

A Survey of IEC 61131-3 Programming Languages and Soft PLC Implementation Technologies

Zhenhao Gu *

University of Shanghai for Science and Technology, Shanghai 200093, China

* Corresponding author

Abstract: Programmable logic controllers (PLCs) are the core control devices of industrial automation systems. With the growing capability of general-purpose computing platforms and the increasing demand for openness and intelligence in industrial control software, soft PLC technology, which reproduces PLC functions in software on general-purpose platforms, has attracted sustained attention from both academia and industry. Taking the programming language system defined by IEC 61131-3 as the main thread, this paper systematically surveys the state of the art of soft PLC implementation technologies. First, the five standard programming languages, the differences among their execution models, and the latest evolution introduced by Edition 4 (2025) of the standard are reviewed. Second, following the two technical routes of compilation and interpretation, the evolution of language transformation research is summarized in terms of structural representation, scope of semantic preservation, and verification means; the program verification toolchain formed by K-ST formal semantics, ESBMC-PLC model checking, and the PLCverif platform is analyzed. Furthermore, the open-source ecosystems of Beremiz and OpenPLC, the commercial system landscape, and real-time implementation techniques are reviewed, with attention to emerging directions including the mainline inclusion of real-time Linux, virtualized deployment, integration with time-sensitive networking (TSN), and large language model assisted programming. Finally, the major technical challenges and development trends are outlined. The survey shows that semantic-preserving language transformation and real-time determinism of the runtime are the two core problems of soft PLC technology, that verification practice is shifting from test-based comparison toward formal verification, and that virtualization and intelligence are becoming key directions of the next stage of soft PLC evolution.

Keywords: IEC 61131-3; Soft PLC; Programming language; Compilation; Formal verification; Real-time system; Runtime; Industrial control software.

1. Introduction

Programmable logic controllers (PLCs) have been deployed on shop floors as dedicated hardware since their emergence in the late 1960s. Relying on deterministic scan-based execution and an industrial design targeted at harsh environments, they have earned a solid engineering reputation for reliability and real-time performance. Hardware PLCs, however, suffer from inherent limitations: dedicated processors and closed toolchains lead to high costs, cross-vendor program migration is difficult, and functional

expansion is constrained by vendor upgrade cycles. Since the beginning of the 21st century, the processing power of general-purpose computing platforms has far surpassed that of traditional PLC processors, and industrial PCs, edge servers, and embedded boards have become increasingly common in production-line environments. Against this background, the soft PLC (also known as PC-based control or software PLC), which runs PLC functions purely in software on general-purpose platforms, emerged as a practical alternative. The development history of PLC and soft PLC technologies is illustrated in Fig. 1.

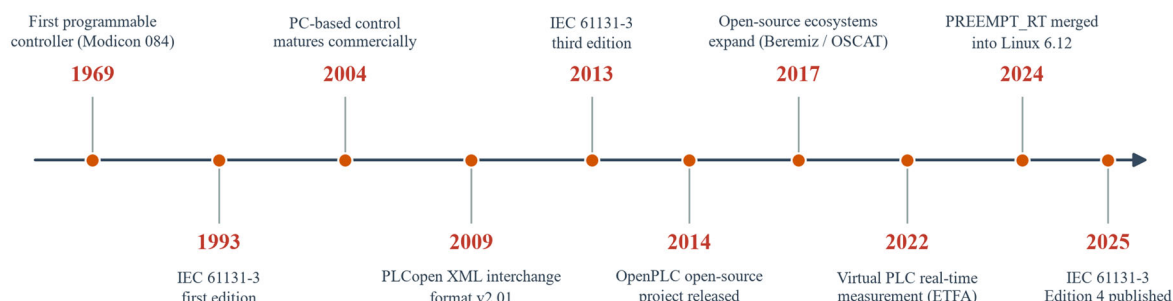


Fig. 1 Development timeline of PLC and soft PLC technologies

The essence of a soft PLC is to build a PLC runtime environment on a general-purpose computing platform: a host development system provides IEC 61131-3 compliant programming and configuration tools, while the runtime system is responsible for compiling or interpreting user programs, scheduling tasks, refreshing the I/O image, and interacting with field devices through real-time buses.

