Emilian ŚWITALSKI, Krzysztof GÓRECKI

Uniwersytet Morski w Gdyni, Wydział Elektryczny, Katedra Elektroniki Morskiej
ORCID: 1. 0000-0002-5437-381X 2. 0000-0002-9857-8235

doi:10.15199/48.2024.10.61

Otwarte i warstwowe projektowanie sterowników PLC na przykładzie sterownika do zastosowań edukacyjnych

Streszczenie: *W artykule porównano najpopularniejsze, zamknięte sterowniki PLC z koncepcją sterowników otwartych, projektowanych warstwowo, których liczba na rynku stale rośnie. Przedstawiono wady i zalety sterowników opartych zarówno na mikrokomputerach, jak i mikrokontrolerach, podkreślając, że obie technologie mają swoje miejsce na rynku. Kluczową częścią pracy jest prezentacja otwartego, warstwowo zaprojektowanego sterownika OpenCPLC opartego na mikrokontrolerze wraz z podkreśleniem jego innowacyjnych cech, zwłaszcza prostą implementację wsparcia poprzez modele językowe AI.*

Abstract: *The article compares the most popular, closed PLC controllers with the concept of layered open controllers, a trend that is gaining momentum. It highlights the strengths and weaknesses of controllers based on both microcomputers and microcontrollers, emphasizing that there is room in the market for both solutions. The core part of the paper is the presentation of the OpenCPLC controller, which is designed in a layered, open manner and based on a microcontroller, with an emphasis on its innovative features, particularly the straightforward implementation of support through AI language models. (**Open and Layered PLC Design as Exemplified by an Educational PLC**)*

Słowa kluczowe: *Sterownik PLC, projektowanie warstwowe, mikrokontroler, IDE, VSCode, Raspberry Pi, Arduino, STM32, OpenCPLC, VRTS, model językowy, AI*
Keywords: *PLC controller, layered design, microcontroller, IDE, VSCode, Raspberry Pi, Arduino, OpenCPLC, STM32, VRTS, Llanguage model, AI*

Wstęp

Sterowniki PLC *(ang. Programable Logic Controllers)* są integralną częścią przemysłu, ponieważ umożliwiają kontrolę i zarządzanie złożonymi procesami produkcyjnymi w czasie rzeczywistym i istotnym elementem systemów wbudowanych [1]. Przy wyborze odpowiedniego sterownika, oprócz wymagań projektowych, należy wziąć pod uwagę jego skalowalność, łatwość programowania, niezawodność, bezpieczeństwo, wsparcie techniczne, warunki eksploatacji oraz koszt. Programiści często preferują sterowniki, które już znają, o ile nie są zobowiązani do użycia konkretnego rozwiązania. Temat ten jest bardziej złożony, niż mogłoby się wydawać, i obejmuje nie tylko aspekty techniczne. Dlatego celem pracy jest analiza dostępnych rozwiązań w kontekście projektowania warstwowego oraz zaproponowanie najbardziej odpowiedniej alternatywy w tym kontekście.

Artykuł podzielony został na dwie części. W pierwszej skupiono się na analizie poprawności technologicznej dostępnych rozwiązań pod kątem projektowania warstwowego. W drugiej części przedstawiono alternatywne, autorskie, otwarte rozwiązanie zaprojektowane w sposób warstwowy wraz z przykładową implementacją. W obu częściach został przeanalizowany zakres środowiska programistycznego i sterownika PLC z pominięciem perspektywy kompleksowych systemów, takich jak systemy **SCADA**.

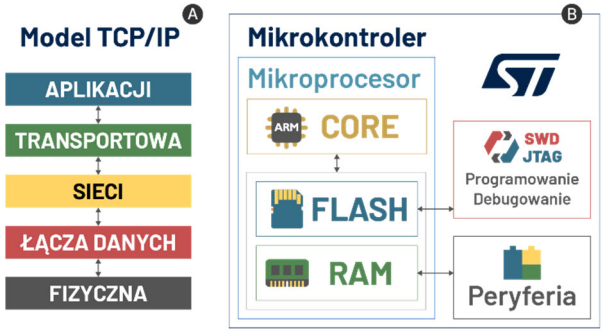
Projektowanie warstwowe

Naturalnym procesem podczas tworzenia większych, bardziej skomplikowanych systemów, aplikacji, języków programowania i sposobów komunikacji jest ich podział na odseparowane funkcjonalnie warstwy abstrakcji, które upraszczają złożone systemy poprzez ukrycie szczegółów niższych poziomów przed wyższymi [2]. Cechami charakterystycznymi projektowania warstwowego są:

