Fundusze Europejskie dla Rozwoju Społecznego
Rzeczpospolita Polska
Dofinansowane przez Unię Europejską

Politechnika Świętokrzyska
Wydział Elektrotechniki, Automatyki i Informatyki

Kierunek studiów:
Elektrotechnika

Tomasz Grzmil

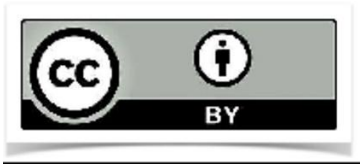
Materiały dydaktyczne do przedmiotu

BUDOWA I OPROGRAMOWANIE KOMPUTEROWYCH SYSTEMÓW STEROWANIA

opracowane w ramach realizacji Projektu
„Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
FERS.01.05-IP.08-0234/23

Kielce, 2025

Spis treści



Materiały dydaktyczne objęte licencją Creative Commons BY 4.0.

Licencja dostępna pod adresem: https://creativecommons.org/licenses/by/4.0/

1. Wprowadzenie do komputerowych systemów sterowania

System automatyki przemysłowej to zespół środków technicznych, który zbiera wartości zmiennych opisujących stan procesu, obserwuje je i alarmuje, przetwarza je matematycznie, wypracowuje decyzje sterujące, umożliwia komunikację człowiek-maszyna i nieustannie testuje własną sprawność. Stosowanie takiego systemu ma konkretny cel: podnieść wydajność i jakość produkcji, poprawić bezpieczeństwo i niezawodność, a przy okazji obniżyć koszty. Jeżeli zasadniczą część obliczeń realizują komputery, mówimy o komputerowym systemie automatyki przemysłowej (KSS). W praktyce zdecydowana większość eksploatowanych dziś układów sterowania ma właśnie tę postać.

Struktura systemu sterowania powinna odzwierciedlać strukturę zakładu. Najczęściej wyróżnia się trzy warstwy:

- zarządzanie przedsiębiorstwem (systemy ERP, MES),
- sterowanie nadrzędne nadzorujące linie i ciągi technologiczne,
- sterowanie bezpośrednie pojedynczych maszyn i urządzeń.

W takiej hierarchii KSS łączy się zarówno z warstwą biznesową, przekładając harmonogramy i receptury na zadania operacyjne, jak i z warstwą urządzeń polowych, zbierając dane procesowe w czasie rzeczywistym, obliczając kluczowe wskaźniki (KPI) i dostarczając operatorom bieżących informacji o wydajności.

Główne funkcje KSS:

1. **Akwizycja danych** – synchroniczny i asynchroniczny odczyt sygnałów analogowych, binarnych, komunikacyjnych.
2. **Przetwarzanie i decydowanie** – algorytmy regulacji (PID, MPC, sterowanie sekwencyjne) oraz logiki bezpieczeństwa.
3. **Aktywna ingerencja** – bezpośrednie wymuszanie wielkości sterujących (zawory, napędy, przekaźniki).
4. **Interakcja z człowiekiem** – wizualizacja, raporty, alarmy, rejestry zdarzeń.
5. **Autodiagnostyka** – ciągłe testy sprzętu i oprogramowania, sygnalizacja usterek

*Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23*

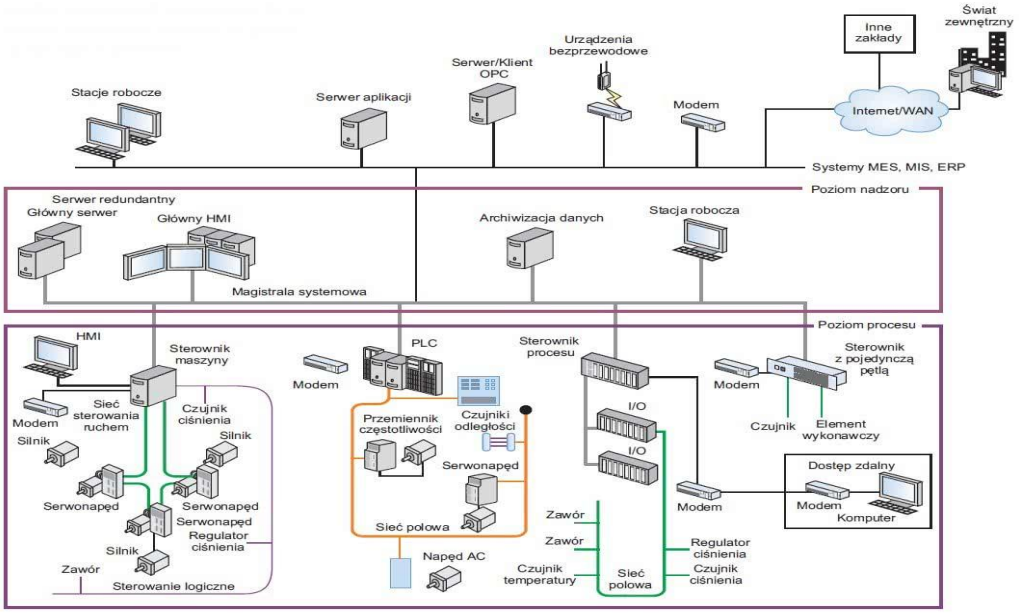
Tabela 1. Krótki zarys rozwoju historycznego.

Okres	Kamienie milowe	Charakterystyka
Pionierski (1959)	Rafineria TEXACO – pierwszy komputer sterujący	Tranzystory, obliczenia ~1 ms, MTBF ≈ 50 h; komputer doradzał operatorowi
DDC – Direct Digital Control (1962)	Ferranti Argus steruje 224 pomiarami, 129 zaworami	Pętle cyfrowe, szybsze obliczenia (100 µs add), MTBF 1000 h, pierwsze specjalizowane języki
Minikomputery (od 1967)	CDC 1700, później MODICON 084 – prekursor PLC	Sumowanie 2 µs, niezawodność 20 000 h, modułowa budowa, odporność przemysłowa
Mikrokomputery (od 1971)	Intel 8080 w sterownikach Allen-Bradley	Spadek ceny sprzętu (PC-PLC), gwałtowny wzrost zastosowań rozproszonych
Standaryzacja (od 1980)	IEC 1131-3, magistrale, sieci przemysłowe, OPC	Ujednolicenie OS (MS-DOS, Windows), rozwój SoftLogic i otwarte interfejsy komunikacyjne

Wykorzystanie komputerów wniosło szybkość i elastyczność obliczeń, możliwość implementacji złożonych algorytmów, integrację z systemami IT oraz łatwą rekonfigurację oprogramowania. Dzięki temu KSS potrafią adaptować się do zmiennych warunków procesu, skracać czas wprowadzenia produktu na rynek i zapewniać pełną widoczność danych od czujnika aż po księgowość.

[Tekst alternatywny. Schemat blokowy. Diagram przedstawia rozproszony system sterowania. W górnej części znajdują się stacje robocze, serwery, urządzenia bezprzewodowe i połączenie z Internetem. Środkowa część pokazuje magistralę systemową z serwerami, HMI i archiwizacją danych. Dolna część ilustruje poziom

procesu: sterowniki, sieci polowe, czujniki, silniki, zawory oraz elementy wykonawcze.]



Rysunek 1. Diagram rozproszonego systemu sterowania. [12].

2. Sprzęt komputerowych systemów sterowania

Programowalne sterowniki logiczne (PLC) - podstawowy element warstwy sterowania bezpośredniego. W zależności od skali procesu spotykamy sterowniki logiczne modułowe klasy „logo” (np. Siemens LOGO!) przeznaczone do prostych sekwencji i aplikacji HVAC (Heating, Ventilation, Air Conditioning), kompaktowe PLC (dawniej S7-200, dziś S7-1200), modułowe PLC średniej klasy (S7-300 zastępowane stopniowo przez S7-1500 z lepszym bezpieczeństwem i procesorami ARM-Cortex) oraz sterowniki procesowe/rackowe (S7-400, redundantne S7-400H, nowsze S7-1500R/H) wykorzystywane w DCS i systemach wysokiej dostępności.

[Tekst alternatywny. Fotografia. Przedstawiono różne modele sterowników PLC firm Siemens, Mitsubishi, Allen-Bradley i Omron.]

Rysunek 2. Różne rodziny sterowników PLC.

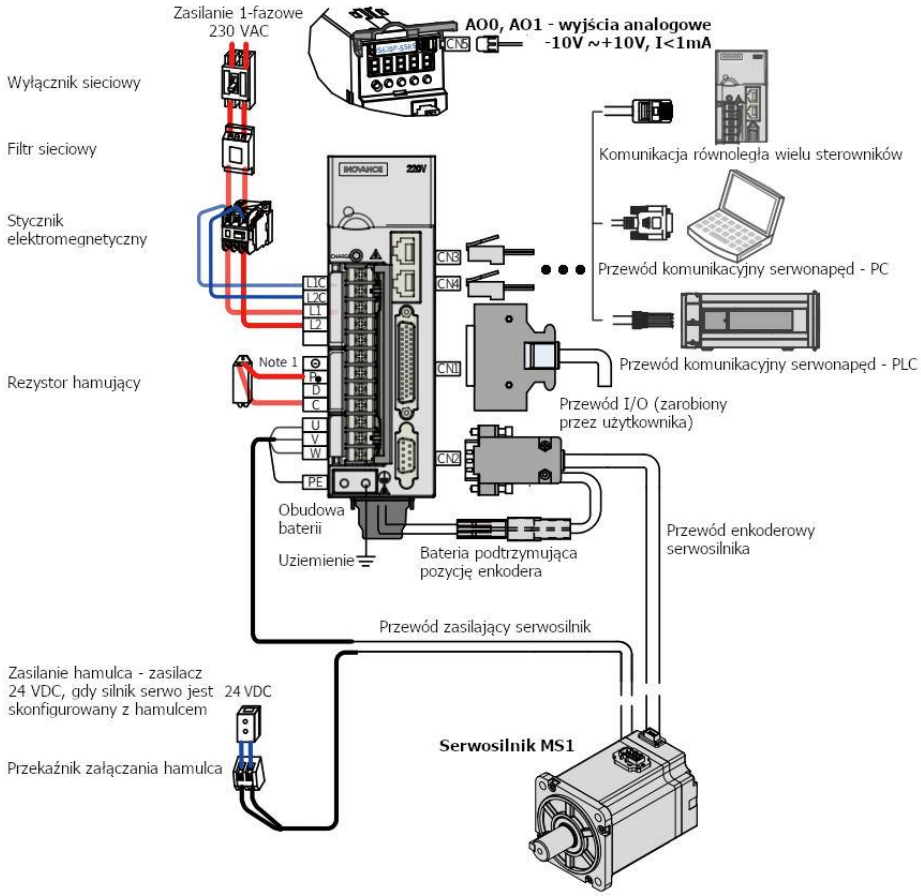
Jednostki I/O rozproszone - rodzina ET 200 mierzy sygnały procesowe i steruje urządzeniami wykonawczymi bezpośrednio przy maszynie, łącząc się z PLC przez PROFIBUS lub PROFINET. Wariant ET 200MP obsługuje już sieci gigabitowe oraz Time-Sensitive Networking (TSN).