Compared with traditional hardware PLCs, soft PLCs offer three advantages. First, the hardware is open: a soft PLC can run on arbitrary x86/ARM platforms and is freed from the constraints of dedicated processors. Second, the data are open: soft PLCs are naturally suited to vertical integration with IT systems such as manufacturing execution systems (MES), digital twins, and industrial internet platforms. Third, iteration

is fast: functional upgrades are delivered as software releases and are not bound by hardware replacement cycles. In recent years, driven by the popularity of the "software-defined manufacturing" concept and the demand for independent and controllable industrial control software, soft PLC research has been warming up noticeably in both academia and industry, and new forms such as the virtual PLC have begun to enter the industrial vision [1,2].

This paper surveys representative research on soft PLC implementation technologies, with the IEC 61131-3 programming language system as the main thread. Section 2 introduces the standard programming languages and their execution models. Section 3 analyzes the key language processing techniques along the compilation and interpretation routes and provides a methodological comparison. Section 4 reviews the open-source ecosystem, commercial systems, and real-time implementation. Section 5 discusses emerging directions such as virtualized deployment, TSN integration, and large language model (LLM) assisted programming. Section 6 summarizes the challenges and trends, and Section 7 concludes the paper.

2. The IEC 61131-3 programming language system

2.1. Five programming languages and their execution models

IEC 61131-3 is the international standard for PLC programming languages issued by the International Electrotechnical Commission (IEC). Its third edition was published in 2013 [3], and the fourth edition [4], published in 2025, is the current version. Two changes in Edition 4 are the most notable: the instruction list (IL) language is formally removed, and the object-oriented extensions (classes, interfaces, and namespaces, among others) that were annexes of Edition 3 are promoted to normative content. In view of the compatibility requirements of existing engineering programs and deployed soft PLC implementations, IL is still used in industrial sites and in most runtimes, and is therefore still covered in this paper. The graphical languages established by Edition 3 and retained by Edition 4 comprise the ladder diagram (LD), the function block diagram (FBD), and the sequential function chart (SFC), while the textual languages comprise structured text (ST) and IL. Systematic textbooks have provided detailed introductions to the standard [5]. The five languages differ markedly in form, expressive strengths, and the roles they play in soft PLC implementations.

The ladder diagram originates from the representation tradition of relay control circuits. Networks are formed by contacts, coils, and function blocks arranged between the left and right power rails, intuitively expressing Boolean logic and signal flow; LD is still the most widely used language among field electrical engineers. The function block diagram connects function blocks in a data-flow style and is suitable for analog computation and continuous control. The sequential function chart describes sequential control processes that advance step by step. On the textual side, ST has a Pascal-like syntax and supports structured programming facilities such as loops, conditionals, and functions; it offers strong expressive power, facilitates compiler optimization and static analysis, and is the target language most frequently chosen by soft PLC runtimes. IL is a low-level, assembly-style language whose short statements map directly onto the

machine execution model. Its removal from the standard reflects the standard makers' orientation toward convergence on high-level languages, but the large stock of existing programs keeps it practically relevant for the foreseeable future. At the program organization level, the standard takes the program organization unit (POU) as the basic building block and organizes programs into three levels—program, function block, and function—and describes the mapping from programs to hardware through configurations and resources, providing a standard framework for the modular design of soft PLC systems.

2.2. Execution model differences and the semantic gap

The difference between the execution models of graphical and textual languages is a core problem that soft PLC implementations must address. Graphical languages execute according to network topology and scan-cycle semantics: the runtime periodically performs the "read inputs—solve logic—write outputs" cyclic scan. A ladder network implicitly carries the parallel semantics of signals propagating along the power rails, with parallel branches evaluated first and then merged; textual languages, in contrast, execute statement by statement, and their temporal behavior is explicitly determined by the order of the statements.

A natural semantic gap exists between the two. If a transformation preserves only the static logic but not the evaluation order within the scan cycle, the output timing of the transformed program will deviate from that of the original program—a risk that is particularly acute for timing-sensitive constructs such as interlocks, edge detection, and timers. Guaranteeing behavioral equivalence before and after language transformation (i.e., semantic preservation) is the common starting point of much of the research reviewed in Section 3, and it is the core yardstick for the maturity of soft PLC language processing technology.