- **specjalizacja** - warstwy mają określone zadania, co pozwala specjalistom skupić się na swoim obszarze;
- **modułowość** - umożliwia łatwe dodawanie, wymianę lub modyfikację poszczególnych części;
- **izolacja** - zmiany w jednej warstwie nie powinny wpływać na inne;
- **skalowalność** - warstwy mogą być rozwijane niezależnie, zapewniając efektywne dostosowanie do zwiększających się wymagań;
- **wzajemna wymienność** - warstwy można zastępować alternatywnymi rozwiązaniami bez naruszania całości systemu;

Te właściwości pomagają w zarządzaniu, testowaniu i rozwoju bardziej złożonych systemów, okazując się przydatne zarówno przy pracy ze sprzętem, jak i z oprogramowaniem.

Wspomniane warstwy mogą być zorganizowane w sposób liniowy, gdzie każda pełni swoją określoną funkcję, a komunikacja między nimi przebiega w układzie hierarchicznym, jak w modelach **TCP/IP** *(rys. 1A)*, **MVC** i oprogramowaniu **KiCad**, które jest zestawem aplikacji. Może się zdarzyć, że wyższa warstwa otacza niższą, zachowując separację, jak w frameworkach **Next.js** i **FastAPI**, a także w **systemach wbudowanych**, o czym może świadczyć sama budowa mikrokontrolera *(rys. 1B)*.



Rys 1. Model TCP/IP (A) oraz uproszczony schemat ideowy mikrokontrolera (B)

Relacje między warstwami mogą być bardziej złożone, jak w przypadku systemów operacyjnych. Kluczowe jest jednak oddzielenie części funkcjonalnych oraz zapewnienie dostępu do poszczególnych modułów.

Sterowniki PLC *"Zamknięte"*

Najpopularniejsze komercyjne sterowniki PLC zazwyczaj są dostarczane z dedykowanym

oprogramowaniem, które umożliwia tworzenie i wgrywanie własnych programów. W centrum każdego sterownika znajduje się mikrokontroler, który wykonuje obliczenia, steruje układami peryferyjnymi oraz komunikuje się z innymi urządzeniami poprzez interfejsy.

Niestety, dedykowane środowiska do programowania narzucają użytkownikom pewne ograniczenia. Umożliwiają korzystanie tylko z funkcji i bloków zapewnionych przez producenta. Takie rozwiązanie oczywiście pozwala szybko tworzyć programy o przejrzystej strukturze, jednak ograniczenie dla programisty dostępu do niższej warstwy odbiera możliwość pełnego wykorzystania mikroprocesora, co czasem jest niezbędne przy bardziej zaawansowanych lub nietypowych zadaniach.

W tego typu rozwiązaniach środowisko programistyczne pozwala na programowanie jedynie sterowników danego producenta, a sterowniki mogą być programowane tylko za pomocą dedykowanego środowiska. Prawdopodobnie istnieje wewnętrzny podział na warstwy, ale dostęp do nich, jak wspomniano wcześniej jest niedostępny dla programistów. Taka strategia jest zdecydowanie bardziej korzystna dla producentów niż dla programistów i konsumentów końcowych. Ogranicza to bowiem wybór i elastyczność dla programistów oraz może prowadzić do większej zależności od jednego dostawcy, co może wpływać na konkurencyjność rynkową i innowacje [3].

Pomimo tych ograniczeń, popularność sterowników takich jak **Siemens S7-1200**, **Wago 750-881** i **Allen-Bradley MicroLogix 1400** świadczy o ich niezawodności i dostosowaniu do potrzeb branży. Dlatego firmy często wybierają te produkty, nawet mimo pewnych wad.

