Annals of the Academy of Romanian Scientists
Series on Engineering Sciences
ISSN 2066-8570 Volume **15**, Number **1**/**2023**

FIRMWARE AND HARDWARE DEVELOPMENT OF AN EDUCATIONAL PROGRAMMABLE LOGIC CONTROLLER

Alexandru-Ioan ANASTASIU,
Florea Dorel ANANIA

Rezumat. *Această lucrare descrie un nou automat programabil (AP) care integrează componente hardware și software de ultimă generație, oferind o experiență educațională imersivă și interactivă. Se detaliază elementele constructive, caracteristicile și beneficiile noului AP dezvoltat, subliniind potențialul acestuia de a oferi studenților o platformă de învățare. Este evidențiată, de asemenea, rentabilitatea deciziilor arhitecturale, scopul final fiind acela de a crea un AP care să fie atât relevant pentru standardele industriei, cât și accesibil pentru universități și școli de învățământ superior.*

Abstract. *This paper describes a novel PLC system that integrates cutting-edge hardware and software components, offering an immersive and interactive educational experience. The design, features, and benefits of the newly developed PLC are detailed, highlighting its potential to empower learners in various educational settings. Moreover, the cost-effectiveness of architectural decisions is also highlighted, with the end goal being to create e PLC that is both relevant to industry standards and affordable for universities and higher education schools.*

Keywords: PLC, Programming, Hardware & Software Development

DOI https://doi.org/10.56082/annalsarscieng.2023.1.65

1. Introduction

Programmable Logic Controllers (PLCs) have long been the cornerstone of industrial automation, driving efficiency and productivity across numerous industries. Despite their widespread use, however, they have remained largely inaccessible to educational institutions due to their cost, complexity, and limited scope of application. Recognizing the critical need for a product that is relevant to the modern-day industry, while remaining cost-effective and affordable, a set of specifications were laid out, quantifying the needs and capabilities of the PLC [1], [2].

2. Programmable logic controllers – literary review

A programmable logic controller represents a computation device that can execute a program in a closed loop, reacting to outside stimuli and being able to influence the environment it is integrated inside of. At the very basic level, a PLC should be able to:

- Be programmable: Allow the user to upload an instruction stream, which dictates the behavior of the device.
- Implement I/O functionality: Allow interaction between the device and the environment, using Input-Output loops.
- Have connectivity to other devices: Whether it be in conjunction with a personal computer, or a sensor array located at a different position, a PLC must have one or multiple dedicated interfaces, implementing respective standards.

Beyond basic requirements, one of the goals of the project was to develop a platform that stays relevant to current-age technology, with an increased accent on the Industry of Things (IoT) and other smart functionality.

According to Bolton (2009, p3), "A programmable logic controller (PLC) is a special form of microprocessor-based controller that uses programmable memory to store instructions and to implement functions such as logic, sequencing, timing, counting, and arithmetic to control machines and processes". This definition provides the first avenue to identifying the general architecture of such a device: at the core lies a microprocessor, an integrated circuit capable of a fetch-decode-execute cycle. Instructions are fed from a memory bank, generally of a non-volatile type, though battery-backed-up SRAM solutions have also been employed historically. [3], [4].

Bolton also describes PLC systems as generally being a networked mesh (2009, p5), allowing for multiple controllers to run individual tasks, while under supervision from a "master" system. It is in this case where one of the proposed innovations of this paper comes into play: deviating from conventional architecture, it was decided to create a "software-defined PLC", essentially allowing for a single board to behave as multiple state machines would, with the hypervisor functionality being built-in.

Electrically, PLCs must provide both logic functionality (allowing for Inputs and Outputs in the form of binary "ON" or "OFF" states) and power-driving functionality: sourcing and sinking (the former meaning providing power from the PLC to the outputs, essentially driving an interface to the positive voltage, the latter meaning allowing for power to flow to the ground, into the PLC). (Bolton, 2009, p11). To lower design complexity, it was decided to bypass current sinking capabilities, allowing the PLC to only source current. An external common ground rail will therefore be required. Dunning (2001, p12) notes that as an industry standard, PLCs must operate at a supply voltage of 24vDC or 220vAC, with a split-power supply: one providing power to the PLC itself and the other to the I/O modules. Due to risks associated with utilizing alternating current in an educational setting, it was decided to set the input voltage to 24v. Separation of

PLC and I/O circuits is done using a step-down voltage source for the controller, with a direct bypass of 24v to the I/O module.

