Internet of Things
for Industry and Human Applications

Internet of Things Technologies
for Cyber Physical Systems

Internet of Things
for Industry and Human Applications

Internet of Things Technologies for Cyber Physical Systems

PRACTICUM

Funded by the Erasmus+ Programme of the European Union

Ministry of Education and Science of Ukraine
Yuriy Fedkovych Chernivtsi National University
National Aerospace University "KhAI"
Zaporizhzhia National Technical University

H. I. Vorobets, V. S. Kharchenko, R. K. Kudermetov,
Ya.M. Klyatchenko, V. E. Horditsa, O. O. Pshenychnyi, I.S. Khamula,
I. M. Lobachev, M. V. Lobachev, M. Y. Tiahunova, O. V. Polska

Internet of Things for Industry and Human Applications

Internet of Things Technologies for Cyber Physical Systems

Practicum

Edited by H. I. Vorobets and V. S. Kharchenko

Project
ERASMUS+ ALIOT "Internet of Things: Emerging Curriculum for Industry and Human Applications"
(573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP)

2019

UDC 004.415/.416:004.89](076.5)=111

T38
Reviewers:
Dr. Ah-Lian Kor, Leeds Beckett University, United Kingdom
Prof., DrS. V. V. Mohor, Corresponding Member of NAS, Ukraine, Director of the G.E. Pukhov Institute of Modeling Problems in Energy

T38 Vorobets H. I., Kharchenko V. S., Kudermetov R. K., Klyatchenko Ya., M., Horditsa V. E., Pshenychnyi O. O., Khamula I. S., Lobachev I. M., Lobachev M. V., Tiahunova M. Y., Polska O. V. Internet of Things Technologies for Cyber Physical Systems: Practicum / Vorobets H. I. and Kharchenko V. S. (Eds.) – Ministry of Education and Science of Ukraine, Yuriy Fedkovych Chernivtsi National University, National Aerospace University "KhAI", Zaporizhzhia National Technical University, 2019. – 172 p.

ISBN 978-617-7361-97-7

The materials of the practical part of the study course MC4 "IoT Technologies for Cyber Physical Systems", developed in the framework of the ERASMUS+ ALIOT project "Internet of Things: Emerging Curriculum for Industry and Human Applications" (573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP).

The course structure, teaching materials, examples of tasks for seminars, practical and laboratory works, as well as methodological recommendations for self-preparation and knowledge testing in the discipline, and criteria for their assessment are given. The material is submitted sequentially t**o** form a holistic picture of the current state, synergy, prospects for research and development of Internet of Things and Cyber-Physical Systems technologies. The focus is on the conceptual issues of modeling, analysis, synthesis and practical implementation of CPS, and the role of IoT at all stages of the life cycle of complex computerized systems.

Designed for Masters of Universities in Information Technology: Computer Science and Information Systems and Technologies, Cybersecurity, Systems Analysis, Software and Computer Engineering, as well as teachers of relevant faculties, engineers and scientists involved in the development and implementation of CPS and IoT technologies.

Ref. – 81 items, figures – 51, tables – 10.

Approved by Academic Council of National Aerospace University "Kharkiv Aviation Institute" (record No 4, December 19, 2018)

ISBN 978-617-7361-97-7.

Міністерство освіти і науки України
Чернівецький національний університет імені Юрія Федьковича
Національний аерокосмічний університет «ХАІ»
Запорізький національний технічний університет

Г. І. Воробець, В. С. Харченко, Р. К. Кудерметов,
Я.М. Клятченко, В. Е. Гордіца, О. О. Пшеничний, І.С. Хамула
І. М.Лобачев, М. В.Лобачев, М. Ю.Тягунова, О. В. Польська

Інтернет речей для промисловості та гуманітарних застосувань

ТЕХНОЛОГІЇ ІНТЕРНЕТУ РЕЧЕЙ ДЛЯ КІБЕРФІЗИЧНИХ СИСТЕМ

Практикум

За редакцією Г. І. Воробця та В. С. Харченко

Проект
ERASMUS + ALIOT «Інтернет речей: нові навчальні програми для промисловості та гуманітарних застосунків»
(573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP)

2019

УДК 004.415/.416:004.89](076.5)=111

Т38

Рецензенти: Доктор Ах-Ліан Кор, Університет м. Лідса, Великобританія
Проф., д.т.н. В. В. Мохор, член-кореспондент НАН України, директор Інституту проблем моделювання в енергетиці ім. Г.Є. Пухова

Т38 Воробець Г. І., Харченко В. С., Кудерметов Р. К., Клятченко Я. М., Гордіца В. Е., Пшеничний О. О., Хамула І. С., Лобачев І. М., Лобачев М. В., Тягунова М. Ю., Польська О. В. Технології Інтернету Речей для кіберфізичних систем. Практикум / За ред. Г. І. Воробця та В. С. Харченко – МОН України, Чернівецький національний університет імені Юрія Федьковича, Національний аерокосмічний університет "ХАІ", Запорізький національний технічний університет, 2019. – 172 с.

ISBN 978-617-7361-97-7.

Викладено матеріали практичної частини курсу МС4 "Технології інтернету речей для кіберфізичних систем", підготовленого в рамках проекту ERASMUS+ ALIOT "Internet of Things: Emerging Curriculum for Industry and Human Applications" (573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP).

Наведено структуру курсу, навчальні матеріали, приклади завдань для семінарів, практичних і лабораторних робіт, а також методичні рекомендації для самопідготовки і перевірки знань з дисципліни, та критерії їх оцінювання. Матеріал подається послідовно для формування цілісної картини сучасного стану, синергії, перспектив розвитку та досліджень технологій інтернету речей і кіберфізичних систем. Увага акцентується на концептуальних питаннях моделювання, аналізу, синтезу і практичного впровадження КФС, та ролі IoT на всіх етапах життєвого циклу складних комп'ютеризованих систем.

Призначено для магістрів університетів у галузі інформаційних технологій: комп'ютерних наук та інформаційних систем і технологій, кібербезпеки, системного аналізу, програмної та комп'ютерної інженерії, а також викладачів відповідних курсів, інженерів та науковців, які займаються розробкою та впровадженням технологій КФС та IoT.

Бібл. – 81, рисунків – 51, таблиць – 10.

Approved by Academic Council of National Aerospace University "Kharkiv Aviation Institute" (record No 4, December 19, 2018)

ISBN 978-617-7361-97-7.

ABBREVIATIONS

ABA –advanced branching algorithms
ACS – automated (automatic) control systems
ADC – analog-to-digital converter
BDD – Block Definition Diagram
CPS(s) – Cyber Physical System(s)
CSpace (CWorld) – cyberspace (world)
ES – embedded system
Esp-IDF – Espressif IoT Development Framework
FPGA – Field-Programmable Gate Array
GCPS – global CPS
GPIO interface – General Purpose Input Output interface
GUI – graphical user interface
I2C interface – Inter-Integrated Circuit interface
IBD – Internal Block Diagram
IDE – integrated development environment
IoT – Internet of Things
LCD – LED Countdown Display
LED – Light-emitted diode
LRM – laboratory research module
LSM – LED traffic signal module
LWIP – Light weight IP Stack
MARTE – Modeling and Analysis of Real-Time and Embedded Systems
MBD – Model-Based Design
MBSE – Model-Based Systems Engineering
NFP – Non-Functional Properties
OMG – Object Management Group
OOP – object-oriented programming
PAD – Pedestrian Audible Device
PCB – printed circuit board
PO – physical object
PoE – Power over Ethernet
PP – physical process
PTL – Pedestrian Traffic Light
PW(PS) – physical world (physical space)
PWM –pulse width modulation
RT process – process in real time
SDE – software development environment
SysML – Systems Modeling Language
UCF-file – User Constraints File
UML – Unified Modeling Language
VSL – Value Specification Language

INTRODUCTION

The materials of the practical part of the study course MC4 “IoT Technologies for Cyber Physical Systems”, developed in the framework of the ERASMUS+ ALIOT project “Internet of Things: Emerging Curriculum for Industry and Human Applications” (573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP)[1] are presented.

The course structure, teaching materials, examples of tasks for seminars, practical and laboratory works, as well as methodological recommendations for self-preparation and knowledge testing in the discipline, and criteria for their assessment are given. The material is submitted sequentially to form a holistic picture of the current state, synergy, prospects for research and development of Internet of Things and Cyber-Physical Systems technologies. The focus is on the conceptual issues of modeling, analysis, synthesis and practical implementation of CPS, and the role of IoT at all stages of the life cycle of complex computerized systems.

Module 1 “Basic Principles for the Organization and Functioning of Ecosystems of the Internet of Things and Cyber-Physical Systems” discusses conceptual issues related to the subject of research and the field of applications, structural organization and construction, as well as the functioning principles of the Internet of Things and Cyber-Physical Systems. The main attention is paid to a comparative analysis of the mutual similarities and differences between the subject area and the technologies used in the field of IoT and CPS. The synergy of IoT and CPS is studied and demonstrated using the example of a complex analysis of hierarchical-modular organization models of complex systems and multi-contour interaction of physical- and cyber-space elements. This module provides one practical lesson and seminar.

The second module “IoT Technology in the Problems of Analysis and Synthesis of CPS” is considered as the basic module in terms of research, design and development of modern cyber-physical systems.

---

[1] *The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.*

The special role of the Internet of Things technologies is emphasized both at the stage of research and development of CPS, and at the stage of implementation and expansion of their functionality. The synergy of IoT and CPS is demonstrated using the hybrid CPS/IoT information-measuring model. The model of through design of CPS/IoT systems from the physical process automation to the intellectualization of information processing and the management process is considered. This module provides for six laboratory works covering the entire cycle of analysis and development of CPS/IoT technical solutions, including the possibility of using reconfiguring computer means. The modern component base is analyzed, and the capabilities of the software and IoT technologies for solving the problems of creating modern cyber-physical smart systems are shown.

The third module “Power-Over-Ethernet Based Transducer Networks for Cyber Physical Systems,” discusses the capabilities of modern network technologies and standard protocols for the intellectualization of spatially distributed CPS. Based on PoE technology, interesting solutions for optimizing information flows, a smart model of energy-efficient CPS are proposed. An extended seminar is offered to familiarize with these questions and to analyze and study them thoroughly.

The fourth module "Model-Based Systems Engineering for the Cyber-Physical Systems" discusses modern CPS/IoT modeling and analysis techniques. Examples of the synthesis of intelligent CPS/IoT solutions and their optimization based on the MBSE approach are given. Two laboratory works are proposed for mastering these methods.

The course is intended for Masters of Universities in Information Technology: Computer Science and Information Systems, Cybersecurity, Systems Analysis, Software and Computer Engineering, as well as teachers of relevant courses, engineers and scientists involved in the development and implementation of CPS and IoT technologies.

Practicum prepared by a team of authors from *Yuriy Fedkovych Chernivtsi National University* – Associate Professor, Head of Computer Systems and Networks Department Heorhii Vorobets, Assistants Valentyna Horditsa and Oleksii Pshenychnyi, Master of Engineering Illia Khamula, Student Volodymyr Buchakchiiskyi; *National Aerospace University named after M. Zhukovsky "KhAI"* – Professor, Head of Computer Systems, Networks and Cybersecurity Department Viacheslav Kharchenko; *Zaporizhzhya National Technical University* – Associate

Professor, Head of Computer Systems and Networks Department Ravil Kudermetov, Associate Professor Mariia Tiahunova, Senior Lecturer Olga Polska; *Odessa National Polytechnic University* – graduate student Ivan Lobachev and Associate Professor, DrS Mykhailo Lobachev.

The authors are grateful to the reviewers, project colleagues, staff of the departments of academic universities, industrial partners for valuable information, methodological assistance and constructive suggestions that were made during the course program discussion and assistance materials. Special thanks to the foreign partners of the ALIoT project – Coimbra University, Portugal; University of New Castle, and Leeds, United Kingdom; KTH Polytechnic University, Stockholm, Sweden; and University of Pisa, Italy for seminars, practical experience on CPS and IoT development and implementation, which they sincerely shared with the project implementers from Ukraine.

1. BASIC PRINCIPLES FOR THE ORGANIZATION AND FUNCTIONING OF ECOSYSTEMS OF THE INTERNET OF THINGS AND CYBER-PHYSICAL SYSTEMS

Assoc. Prof., PhD H. I. Vorobets (ChNU)

1.1 Features of structural and functional synergy of IoT and CPS (Practical Work)

The aim of the Workshop: studying the conceptual foundations of the organization and functioning of IoT and CPS technologies ecosystems, obtaining practical skills in the methodology of developing the structure and functional models of modern high-performance IoT and CPS platforms, and their analysis.

Learning tasks:

- analysis of modern approaches and conceptual diagrams of the structural and functional organization of IoT and CPS;
- study of existing methodologies for the structural synthesis of IoT and CPS based on a hierarchically modular approach to their construction and requirements for the functional completeness of the tasks they implement;
- familiarization with the methodology of the complex approach and prototyping of systems and its application for the tasks of analysis and synthesis of IoT and CPS;
- mastering the studied approaches and methodologies by performing an individual practical task of building a functional CPS model and applying IoT technologies.

Preparation for the practical work includes two stages:

1) thorough study and analysis of theoretical material in a lecture course [1], recommended and self-processed literature and theoretical and methodological calculations given below;
2) implementation and preparation of an individual assignment report due to the recommendations below and its public defense in the audience.

1.1.1 Theoretical aspects of a complex approach to the analysis and synthesis of IoT and CPS

Definition of the essence of IoT and CPS, physical- and cyber-space structure. The concept of IoT and CPS synergy. According to the generally accepted definition, CPS is a technical high-tech solution where the combination of high-performance computing capabilities with informative filling of physical processes creates a new quality of world perception and new added value in the form of new knowledge, products or reducing the cost of its production [1,2]. The basis or the object of research and development here is the "physical process" (PP) of the real world (Fig. 1.1). PP is caused by a change in the state of physical object (PO), observing which a person (user, observer) receives new information about the PO.

The user himself can call some PPs acting on the PO indirectly through certain instrumental (device) interfaces (II, or DI): buttons, knobs, mechanisms, servos, electronic keys, etc., which are also elements of the real physical world/space (PW or PS). In an effort to reduce the amount of manual labor, the user engages mechanisms and systems that perform a certain ordered set of actions/influences on POs. Such tools perform the functions of managing objects or processes. They can be conditionally assigned to cyberspace (control space) and together with PS objects form automated or automatic control systems (ACS). As you can see, the selection of cyberspace objects here is purely conditional – these are real devices that affect the object, but are not directly involved in the implementation of the functions implemented by the PO.

Possibilities of implementing control functions increase significantly when using computer tools as control devices. In this case, their essence is "virtualized", since the control process is more associated with the use of "calculations" than by the direct generation of control signals. Even greater cyber-component virtualization is achieved by implementing network technologies for remote access to computing resources, the so-called "Cloud Technologies". Thus, it is generally accepted that hardware and software resources as objects of the real world create its superstructure in the form of cyber-space (world) (CSpace or CS and CWorld or CW) (Fig. 1.1).

The technology of vertical information exchange from physical processes to computing resources in the Internet on certain computers,

servers or distributed resources in the Cloud, and the results of processing information and control signals in the opposite direction are commonly called the “Internet of Things”.

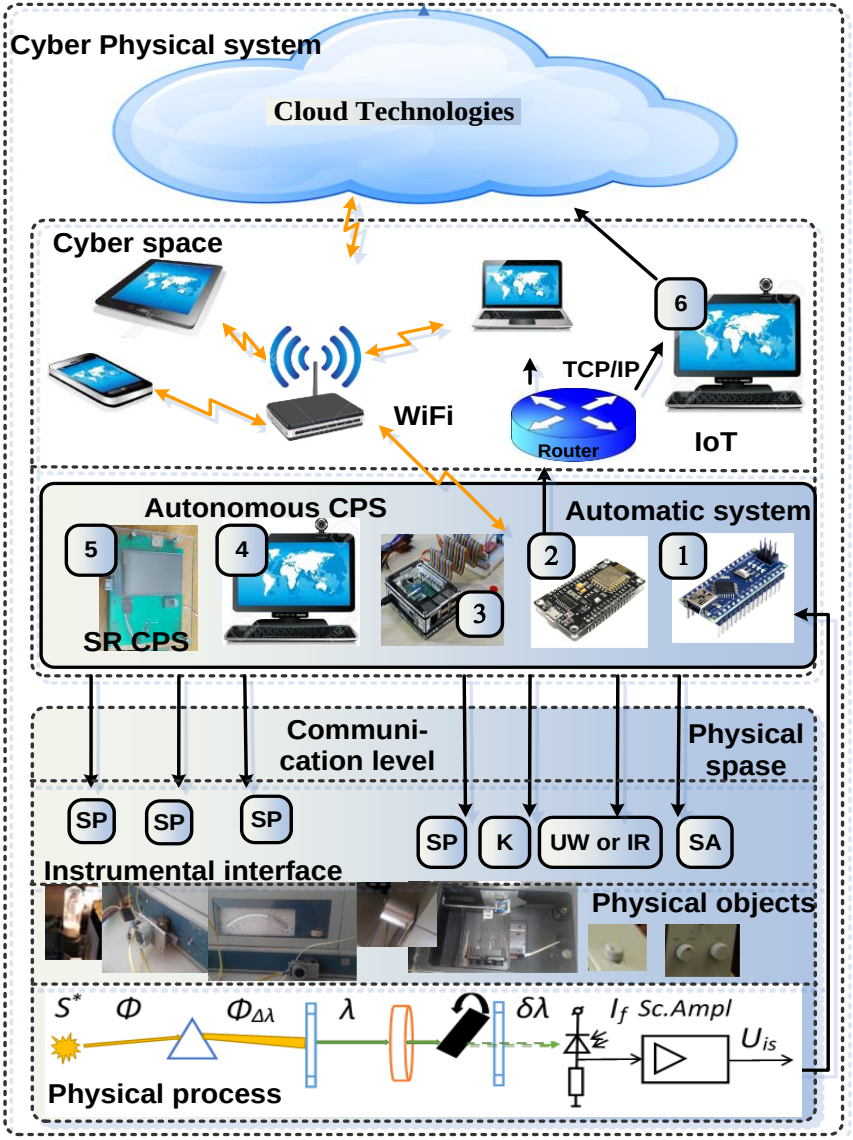


Fig. 1.1 – Structural organization and interaction of physical space and cyberspace components in CPS and IoT

The synergiaiзy of CPS and IoT can be traced already in their model representation: 5C CPS model and three-level IoT [1, 3, 4], and the expansion of the functionality of edge devices and network solutions leads to even greater leveling of differences between them. However, the applied hierarchically modular approach (Fig. 1.1) allows to trace the evolution of CPS and IoT and the differences between them, and to refine their classification depending on the functional completeness of the performed tasks. Let us consider the CPS evolution from mechatronic to self-configuration systems using the example of step-by-step automation of the information-measuring system – the SF series spectrophotometer.

Physical process analysis and modeling features. The essence of the physical process used in the system is to change the optical properties of the probe beam passing through the object under study, and provides the user with information about the characteristic parameters of the object – transmission/absorption of certain radiation wavelengths. To justify CPS platform models of various functional levels, we define the minimum required M_{min} and the full M_{full} PS parameter sets which CS will interact with.

The wavelength λ of the source radiation *S** and the magnitude of the information signal of the photo potential U_{is} can be attributed to the minimum required set. It is the dependency $U_{is}=f(\lambda)$ that provides the necessary information to the user about the research object. However, to obtain it, it is necessary ensure synchronization of the fixation of U_{is} and λ. This is achieved by searching for the signal of the initial value λ_0 in the scanning mode of the scale λ, when applying control pulses to the stepper motor, and setting the necessary offset $\Delta\lambda_i$ (number of pulses n_i) for the corresponding λ_i. In addition, the background value calibration signal U_{is0} is used to quantify U_{is}. It is determined for each value of λ_i before U_{is} measurements. To do this, the frame is shifted with the help of a stepper motor, and the geometric dimensions of the light window, which adjusts the maximum light flux $\Phi_{\Delta\lambda}$ to the photodetectors at a given value λ_i to a maximum $\Phi_{\Delta\lambda max}$, are changed.

Thus, the set

$$M_{min}=\{\lambda_0, \lambda_i, \lambda_f, n_i(\Delta\lambda_i), U_{is0}, U_{is}, U_{isf}, m_j(\Delta U_j), k_m\}$$

contains the minimum required set of PS parameters to automate the measurements of the studied PP at some fixed values of the PO states: $S_{UW/IR}=\{0, 1\}$ – type of radiation source of ultraviolet (UW) or infrared (IR) wavelength range; $PD_{UW/IR}=\{0, 1\}$ – type of photodetector of UW or

IR radiation; $\delta_U=\{0, 1, 2, 3\}_{10}=\{00, 01, 10, 11\}_2$ – photodetector sensitivity on one of the fixed four ranges, that can be encoded with two bits in binary form. In this case, the coordinate $\lambda_i=\lambda_0 + \Delta\lambda_i$ is uniquely determined by the rotation of the wavelength servo motor with the supply of $n_i=N\Delta\lambda_i/(\lambda_f-\lambda_0)$ pulses on it , where *N* is the number of pulses that ensures movement of the scale λ_i from the minimum $\lambda_{min}=\lambda_0$ to the maximum $\lambda_{max}=\lambda_f$ value of the available wavelength range. Similarly, in the calibration mode ($k_m=1$; $k_m \in\{0, 1\}$, 0 – measurements, 1 – calibration), the discretization step of the maximum search $\Phi_{\Delta\lambda max}$ is determined by the range of the photodetector signal change $m_j=M\Delta U_j/(U_{isf}-U_{is0})$ when changing the light window size. For complete automation of the PO, M_{min} needs to be supplemented with state parameters:

$$M_{full}=\{\lambda_0, \lambda_i, \lambda_f, n_i(\Delta\lambda_i), U_{is0}, U_{is}, U_{isf}, m_j(\Delta U_j), k_m, S_{UW/IR}, PD_{UW/IR}, \delta_U\}.$$

Functional-structural analysis and classification of IoT and CPS. Using microcontrollers (1, Arduino, or 2 (ESP32), Fig. 1.1), or computer tools (Sensors – DI (II) – 1 (Arduino) – USB port – 4 (personal computer, PC), Fig. 1.1) for processing sets of M_{min} and M_{full} allows synthesizing mechatronic ACS of respectively automated or automatic type. The application of "intelligent" system control algorithms for searching for given research ranges $\Delta\lambda_i$ and self-calibration of the optical channel is implemented here according to one or another given algorithm and does not provide new knowledge about the studied object, which this device is actually intended for. New knowledge, or their search with the help of new corrected measurements, is obtained after processing the received primary information by the user.

Applying a remote computer for controlling the PO and primary processing of the obtained measurement results (6) according to the scheme PS – Sensors – DI(II) –1 – 2 – 6 (Fig. 1.1), where data are exchanged using TCP/IP protocols, implements the IoT technology . Wherein module 6 can be implemented on the basis of a PC, smartphone, tablet, netbook, and use both wired and wireless communications (WiFi, RF, Bluetooth, IR, etc.).

Using of a control module (3) based on Raspberry Pi 3B+, BeagleBone C or another modification of a single-board microcomputer with the ability to deploy an embedded operating system (OS) as a cyber-component allows implementing an embedded computer system (ES – embedded system).

If advanced branching algorithms (ABA), which give new knowledge about the object under study, or/and allow to refine an already implemented process in real time (RT process) to obtain new knowledge, are used to process measurement data then we can talk about CPS implementation. Two options are possible if the PC hardware and software resources (3, 4, and 6 in Figure 1.1) allow implementing the necessary ABA:

1) autonomous CPS in the configuration Sensors – DI(II) – 1 (Arduino) – USB port – 4 (PC) or Sensors – DI(II) – 1 (Arduino) – USB port – 4 (PC);
2) Open-type CPS according to the scheme PS – Sensors – DI(II) – 1 – 2 – 6, or PS – 1 – USB port – 4 (PC) – Ethernet/Internet Gateway – 6 (PC) (Fig. 1.1) using IoT technology.

Open-type CPS can be classified according to various criteria - used IoT network technologies, implemented algorithms, problem orientation of tasks, architectural complexity, etc.

If local (4) or remote (6) PC resources for processing and simulating the studied objects and processes are not enough, or if high-performance data mining and analysis techniques are required, then it is advisable to use Cloud Technologies, which gives reason to speak about the global CPS (GCPS) formation (Fig. 1.1).

Both in autonomous and in open and global CPS, problem-oriented tasks are mainly solved, the essence of which is determined by the features of PP in the PO as elements of influence or control, and by the ways of processing data and obtaining the expected information. In such cases, it is advisable to use special processors based on FPGA reconfiguration environments to optimize the system in terms of minimizing hardware resources or solution obtaining time, energy efficiency of the system and others. Thus, the reconfigure levels and self-reconfiguration levels CPS (SR CPS) of an autonomous or open type are implemented (module 5, Fig. 1.1). To implement SR CPS, the system must have sufficient built-in hardware resources for the synthesis of special processors according to the implemented problem-oriented ABA. Such systems operate under the 3S model [1, 7] in certain conditions of uncertainty of the initial data regarding environmental influences. They have the properties of self-analysis, self-training/adaptation and self-organization/reconfiguration [8].

However, in all implementation cases of complex systems for information processing and management of objects and processes,

perhaps it's advisable to speak in terms of "systems" about CPS and in terms of "technology" about IoT. This essence is reflected in the names themselves. CPSystems – by the definition of "system", is a collection of physical- and cyber-space objects that interact with each other, and act as a whole when interacting with the environment to achieve a common goal [9,10]. InternetOT – technologies of communication and interaction between objects that can interact with the environment on their own, and are not required to form a system, although they do not exclude it. IoT, as technology can also provide intersystem communications.

The hybrid CPS/IoT model [1, 11] takes into account the synergy capabilities of “systems” and “technologies” and makes it possible to optimize technical solutions according to the generalized economy/speed-performance index [12] of both the cyber component and the CPS as a whole.

At the same time, a clear separation of functions and tasks performed by CPS and IoT will allow to standardize design approaches and technical requirements for individual modules, their interfaces, development and modeling environments, and accordingly to simplify the process of through analysis and synthesis of CPS from sensors and terminal devices to hardware solutions of all levels and cloud models.

Features of IoT and CPS algorithms synthesis. Let's consider an example of a functional algorithm that requires a cyber-physical approach for its implementation, in contrast to the mechatronic one. So for the mechatronic ACS of SF-series spectrophotometer, it is enough to implement a linear algorithm of data measurement to establish the dependence $U_{is}=f(\lambda)$:

1) set the initial data: information about the test sample, scanning range $\lambda : \lambda_1<\Delta\lambda<\lambda_2$, scanning step δ_λ in *nm*, file name to write data;
2) open the data file and record the initial information and carry out the initial settings:
   - move the carriage of the scale λ to the state λ_0;
   - move the carriage of the scale λ from the state λ_0 to the state λ_1;
3) run the measurement management program:
   - write the values of λ_1 to the file;
   - calibrate the optical channel – close the key *K* (Fig.1.1) and set the value $U_{is0}=0\%$, open the key *K* and set the value $U_{is0}=100\%$, close the key *K*;
   - place the studied sample to the optical channel;
   - open the key *K*;

- measure the value of U_{is} at the set λ_1 value;
- write the measured value U_{is} to a file according to λ_1;
- move the scale carriage λ from the state λ_1 to the state $\lambda_i=\lambda_1+\delta_\lambda$;

4) cyclically repeat step 3 until the value $\lambda_i=\lambda_2$ is reached;
5) carry out the last measurement cycle, write the U_{is} value to the file according to λ_2, close the file and finalize the program.

As can be seen, the entire measurement process is carried out automatically. Most information will be obtained when choosing the minimum scan step for the spectrum δ_λ, but the measurement duration will be the highest, within an hour. Increasing the δ_λ step when exploring new unknown samples can lead to information loss between the values of λ_i and λ_{i+1} at unknown intervals of *i*. To solve this problem, it is proposed to intellectualize the system by implementing ABA and "raising" the technical software solution to the level of CPS, or the "analytical/thinking" system.

Several ABA depending on the requirements for measurement accuracy can be implemented. In particular, the algorithm of δ_λ dynamic correction, depending on the change dynamics of the U_{is} photo detector signal, is attractive. Its essence is as follows. With a small calibration characteristic drift in time , it's proposed to measure it with a sufficient step δ_λ once before a series of studies, write it to a memory/file and use it further to process and establish the actual measurement values of the studied samples characteristics $U_{is}=f(\lambda)$. Spectrum measurements are carried out in scanning mode, taking into account the dynamics of changes in the calibration characteristic ($S_k=\Delta U_{isk}/\Delta\lambda_i$), the inertia of mechanics as the spectrum sweep speed ($\upsilon_\lambda\approx2\div5$ nm/s) and the duration of the analog-to-digital conversion ($\tau_{ADC}\approx 1$ ms) of the information signal U_{is}. According to the estimatesthe cyber-component manages to take several hundreds of U_{is} measurements and to estimate the dynamics ($Sm=\Delta U_{is}/\Delta\lambda_i$) in comparison with S_k during the δ_λ=1 nm offset. The initial set λ_i is selected from the condition of the correct visual display $U_{is}=f(\lambda)$ on the monitor or by printing the specified range $\lambda1 <\Delta\lambda <\lambda2$. Changing the step δ_λ of fixing U_{is} at the corresponding λ_i occurs dynamically during the measurement process in accordance with a change in the ratio $\varepsilon=S_m/S_k$ (%). That is, with an increase in ε by X%, the step δ_λ of the next fixation of the measurement results decreases by the same percentage. Measurement synchronization is carried out by the counter of clock pulses, which are fed to the servo of scale λ. Thus, the more features the spectrum under study will have in a certain area $\Delta\lambda$

compared to the background, the more fixation points λ_i will be in the same range.

Sets of parameters M_{min}, M_{full} for the development of functional models and implementation of the proposed ABA require extension –

$$M_{min}^{*}=\{M_{min}, S_m, S_k, \varepsilon\}, M_{full}^{*}=\{M_{full}, S_m, S_k, \varepsilon\}.$$

Unlike the previous algorithm, the measurement process itself is unpredictable, that is, it is realized under certain conditions of uncertainty of the problem by the input parameters, and depends on the properties of the object under study. Accordingly, the cyber component, although it works according to the recorded algorithm, however, provides correction of this algorithm's parameters. Steps 2 and 3 of the linear algorithm description, when modifying it for the proposed ABA, must be rewritten as:

"…- complete the optical channel calibration cycle and store the data;
- place the test sample to the optical channel;
- write the value λ_1 to a file;
- take measurements of the value of U_{is} with the set λ_1;

3) start the measurement management program:
- start the n_i counter;
- start moving the carriage of the scale λ from the state λ_1 to the state $\lambda_i=\lambda_1+\delta_\lambda$;
- take measurements of U_{is} value for the current intermediate value λ_i;
- calculate the parameter S_{mi};
- calculate $\varepsilon=S_m/S_k$ (%);
- make correction for δ_λ^{*};
- stop the counter and, accordingly, the carriage λ, with the corrected values n_i^{*} and $\lambda_i^{*}=\lambda_1+\delta_\lambda^{*}$;
- take measurements of the value of U_{is} when set $\lambda_i^{*}=\lambda_1+\delta_\lambda^{*}$;
- write to file the defined U_{is} values with set λ_i^{*};

4) cyclically repeat step 3 to achieve the value of $\lambda_i=\lambda_2$; … ".

The branching of this algorithm is manifested at the stage of making correction for δ_λ^{*} according to the calculation results $\varepsilon=S_m/S_k$ (%). It is clear that the duration of all the calculations and correction processes should be completed before fixing n_i^{*} and λ_i^{*}. This imposes additional requirements on the productive speed-performance of cyber-component computing resources of the designed CPS.

The disadvantage of this approach is also the "unpredictability" of the data file size, and, accordingly, the amount of memory for storing the

results. The selection of the appropriate modules and parameters of the CPS cyber-component is performed by searching for the maximum value of the target function F_s of the hardware resources description of the designed system [1, 7].

IoT technology is used here if the system configuration is implemented according to structure A: PS – Sensors – DI(II) – 2 – Ethernet/Internet Gateway – 6 (PC) (Fig. 1.1). However, option B is also possible: PS – Sensors – DI(II) – 2 – USB port – 6 (PC). The correct choice of option A or B requires further time parameters exploring of the Ethernet/Internet Gateway, USB port of PC and the ESP32 platform used.

Other ABAs that meet the requirements of "cyberphysics" of systems, that is, the synergy of “computing” and “physical processes”, and the wider use of IoT technologies are also possible. For example, an algorithm for critical points searching (weakly expressed local extrema, inflection points of linear sections $U_{is}=f(\lambda)$, etc.) and the subsequent “thin” scanning of selected sections $\Delta\lambda$, or an algorithm for dynamic modeling of measured characteristics in real time and / or Fourier analysis of the obtained spectra with parallelizing these processes on computational modules. To evaluate and optimize the required resources for the implementation of such algorithms, it is necessary to apply a systematic approach and correctly determine the target functions of the analysis and synthesis of CPS and IoT.

1.1.2 Recommendations for completing an individual practical task

General recommendations for studying the CPS and IoT synergy in a specific subject area. The features of CPS and IoT synergy should be analyzed based on the functional purpose of the systems under study and the essence of the physical processes embedded in their functionality. Applying system decomposition separate structural modules – components – elements allows to better understand the physical nature of the objects being analyzed and to establish causal relationships of their functional. A hierarchically modular model of functional description allows to summarize the requirements for the methods and methodologies of information processing at individual levels, as well as to trace the movement of information flows between the levels.

A generalized integrated approach to the analysis and synthesis of CPS and IoT functional models provides the following steps:

1. A detailed study of the information transformation and transmission features in the analyzed physical system at the physical (natural) level.
2. Establishing multiple physical values and characteristics for the quantitative assessment of physical processes.
3. Determining the minimum required, complete and expanded sets of parameters and characteristics necessary for the basic system functionality implementation.
4. Choosing an algorithm for the basic functionality implementation of data collecting and processing, and process management.
5. Optimization of the selected algorithm in terms of minimizing used hardware resources and system performance.
6. Justification of the requirements for the parameters of the hardware resources used for the implementation of functionally structural solutions.

Such steps can be implemented using the methodology of linear gradual approximation of solution and multiple iterations by the model of multi-circuit CPS interaction with components of the physical world.

As shown in the theoretical part, the attention should be focused on the parametric description of the system in paragraphs 2-4, since they form further requirements for the structural-algorithmic synthesis of CPS and IoT and determine its complexity. It is also worth comparing possible functional algorithms that allow for a qualitative transition from mechatronic systems to CPS and justify the need for synergy in the hybrid CPS/IoT model, for example that is handled in an individual task.

The topics of individual tasks concern the practical applications of CPS and IoT in the environment and everyday life and are not limited with the list below. Student initiative is supported to formulate and expand the proposed topics, especially tasks promising for further promotion in the form of start-up projects. It's worth noting that the proposed tasks should allow for a through analysis of the system from the level of sensors and servos to the possibility of using cloud technologies.

Recommended topics for individual tasks:

1. Autonomous car model for closed-loop racing.
2. Traffic light model with intelligent daily and situational traffic management.

3. Mars-rover model for cross-country traffic and metal search.
4. The system of intelligent lighting control in a multi-room apartment.
5. The system of intellectual zonal control of urban youth area lighting.
6. Climate control system of greenhouse economy.
7. Pet care system.
8. Smart exercise bike with intellectual analysis of the user's functional life activities.
9. Monitoring system of athlete's physical activity in the course of performance of strength exercises.
10. Monitoring system of driver's emotional and physical condition.
11. Express system of quality analysis of one type of the food.
12. Smart-windows energy efficiency monitoring and correction.
13. Gas-analyzer for underground work.
14. System for protecting premises against unauthorized access.
15. The temperature control system for 3D thermal printers.

Requirements for the structure and content of the technical report are determined by the general requirements for the formalization of the research results and scientific reports. The presentation of the material should be consistent and logical. The disclosure completeness of individual issues should be sufficient to justify the main ideas and theses that the author makes for general discussion and coverage of the relevant topic. It is strictly forbidden to compile parts of other authors' publications and study guides, as well as overload the text with well-known reference materials. References to cited literary sources or authors whose ideas are discussed or further developed in the presented study are mandatory. The structure and content of the submitted materials can be as follows:

- Title page, indicating the educational institution, department and specialty where the undergraduate studies, full name of the undergraduate, research titles, in parentheses – subheading: individual assignment from the course IoT technologies for CPS, bottom of the page – city and year of work. Abstract, list of abbreviations and key phrases, content. Volume – 4 pages.
- Introduction (a brief overview of the relevance and processing state of the task is presented; the purpose, object and subject of research are defined; whether this work is a continuation of previous research and publications of the author should be indicated), volume – 2 pages, title is not numbered.

- Main part. The presented material is structured in paragraphs. The paragraphs names are formed according to the topic and purpose of the study, and the specific tasks they solve, and are numbered from position 1. Recommended volume – 3-4 paragraphs, only 11-14 text pages. For example:
  1. Analytical review of modern approaches to the synthesis of CPS (a thorough analysis of well-known scientific publications and Internet materials with emphasis on unresolved problems or tasks requiring further research is presented; the paragraph ends with statement of tasks for our own developments, which will be covered in the following paragraphs). Volume – 3-4 pages.
  2. Method and examples of solving the problem of structural-algorithmic CPS synthesis using IoT technology (the results of research, development of recommendations, models, technical solutions, etc. are presented, proposed by the author, which, in his opinion, are sufficiently substantiated, or confirmed by experiments or practical applications; separate subparagraphs may be highlighted for a more understandable presentation of the material) Volume – 5-6 pages.
  3. Analysis/modeling of the proposed methods and technical solutions (a comparative analysis of the advantages and disadvantages of the models, approaches and solutions proposed and described by the author in Section 2 is carried out; modeling and simulation at the level of program models or sketchy models are possible). Volume – 3-4 pages.
- Conclusions (the achieved 3-5 main results are ascertained, which, in the author's opinion, are sufficiently substantiated with calculations in the main part; suggestions are made for further research and recommendations; the heading is not numbered). Volume – 1-2 pages.
- List of cited references (a list of publications and Internet publications used by the author to substantiate the provided research; the heading is not numbered). Volume –1-2 pages.
- Appendices (materials that are not included in the main part, but are important for a more complete disclosure of their contents are provided; the heading is not numbered). Volume – up to 5 pages.