2.3. Program interchange standards

The XML interchange format defined by the PLCopen organization [6] provides a unified carrier for program exchange between the development environments of different vendors and is the de facto standard input format for soft PLC systems to parse external programs. Parsing of PLCopen XML, construction of the object tree, and engineering data management have accordingly become common technical elements of soft PLC front-end toolchains. With respect to safety, the PLCopen Safety specification provides a standard framework for the definition and use of safety-related function blocks [7], laying the foundation for soft PLCs to carry safety applications.

3. Two technical routes of soft PLC implementation: compilation and interpretation

From the perspective of language processing, soft PLC runtimes fall into two categories: the compilation route and the interpretation route.

3.1. The compilation route and language transformation techniques

The compilation route compiles textual languages such as ST and IL (or graphical languages after conversion) into target

code of the host platform and executes it directly; a typical chain is "ST $\rightarrow$ C/C++ $\rightarrow$ native code." This route offers high execution efficiency and fully exploits general-purpose compiler optimization, at the cost of high toolchain development difficulty and strong platform dependence. Language transformation is the most intensively studied link of this route and can be classified into three categories by transformation direction.

For transformations from graphical to textual languages: Shi et al. [8] studied a new strategy for converting soft PLC ladder diagrams to instruction lists; Huang et al. [9] proposed a transformation algorithm from ladder diagrams to structured text; Zhou et al. [10] proposed a multiway-tree-based algorithm for converting ladder diagrams to instruction lists, organizing series-parallel topologies with a tree structure; and Sun et al. [11] analyzed the data dependencies of ladder

diagrams and extracted their parallelism, providing a methodological basis for preserving execution order during transformation. The common technical key of these works is to describe the ladder network with a graph or tree structure and to obtain statement sequences consistent with the scan semantics through graph traversal or tree transformation. For textual language processing: Li [12] made verification attempts at ST code generation based on a soft PLC model; Xie [13] examined the algorithmic essentials of ST-to-C conversion; and Li [14] designed and implemented an IL compiler based on IEC 61131-3, covering the complete chain from lexical analysis to target code generation.

To summarize, Table 1 compares the above methods along four dimensions: processing object, structural representation, scope of semantic preservation, and verification means.

Table 1. Comparison of language transformation and verification methods.

Reference	Object / direction	Structural representation	Scope of semantic preservation	Verification means
Shi et al. [8]	LD $\rightarrow$ IL	AOV graph + binary tree	Static logic	—
Huang et al. [9]	LD $\rightarrow$ ST	AOV digraph with virtual nodes	Static logic	—
Zhou et al. [10]	LD $\rightarrow$ IL	Multiway tree	Static logic	—
Sun et al. [11]	LD parallelism extraction	Data dependency graph	Execution order within the scan cycle	—
Xie [13]	ST $\rightarrow$ C	—	Statement semantics	—
Li [14]	Full IL compiler chain	Conventional compiler IR	Statement semantics	—
Li [12]	ST code generation	Soft PLC model	Generation correctness	Formal verification
K-ST [15]	ST formal semantics	K-framework operational semantics	Including cross-scan-cycle temporal semantics	Formal executable semantics
ESBMC-PLC [16,17]	LD / IEC 61131-3 program verification	SMT encoding	Behavioral equivalence / property assertions	Bounded model checking
PLCverif [18]	PLC program analysis	Multi-backend integration	Behavioral property assertions	Model checking (multi-backend)

Note: an em dash (—) indicates that the corresponding information is not explicitly reported in the cited publication. The structural representations in the rows of Shi et al. and Huang et al. are based on the patent literature and the published abstract, respectively.

Viewed together, these studies reveal three clear evolutionary threads. First, structural representation evolves from task-specific graph or tree structures toward rigorously defined formal operational semantics. Second, the scope of semantic preservation expands from static logic equivalence toward coverage of cross-scan-cycle temporal semantics such as edges and timers. Third, verification means evolve from "test and compare after transformation" toward "transformation with built-in verification," i.e., formal proof.

The entry of formal methods into this field deserves particular attention. Wang et al. [15] proposed K-ST, a formal executable semantics of ST based on the K framework, which gives a precise operational characterization of the language and provides a rigorous foundation for verifying ST programs and for correct compilation. Dantas et al. [16] developed ESBMC-PLC, which verifies ladder diagram programs by SMT-based bounded model checking; its successor, ESBMC-PLC+, further evolves into a unified multi-dialect IEC 61131-3 verification framework [17]. PLCverif [18], open-sourced by CERN, integrates multiple verification backends for the analysis of safety-critical PLC programs. These works indicate that the assurance of transformation correctness is moving from testing to formal verification, which is the most noteworthy progress in this direction in recent years.