Sterowniki panelowe i panele HMI (SIMATIC C7, panele TP/KTP, MultiPanel) - integrują wizualizację z logiką sterującą, co zmniejsza wymagania montażowe i upraszcza okablowanie. W nowych instalacjach coraz częściej stosuje się web-based HMI oraz panele z systemem Edge.

Komputery przemysłowe (IPC) - klastry Box-PC, Rack-PC i Panel-PC łączą ekosystem Windows IoT LTSC lub Linux RT z kartami akwizycji; dysponują ochroną IP65, odpornością na wibracje i temperaturę. Dzisiaj standardem są procesory x86-64 12-tej/13-tej generacji i sloty PCIe Gen4.

Sterowniki ruchu - platformy CNC (SINUMERIK) i Motion Controller (SIMOTION) realizują interpolację wieloosiową z częstotliwością pętli większej bądź równej 1 kHz, posiadają sprzętowe interfejsy enkoderów i wyjścia ±10 V lub EtherCAT/PROFINET-IRT dla serwonapędów. Od 2024 r. standard PROFIdrive V5 wspiera profil synchronizacji na 250 µs.

[Tekst alternatywny. Schemat. Przedstawia przykładowe podłączenie serwonapędu. Uwzględniono zasilanie, wyłączniki, rezystor hamujący, przewody komunikacyjne do PLC i PC, a także połączenia z serwosilnikiem i jego enkoderem.]



Rysunek 3. Diagram połączenia serwonapędu. [13]

Modułowe systemy akwizycji - NI CompactDAQ (USB, Ethernet, Wi-Fi), PXI/PXIe oraz CompactRIO łączą pomiar czasu rzeczywistego, synchronizację sub-µs i wysokopoziomowe API DAQmx. USB w prezentacji odnosi się do wersji 2.0 (480 Mb/s); obecnie typowe moduły pracują po USB4 40 Gb/s, a Ethernet łączy się z 10 Gb/s backplane.

Karty pomiarowe (DAQ) - od prostych USB-6009 do wielokanałowych PCIe-6363; zapewniają przetworniki ADC do 2 MS/s na kanał i wbudowane silniki DMA. Nowe układy wspierają sprzętową synchronizację IEEE 1588-PTP i akceleracją w FPGA.

[Tekst alternatywny. Fotografia. Zestaw systemów NI DAQ: moduł USB DAQ, CompactDAQ, CompactRIO oraz PXI, widocznych obok komputera z oprogramowaniem do akwizycji danych.]

Rysunek 4. Zestaw modułów NI DAQ. [14]

Magistrale i sieci polowe - klasyczne PROFIBUS DP, CAN FD oraz Modbus RTU wciąż znajdują zastosowanie, lecz dominują sieci Ethernetowe klasy PROFINET IRT, EtherCAT, Ethernet/IP. Coraz częściej implementuje się Time-Sensitive Networking w warstwie 2, a sterowniki S7-1500 i Beckhoff CX7000 mają sprzętowe bloki TSN.

Kluczowe definicje:

- **PLC** - autonomiczny kontroler przemysłowy z cyklicznym skanem programu i wbudowanymi I/O.
- **PAC** - kontroler automatyki programowalnej łączący logikę PLC, przetwarzanie wątkowe i komunikację IT, często w architekturze przemysłowego PC lub SoC.
- **IPC** - komputer PC w wykonaniu przemysłowym, pracujący 24/7 w rozszerzonym zakresie temperatur, stosowany jako SCADA, brama OT-IT albo Soft-PLC.
- **Remote I/O** - moduły wejść/wyjść instalowane przy maszynie, komunikujące się z PLC po sieci polowej.
- **Motion Controller** - układ mikroprocesorowy lub FPGA synchronizujący wiele osi z dokładnością mikrosekundową.

*Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23*

Dzisiejsze systemy sprzętowe łączą klasyczne sterowniki PLC z otwartymi platformami IPC i sieciami czasu rzeczywistego, zapewniając jednocześnie bezpieczeństwo funkcjonalne (Fail-safe) i cyberbezpieczeństwo zgodne z IEC 62443.

3. Oprogramowanie komputerowych systemów sterowania

Współczesny komputerowy system sterowania składa się z warstwy sprzętowej, która wykonuje pomiar i sterowanie, oraz warstwy programowej, która determinuje sposób działania całości. Oprogramowanie przyjmuje trzy główne postaci: firmware zapisany w nieulotnych pamięciach urządzeń, system operacyjny zarządzający zasobami oraz aplikacje użytkownika realizujące logikę procesu i wizualizację.

Firmware - to kod producenta urządzenia (np. BIOS sterownika PLC), inicjalizuje sprzęt, udostępnia tablice rejestrów, zapewnia procedury autodiagnostyki. W nowych jednostkach PLC i I/O firmware obsługuje aktualizacje „over-the-air" z podpisem cyfrowym.

System operacyjny (OS) - oprogramowanie pośredniczące między sprzętem a aplikacjami, zarządzające czasem procesora, pamięcią, urządzeniami we/wy i komunikacją, zapewniające jednolity interfejs programistyczny. Pierwsze PLC pracowały pod własnymi mikrokernami, później pojawiły się DOS i Windows Embedded. Dziś dominują trzy nurty:

- **RTOS** - system operacyjny zapewniający przewidywalny czas reakcji. W klasycznych sterownikach przemysłowych stosuje się m.in. VxWorks 7 i QNX 7.1, które oferują deterministyczne szeregowanie zadań oraz spełniają wymagania bezpieczeństwa SIL 3.
- **Linux PREEMPT_RT / NI Linux RT** w sterownikach PAC i komputerach przemysłowych; jądro 6.8 oferuje latencję < 40 µs (aktualizacja 2025).
- **Systemy kontenerowe** – Siemens Industrial Edge, GE Proficy Operations Hub i CODESYS Virtual Control uruchamiają soft-PLC w kontenerach OCI lub na K3s, pozwalając skalować moc obliczeniową bez zmiany sprzętu.

Tabela 2. Udział systemów operacyjnych (całe spektrum urządzeń).

System	Udział w rynku
Android	~72%
Windows	~70%
iOS	~26-27%
macOS	~5-6%

Linux	~4%
Chrome OS	~1-2%
KaiOS i inne	Poniżej 0,1-0,2%

Języki i środowiska programowania - standard IEC 61131-3 wciąż definiuje pięć klasycznych języków PLC: LD, FBD, ST, IL, SFC. Nowe wersje środowisk (CODESYS 3.5 SP21) dodają obsługę przestrzeni nazw i kontenerów Dockera. Do szybkiego prototypowania i integracji OT-IT używa się Python (asyncua, pydaqmx), Node-RED oraz bibliotek MQTT. Graficzne platformy LabVIEW i Simulink ułatwiają projektowanie algorytmów ciągłych.

Soft-PLC - to program realizujący cykl sterownika na zwykłym PC lub w maszynie wirtualnej. Rozwiązania Paradyne-31 i ISaGRAF z lat 90. zostały wyparte przez:

- CODESYS Control SL / Virtual Control – licencjonowany runtime działający w kontenerze lub hyperwizorze.
- WinCC Unified Runtime Soft-PLC integrujący sterowanie i HMI na tym samym IPC.

Wirtualizacja upraszcza redundancję i pozwala migrować sterownik między węzłami edge i chmurą bez przestoju.

Systemy SCADA i HMI - zbierają dane procesowe, wizualizują je i przekazują operatorowi. Klasyczne pakiety (InTouch, WinCC Classic) ustępują miejsca nowym platformom:

- **WinCC Unified V20** - jeden silnik HTML5 dla paneli i PC, niskolatencyjne WebSockety, widżety Angula.
- **iFIX 2024** - centralna konfiguracja z Proficy Operations Hub, wsparcie kontenerów i trybu „Connected Worker”.
- **Ignition 8.1 LTS / 8.3 (beta 2025)** - modularna architektura bazująca na Java i MQTT Sparkplug-B, tryb Perspective dla urządzeń mobilnych.

SCADA współpracują z bazami danych czasu rzeczywistego, stosują OPC UA PubSub i mechanizmy TSN, aby zsynchronizować dane z dokładnością mikrosekund.

Oprogramowanie akwizycji danych - biblioteka NI-DAQmx, sterowniki EtherCAT i Modbus, a także otwarte API OPC UA tworzą warstwę abstrakcji między sprzętem a logiką. Najnowsze wersje DAQmx 24.0 dodają natywną obsługę IEEE 1588-PTP, a asyncua 1.1 wspiera komunikaty binarne UADP.

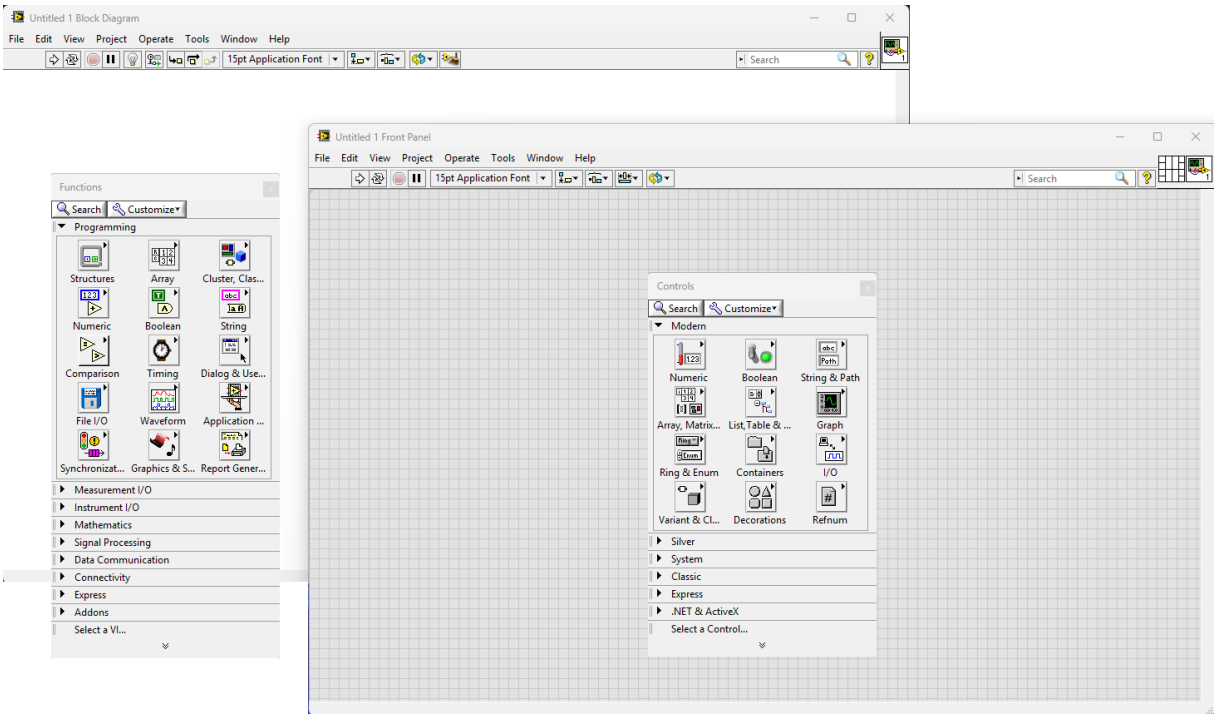
Edge computing - uruchamianie aplikacji sterowania i analityki bliżej źródła danych w kontenerach lub na mikrokontrolerach.