Thus, the total report volume doesn't exceed the size of 1 printed sheet (24 pages) and is within 17-24 standard A4 pages, font – Times New Roman, size – 14, spacing – 1.5, margins – 2 cm.

It is suggested to use this design template for other reports on a seminar, individual and other tasks in this manual, where no special requirements are specified, as well as for the preparation of materials on research work, technical reports, and materials for scientific publications of masters.

Implementation and protection of an individual task. An individual assignment is recommended to start with a thorough study of the lecture material [1] and additional literature. The above theoretical calculations can be used as an example of this task.

First of all, it is necessary to systematize the studied material and structure it according to a previously developed plan. It's recommended to use schemes, diagrams, tables, figures for systematization. This allows you to present more information in a compact form and compare the advantages and disadvantages of various solutions and approaches. Based on this comparison and analysis, proposals are generated to address the identified shortcomings, and methods for their elimination. This approach is individual and depends on the previous experience of the performer.

To protect the completed report, it is recommended to prepare a brief presentation of 15-18 slides, based on 5-7 minutes of the report. The filling of the slides should correspond to the structure and content of the plan developed by the author on his theme according to the above sample. The language of presentation and report is English.

When evaluating the results of an individual task, the following are taken into account:

- originality of approaches and proposed solutions;
- the quality of text design in form and content;
- the quality of the presentation on design and content;
- the quality of the report and the format of the discussion when discussing the report;
- the completeness and correctness of the answers to the questions.

Recommended literature

1. Internet of Things for Industry and Human Application. In Volumes 1-3. Volume 1. Fundamentals and Technologies / V. S. Kharchenko (ed.) – Ministry of Education and Science of Ukraine, National Aerospace University KhAI, 2019. - 605p.
2. K. Schwab, *The fourth industrial revolution*. Crown Publishing Group, Division of Random House Inc, 2017.

3. B. Bagheri, S. Yang, H. Kao and J. Lee, "Cyber-physical Systems Architecture for Self-Aware Machines in Industry 4.0 Environment", *IFAC-PapersOnLine*, vol. 48, no. 3, pp. 1622-1627, 2015. Available: 10.1016/j.ifacol.2015.06.318.
4. N. Suda, “Reconfigurable Architectures and Systems for IoT Applications”, Dissertation Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy. Arizona state university, 2016. – 83 p. – N. Suda, *Repository.asu.edu*, 2019. [Online]. Available: https://repository.asu.edu/attachments/ 164110/ content/Suda_asu_0010E_15651.pdf. [Accessed: 20- Jul- 2019].
5. E. Lee and S. Seshia, "Introduction to Embedded Systems - A Cyber-Physical Systems Approach", *Ptolemy.berkeley.edu*, 2019. [Online]. Available: https://ptolemy.berkeley.edu/books/leeseshia/releases/LeeSeshia_DigitalV1_08.pdf. [Accessed: 05- Jul- 2019].
6. R. D. Sriram, "Toward Internet of Everything: IoT, CPS, and SNSS", OntologPSMW. *Ontologforum.org*, 2019. [Online]. Available: http://ontologforum.org/index.php/ConferenceCall_2015_03_12. [Accessed: 20- Jul- 2019].
7. H. Vorobets and V. Tarasenko, "Self-configuring computer tools in Cyberphysical Systems (Ukrainian)", in *Cyberphysical Systems:: Achievements and Challenges: Proceedings of the Second Science Seminar*, Lviv, 2016, pp. 114-120. Available: http://195.22.112.37/bitstream/ntb/39386/1/20-114-120.pdf
8. N. Wiener, *Cybernetics or control and communication in the animal and the machine*. Mansfield Centre, CT: Martino, 2013.
9. V. Golembo and O. Bochkaryov, "Approaches to Building Conceptual Models of Cyberphysical Systems (Ukrainian)", *Ukrainian Journal of Information Technology*, vol. 864, no. 1, pp. 168-178, 2017. Available: http://science.lpnu.ua/uk/scsit/vsi-vypusky/vypusk-864-nomer-1-2017/pidhody-do-pobudovy-konceptualnyh-modeley-kiberfizychnyh. [Accessed 3 August 2019].
10. V. Melnyk, I. Lopit and A. Keith, "Information exchange protocol for computer devices automatic creation in reconfigurable hardware platforms of the cyber-physical systems computing nodes (Ukrainian)", in *Cyberphysical Systems: Achievements and Challenges: Proceedings of the Second Science Seminar*, Lviv, 2016, pp. 17–22.
11. C. Greer, M. Burns, D. Wollman and E. Griffor, *Cyber-Physical Systems and Internet of Things*. NIST Special Publication 1900-202, 2019, p. 61. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1900-202.pdf
12. B. V. Glushkov, Introduction to Cybernetics.(Russian) Kiev: Publishing House of the Ukrainian SSR Academy of Sciences, 1964.

1.2 Multi- contour interaction of cybernetic and physical space in the CPS/IoT model. Assessment of CPS computing resources (Seminar 1)

The aim of the seminar: studying of the methodology of the systematic approach to the analysis and synthesis of CPS and IoT and the construction of a multi-contour model for the interaction of cyber components with the physical environment, the acquisition of practical skills in the systematic assessment of the computing resources of IoT and CPS and their analysis.

Learning tasks:

- mastering the decomposition methodology for the analysis and synthesis of CPS and IoT and building a hierarchically modular organization of modern CPS in the form of configuration Structure – Device – Module – Process – Function – Procedure/Action (S – D – M – P – F – A);
- studying the methodology for constructing interaction contours in the CS-PS system and the synthesis of a multi-contour model of CS and PS interaction, determining the role and place of IoT technology in such interaction;
- getting acquainted with the methodology of the system analysis problem statement for a comprehensive CPS hardware resource assessment approach;
- practical development of the studied approaches and methodologies by performing an individual task of constructing S – D – M – P – F – A models and multi-contour interaction in CPS using IoT technologies.

Preparation for the seminar.

Preparation for the **seminar** includes the stages:

1) getting acquainted with the purpose and tasks of the seminar;
2) thorough study of theoretical material in a lecture course [1], recommended and self-processed literature;
3) analysis of individual task questions;
4) implementation and preparation of an individual task report due to the recommendations in paragraph 1.1.2;
5) presentation at the seminar and public discussion of the report.

1.2.1 Basic recommendations on theoretical material

When studying the theoretical material, it's worth paying attention to the variety of CPS and IoT application areas, and, accordingly, to the peculiarities of structural-functional solutions, depending on the application. The hierarchical-modular approach simplifies the system synthesis process. However, the issue of a comprehensive assessment of the functionality of the components offered on the market at a certain price and functional segment of modules, a comparative analysis of technical complexity/efficiency, economic indicators (cost, energy consumption, maintenance costs, etc.) comes to the fore. Such an analysis is carried out precisely in retrospect of functional algorithms chosen earlier for implementation in the system. As shown in paragraph 1.1.1, the choice of algorithm can significantly affect the quantitative and qualitative indexes of the hardware resources required in the system.

Important aspects are the requirements for the system multitasking and the effective redistribution of resources. The system approach proposed in [1, 2] allows taking into account the indicated aspects and comprehensively evaluate the resources necessary for the implementation of CPS in accordance with the user-defined F_{CPS} target function.

1.2.2 Recommendations for the practical task

The practical task is performed by students individually and is a continuation of the studies begun in accordance with the selected options for individual tasks in paragraph 1.1.2. Based on the developed functional model and the functional algorithm described in the previous task, students are invited to develop a methodology for synchronizing data processing processes with the corresponding functional modules [3]. Determine the data processing cycling and describe cycles at the level of used modules interaction. Assess the necessary resources for the implementation of the multi-contour model, and justify the capabilities of the well-known Arduino, Discovery, Raspberry Pi, BeagleBone, ESPXXXX base platforms for the CPS project implementation, both with and without using IoT technologies. Compare the latest options for the complexity and efficiency of the main functional algorithm implementation.

For better preparation for the seminar, it is suggested to study the following issues:

1. Justify the functional purpose of the analyzed CPS and briefly describe the essence of the PP used.
2. Identify the possibilities of parallel information processing when managing and obtaining data on PO and PP in the studied CPS.
3. Compile a table of correspondence between the CPS modules/devices/components and the parameter sets of the PS objects with which they interact.
4. Identify which IoT components can be used to provide "intelligent" data processing in the described CPS.
5. What IoT technologies can be used in the processes of cyclic information exchange and data transfer in the CS-PS complex described by CPS?
6. What types of information processing (Parallel, cloud, fog, edge calculations) can be used when considering the CPS/IoT model?
7. What types of hardware resources can be used to process information in the CPS/IoT model?
8. What software (SCADA, MARTE, SysML, UML) is appropriate to use for modeling and simulation of the described CPS/IoT model?

Requirements and recommendations for the preparation and execution of the report are similar to the requirements described in paragraph 1.1.2 for the practice work.

Recommended literature

1. Internet of Things for Industry and Human Application. In Volumes 1-3. Volume 1. Fundamentals and Technologies / V. S. Kharchenko (ed.) – Ministry of Education and Science of Ukraine, National Aerospace University KhAI, 2019. - 605p.
2. G. Vorobets, O. Vorobets and V. Horditsa, "Application of the System Approach for Synthesis of Models of Basic Elements of Reconfigurable Structures at the Information Transmission Systems", *Electrical And Computer Systems*, vol. 28, no. 104, pp. 257-267, 2018. Available: 10.15276/eltecs.28.104.2018.31.
3. A. Melnyk and I. Iakovleva, "Method for Storing in the Memory of a Flowing Graph Algorithm in the Form of a Structural Matrix (Ukrainian)", 96041, 2011.

2. IOT TECHNOLOGY IN THE PROBLEMS OF ANALYSIS AND SYNTHESIS OF CPS

2.1 Basic principles, hardware solutions and application of IoT technologies in CPS synthesis tasks

Assoc. Prof., PhD H. I. Vorobets, Assist. of Lect. V. E. Horditsa, Assist. of Lect. O. O. Pshenychnyi, MsS I. M. Khamula (ChNU)

2.1.1 Basic modules for structure synthesis and CPS/IoT technology testing

In the market of electronic components and functional modules, currently offered by manufacturers, to create models, prototypes of functionally completed projects, there are products of a wide price range, various architectures, complexity, and, accordingly, with a wide range of functionality. These are, for example, entire series (lines) of breadboards from the simplest and cheapest Arduino XXX [1], Discovery XXX [2], Raspberry Pi XXX [3], BeagleBone [4] and others in the price range from 2-3 to 10 dollars and to powerful modern modules with high-speed processors and reconfiguring architecture components such as Nexys Video Artix-7 FPGA [5, 6], Stratix 10 SX SoC Development Kit [7] costing from several hundreds to 8-10 thousand dollars.

An interesting solution for deploying IoT projects is also offered as the NuMaker Brick platform [8], which has extensive expansion/scaling capabilities and supports open-source codes and protocols. Such on-board modules are deployed on the NuMaker Brick platform: a temperature and humidity sensor, a gas detection module, an infrared module, gyroscopes, an accelerator, a sonar, a LED and a buzzer. It is compatible with Android mobile devices and tablets.

The platform is implemented according to the strategy of fully open-source codes for all embedded programs, hardware and application software of the NuMaker Brick platform, consisting of the terminal of the main controller and several modules (Fig. 2.1). The main controller terminal and sensor modules use the high-performance NuMicro® M451 series controllers. This allows implementing the technologies of parallel, distributed and boundary calculations (fog and edge) of data collected from sensors, and offloading data traffic for IoT applications.

Mobile phones and tablets can be connected to the platform through the applications for the platform functional adjustment (Fig. 2.2).

NuMaker Brick also provides greater software flexibility. Each module is already designed to enable a specific function. Users can start using the NuMaker Brick platform without having to rewrite any program, and modules can be spaced or connected randomly. An expansion board is provided where developers can create their own modules.

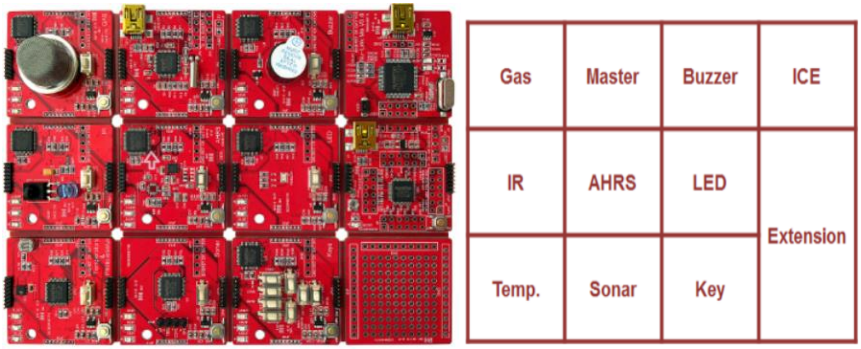


Fig. 2.1 – NuMaker Brick Base Platform Layout [16]

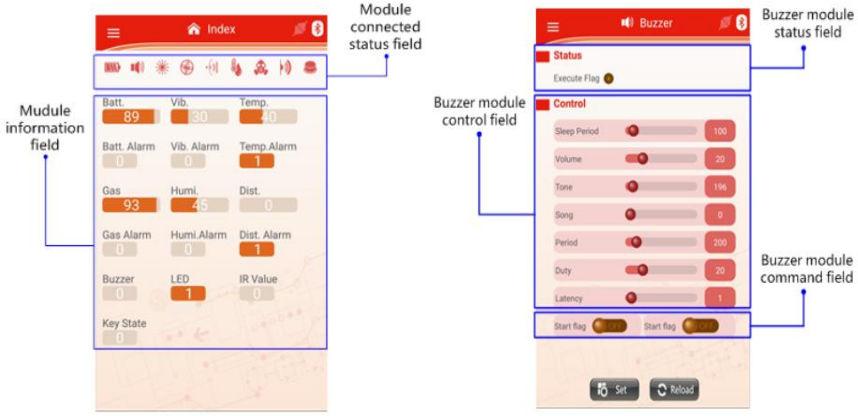


Fig. 2.2 – NuMaker Brick Platform Software [16]

Modification of the NuMaker Uni basic solution uses the 32-bit Cortex®-M0 NANO100NE3BN microcontroller capable of operating at up to 42 MHz [9]. The board contains an RGB LED, infrared transmitter and receiver, 3-axis accelerometer and 3-axis gyroscope (MPU6500), temperature and humidity sensor (HTU21D), Wi-Fi 802.11b/g/n module (ESP-03) which supports TCP, UDP client-server technology, access

point or station mode and AT command, as well as Bluetooth 3.0 module.

The NuMicro NuMaker UNO Development Board [10], which is fully compatible with Arduino equipment, and uses the NuMicro ™ Cortex®-M0 microcontroller to develop and test applications, has also been created. Users can use the Arduino IDE, IAR EWARM, and Keil RVMDK to develop their applications and use a large number of open-source examples. In particular, using the peripheral functions of the ADC, PWM, I²C, SPI, etc., and support for the virtual COM port on USB.

Of course, new developments are also offered on the component market, in particular, Seeed Studios has released a new product under the brand Air602 Wi-Fi Development Board [11]. The core of this board, 12 mm x 10 mm of size and costing $ 1.90, is the tiny WinnerMicro W600 microcontroller, which has a built-in 32-bit ARM Cortex M3 processor. At the hardware level, it implements encryption, as well as two UART interfaces, I2C, SPI and I2S, a real-time clock (RTC), and Wi-Fi support. However, it can be programmed using conventional ARM development tools (such as Keil) using the provided SDK. And although the cost of such a crystal is commensurable with the $ 2-3 cost of an ESP32 crystal, in development tools it is inferior.

As can be seen from the analysis, there are a lot of solutions that are very similar in architecture, technical specifications, price range, but differ only by manufacturers. Therefore, project developers are faced with complex questions of justifying the choice of certain components. Of course, first of all, the necessary functionality, the set of implemented algorithms for a specific set of problem-oriented tasks are determined, and it is reflected further on the set of technical parameters and characteristics of the required modules, as shown in the previous section. However, the designer has to make the decision of selecting the optimal ratio for the technical feasibility/quality/cost of the proposed components/modules for the specific problem being solved. If the requirements for the ratio of quality/cost aren't questions in this case, then the assessment of feasibility is closely related to both technical characteristics and economic indicators. In particular, it is clear that any universal module needs to be adapted to the specifics of problem-oriented tasks. And here there are questions of hardware resources redundancy of the selected module regarding the specific tasks to be

solved, its energy supply and energy efficiency, stability (elasticity), reliability and service, etc.

Based on these considerations, for a comprehensive solution of educational issues, testing of control algorithms and data processing, rapid prototyping and embedded solutions implementation, as well as research and providing communication capabilities of IoT technologies, a technical solution of a laboratory research module (LRM) based on ESP32 crystal is proposed (Fig. 2.3). The created LRM layout corresponds to the price range of 50-80 euros, which is comparable with the 5Stack ESP 32 M5 Kit [12], BeagleBone [4] prototypes, and several advanced models of Arduino XXX [1], Discovery XXX [2], NuMaker Brick [8].

ESP32 is a somewhat extended version of ESP8266 [13]. This series is designed for mobile devices, small-sized electronics, and IoT applications. ESP32 is characterized by ultra-low power consumption due to energy-saving features, including high-resolution synchronization, and several modes of dynamic power correction and dynamic power scaling [14].

The heart of the ESP32 is the Tensilica Xtensa LX6 dual-core processor with a clock speed of up to 240 MHz. The ESP32 is tightly integrated with built-in antenna switches, radio frequency balance, power amplifier, low noise input amplifier, filters, and power management modules. And although it differs from the ARM architecture used in advanced versions [1-3], nevertheless it easily fits with them at the hardware-modular level [15, 16]. In addition, the Arduino IDE is now adapted for programming various modifications of the ESP32 and ESP8266 [17, 18].

A significant advantage of the created LRM (Fig. 2.3) compared to [8-11], which are more focused on IoT technologies, is also its expanded functionality for the synthesis and testing of CPS. Thus, in [8, 11] there are practically no interface modules with servo drives based on PCA9685 and local information display devices – OLED display and LED matrix based on MAX7219, which is implemented in LRM.

The following components are located in the base module for expansion of LRM functionality (Fig.2.3):

1. ESP 32 WROOM-32 main controller module.
2. 8-port PCF8574 control module.
3. Combined accelerometer/gyroscope IMU 6050, accessed via the I2C bus.

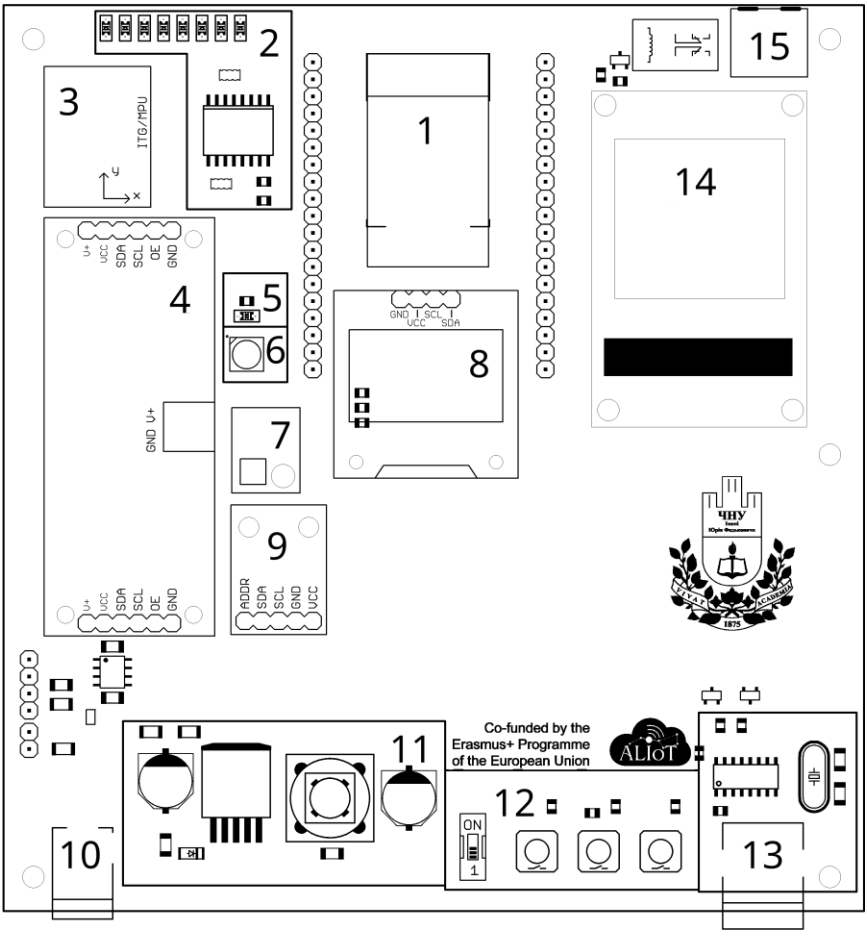


Fig. 2.3 – Layout of the basic laboratory research for the development, sketchy layout and testing of CPS/IoT structures and technologies

4. 16-channel 12-bit PWM/Servo module with I2C interface based on PCA9685.
5. LED-indicator connected to 13 ESP32 output (similar to Arduino compatible boards and modules).
6. Color RGB LED connected directly to ESP32. Using PWM allows up to 16 million colors.
7. BMP-180 atmospheric pressure sensor.
8. OLED display 0.96" I2C 128x64.

9. Digital light sensor GY-302 BH1750FVI.
10. Power supply input for servos, 5V/5A.
11. Schematic of the AC/DC converter based on LM2596 for forming the 3.3V power supply line of the stand.
12. Buttons and switches to interact with the user and reset the stand.
13. USB UART interface based on the CH340 chip.
14. LED matrix based on MAX7219.
15. Relay connector for external loads.

The proposed component set allows using this module as a base for approbation and testing solutions in CPS at the level of the instrumental interface, since it contains a set of basic sensor types and a switching module for connecting power control circuits in the form of a 16-channel 12-bit PWM/Servo module with I2C interface based on PCA9685. In the same version, it can also be used as a base platform for an embedded system with enhanced peripheral switching capabilities in the form of two contact rows along the base crystal (1), as well as a local functional of control for PP parameters and its progress, with visualization of individual results (components 8 and 14) . Using exactly ESP32 also allows for implementing TCP, UDP network layer protocols using IoT technology. Therefore, based on this LRM, a cycle of laboratory works for master students for the basic level mastering of synthesis and analysis of individual solutions and CPS/IoT technologies is proposed.

Recommended literature

1. "Arduino Official Store | Boards Shields Kits Accessories", *Store.arduino.cc*, 2019. [Online]. Available: https://store.arduino.cc/.
2. "STM32 Discovery Kits - STMicroelectronics", *STMicroelectronics*, 2019. [Online]. Available: https://www.st.com/en/evaluation-tools/stm32-discovery-kits.html.
3. *Raspberrypi.org*, 2019. [Online]. Available: https://www. raspberrypi.org/products/raspberry-pi-4-model-b/
4. "BeagleBone Black Rev.C купить Днепр / Украина магазин Xcraft", *Xcraft.com.ua*, 2019. [Online]. Available: https://xcraft.com.ua/mini-kompyuter-beaglebone-black-rev-c.
5. Internet of Things for Industry and Human Application. In Volumes 1-3. Volume 1. Fundamentals and Technologies / V. S. Kharchenko (ed.) – Ministry of Education and Science of Ukraine, National Aerospace University KhAI, 2019. - 605p.

6. "Nexys Video Artix-7 FPGA: Trainer Board for Multimedia Applications", *Digilent*, 2019. [Online]. Available: https://store.digilentinc.com/nexys-video-artix-7-fpga-trainer-board-for-multimedia-applications/.
7. "Stratix 10 SX SoC Development Kit", *Intel.com*, 2019. [Online]. Available: https://www.intel.com/content/www/us/en/programmable/products/boards_and_kits/dev-kits/altera/stratix-10-soc-development-kit.html.
8. "NuMaker Brick (IoT Platform) - Nuvoton Direct", *Nuvoton Direct*. [Online]. Available: https://direct.nuvoton.com/en/numaker-brick.
9. "NuMaker Uni (IoT Wearable Platform) - Nuvoton Direct", *Nuvoton Direct*, 2019. [Online]. Available: https://direct.nuvoton.com/en/numaker-uni.
10. "NuMaker Uno (Arduino Compatible) - Nuvoton Direct", *Nuvoton Direct*. [Online]. Available: https://direct.nuvoton.com/en/numaker-uno.
11. M. Posch, "Tiny WiFi-Enabled ARM MCU For Tiny Projects", *Hackaday*, 2018. [Online]. Available: https://hackaday.com/2018/10/02/tiny-wifi-enabled-arm-mcu-for-tiny-projects/.
12. "M5Stack Official Store. ESP32 Basic Development Kit Core Expandable Micro Control Wifi BLE IoT Prototype Board for Arduino-in Industrial Computer and Accessories from Computer and Office on Aliexpress.com | Alibaba Group", *aliexpress.com*, 2019. [Online]. Available: https://ru.aliexpress.com/item/32837164440.html.
13. S. Santos, "ESP32 vs ESP8266 - Pros and Cons - Maker Advisor", *Maker Advisor*. [Online]. Available: https://makeradvisor.com/esp32-vs-esp8266/.
14. "The Internet of Things with ESP32", *Esp32.net*, 2019. [Online]. Available: http://esp32.net/.
15. "How do I interface ESP8266 to ARM?", *Quora.com*. [Online]. Available: https://www.quora.com/How-do-I-interface-ESP8266-to-ARM.
16. "How can we interface an STM32 F103 black pill with an ESP8266 for data transfer in Arduino IDE?", *Quora.com*. [Online]. Available: https://www.quora.com/unanswered/How-can-we-interface-an-STM32-F103-black-pill-with-an-ESP8266-for-data-transfer-in-Arduino-IDE.
17. "Configuring the Arduino IDE to program the ESP8266 WiFi module (Ukrainian)", *Geekmatic.in.ua*. [Online]. Available: http://geekmatic.in.ua/ua/arduino_ide_with_wifi_esp8266.
18. "ESP8266: What's inside the “people's WiFi”? (Russian)", *Habr.com*, 2014. [Online]. Available: https://habr.com/ru/company/coolrf/blog/238443/.

2.1.2 ESP32 module in CPS/IoT projects. Espressif IoT Development Framework software development environment (Laboratory Work № 1)

The aim of the laboratory work: gaining skills in the Espressif IoT Development Framework (Esp-IDF) software development environment (SDE), getting acquainted with the key features of the ESP32 module.

Recommended hardware and software

Name	Link
ESP32(DevKit)	https://github.com/playelek/pinout-doit-32devkitv1
ESPAsyncWebServer.h	https://github.com/me-no-dev/ESPAsyncWebServer
USB 2.0 cable type A/B	https://store.arduino.cc/usb-2-0-cable-type-a-b

Theoretical Information

Introduction. The basic CPS concept involves using significant computing resources for managing physical objects and processes and mathematical information processing. IoT technology provides the ability to transfer data using standard network protocols. The ESP32 (DevKit) platform was developed to implement such a complex of tasks. It is also attractive that in addition to the Eclipce and Esp-IDF environments, the Arduino IDE supports the software development for ESP32 crystals. Therefore, let's take a closer look at the possibilities of using the technologies provided to ESP32 by the developers for the implementation of CPS/IoT projects.

ESP32 module. The main computing element on board ESP32(DevKit) is the ESP-WROOM-32 module (ESP32) (fig. 1.1), which is already a small-sized printed circuit board (PCB) that contains the main chip (microprocessor with all interfaces), flash memory chip and antenna for WiFi.

The main characteristics of the module:

- microprocessor: Xtensa Dual-Core 32-bit LX6, 160 or 240 MHz;
- memory: 520 KByte SRAM, 448 KByte ROM;
- flash on the module: 1, 2, 4… 64 Mb;

Fig.2.4 – ESP-WROOM-32 module

Wireless technology:
- Wi-Fi: 802.11b/g/n/e/i, to 150 Mbps with HT40;
- Bluetooth: v4.2 BR/EDR and BLE;

Periphery interfaces:
- 12-bit SAR ADC up to 18 channels;
- 2 × 8-bit DAC;
- 10 × touch sensors;
- Temperature sensor;
- 4 × SPI;
- 2 × I²S;
- 2 × I²C;
- 3 × UART;
- 1 host (SD/eMMC/SDIO);
- 1 slave (SDIO/SPI);
- Ethernet MAC with support DMA and IEEE 1588;
- CAN 2.0;
- IR (TX/RX);
- Motor PWM;
- LED PWM up to 16 channels;
- Hall sensor;
- Ultra low power analog pre-amplifier;

Security:
- IEEE 802.11 security WFA, WPA/WPA2 и WAPI;
- Secure boot;
- Flash encryption;
- 1024-bit OTP, including up to 768-bit under the task;
- Cryptographic engine: AES, SHA-2, RSA, ECC, RNG.

As we can see, the ESP32 module is a powerful tool for realization of CPS/IoT projects, and it also has a large enough and powerful set of software components to work with. These methodological materials describe the work with the basic functionality of this system.

Setting up the programming environment. Before you can begin programming ESP32, you must first make some computer settings.

1. First of all we need to download the main tool for working with our board – this is our API interface, or as it is called the framework, which allows you to work with the hardware platform ESP32.

Download link: https://github.com/espressif/esp-idf

After downloading the framework we will briefly review its contents. In the folder "components" we see dozens of other folders, each containing a list of all components of our framework. In data files stored a set of functions that allow the programmer to control the interface device. These files are intended to be compiled with your application.

Next is the folder "docs", which stores compressed documentation on programming all system components, correct use of necessary features information on configuring development environments on different operating systems and other useful information.

Particular attention should be paid to the folder "examples", as it contains some real-life examples of programs that will be very useful for further study of the ESP32 module. It should also be noted that all projects written on this framework are created on the basis of make-up system, that is, each project will contain at least one MakeFile or .mk file which describes the rules of project assembly, compatible compilation of raw files of your program with the necessary for your project to raw framework files.

The "make" folder contains several .mk files that already describe the basic build rules for all projects on this system.

Folder "tools" contains the basic tools and utilities for automatic configuration of projects for different operating systems.

2. The fastest way to start developing for ESP32 is to install the necessary basic tools, such as:

- Compiler – without it, our program would be just a set of text files. This tool creates a binary firmware file from a written program, this file which is loaded into the ESP32 flash memory is the true appearance of the program.

- Windows OS doesn't have a built-in make environment, so we'll use MSYS2 (somewhat similar to the Linux terminal) to work with it. However, we will not need to work with this environment every time, as it will be automatically used by the Eclipse IDE development environment. MSYS2 will mainly be used to generate the configurations of our device, namely which COM port of the computer we have connected to the device, the processor speed in the ESP32 module, switching on/off and configuring certain peripherals of the module.

You can download this tool at the link: https://dl.espressif.com/dl/esp32_win32_msys2_environment_and_toolchain-20170918.zip

3. The next step is to download the Eclipse IDE development environment. Of course, you can use any other environment to your liking, you can even do without it and use only MSYS2. However, this framework is quite convenient to configure it to work with third-party programming microcontrollers and devices as well as Eclipse provides a convenient interface for working with project files, syntax highlighting language and navigate the elements of the program.

For downloading this environment, please visit: http://www.eclipse.org/downloads/packages/eclipse-ide-cc-developers/oxygen1a

4. We need to install a driver for this device so that our physical workstation understands that a specific device is connected to it via USB and that we can program it. For our board, we need to download driver “CP210x USB to UART Bridge VCP Drivers” by the link: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

5. Since all necessary software we already have, we can proceed to its configuration and programming ESP32. First of all, we will select an existing project from the examples presented in the framework and select a folder with the project "hello_world" (…\esp-idf\examples\get-started\ hello_world), copy it in random disk space where projects are stored. The project is written in C programming language and it will be used by us in the future as a ready template program which we will build all future projects. Then run environment msys2.exe (fig.2.5), located in the tools folder.

When the msys2 window opens, the first step is to go to the directory with our project using command **cd** < ***path*** > (fig.2.6). We check the contents of the folder with the **ls** command while already in this location. Now we specify the compiler the path to the framework

directory with the command **export IDF_PATH=<** ***path*** **>**, this is created in order to carry out that the compiler can compile the files of your project with the source code of framework.

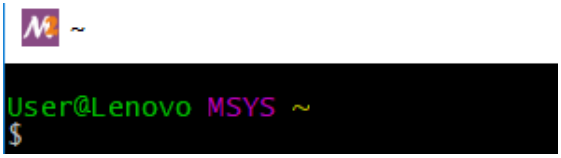


Fig. 2.5 – Window of the environment msys2.exe

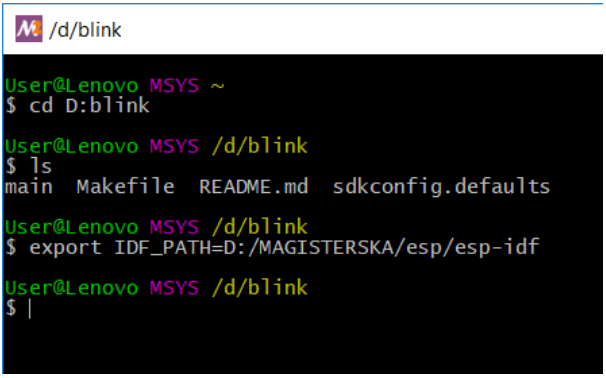


Fig.2.6 – Creating a link to an external resource

At this point, you need to run a special utility - the configurator, which allows you to generate a specific configuration of the project, which is stored in the **sdkconfig** file (fig.2.7). This file stores settings such as the port number to which the card is connected, the built-in programmer settings, and module peripheral settings. The launch of this utility is performed by command **make menuconfig**.

Now we go to the menu “Serial flasher config → Default serial port” and instead “/dev/ttyUSB0” write with which port we need to work with, such as "COM4". Of course, to check which port you want to select, you need to connect the board with a USB cable to your computer and check for an active COM port in Device Manager. Then click on "Exit" and return to msys2. From now, we can already download this sample project to the ESP32 module using the command **make flash**, but we instead move on to setting the programming environment Eclipse IDE.

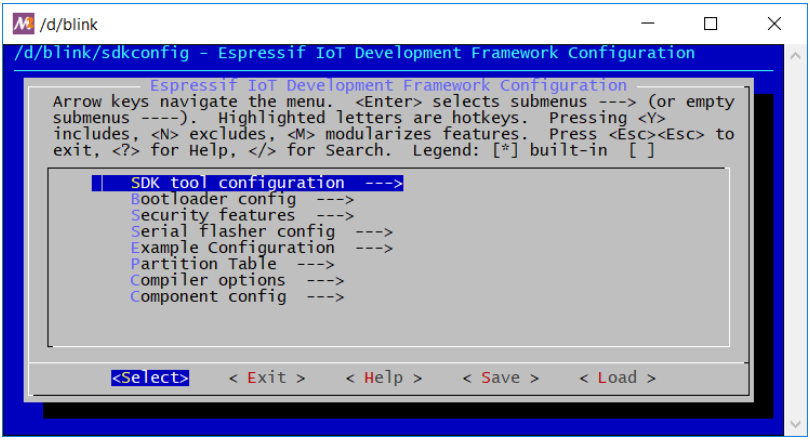


Fig.2.7 – The main window of configurator

6. Open Eclipse and execute the command **File → Import**. In the open window select **C/C++ → Existing Code as Makefile Project** (fig.2.8). In the next window, select **Cross GCC** and specify the path to the directory with the project and click **Finish**. Now our project is open in Eclipse environment, it remains to configure it to work with ESP32. Let's go to the project properties by right-clicking and selecting a command **Properties**. In the open window, go to the bookmark **C/C++ Build → Environment** and create environment variables **IDF_PATH, PATH, V** (fig.2.9). In the variable value **IDF_PATH** prescribe the path to the framework (*.../esp-idf*). It is important that this path be spelled with the character "/" instead of the usual "\". In the **PATH** variable, we list the paths to our tools (compiler and make-up system). In general, this variable should contain three values separated by a semicolon (*...\msys32\usr\bin; ...\msys32\mingw32\bin; ...\msys32\opt\xtensa-esp32-elf\bin*). It should be noted that all paths must be absolute. In the variable **V** enter the value 1.

We return to the tab **C/C++ Build** and uncheck the box **Use default build command** and in box **Build command:** we enter the following text: *bash ${IDF_PATH}\tools\windows\eclipse_make.sh* and click **Apply**.

Then we go to the tab **C/C++ General → Preprocessor Include Pat, Macros etc.** and open the bookmark **Poviders**. We choose an item **CDT GCC Built-in Compiler Setting Cygwin** and enter the following text in the box below: *xtensa-esp32-elf-gcc ${FLAGS} -E -P -v -dD*

"${INPUTS}". This text indicates the name of the compiler and its options for compilation. We do a similar action with an item **CDT GCC Build Output Parser** where we enter the following text: *xtensa-esp32-elf-(gcc)|([gc]\+\+)|(clang)* (fig.2.10).