3.2. The interpretation route and virtual machine technology

The interpretation route compiles user programs into intermediate representations such as bytecode, which are executed instruction by instruction by a virtual machine built into the runtime. Its advantages are platform independence, flexible loading and hot update, and consistent execution behavior across platforms; the cost is interpretation overhead and lower execution efficiency than native code. Sadolewski and Trybus [19] designed a compiler and virtual machine architecture for a multiplatform control environment, demonstrating the feasibility of supporting cross-platform control program execution with intermediate bytecode. The interpretation route is widely used in commercial soft PLC systems; its "compile once, run on many platforms" property fits the demand of industrial control software for cross-hardware deployment. Commercial systems such as CODESYS also adopt the hybrid form of compilation to intermediate code followed by interpretation or immediate execution in the runtime [20].

3.3. Runtime systems and engineering support

Regardless of the route, program organization and runtime scheduling are common engineering problems. Li et al. [21] presented the design of the runtime system software of a programmable controller, covering core mechanisms such as

task scheduling and I/O refresh; Qiao et al. [22] studied integrated soft PLC solutions for embedded scenarios, reflecting the extension of soft PLCs toward the embedded edge. Function block libraries are the carrier of the application capability of a runtime; open-source projects such as OSCAT maintain a large number of standardized function blocks—the Basic library alone contains more than 500 functions and function blocks [23]—significantly reducing application development costs. On the whole, the two routes embody a clear engineering trade-off: the compilation route achieves higher execution efficiency and fully exploits general-purpose compiler optimization, but depends strongly on the target platform and makes hot update and online debugging harder, with the highest development difficulty (language transformation plus code generation plus toolchain); the interpretation route trades execution efficiency for platform independence ("compile once, run on many platforms") and flexible online loading, at the cost of interpretation overhead and a medium development difficulty concentrated in bytecode design and the virtual machine.

4. Open-source ecosystems, commercial systems, and real-time implementation

4.1. The open-source soft PLC ecosystem

Open-source software plays an important role in the soft PLC technology ecosystem. Beremiz [24] is currently the most representative open-source soft PLC project, providing an IEC 61131-3 compliant development environment and runtime. Its language processing core is the open-source *matiec* compiler, which translates IEC 61131-3 programs into C code and generates target code with a general-purpose compiler, and which communicates with field devices over buses such as CANopen. Building industry-customized systems on open-source runtimes has also been practiced in China: Liu et al. [25] implemented safety PLC software for CNC systems based on the PLCopen Safety specification, showing that open-source runtimes can carry safety-related applications after safety extensions. The standardized function block libraries maintained by OSCAT [23] supplement the application-layer resources of the ecosystem.

Another influential route in the open-source community is OpenPLC. OpenPLC [26,27] likewise uses *matiec* as its compilation front-end, compiles ST/LD programs into C code for execution, and provides a complete open-source stack from the development environment (OpenPLC Editor) to the runtime. Owing to its low deployment threshold and open hardware/software interfaces, it has become a widely used experimental platform in industrial control system security research, and a large body of research on PLC vulnerability discovery, intrusion detection, and security hardening takes it as the target environment [27], which has objectively and significantly expanded the academic influence of soft PLC technology. In terms of verification tools, in addition to PLCverif [18] and ESBMC-PLC [16,17] mentioned above, the open-source community has contributed analysis tools covering different language levels, forming a complete open-source chain from development and operation to verification.

Open-source solutions have significantly lowered the entry barrier of soft PLC development, but a clear gap remains compared with commercial systems in terms of engineering maturity, functional safety certification, and long-term maintenance. The open-source camp—Beremiz and

OpenPLC on the runtime side, OSCAT on the function block library side, and PLCverif on the verification side—together covers the complete toolchain, yet none of the runtime projects has completed functional safety certification and their function block libraries remain small; the commercial camp, represented by CODESYS and TwinCAT/BSD, offers mature toolchains and large installed bases, but closed licensing and hardware binding restrict secondary development; domestic vendor solutions, mostly compiled or interpreted and adapted to domestic platforms, are still building their ecosystems [21,22].

