
X3 X2	0	0	0	0
0	0	0	0	0
0	1	0	1	0
1	1	1	0	1
1	0	0	0	1
- (c)

X0	0	1	1	0
X1	0	0	1	1
X3 X2	0	0	0	0
0	0	0	0	1
0	1	0	0	0
1	1	1	0	1
1	0	0	0	0
- (d)

X0	0	1	1	0
X1	0	0	1	1
X3 X2	0	0	0	0
0	0	0	1	0
0	1	1	0	1
1	1	0	0	1
1	0	0	0	1
- (e)

X0	0	1	1	0
X1	0	0	1	1
X3 X2	0	0	0	0
0	0	0	1	1
0	1	1	1	0
1	1	0	0	0
1	0	0	1	0
- (f)

X0	0	1	1	0
X1	0	0	1	1
X3 X2	0	0	0	0
0	0	0	1	1
0	1	0	0	1
1	1	0	1	0
1	0	0	0	0
- (g)

X0	0	1	1	0
X1	0	0	1	1
X3 X2	0	0	1	1
0	0	1	1	0
0	1	0	0	1
1	1	1	0	0
1	0	0	0	0

$$a = \overline{x_1}x_0(\overline{x_3 \oplus x_2}) + \overline{x_3}x_2\overline{x_1}x_0 + x_3\overline{x_2}x_1x_0;$$

$$b = x_2x_1(\overline{x_3 \oplus x_0}) + x_3x_1x_0 + x_2x_1\overline{x_0};$$

$$c = \overline{x_3}x_2\overline{x_1}x_0 + x_3x_2(x_1 + \overline{x_0});$$

$$d = \overline{x_3}x_2\overline{x_1}x_0 + \overline{x_3}x_2x_1x_0 + x_2x_1x_0 + x_3\overline{x_2}x_1\overline{x_0};$$

$$e = \overline{x_3}x_2\overline{x_1} + \overline{x_3}x_0 + \overline{x_2}x_1x_0;$$

$$f = x_3x_2\overline{x_1}x_0 + \overline{x_3}x_2(x_0 + x_1) + \overline{x_3}x_1x_0;$$

$$g = \overline{x_3}x_2\overline{x_1} + x_3x_2\overline{x_1}x_0 + \overline{x_3}x_2x_1x_0.$$

Figure 3.32 Output values for creation the 7-segment decoder logic circuit

From the above truth table (Table 3.3), we get the k-map simplification for the common cathode seven-segment decoder in order to design the

combinational circuit. And from the simplification, we get the output values for creation the 7-segment decoder logic circuit.

For creation the logic circuit of BCD to 7-segment decoder on the tool-bar you find the icon for the insertion of symbols, the Symbol Tool (On the right side of the A – symbol.) Here you can insert NOT, OR and AND gates. (Libraries: primitives->logic) (Fig.3.33).

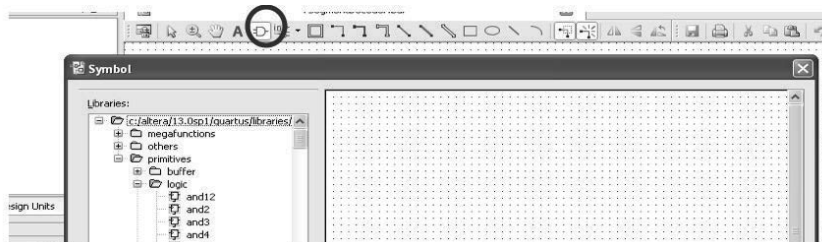


Figure 3.33 Symbol Tool

Next to the Symbol Tool is the icon for the Pin Tool. Insert for Input-Pins and seven Output-Pins. Click twice on pin name that to change it. Connect the elements like shown on Fig. 3.34. Save file again.

Set design as a top level design **Project-> Set as Top-Level Entity**. Compile the design **Processing->Start Compilation**.

In order to use the 7-segment decoder for building a main design, we will create a Symbol File for the 7-segment decoder. With this file we can later insert the full logic of the 7-segment decoder in the other Block Diagram only with one symbol. To do this, select **File ->Create/Update ->Create Symbol Files for Current File**.

For simulation of design the simulation file has to be created: **File->New->University Program VWF**. Save file. Simulation file should have the same name as schematic file – “SevSegmentDecoder”. Add the inputs and outputs to the simulation design file: **Edit->Insert->Insert Node or Bus....** Set values for the each input.

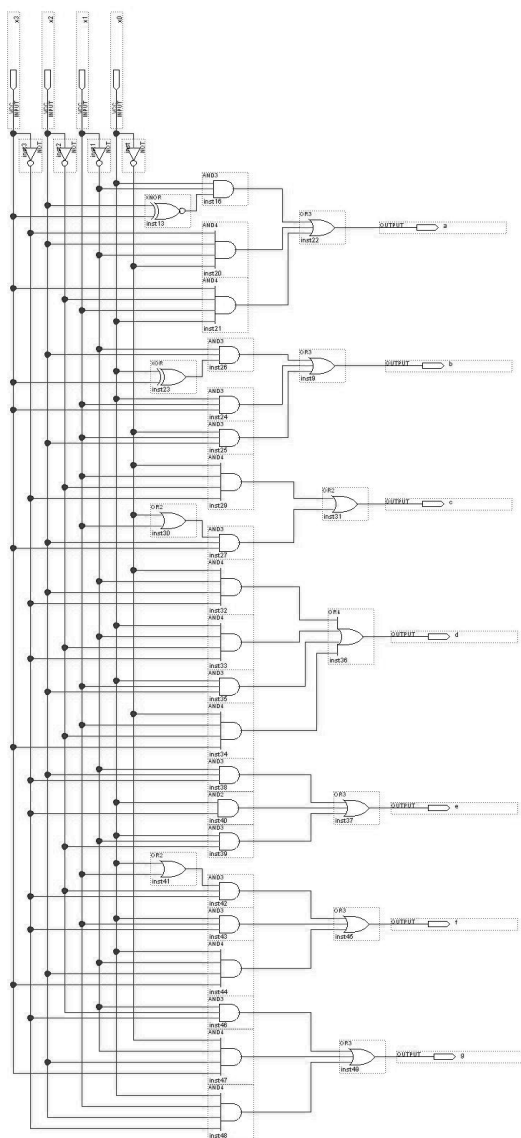


Figure 3.34 Project logic scheme _1

Here click «Node Finder...». Now «List» all Nodes and select them all (Fig.3.35). Click twice «OK».

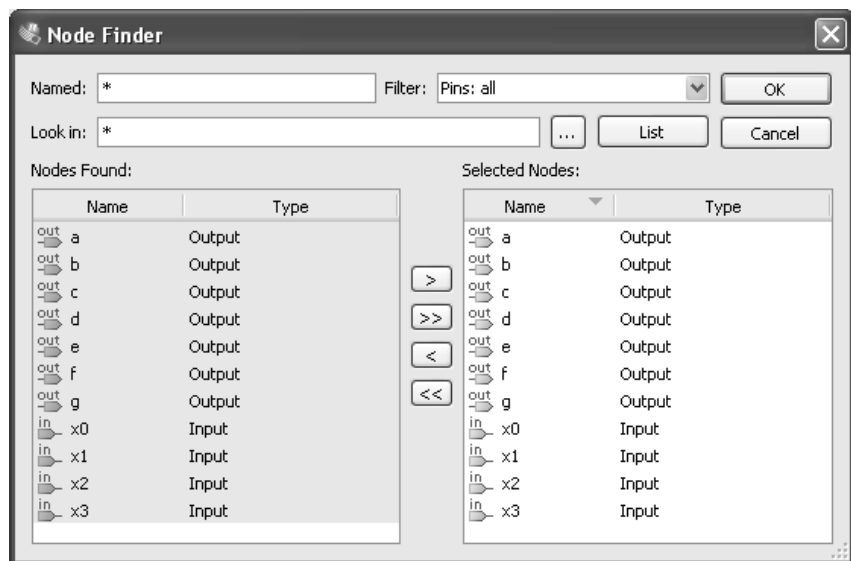


Figure 3.35 Window Node Finder

In this file you can now define the behavior of the inputs. For the all-signals it is easier to use the «Count Value» in the toolbox. To create the group, select all inputs using the “Ctrl” and the right mouse button. Then select **Edit->Grouping->Group...** Set the values from the other inputs like the behavior you want to test. (You can not change the output values.) For example like on Fig. 3.36.

You can start the simulation with **Simulation->Run Functional Simulation**. Check the simulation results (Fig.3.37).

Create new schematic file “Seg_Decoder_test.bdf” (Fig. 3.38).

Set as a Top-Level Entity. Start compilation. After, for the Pinout of you design you use the Pin Planner: **Assignments->Pin Planner** (Fig. 3.39). Number of pin use from the Table 3.1 and Table 3.3. After pinout the design look like on Fig.3.40.

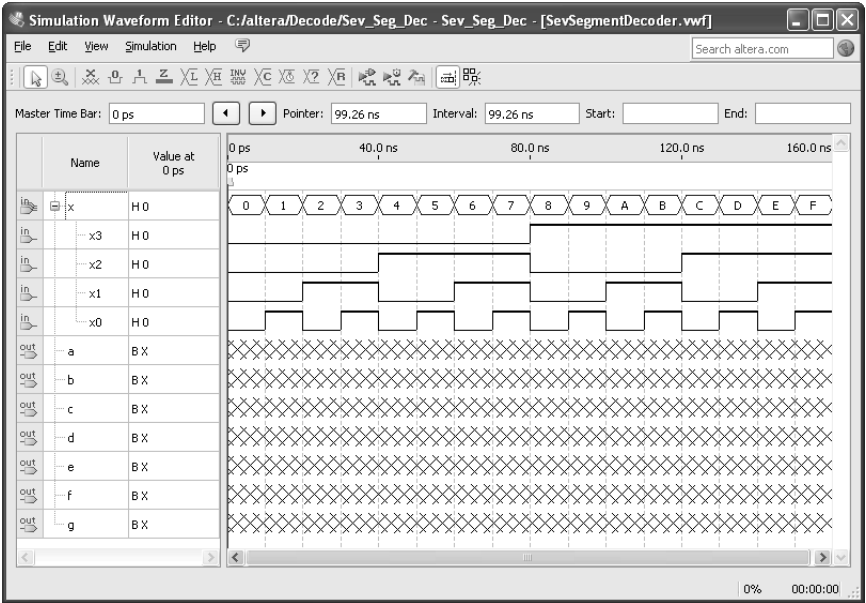


Figure 3.36 Window for the definition of the inputs behavior and creation the inputs group

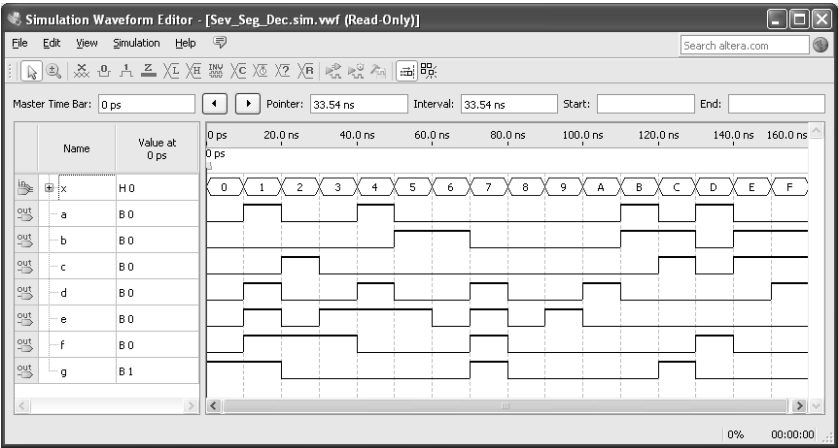


Figure 3.37 Simulation results

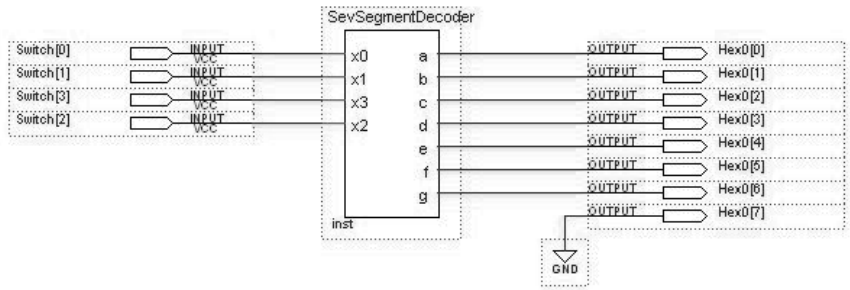


Figure 3.38 Project logic scheme _2.

The screenshot shows the 'Window Pin Planner' tool. The top window displays a top view of a board with pins and labels: 'Top View Wire Board', 'MAX V', and 'SM1202T144CS'. The bottom window shows a table of pin configurations with columns: Node Name, Direction, Location, I/O Bank, Fitter Location, and I/O Standard. The table lists various nodes and their connections.

Node Name	Direction	Location	I/O Bank	Fitter Location	I/O Standard
out Hex0[7]	Output	PIN_84	3	PIN_50	3.3-V LV...default)
out Hex0[6]	Output	PIN_88	3	PIN_67	3.3-V LV...default)
out Hex0[5]	Output	PIN_89	3	PIN_58	3.3-V LV...default)
out Hex0[4]	Output	PIN_91	3	PIN_52	3.3-V LV...default)
out Hex0[3]	Output	PIN_93	3	PIN_55	3.3-V LV...default)
out Hex0[2]	Output	PIN_85	3	PIN_60	3.3-V LV...default)
out Hex0[1]	Output	PIN_86	3	PIN_57	3.3-V LV...default)
out Hex0[0]	Output	PIN_87	3	PIN_61	3.3-V LV...default)
in Switch[3]	Input	PIN_66	4	PIN_74	3.3-V LV...default)
in Switch[2]	Input	PIN_68		PIN_89	3.3-V LV...default)
in Switch[1]	Input			PIN_63	3.3-V LV...default)
in Switch[0]	Input			PIN_62	3.3-V LV...default)

Figure 3.39 Window Pin Planner

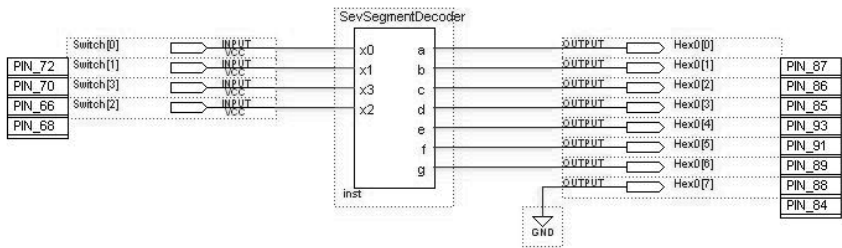


Figure 3.40 Project logic scheme _2 after pinout

Save the project. Compile again. In the project directory in the folder “output_files” you can find file “7_Seg_Decoder.pof”. Use this file to program CPLD board. After the result has been programmed to the RP board of GOLDI lab (Stand alone or Web-based) you can test your design (Fig. 3.41, 3.42).

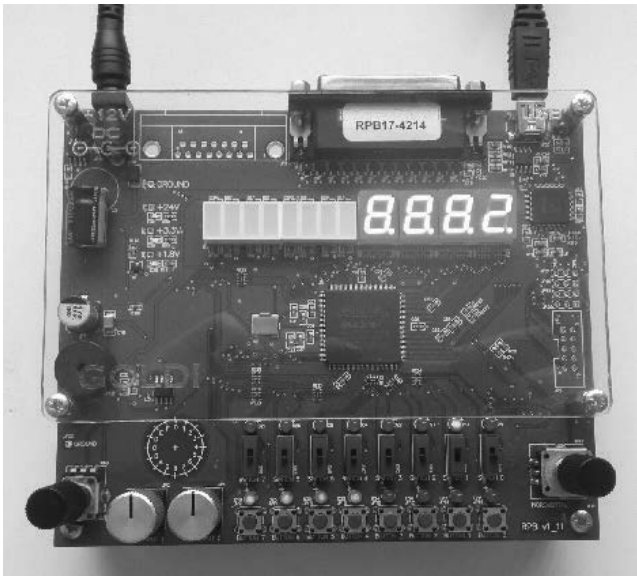


Figure 3.41 Test design on GOLDI Stand alone RP board

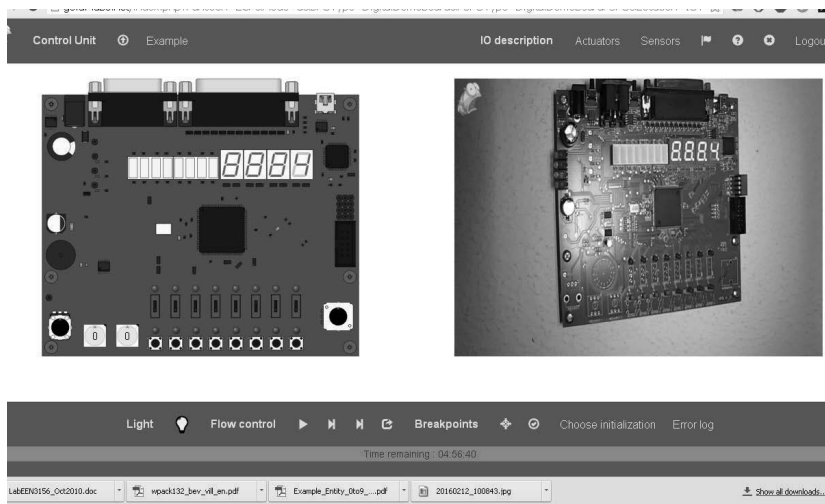


Figure 3.42 Test design on GOLDI Web-based RP board

Other way to create the 7-segment decoder is the use of hardware languages:

- VHDL – Very high speed integrated circuits **H**ardware **D**escription **L**anguage;
- AHDL – Altera **H**ardware **D**escription **L**anguages;
- Verilog HDL.

Here we will consider just two examples of language: VHDL and AHDL. Create the new file **File->New->VHDL File**. Here is a program for BCD to 7-segment display decoder. Copy this code in the VHDL file.

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY seg_vhdl IS
    PORT (
        x : IN      STD_LOGIC_VECTOR(3 DOWNT0 0);--BCD
input
        y : OUT     STD_LOGIC_VECTOR(6 DOWNT0 0));--
7bit decoded output
END seg_VHDL;
ARCHITECTURE TrueTable OF seg_VHDL IS
BEGIN
```

```

WITH x SELECT
    y<=>0000001» WHEN «0000», --0
    «1001111» WHEN «0001», --1
    «0010010» WHEN «0010», --2
    «0000110» WHEN «0011», --3
    «1001100» WHEN «0100», --4
    «0100100» WHEN «0101», --5
    «0100000» WHEN «0110», --6
    «0001111» WHEN «0111», --7
    «0000000» WHEN «1000», --8
    «0000100» WHEN «1001», --9
    «0001000» WHEN «1010», --A
    «1100000» WHEN «1011», --b
    «0110001» WHEN «1100», --C
    «1000010» WHEN «1101», --d
    «0110000» WHEN «1110», --E
    «0111000» WHEN «1111», --F
    «1111111» WHEN others;
END TrueTable;

```

Save file like “seg_vhdl”, set as a top-level entity, compile and create symbol file.

For creation AHDL design select: **File->New->AHDL File**. Copy the next code in the file.

```

SUBDESIGN    seg_ahdl
(
    p[3..0]      : INPUT;
    a,b,c,d,e,f,g : OUTPUT;
)
BEGIN
    TABLE
p[3..0] => a,b,c,d,e,f,g;
H>0» => 0, 0, 0, 0, 0, 0, 1;
H>1» => 1, 0, 0, 1, 1, 1, 1;
H>2» => 0, 0, 1, 0, 0, 1, 0;
H>3» => 0, 0, 0, 0, 1, 1, 0;
H>4» => 1, 0, 0, 1, 1, 0, 0;
H>5» => 0, 1, 0, 0, 1, 0, 0;
H>6» => 0, 1, 0, 0, 0, 0, 0;
H>7» => 0, 0, 0, 1, 1, 1, 1;
H>8» => 0, 0, 0, 0, 0, 0, 0;
H>9» => 0, 0, 0, 0, 1, 0, 0;
H>A» => 0, 0, 0, 1, 0, 0, 0;
H>b” => 1, 1, 0, 0, 0, 0, 0;

```