Sterowniki PLC *"Otwarte"*

Zgodnie z ideą projektowania warstwowego, środowisko programistyczne powinno być łatwo dostosowywane do różnych urządzeń sprzętowych, a sterownik powinien umożliwiać programowanie bez konieczności korzystania z dodatkowej warstwy skoncentrowanej na obsłudze systemów automatyki. Dzięki temu programiści mieliby większą swobodę w wyborze rozwiązań, co sprzyjałoby innowacjom i konkurencyjności na rynku. Sukces otwartych standardów w branży serwerów i sieci jest dowodem na to, że tego typu podejście może działać również w automatyce. Otwarte interfejsy i projektowanie warstwowe w obszarze serwerów znacznie przyczyniły się do rozwoju chmur obliczeniowych i rozwiązań hybrydowych, takich jak np. [4].

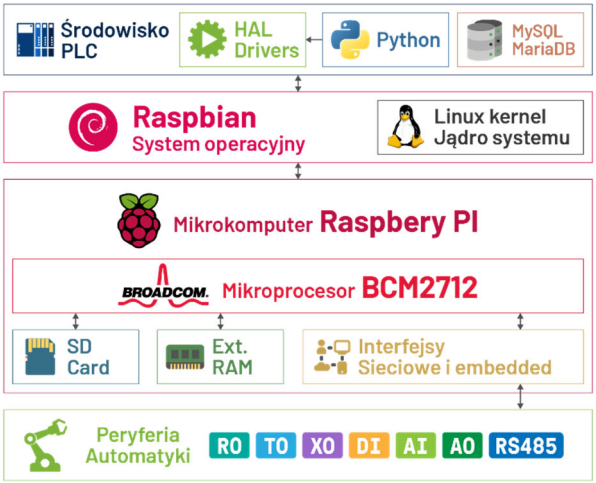
Najbardziej naturalnym kierunkiem rozwoju sterowników PLC w kontekście projektowania warstwowego będzie dodanie warstwy sprzętowej do mikrokomputera z otwartym systemem operacyjnym, takim jak **Linux** [5]. W ostatnim czasie na rynku pojawiło się wiele rozwiązań adaptujących mikrokomputery do zastosowań przemysłowych, takich jak **UniPi**, **Revolution Pi** oparte na **Raspberry P**i oraz **PLCnext** od **Phoenix Contact**.

Zatem opracowanie sterownika opartego na RPI wymaga utworzenia warstwy fizycznej, zgodnej z wyprowadzeniami mikrokomputera Raspberry Pi, obejmującej wybrane peryferia automatyki, takie jak:

- **RO** - Wyjścia przekaźnikowe
- **TO** - Wyjścia tranzystorowe
- **XO** - Wyjścia triakowe
- **DI** - Wejścia cyfrowe
- **AI** - Wejścia analogowe
- **AO** - Wyjścia analogowe
- **RS485** - Interfejs komunikacyjny

Warto także zapewnić pakiet narzędzi i sterowników umożliwiających sterowanie peryferiami automatyki oraz ułatwiających tworzenie aplikacji. Całość należy zawrzeć w automatycznym instalatorze, obrazie systemu lub obrazie Docker, co pozwoli na stworzenie kompletnego środowiska pracy z PLC *(rys. 2)*.

Alternatywną drogą jest umożliwienie bezpośredniego dostępu do programowania mikrokontrolera oraz dostarczenie wysokiej jakości bibliotek obsługujących podstawowe funkcje sterowników PLC i systemu Linux. Taką ścieżką podąża **Arduino**, które we współpracy z firmą **Finder** opracowało moduł **Opta**. Jednak to rozwiązanie jest zintegrowane z dedykowanym **IDE**, co wprawdzie nie uniemożliwia rozdzielenia produktów od siebie, ale na pewno skutecznie do tego zniechęca, gdyż wymusza samodzielne wykonanie konfiguracji na otwartym IDE, co jest bardzo wymagające.



Rys 2. Schemat koncepcyjny otwartego sterownika PLC opartego na mikrokomputerze z podziałem na warstwy

Na rynku jest miejsce dla sterowników PLC zarówno opartych na mikrokomputerach [6], które oferują większą moc obliczeniową, zaawansowaną obsługę interfejsów, takich jak USB, Ethernet czy Wi-Fi, oraz szeroką gamę oprogramowania, w tym języki interpretowane, bazy danych, systemd i cron, jak i tych opartych na mikrokontrolerach [7], które wyróżniają się wyższą niezawodnością, krótszym czasem uruchamiania, niższym zużyciem energii i kosztami.