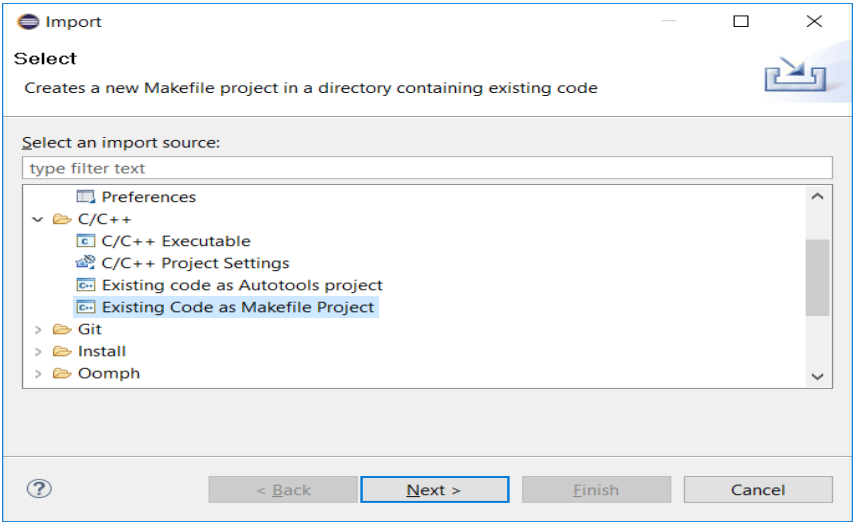


Fig.2.8 – Project type selection window in Eclipse IDE

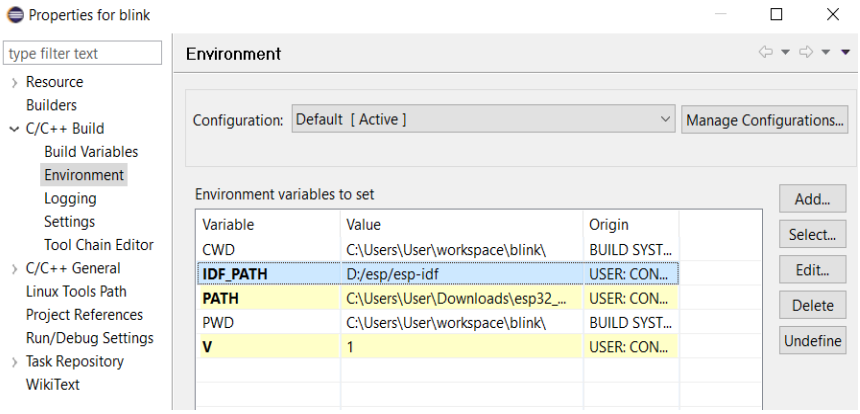


Fig.2.9 – Setting up internal project variables

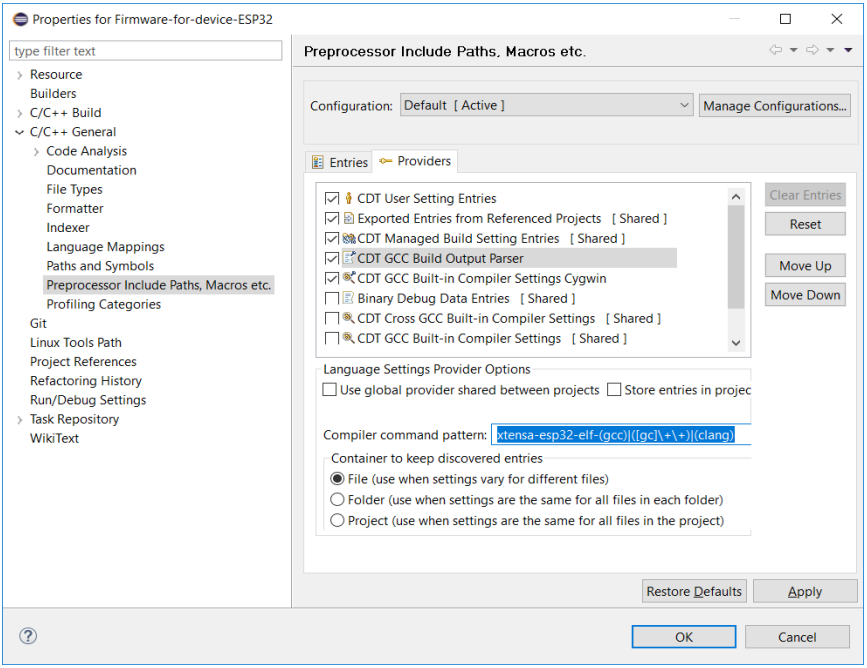


Fig.2.10 – The compiler setting

Now you have to specify the properties of the project by the specific components of the framework, that is, the environment where our search for files with the announcement of the functions we need for the relationship with the hardware capabilities of the module. As mentioned earlier, all the source code for the function declaration is in the corresponding folder in the “components” directory of the framework. Let's go to the tab **C/C++ General → Paths and Symbols**, open the bookmark **Includes** and click **Add**. In an open window, we need to spell out a path to a specific component, but since we already have a variable in the project that stores the path to the framework, we will use it and insert relative paths:

${IDF_PATH}/components/esp32/include
${IDF_PATH}/components/newlib/include
${IDF_PATH}/components/freertos/include
${IDF_PATH}/components/nvs_flash/include
${IDF_PATH}/components/driver/include
${IDF_PATH}/components/log/include

We'll have enough of these components to get started, but if others

suddenly need for the project, we'll just add them to the existing list. The next time you press **Add** tick the box **Is a workspace path** and prescribe the path to the folder where the files of our project lie. Click **Apply** and **OK**. The only thing left is to register commands for the make-up system. There will be three of them: to compile the project, to upload it to ESP32 and to clear it. We do right-click on the project and choose **Make Targets → Create…** and in the field **Target Name** we enter:

- **all** for compiling
- **flash** for downloading
- **clear** for cleaning

As a result, we should have three goals for building the project, as shown in Fig. 2.11.

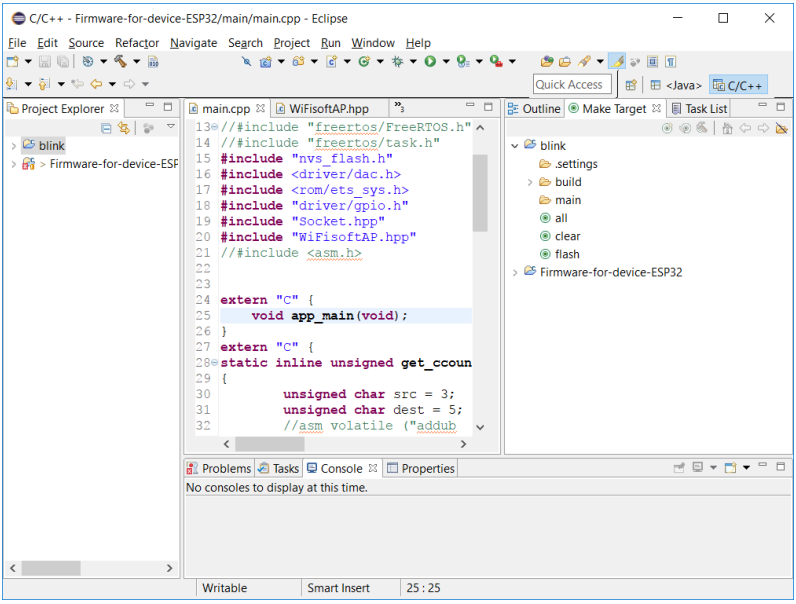


Fig.2.11 – Created goals for the project building

ESP32 programming. The ESP32 module allows you to write applications that can run directly on your device. You can compile programs written in C\C ++ and download them to a device on which they are already running. For your applications to perform some useful action, they must be able to interact with a particular environment, such as the local and global network (Internet) or even the nature. This can be

achieved by setting up a network connection and connecting certain sensors and output devices, which can include either a regular LED or LCD screen or powerful stepper motors. In order to use all the peripherals allowed for this board, ESP32 contains a basic set of functions through which we will programmatically work and exercise control over all our devices. This feature set can be represented as an internal operating system, which provides use our API interface to interact with all the hardware capabilities of the board. Any function may be called in your program at any time when the program is running and it guarantees the provision of the services for which it is intended. This entire interface is fully documented and described in the ESP32 software documentation – https://esp-idf.readthedocs.io/en/latest/api-reference/index.html. In order to successfully write credible applications, you need to know about the availability of specific API interfaces. For example, if we need to connect to a WiFi hotspot, we use specific features in a certain order, which gives us the API to work with the wireless network. Similarly, if we want to connect a third-party device to our board for a specific physical interface, we use the software implementation of the interface in the provided API.

In order to write reliable applications for CPS/IoT device, there is no need for a thorough study of the entire API, it is enough to know that it includes the implementation of the necessary functionality for your application and to be able to use it. This API (framework) was developed by the manufacturer of the ESP32 module and was called the "Internet of Things Development Fremework", better known as ESP-IDF.

1. *The entry point to the program*. When your application is downloaded to your device, it must start from a specific location. In your application, written on this framework, the entry point to the program is a function **int app_main(void)**. This function is analogous to the function **main()**.It is important to note that when you write normal programs running on a regular computer, the program runs from the beginning of the main () function (or defined entry point) to its end and ends there. If this scenario works in a program written for the embedded system, then the program will run in an instant and the rest of the time the device will simply consume current and perform nothing useful. So, the end of the program for embedded systems under normal conditions can be considered the moment when the device power is turned off or the device is simply restarted. In this case, the eternal loop is inserted into the main function, in which the basic work of the device is performed.

2. *Error handling*. Most of the features in the ESP-IDF that we call can return a specific error code if something unexpected happens. Most features result is data in type **esp_err_t**, which can be considered as an integer. If the function call is successful, then returns **ESP_OK** value. Any other value means an error. It is recommended to use a macro **ESP_ERROR_CHECK()** to handle these errors, into which, as a parameter, we can pass a call to a particular function. If our function returns a value other than **ESP_OK**, our program simply stops executing and the corresponding message is displayed on the console. We need to attach the file **<esp_err.h>**, to use this opportunity.

3. *Working with the console*. In general, to program our device, we use a USB cable connection to our computer. This connection on your computer looks like connecting a third-party device to a specific COM port. Downloading the application to ESP32 after setting up the environment will be performed automatically when the project is built, but we can also use this connection to a computer as a tool to display information on a computer monitor. Overall, our program already outputs some information to the PC console by default, but you can also output your own messages. To do this programmatically, we need to call a standard function **printf()**,with which we can work just like in C language:

```
int value = 7;
printf("value is %d \n", value).
```

To work with the console (COM port connection) on a PC, we'll need PuTTY, or some other equivalent. To open the console, we need to enter the appropriate port settings:

- Port number: COM <numbe>
- Transmission speed (bod\sec): 115200
- Information bits: 8
- Stop bits: 1
- Parity: None

These settings depend on the configuration of the device, which will be described below.

4. *Work with the connectors (pins) of the device.* The easiest way to work with input-output information when it comes for embedded systems is to use the GPIO interface (General Purpose Input Output). The purpose of this interface is to output or input binary information from a device or to a device. That is, the simplest algorithm for working with this interface is:

- Set connector for input or output:
  gpio_set_direction(GPIO_NUM_18, PIO_MODE_OUTPUT);
  or
  gpio_set_direction(GPIO_NUM_18, GPIO_MODE_INPUT);
  where **GPIO_NUM_18** is a macro, which means number 18, and **GPIO_MODE_OUTPUT** is a type list **gpio_mode_t**.
- Set the connector to logic zero, or logical unit if mode **GPIO_MODE_OUTPUT** is selected:
  gpio_set_level(GPIO_NUM_18, 0).
- Input mode connectors are a bit trickier and include the development of a feedback function (interrupt function) that needs to be described and logged into the embedded operating system. This function will work every time the signal level on the selected connector changes. An example of how this function works is shown in Listing 1.1 for laboratory work.

Generally, there are a number of additional GPIO features and macros, which an announcement and a description are in the header file **<driver/gpio.h>**.

5. *Built-in operating system FreeRTOS*. Our device, in terms of computing resources, is a 32-bit dual-core processor with a maximum clock speed of 240 MHz. To work with such resources with the ability to run the program concurrently and manage all possible interfaces, this API is built on the operating system FreeRTOS, which communicates with the hardware of the device, monitors the occurrence of certain events, organizes the work of interrupts and system calls. This system is slightly different from the operating system we are familiar with. It doesn't have any GUI elements, Task Manager, etc. Its main task is to allocate microprocessor resources between different program threads (if created) and create relationships with hardware capabilities of the module ESP32. To work with the capabilities of this system, you need to include in your program the required header file, as there are several of them and they are in the path …/esp-idf/components/freertos/ include/freertos. These files contain the necessary announcements of operating system functions and their descriptions. An example of a multi-threaded program using FreeRTOS is listed in Appendix B.

The Practical Part

1. Download and configure the Eclipse desktop and ESP-IDF framework.

2. Compile and download the “hello-world” project for the board by running the **flash** command.

3. View the result of running the program using any terminal program (PuTTY, Terminal or others) with the following COM port settings:

- Speed (baude): 115200
- Data bits: 8
- Stop bits: 1
- Parity: None
- Flow control: XON/XOFF

4. Make changes to the application by removing the automatic reset. Compile and download the application to your device. Describe the program.

5. Put all the code that is in the main function in the eternal cycle. Download the program and describe its behavior.

6. Using Listing Code 1.2, modify the program from item 4 so that its code runs in a separate thread.

7. Create a program that demonstrates the operation of the GPIO interface, such as the activation of the RGB LED in different colors at any interval (using a delay). The code for this program should be generated in a single thread (without the use of FreeRTOS) and in multi-threaded, where switching on/off of a single connector connected to the LED occurs in separate threads.

Report

A report on laboratory work is issued in the form of a short abstract, containing a title page formatted according to the requirements of paragraph 1.1.2; a task description, a brief results description of 2-3 pages; and 2-3 pages of answers to individual control questions. Protection of the results of the work done is carried out orally in the form of an interview with the teacher and answers to questions.

Test questions

1. Describe the characteristics of the ESP32. Which of them are the most important for the implementation of CPS/IoT projects?
2. Describe the structure of the ESP-IDF software environment?
3. Describe the procedure for creating a project based on the ESP32 module in the ESP-IDF for CPS?

4. Describe the features of programming ESP32 and compiling programs in C/C ++.
5. How to register the path to a particular project component in ESP-IDF?
6. What commands are used for the make-system and how are they written in the project?
7. Describe the purpose of the **int app_main (void)** function. In which CPS/IoT project scenarios can it be used?
8. How is error handling implemented in the ESP-IDF? What are the possible program actions for detecting errors?
9. Describe the features of working with the console. How to implement the output of your own messages on a PC?
10. Describe the process of setting up the output of information to individual contacts of the ESP32 board.
11. What are the features of working with connectors in the input signal mode?
12. Describe the main characteristics of the FreeRTOS embedded operating system, and the possibilities of its application in CPS/IoT projects based on ESP32.

Recommended literature

1. "M5Stack Official Store. ESP32 Basic Development Kit Core Expandable Micro Control Wifi BLE IoT Prototype Board for Arduino-in Industrial Computer and Accessories from Computer and Office on Aliexpress.com | Alibaba Group", *aliexpress.com*, 2019. [Online]. Available: https://ru.aliexpress.com/item/32837164440.html.
2. "The Internet of Things with ESP32", *Esp32.net*, 2019. [Online]. Available: http://esp32.net/.
3. "me-no-dev/ESPAsyncWebServer", *GitHub*, 2019. [Online]. Available: https://github.com/me-no-dev/ESPAsyncWebServer.
4. [8]"espressif/esp-idf", *GitHub*, 2019. [Online]. Available: https://github.com/espressif/esp-idf.
5. "Eclipse IDE for C/C++ Developers | Eclipse Packages", *Eclipse.org*, 2019. [Online]. Available: https://www.eclipse.org/downloads/packages /release/neon/2/eclipse-ide-cc-developers.

2.1.3 Creating a local WiFi network based on the ESP32 module (Laboratory Work № 2)

The aim of the laboratory work: gain practical skills in implementing and programming ESP32-based WiFi-networks for embedded systems in CPS/IoT projects.

Recommended hardware and software

Name	Link
ESP32(DevKit)	https://github.com/playelek/pinout-doit-32devkitv1
ESPAsyncWebServer.h	https://github.com/me-no-dev/ESPAsyncWebServer
USB 2.0 cable type A/B	https://store.arduino.cc/usb-2-0-cable-type-a-b

Theoretical Information

Introduction. With the advent of the IEEE 802.11 standard (Wi-Fi), wireless networks have become commonplace. However, this technology has long been used only in the field of personal computers and mobile devices. With the development of embedded systems and the emergence of industries such as CPS/IoT, there was a need for cheap and energy-efficient solutions that could fully meet the standard and interact with any other device.

WiFi. Before you go into creating a WiFi-based device, you need a certain understanding of the concepts behind this technology. At a high level, WiFi is an opportunity to participate in a connection based on a stack of TCP / IP protocols through to high-frequency radio communications. In general, the name WiFi is just a trademark, which means a set of protocols described in the architecture of the IEEE 802.11 wireless LAN. Devices that participate in the process of sharing information over a network are classified in this architecture:

-*Access point*. It usually connects (or acts as) as a TCP/IP router to the rest of the TCP/IP network. Very often, the access point also has a network connection to the Internet and serves as a bridge between the wireless network and the wider TCP/IP network, which is the Internet.

-*Station.* This is a device that connects to an access point and operates on the network solely through a connected access point. That is, in this mode, all incoming and outgoing messages from one station to another

move through the access point. The station can only be connected to one access point at a time.

When several access points operate around a station, the station must know which one to connect to. Each access point has its own network identifier called a BSSID (more commonly used SSID). SSID is a unique name for a wireless network that distinguishes one WiFi network from another. In fact, it is a 32-character value, which is a target for sending packets of information through a created network.

1.*WiFi initialization*. The ability to create and interact with WiFi networks is just one of the many features of ESP32. This way you can only turn on and use WiFi when your device needs it. To allow the program to work with this capability, the header file **<esp_wifi.h>** must first be included in the project and the function **esp_wifi_init()** must be called. The recommended way to do this:

```
wifi_init_config_t config = WIFI_INIT_CONFIG_DEFAULT();
 esp_wifi_init(& config);
```

In this example, **wifi_init_config_t** is a structure that contains presets to initialize the WiFi environment. We used a macro that sets the default object of this structure.

2.*Select a mode for the device*. An ESP32-based device can act as an access point for other devices as well as a station that will connect to a particular access point, or operate in both modes simultaneously, allowing you to organize a point-to-point network. The mode is selected using the function **esp_wifi_set_mode()**. The parameter of this function is a type list **wifi_mode_t**, which may have the values specified in the table 2.1.

3.

Table 2.1 – Lists of the **wifi_mode_t** values specified

Function	Mode
WIFI_MODE_NULL	no mode
WIFI_MODE_STA	station mode
WIFI_MODE_AP	access point mode
WIFI_MODE_APSTA	station mode + access point mode

4.*Connection to the access point*. If our device is going to serve as a station, we will have to connect to an access point. We can ask for a list of available access points to which we can try to connect. This is done using the function **esp_wifi_scan_start()**.The scan results are stored in a dynamically allocated storage in the module memory. The data is

returned to us when we call **esp_wifi_scan_get_ap_records()** as a structure object **wifi_ap_record_t**, which contains information such as the SSID of the access point, its authentication mode, and more.

Once a scan request is issued, we will be informed of the event's completion with an event ID **SYSTEM_EVENT_SCAN_DONE** (regarding ESP32 events, their concepts and their processing will be discussed below). The data associated with this event contains the number of access points, but can also be accessed through a call **esp_wifi_scan_get_ap_num()**.If we want to cancel the scan before it ends on our own, we can call **esp_wifi_scan_stop()**.Example showing the access point search provided in Listing 2.1 for this laboratory work.

5.*Handling system events related to WiFi*. There may be some external events that may occur when using a WiFi device. For example, connecting a third-party device to your access point, or turning off the third-party access point that your device was communicating with. Handling these events is critical to creating reliable software for your device, as they can occur at any point in time that is unknown beforehand and interrupts the current program execution. The occurrence of event data creates an interruption, which in programming is called the emergence of system calls and their processing (handling) is part of the code that will be executed after its occurrence. Such interrupts are followed by the built-in FreeRTOS operating system, which either redirects the current execution of the program to the event handler, or as it is called a callback function.

In order to programmatically handle such situations, we first need to create this function, which will be called when they occur, and register the function in the operating system of the device. It should be noted that the origin and processing of events is not only work with WiFi, but also with many other works of the hardware board. To register the handler function on the system, you must call **esp_event_loop_init()**.We pass the pointer to the handler function (its name) as the first parameter and the null pointer (NULL) as the second parameter. The callback function must match a particular sample:

```
esp_err_t function_name (void *ctx, system_event_t *event)
{
// processing code...
return ESP_OK;
}
```

In order to create and register this function, you need to include

some header files for compatible compilation:

-**#include <esp_event.h>**

-**#include <esp_event_loop.h>**

-**#include <esp_wifi.h>**

-**#include <esp_err.h>**

When this function is called by the system, we get as a parameter some information related to this event. The type of this parameter **system_event_t**, which contains:

-**system_event_id_t event_id** is the unique identifier of the event that triggered the handler. Table 2.2 lists the identifiers and what they mean;

-**system_event_info_t event_info** – is a C language association that can contain different structures of information at some point, depending on the event that occurred. Table 2.3 lists the structures, the names of the objects of these structures in event_info, and at what event we can obtain this information.

Table 2.2 – Lists of the identifiers and what they means

Event_id	Event
SYSTEM_EVENT_STA_START	Start in station mode
SYSTEM_EVENT_STA_STOP	End in station mode
SYSTEM_EVENT_STA_CONNECTED	Connection to the access point
SYSTEM_EVENT_STA_DISCONNECTED	Disconnect from access point
SYSTEM_EVENT_STA_AUTHMODE_CHANGE	Authentication mode changed
SYSTEM_EVENT_STA_GOT_IP	An IP address was obtained from the access point
SYSTEM_EVENT_AP_START	Start in access point mode
SYSTEM_EVENT_AP_STOP	End in access point mode
SYSTEM_EVENT_AP_STACONNECTED	Connecting station
SYSTEM_EVENT_AP_STADISCONNECTED	Disconnecting station
SYSTEM_EVENT_AP_PROBEREQRECVED	We received a test request when we are an access point

Table 2.3 – The structures and names of the objects in event_info

Structure	The object name	Event
system_event_sta_connected_t	connected	SYSTEM_EVENT_STA_CONNECTED
system_event_sta_disconnected_t	disconnected	SYSTEM_EVENT_STA_DISCONNECTED
system_event_sta_scan_done_t	scan_done	SYSTEM_EVENT_SCAN_DONE
system_event_sta_authmode_change_t	auth_change	SYSTEM_EVENT_STA_AUTHMODE_CHANGE
system_event_sta_got_ip_t	got_ip	SYSTEM_EVENT_STA_GOT_IP
system_event_ap_staconnected_t	sta_connected	SYSTEM_EVENT_AP_STACONNECTED
system_event_ao_stadisconnected_t	sta_disconnected	SYSTEM_EVENT_AP_STADISCONNECTED
system_event_ap_probe_req_rx_t	ap_probereq recved	SYSTEM_EVENT_AP_PROBEREQRECVED

It is important that when a particular event occurs, we take the appropriate merge field that fits a particular structure. Below is an example of an event where we display an IP address message in the console to receive an IP address. We use a macro **IPSTR** to get the IP address, which is simply a string **“%d.%d.%d.%d”**, and a macro **IP2STR**, which expands the **ip** field to four integers:

```
if(event->event_id == SYSTEM_EVENT_STA_GOT_IP)
{
   printf("Our IP address is " IPSTR
"\n", IP2STR(&event->event_info.got_ip.ip_info.ip));
}
```

6.*Operation of the device in station mode*. When we think of our device as a WiFi station, we understand that it can only be connected to one access point at any one time. In other words, it makes no sense to say that the device is connected to two or more access points at a time. The access point we want to connect to is set in the data structure **wifi_sta_config_t**. This structure contains two very important fields called **ssid** and **password**. The field **ssid** is the SSID of the access point we connect to. The password field is a clear text value for the password

that will be used to authenticate our device to the destination access point to allow the connection. An example of initialization of this structure:

```
wifi_config_t staConfig = {
.sta = {
.ssid="<access point name>",
.password="<password>",
.bssid_set=false
}
}.
```

Once our structure is initialized, we can provide our device with this data to connect to the access point:

```
esp_wifi_set_config(WIFI_IF_STA,
(wifi_config_t *)&staConfig).
```

Do not forget that you must first set the appropriate mode of operation of the device:

```
esp_wifi_set_mode(WIFI_MODE_STA).
```

Finally, we call the function **esp_wifi_connect()**to connect to the access point. After this function is called, several events will occur after a while, such as connecting to the access point and obtaining the IP address from the point, if it has not been set statically. Only after processing these events are we allowed to participate in the transmission of information over the network. In order to disconnect from the access point, we use the function **esp_wifi_disconnect()**.

There is another issue related to access point connection, and this is the idea of automatic connection. There is a Boolean flag stored in the flash memory that indicates whether ESP32 should attempt to connect automatically to the last access point. If the value set to True, the device will immediately attempt to connect to the last access point used when the device is started. We can enable or disable the auto-connect feature by calling **esp_wifi_set_auto_connect()**.For an example of how to set the unit to station mode, see Listing 2.2 for lab work.

7.*Operation of the device in access point mode*. This mode will allow you to connect to our device. To be an access point, we need to define an SSID that allows other devices to differentiate our network. This SSID can be marked as hidden if we do not want it to be visible during scanning. In addition, we will also need to provide an authentication mode that will be used when the station wishes to contact us. This is used to permit authorized stations and ban unauthorized. Only stations that know our password will be allowed to connect. If we use

authentication, we will also need to choose a password that other devices will need to know to successfully connect.

First of all, we need to set the access point mode:

```
esp_wifi_set_mode(WIFI_MODE_AP);
```

The next step, as in station mode, is to configure the device:

```
wifi_config_t apConfig = {
.ap = {
.ssid="<access point name>",
.ssid_len=0,
.password="<password>",
.channel=0,
.authmode=WIFI_AUTH_OPEN,
.ssid_hidden=0,
.max_connection=4,
.beacon_interval=100
 }
}.
```

Each of the fields in this structure is responsible for the specific characteristics of the access points, which are described in Table 2.4.

Table 2.4 – Specific characteristics of the access points

Field name	Characteristic
ssid	Network ID
ssid_len	The ssid length in bytes. Exposed at NULL
password	Authentication password
channel	The channel we will use to build the network
authmode	Authentication mode
ssid_hidden	Visibility of ssid. The default is NULL
max_connection	Maximum number of access point connections (4)
beacon_interval	The default is 100

After completing the configuration structure, we must notify the device of this configuration:

```
esp_wifi_set_config(WIFI_IF_AP, &apConfig);
```

All you have to do is turn on WiFi on your device:

```
esp_wifi_start();
```

When operating in this mode, we can determine how many stations are currently connected to our device. This is done by calling the

function **wifi_softap_get_station_num()**. If we want to know the details of the connected stations, we can call **wifi_softap_get_station_info()**, which will return a linked list **wifi_sta_list_t**. After receiving this information, we need to clear the memory space with help **wifi_softap_free_station_info()**.Below is an example of getting this information:

```
uint8 stationCount = wifi_softap_get_station_num();
os_printf("stationCount = %d\n", stationCount);
wifi_sta_list_t *stationInfo = wifi_softap_get_station_info();
if (stationInfo != NULL)
{
while (stationInfo != NULL)
{
os_printf("Station IP: %d.%d.%d.%d\n",
IP2STR(&(stationInfo->ip)));
stationInfo = STAILQ_NEXT(stationInfo, next);
}
wifi_softap_free_station_info();
}
```

When ESP32 acts as an access point, it allows other devices to connect to it and make connections. However, it emerges that two devices connected to the same ESP32 acting as an access point cannot interact directly with each other. For example, imagine two devices that connect to our access point. They can be allocated by IP-address 192.168.4.2 and 192.168.4.3 (192.168.4.1 is the default access point address). We can imagine that 192.168.4.2 could communicate with 192.168.4.3 and vice versa, but this is not allowed. It seems that direct networking is only allowed between new connected stations and an access point (ESP32). This seems to limit the use of ESP32 as an access point. The main use of ESP32 as an access point is to allow mobile devices (such as your phone) to connect to ESP32 and talk to an application running on it.

The Practical Part

1.Compile and download the program to the board from listing 2.1, which puts mode "Station" and searches for access points. Analyze how this code works.

2.Modify the application so that after successful search for access points, the program displays all the available access point names in the console.

3.Then the group split into two teams. The first team of students on the basis of code provided in Appendix B is developing a program that will do:

- At the event that characterizes the connection of the device to an access point, displays all available information about the access point and lights the RGB LED in green.
- At the event that characterizes obtain the IP address of the access point displays this address in the console and RGB-LED lit in yellow.
- When disconnecting the device from the access point, programmatically determine the cause of the disconnection and display a message about the cause in the console and illuminate the RGB LED in red..

4. A second team of students develops a program that puts the device in "access point" mode and gives it to another team. Creates a callback function in which:

- In the event that characterizes the access point starts, displays a corresponding message in the console.
- At the event that characterizes the connection of a third-party device, display information about this device in the console and at the same time illuminate arbitrary colors of RGB-LED at an arbitrary interval.
- When disconnecting an external device from this access point, display the corresponding message and stop the RGB LED from flashing.

5. Compile application data and upload it to the appropriate boards. Check the result of executing these programs using an optional terminal program (console). Third-party devices may include smartphones and existing WiFi routers.

6. The team that created the device, which acts as an access point, gives its name and password to the team that created the device "station". The second team modifies its code so that their device with an existing access point of other team, trying to connect to their device.

7. Check the compatibility of devices when different types of events occur, including turning off the access point, or changing the settings of the access point. On the results of execution try to make conclusions.

Report

A report on laboratory work is issued in the form of a short abstract, containing a title page formatted according to the requirements of paragraph 1.1.2; a task description, a brief results description of 2-3 pages; and 2-3 pages of answers to individual control questions. Protection of the results of the work done is carried out orally in the form of an interview with the teacher and answers to questions.

Test questions

1. Describe the Wi-Fi devices classification by their functionality in CPS/IoT projects?
2. How to initialize a Wi-Fi access point for IoT implementation?
3. How is ESP32 mode programmed?
4. Describe the technology for connecting a Wi-Fi station to an access point.
5. Describe the function of **esp_wifi_scan_get_ap_records ()**.
6. How does the event switching of system elements on the WiFi network occur? What is the FreeRTOS role in this?
7. Describe the features of the event identifier function **system_event_id_t event_id event id**.
8. Describe the features of function **system_event_info_t event_info**.
9. Analyze the features of the Wi-Fi device in “workstation” mode.
10. Analyze the features of the Wi-Fi device in “access point” mode.
11. How to determine how many stations are currently connected to a particular device?
12. What are the ESP32 limitations of working with many stations?

Recommended literature

1. "M5Stack Official Store. ESP32 Basic Development Kit Core Expandable Micro Control Wifi BLE IoT Prototype Board for Arduino-in Industrial Computer and Accessories from Computer and Office on Aliexpress.com | Alibaba Group", *aliexpress.com*, 2019. [Online]. Available: https://ru.aliexpress.com/item/32837164440.html.
2. "The Internet of Things with ESP32", *Esp32.net*, 2019. [Online]. Available: http://esp32.net/.
3. "me-no-dev/ESPAsyncWebServer", *GitHub*, 2019. [Online]. Available: https://github.com/me-no-dev/ESPAsyncWebServer.

2.1.4 Implementation of the TCP/IP protocol stack for Wi-Fi data transfer in embedded LWIP systems. Socket technology (Laboratory Work № 3)

The aim of the laboratory work: to explore TCP/IP stack workflow and Wi-Fi network communication for CPS/IoT; get socket programming skills.

Recommended hardware and software

Name	Link
ESP32(DevKit)	https://github.com/playelek/pinout-doit-32devkitv1
ESPAsyncWebServer.h	https://github.com/me-no-dev/ESPAsyncWebServer
USB 2.0 cable type A/B	https://store.arduino.cc/usb-2-0-cable-type-a-b

Theoretical Information

TCP/IP is a systematic stack of network protocols that is divided into four levels acoording to the OSI reference model. For CPS/IoT device, these are protocols that ESP32 understands and uses as rules for transporting data over WiFi. Addressing these rules is by using IP addresses. When we talk about TCP / IP, we are not only talking about TCP protocol that works over IP, but we actually use it as a stack for basic protocols such as IP, TCP and UDP, as well as additional relevant protocols such as DNS, HTTP, FTP, Telnet and more.

LWIP. If we are considering TCP/IP protocol, then we can split it into twodifferent layers: hardware (physical) and software (cyber). Typically, TCP/IP is implemented in software and is intended to collect bits of data from the physical environment at one end of a given protocol stack, and at the other end to reproduce this data in a form already understood by the main program, or even by humans.

One such software implementations exist in ESP-IDF and called LWIP (Light weight IP Stack). This implementation of TCP/IP for ESP32 provides us with the following services: IP, ICMP, IGMP, MLD, ND, UDP, TCP, Sockets API, DNS.

A TCP connection is a bidirectional transmission channel through which data can flow in both directions. Before connecting one side acts as a server - a passive listening for incoming connection requests. The server will simply stall until a connection request. The other side of the

connection is responsible for initiating the same connection. Once created, both sides will be able to send and receive data. In order for a client to request a connection to the server, it must know the address of the server. This address consists of two different parts. The first part is the server IP address and the second part is the client port number. If you make an analogy with your computer, you may have a lot of programs running on it, each of which can receive an incoming connection. Just knowing the IP address of your PC is not enough to access the program you need on this PC. The combination of IP address and port number provides all the necessary address.

We can now look at the settings and organization of the information transmission created by the network. To do this, we need to understand socket technology.

Socket technology. A socket is a programming interface for working with a TCP/IP network. Depending on the use of a particular transport protocol, sockets can be programmed to work with TCP or UDP. In this lab, we will look at how sockets work for TCP connections.

For the organization of the server TCP connections we will need to:

1. Create a TCP socket.
2. Associate the local port with the socket.
3. Set the socket to listening mode.
4. Confirm new connection, if available.
5. Receive, and / or send data.
6. Close the connection.
7. Return to Step 4.

For the organization of the client TCP connections we will need to:

1. Create a TCP socket.
2. Connect to a TCP server.
3. Receive, and / or send data.
4. Close the connection.

Access to work with LWIP and sockets is in the file **<lwip/sockets.h>**. For client and server applications, the process for creating sockets is the same. This is a function call **socket()**:

int sock=socket(AF_INET, SOCK_STREAM, IPPROTO_TCP).

The result of this function is an integer that is now associated with the socket just created and for which we will refer further to the socket. If this function returns "-1", then this indicates an error. When we create a server-side socket, we want it to wait for incoming connection requests. To do this, tell the socket which TCP / IP port number it should

listen to. Please note that we do not provide a port number directly with numeric values. Instead, we provide a value that returns a function **htons()**.In this unit, only one program can be associated with a certain number of the local port. If we want to associate a port number with an application, for example, our server application, in this case we perform a task called "binding", which binds (or assigns) a port number to a socket, which in turn belongs to this program. This is done using the function **bind()**:

```
struct sockaddr_in serverAddress;
serverAddress.sin_family = AF_INET;
serverAddress.sin_addr.s_addr = htonl(INADDR_ANY);
serverAddress.sin_port = htons(portNumber);
bind(sock, (struct sockaddr *)&serverAddress,
    sizeof(serverAddress)).
```

When the socket is now connected to the local port number, we can start listening to incoming connections. We do this by calling a function **listen()**.When we access this function, the server will start listening to the port for connection requests:

```
listen(sock, backlog);
```

backlog is a parameter that specifies the number of connection requests that can occur at one time. Those connections that occur after a given number of existing connections are simply discarded by the server.

Now, from a server perspective, we are ready to do some work. The server application can now block incoming client connections. The bottom line is that the purpose of a server is to process client requests, and when it does not have an active client, it has nothing left to do but wait to receive a request from a new client. That is, there is some blocking of the server program until you receive a connection request. Of course, this makes our executable program dependent on client connectivity, which we cannot predict. This model of sockets is not unique, there are asynchronous connection models, but we will not consider them in this lab.

When our server receives a request to connect to the program, you need to get information about a new client. This is done by calling a function **accept()**:

```
struct sockaddr_in clientAddress;
socklen_t clientAddressLength = sizeof(clientAddress);
int clientSock = accept(sock, (struct sockaddr *);
&clientAddress, &clientAddressLength).
```

In this function, we pass as parameters our socket, a pointer to the structure, which will then contain information about the client and the size of the given structure. The value returned by this function is a new socket, which is a newly created client-server connection. It should be noted that the pre-created socket is a function **socket()** is still active and is used to listen to connections, while the latter is used for data sharing. Like all TCP connections, the newly created one is also symmetrical and bidirectional in relation to the server and the client, so no server and client concepts exist after the connection has been established.

If we want to create a client socket, the procedure for creating a client socket is similar to a server program. Again, we create a socket function **socket()**, but in this case we don't have to use the functions **bind()/listen()/accept()**. All you have to do is call the function **connect()** to connect to a server that is waiting for a connection:

```
struct sockaddr_in serverAddress;
serverAddress.sin_family = AF_INET;
inet_pton(AF_INET, "192.168.1.200",
&serverAddress.sin_addr.s_addr);
serverAddress.sin_port = htons(9999);
int rc = connect(sock, (struct sockaddr *)&serverAddress,
sizeof(struct sockaddr_in)).
```

Once we have a connection, it remains to receive and transmit information through sockets. Information is obtained by means of function **recv()**:

```
char *data = new char[1];
ssize_t sizeRead = recv(clientSock, data, 1, 0);
if (sizeRead < 0);
printf("Error with size reading\n");
printf("%d\n", data[0]).
```

As parameters, the function accepts:

- socket number, which is a bidirectional link;
- a memory pointer that stores information retrieved over the network;
- the amount of memory in bytes in which the information will be written;
- a checkbox indicating that incoming messages are blocked. The default is NULL.

The return value of a function is the size in bytes of information received at one time.

Function for information transfer **send()**:

```
char *data = new char[1];
data[0] = 'A';
ssize_t sizeRead = send(clientSock, data, 1, 0);
if (sizeRead < 0);
printf("Error with size reading\n").
```

This function looks the same as **recv()**. It has similar parameters and a return value. The only difference is that we send data transmitted through the pointer **data**, but not receive.

In general, these functions are not the only ones used to send and receive information over the network, but are the most commonly used. Below is a description of the functions that are often used when using sockets and network programming.

1. The function that receives the incoming request: **int accept(int socket_fd, struct sockaddr* addr, socklen_t* addrlen).**

This function is blocking and is waiting for a request for incoming communication from the server socket. As soon as the client connects, his address is returned to us along with its length. If we try to accept too many connections at one time, ESP32 may return ENFILE to indicate that we have overflow connections.

Function returns socket for client, or -1 in case of error.

The file: **<lwip/sockets.h>.**

2. The function that associates the socket with the address: **int bind(int s, const struct sockaddr* name, socklen_t namelen).**

The **name** parameter is the socket address that will be associated with the socket; the **namelen** provides address length.

If the value returned is less than zero, then this indicates an error.

The file: **<lwip/sockets.h>.**

3. The function that closes the socket: **int close(int s)** or **closesocket(int s).**

s – this is an existing socket.

The file: **<lwip/sockets.h>.**

4. The function that connects to the server: **int connect(int sockFd, const struct sockaddr* partnerAddr, socklen_t addrlen).**

This function is usually called by the client.

