OPENPLC HARDWARE SPEED PERFORMANCE COMPARISON

Saulius Niauronis
Šiaulių valstybinė kolegija
Lithuania

Annotation
Programmable Logic Controllers (PLCs) are critical components in industrial automation. As the technology behind PLCs continues to evolve, there is an ever-growing range of PLC hardware available in the market and their performance and functionality can vary significantly. Choosing the right PLC for a given application is thus a challenging task, requiring a thorough understanding of the available options and their capabilities. This approach involves running benchmarking tests on different PLCs to measure their performance and compare them. In this article, we compare the edge boundaries of time-based parameters of different PLC hardware, including open-source embedded controllers (Arduino and ESP8266) working with software such as OpenPLC and also Wago programmable logic controller PFC200.
Key words: *OpenPLC, Programmable Logic Controller, PLC Benchmark, Speed Test.*

1. Introduction

Programmable Logic Controllers (PLCs) have been used for decades to automate industrial processes, control machinery, and monitor system performance. The technology behind PLCs has evolved significantly over the years, resulting in a wide range of hardware options available in the market. These options include both commercial offerings and open-source solutions such as openPLC. However, the functionality and performance of these devices can vary significantly, making it challenging to choose the right one for a particular application. Benchmarking has emerged as an effective method for evaluating the performance and functionality of PLC hardware [1]. This approach involves designing and running simple test programs on different PLCs to measure their performance and compare them. Usually benchmarking studies focused on comparing the performance of PLCs from different vendors [2]. The studies evaluates the processing speed, memory usage, and I/O capabilities of the PLCs and find significant differences between the devices.

Benchmarking PLC hardware involves creating a set of tests that simulate real-world conditions and measure the performance of different PLCs under these conditions [3]. These tests can evaluate various aspects of PLC performance, including processing speed, memory usage, and input/output (I/O) capabilities. The benchmarking process can help identify the strengths and weaknesses of different PLCs, making it easier to choose the right one for a given application. To achieve accurate performance evaluation of PLCs and other control devices, benchmarking studies usually are conducted to compare different devices [4]. However this requires standardized tests which in an area of electrical engineering products, are not always possible to run because of non-standardized platforms where applications are not meant to be cross-platform compatible [5].

Benchmarking also has been used for evaluating the performance of embedded systems, which are widely used in control applications [6]. In general, performance measurement and benchmarking of industrial control systems have been recognized as important tasks [7]. Regarding IEC 61131-3 based programming, several open-source software platforms such as openPLC are available that can run on various hardware platforms such as Arduino, ESP8266, ESP32 and others [8, 9]. So usage of same standardized programming methods in a situation of comparison of performance demonstrated by both open source software (OpenPLC and related hardware) and commercial PLC solutions, can help determine which platform provides the best value.

More to that, in recent years, the DIY community has become increasingly interested in using microcontrollers like Arduino, ESP8266, and ESP32 for industrial automation applications. These microcontrollers offer a low-cost alternative to traditional PLCs and can be programmed using not only IEC 61131-3, but also other open-source software [10]. However, their performance and reliability for industrial applications are still debatable. This article aims to compare the performance of these DIY microcontrollers against commercial PLC from reputable brands such as Wago, which is selected as one of the latest PLC market new-comers in order not to compare old design systems, which still exist on the market because of long term support and proven reliability.

The task of this study is to compare open source solutions providing IEC 61131-3 compatibility to commercial PLC in terms of speed. Objectives include:

1. To prepare a set of benchmarking applications;
2. To evaluate I/O speed and cycle durations of several hardware options;
3. To compare open source solutions to each other and to commercial solution.

2. Methodology

To compare the performance of different PLC hardware, a range of devices that represented both commercial and open-source solutions were selected. For commercial PLCs, a device from Wago PFC200 family was selected (750-8214 with 8DO module 750-1515). For open-source solutions, hardware controllers, which are commonly used in DIY industrial automation projects, were selected: Arduino UNO (ATmega328 based), Arduino MEGA2560 (ATmega2560 based) and NodeMCU ESP8266. Selected hardware features are provided in table 1. OpenPLC with default options was used in tests. Since it does not support PFC200, multi-platform IDE CODESYS 3.5 (which is very popular as most PLC manufacturers currently base their newest products on CODESYS runtime) was used. All test programs were designed in FBD programming language.