Oprogramowanie ewoluowało od monolitycznych aplikacji DOS do mikrousług w kontenerach, umożliwiając elastyczną skalowalność, wysoką dostępność i płynną integrację OT–IT.

4. LabVIEW – graficzne środowisko inżynierii pomiarowej i sterowania

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) - środowisko firmy National Instruments oparte na graficznym języku G, w którym programy powstają przez łączenie węzłów obliczeniowych połączonych przewodami danych. Wykonanie kodu determinuje przepływ danych (dataflow), a nie kolejność instrukcji. Dzięki temu poszczególne fragmenty VI mogą działać równolegle na wielu wątkach CPU lub na FPGA.

[Tekst alternatywny. Zrzut ekranu. Widok środowiska LabVIEW: po lewej Block Diagram z paletą Functions, po prawej Front Panel z otwartą paletą Controls.]

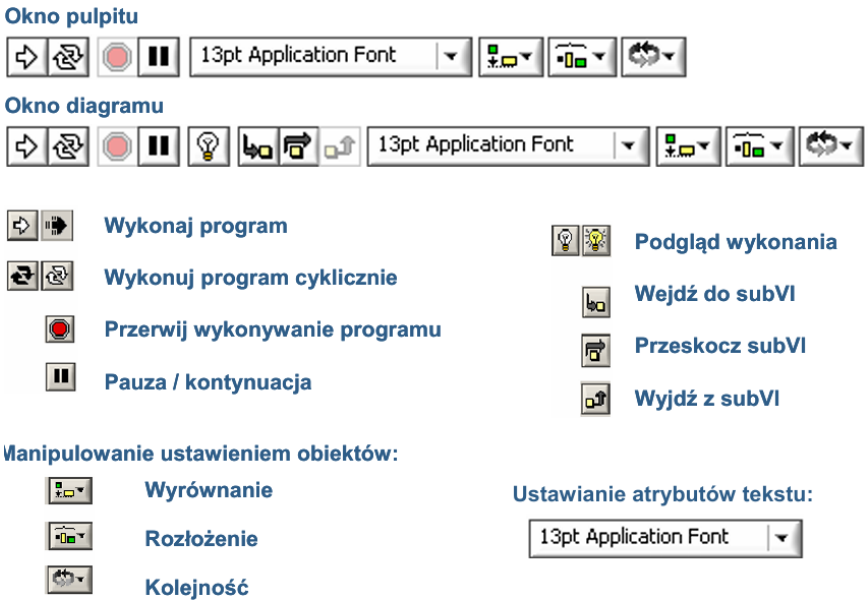


Rysunek 5. Widok środowiska LabVIEW.

Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

Wirtualny przyrząd (VI) - każdy projekt składa się z front panelu (interfejs operatora) i diagramu blokowego (logika) powiązanych ikoną-złączem, która umożliwia użycie VI jako podprogramu. Narzędzia do wprowadzania i łączenia elementów udostępniają palety *Controls* oraz *Functions* (ich układ i skróty klawiaturowe nie zmieniły się od wersji 8, lecz od LabVIEW 2024 wszystkie edytory działają natywnie w 64-bit i obsługują skalowanie DPI).

[Tekst alternatywny. Zrzut ekranu. Belki narzędziowe środowiska LabVIEW z ikonami do uruchamiania, zatrzymywania i debugowania programu oraz ustawiania atrybutów obiektów.]



Rysunek 6. Belki narzędziowe środowiska LabVIEW.

Typy danych i przewody - w języku G stosuje się typy skalarne (I8…I64, SGL, DBL), ciągi, ścieżki, a także struktury array i cluster; kolor i grubość przewodu wskazują typ i wymiar danych. Od wydania 2023Q1 wprowadzono *interfaces* oraz *generics* w bibliotekach klas, co ułatwia stosowanie wzorców projektowych.

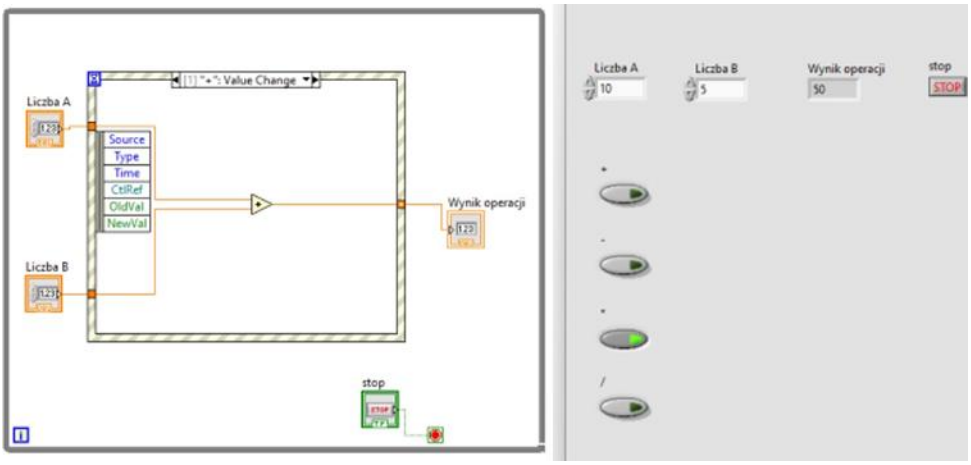
[Tekst alternatywny. Zrzut ekranu. Widok typów danych w środowisku LabVIEW z ikonami i odpowiadającymi im nazwami, m.in. liczby całkowite, zmiennoprzecinkowe, tablice, klastery, wykresy, dane cyfrowe i łańcuchy znakowe.]

Rysunek 7. Typy danych w środowisku LabVIEW.

Struktury sterujące - podstawę sterowania przepływem stanowią pętle while i for z rejestrami przesuwnymi, struktura Case, sekwencje Flat/Stack Sequence, pętla Timed Loop dla kodu deterministycznego oraz Event Structure do reakcji na zdarzenia użytkownika lub urządzeń. W LabVIEW 2024 przyspieszono *Timed Loop* (jitter < 5 µs na NI Linux RT) i dodano monitor czasu rzeczywistego w oknie *Timing Settings*.

Narzędzia inżynierskie - debugowanie ułatwiają tryby *Highlight Execution*, *Probe*, *Breakpoint* i *Step-Into*. Formula Node pozwala wpisać fragmenty składni przypominającej C, a MathScript (obecnie wtyczka community) obsługuje składnię MATLAB-ową.

[Tekst alternatywny. Zrzut ekranu. Przykład prostego kalkulatora w środowisku LabVIEW: po lewej diagram blokowy operacji, po prawej panel użytkownika z polami wejściowymi, wynikiem i przyciskami sterującymi.]



Rysunek 8. Prosty kalkulator w środowisku LabVIEW.

Integracja ze sprzętem - LabVIEW udostępnia jednolite API NI-DAQmx dla kart pomiarowych, VISA dla instrumentów GPIB/USB/LXI oraz biblioteki NI-RIO do systemów CompactRIO/FPGA. Od DAQmx 24.0 możliwa jest synchronizacja PTP IEEE 1588 bez dodatkowego kodu. Węzły Python Node (od 2021) wspierają interpreter Python 3.12 i umożliwiają wywoływanie skryptów *asyncua* lub *pydaqmx* wprost z diagramu blokowego.

Architektura projektu - eksplorator *LabVIEW Project* grupuje VI, biblioteki (.lvlib), klasy (.lvclass) i pliki konfiguracyjne. Struktury *Actor Framework*, *Queued Message Handler* czy *Producer-Consumer* są dostarczane jako szablony i obsługują komunikację kanałami Channel Wires zapewniającą synchronizację i bezpieczeństwo wątków (funkcja dodana w 2016, rozbudowana o kanały strumieniowe w 2024 – aktualizacja 2025).

Rozszerzenia czasu rzeczywistego i FPGA - Pakiet LabVIEW Real-Time tworzy deterministyczne aplikacje działające na kontrolerach NI Linux RT, natomiast LabVIEW FPGA kompiluje kod G do bitstreamu VHDL dla układów Xilinx; w wersji 2024 dodano symulator *cycle-accurate* oraz debug *Instrumented FPGA VI*.

Współpraca z chmurą i DevOps - zarządzanie zależnościami odbywa się przez NI Package Manager; CI/CD ułatwia *G-CLI* oraz oficjalny kontener Docker *ni-labview-cli*. SystemLink™ Cloud oferuje zdalną kompilację, dystrybucję buildów i monitorowanie urządzeń edge.

Kluczowe definicje:

- **VI** - program w LabVIEW składający się z front panelu i diagramu blokowego.

- **G** - graficzny język danych, w którym kolejność wykonania wynika z przepływu danych.
- **Front panel** - wirtualny przyrząd operatora zawierający kontrolki (wejścia) i wskaźniki (wyjścia).
- **Diagram blokowy** - sieć węzłów obliczeniowych i przewodów reprezentująca logikę.
- **Timed Loop** - struktura pętli z harmonogramem okresu, priorytetem i źródłem zegara.
- **Channel Wire** - bezpieczny, typowany kanał komunikacyjny między wątkami w G.

LabVIEW 2024 Q4 stanowi dziś fundament wielu laboratoriów i systemów sterowania dzięki elastycznemu łączeniu kodu graficznego z modułami RT, FPGA i węzłami Python, a także integracji z narzędziami DevOps i chmurą.

5. Akwizycja danych z wykorzystaniem kart pomiarowych

Komputerowy system sterowania rejestruje sygnały procesowe przez **karty pomiarowe** (DAQ) lub moduły CompactDAQ/PXI i generuje sygnały sterujące z dokładnością wyznaczoną rozdzielczością przetworników. Kluczowe parametry karty to liczba i rodzaj kanałów (AI, AO, DIO, CNT), rozdzielczość ADC/DAC, dopuszczalny zakres napięć, maksymalna częstotliwość próbkowania, bufor DMA i możliwości synchronizacji.

