STUDY OF REAL-TIME OPERATING SYSTEM INCLUDING PROGRAMMABLE LOGIC CONTROLLER FUNCTIONALITY

Ulyana Dzelendzyak, Ph. D., As.-Prof., Vladislav Vanyuk, MS Student,
Halyna Vlakh-Vyhrynovska, Ph. D., As.-Prof.,
Lviv Polytechnic National University, Ukraine; e-mail: uliana.y.dzelendziak@lpnu.ua
Mariusz Węgrzyn, Ph. D., As.-Prof.,
Cracow University of Technology, Poland

https://doi.org/10.23939/istcmtm2023.04.052

Abstract. The article describes the developed real-time operating system “VladOS”, which is a set of modules and utilities from which the required system for controlling the eco-house should be assembled. Users can load their written applications into the file system in the form of binary files. System generation involves selecting an active configuration from a list of available user applications, setting values of system configurations, compiling system module parameters using this configuration, and linking system modules into an executable program. Additionally, the performance of the developed operating system is assessed through a comparison with the widely recognized Arduino core, with the “SmartHatka” treehouse serving as a representative example. Essentially, “SmartHatka” is a Ukrainian smart eco-house.

Key words: real-time operating system; programmable logic controller; smart home automation; remote monitoring and control; apple HomeKit.

1. Introduction

Microcontrollers are widely used today for implementing control tasks in various devices, particularly 8-bit and 32-bit microcontrollers have seen significant development. They are designed to control various systems according to the developed microcontroller program. Microcontrollers are used in both domestic and industrial systems. Among enthusiasts and engineers, microcontroller boards from Arduino have gained popularity. Such controllers are used to control all peripherals. The Arduino platform has gained popularity due to its suitability for those who lack extensive knowledge of electronics. However, writing programs for microcontrollers is significantly different from writing programs for a general-purpose computer. When a program is executed on a computer, the operating system takes care of launching programs, interacting with internal and external devices, or human interaction. In contrast, an executed program written for a microcontroller represents a clear sequence of actions. Moreover, the number of tasks that the microcontroller must solve in parallel is constantly increasing. Based on this, microcontroller programs become more complex both during development and debugging. Often, the issue of multitasking arises, which needs to be resolved by the development of a control program. When developing almost any software for microcontrollers, it turns out that the program should consist of several relatively independent tasks with the ability to communicate with each other. In other words, there is a need for a general control program known as a real-time operating system (RTOS).

With the growth of knowledge and the recent introduction of the Internet of Things (IoT), interest in Arduino has experienced another significant development, becoming an essential tool for educating engineers and hobbyists. Now, the Arduino platform has started to evolve to adapt to new needs and challenges, such as IoT applications, 3D printing, and, finally, programmable logic controllers.

A programmable logic controller (PLC) is a specialized computing device designed for use in industrial control systems and other applications where system reliability is crucial. Originally, they were developed to replace hardwired relays, sequencers, and timers used in industrial automation processes, but today they are scalable and used for building automation (access control, lighting control, heating, ventilation, air conditioning, elevator and escalator control, etc.).They also serve as a good example of real-time operating systems as they have a high capability to produce outputs in response to specific inputs in short timeframes, which is a key requirement for most settings, as a second delay can disrupt the entire process.

One of the key characteristics of the PLC is its low technical programming and operational requirements. PLCs were designed to be used by both highly skilled automation professionals and ordinary technicians. Microcontrollers, on the other hand, require a careful approach. Designers need to have in-depth knowledge of electrical engineering principles and programming to complete projects using microcontrollers. Even though there are simplified platforms such as Arduino, it is still much more complex than the “plug and play” nature of PLCs both in terms of connection and ease of use.

Despite the wide variety of proposed real-time operating systems, none of them come close to the simplicity of the PLC's software model. Therefore, the problem of implementing a reliable real-time operating system with low technical requirements for operation on controllers remains relevant today.

2. Analysis of the issue

A Programmable Logic Controller (PLC) is typically a single-board mini-computer constructed based on a single-chip microcontroller and housed in a standard-sized (brick-sized) enclosure. Inputs to the PLC can be connected to buttons, joystick contacts, switches (i. e., control elements), sensors, and actuating mechanisms (motors, lamps, heating elements, valves, fans, actuators, etc.). The PLC cyclically polls input signals (control elements and sensors), executes the user program (calculates variable values) and outputs the obtained values to the actuating mechanisms [2]. In other words, the PLC repeatedly executes the same user program in a cycle.

On the market, there are many different types of PLCs available to meet customer requirements. After analyzing existing PLC boards and operating systems [3], the Arduino Mega [7] was selected as the main development platform. This choice makes it more cost-effective to implement the PLC strategy for budget-conscious systems. The board features the ATmega2560 [6] microcontroller, which includes 54 digital input/output pins (15 of which can be used as PWM outputs), 16 analog inputs, 256 kb of flash memory (quite a large capacity relative to other AVR controllers), and a clock frequency of only 16 MHz. The limited processing speed posed a challenge in implementing a reliable system, leading to a rewrite of the system kernel since the selected development environment was Arduino IDE. Thus, the entry threshold for users who decide to write an application for the “VladOS” system remained the same as with regular Arduino-core. The clock frequency can only be changed by soldering a different resonator to the board, which can be fatal for the controller. Additionally, effective dynamic stack allocation can only be implemented using assembly functions, making changes to the bit registration functionality for user processes is not recommended. Furthermore, when the code address is stored in the stack on the ATmega2560, it occupies three bytes instead of two as on other ATmega microchips. Therefore, porting to other schematics without changes to the OS is impossible.