To conclude the literary review, one must also bring to attention the programmability of PLCs. Numerous general-purpose programming languages exist, which allow low-level access for programmers. PLCs have, however, to be easy to program by design: an abstraction level had to be devised, and therefore a matching instruction set. Standardization was achieved with IEC 61131-3, which is "the first real endeavor to standardize programming languages for industrial automation, … independent of any single company" [5]…[9]. ABB identifies 6 different languages, however, to allow for faster development and to simplify the firmware, it was decided to implement only the textual types: Instruction List, IL, and Structured Text, ST. There are plans for a symbol-based Functional Block Diagram (FBD) implementation, however, they are still in early development.

3. General design

3.1. Controller

In choosing the main processing unit of the controller, numerous factors must be considered: raw processing power (the speed at which a microprocessor can parse and execute instructions, usually a function of processor frequency and architecture), core count (multi-core designs allow for parallelization of tasks, enabling synchronous execution of multiple threads) and memory characteristics. Memory (divided into volatile, or random-access-memory, used for data processing, and non-volatile, used for program storage) is taken as a factor into account at this stage since most often a microcontroller will ship with onboard flash and RAM.

With this in mind, a specification was drafted, featuring the following requirements:

- **Overall core performance >= 100 CM** (Core-Mark), due to the nature of handling many tasks, communication interfaces, as well as internal housekeeping.
- **Multi-core preferred**: the ability to offload tasks to a second core means higher responsiveness of a system under heavy load.
- **3.3v operating voltage**: 3.3v allows devices with a (theoretical) lower power draw. While power consumption was not a deciding factor, it is relevant since a lower power draw means lower heat production: a concern in closed environments like the inside of a casing.
- **RAM Size > 50KB**: this was determined experimentally and will be detailed in section 3.3.
- **Price < 5€**: this proved to be an ambitious yet unattainable goal. The misestimation was attributed to inexperience in the field and was a valuable learning experience.

- **I/O count of 32**: A higher I/O count allows for higher flexibility of the PLC. The targeted I/O count was chosen to create a competitive design.

3.2. Program memory (PROGMEM)

PROGMEM represents the physical location of the bytes which represent the program to be run by the PLC. This generally takes three forms: Flash memory, EEPROM, or battery-backed-up SRAM.

Flash has the highest write speeds, as well as generally very large capacities, but poses the disadvantage of lower lifetime expectancy (estimated at close to 10k write-erase cycles). EEPROM (electrically erasable read-only memory) trades speed for endurance (usually an order of magnitude higher than flash, at 100k cycles). Battery-backed up static RAM (SRAM) has the highest speeds of them all, as well as having a virtually infinite lifetime, however, necessitates the use of an external battery, to keep the data from being wiped. This becomes an issue in the case of applications designed to run for 5+ years without reprogramming.

Taking these factors into account, the solution was chosen to be an EEPROM-based PROGMEM, as especially during development there will be frequent write cycles. Size was chosen to fit an average number of IL programs, at 125KB, or 2MiB.

3.3. Interpreter and Hypervisor

As highlighted earlier, the use of ST and IL instructions requires decoding and execution. Utilizing the previous experience of the authors, a simple yet efficient interpreter was created, which operates in a dual pipeline, 2-step execution. The dual pipelines refer to the two moments when code is loaded: when writing to the EEPROM, an optimizing compiler translates textual instructions and parameters into bytecode, reducing instruction size. The second pipeline appears when loading instructions from EEPROM into the processor, to be decoded and executed.

In a regular processor, a fetch-decode-execute cycle must be implemented: instructions are fetched from memory, decoded, then executed. By employing an optimizing compiler, the decode step is bypassed: instructions are fetched into a stack, then, via switch statements, executed.