Pojawia się zatem wyraźna luka, której wypełnienie mogłoby korzystnie wpłynąć zarówno na konkurencyjność, jak i na edukację w obszarze komputerowych systemów sterowania. Projektowanie warstwowe, z klarownym podziałem na poszczególne komponenty, pozwala lepiej zrozumieć działanie systemów i ułatwia proces ich nauki. Dzięki temu uczniowie i specjaliści mogą szybciej zdobywać praktyczne umiejętności, zwiększając swoją biegłość w nowoczesnych technologiach sterowania.

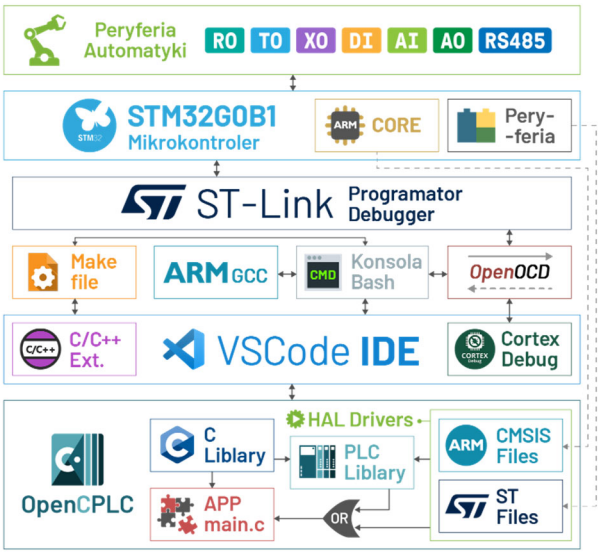
Projekt OpenCPLC

Zaproponowany w tej pracy projekt OpenCPLC może wypełnić wskazaną w poprzednim rozdziale lukę *(https://opencplc.org)*. W dużej mierze korzysta on z uznanych i popularnych rozwiązań. Język programowania używany w tym projekcie to C, który, pomimo swojego zaawansowanego wieku, wciąż jest najczęściej wybieraną technologią przez programistów systemów wbudowanych. Przykładowa konfiguracja obejmuje pracę z IDE **VSCode** z rozszerzeniami **Cortex Debug** i **C/C++**, ale nic nie stoi na przeszkodzie, aby skonfigurować projekt pod środowisko programistyczne **STM32CubeIDE**. W obu przypadkach do programowania i debugowania zostanie wykorzystany programator **ST-Link**, który standardowo używany jest do

pracy z mikrokontrolerami **STM32**. Do kompilacji projektu standardowo używany jest kompilator **ARM GCC** wraz z programem **Make**, który generuje i uruchamia odpowiednie komendy. Następnie program jest wgrywany do mikrokontrolera za pomocą aplikacji **OpenOCD**, która również pełni funkcję debugera.

W celu zautomatyzowania konfiguracji środowiska stworzono narzędzie **wizard.exe**. Jest ono dostępne na platformie **GitHub** wraz ze szczegółowym opisem konfiguracji i wskazówkami dla początkujących *(https://github.com/OpenCPLC)*.

Rysunek 3 przedstawia wszystkie opisane wyżej narzędzia w celu stworzenia sterownika PLC. Podobnie jak w przypadku mikrokomputera, system został rozbudowany o warstwę sprzętową, którą można sterować za pośrednictwem zaimplementowanej warstwy programowej.



Rys. 3 Mapa technologiczna projektu OpenCPLC z podziałem na warstwy

OpenCPLC *hardware*

Pierwszą autorską konstrukcją sprzętową jest demonstracyjny sterownik **Uno** *(rys. 4)*. Wykorzystano w nim mikrokontroler **STM32G0B1RE**, który ma **512kB** pamięci FLASH oraz **128kB** pamięci RAM. Posiada zintegrowany zegar czasu rzeczywistego *(**RTC**)*. Dołączono również zewnętrzny kwarc **18.432MHz**, którego częstotliwość zmniejsza błędy w komunikacji **RS485**.