Table 1.

Hardware specifications

Controller	CPU	RAM	ROM	I/O count	Comm. interfaces	Price in 1pc quantity
Arduino UNO R3 (ATmega328)	16 MHz	2 kB	32 kB + 1 kb	13 digital I/O + 6 analog	+	from 2.50€ + VAT
Arduino MEGA2560	16 MHz	8 kB	256 kB + 4 kb	54 digital I/O + 16 analog	+	from 9.00€ + VAT
NodeMCU ESP8266	80 – 160 MHz	128 kB	4 Mb	16 digial I/O + 1 analog	+ incl. wireless	from 1.50€ + VAT
Wago PFC200 (750-8214 + 750-1515)	1 GHz	512 Mb	4 Gb	8 digital outputs	+	from 1150.00€ + VAT
Wago PFC200 (750-8214 + 5 modules)	1 GHz	512 Mb	4 Gb	12 DO + 12 DI + specialized interface	+	from 1630.00€ + VAT

Program A (fig. 1) was designed to be as simple as possible. It was used to measure real output signal parameters, such as signal amplitude, I/O output pulse rise time, minimum duration (I. e. maximum speed) of one program cycle.

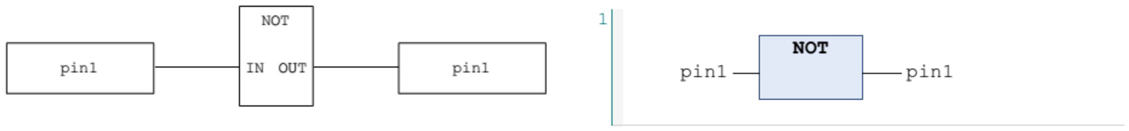

Fig. 1. Program A – output inversion each cycle. OpenPLC (left); CODESYS 3.5 (right)

Program B (fig. 2) was designed as a mean to analyze situation when controller needs more processing for background tasks (in this situation, time monitoring) with same I/O load. Program B results compared to program A provides information on if the performance is limited by I/O interface, or by CPU and memory throughput. Cycle period is measured.

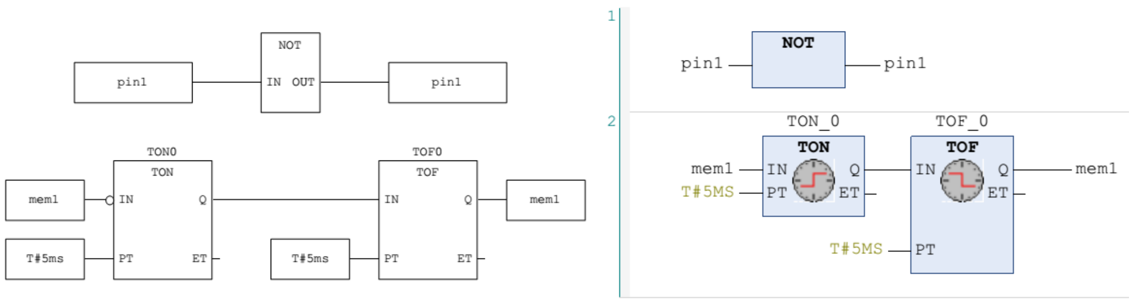

Fig. 2. Program B – output inversion each cycle with additional background task. OpenPLC (left); CODESYS 3.5 (right)

Program C (fig. 3) was designed to represent more CPU and memory consuming process, where I/O operations are rare and used only to externally measure multi-cycle duration (period).