The file: **<lwip/sockets.h>.**

5. The function that sets the socket property: **fcntl(int s, int cmd, int val).**

For example, to install a non-blocking socket: **s** is a socket, **cmd** – **F_SETFL, val** – **O_NONBLOCK.**

The file: **<lwip/sockets.h>.**

6. Function that searches for a host by its name: **struct hostent *gethostbyname(const char *name).**

It returns a pointer to a structure **hostent**.

The file: **<lwip/netdb.h>.**

7. The function that converts the integer port number in a sequence of bytes, which is associated with a socket port number: **uint32_t htonl(uint32_t netLong)** for the number of type **long, uint16_t htons(uint16_t hostShort)** for the number of type **short.**

The file: **<lwip/sockets.h>.**

8. The function that performs structure conversion IP-addresses in a string of characters: **char *inet_ntop(int af, const char* src, char* dst, socklen_t size).**

The **af** indicates the address family. It can take on value:

AF_INET – this is **IPv4;**

AF_INET6 – this is **IPv6;**

src – this is a pointer to the address structure;

dst – this is a buffer that will be filled with text;

size – this is the length of this buffer that can be filled. For **AF_INET** the buffer must be by length **INET_ADDRSTRLEN** bytes, and for **AF_INET6** it should be at least **INET6_ADDRSTRLEN** bytes long.

The file: **<lwip/sockets.h>.**

9. The function which is listening incoming connections: **int listen(int socket_fd, int backlog).**

The file: **<lwip/sockets.h>.**

10. The function that performs data acquisition on the net: **ssize_t read(int s, void* mem, size_t len).**

It is similar to the function **recv()**.

The file: **<lwip/sockets.h>.**

11. The function for receiving data over the network: **ssize_t recv(int s, void* mem, size_t len, int flags).**

This function returns the number of received bytes. A value of -1 indicates an error. A value of zero indicates that the connection is closed. Where: **s** is socket number, which is a bidirectional link; **mem** is a memory pointer in which is recorded information that obtained through the network; **len** is the amount of memory in bytes in which the

information will be written; **flag** is a flag that indicates to block incoming messages. The default is NULL. It may mean:

MSG_DONTWAIT – do not block waiting for a message that has not yet been received;

MSG_OOB – check that the message is out of range;

MSG_PEEK - get the first message without wasting it from memory.

The file: <**lwip/sockets.h**>.

12. The function that sends a byte set through the network: **ssize_t send(int s, const void* dataptr, size_t size, int flags).**

The data specified in **dataptr** for the size **size** bytes is transmitted by the network.

The file: **<lwip/sockets.h>.**

13. The function that implements sending a message through the network: **ssize_t sendmsg(int s, const struct msghdr* msg, int flags).**

To send, you must complete the structure **msghdr**.

The file: **<lwip/sockets.h>.**

14. The function that performs sending data over UDP protocol: **ssize_t sendto(int sock, const void* dataptr,** where **size_t size, int flags, const struct sockaddr* to, socklen_t tolen); sock** is a socket that was created by the function **socket(); dataptr** is a pointer to the data to be transmitted; **size** is a size of the datagrams in bytes (the maximum size is 64 Kb); **flag** is a flag that can take on a value **MSG_DONTWAIT** or **MSG_OOB; to** is a pointer to a structure containing the address of the recipient of the information; **tolen** is a length of structure with address.

The file: **<lwip/sockets.h>.**

15. The socket creation function: **int socket(int domain, int type, int protocol); domain** is the default in **AF_INET**; **type** is a value **SOCK_STREAM** or **SOCK_DGRAM** or **SOCK_RAW**; **protocol** is a value **IPPROTO_IP** or **IPPROTO_TCP**, or **IPPROTO_UDP.**

The result that is returned is the just created socket. A value less than zero indicates an error.

The file: **<lwip/sockets.h>.**

The Practical Part

1. The group is divided into two teams. The first team creates a program that configures the device to work with the network as an “access point”. The second team creates a program to work with the

network in the “station” mode (see lab 2). You must compile your applications and test their performance for network events (device connectivity, address retrieval, etc.).

2. You must add the socket algorithm to your programs. In each program it must be one, the number of concurrent requests is not less than two at a time. Use socket protocol for TCP. The team that develops the "station" creates a socket for work in client mode. Another team configures the socket to work in server mode (listening mode).

3. To test the sockets, you can use the Android application "Socket Protocol".

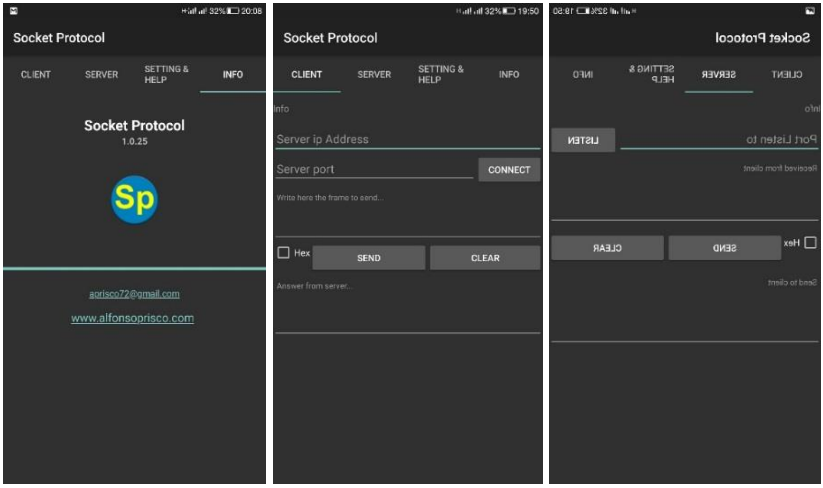


Fig.2.12 – Socket Protocol main windows

This program works both in client mode and in server mode.

It should be recalled that if your device (test stand) is operating in access point mode, its default address is 192.168.4.1.

If the device is in "station" mode, then it already needs to know the address of the device (server or access point) to which it will try to connect, i.e. in the case of "Socket Protocol" application in server mode, you need to configure your Android device as an access point and find out his address on his own network.

4. You must test the performance of your applications by sending arbitrary bytes from one device to another. If the device works as an access point, you must display the message as received in the console.

5. The team, whose device works with the network as an access point, organizes the control of the RGB LED by sending certain commands over the network. The format of commands is arbitrary. The number of commands is also arbitrary, but there must be at least three commands to enable each color.

6. The team, working with a “station”, agrees the format command with another team and organizes the work of your device so that intervals of two seconds of fame random command device to device "access point".

7. Check the work of the created devices. Try to connect to the access point device at the same time, both from your Android device and from the "station" device, and at the same time send commands to control the LED.

Report

Reporting and defense of laboratory work is performed according to the requirements described in work 2.

Test questions

1. Describe the TCP/IP protocol stack hierarchy?
2. What services does LwIP provide in ESP-IDF for organizing data exchange in embedded systems?
3. What is the essence of socket technology?
4. Describe the technology for organizing the server with a TCP connection.
5. Describe the technology for organizing client work with TCP connections.

Recommended literature

1. "M5Stack Official Store. ESP32 Basic Development Kit Core Expandable Micro Control Wifi BLE IoT Prototype Board for Arduino-in Industrial Computer and Accessories from Computer and Office on Aliexpress.com | Alibaba Group", *aliexpress.com*, 2019. [Online]. Available: https://ru.aliexpress.com/item/32837164440.html.
2. "me-no-dev/ESPAsyncWebServer", *GitHub*, 2019. [Online]. Available: https://github.com/me-no-dev/ESPAsyncWebServer.

2.1.5 Applying ESP32 for working with sensors and peripherals in CPS/IoT projects. GPIO PCF8574 extender for sensor networks (Laboratory Work № 4)

The aim of the laboratory work: to study the features of using the I2C interface in ESP32 for data exchange with sensors of various types; get practical skills in programming sensor networks.

Recommended hardware and software

Pressure and temperature sensors BMP180, light sensor BH1750, GPIO expander PCF8574, accelerometer and gyroscope MPU-6050, PWM servo controller on PCA9685, OLED display SSD1306, MAX7 LEDs matrix MAX7219/MAX7221, laboratory research module (LRM), Eclipse and Arduino IDE programming environment.

Theoretical Information

Physically, the **I2C (Inter-Integrated Circuit) interface** is a serial data bus developed by Philips in the early 1980s to control low-speed electronics peripherals from motherboards, embedded and mobile systems. In ESP32 I2C is implemented according to the standard scheme [1]. Devices connected using I2C (Fig.2.13) are divided into one master device (Master) and the rest of the slave devices (Slaves).

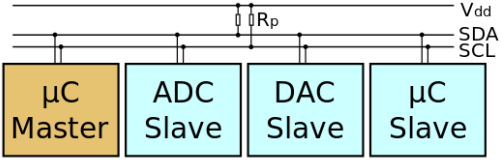


Fig.2.13 – I2C interface wiring diagram

Classical addressing includes a 7-bit address space with 16 reserved addresses; it allows connecting up to 112 different devices using two communication lines. When the address is considered by all slave devices, only the address of one of them must match the one supplied by the master. Other devices ignore the request.

I2C programming. The ESP-IDF development kit provides a driver that allows you to control the operation of the I2C interface using C language functions without resorting to low-level register manipulation. In ESP32, this interface is implemented in hardware, and

the drivers significantly simplify working with I2C. ESP32 can act both as a Master and as a Slaves device. It has two independent I2C ports (**I2C_NUM_0** and **I2C_NUM_1**) so two independent buses can be simultaneously implemented, being in different or identical roles. In many microcontrollers, the connectors where a particular interface port is running are hardware-fixed. In ESP32, there is no such restriction, so two module pins for the port can be chosen which are more convenient. To do this, one needs to decide which pin will be responsible for the SDA line, and which – for SCL.

To configure I2C, we must first call the function **i2c_param_config ()**, which accepts as the parameters which port we will configure, and the structure with the configuration. This structure has the following fields:

- **mode** – the role that our device plays when working with I2C. It can be **I2C_MODE_MASTER** for the master or **I2C_MODE_SLAVE** for the slave;
- **sda_io_num, scl_io_num** – determine which contact numbers will match specific port lines;
- **sda_pullup_en, scl_pullup_en** – these two fields determine whether selected contacts are pulled to power, that is, whether they are set by default to a high signal state on them;
- **clk_speed** – this field assigns the clock frequency when the device is Master. A value of 100000 is normal for standard I2C mode at 100 kHz bus frequency; however, ESP32 also supports 400 kHz I2C bus operation.

Configuration Example:

```
i2c_config_t conf;
conf.mode = I2C_MODE_MASTER;
conf.sda_io_num = 25;
conf.scl_io_num = 26;
conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
conf.master.clk_speed = 100000;
i2c_param_config(I2C_NUM_0, &conf).
```

After configuring, the **i2c_driver_install ()** function is called. While calling, indicate the port, regardless of whether the device is Master or Slave, and if it is Slave, then use the buffer size:

```
i2c_driver_install (I2C_NUM_0, I2C_MODE_MASTER, 0, 0, 0).
```

Once we initialize the I2C port, you can perform the necessary actions. If the device is Master, then you can perform a write/read

operation to the identified Slaves. The read operation is the value "1" of the eighth bit after the seven-bit address, and the write operation is "0". To set this bit, there are constants **I2C_MASTER_READ** and **I2C_MASTER_WRITE**. If you need to read data from the device at the address 0x12, then the transmitted I2C address will be **(0x12 << 1) | I2C_MASTER_READ**, and for writing to the device at 0x12 – the I2C address **(0x12 << 1) | I2C_MASTER_READ**.

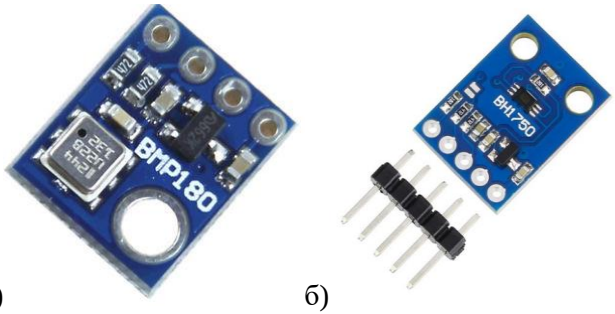
To send a command, we create the structure of this command and then transmit it. The command building begins with the **i2c_cmd_link_create()** function. Then we call **i2c_master_start(cmd)**, which means permission to transfer. Now you can associate the data that you want to send using the **i2c_master_write_byte()** and/or **i2c_master_write()** functions. Next, we indicate that the command transfer is completed – **i2c_master_stop()**. Now you can really ask ESP32 to execute the command by calling **i2c_master_cmd_begin()**. This function sends all buffered commands. After calling **i2c_master_cmd_begin()** you need to release the command register and create a new one for subsequent transfer.

This example shows the transfer of 0x34 byte to the Slave-devices at 0x12 address:

i2c_cmd_handle_t cmd = i2c_cmd_link_create();
i2c_master_start(cmd);
i2c_master_write_byte(cmd, (0x12 << 1) | I2C_MASTER_WRITE, 1); i2c_master_write_byte(cmd, 0x34, 1);
i2c_master_stop(cmd);
i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000/portTICK_PERIOD_MS); i2c_cmd_link_delete(cmd).

Working with sensors. When implementing CPS/IoT projects, the main task is to implement the instrumental Interface between the sensors and the computer component. Now the sensor elements are manufactured with an integrated digital signal processing module, containing ADC and registers, making them easier to handle with.

The barometric pressure and temperature sensor BMP180 (Fig.2.14, a) [2] works with a voltage of 3.3 V. Its address when working on the I2C interface is 0x77. The I2C interface commands for reading information from the device are described in detail in the BMP180 specification.



а) б)

Fig.2.14 – Sensors of: a) pressure and temperature BMP180; b) the level of lighting BH1750

According to the specification, BMP180 registers contain certain parameters; each of them is responsible for certain data for calibration and further reading of information that will need to be converted into a familiar representation of pressure and temperature. The address correspondence of these registers and parameters according to the specification is given in table 2.5, and a set of ready functions for working with the BMP180 sensor are shown in Appendix B for Laboratory Work No. 4.

Table 2.5 – Addressing sensors BMP180 and BH1750

BMP180			**BH1750**	
Parameter	**Type**	**Register**	**Command**	**Meaning**
AC1	short	0xAA, 0xAB	0x00	Turn-off the power
AC2	short	0xAC, 0xAD	0x01	Turn-on the power
AC3	short	0xAE, 0xAF	0x07	Reset
AC4	unsigned short	0xB0, 0xB1	0x10	Mode H
AC5	unsigned short	0xB2, 0xB3	0x11	Mode 2H
AC6	unsigned short	0xB4, 0xB5	0x13	Mode L
B1	short	0xB6, 0xB7	0x20	One time Mode H
B2	short	0xB8, 0xB9	0x21	One time Mode 2H
MB	short	0xBA, 0xBB	0x23	One time Mode L
MC	short	0xBC, 0xBD		
MD	short	0xBE, 0xBF		

The environmental light sensor BH1750 (Fig.2.14, b) [3] operates with a voltage of 3.3 V and is based on the dependence of the semiconductor resistance on the magnitude of the incident light flux. It

has a built-in sixteen-bit ADC, I2C address – 0x23. The list of commands is shown in Table 2.5, and for programming example, see Listing 4.1 for Laboratory work No. 4.

Input and output port expander module PCF8574 [4]. In CPS/IoT projects, it is often necessary to obtain information from dozens and hundreds of information channels. The ESP32 module has 38 contacts in total and only contacts number 34-39 are inputs. The PCF8574 chip operates on the I2C interface, which requires the microprocessor to use only two contacts, and provides eight additional general-purpose input/output ports(GPIO) with the corresponding address field extension (Table 2.6, 2.7). When addressing via the I2C ESP32 interface, the A0-A2 contacts aren't used and are connected to the ground.

Table 2.6 – Assigning PCF8574 Contacts

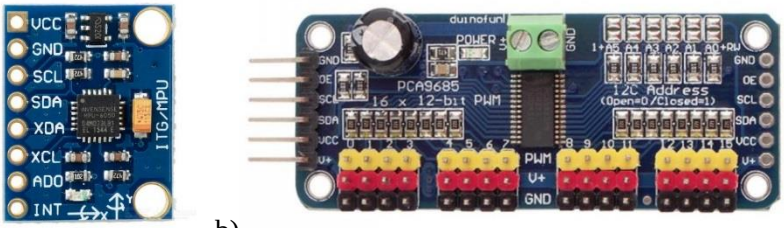
Symbol	Contact number	Appointment
A0-A2	1, 2, 3	Addressing
P0-P7	4, 5, 6, 7, 9, 10, 11, 12	I/O ports
INT	13	Interruption output
SCL	14	Timing line I2C
SDA	15	Data line I2C
VDD	16	Power (2.5 – 6 volt)
Vss	8	Ground

Table 2.7 – Standard PCF8574 pin addressing

A2	A1	A0	Адреса
0	0	0	0x38
0	0	1	0x39
0	1	0	0x3a
0	1	1	0x3b
1	0	0	0x3c
1	0	1	0x3d
1	1	0	0x3e
1	1	1	0x3f

The accelerometer and gyroscope MPU-6050 [5] (Fig.2.15, a) combines the possibilities of measuring linear acceleration and angular velocity in three-dimensional space, giving 6-step freedom of measurement. Measurement resolution is 16 bits, power is 3.3 V. The

MPU-6050 connects to the main ESP32 module using the I2C interface at 0x68. The device contains a set of registers, which is responsible for maintaining a certain measured value. The values of these registers and their descriptions are given in Table 2.8.



a) b)

Fig. 2.15 – Modules: a) MPU-6050 accelerometer and gyroscope; b) PWM controller

Table 2.8 – Description of MPU-6050 registers

Register	Offset	Name	Description
0x3B	0	ACCEL_XOUT_H	AccelX High
0x3C	1	ACCEL_XOUT_L	AccelX Low
0x3D	2	ACCEL_YOUT_H	AccelY High
0x3E	3	ACCEL_YOUT_L	AccelY Low
0x3F	4	ACCEL_ZOUT_H	AccelZ High
0x40	5	ACCEL_ZOUT_L	AccelZ Low
0x41	6	TEMP_OUT_H	Temp High
0x42	7	TEMP_OUT_L	Temp Low
0x43	8	GYRO_XOUT_H	GyroX High
0x44	9	GYRO_XOUT_L	GyroX Low
0x45	10	GYRO_YOUT_H	GyroY High
0x46	11	GYRO_YOUT_L	GyroY Low
0x47	12	GYRO_ZOUT_H	GyroZ High
0x48	13	GYRO_ZOUT_L	GyroZ Low

The use of the MPU-6050 accelerometer and gyroscope module in CPS/IoT projects is especially effective for CPS dynamic physical space. Such sensors allow predicting the behavior of a physical system and correcting its trajectory. For this purpose dynamic models of adaptive systems and a technique for parallelizing computations both on built-in resources and for Cloud technologies are used.

A PWM controller for servo drive controlling (Fig.2.15, b) is implemented according to the timer scheme [6]. The essence of such an approach is that the duration/frequency of the controlled process, as for example, and the angle of the motor rotor rotation, is determined by the control pulse duration τ. In turn, τi is set by a digital timer, that is, the duration of recalculating the number ki of standardized pulses supplied from a calibrated high-precision generator to the counter's input. The board of the 16-channel PWM driver PCA9685 (Fig. 2.15, b) is controlled via the I2C interface at the base address 0x40. ESP32 for working with the PCA9685 driver in CPS/IoT projects can be programmed both in the Eclipse environment and in the Arduino IDE. To work with the Arduino IDE based on the created LRM, we modify the existing library, which holds two files *Adafruit_PWMServoDriver.h* and *Adafruit_PWMServoDriver.cpp.*

The file with the extension *.h* to contains macros defining the names of commands and their digital representation (**#define PCA9685_SUBADR3 0x4**, etc.). These commands are responsible for setting the PWM frequency, selecting a channel, turning PWM on and off on a particular channel. This information will be useful in the implementation of PWM-controller control when programming using the ESP-IDF framework.

In addition to these macros, the header file contains the class **Adafruit_PWMServoDriver**, which creates an abstraction between the I2C interface and the PWM controller. In this class, there is the **setPWMFreq()** method, which sets the frequency for the PWM, and the **setPWM()** method, which sets the specific PWM channel to a specific PWM signal. Also in this class there is a single private field of the class that stores the device address. Let’s add two more private fields to this class, which will determine the pin numbers for the SDA and SCL lines:

```
private:
  uint8_t _i2caddr;
  uint8_t _sda_pin;
  uint8_t _scl_pin.
```

Next, we add two parameters to the prototype of the class constructor, which will set the values of the class fields:

```
Adafruit_PWMServoDriver(uint8_t addr = 0x40,
uint8_t sda = -1, uint8_t scl = -1).
```

In the file with the extension *.cpp* we will change the body of the constructor:

```
Adafruit_PWMServoDriver::Adafruit_PWMServoDriver(uin
t8_t addr,
    uint8_t sda = -1, uint8_t scl = -1) {
      _i2caddr = addr;
      _sda_pin = sda;
      _scl_pin = scl;
    }
```

Finally, let's change the function that triggers the I2C interface enable:

```
    void Adafruit_PWMServoDriver::begin(void) {
      if (_sda_pin == -1 || _scl_pin == -1)
     WIRE.begin();
      else
     WIRE.begin(_sda_pin, _scl_pin);
      reset();
    }
```

Such a modification of the library is sufficient for direct control of the servomotor. However, since each servomotor model has different characteristics, the PWM-controller settings are different for them. For example, when setting up the SG90 servomotor, the frequency of the PWM signal which it works with, and the minimum and maximum pulse width of the signal should be specified: frequency 60 Hz, and the minimum and maximum pulse duration, respectively, 150 and 600 units for working with this library. That is, the operation algorithm is as follows:

- declare a class object and specify its address and pin numbers for the I2C interface:

```
Adafruit_PWMServoDriver pwm =
Adafruit_PWMServoDriver(0x40, 16, 17);
```

- enable this object in the **setup()** function and set the engine speed:

```
pwm.begin();
pwm.setPWMFreq(60);
```

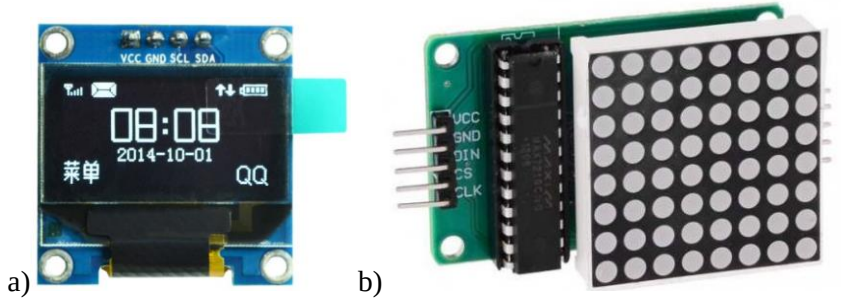
- enable in **loop()** function the appropriate port to which the servomotor is connected and set the required pulse duration, which in turn will set the servomotor axle to the appropriate angle:

```
uint8_t servoNum = 0;
// sets the axle to an angle of 0 degrees
pwm.setPWM(servoNum, 0, 150);
// sets the axle to an angle of 90 degrees
```

pwm.setPWM(servoNum, 0, 375);
// sets the axle to an angle of 180 degrees
pwm.setPWM(servoNum, 0, 600).

Of course, for other types of engines the settings will be different too.

The monochrome OLED display is controlled by the SSD1306 driver and has a resolution of 128x64 pixels (Fig. 2.16, a) and is controlled by SPI and I2C interfaces when working with ESP32 [9].



a) b)

Fig. 2.16 – Information displaying means: a) OLED display SSD1306; b) LED matrix MAX7219/MAX7221

To simplify image management on the OLED display, let's use the Arduino graphics core and the source code library for working with this display [10,11], which will be loaded to the directory *C:\Users\<user>\Documents\Arduino\libraries*.

To customize the code for a specific display model and ESP32 module using the I2C interface, we will make certain changes in the *Adafruit_SSD1306.h* file. On line number 73, uncomment the macro that is responsible for supporting the display with a resolution of 128x64 pixels and comment the bottom macro:

#define SSD1306_128_64
//#define SSD1306_128_32

The next step is to change the initialization of the I2C interface in the Adafruit_SSD1306.cpp file. Since in the laboratory bench implementation the display is connected via the I2C interface, where the SDA line corresponds to 25 pin number and the SCL line - to 26, we will add information about specific connectors to this interface call. To do this, open the file and go to line number 206 and change the interface call to the following: **Wire.begin (25, 26)**.

The modified library is ready for use. You can check its

functionality using the project example in the *...\examples\ssd1306_128x64_i2c* folder for the Arduino IDE. The description of functions for working with this graphic core and the creation of an arbitrary image on this OLED display can be found in the file *C:\Users\User\Documents\Arduino\libraries\Adafruit-GFX-Library-master\Adafruit_GFX.h.*

The LED matrix MAX7219/MAX7221 (Fig. 2.16, b) [12] is controlled by the SPI drivers of the same name with ESP32 and has a limited voltage rise rate for segment drivers. Drivers allow cascading and allow the user to determine the decoding mode of each bit. Each of the indicator bits has independent addressing and its contents can be updated without having to overwrite the entire indicator. In addition, these drivers have a sleep mode with information storage, analog and digital brightness control of connected indicators and a test mode switching on all LED segments. The driver can control eight seven-segment indicators with a dot, or separately 64 LEDs in LED panels with a common cathode. Thus, these microcircuits are suitable not only for seven-segment but also for matrix indicators.

Work with this LED matrix comes down to working with its specific driver. Formatted data (Table 2.9) is sent over the SPI bus from ESP32 by 16 bits, the most significant bit forward.

Table 2.9 – Data Matrix Format MAX7219/MAX7221

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
X	X	X	X	ADDRESS				DATA							

In bits D15...D12, useful information is not transmitted. The ADDRESS field specifies the instruction for action: 1...8 (0001...1000) – select a character-place. In this case, in the DATA field, information about the state of the character-place segments is transmitted: 0 (00000000) – segments do not light up; 255 (11111111) – all segments of the selected character-place, including point, light up. The ADDRESS value of 9...15 (1001...1111) indicates the execution of the service instruction (table 2.10).

Table 2.10 – List of instructions MAX7219/MAX7221 and their meanings

ADDRESS					Command Description
D11	D10	D9	D8	HEX CODE	
1	0	0	1	0x09	Decoding mode. The DATA bits select which character-places to decode and which not. Dx=1 – decode sign x, Dx=0 – do not decode sign x.
1	0	1	0	0x0A	Glow intensity (brightness). Bits D0...D3 select the brightness of the glow. At D0=D1=D2=D3=0, the brightness is minimal. When D0=D1=D2=D3=1, the brightness is maximum.
1	0	1	1	0x0B	Selecting the number of the displayed character-places. Bits D0...D2 select the displayed locations. At D0=D1=D2=1 all eight character-places are displayed
1	1	0	0	0x0C	Sleep mode. DATA=0, the microcircuit goes into sleep mode. DATA=1 – normal mode.
1	1	0	1	0x0D	Not used
1	1	1	0	0x0E	Not used
1	1	1	1	0x0F	Test. DATA=1 – test activated, DATA=0 – turned off.

For normal operation of the board, after power is supplied, it must be initialized by running the following commands:

- ADDRESS = 0x0F, DATA = 0x00 – the indicator test is off;
- ADDRESS = 0x0C, DATA = 0x01 – get out of sleep mode;
- ADDRESS = 0x0B, DATA = 0x07 – the number of used characters– 8;
- ADDRESS = 0x09, DATA = 0x00 – the decoder is off;
- ADDRESS = 0x0A, DATA = 0x0F – the maximum brightness.

After these steps, different LEDs can be randomly highlighted on the matrix. This must be taken into account and immediately cleaned by setting all LEDs to zero using ADRES=0x01...0x08, DATA=0x00.

Practical part

1. Create and test a program that works with the BMP180 pressure and temperature sensor. The program should update the sensor information every second and display it to the console.
2. Complete the program with a light sensor. Modify it so that the information from two sensors is displayed to the console at the interval of 5 seconds. Save it.
3. Create a program for networking in the combined mode – "workstation\access point". The essence of this mode is that, at the same time, the device acts as an access point for certain outside connected devices, and for others – as a station. That is, the device is Master on one's own local subnet, and Slave on another
4. Extend this program with a socket that works in server mode, that is, it listens to the port for connection. Set up the program for each device (stand) so that all devices can exchange information with each other.
5. Add functionality for working with sensors to this program. Organize the work of programs on devices so that the information about pressure, temperature and lighting from one stand, with a certain interval, would be transmitted to another stand. On the other stand at the time of receiving information, the information from its own sensors is being read. Check the information from the sensors for errors (for example, in the indication difference of neighboring sensors), calculate the arithmetic average of each value and display it to the console of one (terminal) device.
6. Create a program for the device that establishes communication with the MPU6050 module, and reads data from the accelerometer and gyroscope. Arrange the information output on all three axes of the accelerometer and gyroscope to the console with a frequency of 1 second (note that the first few seconds are necessary for calibrating the module). Create a program that changes the color of the RGB LED glow, depending on the inclination angle of the device in space.
7. Install Arduino IDE and configure the environment for the work with ESP32.
8. Using the **Wire** class, establish communication with the MPU6050 module and read the data from the accelerometer and display them to the console.
9. In the Eclipse environment, using the LEDC/PWM driver, organize the RGB LED glow of different brightness. Each color should

alternately glow from minimum to maximum brightness. The parameters for the timer are arbitrary.

10. Open “File” -> “Examples” in the Arduino IDE; find the tab “Examples for ESP32 Dev Module” in the list; open the program code with the tab “WiFi” -> “SimpleWiFiServer”. Explore this program working. Modify it so that it exposes the servomotor to the angle, information about which comes over the network from an external device. An Android device, or any other device with the ability to work in a Wi-Fi network, can act as an outside device.
11. In the Arduino IDE, select “File” -> “Examples” -> “Examples from user libraries” -> “Adafruit SSD1306” -> “ssd1306_128x64_i2c”. Download this program to the device and check its functionality. Display the specified animation using the graphic library "Adafruit_GFX.h".
12. In the Eclipse environment, create a program for static and dynamic data output from the BMP180 and BH1750 sensors to the LED matrix. Select image according to variant.

Report

The report and protection of laboratory work is carried out according to the requirements described in laboratory work 2. Tasks №№ 1, 3, 5, 8 are obligatory for execution. Other tasks are performed individually as additional ones.

Test questions

1. What functions can BMP180 and BH1750 modules perform in CPS/IoT projects? Describe the features of their programming with ESP32.
2. How to implement a combined "workstation/access point" mode on ESP32?
3. Make a comparative analysis of Eclipse and Arduino IDE libraries for ESP32.
4. How to use file sockets to implement file exchange between ESP32 devices?
5. How is the reading of the accelerometer and gyroscope MPU6050 synchronized with LED smart-control?
6. Describe the **Wire** class and procedure for establishing communications and exchanging data with the MPU6050 module.

7. Justify with what accuracy and resolution information can be displayed on the OLED?
8. Describe the functionality of “SimpleWiFiServer”.
9. How is image output implemented in "ssd1306_128x64_i2c"?
10. What is the difference between static and dynamic data output?

Recommended literature

1. "I2C bus description", *Itt-ltd.com*, 2019. [Online]. Available: http://itt-ltd.com/reference/ref_i2c.html.
2. "BMP180 Digital pressure sensor. Data sheet", *Cdn-shop.adafruit.com*, 2019. [Online]. Available: https://cdn-shop.adafruit.com/datasheets/BST-BMP180-DS000-09.pdf.
3. "Digital 16bit Serial Output Type Ambient Light Sensor IC. BH1750FVI", *Mouser.com*, 2019. [Online]. Available: https://www.mouser.com/ds/2/348/bh1750fvi-e-186247.pdf.
4. "PCF8574; PCF8574A Remote 8-bit I/O expander for I2C-bus with interrupt", *Nxp.com*, 2019. [Online]. Available: https://www.nxp.com/docs/en/data-sheet/PCF8574_PCF8574A.pdf.
5. "MPU-6000 and MPU-6050 Register Map and Descriptions Revision 4.2", *Invensense.com*, 2019. [Online]. Available: https://www.invensense.com/wp-content/uploads/2015/02/MPU-6000-Register-Map1.pdf.
6. "Adafruit PWM Servo Driver Library", *Arduinolibraries.info*, 2019. [Online]. Available: https://www.arduinolibraries.info/libraries/adafruit-pwm-servo-driver-library.
7. "Arduino – Software. Arduino WEB editor ", Arduino.cc, 2019. [Online]. Available: https://www.arduino.cc/en/Main/Software.
8. "Arduino core for ESP32 WiFi chip", *GitHub*, 2019. [Online]. Available: https://github.com/espressif/arduino-esp32.
9. "ESP32 OLED Display with Arduino IDE | Random Nerd Tutorials", *Random Nerd Tutorials*, 2019. [Online]. Available: https://randomnerdtutorials.com/esp32-ssd1306-oled-display-arduino-ide/.
10. "adafruit/Adafruit-GFX-Library", *GitHub*, 2019. [Online]. Available: https://github.com/adafruit/Adafruit-GFX-Library.
11. "adafruit/Adafruit_SSD1306", *GitHub*, 2019. [Online]. Available: https://github.com/adafruit/Adafruit_SSD1306.
12. "MAX7219/MAX7221. Serially Interfaced, 8-Digit LED Display Drivers", *Datasheets.maximintegrated.com*, 2019. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX7219-MAX7221.pdf.

2.2 Applying reconfigurable environments in CPS/IoT project synthesis tasks (Laboratory Work №5)

Assoc. Prof., PhD Ya. Klyatchenko (NTUU "KPI"),
Assoc. Prof., PhD H. Vorobets, Assist. of Lect. V. Horditsa (ChNU)

The aim of the laboratory work: mastering the methodology for implementing modules with integrated information processing in CPS/IoT projects, improving the practical development skills for solutions based on FPGA programmable environments.

Recommended hardware and software

Debugging stand based on the programmable logical environment XC3S700AN; WebPACK ISE CAD software for developing FPGA projects from Xilinx.

Theoretical Information

The implementation of a comprehensive information processing project in a configurable FPGA environment significantly expands the possibilities of the practical CPS application [1]. This approach allows expanding the application field of CPS/IoT technologies and diversifying the algorithm sets and types of tasks to be solved. The FPGA projects' application is particularly effective for high-performance computing, video processing, and tasks requiring algorithm parallelization. In particular, for example, such solutions are implemented in modern ultrasound diagnostics, space technology, and systems of critical application to increase the reliability and resilience of control systems [2].

In CPS of ecological monitoring, observation and control of technological processes, real-time intelligent information analysis, it often needs to visualize the results, generate additional signals, and make decisions. Consider an example of implementing a comprehensive solution using XC3S700AN environments on the Xilinx Spartan 3AN platform [3]. Of course, there are more powerful tools available now – Virtex 4/5/6, Spartan-6 [4]. But when choosing a platform for real CPS/IoT projects, you need to consider cost-effective system performance as well.

Let's suppose that for a monitoring system it is necessary to monitor objects distributed in space, to record their proximity to a critical state and to signal a critical situation. Such a system can be relatively easy simulated using the signal processing of the VGA video terminal, PS/2 keyboard, and generating modulated acoustic oscillations on the XC3S700AN chip of the Xilinx platform [3].

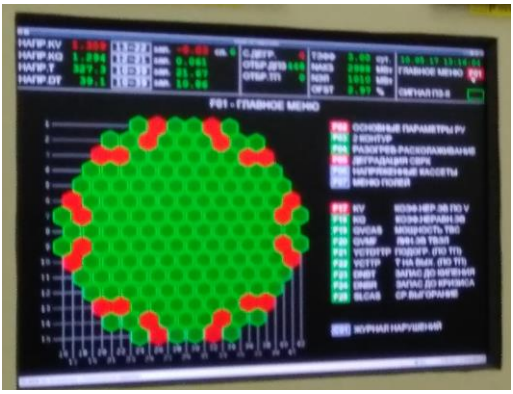


Fig. 2.17 – Visualization example of CPS status monitoring results

Fig. 2.18 – PS/2 layout on the Spartan-3AN Starter Kit board [3]

The PS/2 port, located on the Spartan-3AN Starter Kit (Figure 2.18), allows two devices, such as a keyboard and a mouse, to be connected simultaneously with a splitter (Figure 2.19). The UCF-file for initialization is as follows:

```
# Main connection
NET "PS2_CLK1" LOC = "W12" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
NET "PS2_DATA1" LOC = "V11" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW;
```

```
# Additional connection
NET "PS2_CLK2" LOC = "U11" | IOSTANDARD = LVCMOS33 | DRIVE = 8
| SLEW = SLOW;
NET "PS2_DATA2" LOC = "Y12" | IOSTANDARD = LVCMOS33 | DRIVE =
8 | SLEW = SLOW;
```

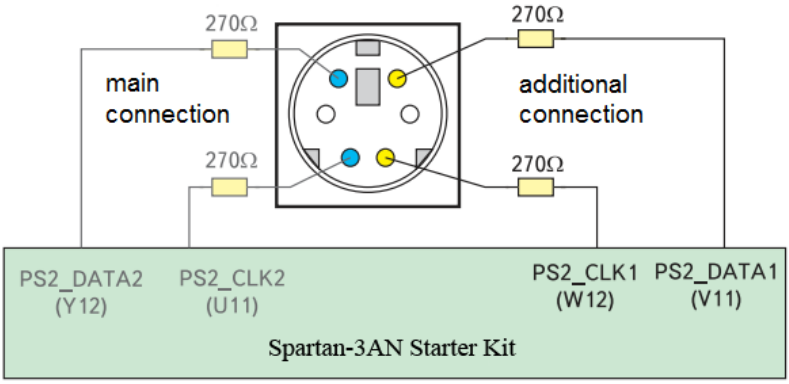


Fig. 2.19 – Wiring diagram for connecting PS/2 contacts to the Spartan-3AN Starter Kit

PS/2 protocol operation. Data exchange between the keyboard and the controller is carried out asynchronously using a serial protocol when a certain key is activated. Two lines are used for data exchange - KBData and KBSync. When transmitting scan codes, the keyboard sets the next bit of data on the KBData line and confirms the transfer by switching the signal from the “1” to “0” on the KBSync line. When receiving data from the controller, the keyboard reads bit of data from the KBData line and issues a confirmation of receipt by transferring the signal on the KBSync line from "1" to "0". The controller may signal that it is not ready to transmit/receive by low-level data on the KBSync line. The rest of the time, when there is no data to transmit, both lines have a high signal level. The pulse repetition rate of the KBSync line is about 10-25 KHz.