```
H"C" => 0, 1, 1, 0, 0, 0, 1;  
H"d" => 1, 0, 0, 0, 0, 0, 1;  
H"E" => 0, 1, 1, 0, 0, 0, 0;  
H"F" => 0, 1, 1, 1, 0, 0, 0;  
END TABLE;
```

END;

Save file like “seg_ahdl”, compile and after successful compiling create symbol file, you can include this symbol into the project file.

3.3. Control Questions

- Main components of ES Integrated Development Environment.
- General characteristics of the most known ECAD-systems.
- General characteristics of the most known MCAD-system.
- Main stages of ES based on MCU design.
- Possibilities of Arduino development environment.
- Advantages of Arduino platform.
- RELDES architecture and set of experiments.
- GOLDI architecture and set of experiments.

3.4. Remote experiments

- RE 1-4. Implementation the experiment «Traffic Lights», «Servomotor», «LCD display» and «Ultrasonic sensor distance» in the RELDES remote lab environment using different scenarios.
- RE 5. Implementation the experiment «Rapid Prototyping Board» in the GOLDI remote lab environment using Text based design methods (AHDL, VHDL) and Graphically based design method.

3.5. Lab Works

- Lab work 1. Embedded systems software development by using Atmel Studio.
- Lab work 1. Embedded systems software development by using Arduino IDE.
- Lab work 2. Embedded systems hardware development by using Altium Designer.
- Lab work 3. Embedded systems hardware development by using PTC Creo.

CONCLUSIONS

Development and practical application of new electronic educational resources for learning of embedded systems complex hardware/software design is an urgent task, because existing problems in this area require a qualitatively new methods, technologies and design tools and, accordingly, new methods of future professionals training.

The practical value of the work lies in the fact that the use of established in ZNTU remote lab RELDES allows students to acquire knowledge and practical experience in embedded systems design: to receive and analyze information about ready hardware/software components to be used, to perform range of hardware platforms, to develop a program code, to check for compatibility and performance of the system hardware and software. Remote components testing allow assessing quickly the appropriateness of their use, and deciding on their integration into the project, thus reducing the risk and design time.

Such specialized remote labs are now more than just advanced educational resources. Their potential is useful not only in education but also in design and remote maintenance of embedded systems as they provide opportunities for sharing various equipment and specialized software. The use of such laboratories allows professionals to make the right decision about the possible development of the project.

The proposed set of laboratory experiments RELDES will be further extended to other hardware/software platforms, as well as problems associated with the implementation of various classes of embedded systems.

LITERATURE FOR PART 1

1. Abdurohman, M. The New Embedded System Design Methodology For Improving Design Process Performance / M. Abdurohman, Kuspriyanto, Sutikno, S., Sasongko, A. // International Journal of Computer Science and Information Security, Vol. 8, No. 1, 2010. – pp. 35-43,
2. Altium Designer (Protel) – pass-through system design of printed circuit boards [Available electronically] / Eurointech. – Access mode: <http://www.eurointech.ru/protel>

3. Altium Designer [Available electronically]/ CSoft. – Access mode: <http://www.csoft.ru/catalog/soft/altium-designer/altium-designer-10.html>
4. Arduino Simulator [Available electronically] : Access mode : www/ URL: <http://schogini.in/app-websites/arduino-simulator/>
5. Arduino [Available electronically]: Access mode: www/ URL: <https://ru.wikipedia.org/wiki/Arduino>
6. Atmel Studio [Available electronically]/ Atmel. – Access mode: www/ URL: <http://www.atmel.com/tools/atmelstudio.aspx>
7. A tradeoff between microcontroller, DSP, FPGA and ASIC technologies [Available electronically]/ EE Times. – Access mode: www/ URL: http://www.eetimes.com/document.asp?doc_id=1275272
8. Creo Parametric. [Available electronically]. – Access mode: . http://www.pro-technologies.ru/product/Creo_Parametric/ – Загл. с экр.
9. Difference Between FPGA and Microcontroller [Available electronically]/ DifferenceBetween.net. – Access mode: www/ URL: <http://www.differencebetween.net/technology/difference-between-fpga-and-microcontroller/>
10. Ebert, C. Software: Facts, Figures, and Future/ C. Ebert, C. Jones// Computer, vol. 42, No. 4, April 2009. – pp. 42-52, doi:10.1109/MC.2009.118
11. Embedded Systems: Technologies and Markets [Available electronically]/ BCC Research. – Access mode: www/ URL: <http://www.bccresearch.com/market-research/information-technology/embedded-systems-technologies-markets-ift016d.html>
12. Embedded System Market – Global Industry Analysis, Size, Share, Growth, Trends and Forecast, 2012 – 2018 [Available electronically]/ Transparency Market Research. – Access mode: www/ URL: <http://www.transparencymarketresearch.com/embedded-system.html>
13. Electronics from beginner to pro. <http://123d.circuits.io/>
14. Freeduino [Available electronically]: Access mode: www/ URL: http://www.freeduino.org/freeduino_open_designs.html
15. Henke, K. Fields of Applications for Hybrid Online Labs / Karsten Henke, Steffen Ostendorff, Heinz-Dietrich Wuttke, Tobias Vietzke, Christian Lutze // [International Journal of Online Engineering](#)

- (iJOE), Vol 9 (2013) – Access mode: <http://online-journals.org/i-joe/article/view/2542>
16. Henke, K. Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems / K. Henke, G. Tabunshchyk, H.-D. Wuttke, T. Vietzke, St. Ostendorff // Remote Engineering & Virtual Instrumentation–REV2014, Porto, Portugal, February 2014.-pp.61-66
 17. Henke, K. A Grid Concept for Reliable, Flexible and Robust Remote Engineering Laboratories/ Karsten Henke, Steffen Ostendorff, Heinz-Dietrich Wuttke, Stefan Vogel // Remote Engineering & Virtual Instrumentation – REV2012 , Bilbao, Spain, July 2012
 18. Henke K. Web-based Rapid Prototyping of Digital Systems/ Karsten Henke, Silvia Krug // International Conference on Interactive Computer-Aided Blended Learning, Antigua, Guatemala, November 2011.- pp. 22-27
 19. Leffingwell D. Managing Software Requirements: A Use Case Approach / Dean Leffingwell, Don Wording. – Addison-Wesley, 2003. – P. 402 – ISBN 0-321-12247-X.
 20. Mbed [Available electronically]: Access mode : www/ URL: <http://en.wikipedia.org/wiki/Mbed>
 21. Parkhomenko A. Complex requirements analysis for the high-level design of Embedded Systems / A. Parkhomenko, O. Gladkova // Bulletin «Lviv Polytechnic» – Computer systems design. Theory and Practice. – Lviv, Lviv Polytechnic, №808 (2015). – pp.3-10
 22. Parkhomenko A.V. Virtual Tools and Collaborative Working Environment in Embedded System Design / A.V. Parkhomenko, O.N. Gladkova // Proceedings of XI International Conference on Remote Engineering and Virtual Instrumentation (REV2014) (26-28 February, 2014, Porto, Portugal) Porto: Polytechnic, 2014. –pp. 91-93.
 23. Parkhomenko A. Development and Application of Remote Laboratory for Embedded Systems Design / A.Parkhomenko, O. Gladkova, E. Ivanov, A. Sokolyanskii, S. Kurson// iJOE – Volume 11, Issue 3, 2015.- pp.27-31
 24. Parkhomenko A. Internet-based Technologies for Design of Embedded Systems / A.Parkhomenko, O. Gladkova, S. Kurson, A. Sokolyanskii, E. Ivanov // Journal of Control Science and Engineering 2 (2015) pp. 55-63

25. Parkhomenko, A. Analysis and application of existent approaches in microcontroller system designing / Anzhelika Parkhomenko, Olga Gladkova // Proceedings of IX-th International Conference «PERSPECTIVE TECHNOLOGIES AND METHODS IN MEMS DESIGN (MEMSTECH 2013)», Lviv: Lviv Polytechnic, 2013. – pp.268-270.
26. Proteus VSM. Step-through debugging [Available electronically] – Access mode: <http://we.easyelectronics.ru/CADSoft/proteus-vsm-poshagovaya-otladka.html>.
27. RELDES [Available electronically]. – Access mode: <http://youtu.be/u2anq--UYFg>
28. Simulator for Arduino [Available electronically]: Access mode : [www/ URL: http://www.virtronics.com.au/Simulator-for-Arduino.html](http://www.virtronics.com.au/Simulator-for-Arduino.html)
29. Simuino [Available electronically]: Access mode : [www/ URL: https://code.google.com/p/simuino/](http://www.code.google.com/p/simuino/)
30. Shih, Randy H. Parametric Modeling with Creo Parametric 1.0 / Randy H. Shih. – SDC Publisher: Stepheer Schroff, 2011. – P. 432
31. Teich, J. Hardware/Software Codesign: The Past, the Present, and Predicting the Future”/ J. Teich // Proceedings of the IEEE, Vol.100, Germany, May 13, 2012. – pp. 1411-1429
32. Wiegers K. E. Software Requirements. / Karl E. Wiegers. – 2nd Edition.–Microsoft Press, 2003. – P. 544. ISBN:978-0-7356-1879-4.
33. Wolf H., Wayne, Hardware-Software Co-Design of Embedded Systems / H. Wayne Wolf // Proceedings of the IEEE, Vol.82, NO. 7, July 1994.- pp. 967-989
34. Wuttke H- D. Remote and Virtual Laboratories in Problem-Based Learning Scenarios / Heinz- Dietrich Wuttke, Raimund Ubar, Karsten Henke// IEEE International Symposium on Multimedia, Taichung, Taiwan, December 2010.- pp.377-382
35. Zielczynski P. Requirement Management Using IBM Rational RequisitePro. / Peter Zielczynski – IBM Press, 2007. – P. 360, ISBN 0-321-38300-1.
36. Belov, A.V. Create a device microcontroller / A.V. Belov. – SPb.: Science and Technology, 2007. – 304 c.

37. Blog rozrobnika Emulino [Available electronically]: Access: [www / URL: http://hewgill.com/journal/entries/507-emulino-arduino-cpu-emulator](http://www.hewgill.com/journal/entries/507-emulino-arduino-cpu-emulator)
38. Burmistrov, A.V. Real time operating system for microcontroller systems / A.V. Burmistrov, I.V. Filin // The successes of modern science, №6, 2012. – pp. 87-88
39. Vasiliev, A.E. Microcontrollers. Embedded Development / A.E. Vasiliev. – SPb.: BHV-Petersburg, 2008. – 304c.
40. Embedded Systems [Available electronically] / Spring. – Access: [www / URL: http://www.rodnik.ru/product/spa/embedded-solutions/](http://www.rodnik.ru/product/spa/embedded-solutions/)
41. High 8-bit RISC microcontroller family AVR [Available electronically]/microelectronics market. –Access mode: [www / URL: http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/avr/about.htm](http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/avr/about.htm)
42. ISO / MES 12207-2010 Information technology. System and software engineering. Software life cycle processes (ISO / IEC 12207: 2008). – Enter. 01.03.2012. – Moscow: Standartinform, 2011.- 104c.
43. GOST 34.602-89. Information technology. Set of standards for automated systems. Terms of reference for the creation of an automated system. – Enter. 01.01.1990. – M.: Publishing IEC Standards, 2004. 11 p.
44. GOST R 41904-2002 software embedded systems. General requirements for the design and documentation. – Enter. 24.07.2002. – Publisher IPC Standards, 2002. – 62c
45. Two approaches to the implementation of software for embedded [Available electronically] / Programming microcontrollers. – Access mode: [www / URL: http://habrahabr.ru/post/148805/](http://habrahabr.ru/post/148805/)
46. Destribyutor Freeduino in Russia [Available electronically]: Access: [www / URL: http://freeduino.ru/arduino/products.html](http://freeduino.ru/arduino/products.html)
47. State Standard ISO/IEC 14288: 2004. Information technologies. Life cycle process of the system (ISO / IEC 14288: 2002, IDT). – Enter. 01.07.2007. – K.: Derzhstandart Ukraine, 2004. – P.48
48. Gunwi County, Lee. Basics CAD (CAD/CAM/CAE) / Gunwi County Lee. – SPb.: Peter, 2004. – P. 560.: ill.
49. Lambert, H. 8-bit AVR microcontrollers Corporation Atmel: news trends of development / Helen Lambert // Components and technologies, № 6, 2009. – pp. 62- 65

50. The real time operating system [Available electronically] / CIT Forum. – Access mode: www / URL: http://citforum.ru/operating_systems/rtos/1.shtml
51. Fundamentals of software engineering [Available electronically] / SWEBOK. – Access: www / URL: <http://swebok.sorlik.ru>
52. The official website for Raspberry Pi [Available electronically]: Access: www / URL: www.raspberrypi.org
53. The official website for Altera Cyclone [Available electronically]: Access: www / URL: <http://www.altera.com/devices/fpga/cyclone/cyc-index.jsp>
54. Parkhomenko A. Interactive remote lab for hardware and software platforms research / A.Parhomenko, O.Gladkova // Theses of the International Scientific Conference «Internet-Education-Science-2014» «IES 2014» (14 – 17 October 2014 , Vinnitsa), 2014.- pp. 248-250
55. Parkhomenko A.V. Automated design of electronic devices in Creo and Altium Designer environments /A.V. Parhomenko, A.V. Pritula, V.M. Krischuk. – Zaporizhzhya: Dike pole, 2013. – P. 240.
56. Platunov, A.E. High-level design of embedded systems. Part 1 / A.E. Platunov, N.P. Postnikov – SPb .: ITMO, 2011. – P.121.
57. Platunov, A.E. Embedded Systems Management. / A.E. Platunov // CONTROL ENGINEERING RUSSIA. 2013. – № 1 (43). – pp. 16-24
58. Postnikov, E.B. Overview of international experience of creation and operation of remote access labs. [Available electronically]: Access: www / URL: http://www.efmsb.ru/download/Mirovoy_opit_sozdaniya_i_ekspluatacii_laboratoriy_udalennogo_dostupa.pdf
59. Draft imager [Available electronically]: Access: www / URL: <https://www.tindie.com/products/PureEngineering/flir-lepton-thermal-camera-breakout-2/>
60. Development of the microprocessor system based on microcontrollers [Available electronically] / Online University. – Access: www / URL: <http://www.intuit.ru/department/hardware/mpbasics/11/>
61. Russian-language resources Raspberry Pi [Available electronically]: Access: www / URL: <http://raspberrypi.su/description.html>

100. A. Parkhomenko, G. Tabunshchyk, M. Poliakov, etc

62. Sabunin, A.E. Altium Designer. New solutions in the design. /A.E. Sabunin – Solon Press, 2009. – 432s.
63. Systems for smart homes using Rasperry Pi [Available electronically]: Access: www / URL: <http://habrahabr.ru/post/237597/>
64. Sukhodolskiy, V.Y. Altium Designer. Designing functional units RES on PCBs / V.Y. Sukhodolskiy. – SPb.: BHV-Petersburg, 2009. – P.480
65. Tatarchevsky V. Application Switch-technology in the development of application software for microcontrollers. Part 1 / V. Tatarchevsky // Components and Technology, № 11, 2006.
66. Truhin A.V. On the use of virtual laboratories in education // A.V. Truhin // Open and Distance Education, № 4 (8), 2002.

PART 2

CONTROL SYSTEMS WITH PROGRAMMABLE LOGIC CONTROLLERS

M. Poliakov, T. Larionova

INTRODUCTION TO PART 2

Nowadays, it is hard to overestimate the role of electrical machines and apparatuses. The study of their operation principles, maintenance and design is an important element of electrical engineering course.

One of the tendencies in the development and improvement of electrical machines and apparatuses efficiency is system integration in an application process using “smart” on the basis of information technology. Control systems of electrical machines and apparatuses in industry and electroenergetics have multilevel structure. Industrial computers, networks and PC software, including packages of man-machine interface, represent the upper levels of these systems. Mean level contains industrial controllers – computing devices with specific programming languages oriented on process or equipment control in an industrial environment. Intelligent sensors and actuators occupy lower levels of these systems.

For design of control systems with programmable logic controllers, it is required to know the principle and architecture of industrial controllers, their programming languages, methods and tools for the project solutions formalization, methods of control problem solving with the help of man-machine interface. These issues are the course subjects.

The course is developed for professionals of industrial enterprises and organizations who want to use modern technology of electrical machines and apparatuses control in their work. It may be useful for independent work of students with specialization in Electromechanics. The course is developed for distance learning, but, if necessary, may be complemented with laboratory works in the laboratories at Zaporizhzhya National Technical University (ZNTU).

The course is based on lectures given in 1999 to students of Electrical Engineering Department (ZNTU) and specialists of Ukrainian and Russian industrial enterprises. Knowledge and skills obtained during the course “Control Systems with programmable logic controllers”, are applied by our graduates in control system of the hydroelectric power plant, electromechanics in cruise liner, design of monitoring systems of power and instrument transformers, controllers for motor control systems, regional metallurgical plants and machine-building enterprises.

4 INTRODUCTION TO CONTROLLER CONTROL SYSTEMS

4.1. Types and Properties of Controllers

Programmable controller is a semiconductor control system, which comprises user-programmable memory for filling instructions to perform special functions such as I/O control, logic, switching functions, arithmetic and manipulation with data files. The controller includes a CPU, I/O interface and memory. The controller is designed as an industrial control system. Abbreviation “PLC” (Programmable Logic Controller), introduced by Allen Bradley (USA), reflects one of the programmable controllers applications – logic control. However, modern programmable controllers are able to solve broader problems, for example, implement an automatic control system, artificial intelligence etc.

The hardware and software implementation of the controller contains the solution of typical control problems, such as interfacing with the control object, the operation control of the control device of the typical faults itself, emergency switching of the object, recovery of control process after controller power failure and other. Most modern PLC are the network ones. PLC is connected to the programming station, operator interface devices and other elements of industrial automation systems (controllers, sensors, information-measuring devices, actuators, and others) via the network.

Controllers are available in the following designs characterizing the possibility of their functional expansion:

- fixed controllers – functionally and structurally complete devices with fixed characteristics;

- modular controllers— devices that are complemented and assembled by the consumer from standard units, based on the specific management tasks, in few minutes.
- fixed controllers with the ability to connect additional I/O modules.

In Fig. 4.1 module controller SLC500, produced by Rockwell Automation Company (USA), is shown [11].



Figure 4.1 – Module controller SLC500

Models are produced by families and differ in design and system parameters, shown in the Table 4.1.

If the required number of modules in the system exceeds the slots number in the chassis, the system bus of several (usually three) chassis are joined among themselves with short (tens of centimetres) cable. Such system is called a local chassis and its input chassis are called racks. It is used in concentrated control systems.

For distributed single-processor control systems design one or more scanners are included to controller. Scanner is a network module that functionally is an input-output processor. The scanner provides

information exchange between CPU and input-output modules installed in remote chassis, using industrial network. The purpose of scanner application is CPU release from network data communication routine.

Structure of distributed single-processor controller is shown in Fig. 4.2.

Table 4.1 – System parameters of controller modules

Module	System parameters
Power supply module	Voltage, load current and voltage for power supply modules and sensors.
Chassis	Slot number for module installation
Processor	Operation speed, data capacity, user memory capacity, system channels, system instructions, consumption current
Input, output module	Number of channels, channel parameters, configuration method, consumption current
Network module	Network type, number of channels, consumption current

There is no CPU in remote chassis. Adapter in the remote chassis is also microprocessor unit, but it doesn't execute user program, it only supports the network protocol and manages the reading (writing) of information from/to input/output modules and information transfer through the network.

One of the purposes of remote chassis application is using in a single controller structurally different input/output modules. For example, one of the remote chassis must be placed in explosion hazard zone of control object, designed on the explosive elements. The rest of the chassis can be implemented in a standard (not explosion-proof), usually cheaper, embodiment. Most modern PLC are network ones. Through network PLC are connected to programming stations, operator interface devices, other elements of industrial automation systems (controllers, sensors, information-measuring devices, actuators and others). Network equipment are placed in central processor as well as in specialized modules, installed in the chassis. If there more than one channel in the controller, it can serve as a bridge from one network to another. To

the network system parameters may be referred the number of nodes in a network, length of network segments, the speed of information transfer, response time, network topology, transfer environment. In modern industrial networks the number of nodes can be up to several tens, the total length of the network segments – up to several kilometres, the information transmission time – tens and hundreds of kilobytes per second. If the control system controllers are remote from each other at farther distances, the connection between them is carried out with the help of modems using telephone lines, radio channels, satellite channels and communications through power lines. There are networks with unguaranteed (e.g. Ethernet) and guaranteed (e.g. Controlnet) message delivery time.

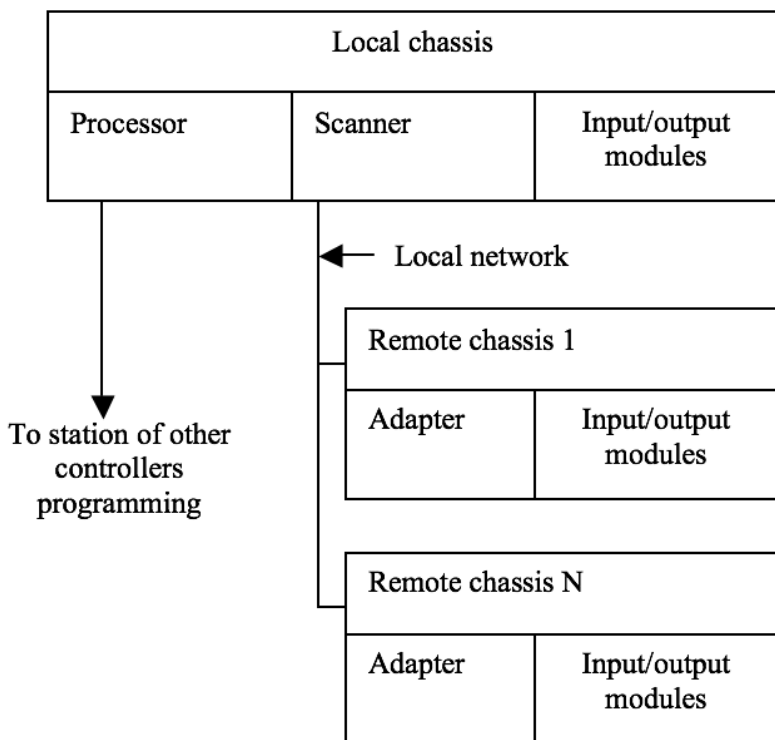


Figure 4.2 – Structure of distributed single-processor controller

Controller application became alternative to direct control of objects with the help of personal computer (PC). Proponents of direct PC-based control pay attention to possibility of larger memory capacity use, including disk, advanced software, relative cheapness of PC. Their opponents note lack of reliable, excessive complexity of both PC hardware and software, PC non-conformance of real-time system requirements.

As a result of noted shortcomings overcoming a hybrid of two control techniques, called software controller, was developed. In this controller functions of industrial controller processor are emulated by programme executed on PC processor in real-time OS environment, such as Windows CE. This PC is equipped with network cards for appropriate industrial networks. Through these networks the exchange of information with the removed chassis, where input/output modules are installed, is carried out.

In some cases, personal computer, which implements software controller, is available in a special embodiment without ventilation and moving parts, with solid Windows and ability of direct input/output modules connection. To such devices the controller EZ1131, based on computer "TECLA" (company Online Development Corp, USA, website: www.aldi.com), shown in Figure 4.3, is referred.

In reasonable control cases backup of information transfer channels, power supplies, processors and controller input/output modules is used.

4.2. Hardware of Industrial Controller

As noted above, the industrial controller is a computing device with specific peripherals (modules or blocks of input and output), which is adjusted for a specific user task solution with the help of programming. Software and hardware implementation of controller contains solutions of typical tasks of control systems design, such as communication with control object, operability control of the controlled unit, emergency shutoff of control object, recovery of control process after controller power failures and so on.



Figure 4.3 – Controller EZ1131, based on computer “TECLA”

Task of connection with control object is resolved by choice and configuration of input-output modules. Classification of input-output modules is shown in Fig. 4.4.

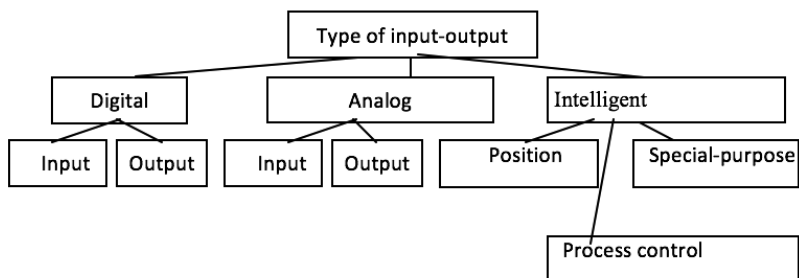


Figure 4.4 – Types of input-output modules

I/O modules are divided into digital, analogue, and intelligent. Digital module is a module, in which each input (output) channel corresponds to the tag of the Boolean-type data in the processor memory. In the analogue modules each input (output) channel corresponds to the data type of an

integer type. Intelligent modules provide some internal processing of input values or control of the output values of the user program which executes in the controller.

Selectors, sensors limiters, limit switches, push buttons, switches, circuit breakers, motor starter contacts, photocells, etc. are connected to the controller with the help of the digital input modules. Digital modules control the turning on/off of the emergency devices, control relays, fans, lights, sirens, valves, motor starters, solenoids, etc. are connected to the controller with the help of the digital output modules.

Analogue input modules perform the function of the analogue-to-digital conversion of the input signals. They are designed to interface with sensors converting the value of the physical quantity (temperature, velocity, pressure, humidity, etc.) into an electrical signal and measuring the values of the electric currents and voltages. Analogue output modules control analogue valves, actuators, etc. They carry out the conversion of digital data values into the controller electric voltage or current signal at the output of module.

Intelligent modules control such devices as encoders, flow meters, weighers, ASCII-devices, displays, power analysers, etc. They are used in specific applications such as positioning control, PID control, and communication with external devices.

The structure of basic data flows between the module and processor is shown in Fig. 4.5. The direction of data transfer between the module and processor depends on the module type: for input ones – from the module to the processor; for output ones – from the processor to the module.

Data present the information entered into the processor through the input module or derived from the processor through the output module. Configuration word is a set of fields or tags that define the operation mode of the module. Status word is a set of fields or tags, which reflect the module state.

Electric circuits of digital and analog input and output modules contain the interface node with the controller processor and channel circuits.

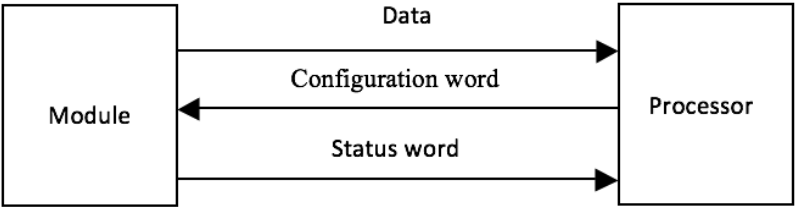


Figure 4.5 – Data flows between input/output module and processor

Structure chart of the typical quantized inputs shown in Fig. 4.6.

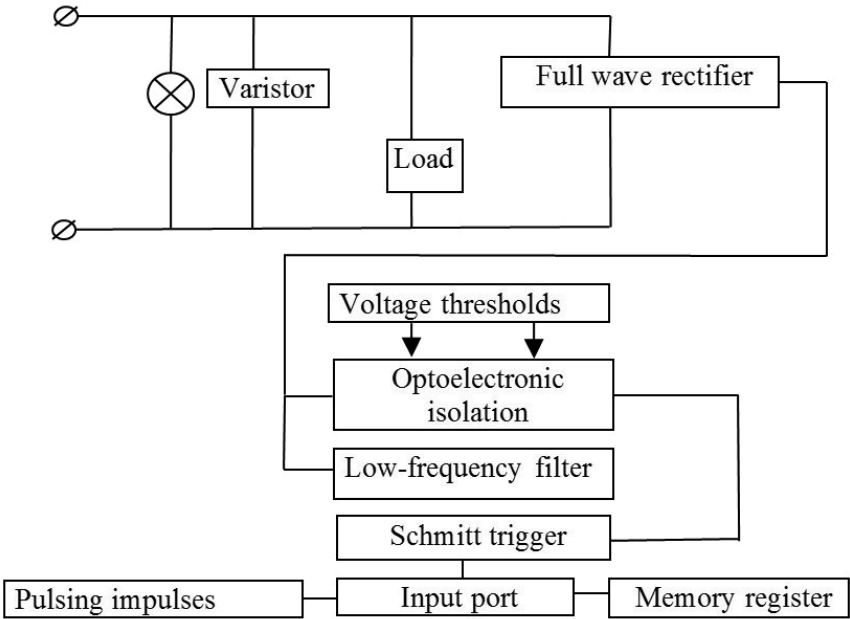


Figure 4.6 – Structure chart of the typical quantized input

As it is seen in Fig. 4.6, diagram of quantized input provides: indication of input logic level, varistor overvoltage protection, ohmic load of AC input voltage source, specification of logic levels “0” and “1” voltage thresholds, the low-frequency filtering of the input signal, optoelectronic isolation of controller main part from the input signal source, formation of signal fronts (Schmitt trigger), periodic (with a period of pulsing

impulses) savings of the current logic levels in the memory register.

Some manufacturers provide the ability to connect/disconnect input and output modules to controllers on operating RIUP (Remove In Up Power), monitoring of sensor circuit integrity and short circuits, troubleshooting of channel circuit, system time marking of channel logical level changes etc.

For quantized DC inputs it is significant the sensor type (circuit connection to the power supply) and presence/absence of interchannel isolation. Depending on the sensor type, input circuits are divided into receiving (sink) and those that eject (source) current. Diagrams of sensor connection to inputs of modules are shown in Fig. 4.7.

Quantized output modules should provide a specific power output in the load R. Switching of this power is carried out by transistor, relay or triac. The respective circuits of outputs and their connections to the load are shown in Fig. 4.8. They differ in the structure of the output cascade. The output cascade in Fig. 4.8a built on the field (FET) transistor VT with protective diode VD, fuse F and indicator L.

The output cascade circuit in Fig. 4.8b is also built on the field-effect transistor VT, which controls the relay R. R1.1 relay contacts can be connected to the AC or DC circuit.

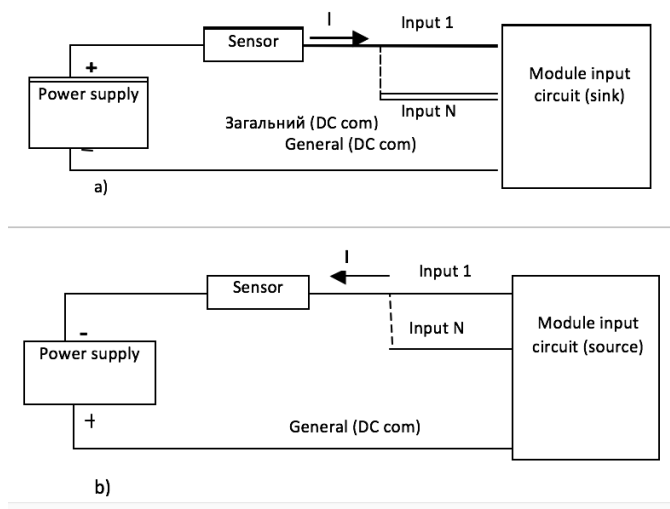


Figure 4.7 – Diagrams of sensor connection to inputs of modules:
a – sink type; b – source type

The output cascade circuit in Fig. 4.8c is built on triac VS. It contains an indicator L and safety elements –varistor VD and capacitor C.

Structure chart of analog input module channels is shown in Fig. 4.9. The diagram contains N (as a rule, $N = 4, 8, 16$) amplifier channels, connected via multiplexer to the ADC consistently in time. Multiplexing of channels allows reducing the number of ADC, but increases the data update time at the ADC output.

Channel configuration of the analog input module includes permission/prohibition channel task, acceptable range of input signals, the data presentation format in the processor, the parameters of frequency filter channel, the channel data value at sensor breakage and others.

Typically, channel configuration is carried out by entering of the configuration word to channel circuit from processor. Less frequently channel configuration is done by switches, installed on module board. For example, switch “current – voltage”.

Choice “prohibition” in the permission/prohibition channel allows disconnecting channel and thereby reduces the data update time at the ADC output.

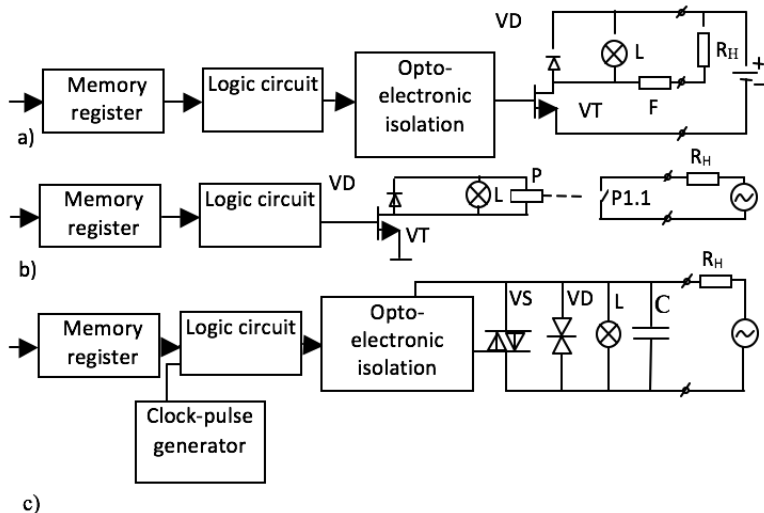


Figure 4.8 – Structure chart of the quantized output: a – transistor; b – relay; c – triac

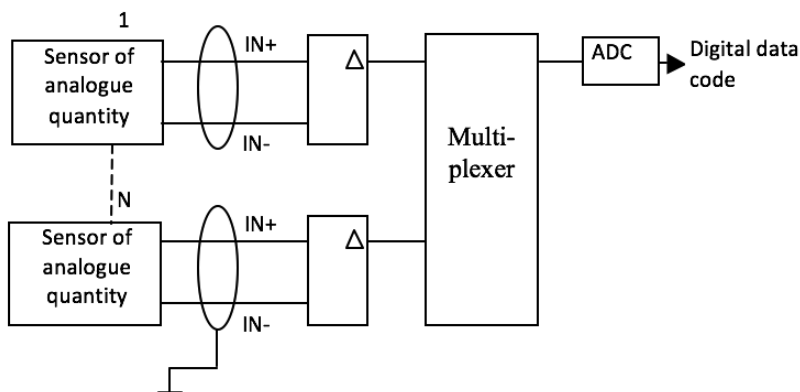


Figure 4.9 – Structure chart of analog input module channels

The range of acceptable values of input signals must match the actual signals. This allows carrying-out analog-to-digital conversion as accurately as possible. For ADC, connected to the temperature sensors, it is configured thermocouple or thermal resistance type.

The most common data formats in processor are:

Engineering units – an integer equals to the number of millivolts (microamperes) of converting voltage (current). For example, if the converted voltage equals to 2.53 Volts, this corresponds to 2530 engineering units.

Proportional count – the maximum and minimum possible output codes of the ADC correspond to limiting values of acceptable range of input signals. For example, the acceptable range of input values $[-10, +10]$ V, the range of decimal equivalent digital codes at output of 14-bit ADC $[-32768 + 32768]$, at ADC input signal voltage $U_{in} = 1V$. Then decimal equivalent of the signal code is defined according to the formula:

$$code = 1V \cdot \frac{[32767 - (-32768)]}{[10 - (-10)]V} \approx 3268.$$

For the channel frequency filter the cut-off frequency is configured. The lower this frequency, the more effective filtering of high frequency noise at the channel input, but at the same ADC reaction time to input signal jumping increases. In Fig. 4.10 it is presented the dynamics of

code changes at ADC output for two filter variants. Curve 1 corresponds to the filter with higher cut-off frequency than in the filter corresponded to curve 2.

To reflect via the data channel register the sensor breakage from the channel input, it is needed to enter in the register an extraordinary value, which does not occur during normal use. For example, 0°C, when measuring the metal temperature in the furnace. This value is chosen at module configuration.

Status word of output module contains tags confirming reception of configuration word by module, tags of module faults and its individual channels.

Analog output modules consist of interface node with controller bus, control nodes and channel nodes. Channel node includes memory register, logic circuit and optically isolated from the controller main circuit DAC, buffer amplifier and converter voltage-current.

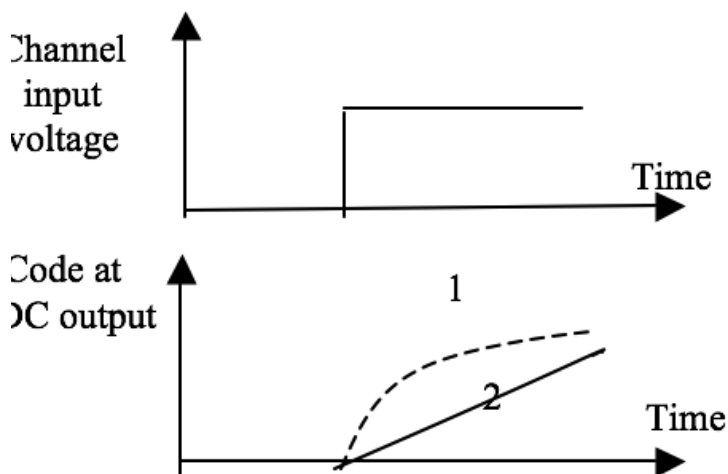


Figure 4.10 – Response at ADC output

Output modules, as well as input, support various data formats in the processor and have a number of additional features. For example, the module family Controllogix [12] allow configuring the maximum rate

of signal change, perform fixation (restriction) of output signal if the digital code at the module input go beyond the specified limits, capture (definition) of output signal value to initialize the module at fault or during programming, detect breakage of wire connecting the module output with the load, form the alarms (status bits).

4.3. Functional Organization of Controller

During solving of control problem, a set of hardware and software is used in the controller, which should be divided into functional levels (interface with the control object, hardware control, system typical complex operations, states and adaptation), as it is shown in Table 4.2.

On the lower level there is control object, unit, which technical process is described with inputs and outputs.

Inputs are divided into controlled and uncontrolled ones. Through uncontrolled inputs distributing actions, such as ambient temperature, load unit settings, etc. come to system. CO controlled inputs are associated with actuators. CO outputs carry information about CO condition which comes to control system via sensors.

Quality of CO interaction with control system is described by controllability and observability parameters.

Controllability indicates whether only one set of controlled inputs is enough for control implementation, i.e., achieving of the necessary CU conditions.

Observability characterizes sufficiency of a set of outputs and sensors connected to them for assessment of CO conditions in order to achieve control goals.

On the next level controller hardware are presented: input, output modules, CPU, network module, power supply, operator interface node. In terms of control it is essential that these aids are configured ones. In addition, through the controller hardware additional control circuit may be closed in. For example, by analyzing color and lighting (blinking, continuous) of processor status LEDs the operator makes a decision which comes to the controller via operator interface node. And the information from the voltage network sensor controls the program procedure of power failures processing.

Table 4.2 – Levels of controller control

Level	Elements	Informational arrays	Location
Control of adaptation	Productive system	Knowledge base of application, objective function	User's memory
Control of states	Control automata (CA)	CA table of inputs and transfers	
Control of complex operations	Input, output operational automata (OA)	Data files (structures), forcing files, OA parameter files	
Control of standard operations	Instruction of programming language	Control structures	
Control of system operations	Controller operation system	Controller status file (of system variables)	Controller system memory
		-	
Hardware control	Processor; input, output, network modules; power supply unit; operator interface	-	Chassis, control
Connection with control object	Sensors, actuators		Customer object
Control object (CO)	Equipment, physical, technical process		

Via network module controller communicates with programming station, other controllers, intelligent devices (e.g. drives, power quality analyzers, operator interface) computers and operator workstations.

Sources and consumers of this information may be on all higher levels of controller control.

Control of system controller operations is an important control level. This control is executed by the controller operating system (OS) that performs following functions:

- loading, remote control of modes of user's application software automata;
- saving/recovery of user application conditions at failures/controller power switching;
- organization of software scan, including the information exchange between hardware and user memory, control of multitasking and interruption, exchange via networks and operability diagnostics of controller hardware.

OS elements are not available for the use. User can only read and write system variables, stored in controller status file.

All levels that are higher than OS level is software automata stored in user memory.

Most programming language instructions have rather complex execution algorithm, with internal states that express through control structure elements. From this point of view, they can be seen as automata of typical user operations combine they can be combined in the separate control level.

Automata of typical operations are used in operational and controlled automata.

In level of operations control let's single out input and output OA. Input OA execute operations with different types of input variables.

Typical tasks, solved with input OAs:

- calculation of logical functions;
- calculation of predicates;
- count of events;
- calculation of functions using arithmetical and logical operations, trigonometrical and other functions;
- data base processing (search, filtration, sorting);
- decompression, transformation of codes and values, data control, recognition, statistical data transformation, identification of priorities.

OA operation results come to control automation (CA) inputs in a

form transfer conditions and (or) in a data form come to inputs of other OA.

Output OA forms values of output controller variables. Let's distinguish controlled and uncontrolled output OA. Controlled OA are activated by output signal of control automaton, and uncontrolled ones – by inputs of output OA. In a separate case if it is required permanent OA activity they have no inputs at all.

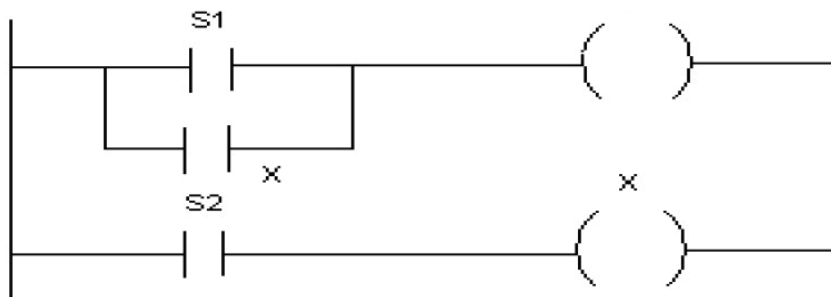
Typical output OA: generators and former of stimulating actions on CO; qualificator steps of control automaton; regulators of CO parameters; synthesizers of data structures and arrays.

Control level of control system states is represented as control automata in the control structure. Each CA usually controls states of individual task, program, procedure that is automata can be used in parallel. Only single-task application without procedures for processing of interrupts can be controlled by single CA.

At each instant of time CA can be in only one state. If vector of inputs X , which comes from OA, changes, CA transfers to a new state. Simultaneously at CA vector of outputs Y is formed which controls the activity of output OAs.

CA can be designed as Mealy or Moore machine. Moore machine connects separate state with vector of outputs, i.e. with the actions that performs output OA, activated by CA. In Mealy machine state defines the interpreting method of the vector of inputs X . Each of allowed in a given state vector X is associated with vector Y and a new state.

In both cases – state is information about previous CA, needed for determination of its future behavior.



a)

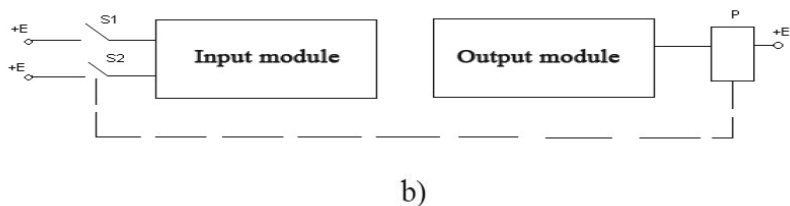


Figure 4.11 – Example of “parasitic” states formation: a) – program;
b) – controller connection diagram

When designing CA it is needed to take into consideration the possibility of “parasitic” states formation via feedbacks at lower control levels. For example, let’s using controller it should be implemented a function of relay R switching control with the help of contact S1 or software indicator H. In Fig. 4.11 it is given program fragment in LD language (see Section 5.1), implemented by corresponding CA.

Relay R controls closure of contact S2 (Fig. 4.11b). Separately subcircuits Fig. 4.11b are combinational relay circuit, and the program 4.11a implements logical functions and status of output R has not a memory property. However, due to the hardware feedback, contact S2 state depends on the relay R state. According to the program state of relay R depends on the contact S2. Therefore, at the first contact S1 closure, relay R will switch on and will be self-held. A new stable controller state will appear and function of relay control will be lost.

Data processing algorithms and state control can be changed during operation of controller control system. For this purpose two methods are used: reprogramming and adaptation.

Reprogramming is carried out by programmer, who replaces the executable code of application. Adaptation can be performed automatically without human intervention. When adapting control results are accumulated and evaluated, changes of controlled and uncontrolled CU parameters are predicted. Based on these data information arrays that represent the automata (control and operational ones) structure, controller system variables are changed.

Adaptation control can be singled out as a separate control level that uses the tools of artificial intelligence. These tools require significant CPU resources, and therefore the adaptation level can be implemented in application, which is external to the controller, such as model in

MATLAB software package, which communicates with the controller via Dynamic Data Exchange (DDE) features.

Let's consider interactions of hardware and controller operational system at level of system operations control.

Important elements of the hardware interaction and controller operating system are the principles of the controller memory splitting, the structure of the user application, the operating modes of the controller, the concept of the program scan and forcing. In memory of the controller, one can identify the system and the user area. System memory contains the controller operating system and this part of memory is unavailable to the user. The user memory contains data and user program. There are two types of data organization in memory. In the first method, the same types of data are combined into files: the bit data, integer data, timer structures, etc. In this case, to access the data it is necessary to specify the file name, the serial number of words in the file and the number of bits. The second method lies in assigning names to all data elements. Named data are called tags. Addressing to the tags in a program is carried out by their names.

Program organization in the user memory depends on the type of application: single-tasking or multi-tasking. In single-tasking application, one indicates the major program (procedure), procedures of subprograms, time and event interruptions and procedures of error handling. The main program (procedure) is performed continuously, that is, it automatically restarts after completion. Subprograms are run from the main program or subprogram (nested subprograms). Timed procedures are replenished periodically, after the predetermined time interval. Configuring the event in order to start an interrupted program by this event includes setting the address of the input channels of digital modules and data values. For the time of the procedure interruption, the main program is stopped. Interrupts can be banned from the main or other procedures.

One cycle of the main program performance is called program scan. Its stages is shown on Fig. 4.12

In the first phase of the scan, data from the memory registers of the input modules is written to the user memory of the processor via the system of buses. In a second step, the control program is scanned; and, in accordance with logic embedded in it, the new values of the output variables are formed and written in the user memory of the processor. Only at the third phase of the scan, the values of these variables through

the bus system are written to the memory registers of the output modules. As a result, the new exposures are given to the controlled object. In the fourth stage of the program scan, messaging via networks is performed; the fifth is the performance monitoring and fault diagnosis of controller hardware.

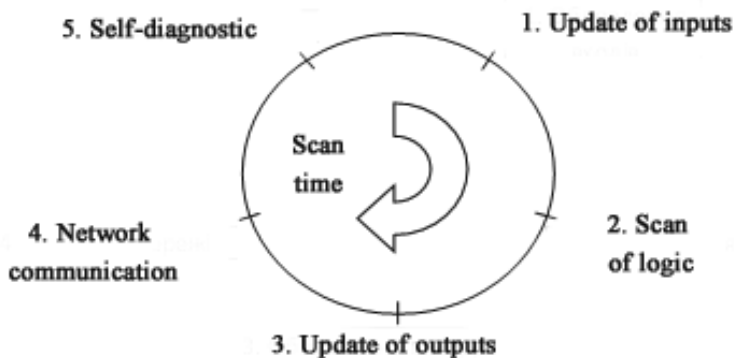


Figure 4.12 – Program scan stages

Typical structure of a multivalued application contains tasks, errors, power failures evaluator, and system data.

System data of the application includes controller tags, data input/output system, data controller. Task is divided into the programs. The application has only one cyclic (restarting after execution) task with the lowest priority; thus, it can be interrupted by any periodic task. Programs consist of procedures, program tags and data of the task. Data of the task include configuration data, status, and the watchdog timer. The configuration data of the task contains information on the order of the start, priority, execution time and the type of task (periodic or cyclic). Program tags of tasks, unlike a controller tag, are not available for other application tasks. Composition of program procedures is similar to that of single-task application procedures.

Feature set of controller is associated with modes of its processor. Description of typical processor modes is presented in Table 4.3.

Processor mode is set by the position of the key on the front panel of the CPU.

Table 4.3 – Description of typical processor modes

Mode	Description
Program	Program change with the help of programming package. User program is stopped
Run	User program (application) runs. Program execution results can be viewed in programming package. Program edition and processor mode change are impossible in programming package.
Remote run	User program (application) runs. Program execution results can be viewed and processor mode can be changed in programming package. Program edition is impossible in programming package.
Test	The program is executed in the same way as in the mode Remote Run, but output values are calculated in the user program and are not transited to te output modules.

Under forcing of I/O one understands the ability to change the values of the input and output of the controller. If forcing is enabled, these values are taken from the tables of forcing, which are the part of the user's application. Forcing is used, for example, to force a shutdown, stop, and transfer to a safe condition of the unit, the user process in case of accident.

5 CONTROLLER PROGRAMMING AND FORMALIZATION OF DESIGN SOLUTIONS

5.1. IEC61131 programming language overview

Programming of applications for controller control systems is a separate line in the programming, which has its own peculiarities, and significantly differs from, for example, programming of microcontroller systems or solving of data computing problem on PC.

The International Electrotechnical Commission (IEC) solves the programming languages standardization problem. IEC 61131-3 defines the general requirements for programming languages and types of languages for programmable controllers. Programming languages defined by IEC 61131-3 are shown in Table 5.1.

LD Language presents the program logic in a form similar to an electrical circuit with circuits consisting of contacts and coils of the relay. This form was especially popular during the transition from relay automata to programmable controller. Program in LD consists of rungs. Each rung is a program being analogues to the logic of the electrical circuit. Rung contains conditional and executive instructions and is equivalent to the conditional execution operator IF condition THEN action, where the condition is represented by one or more conditional instructions, logical connections between which are defined by graphical branches of rung, and the action is represented by one or more executive instructions execution, sequence of which is also set by the graphics branches of a rung. Example of the rung is shown in Fig. 5.1.

Modern LD is a high level programming language, which contains about a hundred instructions, including such complicated instructions as PID – control, the transmission of messages over a data network to other controllers, operations under the matrix, the synchronous movement of the axes.

Table 5.1 Standard types of controller programming languages

Designation	Denomination	Type of language	The area of the most effective application
LD (RLL)	Ladder Diagram Relay Ladder Logic	Graphic	Discrete logic.
SFC	Sequential Function Chart	Graphic	Building of a functional specification of the system, control organization as a sequence of steps or states through which the system runs.
ST	Structured Text	Textual	Complicated algorithms, work with string data, files, data structures.
FBD	Function Block Diagram	Graphic	The processes of continuous control, control algorithms, which are performed continuously.
IL	Instruction List	Textual	Where minimizing of the execution time and memory code volume is required.

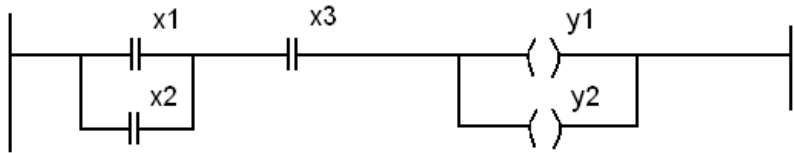


Figure 5.1 – Rung of programs in LD

Language FBD also represents the program logic in the form similar to an electrical circuit, but unlike the LD, it resembles the unit with

digital logic elements, adders, counters, etc. FBD diagram describes the function of the output variables on the input variables using standard elementary functional units and connection lines. Nomenclature of functional units approximately corresponds to the nomenclature of LD language instructions, i.e. includes functional blocks that perform data management, Boolean, arithmetic and comparison operations, converting of data, mathematical function, blocks of timers and counters, and others. Standard function block is recorded on FBD diagram as a rectangle that has a certain number of inputs and outputs. Inputs of the standard function block are located on the left side of a rectangle and are connected to the output variables or outputs of other functional units. Outputs of the standard function block are located on the right side of a rectangle and are connected to the output or input variables of other functional units. The connection line is directed and transmits its data from input to output.

FBD Diagram may comprise several subcircuits having non-crossing subsets of input variables. These subcircuits are called chains. Programs corresponding to the circuits are performed sequentially from the top to the down in the order of the FBD diagram.

Example of circuit in FBD diagram is shown in Fig. 5.2. Here dependence of output y from input variables $in1$, $in2$, $in3$ corresponds to formula:

$$y = (in1 + in2) / 2 - in3$$

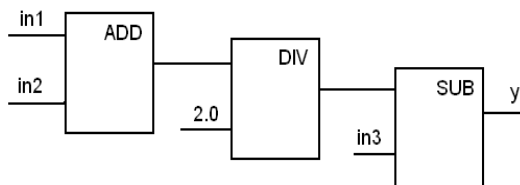


Figure 5.2 – Example of circuit in FBD diagram

SFC language presents the program logic in the form of a directed graph. The vertices of the graph correspond to the steps, and the arcs correspond to the transitions of the control process. The execution process of the SFC program can be represented as a process of transition from step to step. When the step becomes active, it triggers certain

actions, or may cause a child SFC program. Transitions of SFC programs describe conditions for termination of the current step activity and the transferring it to the next step. Actions of the steps and logical conditions of the transition are programmed in LD, FBD, ST, and IL.

Alternative branches of the SFC column are enclosed in single lines of multiple connections. Each branch begins and ends with its own transition condition. Testing of the alternative conditions is performed from left to right. If the true condition is found, the other alternatives are not considered. Double horizontal lines designate parallel executing branches of the SFC graph. These branches have the same input and output condition of the branch. Transition condition, terminating the parallelism, is checked only when the last steps in each parallel branch are active.

In Fig. 5.3 example of SFC program is presented. Step with double square outlining is initial one, steps 1 and 2 are executed sequentially, steps 3, 4 are located in alternative branches, fragments with steps 6, 8 and 7, 9 are executed in parallel.

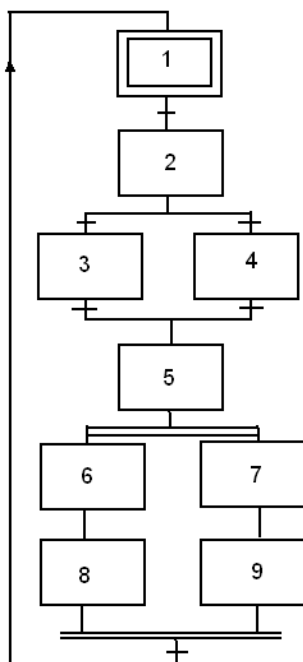


Figure 5.3 – Example of SFC program

ST language is often called “Pascal-like” because of the syntax and operators similarities. ST program is a list of operators ending with a semicolon. ST language is used to describe the conditions of the transitions and actions within the SFC program steps. From the ST program, it is easy to call programs in ST, IL, LD or FBD languages, and in some software and “C – functions”. ST language uses traditional operations (arithmetic, logical, comparison) and operators (assignment, return, branches, and loops).

Main components of ST language are presented in Table 5.2.

Table 5.2 – Main components of ST language

Name	Description	Examples
Assignment	For assignment values to tags	tag:= expression
Expression	Expression includes: - tags – designation of memory space, where data is stored; - immediate values – numbers, symbols; - operators – symbols or mnemonics for specification of operations; - functions – at execution they return one value.	Value 1 4 tag1+tag2 tag1>= value 1 function (tag1)
Instruction	As opposed to functions instructions cannot be used in expressions	Instruction (); Instruction (operand);
Construct	Specify type of algorithm processing (branched, cyclic)	IF...THEN CASE FOR...DO WHILE...DO REPEAT...UNTIL EXIT.
Comment	Text which doesn't influence on program execution	// comment (* start of comment...end of comment*)

Example of ST program is shown in Fig. 5.4.