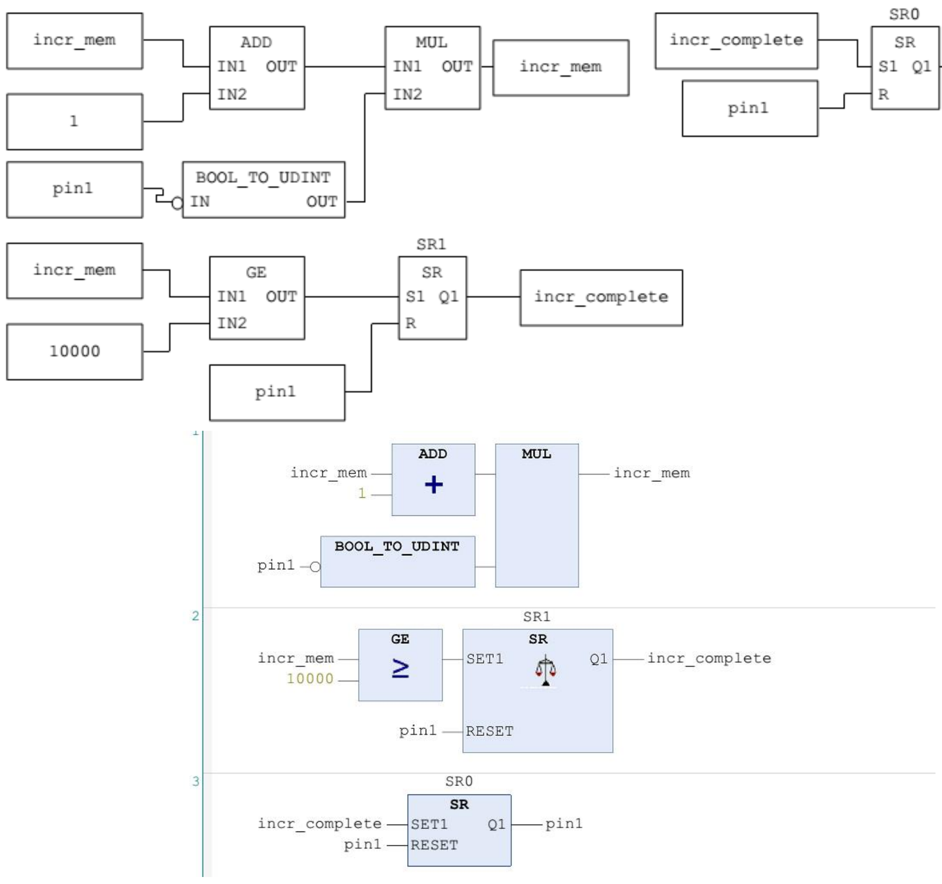

Fig. 3. Programs C & D – counting each cycle (0-10000) and single cycle pulse together with value reset to 0. OpenPLC program D (upper); CODESYS 3.5 program D (bottom). Program C is same, but GE block input is 100 instead of 10000

Program D is same as program C, but 100x longer in cycle count until reset and repeat. It was used in order to remove the delays concerning I/O and reset to 0 before repetition of multi-cycle operations. This can be done by comparing C and D results with some simple calculations.

Programs were set so, that they are repeated as fast as possible. OpenPLC has no freewheeling option for task execution, so minimum allowed interval of 1 us was set. PFC200 controller allowed 50 us minimum cycle repetition interval and also freewheeling option, so both were used. No any other task was set to run in background – only one of shown programs at a time. If to compare to real-world applications, programs are basic and small, but the idea is to investigate fastest possible performance speeds and their deterioration when CPU and memory consumption is being increased. I. e. only maximum performance and no minimum performance was analyzed.

Rigol DS1052E digital oscilloscope was used for timing measurements. 50 MHz bandwidth allows to measure digital signals with ~20 ns rise times without the distortion of significant frequency components of a signal and 1GSa/s sampling rate allows signal acquisition with 1 ns resolution.

Each measurement was repeated 10 times in order to ensure that there is no variation because of different conditions for measurement algorithm or a state of device being tested.

3. Benchmark results

Since I/O operation is hardware determined, no change in output pulse rise (or fall) durations was observed while changing CPU load. Fig. 4. Shows typical output pulses (measured using program A). Please note, that different hardware has different voltage levels (3V, 5V, 24V), so rise times should not be compared directly. In this experimental trial, main purpose of this data was to understand if this duration can be important while measuring cycle times of different platforms. As pulse rise times observed are in 15-20 ns range (which at the same time is at a limit of measurement equipment bandwidth) and periods are in tens of micro seconds, it can be assumed, that signal rise and fall times are not needed to account for in later trials. Overshoot is different and is less in systems with higher output voltage.