Data transmission order: one start-bit – "0", eight data bits, parity bit (the sum of all bits +1), one stop-bit – "1". After receiving each byte of data, the controller sets a low level on the KBSync line signaling that it is busy processing the received data and is not ready to accept the next. This can be considered a confirmation of acceptance. The keyboard confirms every byte of the received command with the 0FAh code. If an

error occurs during transmission, the controller may require the last byte to be transmitted again by issuing the 0FEh command. The keyboard behaves differently – it simply ignores errors. Each key of the PS/2 keypad has a unique scanning code (Fig. 2.20), which is sent each time the corresponding key is pressed [3].

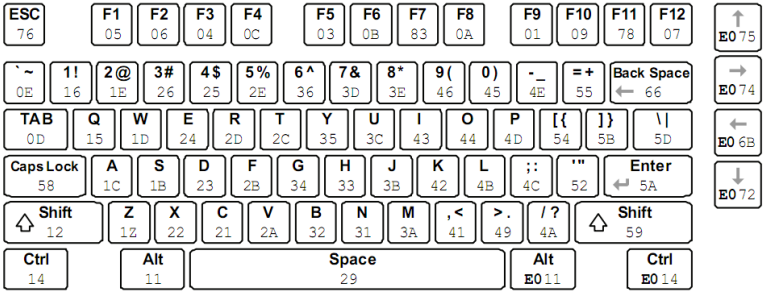


Fig. 2.20 – Keypad scan codes [3]

VGA-port on Spartan-3AN FPGA Starter Kit (Fig.2.21) is made as standard HD-DB15 connector. Using this port allows connecting most CRT and LCD monitors that come with a standard VGA cable.

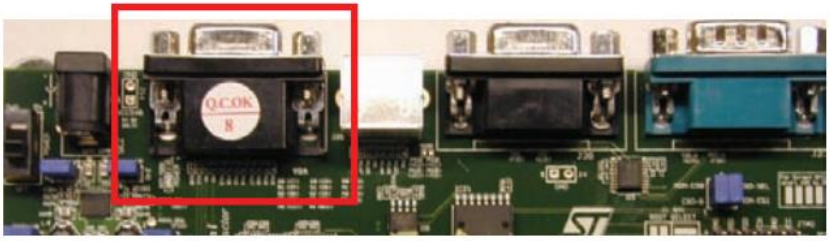


Fig. 2.21 – VGA-port layout on the Spartan-3AN Starter Kit board [3]

The principle of controlling a VGA port is to generate a tape and frame scan using two counters, which provide pixel-by-pixel scanning of the image, respectively, by rows and columns. When addressing each pixel, the RGB signal is activated in a certain proportion, which ensures the reproduction of the corresponding color cast. FPGA directly controls the five VGA signals through resistors (Fig. 2.22). Each red, green and blue signal has four outputs from the FPGA. The resistor values provide a binary coded output level. Thus, at 4-bit resolution, each base color

generates a 12-bit code or 4096 possible color tones. A series resistor, combined with a 75 Ohm resistor built into the VGA cable, ensures that VGA color signals remain in the 0V to 0.7V range. The control of the signal level VGA_R [3:0], VGA_G [3:0], VGA_B [3:0] provides generation of the desired color (Fig. 2.23).

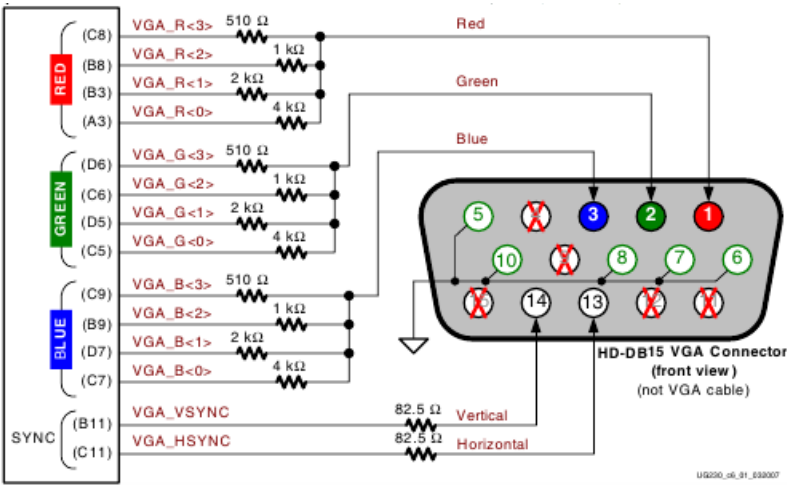


Fig.2.22 – VGA-port to FPGA connection scheme [3]

VGA_R[3:0]	VGA_G[3:0]	VGA_B[3:0]	Color
0000	0000	0000	
0000	0000	1111	
0000	1111	0000	
0000	1111	1111	
1111	0000	0000	
1111	0000	1111	
1111	1111	0000	
1111	1111	1111	

Fig 2.23 – Formation of colors

VGA signal generation is described in detail in Chapter 6 of the technical documentation of the Spartan-3A/3AN FPGA Starter Kit Board User Guide [3].

UCF-file example:

```
NET "VGA_R<3>"   LOC = "C8" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_R<2>"   LOC = "B8" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_R<1>"   LOC = "B3" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_R<0>"   LOC = "A3" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_G<3>"   LOC = "D6" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_G<2>"   LOC = "C6" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_G<1>"   LOC = "D5" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_G<0>"   LOC = "C5" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_B<3>"   LOC = "C9" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_B<2>"   LOC = "B9" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_B<1>"   LOC = "D7" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_B<0>"   LOC = "C7" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = FAST ;
NET "VGA_HSYNC" LOC = "C11" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW= FAST;
NET "VGA_VSYNC" LOC = "B11" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW= FAST.
```

Examples of interfaces with the VGA port for contour formation and polygon fills can be used as blocks for arbitrary image configuration.

Generation of tone signals is realized through the audio output of the Spartan-3AN Starter Kit (Fig. 2.24). UCF-file example:

```
NET "AUD_L" LOC = "Y10" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
NET "AUD_R" LOC = "V10" | IOSTANDARD = LVCMOS33 | DRIVE = 8 | SLEW = SLOW ;
```

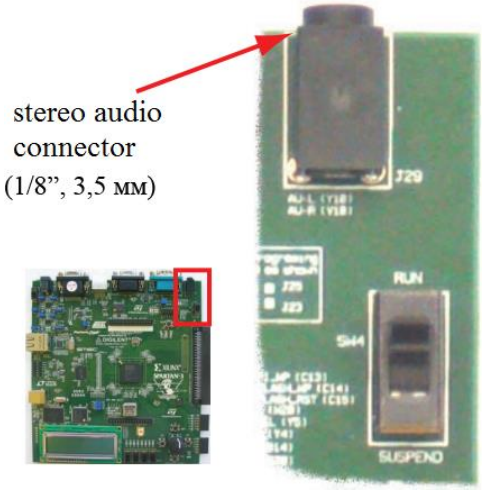


Fig. 2.24 – Audio output of the Spartan-3AN Starter Kit

The principle of sound generation lays in a multiple division of the oscillation frequency of the reference high-frequency clock generator to the sound range. So a single-tone signal can be easily implemented on the basis of binary counters in FPGA. With a 25 MHz counter, the

frequency can be divided in half, using a 16-bit counter, which counts from 0 to 65536 different values. Counter switching will occur at a frequency of 25000000/65536=381 Hz.

Listing of the program of single-tone signal:

```
module music(clk, speaker);
input clk;
output speaker;
// 16 – bit counter
reg [15:0] counter;
always @(posedge clk) counter <= counter+1;
// at a high level, monophonic sound output
assign speaker = counter[15];
endmodule
```

Listing of the program of two-tone signal:

```
module music(clk, speaker);
input clk;
output speaker;
parameter clkdivider = 25000000/440/2;
reg [23:0] tone;
always @(posedge clk) tone <= tone+1;
reg [14:0] counter;
always @(posedge clk) if(counter==0) counter <= (tone[23] ? clkdivider-1 : clkdivider/2-1); else counter <= counter-1;
reg speaker;
always @(posedge clk) if(counter==0) speaker <= ~speaker;
endmodule.
```

To generate oscillations of variable tonality let's use the first example of a simple single-tone sound and 7-bit tonal counters in tone [21:15] with a shift in the reference point. To switch between the two signals, we use tone [22]. As soon as the first signal counter reaches 127, we move to the second signal until the counter goes to 0, and then return to the first signal.

Listing of the program of multitone signal:

```
module music(clk, speaker);
input clk;
output speaker;
reg [22:0] tone;
always @(posedge clk) tone <= tone+1;
wire [6:0] ramp = (tone[22] ? tone[21:15] : ~tone[21:15]);
```

wire [14:0] clkdivider = {2'b01, ramp, 6'b000000};
reg [14:0] counter;
always @(posedge clk) if(counter==0) counter <= clkdivider; else counter <= counter-1;
reg speaker;
always @(posedge clk) if(counter==0) speaker <= ~speaker;
endmodule.

The Spartan-3AN Starter Kit basic module also has built-in potentiometers, analog-to-digital converters, and other units that are programmed in a similar way. projects of varying complexity can be simulated, combining their functionality. You can check the interaction correctness of VGA port, mouse, keyboard, and oscillation generation by downloading the appropriate modules, or example of the ping-pong program listing from Appendix B.

Report

The report and protection of laboratory work is carried out according to the requirements described in laboratory work 2.

Practical part

1. Read the technical documentation about working with the VGA port in the technical description of the Spartan-3A/3AN FPGA Starter Kit Board User Guide, section 6. Using samples, write a software module that selects on the screen an area similar to the real one being studied, and statically clusters it to the observation zones, where the sensors are located, and paints them.
2. Modify the created module to simulate the dynamic display of sensor readings in clusters on a color scale.
3. Add a keyboard and mouse control module. Compile and download presented examples on FPGA. Check its functionality. Tasks by variants
4. Synthesize the module for generating sound vibrations. Provide for the possibility of its activation on given conditionally critical states of clusters.
5. Compile a complex project with all three synthesized software modules involved. Test it.

Test questions

1. What are the principles of display control in FPGA driver synthesis?
2. What principles are implemented in FPGA for keyboard and mouse operation?
3. What principles are implemented in the FPGA to generate sound vibrations?
4. How can the keyboard and VGA monitor function be synchronized in the FPGA environment when implementing a CPS/IoT project?
5. What are the functionality features of implementing dynamic visualization management in CPS/IoT projects?
6. Justify with what accuracy and resolution the information can be outputted using the built-in tools in the Spartan-3A/3AN FPGA Starter Kit Board User Guide?
7. Give and describe application examples of known to you FPGAs of Intel/Altera and Xilinx series in CPS/IoT projects?
8. What are the advantages of modern dynamically reconfiguring FPGAs of the Intel/Altera and Xilinx series?

Recommended literature

1. *Xilinx.com*, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds557.pdf.
2. F. Vanderhaegen, "Towards increased systems resilience: New challenges based on dissonance control for human reliability in Cyber-Physical&Human Systems", 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578817300275
3. *Xilinx.com*, 2019. [Online]. Available: https://www.xilinx.com/support/documentation/boards_and_kits/s3astarter_schematic.pdf.
4. *Rssp.ru*, 2019. [Online]. Available: http://www.rssp.ru/upload/iblock/916/13_Otladochnie_plati%20f%20nabori.pdf.

2.3 Methodology for implementation of the CPS/IoT complex project (Laboratory work № 6)

Assoc. Prof., PhD H. I. Vorobets, Assist. of Lect. O. O. Pshenychnyi, Bachelor student V. V. Buchakchiyskyi (ChNU)

The aim of the laboratory work: generalization of knowledge about functional-algorithmic CPS models and application of IoT in problems of their analysis, synthesis and functionality expansion; gaining practical skills in applying a complex approach to CPS/IoT projects.

Recommended hardware and software

Name	Link
Arduino Nano	https://store.arduino.cc/arduino-nano
Arduino IDE	https://www.arduino.cc/en/main/software
USB 2.0 cable type A/B	https://store.arduino.cc/usb-2-0-cable-type-a-b
ESP32(DevKit)	https://github.com/playelek/pinout-doit-32devkitv1
ESPAsyncWebServer.h	https://github.com/me-no-dev/ESPAsyncWebServer
Laboratory stand	

Theoretical information and example of work execution

This laboratory work is proposed as a final lesson in the module "IoT technology in the problems of synthesis and analysis of CPS". Based on the results of its implementation, a holistic view should be formed on the features of the structure and functional-algorithmic organization of CPS, their classification and fundamental differences from automation and mechatronics systems, the role of IoT technologies both in expanding the functionality of CPS and in solving problems of their analysis and synthesis.

As a final example, it is proposed to consider the synthesis of information-analytical CPS spectroscopy of physical objects and bioactive environments for scientific research, the structural and functional-algorithmic analysis of which is given in paragraph 1.1.1. The synthesis of such a system involves the implementation of three stages: 1) synthesis of a minimally- or fully-functional automaton for controlling physical processes in the optoelectronic channel of a standard

spectrophotometer of the SF series; 2) the justification and implementation of cyber components for the intellectualization of the functionality of the control automaton and/or the creation of an autonomous version of CPS; 3) the use of IoT technology to expand the functionality and implementation of open-type CPS.

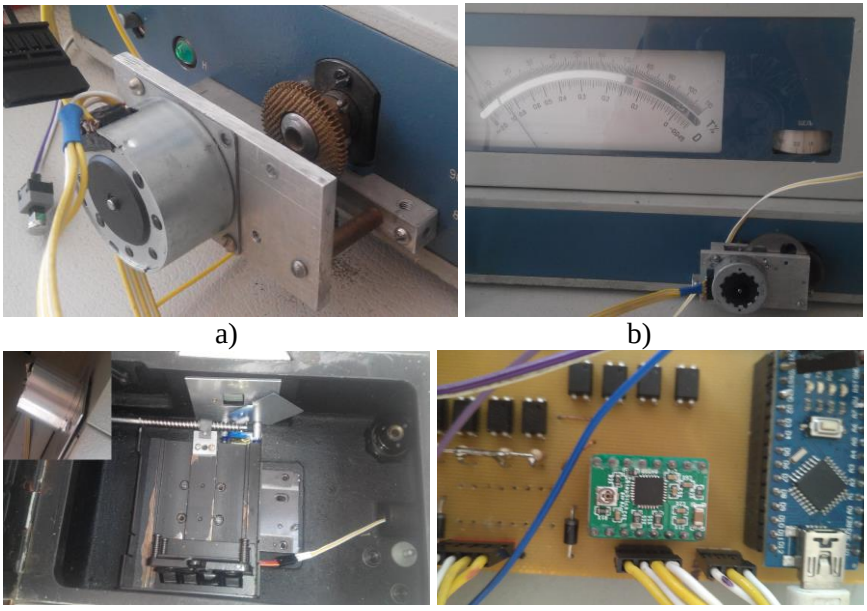
To implement the first stage in the form of an automaton with the minimum required set of functionality:

$$M_{min}=\{\lambda_0, \lambda_i, \lambda_f, n_i(\Delta\lambda_i), U_{is0}, U_{is}, U_{isf}, m_j(\Delta U_j), k_m\},$$

it is necessary to provide controlling over the sweep of the wavelength spectrum in the range from λ_{min}=180 nm to λ_{max}=1100 nm with a given step n_{min}, controlling the geometric dimensions of the optical window from minimum to maximum value with a step m_{min}, commuting k_m calibration/measurement mode, and ensuring potential measurements U_{is} on the photosensor minimum U_{is0} to maximum U_{isf} value with the analog-to-digital converter (ADC).

To sweep the spectrum and adjust the size of the optical window, one can use unipolar stepper motors of various types, for example EM-211, 42SIM-24Dxxx, etc. (Fig. 2.25, a, b), with additional mechanical gearboxes [1, 2]. The formation of control pulses with the necessary parameters for the engines is provided by Darlington assemblies ULN203xx, ULN2803A or the like. The calibration mode of the optical channel is carried out when there is no sample in it and in two positions – closed and open channel, that is provided by a curtain with an electromagnetic drive (Fig. 2.25, c). The feed into the optical channel of the carriage with the samples under study is automated by means of a worm gear driven by an EM142 engine, where the control signals are formed by the A4988 driver [3].

Thus, for a minimum set of physical object automation, 12 digital lines are required to control the engines, one – for the curtain, one analog input for measuring the photopotential on the sensor, and 2 single-bit inputs for fixing zero states n0 and m0, A minimal set of the Arduino Nano module is enough for this. The connection of the digital lines of the Arduino board with the power drives of the stepper motors is implemented using optocouplers (Fig. 2.25, d). The software is formed as a set of management functions that can be activated according to the algorithm of the main control program. Arduino Nano built-in memory is enough for it and no additional resources are required.



a) b)

c) d)

Fig. 2.25 – An example of providing the minimum required level of PO automation for its integration into a CPS/IoT project; the photo shows the control modules: a) spectrum sweep along λ; b) the size of the optical window; c) a carriage with a test sample and an optical window curtain; d) Arduino Nano with A4988 driver and optical connection of power drives.

The cyber component for the implementation of the CPS/IoT project can be built in several versions. The simplest approach is to use a universal PC switched with Arduino Nano via a USB port and an additional interface board. Installed on a PC, the Arduino IDE allows programming the Arduino Nano for performing basic management functions. PC applications are enough to implement the mechatronic ACS model in the form of standard measurement algorithms. For example, to work with Arduino modules in this system, one needs to install the CH340/CH341SER driver and the termite-3.1/3.3 or any other remote access program. Measurement data is accumulated in the form of text files, which are then processed by other application software.

For the synthesis of stand-alone CPS, which implements “intelligent” branched algorithms for modeling processes and

characteristics of POs, controlling POs and processes, and processing data on PC, object-oriented programming (OOP) tools (C++, Java#, JavaScript) and other universal software packages (MatLab2011, MathCad, etc.) are used.

Using ESP32 in this CPS/IoT project also has certain features. The basic variant involves the implementation of the CS-PS interaction model according to the scheme (Fig. 2.26): *Arduino Nano as controller of mechanical part and ESP32 as a web server and a device for implementing an "intelligent" measurement control algorithm*.

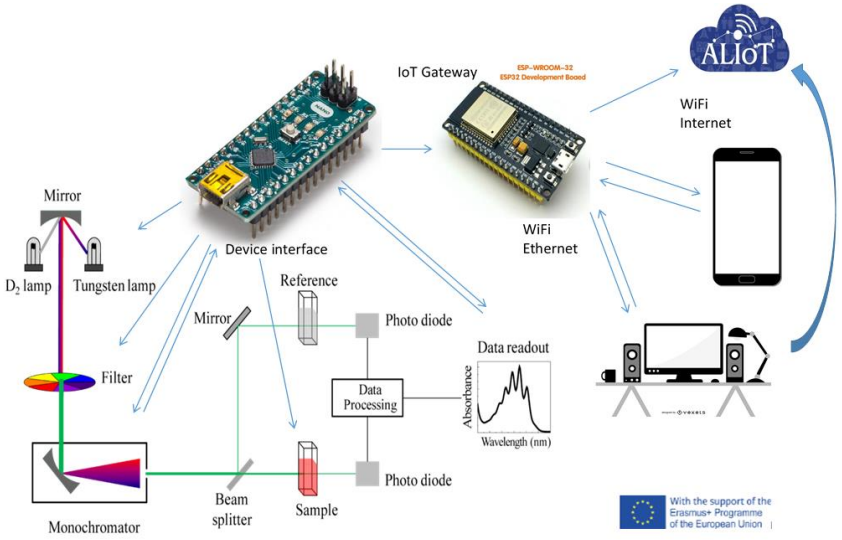


Fig. 2.26 – An example of CPS/IoT project – Arduino Nano as controller of mechanical part and ESP32 as a web server

The Arduino "Source code" software module performs the following functions:

- monitoring and control of the mechanical servos of the spectrophotometer according to the specified CPS functional program;
- measurement of the potential of U_{is} on the photosensor using the built-in ADC;
- Transmit U_{is} and λ data through the serial port (UART) according to the protocol of the basic algorithm.

To create the programs-functions of the Arduino Nano module, the *SerialCommand.h* library, which simplifies the development of programs using the serial port, and *Servo.h* library, which allows easy servos control, were used.

The Arduino controller accepts/identifies the following basic commands on the serial port:

- Init – starts the device initialization algorithm, and puts all mechanisms in the starting position;
- Go xxxx yyyy zz n - launches the standard algorithm for measuring the spectrum in the range $\Delta\lambda$ from xxxx (nm) to yyyy (nm) with a step zz (nm) for the sample in the n-th carriage;
- GoFi xx m – launches a non-standard spectrum measurement algorithm with scanning by λ, but with a step xx (%) by ΔU_{is};
- GoDm xxxx yyyy zz nm – launches a non-standard algorithm for measuring the spectrum with dynamic correction of the scan step by λ using the results of the dynamics evaluating ($S_m=\Delta U_{is}/\Delta\lambda_i$).

The execution of the main algorithm is initialized from ESP32 by the *Start* command. The measurement results are recorded on an SD card or streamed in parallel to a host computer via Wi-Fi network.

The EPS32 controller also plays the role of a web server. The following libraries were used for this purpose: *ESPAsyncWebServer.h* – to run an asynchronous web server based on a microcontroller; *FS.h* – for interacting with the file system; *SPI.h* – for interaction with the SPI interface; *HardwareSerial.h* – for interacting with the serial port. The web server created on EPS32 is identified by the local IP address of the router and allows to have access to measurement data or to start new measurements without having physical access to the CPS PO.

The advantage of this solution is the low cost, and the disadvantage is the relatively high development complexity. It's also required to investigate the reliability of the ESP32 with two or more end-devices due to its limited resources.

The CPS/IoT model has advanced capabilities when it's implemented according to the scheme: *Arduino Nano as a mechanical part controller and Raspberry Pi as a web server and a device for implementing an "intelligent" measurement control algorithm*.

In this case, the role of the functions of the Arduino module remains unchanged, while the functions of measurement management and web services (nginx etc.) are performed by the *Raspberry Pi* module. Requirements for the last one do not have much significance since each

of them has enough memory to start and run UNIX-based RTOS [4]. The advantage of this solution is the high reliability of the CPS/IoT project due to the lack of user web-server libraries, which are necessary in the case of ESP32. In addition, edge computing features can be implemented on the *Raspberry Pi*, reducing Wi-Fi traffic in complex CPS/IoT solutions. For relatively simple tasks, the question of the appropriateness/cost-effectiveness of using excess *Raspberry Pi* resources in CPS/IoT projects arises.

Notes for practical work

When performing a practical task on programming and setting up a laboratory bench of a spectrophotometer with a control system based on Arduino Nano and ESP32 (DevKit), **REMEMBER** that the serial port of Arduino is used both for communication with another controller and for communication with a computer. Therefore, in order to avoid collisions, the RX0 and TX0 lines that are used for ESP32–Arduino communications must be broken when programming the ESP32. That is, the ESP32–Arduino system configuration algorithm can be as follows:

- disconnect the RX0 and TX0 lines which connect the ESP32–Arduino;
- program the Arduino Nano using the Arduino IDE;
- program ESP32 using Arduino IDE;
- Switch the RX0 and TX0 lines that connect the ESP32–Arduino to enable them to exchange information.

Note that in this case, we only download measurement algorithms and spectrophotometer control subprograms to the Arduino Nano, and all the control functions are assumed by ESP32.

Tasks for individual execution

The practical tasks that students have to complete consist of two parts.

Part A - development of intelligent measurement control and data processing algorithm for CPS/IoT based on the laboratory stand of an automated spectrophotometer, or its simulation model. The following algorithms are offered:

1. Quick search for the extrema of the function $U_{is}=f(\lambda)$, the selection of three points λ_i with the maximum value of U_{is} among them and additional scanning of the spectrum in the vicinity of $\Delta\lambda_i \pm 10$ nm for the selected points.

2. Scanning the spectrum $U_{is}=f(\lambda)$ with a given step $\Delta\lambda_i$, searching for all maxima and minima of the function $U_{is}=f(\lambda)$ and establishing the maximum similarity of the test sample to the known teacher-defined tables.
3. Search for all maxima of the function $U_{is}=f(\lambda)$ and determine the groups with the largest and smallest absolute values of U_{is}.
4. Identification of sections of the function $U_{is}=f(\lambda)$, which can be approximated by piecewise-linear approximations, and transitions between sections with different angular coefficients.
5. The selection of areas with adjacent maxima or minima at a distance no farther than the specified value $\Delta\lambda$, and the study of the "fine structure" of the spectrum in these areas.
6. Differential processing of the 1st and 2nd order spectra in the given ranges $\Delta\lambda$.
7. Software noise filtering in the measurement and graphic display of spectra.
8. Fourier analysis of the spectra in the selected range $\Delta\lambda$.
9. Creating user interfaces for displaying control and measurement results on a network PC.
10. Development of additional dynamic measurement control commands for the Arduino controller.
11. Investigation of the fine structure of the spectra at points of local extrema weakly expressed in ΔU_{is}.
12. Determination of the inflection points of the function $U_{is}=f(\lambda)$ and investigation of their fine structure of the spectra.

Part B - substantiation and simulation of the CPS/IoT project technical solution and the development of an intelligent algorithm of its functionality according to the options for practical tasks given in 1.1.2. As an example of development, the approach described above in a theoretical review is proposed.

Report

An individual task is performed at the expense of non-audit hours for independent work of students. The report is formulated as an appendix to the report on the practical task in paragraph 1.1.,and the results in the form of a software product, tables, justifications are protected and demonstrated by students during a laboratory workshop in the audience.

Test questions

1. How can you justify the choice of the minimum required function set of the projected CPS?
2. What technical means are required to implement the selected function set?
3. What type of signals is used to control the PO in the projected CPS?
4. Justify the specifications requirements for the automation base module of the appropriate variant of PO.
5. Describe the block diagram of the intelligent control and data processing algorithm of the selected CPS/IoT variant.
6. What type of CPS/IoT project is appropriate to implement by cost-effectiveness for a given option?
7. Describe the set of commands for the basic PO-control module based on the selected Arduino/ESP32 platform required for its functionality.
8. What libraries in the Arduino IDE are included for developing Arduino Nano functional algorithms?
9. What libraries in the Arduino IDE are enabled for the development of ESP32 functional algorithms?
10. How to create an ESP32-based web server to implement IoT?

Recommended literature

1. "uln203 datasheet pdf, Datasheet4U.com", *Datasheet4u.com*, 2019. [Online]. Available: https://datasheet4u.com/share_search.php?sWord=uln203.
2. "Hack Stepper Motor EM-211 and EM-210", *blog.zerokol.com*, 2019. [Online]. Available: https://blog.zerokol.com/2012/09/hack-stepper-motor-em-210-and-em-211.html.
3. A4988. DMOS Microstepping Driver with Translator and Overcurrent Protection. *Pololu.com*, 2019. [Online]. Available: https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf.
4. Internet of Things for Industry and Human Application. In Volumes 1-3. Volume 1. Fundamentals and Technologies / V. S. Kharchenko (ed.) – Ministry of Education and Science of Ukraine, National Aerospace University KhAI, 2019. - 605p.

3 POWER-OVER-ETHERNET BASED TRANSDUCER NETWORKS FOR CYBER PHYSICAL SYSTEMS (SEMINAR 2)

I. M. Lobachev, DrS, Assoc. Prof. M.V. Lobachev (ONPU), DrS, Prof. V. Kharchenko (KhAI), Assoc. Prof., PhD H. Vorobets (ChNU)

The aim of the seminar: study the features of Power over Ethernet (PoE) architecture and technology as a hybrid CPS/IoT model, get acquainted with the concepts of configuring and adapting the system to different PoE scenarios.

Learning tasks:

- mastering the analysis methodology and technical implementation of the concept of hierarchically modular PoE architecture as a smart CPS/IoT system;
- gaining practical skills in determining the requirements for network infrastructure components when using them to implement a hybrid CPS/IoT system with PoE technology;
- studying ways to automate the process of accounting data about switches, hubs and sensors in an Ethernet PoE network;
- familiarization with the methodology of intelligent processing and storage of data from sensors in branched networks with PoE technology;
- study of architectural features and methods of the practical application of neural networks in PoE implementation of CPS/IoT;
- mastering testing methods for architecture and technical solutions of intelligent PoE network.

Preparation for the seminar includes the following steps:

1) familiarization with the purpose and objectives for the seminar;
2) a thorough study of theoretical material at a lecture course [1], recommended and independently processed literature;
3) analysis of the general list of questions for the seminar, as well as for the individual task in the context of the concept of PoE technology as a hybrid CPS/IoT model;
4) implementation and preparation of the report on an individual task according to the recommendations given in paragraph 1.1.2;
5) presentation at the seminar and public discussion of the report.

Basic recommendations for theoretical material

When studying theoretical material on this topic, first of all, attention should be paid to the goal of PoE technology implementation – reducing the complexity level of hybrid CPS/IoT models for implementing smart urban modern infrastructure projects [2, 3]. The method that is proposed to be taken as a basis is to diversify [4] existing technical solutions and expand their functionality [5, 6]. Similar approaches were previously used, for example, in the field of radio communications and telecommunications. However, the modern possibilities of using high-performance computing on embedded and distributed resources, as well as network technologies [7, 8], allow increasing the amount of information transmitted between infrastructure objects by several orders of magnitude, and intellectualizing its processing. The use of Raspberry Pi, BeagleBone, and the like modules allow optimizing the computational load distribution between edge devices and cloud resources, optimizing data marshaling and improve the energy efficiency of smart systems. A unique solution, in this regard, is the application of the potential of neural networks. This approach allows to implement complex algorithms for data analysis and to adapt and reconfigure the system in various scenarios, depending on the information obtained. The results of modeling the proposed solutions are also interesting and significant.

Recommendations for the seminar preparation

In order to form students' holistic view about the generalized CPS/IoT model using PoE technology, it is recommended to familiarize the full list of self-study questions at the lectures [1] and the recommendations of this section and find short answers to them. The recommendations in [9, 10] will also be useful. The individual task for the seminar requires a thorough study of the material and preparation for a 5-7 minutes report and a presentation of 12-15 slides.

For better seminar preparing, it is suggested to study the following questions:

1. What problems were reviewed in the conceptual development of the PoE architecture as a smart CPS/IoT system?
2. What hardware and software have different authors used to test the validity and feasibility of the PoE concept?
3. What are the advantages and disadvantages of the PoE method?

4. What is the essence of computing load optimization when using the concept of hierarchically-modular implementation of PoE architecture?
5. Justify what are the component requirements when using them to implement a hybrid CPS/IoT system using PoE technology?
6. Describe how to classify the sensor hubs in the PoE architecture? What is the functionality of the Master Master Slave class?
7. Describe the features of the PoE-system configuring. How is the main PoE-system configuration file formed and where is it stored?
8. What is called a logical cluster of sensors? How is it formed?
9. What are the functions of the server, how is it implemented and where is it located?
10. Describe the modes for adapting the system to any PoE scenario. Compare their advantages and disadvantages.
11. What is the difference between the algorithms of adaptative modes and system reconfiguration?
12. How is the data processing on the web site organized?
13. How to automate the process of accounting data from a configuration file containing information about switches, hubs, and sensors?
14. What tasks does CRON perform in PoE technology? Describe possible scenarios.
15. What format is the data stored in? Analyze the features of data storage technology used.
16. What is the essence of a hierarchical data format? How can this affect the speed of data access?
17. Describe a high-level diagram that describes the hierarchy of the PoE system.
18. What parameters of the Movidius (Intel) mobile neural computing chip provide it with advantages for implementing the neural network concept based on it?
19. Describe the architecture and method used to optimize the PoE neural network.
20. Describe the methodology for PoE network testing. How were the network scaling capabilities tested?
21. Which sensors and which conceptual issues of system energy efficiency were inspected during testing?
22. Justify the recommendations for deploying an intelligent PoE network.

Requirements and recommendations for the preparation and execution of the report are similar to the requirements described in paragraph 1.1.2. for the practical work.

Recommended literature

1. Internet of Things for Industry and Human Application. In Volumes 1-3. Volume 1. Fundamentals and Technologies / V. S. Kharchenko (ed.) – Ministry of Education and Science of Ukraine, National Aerospace University KhAI, 2019. - 605p
2. R. Goldstein and D. Neuman, “Mega-buildings: Benefits and opportunities of renewal and reused the essential role of existing buildings in a carbon neutral world,” in Proceedings of the American Institute of Architects National Convention and Design Exposition held in Miami, Florida, USA, 10-12 June, 2010.
3. M. Magno, T. Polonelli, L. Benini, and E. Popovici, “A low cost, highly scalable wireless sensor network solution to achieve smart led light control for green buildings,” IEEE Sensors Journal, vol. 15, no. 5, pp. 2963–2973, 2015.
4. U. M. Kulkarni, D. V. Kulkarni, and H. H. Kenchannavar, “Neural network based energy conservation for wireless sensor network,” in 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon). IEEE, aug 2017.
5. N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan, and D. Estrin, “A wireless sensor network for structural monitoring,” in Proceedings of the 2nd international conference on Embedded networked sensor systems. ACM, 2004, pp. 13–24.
6. Cisco Catalyst 4500E Supervisor Engine 8-E Configuration Guide (Wireless), Cisco IOS XE Release 3.7E, 2nd ed. Cisco INC, 2014. [Online]. Available: http://www.cisco.com/c/en/us/td/docs/switches /lan/-catalyst4500/ XE3-7-0E/wireless/configuration-guide/b_37e_ 4500sup8e_cg.html
7. Cisco Systems INC, Cisco Catalyst UPOE Power Splitter, 2015. [Online]. Available: https://-developer.cisco.com/fileMedia/download /99c67d92-8089-44b9-980a-9bc304abf739
8. Related Programmes to Embedded Systems and Internet of Things (ES-IoT) MSc [https://www.ncl.ac.uk/postgraduate/courses /degrees/embedded-systems-internet-of-things-msc/relateddegrees.html]
9. Master's programme in Information and Network Engineering [https://www.kth.se/en/studies/master/information-and-network-engineering/master-s-programme-in-information-and-network-engineering-1.673817]

4. MODEL-BASED SYSTEMS ENGINEERING FOR THE CYBER-PHYSICAL SYSTEMS

Assoc. Prof., Dr. R. K. Kudermetov, Assoc. Prof. Dr. M. Yu. Tiahunova, Senior Lecturer O. V. Polska (ZNTU)

4.1 Model based design of CPS using UML 2 and MARTE profile (Laboratory Work № 7)

The aim of the laboratory work: to learn and gain the skills in a model-based design approach, learn how to use the UML 2 and the MARTE profile for specification of CPS components, in particular its temporal characteristics and resources, which implement functional and non-functional requirements for CPS.

Training participants: lecturers, scientists, technical staff, MSc students department of the university.

Theoretical information

Model-Based Design (MBD) is a powerful design technology for CPS. The basis of this technology is the models that specify the basic structural, behavioral, operational, qualitative, parametric and other properties of the system. Models can have different degrees of abstraction, complexity and detail, depending on the stage of development. Consequently, models can evoluate along with system evolve or system development stadies. Therefore, models can be used for early project analysis; assist in the separation of problems, traceability, tracking, impact analysis and synthesis. Using models, it is possible to identify structural defects earlier than at the prototyping stage, with a much higher cost. In the later stages of development the model are the basis for testing, verification and validation of the system. MBD is part of an even more fundamental methodology for creating complex systems – model-based systems engeneering (MBSE).

UML was developed to support the modeling of software applications with a program-oriented "logical" view of the world and with little attention paid to the characteristics of underlyin computing technologies. [1]. In the domains of real-time embedded systems and CPS which have more stringent requirements for quality of service, due attention should be paid to technological problems, since they can play a fundamental role in the design. MARTE has been designed to meet this

need. MARTE supplements the standard UML with the following main features, which are of great importance in the design of real-time embedded systems, but are not sufficiently supported in the UML standard [1]:

- the ability to define and specify different types of quantitative and qualitative measures associated with a UML model and its various elements, as well as any functional relationships that may exist between them, g.e. the worst-case execution time of a code or required throughput;
- a precise, comprehensive, and flexible model of time, which can be adapted to suit application needs, including physically distributed systems;
- the ability to accurately model hardware resources, that is, model elements that represent entities with a physical underpinning, such as processors, memory, input and output devices, networks, and so forth;
- the ability to accurately model hardware resources, that is, model elements that represent entities with a physical underpinning, such as processors, memory, input and output devices, networks, and so forth;
- the ability to accurately model software resources specific to real-time and embedded software, such as threads, processes, or mutexes;
- the ability to accurately capture the relationships between software applications and the computing platforms that support them.

With these added capabilities and appropriate formal analysis methods and corresponding tools, it is possible to automatically or semi-automatically predict or validate key performance indicators of a proposed design, long before committing to its implementation. This enables early detection of design flaws thereby greatly reducing engineering risk.

MARTE [2] replaces an earlier standardized UML profile called the UML Profile for Scheduling, Performance, and Time (SPT) [3].

Because MARTE is defined as a proper profile of UML, it can in principle be used with any UML tool. Moreover it can be combined with other complementary profiles, such as the SysML profile for system engineering.

Example of work execution

In this laboratory work the Pedestrian Traffic Light is considered as an example of CPS. Our simple Pedestrian Traffic Light (PTL) model includes a controller that controls the work of the PTL, three Light-

emitted diode (LED) traffic signal modules (LSM) and LED Countdown Display (LCD). Thus, the object-oriented UML model of the system can consist of three classes: `TrafficLightController`, `LightTrafficSignalModule` and `CountdownDisplay`.
The UML class-diagram is shown in Fig. 4.1. This is a very simple class-diagram, suitable only for studying the basic properties of PTL. It carries information to the developer of the system only about the basic properties and functions of the elements of the system. When developing such a system, the developer needs to know many other properties of the system, for example, what speed the controller should have, the sequence of switching on and off of LSM and LCD, the power consumption of system components, etc.