```

PROGRAM ST_Example
VAR
  in1,in2,in3,y:REAL;
END_VAR
BEGIN
  y:=((in1+in2)/2)-in3;
END.

```

Figure 5.4 – Example of ST program

Language IL is an “assembler-like” language. Assembler languages are widely used in programming of microcontroller systems [13], which can be applied as prototypes when designing control systems based on industrial controllers. As it was stressed, code compactness and short runtime of IL programs – is a reason or application and learning of IL language.

The program in IL is a sequence of instructions. Instruction IL may comprise from one to four fields: label, operator, operand and comment, which type and purposes are given in Table 5.3.

Table 5.3 – Structure of IL instruction fields

Name	Type	Purpose
Label	optional	Specifies address of transfer
Operator	mandatory	Defines actions, executed by instruction
Operand	optional	Defines data, on which actions are performed. If field is absent, then operand is chosen by default.
Comment	optional	Description, explanation of actions performed by instruction

Standard IL operators are presented in Table 5.4.

Example of IL program functionally similar to FBD program (Fig. 5.2) and ST program (Fig. 5.4) is shown in Fig. 5.5.

Table 5.4 – Standard IL operators

Operator	Description
LD	Load value of operand to accumulator
ST	Assign accumulator value to operand
S	If accumulator is TRUE, set operand value TRUE
R	If accumulator is FALSE, set operand value FALSE
AND	Bit-by-bit AND
OR	Bit-by-bit OR
XOR	Bit-by-bit exclusive OR
ADD	Addition
SUB	Subtraction
MUL	Multiplication
DIV	Division
MOD	Modulus
GT	>
GE	>=
EQ	=
NE	≠
LE	<=
LT	<
JMP	Transfer to label
CAL	Call of subprogram, functional block
RET	Exit and return to program, which have called subprogram