Podstawowe pojęcia:

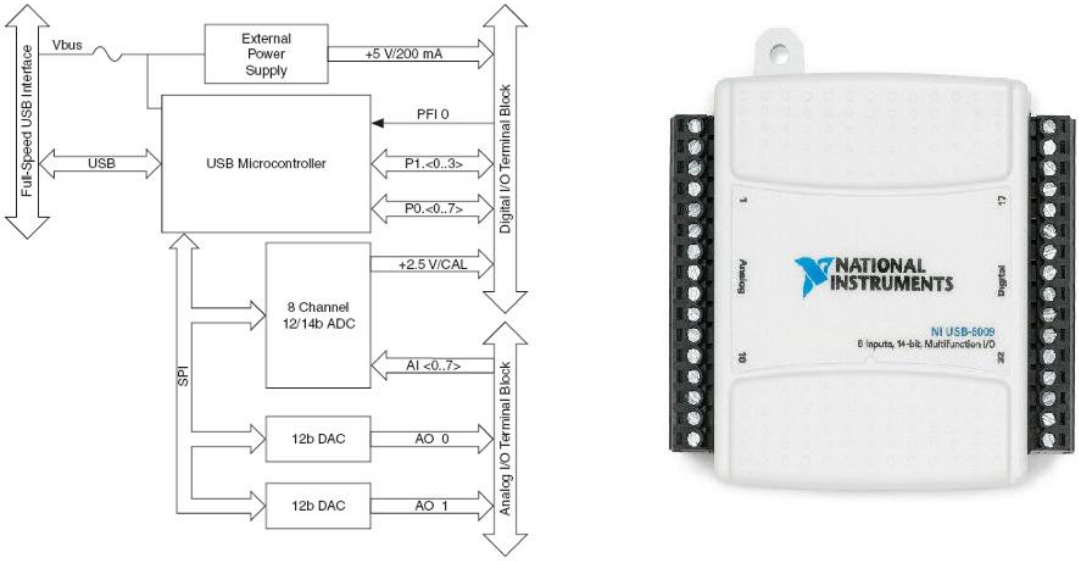
- **Rozdzielczość N-bit** - najmniejszy krok kwantyzacji wynosi $1/2^N$ pełnej skali, a zakres dynamiczny to $20log_{10}(2^N)$ dB
- **LSB / FS** - najmłodszy bit i pełna skala odniesienia sygnału.
- **Twierdzenie Nyquista** - prawidłowa rekonstrukcja sygnału wymaga próbkowania większego od 2 mnożone razy najwyższej składowej widma.
- **Błędy ADC** - przesunięcia i wzmocnienia, nieliniowości INL/DNL, szum i dryft temperaturowy to główne źródła niepewności pomiaru.

Przykładowe urządzenia:

- **Advantech PCI-1718 - 16-bit, 100 kS/s** - programowanie przez sterownik DLL lub rejestry I/O.
- **NI USB-6008 - 12-bit, 10 kS/s, 8 AI + 2 AO + 12 DIO** - popularna w laboratoriach dydaktycznych lecz wycofana z produkcji w 2024.

- **CompactDAQ (cDAQ-9189, 4-slot LAN)** - wbudowany układ czasowy i synchronizacja IEEE 1588; obsługa w NI-DAQmx od wersji 20.7, a od 24.5 umożliwia sprzętowe wyzwalanie przez TSN.

[Tekst alternatywny. Schemat i fotografia. Karta pomiarowa NI USB-6009: po lewej schemat blokowy z układem mikrokontrolera, przetwornikami ADC/DAC i interfejsami I/O, po prawej zdjęcie urządzenia.]



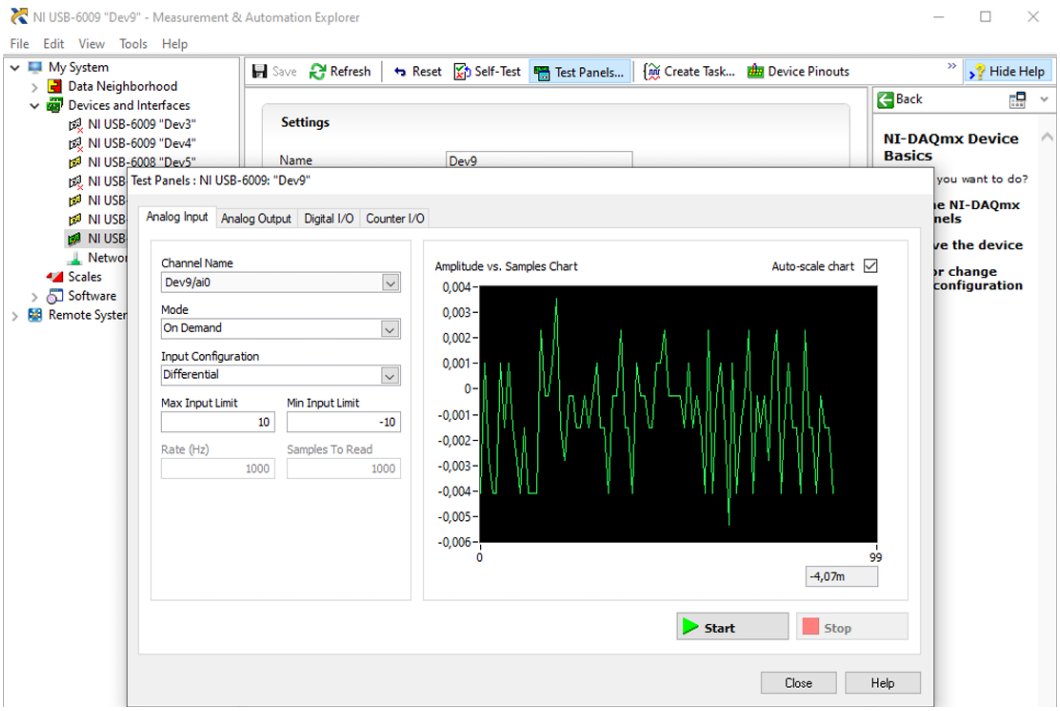
Rysunek 9. Karta pomiarowa NI USB-6009 wraz z schematem blokowym.

Sterowniki i API - karty DAQ współpracują z warstwą sterowników:

- **Traditional NI-DAQ** - biblioteka 32-bit.
- **NI-DAQmx** - aktualna architektura; wersja 24.5 (2024 Q3) wprowadza kanały czasu rzeczywistego z PTP / TSN oraz usprawnienia w Python API.
- **NI-DAQmx Base** - uproszczony pakiet dla Linux/macOS (ostatnia wersja 19.1, brak wsparcia dla nowych urządzeń).
- **Advantech ActiveDAQ / GeniDAQ** - graficzne konfiguratory, niższy poziom zapewnia DLL Driver i dostęp do rejestrów sprzętu.

Języki programowania - C/C++ i C# korzystają z bezpośredniego API. LabVIEW udostępnia gotowe VI: Cont Acq&Graph Voltage, Write Dig Port itp. Python używa pakietu nidaqmx *(pip install nidaqmx)* lub PyDAQmx z pełnym mappingiem funkcji graficznych przykładów LabVIEW.

[Tekst alternatywny. Zrzut ekranu. Widok programu NI MAX z listą urządzeń USB-6009 oraz otwartym panelem testowym z konfiguracją wejść analogowych i wykresem sygnału.]



Rysunek 10. Program NI MAX do testowania kart pomiarowych.

Konfiguracja i zadania - program Measurement & Automation Explorer (MAX) pozwala utworzyć wirtualne kanały, zadania i skalowania, a także podłączyć urządzenia symulowane do testów bez sprzętu. W kodzie NI-DAQmx zadanie tworzy się przez: CreateTask → CreateAI*Chan → CfgSampClkTiming → StartTask. Blok DAQ Assistant generuje ten sam kod automatycznie, ale przy przenoszeniu aplikacji na inny komputer wymaga powtórnej konfiguracji.

Zegar, wyzwalanie i synchronizacja - karty PCIe/PXI dzielą wewnętrzny pacer clock (typowo 20 MHz) na wymaganą częstotliwość próbkowania. Tryby oversampling i ciągły transfer DMA minimalizują jitter. W systemach rozproszonych cDAQ-9189/cRIO-905x udostępniają zegary PTP do synchronizacji z siecią (<1 µs), a najnowsze chassis obsługują nadrzędny zegar TSN. Zewnętrzne wejścia START TRIG lub REF TRIG pozwalają zsynchronizować pomiar wielu urządzeń.

Typowe błędy konfiguracji - niewłaściwy tryb wejścia (single-ended zamiast differential) pogarsza CMRR i powoduje aliasing zakłóceń sieciowych. Nieodpowiednia szybkość próbkowania (< 2×-max f sygnału) prowadzi do aliasingu

zgodnie z Nyquistem. Błędny wybór zakresu ±10 V dla czujnika ±100 mV zubaża rozdzielczość o ~7 bitów. Wreszcie brak kalibracji termicznej skutkuje dryftem offsetu (do kilkunastu LSB/dobę).

Kluczowe definicje:

- **AI / AO / DIO / CNT** - kanały analogowe wejściowe, analogowe wyjściowe, cyfrowe oraz liczniki.
- **Pacer clock** - sprzętowy generator tyknięć wyzwalający kolejne próbki.
- **Task** - kontener NI-DAQmx łączący kanały, taktowanie i warunki wyzwalania.
- **INL / DNL** - miary nieliniowości całkowej i różnicowej przetwornika.
- **PTP / TSN** - sieciowe protokoły synchronizacji czasu z dokładnością mikrosekundową.

Nowoczesna akwizycja danych wykorzystuje sterowniki NI-DAQmx 24.x, Python lub LabVIEW do elastycznej konfiguracji, a sprzęt wspiera PTP, TSN i szybkie magistrale PCIe Gen4/USB4. Dzięki temu projektant może zbudować skalowalny łańcuch od czujnika aż po chmurę bez rezygnacji z dokładności ani deterministycznego czasu próbkowania.

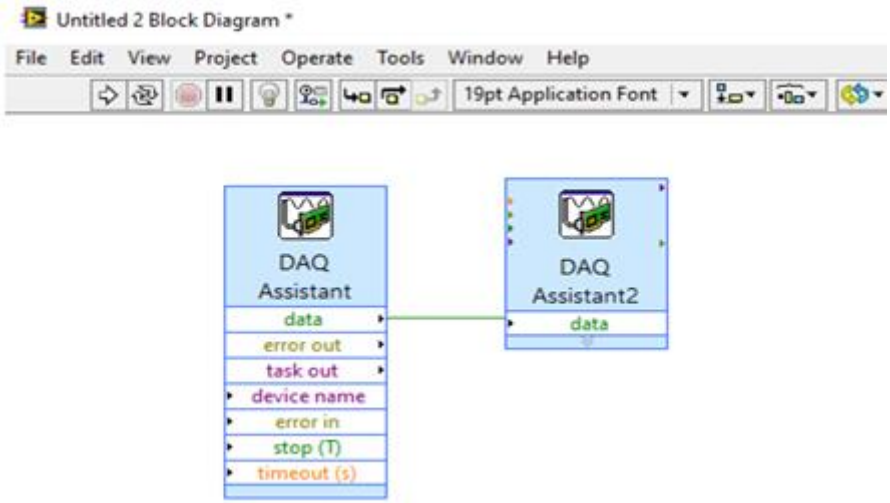
5.1. Obsługa karty pomiarowej NI USB-6009

NI USB-6009 to wielokanałowa karta pomiarowa z interfejsem USB, umożliwiająca akwizycję sygnałów analogowych, cyfrowych oraz generację napięć. Urządzenie współpracuje z oprogramowaniem LabVIEW oraz środowiskami programistycznymi opartymi na NI-DAQmx.