Previously, the term “software-defined PLC” was mentioned. This defines the architecture utilized by the firmware. A special operating system was deployed, called FreeRTOS. This paper shall not detail the functionality of the OS in full but will refer to rather a detail of what has been utilized, introducing the concept of Tasks.

Traditionally, microcontrollers (even multi-cored designs), are limited to running a single program loop per core. FreeRTOS allows the creation of "Tasks", which are independent program loops, which can run pseudo-simultaneously. This is achieved by taking the processor's rate and dividing it into "ticks". These are time slices, during which a certain task is active. Once the slice of a task has expired, it is halted, and execution is handed over to the next task.

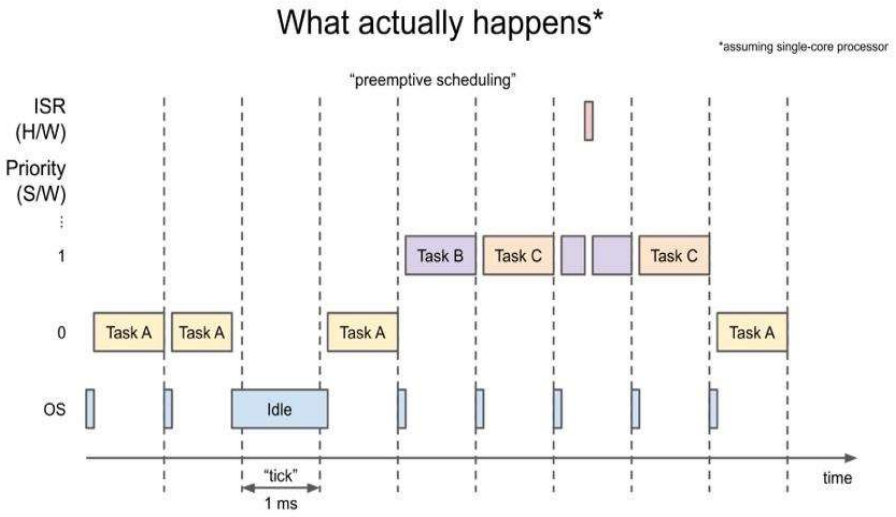

Fig. 8. FreeRTOS time slicing

By using this mechanism, a special architecture can be devised: a "hypervisor" task that runs general housekeeping tasks (memory management, serving of the web interface, communication to the outside) and the separate virtual PLC tasks, which can independently run their distinct code loops.

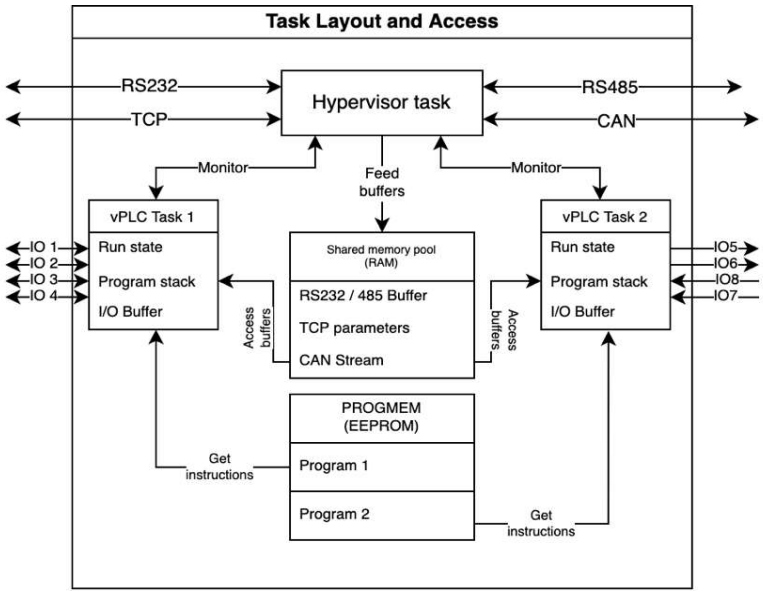

Fig. 9. Task layout inside the PLC

Fig.2 presents the advantage of this approach: A much larger number of virtual PLCs (vPLC) can be created as tasks, each with its own instruction stack, its own memory, and even selective access to pins. The hypervisor is the arbitrator, being the only one to be able to access the interfaces directly [10], [11], [12].