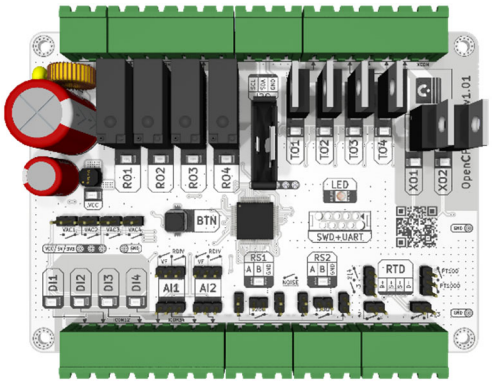
Po zakupie urządzenie jest zaprogramowane jako moduł rozszerzeń do współpracy z zewnętrznym sterownikiem lub komputerem. Jednak możliwe jest jego bezpośrednie programowanie, co pozwala wykorzystać je jako sterownik PLC. Jest ono kompatybilne zarówno z systemami pracującymi zarówno przy napięciu zasilania równym **24V**, jak i **12V**, co wyróżnia ten produkt na rynku. Urządzenie może być zasilane z obu tych źródeł, można sterować nimi płynnie z wyjść, a także odczytywać napięcie jako logiczną 1 na wejściach. Dzięki temu może wdrażać systemy automatyki, w których standardem zasilania jest **24VDC**, oraz systemy maszyn przemysłowych ze standardem **12VDC**. Urządzenie współpracuje z napięciami przemiennymi do **230V** i charakteryzuje się wszechstronnością także ze względu na różnorodność peryferii, co obrazuje Tabela 1.

Sterownik nie jest wyposażony w wiele peryferiów, dlatego najlepiej sprawdzi się w mniejszych projektach automatyki, takich jak:

- System nawadniania/naświetlania roślin
- Regulator temperatury/natlenienia w akwarium
- System monitoringu zużycia energii
- Automatyczny system podlewania trawnika/szklarni
- Automatyczny regulator prędkości wentylatora
- Inteligentny termostat - sterowanie piecem
- Automatyczny system podawania karmy
- Inteligentny kurnik, symulujący krótszą dobę
- Inkubator jajek

Tabela 1. Zestawienie układów peryferyjnych sterownika OpenCPLC Uno

Układ	Opis	Ilość
RO	Wyjścia przekaźnikowe: 5A 230VAC, 7A 30VDC. Licznik przełączeń i funkcja zapamiętywania stanu.	2×2=**4**
TO	Wyjścia tranzystorowe: 5A. Sterowane napięciem zasilania. Wszystkie mogą pracować w trybie power PWM lub jedno może pracować jako wyjście częstotliwościowe VFPWM.	**4**
XO	Wyjście triakowe: 12-230VAC. Brak detekcji przejścia przez zero.	2×1=**2**
DI	Wejścia cyfrowe: napięcia 12VDC, 24VAC i 230VAC są traktowane jako stan wysoki. Wszystkie mogą działać jako szybkie liczniki.	2×2=**4**
AI	Wejścia analogowe: 0-10V, 4-20mA lub 0-10V z wtórnikiem napięciowym.	**2**
RS485	Interfejs komunikacyjny z obsługą Modbus RTU, Profibus, BACnet lub do zastosowań niestandardowych.	**2**
RTD	Wejście czujników rezystancyjnych, dostosowane do PT100 i PT1000.	**1**
LED	Dioda informacyjna RGB.	**1**
BTN	przycisk, który działa jak wejścia DI.	**1**



Rys 4. Wizualizacja sterownika Uno OpenCPLC

OpenCPLC *software*

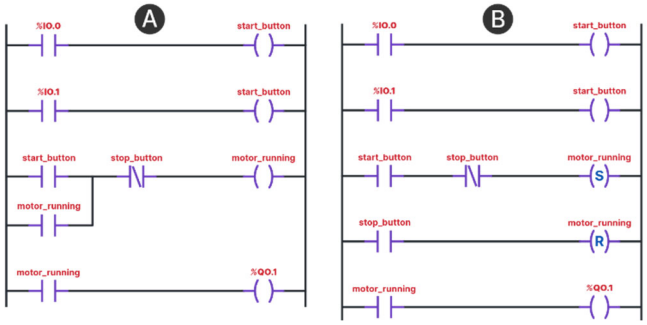
Projekt OpenCPLC zawiera minimalny zestaw plików potrzebnych do pracy z mikrokontrolerami STM32. Pliki te są dostarczane przez firmę **ARM**, projektującą rdzenie, oraz **STMicroelectronics**, producenta mikrokontrolerów. Pozostałe biblioteki to autorskie, otwartoźródłowe *(open source)* rozwiązania autora publikacji, dostępne w repozytorium na GitHubie.

