p

2018

Oprogramowanie systemów pomiarowych

OPROGRAMOWANIE WIRTUALNYCH PRZYRZĄDÓW I SYSTEMÓW POMIAROWYCH

ANDRZEJ MAJKOWSKI, REMIGIUSZ RAK

OPROGRAMOWANIE , LABWINDOWS, LABVIEW, SCPI,SCADA

Oprogramowanie systemów pomiarowych pełni równie istotną rolę co wykorzystane w systemach przyrządy pomiarowe. Umożliwia ono łatwe definiowanie funkcji systemu. W szczególności dotyczy to wirtualnych przyrządów pomiarowych, których istotę stanowi inteligentne połączenie sprzętu i oprogramowania. Środowiska programistyczne często mają formę graficznych języków programowania, jak np. LabVIEW firmy National Instruments. Jest to przyjazne dla użytkownika narzędzie programowe, które umożliwia tworzenie stosunkowo zaawansowanych aplikacji pomiarowych, z systemami SCADA włącznie. Bardzo ważną rolę pełni też język programowania autonomicznych przyrządów pomiarowych SCPI – Standard Commands for Programmable Instruments.

Spis treści

1 Wstęp ... 2

1.1 Oprogramowanie na poziomie rejestrowym ... 2
1.2 Oprogramowanie na poziomie sterownika ... 3
1.3 Oprogramowanie narzędziowe ... 3

2 Język SCPI ... 4

2.1 Model urządzenia SCPI ... 4
2.2 Składnia języka SCPI ... 5
2.3 Przykład konfiguracji podsystemu SCPI ... 7
2.4 System statusu SCPI ... 8

3 LabWindows/CVI ... 9
4 LabVIEW ... 10
5 System SCADA ... 11

5.1 Funkcje systemu SCADA: ... 11
5.2 Oferta rynkowa systemów SCADA: ... 11
5.3 Architektura systemów SCADA: ... 12
5.4 Prosty system SCADA ... 12
5.5 Przykłady architektury SCADA na bazie Intouch ... 14

5.5.1 Pojedynczy komputer (najprostsza architektura) ... 14
5.5.2 Architektura typu „klient” ... 14
5.5.3 Architektura bazująca na serwerze ... 14
5.5.4 Architektura sieciowa (NAD: Network Application Development) ... 15

5.6 Komponenty systemu SCADA ... 15

Komponenty sprzętowe systemu SCADA: ... 15
Komponenty programowe systemu SCADA: ... 15
5.6.1 Oprogramowanie SCADA: ... 16
5.6.2 Podstawowe elementy aplikacji SCADA: ... 16

5.7 System Wizcon ... 20
5.8 Zintegrowany system monitorowania i sterowania iFIX ... 22

5.8.1 Właściwości i architektura systemu iFIX ... 22

5.9 SCADA - podsumowanie ... 23

6 Ćwiczenia do modułu (rozwiązane problemy praktyczne - zadania, projekty) ... 25

6.1 Program do odszumiania sygnału napisany w środowisku LabWindows CVI ... 25
6.2 Prosty analizator widmowy napisany w środowisku LabView ... 31

7 Pytania kontrolne ... 40
8 Bibliografia ... 43

1 Wstęp

Zarówno oprogramowanie systemu pomiarowego, jak i oprogramowanie tworzące wirtualny przyrząd pomiarowy dzieli się na kilka odrębnych części (modułów). Należą do nich:

1. Graficzny interfejs użytkownika (obsługa przyrządu, zobrazowanie wyników),
2. Moduł obsługi zasobów sprzętowych (sterowanie/programowanie, zbieranie i wstępne przetwarzanie danych pomiarowych),
3. Moduł zaawansowanego przetwarzania i analizy sygnałów.

Dla wygody można uznać, że tradycyjnie, tylko dwa pierwsze spośród nich stanowią integralną część elementów „konstrukcyjnych" przyrządu. Ostatni, zawierający algorytmy cyfrowego przetwarzania i analizy sygnałów, jako niezwykle ważny i aktualny z punktu widzenia metrologa, został omówiony w odrębnym, dedykowanym jemu module pracy zatytułowanym: „Przetwarzanie i analiza sygnałów pomiarowych". Taki sposób podejścia nie zakłóci strategii zrównoważonego eksponowania cech wirtualnych przyrządów pomiarowych.

Nazwa „graficzny interfejs użytkownika", odnosi się do umieszczonego na ekranie monitora panelu obrazującego najczęściej wygląd płyty czołowej, która najbardziej wiernie odwzorowuje cechy danego przyrządu lub systemu. Mogą się tam znaleźć różnego typu obiekty jak: manipulatory, przyciski, przełączniki, potencjometry, a także pola tekstowe i odczytowe, okna graficzne i inne produkty wyobraźni projektanta. Obsługa wszystkich manipulatorów odbywa się z użyciem klawiatury lub najczęściej, myszy.

Bardzo istotny element oprogramowania stanowią funkcje realizujące sprzężenie modułu graficznego interfejsu użytkownika z pozostałymi modułami programu. Chodzi tu zarówno o moduły obsługi sprzętu jak również moduły wstępnego przetwarzania i analizy sygnałów. Obsługa modułów sprzętowych tradycyjnie może być zrealizowana na trzech poziomach: rejestrowym, rozkazowym (sterownika) oraz oprogramowania narzędziowego. Schemat blokowy strategii oprogramowania sprzętu, z podkreśleniem jego hierarchicznej struktury, przedstawiony jest na rys. 1.1.

Rys. 1.1 Hierarchiczna struktura oprogramowania do obsługi sprzętu

1.1 Oprogramowanie na poziomie rejestrowym

Oprogramowanie na poziomie rejestrowym wymaga znajomości wewnętrznej struktury urządzeń, (takich jak autonomiczny przyrząd pomiarowy, moduł VXI czy karta zbierania danych). Po to, aby oprogramować ich funkcje pomiarowe, należy wpisać do wskazanych rejestrów, odpowiednie i na ogół złożone kombinacje bitów. Jak już wspomniano wcześniej jest to zajęcie bardzo uciążliwe i wymaga stałego kontaktu z instrukcją obsługi.

1.2 Oprogramowanie na poziomie sterownika

Jednym z najbardziej popularnych składników współczesnej techniki pomiarowej, bardzo pomocnym w procesie tworzenia oprogramowania do obsługi różnego typu dedykowanych zasobów sprzętowych, jest sterownik. Jest to rodzaj, zwykle firmowego, łatwego w implementacji oprogramowania, zawierającego instrukcje realizujące pełny dostęp do złożonych funkcji pomiarowych konkretnego urządzenia (przyrząd autonomiczny, przyrząd modułowy, karta DAQ), za pośrednictwem intuicyjnych modułów API. Obecnie sterowniki stanowią nieodłączny element wszelkiego typu sprzętu, który daje się obsłużyć w sposób programowy.
W roku 1998, kilka przodujących firm w dziedzinie produkcji aparatury kontrolno-pomiarowej powołało do życia organizację znaną pod nazwą „Interchangeable Virtual Instrument Foundation: IVIF". Powołana ona została, między innymi, do stanowienia standardów w zakresie sterowników programowych.

1.3 Oprogramowanie narzędziowe

W zakresie oprogramowania narzędziowego wspomagającego projektowanie wirtualnych przyrządów pomiarowych oferta wyspecjalizowanych firm światowych jest niezwykle bogata. Dochodzenie do tego stanu zajęło wiele lat. W pierwszym etapie, nastąpiło wyposażenie (wzbogacenie) klasycznych języków programowania w biblioteki procedur o specyfice pomiarowej (komunikacji z urządzeniami). W ten sposób rozszerzone zostały możliwości wybranych języków programowania jak: Basic, C lub Pascal. Szczególną pozycję zajmuje tu Basic, przyjęty za standardowy język programowania systemów pomiarowych przez firmę Hewlett-Packard. Wprowadzane procedury wzbogacały kolejne wersje Basica, tworząc nowy standard tego języka HPBasic i jeszcze nowsze, bazujące na nim wersje HPBasicPlus, IBasic (obiektowa, uproszczona wersja HPBasica). Zwieńczeniem tej drogi rozwoju Basica jest przystosowany do środowiska Windows Instrument Basic.

Odpowiednikiem "wzbogacania" Basica dla innych języków programowania jest wyposażanie ich w biblioteki do obsługi pomiarów. Dobrym przykładem są tu produkty IOLib488 i RTLib488 firmy I/Otech. IOLib488 jest biblioteką sterowników przyrządów I/Otech (Data Acquisition Instruments). Produkt oferowany jest w wersji źródłowej (C/Pascal/QBasic). RTLib488 jest z kolei graficzną biblioteką czasu rzeczywistego dla C i Pascala, odpowiadającą funkcjom graficznym zaimplementowanym w HPBasicPlus.

Kolejne rozwiązanie prezentowanego rodzaju oprogramowania stanowi ASYST. Twórcą oprogramowania jest Keithley - Asyst Software Technologies. Wywodzi się on z języka Forth i stanowi samodzielny język programowania systemów pomiarowych, z możliwością przyłączania modułów programowych C i Fortranu. Zasadniczy ciężar obsługi systemu przejmuje na siebie ASYST wspomagany jedynie procedurami zewnętrznymi użytkownika.

Wymienione, wybrane przykłady oprogramowania wskazują na różne drogi dojścia do tego samego celu: stworzenia wygodnych narzędzi realizacji systemów pomiarowych, wszystkie mają wspólną cechę - wymagają pisania programu.

Proces tworzenia programu można dalej uprościć stosując mechanizmy wywoływania i łączenia gotowych procedur metodą paneli funkcyjnych lub ikon. Taki dodatkowy mechanizm uwalnia programistę od wnikania w szczegóły stosowanych przez niego narzędzi, w skrajnym przypadku doprowadzając do tego, że stworzy on program np. w języku C, bez znajomości tego języka. Oczywiście doświadczony programista jest w stanie nawet tak przygotowany produkt świadomie modyfikować i „ulepszać".

Rewolucja w dziedzinie oprogramowania narzędziowego dokonała się, bez wątpienia z chwilą wprowadzenia systemu Windows oraz języków programowania obiektowego. Klasycznym przykładem oprogramowaniem narzędziowego zawierającego wszystkie nowoczesne mechanizmy wspomagania projektowania wirtualnych przyrządów pomiarowych w środowisku Windows są: LabWindows/CVI oraz LabVIEW firmy National Instruments.

Ważnym osiągnięciem jest opracowanie uniwersalnego języka programowania przyrządów pomiarowych o nazwie Standard Commands for Programmable Instruments: SCPI.

2 Język SCPI

Dzięki porozumieniu (konsorcjum) największych światowych producentów aparatury pomiarowo-kontrolnej powstał uniwersalny język programowania przyrządów pomiarowych znany pod skrótową nazwą:
SCPI (Standard Commands for Programmable Instruments).

Za pierwowzór do utworzenia tego języka posłużyły tzw. rozkazy uniwersalne i zapytania zdefiniowane w standardzie IEC-625.2. SCPI zawiera zestaw instrukcji, które niezależnie od rodzaju przyrządu i typu interfejsu, wysłane z kontrolera pozwalają na pełne zaprogramowanie jego pracy.

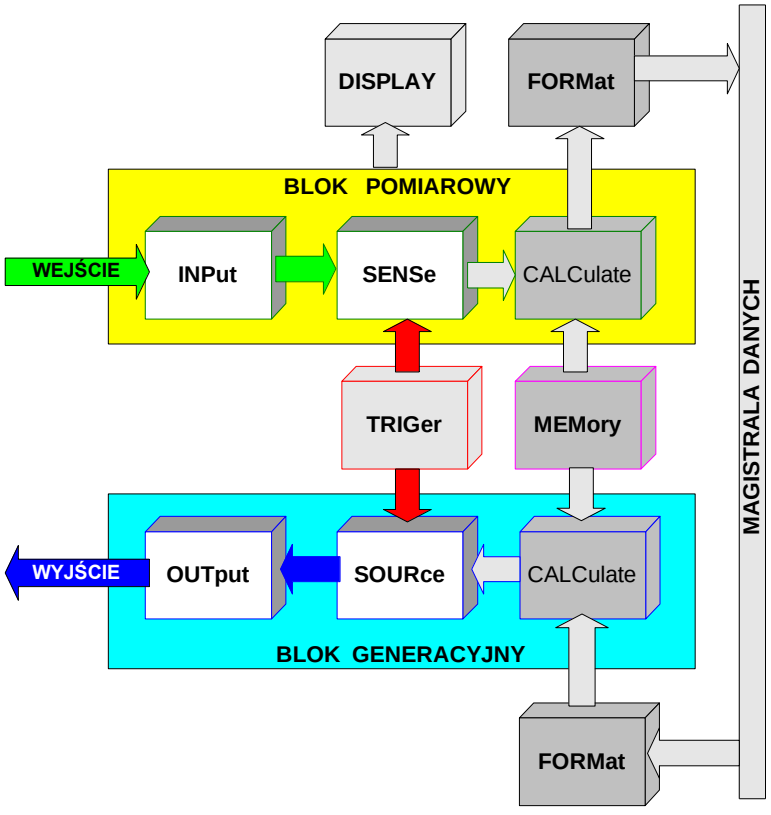
2.1 Model urządzenia SCPI

Do opracowania mnemoniki i składni tego języka użyty został oryginalny schemat blokowy uniwersalnego przyrządu pomiarowego, zwany modelem urządzenia SCPI (rys.2.2). Zawiera on dwa podstawowe bloki: pomiarowy i generacyjny. Wewnątrz bloku pomiarowego znajdują się:

INPut - moduł wejściowy definiujący impedancję wejściową, wzmocnienie i model wstępnego przetwarzania sygnału,
SENSe – moduł konwersji do postaci danych wewnętrznych (dane cyfrowe) definiujący rozdzielczość,
CALCulate – moduł (cyfrowego) przetwarzania danych.