4.2. Real-time implementation techniques

Real-time capability is a key issue in the engineering application of soft PLCs. The default scheduling policies of general-purpose operating systems cannot guarantee scan-cycle determinism, so real-time extension technologies are required. Mainstream approaches include Linux with the PREEMPT_RT real-time patch, dual-kernel solutions such as Xenomai/RTAI, and commercial real-time operating systems. Notably, after two decades of long-term evolution, PREEMPT_RT was merged into the mainline Linux 6.12 kernel in November 2024 [29], marking the entry of real-time Linux into the "native support" stage—a change with far-reaching significance for the deployment and maintenance of soft PLCs. Perez et al. [30] systematically measured the real-time performance of virtual PLCs; their experiments show that virtual PLCs running on Linux with real-time patches can achieve deterministic responses below the millisecond level, providing empirical support for the feasibility of soft PLCs in real industrial scenarios. The deep coordination of real-time Ethernet buses (such as EtherCAT) with soft PLC runtimes further pushes soft PLCs into high-real-time scenarios such as motion control [28].

Overall, "real-time Linux kernel + standard real-time bus + open runtime" has become a mature paradigm for soft PLC engineering implementation, and the entry threshold for real-time capability is being lowered significantly.

5. Emerging directions: virtualization, TSN, and intelligence

Virtualization and cloud-edge deployment. Container technology is changing the delivery form of soft PLCs. Gaffurini et al. [1] deployed virtual PLCs in containers on industrial edge platforms and systematically evaluated their monitoring and communication performance; the results show that virtual PLCs can meet the performance requirements of supervisory control tasks in edge scenarios. Müller et al. [2] conducted a systematic survey of virtualization in industrial production, outlining the benefits and risks of virtual and virtualized industrial controls compared with traditional deployments in terms of flexibility, scalability, and operation and maintenance cost. Industry analysis also predicts that virtual PLCs will form a considerable market by 2030 [31]. Virtualization completely decouples control programs from hardware, providing new possibilities for rapid replication, gray-scale upgrade, and disaster-recovery switching of production line controllers, but the real-time boundaries and security isolation mechanisms involved still need in-depth study.

Integration with TSN and 5G networks. Time-sensitive networking (TSN) provides deterministic transmission capabilities for Ethernet through the IEEE 802.1 series of

standards and is being deeply integrated with soft PLC architectures; the white paper released by 5G-ACIA systematically discusses typical scenarios of 5G and TSN integration for industrial control [32]. For soft PLCs, TSN/5G means that the communication between the runtime and remote I/O, or even cloud-side controllers, can have bounded latency, further expanding the deployment radius of soft PLCs.

LLM assisted programming. Large language models have begun to enter PLC programming. Liu et al. [33] proposed Agents4PLC, which automatically generates ST code with a multi-agent framework and embeds formal verification into the generation loop to form a closed loop. AutoPLC [34] generates ST code conforming to vendor-specific code styles, targeting the style differences across vendors. Such work is still in its infancy, and the correctness and safety of the generated code remain open problems, but the paradigm shift it points to—from natural language to control logic—deserves continuous attention from soft PLC toolchain researchers.

6. Challenges and prospects

Although soft PLC technology has completed the leap from proof of concept to engineering practice, several issues deserve continued attention.

First, the degree of formalization of semantic preservation is insufficient. Existing research on the transformation from graphical to textual languages mostly targets structural equivalence and static logic equivalence, and lacks systematic verification methods for the strict preservation of cross-scan-cycle temporal semantics. The emergence of formal semantics such as K-ST [15] and of the model checking tools of the ESBMC-PLC series [16,17] provides a new technical basis; deeply embedding formal verification into the language transformation toolchain to achieve "transformation with built-in verification" is an important direction for improving the trustworthiness of soft PLCs.

Second, the engineering threshold of functional safety certification is high. Soft PLCs already have solid empirical evidence of millisecond-level deterministic response on general-purpose platforms, but completing systematic certification under functional safety standards (such as IEC 61508) still requires the overall construction of the runtime kernel, the compilation toolchain, and online diagnostic mechanisms. How open-source runtimes can complete the safety evidence chain is the key to whether they can enter high-end production lines.

Third, virtualization brings new problems. Containerized deployment introduces new topics such as resource contention, real-time boundary characterization, and security isolation; the responsibility boundary between virtual PLCs and traditional control systems, configuration migration, and the operation and maintenance system still await industry consensus.