- biblioteki **HAL** *(Hardware Abstraction Layer)* - foldery `lib/per/` i `lib/ifc/`;
- biblioteki ogólnego przeznaczenia w języku C - foldery `lib/ext/` i `lib/dev/`;
- System **VRTS** *(https://github.com/Xaeian/VRTS)*, umożliwiający programowanie wielowątkowe [8] - folder `lib/sys/`;
- Biblioteki **PLC**, które tworzą warstwę abstrakcji dedykowaną automatyce - folder `/plc/`

Każda konfiguracja sprzętowa jest połączona z odpowiednimi bibliotekami poprzez pliki mapujące wyprowadzenia mikrokontrolera oraz pinout sterownika PLC, zawierający skróty używane w automatyce. Dla sterownika **Uno** są to pliki `opencplc-uno.h` i `opencplc-uno.c`.

Oczywiście, w duchu warstwowego projektowania, można stworzyć własny plik mapujący dla indywidualnej implementacji sprzętowej i korzystać z dostępnych bibliotek OpenCPLC. Zaprojektowany sterownik można też programować jak zwykły mikrokontroler, bez dodatkowych bibliotek, jeśli z jakiegoś powodu nie spełniają one oczekiwań programisty lub wymogów projektowych. Można nawet zrezygnować z warstwy PLC i programować inne aplikacje systemów wbudowanych, wykorzystując pozostałe warstwy. Pełna elastyczność pozwala na szerokie dostosowanie do różnorodnych potrzeb.

Zapotrzebowanie na specjalistów w dziedzinie automatyki zawsze było i będzie wysokie. W przeszłości, gdy programistów było niewielu, a zadania automatyki przejmowali elektrycy, język drabinkowy ***(LAD)*** okazał się idealnym rozwiązaniem, ponieważ opierał się na logice znanej z elektryki. Dziś sytuacja się zmieniła, a kod w języku C bywa bardziej przystępny dla absolwentów kierunków technicznych niż drzewo logiczne złożone ze styków i cewek. Rysunki 5 i 6 przedstawiają oprogramowanie systemu start-stop odpowiednio w językach LAD oraz C. System uruchamia silnik po naciśnięciu jednego przycisku i wyłącza go po naciśnięciu drugiego.



Rys 5. Oprogramowanie systemu start-stop w języku LAD: klasycznie (A) oraz z użyciem cewek typu set i reset (B).

A

```
bool start_button = false;
bool stop_button = false;
bool motor_running = false;

void start_stop(void)
{
  while(1) {
    start_button = DIN_State(&DI1);
    stop_button = DIN_State(&DI2);
    if(stop_button) {
      motor_running = false;
    }
    else if(start_button || motor_running) {
      motor_running = true;
    }
    RELAY_Preset(&RO1, motor_running);
  }
}
```

B

```
DIN_t *start_button = &DI1;
DIN_t *stop_button = &DI2;
DOUT_t *motor_running = &RO1;

void start_stop(void)
{
  while(1) {
    if(DIN_Rais(stop_button)) {
      DOUT_Rst(motor_running);
    }
    else if(DIN_Rais(start_button)) {
      DOUT_Set(motor_running);
    }
  }
  let();
}
```

Rys. 6 Oprogramowanie systemu start-stop napisany w języku C z mapowaniem na zmienne (A) i wskaźniki (B)

Wsparcie AI

Od strony programistycznej sterownik OpenCPLC jest całkowicie otwarty, charakteryzując się minimalistyczną, monolityczną strukturą, a wszystkie zależności są przechowywane w plikach tekstowych. Bogactwo elementarnych przykładów i prostych projektów sprawia, że stanowi doskonały materiał dla **modeli językowych**, które mogą łatwo nauczyć się wspomagać automatyków w programowaniu tych sterowników.