Blok generacyjny zawiera:

CALCulate – moduł (cyfrowego) przetwarzania danych,
SOURce – moduł generacji definiujący źródło sygnału generowanego,
OUTput – moduł wyjściowy definiujący charakterystyki wyjściowe przyrządu (np. impedancja wyjściowa).



Rys. 2.2 Model urządzenia SCPI

Model określa ponadto dwa moduły wspólne dla obydwu bloków:

TRIGer – moduł ten definiuje model wyzwalania przyrządu (synchronizacja ze zdarzeniami wewnętrznymi lub zewnętrznymi),

MEMory – moduł pamięci (przechowywanie danych wewnętrznych).

Istnieją też dwa różne moduły o tej samej nazwie skojarzone z obydwoma blokami:

FORMat – moduł konwersji danych przeznaczony do współpracy z magistralą.

Polecenia języka SCPI, o strukturze hierarchicznej, pogrupowane są w podsystemy, które w większości są odpowiednikami poszczególnych modułów modelu SCPI (wskazuje na to również mnemonika poleceń nadrzędnych). Istnieje ponadto zbiór podsystemów nie objętych strukturą modelu SCPI. Polecenia tych podsystemów służą do ustalania zewnętrznych warunków pracy przyrządu (pomiaru). Jako przykład niech posłuży zestaw poleceń skierowanych do multimetru:

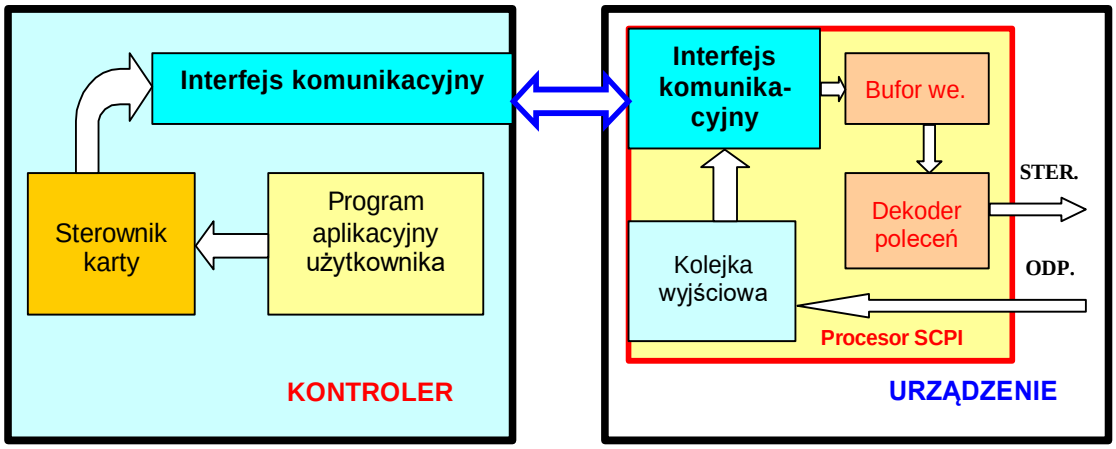
INPut:IMPedance 5e6; SENSe:VOLTage:DC 100, .001

Pochodzą one z dwu różnych podsystemów INPut i SENSe. Pierwszy ustala wartość modułu impedancji (rezystancji) wejściowej woltomierza na 5MΩ. Drugi ustala warunki pomiaru napięcia stałego na zakresie 100V, z rozdzielczością 0,001.

Drugi przykład pokazuje sposób pomiaru napięcia przemiennego na zakresie 20V, z rozdzielczością 0.005:

MEASure:VOLTage:AC? 20, 0.005

Urządzenia rozpoznają i interpretują wysłane przez kontroler rozkazy języka SCPI z pomocą tzw. procesora SCPI. Schemat współpracy kontrolera z urządzeniem przedstawiony jest na rysunku 2.3 jest on na tyle prosty, że nie wymaga specjalnego opisu.

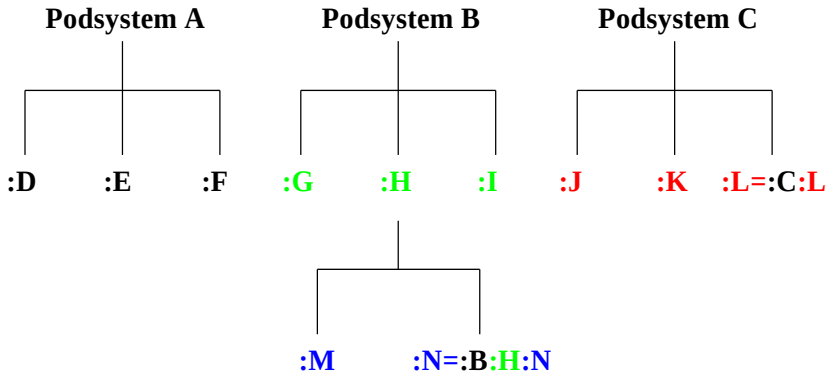


Rys. 2.3 Współpraca kontrolera z urządzeniem za pomocą SCPI

Polecenia SCPI zgrupowane w podsystemy odpowiadające komponentom modelu urządzenia SCPI (INPut, SENSe itd.) oraz dodatkowe podsystemy zwane systemowymi. Jak np.: SYSTem - ustawienie czasu, daty, sposobu informowania o błędach, STATus - informowanie o stanie urządzenia.

2.2 Składnia języka SCPI

Składnia języka SCPI ma strukturę hierarchiczną. Przykłady takiej struktury przedstawiono na rysunku 2.4.



Rys. 2.4. Obraz struktury hierarchicznej języka SCPI

Najwyżej w hierarchii położone są rozkazy poziomu korzenia (root level commands) zwane podsystemami np.: A, B, C. Dalej znajdują się rozkazy niższego poziomu (lower level commands), prowadzące do listków tej struktury zwanej inaczej drzewiastą. Obowiązkiem jest podanie pełnej ścieżki dostępu np.: N =:B :H :N
Separatory używane w SCPI mają następujące znaczenie:

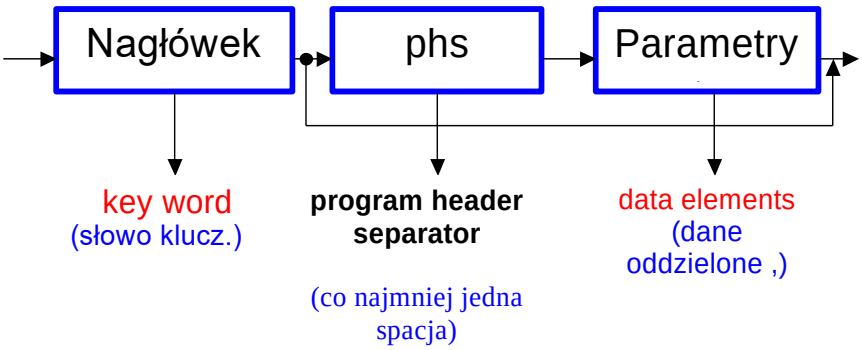
: - oddziela rozkazy niższego poziomu od wyższego

␣, tab - oddziela parametry od rozkazu

, - oddziela parametry (lista parametrów)

; - oddziela polecenia z różnych podsystemów.

Składnia informacji programowych przedstawiona jest na rysunku 2.5. Pojedynczy komunikat zwany jest jednostką programującą (program message unit).



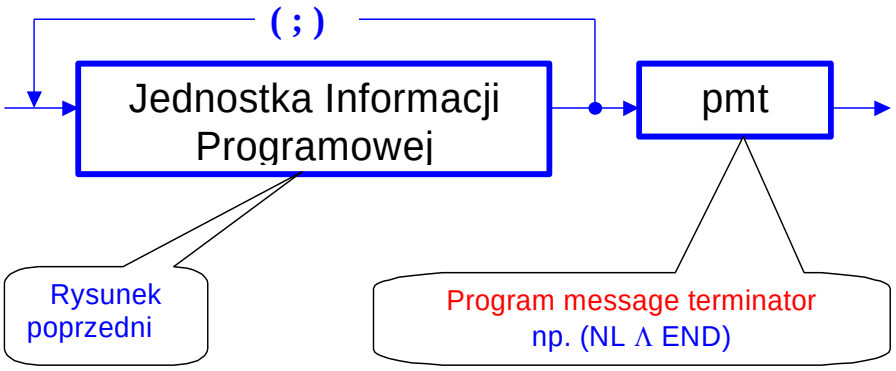
Rys. 2.5 Blok informacji programowej SCPI

Jako przykład niech posłuży polecenie:

:INPut : IMPedance 1e6

Programuje ono wartość impedancji wejściowej przyrządu na 1MΩ.

Pojedynczy komunikat może zawierać cały blok informacji programowej. Przykład takiego bloku zamieszczono na rysunku 2.6.



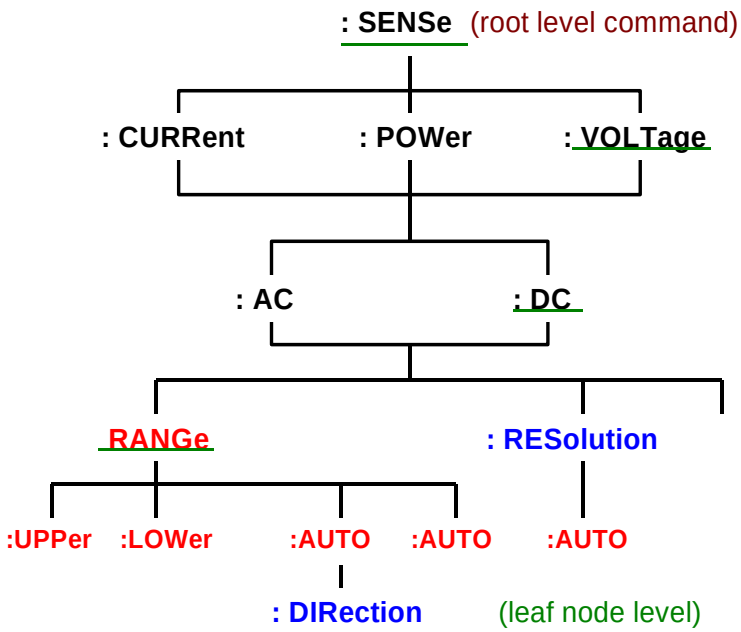
Rys. 2.6 Blok informacji programowej SCPI

Poniżej podano przykład bloku informacji programowej:

:INPut : IMPedance 1e6; :SENse :VOLTage :RANGe 200

2.3 Przykład konfiguracji podsystemu SCPI

Przykład konfiguracji konkretnego podsystemu pokazano na rysunku 2.7.



Rys. 2.7 Blok informacji programowej SCPI

Przykład użycia podsystemu SENSE podano poniżej:

SENSe :VOLTage :DC :RANGe :UPPer

Przy korzystaniu z języka SCPI warto pamiętać ponadto, że:

- Pierwszy rozkaz w nowym komunikacie programowym zawsze zawiera pełny nagłówek.
- Moduły informacji programowej (w tym samym komunikacie programowym), które nie zostały poprzedzone dwukropkiem są traktowane jako polecenia z tego samego poziomu co poprzedzające.