```
LD in1
ADD in2
DIV 2
SUB in3
ST y
```

Figure 5.5 – Example of program in IL language

Besides standard operators, there are additional ones that are not supported by all controller platforms.

Some operators may join modifiers that change the meaning of the operation. To such modifiers refers: N – Boolean negation of the operand; “(“ – delayed operation; C – conditional operation. For example, while

execution of the instructions AND with modifier N (ANDN x), AND operation is performed on an inversion of the operand x. A sequence of instructions

$$\begin{array}{c} \text{AND}(x1 \\ \text{OR } x2 \\) \end{array}$$

means that operation AND is performed on result of OR operation with operators x1 and x2

C modifier is attached to the operators JMP, CALL, RET. In presence of modifier these operators take the form JMPC, CALC, RETC and corresponding instructions are executed only if the accumulator register (result) has Boolean value TRUE (or nonzero not Boolean value). C modifier can be used in conjunction with modifier N. For example, an instruction with operator JMPCN will execute only if the accumulator register has a Boolean value FALSE.

5.2. Software Structure of Industrial Automation Systems

Software of industrial automation systems includes user application, tools for development and usage of this application in the environment of industrial automation system. In general, the application consists of several user applications executable on controllers, intelligent information-measuring devices, operator interface devices and on personal computers in the environment of man-machine interface programs and other programs such as Excel, Access, MATLAB and others. Software for development and usage of user applications can be imagined as a house (Fig. 5.6).

Operating system of controller occupies the first floor of the house. Functions of controller OS was discussed above.

Software of all the further floors is located outside the controller – in PC programming station or PC operator workstation.

Tasks of communication server are configuring and control of the status of direct communication drivers for various versions of controllers associations in the network; display of active nodes in networks connected to the PC; data configuring for DDE data exchange between applications included in the control system and network monitoring.

Programming environment provides creating and documenting of

control application, its verification, translation of programming language to downloadable code, downloading of this code to the controller, control of controller modes, online monitoring of application execution in the controller, unloading code that runs in the controller to programming package and its representation in standard programming language. Application development involves the choice of processor, the method of communication with the controller, chassis size, configuring of input/output modules (selection and placement in chassis slots, assignment of channels parameters), specification of application structure (configuration of tasks, programs, procedures, data), insertion of user logic and comments in standard programming language and other operations.

Floor	Name	Description
6	Office	Packages of information interchange with control application via Internet and mobile networks
5	Informational	Packages of information interchange with control application and database of higher enterprise levels
4	Visualization	Packages of man-machine interface, configuring of operator interface means
3	Programming	Programming environment. Controller emulator
2	Communication	Communication server
1	Controller OS	Real-time OS

Figure 5.6 – Software structure for development and execution of user applications

In cases when real controller is absent run and debug of the user program can be performed using the emulator. The emulator is a software product that runs on the PC programming station and emulates user application execution for one or more controllers. The results of the user program execution using the controller emulator can be viewed

with the help of programming package. Emulator screen allows mode control of emulated controllers that is run and stop of the program, set file name of executable program. The emulator doesn't always provide 100% compliance with the real controller, especially when it concerns peculiarities of the real-time program run. However, this is a very useful tool for developers.

The visualization package provides the creation, documentation and execution of visualization application on PC or its uploading to the operator interface panel. These packages are specialized for each type of device, with the help of which visualization is performed.

Creating a visualization project includes the creation of tag database, system of virtual control screens, alarm subsystems, logging of data, actions and other subsystems.

Tag visualization project is an analogue of variable in the standard programming languages. They are divided into the instrumental and system tags. Instrumental tags correspond to a named memory space in the controller; system tag is generated by the visualization system. Typical examples of system tags: seconds (System/second), minutes (System/minute) of system time. Tag value is entered to the tag database at visualization application runtime on a periodic predetermined base. Information transfer in the opposite direction, i.e. from the tag database to the controller is made in response to some event in the visualization system. For example, at a specific time, under certain conditions, mouse click on the screen.

The system of virtual control screens contains graphical objects, displays and controls that in real time present parameters, essential for operator decision-making.

At the fifth (information floor) level software performs real-time control of data exchange between the components of user control application (of object or process) and databases of higher levels, such as the enterprise level.

At the sixth (office floor) level software provides services for remote access via the Internet to information of control application. Among the typical services there are forming and sending of prepared by system emails and SMS on mobile phones of staff working with user application object, remote browser access to virtual control screens of visualization application. Roughly speaking, staying at the resort or in a distant trip, you can go to the Internet cafe, type cherished IP address and password,

open the Control screen of visualization application, obtain the necessary information, and for example, switch off the actuator in the control object.

5.3. Ladder Diagram (LD) language: construction and basic instruction

Ladder Diagram (LD) language is a high-level graphical language. As it was noted before, the program in LD consists of rungs executed sequentially. To change the order of rungs execution, organization of branching and looping in the program instructions for the program flow control are used.

All instructions of the LD language are divided into conditional – checking verity of certain conditions and executive – to perform operations with data. Language instructions are placed in series and (or) a parallel branches of conditional and executive parts of the rung.

By functions of LD language instructions are combined into groups: bit, timers, counters, comparison, math, logic, program flow control, replacement, conversion, special operations, ASCII, and other instructions. List of the available for the selected processor instructions is provided in the manual of the controller and in the reference section of the programming environment.

Each instruction has a graphical designation, mnemonic, type (executive, conditional), operands, and in some cases, the control structures. The performance results in the instructions execution, especially mathematical, may affect the condition of the mathematical processor register.

Bit instructions are used to monitor and control the status of bits. Descriptions of some bit instructions are given in Table. 5.5.

Timers and counters control the operations based on time or number of events. All instructions in this category are the output ones and all, except instruction RES, have a number of parameters, combined in structure associated with the tag of a timer (counter). Denomination and description of the instructions given in the Table 5.6 and the parameters of the structures – in Table 5.7.

Table 5.5 – Bit instructions

Mnemonic	Denomination	Type	Description
XIC	eXamine If Close. Check on the status ON	Conditional	Truth (enables output) when the bit is set.
XIO	eXamine If Open. Check on the status OFF	Conditional	Truth (enables output) when the bit is cleared.
OTE	OuTput Energize. Enable output	Executive	When the instruction is enabled, it sets the data bit, and when prohibited – resets.
OTL	OuTput Latch. Freeze output	Executive	When the instruction is enabled, it sets the data bit. Prohibition of instruction does not change the state of a data bit.

Instructions of comparison are a category of input instructions intended for comparison of the values. It includes the following instructions: EQUAL (EQU) – checks the equality of two values; Not Equal (NEQ) – checks the inequality of two quantities; Greater than or Equal (GEQ) – checks when the value of A is greater than or equal to the value B; GReTer then (GRT) – checks when the value of A is greater than the value of B; Less than or Equal (LEQ) – checks when the value of A is smaller than B or equal; Less than (LES) – checks when the value of A is smaller than B; LIMit test (LIM) – checks when the value Test is within the range Low Limit and High Limit.

Math instruction is a category of output instructions that perform arithmetic operations of addition (ADD), subtraction (SUB), division (DIV), multiplication (MUL), taking the root (SQR), calculate the logarithm (LOG), sine (SIN) and other. They are executed each time the instruction is scanned until the rung conditions are true. The results of the computation instructions are entered in the operand Dest (Destination), and change the arithmetic status flags such as C – carry (Carry), V – Overflow, Z – Zero (Zero), and S is the sign (Sign).

Table 5.6 – Instructions timers and counters

Mnemonic	Denomination	Description
TON	Timer ON delay	Non-retentive timer that accumulates time at the resolution of instruction execution until the execution is forbidden or accumulated time ACC reaches the setting PRE. Since the equality of ACC and PRE bit of performance DN is set. When the execution of instruction TON is forbidden, accumulated time ACC and bit performed DN is reset.
TOF	Timer OFF delay	Non-retentive timer that accumulates time when the resolution of instruction until the execution is forbidden or accumulated time ACC reaches the setting PRE. Execution bit DN is set from the moment of permission of the instructions until equality of ACC and PRE.
RTO	Retentive Timer On	Timer accumulates time at the resolution of the instruction execution. Since equality ACC and setting PRE, execution bit DN is set. When execution of instruction RTO is forbidden, accumulated time ACC is saved. To reset the value of the ACC instruction RES is used.
CTU	CounT Up	At each resolution of STU instruction execution, the accumulated value ACC is incremented by one. From the equality of ACC and setting PRE, the execution bit DN is set. To reset the value ACC instruction RES is used.
CTD	CounT Down	AT each resolution of CTD instruction execution, the accumulated value ACC is decremented by one. Execution bit is set when the ACC is more or equal to PRE. To reset the value ACC, the instruction RES is used.
RES	RESet	In timers and counters, control status bits also reset the accumulated value ACC, and in the structures of control – the value of the position of the current element POS and control status bits.

Table 5.7 – Structure parameters of timers and counters

Designation	Instruction	Description
ACC	TON, TOF, RTO	Accumulator (ACCumulator), the value of accumulated time
	CTU, CTD	The value of the accumulated events
PRE	TON, TOF, RTO	Setpoint (PREset)of time
	CTU, CTD	Setpoint (PREset) of events
DN	TON, TOF, RTO	Bit of execution (DONE)
EN		Bits of resolution (ENable)
TT		Bit of count (Timer Timed)
CU	CTU	Bit of resolution of direct account (Count Up)
CD	CTD	Bits of resolution of countdown (Count Down)
OV	CTU, CTD	Overflow bit (Over flow), is established at the time of transition from the maximum positive value in the accumulator to a negative value
UN		Underflow bit (Under flow), established at the time of transition from the maximum negative value in the accumulator to a positive value

Logic output instructions, such as bitwise AND (AND), OR (OR), exclusive OR (XOR) perform logical operations on bits. The result of the instruction is stored in the operand Dest, and changes the arithmetic flags status.

Flow control program instruction is used to change the logic of the program, i.e. the sequence of rungs. Typical representatives of this category: Jump (JMP), Label (LBL), Jump to Subroutine (JSR), Subroutine (SBR), Return (RET).

When the JMP instruction performance, the program skips to the operand specified by label name instruction LBL. Next, the program

continues to run within the rung, with suitable instruction LBL. Jump forward and jump back instructions should be distinguished. At forward transition a number of rungs are skipped (not executed). Move back through the program is executed when the rung number with LBL instruction is not greater than the rung number with JMP instruction. At the back transition program loop is formed and increased during program session.

JSR instruction in conjunction with SBR and RET instruction manages the execution of a single subroutine within the program so that the subroutine is executed once, and program execution continues with the place from which it was called when the JSR instruction moves to the execution of the subprogram.

Sequencers, shift bit strings and PID-control we refer to the extended instruction manual.

Sequencer instructions are used to perform sequential and repetitive operations with elements of data arrays. Each time the instruction Sequencer Output (SQO) operates at the destination operand the next word from the array data is sent. The bits of the word can contain instructions for turning on/off of the control system actuators.

Instructions of bit string shift, for example, Bit Shift Left (BSL) operates each time the instruction produces a shift by one digit of bit strings, which are given in the instructions by the starting address and length.

Proportional Integral Derivative (PID) instructions is an output instruction that controls physical properties such as temperature, pressure, liquid level, or rate using process loops. During programming, one enters parameters Control Block, Process Variable and Control Variable. Control Block is the file that stores the data required to operate the instruction. Process Variable – is an element address that stores the process input value. Control Variable is an element address that stores the output of PID instruction.

5.4. Formalization of control tasks

One of the important design stages, including designing of control controller systems, is the formalization of control tasks. The variety of formalization description means is used for the following reasons:

- As a result of theoretical study of the problem, there is an analytic or graphic description of the problem or its fragments in the form of Boolean expressions, predicates, automatic tables, graphs of automatic machines, algorithms, Petri nets, etc.
- Before design stage there is documentation of the functional analog control system on another element base. For example, in the form of the electrical circuit of logical assembly, relay-contactor assembly, program in other programming language (not the same that is used in the controller).
- Functional analog of control system is obtained by modeling the object and the controller in simulation package such as Simulink, Stateflow.
- Design of control system is carried out using one of the methods of architectural design, such as COMET (Concurrent Object Modeling and Architectural Design Method) and documented using the Unified Modeling Language (UML). Therefore, software developer should be familiarized with these types of control system description and methods of their conversion into the program in language, interpreted with controller.

The term “algorithm” is derived from the name of a medieval Uzbek mathematician Al-Khwarizmi and is applied for the definition of formal requirements to the order of some operations for solving of all the problems of specific type, in particular, the order of execution of operations during the operation of the system “the control unit – the control object”.

For a formal description of the algorithms the languages of circuits, transition formulas, matrix and flowcharts of algorithms and their modifications are used. Logic circuit (LC) is an expression composed of consecutive operators and logical conditions, and arranged in a certain way by the branching of numbered arrows. For example, let LC is given:

$A0\ z1\ \uparrow^1\ A1\ \downarrow^1\ z2\ \uparrow^2\ A2\ A3\ z3\ \uparrow^2\ A4\ \downarrow^2\ A5\ Ak$

This LA has start operators A0 and Ak, five operators (A1 – A5) and three logical conditions (z1 – z3). Operators of the algorithm are performed by the LSA from left to right with taken into account the values of logical conditions. Thus, from the initial state A0 condition z1 is checked. If z1 is executed, the next statement is executed, standing at the right of the LSA z1, i.e. A1. Otherwise, it changes along the way:

the upper arrow ($\uparrow^1$) – bottom arrow with the same number ($\downarrow^1$). Next, condition z_1 is checked, which is from the right side of the bottom of the arrow, i.e. z_2 . The algorithm usually ends when reaching A_k , or when, at a certain stage there is no member of the diagram that should be performed.

5.5. FSM models of the control system behavior

Automata theory is the foundation of modern theoretical and practical computer science. Automaton description is widely used in the design of both hardware and software logic control systems. Unlike functional converters, on basis of which digital combinational assemblies are constructed, automata have the memory property. That is, their response depends not only on the input at any given time, but also on the values of inputs at previous time moments. Automaton saves its history in the states. Condition is a characteristic that uniquely defines the reaction of the automaton on subsequent input events. In practice, automata with a finite number of states are widely used. They are called Finite State Machines (FSM).

An important place in the theory two main machines – Mealy and Moore – is occupied. Let a finite automaton is given by six objects $A = \langle S, X, Y, \sigma, \lambda \rangle$, where S – the set of states, X – the set of input signals, Y – set of output signals; $\sigma: S \times X \rightarrow S$ – transition function; λ – the output function. For Mealy function $\lambda: S \times X \rightarrow Y$, and for Moore machine $\lambda: S \rightarrow Y$. It is convenient to give transition and output functions in tabular form, and for graphic description of the machine it is convenient to use the directed graph, whose vertices correspond to the states and arcs correspond to the transition from one state to the other on. With the help of Moore machine it is possible to construct the equivalent Mealy machine, and vice versa. The equivalent machine can has different number of states and semantics. It is often said that Mealy FSM is tied to transitions, while Moore FSM – to the states. Machines can be implemented in software and hardware. Software implementation of the automaton is documented in the form of flowcharts, hardware – in the form of circuit diagram of device. Because the controller systems are often designed to replace existing hardware and software machines, methods of converting these documents into graphs of automata and vice versa are of great interest.

The machine graph design procedure according to algorithm flow chart includes mark-up of algorithm flowchart for defining a set of states of the machine and determination of ways on algorithm flowchart for determination of the machine transitions. Design of Mealy and Moore machine differs in mark-up rules for algorithm flowchart. For Mealy machine mark-up there are following rules:

1. The symbol S_0 indicates the input vertices following the initial and the final input vertices;
2. The input of each vertex that follows the operator marked by a symbol S_p and different vertices are assigned with different symbols;
3. If the input vertex is marked, it is done only with one character.

For Moore machine mark-up rules of algorithm flowchart are:

1. The symbol S_0 indicates the start and end vertices of algorithm flowchart;
2. Each operator vertex is marked by S_i which is unique and distinct from the other symbols;
3. Each symbol S_p , which is a result of mark-up, is assigned a state of machine, and symbol S_0 correspond to the initial state.

According to the algorithm flow chart path from the mark S_i to the mark S_j corresponds to a transition of the machine from state S_i to state S_j . For Mealy machine this way is represented as:

$$S_i x(S_i, S_j) y(S_i, S_j) S_j,$$

where $x(S_i, S_j)$ – conjunction of logical conditions X_k corresponding to conditional vertices on the path from S_i to S_j ; $y(S_i, S_j)$ – the contents of the vertex operator located on the algorithm flowchart between marks S_i and S_j .

If the path passes through the output “yes” (“1”, “true”) conditional tops of algorithm flow chart, the condition x_k is taken without inversion, and if through the output “no” (“0”, “false”) – then the condition x_k is included in conjunction with inversion. Ways in which some condition x_i is included in both direct and inverse form, are excluded from the description of the automaton. Paths which do not contain conditional vertices correspond to unconditional jump in the graph of the automaton, and the paths that do not contain the vertex operator – an empty operator.

For Moore machine path from mark-up S_i to mark-up S_j is signed as: $Six(S_i, S_j) S_j$

Input signal y_i in each state corresponds to content of operator vertices, which is marked as S_i .

5.6. Tools for work with remote laboratory

The hybrid Online Lab GOLDi (Grid of Online Lab Devices Ilmenau) was developed at the Department of Integrated Communication Systems at the Ilmenau University of Technology. It provides a tool set, supporting all design steps for complex control tasks (e.g., in the fields of control engineering, robotics, telecontrol engineering). Goal of the GOLDi system is to show state of the art ways and challenges of remote controlling and remote observation of real processes, dealing with the integrated and interactive usage of modern Internet and intranet technologies like HTML5, JavaScript, etc. It offers various features like visualization and animation, which allows observing and to test all the properties of the design. In connection with formal design techniques, simulation and prototyping are used to establish a foundation for the development of a reliable system design. Structure of GOLDi laboratory is shown on Fig. 5.7.

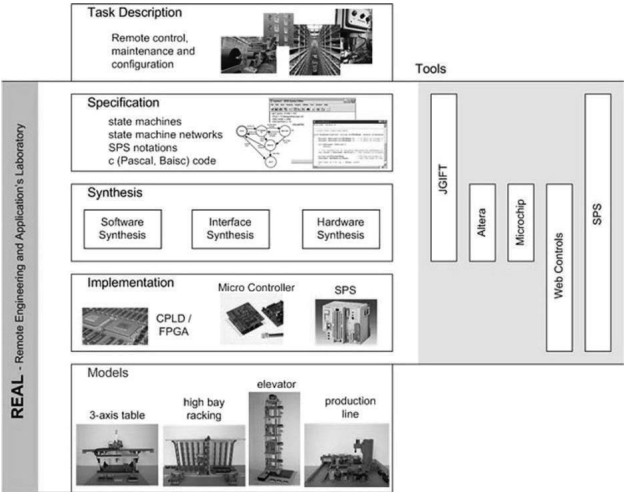


Figure 5.7 – Structure of GOLDi laboratory

The student can use Finite State Machines as a specification technique, based upon an automaton graph. By accessing the Web browser interface, he is able to enter his created control algorithm (in the form of Boolean equations), handle the laboratory procedure (initialize, start, stop and reset) and change environmental variables if necessary. The physical system will be controlled by the interpreter running inside the client (implemented e.g. as Java applet on the student's PC at home). In this case, only the input and output signals of the model will be transferred via Internet.

To use the lab GOLDi one must be connected to the Internet by means of PC with installed software and the browser Java-machine. To enter the laboratory one should use the following link <http://goldi-labs.net> or other link, which is showed by the lecturer.

On the site home page (Home) you can see the appointment (Welcome), a presentation of the architecture (Presentation Lab) the lab, a sequence of steps with it (How to use ESP), monitor running experiments (Visitor mode) and team (Team GOLDi) the creators of the laboratory. On the page Control Unit describes the available technologies for creating a controlling device: Manual Control, Digital Demo Board, PSOC, PLD, Finite State Machine and Microcontroller.

The page Physical system describes the physical and virtual model of the laboratory with a list of their inputs and outputs: 3Axis Portal, Elevator A (3 floors), Elevator B (4 floors), Elevator C (4 floors), Production Sell, Storage Warehouse, Water Level Control, Digital Demo Board and Maze.

On the page Start Experiment need to choose the instance lab to be an experiment. These instances set the names of the universities which are located in the laboratory GOLDi.

On the page of GIFT (Graphical Interactive Tool FSM) describes how to work with this software tool in the process of creating management applications. RUN Screen example for conveyor control system design is shown in Fig. 5.8.

The screen contains a box for entering project data and control of its loading and running, the window of physical model WEB image of control object, the window of virtual model. To study methods of control system design based on FSM models it is required to choose the language of communication, download the sample project, select "Only simulate" to initialize, run the example and observe the work of the virtual model.

Sample project can be saved on computer and viewed using standard Windows Notepad. In describing of the FSM-model following notation is used: x – input variable, y – output variable, z – state, # – logical operation OR, & – logical operation AND, ! – Logical operation NOT.

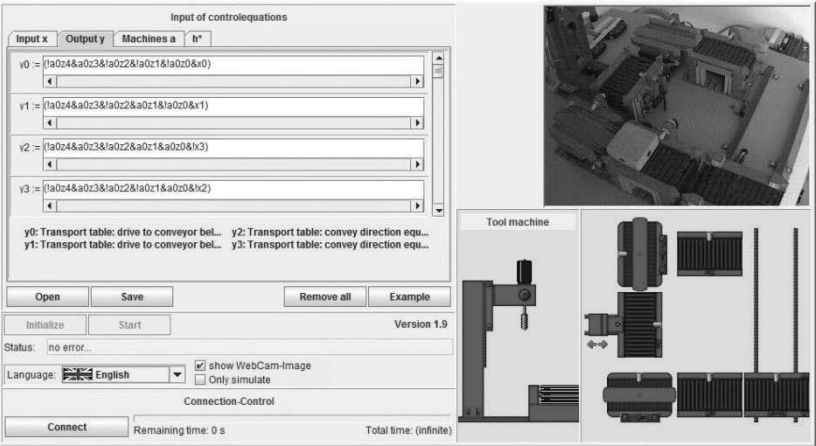
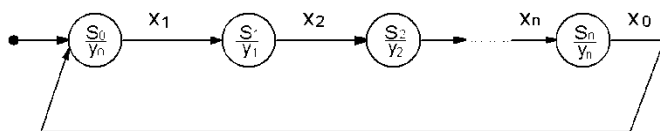


Figure 5.8 – Display structure for FSM design

6 TYPICAL CONTROL TASKS

6.1. FSM modeling on LD language

Let's consider the features of FSM – simulation of control systems in the LD language. A feature of the machine with a fixed change of state is the presence of only one path from the initial vertex to the final one. The graph of this automaton is shown in Fig. 6.1.



S_i – state ($i=0,n$); y_i – output into state S_i ; x_i – transition condition from state S_{i-1} to state S_i

Figure 6.1. – The graph of the FSM with a fixed changing of states

Transition of x_i from state S_{i-1} to S_i can be realized by the event, and (or) by the time. In the machine program, it is usually possible to allocate the rungs of initial machine state setting, state changes at a new input event and generating of output signals in each state. Options for programming machines with fixed changing states differing in the way of representing machine states in the program.

Fig. 6.2 presents a program, which implements the change of states by event using the instructions OTU, OTL, XIC and XIO. With usage of the software equivalent of counters – Instructions CTU, CTD it is also possible to construct a software implementation of the machine, such as shown in Fig. 6.3 for a graph of the FSM shown in Fig. 6.1.

We assume that the state of the automaton is encoded with the value of the accumulator meter Count1. The rung of machine setting to its original state at the first running of the program is not shown in Fig. 6.3. The conditional part of the first rung of the program in Fig. 6.3 contains n branches (the number of states of the automaton and counter instructions to CTU). Each branch defines the conditions of increasing counter status Count1, namely the current state of the counter and the corresponding input signal. Rung with RES instruction resets the battery meter to zero,

i.e. sets machine to zero. Subsequent rungs with EQU instruction and OTE form machine outputs in each state.

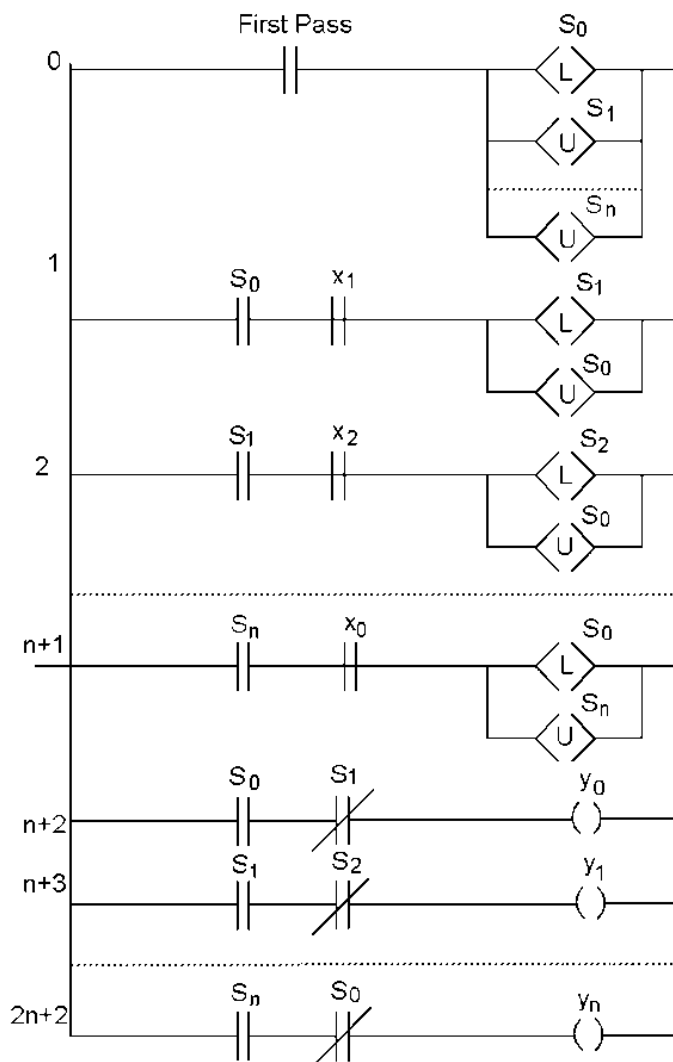


Figure 6.2. – The machine program – analogue of the digital assembly with storage in RS-triggers

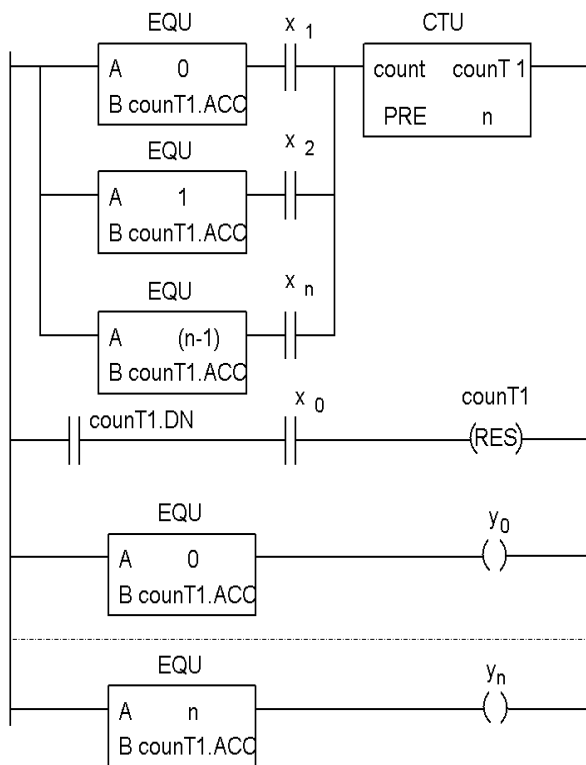


Figure 6.3 – FSM program with storage of states in accumulator of instruction counter

Unlike the machines with the fixed change of states machines with an arbitrary change have more than one path from the initial vertex to the final one, and therefore, in general, a vertex of a machine can be incident to more than one ingoing and (or) the outgoing arc. In addition, the type of machine becomes important – Mealy or Moore machine.

The process of programming the machine with an arbitrary change of states contains the same steps that the machine with fixed changing states: definition of memory elements for storage of the states; programming transitions from one state to another, including the transition to the initial state at the initiation of the machine; programming actions (outputs) in each state.

Thus, the machine comprising N conditions and M steps it is needed (N+M+1) rungs for programming. Number of rungs can be reduced if combined in one rung transitions (arcs) incident to one state (top) and actions in this state. Two approaches for such an association: “for all incoming” and “for all outgoing” from Si-th vertices of Mealy and Moore machines arcs are shown in Fig. 6.4 – 6.7.

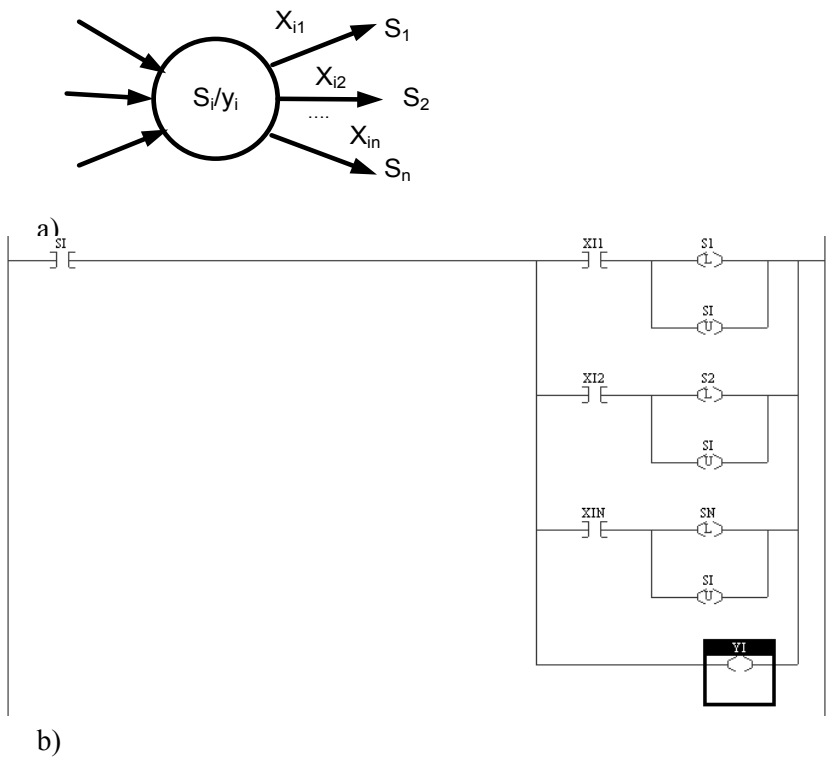
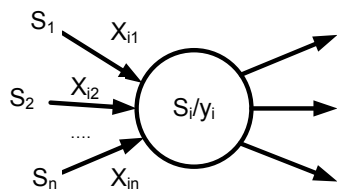
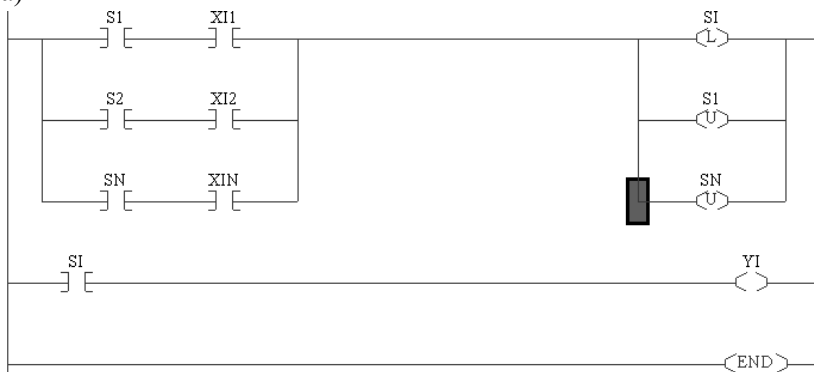


Figure 6.4 – Programming of the Moore machine according to the principle “all outgoing”: a – a fragment of the graph with outgoing arcs incident to state S_i ; b – part of the program

The program, which implements a Moore machine on the principle “all outgoing”, has N rung (the number of states of the automaton). Each rung (Fig. 6.4b) describes one outgoing transitions, Si-th state (Fig. 6.4a) and the output value of the machine in this state.

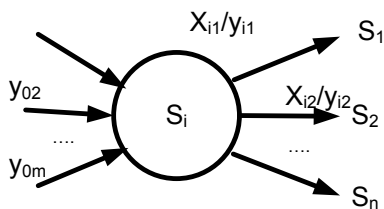


a)

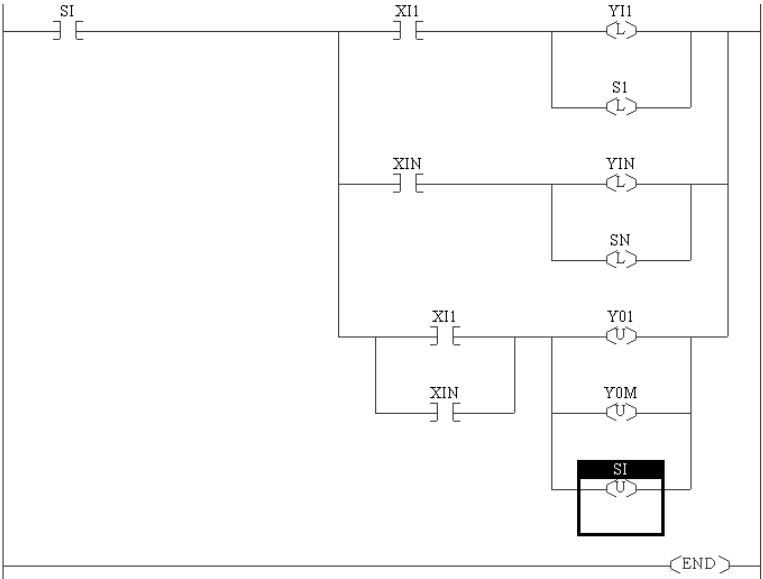


b)

Figure 6.5 – Programming of the Moore machine according to the principle “all ingoing” a – a fragment of the graph (ingoing arcs incident to state S_i); b – part of the program

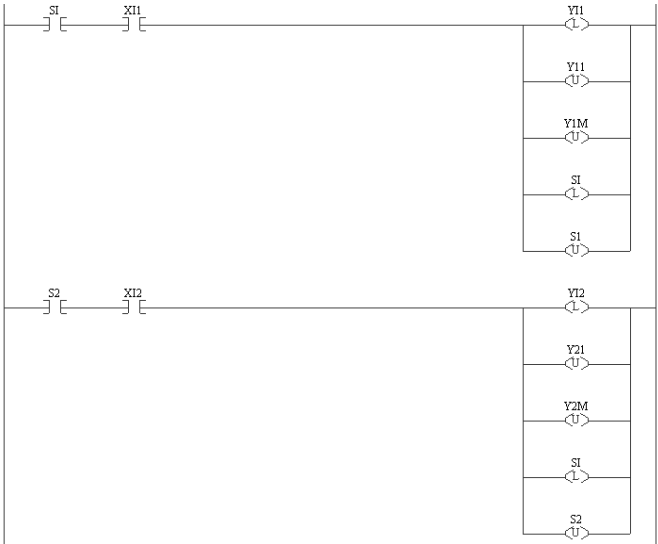


a)



b)

Figure 6.6. – Programming of the Mealy machine according to the principle “all outgoing”: a – a fragment of the graph with outgoing arcs incident to the state of S_i; b – part of the program



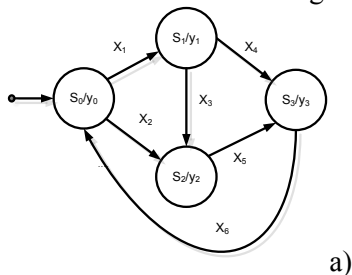


b)

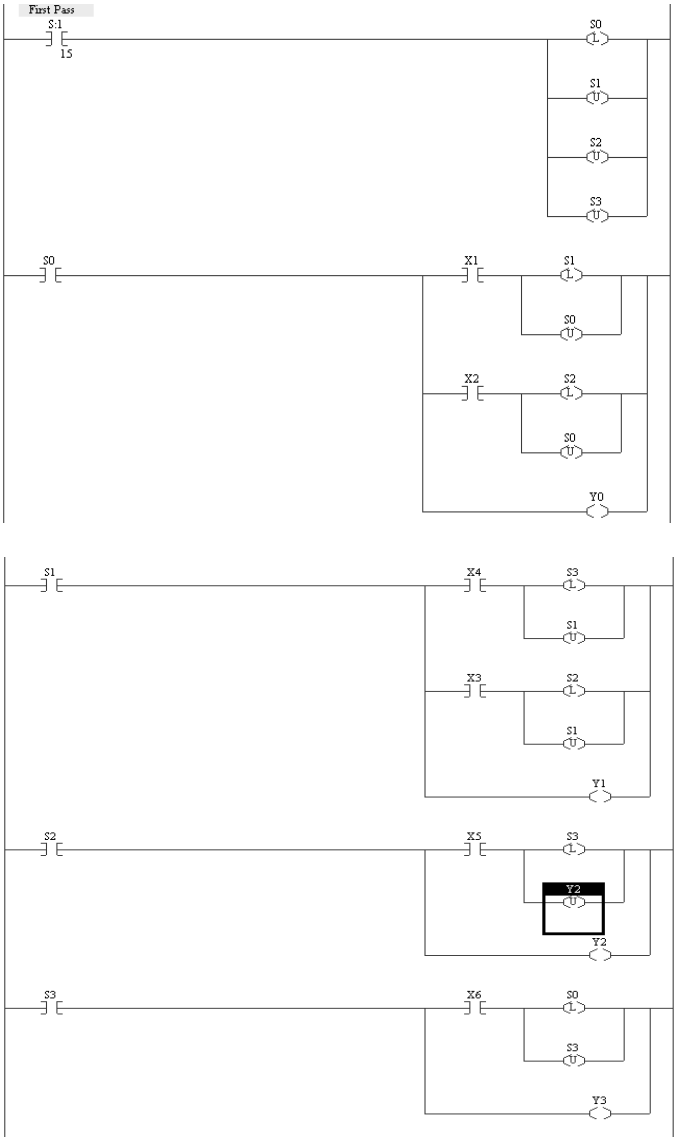
Figure 6.7. – Programming of the Mealy machine according to the principle of “all ingoing” a – a fragment of the graph with ingoing arcs incident to the state of S_i ; b – part of the program

As seen from Fig. 6.4 – 6.7, the program of the Mealy machine requires not only a new state, but also output values in the new state, and, therefore, the reset of the previous state and the previous output. For example, if the Mealy machine in Fig. 6.6 is at state S_i , then the activity may be one of the outputs y_{01} , y_{02} , y_{0m} and their activity must be reset at any transition from state S_i : in S_1 to x_{i1} ; in S_2 on x_{i2} ; ... in S_n by x_{in} . If Mealy machine in Fig. 6.7 is in a state S_i , then the activity of one set of outputs y_{i1} , ... y_{in} , and for each of the outputs requires reset activity of the prior state. For example, if machine transits in state S_i through input x_{i2} from state S_2 , then the out activity is set in y_{i2} , and the activities of outputs y_{21} , ... y_{2m} are reset.

As an example, Fig. 6.8 shows a graph and program of the Moore machine constructed according to the principle “all outgoing.”



a)



b)

Figure 6.8 – Example programming of the Moore machine according to the outgoing principle a – graph of the automaton; b – the program

Considered above machines are not the only ones. The other solutions discussing at design of machines with fixed changing states and sources [14,16,17] can be used.

6.2. Standard tasks of control

To the standard control tasks refer logic functions programming, effects generators programming.

As a result of programming one or more rungs of the program with XIC, XIO, OTE instructions is composed. Operands of rungs are bit tags of input variables and functions. If the truth of the logic function allows performing other output instructions, these instructions are additionally recorded into rung or used instead of OTE instruction rung.

Let the original formula of the logic function is given in Disjunctive normal form (DNF) where each term k_i (or conjunct) represents conjunction taken with or without negation functions of binary variables. Then the terms $k_1, \dots, k_n$ are presented in rung parallel to input branch of the program, and function like output branch.

The branch representing k_i term includes a chain of series-connected XIC instruction and (or) XIO. Operands of instructions are tags of binary variables functions. In the output branch the OTE instruction is placed, tag of function is its operand. It is allowed to put only one XIC (XIO) instruction in the k_i term branch the operand of which is tag of intermediate variable representing the truth of the k_i term. The intermediate variable calculation is done in a separate rung. Fig. 6.9 shows the example of logic

function programming: $y = \overline{(x_1 \wedge x_2)} \vee (x_3 \wedge x_4)$ using intermediate variables y_1 and y_2 .

Let the logic function given in conjunctive normal form (CNF), where each term D_i is a disjunction of binary variables function taken with or without negations. Then, the terms $D_1 \dots D_n$ are presented in program rung like sequence of input branches, and function as an output branch. Branch representing D_i term contains parallel branches with XIC and (or) XIO instructions which operands are binary variables functions tags. OTE instruction is placed in the output branch. Logic function

programming example: $y = (\bar{x}_1 \vee x_2) \wedge (x_3 \vee \bar{x}_4)$ is shown in Fig. 6.10.

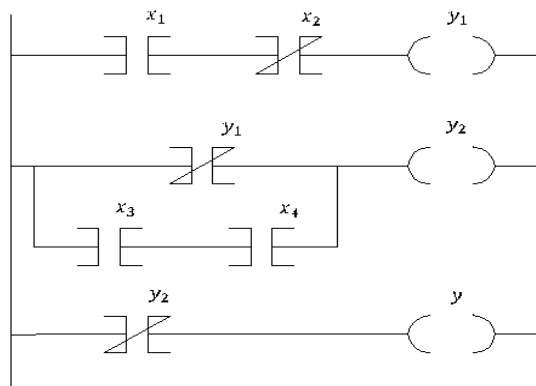


Figure 6.9 – The logic function in programming using intermediate variables

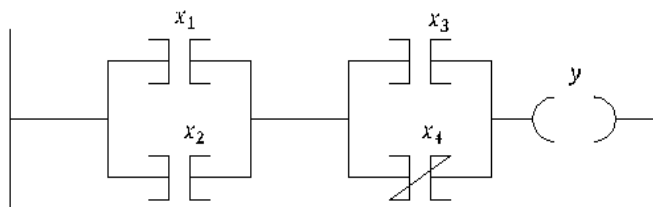


Figure 6.10 – The logic function shown in CNF programming

A stimulus generator (SG) means OA output, the output signal level of which changes during the active OA. The SG outputs are connected to inputs of control object. The generated signals change the state of the control object that is fixed with analysis of the CU outputs. In addition, the SG outputs can be connected to the input of OA, and simulate the disturbances changes or output signals of the control object. In Fig.6.11 the program of infinite sequence generator of rectangular pulses with a duty factor 0.5 is shown.

In the rung of 0, with the first run of the program, on the generator output Y initial values of the code bits X are set. Rung 1 generates a time interval $\text{TIMEBASE} * \text{PRE}$, during which the Y outputs current values are saved. We call this interval phase 1. In rung 2, these values are reversed for orders Yi for which the mask order value Mi is one. In the next program scan, in rung 1 count of the phase 2 of the time sequence

begins. After the completion of phase 2, the orders outputs Y values are reversed again.

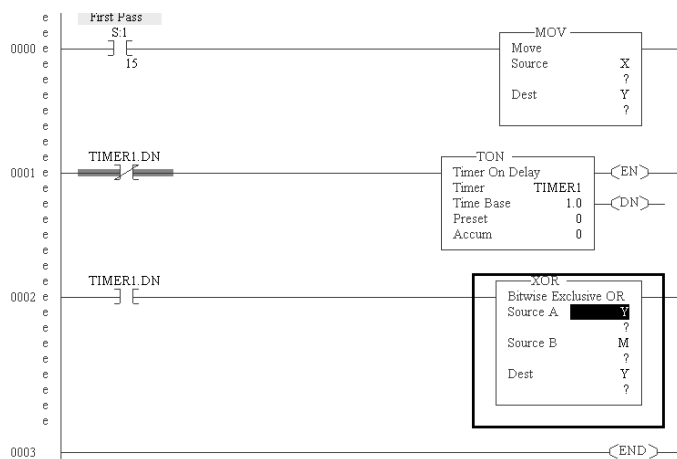


Figure 6.11 – Sequence generator pulse program with duty factor 0.5.

Fig.6.12 is a program of the sawtooth signal generator (linearly increasing) forms. In this program, the counter accumulator instruction COUNT1.ACC varies from 0 to COUNT1.PRE, increased by one after a time interval TIME BASE * PRE given with timer instructions operand TIMER1. Then, the accumulator meter is reset, and the cycle repeats forming a sawtooth voltage.

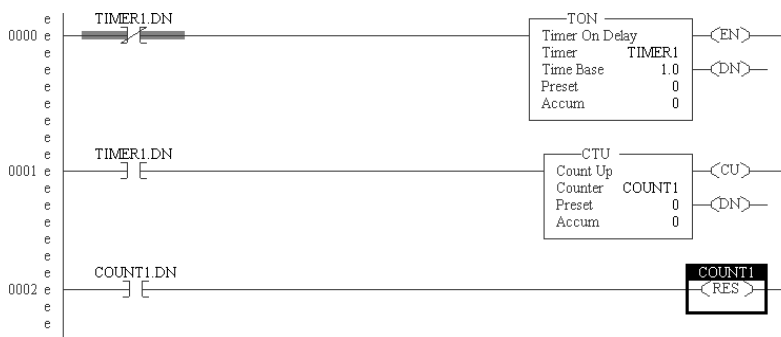


Figure 6.12. – Sawtooth (linearly increasing) signal forms generator program.

Fig.6.13 shows the triangular shape signal generator program with a maximum (N1) and minimum (N2) values of the output signal.

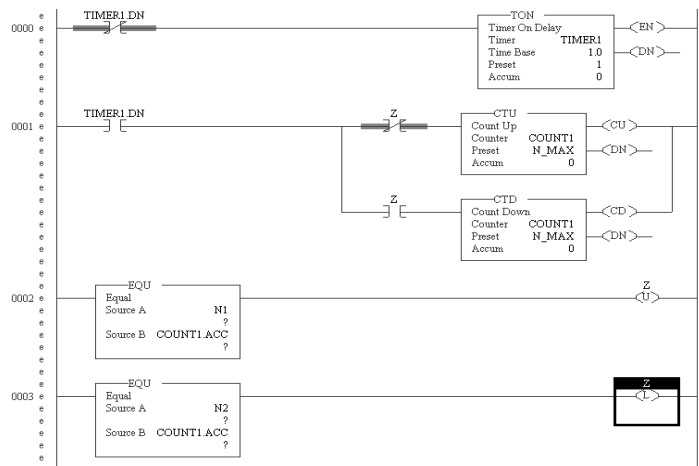


Figure 6.13. – Program of generator of triangular shape signals.

Fig. 6.14 illustrates method of constructing a trigonometric function generator. It is required to form a piecewise linear approximation of a sinusoidal signal $Y=A\cdot\sin(\omega t)$. The initial data for setting the operands instruction in the generator program is a function period T and the approximated function linear sections number n of over its period. Whence $\omega=2\pi/t$ and duration of the basic block is $\Delta t=T/n$. The duration Δt is set up in the program as the product of the values of the operands `TIME BASE` and `PRE` instructions of timer `TON`.

If an arbitrary function $f(t)$ is set in tabular form, than for the function definitions generator construction it is convenient to use the `SQO` sequencer instruction, as it is shown in Fig. 6.15. In this program, the timer `TIMER1` sets the duration of the approximated function lineal department interval. At the interval end on `Y` output comes new function value $f(t)$ taken from the next line of the table `TABL`.

After the whole table “reading” the reset structure `SQO` instruction is done and new formation period of a function $f(t)$ starts.

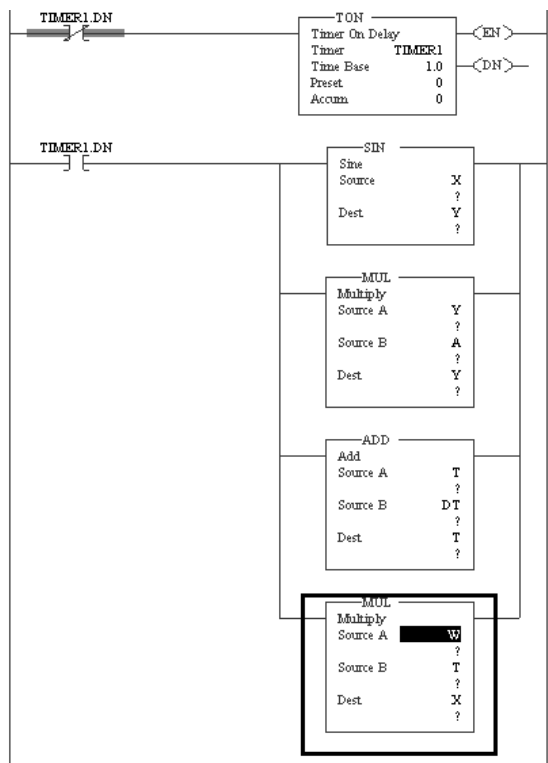


Figure 6.14 – Generator of sinusoidal functions

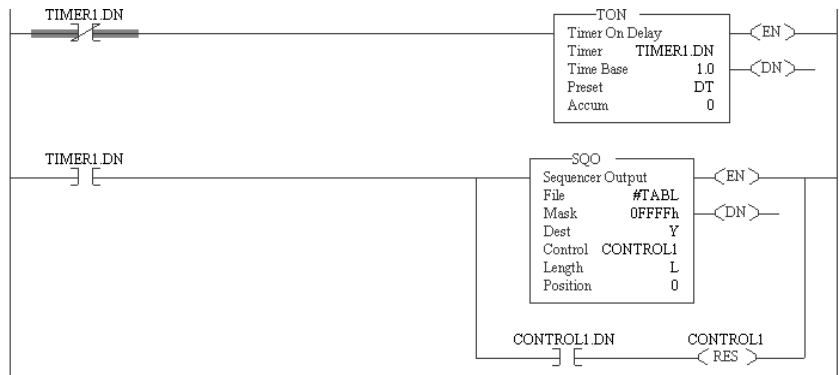


Figure 6.15. – Tabular computation of arbitrary function

6.3. Man Machine Interface Software

Characteristic features of modern controller control systems are multilevel structure, distribution, human involvement in the control process, control algorithms software implementation. Such systems exist in the areas of generation, transmission and distribution of electric energy, industrial production, recovering, transportation and distribution of oil and gas, control of transport, public establishment, telecommunications etc. As the rule, the problem of MMI (MMI – Man Machine Interface or HMI – Human Machine Interface), SCADA (SCADA – Supervisory Control and Data Acquisition) is to be solved at the top level of control system. The main structural components of SCADA system is RTU (RTU – Remote Terminal Unit), dispatch control point (MTU – Master Terminal Unit, MS – Master Station) and CS (CS – Communication system). The main element of MTU software is the user application, which is created and performed in visualization software packages environment.

For evaluation and selection of SCADA-system, it is important to know characteristics that are divided in four categories: basic properties, the possibility of the development environment, the possibility of system configuration, performance support / operating system. Openness, support controllers, registration, archiving, work with trends, alarm system, opportunities HMI, WEB technology, supporting of statements, operating system, scale, redundancy, access protection, cost etc. belongs to basic properties.

SCADA-system is considered open if it describes the data formats and interface for connecting independently developed components such as custom software modules, drivers, input-output, components, technologies developed by DDE (Dynamical Data Exchange), OLE (Object Linking and Embedding), OPC (OLE for Process Control), COM / DCOM (Component Object Model / Distributed COM); data exchange with real-time databases of different levels and configuration parameters ASCII – export (import) of the project manual is available.

Data registration is the process when the data selection occurs due to the range and time of its entering, structuring and storage organization on a computer for further processing. Archiving is a way to reduce the amount of store data memory.

Alarms are signaling subsystem messages. Alarm configuration means differs in priorities assignment possibilities, grouping, online filtering, masking, frame messages to the operator, etc.

Trends are the means of data visualization usually depended on time, rarely from another variable (called XY – dependence). Trends are divided into real-time and historical. Number of curves in the trend, the possibility of rapid change in the scale window, forming a trend using ActiveX object etc. belongs to characteristics of these features.

Features of HMI SCADA-systems are characterized by the presence of object-oriented graphics editor, a library of standard graphical symbols, the technique of “drug-and-drop”, separate screen navigation tools, animation types of graphic objects, multimedia, etc.

WEB technology used in SCADA systems allows data monitoring and control of object from a remote computer that is connected to the control system via computer network.

First of all, support of reports is an opportunity of printing and archiving of reports that, in general, contain raw data received from the controlling object, their processing results, event recording, alarm and control system activities. Scalability assumes the possible of existence of multiple implementations of one SCADA-system, which differ in the number of tags in the real time database or in number of assemblies in the control system.

Reservation is a way to ensure the operability of control system under conditions of possible failures of its elements. As for SCADA systems, this is units and their components (for example, CPU, power supply, magnetic disc store, etc.), data exchange channels. Access Protection is a system of measures to limit unauthorized access to data and configuring tools of user application. These measures include passwords and access levels to the data and some types of activities. Other properties one can note is the presence of statistical control options, support “recipes”, tasks planning, fuzzy logic tools and automatic event processing.

Development of user application is carried out in MMI package with using the so-called project manager from which problem-oriented editors and already made part of the project are called. Screen, selected by designer, opens in the graphics editor environment. Typical problem-oriented editors are communication channel editors with controllers, nodes in networks, tag editor, events, user rights, properties of data logging subsystems etc. The design results are graphic, logic and tag

tasks control models. Graphical model is a graphic screen system and navigation tools between them. Graphic screen contains graphical objects that have behavior, i.e. they change their shape, screen position, color and other attributes with the change of control tag value. Logic model links certain system conditions with specific actions. Tagged models are described for each tag that is used in the control process, namely, its type, source, range of values, and others.

6.4. Behavioral synthesis and scripting of visualization problem

In order to broaden the functional abilities of SCADA package commands, macros and scripting are used. Commands of the package are subprograms, which are used as a reaction element for a particular event, appearing at application launching or a part of a macros or program. Actions, proposed by functional menu of SCADA package are realized in terms of commands.

Macros is lists of commands being automatically launched in the order of their position. Command “execute macros <macros name>” allows including a macros to other macros.

Scripting consists of object model of an application and a software language. The most widespread software languages are Microsoft Visual Basic for Application (VBA) and C. Scripting example: scenarios development (VBA or C subprograms, performing conditional branching); binding of visualization application data to other applications, such as Microsoft Access or Microsoft SQL Server; control of visualization application from VBA subprogram by calling the commands to visualization application.

6.5. Self-evaluation in form of quiz, exercises

1. Structure of industrial automation systems with the use of controllers.
2. The principle of a programmable controller operation.
3. Overview of languages according to the standard IEC 61131-3.
4. Ladder Diagram Language.
5. Structural diagram of an industrial controller.
6. The programming software: the purpose, the basic steps for creating projects.

7. The bit instructions.
8. The instructions of timers.
9. The instructions of counters.
10. The compare instructions.
11. The mathematical instructions.
12. Instructions for the control program.
13. Sequencer, shift, PID instructions.
14. Means of control tasks formalization.
15. Representation of the control algorithm in the form of a logic diagram
16. Representation of the control algorithm in the form of a finite state machine.
17. Tools for work with remote laboratory
18. FSM modeling on LD language for automaton with a fixed states changing.
19. FSM modeling on LD language for automaton with a random states changing.
20. Logic function programming.
21. Programming of exposure generators of rectangular, saw tooth, triangular, sinusoidal and freeform.
22. The purpose and characteristics of packages MMI and SCADA – systems.
23. Editors of the project manager of MMI package.
24. The structure of the virtual screen control.
25. Implementation of the behavior in the environment of MMI using the event editor.
26. Implementation of the behavior in the environment of MMI using the scripting.
27. Practical task “Control system of a train station”.
28. Practical task “Traffic light”.
29. Practical task “Water level control in the tank”.
30. Practical task “Formation of a sequence of pulses”.
31. Practical task “Logic function”.

6.6. Self-evaluation in multiple-choice form

1. Which languages is not a standard programming language of industrial controllers?

1 – LD; 2 – UML; 3 – SFC; 4 – ST; 5 – FBD; 6 – all of the listed languages are not the standard controller programming languages.

2. The program in LD language visually resembles:

1 – listing of assembler program; 2 – graph of automaton; 3 – circuitry of the digital node; 4 – circuitry of relay automation; 5 – flowchart; 6 – none of the above.

3. Which one of the modules is not a part of industrial controller?

1.- processor; 2 – output module; 3 – input module; 4 – network module; 5 – hard drive; 6 – all of the listed modules are included into industrial controller.

4. Which software product is not required at development of the controller control program?

1 – RSLinx; 2 – RSLogix; 3 – RSEmulate; 4 – RSView; 5 – Microsoft Word; 6 – all products are required.

5. Which of the listed LD programming language instruction is not input (conditional) one?

1. CTD; 2 – EQU; 3 – GEQ; 4 – LEQ; 5 – NEQ; 6 – all instructions are input ones.

6 Which of the listed LD programming language instruction is not output (executive) one?

1. CTU; 2 – ADD; 3 – TON; 4 – MOV; 5 – LIM; 6 – all instructions are output ones.

7. What address in the controller memory has the third channel of 16-channel input module, placed in the second slot:

1 – O:3/2; 2 – I:2.3; 3 – I:3.0/2; 4 – I:2.0/3; 5 – I:2:0/3; 6 – other answer.

8. Which components are not included in the description of the finite state machine?

1 – set of inputs; 2 – set of outputs; 3 – set of states; 4 – function of outputs; 5 – function of transfers; 6 – all components are included in description.

9. For what values of Boolean variables a, b result of expression $!(a+b)$ equals to 1:

1 – a=0, b=0; 2 – a=1, b=0; 3 – a=0, b=1; 4 – a=1, b=1; 5 – other answer.

10. For what values of Boolean variables a, b result of expression $!(ab)$ equals to 0:

1 a=0 b=0; 2 a=1 b=0; 3 a=0 b=1; 4 a=1 b=1; 5 – other answer.

11. What is not a step of the controller program scan?

1 – update of inputs; 2 – scan of user logic; 3 – update of outputs; 4 – network traffic; 5 – self-diagnostic; 6 – all of the listed are the steps.

12. What of the listed is not control type of the graphical object of the virtual control screen?

1 – rotation control; 2 – touch control; 3- filling control 4 – horizontal slider; 5 – object width control; 6 – all are control types.

13. At what graphic object control object position on the screen is set with the tag value?

1 – rotation control; 2 – touch control; 3- filling control 4 – horizontal slider; 5 – object width control; 6 – at none.

14. Which of the listed components is not included in the HMI application?

1 – alarm; 2 – graphical control screen; 3 – program in LD language; 4 – event; 5 – trend; 6 – all are included.