This workaround is done in order to avoid scheduling conflicts, where two tasks might attempt to access the same interface at the same time: a situation called a "race condition". The buffers are implemented as cyclical buffers, with a predefined wrap-around time, of which each task is aware. A flaw in this approach can be that, while interfaces are protected, the EEPROM is left unprotected. The workaround implemented in its case is to limit access to the interface through software methods (mutexes and semaphores), which trades off access speed (as each vPLC has to wait as the currently accessing one pulls an instruction out of the EEPROM) for memory size. The use of pre-fetched instruction stacks has been considered (having each vPLC read ahead all of its instructions, then proceed with execution), but was abandoned after identifying that it would be a strenuous task to manage the memory. Further developments could tackle this design oversight.

3.4. Interfaces

To finalize the specification, it was necessary to determine the connectivity of allotted to the PLC. While there is no "one size fits all" solution, it was found that, in general, industrial controllers rely on a number of interface-protocol pairs: RS232/485 Serial, to achieve either UART or PROFIBUS, Ethernet / WiFi to

achieve connectivity via TCP/IP and often CAN Bus, to achieve low-interference long-range transmissions.

The following interfaces were therefore selected, in order to best fit educational applications:

- RS232/485, in order to support the ubiquitous Serial protocols, as well as older PROFIBUS and PROFINET protocols, for backward compatibility.
- WiFi, in order to achieve compatibility with modern systems, as well as allow for the creation of a user-friendly web interface.
- CAN, in order to allow for interfacing over long ranges.

An important design consideration appeared at this stage: it would be useful to maintain a log of errors, complete with timestamps. This introduced the need to implement a Real-Time Clock module.

3.5. Hardware selection

To finalize the design process, it was necessary to do a cost analysis of the solutions to be employed, as well as a side-by-side comparison of the modules to be chosen. While this was done for every part chosen, for brevity, this section will only detail what was done for the microcontroller, from which the others can be extrapolated.

A grading system was designed, where parts were graded according to the criterions described in section 3.1:

Table 4. Device choice and specification

No.	Name	CM	Multicore?	Voltage	CAN?	RAM(KB)	Price	Wifi?
1	Arduino Due	95	No	3.3v	Yes, XCV	96	45.00 €	No
2	Teensy 3.5	265	No	3.3v	No	256	27.50 €	No
3	ESP32 DvKit v4	351	Yes, 2	3.3v	Yes, XCV	520	7.00 €	Yes
4	RPI Pico W	235	Yes, 2	3.3v	Yes, XCV	264	6.00 €	Yes
5	dsPIC33EV	234	Yes, 2	3.3v / 5v	Yes	16	0	No

After initial selection, grades were awarded from 1 to 10. Grading methodology is as follows:

- Values under specification: automatic fail, 1
- Values over specification: the maximum value is considered a 10, others are referenced to it

- Since none of the modules fit the price budget, grades were awarded by closeness to the value specified.

Table 5. Grades awarded and final selection

No.	Component n	CM	Multicore?	Voltage	CAN?	RAM	Price	Final grade
1	Arduino Due	1	1	9	9	1.85	1	3.808
2	Teensy 3.5	7.55	1	9	1	4.92	4.5	4.662
3	ESP32 Devkit	10	10	9	9	10	8.6	9.433
4	Raspberry Pi	6.7	10	9	8.5	5.08	8.8	8.013
5	dsPIC33EV C/	6.67	10	10	10	5	10	8.612

A similar process was employed for each other component, resulting in a BOM which totals close to 120€. Comparative solutions are priced at almost triple values, which proves that the design is viable economically.

4. Hardware design

With the specification locked in, the process of hardware development begins, by following the general architecture of a PLC, as described in section 1.