Przykład zastosowania drugiej uwagi zamieszczono poniżej:

SENSe :VOLTage :DC :RANGe :UPPer; LOWer;

Drugi sposób definiowania podsystemu polega na użyci tabeli. Przykład dla systemu SENSE zamieszczono w tabeli. 2.1

Tabela 2.1: Inny sposób definiowania podsystemu

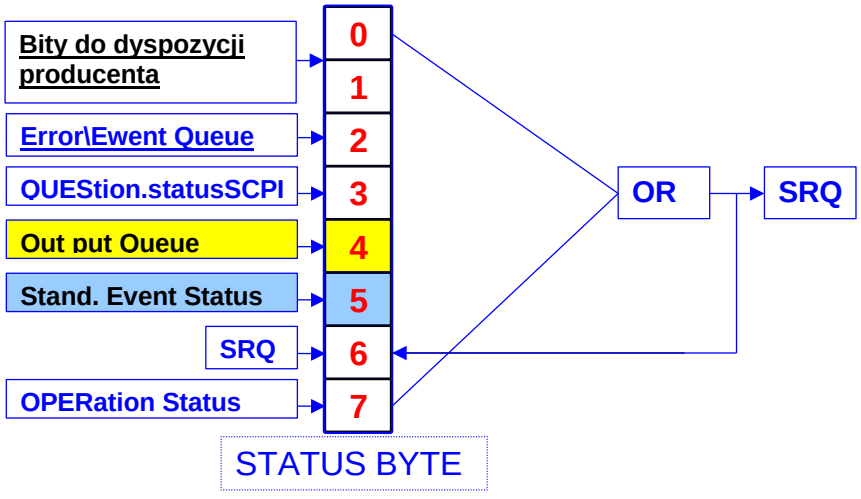
Rozkaz	Rodzaj paramatru	Wartości
:SENSe		
:CURRent		
:POWer		
:VOLTage		
:DC		
:AC		
:RANGE		
:UPPer	<numeryczny>	wartości
:LOWer	<numeryczny>	zależne od
:AUTO	<logiczny>	urządzeń
:DIRection	UP | DOWN |	
:RESolution	<numeryczny>	
:AUTO	<logiczny>	

Przy stosowaniu języka SCPI warto też wiedzieć, że:

- Kontroler wysyła komunikaty w dowolnej chwili;
- Urządzenie wysyła odpowiedź na życzenie;
- Jedynie komunikaty zakończone ? upoważniają urządzenie do umieszczania odpowiedzi w kolejce wyjściowej;
- Informację odczytuje się za pomocą Receive ( );
- Kolejka może się zapełniać: utrata poprzedniej informacji;
- Kolejkę zeruje DevClear( ).

2.4 System statusu SCPI

System statusu języka SCPI wzbogacony jest, w stosunku do swojego odpowiednika w IEC-625.2, o dodatkowe rejestry oraz dodatkowe bity rejestru statusu. Schemat blokowy tego systemu przedstawiono na rys. 2.8.



Rys. 2.8 Schemat blokowy systemu statusu SCPI

Interpretacja nowych elementów jest następująca:

Operation Status (Rejestr operacji) - rejestr zawierający informacje o wykonywanych przez przyrząd operacjach;

Questionable status (Rejestr niepewności pomiaru) - rejestr zawierający informacje o przyczynach błędnych pomiarów (np. przepełnienie);

Error\Ewent Queue (Kolejka błędnych zdarzeń).

3 LabWindows/CVI

Element rozszerzenia nazwy CVI (C for Virtual Instrumentation) stosunkowo wiernie oddaje istotę tego środowiska programistycznego – wskazuje na możliwość dostępu do kodu źródłowego w języku C. Wydaje się, że ta cecha wpłynęła na uzyskanie tak dużej popularności CVI w środowisku akademickim.

W istocie o sile LabWindows/CVI decydują funkcje biblioteczne osadzone w środowisku języka C. Biblioteki te zawierają elementy:

- dostępu do interfejsów pomiarowych (GPIB, GPIB-488.2, RS-232, VXI, DAQ),
- dostępu do przyrządów autonomicznych: Instrument Library,
- grafiki komunikacji z użytkownikiem (Graphics Library, User Interface Library, Formatting and I/O Library),
- analizy matematycznej (i cyfrowego przetwarzania sygnałów): Analysis Library, Advanced Analysis Library,
- dostępu do sieci komputerowej (TCP/IP, DataSocket, ActiveX),
- międzyprocesowej wymiany danych DDE (Dynamic Data Exchange).

O atrakcyjności programu stanowi przemyślana "grafika pomiarowa", pozwalająca tworzyć na ekranie i wkomponowywać w oprogramowanie panele sterujące sprzętem pomiarowym, wyglądające i funkcjonujące jak płyty czołowe klasycznych przyrządów laboratoryjnych.

Użytkownik w łatwy sposób może skomponować panel (określić jego rozmiary, ustawić na nim we właściwych miejscach elementy nastawcze i obserwacyjne, przydzielić im aktywność i nastawy, oraz przyporządkować obiekty i funkcje dostępne dla realizowanego oprogramowania), wprowadzić go do swej biblioteki i wkomponować w sposób interakcyjny w strukturę tworzonego programu. Program wywołujący panel może zarządzać aktywnością jego obiektów (blokować i otwierać dostęp użytkownika do elementów regulacyjnych), czytać stan nastaw, oraz sterować jego wskaźnikami. Program jest informowany o zmianie stanu panelu ("poruszenie" nastawy lub wprowadzenie nowej wartości danej) przez wywoływaną funkcję GetUserEvent. Funkcja ta może sprawdzać czy nastąpiła ingerencja operatora systemu czy nie i ją identyfikować, bądź też wstrzymywać działanie programu w oczekiwaniu na taką ingerencję.

Kolejną zaletą CVI jest wysoka jakość interakcyjnego narzędzia do tworzenia oprogramowania. Program ten jest bardzo funkcjonalny, przejrzysty i konsekwentny w użyciu. Użytkownik ma bezpośredni dostęp do pola edycyjnego programu i dodatkowego pola edycyjnego podprogramów. Z obydwu pól może kompilować i uruchamiać program bądź jego fragmenty. Edycję programu ułatwia oryginalna koncepcja tzw. paneli funkcyjnych pozwalających "pisać" program poprzez kolejne wywoływanie paneli definiujących wywoływane funkcje przepisywane natychmiast do pól edycyjnych. Panel funkcyjny jest podobny do panelu przyrządu- elementy nastawcze panelu funkcyjnego (sterowane myszą), określają wartości/nazwy parametrów aktualnie wywoływanej funkcji. Treść linii programu (w języku źródłowym), którą panel aktualnie zarządza, widoczna jest równocześnie z panelem na ekranie monitora. Pozwala to użytkownikowi kontrolować treść fragmentu programu źródłowego, który aktualny panel generuje, przed wprowadzeniem go do pól edycyjnych.

LabWindows/CVI pozwala definiować własne sterowniki przyrządów użytkownika, a dla realizowanych przez nie funkcji tworzyć stosowne panele funkcyjne. Podobne panele funkcyjne użytkownik może też zdefiniować dla swych własnych funkcji, niekoniecznie dotyczących sprzętu pomiarowego. Takie funkcje użytkownika mogą tworzyć jego prywatną bibliotekę i być używane na równi z oprogramowaniem bibliotecznym CVI.

Wreszcie, LabWindows/CVI daje możliwość realizacji dostępu do sieci komputerowej. Przez wykorzystanie specjalistycznego narzędzia o nazwie DataSocket, włączonego do architektury internetowej, daje możliwość publikacji wybranej informacji w sieci globalnej Internet, jak również tworzenia aplikacji rozproszonych w tej sieci. CVI zawiera też funkcje biblioteczne niższego poziomu do bezpośredniej obsługi protokołu TCP/IP, a więc transmisji danych w sieci komputerowej.

Istnieje cały szereg programów narzędziowych. Oprogramowanie LabWindows/CVI zostało tu wyróżnione głównie z uwagi na dużą popularność w środowisku akademickim i niewątpliwy sentyment do niego ze strony autorów.

4 LabVIEW

LabVIEW (firmy National Instruments), jest jednym z najpopularniejszych środowisk przeznaczonych do tworzenia szeroko aplikacji przeznaczonych do rejestracji, przetwarzania i prezentacji danych pomiarowych. Budowanie aplikacji nie jest oparte na klasycznym języku programowania (C , BASIC lub Pascal) a polega na kodowaniu zadanych czynności w sposób graficzny (język G). Projektowanie aplikacji odbywa się z użyciem diagramu blokowego. Środowisko zawiera bogate biblioteki graficznego interfejsu użytkownika, bogate biblioteki analiz (porównywalne z CVI)), obsługi interfejsów (IEC-625.x, RS-232, RS-485, VXI/PXI, GPIB), dostępu do sieci komputerowej (TCP/IP, ActiveX), dostępu do DDE. Umożliwia wielozadaniowość. Głównymi oknami środowiska LabVIEW są: okno płyty czołowej (Front Panel) oraz okno diagramu (Block Diagram). Pierwsze z nich służy do projektowania płyty czołowej (panelu) aplikacji, drugie natomiast jest oknem, w którym buduje się w sposób graficzny kod programu.

LabVIEW ułatwia integrację ze sprzętem pomiarowym, umożliwiając natychmiastową akwizycję danych i ich późniejszą wizualizację. Jest to możliwe dla niemalże wszystkich urządzeń I/O, niezależnie od tego czy wyprodukowane zostały przez NI czy innych producentów. W połączeniu z paradygmatem programowania graficznego, redukującego czas rozwoju aplikacji, LabVIEW 2017 wspiera tworzenie zaawansowanych projektów dzięki dopasowanym narzędziom – stając na czele dzisiejszych technologii.

Pozostałe, godne uwagi, pakiety oprogramowania to:
VEE (Hewlett-Packard/Agilent): Środowisko graficzno-tekstowe, sterowanie działaniem programu na zasadzie propagacji danych między obiektami, atrakcyjna możliwość symulacji działania przyrządu, brak możliwości dołączania programowych modułów zewnętrznych, obsługa interfejsów (IEC-625.x, RS-232, RS-485, GPIB, FirWire – IEEE 1394 i in.).
TestPoint (Kithley Instruments): Programowanie obiektowe, forma czterech okien projektowych (magazyn obiektów, panel płyty czołowej przyrządu, zbiór już użytych obiektów, algorytm zadania pomiarowego), przeciąganie obiektów metodą Drag&Drop, obsługa interfejsów (IEC-625.x, RS-232, RS-485, GPIB, możliwość symulacji rzeczywistych przyrządów, możliwość importowania modułów programowych, wielozadaniowość.
DasyLab (Dasytec): Graficzne programowanie aplikacji, obsługa interfejsów (IEC-625.x, RS-232, RS-485, VXI), dostęp do sieci komputerowej (TCP/IP), dostęp do DDE, wielozadaniowość.
Specjalne miejsce zajmują programy ukierunkowane na sterowanie i automatykę przemysłową: BridgVIEW (National Instruments), Lookout (National Instruments), GeniDAQ (Advantech).

Wspomniany wcześniej etap projektowania oprogramowania związany ze zobrazowaniem i rejestracją wyników jest ściśle uzależniony od wymagań użytkownika. Zasadniczo, ta część projektowania nie stwarza większych

trudności. Od strony sprzętowej niezbędny jest wybór monitora i odpowiedniej karty graficznej, drukarki, plotera, sterownika dysku czy też rejestratorów magnetycznych. Więcej pracy wymaga przygotowanie stosownego oprogramowania. Jednakże w przypadku wykorzystania zintegrowanego oprogramowania firmowego zadania projektowe zostają zminimalizowane lub wyeliminowane zupełnie.

5 System SCADA

Termin SCADA wywodzi się z określenia Supervisory Control And Data Acquisition. Termin ten oznacza zdalny system monitorowania i nadzoru procesów technologicznych. Systemy SCADA zastępują tablice synoptyczne stosowane w procesach sterowaniach i zawierają wiele dodatkowych funkcji. Systemy SCADA to złożone systemy komputerowe do zastosowań automatyki przemysłowej, zrealizowane na bazie tzw. oprogramowania wizualizacyjnego. System SCADA współpracuje zawsze z urządzeniami mającymi możliwości komunikacji cyfrowej (regulatory cyfrowe, przetworniki inteligentne) i połączonymi z nadzorowanym procesem za pośrednictwem interfejsu procesowego. SCADA zbiera informacje o procesie wyłącznie za pośrednictwem tych urządzeń! Pracuje zawsze w czasie rzeczywistym, część jego funkcji (np. alarmy) musi spełniać wymagania „Hard Real Time".

5.1 Funkcje systemu SCADA:

- Wizualizacja pracy urządzenia/instalacji,
- Uruchamianie i zatrzymywanie całej instalacji lub jej części,
- Monitorowanie i nadzór przebiegu procesu,
- Wykrywanie i sygnalizacja stanów awaryjnych,
- Archiwizacja danych procesowych,
- Zadawanie parametrów regulatorów pracujących w stopniu bezpośrednim,
- Sterowanie zdalne (automatyczne lub ręczne).