Additionally, some functionality in the implemented kernel, although faster than the original, consumes more Flash memory. The main problem can be summarized as follows: how to select the executed software modules for the available system resources to minimize the cost-to-performance ratio [1].

To solve this problem, the following approaches were used:

1. Program code configuration. A specific set of services is chosen by conditional loading of applications and conditional linking. This approach allowed the con- struction of a minimal system, which is especially useful in real-time system generation requiring high performance.
2. Dynamic configuration. The operating system can respond to changing user requirements by efficiently altering its configuration.

3. Goal

Develop a real-time operating system for AVR ATMEGA2560 microcontrollers as a programmable electronic control unit, and evaluate its performance in managing an eco-house system built in the Arduino IDE, conducting real-world analysis through a practical case study.

4. The structure of the “VladOS”

The “VladOS” operating system consists of a kernel, main and additional services. Main system services are the modules that cannot be disabled. Errors in these modules are critical for the entire system.

Main system services include:

1. Peripherals services.
2. “Control Menu” service.
3. EEPROM Configuration DB.
4. Main “SD_File System”: bootloader, JSON configurations for objects, groups, and other settings.
5. Service of system error codes [5] “Status Code System”.
6. User application.

Additional system services are modules that can be disabled during execution or excluded from system compilation. Their absence will not trigger critical errors, and the system will continue to operate.

Additional system services include:

1. “Keylock”: Security Lock System;
2. “iHatka”: UART json [9] Serialization with WemosD1 Mini [8] and Apple Home kit;
3. Additional “SD_File System”: logger, configurations data backup;

4.1. Groups for peripherals services

Fig. 1 shows a block diagram of the devices connected to the control unit.

Input group services:

1. Button polling service “ButtonsHandler”.
2. Keypad polling service “KeypadHandler”.
3. Infrared (IR) signal processing service “IRRecvHandler”.
4. Analog signal processing from potentiometers service “PotsHandler”.
5. Analog sensor signal processing service “SensorsHandler”.
6. I2C sensor processing service “I2CHandler”.
7. Optional additional rotary encoder processing service “EncoderHandler”.

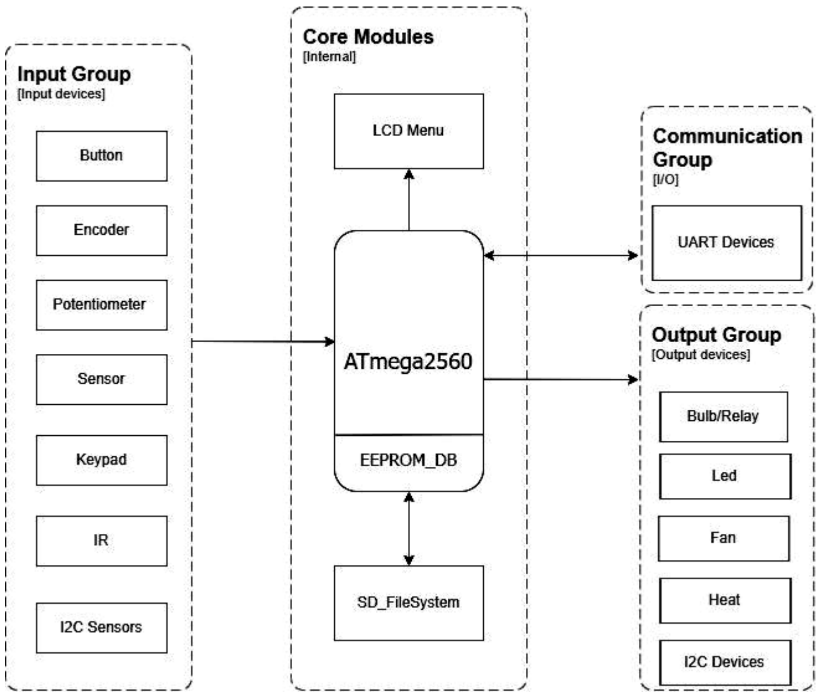

Fig. 1. Block diagram of services for connected devices

Input group devices:

1. Pull-up buttons, regular buttons, switches.
2. Keypad.
3. IR receiver.
4. Analog potentiometers: regular potentiometers, joysticks.
5. Analog sensors: temperature, humidity, pressure sensors, and others.
6. I2C sensors: sensors and devices connected to the controller via the I2C bus.
7. Additional rotary encoder (EncButton, Encoder).

Output group services:

1. Room lighting control service “Bulbs”.
2. LED lighting control service “Leds”.
3. Ventilation control service “Fans”.
4. Heating control service “Heat” (service is disabled by default).
5. Additional I2C devices control service (service is disabled by default).

Output group devices:

1. Light bulbs: controlled with impulse signals on relays or static signals on relays.
2. LED strip lights.
3. Ventilation fans (with or without connected control potentiometer) and air conditioner IR transmitter.
4. Heating control: with a connected temperature sensor “autoHeat” or with a connected control input device “manualHeat”.
5. I2C devices: additional screens and subordinate devices.