Obsługa w LabVIEW:

1. **Sterowniki:** Wymagane jest zainstalowanie pakietu **NI-DAQmx** (wersja zgodna z urządzeniem i LabVIEW).
2. Konfiguracja:
   - W NI MAX (Measurement & Automation Explorer) karta jest wykrywana jako „Dev1” lub podobna.
   - Można przeprowadzić testy kanałów oraz sprawdzić konfigurację linii wejścia/wyjścia.
3. Programowanie:
   - W LabVIEW używa się VI z palety DAQmx – np. DAQ Assistant, DAQmx Create Channel, DAQmx Read, DAQmx Write.
   - Przykład: odczyt napięcia z wejścia analogowego AI0: DAQmx Create Channel → DAQmx Start Task → DAQmx Read → wyświetlenie wyniku na wskaźniku.
   - Typowy typ kanału: Voltage, Analog Input, z zakresem np. 0–5 V.

[Tekst alternatywny. Zrzut ekranu. Diagram blokowy w środowisku LabVIEW przedstawiający konfigurację DAQ Assistant do odczytu wejść cyfrowych i ich przekazania na wyjścia karty NI USB-6009.]



Rysunek 11. Obsługa karty pomiarowej w środowisku LabVIEW.

Obsługa w języku C++:

1. Wymagania
   - Zainstalowany sterownik **NI-DAQmx** oraz jego C++ API (dostępne w pakiecie jako nagłówki i biblioteki statyczne).
   - Kompilator wspierający standard C++ (np. MSVC, GCC).
2. Programowanie:
   - Używa się funkcji z biblioteki NIDAQmxBase.h, (listing 1).
   - Wartości odczytywane są do zmiennych typu double, po czym można je przetworzyć lub zapisać do pliku.
   - Po zakończeniu należy wywołać DAQmxClearTask(taskHandle).

```
#include <iostream>
#include "NIDAQmx.h"
#include <thread>
#include <chrono>

int main()
{
    TaskHandle taskHandle = 0;
    uInt8 data[2] = { 0 };
    uInt8 outputData[3] = { 0 , 0, 0};
    float64 analogData = 0.0;
    int32 read = 0;
```

```
    while (true) {
        // 1. Ustaw port jako wejście i odczytaj dane
        DAQmxCreateTask("", &taskHandle);
        DAQmxCreateDIChan(taskHandle, "Dev1/port0/line3:4", "",
DAQmx_Val_ChanPerLine);
        DAQmxStartTask(taskHandle);

        DAQmxReadDigitalLines(taskHandle, 1, 10.0, DAQmx_Val_GroupByChannel,
data, 2, NULL, NULL, NULL);
        std::cout << "Odczyt z wejścia: " << (int)data[0] << (int)data[1] <<
std::endl;

        DAQmxStopTask(taskHandle);
        DAQmxClearTask(taskHandle);

        std::this_thread::sleep_for(std::chrono::milliseconds(50));

        // 2. Ustaw port jako wyjście i zmień stan
        DAQmxCreateTask("", &taskHandle);
        DAQmxCreateDOChan(taskHandle, "Dev1/port0/line0:2", "",
DAQmx_Val_ChanPerLine);
        DAQmxStartTask(taskHandle);

        uInt8 output = data[0];  // np. kopiujemy stan wejścia na wyjście
        DAQmxWriteDigitalLines(taskHandle, 1, 1, 10.0,
DAQmx_Val_GroupByChannel, outputData, NULL, NULL);

        DAQmxStopTask(taskHandle);
        DAQmxClearTask(taskHandle);

        std::this_thread::sleep_for(std::chrono::milliseconds(50));


        // 3. Odczyt z wejścia analogowego (np. napięcia na ai0)
        DAQmxCreateTask("", &taskHandle);
        DAQmxCreateAIVoltageChan(taskHandle, "Dev1/ai0", "",
DAQmx_Val_Cfg_Default,
            0.0, 5.0, DAQmx_Val_Volts, NULL);
        DAQmxStartTask(taskHandle);
        DAQmxReadAnalogF64(taskHandle, 1, 10.0, DAQmx_Val_GroupByChannel,
            &analogData, 1, &read, NULL);
        DAQmxStopTask(taskHandle);
        DAQmxClearTask(taskHandle);
```


*Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23*

```
        std::cout << "[AI] Zmierzone napięcie: " << analogData << " V" <<
std::endl;

        std::cout << "---------------------------" << std::endl;
        std::this_thread::sleep_for(std::chrono::milliseconds(50));
    }

    return 0;
}
```

Listing 1. Program w C++ obsługujący odczyt wejść cyfrowych i analogowych oraz zapis wyjść cyfrowych.

6. Architektura i magistrale komputerów przemysłowych

Komputer przemysłowy (IPC) składa się z płyty głównej, pamięci, procesora, warstwy magistral rozszerzeń oraz interfejsów polowych, a jego zadaniem jest niezawodne wykonywanie kodu sterującego w środowisku o podwyższonych wymaganiach temperaturowych i wibracyjnych. W kolejnych akapitach omawiamy ewolucję tej architektury, od klasycznego podziału „mostek północny / południowy" po współczesne system-on-chip i magistrale PCIe 6.0, uwzględniając formy montażu popularne w automatyce.

[Tekst alternatywny. Fotografia. Cztery płyty główne ASUS w różnych formatach: EATX, ATX, micro-ATX i mini-ITX, ustawione obok siebie dla porównania wielkości.]

Rysunek 12. Płyty główne w różnych formatach budowy. [15]

Formaty płyt i modułów - lista historycznych standardów AT, ATX, BTX czy micro-ATX pozostaje przydatna do serwisowania starszych maszyn, lecz w nowych sterownikach królują:

Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki" nr FERS.01.05-IP.08-0234/23

- **Mini-ITX** (170 x 170 mm) - kompaktowe IPC z procesorami Alder Lake-N/P i maksymalnie 2 x PCIe Gen5 x16.
- **COM Express & COM-HPC** - moduły CPU montowane na płytkach carrier. Nowa rewizja 2.1 wspiera 65 lanes PCIe 5.0.
- **PC/104 & PC/104-Plus** - stosowane w pojazdach i wojskowych sterownikach, korzystają nadal z magistrali ISA lub PCI-104, zapewniając > 15-letnią dostępność komponentów (standard wciąż utrzymuje specyfikację 3.1).

Chipsety (od mostków do SoC) - klasycznej architekturze linia Front Side Bus (FSB) łączyła CPU z mostkiem północnym, a północny z południowym obsługiwał wolniejsze peryferia.

- Intel zastąpił FSB łączem QuickPath Interconnect (QPI), a w platformach Alder/Raptor/Ion Lake cały kontroler pamięci i część logiki południowej zintegrowano w CPU; komunikację z chipsetem przejął DMI 4.0 (x8 PCIe 4.0).
- AMD porzucił HyperTransport na rzecz Infinity Fabric 3.0, łącząc rdzenie i kontrolery IO bezpośrednio w układach EPYC 9005.

Rezultat: znika fizyczny mostek północny, a południowy (Platform Controller Hub) ewoluuje w pojedynczy chip obsługujący USB 4, SATA i Ethernet 2.5/5 Gb/s.

Tabela 3. Magistrale wewnętrzne - przegląd i obecny status.

Magistrala	Przepustowość	Status w automatyce	Uwagi
ISA, EISA	8–16 Mb/s	utrzymywana w PC/104 i starszych sterownikach	prosta dekodowana I/O, brak hot-plug
PCI 32-bit 33 MHz	132 MB/s	schyłkowa, spotykana w retrofitach	DMA, Plug-and-Play
PCI-X 133/266	do 2.1 GB/s	jeszcze w serwerach pamięciowych	64-bit, równoległa
PCI Express Gen3/4/5	8/16/32 GT/s/lane	standard w IPC, Gen5 powszechny od 2024	szeregowa point-to-point
PCI Express Gen6	64 GT/s/lane (256 GB/s x16)	pierwsze backplane CompactPCI Serial	
CXL 2.0/3.0	zależna od PCIe 5/6	testowe serwery edge,	koherentny link CPU-GPU-DDR

		przygotowanie do pamięci- jako-usługi	
TSN-PCIe / TSN-Ethernet	1–10 Gb/s z gwarancją czasu	karty NIC TSN dla sterowania ruchem	deterministyczne okna czasowe

Kontroler przerwań, DMA i gorące wątki - w nowoczesnych CPU rolę kontrolera przerwań pełni Advanced Programmable Interrupt Controller (APIC) lub x2APIC, a DMA realizowany jest przez wielokanałowe Intel® VT-d / AMD IOMMU.

Rola magistrali w systemach sterowania:

- Magistrale CPU ↔ RAM (FSB/QPI/IF) determinują czas cyklu sterowania w miękkim PLC.
- Magistrale rozszerzeń (PCIe) przenoszą strumienie z kart DAQ, GPU lub TSN NIC.
- Szyny niskopoziomowe (SPI, I²C) konfigurują czujniki, a Ethernet-TSN łączy moduły I/O w czasie ≤ 1 µs.
-

Deterministyczne sterowanie wymaga, aby cała ścieżka od magistrali procesora, przez przełączniki PCIe, aż do Ethernetu TSN była zegarowo zsynchronizowana. Nowe backplane CompactPCI Serial z Gen5 i karty Kontron PCIE-0400-TSN zapewniają hardware-assist dla IEEE 802.1AS-Rev i precyzyjnego kształtowania ruchu.

Kluczowe definicje:

- **Front Side Bus (FSB)** - równoległa magistrala CPU-Northbridge (zastąpiona przez QPI/DMI/IF).
- **PCI Express (PCIe)** - szeregowe, punkt-do-punkt łącze modułowe; przepustowość rośnie co generację x2.
- **CXL** - nadbudowa nad PCIe zapewniająca koherencję pamięci między CPU a akceleratorami.
- **TSN** - rozszerzenia Ethernet gwarantujące transmisję deterministyczną dla automatyki.
- **PC/104** - zestandaryzowany, sztaplowany format płytek z magistralą ISA/PCI dla środowisk ekstremalnych.

Architektura współczesnych komputerów przemysłowych łączy więc tradycyjną wytrzymałość (PC/104, COM modules) z najnowszymi magistralami PCIe 6.0 i CXL, dodając warstwę TSN dla synchronizacji w sieci OT. Zrozumienie tych warstw jest

niezbędne przed przejściem do projektowania oprogramowania czasu rzeczywistego i diagnostyki magistral.

7. Systemy operacyjne w komputerowych systemach sterowania

System operacyjny - jest oprogramowaniem pośredniczącym pomiędzy sprzętem a aplikacjami sterującymi; zarządza pamięcią, czasem procesora, urządzeniami we/wy i komunikacją, tworząc środowisko, w którym inżynier może uruchamiać kod regulacji i wizualizacji. Dla układów automatyki kluczowe są **przewidywalność czasowa** i **niezawodność**, dlatego w praktyce wykorzystuje się zarówno systemy ogólnego przeznaczenia z rozszerzeniami czasu rzeczywistego, jak i natywne systemy RTOS.