Fourth, ecosystem construction and domestic adaptation remain long-term tasks. The competitiveness of commercial systems lies not only in the runtime itself, but also in the complete ecosystem composed of function block libraries, device drivers, and industry templates. The fusion of the open-source and commercial ecosystems and adaptation to domestic chips and operating systems will provide latecomers with differentiated development paths, while new tool forms such as LLM assisted programming may reshape the development toolchain landscape of control software. At

the standard level, the removal of IL and the promotion of object-oriented capabilities in Edition 4 will continue to drive the renewal of language processing toolchains.

7. Conclusion

With the IEC 61131-3 programming languages as the main thread, this paper has surveyed the research progress of soft PLC implementation technologies in language transformation and formal verification, compilers and virtual machines, open-source ecosystems and commercial systems, and real-time and networked implementation. Overall, rigorous verification of language semantic preservation and functional safety capability are the core technical issues at present; verification evolving from testing to formal methods, deployment evolving from dedicated hardware to virtualization, and development evolving from manual configuration to intelligent assistance are three clearly visible mainlines of progress. With real-time Linux entering the mainline kernel, the gradual popularization of TSN/5G deterministic networks, and the deepening demand for openness and autonomy in industrial control software, soft PLC technology is expected to undertake core control functions in a wider range of industrial scenarios.

References

- [1] Gaffurini, M., Bellagente, P., Depari, A., Ferrari, P., Flammini, A., & Sisinni, E. (2024). Virtual PLC in industrial edge platform: Performance evaluation of supervision and control communication. *IEEE Transactions on Instrumentation and Measurement*, 73, 1-10. <https://doi.org/10.1109/TIM.2024.3370746>
- [2] Müller, J., Giani, M., Deubert, D., & Sauer, A. (2024). Virtualization in industrial production: A survey focusing on virtual and virtualized industrial controls. In *2024 IEEE 29th International Conference on Emerging Technologies and Factory Automation (ETFA)* (pp. 1-7). IEEE. <https://doi.org/10.1109/ETFA61755.2024.10710668>
- [3] International Electrotechnical Commission. (2013). *Programmable controllers – Part 3: Programming languages (IEC 61131-3:2013, Edition 3.0)*. IEC.
- [4] International Electrotechnical Commission. (2025). *Programmable controllers – Part 3: Programming languages (IEC 61131-3:2025, Edition 4.0)*. IEC.
- [5] Peng, Y., & He, Y. Q. (2009). *IEC 61131-3 programming languages and application fundamentals [In Chinese]*. China Machine Press.
- [6] PLCopen. (2009). *XML interchange format for IEC 61131-3 (Version 2.01)*. PLCopen Technical Committee 6.
- [7] PLCopen. (n.d.). *PLCopen Safety: Safety software specification*. <https://plcopen.org>
- [8] Shi, R., Zhou, L., & Yang, Z. Y. (2010). Research on a new conversion strategy from soft PLC ladder diagram to instruction list [In Chinese]. *Computer Engineering and Applications*, 46(18), 244-248.
- [9] Huang, L. W., Liu, W., & Liu, Z. Q. (2009). Algorithm of transformation from PLC ladder diagram to structured text. In *2009 9th International Conference on Electronic Measurement & Instruments (ICEMI)* (pp. 778-782). IEEE.
- [10] Zhou, W. Q., He, T. N., & Chen, D. F. (2017). Ladder diagram to instruction list conversion algorithm based on multiway tree [In Chinese]. *Computer Measurement and Control*, 25(8), 191-194.