Interfejsy graficzne, formularze i złożone, wielopoziomowe menu, choć ergonomiczne i przyjazne dla użytkowników, stanowią znaczące wyzwanie dla systemów sztucznej inteligencji. Istnieje niepewność, czy te elementy są w ogóle możliwe do nauki i zrozumienia przez algorytmy **AI**. Projekt OpenCPLC został zaprojektowany z myślą o maksymalnym ułatwieniu nauki dla sztucznej inteligencji.

Obecnie wdrożono najprostsze rozwiązanie, polegające na wykorzystaniu własnych modeli **GPTs**, które można stworzyć przy użyciu modelu **GPT-4** *(https://chatgpt.com)*. Model ten funkcjonuje pod nazwą OpenCPLC Assistant. Niestety, to podejście wymaga od potencjalnych użytkowników wykupienia płatnej wersji aplikacji.

W przyszłości bardziej odpowiednim rozwiązaniem będzie implementacja wbudowanego modelu językowego lokalnie. Można to zrealizować stosunkowo łatwo, korzystając z narzędzia **nanoGPT** *(https://github.com/karpathy/nanoGPT)*. Dzięki temu model językowy działałby lokalnie, zapewniając szybsze odpowiedzi oraz wyższy poziom prywatności danych użytkownika. NanoGPT umożliwia tworzenie wydajnych i zoptymalizowanych modeli, które można zintegrować z istniejącą aplikacją, zwiększając jej funkcjonalność bez konieczności korzystania z usług zewnętrznych.

Uwagi końcowe

W pracy przedstawiono autorski, otwarty i warstwowo zaprojektowany projekt OpenCPLC, który zawiera sterownik Uno oparty na mikrokontrolerze. Podkreślono jego zalety i unikalność w porównaniu z dostępnymi na rynku rozwiązaniami, w tym możliwość łatwego wsparcia przez modele językowe AI. Sterownik ten jest odpowiedni zarówno do zastosowań edukacyjnych, jak i do prostych systemów sterowania.

Autorzy: *mgr inż. Emilian Świtalski, prof. dr hab. inż. Krzysztof Górecki, Uniwersytet Morski w Gdyni, Wydział Elektryczny, Katedra Elektroniki Morskiej, ul. Morska 83, 81-225 Gdynia, E-mail: e.switalski@we.umg.edu.pl, k.gorecki@we.umg.edu.pl*

LITERATURA

1. M. Pawlak, "Sterowniki programowalne", Politechnika Wrocławska, 2010
2. Alvine B. Belle, Ghizlane El Boussaidi, Timothy C. Lethbridge, Segla Kpodjedo, Hafedh Mili, Andres Paz, "Systematically reviewing the layered architectural pattern principles and their use to reconstruct software architectures", arXiv preprint, DOI: 10.48550/arXiv.2112.01644
3. Melissa Schilling, "Strategic Management of Technological Innovation", McGraw-Hill Education, Edycja 4, 16 Luty 2013, ISBN-10: 0071326448, ISBN-13: 978-0071326445
4. Liu Xiang, Liu Shaobin, "Hybrid Cloud Networking Design Based on Openstack Architecture", Journal of Physics: Conference Series 1693 012011, DOI: 10.1088/1742-6596/1693/1/012011
5. I. Smołka, "Analiza metod implementacji sieci programowalnych w komputerowych systemach przemysłowych wykorzystujących przemysłowy Internet Rzeczy", 2023
6. D. Yulin, Z. Chunjiao, "Design and research of embedded PLC development system", ICCRD 2011, DOI: 10.1109/ICCRD.2011.5764286
7. B Ahmad; G Goswami; M. F Khan, "An Alternative Option to Commercial Programmable Logic Controllers", SMART, 2021, DOI: 10.1109/SMART52563.2021.9676213
8. E. Świtalski, K. Górecki: System zwalniania wątków VRTS jako alternatywa dla RTOS. Przegląd Elektrotechniczny, R. 99, nr 9, (2023), s. 258-261. doi:10.15199/48.2023.09.52