Klasyfikacja i architektura - w zależności od wymagań wyróżnia się **soft-** i **hard-real-time**. Pierwszy toleruje sporadyczne opóźnienia, drugi musi zachować nieprzekraczalne czasy reakcji. Pod względem budowy jąder spotykamy trzy podejścia:

- **Monolityczne** (Linux, FreeBSD) - wysoka wydajność, ale ryzyko destabilizacji całego systemu przy błędzie sterownika.
- **Mikrojądro** (QNX, seL4) - minimalny kod w przestrzeni jądra, usługi plików i sieci w procesach użytkownika; łatwiejsza weryfikacja formalna, nieco większy narzut komunikacyjny.
- **Hybrydowe** (Windows NT-rodzina) - kompromis łączący modułowość z wydajnością.

Systemy ogólnego przeznaczenia z opcją RT - Windows IoT pozostaje domyślną platformą dla wielu stacji SCADA. Najnowsza gałąź Windows 11 IoT Enterprise LTSC 2024 oferuje dziesięcioletnie wsparcie, lockdown aplikacji i usprawnione mechanizmy Device Update Center. Aby uzyskać twarde czasy reakcji, producenci dodają nakładki, np. IntervalZero RTX64 4.5, która tworzy równoległe jądro RT obok Windows i pozwala uruchamiać wątki z jitterem mniejszym niż 10 µs. W świecie open-source Linux stał się pełnoprawnym RTOS dzięki włączeniu zestawu łat PREEMPT_RT do gałęzi głównej (kernel 6.6) - typowe najgorsze opóźnienie tasku wysokiego priorytetu na sprzęcie x86 to obecnie 40–60 µs bez specjalnego strojenia. Dla potrzeb twardszych gwarancji stosuje się nadal Xenomai 4 (patch Adeos) lub RT-Linux, które przechwytują przerwania przed jądrem ogólnym.

Natywne systemy czasu rzeczywistego:

- **QNX Neutrino** - mikrojądro ≈ 10 kB, deterministyczna komunikacja message-passing i harmonogram 32 poziomów priorytetu; wersja 8 rozszerzyła obsługę TLS 1.3 i CAN-FD ISO.
- **VxWorks 7** - certyfikaty SIL 3, kontenery OCI i szyfrowany system plików; używany w sterownikach ruchu i satelitach.
- **Zephyr 4.2** - lekki RTOS pod MCU (ARM, RISC-V, RX), wyposażony w Bluetooth LE 5.4 i obsługę USB Video Class.

Mechanizmy krytyczne dla automatyki - pamięć wirtualna poprawia wykorzystanie RAM, lecz stronicowanie na żądanie wprowadza nieprzewidywalne opóźnienia - w systemach hard-RT funkcję tę ogranicza się lub całkiem wyłącza. Planowanie odbywa się według priorytetów. W Windows NT dostępne są 32 poziomy, z czego 16 przeznaczono na wątki real-time, a w jądrze Linux 2.6+ aż 140 (1-99 RT, 100-139 FIFO/RR dla zadań zwykłych). Wywłaszczanie i przełączanie kontekstu muszą być krótsze od cyklu sterowania - współczesne RTOSy osiągają < 2 µs na ARM-Cortex-A53.
Synchronizacja odbywa się przez semafory, kolejki, zdarzenia; mikrojądro QNX dodatkowo udostępnia pełnomocników (proxy) i natywny mechanizm message-passing, który eliminuje potrzebę współdzielonej pamięci i zapobiega odwrotnemu dziedziczeniu priorytetu.

Trendy 2023–2025:

- **Konteneryzacja Edge** - Siemens Industrial Edge 2.0 czy CODESYS Virtual Control potrafią uruchamiać sterownik PLC jako kontener OCI na Linux PREEMPT_RT, a sched-policies cgroups v2 mapują się na priorytety RT.
- **Koherencja czasu w sieci** - TSN oraz PTP IEEE 1588 integrowane są bezpośrednio w sterowniki (Linux phc2sys, Windows Precision Time Protocol).
- **Bezpieczeństwo** - RTOSy uzyskują certyfikaty IEC 62443-4-2, wprowadzają ASLR i podpisy binarne; Windows 11 IoT posiada Secure Boot DMA Protection.

Kluczowe definicje:

- **RTOS** - system operacyjny gwarantujący deterministyczny czas reakcji.
- **Hard/Soft RT** - odpowiednio: czas graniczny niepodlegający przekroczeniu/czas, którego przekroczenie jest tolerowane.
- **PREEMPT_RT** - zestaw poprawek nadających jądrze Linux właściwości czasu rzeczywistego.
- **IntervalZero RTX64** - rozszerzenie do Windows tworzące równoległe jądro RT korzystające z tych samych zasobów sprzętowych.
- **Microkernel** - jądro zawierające jedynie podstawy zarządzania zadaniami i IPC; pozostałe usługi działają jako procesy użytkownika.

Nowoczesne systemy operacyjne dla automatyki łączą dojrzałe mechanizmy wielozadaniowe z coraz lepszą deterministyką (Linux 6.x RT, Windows RTX64) i narzędziami konteneryzacji, ułatwiając budowę rozproszonych, bezpiecznych platform sterowania w erze Przemysłu 4.0.

8. Komunikacja i wymiana danych w komputerowych systemach sterowania

Niezawodne sterowanie wymaga płynnego przepływu informacji od sensorów i sterowników do aplikacji SCADA, MES i chmury. Sposoby przekazywania danych można pogrupować według zasięgu.

Metody lokalne - pliki tekstowe lub CSV, biblioteki LIB/DLL ładowane dynamicznie, historyczne mechanizmy DDE / OLE / COM oraz specyficzne interfejsy producentów służą wymianie danych wewnątrz jednego komputera lub między procesami działającymi w tym samym systemie. DDE i NetDDE są dziś praktycznie nieużywane, zastąpione przez COM i nowsze .NET-owe interfejsy.

Połączenia zdalne - podstawowe protokoły sieciowe to UDP / TCP/IP, FTP, HTTP i gniazda WinSock, którym towarzyszą niskopoziomowe zdalne wywołania procedur (RPC). Współcześnie do transmisji struktur binarnych i strumieni telemetrycznych częściej wybiera się gRPC lub WebSocket.

Model komponentów - technologia COM/DCOM umożliwiła obiektowe wywołania metod między aplikacjami, a rozszerzenie DCOM dodało przeźroczystość sieciową. Rozwiązania .NET Remoting ustępują miejsca usługom REST i gRPC, które są prostsze w konfiguracji zapór sieciowych i łatwiejsze do konteneryzacji.

Klasyczny standard OPC - OPC Classic powstał, by ujednolicić dostęp do urządzeń poprzez wspólny interfejs klient-serwer. Jego zaletą jest odciążenie urządzeń i likwidacja konieczności pisania dedykowanych sterowników.

- **OPC DA 3.0** - dostęp do bieżących wartości tagów, tryb synchroniczny lub asynchroniczny, transport COM/DCOM lub SOAP.
- **OPC HDA** - zapytania o dane historyczne oraz proste agregacje.
- **OPC A&E** - obsługa alarmów i zdarzeń z warunkami i potwierdzeniami.
- **OPC Security** - definiuje role i uprawnienia na poziomie obiektów, jednak w wersjach Classic zależy od DCOM.

Utrzymywanie DCOM we współczesnych sieciach OT-IT jest kłopotliwe (firewalle, wersje systemu), dlatego dziś zaleca się migrację do OPC UA.
OPC Unified Architecture - łączy funkcje DA, HDA i A&E w jednym modelu usług i porzuca DCOM na rzecz natywnych ramek TCP lub HTTPS. Model obiektowy opisuje zmienne, metody i relacje w postaci węzłów, a wbudowane podpisy oraz szyfrowanie zapewniają poufność i integralność. Aktualne rozszerzenia:

- PubSub - komunikaty UADP lub JSON wysyłane wieloodbiorczo przez UDP, MQTT lub AMQP, idealne do IIoT.
- OPC UA FX - profil do deterministycznego sterowania urządzeniami po Ethernet TSN, synchronizacja < 1 µs.
- Reverse Connect - połączenia inicjowane z wysokostronnych sieci OT do stref DMZ, upraszczające konfigurację zapór.

Pliki, XML i usługi sieciowe - XML, SOAP i język WSDL opisują interfejsy tzw. usług sieciowych, umożliwiając integrację heterogenicznych aplikacji. W praktyce bogate dokumenty SOAP zostały w wielu zastosowaniach wyparte przez lżejsze REST + JSON oraz schematy OPC UA Discovery.

Kluczowe definicje:

- **RPC** - wywołanie procedury w innym procesie lub na innym komputerze bez jawnej obsługi kanału transportowego.
- **COM/DCOM** - binarny model komponentów w Windows; DCOM rozszerza go o transmisję sieciową.
- **SOAP / WSDL** - XML-owy protokół zdalnych wywołań i język opisu usług.
- **OPC Classic** - zestaw specyfikacji DA, HDA, A&E opartych na COM/DCOM.
- **OPC UA** - niezależna od platformy architektura usługowa z wbudowanym modelem semantycznym i zabezpieczeniami.
- **PubSub** - sposób transferu danych w OPC UA polegający na publikacji-subskrypcji, bez sesji klient-serwer.

Dzisiejsze systemy sterowania łączą więc klasyczne biblioteki i pliki konfiguracji z bezpieczną komunikacją OPC UA, MQTT i gRPC, a deterministyczny Ethernet TSN oraz profil UA FX pozwalają przenieść wymianę danych w czasie poniżej 1 ms z urządzeń polowych aż do chmury analitycznej.

8.1. Tworzenie serwera OPC UA z wykorzystaniem języka Python

Poniższy opis przedstawia sposób zbudowania przykładowego środowiska OPC UA w Pythonie z wykorzystaniem biblioteki opcua. Zaprezentowano procedurę

uruchomienia serwera, dodania zmiennej do przestrzeni adresowej oraz dynamicznej modyfikacji jej wartości.

W pierwszym etapie instaluje się wymagane pakiety czyli bibliotekę python-opcua oraz narzędzie wiersza poleceń opcua-client korzystając z menedżera pip w konsoli CMD: pip install opcua opcua-client. Listing 1 demonstruje aplikację kliencką odczytującą i wyświetlającą wartości udostępniane w sieci, natomiast Listing 2 przedstawia minimalistyczny serwer publikujący zmienną i aktualizujący jej wartość w czasie rzeczywistym.

```
import sys
sys.path.insert(0, "..")
import time
from opcua import ua, Server

if __name__ == "__main__":
    server = Server()
    server.set_endpoint("opc.tcp://127.0.0.1:5840/freeopcua/server/")

    uri = "http://examples.freeopcua.github.io"
    idx = server.register_namespace(uri)

    objects = server.get_objects_node()

    myobj = objects.add_object(idx, "MyObject")
    myvar = myobj.add_variable(idx, "MyVariable", 6.7)
    myvar.set_writable()    # Set MyVariable to be writable by clients

    myvar2 = myobj.add_variable(idx, "MyVariable2", 20)
    myvar2.set_writable()

    myvar3 = myobj.add_variable(idx, "MyVariable3", False)
    myvar3.set_writable()

    # starting!
    server.start()

    try:
        count = 0
        while True:
            time.sleep(1)
            count += 0.1

    finally:
```


Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

```
        server.stop()
```

Listing 2. Kod serwera OPC UA opcua_server.py.