4.1. Inputs

An important part of PLC design involves the choice of input type: either analog (able to take discrete samples of a wave), or digital (able to measure position between a set of thresholds, obtaining a binary value). It was decided that all analog pins would be utilized on the microcontroller, allowing for all inputs to be analog (and by a simple thresholding operation, reduced to digital values). It is, however, important to clarify that this could not be established directly: input values in the industrial space range from 0 to 24v, which would instantly destroy the microcontroller, which operates at 3.3v. A special step-down configuration would be necessary, in order to measure accurately.

Traditionally, a resistor-based voltage divider would be used, however, this introduces the issue of current draw by the resistors, which can interfere with weak signals. A special circuit was devised, using a voltage follower operation amplifier, to repeat the voltage, followed by the divider. The amplifier chosen was of the RRIO type, allowing for full rail-to-rail input-output operation.

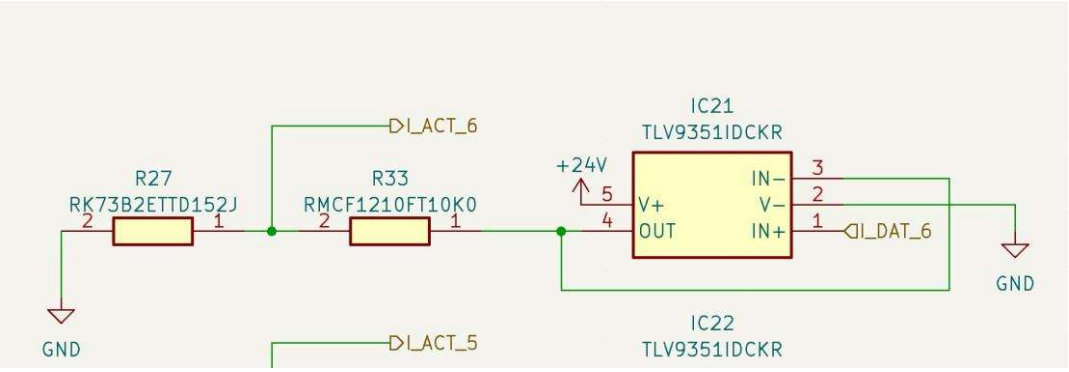

Fig. 10. Voltage follower and divider circuit

The microcontroller has a resolution of 12 bits, which equates to total of 4096 possible discrete voltage levels. By dividing the voltage range (3.3/4095), the resolution is calculated at 0.8mV / division, a resolution considered to be acceptable.

4.2. Outputs

When designing the output circuit, an important condition was imposed: the output must be galvanically isolated from the rest of the circuit, in order to not risk any kind of damage by the high voltage rail (+24v) to the (+3.3v) one. This was achieved using optoisolated solid-state relays.

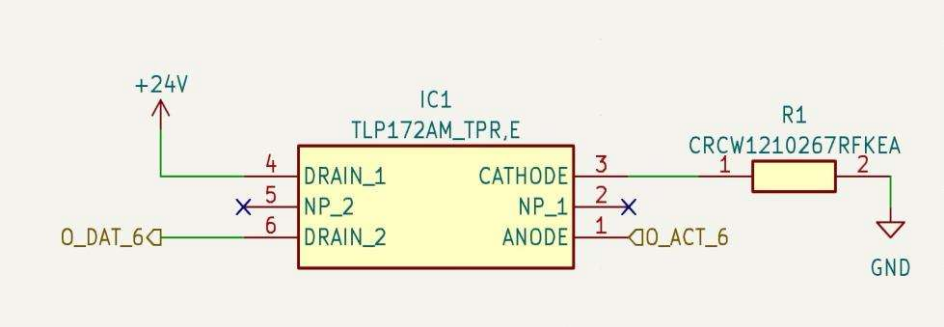

Fig. 11. Solid state relay

This design has, however, one major oversight, which was not caught until the initial prototype run: when unpowered, the output pins are left floating, neither at a logic level 1 nor at a logic level 0. This has, thus far, not caused any issues in testing, but could be a point of failure in the future.