4.2. LCD Navigation Control Menu

The System Engineering Management function has the responsibility for the design of the complete system’s architecture. It develops and maintains system requirements and its internal and external interfaces [11]. That’s why the multi-level navigation menu system should be highlighted separately.

The LCD20x4 “EncMenu” is a display module board with a rotating encoder that was specifically designed for the operation of this service. The presence of the main elements of this module is a minimal requirement for launching the entire system. The navigation menu service (EncMenu) also functions with other types of LCDs on both the Arduino Mega and other platforms. Fig. 2 shows the PCB schematic of the Arduino Mega LCD2004 shield.

The main control element is the encoder, the handle of which can be rotated and pressed (it serves as a button).

Use scenarios:

1. Navigate through the menu by rotating the pressed encoder.
2. When rotating the encoder’s handle, the selection cursor (arrow) moves between menu items.
3. Pressing the encoder selects a variable for modification.
4. Multiple presses – toggling a mode (based on the number of presses).
5. Changing the selected variable by rotating the encoder.

6. Rapidly rotating the encoder changes the value with a large step.

The block diagram of the linked list of the main navigation menu is shown in Fig. 3. The initialization of groups and objects within each group, followed by queuing in the RTOS, is done in the same order as the displayed menu elements. The main advantage of the developed menu is its user-friendly navigation between groups and the ability to interact with objects within these groups by pointers, the number of which can vary. These interactions are much easier with a linked list because the structures themselves aren't manipulated, only the associated pointers [10].

The main menu contents include:

1. Main navigation screen.
2. BULB (relay) light control screen.
3. LED (PWM) control screen.
4. FAN (ventilation) monitoring and control screen.
5. Heating control screen (relay and phase control dimmer).
6. When there are no changes for a certain period, an information screen is activated, displaying the current time and values from connected sensors (temperature, pressure, humidity).
7. Settings menu (accessed through the main screen).

Each menu item contains the following information: the parameter's ordinal number, its name, and the display of the current value (with multiple display modes). Users can configure the display mode and ordinal number of each value in the Settings menu. The block diagram of the settings navigation menu is shown in Fig. 4.

Settings menu:

1. Creating new device groups and configuring their display in the menu.
2. Setting up or creating new connected devices.
3. Creating automation between devices for connected groups.
4. Configuring active services.
5. Time settings.
6. Password settings.
7. Resetting settings in EEPROM to default.

All values and parameters are stored in static EEPROM memory.