Opis kroków działania:

1. Konfiguracja serwera

Tworzenie instancji serwera za pomocą Server() i ustawienie jego punktu końcowego na opc.tcp://127.0.0.1:5840/freeopcua/server/. Ten adres można zmienić według potrzeb.

2. Dodanie przestrzeni nazw

Rejestracja własnej przestrzeni nazw (http://examples.freeopcua.github.io) i uzyskujemy jej indeks (ID).

3. Dodanie obiektu i zmiennej

Tworzenie nowego obiektu o nazwie MyObject i dodanie do niego zmiennej MyVariable z początkową wartością 6.7. Ustawienie zmiennej jako zapisywalną (ang. writable), co pozwala klientom na jej modyfikację.

4. Uruchomienie serwera:

Uruchomienie serwera za pomocą server.start().

5. Dynamiczna zmiana wartości zmiennej:

W pętli while co sekundę zmieniana jest wartość zmiennej MyVariable. Wartość jest zwiększana o 0.1 i aktualizowana za pomocą metody set_value.

6. Zatrzymanie serwera:

W bloku finally serwer jest zatrzymywany, a połączenia i subskrypcje zamykane.

```
import sys
sys.path.insert(0, "..")
from opcua import Client
import time

if __name__ == "__main__":
    client = Client("opc.tcp://localhost:5840/freeopcua/server/")

    try:
        client.connect()

        root = client.get_root_node()
        print("Objects node is: ", root)
        print("Children of root are: ", root.get_children())

        myvar = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable"])
        myvar1 = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable2"])
        myvar2 = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable3"])
        obj = root.get_child(["0:Objects", "2:MyObject"])
```


Projekt „Dostosowanie kształcenia w Politechnice Świętokrzyskiej do potrzeb współczesnej gospodarki”
nr FERS.01.05-IP.08-0234/23

```
        print("myvar is: ", myvar)
        print("myvar is: ", myvar1)
        print("myvar is: ", myvar2)
        print("myobj is: ", obj)
        while True:
            print(myvar.get_value())
            print(myvar1.get_value())
            print(myvar2.get_value())
            time.sleep(1)

    finally:
        client.disconnect()
```

Listing 3. Kod klienta OPC UA opcua_client.py.

Opis kroków działania:

7. Dodanie ścieżki do modułów OPC UA

Importowanie modułów i dodanie ścieżki do nadrzędnego katalogu za pomocą sys.path.insert(0, ".."). To umożliwia dostęp do modułów z nadrzędnego katalogu.

8. Import wymaganych modułów

Importowanie klasy Client z biblioteki opcua do komunikacji z serwerem OPC UA. Importowanie modułu time do zarządzania opóźnieniami w pętli.

9. Inicjalizacja klienta OPC UA

Tworzenie instancji klienta za pomocą Client("opc.tcp://localhost:5840/freeopcua/server/"). Ustawienie adresu URL serwera OPC UA, z którym klient będzie się łączył.

10. Połączenie z serwerem

Wywołanie client.connect(), aby nawiązać połączenie z serwerem.

11. Uzyskanie dostępu do przestrzeni adresowej

Wywołanie client.get_root_node(), aby pobrać węzeł korzeniowy przestrzeni adresowej serwera OPC UA. Wyświetlenie węzła korzeniowego za pomocą print("Objects node is: ", root).

12. Przeglądanie dzieci węzła korzeniowego:

Wywołanie root.get_children() w celu uzyskania listy dzieci węzła korzeniowego. Wyświetlenie dzieci za pomocą print("Children of root are: ", root.get_children()).

13. Uzyskanie dostępu do zmiennych i obiektu

Pobranie węzłów zmiennych za pomocą ich ścieżek:

- myvar = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable"])
- myvar1 = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable2"])
- myvar2 = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable3"])

Pobranie węzła obiektu za pomocą:

- obj = root.get_child(["0:Objects", "2:MyObject"])
- wyświetlenie informacji o zmiennych i obiekcie za pomocą funkcji print.

14. Odczytywanie wartości zmiennych w pętli:

W nieskończonej pętli while True:

- Odczytywanie wartości zmiennych myvar, myvar1 i myvar2 za pomocą metody get_value().
- Wyświetlanie ich wartości za pomocą print.
- Wprowadzenie jednosekundowego opóźnienia między odczytami za pomocą time.sleep(1).

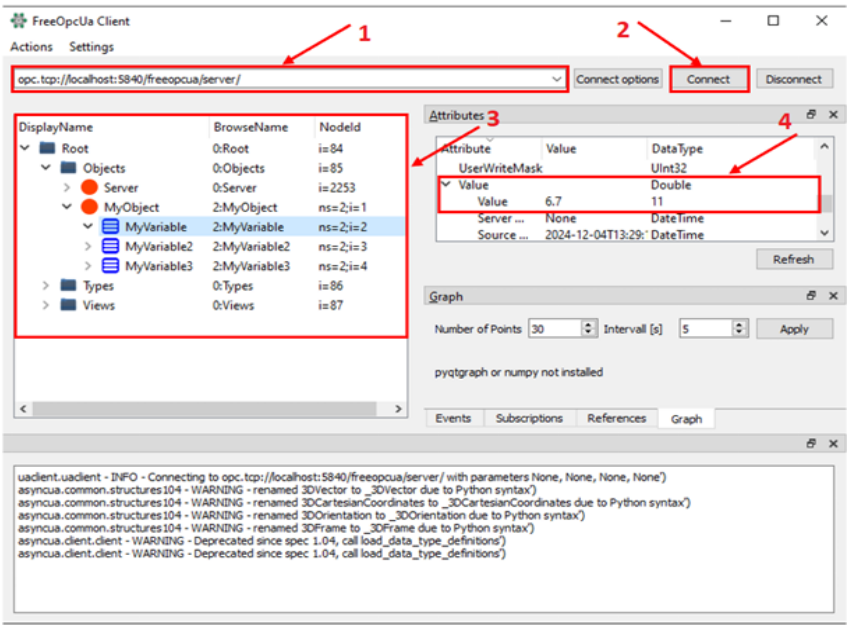
15. Zakończenie pracy klienta:

W bloku finally wywołanie client.disconnect(), aby zakończyć połączenie z serwerem i oczyścić zasoby.

W obu przypadkach należy w konsoli CMD uruchomić skrypt:

- Dla serwera będzie to: python server_opc.py,
- Dla klienta będzie to: python client_opc.py.
- Dodatkowo można skorzystać ze gotowego klienta FreeOpcUa Client dostępnego w bibliotece wpisując w terminal opcua-client (podgląd programu na rysunku 13).

[Tekst alternatywny. Zrzut ekranu. Widok programu FreeOpcUA Client: panel połączenia z serwerem OPC UA, lista obiektów i zmiennych oraz okno atrybutów z wartościami i typami danych.]



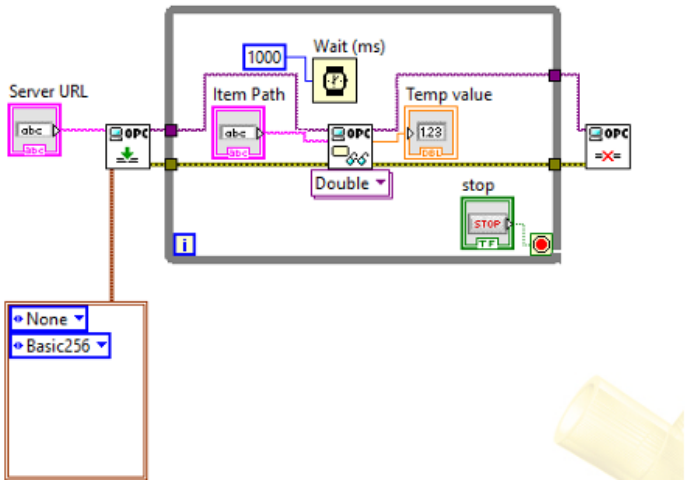
Rysunek 13. Program FreeOpcUA Client.

8.2. Tworzenie klienta OPC UA z wykorzystaniem programu LabVIEW

LabVIEW umożliwia tworzenie klienta OPC UA, który pozwala na odczyt i zapis danych udostępnianych przez serwery OPC UA. Dzięki graficznemu środowisku i modułowi OPC UA Toolkit, proces ten jest intuicyjny i nie wymaga głębokiej znajomości protokołu. Tworzenie klienta OPC UA w LabVIEW obejmuje następujące kroki:

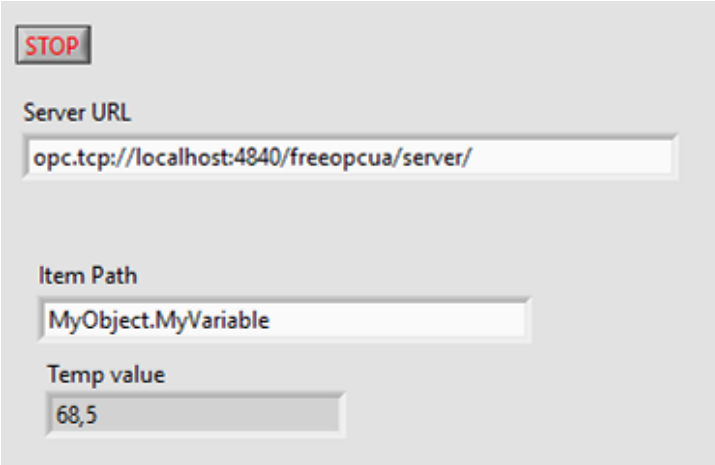
1. Konfiguracja połączenia z serwerem OPC UA
   - Należy skorzystać z palety OPC UA Client Toolkit.
   - Użyć bloku Open Connection.vi, aby nawiązać połączenie z serwerem.
   - Wprowadzenie punktu końcowego serwera (npopc.tcp://localhost:5840/freeopcua/server/) oraz opcjonalne dane uwierzytelniające.
2. Przeglądanie przestrzeni adresowej serwera
   - Wykorzystać funkcję Browse.vi, aby odczytać strukturę przestrzeni adresowej serwera OPC UA.
   - Identyfikacja zmiennych i węzłów, które będą monitorowane lub modyfikowane.
3. Odczyt danych z serwera
   - Należy skożystać z bloku Read.vi, aby pobierać wartości wybranych zmiennych z serwera.
   - Można skonfigurować odczyt cykliczny (np. w pętli) w celu monitorowania zmieniających się danych w czasie rzeczywistym.
4. Zapis danych na serwerze
   - Wykorzystanie bloku Write.vi do zmian wartości zmiennych zapisywalnych (ang. writable) na serwerze OPC UA.
   - Upewnienie się, że klient ma odpowiednie uprawnienia do modyfikacji danych.
5. Zamykanie połączenia
   - Po zakończeniu pracy z klientem należy użyć bloku Close Connection.vi, aby zamknąć połączenie z serwerem i zwolnić zasoby.