W typowej sytuacji różne grupy użytkowników mają dostęp do różnych funkcji systemu. Należy tu wyróżnić:

- technologów,
- automatyków,
- informatyków,
- kierownictwo wydziału produkcyjnego,
- zarząd.

Każda z tych grup powinna mieć dostęp do „swojej" części i nie powinna mieć dostępu do innych.

5.2 Oferta rynkowa systemów SCADA:

Oferta rynkowa sprzętu i oprogramowania SCADA jest bardzo rozbudowana. Systemy SCADA dostępne na rynku można podzielić na dwie zasadnicze grupy:

„**Niezależne**" – produkowane przez firmy nie produkujące innego sprzętu automatyki, dostosowane do współpracy ze sprzętem różnych producentów. Przykłady: WONDERWARE INTOUCH, CITECT.

„**Zależne**" - Sprzęt i oprogramowanie produkowane przez producentów sprzętu PLC (Programmable Logic Controller). Są one dedykowane do sprzętu określonej firmy, ale mają też możliwość współpracy z innym sprzętem. Przykłady: ProTool/Pro i WinCC (SIEMENS), Proficy HMI/SCADA-iFIX (GE FANUC), LabView (National Instruments).

5.3 Architektura systemów SCADA:

Każdy system SCADA jest budowany ściśle pod kątem wymagań określonego urządzenia, linii produkcyjnej lub zakładu, który ma go nadzorować. W praktyce stosowane są systemy bardzo proste (przeznaczone do współpracy np. z pojedynczym PLC) jak i rozbudowane, nadzorujące pracę całego wydziału produkcyjnego lub zakładu. Podobnie, jak każdy system informatyczny, system SCADA zawiera dwa zasadnicze komponenty: sprzętowy i programowy.

Komponenty sprzętowe SCADA:

Regulator cyfrowy lub urządzenie inteligentne: PLC, regulator PID, system akwizycji danych, przetwornik inteligentny, itp. (Zawsze jest to urządzenie mające bezpośredni kontakt ze sterowanym procesem i otoczeniem).

System komunikacyjny: Sieć lub system łączności bezprzewodowej,

Serwer do zbierania danych z procesu,

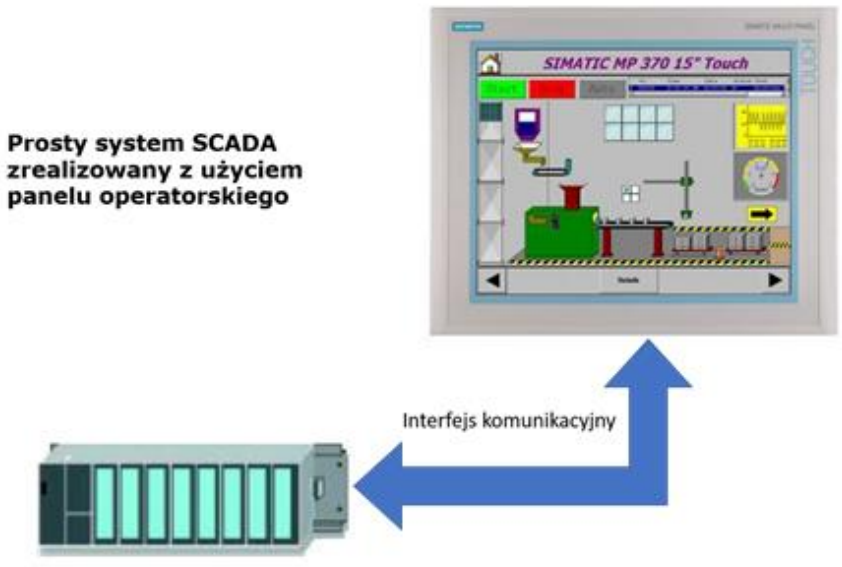
Stacja robocza realizująca pozostałe funkcje.

W prostszych i mniejszych systemach SCADA funkcje serwera i stacji klienckiej są zintegrowane w jednym urządzeniu: komputer, panel operatorski. W bardziej złożonych systemach SCADA funkcje serwera i stacji roboczej (klienckiej) są realizowane przez różne urządzenia. Dedykowany sprzęt SCADA budowany jest zgodnie z wymaganiami przemysłowymi.

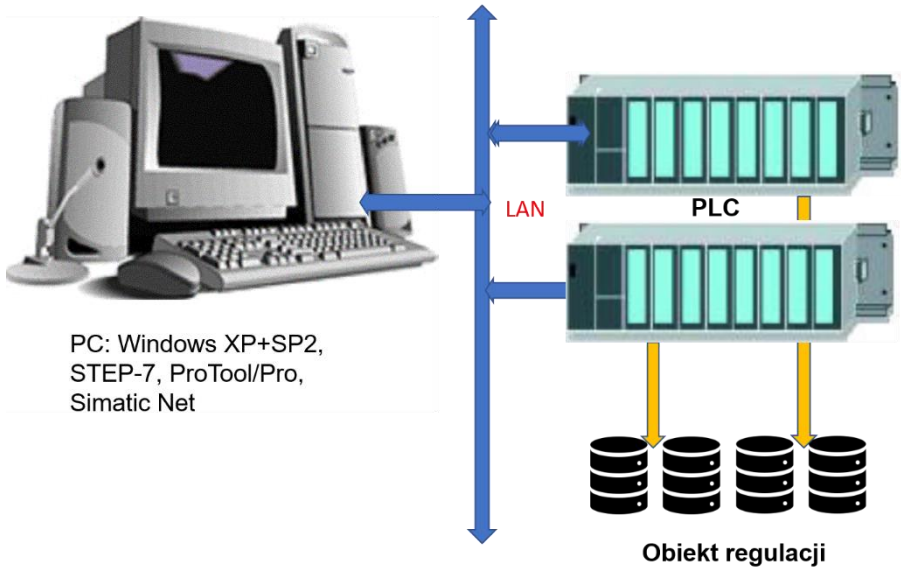
5.4 Prosty system SCADA

W najprostszym przypadku system SCADA zawiera (rys. 5.1 i 5.2):

- sterownik PLC, system akwizycji danych lub urządzenie inteligentne,
- system komunikacyjny: port komunikacyjny, sieć, łączność bezprzewodowa,
- komputer, który pełni następujące funkcje: stacja robocza, serwer komunikacyjny, serwer danych.



Rys. 5.1 Prosty system SCADA zrealizowany z użyciem panelu operatorskiego



Rys. 5.2 Prosty system SCADA zrealizowany z użyciem komputera PC

Rozbudowany system SCADA (rys. 5.3) zawiera 5 podstawowych poziomów:

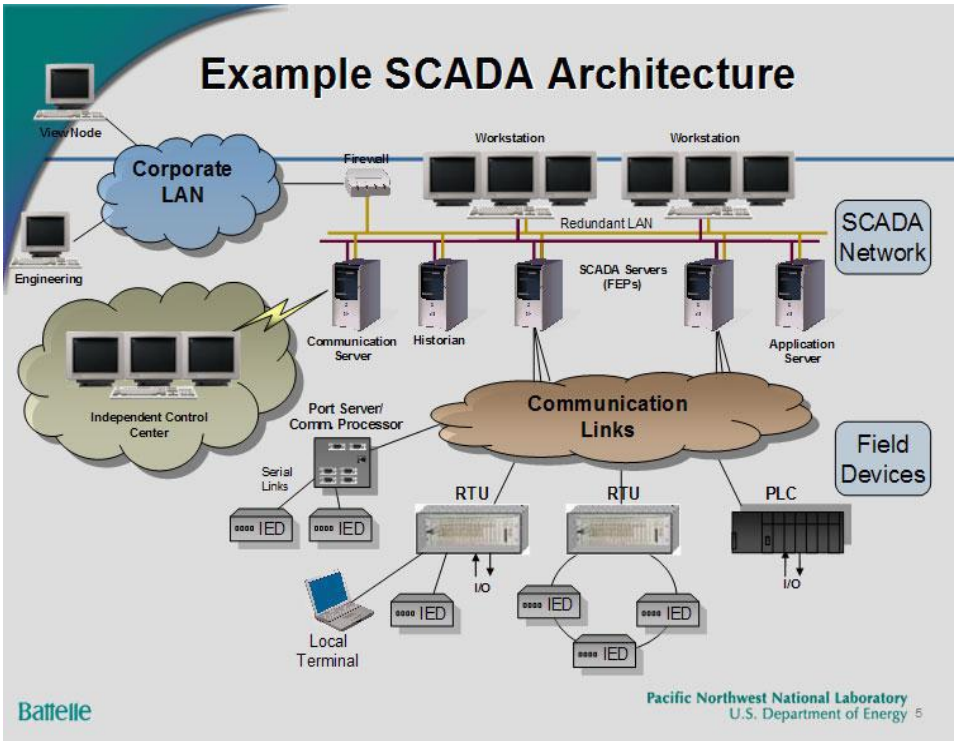
Poziom 1. Urządzenia polowe: przetworniki inteligentne, sterowniki PLC/PAC, regulatory, systemy akwizycji danych, itp. (urządzenia mające bezpośredni kontakt ze sterowanym procesem i możliwość komunikacji cyfrowej).

Poziom 2. Stacje akwizycji danych z urządzeń polowych (RTU – Remote Terminal Unit).

Poziom 3. System komunikacyjny zrealizowany z użyciem sieci przemysłowej lub urządzeń bezprzewodowych, itp.

Poziom 4. Stacje operatorskie z zaimplementowanymi aplikacjami użytkownika.

Poziom 5. System komputerowy na poziomie zarządu wydziału/zakładu.



Rys. 5.3 Przykład architektury SCADA (źródło: Pacific North west National Laboratory, USA)

5.5 Przykłady architektury SCADA na bazie Intouch

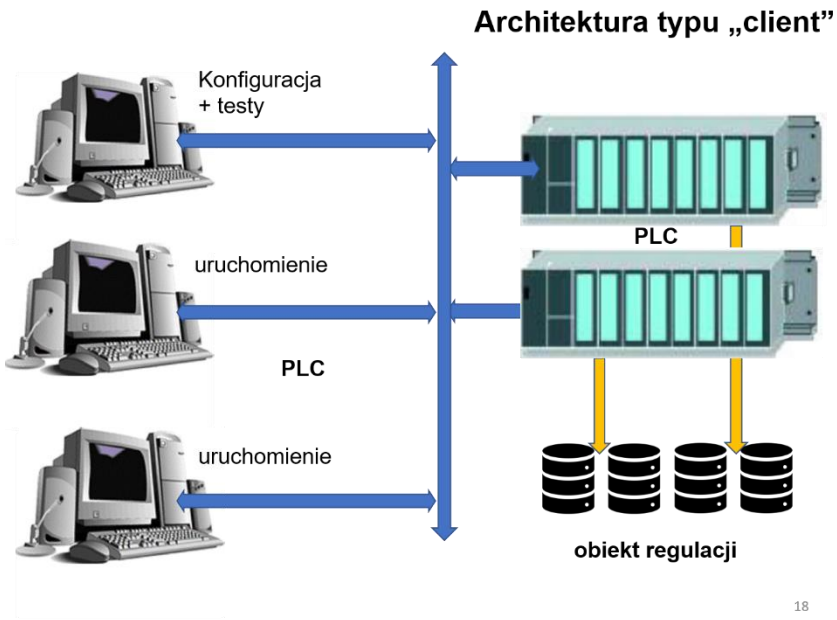
5.5.1 Pojedynczy komputer (najprostsza architektura)

Najprostsza architektura SCADA zawiera pojedynczy komputer, na którym jest budowana, testowana i uruchamiana aplikacja (rys. 5.4)

Rys. 5.4 Architektura z pojedynczym komputerem

5.5.2 Architektura typu „klient”

Aplikacja jest budowana i testowana na jednej stacji, następnie kopiowana i uruchamiana na pozostałych stacjach. Zaleta: Łatwe zapewnienie redundancji, gdyż każda stacja działa całkowicie niezależnie od innych i można dowolnie zwiększać ich liczbę. Wady: Modyfikacja aplikacji wymaga wyłączenia aplikacji, skopiowania nowej wersji i ponownego uruchomienia wszystkich stacji. Na każdej stacji musi być osobna licencja (rys. 5.5).



Rys. 5.5 Schemat architektury typu „klient”

5.5.3 Architektura bazująca na serwerze

Przykład architektury bazującej na serwerze zamieszczono na rys. 5.6.