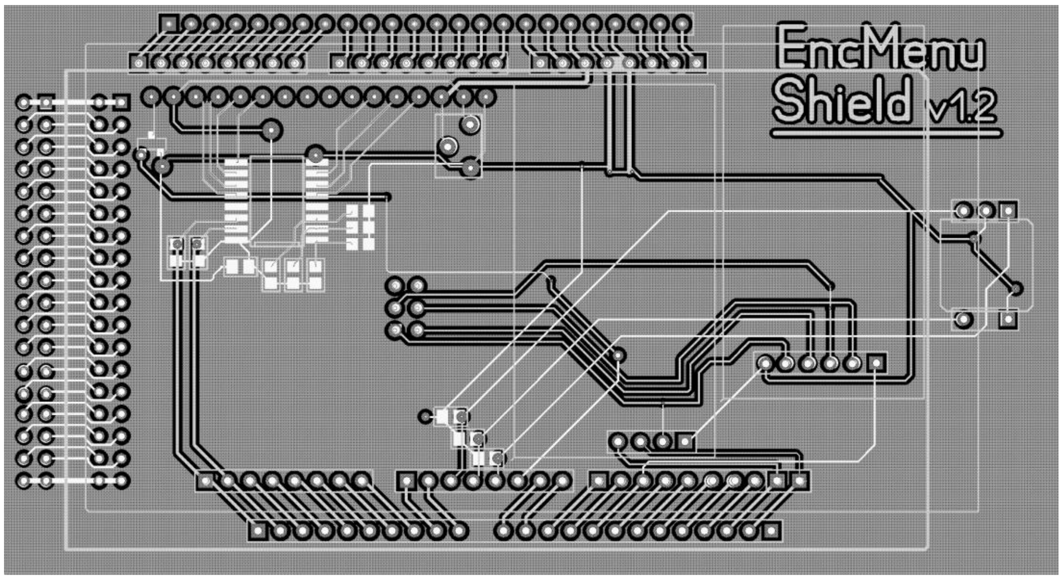

Fig. 2. Schematic of the LCD20x4"EncMenu" printed circuit board

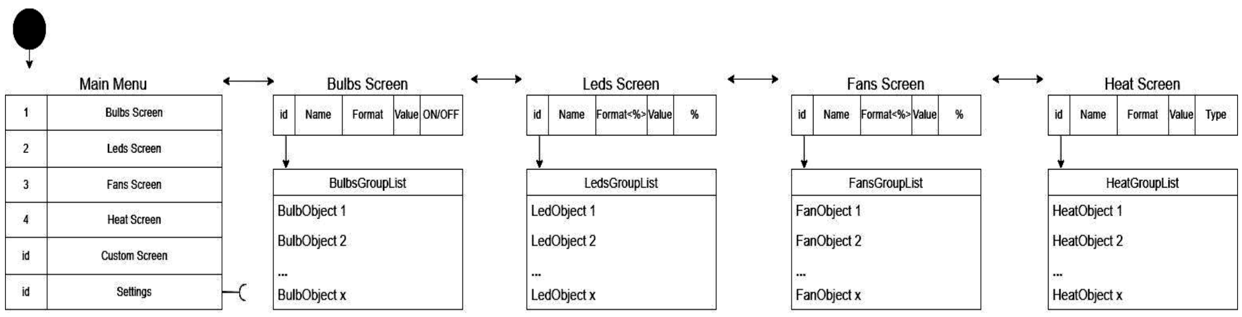

Fig. 3. The structural diagram of the main navigation menu

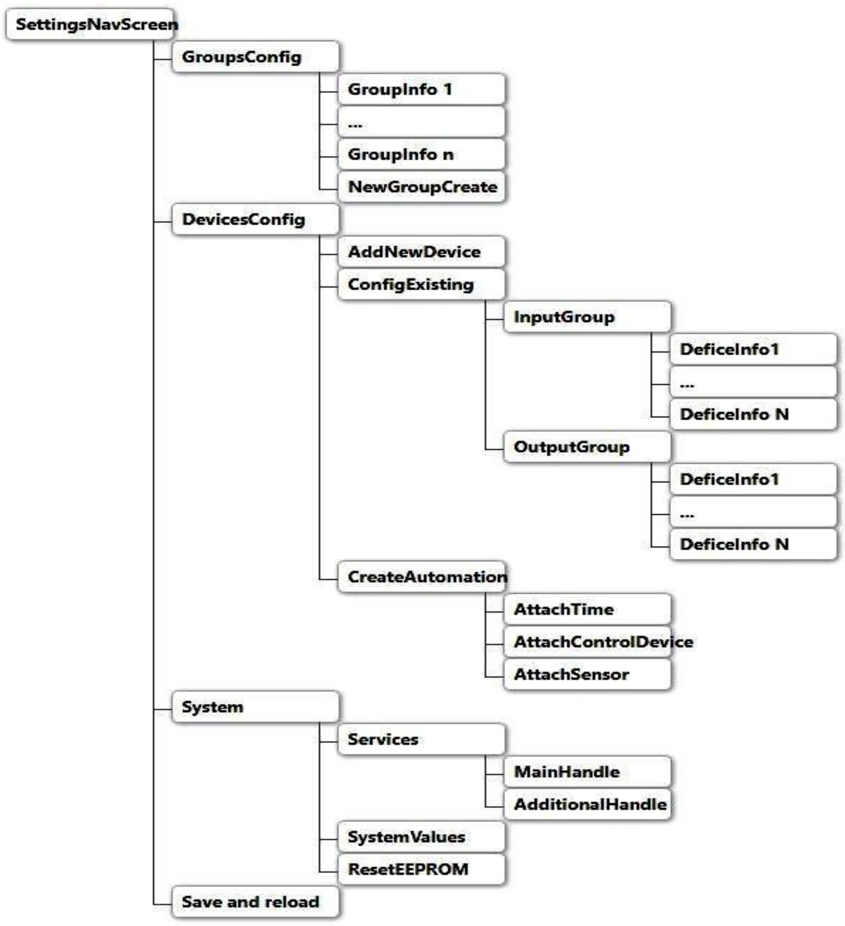

Fig. 4. The structural diagram of the settings menu

4.3. System configuration parameters and OS initialization

On the first system startup, if the SD card module is connected, the values of the configuration parameters are taken from the template files located on the SD file system. These files have the “JSON” extension. After that, the system checks for the presence of a new binary firmware that can be loaded into the system as a user service application. If the memory card is not connected, a system error can occur, and the file system service could be disabled, but the system initialization will be continued. In this case, the minimum configuration could be used to create the database.

File system hierarchy:

1. Apps:
- “app.json” – user application description;
- “app.bin” – user-compiled application.
2. Import:
- “new_group.json” – new device group configuration;
- “new_objects.json” – new connected devices configuration;
- “settings.json” – new system settings.
3. Export:
- “db.json” – current pin settings for connected devices;
- “settings.json” – a merge of files from the “Settings” directory.
4. Logs (Logs Service):
- “setup_log” – errors that occurred during system initialization;
- “system_log” – key system events;
- “event_log” – events triggered by devices from the Input group;
- “trigger_logs” – events triggered by timers or timeouts.
5. Settings:
- “time_conf.json” – time and timer settings;
- “main_config.json” – some system values (e.g., signal processing coefficients);
- “wifi_conf.json” – values to be sent via UART (only if the “iHatka” service is active).

The database is created in an electrically erasable programmable memory (EEPROM), as it is a type of non-volatile memory, and the values will be retained even in case of an emergency shutdown. Upon the next system startup, parameter values are retrieved from the EEPROM database.

Later, after the system starts, the user will have the option to access the settings menu, where the selection of included modules in the executable version of the system can be changed. However, it should be noted that choosing the correct values requires an understanding of the system's internal structure and even the impact of the parameters on the system. As an example of customization: the system can load peripheral maintenance programs depending on the situation.