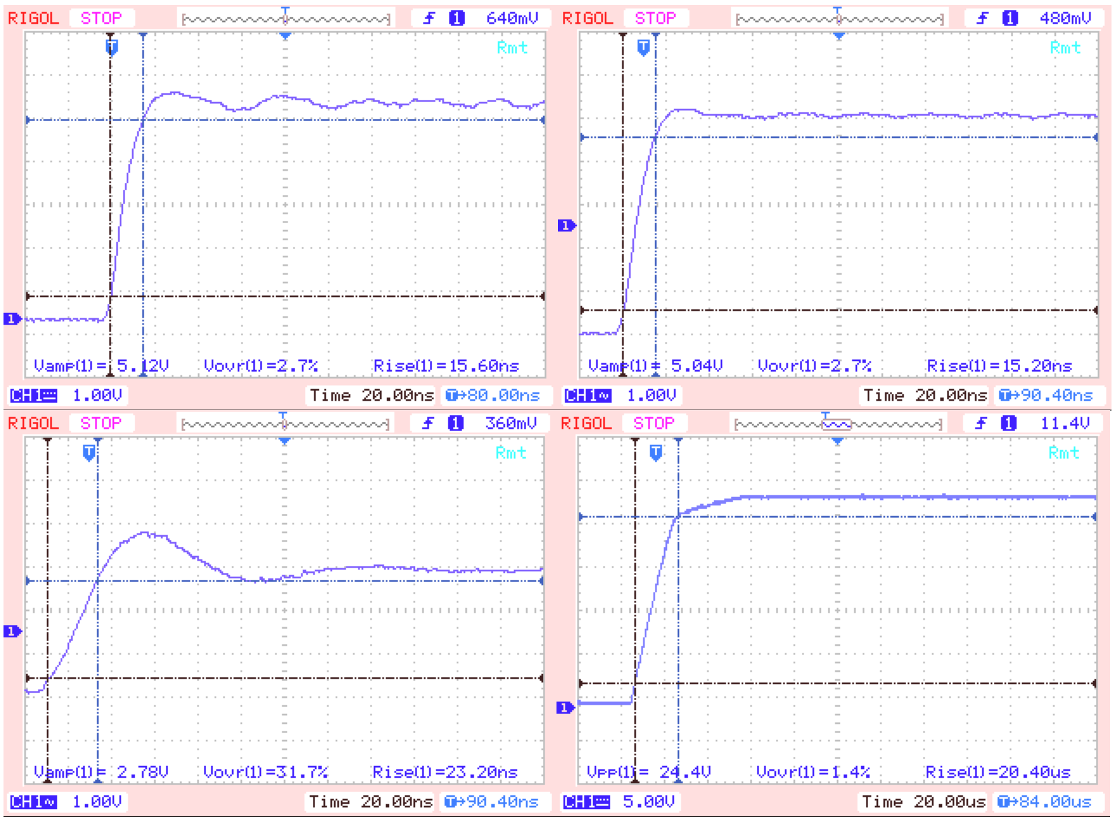

Fig. 4. Output pulse shape, amplitude and rise duration. UNO (upper left); MEGA2560 (upper right); ESP8266 (bottom left); Wago 750-1515 (bottom right) (mind different time unit)

PFC200 output pulse fall duration has capacitor discharge based waveform (Fig. 5), but it is not important as signal period values will be monitored, so the rising edge is enough to set the trigger of the oscilloscope.