- [11] Sun, D. Q., Yan, Y., & Wu, H. F. (2014). Data dependency analysis and parallel extraction of ladder diagram [In Chinese]. *Computer Engineering*, 40(2), 67-70, 76.
- [12] Li, Z. C. (2021). ST language code generation and verification based on a soft PLC model [Master's thesis, East China Normal University] [In Chinese].
- [13] Xie, L. N. (2024). Research on the conversion algorithm from ST language to C language [In Chinese]. *Computer Programming Skills and Maintenance*, (4), 35-37.
- [14] Li, H. L. (2014). Design and implementation of an IL language compiler based on the IEC 61131-3 standard [Master's thesis, Xidian University] [In Chinese].
- [15] Wang, K., Wang, J., Poskitt, C. M., Chen, L., & Xu, Z. (2023). K-ST: A formal executable semantics of the structured text language for PLCs. *IEEE Transactions on Software Engineering*, 49(10), 4796-4813. <https://doi.org/10.1109/TSE.2023.3300762>
- [16] Dantas, P., Cordeiro, L., & Junior, W. (2026). ESBMC-PLC: Formal verification of IEC 61131-3 ladder diagram programs using SMT-based model checking. *arXiv*, arXiv:2606.15461. <https://arxiv.org/abs/2606.15461>
- [17] Dantas, P., Cordeiro, L., & Junior, W. (2026). ESBMC-PLC+: A unified IEC 61131-3 formal verification framework as a PLCverif successor. *arXiv*, arXiv:2606.23870. <https://arxiv.org/abs/2606.23870>
- [18] CERN. (n.d.). PLCverif: Open-source framework for the verification of PLC programs. <https://github.com/cern-logs/PLCverif>
- [19] Sadolewski, J., & Trybus, B. (2022). Compiler and virtual machine of a multiplatform control environment. *Bulletin of the Polish Academy of Sciences: Technical Sciences*, 70(2), Article e140554. <https://doi.org/10.24425/bpasts.2022.140554>
- [20] CODESYS. (n.d.). CODESYS: IEC 61131-3 development environment [Product documentation]. <https://www.codesys.com>
- [21] Li, X., Chen, H. J., Zeng, K., Wang, Y., & Liu, Z. (2023). Design of runtime system software for programmable logic controllers [In Chinese]. *Application of Microcontrollers and Embedded Systems*, 23(7), 41-43, 75.
- [22] Qiao, Z. G., Bu, L. W., Ye, Y., Zhang, H., & Li, W. (2025). Research and application of integrated embedded soft PLC [In Chinese]. *Automation Application*, 66(11).
- [23] OSCAT. (n.d.). Open source community for automation technology. <https://www.oscat.de>
- [24] Beremiz. (n.d.). Beremiz: Open source PLC IDE. <https://github.com/beremiz/beremiz>
- [25] Liu, W. Y., Zhu, Z. H., Lyu, X. F., Wang, H., & Chen, L. (2022). Safety PLC software system for CNC systems based on PLCopen Safety [In Chinese]. *Manufacturing Technology and Machine Tool*, (4), 114-117.
- [26] Alves, T. R., Buratto, M., Souza, F. M., & Rodrigues, T. V. (2014). OpenPLC: An open source alternative to automation. In 2014 IEEE Global Humanitarian Technology Conference (GHTC) (pp. 585-589). IEEE. <https://doi.org/10.1109/GHTC.2014.6970342>
- [27] Alves, T., & Morris, T. (2018). OpenPLC: An IEC 61131-3 compliant open source industrial controller for cyber security research. *Computers & Security*, 78, 364-379. <https://doi.org/10.1016/j.cose.2018.07.007>
- [28] Beckhoff Automation. (n.d.). TwinCAT/BSD: PC-based control platform [Product documentation]. <https://www.beckhoff.com>
- [29] Larabel, M. (2024, September 20). Real-time "PREEMPT_RT" support merged for Linux 6.12. Phoronix. <https://www.phoronix.com/news/Linux-6.12-PREEMPT-RT>
- [30] Perez, D. J., Walzl, J., Prenzel, L., & Kaiser, J. (2022). How real (time) are virtual PLCs? In 2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA) (pp. 1-8). IEEE.
- [31] IoT Analytics. (2024, December 11). Virtual PLC and soft PLC market report 2024–2030. <https://iot-analytics.com/virtual-plcs>
- [32] 5G-ACIA. (2021, May 4). Integration of 5G with time-sensitive networking for industrial communication [White paper]. <https://5g-acia.org>
- [33] Liu, Z., Zeng, R., Wang, D., Zhang, Y., & Li, J. (2026). Agents4PLC: Automating closed-loop PLC code generation and verification in industrial control systems using LLM-based agents. *IEEE Transactions on Software Engineering*, 52(5), 1672-1687. <https://doi.org/10.1109/TSE.2026.3667895>
- [34] Yang, D., Wu, A., Zhang, T., & Li, H. (2024). AutoPLC: Generating vendor-aware structured text for programmable logic controllers. *arXiv*, arXiv:2412.02410. <https://arxiv.org/abs/2412.02410>