After configuring the database, the process of system initialization begins. During the initialization process, the core modules of the operating system are loaded into memory. The system manager sets the values of the configuration parameters that determine how many resources of what type will be included in the system. The stages of system initialization include:

1. "StatusCode System" initialization (checking for errors at each program step).
2. Initialization of the control block: I2C LCD menu with a rotary encoder, RTC module, and SD card module.
3. If the "FileSystem" service is enabled: check for new firmware on the SD card, check for new configurations, and initialize the logging system.
4. Initialization of the database from EEPROM memory.
5. Configuration of the internal timer and threads.
6. Peripheral initialization and task creation.
7. If the "iHatka" service is enabled: initialization of the UART connection with the Wi-Fi module.
8. Real-time system operation.

4.3. Task control

Fig. 5 shows the structural diagram of real-time task management for system modules.

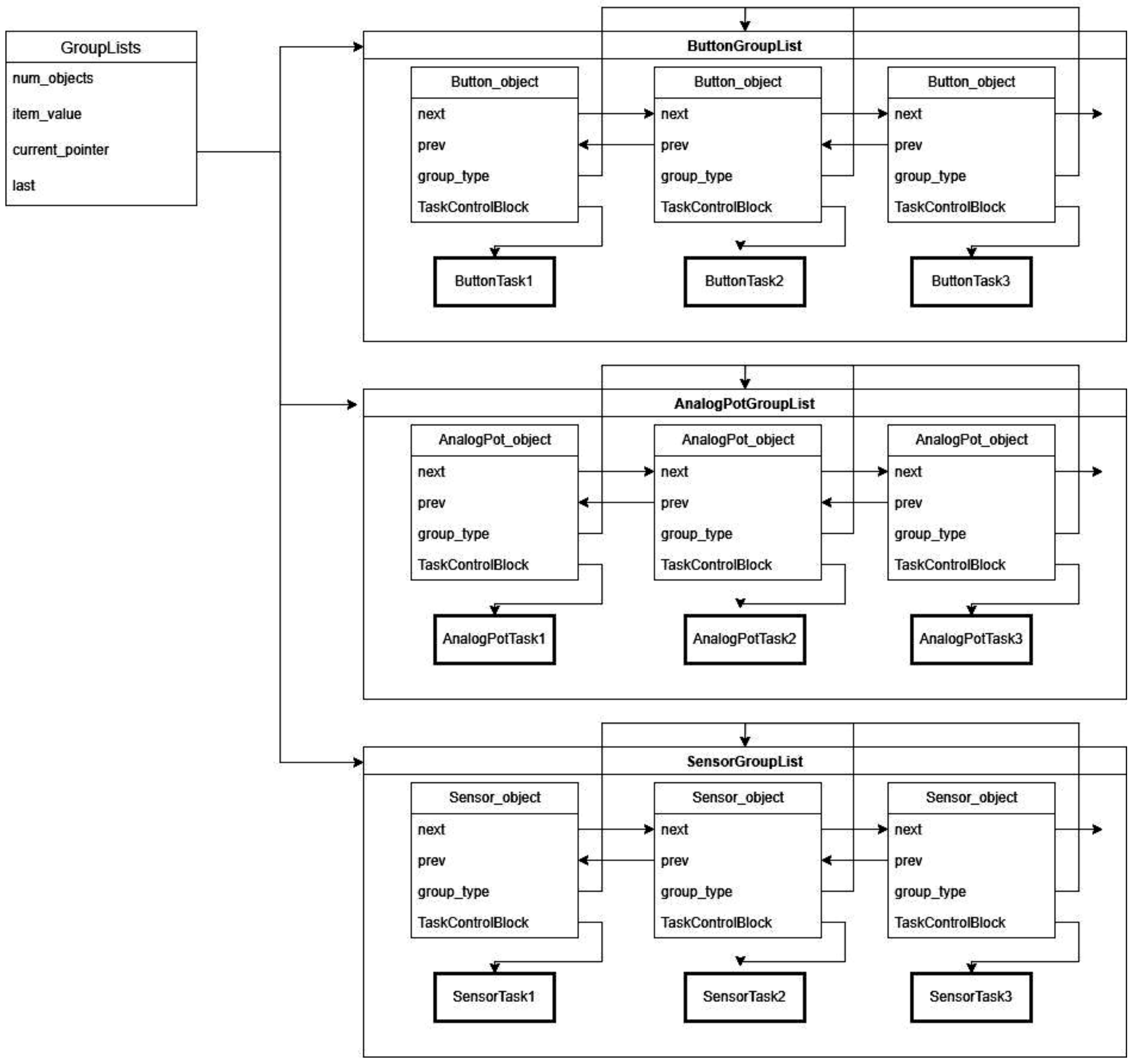

Fig. 5. Structural diagram of processing system modules

Priorities of system module operation:

1. Polling the main encoder occurs as an interrupt signal.
2. Periodic reading from communication channels: I2C (time module, temperature, and other sensors), UART (signal presence, and hence new parameters from the internet module).
3. Polling INPUT channels for changes in button/extra encoder state.
4. Polling analog INPUT for changes in potentiometer state.
5. Polling INPUT channels for changes in Keypad state.
6. Reading the signal from the IR receiver.
7. If changes are detected, a signal is sent to other threads (8–14).
8. Changing the state of the light relay OUTPUT channel.
9. Changing the state of the LED strip PWM OUTPUT signal.
10. Changing the state of the ventilation PWM OUTPUT signal.
11. Changing the state of the INPUT door relay channel when the correct password is entered.
12. Changing the state of the IR fan and sending a signal through the IR transmitter.
13. Signal for generating a JSON document [9] and sending it via UART to WemosD1 mini.
14. Signal to update the LCD menu.