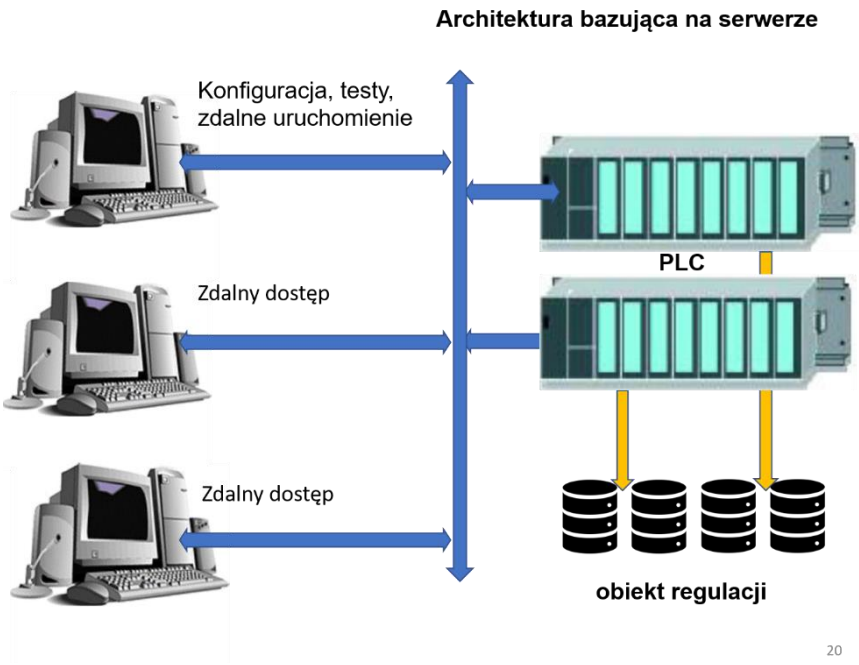
Zalety:

- Nie trzeba kopiować aplikacji,
- Niższe ceny licencji,
- Wszystkie modyfikacje są natychmiast widoczne.

Wady:

- Trudne modyfikacje (pracujemy na jedynej działającej wersji),

• Brak redundancji,
• Wszystkie stacje muszą mieć te same parametry (np. rozdzielczości ekranów).



Rys. 5.6 Architektura bazująca na serwerze

5.5.4 Architektura sieciowa (NAD: Network Application Development)

Aplikacja jest budowana i testowana na centralnej stacji („master"), a następnie kopiowana do innych, gdzie jest wykonywana. Różnice w stosunku do architektury typu „klient":

• Licencja jest zainstalowana tylko na centralnej stacji,
• Praca stacji jest możliwa wyłącznie przy połączeniu ze stacją „master",
• Dane mogą być archiwizowane tylko na głównej stacji (bardzo duże obciążenie sieci!) lub na każdej stacji z osobna.

5.6 Komponenty systemu SCADA

Każdy system SCADA (podobnie jak każdy system komputerowy) zawiera dwa podstawowe typy komponentów sprzętowe i programowe.

Komponenty sprzętowe systemu SCADA:

- sprzęt sieciowy,
- serwery danych,
- serwery aplikacji,
- stacje robocze.

Stacją roboczą może być panel operatorski o różnych możliwościach i złożoności lub komputer osobisty.

Komponenty programowe systemu SCADA:

- oprogramowanie komunikacyjne,
- środowisko uruchomieniowe,
- system operacyjny,
- oprogramowanie baz danych.

Każda aplikacja SCADA musi być zbudowana z wykorzystaniem odpowiedniego środowiska konfiguracyjnego, które jest niezależną częścią oprogramowania. Zwykle i środowiska uruchomieniowe i konfiguracyjne muszą pochodzić od tego samego producenta.

5.6.1 Oprogramowanie SCADA:

Każde, konkretne środowisko programistyczne SCADA zawiera trzy zasadnicze elementy:

- Środowisko konfiguracyjne, umożliwiające budowę i testowanie aplikacji (symulator).
- Środowisko uruchomieniowe („runtime"), zapewniające uruchomienie i działanie aplikacji w czasie rzeczywistym.
- Driver komunikacyjny, pozwalający na realizację komunikacji pomiędzy aplikacją i urządzeniem.

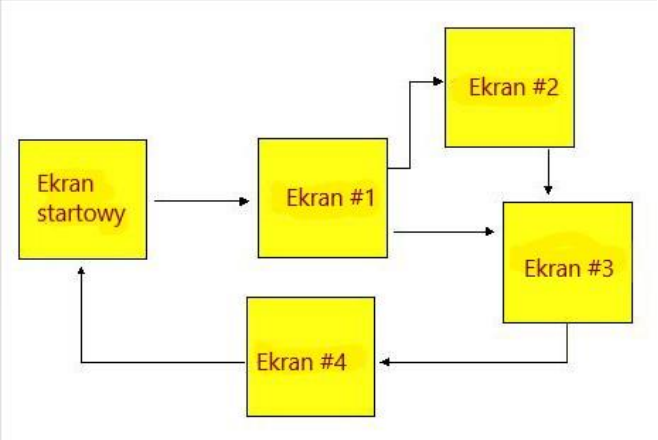
Wersja wykonywalna jest kompilowana, ale musi pracować pod nadzorem środowiska uruchomieniowego.
Wersje komercyjne „runtime'u" mogą się różnić liczbą zmiennych mających możliwość komunikacji z otoczeniem.

Każde oprogramowanie SCADA zawiera duży zestaw driverów komunikacyjnych, które umożliwiają łączenie go z urządzeniami różnych producentów.

5.6.2 Podstawowe elementy aplikacji SCADA:

- Zmienne (Tags),
- Obiekty graficzne,
- Skrypty i funkcje,
- Receptury,
- Archiwa.

Znaczna część aplikacji użytkownika ma charakter graficzny i pogrupowana jest na „ekrany", połączone z sobą w różny sposób. Ogólny układ aplikacji SCADA z punktu widzenia użytkownika przedstawiono na rys. 5.7.



Rys. 5.7 Przykładowy układ ekranów aplikacji SCADA

Zmienne (Tags)

Wewnętrzne, lokalne: Są to zmienne „wewnętrzne" aplikacji, które nie mają możliwości połączenia ze zmiennymi sterownika. Ich liczba zwykle nie jest limitowana.

Zewnętrzne, globalne: Są to zmienne, które mają przypisane adresy w przestrzeni adresowej urządzenia, z którym się komunikują i ich wartości mogą być zmieniane zarówno z poziomu urządzenia, jak i z poziomu aplikacji.

Obydwie wymienione powyżej klasy zmiennych mogą stanowić następujące typy:

- binarne (dyskretne),

- całkowite,
- rzeczywiste,
- czasowe,
- tekstowe,
- liczniki

Dodatkowo, w aplikacjach stosowane są także zmienne opisujące zdarzenia, itp. Jeżeli zmienne te są typu globalnego (wejście/wyjście) to każda z nich jest przypisana do konkretnego urządzenia (np. sterownika PLC). Podczas definiowania takiej zmiennej określamy następujące elementy:

- Urządzenie, do którego zmienna jest przypisana,
- Adres zmiennej w przestrzeni adresowej urządzenia zdefiniowanego wyżej.

Powyższa procedura wynika z faktu, że jedna aplikacja SCADA powinna mieć możliwość współpracy z wieloma urządzeniami (takimi samymi lub różnymi). Szczegóły konfiguracji zmiennych globalnych są zależne od konkretnego środowiska programowego i sprzętu. Zmienne lokalne i globalne mogą też mieć przypisane pewne dodatkowe atrybuty, ułatwiające budowę aplikacji. Przykładowo mogą to być:

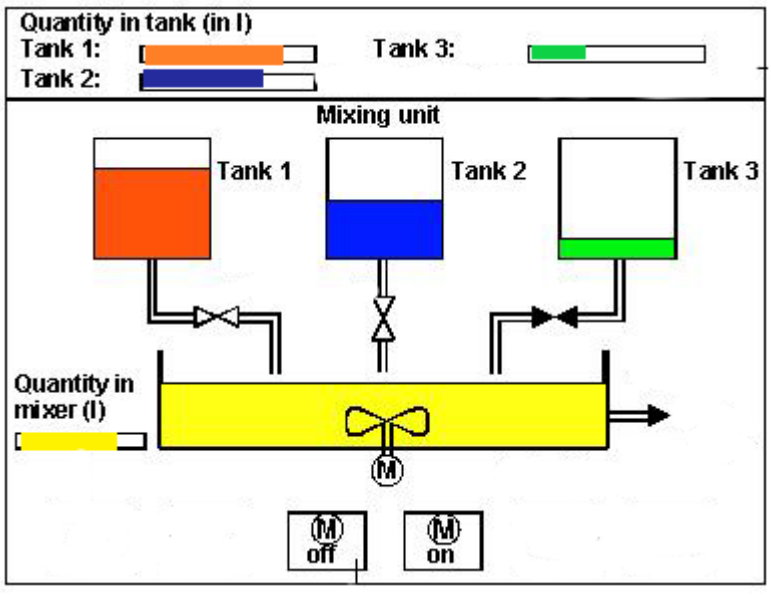
- Wartości początkowe,
- Zakresy dopuszczalnych wartości,
- Alarmy różnych typów,
- Przypisanie zmiennej do archiwum,
- Częstotliwość odczytu/zapisu (tylko zmienne globalne).

Atrybuty te są definiowane podczas deklaracji zmiennych.

Obiekty graficzne:

W oprogramowaniu SCADA stosuje się dużo różnych elementów graficznych. Zaliczamy do nich:

- Okienka wejściowe (do wprowadzania wartości zmiennych),
- Okienka wyjściowe (do wyświetlania wartości zmiennych),
- Elementarne figury geometryczne,
- Przyciski,
- Przełączniki,
- Lampki,
- itp.



Rys. 5.8 Przykładowy ekran prostej aplikacji SCADA – mieszalnik płynów

Elementy graficzne:

- Suwaki,
- Bargrafy,
- Wskaźniki analogowe różnych typów,
- Gotowe elementy graficzne reprezentujące np. różne części instalacji,
- Pola tekstowe do tworzenia etykietek, opisów, itp.

Wszystkie elementy tych obiektów są definiowane podczas ich konfiguracji.

Trend bieżący:

Służy do wyświetlania bieżącego przebiegu określonej zmiennej.

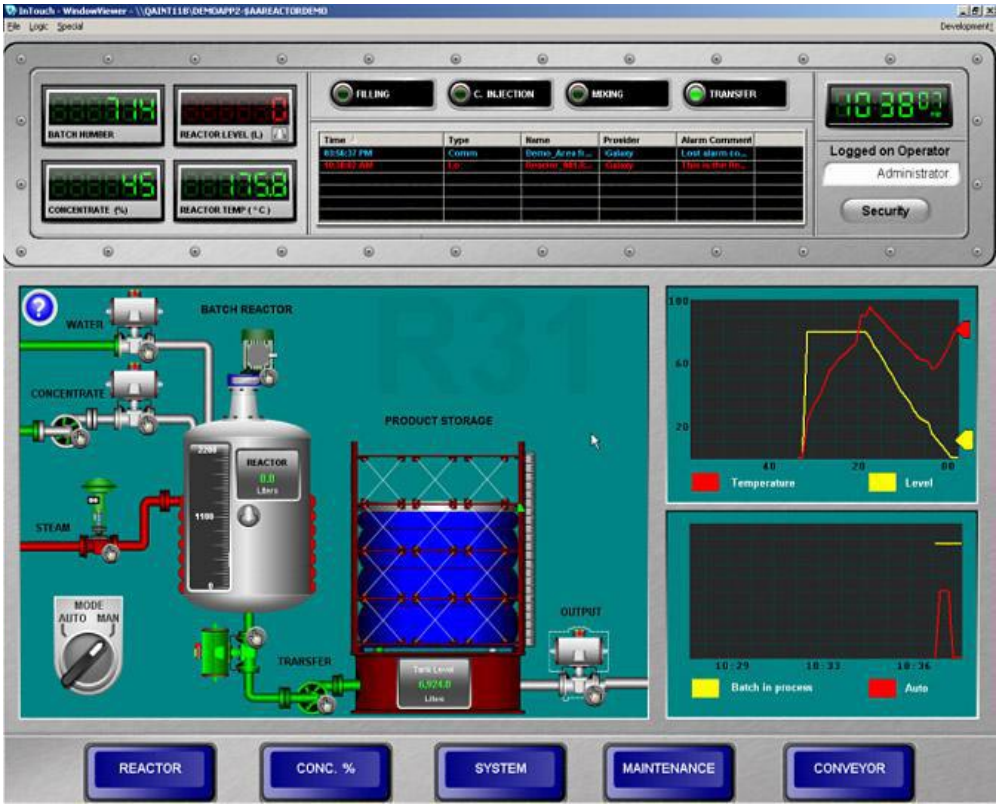
Trend historyczny:

Służy do wyświetlania przebiegu określonej zmiennej w zadanym przedziale czasowym z przeszłości. Wykorzystuje dane pobierane z archiwum.