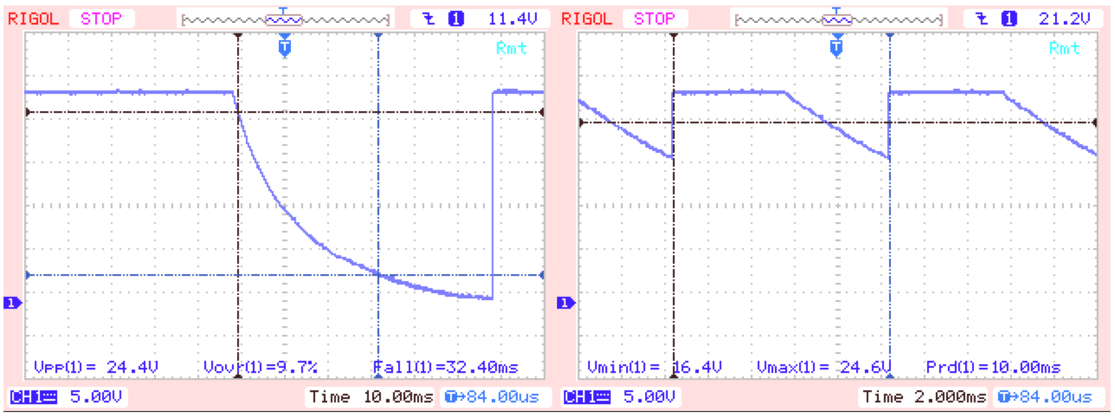

Fig. 5. Wago 750-1515 output pulse trailing edge shape (left); Its insignificance while measuring cycle period (right)

By analyzing program cycle durations of program A-D (Fig. 6), it can be seen that program A cycle time is no close to scheduled 1 us and is 17.6 us (35.2us / 2). Program B cycle time increases as additional CPU load is added and is 94.8 us (189.6 us / 2). It can be stated that processing power is the limiting factor and not I/O delays. Program C cycle time can be calculated as 6.32 ms / 100 cycles and is 63.2 us. Program D cycle time is 612 ms / 10000 = 61.2 us. It can be calculated that 99 additional (program C as compared to D) variable and output resets take 632-612 ms, so one reset is 0.202 ms (I. e. (632-612) / 99.).