Due to the ambitious pin count, an extra component was needed, in order to extend those available on the microcontroller: the PCA9555 is an inter-integrated-circuit (i2c) I/O extender, allowing for outputs to be switched on through digital commands from the microcontroller. The added latency associated with utilizing such a module was considered, tested, and found acceptable, in the region of 1-2ms.

4.3. General diagram

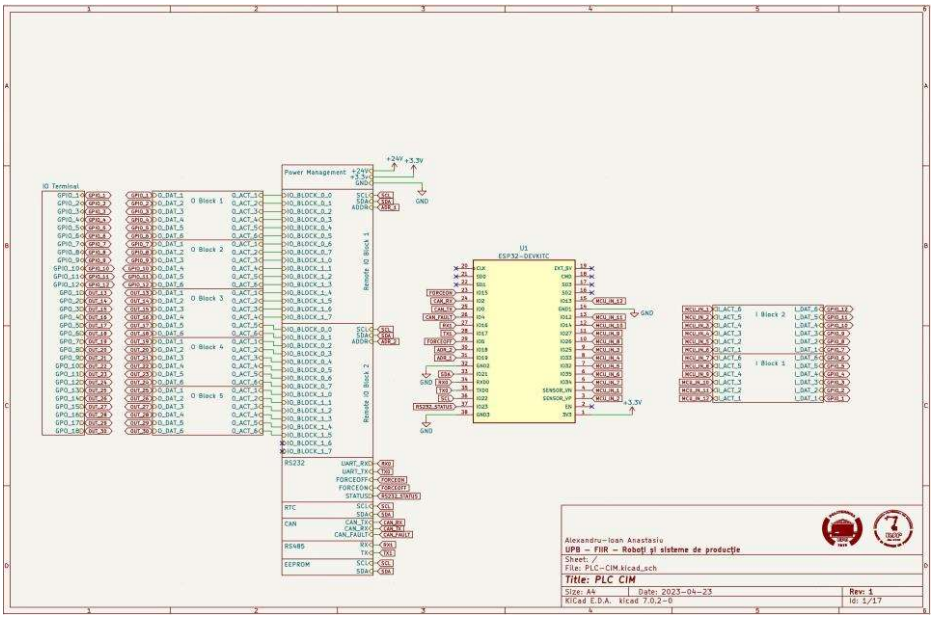

Fig. 12. The complete diagram

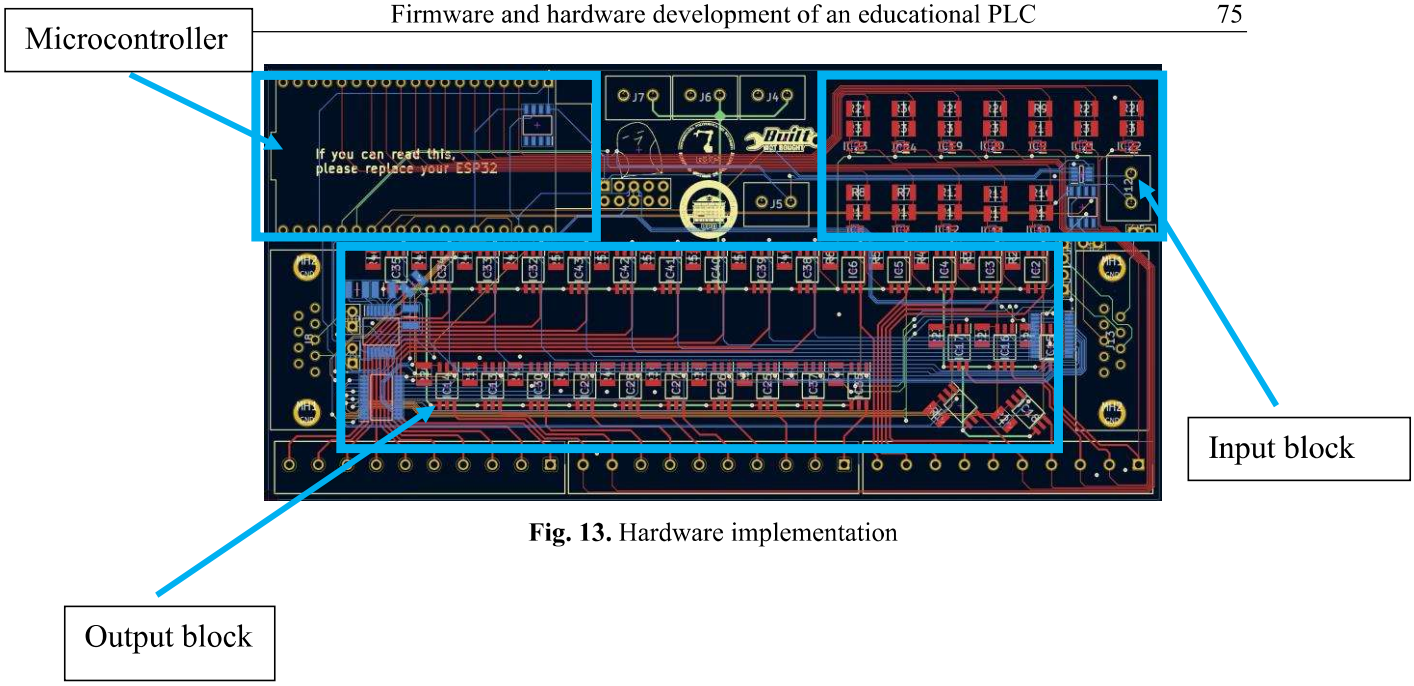

Fig. 13. Hardware implementation

5. Closing thoughts

This paper presented a novel approach to build a low-cost, educational programmable logic controller that aimed to implement as much industrial functionality as possible, while remaining an open, flexible platform. Both hardware and firmware will be made available as an open-source repository hosted online, in order to allow students as well as teachers to reproduce elements described.

The field of Programmable Logic Controllers opens up a vast array of possibilities, both in automation, robotics, and factory processes. By having a low-cost alternative to a commercially available product, it is possible to develop the necessary skills to program such devices.

REFERENCES

[1] Gary Dunning, *Introduction to Programmable Logic Controllers [2 ed.]* (2001).

[2] William Bolton, *Programmable Logic Controllers* (Elsevier, Burlington, USA, 2009)