15. Which of the listed is not necessary to define when configuring the memory tag?

1 – tag name; 2 – tag type; 3 – controller network number; 4 – initial value of the tag; 5 – an expression for calculating of the tag value; 6 – all are needed to be defined.

16. With the help of what element it is possible to implement FSM-based control in the HMI application?

1 – subsystem of virtual control screens; 2 – subsystem of events; 3 – alarm subsystem; 4 – data archiving subsystem; 5 – all elements are suitable; 6 – none of the listed elements.

Numbers of correct answers are presented in chapter 6.8 and estimated result – in Table 6.1.

Table 6.1 – Estimation of test results

Quantity of correct answers	15-16	13-14	10-12	Less than 10
Mark	Excellent	Good	Satisfactory	Poor

6.7. Laboratory Work

It is required to complete two laboratory works, the purpose of which is the study of implementation methods for control algorithms using FSM and design practice. “Elevator” and “Conveyor” virtual models in **GOLDI** remote laboratory are to be used in the laboratory work performance.

Task for the laboratory work:

1. Study the sensors and executive mechanisms content of the simulated model.
2. Study the simulated model functions realized in the built-in example. To design a system FSM model on the basis of example data.
3. Complement the FSM model by functions, realized in the example. Proposed functions are to be agreed with an assistant of lecturer by an e-mail.
4. Configure a controller; develop a controller connection pattern to the designed system equipment.
5. Propose a modified FSM model realization in terms of LD language.

The laboratory work report should contain:

1. Purpose, objectives and conclusions of the performed task.
2. The description of FSM model, including the inputs, outputs and states names, logical equations for outputs and transitions, verbal description for realized functions and the graph of FSM example.
3. PLC specification, its connection pattern to the designed system equipment, programs realizing the control algorithm in terms of LD language.

6.8. Practical Work

The topic for practical works: “Logical functions programming in terms of LD language”, “Conceptual machine programming in terms of LD language” and “MMI project configuration”.

The works are performed by individual tasks, obtained from the assistant by the e-mail. The practical work reports are also e-mailed to assistant. The work is accepted after assistant’s or lecturer’s approval.

In the work “Logical function programming in terms of LD language” it is necessary to develop a truth table, logical elements scheme, logical program inputs and outputs connection pattern to PLC, the table of inputs and outputs addresses, control flow diagram and LD language program on the basis of initial logical equation.

In the work “Conceptual machine programming in terms of LD” it is necessary to develop FSM model, input and output connection pattern of a conceptual machine to PLC and LD language program on the basis of initial tabular description.

In the work “MMI project configuration” it is necessary to develop FSM model of visual application in term of a control task graph, submit a tag database for a control task with their characteristics, graphical representation of a task in the form of visual command screens with graphical objects and the types of their control indication, list of events of control algorithm with indication of condition and action expressions of an event.

Answers to test in chapter 6.6:

Question	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Answer	2	4	5	4	1	5	4	6	1	4	6	6	4	3	3	2

LITERATURE FOR PART 2

1. Rassalsky A. N. Система мониторинга и управления силовых трансформаторов [Control and monitoring system of power transformers]. – Kharkiv, Electrical Engineering & Electromechanics, 2005, № 2, p.54–58.
2. Mishel’Zh., LorzhоK., Jesp’оB. Программируемые контроллеры. Пер. с франц. А. П. Сизова [Programmable controllers] – Moscow, Mashinostroenie, 1986 – 176 p.

3. Parr E. A. Programmable Controllers. An engineer's guite. Third edition. Oxford: Newnes, 2003, 429 p.
4. Lewis R. W. Programming industrial control systems using IEC1131-3/Revised ed. The Institution of Electrical Engineers. London, United Kindom, 1998, 344 p.
5. John K. – H., Tiegelkamp, M. IEC 61131-3: Programming Industrial Automation Systems. Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-Making Tools. – Shpringer. 2001 VI, 376 pp., 139 figs.
6. Petrov I. V. Программируемые контроллеры. Стандартные языки и инструменты [Programmable Controllers. Standard languages and tools] – Moscow, SOLON-Press, 2003, – 256p.
7. Olson G., Piani D. Цифровые системы автоматизации и управления [Digital automation and control systems]. – Saint Petersburg, Nevskij Dialekt, 2001. – 557p.
8. Rassalsky A. N., Polyakov M. A. Опыт обучения промышленной информатике в учебно-научной лаборатории АСУ ТП Запорожского технического университета [Teaching experience of industrial informatics in ACS TP educational and scientific laboratory at Zaporozhye Technical University]. – “New information technologies in electrical engineering education”: collected papers of the 5th international methodological conference, Astrakhan, CNTJeP, 2000, с.199–201.
9. Polyakov M. A., Rassalsky A. N. Перспективы применения средств промышленной автоматизации для целей обучения [Application prospects industrial automation facilities for educational needs]. – “New information technologies in regional infrastructure and education”: collected papers of the 4th international methodological conference, Astrakhan, CNTJeP, 2001, p.123-124.
10. Rassalsky A. N., Polyakov M. A. Опыт обучения контроллерным системам управления в учебно-научной лаборатории АСУ ТП Запорожского технического университета [Teaching experience of controller control systems in ACS TP educational and scientific laboratory at Zaporozhye Technical University]. – Visnyk of National Technical University of Ukraine “Kyiv Polytechnic Institute”, Kharkiv, 2001, № 16, p.141.

11. Rockwell automation. User manuals/ Access: [www/URL: http://literature.rockwellautomation.com/idc/groups/public/documents/webassets/browse_category.hcst](http://literature.rockwellautomation.com/idc/groups/public/documents/webassets/browse_category.hcst) (EN).
12. Modules of analog input/output ControlLogix. Allen Bradley company. Publication 1756–6.5.9. – November 1998.
13. Stashin V. V. and others. Проектирование цифровых устройств на однокристальных микроконтроллерах [Design of digital devices on single-chip microcontrollers]. – Moscow, Jenergoatomizdat, 1990, 224 p.
14. Goma H., Designing Concurrent, Distributed, and Real-Time Application with UML. 1st Edition, Addison-Wesley, 2000, 816 p.
15. Buch G., Rambo D., Dzhekbson A., Jazyk UML. – Язык UML. Руководство пользователя. Пер. с англ. [UML language. User Guide. Translation from English] – Moscow, DMK, 2001, 432 p.
16. Karpov Ju. G. Теория автоматов [Automata theory]. – Saint Petersburg, Piter, 2002. – 224 p.
17. Hugh Jack Automating Manufacturing Systems with PLCs (version 5.0, may 4, 2007).
18. Polyakov M. A., Larionova T. Ju. Теоретико-множественная модель функциональной структуры удаленной лаборатории для обучения проектированию систем управления [Set-theoretic model of the functional structure of a remote laboratory for teaching of the control systems design]. – Dnipropetrovsk, “System technologies”, 2015, 3 (98), p.48-56.

PART 3

QUALITY OF INFORMATIONAL SYSTEMS

Galyna Tabunshchyk, Tetiana Kapliienko

INTRODUCTION TO PART 3

Quality means that a product satisfies the demands of its specifications. It also means achieving a high level of customer satisfaction with the product.

Quality management procedure consists from:

- quality assurance – establishing organizational quality standards and procedures;
- quality planning – selecting and modifying applicable quality standards and procedures for a particular project;
- quality control – ensuring quality standards and procedures are followed by development team.

Verification and Validation activities focus on both the quality of the the software product and of the engineering process.

Concurrent informational systems are compound systems which integrate software and hardware in one embedded system but existed approaches don't consider complex verification of hardware and software for them.

For complex automatization of the informational system it's important to have formalized model for objects included in the system as well as the description of the processes of data transmission, so the chapter 7 consists of the description of the existing approaches to informational systems verification and developed by the authors the model of verification as well.

One of the most effective instruments of verification is testing, which allow to provide verification of all modified parts of the developing system. Chapter 8 depicts software and hardware techniques for testing, including modified method for regression testing, which allow

forming automatically connected components of the system within the repeated testing of the informational system.

Chapter 9 contains the explanation of teaching approaches with the usage of the remote laboratories. Possibilities of online lab GOLDi developed by Integrated Communication Systems Group at the Ilmenau University of Technology are described. Examples of model-based testing and functional testing with the usage of the modes of grid remote online laboratories and tasks for further development are also suggested by the authors.

Authors want to thank Karsten Henke, Dieter-Heintz Wuttke and all Goldi team for provided support during investigation of the GOLDi facilities.

7 BASICS OF EMBEDDED SYSTEMS VERIFICATION

Verification and Validation activities focus on both as quality of the software product as quality of the engineering process.

Validation is the process of evaluating a software system during, or at the end of, the development cycle in order to determine whether it satisfies specified requirements [10].

Verification is the process of evaluating a software system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase [10].

7.1. Embedded Systems verification methods

Verification methods can be classified as static verification method, formal verification, dynamic verification methods, synthetic methods (fig. 7.1).

Among expertises the most popular are management review, technical review, walkthrough, inspection and audit. From the beginning of 1990s scenario based software architecture evaluation methods have been rapidly developing, which are not considered as “typical” expertises.

Expertise differs from all other verification methods by type of how it's conducting. It uses only real artifacts of life cycle and not models like in formal methods or results like in dynamic methods. It can be applied for any Software Characteristics and for any artifacts of all life cycle.



Figure 7.1 – Verification methods

It can detect a wide variety of errors, what is mostly important it is provide it at the design stage for the defined artifact, thereby minimizing lifetime of a defect and its impact on the quality of original artifacts. At the same time expertise can be automated and requires the active participation of the people.

Efficiency of the examination essentially depends on the experience and motivation of the participants of the process and of ensuring proper interoperability between participants. This imposes additional restrictions on the distribution of resources in the project and can lead to conflicts between developers if project management pays little attention to the communicative aspects of the examination.

Software reviews and audits are a basic part of the ongoing evaluation of software products as they pass along the software development life cycle.

For different goals there are used different types of expertise.

- *technical review* – according to IEEE Std 1028-2008 [20] it is « A process or meeting where a software product is presented for comments or approval». The goal of technical review is to evaluate conformance to specification and standards. It is performed by technical leadership and peers.
- *walkthrough* – it is a type of static analysis where participants are asking questions and make comments, search for anomalies, improve

the product, consider alternative implementations, evaluate conformance to standards or specifications;

- *software inspection* – a visual examination of a software product to detect and identify anomalies including errors and deviations from standards and specifications. It is proceeded in the way of peer examined, led by impartial and trained facilitators For inspections determination of remedial or investigative action for an anomaly is mandatory. Solutions are not determined during inspection meeting. Management position ‘over’ participants shall not participate. During inspection strongly recommended to collect and analysis data;
- *audit* – is a type of software review in which one or more auditors who are not members of the software development organization conduct «An independent examination of a software product, software process, or set of software processes to assess compliance with specifications, standards, contractual agreements, or other criteria” [10].

Static analysis is used for checking formalized rules for Informational Systems artifact construction as well as for bugs searching according to defined patterns.

This type of analysis easily done through automation and can be totally done by software (SW), therefore sometimes its need human interfere for instance for applying internal coding standards. But it can be used only for code or for specified IS artifacted and can find only small number of bugs. The disadvantage of these methods or great redundancy of strict analysis methods or high probability to miss error when error messages approach is used. Automation verification techniques which use static analysis approach are widely use as they don’t require special prerequisites and very convenient. Most of the techniques which prove their effectiveness usually become parts of the compilers and transfer to the semantic rules for software languages.

Formal verification is the process of checking whether a design satisfies some requirements (properties). Formal verification is conducted with the the special approaches such as theorem proving, model checking, abstract interpretation.

Formal methods are applicable only to those properties that are expressed formally within a certain mathematical model, and to those artifacts for which we can construct an adequate formal model. Accordingly, for the implementation of such methods in the project

must be put considerable effort on building formal models. Moreover, to construct such models and conduct analysis can only specialists. The process of formal models construction is difficult to automate, it always need a human. Analysis of their properties can be automated, and now there are existing tools which are able to analyze formal models of industrial complexity, but to effectively use them it is often also require a very specific set of skills and knowledge (in specific sections of mathematical logic and algebra). However, in some fields where the consequences of error in the system can be extremely expensive, formal verification methods are widely used. They are able to detect complex errors almost not detectable by examination or testing. In addition, formalization of requirements and design solutions possible only with their deep understanding and therefore should be done a thorough analysis of the artifacts, which often requires collaboration of experts in formal methods and analytic of system domain.

In practice, formal verification techniques more often used for hardware than for software testing. This is due to more higher cost of errors in hardware, greater uniformity of its structure, the wider use of multiple project information and greater familiarity strict limits and precise descriptions for engineers.

Property-based testing (PBT) allows programmers to capture informal conjectures about their code as executable specifications and to thoroughly test these conjectures on a large number of inputs, usually randomly generated. When a counterexample is found it is shrunk to a minimal one, which is displayed to the programmer.

Executable models characterized by the fact that they can be realized in a certain way to trace changes in properties of simulated software. Each performed model is essentially a program for some well-defined virtual machine. All executable models can be considered as an extension and generalization models based on finite automata. Intermediate models have both features as an algebraic logical and as executable models. Some of the above examples may be a number of properties attributable to both classes.

Model checking is an automatic technique for verifying finit state concurrent systems. **Software model checking** is the algorithmic analysis of programs to prove properties of their executions. It traces its roots to logic and theorem proving, both to provide the conceptual framework in which to formalize the fundamental questions and to provide algorithmic

procedures for the analysis of logical questions.

The main goal of software model checking is to prove properties of program computations. Examples of properties are simple assertions, that state that a predicate on program variables holds whenever the computation reaches a particular control location (e.g., “the variable x is positive whenever control reaches”), or global invariants, that state that certain predicates hold on every reachable state (e.g., “each array access is within bounds”), or termination properties (e.g., “the program terminates on all inputs”). Broadly, properties are classified as safety and liveness. Informally, safety properties stipulate that “bad things” do not happen during program computations, and liveness properties stipulate that “good things” do eventually happen.

Persistence properties are special types of liveness properties that assert that from some moment on a certain state condition F holds continuously.

For many systems, termination can be proved only under certain assumptions about the non-deterministic choices made during program execution. The programmer often models certain aspects of the system through non-deterministic choice, with an implicit assumption that such choices are resolved in a “fair” manner.

The standard way to rule out certain undesirable infinite behaviors from the scope of verification is through *fairness conditions*. Fair termination is the property that a program terminates on all runs that satisfy the fairness requirements. Just as safety properties can be reduced to reachability problems, liveness properties can be reduced to checking termination under fairness requirements. The techniques for proving termination generalize to fair termination by taking a product of the program with an automaton on infinite words modeling the fairness condition, and checking for well-foundedness only for final states of the automaton.

Methods and tools for the **consistency checking**. When checking the consistency, there is conducting the analysis of the correspondence between the two models, one of which simulates the audited artifact, usually it is a real project or the system (its components), and the second – tested properties. In this case checked properties are the requirements for the behavior of the system or its components, which are presented in a generalized automata (transitions system, Petri nets and others.). All scenarios for which are declared valid. In this case it is usually checked

that all the possible scenarios for the implementation of behavior are also possible in the specification. Sometimes their equivalence is set, this mean that additionally is checked that all behavior specification scenarios are implemented. Most of the methods and tools for consistency check are used for this testing, and therefore relate to synthetic methods of verification – model-based testing.

Dynamic verification is performed during the execution of software, and dynamically checks its behaviour; it is commonly known as the test phase.

To use dynamic methods need to have a working system, or at least some of its components, or their prototypes, and they usually are not used in the early stages of development. With these methods can control the characteristics of IS in its real environment where other approaches can not be used.

Dynamic verification techniques can detect only errors that occur during its operation. The use of dynamic verification methods typically require additional qualifications – skills to create tests, skills for developing test systems for test performing or possibilities to monitor system that allows you to control certain behaviors of the IS. However, testing or monitoring systems can be made once and used repeatedly for broad classes of IS, only the tests should be prepared afresh for each IS under test. At the same time preparing tests for the early stages of IS can detect many defects in the description of the requirements and design documents – in fact, developers have to test in the ordinary course of business to carry out an examination of artifacts that are the basis for the tests. As testing is very time-consuming process, so commonly there are use not too reliable, but cheap enough tools, such as probability testing in which test data is generated randomly or testing on the basis of simple scenarios that examine only the most simple situation.

Dynamic methods that use formal elements create separate field of verification – model-based testing, model driven testing and monitoring of the formal properties (runtime verification and passive testing). Several tools of test construction essentially use a formalization of some software properties, and static code analysis. The general idea of these methods can be described as next – to combine the benefits of basic approaches to verification to balance their shortcomings.

In **Model-Driven Development** (MDD), models are the basic blocks of software development, and model transformations are used to map

between models conforming to different metamodels. Given their key role in MDD, verification of transformations is becoming of increasing interest to researcher.

Model based testing is used to build formal models test requirements for IS. As a criteria for completeness of testing, as other criteria are based on the information which is contained in these models. The resulting tests are usually weakly associated with specific features of the code test system, but contain a representative sample of the situations in terms of the original model.

Currently testing methods based on models using these types of models and techniques as methods for checking consistency of machines and systems conversions. These methods belong to one of three types, depending on the model used.

Methods of testing based on finite automata developed most deeply known their exact limits and guarantees the completeness checks performed. Methods that use advanced machines, reduce them to normal, but used a more detailed coverage criteria based on the use of data in advanced machines.

Transition systems are often used in computer science as models to describe the behavior of systems. They are basically directed graphs where nodes represent states, and edges model transitions, i.e., state changes. A state describes some information about a system at a certain moment of its behavior.

System transitions. Such methods often used for testing of the distributed systems because modeling of such systems with finite automata is very time consuming. Most of these methods are not define practically applicable criteria for the determination of the completeness and does not give a final test sets for real systems because they use tools based on certain heuristics for test suite finalizing.

Methods of testing based on formal analysis of SW properties use formal analysis for classification of test situations and targeted test generation.

Methods based on model checking. In these test methods test cases are chosen as representatives of equivalence classes, defined by covering criteria. Concrete test sequence is constructed as an example of testing model (model checking) pursuant properties, which is the negation of the conditions of this situation achievement.

Methods based on **deductive analysis**. In these methods selected test cases is in special cases deductive analysis of the properties of the tested system.

7.2. Model for verification of the Embedded System

For automation of compound verification methods authors suggested model of verification, based on basic model for web-systems [12-14], for definition of the main concept of the system.

Conceptual base and semantic of the IS is presented by the domain model, which is

$$DM = (C, D, RC, RD),$$

Where $C = CC \cup CA$ – set of definitions (composition and atomic), D – set of domain elements, RC – set of relations between concepts, RD – set of relations between concept instances.

Element of the IS ed – is any artifact, for example it can be a document, available in the NET with unique URL, and defined ed as function:

$$rd : CA \rightarrow Ed | \forall c \in CA : ed = rd(c), ed \in Ed,$$

where CA – set of atom conceptions, Ed – set of the IS software (SW) and hardware (HW) resources. In the other words it is a set of integrated SW-HW techniques and information, which should be published in the net in the predefined form (text, figure, audio).

Model of stakeholders requirements have special architecture, which consists from requirements to the information types and required methods of proceedings. Let's divide stakeholders information into two parts – user profile and implicit requirements of the user.

Explicit requirements (user profile) u – set $UP(u) = (A, V, rp)$

$$rp: A \rightarrow V | \forall a \in A : rp(a) \in Va,$$

where A – set of attributes, defined as user characteristics dictionary, $V = \bigcup_{a \in A} V_a$ – set of attributes values i V_a – attribute range a .

Implicit user requirements in these terms denote model that includes implicit user characteristics. While the obvious characteristics are set by the user, implicit characteristics are determined by adaptive system. In the system different values relating to a specific subject domain are collected.

Implicit user requirements u – is a set $UM(u) = (D, A, V, rm)$

$$rm: D \times A \rightarrow V | \forall a \in D \times A : rm(a) \in Va,$$

where D – is a set of domain concept instances;
 A – set of attributes, which are determined as dictionary of implicit user features;

$V = \bigcup_{a \in A} V_a$ – set of attributes values;

V_a – range for attribute a .

The set of explicit $UP(u)$ and implicit $UM(u)$ user requirements, which are depicted as a set of prototype interface elements form the basis for the IS that is used to demonstrate the concepts embodied in a system, check of the user cases and is used for the challenges search:

$$interprototype = (A, D, V, rp, rm).$$

Adaptive function fa is describing transmission between IS elements and stakeholders requirements. IS element, for example can be part of the code, which is also a part of some subsystem:

$$fa : ea \rightarrow e,$$

where $fa \in AF$,

ea – elementary unit of SW of IS interprototype, $ea \in \{rp(a), rm(a)\}$;

$e = \{ed\}$ – subset of SW of IS resources, $e \subset E_d$, ed – IS element, E_d – set of web-resources.

Effectiveness of IS functioning can be evaluated by such important criteria as user compliance and accuracy of information, so in a formal statement the problem of efficiency estimation can be represented as follows:

$$\begin{cases} interprototype \rightarrow E_d \\ DB \subset (DB_1 \cap DB_2 \cap \dots \cap DB_N), \end{cases}$$

where interprototype – combined set of explicit and implicit user requirements of the IS software functionality;

$E_d = \{ed\}$ – existing functionality of the IS software, described as a set of IS elements;

$DB_1, DB_2, \dots, DB_n$ – IS datastore in case of data distribution or use of federated approach usage to data integration (data approaches being

used at the opportunity to get the most current information without delay in updating, as is the case with the consolidation of data). Each *DB* includes a set *D*.

For solving this task it is important to develop following areas:

– considering that user interface prototype is the best way for presenting of user requirements, there are appear the task of comparing of existing functional elements and elements of the prototype:

$$interprototype \equiv E_d$$

– for identification of the uncertainty of the IS and to the requirements it is necessary to develop methods for risk of IS development identification, and methods which should reduce consequences of those risks, considering the specific characteristics of the model and its requirements;

– for data accuracy control, for data which are stored in different DB it is important to organize test process for integrated DB, which allow to provide accuracy and actuality of the data with min time losses.

Let's consider $f = \bigcup_{d \in D} \{e_d, p\}$, $e_d \in E_d$, $f \in F$, p – test case for f , set of one or number of IS elements, as «system functional element» – elementary structure component of IS, which realize finished functional block. For such block can be developed one automated test or a test suits:

$$F = (E_d, P, r) | \forall e_d, e_d \in E_d, \exists p, p \in P$$

$$rf : ed \rightarrow p.$$

Functional element (FE) of development should be chosen depending from the type of the IS. For example, if IS required DB or combine different hardware platforms, then FE is a function of data exchange between these components.

Unlike *functional point (FP)* usage for assessment of FE give opportunity to use more unified functional module, specified to concrete IS. These can simplify the procedure of planning and testing of developing elements.

Considering all defined advantages of FE usage and that it is not present in existing models, verification model *MV* will be:

$$\begin{aligned}
 MV &= (interprototype, Ed, DB, rmv) \\
 rmv: interprototype &\rightarrow Ed \times DB \mid \forall ea, ea \in \{rp(a), rm(a)\}, ea \\
 &\rightarrow e, e = \{ed\}, \\
 e_d &\in E_d, E_d \in W_{res}, DB \in W_{res}
 \end{aligned}$$

Where ea – elementary part of IS interprototype, $ea \in \{rp(a), rm(a)\}; e = \{ed\}$ – subset of resources of IS , $e \in E_d$, ed – IS element, Ed – set if web-resources.

7.3. Individual tasks

1. Prepare a review on the history of formal verification methods development.
2. Give extended review for one of the following verification method:
 - a. Kripke Structures and Labeled Transition Systems
 - b. Temporal logics
 - c. Model Checking
3. For the existing software project apply one of the static verification method:
 - a. technical review;
 - b. walkthrough;
 - c. inspection;
 - d. audit
4. Devide existing software project into the functional points and functional elements. Explain the difference, provide analysis of number of element and connections between the elements.

7.4. Control questions

1. What is the difference between verification and validation?
2. What approaches are used for static verification?
3. What is the aim of formal verification methods?
4. What is the difference between static and dynamic verification?
5. What are the common characters and the differences between technical review, walkthrough, inspection and audit?

6. What are the most common approaches for formal verification?
7. What are 5 basic characteristics of model-based checking?
8. What is the difference between functional point and functional element?
9. What is the purpose of user requirements prototype in the formal verification?
10. What type of verification “model of verification of the ES” is referred to?

8 EMBEDDED SYSTEMS TESTING

Embedded systems (ES) are compound systems, which can combine different architectures. Moreover the most of the ES – are real time systems, so the result depends not only on logical algorithm of functioning but from time of response as well.

Mainly designing of Software and Hardware for Embedded Systems are separated processes. But for system diagnostic purposes correlation between software and hardware are evident. Faults can appear as sequences of software defects as hardware faults. Most of the approaches don't consider testing at system level and implementation level as a common approach. So this chapter is devoted to both software and hardware testing.

To support the engineers during the development of software, there are used different models. One of the most commonly used is the V-model (fig. 8.1).

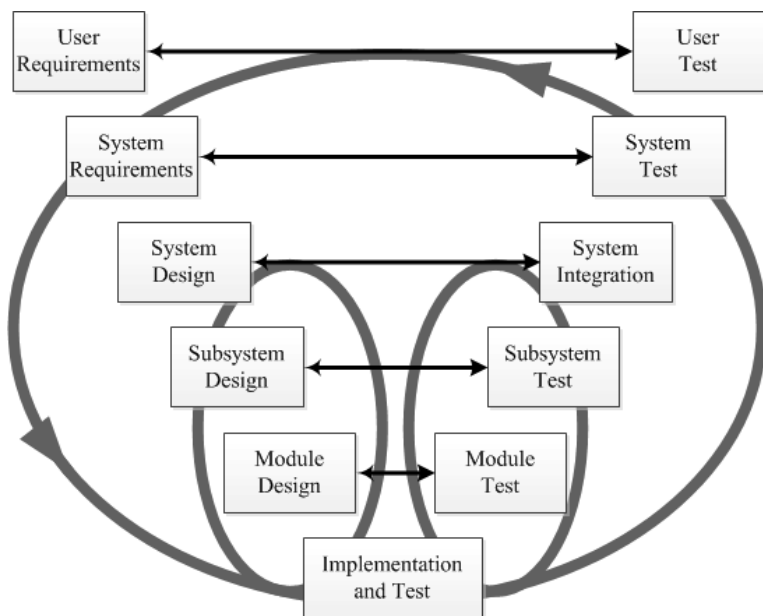


Figure 8.1 V-model

8.1. Terms and definitions

Test is a sequence of operations and sequence of events which should be proceeded after conduction the operation

According to IEEE 829-2008 [20] **test case** is:

- a set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement;
- documentation specifying inputs, predicted results, and a set of execution conditions for a

Test suite is a set of the test cases.

Test specification is a description of the system environment or the expected system behavior. It is used to derive test suites and to compare expected and observed system behavior.

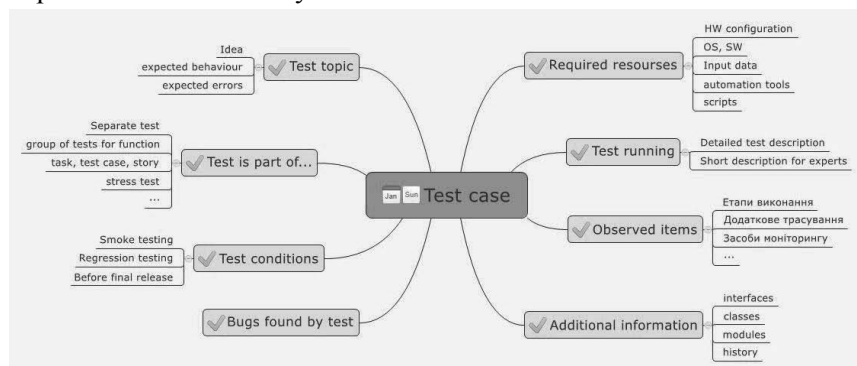


Figure 8.2 Test case template

Test case specification varies by organization. But using a standard test case specification for writing test cases is one step closer to setting up testing process on your project.

With the option of testing should be linked to the following features:

- it must have a high probability of the defect detection;
- it must be reproducible;
- it must have clearly defined expected results and criteria of successful or failed the test exit;
- it should not be excessive.

Let consider the main tasks of the testing:

Find defects. This is the classic objective of the testing. A test is run in order to trigger failures that expose defects. Generally, we look for defects in all interesting parts of the product.

Maximize bug count. The distinction between this and “find defects” is that total number of bugs is more important than coverage. We might focus narrowly, on only a few high-risk features, if this is the way to find the most bugs in the time available.

Block premature product releases. This tester stops premature shipment by finding bugs so serious that no one would ship the product until they are fixed. For every release decision meeting, the tester’s goal is to have new showstopper bugs.

Help managers make ship/no-ship decisions. Managers are typically concerned with risk in the field. They want to know about coverage, and how important the known problems are. Problems that appear significant on paper but will not lead to customer dissatisfaction are probably not relevant to the ship decision.

Minimize technical support costs. Working in conjunction with a technical support or help desk group, the test team identifies the issues that lead to calls for support. These are often peripherally related to the product under test--for example, getting the product to work with a specific printer or to import data successfully from a third party database might prevent more calls than a low-frequency, data-corrupting crash.

Assess conformance to specification. Any claim made in the specification is checked. Program characteristics not addressed in the specification are not (as part of this objective) checked.

Conform to regulations. If a regulation specifies a certain type of coverage the test group creates the appropriate tests. If the regulation specifies a style for the specifications or other documentation, the test group probably checks the style. In general, the test group is focusing on anything covered by regulation and (in the context of this objective) nothing that is not covered by regulation.

Minimize safety-related lawsuit risk. Any error that could lead to an accident or injury is of primary interest. Errors that lead to loss of time or data or corrupt data, but that don’t carry a risk of injury or damage to physical things is out of scope.

Find safe scenarios for use of the product. Sometimes, all that you’re looking for is one way to do a task that will consistently work--one set

of instructions that someone else can follow that will reliably deliver the benefit they are supposed to lead to. In this case, the tester is not looking for bugs. He is trying out, empirically refining and documenting, a way to do a task.

Assess quality. This is a tricky objective because quality is multi-dimensional. The nature of quality depends on the nature of the product.

Verify correctness of the product. It is impossible to do this by testing. You can prove that the product is not correct or you can demonstrate that you didn't find any errors in a given period of time using a given testing strategy. However, you can't test exhaustively, and the product might fail under conditions that you did not test. The best you can do is assessment-test-based estimation of the probability of errors.

Assure quality. Despite the common title, quality assurance, you can't assure quality by testing. You can't assure quality by gathering metrics. You can't assure quality by setting standards. Quality assurance involves building a high quality product and for that, you need skilled people throughout development who have time and motivation and an appropriate balance of direction and creative freedom. This is out of scope for a test organization. It is within scope for the project manager and associated executives. The test organization can certainly help in this process by performing a wide range of technical investigations, but those investigations are not quality assurance.

As it was mention earlier the main aim of the testing is finding defects. But there are a lot of types of defects defined in various ways in different literature. Let's define such terms as failure, error and fault.

Failures. A failure occurs when the user perceives that a software program is unable to deliver the expected service. The expected service is described by a system specification or a set of user requirements.

Errors. An error is part of the system state which is liable to lead to a failure. It is an intermediate stage in between faults and failures. An error may propagate, i.e., produce other errors.

Faults. A fault, sometimes called a bug, is the identified or hypothesized cause of a software failure. Software faults can be classified as design faults and operational faults according to the phases of creation. Although the same classification can be used in hardware faults, we only interpret them in the sense of software here.

Design faults. A design fault is a fault occurring in software design and development process. Design faults can be recovered with fault

removal approaches by revising the design documentation and the source code.

Operational faults. An operational fault is a fault occurring in software operation due to timing, race conditions, workload-related stress and other environmental conditions. Such a fault can be removed by recovery, i.e., roll-back to a previously saved state and executed again.

Faults in embedded systems can be classified as:

- mechanical;
- electronical;
- software;
- people.

8.2. Hardware faults models

As the complexity and integration of hardware are increasing with technology, defects are too numerous and very difficult to analyze. A fault model helps us to identify the targets for testing and analysis of failure. Further, the effectiveness of the model in terms of its relation to actual failures should be established by experiments. Faults in a digital system can be classified into three groups: design, fabrication, and operational faults. Design faults are made by human designers or CAD software (simulators, translators, or layout generators), and occur during the design process. These faults are not directly related to the testing process. Fabrication defects are due to an imperfect manufacturing process. Defects on hardware itself, bad connections, bridges, improper semiconductor doping and irregular power supply are the examples of physical faults. Physical faults are also called as defect-oriented faults. Operational or logical faults are occurred due to environmental disturbances during normal operation of embedded system. Such disturbances include electromagnetic interference, operator mistakes, and extremes of temperature and vibration. Some design defects and manufacturing faults escape detection and combine with wearout and environmental disturbances to cause problems in the field.

Hardware faults are classified as stuck-at faults, bridging faults, open faults, power disturbance faults, spurious current faults, memory faults, transistor faults etc. The most commonly used fault model is that of the “stuck-at fault model” [1]. This is modeled by having a line segment stuck at logic 0 or 1 (stuck-at 1 or stuck-at 0).

Stuck-at Fault: This is due to the flaws on hardware, and they represent faults of the signal lines. A signal line is the input or output of a logic gate. Each connecting line can have two types of faults: stuck-at-0 (s-a-0) or stuck-at-1 (s-a-1). In general several stuck-at faults can be simultaneously present in the circuit. A circuit with n lines can have $3n - 1$ possible stuck line combinations as each line can be one of the three states: s-a-0, s-a-1 or fault free. Even a moderate value of n will give large number of multiple stuck-at faults. It is a common practice, therefore to model only single stuck-at faults. An n -line circuit can have at most $2n$ single stuck-at faults. This number can be further reduced by fault collapsing technique.

Single stuck-at faults is characterized by the following properties:

1. Fault will occur only in one line.
2. The faulty line is permanently set to either 0 or 1.
3. The fault can be at an input or output of a gate.
4. Every fan-out branch is to be considered as a separate line.

Bridging faults: These are due to a short between groups of signal. The logic value of the shorted net may be modeled as 1-dominant (OR bridge), 0-dominant (AND bridge), or intermediate, depending upon the technology in which the circuit is implemented.

Stuck-Open and Stuck-Short faults: MOS transistor is considered as an ideal switch and two types of faults are modeled. In stuck-open fault a single transistor is permanently stuck in the open state and in stuck-short fault a single transistor is permanently shorted irrespective of its gate voltage. These are caused by bad connection of signal line.

Power disturbance faults: These are caused by inconsistent power supplies and affect the whole system.

Spurious current faults: that exposed to heavy ion affect whole system.

Operational faults are usually classified according to their duration:

Permanent faults exist indefinitely if no corrective action is taken. These are mainly manufacturing faults and are not frequently occur due to change in system operation or environmental disturbances.

Intermittent faults appear, disappear, and reappear frequently. They are difficult to predict, but their effects are highly correlated. Most of these faults are due to marginal design or manufacturing steps. These faults occur under a typical environmental disturbance.

Transient faults appear for an instant and disappear quickly. These are not correlated with each other. These are occurred due random environmental disturbances. Power disturbance faults and spurious current faults are transient faults.

8.3. Testing of Embedded Core-Based System-on-Chips

The system-on-chip (SoCs) test is a single composite test comprised of the individual core tests of each core, the User Defined Logic UDL tests, and interconnect tests. Each individual core or UDL test may involve surrounding components. Certain operational constraints (e.g., safe mode, low power mode, bypass mode) are often required which necessitates access and isolation modes [11].

In a core-based system-on-chip, the system integrator designs the UDL and assembles the pre-designed cores provided by the core vendor. A core is typically hardware description of standard IS e.g., DSP, RISC processor, or DRAM core. Embedded cores represent intellectual property (IP) and in order to protect IP, core vendors do not release the detailed structural information to the system integrator. Instead a set of test pattern is provided by the core vendor that guarantees specific fault coverage. Though the cores are tested as part of overall system performance by the system integrator, the system integrator deals the core as a black box. These test patterns must be applied to the cores in a given order, using a specific clock strategy.

In SOC's cores are often embedded in several layers of user-defined or other core-based logic, and direct physical access to its peripheries is not available from chip I/Os. Hence, an electronic access mechanism is needed. This access mechanism requires additional logic, such as a wrapper around the core and wiring, such as a test access mechanism to connect core peripheries to the test sources and sinks.

A conceptual architecture for testing embedded-core-based SOC's is shown it consists of three structural elements: (1) test pattern source, (2) test access mechanism, and (3) core test wrapper.

The test pattern source generates the test stimuli for the embedded core, and the test pattern sink compares the response(s) to the expected response(s). Test pattern source as well as sink can be implemented either off-chip by external Automatic Test Equipment (ATE), on-chip by Built-

In Self-Test (or Embedded ATE), or as a combination of both. Source and sink do not need to be of the same type, e.g., the source of an embedded core can be implemented off-chip, while the sink of the same core is implemented on-chip. The choice for a certain type of source or sink is determined by the type of circuitry in the core, the type of pre-defined tests that come with the core and quality and cost considerations. The type of circuitry of a certain core and the type of predefined tests that come with the core determine which implementation options are left open for test pattern source and sink. The actual choice for a particular source or sink is in general determined by quality and cost considerations. On-chip sources and sinks provide better accuracy and performance related defect coverage, but at the same time increase the silicon area and hence might reduce manufacturing yield.

The test access mechanism takes care of on-chip test pattern transport. It can be used (1) to transport test stimuli from the test pattern source to the core-under-test, and (2) to transport test responses from the core-under-test to the test pattern sink. The test access mechanism is by definition, implemented on-chip. Although for one core often the same type of test access mechanism is used for both stimulus as well as response transportation, this is not required and various combinations may co-exist. Designing a test access mechanism involves making a trade-off between the transport capacity (bandwidth) of the mechanism and the test application cost it induces. The bandwidth is limited by the bandwidth of source and sink and the amount of silicon area one wants to spend on the test access mechanism itself.

The core test wrapper forms the interface between the embedded core and its system chip environment. It connects the core terminals both to the rest of the IS, as well as to the test access mechanism. By definition, the core test wrapper is implemented on-chip.

The core test wrapper should have the following mandatory modes.

- Normal operation (i.e., non-test) mode of the core. In this mode, the core is connected to its system-IS environment and the wrapper is transparent.
- Core test mode. In this mode the test access mechanism is connected to the core, such that test stimuli can be applied at the core's inputs and responses can be observed at the core's outputs.

- **Interconnect test mode.** In this mode the test access mechanism is connected to the interconnect wiring and logic, such that test stimuli can be applied at the core's outputs and responses can be observed at the core's inputs.

Apart from these mandatory modes, a core test wrapper might have several optional modes, e.g., a detach mode to disconnect the core from its system chip environment and the test access mechanism, or a bypass mode for the test access mechanisms. Depending on the implementation of the test access mechanism, some of the above modes may coincide. For example, if the test access mechanism uses existing functionality, normal operation and core test mode may coincide.