Wyświetlanie alarmów:

Wyświetla informacje o niebezpiecznych zdarzeniach.

Przykładowy ekran SCADA, zawierający wspomniane elementy, zamieszczono na rys. 5.9.



Rys. 5.9 Przykładowy ekran SCADA (Intouch)

Powiązanie elementów graficznych z elementami aplikacji

Każdy z elementów graficznych może być powiązany z określonymi zmiennymi aplikacji.

Powiązanie elementów graficznych ze zmiennymi jest stosowane do:

- Animacji grafiki z użyciem tych zmiennych,
- Zmiany wartości zmiennych przez użytkownika.

Dodatkowo, elementy graficzne mogą być powiązane z innymi elementami aplikacji, np. funkcjami lub skryptami.

Wykaz typowych elementów ekranu (GIU) SCADA:

- Przycisk (przełącznik) – BOOL. Zmiana wartości zmiennej binarnej w aplikacji lub w sterowniku PLC przez użytkownika. Możliwe różne opcje.
- Suwak- INT . Zmiana wartości zmiennej typu INT w zadanym zakresie.
- Bargraf, wskaźnik analogowy - INT Wyświetlanie wartości typu INT.
- Lampka – BOOL. Świecenie w różnych kolorach w zależności od wartości zmiennej.
- Okienko wejściowe - INT, REAL, STRING, TIME. Przypisanie wartości wpisanej przez użytkownika do zmiennej.
- Okienko wyjściowe - INT, REAL, STRING, TIME. Wyświetlenie wartości zmiennej.
- Elementarna figura geometryczna - BOOL, INT. Różne akcje w zależności od typu zmiennej: BOOL- działanie takie jak lampka, INT - 1. zmiana koloru wypełnienia odcieniami w zależności od wartości zmiennej, 2. zmiana wysokości wypełnienia w danym kolorze, 3. Zmiana rozmiaru obiektu, 4. Zmiana położenia obiektu na ekranie.

Metody aktywacji skryptów/funkcji:

- Jednorazowo podczas startu lub zatrzymania aplikacji,
- Cyklicznie podczas pracy aplikacji (w sposób ciągły lub z zadanym okresem próbkowania),
- Z użyciem przycisku – różne opcje:
  - przy naciśnięciu,
  - przy zwolnieniu,
  - podczas trzymania wciśniętego
- Przy zmianie wartości określonej zmiennej,
- Przy wystąpieniu alarmu/ostrzeżenia.

Archiwa i bazy danych:

Archiwizacja różnych danych związanych z prowadzeniem procesu jest jednym z fundamentalnych zadań każdej aplikacji SCADA. Dane procesowe są archiwizowane w typowych formatach, np. *csv. Szczegóły związane z tworzeniem archiwów są różne w każdym środowisku programowym, natomiast pewne ogólne zasady są wspólne.

Archiwizowane dane mogą być gromadzone:

- W pojedynczym pliku np. *.csv,
- W bazie danych typu:· MS Data Engine 97, MS Access 97, MS SQL Server 7.0.

Proces archiwizacji może być uruchamiany w różny sposób, podobnie, jak wykonanie wszystkich innych funkcji:

- cyklicznie,
- po aktywacji np. przyciskiem,
- po zmianie wartości określonej zmiennej.

Receptury:

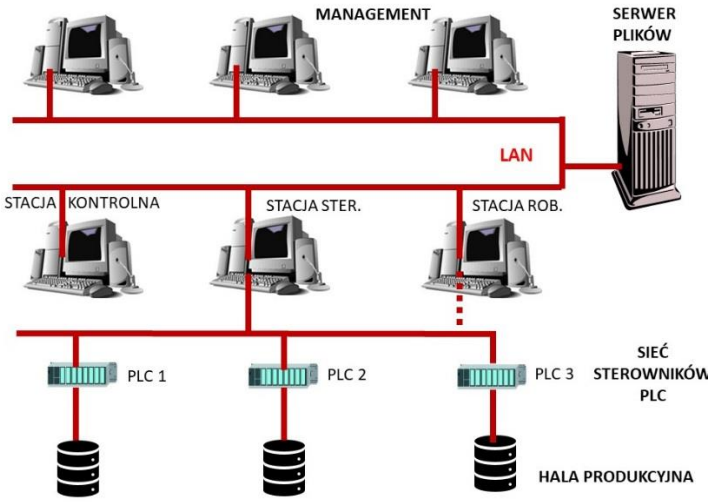
Receptury są stosowane przy wytwarzaniu produktów, które są zestawiane z pojedynczych składników w różnych proporcjach i w zależności od proporcji tych samych składników wejściowych możemy otrzymać różne produkty. Receptura ma postać rekordu danych, w którym są opisane poszczególne składniki oraz ich liczby.

5.7 System Wizcon

Wizcon jest zaawansowanym systemem sterowania nadrzędnego i akwizycji danych (SCADA) umożliwiającym integratorom systemów tworzenie wyrafinowanych aplikacji sterująco - monitorujących dla wszystkich gałęzi przemysłu. Wizcon jest generatorem aplikacji. Oznacza to, że wszystkie funkcje sterowania i monitorowania są już wbudowane w system, natomiast projektant musi tylko zaprojektować aplikację. Wymagane są jedynie minimalne umiejętności w zakresie programowania i obsługi komputera. System ten wykorzystuje wielozadaniowość i możliwości systemów operacyjnych oraz ma wbudowany mechanizm zdarzeniowy (event - driven), umożliwiający osiągnięcie wysokiej sprawności pracy i utrzymania integralności danych. Także graficzny interfejs użytkownika systemu Wizcon zapewnia przejrzystość i wydajność procesu wizualizacji danych technologicznych. Wizcon dla Windows i Internetu umożliwia integrację aplikacji z siecią Internet/Intranet oraz wykorzystanie grafiki czasu rzeczywistego i kierowane zdarzeniowo aktualizacje informacji, a wszystko w dowolnym systemie operacyjnym. Łączy w tym rozwiązaniu zalety systemów SCADA, Javy, HTML i Internetu.

Praca z aplikacjami Wizcon dla Internetu nie wymaga żadnego dedykowanego oprogramowania ani żadnych dodatków. Monitorować i sterować danymi z hali produkcyjnej można teraz z poziomu zwykłej przeglądarki internetowej. Można również przeglądać dane ogólnofirmowe przy użyciu tego samego interfejsu, z dowolnego stacjonarnego lub przenośnego komputera.

Aplikacje systemu Wizcon komunikują się z urządzeniami obiektowymi, takimi jak sterowniki PLC, przyrządy pomiarowe i inne. Ponieważ wszystkie dane są monitorowane i zapisywane, system Wizcon szybko reaguje na zaistniałe sytuacje zgodnie z zaprogramowaną procedurą i żądaniami operatora. Typowa konfiguracja systemu Wizcon SCADA zamieszczona jest na rys. 5.10.



Rys. 5.10 Typowa konfiguracja systemu Wizcon SCADA

Uważnie zaprojektowana aplikacja umożliwia użytkownikowi łatwe, wydajne sterowanie i nadzór nad przebiegającym procesem technologicznym. Przed rozpoczęciem projektowania własnej aplikacji, należy zapoznać się ze specyfikacją procesu, dla którego tworzymy tę aplikację, tzn.:

- uzyskać listę zmiennych, których system Wizcon będzie używał do czytania i zapisywania z/do sterowników PLC (lista we/wy lub lista bramek),
- przejrzeć schematy technologiczne, aby dowiedzieć się jakie wyposażenie jest używane,
- przejrzeć plany fabryki, w celu określenia rozkładu hali produkcyjnej, dla której projektuje się system,

- dowiedzieć się, jaki typ raportów wymagany jest przez użytkowników i kierowników, w jakim ma być formacie i jakie informacje ma zawierać,
- dowiedzieć się jakiego typu sieć jest używana, czy wykorzystuje protokół NetBIOS czy TCP/IP i jakie nazwy mają stacje sieciowe,
- określić wszystkich użytkowników,
- określić ustawienia sterowników PLC, takie jak: bloki i adresy.

Kiedy już zdobyte zostaną wszystkie potrzebne informacje, można przystąpić do budowania aplikacji. Proces wizualizacji musi dokładnie odzwierciedlać działający proces produkcyjny, umożliwiając użytkownikowi wyraźne zrozumienie stanu procesu w każdej chwili. Przy tworzeniu aplikacji powinny być wykonane następujące kroki:

1. Definiowanie sterowników i bloków komunikacyjnych,
2. Definiowanie grup użytkowników,
3. Definiowanie bramek i alarmów,
4. Budowanie obrazu aplikacji,
5. Logika,
6. Testowanie aplikacji,
7. Formowanie wykresów, raportów i receptur,
8. Końcowe strojenie aplikacji

Pierwszym krokiem w projektowaniu aplikacji systemu Wizcon jest zdefiniowanie sterowników i bloków komunikacyjnych. Sterowniki komunikacyjne są używane do obsługi komunikacji z urządzeniami zewnętrznymi, takimi jak sterowniki PLC, urządzenia i przyrządy stosowane w przemyśle, zdalne komputerowe stacje robocze i stacje sieciowe. Sterowniki komunikacyjne funkcjonują w systemie jako oddzielne pliki programów i są kopiowane do katalogu systemu Wizcon podczas jego instalacji. Można zdefiniować bloki komunikacyjne, aby zwiększyć wydajność sterowników, podczas pracy z dużą ilością bramek. Bloki te umożliwiają transfer dużych bloków informacji zamiast indywidualnych elementów danych.

Obrazy Wizcona są dynamicznymi grafikami reprezentującymi przebiegi procesów technologicznych. W procesach tych bramki są reprezentowane przez obiekty obrazu, a obiekty mogą reprezentować wartości występujące w czasie trwania procesu, co prowadzi do graficznego i dynamicznego obrazowania jego przebiegu. Okno obrazu jest standardowym oknem Wizcona, w którym można tworzyć, edytować i przeglądać obrazy. Okna obrazów mogą być przenoszone, można zmieniać ich rozmiar i manipulować nimi przy użyciu standardowych technik stosowanych w odniesieniu do okien.

Termin „Bramka” w systemie Wizcon odnosi się do wartości sterujących monitorowanych przez system. Wartości te są podobne do zmiennych w języku programowania (np., Basic, Pascal lub C) lub do nazw rejestrów w PLC. Każda wartość jest identyfikowana przez unikatową nazwę i może być wyrażona jako, np. liczba całkowita, liczba rzeczywista lub wartość logiczna (Boolean). Bramki różnią się od innych zmiennych tym, że mogą być powiązane z urządzeniami zewnętrznymi takimi, jak rejestry, wejścia/wyjścia w PLC, czy adresami pamięci w zdalnych urządzeniach. Wartość bramki reprezentuje wartość w składniku zewnętrznym lub urządzeniu w taki sposób, że odwołania do bramki są równoważne odwołaniom do takiego elementu lub urządzenia.

Animacja obrazu jest procesem łączenia obiektów obrazu, stworzonych przy pomocy edytora obrazu, z kontrolowanym procesem poprzez bramki. Istnieją dwie drogi stworzenia animacji:

- Obiekty dynamiczne - obiekty w obrazie skojarzone są z bramkami. Są elementami reagującymi na zmiany, jakie zachodzą w bramkach takie jak: położenie, rozmiar, kolor i orientacja. Ostatecznie może zostać osiągnięta dynamiczna, graficzna ilustracja procesu w fabryce. Każdy obiekt w obrazie może być animowany dynamicznie, włączając w to komunikaty procesu.

- Aktywatory - Obiekty w obrazie są asygnowane jako aktywatory. Kiedy tylko obiekty te zostaną aktywowane, wykonane zostaną operacje takie jak: zmiana wartości bramek, a przez to zmieni się graficzna prezentacja obrazu.

5.8 Zintegrowany system monitorowania i sterowania iFIX

IFIX firmy Intellution jest wiodącym na rynku oprogramowaniem SCADA realizującym wszystkie funkcje wizualizacji, akwizycji danych i nadrzędnego sterowania procesami technologicznymi. System pozwala na precyzyjne monitorowanie i kontrolę wszelkich parametrów procesów produkcyjnych oraz urządzeń i zasobów w celu zwiększenia wydajności i elastyczności produkcji.

IFIX jest modułem HMI/SCADA – obiektowej rodziny rozwiązań przemysłowych obejmującej również wydajne rozwiązania do zarządzania i sterowania procesami wsadowymi, gromadzenia i udostępniania danych archiwalnych, monitorowania przestojów oraz narzędzia internetowe do analizy danych bieżących i archiwalnych, a także bieżącego podglądu stanów procesu produkcyjnego. Wszystkie te komponenty mogą być łatwo zintegrowane w jeden kompleksowy system zapewniający kontrolę w czasie rzeczywistym nad każdym elementem najbardziej złożonych procesów przemysłowych.