[3] ABB, *Overview of the IEC 61131 Standard*

[4] Bruce Carter and Thomas R. Brown, *Handbook of Operational Amplifier Applications* (2001)

[5] Cozmin Cristoiu, Adrian Nicolescu*, "New approach for forward kinematics modeling of industrial robots with closed kinematic chain", 2017*

[6] Balasubramanian, Ravi. *"The Denavit Hartenberg Convention.",* USA: Robotics Institute Carnegie Mellon University (2011)

[7] Slabaugh, Gregory G. "Computing Euler angles from a rotation matrix." Retrieved on August 6.2000 (1999): 39-63.

[8] Reddy, Venu Gopal. "Neon technology introduction." ARM Corporation 4.1 (2008): 1-33

Kim, Dae-Hwan. "ARM NEON Assembly Optimization."

[9] Dobrescu, Tiberiu Gabriel & Dorin, Alexandru & Nicoleta-Elisabeta, Pascu & Ivan, Ioana. (2011). CINEMATICA ROBOTILOR INDUSTRIALI.

[10] Blanchette, Jasmin, and Mark Summerfield. *C++ GUI programming with Qt 4.* Prentice Hall Professional, 2006.

[11] Garbev, Atanas, and Atanas Atanassov. *Comparative Analysis of RoboDK and Robot Operating System for Solving Diagnostics Tasks in Off-Line Programming*. 2020 International Conference Automatics and Informatics (ICAI). IEEE, 2020.

[12] Holubek, Radovan, et al. *Offline programming of an ABB robot using imported CAD models in the RobotStudio software environment*. Applied Mechanics and Materials. Vol. 693. Trans Tech Publications Ltd, 2014.