Pre-designed cores have their own internal clock distribution system. Different cores have different clock propagation delays, which might result in clock skew for inter-core communication. The system-IS designer should take care of this clock skew issue in the functional communication between cores. However, clock skew might also corrupt the data transfer over the test access mechanism, especially if this mechanism is shared by multiple cores. The core test wrapper is the best place to have provisions for clock skew prevention in the test access paths between the cores.

In addition to the test integration and interdependence issues, the system chip composite test requires adequate test scheduling. Effective test scheduling for SOC's is challenging because it must address several conflicting goals: total SOC testing time minimization, power dissipation, precedence constraints among tests and area overhead constraints. Also, test scheduling is necessary to run intra-core and inter-core tests in certain order not to impact the initialization and final contents of individual cores.

8.4. Boundary scan

Boundary Scan is a family of test methodologies aiming at resolving many test problems: from chip level to system level, from logic cores to interconnects between cores, and from digital circuits to analog or mixed-mode circuits. It is now widely accepted in industry and has been considered as an industry standard in most large IS system designs. Boundary-scan, as defined by the IEEE Std. 1149.1 standard, is an

integrated method for testing interconnects on printed circuit board that is implemented at the IS level.

The Joint Test Action Group (JTAG) has developed a specification for boundary-scan testing that was standardized in 1990 by IEEE as the IEEE Std. 1149.1-1990. In 1993 a new revision to the IEEE Std. 1149.1 standard was introduced (1149.1a) and it contained many clarifications, corrections, and enhancements. In 1994, a supplement that contains a description of the boundary-scan Description Language (BSDL) was added to the standard. Since that time, this standard has been adopted by major electronics companies all over the world.

Boundary-scan was first proposed in 1985 and became the IEEE 1149.1 standard in 1990. During the first few years after 1990, boundary-scan gradually gained in popularity as IS silicon vendors provided more devices compliant with the IEEE 1149.1 standard. At this time, a great many boundary-scan devices are available, and literally thousands of production lines around the world use boundary-scan routinely in board testing and in-system programming.

To comply with the standard, a device must include the 4-wire (5-wire if optional reset signal is included) Test Access Port (TAP), internal boundary-scan cells for each pin and associated internal boundary-scan registers and additional multiplexing circuitry. In addition, the device vendor must provide Boundary Scan Description Language (BSDL) files that fully describe the boundary-scan implementation in the associated devices.

The Figure 8.2 shows the following elements:

- Test Access Port (TAP) with a set of four dedicated test pins: Test Data In (TDI), Test Mode Select (TMS), Test Clock (TCK), Test Data Out (TDO) and one optional test pin Test Reset (TRST*);
- a boundary-scan cell on each device primary input and primary output pin, connected internally to form a serial boundary-scan register (Boundary Scan);
- a TAP controller with inputs TCK, TMS, and TRST*;
- an n-bit ($n \geq 2$) instruction register holding the current instruction;
- a 1-bit Bypass register (Bypass);

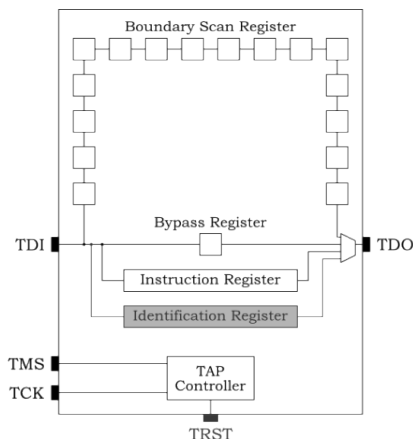


Figure 8.2 JTAG Architecture

- an optional 32-bit Identification register capable of being loaded with a permanent device identification code.

The test access ports (TAP), which define the bus protocol of boundary scan, are the additional I/O pins needed for each chip employing Std.1149.1a. The TAP controller is a 16-state final state machine that controls each step of the operations of boundary scan. Each instruction to be carried out by the boundary scan architecture is stored in the Instruction Register. The various control signals associated with the instruction are then provided by a decoder. Several Test Data Registers are used to stored test data or some system related information such as the chip ID, company name, etc.

Boundary Scan Description Language (BSDL) has been approved as the IEEE Std. 1149.1b (the original boundary scan standard is IEEE Std. 1149.1a) [8,9]. This VHDL compatible language can greatly reduce the effort to incorporate boundary scan into a chip, and hence is quite useful when a designer wishes to design boundary scan in his own style. Basically for those parts that are mandatory to the Std. 1149.1a such as the TAP controller and the BYPASS register, the designer does not need to describe them; they can be automatically generated. The designer only has to describe the specifications related to his own design such as the length of boundary scan register, the user-defined boundary scan instructions, the decoder for his own instructions, the I/O pins

assignment. In general these descriptions are quite easy to prepare. In fact, currently many CAD tools already implement the boundary scan generation procedure and thus it may even not be needed for a designer to write the BSDL file: the tools can automatically generate the needed boundary scan circuitry for any circuit design as long as the I/O of the design is specified.

Any manufacturer of a JTAG compliant device must provide a BSDL file for that device. The BSDL file contains information on the function of each of the pins on the device – which are used as I/Os, power or ground. BSDL files describe the Boundary Scan architecture of a JTAG-compliant device, and are written in VHDL. The BSDL file includes:

1. **Entity Declaration:** The entity declaration is a VHDL construct that is used to identify the name of the device that is described by the BSDL file.
2. **Generic Parameter:** The Generic parameter specifies which package is described by the BSDL file.
3. **Logical Port Description:** lists all of the pads on a device, and states whether that pin is an input(in bit;), output(out bit;), bidirectional (in/out bit;) or unavailable for boundary scan (linkage bit;).
4. **Package Pin Mapping:** The Package Pin Mapping shows how the pads on the device die are wired to the pins on the device package.
5. **Use statements:** The use statement calls VHDL packages that contain attributes, types, constants, etc. that are referenced in the BSDL File.
6. **Scan Port Identification:** The Scan Port Identification identifies the JTAG pins: TDI, TDO, TMS, TCK and TRST (if used).
7. **TAP description:** provides additional information on the device's JTAG logic; the Instruction Register length, Instruction Opcodes, device IDCODE, etc. These characteristics are device specific.
8. **Boundary Register description:** provides the structure of the Boundary Scan cells on the device. Each pin on a device may have up to three Boundary Scan cells, each cell consisting of a register and a latch.

Many of the IS devices in use on your products may already support boundary-scan (or JTAG as some IS vendors call it), whether you're using it or not. For example, the PowerPCs of IBM and Motorola, programmable logic devices from Altera, Xilinx, Lattice and STMicroelectronics, digital signal processors (DSPs) from Texas Instruments, wireless

processors from Qualcomm, most application specific ICs (ASICs), and many others fully comply with IEEE 1149.1. So, it's quite possible that you can easily tap into the power of boundary-scan to improve board testability, dismantle the bone-pile, speed up board design and prototype debug, and even perform in-system programming of flash memory and programmable logic devices via the boundary-scan chain.

8.5. Embedded systems software testing

Software testing (software testing) – a planned, ordered activity, it is process of analysis or usage of software aimed to identify defects.

All types of embedded systems software testing can be classified, in terms of the inspected elements according to quality characteristics, to the data sources or on the role of the team.

Software testing can be divided roughly to four levels. They are, from low-level to high-level: unit testing, integration tests, system testing and acceptance testing.

Unit testing is the lowest level of software testing process in which individual units of software is tested. A unit is the smallest testable part of software, which is usually a single function. This level of testing is normally carried out by developers in parallel with development process as a part of test driven development practice. In unit testing a white box testing method is often used due to the small size and simplicity of the code under test.

Integration testing is a level of software testing process where the units are combined and tested as functional groups. At this level the emphasis is on testing the interface and interaction between the units. Testing method can be black box testing, white box testing or grey box testing.

System testing is a higher level of the software testing process where a complete integrated system is tested. On embedded systems this phase is usually carried out on prototypes where the functionality of the embedded software in the destination platform is tested in black box method. System testing aims to reveal defects in which the software does not meet the requirements set for the product and defected interaction between the embedded system and the plant. System testing is usually carried out by independent testing team but on small projects this phase can be carried out by the developers.

Acceptance testing is the final stage of testing where the final product is put under test and the design is accepted for release. The aim is to find the defects in which the product does not fulfil its original requirements and purpose for the customer.

According to performance characteristics there are distinguished functionality testing, reliability tests, portability tests, usability testing.

Based on data data sources there are exist – black box testing, grey box testing and white box testing.

Functional tests are based on software functions, features and interoperability with other systems, and consider the external behavior of the system. The most common types of functional tests are: functional testing; security and access control testing; interoperability testing.

Non-functional testing describes the tests required to determine the characteristics of software that can be measured by different variables. Overall, this testing of “how” the system works. This is all kinds of performance testing such as: performance and load testing; stress testing; stability/reliability testing; volume testing; installation testing; usability testing; failover and recovery testing; configuration testing.

In order to be sure that the developed system is really usable, one have to assure that the system for example responds in a certain amount of time, handles a specified number of transactions per minute or uses no more memory than specified. Otherwise, the end user might find the system unusable. The purpose of **performance testing** is to find such defects in the system according to specified requirements. It is very important that those requirements are measurable.

When a system is input with a large load of data in a short period of time, it is called a **stress test**. The purpose is trying to find in which situation the system breaks.

Another type of system testing is loading the software with extremely large data. The purposes of **volume testing** are to show that the software cannot handle the volume of data specified in its requirements and to find how much data it can handle.

Usability testing is a rather broad category, since it includes all user interface questions, e.g. is the data input syntax consistent and are the possible error messages and indicators logical.

The design purpose of systems with recovery facilities is to minimize the mean time to recovery, because downtime often causes the company to lose money since the system is inoperable. When performing **recovery**

testing the system is subjected to device failures in order to check if the software can detect such device failures and continue from a known state.

Often, systems are required to run under different types of software or hardware configurations and they have to be tested on all possible configurations in order to be sure that they fulfil the configuration requirements. When developing a web based application for example, the system can have requirements that it can be run in one or many web browsers. When writing test cases for configuration testing, tests should be focused to find defects in the areas where the different platforms are known to have differences.

After improvement, software should be tested once more for conformance the fact that the problem was solved. For these aims there are used: smoke testing; regression testing; build verification test; sanity testing.

The purpose of **smoke testing** is to determine whether the new software build is stable or not so that the build could be used for detailed testing by the QA team and further work by the development team. Smoke Testing is generally done by the QA team but in certain situations, can be done by the development team. In that case the development team checks the stability of the build and deploy to QA only if the build is stable.

Sanity testing is the term which is related to smoke testing but they are different. One similarity between two is that both are used as criteria for accepting/rejecting the new build. If sanity test cases fail then build is rejected because if the build is not containing the required changes then there is no point of doing regression testing on the build. Smoke is being part of regression testing, checks the crucial functionality while sanity testing is part of acceptance testing, checks whether the newly added functionality is working or not. Generally smoke is performed on relatively unstable product while sanity testing is done on the relatively stable product.

Moreover generally only one of them is performed but if we need to do both then smoke testing is performed first and then sanity testing is done.

Regression testing is not defined as a level of testing, but it is the retesting of software that occurs when changes are made to ensure that the software still works according to the specification. Regression testing can be implemented at any level of test. According to Burnstein, regression testing is especially important when multiple software releases are developed.

Testing is conducted at various levels throughout the life cycle development and support of ES software. Level test identifies something that made tests: on a separate module or group of modules of the system as a whole. Testing at all levels of the system – is the key to successful project completion.

8.6. Regression testing for Informational Systems with web-interface

Let's consider the method of regression testing of IS, including IS with web-oriented interface (*WOIS*). This class of IS is constructed based on the evolutionary prototyping principle. This means that consistently a prototype of the system is created, which will be with each iteration closer to the real product. This approach has the advantage that the developers at every step working with executing system that has partial functionality which can be tested and be improved with each iteration.

For testing lets highlight Functional Elements (FE), which is atomic object connected to the user requirements (IS prototype) and functionality of the system by the adaptive rule, depending from logic of the application. Thus, if we accept for the F – entity selects in IS FE, and F_k – as entity implemented and tested at k -version FE, then IS development progress at the first assumption can be assessed by:

$$M_{progress1} = \frac{|F_k|}{|F|} \times 100\% . \quad (8.1)$$

More accurate information about development state of the IS can give the other criteria which include complexity of FE integration and percentage of fulfillment of FE development and depicts the progress of the IS development according to the project timetable:

$$M_{progress2} = \frac{\sum_{i=1}^N (tp_i \times p_i + p_i^{(int)} \times p_i^{(int)})}{100 \times \sum_{i=1}^N (tp_i + p_i^{(int)})} , \quad (8.2)$$

where tp – FE planned laboriousness; $tp^{(int)}$ – planned labor costs for integration of FE in IS; N – overall number of FE, $N = |F| - |F_{cancel}|$, $|F|$ – overall number of defined FE in IS; $|F_{cancel}|$ – number of FE which are checked; p_i – The percentage of development completion of the of FE; $p_i^{(int)}$ – The percentage of integration of FE completion.

Values are defined after testing, form the number of passed tests for defined FE:

$$\frac{n_test_passed_i}{n_test_total_i} \times 100\%$$

where $n_test_passed_i$ – number of tests for which success criteria are reached for FE_i, $n_test_total_i$ – total number of tests for FE_i.

Deviations of actual labor costs for the IS development from timetable can be determined by criteria:

$$M_{dev} = \sum_{i=1}^N \left[\left(\frac{tf_i \times p_i + tf_i^{(int)} \times p_i^{(int)}}{100} \right) - (tp_i + tp_i^{(int)}) \right], \quad (8.3)$$

where tf – current actual labor costs for FE; $tf^{(int)}$ – current actual labor costs for integration of FE.

All proposed criteria for progress assessment striving to 100% (this corresponds to fully developed IS which meets all requirements), and the value of criteria of deviation from timetable allow to determine not correct assessment of the FE (negative value indicates an inflated estimate labor, positive determines the excess of the current schedule).

These criteria are effort measures and they are measured in relative scale with respect to the primary task and duration. The calculated values (8.1-8.3) at the intermediate stages give IS development manager quantitative assessment of the project state and possibility to determine

the compliance of process development schedule. Separate usage of the criteria can lead to incorrect conclusions, while their combination allow accurately determine the current state of development of the IS.

Thus, the proposed criteria are objective calculated dynamic criteria for progress state measurement and allow at any iteration of the IS development to determine whether the actual state of the process fit the plan. The values can determine whether all currently scheduled tasks were performed, that there are no not exceeded deadlines and unfinished tasks. The criteria values allow to overestimate the complexity of the individual modules and integrated circuits in general, then adjust the development process in order to reduce the deviation from the scheduled date for the transfer operation, thus reducing the risk of violations of scheduling.

For IS with web interface (WOIS) level of critical web-resources of different levels significantly different and therefore must be considered individually.

The embedding level is a characteristic that is responsible for web pages state in total system structure. The level of embedding is determined by the minimum number of transitions, you need to do to reach this page from the home page. Home page always has level 1. All other pages that you can get out of it in one click – level 2. Page of levels 4 in terms of optimal design is illegal as search engines mainly not proceed their indexing.

Let assume that the quality of a developed web-interface can be measured by the set of metrics based on the structure of the web-oriented informational system (WOIS):

$$WOIS = \langle structure0, S_p, structure, S \rangle ,$$

where $structure0 = \langle Ed2_p, EdN3_p, EdN_{\rightarrow \infty}^p \rangle$ – basic structure of the system; $Ed2_p$ – first level resources; EdN_p – resources located deeper then level 3d; S_p – relationship matrix between IS resources.

$structure = \langle Ed2, EdN3, EdN_{\rightarrow \infty} \rangle$ – structure of WOIS SW;

$Ed2$ – web-resources of the 2 level of WOIS SW;

$Ed3$ – web-resources of the 3d level of WOIS SW;

$EdN_{\rightarrow \infty}$ – More deep level web-resources for of WOIS SW;

S – matrix that contains links between resources for WOIS SW.

Accordingly, $|F|$ – number of selected FE in the SI will be calculated as:

$$|F'| = N2_p + N3_p + N_{\rightarrow\infty}^p,$$

where $N2_p$ – the number of elements of the prototype second level of nesting; $N3_p$ – the number of elements of the prototype third level of nesting; $N_{\rightarrow\infty}^p$ – the number of elements of the prototype third deeper level of nesting;

a F_k – number of realized FE and FE that passed testing in the k-version of the IS will be:

$$|F'_k| = N2 + N3 + N_{\rightarrow\infty},$$

Where $N2$ – the number of web-resources of WOIS SW second level of nesting; $N3$ – the number of web-resources of WOIS SW third level of nesting; $N_{\rightarrow\infty}$ – the number of web-resources of WOIS SW third deeper level of nesting.

Similarly the criteria for conformity assessment testing prototype, the following criteria condition are proposed:

1. – criteria of the number of web-resources prototype version of WOIS second level of nesting:

$$M_{N_2} = \frac{N2}{N2_p} \times 100\%, \quad \lim(M_{N_2}) \rightarrow 100\% \quad (8.4)$$

where $N2 = |Ed2|$ – the number of web-resources of WOIS SW second level of nesting;

$N2_p = |Ed2_p|$ – the number of elements of the prototype second level of nesting;

2. – criteria of the number of web-resources prototype version of WOIS third level of nesting:

$$M_{N_3} = \frac{N3}{N3_p} \times 100\%, \quad \lim(M_{N_3}) \rightarrow 100\% \quad (8.5)$$

where $N3$ – the number of web-resources of WOIS SW third level of nesting;

$N3p$ – the number of elements of the prototype third deeper level of nesting;

3. – criteria of the number of web-resources prototype version of WOIS third deeper level of nesting:

$$M_{N \rightarrow \infty} = \frac{1}{N2_p + N3_p} \times \sum_{i=4}^q N_i \times 100\%,$$

$$\lim(M_{N \rightarrow \infty}) \rightarrow 0 \quad (8.6)$$

where q – maximum of nesting of WOIS SW;

$N2p$ i $N3p$ – in accordance with the number of elements of the prototype second and third level of nesting;

for better indexing WOIS $\lim M_{N \rightarrow \infty} \rightarrow 0$, is the number of web-resources of more than a third nesting level as possible should be less than the number $N3$ i $N2$.

4. MS – criteria of compliance structure of WOIS SW and the prototype interface (because of symmetry relative to the main diagonal considered only the lower triangular matrix):

$$M_s = 2 \times \frac{\sum_{i=1}^N \sum_{j=1}^{i-1} a[i][j]}{N^2 - N}, \lim(M_s) \rightarrow 1 \quad (8.7)$$

where N – number of web-resources of WOIS SW and prototype (if the number of web-resources does not match, to add row or column and to fill it with zeros);

$$a[i][j] = \begin{cases} 1, & \text{if } S_p[i][j] = S[i][j] \\ 0, & \text{if } S_p[i][j] \neq S[i][j] \end{cases}$$

S – Matrix links between web-resources of WOIS;

Sp – Matrix links between elements of the prototype.

In the header rows and columns of the matrix Sp are all elements names of a prototype WOIS:

$$S_p[0][i] = S_p[i][0] = name_page_i,$$

where $i \in [1; N]$ – number of web-resource,

$name_page_i$ – name of the i-th web-resource.

All elements of the main diagonal =0. Other elements of the matrix can have the following values:

$$S_p[i][j] = \begin{cases} 0, & \text{if the relationship between i and j is absent;} \\ 1, & \text{if there is a transition on j from i} \\ 2, & \text{if there is a transition on i from j} \\ 3, & \text{if there is a two-way communication.} \end{cases}$$

Matrix S is constructed similarly.

5. Value complexity developed and approved web-resources version of WOIS in general planned labor WOIS calculated similarly (8.6):

$$M_{progress} = \sum_{i=1}^N \left[\left(\frac{tp_i \times pct_i}{100} + tS_p[i] \right) - (tf_i + tS[i]) \right], \lim(M_{progress}) \rightarrow 0,$$

where $i = 1..N$, $N = |F|$ – the total number of scheduled web-resources;

tp_i – planned labor costs FE;

pct_i – the percentage of completion of the i-th web-resource;

$tS_p[i]$ – planned labor relations for the implementation of the i-th web-resource;

tf_i – current actual labor on the web-resource;

$tS[i]$ – current actual labor relations for the implementation of the i-th web-resource.

These criteria are checked at every stage of life cycle WOIS by means of automated testing WOIS links. This allows you to verify that the number of web-resources at all levels of required number.

Regression testing methods for web-oriented systems

The method laid prototype interface ICs, which are calculated on the basis of quality assessment criteria (8.1 – 8.7), connecting the IS logical model, developed with artifacts in various stages of WOIS.

Consider computing circuit method:

Step 1. Construct a matrix of requirements attributes - attributes to the functional requirements of V added the attribute “Implement the page or FE”, which indicates the web-resource identifier, which will be implemented this functionality.

Step 2. In the field “Implement the page or PI” extracted ID web-resources, and removed duplicate names formed array of web-resources of the future of IS structure0:

$$V_a^{e_j} = \text{"b realize for E"} \Rightarrow structure0,$$

where $V_a^{e_j}$ –web-resource attribute Set; $j = 1..N$, N – the number of web-resources;

structure 0 – IS primary structure that contains information about the nesting level web-resources.

Step 3. Construct a modified structure prototype Sp , i.e. structure interaction web-resource matrix of possible transitions between them $Sp [N+1][N+1]$.

Step 4. As a result of the development in k -th iteration are modified structure PC IS matrix structure and possible transitions implemented web-resources $S [N + 1] [N + 1]$.

Step 5. After the development phase, each iteration is calculated during testing modified criteria for the assessment of the development process PC IS recorded in formulas (8.1 - 8.6), that compared primary structure and modified prototype and structure derived from the analysis developed in the current iteration versions of:

$$\left\{ \begin{array}{l} S [N + 1][N + 1] == Sp [N + 1][N + 1] \\ structure == structure0 \end{array} \right.$$

Step 6. The testing methods are chosen for the changed FE IS. A tested and found errors, depending on the test results are assigned the following statuses:

$$status(bug_i) \in \{ "open", "fixed", "verified fixed", "verified closed" \},$$

where «open» – open and requires testing; «fixed» – reformed; «verified fixed» – tested corrected; «verified closed» – closed corrected.

Step 7. Chosen methods of testing for IS related FE by analyzing the structure of relationships between resources IS.

Formally, the process of selecting test methods as follows:

1. For defects (errors) with the status “*fixed*” should test the web-resource, which is due to functional requirements to which the defect is corrected:

$$if (fixed(bug_i) \& bug_i \in V_d^{e_d^g}) \Rightarrow test(e_d^g),$$

where bug_i – any of no-closure defects, $i=1..Nb$, Nb – number of defects;

$fixed(bug_i)$ – appropriation defect status «*fixed*»;

$V_d^{e_d^g}$ – functional requirement that is placed under the resource e_d^g , $g = 1..N$, N – the number of web-resources;

$test(e_d^g)$ – launching a set of tests for web-resource e_d^g .

2. Identify resources associated with altered for testing:

$$if (S_p[g][k] > 0) \Rightarrow test(e_d^k),$$

where $k = 0..N$ – number of web-resource.

3. If the tested web-resource and all associated with it, have been tested, this defect is assigned the status of “verified fixed” and then “verified closed”:

$$\begin{aligned} & \text{if}(\text{test}(e_d^g) == \text{passed} \ \& \ \text{test}(e_d^k) == \text{passed}) \\ & \Rightarrow \text{verified fixed}(\text{bug}_i) \ \& \ \text{fixed closed}(\text{bug}_i), \end{aligned}$$

where $\text{verified fixed}(\text{bug}_i)$ – appropriation bug_i status “verified fixed”;

$\text{fixed closed}(\text{bug}_i)$ – appropriation bug_i status «fixed closed».

Step 8. The decision to finish development.

Stages 1–7 are performed at each iteration as long as the test results will converge with the prescribed test criteria in terms of completion of testing. The criteria of successful development are $M_{N_{2,3}} = 100\%$, $\lim M_S = 100\%$ and $\lim M_{N \rightarrow \infty} \rightarrow 0$ (the threshold can be set by the customer).

It is necessary that criteria reach $\lim M_{N_{2,3}} \rightarrow 100\%$, $\lim M_S \rightarrow 100\%$ and $\lim M_{N \rightarrow \infty} \rightarrow 0\%$, so for the other values it is important to determine the reasons of deviation from the optimal value and correct the discrepancy prototype (stage 5) or irrelevant prototype (phase 2). In case of changing or adding requirements for the system you should return to the Stage 1.

The proposed method goes beyond the standard goals of quality control and allows ensuring quality at different stages of embedded software life cycle. This is possible because of the systematic monitoring and tracking of the developed IS according to customer requirements of functionality and structure of the IS. Created at each stage artifacts are tested for compliance with prototype and thus, this method provides means of assessing the quality at various stages of the IS by analysis of compliance of the logical model of the developed system adopted by the customer.

8.7. Individual tasks

1. Develop software for regression testing of java application.
2. Implement software realization for regression testing for the project suggested by the teacher. Calculate main criteria. Provide conclusions.
3. Develop test suits for the suggested project.

4. Provide expended review on the hardware fault models.
5. Conduct comparison analysis of the functional testing methods.
6. Give comparison analysis of the non-functional testing methods.

8.8. Control questions

1. What are main elements of test case?
2. What types of hardware errors do you know?
3. What is the purpose of the boundary scan?
4. What peripheral equipment should be used for boundary scan and for what purpose?
5. What is the difference between black box testing, grey box testing and white box testing?
6. What is the purpose of regression testing?
7. What are difference between sanity and smoke testing?
8. What is the aim of conformance testing?
9. In what levels all methods for software testing are divided? Give short characteristic of each level.
10. What is a conceptual architecture for testing embedded-core-based SOC's?
11. What are the difficulties in testing SOC's?

9 TEACHING OF EMBEDDED SYSTEMS TESTING BASED ON GOLDI

9.1. Analysis of RL GOLDi facilities for Quality of Informational System Study

In educational process Informational communicational technologies (ICT) each year are used more and more.

Each educational phase, such as – introduction phase, motivation phase, explanation phase, fixation phase and diagnostic phase, consists from its ICT.

One of the most important parts in engineering disciplines are fixation phase, when students with the help of labs fix discipline material.

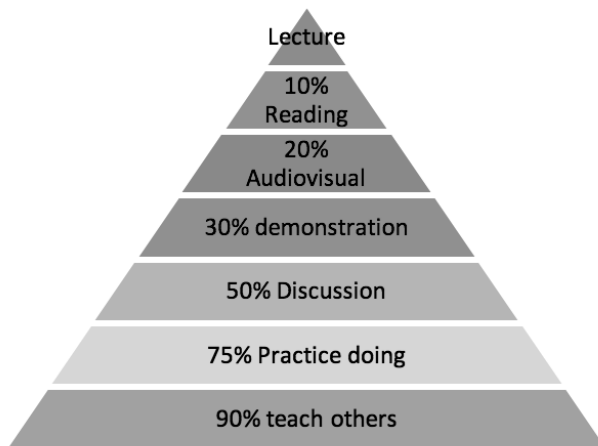


Figure 9.1 Learning Pyramid

Nowadays when there are a lot of unique and expensive equipment it is quite difficult to provide constant access to it. In this case very useful instruments are remote laboratories.

Remote Laboratories (RL) have been successfully developed and implemented in university education worldwide [1-7]. The availability

and flexibility of the remote lab usage in comparison with hands-on labs can significantly contribute to the overall knowledge and boost the creativity of students.

There can be two different starting points for the RL development [4]: building a new RL by conversion or rebuild (remotely) of an existing hands-on equipment with educational scenario and competences to be reached by students; starting with the selection of an existing remotely controlled equipment, next, construction of the scenario for the equipment usage: (re)use of an existing RL for new learning outcomes.

Both approaches have major advantages and disadvantages. When searching a solution teacher should answer questions concerning the Remote laboratory:

1. What are the possibilities of experimenting envisaged in the remote laboratory?
2. What is the variety of input and output data?
3. What methods can be used in the remote laboratory to transform input to output?

The second stage should answer the questions concerned the teaching material:

1. What are the learning objectives of the course/module?
2. What are the learning outcomes?
3. What outcomes can be reached with the usages of the RL?
4. Should there be possibilities for evaluation built in in the RL?

These questions can be systemized with the SWOT table. Depending on the starting point (construction of a new RL or (re)use of an existing device) it looks a little different.

The conversion/rebuild of an existing hands-on experiment has several advantages: the goal of the experiment is well known, the learning outcomes to be reached are described, the experience gained over time in the hands-on experiment can be integrated in the RL, the parameters of the experiment are well known. As such a rebuild is an easy step up to a successful implementation. At table 9.1 SWOT analysis template for implementation of existing remote lab into existing discipline is provided.

Starting with the construction of a new RL has other advantages: no boundary conditions of existing experiments, out-of-the-box new

experiments can be made, innovative approaches can be adventured. The set back is that once the RL has been conceived, a new pedagogical and didactical setting has to be constructed to fit it in the curricula.

Table 9.1 SWOT analysis template for integration of existing courses with existing remote laboratories

	External opportunity <i>What are course objectives?</i> <i>What are the learning outcomes?</i> <i>What methods and technologies students should use?</i> <i>What ICT are used in the course?</i> <i>Where ICT can be used?</i>	External Threats <i>What are prerequisites for a course?</i>
Internal Strength <i>What are the necessary remote Laboratory possibilities?</i> <i>What are the fields of knowledge with what the RL deals with?</i>	How learning outcomes can be reached with the automation in the RL	How can you use RL to avoid discipline pre-requisites?
Internal Weakness <i>What are constraints for input data and output data?</i> <i>What are prerequisites for RL usage?</i> <i>Is there material of the course for what it is impossible to use ICT?</i>	How can the usage of additional ICT can be used to avoid the requirement of constraints and prerequisites?	How can you minimize your weakness to avoid threats?

In ICo-op, facilities of the Ilmenau Interactive Hybrid Online Lab GOLDi are to be used for the industrial study and in the Desire project it should be integrated into the curricula in Embedded systems at Zaporizhzhia National Technical University [15-18].

The concept and the architecture as well as different fields of application of the (GOLDi) were presented in various publications during the last years in detail. A special rapid prototyping board for digital systems was developed for the new operation mode of the Ilmenau Interactive Hybrid Online Lab, presented to fulfill the mentioned design tasks [9].

We suggest to use RL GOLDi for the discipline “Quality of Informational Systems”.

First – there was the analysis of possibilities of the GOLDi’s operating modes: Stand-alone Mode (visual prototyping); Remote Control Mode (via Web-client); Remote Control Mode (via control unit); Virtual Control Mode (visual prototyping); Virtual Control Mode (visual prototyping); Local Control Mode (via control unit); Local Control Mode (manually); Rapid Prototyping Mode; Visitor Mode[7].

GOLDi provides additional functionalities for a Web-based rapid prototyping of digital systems as well as the Web-based verification of such systems. At the fig. 9.2 you can see the variety of possibilities for GOLDi usage.

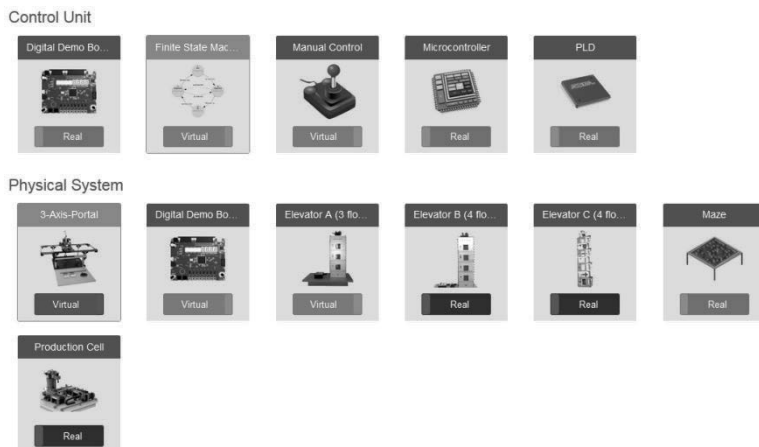


Figure 9.2 – GOLDi operating modes

Next there was an analysis of the Learning Objectives and learning outcomes (fig. 9.3).

Learning Objectives: to explain differences between validation and verification, to describe the common types of faults that occur in embedded systems, to give basics of model-oriented testing, to give knowledge

about boundary scan methods, to describe different approaches to the embedded software testing.

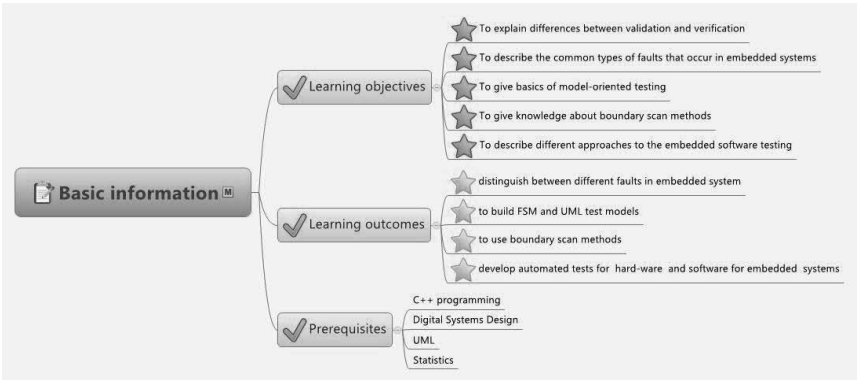


Figure 9.3 Basic information about discipline «Quality of Informational Systems»

All was collected in table 9.2.

Table 9.2 SWOT for GOLDi adaptation for discipline «Quality of informational systems»

	External opportunity	External Threats
Internal Strength	Plan to use: • operation mode “Stand alone mode” for FMS test generation sequence • Operation mode “Remote control Mode” for tests experiments • Rapid Prototype mode for automated functional tests	Plan to developed additional lab session for software test generation that can be applied at the Rapid Prototype mode
Internal Weakness	Plan to use operation mode “Virtual control Mode” for pre-testing knowledge of the students	Plan to develop additional virtual software and tests in LMS Moodle in FMS, VHDL and C from ES.

Learning Outcomes: after going through this module the student would be able to distinguish between different faults in embedded systems; to build FSM and UML test models; to use boundary scan methods; develop automated tests for hardware and software for embedded systems.

Internal weakness are requirements of pre-knowledge in C++,UML, FSM, Digital System Design.

External threat were defined as: impossible to get access to the Remote and Virtual control modes due to the Internet connections.

Let's consider further the usage of GOLDi lab for teaching FSM for model-based testing, rapid prototype board and microcontroller programming for functional testing.

9.2. RL GOLDi usage for model-based testing

Model-based functional testing is based on comparing the system under the test to a test model. This comparison usually consists of automatically generating a test suite from the test model, executing the test suite, and comparing the observable behavior to the expected one. Important advantages of model-based testing are formal test specifications that are close to requirements, traceability of these requirements to test cases, and the automation of test case design, which helps reducing test costs.

Let's consider typical workflow for model-based testing:

1. Build the model; e.g. finite-state machine abstraction of system's event structure
2. Generate expected inputs; e.g. trace of events for finite-state machine
3. Generate expected output; e.g. target state
4. Compare actual output with expected one, e.g. was target state reached?
5. Decide on further actions; e.g. modify model, generate more tests, estimate reliability

In constructing the model, it must meet the following requirements:
– simplicity – the model should be so simple that its construction

costs could be recouped. Good simple model does not require large investments in training. The model should be intuitive to each employee;

- detail – the model should be so detailed that it can be described all states, and program settings for full testing;
- testability – the model should be built so that the generation tests could receive test kits, suitable for manual testing (and later, and automated);
- automation – potential model must support automated testing. That transitions between states should be so “low-level” so that they could transmit the input test automation tool.

To build options for testing based on models of effective tools are: state diagram UML, finite state machines, machines Mealy, Moore automaton, context-independent grammar Markov networks, the table making.

When using finite automata test conditions can be selected on the following criteria:

What value can take an option? Check the limits invalid value, normal values

What set of values have special meaning? If the parameters depend on each other – depending on track

What combination of options leads to the correct operation of the program / method / module?

What combinations of cause error return code is incorrect or abnormal behavior?

Figure 9.4 reflects possible scenarios in the formation models.

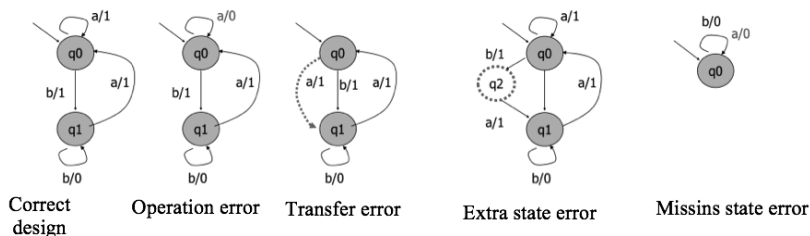


Figure 9.4 FSM for errors

The most common FSM is used for conformance testing, transition testing and coverage testing.

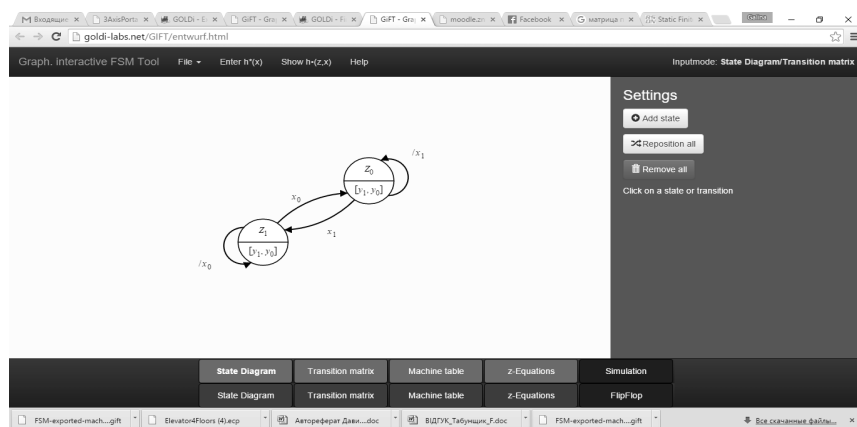


Figure 9.4 GOLDi Graphical Interactive FSM Tool

Manipulating with GOLDi you can use Finite State Machines (FSM) as specification for real or virtual modes (fig. 9.5-9.6).

For teaching FSM GOLDi contains Graphical Interactive FSM Toolset (GIFT) (fig. 9.4) and provide such functionality as:

- general FSM administration,
- input the FSM as automaton graph or transition table,
- handling of edge weights and coding of nodes;
- simulation of single and parallel automata;
- simulation of single and parallel automata,
- generation of the next-state (z) and output (y) equations.
- generation equations for D and JK flip-flops,
- export of z/y equations to the ECP of the *GOLDi* system.

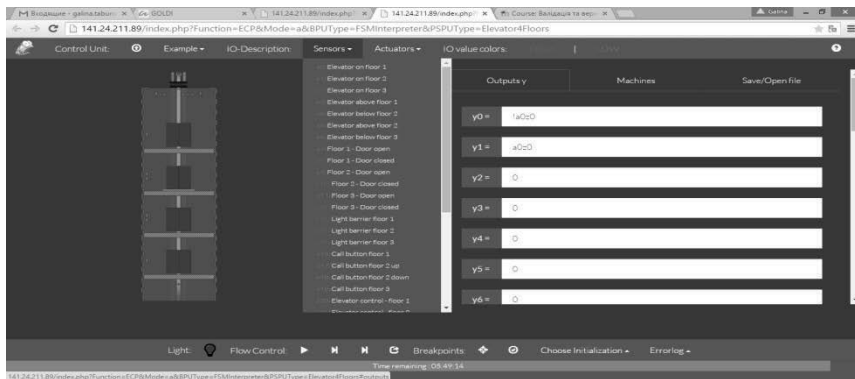


Figure 9.5 FSM for virtual Lift

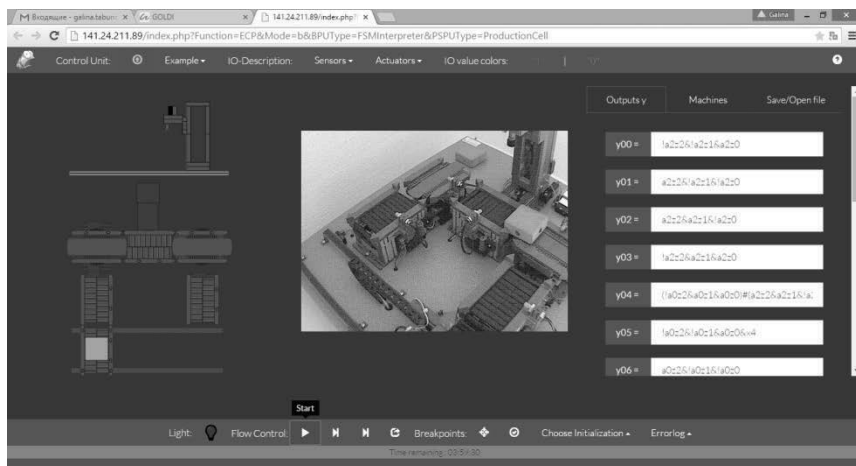


Figure 9.6 FSM for remote Process Cell experiment

Let's consider the example modelling continuous opening and closing the door at the floor 1.

1. So let's define two states Z0 – door on the first floor is opening, Z1 – door on the first floor is closing
2. Actuators: y3=1 make door first floor opening; y4=1 make door first floor closing;
3. Sensors: x00=1 when lift on the first floor; x07=1 indicate that door first floor is open, x08=1 indicate that door first floor is closed

For FSM modeling we can use GIFT, where $x_1=x_0$, $x_2=x_8$, $y_0=y_3$ and $y_1=y_4$; (fig. 9.7)

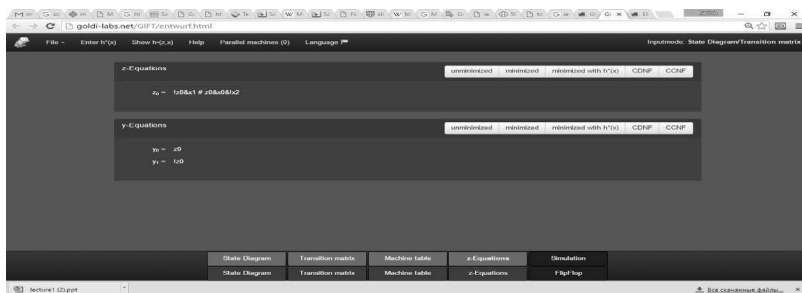


Figure 9.7 GIFT z-equations

As the result we got a machine which we can apply as to virtual as to real mode:

$$a_0z_0 = !a_0z_0 \& x_7 \# a_0z_0 \& x_8 \& !x_8$$

$$y_3 = !a_0z_0; y_4 = a_0z_0;$$

9.3. RL GOLDi usage for functional testing of Embedded systems

Testing is an organized process to verify the behavior, performance, and reliability of a device or system against designed specifications.

Testing is comparing actual and expected behavior: Without that comparison it would be impossible to detect functional failures.

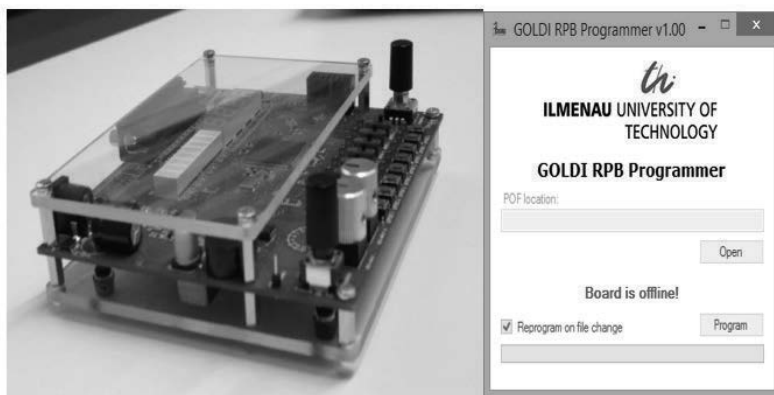


Figure 9.9 – Stand-alone board

The heart of hybrid online remote lab GOLDi is rapid prototype board. There are existing possibilities to work with it as with stand-alone regime (fig. 9.9) as in virtual and remote experiments within laboratory (fig. 9.10 – 9.11).

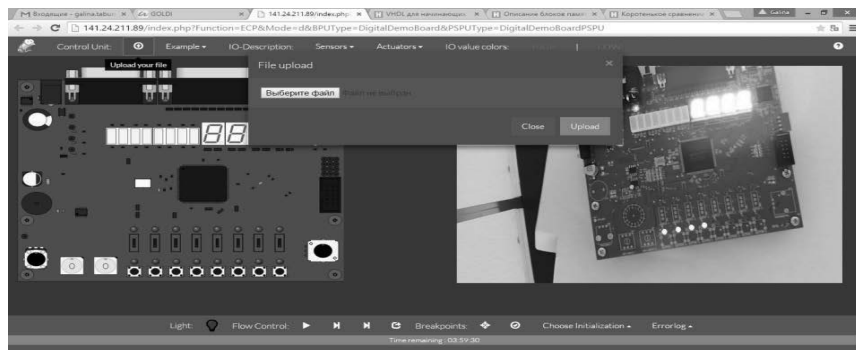


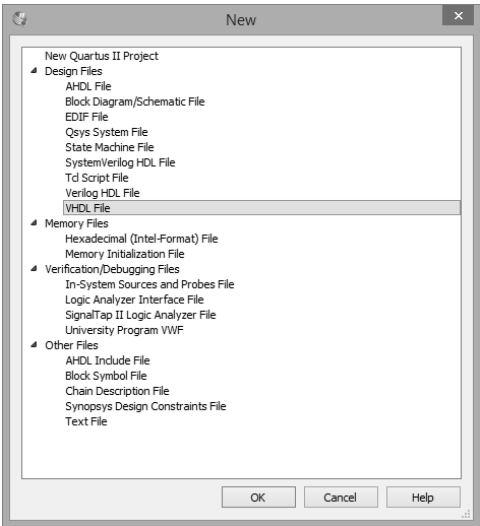
Figure 9.10 Remote laboratory GOLDi

The result Quartus II. file is stored at output folder in .pof (fig. 9.10). Design in Quartus II consists form following steps:

1. *Design Entry* – the desired circuit is specified either by means of a schematic diagram, or by using a hardware description language, such as Verilog or VHDL.
2. *Synthesis* – the entered design is synthesized into a circuit that consists of the logic elements (LEs) provided in the FPGA chip.
3. *Functional Simulation* – the synthesized circuit is tested to verify its functional correctness; this simulation does not take into account any timing issues.
4. *Fitting* – the CAD Fitter tool determines the placement of the LEs defined in the netlist into the LEs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs.
5. *Timing Analysis* – propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit.
6. *Timing Simulation* – the fitted circuit is tested to verify both its functional correctness and timing.
7. *Programming and Configuration* – the designed circuit is implemented in a physical FPGA chip by programming the configuration switches

that configure the LEs and establish the required wiring connections
Let's consider example.

Step 1. Create VHDL project (fig 9.11)



Step 2. Synthesis (fig. 9.12).

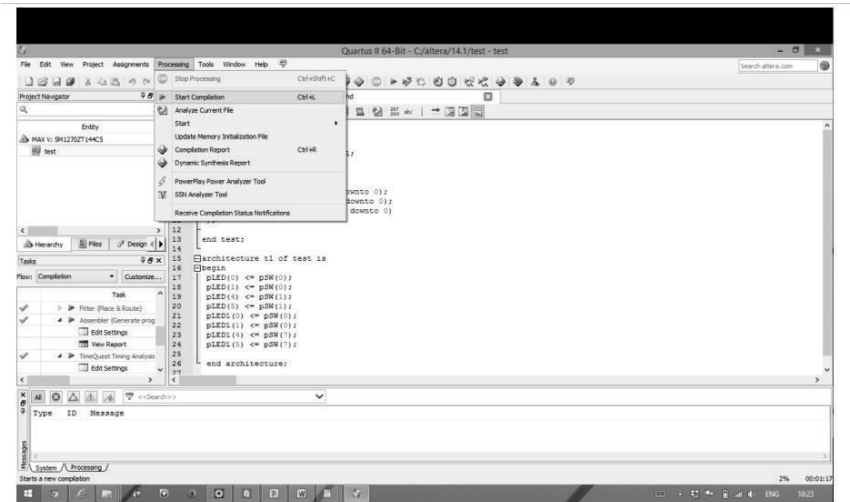


Figure 9.12 Project compilation

Step 3. Assignment Pins in Pin Planner (fig. 9.13). Information about pins you can get from GOLDi documentation for Rapid Prototype Board (fig. 9.14).

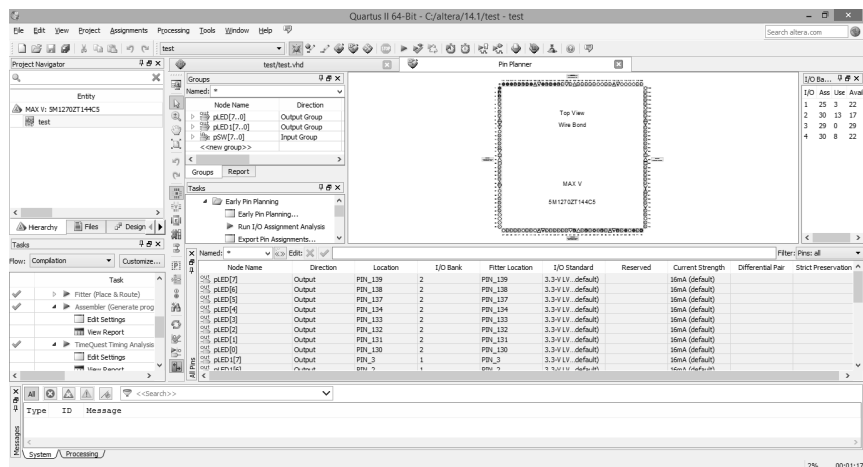


Figure 9.13 Pin Planner for Quartus II

Step 4. Recompiling the program and get .pof file for future programming board rapid prototyping.

The team responsible for the development and testing of software for embedded systems (Fig. 9.15) have developed materials for use in the international project ICo-op. Let's consider the examples of tasks for functional testing.

Task 1

Develop test cases and modify project test1 according to the task to change the number of functional switch and the number of IEDs that light up according to the number equal to the remainder of the division to give birth to 8.

Test case 1

Unique identifier – test1.

Short description – far right switch sw [0] will turn on / off the far right LED ld [0].

Name	Input/Output	Pin
Hex0[2]	Output	PIN 85
Hex0[1]	Output	PIN 86
Hex0[0]	Output	PIN 87
Hex1[7]	Output	PIN 96
Hex1[6]	Output	PIN 102
Hex1[5]	Output	PIN 103
Hex1[4]	Output	PIN 104
Hex1[3]	Output	PIN 105
Hex1[2]	Output	PIN 97
Hex1[1]	Output	PIN 98
Hex1[0]	Output	PIN 101
Hex2[7]	Output	PIN 108
Hex2[8]	Output	PIN 112
Hex2[5]	Output	PIN 113
Hex2[4]	Output	PIN 114
Hex2[3]	Output	PIN 117
Hex2[2]	Output	PIN 109
Hex2[1]	Output	PIN 110
Hex2[0]	Output	PIN 111
Hex3[7]	Output	PIN 118
Hex3[6]	Output	PIN 122
Hex3[5]	Output	PIN 123
Hex3[4]	Output	PIN 124
Hex3[3]	Output	PIN 125
Hex3[2]	Output	PIN 119
Hex3[1]	Output	PIN 120
Hex3[0]	Output	PIN 121
HexCode0[3]	Input	PIN 52
HexCode0[2]	Input	PIN 50
HexCode0[1]	Input	PIN 51
HexCode0[0]	Input	PIN 49
HexCode1[3]	Input	PIN 48
HexCode1[2]	Input	PIN 44
HexCode1[1]	Input	PIN 45
HexCode1[0]	Input	PIN 43
Incremental[1]	Input	PIN 80
Incremental[0]	Input	PIN 81
Switch[7]	Input	PIN 55
Switch[6]	Input	PIN 58
Switch[5]	Input	PIN 60
Switch[4]	Input	PIN 62
Switch[3]	Input	PIN 66
Switch[2]	Input	PIN 68
Switch[1]	Input	PIN 70
Switch[0]	Input	PIN 72

Name	Input/Output	Pin
Bargraph0[7]	Output	PIN 139
Bargraph0[6]	Output	PIN 138
Bargraph0[5]	Output	PIN 137
Bargraph0[4]	Output	PIN 134
Bargraph0[3]	Output	PIN 133
Bargraph0[2]	Output	PIN 132
Bargraph0[1]	Output	PIN 131
Bargraph0[0]	Output	PIN 130
Bargraph1[7]	Output	PIN 3
Bargraph1[6]	Output	PIN 2
Bargraph1[5]	Output	PIN 1
Bargraph1[4]	Output	PIN 144
Bargraph1[3]	Output	PIN 143
Bargraph1[2]	Output	PIN 142
Bargraph1[1]	Output	PIN 141
Bargraph1[0]	Output	PIN 140
Button[7]	Input	PIN 53
Button[6]	Input	PIN 57
Button[5]	Input	PIN 59
Button[4]	Input	PIN 61
Button[3]	Input	PIN 63
Button[2]	Input	PIN 67
Button[1]	Input	PIN 69
Button[0]	Input	PIN 71
Buzzer	Output	PIN 4
Clock	Input	PIN 18
Frequency	Input	PIN 20
FTDI nCTS	Output	PIN 5
FTDI nRTS	Input	PIN 6
FTDI RXD	Output	PIN 7
FTDI TXD	Input	PIN 8
GPIO[11]	Output	PIN 127
GPIO[10]	Output	PIN 73
GPIO[9]	Output	PIN 74
GPIO[8]	Output	PIN 75
GPIO[7]	Output	PIN 76
GPIO[6]	Output	PIN 77
GPIO[5]	Output	PIN 79
GPIO[4]	Output	PIN 94
GPIO[3]	Output	PIN 95
GPIO[2]	Output	PIN 106
GPIO[1]	Output	PIN 107
GPIO[0]	Output	PIN 129
Hex0[7]	Output	PIN 84
Hex0[6]	Output	PIN 88
Hex0[5]	Output	PIN 89
Hex0[4]	Output	PIN 91
Hex0[3]	Output	PIN 93

Figure 9.14 Pin assignment MAX® V – 5M1270Z CPLD



Figure 9.15 Information Systems Reliability Team

The execution – enable fee, turn right switch.

Requirements – test loaded on board, the board is connected to the computer.

Criterion completion – switches off and the right LED lights up right at the off – is off.

Category testing – testing system components fees.

Author – Ivanov II

Automated – yes.

Listing 9.1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity test is
port (
    pSW: in std_logic_vector (7 downto 0);
    pLED: out std_logic_vector(7 downto 0);
    pLED1: out std_logic_vector(7 downto 0)
);
end test;
architecture t1 of test is
begin
    pLED(0) <= pSW(0);
    pLED(1) <= pSW(0);
    pLED(4) <= pSW(1);
    pLED(5) <= pSW(1);
    pLED1(0) <= pSW(0);
    pLED1(1) <= pSW(0);
    pLED1(4) <= pSW(7);
    pLED1(5) <= pSW(7);
end architecture;
```