5.8.1 Właściwości i architektura systemu iFIX

Graficzne narzędzia systemu pozwalają szybko budować złożone aplikacje i dostosowywać je w trybie „On-line" do zmieniających się potrzeb użytkowników. IFIX zawiera wszystko, co jest potrzebne do efektywnego i szybkiego tworzenia aplikacji dowolnego typu i wielkości, począwszy od pojedynczych stacji HMI (Human Machine Interface) a skończywszy na rozbudowanych, wielostanowiskowych, sieciowych systemach SCADA. Rozproszona architektura sieciowa klient – serwer zapewnia maksymalną elastyczność w projektowaniu rozwiązań sieciowych: od pojedynczego komputera pracującego jako samodzielna stacja HMI, aż po olbrzymie systemy SCADA pracujące z wieloma rozproszonymi serwerami i klientami.

Serwer SCADA systemu iFIX łączy się z fizycznymi wejściami i wyjściami odwzorowując ich stan w procesowej bazie danych czasu rzeczywistego. Baza może zawierać różne typy bloków:

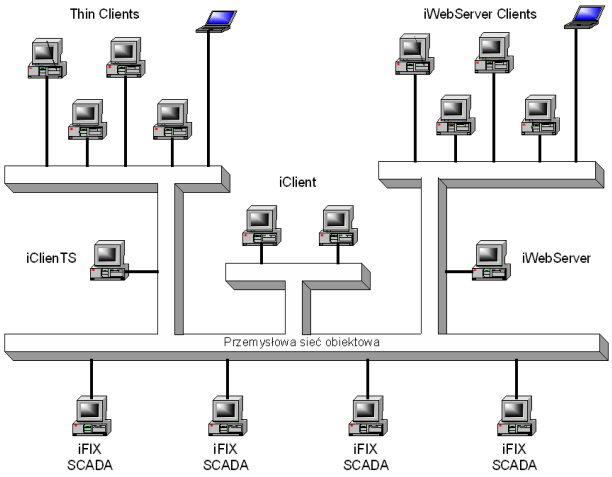
- Wejścia i wyjścia analogowe i cyfrowe,
- Bloki kalkulacyjne,
- Bloki alarmowe,
- Liczniki czasu pracy,
- Funkcje automatycznej regulacji i sterowania,
- Funkcje kontroli statystycznej,
- Bloki poleceń SQL.

Zbierane dane są udostępniane w czasie rzeczywistym lokalnie i sieciowo innym aplikacjom klienckim. Aplikacje iFIX obejmują:

- animowaną grafikę,
- wykresy,
- raporty,
- obsługę alarmów,
- archiwizacje danych,
- zarządzanie recepturami i produkcją wsadową,
- analizę i przetwarzanie danych,
- oraz wiele innych.

Wszystkie aplikacje mogą pracować na wspólnym komputerze z serwerem SCADA albo na wielu komputerach serwerów i klientów w sieciach LAN, WAN, Intranet, Internet. Aplikacja „iClient" instalowana na komputerze klienta, zależnie od skonfigurowanych uprawnień umożliwia dostęp do danych gromadzonych we wszystkich serwerach SCADA podłączonych do sieci przemysłowej. Klient może korzystać ze wszystkich funkcji systemu iFIX, włącznie z animowaną grafiką wykresami, alarmami, raportami i sterowaniem.

Dodatkowo ze stacji klienckiej można dokonywać zmian aplikacji w całym systemie w czasie jego pracy, włączając w to modyfikacje baz danych serwerów SCADA. Aplikacja „iWebServer" wykorzystuje technologie WEB dla udostępniania danych wielu użytkownikom jednocześnie. Animacje są konwertowane do postaci stron HTML i udostępniane na serwerze internetowym w czasie rzeczywistym. Rozproszona architektura systemu iFIX przedstawiona została na rys. 5.11.



Rys. 5.11 Rozproszona architektura systemu iFIX

Rozproszona architektura klient – serwer systemu iFIX umożliwia budowę dowolnej kombinacji rozproszonych serwerów SCADA i rozproszonych klientów. Niezależnie od stopnia rozproszenia użytkownicy widzą jeden, zintegrowany system. Adresowanie zmiennych w systemie obejmuje także nazwę źródłowego serwera SCADA, dzięki czemu każda aplikacja uruchomiona na dowolnym komputerze w sieci działa bezproblemowo odnajdując odpowiednie źródła danych. Raz zaprojektowane aplikacje mogą być uruchomiane na dowolnym komputerze pracującym w sieci bez żadnych modyfikacji. Jako język skryptów system wykorzystuje standardowy „VISUAL BASIC FOR APLICATION", którego pełny edytor wbudowany jest w architekturę. Zapisywanie skryptów każdego obiektu umożliwia przenoszenie obiektów pomiędzy aplikacjami bez utraty ich właściwości.

5.9 SCADA - podsumowanie

Charakterystyka ogólna systemów SCADA:

- Współpraca ze sterownikami PLC, regulatorami mikroprocesorowymi i innymi urządzeniami tzw. centralnej części komputerowego systemu automatyki różnych producentów.
- Możliwość realizacji zdecentralizowanych systemów automatyki przemysłowej, głównie dla średnich i małych instalacji technologicznych.
- Duża otwartość, okupiona brakiem integracji oprogramowania wizualizacyjnego i sterującego.

Konkretne rozwiązania różnią się zakresem i sposobem realizacji poszczególnych funkcji. Do czołowych reprezentantów należą:

- In Touch,

- FIX,
- Wizcon,
- WinCC.

Funkcje systemów SCADA:

- komunikacja z aparaturą sterującą i stacjami operatorskimi,
- przetwarzanie zmiennych procesowych,
- oddziaływanie na proces (sterowanie, regulacja),
- kontrola procesu i sygnalizacja alarmów,
- raportowanie i archiwizacja danych,
- wizualizacja graficzna przebiegu procesu na schematach, wykresach, itp.,
- konfigurowanie struktur algorytmicznych i obrazów synoptycznych,
- wymianę danych z innymi systemami poprzez sieci FAN, LAN, WAN itd.

Cechy systemów SCADA:

- Wielozadaniowy system operacyjny z wywłaszczaniem,
- Praca w strukturze sieciowej,
- Możliwość ewolucyjnej rozbudowy,
- Możliwość rozszerzania i modyfikacji aplikacji w trybie on-line,
- Projektowany stopień niezawodności i zabezpieczenie dostępu,
- Otwartość i skalowalność:
    - wykorzystanie standardowego systemu operacyjnego i oprogramowania sieciowego,
    - wykorzystanie standardowego sprzętu,
    - możliwość opracowania przez użytkownika oprogramowania komunikacyjnego do nietypowych urządzeń lub szeroka dostępność driverów komunikacyjnych,
    - możliwość wymiany danych z bazami danych i innymi systemami z pomocą standardowych mechanizmów np. DDE, OLE, OPC itp.).

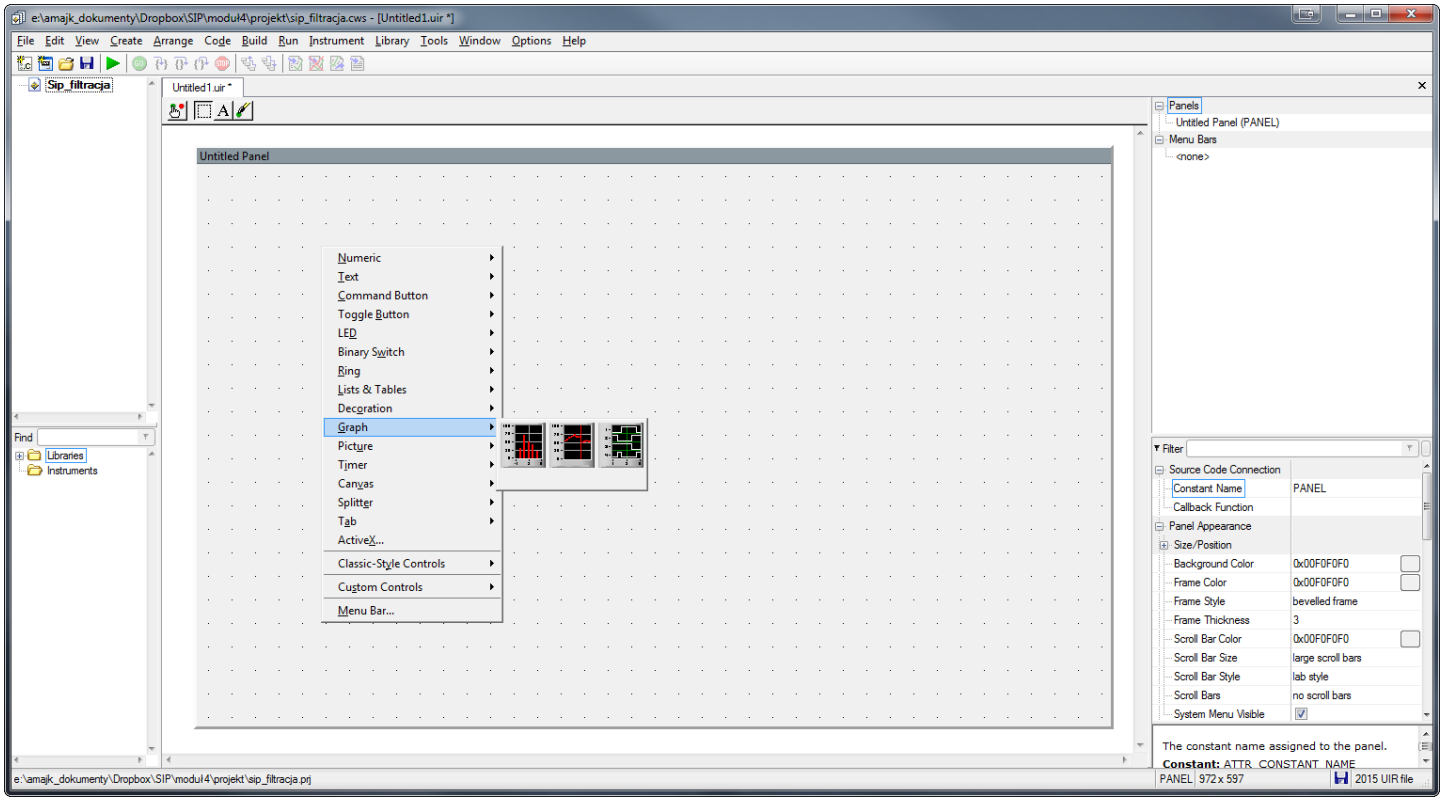
6 Ćwiczenia do modułu (rozwiązane problemy praktyczne - zadania, projekty)

W tej części zostaną zaprezentowane dwa przykłady prostych programów realizujących przyrządy wirtualne. Jeden z nich jest napisany w środowisku LabWindows CVI, drugi w środowisku LabView. Celem ćwiczeń jest zilustrowanie metodyki projektowania i programowania takich przyrządów. Natomiast nie jest to próba opisu podanych środowisk, czy też próba nauki programowania. Te zagadnienia są zbyt szerokie, aby nawet w sposób skrótowy je tu ująć. Ćwiczenia mają raczej zachęcić do własnych studiów i prób programowania. Środowiska LabWindows CVI i LabView można pobrać w wersji testowej ze stron internetowych National Instruments. Prezentowane programy są dołączone do podręcznika.