The built-in programs mentioned above represent separate and independent services of the operating system. In case of a system error during the initialization of these services or while using the system, the following will occur: the error code will be registered in a separate service "StatusCodeSystem", and the respective service will be disabled until the next system reboot.

5. IoTeco-house "SmartHatka"

This work proposes the implementation of various sensors and devices connected to an Arduino Mega 2560 development board for home monitoring, as well as the use of an ESP8266 board [8] to monitor and control other devices remotely via Wi-Fi and an iPhone built-in application called "Homekit". The proposed real-time automation operating system offers a cost-effective solution that can be used and configured locally through a navigation LCD menu and rotary encoder, or used remotely via various iOS-based devices over Wi-Fi, giving users the ability to control all devices with minimal effort. Configuration settings for active services, timing, device quantity, tasks, types, connections, and other settings are indicated in the display menu and are stored in the system's memory even after emergency shutdown. This makes the system adaptable to specific requirements and suitable for integration into both eco-friendly and conventional homes.

In other words, "SmartHatka" is a group of interconnected devices managed by a central control unit (Arduino Mega with the "EncMenu Shield") operating within a "VladOS" environment, which runs a single user application, and an actively running service, "iHatka" which is hosted on a connected to the internet ESP8266. "SmartHatka" has a Software Defined Networking (SDN) architecture as an approach to network management [12].

5.1. Homekit + Wifi WemosD1mini = "iHatka"

Homekit by Apple was chosen for monitoring and control over the internet for several reasons:

1. The presence of a built-in app on Apple devices, enables users to set up an Apple tablet to control their home from anywhere in the world. This eliminates the need to purchase additional applications, servers, hosting, and IP addresses, and connect them.

2. By designing rooms, objects, and actions within the HomeKit service, users can enable automatic actions in their home, which helps ensure the periodic execution of tasks and the correctness of system concepts such as queues, semaphores, mutexes, interrupts, and, most importantly, reliable data transmission via UART and stable controller operation.

3. Built-in logging systems are available on Apple devices, allowing easy comparison with logs generated on the controller to quickly identify the cause of any issues.

4. Homekit offers a wide range of device categories and no limitations on the number of connected devices. This allows users to check system "Groups" in dynamically created objects and obtain a well-displayed representation of correctly configured groups and objects on their smartphone.

It is worth noting that the "SmartHatka" system can connect to almost all applications and servers via various IoT protocols, from MQTT to deploying a web page on ESP8266[4]. This adaptability is possible because the system itself doesn't need to know where it's sending or receiving data. It only needs to know if new data has arrived from the WiFi module via UART (requiring an update of these object values) and whether there have been changes in system objects that need to be sent to the ESP controller. This adaptability is one of the reasons why the JSON format [9] was chosen for data transmission between controllers.

5.2. Building "SmartHatka"

The following components were utilized to build the "SmartHatka" smart eco-house:

1. Arduino Mega microcontroller (based on Atmega2560).
2. LCD-20X4B.
3. I2C module pcf8574.
4. Wemos D1 mini.
5. RTC_DS3231.
6. Digital latch-based memory circuits using HEF4013.
7. 4 tactile buttons.
8. 4 analog potentiometers.
9. 4 different fans.
10. Infrared receiver.
11. Infrared transmitter.
12. Bme280 sensor.
13. SD/TF module.
14. 5 relays HK19F-DC5V-SHG.
15. 3×4 matrix keypad.
16. Rotary encoder EC12.
17. Power supply unit for the system.

The compiled system of objects includes:

1. Calling selected functions through buttons (4 related objects).
2. Lighting control: 4 pulse relays (4 objects).
3. PWM LED strips (3 objects).
4. PWM ventilation: fans and potentiometers (8 related objects).
5. Infrared receiver and transmitter for air conditioner control (2 related objects).
6. Temperature, pressure, and humidity sensor (1 object).
7. Security system with keypad-based password identification and door relay control (2 related objects).
8. Additional real-time module.
9. Logging and firmware update system via the SD card (SPI).
10. Implementation of internet control through Apple Homekit with Esp8266 via UART.

Fig. 6 and Fig. 7 depict the schematics of the main control unit and the connection of the “SmartHatka” modules, respectively.

5.3. The analysis of the effectiveness of the developed system

Table 1 presents a comparison of the processing time for basic modules connected to Arduino Mega.

Table 1. Execution time of individual modules

Module	Arduino, us	VladOS, us
Bulb	2.40	0.125
Button	2.06	0.063
Led	5.9	0.63
Fan	107.2	5.94

However, since in real-time systems, the transition between task execution occurs through interrupts, a complete analysis of efficiency should also consider the time the system spends processing interrupts. The analysis was conducted using a logical 24MHz 8-channel analyzer Saleae Logic. Fig. 8 shows the results of the analysis.