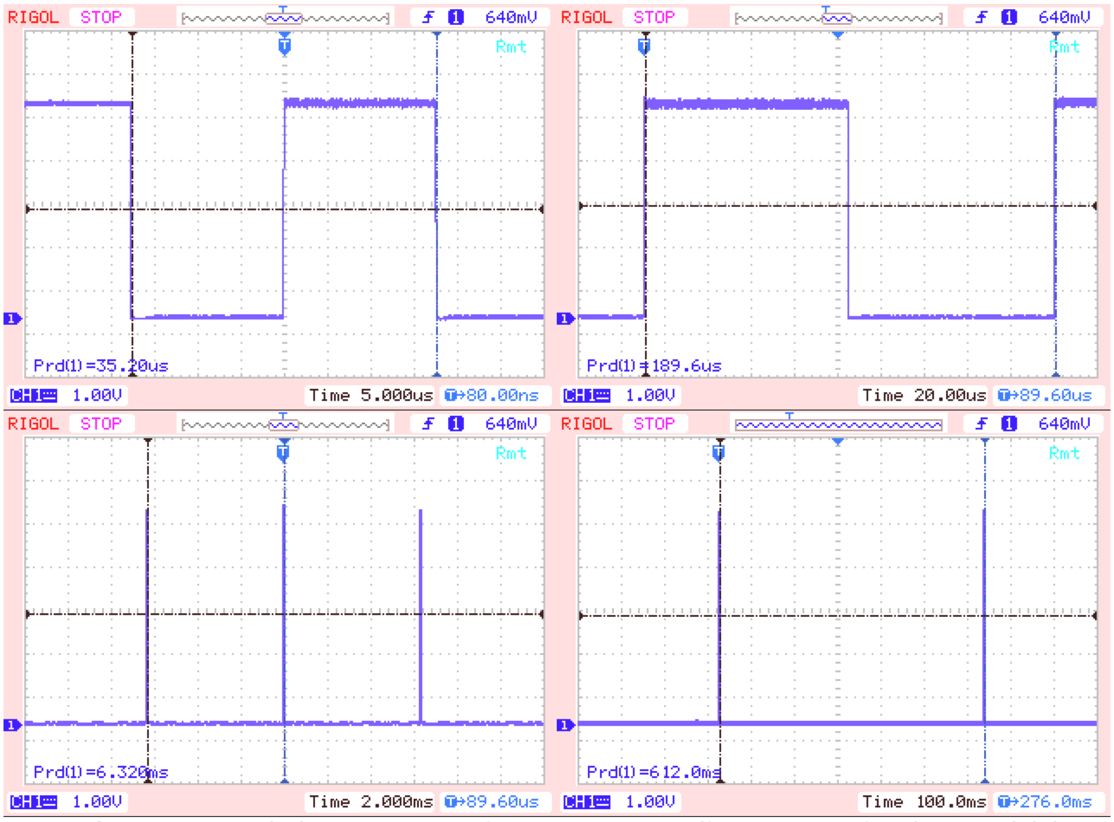

Fig. 6. UNO period measurement for programs A-D (from upper left to bottom right)

MEGA2560 and ESP8266 measurements are corresponding and they are provided in table 2. However situation while observing results for Wago PFC200 is different. This system's type is distributed I/O system, meaning that in between PFC200 controller and I/O module (750-1515 in this setup), there is a CANopen bus. Also the PLC itself is running some operating system and runtime runs only as a service (which then execute test programs). Wago's popular 750 system has no standalone PLC controllers, so these system type based differences are not possible to avoid. Most of industrial PLCs from other manufacturers also adopt such type of technology with similar topology.

All 4 programs in PFC200 PLC were run with PC connected in online monitoring state and later also with PLC running in offline (no PC connected) mode. No differences in cycle times were observed. However, cycle times were shorter in sessions, when system consisted of only PFC200 controller and 750-1515 digital output module alone. Trials were also ran with 5 modules on PLC internal bus as this allowed more close scenario to Arduino or ESP8266 comparison, where there is comparable count of inputs and outputs. All this data is provided in table 2. Program C & D durations are average.

Table 2.

Benchmark results. Single cycle durations

Controller	**Set task cycle**	**Program A (inversion)**	**Program B (inversion with additional CPU load)**	**Program C (count to 100)**	**Program D (count to 10000)**	**Program C & D (reset duration)**
UNO	1 us	17.6 us	94.8 us	63.2 us	61.2 us	202 us
MEGA2560	1 us	59.6 us	134.5 us	98.4 us	95.2 us	323 us
ESP8266	1 us	19.3 us	23.5 us	22.28 us	24 us	-
PFC200:						
	50 us	740-830 us	740-800 us	792 us	784 us	808 us
(single module on internal CANbus)	50 us	528 us	490 us	499 us	504 us	505 us
	1 ms			1008 us	1000 us	
(single module on internal CANbus)	1 ms			1012 us	1000 us	
	free-wheeling	1860 us	1880 us	1920 us	1880 us	4040 us
(single module on internal CANbus)	free-wheeling	1600 us	1580 us	1600 us	1600 us	-
	5 ms	5000 us				

4. Comparison and conclusions

Different hardware of controllers has different output drivers, so output pulse edge timing and amplitude voltage varies. AVR family controllers (ATmega328 and ATmega2560) has rapid output, with a leading edge rise time of only 15-16 ns, another embedded solution (ESP8266) has similar value of 23 ns. But commercial PLC output (Wago 750-1515) is much (approx. 1000 times) slower as most probably it has not only additional 24V circuitry, but also some additional features.

In case of embedded controllers, even with IEC61131-3 programming, their minimum program cycle times are much faster (18-60 us) when compared to Wago distributed I/O system 750 (528 us). On the other hand, even minimum CPU load increases cycle time for embedded controllers significantly (up to 5 times observed), but does not increase for commercial PLC at all. This means, that commercial PLC performance speed is even if more slow, but more stable and not so dependent on CPU load. The same can be seen while comparing embedded solution with more available CPU power (ESP8266) – cycle times are almost constant (and still 20 times faster than commercial PLC).