The PTL system is a simple example of CPS as it contains a cyber-part (controller) and a physical part (LSM and LCD). The task of the designer is to design these parts and design a system from these parts or design the system using off-the-shelf components. For this, it is necessary to take into account and coordinate many dissimilar and even articulate requirements for cyber and physical parts in order to synthesize a full-featured system. For this, it is necessary to take into account and harmonize many heterogeneous and even contradiction requirements for cyber and physical parts in order to synthesize a full-featured system. PTL is a built-in real-time system, since it contains embedded intellectual components that hardware or software implement the specified system operation algorithms that respond to changes in operating conditions (time of day, traffic intensity of vehicles and pedestrians, the need for constant self-diagnosis of the system). Taking into account these arguments, the design of such, in nature interdisciplinary system should be carried out using the system engineering approaches, which are advisable to carry out using the methodology of designing systems based on models.

PTL is only part of a wider vehicle and pedestrian control system and can be part of smart city control systems. Various modifications of the PTL elements can be included to automotive, rail, sea and air vehicle control systems, parking systems, lighting control systems, etc. For fixing and reusing models of the controller, embedded computer, LSM and LCD, it is advisable to create a profile with the stereotypes of these components. A variant of the profile, which the stereotypes of the listed components and was created with the help UML 2, is shown in Fig. 4.2.

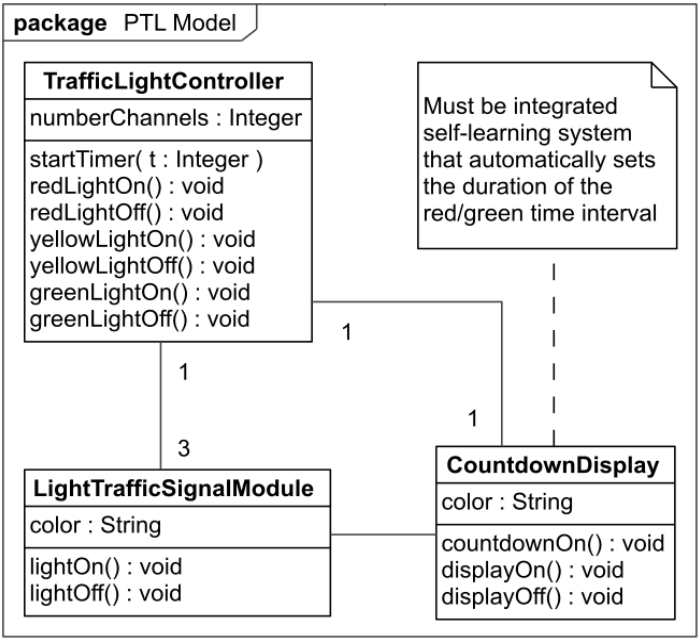


Fig. 4.1 – Simple class UML- diagram of PTL

This profile includes stereotypes of «`TrafficController`», «`Computer`», «`TrafficLighter`» and «`CountdownDisplay`». Some constraints are defined for these stereotypes, for example, «`name must be unique`», «`architecture=x86`» and others. These constraints must be *true* and can be verified by the automated modelling tools. The optional tags are defined for the stereotypes, for example, «`power`» and «`minTemperature`», which can be assigned specific values when the instantiation, i.e. the creation of instances of components. The constraints and the tags introduced will now always be the properties of classes and instances of classes that will be stereotyped by stereotypes from this profile.

The disadvantage of stereotypes introduced in the profile in Fig. 4.2, is that tag types do not carry useful information. Almost all of them are of type `String`. This choice is due to the fact that when creating instances of classes stereotyped by these stereotypes, it is necessary in addition to determination tags values to define the units of measure of these tags.

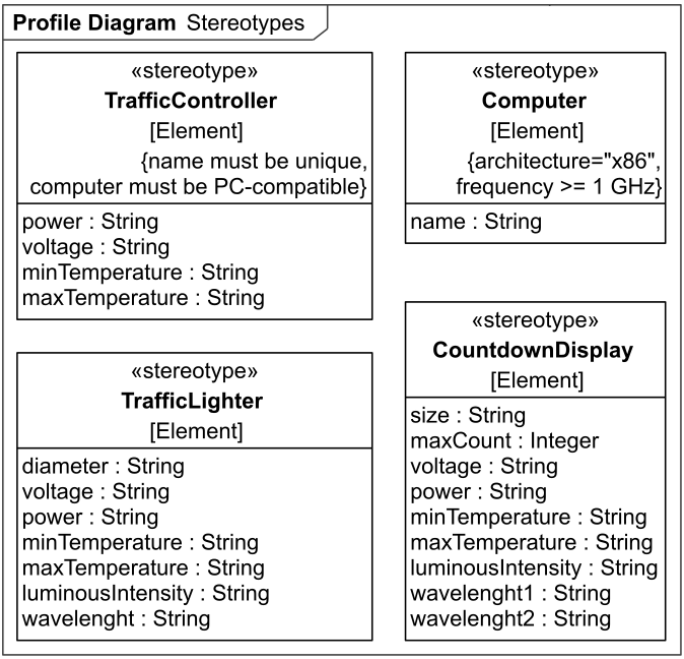


Fig. 4.2 – Profile for PTL domain

UML 2 and profile MARTE allow to introducing new physical data types. MARTE provides standardized mechanisms for determining physical quantities, which are called *non-functional properties* (NFP). For this MARTE provides the following features:

- a standard library of physical data types that represent physical dimensions, such as volume, energy, duration, mass, and so on; the ability to specify concrete literal values for physical data types, e.g., `"500 ms"`;
- the ability to extend the library of physical data types with new domain-specific and application specific types.

In addition, MARTE has its own *Value Specification Language* (VSL) for expressing physical units. Using the physical units predefined in MARTE, it is possible to create units derived from them. For this, the MARTE profile has the «`Unit`» stereotype to identify the enumeration literals that represent the measurement units. This stereotype has three optional attributes: `baseUnit` – attribute representing a unit that serves as a base for this unit; `offsetFactor` – attribute specifying a numerical

offset for computing the value of this unit relative to the base unit; `convFactor` – optional attribute representing the quantitative relationship to the `baseUnit`.

Figure 4.3 demonstrates the using of MARTE standard physical unit types and the adding of new physical data types. Here we have created new units of measurement for the specification of such quantities as the wavelength and luminous intensity of LSM, the allowable ambient temperature of the instruments. To express the time characteristics of the PTL algorithm, we divided the time units into seconds and milliseconds using standard MARTE units of time.

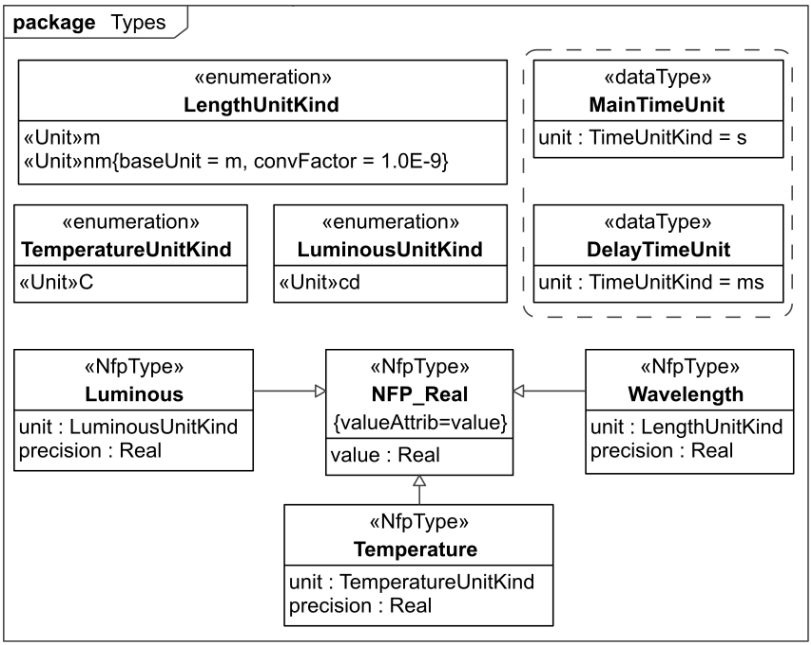


Fig. 4.3 – Adding new physical data types

Using the new physical data types introduced, we can specify the tag dimensions for traffic light stereotypes. Now all tags will have uniquely defined dimensions. The new version of the traffic light profile is shown in Fig. 4.4.

MARTE has one of the fundamental mechanisms for specifying a platform that provides system implementation. This mechanism is provided with a resource concept. It is known that any resource of the

system or device may be exhausted. Engineers should answer the question: will there be enough resources available to meet system requirements? That is, will the supply of resources meet the demand? MARTE uses the well-known client-server pattern for modeling the relationship between an application and the underlying platform. The MARTE «Resource» stereotype is the common predecessor (parent) of a very large variety of specializations based on nature and purpose of the resource: *processing resources*, *storage resources*, *communication resources*, *concurrency resources*, *mutual exclusions resources*, *device resources*, *timing resources*. These types can be applied to model elements: classifiers as well as to elements that represent instances (objects and links, lifelines in sequence, etc.). Many resource stereotypes are related by relation of "inheritance" (generalization). Resource stereotypes have many specific for each of them restrictions and tags that allow you to specify the requirements (properties) to the platforms that are necessary to ensure the feasibility of systems.

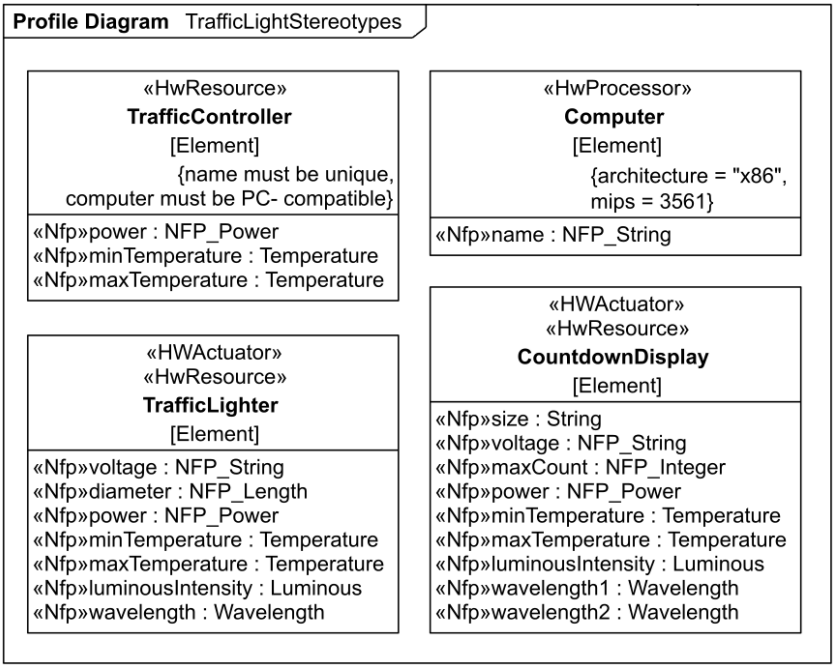


Fig. 4.4 – New version profile for traffic light domain

Using the MARTE resource stereotypes we additionally stereotyped the stereotypes of profile `TrafficLight`. As can be seen from the diagram in Fig. 4.4, we used the «`HwResource`», «`HwProcessor`», «`HwActuator`» stereotypes. Thus, the stereotypes of the `TrafficLight` profile have the constraints and tags introduced by us, as well as the constraints and the tags of the MARTE stereotypes.

Now we can use the created profile for stereotyping the classes of our model. The class diagram created using the new profile is shown in Fig. 4.5. This can be seen from the class headings, where the stereotypes used are indicated above the class name. Although stereotype tags are not displayed in the diagram, they are intrinsic parts of stereotyped classes. We will use these tags explicitly to specify properties when creating instances of classes.

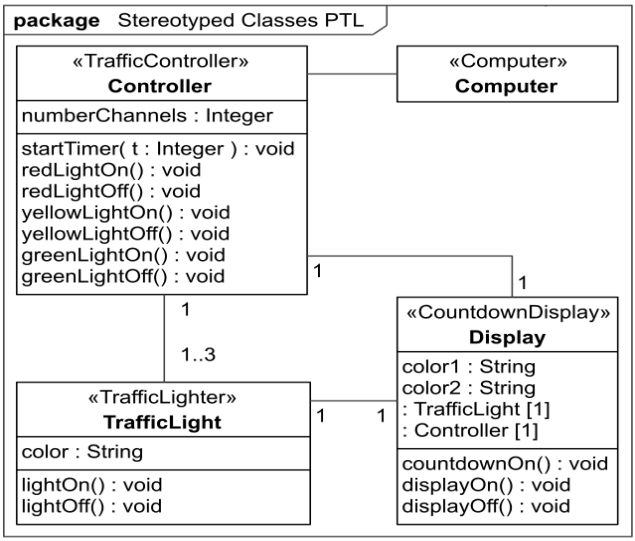


Fig. 4.5 – Class-diagram using Traffic Light profile

System analysis cannot always be accurately performed using common qualitative models, such as class diagrams. It is not enough to know that a certain class of applied tasks is deployed on a certain class of hardware nodes. Instead, you need to know not only the number of instances of these tasks and nodes, but also how individual task instances are distributed across specific node instances. In fact, the vast majority of engineering analyzes are performed on instance-based models. One way

to represent instance-based models is through an object diagram. Object diagram of PTL is shown in Fig. 4.6. Due to the presence of tags introduced using stereotypes, we are able to specify non-functional properties of the system, such as power consumption, voltage, etc. Also, thanks to the MARTE concept of resources, we have identified specific platforms for the system components that, according to the engineer's calculations, satisfy the system requirements.

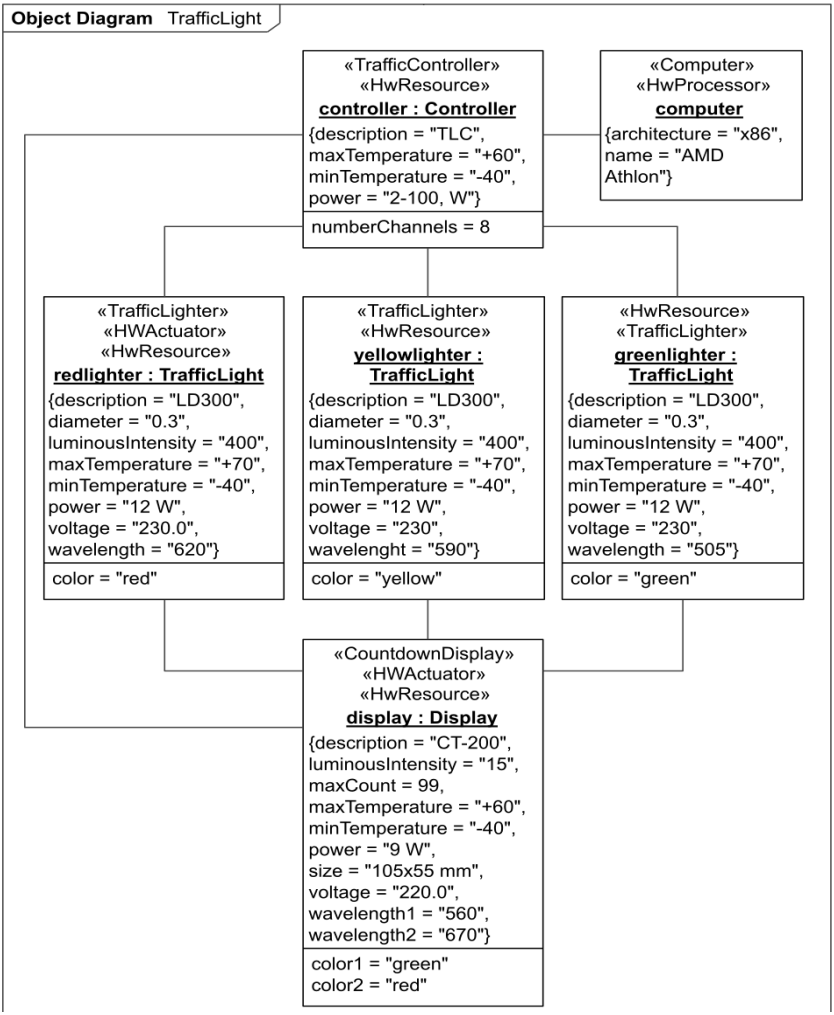


Fig. 4.6 – Object-diagram of PTL model

To modeling the behaviour of a traffic light system, we first consider in what working states the traffic light can be. It has only four simple states that represent the phases of a traffic light system in which different colours (or colour combinations) are displayed. The state machine reacts to "switching" events. The flow of the finite state machine will start from the initial pseudo state and then switching to the "Red" state. When the "switch to red-yellow" signal is received, the "Red" state will finished and the "Red-yellow" state will be executed. When the "switch to green" signal is received, the "Red-yellow" state is finished and the "Green" state is entered and executed. When the "switch to yellow" signal is received, the "Green" state is finished and the "Yellow" state is entered and executed. When the state machine receives "switch to red", the state "Yellow" will be finished and in the next state the "Red" will be active again. The representation of a traffic light system as finite state machine is shown in Fig. 4.7.

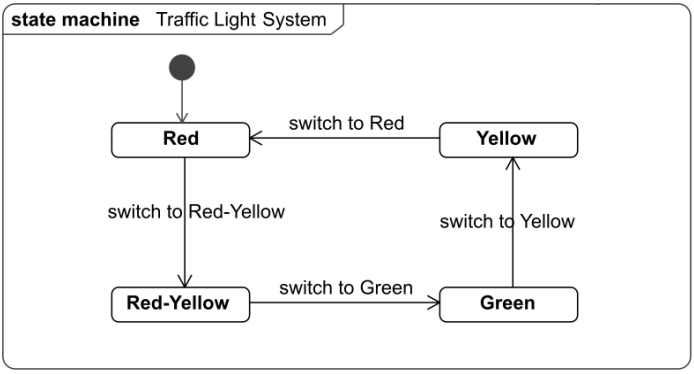


Fig. 4.7 – State machine diagram of PTL

To further refine the behavioural properties of the PTL system, we will perform modelling using an activity diagram. Activity diagram precisely specify behaviour and can depict behaviour without explicit reference to which structural elements are responsible for performing the behaviour. The activity diagram for PTL system is shown in Fig. 4.8. This diagram represents the behaviour of the LSM and LCD components. i.e. what they do and how they react to the receipt of signals arriving on their inputs. Such modeling facilitates for schedulability analysis.

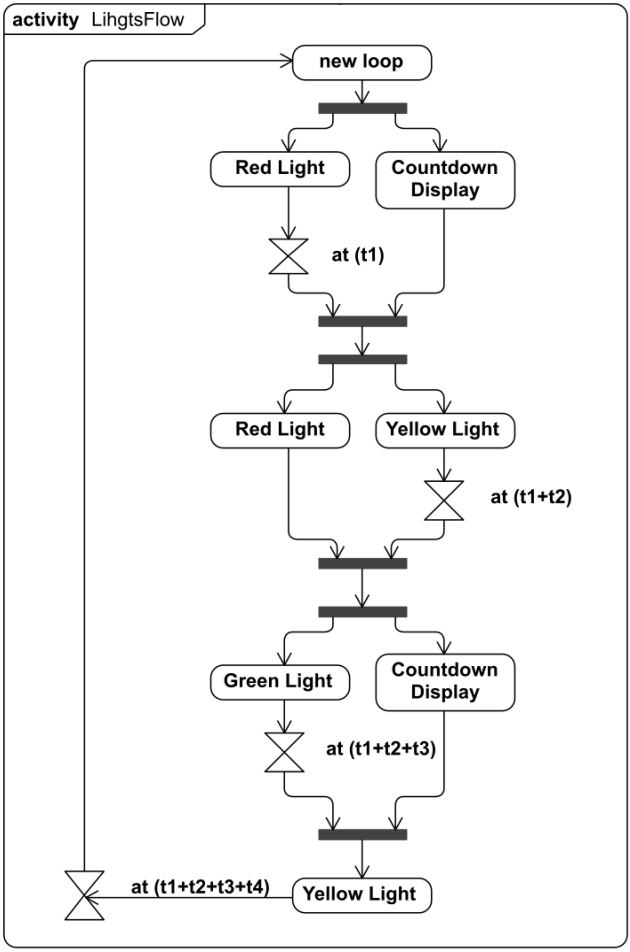


Fig. 4.8 – Activity diagram PTL

In this diagram, we have shown the relationship between activities and the flow of their control, details of the sequential and parallel operation of the PTL components, without explicitly indicating which component is responsible for the activity. This gives us the opportunity to focus on the analysis of events occurring in the system. In addition, we decided on the temporal sequence of events, and marked the moments of the occurrence of events with the time events. Specifying the time events (t1, t1+t2, etc.) allows us to determine the time intervals

during which one or another activity occurs (activity "Red Light", activity "Yellow Light", etc.).

Now we have sufficiently studied the structure, behaviour and logic of the PTL. The PMT logic is implemented by software that runs on a computer embedded in the controller. MARTE has the means to more accurately simulate properties that are typical of many real-time software applications, but which are usually under-served by general-purpose modeling languages, including UML.

In particular, MARTE offers concepts for modeling properties such as c*oncurrency*, *timeliness*, *asynchrony and interaction with physical components*, *resource limitations*. For the problems of real-time application modeling in MARTE, the concept of a software resource has been introduced, which serves as the basis for the many different manifestations required when modeling applications. In MARTE, software resources are represented by the «`SwResource`» stereotype, which is a refinement of the general «`Resource`» stereotype. The core «`SwResource`» stereotype is refined further to represent various specialized resource types, such as concurrent tasks, mutual exclusion devices, memory buffers, communication channels, etc. Software resources are usually created by a program that, after all, requires real physical processors and real physical memory. However, this physical basis is hidden behind the levels of software, the purpose of which is to provide a more abstract or "logical" view of hardware. This provides a more convenient view of the resource and the portability of the application to different hardware platforms. «`SwResource`» stereotype is an abstract stereotype, so it is necessary to use its heirs. In our case, we will not analyze PTL software more deeply, because in this PTL model we do not use the properties of concurrency or timeliness. Although in the deep design of the PTL software, it may be necessary to simulate the multi-thread software that runs concurrently tasks such as traffic light control, diagnosis, and communication with the city traffic management service.

We will consider PTL software as a sequential program. In the activity diagram (Fig. 4.8), we determined the times at which the program should generate traffic light switching events. Similarly the stereotypes introduced above, we create a stereotype of the controller software «`TrafficLightSoftware`». In this stereotype, we define the time interval tags (`t1`, `t2`, `t3`, `t4`) for each traffic light states and for the allowable time interval of the switching process "`dt`", as well as the

constraint associated with this interval `{dt<=300}`. The stereotype, class and particular instantiation of this class are shown in the diagram (Fig. 4.9).

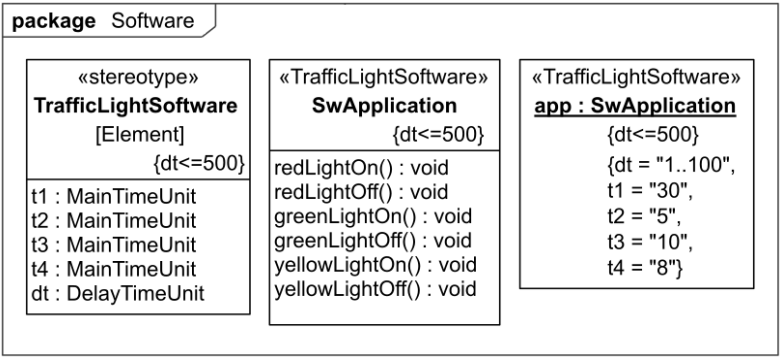


Fig. 4.9 – Stereotype, class and instantiation of software

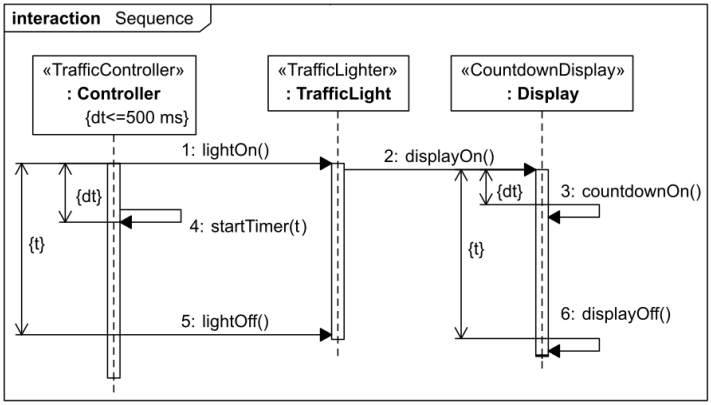


Fig. 4.10 – The sequence of interaction of LSM and LCD

Now we focus on modeling the sequence of time events and the relationship between them. To do this, first consider the sequence diagram, which represents the interaction of the LSM and LCD. As indicated in the note in Fig. 4.1, LCD must be integrated self-learning system that automatically sets the duration of the red/green time intervals. This means that the countdown counter should start when the red and green LSMs turns on. The sequence diagram of interaction LCD and LSM is shown in Fig. 4.10. Note that there are other variants of the

LCD scheme in which the interval value is transmitted from the controller via the RS-485 or RS-232 interface. The sequence diagram of interaction of controller and LSMs is shown in Fig. 4.11 (interaction with LCD is not presented here).

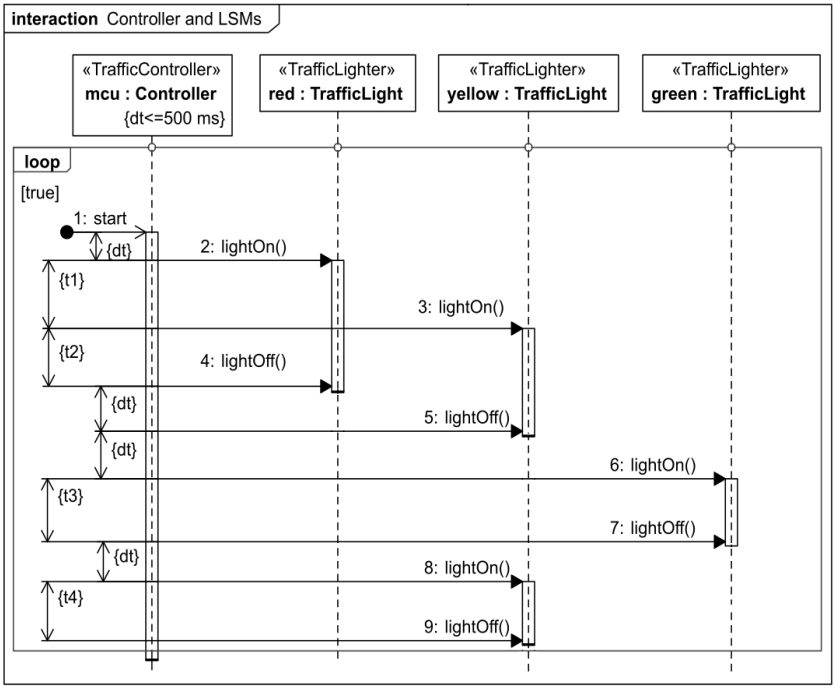


Fig. 4.11 – The sequence of interaction of controller and LSMs

Tasks for individual execution

Performing this work involves basis knowledge of the UML language. Modeling can be performed manually or using tool environments such as Modelio, Papyrus, MagicDraw, etc.

1. In this laboratory work develop your own model of the PTL system, which additionally includes pedestrian audible device (PAD). Switching on and off the PAD should be performed in parallel with turning on and off the green signal of PTL.

2. The PAD model should take into account the following PAD parameters:

- supply voltage – 220 V;
- maximum power consumption – 6 W;

- operating temperature – -45/+45 ºC;
- the sound signal (otherwise known as a rapid tick) should be a percussion sound with a frequency of 500 Hz which repeats from 8 to 10 times per second;
- the sound level of the signal should be in the range 30..90 dBA and should be more 5 dBA than ambient sound;
- the source of sound signals should be located at a height of 0.9–3.5 m from the ground level.

3. In the model, you must reproduce all the above diagrams with the added device PAD.

4. Based on the sequence diagram shown in Fig. 4.11 develop the algorithm of the controller and the program (on language C / C ++, Java, Pyton, etc).

Test questions

1. What means the abbreviation MARTE?
2. What is the purpose of developing an MARTE profile?
3. Why can the MARTE profile be combined with other UML profiles, for example with the SysML profile?
4. What entities can be included in a profile?
5. What are the stereotypes in UML used for?
6. What does MARTE add to UML?
7. What does non-functional quality of the system mean?
8. How to define new physical data types in MARTE?
9. What does the term "resource" mean in MARTE?
10. Is it possible to use stereotypes MARTE to stereotypize other stereotypes?
11. What means in UML and MARTE do you know for modeling system behaviour?

Report

The report should contain: a title page with the name of the laboratory work; goal of the work; statement of the task for individual execution; answers to test questions; results and analysis of work on an individual task; a brief analysis of the recommended literature and conclusions.

Recommended literature

1. Selic, S. Gérard, S. *Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems.* Morgan Kaufmann, Burlington, 2013.
2. "About the UML Profile for MARTE Specification Version 1.0", *omg.org*, 2009. [Online]. Available: https://www.omg.org/spec/ MARTE/1.0. [Accessed: 26- Jul- 2018].
3. P. Douglass. *Real-Time UML Workshop for Embedded Systems, Second Edition*, Newnes, Newton, MA, 2013.
4. A. Silvestre, S. S. Soares. Modeling Road Traffic Signals Control Using UML and the MARTE Profile. in *Proceedings of the International Conference on Computational Science and Its Applications, ICCSA 2012*, pp. 1-15.

4.2 Model based design CPS using SysML (Laboratory Work № 8)

The aim of the laboratory work: to learn and gain the skills in a model-based design approach with SysML, learn how to use the SysML for specification requrements for the CPS, CPS components and its inner structures, parametric analisys of the CPS.

You will learn how to organize a set of model elements in a project with package diagram, how to model the CPS structure using a block definition diagram, how to specify the internal structure of a single block with internal block diagram, how you can model and specify the requirements for the CPS, how to bind the CPS parameters and constraints using a parametric diagram, etc. As an example, the model uses PTL, whose work was analyzed in laboratory work № 7 of this module.

Training participants: lecturers, scientists, technical staff, MSc students department of the university.

Theoretical information

SysML is defined by Object Management Group (OMG) as '*a general purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behaviour,*

structure, and parametrics, which is used to integrate with other engineering analysis models [1,2]. The SysML is based on the UML.

Two types of profiles can be constructed: language profiles add new concepts to UML, or, in some cases, displace corresponding UML concepts, while annotation profiles are used to provide domain specific reinterpretations of UML models. The SysML is a language profile; it designed to be used as a full-fledged, self-contained language by systems engineers for modeling at the system level [3]. SysML uses and extends many of the features of UML 2 in such a way that it can be used to develop all kinds of technical systems. In particular, in SysML diagrams of requirements, blocks, parameters were introduced, activity diagrams were changed, the properties of standard ports were expanded (Fig. 4.12).

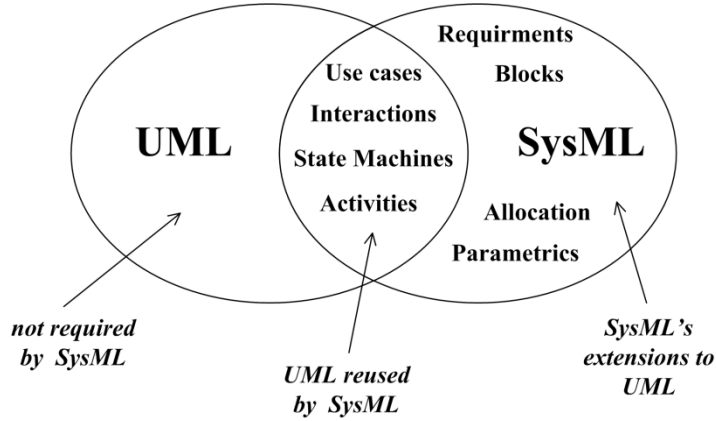


Fig. 4.12 – Relationship between SysML and UML

SysML extensions include [1]:

- the *block* is the basic unit of structure in SysML and can be used to represent hardware, software, facilities, personnel, or any other system element. The system structure is represented by *block definition diagrams* and *internal block diagrams*. A block definition diagram describes the system hierarchy and system/component classifications. The internal block diagram describes the internal structure of a system in terms of its *parts*, *ports*, and *connectors*. The *package diagram* is used to organize the model;
- *requirements diagram* captures *requirements* hierarchies and

requirements derivation. It allows a requirement to be related to model elements that satisfy or verify the requirement;

- *parametric diagram* represents *constraints* on system *property values*, allowing engineering analysis models to be produced as well as defining complex constraint relationships that can be used in verification and validation of activities.
- *allocation* relationship to represent various types of allocation, including allocation of functions to components, logical to physical components, and software to hardware.

Example of work execution

Since SysML is designed to model technical systems in any application area, in order to feel the difference between SysML and UML, consider the concept of a *block*. On the one hand, the concept of a block in SysML is similar to the concept of a class in UML. On the other hand, the concept of a class is more adapted to modeling software, and the concept of a block allows you to simulate not only *DataType* elements, but also the types of things that can flow, such as matter and energy, which is more natural for technical systems.

The *block definition diagram* (BDD) is used to represent elements such as blocks, value types, relationships between them (e.g. association, generalization and dependency). Value types are used to determine the types of things that can exist in the system. BDD is typically used to display system hierarchy trees and classification trees.

The PTL classification tree represented by BDD shown in Fig. 4.13.

It is important to build and show BDD at the very beginning of the modeling, because the elements defined in this diagram form the basis of everything else in the model of our system. The structure of the PTL with the specification of elements, types of values, constraints and the relationship between them is shown in Fig. 4.14. Note that the units of values are taken from the SysML *SI Value Type Library*. This library provides a common set of units for use throughout the model. In addition, in SysML, you can create modules derived from these types, as well as create your own types of modules used in a particular domain. Recall that in order to use standard UML, SysML, MARTE or other native libraries you have previously developed, or model library elements, you need to import packages that define these libraries and elements.

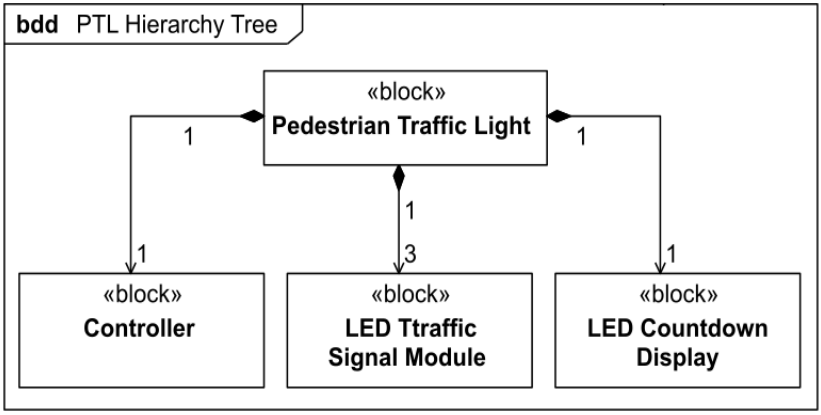


Fig. 4.13 – Block definition diagram represents the PTL and its components

A system model in MBSE can consist of hundreds, thousands, or more elements. For convenient navigation between model elements, the ability to reuse model elements, model configuration management, it is important to ensure effective organization of the model. For organizing model in SysML are used the packages. Packages are used to partition elements of the model into coherent units that can be subject to access control, model navigation and configuration management. Note that the package defines the namespace for the various elements of the model. The package is a model element, i.e. it may have other model elements nested in it according to the model hierarchy. The most significant kinds of packages used to organize models in SysML are models, packages, model libraries, and views.

The package diagram example in Fig. 4.15 describes the organization for model PTL. The packages in this diagram are primarily organized based on the types of artifacts that are planned to be developed during the design of the system based on the MBSE, including requirements, use cases, structural and behavioural models.

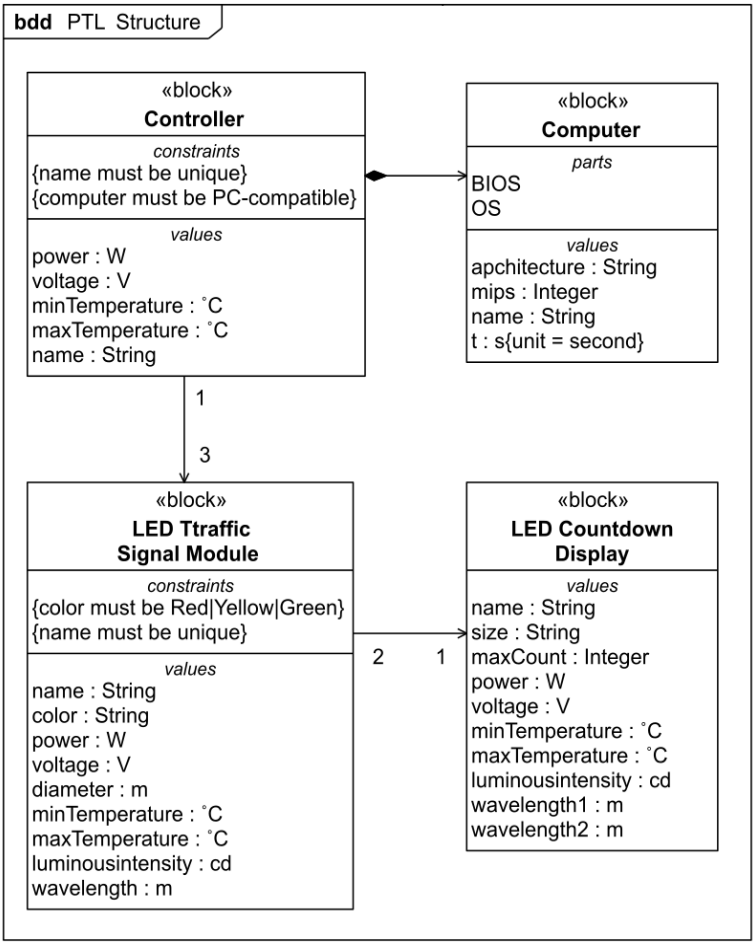


Fig. 4.14 – BDD diagram of PTL structure

The "`Value Types`" package imports the "`SI Value Type Library`" package, which is a reusable model library package available in SysML. The "`Value Types`" package uses the imported definitions of units and quantity kinds to create specific value types, which are then applied to value properties with consistent units throughout the model.