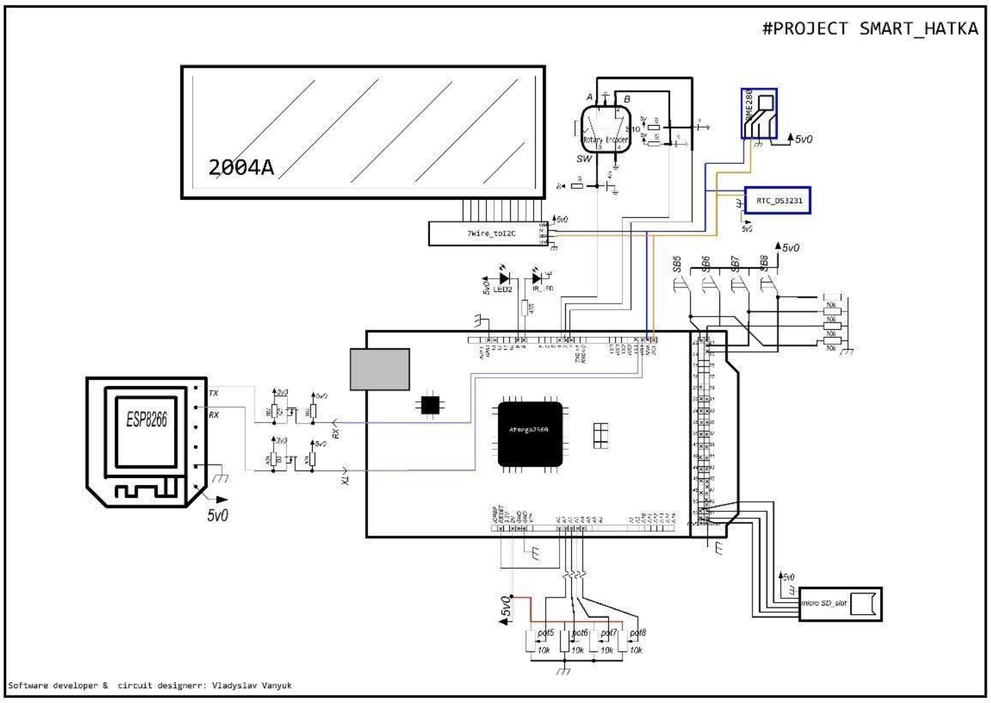

Fig. 6. The schematic of the connections to the main control unit

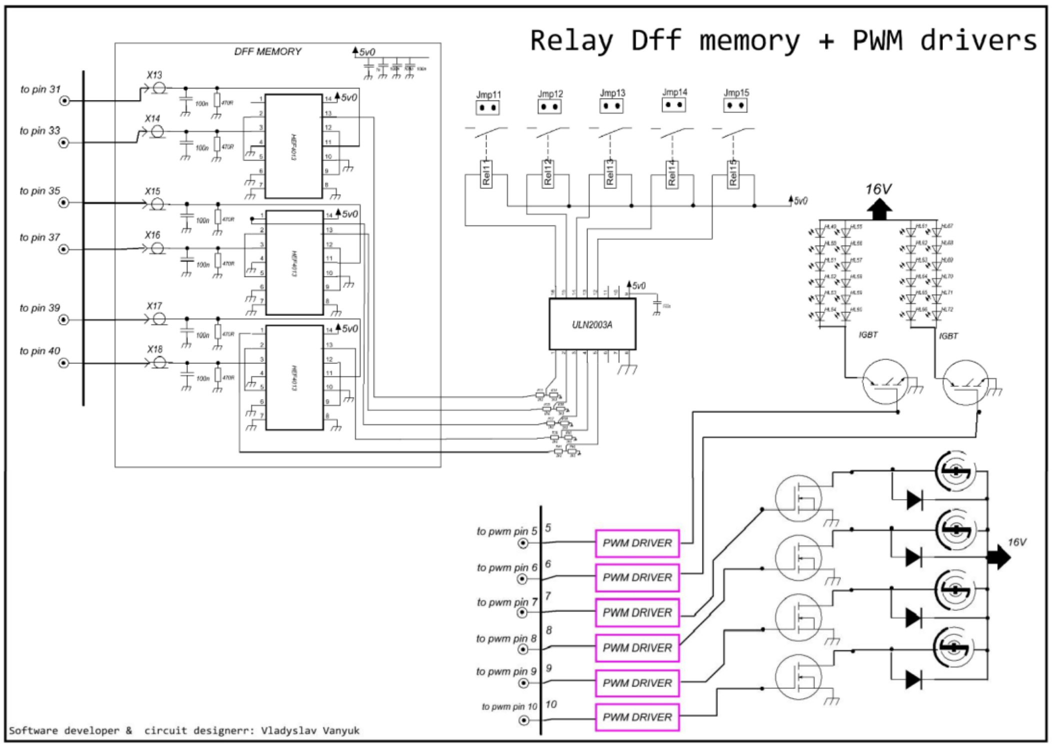

Fig. 7. The schematic of module connections

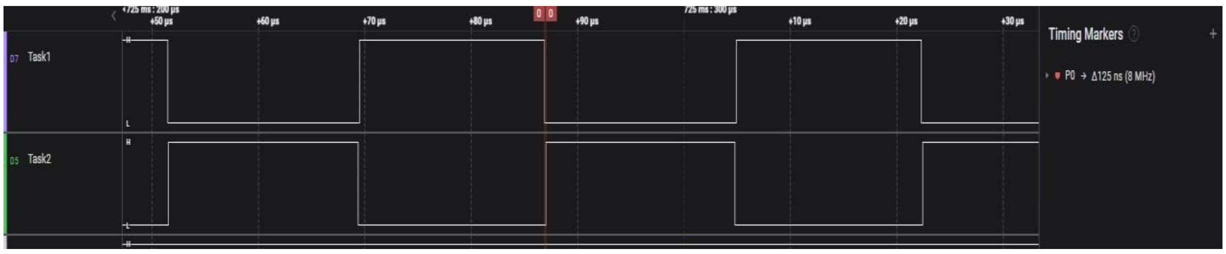

Fig. 8. Analysis of task switching time

The approximate task switching time is 0.125 microseconds. Therefore, the formula for the processing time of a compiled group of modules is as follows:

$$T = \frac{\sum\left(t_{(task)} + t_{(interrupt)}\right)}{\sum\left(t_{(task)}\right)}.$$

Table 2 shows how many times "VladOS" processes the connected modules faster.

Table 2. Comparative analysis of system efficiency

Module groups	Bulbs	Leds	Fans	Buttons
Times faster	6.98	6.2	59.5	6.27

Even with interruptions between tasks, "VladOS" is significantly faster by orders of magnitude.

The "SmartHatka" system has been implemented in an IoT eco-house as the primary control element. The versatility of the proposed solution lies in the operating system's ability to dynamically respond to changing user requirements, efficiently altering its configuration.

6. Conclusions

Through the modernization of the Arduino Mega controller, integration of peripheral modules into the real-time operating system for Atmega2560 microchips, and optimization of data processing, a cost-effective alternative to expensive PLCs named "VladOS" has been developed, which, while not matching the speed of 32-bit systems, proves significantly faster and more reliable than the widely-used Arduino-avr core, making it a viable choice for home automation applications.

7. Gratitude

The authors thank their colleagues: Ihor Nau- menko for his valuable contributions and advice, Ihor Yarish for his professional insights and guidance on sci- entific literature, and Valentin Turkovsky for providing the logic analyzer.

8. Mutual claims of authors

The authors have no claims against each other.

References

[1] A. C. A. Y. Putra, H. Wijanto and Edwar, "Design and Implementation RTOS (Real Time Operating System) as a Nano Satellite Control for Responding to Space Environmental Conditions", IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob), 2021, pp. 178- 182.DOI: 10.1109/APWiMob51111.2021.9435247.