Wago PFC200 family controller cycle time is dependent on internal CANopen bus load. Observed latency difference was observed to be +60% (when 5 modules were in system in comparison to 1 module). In any scenario this PLC was not able to maintain minimum task cycle time of 50 us and was at least almost 10 times longer: 490 us. If task cycle time was set to 1 ms or to 5 ms, it was maintained in an outstanding accuracy. Freewheeling mode was expected to assign all resources of CPU to get as low as possible cycle times, but it appears, that in this mode, other system processes obtained most of processing power or it was by default optimized for low energy consumption instead of low cycle time. Freewheeling mode appeared to be neither fast neither very stable in terms of program cycle durations. On the other hand, it might be the right choice in order to have low cycle time and fail-free long-term operation, as problems can theoretically occur if task execution takes longer than scheduled cycle time. In neither trial, program instabilities were observed, but for production designs this approach of setting lowest possible cycle time is potentially a non-stable way of achieving speed performance as cycles are skipped, or delayed.

In overall it can be stated, that open source controllers has their own performance benefits over full-scale PLC systems. They are not only much cheaper, but allow to achieve even faster (up to 28 times as observed in this study) performance in small scale applications (up to some point which is limited by CPU resources available). On the other hand, commercial PLCs have more robust programming environments, more features and more flexibility. All this is important as projects in industry are typically non mass production, so comfortable use and support more important than a unit price.

References

1. Iqbal, S.; Khan, S. A.; Khan, Z. A., 2013. Benchmarking industrial PLC & PAC: An approach to cost effective industrial automation, International Conference on Open Source Systems and Technologies, Lahore, Pakistan, pp. 141-146, doi: 10.1109/ICOSST.2013.6720620.

2. Marasek, S.; Kulesza, Z., 2010. Universal tool for estimation of programmable logic controllers processing power, Proceedings of the 17th International Conference Mixed Design of Integrated Circuits and Systems - MIXDES 2010, Wroclaw, Poland, pp. 447-450.

3. Sunder, C.; Zoitl, A.; Rofner, H.; Strasser, T.; Brunnenkreef, J., 2007. Benchmarking of IEC 61499 runtime environments, IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007), Patras, Greece, pp. 474-481, doi: 10.1109/EFTA.2007.4416806.

4. Luyan, W.; Zhangguo, S.; Long, C., 2013. The Performance Analysis for Embedded Systems using Statistics Methods. *TELKOMNIKA Indonesian Journal of Electrical Engineering*. 11. 10.11591/telkomnika.v11i7.2864.

5. Weiss, A. R., 1999. The standardization of embedded benchmarking: pitfalls and opportunities, Proceedings 1999 IEEE International Conference on Computer Design: VLSI in Computers and Processors (Cat. No.99CB37040), Austin, TX, USA, pp. 492-508, doi: 10.1109/ICCD.1999.808586.

6. Iqbal, S. M. Z.; Liang, Y.; Grahn, H., 2010. ParMiBench - An Open-Source Benchmark for Embedded Multiprocessor Systems, *IEEE Computer Architecture Letters*, 9(2) pp. 45-48, doi: 10.1109/L-CA.2010.14.

7. Jelali, M., 2012. *Control performance management in industrial automation*: assessment, diagnosis and improvement of control loop performance.

8. Tisserant, E.; Bessard, L.; de Sousa, M., 2007. An Open Source IEC 61131-3 Integrated Development Environment, 2007 5th IEEE International Conference on Industrial Informatics, Vienna, Austria, pp. 183-187, doi: 10.1109/INDIN.2007.4384753.

9. Alves, T.; Morris, T., 2018. OpenPLC: An IEC 61,131–3 compliant open source industrial controller for cyber security research, *Computers & Security*, 78, pp. 364-379, ISSN 0167-4048, https://doi.org/10.1016/j.cose.2018.07.007.

10. Javed, M.Y.; Rizvi, S.T.H.; Saeed, M.A.; Abid, K.;Naeem, O. B.; Ahmad, A.; Shahid, K., 2015. Low cost computer numeric controller using open source software and hardware. *Sci. Int. (Lahore)*, 27(5), pp.4041-4045. ISSN 1013-5316.

Received: 7 April 2023.
Accepted: 18 April 2023.