[Tekst alternatywny. Zrzut ekranu. Diagram blokowy w środowisku LabVIEW przedstawiający implementację klienta OPC UA: konfigurację adresu serwera, ścieżki elementu, odczytu wartości i przycisku zatrzymania.]



Rysunek 14. Implementacja klienta komunikacji OPC UA.

[Tekst alternatywny. Zrzut ekranu. Panel użytkownika w środowisku LabVIEW pokazujący uruchomienie klienta OPC UA: pole adresu serwera, ścieżkę zmiennej oraz aktualną wartość temperatury.]



Rysunek 15. Panel użytkownika komunikacji OPC UA.

9. Wizualizacja procesów przemysłowych

Nowoczesna wizualizacja to nie tylko ekrany synoptyczne w sterowni. Jej zadaniem jest ułatwienie operatorowi szybkiej, prawidłowej reakcji na zdarzenia, a inżynierowi – dostęp do danych, które wspierają decyzje produkcyjne i biznesowe. System wizualizacji, zwykle część pakietu **SCADA/HMI**, łączy cztery funkcje: prezentację bieżących wartości, alarmowanie, gromadzenie historii oraz interakcję człowieka z maszyną.

Podstawowe pojęcia:

- **SCADA** (Supervisory Control and Data Acquisition) - oprogramowanie nadzorujące proces, zbierające i archiwizujące dane.
- **HMI** (Human–Machine Interface) - warstwa graficzna, przez którą operator obserwuje i steruje instalacją.
- **Dashboard** - ekran skupiający najważniejsze wskaźniki (KPI, OEE) prezentowane w formie wykresów lub kafelków.
- **Alarm** - zdarzenie przekroczenia warunku technologicznego; system alarmowy definiuje priorytety, opóźnienia i procedury potwierdzeń.
- **Historian** - zoptymalizowana baza danych czasu rzeczywistego przechowująca zmienne procesowe w wysokiej rozdzielczości.

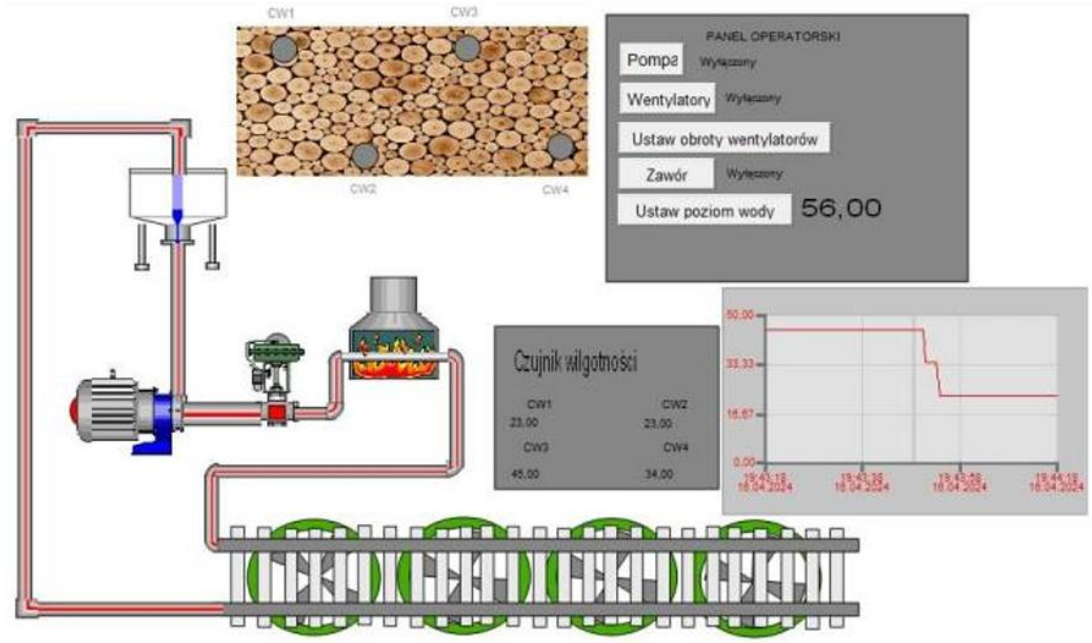
Kluczowe wymagania wobec wizualizacji - w projektach przemysłowych obowiązuje dziś filozofia High-Performance HMI (norma ISA-101). Kolorystyka ograniczona do odcieni szarości z akcentami dla stanów alarmowych, minimalizm graficzny i kontekstowe detale wyświetlane dopiero po powiększeniu widoku. Dzięki temu operator skupia się na odchyleniach zamiast na ozdobnikach graficznych. Ekrany muszą reagować z opóźnieniem poniżej 500 ms, a system alarmów gwarantować rejestrację i dystrybucję w czasie < 1 s.

iFIX w roli centralnej platformy wizualizacji - Proficy iFIX firmy GE Vernova (wcześniej GE Digital). Wersja 2024 wprowadziła m.in. do 80 % szybsze przetwarzanie alarmów, integrację z Configuration Hub i licencję, która od razu obejmuje web-owe Operations Hub. Nadal dostępny jest tradycyjny klient desktop, ale użytkownik może też korzystać z widoków HTML5 na dowolnym urządzeniu mobilnym, bez wtyczek i bezpłatnie w ramach jednej licencji. W nadchodzącej linii Proficy 2025 producent zapowiada scentralizowane zarządzanie licencjami, dalszą poprawę wydajności i gotowe szablony ekranów ISA-101.

Architektura iFIX obejmuje:

- iFIX SCADA Server - gromadzi dane, prowadzi alarmy, udostępnia je przez OPC UA oraz REST.
- Historian -kompresja blokowa, zapytania SQL-like; składowanie w AWS/Azure lub na serwerze on-premise.
- Clients - Workbench (projektant), lokalny Runtime, przeglądarka HTML5; każdy klient korzysta z jednoźródłowego repozytorium konfiguracji, co upraszcza wersjonowanie.
- Operations Hub - narzędzie „drag-and-drop” do tworzenia dashboardów KPI bez kodowania.

[Tekst alternatywny. Schemat. Przykładowy projekt suszarni drewna w programie iFix: przedstawiono układ wentylatorów, pompę, zawory, czujniki wilgotności, panel operatorski oraz wykres zmian parametrów procesu.]



Rysunek 16. Projekt suszarni drewna.

Przykładowy przepływ projektowania ekranu:

1. W Workbench definiujemy strukturę tagów i mapujemy je na źródła danych (OPC UA, MQTT, Modbus).
2. Tworzymy szablon ekranu i przypisujemy obiekty graficzne (pompę, zawór) do tagów procesowych.
3. Konfigurujemy reguły kolorów i alarmów według ISA-101 (np. żółty - stan ostrzegawczy, czerwony - krytyczny).
4. W Operations Hub przeciągamy widget Trend i łączymy go z danymi Historiana. Definiujemy zakres czasu i agregację.
5. Publikujemy projekt; użytkownik może wyświetlić ten sam ekran w desktop Runtime lub w przeglądarce na tablecie.

Integracja z OT-IT i chmurą - iFIX udostępnia REST API i klienta MQTT, co pozwala wysyłać zdarzenia do platform analitycznych. Wersja 2024 umożliwia przesyłanie strumieni danych do Proficy CSense albo zewnętrznego lakehouse (Snowflake, Databricks) bezpośrednio z Historiana. Obsługiwany jest też profil OPC UA PubSub dla wymiany danych z urządzeniami czasu rzeczywistego po TSN.

Trendy w wizualizacji:

- **Web-native** - przejście z ActiveX na HTML5/WebSocket (Ignition Perspective, WinCC Unified, iFIX HTML5).
- **Mobilność i Connected Worker** - interfejsy responsywne, tryb offline i powiadomienia push.
- **Ujednolicone repozytoria konfiguracji** - centralny Hub: jeden model danych dla SCADA, Historiana, raportów i CMMS.
- **Grafika wektorowa i tematy nocne** - skalowanie DPI, tryb dark poprawiający ergonomię w sterowni.
- **Analityka wizyjna** - integracja kamer AI do wykrywania anomalii i generowania alarmów.

Najczęstsze błędy projektowe - zbyt jaskrawe kolory, nadmiar animacji oraz brak hierarchii alarmowej prowadzą do “alarm flood” i zmęczenia operatora. Inny błąd to wyświetlanie zbyt wielu trendów bez kontekstu. Skuteczniejsze jest stosowanie sparklines i KPI z tolerancją ± %, które natychmiast wskazują odchylenie. Wizualizacja jest zatem ostatnią, ale kluczową warstwą łańcucha SCADA. Dzięki narzędziom takim jak iFIX 2024/25 możemy szybko budować skalowalne, web-owe panele zgodne z ISA-101, zintegrowane z Historianem, OPC UA PubSub i platformami analitycznymi, co przekłada się na wyższą czujność operatora i krótszy czas reakcji na incydenty.

10. Literatura

1. Niederliński A. (1984). Systemy komputerowe automatyki przemysłowej, Wydawnictwa Naukowo-Techniczne.
2. Mielczarek W. (1993). Szeregowe interfejsy cyfrowe, Helion.
3. Winiecki W. (1997). Organizacja komputerowych systemów pomiarowych, Oficyna Wydawnicza Politechniki Warszawskiej.
4. Grega W. (1999). Sterowanie cyfrowe w czasie rzeczywistym, Wydawnictwo AGH.
5. Bailey D., Wright E. (2003). Practical SCADA for Industry, Newnes (Elsevier).
6. Mahnke W., Leitner S.-H., Damm M. (2009). OPC Unified Architecture, Springer.
7. Travis J., Kring J. (2007). LabVIEW for Everyone: Graphical Programming Made Easy and Fun (3 ed.), Pearson.
8. Zurawski R. (red.) (2015). Industrial Communication Technology Handbook (2 ed.), CRC Press.
9. Meyer H., Fuchs F., Thiel K. (2009). Manufacturing Execution Systems (MES): Optimal Design, Planning and Deployment, McGraw-Hill.
10. Hohpe G., Woolf B. (2003). Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley.
11. Gilchrist A. (2016). Industry 4.0: The Industrial Internet of Things, Apress.
12. *Rozproszone systemy sterowania – wybór, integracja i migracja DCS-ów*. Automatykab2b.pl. [online] Dostępne: https://automatykab2b.pl/temat-miesiaca/43876-rozproszone-systemy-sterowania-wybor-integracja-i-migracja-dcs-ow [Dostęp: 18.07.2025].
13. *Eldar – dystrybutor komponentów i systemów automatyki przemysłowej*. Eldar.biz. [online] Dostępne: https://www.eldar.biz/ [Dostęp: 18.07.2025].
14. *National Instruments – Strona główna producenta systemów pomiarowych i sterowania*. Ni.com. [online] Dostępne: https://www.ni.com/ [Dostęp: 18.07.2025].
15. *Ini Perbedaan ATX, MicroATX dan Mini-ITX*. Medcom.id. [online] Dostępne: https://www.medcom.id/teknologi/tips-trik/aNrQg61K-ini-perbedaan-atx-microatx-dan-mini-itx [Dostęp: 18.07.2025].