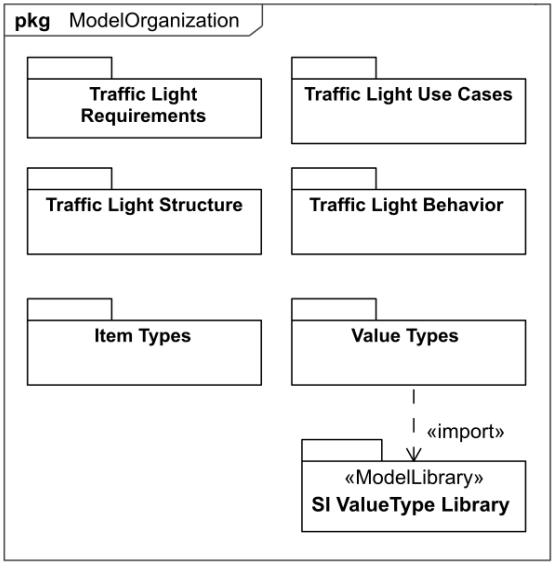


Fig. 4.15 – Package diagram of the organization of the PTL model

Package "`Traffic Light Requirements`" contains a hierarchical structure of the requirements for PTL. A feature of SysML is the presentation of system requirements in the form of model elements that have their own identifiers and requirements text. This makes it possible to organize the traceability of satisfying each requirement by other elements of the model, for example, by blocks. The requirements for the PTL system need to be captured and traced in the system model. A mission statement set provides the basis for more specific mission requirements. These mission requirements are used to identify performance measures, and then through analysis lead to a comprehensive set of system requirements for the specification of the PTL. The set of mission statements contains the statements about the paramount importance of ensuring the safety of pedestrians, who and how should ensure the safety of pedestrians, the definitions of a pedestrian, a pedestrian traffic light, etc. Figure 4.16 shows an example of the top level of the "`Mission Requirements`" package structure. Note that this is only a demo diagram, and it does not cover the entire set of system requirements, it shows only some possible examples of requirements.

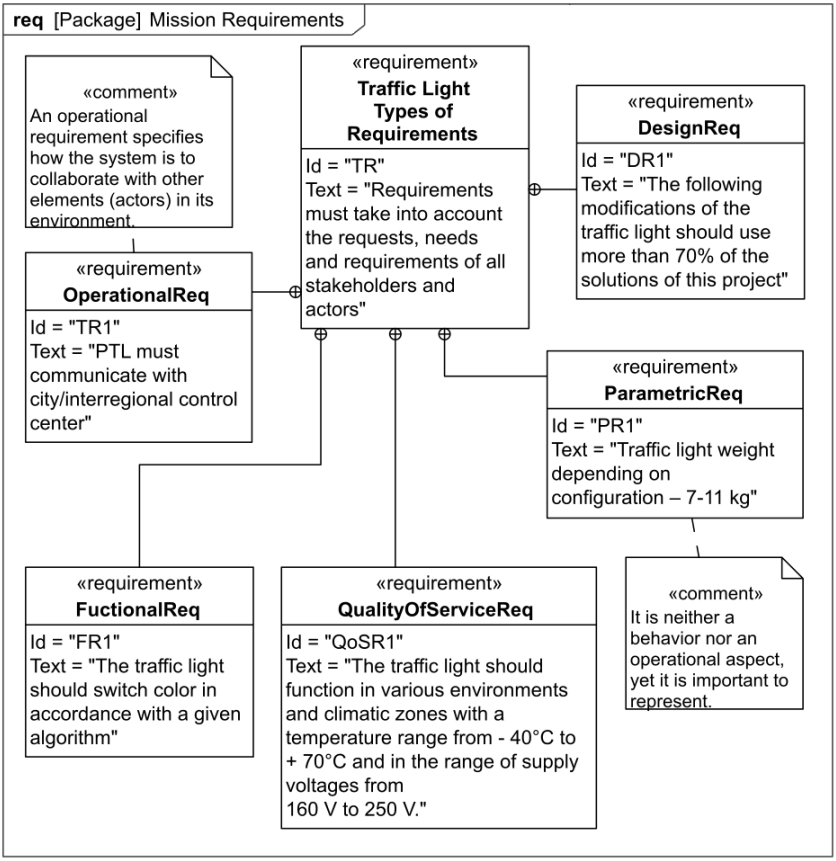


Fig. 4.16 – Mission requirements package with requirements examples

For the presentation and modeling of the internal structure of a separate unit, SysML provides a feature *internal block diagram* (IBD). Like a BDD, an IBD is a structural view of the system or one of its parts. Unlike a BDD, an IBD does not display blocks; it displays usages of blocks, i.e. the part properties of the block that is named in the header of the IBD. IBD enables you to convey additional information that you can't convey on a BDD: the connections among part properties and reference properties; the types of matter, energy, or data that flow across the connections; and the services that are provided and required across the connections. So, a BDD and an IBD provide complementary views of a block [5]. Example of IBD diagram of PTL is shown in Fig. 4.17. In

this diagram the components of PTL are presented. In addition, you can see that the components are interconnected using ports. The concept of port for is used to model interfaces. SysML allows modelers to specify a diverse set of interfaces, including mechanical, electrical, software, and human-machine interfaces. Ports are shown as rectangles intersecting the boundary of their block.

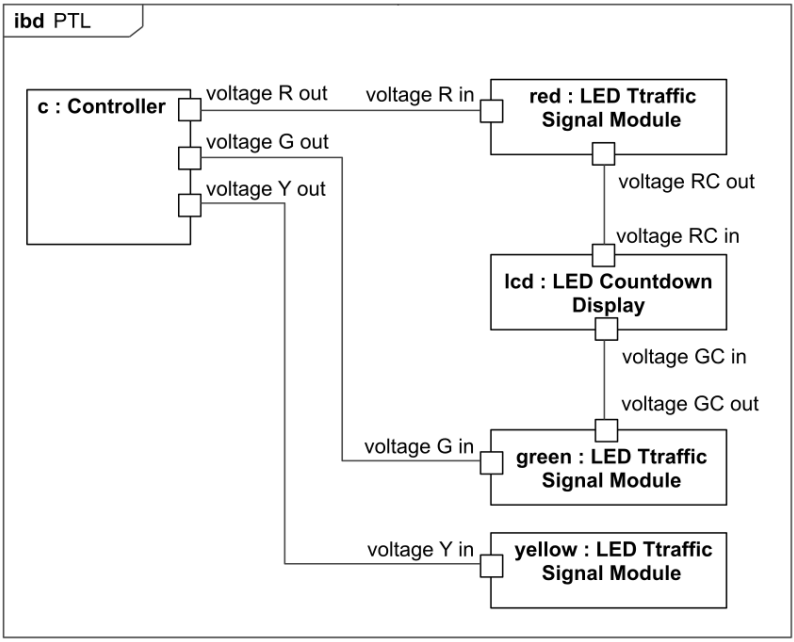


Fig. 4.17 – Internal block diagram of PTL

Another example of an IBD diagram is shown in Fig.4.18. The simple internal structure of the controller block is shown here. The computer runs the PTL operation algorithm and outputs the necessary signals to the power switches that commute the line voltage to the power circuit outputs.

Unlike UML and MARTE, SysML allows you to represent the properties of a system in the form of mathematical equations, which is very important for various types of engineering analysis and simulations. SysML introduces a *constraint block*, which is a special kind of block used to define equations so that they can be reused and interconnected. The parameters of the equations in these constraints are related to the

properties of the system that is being modeled. Constraint blocks have two main features: a set of parameters and an expression that constrains those parameters. Constraint block is shown as a round-cornered rectangle known as a *constraint property*. The small rectangles attached to the inside edge of the constraint property represent each constraint parameter and their names correspond to the parameters defined for the constraint block in its definition.

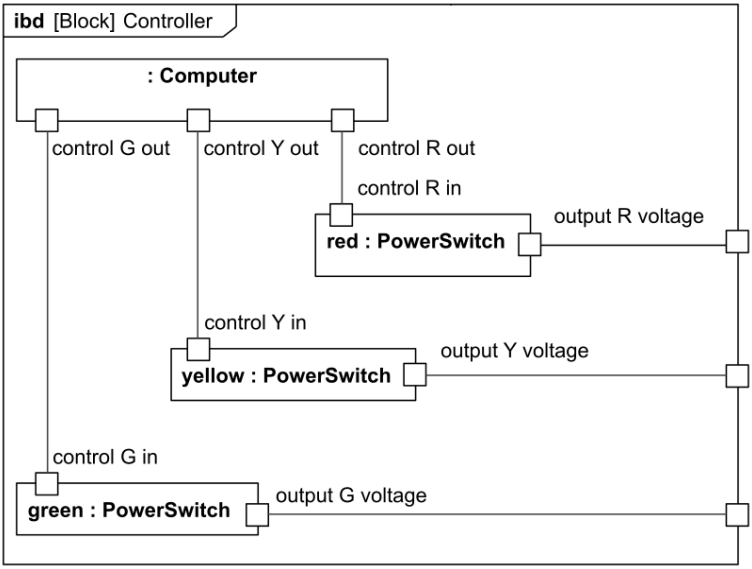


Fig. 4.18 – Internal block diagram of PTL controller

The definition of constraint blocks is performed in the BDD diagram, and the use of these blocks is carried out in a *parametric diagram*. An example of a block definition diagram containing constraint blocks is shown in Fig. 4.19. This diagram shows four constraint blocks. "`Joule-Lenz Law`", "`Total Power`" and "`Peek Power`" constraint blocks define the equations and its parameters. "`Power Consumption`" is a more complex constraint block; it includes "`Joule-Lenz Law`", "`Total Power`" and "`Peek Power`" constraint blocks.

Constraint parameters provide connection points that can be connected, via connectors, to other constraint parameters on the same or other constraint properties (e.g. in Fig. 4.20 constraint parameter "`P`" connected to "`tp`" an "`pp`" constraint parameters).

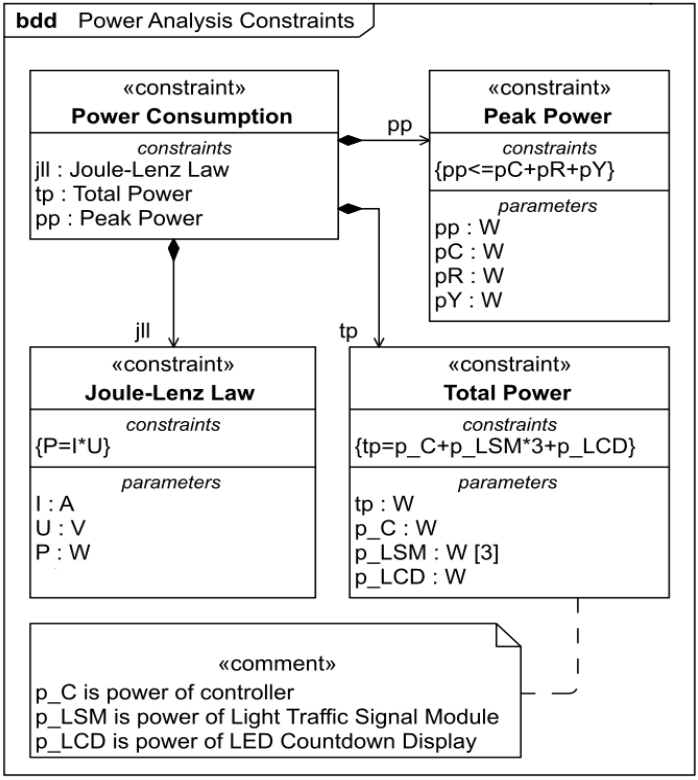


Fig. 4.19 – Definition constraint blocks for power analysis

Figure 4.20 shows a parametric diagram for "`Power Consumption`" constraint block originally introduced in Fig. 19. This block depicted as context of a parametric diagram. The diagram shows how the parameters of constraint properties "`tp`", a usage of "`Total Power`", "`pp`", a usage of "`Peak Power`" and "`P`", a usage of "`Joule-Lenz Law`", are bound together.

The block definition diagram does not show all the required information needed to interconnect its constraint properties. Particularly, it does not show the relationship between the parameters of constraint properties and the parameters of their parent and siblings. This information is provided on the parametric diagram using binding connectors, which signify *equality* relationships between their two ends. In a parametric diagram, the block is designated by the enclosing frame

and the constraint properties represent usages of the constraint blocks. The parameters of the constraint properties are bound to the value properties of the block using binding connectors. A value property is depicted as a rectangle displaying its name, type, and multiplicity.

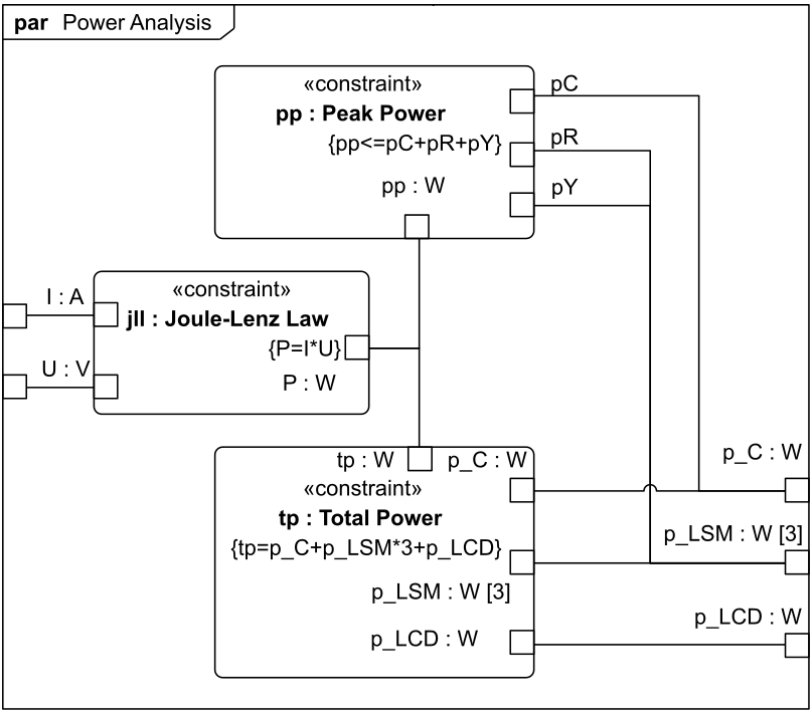


Fig. 4.20 – Parametric diagram for "`Power Consumption`" constraint block

In Fig. 4.21 the constraint on total power is shown. The "`Total Power`" constraint block is used, via a constraint property demand equation. The power demand values of all the powered devices are bound to corresponded requisite parameters of demand equation.

Figure 4.22 shows a parametric diagram for the "`Peek Power`" limitation. The equation determines the possible range of peak power, which can be only with simultaneous power consumption of the controller, red and yellow LSMs.

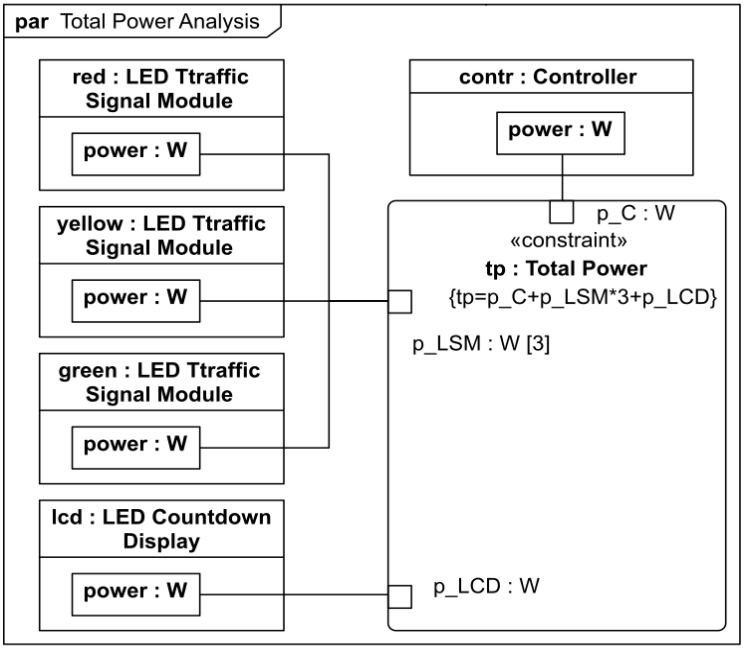


Fig. 4.21 – Parametric diagram for "`Total Power`" constraint block

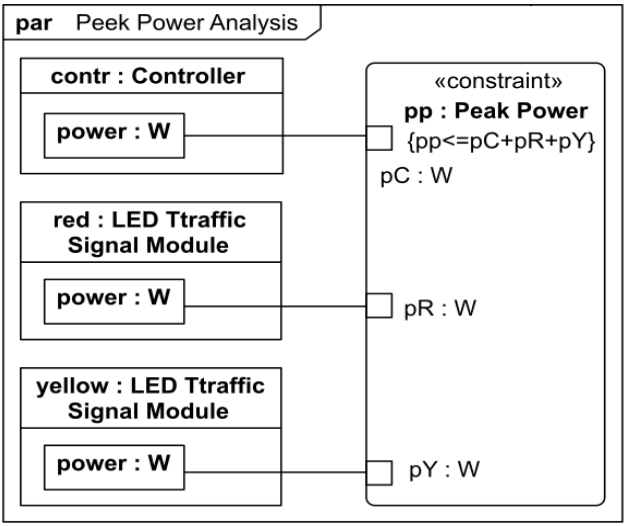


Fig. 4.22 – Parametric diagram for "`Peek Power`" constraint block

The parameters of the constraint properties "pC" (power of controller), "pR" (power of red LSM) and "pY" (power of yellow LSM) are bounded to the value property "power" of the corresponding blocks using binding connectors.

BDD diagram in Fig. 4.23 and the parametric diagram in Fig. 4.24 illustrate the use of a constraint that defines a requirement for a PTL algorithm. On BDD diagram, constraint parameters "t1", "t2", "t3" and "t4" in the "Algorithm" constraint block are coupled by demand algorithm. For these parameters are correspond the values "t1", "t2", "t3" and "t4" in "Software" block. The "Software" block has its own constraints {SW must be implemented in C language} and constraint, which defines a delay() function. On parametric diagram the relationships between these constraint parameters and the values of "Software" block constraint parameters "t1", "t2", "t3" and "t4" are specified. In addition, note that this diagram shows the time source for the "Software" block in the form of an arrow on the connector, which is called, conveyed information.

SysML introduced a very useful relationship between system model elements called *allocation*. Allocations can be considered in the following contexts: allocating requirements to structures, allocating behaviours to structures, allocating logical structures to physical structures, and allocating resources to structures [5]. An allocate relationship is a kind of dependency used to allocate one model element to another. An allocate relationship may be established between any two named model elements and provides a general purpose assignment mechanism [4].

In BDD diagram (Fig. 4.23) model element "Algorithm" is said to be "*allocated to*" model element "Software" and model element "Software" "*allocated to*" model element "Computer". In this case, allocation is similar to the concept of *deployment* in UML. An indication of the deployment of software components to equipment nodes is a general need for the development of systems such as CPS.

Thus, parametric diagrams allow you to limit the properties and behaviour of the system and can be invaluable in understanding the often complex relationships between the properties of the system. Modeling such relationships allows decisions to be made on analysis and design, and can also be used to verify whether requirements have been met or can actually be met [3].

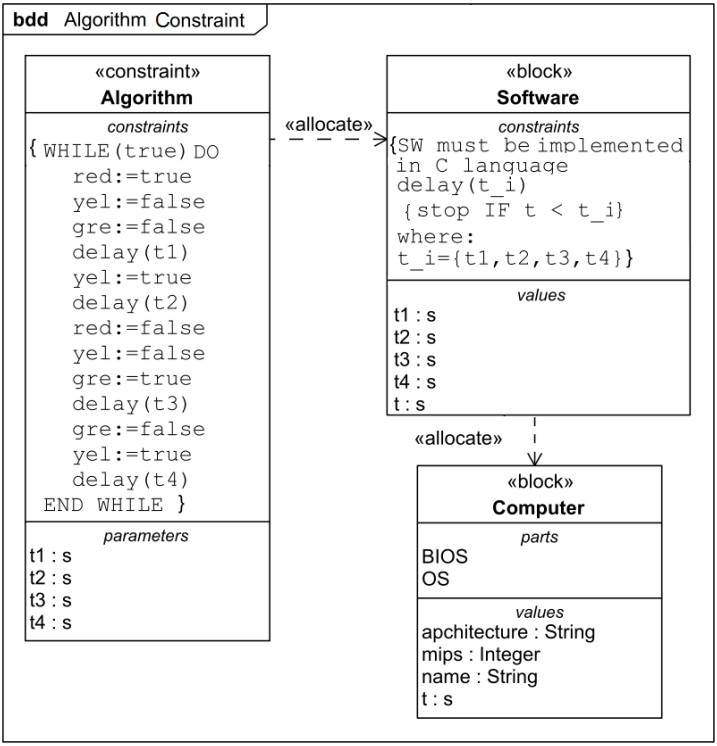


Fig. 4.23 – Definition "Algorithm" constraint block

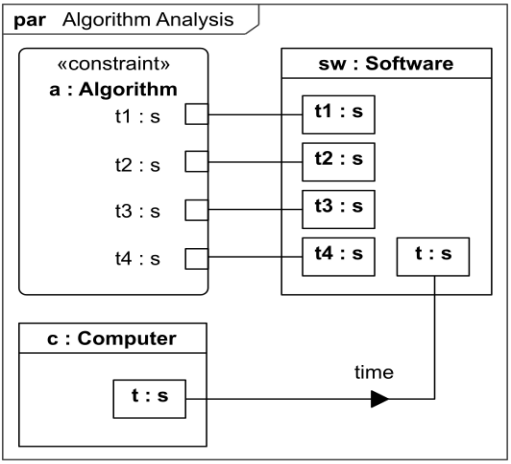


Fig. 4.24 – Parametric diagram for "Algorithm" constraint block

Tasks for individual execution

Performing this work involves basis knowledge of the UML language. Modeling can be performed manually or using tool environments such as Modelio, Papyrus, MagicDraw, etc.

1. In this laboratory work develop your own model of the PTL system, which additionally includes pedestrian audible device (PAD). Switching on and off the PAD should be performed in parallel with turning on and off the green signal of PTL.
2. Given the introduced device:

2.1. Design the BDD diagram for PTL model analogous to Fig. 4.14;
2.2. Design the IBD diagram for PTL model analogous to Fig. 4.17;
2.3. Design the IBD diagram for PTL controller model analogous to Fig. 18;
2.4. Design the IBD diagram for the PAD;
2.5. Develop requirements to the PAD and represent them in requirement diagram;
2.6. Extend the constraint block Total Power with constraint parameter "p_PAD", and design new parametric diagram for Total Power constraint block.

3. Suppose a PAD is controlled by PTL controller. Design:

3.1. Activity diagram PTL;
3.2. Sequence diagram interaction controller and PAD;
3.3. Develop the algorithm of the controller and the program (on language C/C ++, Java, Pyton, etc).

Test questions

1. What is the difference between the type of SysML profile and the type of MARTE profile?
2. What is a namespace and how is it provided in SysML?
3. What types of model elements can be represented on the block definition diagram?
4. What types of model elements can be represented on the internal block diagram?
5. What is the difference between a value and a part in a block representation?
6. How are constraint parameters represented on a block definition diagram?
7. What types of model elements can a parametric diagram represent?

8. What are the semantics of a binding connector?
9. How can constraint blocks be used to constrain the value properties of blocks?
10. Which standard properties are expressed in a SysML requirement?
11. What types of relationships can exist between requirements?

Report

The report should contain: a title page with the name of the laboratory work; goal of the work; statement of the task for individual execution; answers to test questions; results and analysis of work on an individual task; a brief analysis of the recommended literature and conclusions.

Recommended literature

1. "What is SysML", *omg.org*, 2018. [Online]. Available: https://www.omg.org/what-is-sysml.htm. [Accessed: 12- Apr- 2019].
2. "SysML Specifications - Current Version: OMG SysML 1.5", *SysML.org*, 2017. [Online]. Available: https://sysml.org/sysml-specifications/. [Accessed: 18- Jul- 2018].
3. J. Holt and S. Perry, *SysML for Systems Engineering: A model-based approach (2nd Edition)*, The Institution of Engineering and Technology, 2014.
4. S. Friedenthal, A. Moore, and S. Rick, *A Practical Guide to SysML: The Systems Modeling Language (3rd Edition)*, Morgan Kaufmann Publishers, Inc.: San Francisco, CA, 2012.
5. L. Delligatti, *SysML Distilled: A Brief Guide to the Systems Modeling Language*, Addison-Wesley, 2013.

APPENDIX A

TEACHING PROGRAM OF THE COURSE MC4 "IOT TECHNOLOGIES FOR CYBER PHYSICAL SYSTEMS"

DESCRIPTION OF THE COURSE

TITLE OF THE COURSE	Code
IoT Technologies for Cyber Physical Systems	**MC4**

Teacher(s)	Department
Coordinating: Assoc. Prof., Dr. H. I. Vorobets **Others:** Assist.of Lect. V. E. Horditsa, Assist.of Lect. O. O. Pshenychnyi	Computer Systems and Networks
Coordinating: Prof., DrS. V. S. Kharchenko	Computer Systems, Networks and Cybersecurit
Coordinating: Assoc. Prof., Dr. R. K. Kudermetov **Others:** Assoc. Prof., Dr. M. Yu. Tiahunova, Senior Lecturer O. V. Polska	Computer Systems and Networks

Study cycle	Level of the module	Type of the module
MsS	A	Bounden

Form of delivery	Duration	Langage(s)
Full-time tuition	One semester	English

Prerequisites	
Prerequisites: Computer Electronics; Computer circuitry; Microcontrollers; Computer Networks; Architecture of Computers	**Co-requisites (if necessary):** Cryptography; Means of Artificial Intelligence; Protection of information & Cybersecurity

Credits of the course	Total student workload	Contact hours	Individual work hours
4	120	60	60

Aim of the course: competences foreseen by the study program		
The purpose of the course is: deep understanding of the peculiarities of Cyber-Physical Systems and the Internet of Things technologies as specialized computer systems and networks; gaining new theoretical knowledge and practical skills of independent scientific activity and the development of new ideas in the field of IoT and CPS; mastering the methods of design, analysis and synthesis of intelligent computer systems. Acquiring relevant knowledge and skills is based on the assimilation and application of modern methods and technologies of system analysis, machine learning, teamwork, multi-criteria decision making. Attention is focused on the practical application of modern network information IoT technologies and distributed information resources (cloud, fog, edge computing) for solving scientific and applied tasks both in information support and in the modeling and design of complex CPS. A separate sub-task is the programming of computer networks for general and specialized purposes (sensor networks, mesh networks, etc.), the functioning of telecommunication and telemetric data transmission systems, and intellectual analysis and information processing. As a result of studying this course, a base of knowledge and skills for decision making and project management is formed to create complex self-organizing, self-configuring intelligent CPS based on IoT technologies application.		
Learning outcomes of course	**Teaching/learning methods**	**Assessment methods**
At the end of course, the successful student will be able to: 1. Justify the basic criteria for choosing a platform, tools and technical solutions for the development of CPS/IoT projects.	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Course Evaluation Questionnaire Testing based on alternative method of assessment
2. Analyze physical objects and processes and justify the selection of optimal algorithms for the CPS/IoT technical solutions implementation.	Interactive lectures, Learning in laboratories	Course Evaluation Questionnaire
3. Evaluate the necessary hardware and software resources for the CPS/IoT project implementation.	Interactive lectures, Learning in laboratories	Course Evaluation Questionnaire
4. Select the technical base, hardware-software tools, and platforms for preliminary	Interactive lectures, Learning in laboratories	Course Evaluation Questionnaire

prototyping and construction of CPS/IoT systems.		
5. Evaluate the feasibility and technical-economic efficiency of the proposed solutions.	Interactive lectures, Just-in-Time Teaching	Course Evaluation Questionnaire,
6. Optimize projects based on a systematic approach to the CPS/IoT analysis and synthesis.	Interactive lectures, Learning in laboratories	Course Evaluation Questionnaire
7. Identify CPS information models, describe their functionality and limitations, using existing tools, means, and technologies.	Interactive lectures, Learning in laboratories	Course Evaluation Questionnaire, Testing based on alternative method of assessment
8. Carry out modeling and simulating of individual algorithms, modules and systems of CPS/IoT using Model-Based Systems Engineering for the Cyber-Physical Systems technologies - UML, MARTE, SysML.	Interactive lectures, Learning in laboratories, Just-in-Time Teaching	Course Evaluation Questionnaire
9. Analyze cybersecurity methods and technologies of CPS/IoT systems.	Interactive lectures, Learning in laboratories	Course Evaluation Questionnaire
10. Evaluate and choose protocols and standards for the implementation of communications, processing and data transmitting for the designed CPS/IoT system using cloud, fog, edge computing.	Interactive lectures, Just-in-Time Teaching	Testing based on alternative method of assessment
11 Propose technical solutions and apply PoE technologies to create distributed smart CPS/IoT systems.	Interactive lectures, Just-in-Time Teaching	Course Evaluation Questionnaire
12. Manage the development and implementation of modern smart CPS/IoT systems in various problem-oriented industries.	Interactive lectures, Just-in-Time Teaching	Testing based on alternative method of assessment

Themes	Contact work hours							Time and tasks for individual work	
	Lectures	Consultations	Seminars	Practical work (training)	Laboratory work	Placements	**Total contact work**	**Individual work**	**Tasks**
1. CPS and IoT as a basis Industry 4.0. Basic principles for the organization and functioning of IoT and CPS ecosystems 1.1. Evolution, Standards, Development Prospects for IoT and CPS 1.2. Conceptual diagrams of IoT and CPS 1.3. Motivation and examples of IoT and CPS for industry and the human applications 1.4 IoT services and technologies for CPS	2			2			**4**	**4**	1.5. Operation of the CPS in conditions of uncertainty of input data 1.6. Built-in computer facilities reconfigurable CPS 1.7.Evolution of the cyber-component of mechatronic systems
2. System approach for the analysis and synthesis of IoT and CPS structures 2.1 Setting problem-oriented tasks 2.2 Definition and study of the target function of the CPS synthesis problem 2.3. Self-organization principles of CPSs 2.4. 3S model of CPS 2.5. Examples of structural solutions	2			2			**4**	**4**	2.6. Linear and branched algorithms for conditionally defined input data 2.7. Methods for finding optimal trajectories
3. Data processing in the CPS 3.1. Estimation of	2		2		2		**6**	**4**	3.4 Assessment

computing resources 3.2 Transmission, processing, display, storage of data 3.3. Parallel, cloud, fog, edge calculations and resources									of the effectiveness of artificial intelligence
4 Mathematical and informational support of IoT and CPS technologies 4.1. Stages and tasks of modeling of information processing 4.2. Functional IoT and CPS algorithms (in terms of application) 4.3. Mathematical models of CPS 4.4 Information models of mass service systems (MSS) in CPS 4.5 Models of Petri Networks for IoT and CPS technologies	2				2		**4**	**5**	4.6. Software package Symulink 4.7.Genetic Algorithms 4.8. Neural Networks 4.9 Methods of fuzzy logic 4.10 Bio-inspiring and bio-integrated CPS and IoT technologies
5. IoT technology in the problems of synthesis and analysis of CPS. Modern elemental and technological base for CPS and IoT 5.1. Evolution of microcontroller facilities and systems. 32- and 64-bit ARM architecture 5.2. Principles of synthesis of CPS based on industrial microprocessor modules 5.3. Principles of synthesis of CPS on the basis of programmable logic environments CPLD, FPGA	2				4		**6**	**6**	5.4. Architec-tural decisions of reconfigurable CPS 5.5. Means of artificial intelligence in CPS 5.6. Dynamic redistribution of computing load
6. Interfaces of open systems	2				4		**6**	**7**	6.5. Methods

and network protocols IoT 6.1 Sensor networks, nonstandard protocols of physical level in CPS 6.2. Mesh networks, Zigbee protocols in CPS 6.3. IR, Bluetooth, RFID for local data transmit in CPS 6.4. Network protocols and computer network programming for CPS									of information protection in IoT technology for CPS
7. Specialized software packages for simulation and synthesis of IoT and CPS 7.1. RTOS 7.2. Features of FPGA programming by Altera 7.3 Features of FPGA Programming by Xilinx 7.4 Means for the synthesis and analysis of analog and digital circuits Altium Designer	2				4		**6**	**6**	7.5. Software package Ptolemy II 7.6. Linux RTOS
8.1 IoT and scalability of CPS 8.2 Conception and advantages of Power over Ethernet 8.2.1 Conception of PoE method. State of the art 8.2.2 Advantages of the PoE method	2		2		0		**4**	**4**	8.3 Examples of PoE applications
9. System Power of Ethernet based architecture 9.1 General architecture view 9.2 System requirements 9.3 Sensor hub classes 9.4 Configuration and operation modes of the system 9.5 Parameters,	2		2				**4**	**4**	9.6 Data processing and presentation 9.7 The systems hierarchy

organization and data processing									
10. Neural networks incorporation, network testing, general integration flow 10.1 Incorporation of neural networks 10.2 Testing of the network 10.3 General integration flow	2		2				**4**	**4**	
11. Model-based systems engineering for CPS. Modeling methodologies for CPS 11.1 Rationale MBSE approaches for analysis, specification, design, and verification of CPS 11.2 An overview of the general-purpose modeling languages and its benefits for CPS 11.3 Technology platforms for CPS modeling	2				2		**4**	**4**	
12. MARTE profile of UML foundations 12.1 An introduction to UML profiles 12.2 Specifying non-functional properties 12.3 Modeling time and resources	2				2		**4**	**4**	
13. Modeling CPS with SysML and MARTE 13.1 The SysML profile 13.2 Methods of combining SysML and MARTE for modeling CPS 13.3 Basics of model-based analysis of CPS	2				2		**4**	**4**	
On the whole	**26**		**8**	**4**	**22**		**60**	**60**	

Assessment strategy	Weight in %	Deadlines	Assessment criteria
Lecture activity, including fulfilling special self-tasks	10	7,14	85% – 100% Outstanding work, showing a full grasp of all the questions answered. 70% – 84% Perfect or near perfect answers to a high proportion of the questions answered. There should be a thorough understanding and appreciation of the material. 60% – 69% A very good knowledge of much of the important material, possibly excellent in places, but with a limited account of some significant topics. 50% – 59% There should be a good grasp of several important topics, but with only a limited understanding or ability in places. There may be significant omissions. 45% – 49% Students will show some relevant knowledge of some of the issues involved, but with a good grasp of only a minority of the material. Some topics may be answered well, but others will be either omitted or incorrect. 40% – 44% There should be some work of some merit. There may be a few topics answered partly or there may be scattered or perfunctory knowledge across a larger range. 20% – 39% There should be substantial deficiencies, or no answers, across large parts of the topics set, but with a little relevant and correct material in places. 0% – 19% Very little or nothing that is correct and relevant.
Learning in laboratories	30	7,14	85% – 100% An outstanding piece of work, superbly organized and presented, excellent achievement of the objectives, evidence of original thought. 70% – 84% Students will show a thorough understanding and appreciation

			of the material, producing work without significant error or omission. Objectives achieved well. Excellent organization and presentation. 60% – 69% Students will show a clear understanding of the issues involved and the work should be well written and well organized. Good work towards the objectives. The exercise should show evidence that the student has thought about the topic and has not simply reproduced standard solutions or arguments. 50% – 59% The work should show evidence that the student has a reasonable understanding of the basic material. There may be some signs of weakness, but overall the grasp of the topic should be sound. The presentation and organization should be reasonably clear, and the objectives should at least be partially achieved. 45% – 49% Students will show some appreciation of the issues involved. The exercise will indicate a basic understanding of the topic, but will not have gone beyond this, and there may well be signs of confusion about more complex material. There should be fair work towards the laboratory work objectives. 40% – 44% There should be some work towards the laboratory work objectives, but significant issues are likely to be neglected, and there will be little or no appreciation of the complexity of the problem. 20% – 39% The work may contain some correct and relevant material, but most issues are neglected or are covered incorrectly. There should be some signs of appreciation of the laboratory work requirements.