Task 2

Develop test cases and modify project test2 according to the task.

Test case 2

Unique identifier – test2.

Short description – switching on the extreme right switch sw [0] will bring to room DISPLAY option.

The execution – enable free turn-on of the right switch.

Requirements – test is loaded on board, the board is connected to the computer.

The criterion of completion – ‘when the right switches turn on 7 segment displays show the user number when switched off – is off.

Category – testing of the system component elements.

Author – Ivanov II

Automated – yes.

Listing 9.2

```
library IEEE;

use IEEE.STD_LOGIC_1164.all;

use ieee.std_logic_arith.all;

use ieee.std_logic_unsigned.all;
entity btn_test is
port(
  pbtn: in STD_LOGIC_VECTOR( 7 downto 0);
  pHex0 : out      STD_LOGIC_VECTOR( 7 downto 0);
  pHex1 : out      STD_LOGIC_VECTOR( 7 downto 0);
  pHex2 : out      STD_LOGIC_VECTOR( 7 downto 0);
  pHex3 : out      STD_LOGIC_VECTOR( 7 downto 0)
);
end btn_test;
architecture numbers of btn_test is
begin
  if (pbtn = "000000001") then
    pHex0<="11110000";
    phex1 <="00000000";
```

```

phex2 <="00000000";
phex3 <="00000000";
elsif (pbtn="00000010") then
  pHex1<="11110000";
phex0 <="00000000";
phex2 <="00000000";
phex3 <="00000000";
end if;
end numbers;

```

9.4. Individual tasks

For each task you should develop test case, develop VHDL code for the test and run the test within online laboratory GOLDi. (For guest access you can use user: GOLDi, password:goldi).

1. Build a model of the 3 axes portal on schedule in Fig. 9.7.
2. Expand the test sequence and use them to cover all transitions design model 3 axle portal.
3. Develop a simple criterion for test coverage for 3 axle portal.
4. Expand the test sequence and use them to cover all transitions model elevator.
5. Develop a simple criterion the test coverage for the model elevator.
6. Expand the test sequence and use them to cover all transitions model conveyor line.
7. Develop simple criterion test coverage for a conveyor line.

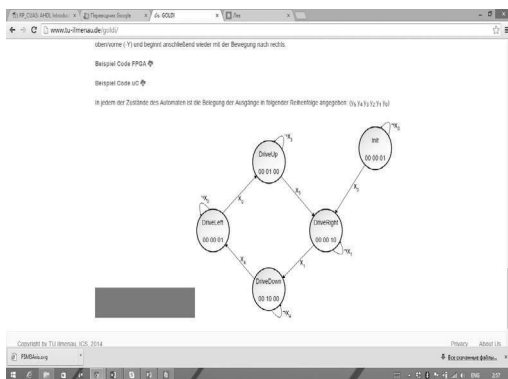


Figure 9.7 – Model of 3-D Axil portal functioning

9.5. Control questions

1. What is Model-based testing?
2. What is the difference between Mealy and Moor Automata?
3. What coverage criteria can be used?
4. What is the structure of VHDL file?
5. Give short description of functional testing.
6. What approaches allow automating functional testing?
7. How to upload testing software on the FPGA?
8. What regime modes are possible in RL GOLDi?
9. What are the possibilities of the can bus?
10. What software can be used for PLD programming?

LITERATURE FOR PART 3

1. Arras, P. E-learning concept for the properties of materials remote study/ Arras, P. ; Tabunshchyk, G. ; Kozik, T.// IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS), 2013 Volume: 02, 2013, P3: 742 – 747. DOI: 10.1109/IDAACS.2013.6663024
2. Arras, P. E-learning environment for the remote study in material properties courses/ Arras, P., Kolot, Y., Galyna, T., Kozik, T // International Journal of Computing, 12 (3), 2014, 233-238.
3. P. Arras, Architectural Characteristics and Educational Possibilities of the Remote Laboratory in Materials Properties/ P. Arras, G. Tabunshchyk, Ye. Kolot, B. Tanghe// REV2014 Conference 26 – 28 February 2014, Porto, Portugal, PP. 94-97
4. Arras, P. Iterative Pattern for the Embedding of Remote Laboratories in the Educational Process / P. Arras, K. Henke , G.Tabunshchyk, D. V.Merode // 12th International Conference on Remote Engineering and Virtual Instrumentation (REV 2015) 25-28 February 2015, Bangkok, Thailand, PP. 52-55 (DOI 978-1-4799-7838-0/15)
5. Henke K.,: Using Interactive Hybrid Online Labs for Rapid

- Prototyping of Digital Systems/ G. Tabunshchuk, H-D Wuttke, St. Ostendorff, T. Vietzke //REV2014 Conference 26 – 28 February 2014, Porto, Portugal, PP.48-59
6. Henke K.,: Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems/G. Tabunshchuk, H-D Wuttke, St. Ostendorff, T. Vietzke // iJOE, Volume 10, Issue 5, 2014, -PP. 57-62
7. Kozik T., Techniques and tools for virtual and remote experiments/ Kozik T., Arras P., Tabunshchuk G. // Suchasni problemy i dosiahnennia v haluzi radiotekhniki, telekomunikatsii ta informatsiinykh tekhnolohii: zb. tez dop. VII mizhnar. nauk.-prakt. konf. (Zaporizhzhia, 17–19 veresnia 2014 r.). – Zaporizhzhia: ZNTU, 2014. – PPa. 112-113.
8. “JTAG Boundary – Testing scanning for MAX V пристроїв” <http://www.altera.com/literature/hb/max-v/mv51008.pdf>
9. JTAG Boundary – Testing scanning for Cyclone IV пристроїв <http://www.altera.com/literature/hb/cyclone-iv/cyiv-51010.pdf>
10. Modelling, diagnostics and testing of digital systems [Available electronically]/ INTUIT. Access mode: <http://www.intuit.ru/studies/courses/3440/682/info> [RU]
11. Yu. Skobtsov . Logical modelling and testing of digital systems /Yu. Skobtsov,. V. Skobtsov.- Donetsk:IPMM NASU, DonNTU, 2005.- 436c [RU]
12. Bragina T. Informational technology for risk-oriented assessment of web-oriented system functionality /T. Bragina, G. Tabunshchuk // Systemy obrobky informatsii. Vyp.2 (118). – 2014.– С. 245-252. (In Russian)
13. Bragina T. Risk-oriented method for integrated database testing / Bragina T. I., Tabunshchuk G. V.// «Electrotechnical and computer systems». – Odessa, 2014. – №13 (89). – С. 223-230. (In Russian)
14. Tabunshchuk G.V. Software quality engineering: students textbook/ Tabunshchuk G. V., Kudermetov R. K., Bragina T. I. ; Zaporiz. nats. tekhn. un-t. – Zaporizhzhia : Dyke pole, 2013. – 173 s. (In Ukrainian)
15. Tabunshchuk, H.V. Perspektyvy vprovadzhennia virtualnoi inzhenerii u navchalnyi protses ZNTU/ H.V. Tabunshchuk, A.V. Parkhomenko //Tyzhden nauky – 2014: zb. tez dop. shchorich. nauk.-prakt. konf.

- vykladachiv, naukovtsiv, molodykh uchenykh, aspirantiv, studentiv ZNTU (Zaporizhzhia, 20–25 kvit. 2014 r.). – Zaporizhzhia: ZNTU, 2014. (In Ukrainian)
16. Prytula A.V. Praktychno-orientovani metody vykladannia v haluzi vbudovanykh system/ Prytula A.V., Parkhomenko A.V., Tabunshchik G.V // Suchasni problemy i dosiahnennia v haluzi radiotekhniky, telekomunikatsii ta informatsiinykh tekhnolohii: zb. tez dop. VII mizhnar. nauk.-prakt. konf. (Zaporizhzhia, 17–19 veresnia 2014 r.). – Zaporizhzhia: ZNTU, 2014. – P. 216 (In Ukrainian)
 17. Shcherbak N.V. Educational experiment for interactive environment / Shcherbak N.V., Tabunshchik G.V // Suchasni problemy i dosiahnennia v haluzi radiotekhniky, telekomunikatsii ta informatsiinykh tekhnolohii: zb. tez dop. VII mizhnar. nauk.-prakt. konf. (Zaporizhzhia, 17–19 veresnia 2014 r.). – Zaporizhzhia: ZNTU, 2014. – PP.254-255 (In Russian)
 18. Tabunshchik G.V. Novi tekhnolohii v pidhotovtsi fakhivtsiv z vbudovanykh system / G.V. Tabunshchik, A.V. Parkhomenko // INTERNET-OSVITA-NAUKA-2014: tezy dop. 19-oi mizhnar. nauk.-prakt. konf., ION–2014, zhovten 14–17, 2014. – Vinnytsia: VNTU, 2014. – PP.248-250 (In Ukrainian)
 19. Bragina T. Testing methods library for applications with web-based interfaces / T. Bragina, G. Tabunshchik, D. Moroka // Central European Researchers Journal. – 2015. Vol. 2. -pp.90-94
 20. IEEE-829-2008. Standard for Software and System Test Documentation. 2008. 150p.
 21. Henke K. The Remote Lab Cloud “goldi-labs.net” /K. Henke, T. Vietzke, R. Hutschenreuter, H.-D. Wuttke // Proc. of XIII Int. Conf. on Remote Engineering and Virtual Instrumentation (REV2016) (24–26 February, 2016, Madrid, Spain) UNED: – pp. 31-36

ALPHABETICAL-SUBJECT INDEX

A

Actuators, 108
Altera, 74
Altera Hardware Description Languages (AHDL), 91, 92
Altium Designer, 54
Analogue input modules, 111
Analysis
 - static, 167
 - dynamical, 170
Arduino, 45, 56
Application-specific integrated circuit (ASIC), 44
Atmel Studio, 56
Audit, 167

B

Bit instructions, 132

C

Chassis, 104
Checking
 - consistency, 171
 - model, 170
Conjunctive normal form (CNF), 151
CoDesign, 39
Concurrent Object Modeling and Architectural Design Method (COMET), 137
Configuration word, 108
Control automata (CA), 115
Controller tags, 120
Controlnet, 105
Control systems, 102
Complex Programmable Logic Device (CPLD), 83, 90
Creo, 57

D

Design
 - flow, 76, 77
 - simulation, 79
Design methods
 - text based 77, 78
 - graphically based 77, 78
 - FSM based 77, 79

Digital

- modules, 108
- simulation, 11

Display, 66, 79, 80

E

Electronic Computer-Aided Design (ECAD), 41, 54

Embedded System

- area of application, 26
- concept for design, 38
- hardware realization, 44
- life cycle, 28
- programming languages, 50
- requirements, 27
- software realization, 49
- term, 26, 179

Engineering units, 112

Proportional count, 112

Environment

- integrated development, 45, 76

Errors, 120, 182

Ethernet, 105

F

Failures, 106, 182

Family Controllogix, 113

Faults, 102, 182, 184

Field-programmable gate array (FPGA), 44,74

Finite State Machine (FSM), 138

Fixed controllers, 102

Flow control program instruction, 135

FSM with a fixed changing of states, 143

Function Block Diagram, 123

Functional element, 176

G

Grid of Online Lab Devices Ilmenau (GOLDI), 13, 204

H

Hardware, 106

Hardware Realization, 44

Hardware-Software

- CoDesign, 39
- platform, 39, 44

I

IEC 61131-3, 122
Industrial automation systems, 129
Industrial controllers, 102
Industrial networks, 105
Input and output OA, 115
Input, output module, 107
Inspection, 169
Instruction List, 127
Instructions of bit string shift, 136
Instructions counters, 132
Instructions of comparison, 133
Instructions timers, 132
Integrated Development Environment (IDE), 45,76
Intelligent modules, 108

J

Joint Test Action Group (JTAG), 188

L

Laboratory
 - GOLDI, 13, 206
 - RELDES, 58
 - remote, 12
 - virtual, 12
Ladder Diagram, 132
Levels of controller control, 115
Logic output instructions, 133

M

Main program, 119
Man-machine interface, 156
Mealy machine, 117
Microcontroller unit (MCU), 44, 48
Model-Driven Development, 172, 210
Modular controllers, 103
Moore machines, 117

O

Operational automata (OA), 115
Operator interface, 156

P

Periodic task, 120
Platform, 39, 44
Programmable Logic Controller (PLC), 102
Programming languages, 122
Program scan, 119
Proportional Integral Derivative (PID) instructions, 136
PROTEUS VSM, 54

Q

Quality

- assurance, 166
- planning, 166
- control, 166, 211

Quartus II, 80, 214

R

Racks, 103
Rapid Prototyping Board

- main components, 74,80
- operation modes, 74
- stand alone, 90, 213
- web-based, 90, 91, 214

Raspberry Pi 46,49
Real-time operation system (RTOS), 50
REmote Laboratory for Design of Embedded Systems (RELDES)

- API, 72
- description, 58
- development team, 72
- set of remote experiments, 60
- use case diagram 72, 73

Remote

- laboratory, 12, 205
- experiment
 - Traffic Lights, 60
 - Servomotor, 64
 - LCD display, 66
 - Sonic distance sensor, 68

Remove In Up Power (RIUP), 110
Requirements

- analysis, 28
- creation methodology, 32

- type, 29
- management tools, 34
- model, 30,31,32
- level, 30

Reuse, 41

Rich Internet Application (RIA), 74

Rung,122

S

Scan, 119

- boundary, 189

Sensors, 108

Sequencer instructions, 136

Sequential Function Chart,123

Simulation 83, 87

Sink type, 110

Slot, 103

Software realization, 49

Software testing, 191

Source type, 110

Stimulant generator (FGP), 152

Structured Text, 123

Switch-technology, 52

System

- design 28
- requirements 29

System transitions, 173

T

Tag, 120

Technical review 168

Test, 179

Test-case, 179

Testing

- acceptance, 191
- configuration, 192
- integration, 191
- performance, 192, 213
- property-based, 170
- regression, 193, 195
- sanity, 192
- system, 191

- smoke, 193
- unit, 191
- usability, 192
- volume, 192

Time chart, 41,43

U

Unified Modeling Language (UML), 209

V

Validation, 167

Verification, 167

- dynamic, 172
- formal, 169

Verilog HDL, 91

Very high speed integrated circuits Hardware Description Language (VHDL), 91

Virtual

- design, 10
- engineering, 10
- laboratory, 12
- manufacturing, 10
- plant, 11
- prototyping, 11
- simulation, 41

W

Walkthrough, 168

Web-based

- programming, 76
- rapid prototyping, 74
- user interface, 76
- validation, 74

ABOUT AUTHORS



Karsten HENKE
Dr.-Ing., Integrated Communication Systems Group , TU
Ilmenau, Germany
karsten.henke@tu-ilmenau.de

Honor with the Lehrpreis der TU Ilmenau 2010 in the category
staff member

Executive Committee Member of the Global Online Laboratory
Consortium (GOLC)

Steering Committee or Program Committe Member of:

- IEEE Engineering Education 2010 – The Future of Global Learning in Engineering Education (IEEE EDUCON2010)
- Annual International Conference on Remote Engineering and Virtual Instrumentation (REV)
- Annual International Conference on Interactive Mobile and Computer Aided Learning (IMCL)
- Annual International KES Conference on Smart Technology based Education and Training>International KES Conference on Smart Technology based Education and Training (KES STET)
- Annual International Conference on Computers and Advanced Technology in Education (CATE)
- Annual International Conference on Interactive Computer Aided Blended Learning (ICBL)

Editorial Board Member of:

- International Journal of Online Engineering (iJOE, IAOE)
- Advanced Technology for Learning (ATL, Acta Press)



Anzhelika V. PARKHOMENKO, Ph.D., Associate professor of Software Tools Department of Zaporizhzhya National Technical University. A graduate of Zaporizhzhya Machine-Building Institute named V.Ya.Chubarya. Defended the PhD thesis on topic «Development of complex models of microelectronic equipment components for CAD». Author of over 100 scientific publications and educational works, including one textbook approved by Ministry of Education and Science of Ukraine, 2 Certificates for invention. Main research fields: Embedded Systems for moving objects control; technologies and systems of virtual and remote engineering; CAD/CAM/CAE-system.