[2] M. S. Saleh, K. G. Mohammed, Z. S. Al-Sagar, and A. Z. Sameen. "Design and Implementation of PLC-Based Monitoring and Sequence Controller System", Journal of Advanced Research in Dynamical and Control Systems, Vol. 10, 2018. Available from: https://www.researchgate.net/publication/357451701_Pengembangan_Embedded_De vice_Berbasis_PLC_untuk_Simulator_Rejection_System_dengan_Penambahan_Human_Machine_Interface [accessed Oct 24, 2023].

[3] Khan, S. (2021). Real-Time Operating System (RTOS) with Different Application: A Systematic Mapping: A Systematic Mapping. European Journal of Engineering and Technology Research, 6, 1 (Jan. 2021), 100–103. DOI: https://doi.org/10.24018/ejeng.2021.6.1.2322.

[4] Hahm S.-I., Kim J., Jeong A., Yi H., Chang S., Kishore S. N., Chauhan A., Cherian S. P. Reliable real-time operating system for IoT devices. IEEE Internet of Things Journal, Vol. 8, No. 5, 3705–3716, 2021. DOI: 10.1109/JIOT.2020. 3025612.

[5] Luna, R., Islam, S. A. Security and Reliability of Safety- Critical RTOS.SN COMPUT. SCI.2, 356. 2021. DOI: https://doi.org/10.1007/s42979-021-00753-y

[6] ATmega2560 Datasheet; ATmega640/V-1280/V-1281/V- 2560/V-2561/V; Atmel: San Jose, California, USA, 2014 [Online]. Available: https://ww1.microchip.com/downloads/ en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller- ATmega640-1280-1281-2560-2561_datasheet.pdf (accessed on 23 October 2023).

[7] Arduino Mega 2560 Rev3: Product Reference Manual; Arduino S.r.l.: Scarmagno, Italy, Year: 2023 [Online]. Available: https://docs.arduino.cc/resources/datasheets/ A000067-datasheet.pdf (accessed on 23 October 2023).

[8] ESP8266EX Datasheet; Version 7.0; Espressif Systems: Shanghai, China, 2023 [Online]. Available: https://www.espressif.com/sites/default/files/documentation/0a- esp8266ex_ datasheet_en.pdf (accessed on 23 October 2023).

[9] Benoît Blanchon, Mastering ArduinoJson: Efficient JSON serialization for embedded C++. Ebook, 2017.

[10] James M. Fiore, Embedded Controllers Using C and Arduino – 2e. Dissidents, 2018.

[11] U. S. Air Force, SMC Systems Engineering: Primer & Handbook. 2nd Edition, 77–78. Space & Missile Systems Center, 2004.

[12] Academic colleagues of the ALIOT consortium, Internet of Things for Industry and Human Applications, Vol. 2, National Aerospace University "KhAI", 2019.