			0% – 19% Very little or nothing that is correct and relevant and no real appreciation of the laboratory work requirements.
Course Evaluation Quest	60	8,16	The score corresponds to the percentage of correct answers to the test questions

Author	Year of issue	Title	No of periodical or volume	Place of printing. Printing house or internet link
Compulsory literature				
I. Kalyaev, V. Lohin, I. Makarov et al.	2007	Intelligent robots: a manual for universities (Russian)		Moscow : Mashinostroenie, (Mechanical Engineering),
D. Robbins and M. Tanik	2014	Cyber-Physical Ecosystems: App-Centric Software Ecosystems in Cyber-Physical Environments		Springer Science+Business Media New York
D. Moldovan, G. Copil and S. Dustdar	2018	Elastic systems: Towards cyber-physical ecosystems of people, processes, and things	vol. 57	Computer Standards & Interfaces https://www.infosys.tuwien.ac.at/Staff/sd/papers /Zeitschriftenartikel_2018_D_Moldovan_Elastic.pdf
C. Greer, M. Burns, D. Wollman and E. Griffor	2019	Cyber-Physical Systems and Internet of Things		NIST Special Publication 1900-202 https://nvlpubs.nist.gov/nistpubs/ SpecialPublications/NIST.SP.1900-202.pdf
E. Lee and S. Seshia	2019	Introduction to Embedded Systems - A Cyber-Physical Systems Approach		

N. Wiener	2013	Cybernetics or control and communication in the animal and the machine		Mansfield Centre, CT: Martino
E. Lee	2006	Cyber-Physical Systems - Are Computing Foundations Adequate		Ptolemy.eecs.berkeley.edu https://ptolemy.eecs.berkeley.edu/publications/papers/06/CPSPositionPaper/Lee_CPS_PositionPaper.pdf
S. Sarma, D. Brock and K. Ashton	2001	white paper The Networked Physical World Proposals for Engineering the Next Generation of Computing, Commerce & Automatic-Identification",		*Semanticscholar.org* https://www.semanticscholar.org/paper/The-Networked-Physical-World-Proposals-for-the-Next-Sarma-Brock/88b4a255082d91b3c88261976c85a24f2f92c5c3
J. Morra,	2019	Xilinx Adapts to an Adaptive Future of Computing		Electronic Design https://www.electronicdesign.com/industrial-automation/xilinx-adapts-adaptive-future-computing
B. Bagheri, S. Yang, H. Kao and J. Lee	2015	Cyber-physical Systems Architecture for Self-Aware Machines in Industry 4.0 Environment	vol. 48, no. 3	IFAC-PapersOnLine
N. Suda	2016	Reconfigurable Architectures and Systems for IoT Applications (Partial Fulfillment of the Requirements for		Dissertation Presented in Repository.asu.edu, https://repository.asu.edu/attachments/164110/

		the Degree Doctor of Philosophy. Arizona state university)		content/Suda_asu_0010E_15651.pdf
R. D. Sriram	2019	Toward Internet of Everything: IoT, CPS, and SNSS		Ontologforum.org http://ontologforum.org/index.php/ConferenceCall_2015_03_12
V. Harchenko, N. Zahorodna and R. Kozak	2017	Fundamentals of security and resilient computing		Diit.edu.ua http://diit.edu.ua/sites/tempus/files/full/1%20(1).pdf
	2013	CyPhERS. Cyber-Physical European Roadmap & Strategy. Research Agenda and Recommendations for Action		Cyphers.eu http://cyphers.eu/sites/default/files/ d6.1+2-report.pdf
		Edge computing primer: IoT intelligence starts at the edge		Processonline.com.au https://www.processonline.com.au/content/industrial-networks-buses/article/edge-computing-primer-iot-intelligence-starts-at-the-edge-736246826
S. Lin	2019	FPGAs, SoCs, Microcontrollers – A Quick Rundown of IoT Devices		Bitcoin Insider https://www.bitcoininsider.org/article/53125/fpgas-socs-microcontrollers-quick-rundown-iot-devices
I. Horváth and B. Gerritsen	2019	Cyber-Physical Systems: Concepts, Technologies and Implementation Principles		https://www.academia.edu/14501665/Cyber-Physical_Systems_Concepts_technologies_and_implementation_principl

				es
	2019	IoT – Using Cloud IoT Core to connect a microcontroller (ESP32) to the Google Cloud Platform		Nilhcem.com http://nilhcem.com/iot/cloud-iot-core-with-the-esp32-and-arduino
P. Papcun, E. Kajáti, C. Liu, R. Zhong and I. Zolotová	2019	Cloud-Based Control of Industrial Cyber-Physical Systems		https://www.researchgate.net/publication/332606551 _Cloud-based_Control_of_Industrial_Cyber-Physical_Systems
	2019	Zybo Zynq-7000 ARM/FPGA SoC Trainer Board (RETIRED)		Digilent https://store.digilentinc.com/zybo-zynq-7000-arm-fpga-soc-trainer-board/
	2015	Laying the groundwork for a new level of Power over Ethernet		COMMSCOPE INC http://www.commscope.com/Docs/POE_Groundwork_WP-107291.pdf
M. Magno, T. Polonelli, L. Benini, and E. Popovici	2015	A low cost, highly scalable wireless sensor network solution to achieve smart led light control for green buildings	vol. 15, no. 5	IEEE Sensors Journal
K. Chintalapudi, T. Fu, J. Paek, N. Kothari, S. Rangwala, J. Caffrey, R. Govindan, E. Johnson, and S. Masri	2006	Monitoring civil structures with a wireless sensor network	vol. 10, no. 2	Internet Computing, IEEE
U. M. Kulkarni,	2017	Neural network based energy		International Conferenc On Smart Technologies

D. V. Kulkarni, and H. H. Kenchannavar		conservation for wireless sensor network		For Smart Nation (SmartTechCon). IEEE
F. G. Osorio, M. Xinran, Y. Liu, P. Lusina, and E. Cretu	2015	Sensor network using power-over-ethernet		Computing and Communication (IEMCON), International Conference and Workshop on. IEEE.
I. Quadri, A. Bagnato, E. Brosse, and A. Sadovykh	2015	Modeling methodologies for Cyber-physical Systems: research field study on inherent and future challenges	vol. 36, no. 4	Ada User Journal
J. C. Jensen, D. H. Chang, and E. A. Lee	2011	A Model-Based Design Methodology for Cyber-Physical Systems		7th International Wireless Communications and Mobile Computing Conference (IWCMC 2011), Istanbul, Turkey
H. Posadas, P. Peñil, A. Nicolás, and E. Villar	2013.	System synthesis from UML/MARTE models: the PHARAON approach		Proceedings of the 2013 Electronic System Level Synthesis Conference (ESLsyn), Austin, TX, USA
		Object Management Group (OMG): Modeling and Analysis of Real Time and Embedded systems, version 1.1 (MARTE).		http://www.omg.org/spec/MARTE/1.1/
F. Boutekkouk, M. Benmohammed S. Bilavarn,	2009	UML2.0 profiles for embedded systems and systems on a chip	vol. 8, no. 1	Journal of Object Technology

and M. Auguin		(SoCs)		
F. Mallet, E. Villar, and F. Herrera	2017	MARTE for CPS and CPSoS: present and future, methodology and tools		Springer
J. Sztipanovits et al.	2018	Model and Tool Integration Platforms for Cyber–Physical System Design	vol. 106, no. 9	Proceedings of the IEEE
Additional literature				
B. V. Glushkov	1964	Introduction to Cybernetics (Russian)		Kiev: Publishing House of the Ukrainian SSR Academy of Sciences
	2019	Opto22 - 2173 Your IoT Primer: Bridge the Gap between OT and IT		Opto22.com https://www.opto22.com/support/resources-tools/documents/2173-your-iot-primer-bridge-the-gap-ot-and-it.
G. Chen	2010	Internet of Things towards Ubiquitous and Mobile Computing		https://www.microsoft.com/en-us/research/wp-content/uploads/2010/07 /Guihai-Chen_Oct19.pdf
Nam Yong Kim, Shailendra Rathore, Jung Hyun Ryu, Jin Ho Park and Jong Hyuk Park	2018	A Survey on Cyber Physical System Security for IoT: Issues, Challenges, Threats, Solutions	vol. 14, no. 6	J,Inf Process Syst.
D. Ratasich,	2019	A Roadmap	vol. 7	IEEE

F. Khalid, F. Geissler, R. Grosu, M. Shafique and E. Bartocci		Toward the Resilient Internet of Things for Cyber-Physical Systems		https://publik.tuwien.ac.at/files/publik_275267.pdf
J. Wan, M. Chen, F. Xia, L. Di and K. Zhou		From machine-to-machine communications towards cyber-physical systems	vol. 10, no. 3	Computer Science and Information Systems https://pdfs.semanticscholar.org/3902/a278567a7f29660e9a46ea4377c66d3414c0.pdf
J. Deshmukh		How can CPS education provide what the industry needs?		https://www.kth.se/polopoly_fs/1.518392.1550156534!/CPS%20Ed%20Workshop_JyoDeshmukh.pdf
M. Grimheden		Mechatronics Education at KTH (and Embedded Systems)		https://www.kth.se/polopoly_fs/1.518408.1550157760!/CPSED2014_Berkeley_MartinGrimheden.pdf
H. Vorobets and V. Tarasenko	2016	Self-configuring computer tools (Ukrainian)		Cyberphysical Systems: Achievements and Challenges: Proceedings of the Second Science Seminar, Lviv http://195.22.112.37/bitstream/ntb/39386/1/20-114-120.pdf
V. Golembo and O. Bochkaryov	2017	Approaches to Building Conceptual Models of Cyberphysical Systems (Ukrainian)	vol. 864, no. 1	Ukrainian Journal of Information Technology http://science.lpnu.ua/uk/scsit/vsi-vypusky/vypusk-864-nomer-1-2017/pidhody-do-

				pobudovy-konceptualnyh-modeley-kiberfizychnyh
V. Melnyk, I. Lopit and A. Keith	2016	Information exchange protocol for computer devices automatic creation in reconfigurable hardware platforms of the cyber-physical systems computing nodes (Ukrainian)		Cyberphysical Systems: Achievements and Challenges: Proceedings of the Second Science Seminar, Lviv.
L. Chen, L. Shi and W. Tan	2012	Modeling and Performance Evaluation of Internet of Things based on Petri Nets and Behavior Expression	vol. 4, no. 18	Research Journal of Applied Sciences, Engineering and Technology https://maxwellsci.com/print/rjaset/v4-3381-3385.pdf
	2019	Intel® SoC FPGAs Programmable Devices		Intel https://www.intel.com/content/www/us/en/products/programmable/soc.html
	2019	Intel® Quartus® Prime Standard Edition Handbook Volume 3		People.ece.cornell.edu https://people.ece.cornell.edu/land/courses/ece5760/DE1_SOC/Power_Estimation/qts-qps-5v3.pdf
	2019	FPGA		Xilinx.com https://www.xilinx.com/support/documentation-navigation/silicon-devices/fpga.html
H. Vorobets,	2015	The computerized	vol. 2,	Eastern-European

R. Hurzhui and M. Kuz		system with the reconfigurable architecture for monitoring environmental parameters	no. 674	Journal of Enterprise Technologies
	2019	Ptolemy Project Home Page		Ptolemy.berkeley.edu https://ptolemy.berkeley.edu/
Edward A. Lee		System Design, Modeling, and Simulation using Ptolemy II		https://ptolemy.berkeley.edu/books/Systems/chapters/IGettingStarting.pdf
	2019	Why RTOS and What is RTOS?		FreeRTOS https://www.freertos.org/about-RTOS.html
	2019	Altera Quartus II | IT Department		Information-technology.web.cern.ch, http://information-technology.web.cern.ch/services/software/quartus-ii
	2019	ISE Design Suite		Xilinx.com https://www.xilinx.com/products/design-tools/ise-design-suite.html
	2019	Altium Designer 19 - Best PCB Design Software for Engineers		Computer Aided PCB Design Software https://www.altium.com/altium-designer/
	2019	Altium Designer Documentation | Altium Designer 19.0 User Guide | Documentation		Altium.com https://www.altium.com/documentation/ru/19.0/display/ADES/Altium+Designer+Documentation
	2014	Cisco Catalyst 4500E Supervisor Engine 8-E Configuration		Cisco INC http://¬www.cisco.com/¬c/¬en/¬us/¬td/¬docs/¬switches/¬lan/-

		Guide (Wireless), Cisco IOS XE Release 3.7E, 2nd ed		catalyst4500/¬XE3-7-0E/¬wireless/¬configuration-guide/¬b_37e_4500sup8e_cg.html
	2015	Cisco Systems INC, Cisco Catalyst UPOE Power Splitter		https://-developer.cisco.com/¬fileMedia/¬download/¬99c67d92-8089-44b9-980a-9bc304abf739
J. Fitzgerald et al.	2016	Collaborative model-based systems engineering for cyber-physical systems, with a building automation case study	vol. 26	INCOSE International Symposium
P. G. Larsen et al.	2016	Integrated tool chain for model-based design of Cyber-Physical Systems: the INTO-CPS project		2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data), Vienna, Austria
S. Friedenthal, A. Moore, and S. Rick	2012	A practical guide to SysML: the systems modeling language.		San Francisco, CA: Morgan Kaufmann Publishers, Inc.
B. Selic and S. Gérard	2013	Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: developing Cyber-Physical Systems		Burlington: Morgan Kaufmann

APPENDIX B

Appendix B Code of the programs for laboratory work

1. Listing for laboratory work № 1.1

```
#include "driver/gpio.h"
static QueueHandle_t q1;
#define TEST_GPIO (25)
static void handler(void *args)
{
gpio_num_t gpio;
gpio = TEST_GPIO;
xQueueSendToBackFromISR(q1, &gpio, NULL);
}
void test1_task(void *ignore)
{
ESP_LOGD(tag, ">> test1_task");
gpio_num_t gpio;
q1 = xQueueCreate(10, sizeof(gpio_num_t));
gpio_config_t gpioConfig;
gpioConfig.pin_bit_mask = GPIO_SEL_25;
gpioConfig.mode = GPIO_MODE_INPUT;
gpioConfig.pull_up_en = GPIO_PULLUP_DISABLE;
gpioConfig.pull_down_en = GPIO_PULLDOWN_ENABLE;
gpioConfig.intr_type = GPIO_INTR_POSEDGE;
gpio_config(&gpioConfig);
gpio_install_isr_service(0);
gpio_isr_handler_add(TEST_GPIO, handler, NULL);
while(1)
{
ESP_LOGD(tag, "Waiting on queue");
BaseType_t rc = xQueueReceive(q1, &gpio, portMAX_DELAY);
ESP_LOGD(tag, "Woke from queue wait: %d", rc);
}
vTaskDelete(NULL);
}
void app_main()
{
   xTaskCreate(&test1_task, "test1_task", configMINIMAL_STACK_SIZE, NULL,
5, NULL);
}
```

2. Listing for laboratory work № 1.2

```
////////////////////////////////////////////////////////////////////////////////
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "sdkconfig.h"
void first_task(void *pvParameter)
{
while(1)
{
// some actions
}
vTaskDelete(NULL);
}
void second_task(void *pvParameter)
{
while(1)
{
// some actions
}
vTaskDelete(NULL);
}
void app_main()
{
   xTaskCreate(&first_task, "first_task", configMINIMAL_STACK_SIZE, NULL, 5,
NULL);
xTaskCreate(&second_task, "second_task", configMINIMAL_STACK_SIZE, NULL,
5, NULL);
}
```

3. Listing for laboratory work №2.1

```
#include "sdkconfig.h"
#include "esp_wifi.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_event_loop.h"
#include "nvs_flash.h"
esp_err_t event_handler(void *ctx, system_event_t *event)
{
if (event->event_id == SYSTEM_EVENT_SCAN_DONE)
```

```
{
printf("Number of access points found: %d\n", event-
>event_info.scan_done.number);
uint16_t apCount = event->event_info.scan_done.number;
if (apCount == 0)
{
return ESP_OK;
}
wifi_ap_record_t *list = (wifi_ap_record_t *)malloc(sizeof(wifi_ap_record_t) *
apCount);
ESP_ERROR_CHECK(esp_wifi_scan_get_ap_records(&apCount, list));
int i;
for (i=0; i<apCount; i++)
{
char *authmode;
switch(list[i].authmode)
{
case WIFI_AUTH_OPEN:
authmode = "WIFI_AUTH_OPEN";
break;
case WIFI_AUTH_WEP:
authmode = "WIFI_AUTH_WEP";
break;
case WIFI_AUTH_WPA_PSK:
authmode = "WIFI_AUTH_WPA_PSK";
break;
case WIFI_AUTH_WPA2_PSK:
authmode = "WIFI_AUTH_WPA2_PSK";
break;
case WIFI_AUTH_WPA_WPA2_PSK:
authmode = "WIFI_AUTH_WPA_WPA2_PSK";
break;
default:
authmode = "Unknown";
break;
}
printf("ssid=%s, rssi=%d, authmode=%s\n", list[i].ssid, list[i].rssi, authmode);
}
free(list);
}
```

```
return ESP_OK;
}
int app_main(void)
{
nvs_flash_init();
tcpip_adapter_init();
ESP_ERROR_CHECK(esp_event_loop_init(event_handler, NULL));
wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
ESP_ERROR_CHECK(esp_wifi_init(&cfg));
ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));
ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
ESP_ERROR_CHECK(esp_wifi_start());
// Let us test a WiFi scan ...
wifi_scan_config_t scanConf = {
.ssid = NULL,
.bssid = NULL,
.channel = 0,
.show_hidden = 1
};
ESP_ERROR_CHECK(esp_wifi_scan_start(&scanConf, 0));
return 0;
}
```

4. Listing for laboratory work №4.1

```
#include <driver/i2c.h>
#include <esp_log.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <math.h>
#include "sdkconfig.h"
#define PIN_SDA 21
#define PIN_CLK 22
#define I2C_ADDRESS 0x23
// No active state
#define BH1750_POWER_DOWN 0x00
// Wating for measurment command
#define BH1750_POWER_ON 0x01
// Reset data register value - not accepted in POWER_DOWN mode
#define BH1750_RESET 0x07
// Start measurement at 1lx resolution. Measurement time is approx 120ms.
#define BH1750_CONTINUOUS_HIGH_RES_MODE  0x10
```

```
// Start measurement at 0.5lx resolution. Measurement time is approx 120ms.
#define BH1750_CONTINUOUS_HIGH_RES_MODE_2  0x11
// Start measurement at 4lx resolution. Measurement time is approx 16ms.
#define BH1750_CONTINUOUS_LOW_RES_MODE  0x13
// Start measurement at 1lx resolution. Measurement time is approx 120ms.
// Device is automatically set to Power Down after measurement.
#define BH1750_ONE_TIME_HIGH_RES_MODE  0x20
// Start measurement at 0.5lx resolution. Measurement time is approx 120ms.
// Device is automatically set to Power Down after measurement.
#define BH1750_ONE_TIME_HIGH_RES_MODE_2  0x21
// Start measurement at 1lx resolution. Measurement time is approx 120ms.
// Device is automatically set to Power Down after measurement.
#define BH1750_ONE_TIME_LOW_RES_MODE  0x23
static char tag[] = "bh1750fvi";
#undef ESP_ERROR_CHECK
#define ESP_ERROR_CHECK(x)   do { esp_err_t rc = (x); if (rc != ESP_OK) {
ESP_LOGE("err", "esp_err_t = %d", rc); assert(0 && #x);} } while(0);
void task_bh1750fvi(void *ignore) {
printf(">> bh1750fvi");
i2c_config_t conf;
conf.mode = I2C_MODE_MASTER;
conf.sda_io_num = GPIO_NUM_23;
conf.scl_io_num = GPIO_NUM_22;
conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
conf.master.clk_speed = 100000;
ESP_ERROR_CHECK(i2c_param_config(I2C_NUM_0, &conf));
ESP_ERROR_CHECK(i2c_driver_install(I2C_NUM_0, I2C_MODE_MASTER, 0, 0,
0));

uint8_t data[2];
i2c_cmd_handle_t cmd = i2c_cmd_link_create();
i2c_master_start(cmd);
i2c_master_write_byte(cmd, (I2C_ADDRESS << 1) | I2C_MASTER_WRITE, 1);
i2c_master_write_byte(cmd, BH1750_CONTINUOUS_HIGH_RES_MODE, 1);
i2c_master_stop(cmd);
i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000/portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd);
while(1) {
cmd = i2c_cmd_link_create();
```

```
ESP_ERROR_CHECK(i2c_master_start(cmd));
ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (I2C_ADDRESS << 1) |
I2C_MASTER_READ, 1));
i2c_master_read_byte(cmd, data,   0);
i2c_master_read_byte(cmd, data+1, 1);
ESP_ERROR_CHECK(i2c_master_stop(cmd));
ESP_ERROR_CHECK(i2c_master_cmd_begin(I2C_NUM_0, cmd,
100/portTICK_PERIOD_MS));
i2c_cmd_link_delete(cmd);
int value = ((data[0] << 8) | data[1])/1.2;
printf("lx value: %d\n", value);
vTaskDelay(200/portTICK_PERIOD_MS);
}
vTaskDelete(NULL);
} // task_hmc5883l
```

5. Listing for laboratory work №4.2

```
#include <driver/i2c.h>
#include <esp_log.h>
#include <freertos/FreeRTOS.h>
#include <freertos/task.h>
#include <math.h>
#include "sdkconfig.h"
#define PIN_SDA 21
#define PIN_CLK 22
#define I2C_ADDRESS 0x68 // I2C address of MPU6050
#define MPU6050_ACCEL_XOUT_H 0x3B
#define MPU6050_PWR_MGMT_1   0x6B
/*
 * The following registers contain the primary data we are interested in
 * 0x3B MPU6050_ACCEL_XOUT_H
 * 0x3C MPU6050_ACCEL_XOUT_L
 * 0x3D MPU6050_ACCEL_YOUT_H
 * 0x3E MPU6050_ACCEL_YOUT_L
 * 0x3F MPU6050_ACCEL_ZOUT_H
 * 0x50 MPU6050_ACCEL_ZOUT_L
 * 0x41 MPU6050_TEMP_OUT_H
 * 0x42 MPU6050_TEMP_OUT_L
 * 0x43 MPU6050_GYRO_XOUT_H
 * 0x44 MPU6050_GYRO_XOUT_L
```

```
 * 0x45 MPU6050_GYRO_YOUT_H
 * 0x46 MPU6050_GYRO_YOUT_L
 * 0x47 MPU6050_GYRO_ZOUT_H
 * 0x48 MPU6050_GYRO_ZOUT_L
 */
static char tag[] = "mpu6050";
#undef ESP_ERROR_CHECK
#define ESP_ERROR_CHECK(x)   do { esp_err_t rc = (x); if (rc != ESP_OK) {
ESP_LOGE("err", "esp_err_t = %d", rc); assert(0 && #x);} } while(0);
void task_mpu6050(void *ignore) {
ESP_LOGD(tag, ">> mpu6050");
i2c_config_t conf;
conf.mode = I2C_MODE_MASTER;
conf.sda_io_num = GPIO_NUM_21;
conf.scl_io_num = GPIO_NUM_22;
conf.sda_pullup_en = GPIO_PULLUP_ENABLE;
conf.scl_pullup_en = GPIO_PULLUP_ENABLE;
conf.master.clk_speed = 100000;
ESP_ERROR_CHECK(i2c_param_config(I2C_NUM_0, &conf));
ESP_ERROR_CHECK(i2c_driver_install(I2C_NUM_0, I2C_MODE_MASTER, 0, 0,
0));
i2c_cmd_handle_t cmd;
vTaskDelay(200/portTICK_PERIOD_MS);
cmd = i2c_cmd_link_create();
ESP_ERROR_CHECK(i2c_master_start(cmd));
ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (I2C_ADDRESS << 1) |
I2C_MASTER_WRITE, 1));
i2c_master_write_byte(cmd, MPU6050_ACCEL_XOUT_H, 1);
ESP_ERROR_CHECK(i2c_master_stop(cmd));
i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000/portTICK_PERIOD_MS);
i2c_cmd_link_delete(cmd);

cmd = i2c_cmd_link_create();
ESP_ERROR_CHECK(i2c_master_start(cmd));
ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (I2C_ADDRESS << 1) |
I2C_MASTER_WRITE, 1));
i2c_master_write_byte(cmd, MPU6050_PWR_MGMT_1, 1);
i2c_master_write_byte(cmd, 0, 1);
ESP_ERROR_CHECK(i2c_master_stop(cmd));
i2c_master_cmd_begin(I2C_NUM_0, cmd, 1000/portTICK_PERIOD_MS);
```

```
i2c_cmd_link_delete(cmd);
uint8_t data[14];
short accel_x;
short accel_y;
short accel_z;

while(1) {
// Tell the MPU6050 to position the internal register pointer to register
// MPU6050_ACCEL_XOUT_H.
cmd = i2c_cmd_link_create();
ESP_ERROR_CHECK(i2c_master_start(cmd));
ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (I2C_ADDRESS << 1) |
I2C_MASTER_WRITE, 1));
ESP_ERROR_CHECK(i2c_master_write_byte(cmd, MPU6050_ACCEL_XOUT_H,
1));
ESP_ERROR_CHECK(i2c_master_stop(cmd));
ESP_ERROR_CHECK(i2c_master_cmd_begin(I2C_NUM_0, cmd,
1000/portTICK_PERIOD_MS));
i2c_cmd_link_delete(cmd);
cmd = i2c_cmd_link_create();
ESP_ERROR_CHECK(i2c_master_start(cmd));
ESP_ERROR_CHECK(i2c_master_write_byte(cmd, (I2C_ADDRESS << 1) |
I2C_MASTER_READ, 1));
ESP_ERROR_CHECK(i2c_master_read_byte(cmd, data,   0));
ESP_ERROR_CHECK(i2c_master_read_byte(cmd, data+1, 0));
ESP_ERROR_CHECK(i2c_master_read_byte(cmd, data+2, 0));
ESP_ERROR_CHECK(i2c_master_read_byte(cmd, data+3, 0));
ESP_ERROR_CHECK(i2c_master_read_byte(cmd, data+4, 0));
ESP_ERROR_CHECK(i2c_master_read_byte(cmd, data+5, 1));
//i2c_master_read(cmd, data, sizeof(data), 1);
ESP_ERROR_CHECK(i2c_master_stop(cmd));
ESP_ERROR_CHECK(i2c_master_cmd_begin(I2C_NUM_0, cmd,
1000/portTICK_PERIOD_MS));
i2c_cmd_link_delete(cmd);
accel_x = (data[0] << 8) | data[1];
accel_y = (data[2] << 8) | data[3];
accel_z = (data[4] << 8) | data[5];
printf("accel_x: %d, accel_y: %d, accel_z: %d\n", accel_x, accel_y, accel_z);
vTaskDelay(500/portTICK_PERIOD_MS);
}
```

```
vTaskDelete(NULL);
} // task_hmc5883l
```

6. Listing for laboratory work №4.3

```
#include <driver/spi_master.h>
void test_spi_task(void *ignore)
{
ESP_LOGD(tag, ">> test_spi_task");
spi_bus_config_t bus_config;
bus_config.sclk_io_num = clkPin; // CLK
bus_config.mosi_io_num = mosiPin; // MOSI
bus_config.miso_io_num = misoPin; // MISO
bus_config.quadwp_io_num = -1; // Not used
bus_config.quadhd_io_num = -1; // Not used
ESP_LOGI(tag, "... Initializing bus.");
ESP_ERROR_CHECK(spi_bus_initialize(HSPI_HOST, &bus_config, 1));
spi_device_handle_t handle;
spi_device_interface_config_t dev_config;
dev_config.address_bits = 0;
dev_config.command_bits = 0;
dev_config.dummy_bits = 0;
dev_config.mode = 0;
dev_config.duty_cycle_pos = 0;
dev_config.cs_ena_posttrans = 0;
dev_config.cs_ena_pretrans = 0;
dev_config.clock_speed_hz = 10000;
dev_config.spics_io_num = csPin;
dev_config.flags = 0;
dev_config.queue_size = 1;
dev_config.pre_cb = NULL;
dev_config.post_cb = NULL;
ESP_LOGI(tag, "... Adding device bus.");
ESP_ERROR_CHECK(spi_bus_add_device(HSPI_HOST, &dev_config, &handle));
char data[3];
spi_transaction_t trans_desc;
trans_desc.address = 0;
trans_desc.command = 0;
trans_desc.flags = 0;
trans_desc.length = 3 * 8;
trans_desc.rxlength = 0;
```

```
trans_desc.tx_buffer = data;
trans_desc.rx_buffer = data;
data[0] = 0x12;
data[1] = 0x34;
data[2] = 0x56;
ESP_LOGI(tag, "... Transmitting.");
ESP_ERROR_CHECK(spi_device_transmit(handle, &trans_desc));
ESP_LOGI(tag, "... Removing device.");
ESP_ERROR_CHECK(spi_bus_remove_device(handle));
ESP_LOGI(tag, "... Freeing bus.");
ESP_ERROR_CHECK(spi_bus_free(HSPI_HOST));
ESP_LOGD(tag, "<< test_spi_task");
vTaskDelete(NULL);
```

7. Listing for laboratory work №5.1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity vga is
port(clk50_in  : in std_logic;
     red_out   : out std_logic;
     green_out : out std_logic;
     blue_out  : out std_logic;
     hs_out    : out std_logic;
     vs_out    : out std_logic);
end vga;

architecture Behavioral of vga is

signal clk25            : std_logic;
signal horizontal_counter : std_logic_vector (9 downto 0);
signal vertical_counter   : std_logic_vector (9 downto 0);

begin

-- generate a 25Mhz clock
process (clk50_in)
begin
```

```
if clk50_in'event and clk50_in='1' then
if (clk25 = '0') then
clk25 <= '1';
else
       clk25 <= '0';
end if;
end if;
end process;

  process (clk25)
  begin
    if clk25'event and clk25 = '1' then
      if (horizontal_counter >= "0010010000" ) -- 144
and (horizontal_counter < "1100010000" ) -- 784
and (vertical_counter >= "0000100111" ) -- 39
and (vertical_counter < "1000000111" ) -- 519
then
--here you paint!!
red_out <= '1';
green_out <= '0';
blue_out <= '0';
else
red_out <= '0';
green_out <= '0';
blue_out <= '0';
     end if;
      if (horizontal_counter > "0000000000" )
and (horizontal_counter < "0001100001" ) -- 96+1
then
hs_out <= '0';
else
hs_out <= '1';
      end if;
      if (vertical_counter > "0000000000" )
and (vertical_counter < "0000000011" ) -- 2+1
then
vs_out <= '0';
else
vs_out <= '1';
      end if;
```

```
horizontal_counter <= horizontal_counter+"0000000001";

      if (horizontal_counter="1100100000") then
        vertical_counter <= vertical_counter+"0000000001";
        horizontal_counter <= "0000000000";
      end if;

      if (vertical_counter="1000001001") then
        vertical_counter <= "0000000000";
      end if;
    end if;
  end process;

  end Behavioral;
```

The following example shows the output of 8-pixel squares. In the middle of each square there are other multicolored squares.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity square is
  port(clk50_in  : in std_logic;
       red_out   : out std_logic;
       green_out : out std_logic;
       blue_out  : out std_logic;
       hs_out    : out std_logic;
       vs_out    : out std_logic);
end square;

architecture Behavioral of square is

signal clk25            : std_logic; -- the 25Mhz clock
signal horizontal_counter : std_logic_vector (9 downto 0);
signal vertical_counter   : std_logic_vector (9 downto 0);

begin
```

```
-- generate a 25Mhz clock
process (clk50_in)
begin
  if clk50_in'event and clk50_in='1' then
    if (clk25 = '0') then
      clk25 <= '1';
    else
      clk25 <= '0';
    end if;
  end if;
end process;

process (clk25)
begin
  if clk25'event and clk25 = '1' then
    if (horizontal_counter >= "0010010000" ) -- 144
    and (horizontal_counter < "1100010000" ) -- 784
    and (vertical_counter >= "0000100111" ) -- 39
    and (vertical_counter < "1000000111" ) -- 519
    then
      red_out <= horizontal_counter(3)
          and vertical_counter(3);
      green_out <= horizontal_counter(4)
          and vertical_counter(4);
      blue_out <= horizontal_counter(5)
          and vertical_counter(5);
    else
      red_out <= '0';
      green_out <= '0';
      blue_out <= '0';
    end if;
    if (horizontal_counter > "0000000000" )
      and (horizontal_counter < "0001100001" ) -- 96+1
    then
      hs_out <= '0';
    else
      hs_out <= '1';
    end if;
    if (vertical_counter > "0000000000" )
      and (vertical_counter < "0000000011" ) -- 2+1
```

```
   then
    vs_out <= '0';
   else
    vs_out <= '1';
   end if;
   horizontal_counter <= horizontal_counter+"0000000001";
   if (horizontal_counter="1100100000") then
    vertical_counter <= vertical_counter+"0000000001";
    horizontal_counter <= "0000000000";
   end if;
   if (vertical_counter="1000001001") then
    vertical_counter <= "0000000000";
   end if;
  end if;
end process;

end Behavioral;
```

8. Listing for laboratory work №5.2

```
{Keyboard controller}
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity KeyboardController is
   Port ( Clock : in STD_LOGIC;
      KeyboardClock : in  STD_LOGIC;
        KeyboardData : in  STD_LOGIC;
        LeftPaddleDirection : buffer  integer;
        RightPaddleDirection : buffer  integer
);
end KeyboardController;

architecture Behavioral of KeyboardController is

signal bitCount : integer range 0 to 100 := 0;
signal scancodeReady : STD_LOGIC := '0';
signal scancode : STD_LOGIC_VECTOR(7 downto 0);
signal breakReceived : STD_LOGIC := '0';
```

```
constant keyboardA : STD_LOGIC_VECTOR(7 downto 0) := "00011100";
constant keyboardY : STD_LOGIC_VECTOR(7 downto 0) := "00011010";
constant keyboardK : STD_LOGIC_VECTOR(7 downto 0) := "01000010";
constant keyboardM : STD_LOGIC_VECTOR(7 downto 0) := "00111010";

begin

keksfabrik : process(KeyboardClock)
begin
if falling_edge(KeyboardClock) then
if bitCount = 0 and KeyboardData = '0' then --keyboard wants to send data
scancodeReady <= '0';
bitCount <= bitCount + 1;
elsif bitCount > 0 and bitCount < 9 then -- shift one bit into the scancode from
the left
scancode <= KeyboardData & scancode (7 downto 1);
bitCount <= bitCount + 1;
elsif bitCount = 9 then -- parity bit
bitCount <= bitCount + 1;
elsif bitCount = 10 then -- end of message
scancodeReady <= '1';
bitCount <= 0;
end if;
end if;
end process keksfabrik;

kruemelfabrik : process(scancodeReady, scancode)
begin
if scancodeReady'event and scancodeReady = '1' then
-- breakcode breaks the current scancode
if breakReceived = '1' then
breakReceived <= '0';
if scancode = keyboardA or scancode = keyboardY then
LeftPaddleDirection <= 0;
elsif scancode = keyboardK or scancode = keyboardM then
RightPaddleDirection <= 0;
end if;
elsif breakReceived = '0' then
-- scancode processing
```

```
if scancode = "11110000" then -- mark break for next scancode
breakReceived <= '1';
end if;

if scancode = keyboardA then
LeftPaddleDirection <= -1;
elsif scancode = keyboardY then
LeftPaddleDirection <= 1;
elsif scancode = keyboardK then
RightPaddleDirection <= -1;
elsif scancode = keyboardM then
RightPaddleDirection <= 1;
end if;
end if;
end if;
end process kruemelfabrik;
end Behavioral;
```

АНОТАЦІЯ

УДК 004.415/.416:004.89](076.5)=111

Воробець Г. І., Харченко В. С., Кудерметов Р. К., Клятченко Я. М., Гордіца В. Е., Пшеничний О. О., Хамула І. С., Лобачев І. М., Лобачев М. В., Тягунова М. Ю., Польська О. В. Технології Інтернету Речей для кіберфізичних систем. Практикум / За ред. Г. І. Воробця та В. С. Харченко – МОН України, Чернівецький національний університет імені Юрія Федьковича, Національний аерокосмічний університет "ХАІ", Запорізький національний технічний університет, 2019. – 172 с.

Викладено матеріали практичної частини курсу МС4 "Технології інтернету речей для кіберфізичних систем", підготовленого в рамках проекту ERASMUS+ ALIOT "Internet of Things: Emerging Curriculum for Industry and Human Applications" (573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP).

Наведено структуру курсу, навчальні матеріали, приклади завдань для семінарів, практичних і лабораторних робіт, а також методичні рекомендації для самопідготовки і перевірки знань з дисципліни, та критерії їх оцінювання. Матеріал подається послідовно для формування цілісної картини сучасного стану, синергії, перспектив розвитку та досліджень технологій інтернету речей і кіберфізичних систем. Увага акцентується на концептуальних питаннях моделювання, аналізу, синтезу і практичного впровадження КФС, та ролі IoT на всіх етапах життєвого циклу складних комп'ютеризованих систем.

Призначено для магістрів університетів у галузі інформаційних технологій: комп'ютерних наук та інформаційних систем і технологій, кібербезпеки, системного аналізу, програмної та комп'ютерної інженерії, а також викладачів відповідних курсів, інженерів та науковців, які займаються розробкою та впровадженням технологій КФС та IoT.

Бібл. – 81, рисунків – 51, таблиць – 10.

ЗМІСТ

ABSTRACT

UDC 004.415/.416:004.89](076.5)=111

Vorobets H. I., Kharchenko V. S., Kudermetov R. K., Klyatchenko Ya. M., Horditsa V. E., Pshenychnyi O. O., Khamula I. S., Lobachev I. M., Lobachev M. V., Tiahunova M. Yu., Polska O. V. Internet of Things Technologies for Cyber Physical Systems: Practicum / Vorobets H. I. and Kharchenko V. S. (Eds.) – Ministry of Education and Science of Ukraine, Yuriy Fedkovych Chernivtsi National University, National Aerospace University "KhAI", Zaporizhzhia National Technical University, 2019. – 172 p.

The materials of the practical part of the study course MC4 "IoT Technologies for Cyber Physical Systems", developed in the framework of the ERASMUS+ ALIOT project "Internet of Things: Emerging Curriculum for Industry and Human Applications" (573818-EPP-1-2016-1-UK-EPPKA2-CBHE-JP) are presented.

The course structure, teaching materials, examples of tasks for seminars, practical and laboratory works, as well as methodological recommendations for self-preparation and knowledge testing in the discipline, and criteria for their assessment are given. The material is submitted sequentially to form a holistic picture of the current state, synergy, prospects for research and development of Internet of Things and Cyber-Physical Systems technologies. The focus is on the conceptual issues of modeling, analysis, synthesis and practical implementation of CPS, and the role of IoT at all stages of the life cycle of complex computerized systems.

Designed for Masters of Universities in Information Technology: Computer Science and Information Systems, Cybersecurity, Systems Analysis, Software and Computer Engineering, as well as teachers of relevant faculties, engineers and scientists involved in the development and implementation of CPS and IoT technologies.

Ref. – 81 items, figures – 51, tables – 10.

CONTENTS

Воробець Георгій Іванович
Харченко В‘ячеслав Сергійович
Кудерметов Равіль Камілович
Клятченко Ярослав Михайлович
Гордіца Валентина Емануїлівна
Пшеничний Олексій Олександрович
Хамула Ілля Сергійович
Лобачев Михайло Вікторович
Лобачев Іван Михайлович
Тягунова Марія Юріївна
Польська Ольга Володимирівна

ТЕХНОЛОГІЇ ІНТЕРНЕТУ РЕЧЕЙ ДЛЯ КІБЕРФІЗИЧНИХ СИСТЕМ

Практикум
(англійською мовою)

Редактори Воробець Г. І., Харченко В. С.

Комп'ютерна верстка
Г. І. Воробець,
В. Е. Гордіца

Зв. план, 2019
Підписаний до друку 27.08.2019
Формат 60x84 1/16. Папір офс. No2. Офс. друк.
Умов. друк. арк. 9,88. Уч.-вид. л. 10,62. Наклад 150 прим.
Замовлення 270819-9.

Національний аерокосмічний університет ім. М. Є. Жуковського
"Харківський авіаційний інститут"
61070, Харків-70, вул. Чкалова, 17
http://www.khai.edu

Випускаючий редактор: ФОП Голембовська О.О.
03049, Київ, Повітрофлотський пр-кт, б. 3, к. 32.
Свідоцтво про внесення суб'єкта видавничої справи до державного реєстру видавців,
виготовлювачів і розповсюджувачів видавничої продукції
серія ДК No 5120 від 08.06.2016 р.

Видавець: ТОВ «Видавництво Юстон»
01034, м. Київ, вул.. О. Гончара, 36-а, тел.: +38 044 360 22 66
www.yuston.com.ua
Свідоцтво про внесення суб‘єкта видавничої справи до державного реєстру видавців,
виготовлювачів і розповсюджувачів видавничої продукції
серія ДК No 497 від 09.09.2015 р.