Olga M. GLADKOVA, a postgraduate student of Software Tools Department of Zaporizhzhya National Technical University. She has a master's degree in the specialty «Software systems». The author of 18 scientific publications, 1 Certificate for invention, three educational works. Main research fields: CAD/CAM/CAE systems; Embedded Systems for moving objects control; Internet of Things; technologies and systems of virtual and remote engineering;



Mykhailo A. POLIAKOV associate professor, PhD. , Assistant professor of electrical and electronic devices ZNTU. After graduating in 1974 ZNTU worked developer of control systems at the Research Institute «Mars» in Russia. Scientific – teaching experience – 40 years in universities Kiev, Ulyanovsk, Zaporozhe, Dnepropetrovsk, author of over 100 scientific works. Research interests: design of control systems with controllers, monitoring and forecasting of technical states of transformers.



Tetiana LARIONOVA – teaching assistant at department of Electrical and electronic devices, postgraduate student at the Zaporozhye National Technical University (ZNTU). Graduated from ZNTU with a master degree in “Electrical machines and apparatuses” in 2011, has 7 scientific papers. Scientific interests: design of intelligent control systems with controllers, development of DC-DC converters, power efficiency of electricity supply systems.



Galyna TABUNSHCHYK – PhD, Prof of Software Tool Department of Zaporizhzhya National Technical University. Graduated from Zaporizhzhya National Technical University with speciality Software Engineering, in 2004 finished PhD work in control systems and process. Have more than 100 scientific works. Scientific interests – Software Engineering, System Verification, Embedded Systems, Risk Management. E-mail: galina.tabunshchik@gmail.com



Tetiana KAPLIHENKO, Ph.D., an associate professor of Software Tools Department at Zaporizhzhya National Technical University. She has a master's degree in "Software Engineering". She finished PhD study in "Information technology", and defended thesis «Information technology for dynamic planning and monitoring the development process of web-oriented information systems» in 2015. She is the author of 33 scientific publications, 3 Certificates for invention and 1 patent. Main research fields: the analysis and verification for the software quality; software design managements; risks management. E-mail: tabr007@yandex.ua

APPENDIX. THE REMOTE LAB CLOUD “GOLDI-LABS.NET”

***Karsten Henke, Tobias Vietzke, René Hutschenreuter,
Heinz-Dietrich Wuttke***

Integrated Communication Systems Group
Technische Universität Ilmenau
Ilmenau, Germany [karsten.henke|tobias.vietzke|rene.hutschenreuter|dieter.
wuttke]@tu-ilmenau.de

Abstract – After a short discussion of the GOLDi remote lab infrastructure, the new experiment control panel (ECP) and the integrated Graphical Interactive Finite State Machine toolset (GIFT) main focus of the paper is the interconnection of multiple remote labs (based on the GOLDi infrastructure) to a remote lab cloud. Within two running TEMPUS projects ten remote labs in four countries can be used worldwide. Based on this cloud structure it is necessary to adapt the complex GOLDi access control system. With this new reservation system possibilities for an effective worldwide experiment management will be available, which will be described in the article.

Keywords – *control engineering education; laboratories; Web-based education; virtual and remote labs; Web-based design tools; distance learning.*

I. INTRODUCTION

Our interactive hybrid online lab – called GOLDi (Grid of Online Lab Devices Ilmenau) – stands for a grid concept to realize a universal remote lab infrastructure (see Figure 1).

The remote lab is used for teaching practical lessons as well as to give hands-on experiences for the development of embedded electronics. This is not done using on-site lessons, but remotely via the Internet. This has the advantage that courses can be offered internationally world-wide and gives students from different countries, speaking different languages, the same access and equal possibilities in the lab. Additionally, even for local students the lab offers extended opening hours (twenty-four-seven) when compared to a regular lab. Besides the advantages for students, this also reduces the costs for academic teaching and improves the quality by offering more practical training possibilities.



Fig. 1. The official GOLDi website (www.goldi-labs.net)

This gives students the chance to realize correct designs, to organize self-study process of the student more efficiently, to control student's work and to broaden the ways of communication in research work with companies as well.

With our hybrid¹ online lab we support the design process of digital control systems which usually consists of the conceptual formulation and the design of the control algorithm to finally achieve a validated control. For the functional description we offer different specification techniques by using different control units:

- a finite state machine (FSM) based design on the basis of digital automata,
- a software-oriented design in C or assembler by using microcontrollers,
- a hardware-oriented design in hardware description languages or schematic block design by using FPGAs.

Simulation and visual prototyping help to find functional errors before starting practical work on real physical systems (the electro-mechanical models) in the lab room with one of the selected control units.

With our GOLDi remote lab we offer the students a working environment that is as close as possible to a real world laboratory. Under

¹ Hybrid online labs provide both remote experiments on real electro-mechanical models (physical systems) in the remote lab as well as simulation models of these physical systems in virtual labs.

real laboratory conditions disturbances can appear and lead to failures of the control algorithm that cannot be detected under virtual lab conditions. Therefore, it is important to include such real disruptive factors for a closer relation to practical conditions. Furthermore, the lab infrastructure should offer the students stimulus with regards to the design of safety critical control systems. By the conception of our lab, the virtual worlds were embedded within a real laboratory.

II. GOLDI ARCHITECTURE

A. Serversideinfrastructure

This GOLDi infrastructure, shown in Figure 2, consists on the server side of three parts:

- an internal serial **remote lab bus**,
- a **bus protection unit** (BPU) to interface the selected control unit with the remote lab bus and to protect the remote lab bus from misuse and damaging as well as
- a **physical system protection unit** (PSPU), which protects the physical systems (the electro-mechanical models in the remote lab) against deliberate damage or accidentally wrong control commands and which offers different access and control mechanisms.

The interconnection between the web-control units and the selected physical hardware model during a remote lab work session (experiment) as well as the user management is done by the remote lab server, which also handles the webcams.

By using the ECP, the student is able to:

- upload the synthesized/compiled designs to the corresponding Web-control units (e.g. microcontroller, FPGA) in the lab room,
- handle the experiment (e.g., start, stop, reset),
- use the interactive debugging features (break on sensor/actuator changes or special conditions),
- single step processing, by pausing the execution on every sensor/actor change,
- change environmental variables if necessary and watch the experiment by manipulating environmental variables inside an

I/O monitor (Figure 4 shows an example to monitor the sensor signals of the 3-axis portal) or by observing the control of the physical system directly via a webcam.

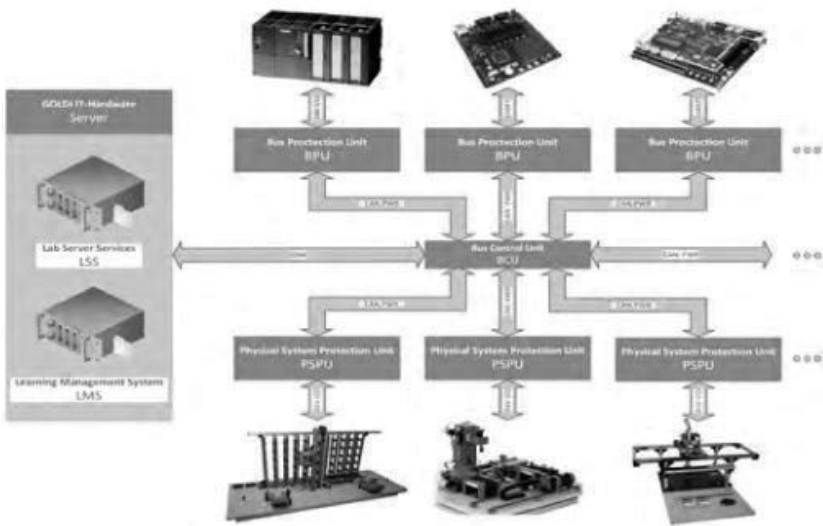


Fig. 2. GOLDi internal architecture

A detailed description of the whole GOLDi architecture as well as different working modes can be found in [1-4].

B. Client side infrastructure

On client side (student’s PC) GOLDi offers a Web-based environment – the *experiment control panel* (ECP) – supporting the above mentioned features to generate and execute a design by using simulation models. Figure 3 gives an impression for the 3-axis portal physical system.

The increasing capacity of wireless communication and the growing number of mobile devices (e.g. smartphones and tablets) on the one hand as well as modern Internet technologies like JavaScript, HTML5 and Web Sockets on the other hand provides new possibilities and challenges in the area of mobile learning. Therefore a realization as HTML5 application was chosen for the Web-interface.

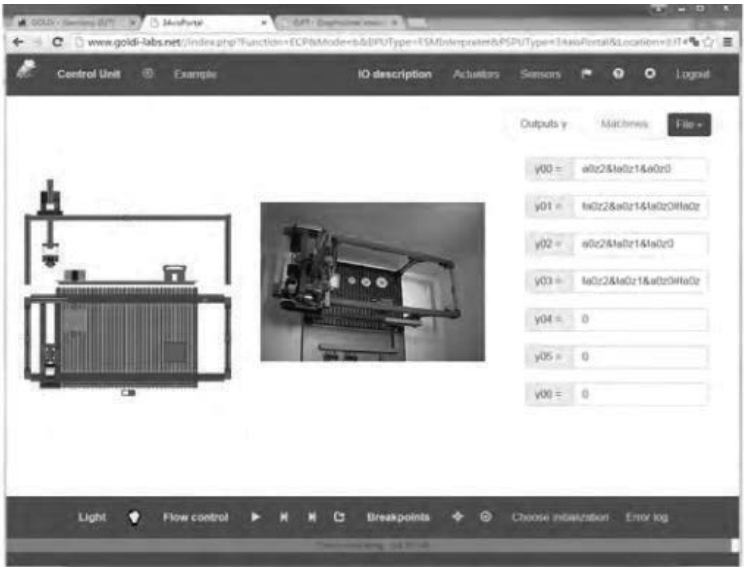


Fig. 3. GOLDi Experiment Control Panel for the 3-axis portal

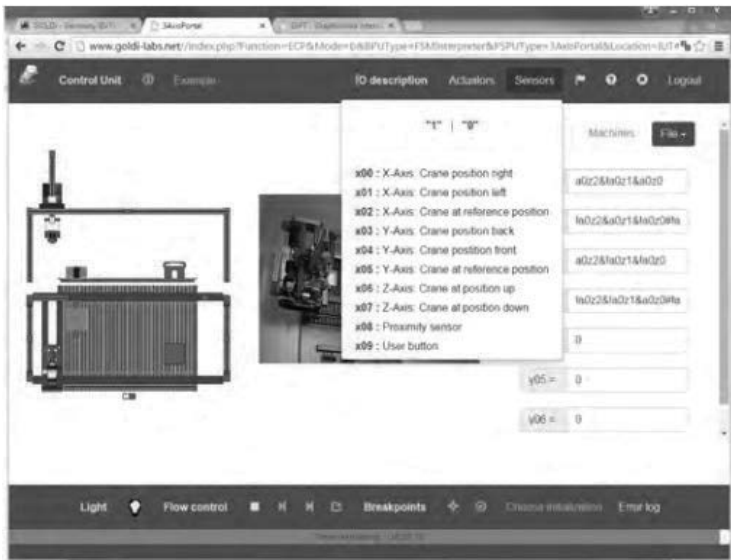


Fig. 4. I/O monitor (e.g. sensor signals) of the ECP for the 3-axis portal

Beside the possibility to bring the physical system in a predefined initial state the user is also able to choose an individual initial situation for his experiment. For that he can manipulate the physical system via mouse or keypad. Figure 5 shows an example to move the trolley of the 3-axis portal over the worktable to pick up a workpiece via the electromagnet.

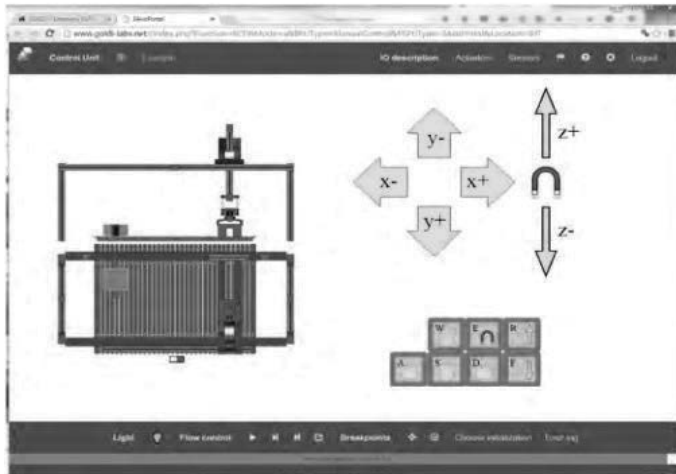


Fig. 5. ECP for manual control to select an individual initial situation

C. GIFT – Graphical Interactive Finite State Machine Toolset

For the design of finite state machines (FSM) a graphical interactive FSM toolset (called GIFT) can be used directly within the GOLDi infrastructure which provides methods for

- general FSM administration,
- input the FSM as automaton graph or transition table,
- handling of edge weights and coding of nodes,
- simulation of single and parallel automata,
- generation of the next-state (z) and output (y) equations,
- export of z/y equations to the ECP of the GOLDi system,
- generation equations for D and JK flip-flops.
-

As an example we will discuss a FSM design task for students in the second semester by using the 3-Axis portal:

“On the spindle of a portal crane, a tool carriage can be moved to the right and to the left. Limit switches provide information on the left end position (x_l) as well as the right end position (x_r) of the tool carriage (x_r, x_l).

The motion can be controlled via the binary control outputs (y_l, y_r) between

- *motiontotheleft*($y_l=1, y_r=0$),
- *motiontotheright*($y_l=0, y_r=1$) and
- *stop*($y_l=y_r=0$).

An additional input variable x_s signalizes

- *stop motion* ($x_s = 0$) or
- *movement* ($x_s = 1$) to the left or right.

After a possible break, you should continue the movement in the original direction.”

Such kinds of tasks are solved by developing a formal description of the control algorithm on the basis of an automaton graph and the corresponding Boolean equations. Figure 6 shows the corresponding automaton graph by using the graph editor of the GIFT system.

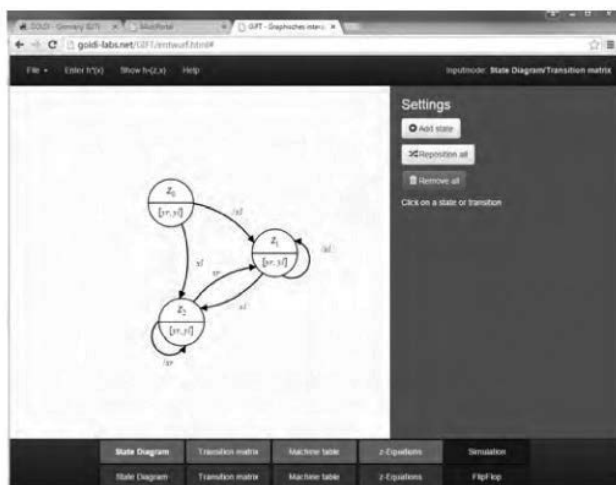


Fig. 6. Automaton graph editor of the integrated GIFT system

After that, the student can proceed to the simulation process using graphical controls. That leads to appropriate state transitions caused by the changed set of input variables. Via waveform simulation (shown in Figure 7) the temporal progression of input and output variables and the internal system variables (states) of partial automata can be shown.

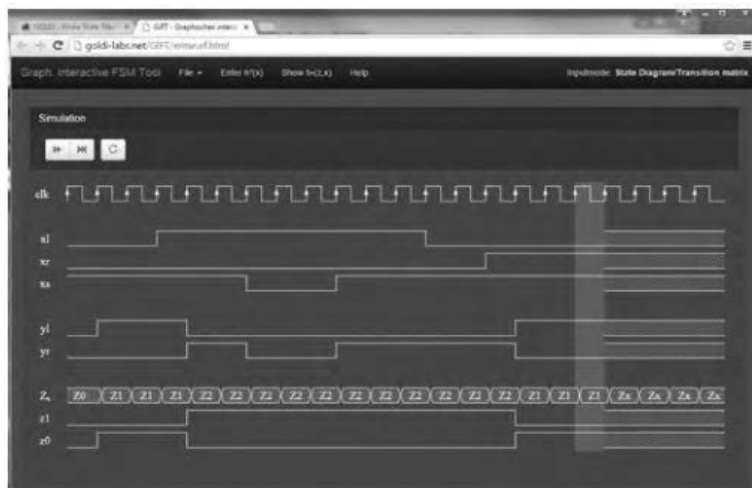


Fig. 7. Simulation tool of the integrated GIFT system

Most of the students are highly motivated in realizing such tasks in a real laboratory because they can see the results of their design immediately. That's why the GIFT system offers the possibility to export the generated next-state and output equations directly to the ECP of the GOLDi system (Figure 8).



Fig. 8. ECP export functionality of the GIFT system

Because the design is typically made with user-defined variables (e.g. x_l , y_r) the student has to adapt them to the real sensor and actuator interface notations (e.g. x_r must be replaced by $x00$ of the 3-Axis portal sensor signals, which means X-Axis: Crane position right). This can be done interactively within the ECP (see Figure 9).

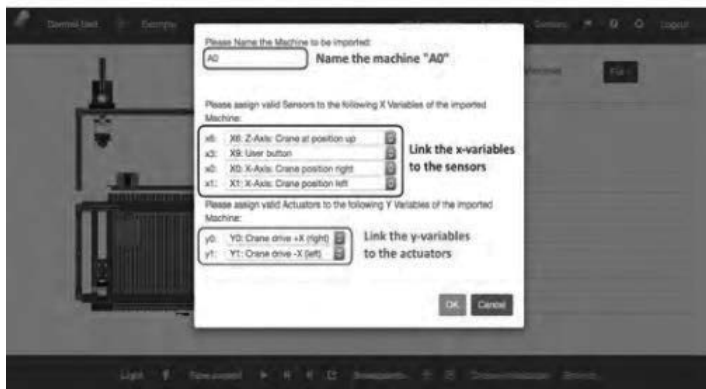


Fig. 9. GIFT import functionality of the ECP of the GOLDi system

III. GOLDI EXPERIMENT CONFIGURATION

To perform a remote experiment the first step is the user login. The GOLDi infrastructure supports the following authentication mechanisms:

- login via the university LDAP user account,
- login via the eduroam login mechanism (will be implemented soon),
- login after an individual registration (with freely selectable username and password).

The next step is the experiment configuration. Based on the flexible grid structure of the GOLDi system an experiment consists of two components: on the one hand there are various control units (e.g. FSM, microcontroller, FPGA). On the other hand there are the electromechanical physical systems (e.g. elevator, 3-axis model or warehouse). Each real physical system can also be used as simulated virtual model. Figure 10 shows an example for the experiment configuration with available control units as well as physical systems.

B. Microcontroller based experiment configuration

For a software-oriented implementation students can implement their control algorithm directly into a microcontroller. Therefore, they use common (non- commercial) development tools, for example the Atmel Studio from Atmel [5], to develop Assembly- and/or C-coded software projects for the Atmel microcontroller in the remote lab. After compilation, the generated software control algorithm can be uploaded via the ECP interface to the remote lab, where the hex code is programmed into the microcontroller automatically in the background (see Figure 7).



Fig. 10. GOLDi experiment configuration

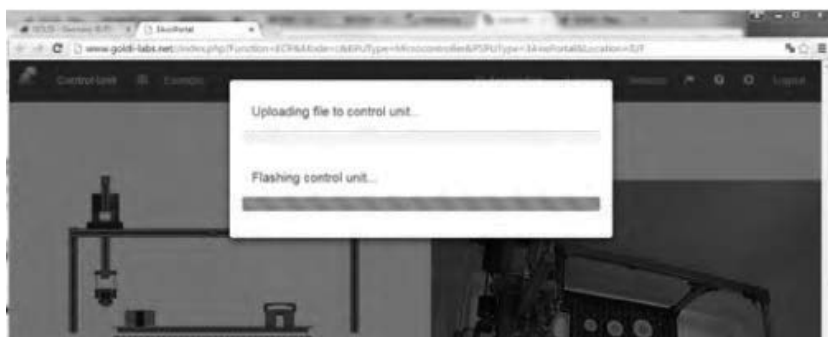


Fig. 11. Programming the microcontroller in the remote lab via the ECP

Now, the student can begin with his experiment, to check if his algorithm fulfills the requirements of the given control task.

C. Microcontroller based experiment configuration

If a student prefers an more hardware-oriented design using an FPGA and applying a hardware description language like VHDL as specification technique, he can prepare his design with common development tools, for example Quartus II [6] or others. The generated bit file is uploaded via the ECP interface to the remote lab, where the FPGA will be programmed in the background automatically. After programming the connected FPGA, the FPGA board operates as control unit for the designed control algorithm, and the student can start his experiment.

D. Watching an experiment as guest

Visitors or members of a student project team have the opportunity to observe running experiments without direct manipulation of these experiments. Figure 12 gives an example where the visitor can choose between two running experiments.

A. FSM-based experiment configuration

In case of using finite state machines (FSM) for specification, based upon an automaton graph, a student can use the described GIFT design environment of the GOLDi system. This system generates the required next state and output equations, which can be exchanged within the GOLDi infrastructure from the GIFT system directly to the corresponding ECP. In case of a selected virtual physical system the control algorithm is executed by an interpreter running inside the student's client PC. No Internet connectivity to the laboratory and the physical systems in remote lab is necessary for this experiment. This mode is especially for face-to-face lectures in a classroom where each student can work with the same physical system simultaneously.

Finally the student (but only one student) can choose the real physical system in the remote lab. In this case, the electromechanical hardware model will be controlled via the Internet "from a distance" through the

interpreter running inside the student's client PC. No additional control units in the remote lab are necessary. In this case, only the input and output signals of the physical system will be transferred via Internet.

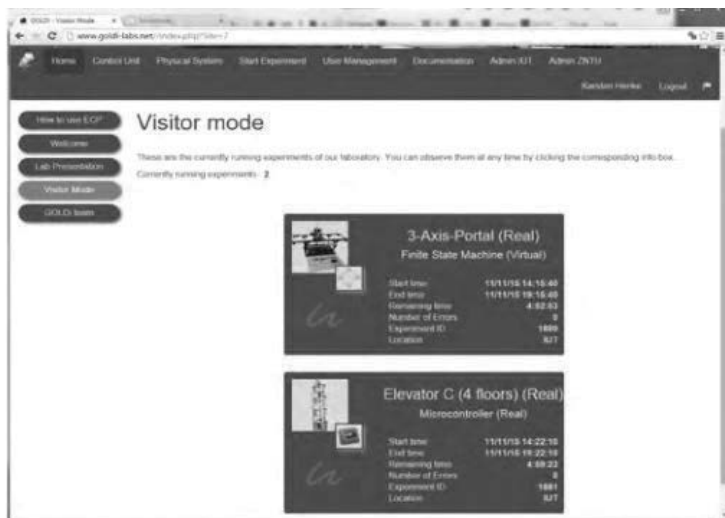


Fig. 12. GOLDi Visitor mode

IV. GOLDI LABS CLOUD

Actually the GOLDi infrastructure is used within two running TEMPUS projects: Wider objective of the ICo-op project [7] is e.g. to empower university-enterprises partnerships in Armenia, Georgia, and Ukraine by modernizing engineering education based on remote engineering and virtual instrumentation; enhanced with transversal knowledge and competences at universities and offering contemporary methods of the vocational education and training for adults in enterprises. One of the specific objectives of the DesIRE project [8] is e.g. to create remote and virtual laboratories in the area of Embedded System in Armenia, Georgia, and Ukraine for distance and e-learning.

This means, that each project partner receives a complete remote lab. Totally GOLDi remote labs are running at ten universities in Armenia, Georgia, Germany, and Ukraine. One implementation problem we have had to solve is the maintenance of each partner remote lab:

- Any new functionality or a new firmware must be implemented manually on each partner website.
- Each institution has specific modifications according their own requirements.
- Each university has different network architectures.
- Each university has different remote lab configurations.
- To have access to an experiment of the partner labs, a separate user account on the corresponding partner website is necessary.

Each GOLDi partner has to handle the maintenance itself – supported by the Ilmenau GOLDi project team. We realized very quickly that this procedure is very ineffective.

We decided to reengineer the network of all GOLDi remote labs to a GOLDi cloud system [4, 9]. Available GOLDi Servers (it means available partner remote labs) are registered in the GOLDi cloud. They are highlighted colored as ready to be used on the central GOLDi website (see Figure 13).



Fig. 13. GOLDi cloud Website

The new cloud structure (see Figure 14) has the following advantages:

- Maintenance of the whole system on one central location: www.goldi-labs.net.
- All partner universities have the same GOLDi version.
- New functionalities are immediately available for all partners.
- Usage of all partner labs with one central [goldi-labs.net](http://www.goldi-labs.net) user account.
- The user can determine which lab has the lowest delay (best reaction time) regarding his current geographic position and internet connection.
- Overview about all running experiments in all partner labs worldwide with the possibility to observe these experiments.

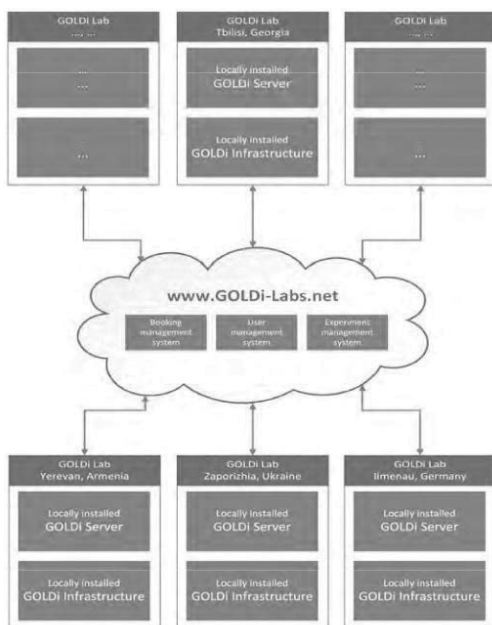


Fig. 14. GOLDi cloud architecture

Each GOLDi client communicates with the GOLDi cloud for the central user and experiment management and the booking management. All privacy-relevant data of the cloud are located in the computing department of the Ilmenau University of Technology with highest privacy-requirements and will never be exchanged with the local GOLDi Servers of the partner institutions.

With the local GOLDi Server each client exchanges only the sensor/actuator signals during a running lab experiment directly.

V. GOLDI BOOKING SYSTEM

Based on the flexible grid structure of the GOLDi system, a complex access control system must be available. The implementation challenge is the two resource-constraint- scheduling. On the one hand there are the electromechanical physical systems (e.g. elevator, 3-axis model and warehouse). Each system can also be used as simulated virtual model. On the other hand there are the control units (e.g. microcontroller, FPGA). For a specific experiment the user can combine these components, which finally means that the maximum of different experiments is the cross product of all real/virtual physical systems and the control units (see section III).

Besides this complex experiment configuration, the user can decide when he wants to start his own specific experiment. There are actually three different start options which will be described in the following.

A. Direct start mechanism

The user can start the experiment directly just-in-time. This option is only possible if the selected devices are not in use. The experiment time is limited to 30 min., but will automatically increase if no other user is waiting for the hardware (control unit / physical system).

B. Queue-based start mechanism

The queue-based start provides additional functionality compared to the direct start. After combining the experiment, the user will be appended to a special queueing system, which changes the user priorities dynamically based on the current load of the lab. It always tries to get the highest possible load.

C. Pre-reserved start mechanism

In this case the user can reserve a specific experiment in advance for a certain time slot. Then the selected hardware will be blocked during

this reserved time. Furthermore, teachers can use this feature to get an exclusive reservation for a group of students to work together on a pre-configured experiment.

VI. CONCLUSION

The Integrated Communication Systems Group at the Ilmenau University of Technology is an expert in the field of Internet-supported teaching of digital system design and is well experienced in the area of integrated hard- and software systems. Students have to pass hands-on examinations in a lab to complete the learning outcomes by own experiences. For all students, hands-on experiences are important to deepen their knowledge about topics they learned during lectures. But most of the students are highly motivated in realizing their design tasks in a real laboratory because they can see the results of their design immediately. With our interactive hybrid online lab, called GOLDi (Grid of Online Lab Devices Ilmenau), we want to offer the students a working environment that is as close as possible to a real world laboratory. Under real laboratory conditions disturbances can appear and lead to failures of the control algorithm that cannot be detected under virtual lab conditions. Facilities of hybrid online labs provide permanent online access for students and supervisors, and give possibilities to check different parts of the designs most easily. Besides the advantages for students, this also reduces the costs for academic teaching and improves the overall quality by offering more practical training options.

This gives students the chance to realize correct designs, to organize self-study process of the student more efficiently, to control student's work and to broaden the ways of communication in research work with companies as well. This solution is intended for the use in teaching materials dealing with the design of digital control systems and embedded systems – from the basics up to complex design tasks.

Within several European projects in the area of “Remote Engineering” e.g. ICo-op [7] and DesIRE [8] it is increasingly necessary to allow and organize a shared use of equipment. Therefor the main focus is a Web-wide usage of design tools and remote labs for the design of digital systems. For that, all installed GOLDi remote labs of participating universities were interconnected to the so called GOLDi cloud, described in section IV.

For an effective usage of the GOLDi infrastructure with the possibility of a flexible experiment configuration a complex booking system must be available which is described in section V.

ACKNOWLEDGMENT

The authors would like to acknowledge the work of Alexander Härtel, Tobias Fäth, René Hutschenreuter, Lisa- Marie Schilling, Nicole Ponischil, Lennartr Planz, Stephen Ahmad and Bastian Hellweg for their work within the GOLDi framework.

This work was supported in part by the European Commission within the program “Tempus”, “ICo-op – Industrial Cooperation and Creative Engineering Education based on Remote Engineering and Virtual Instrumentation”, Grant No. 530278-TEMPUS-1-2012-1-DE-TEMPUS-JPHES as well as “DesIRE – Development of Embedded System Courses with implementation of Innovative Virtual approaches for integration of Research, Education and Production in UA, GE, AM”, Grant No. 544091-TEMPUS-1-2013-1-BE- TEMPUS-JPCR.

REFERENCES

- [1] K. Henke, St. Ostendorff, St. Vogel, H.-D. Wuttke, “A Grid Concept for Reliable, Flexible and Robust Remote Engineering Laboratories”, *International Journal of Online Engineering (iJOE)*, Vol 8 (2012), pp. 42–49, Vienna, December 2012.
- [2] K. Henke, St. Ostendorff, H.-D. Wuttke, T. Vietzke, Ch. Lutze, “Fields of Applications for Hybrid Online Labs”, *International Journal of Online Engineering (iJOE)*, Vol 9 (2013); pp. 20–30, Vienna, May 2013.
- [3] K. Henke, H.-D. Wuttke, T. Vietzke, St. Ostendorff, “Using Interactive Hybrid Online Labs for Rapid Prototyping of Digital Systems”, *International Journal of Online Engineering (iJOE)*, Vol 6 (2014), pp. 57–62, Vienna, October 2014.
- [4] K. Henke, T. Vietzke, H.-D. Wuttke, St. Ostendorff, “GOLDi – Grid of Online Lab Devices Ilmenau”, *Demonstration of Online Experimentation, exp.at’15 International Conference*, São Miguel Island, Azores, Portugal, Juni 2015.
- [5] Atmel Corporation: <http://www.atmel.com>
- [6] Altera Corporation: <http://www.altera.com>
- [7] ICo-op project Website: <http://www.ICo-op.eu>
- [8] DesIRE project Website: <http://tempus-desire.thomasmore.be> [9] GOLDi-labs cloud Website: <http://goldi-labs.net>

Навчальне видання

**Parkhomenko A.V.,
Tabunshchik G.V.,
Poliakov M.O.,
Gladkova O.M.,
Kaplienko T.I.,
Larionova T. Y.**

REMOTE AND VIRTUAL TOOLS IN ENGINEERING

Textbook

(АНГЛІЙСЬКОЮ МОВОЮ)

Технічний редактор

Л. А. Рябоконь

Коректор

Н. В. Чечеко

Художник

А. П. Кондаков

Формат 60х84/16.

Папір офсетний. Гарнітура *Times*. Друк офсетний.

Підписано до друку 15.04.2016. Ум. друк. арк. 14,53.

Наклад 200 прим.

Видавництво «Дике Поле»

Україна, 69063, м. Запоріжжя, вул. Троїцька, 31-А.

Тел.: (061) 213-75-95; 213-75-05.

Свідоцтво суб'єкта видавничої справи

ЗЗ № 004 від 23.08.2001 р.