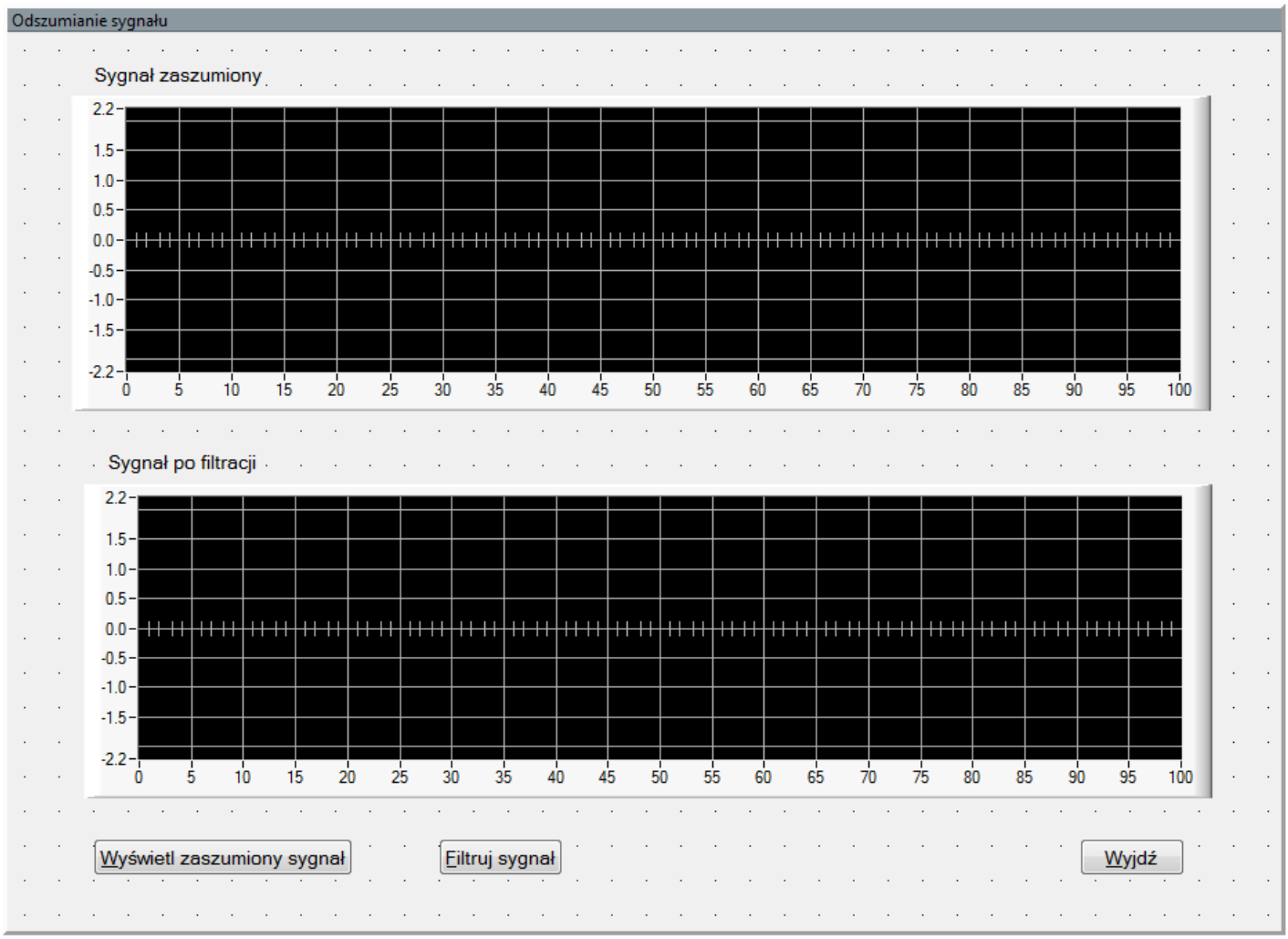
6.1 Program do odszumiania sygnału napisany w środowisku LabWindows CVI

Zadanie jest postawione w następujący sposób. Należy wygenerować sygnał sinusoidalny o częstotliwości 10 Hz. Następnie generujemy szum biały o częstotliwości większej niż 150 Hz. Dodajemy do siebie te sygnały. W ten sposób otrzymujemy sygnał zaszumiony. Następnie należy zaprojektować filtr cyfrowy, który odfiltruje szum i pozostawi czysty sygnał sinusoidalny.

Zaczynamy od stworzenia nowego projektu. Możemy go od razu nazwać, np. *Sip_przykład1*. Na razie projekt jest pusty, nie ma dołączonych żadnych plików. Stwórzmy nowy plik z definicją interfejsu użytkownika (*File -> New -> UserInterface*). Powstanie plik o rozszerzeniu .uir. Umożliwia on stworzenie interfejsu użytkownika. Robi się to w bardzo prosty sposób wybierając i odpowiednio układając potrzebne elementy graficzne. Na tym etapie należy się dobrze zastanowić co będzie potrzebne i jak rozmieścić elementy interfejsu graficznego. W naszym przypadku musimy utworzyć dwa wykresy. Jeden z nich będzie przedstawiał zaszumiony, drugi przefiltrowany sygnał. Potrzebne będą też trzy przyciski do sterowania programem. Naciśnięcie pierwszego spowoduje generację i wyświetlenie sygnału zaszumionego, naciśnięcie drugiego zainicjuje filtrację sygnału i jednocześnie sygnał po filtracji zostanie wyświetlany na drugim wykresie. Trzeci przycisk będzie służył do zakończenia działanie programu. Najpierw stworzymy pierwszy wykres. Klikamy lewym przyciskiem myszy na szarym polu panelu interfejsu graficznego i wybieramy element *Graph* (rys. 6.1). Zmieniamy rozmiar wykresu do postaci, która nas zadowala. Podobnie umieszczamy drugi wykres. Następnie tworzymy trzy przyciski sterujące. W tym celu wybieramy, w taki sam jak wyżej sposób, element *Command Button -> OK*. Tworzymy trzy przyciski. Można je również dowolnie rozmieścić na panelu, a także zmienić ich rozmiar. Mamy już wszystkie potrzebne elementy. Teraz należy im przypisać odpowiednie atrybuty. Z każdym elementem związane jest wiele różnych atrybutów. Niektóre są bardzo istotne, inne mniej ważne. Okienko atrybutów jest dostępne po podwójnym kliknięciu danego elementu lewym przyciskiem myszki, albo z prawej strony okna środowiska LabWindows CVI (rys. 6.1). W przypadku panelu zmieniamy jego nazwę. Jeśli chcemy, żeby program się zamykał po kliknięciu krzyżyka, należy również przypisać do panelu odpowiednią funkcję zamykającą. Dla elementów typu *Graph* zmieniamy ich nazwy i ukrywamy legendę (opcja *Legend Visible*). Możemy też odpowiednio ustawić skalowanie osi na wykresie i sposób wyświetlania siatki z podziałką. W każdym przypadku ważne są nazwy stałych identyfikujących poszczególne elementy w programie (opcja *Constant Name*). Muszą one być unikalne. Nazwy tych stałych można zmienić, albo pozostawić przypisane automatycznie. Na końcu opisujemy przyciski. Tu musimy wykonać trochę więcej rzeczy, bo są to elementy sterujące programem. Dla pierwszego przycisku ustalamy nazwą (*Properties -> Lebel*) na *_Wyświetl zaszumiony sygnał*, drugi na *_Filtruj sygnał* i trzeci na *_Wyjdź*. Efekt końcowy jest pokazany na rys. 6.2.



Rys.6.1 Tworzenie interfejsu graficznego



Rys.6.2 Gotowy interfejs GUI programu

Teraz dołączymy plik interfejsu graficznego do projektu. Najpierw zapisujemy plik pod dowolną nazwą, np. *sip_przyklad1.uir*. Następnie dołączamy go do projektu (klikamy lewym przyciskiem myszy na nazwie projektu i wybieramy *Add Existing file* i nasz plik. Inną opcją dodawania plików do projektu jest polecenie *Edit -> Add Files to project*.

W projekcie musimy też oczywiście mieć plik źródłowy. Tworzymy go i dodajemy do projektu. Klikamy prawym przyciskiem myszy na nazwie projektu i wybieramy opcję *Create New File* i następnie *Source File*. Nazwa pliku źródłowego może być taka sama jak nazwa projektu (ale nie musi). Po kliknięciu *OK* zostanie utworzony nowy plik źródłowy. Jednocześnie, automatycznie zostanie dodany plik nagłówkowy. Mamy już wszystkie potrzebne pliki. W LabWindows CVI programujemy w języku C. Środowisko dostarcza jednak wielu narzędzi pomocnych w programowaniu. Część kodu można też wygenerować automatycznie.

W celu wygenerowania funkcji main() wracamy do pliku *.uir*. Klikamy w panel. Z menu wybieramy polecenie *Code -> Generate_Main Function*. W pliku źródłowym (z rozszerzeniem .c) powinny się pojawić definicje potrzebnych zmiennych oraz definicja funkcji *main ()* (Listing 1).

```
#include <cvirte.h>
#include <userint.h>
#include "prz1.h"
static int panelHandle;

int main (int argc, char *argv[])
{
	if (InitCVIRTE (0, argv, 0) == 0)
		return -1;	/* out of memory */
	if ((panelHandle = LoadPanel (0, "prz1.uir", PANEL)) < 0)
		return -1;
	DisplayPanel (panelHandle);
	RunUserInterface ();
	DiscardPanel (panelHandle);
	return 0;
}
```

Listing 1. Wygenerowana funkcja *main()*

Usuwamy niepotrzebne linie kodu:

```
/// HIFN  What does your function do?
/// HIPAR x/What inputs does your function expect?
/// HIRET What does your function return?
int Define_Your_Functions_Here (int x)
{
	return x;
}
```

Tak utworzony program można już skompilować. Po kompilacji do pliku nagłówkowego *sip_przyklad1.h* zostaną automatycznie dołączone definicje odpowiednich stałych związanych z utworzonym wcześniej interfejsem użytkownika (panelem). Program można też uruchomić. Jednak po jego uruchomieniu pokazuje się panel główny i … nie ma jak z niego wyjść. Jest to związane z działaniem wygenerowanego kodu (funkcja *main()*). Funkcja ta ładuje panel do pamięci (*LoadPanel()*) i wyświetla ten panel na ekranie monitora (*DisplayPanel()*). Następnie wykonywana jest funkcja *RunUserInterface()*. W dużym skrócie funkcja ta sprawdza w nieskończonej

pętli, czy nie nastąpiły jakieś zdarzenia (*callback()*). My nie mamy zdefiniowanych żadnych zdarzeń więc w programie nie można nic zrobić. Program możemy zatrzymać naciskając ikonkę STOP ze środowiska LabWindows CVI.

W następnym kroku należy dodać zdarzenia powiązane z naciśnięciami zdefiniowanych wcześniej przycisków. Na początek zaprogramujmy możliwość wyjścia z programu. W tym celu najpierw musimy zdefiniować funkcję *callback()* (czyli zdarzenie) związane z przyciskiem *Wyjdź*. Klikamy plik *.uir*. Na panelu interfejsu użytkownika klikamy dwa razy przycisk *Wyjdź* i otwieramy okienko właściwości. Wpisujemy nazwę funkcji callback – *quit*. Należy też ustawić *Control mode* na *Hot*. Zamykamy okienko właściwości. Następnie prawym przyciskiem myszy klikamy przycisk *Wyjdź* i wybieramy opcję *Generate Control Calback*. W pliku *.c* pojawi się kod odpowiedniej funkcji *callback()* jak na Listingu 2.

```
int CVICALLBACK quit (int panel, int control, int event,
  void *callbackData, int eventData1, int eventData2)
{
        switch (event)
        {
                case EVENT_COMMIT:

                        break;
        }
        return 0;
}
```

Listing 2. Wygenerowana funkcja *callback()*

Jednak funkcja ta nadal nic nie robi. Trzeba jeszcze dopisać funkcję kończącą program (Listing 3).

```
int CVICALLBACK quit (int panel, int control, int event,
                                        void *callbackData, int eventData1, int eventData2)
{
        switch (event)
        {
                case EVENT_COMMIT:

                        QuitUserInterface(0);

                        break;
        }
        return 0;
}
```

Listing 3. Kompletna funkcja *callback()*

W tym momencie można już zakończyć program przyciskiem *Wyjdź*. Aby można było zakończyć program kliknięciem krzyżyka trzeba jeszcze powiązać z panelem interfejsu użytkownika stworzoną funkcję kończenia programu. Robi się to we właściwościach panelu (*Panel Settings -> CloseControl*). Ustawiamy funkcję *_Wyjdz(QUIT)*. Przy okazji możemy też zablokować maksymalizację panelu i zmianę jego wielkości (opcje *Can Maximize* i *Sizable*).

Musimy jeszcze oprogramować pozostałe przyciski. LabWindows CVI oferuje wiele różnych funkcji. Warto się z nimi zapoznać. Wszystkie mają tzw. panel funkcji, poprzez który można wprowadzać zmienne i uzyskać pomoc

odnośnie działania funkcji. Funkcje są zgromadzone w zakładce *Library* i pogrupowane w kategorie. Jest ich dość dużo. Oczywiście można też używać standardowych funkcji języka C. Ponieważ celem ćwiczenia nie jest nauka programowania, tu zostaną tylko opisane konkretne fragmenty kodu związane z przyciskami *Wyświetl zaszumiony sygnał* i *Filtruj sygnał*.

Najpierw tworzymy zdarzenia (funkcje *callback()*) dla przycisków *Wyświetl zaszumiony sygnał* i *Filtruj sygnał* w sposób opisany wcześniej. Fragmenty kodu definiujące działanie przycisków należy wstawić ręcznie (Listing 4).

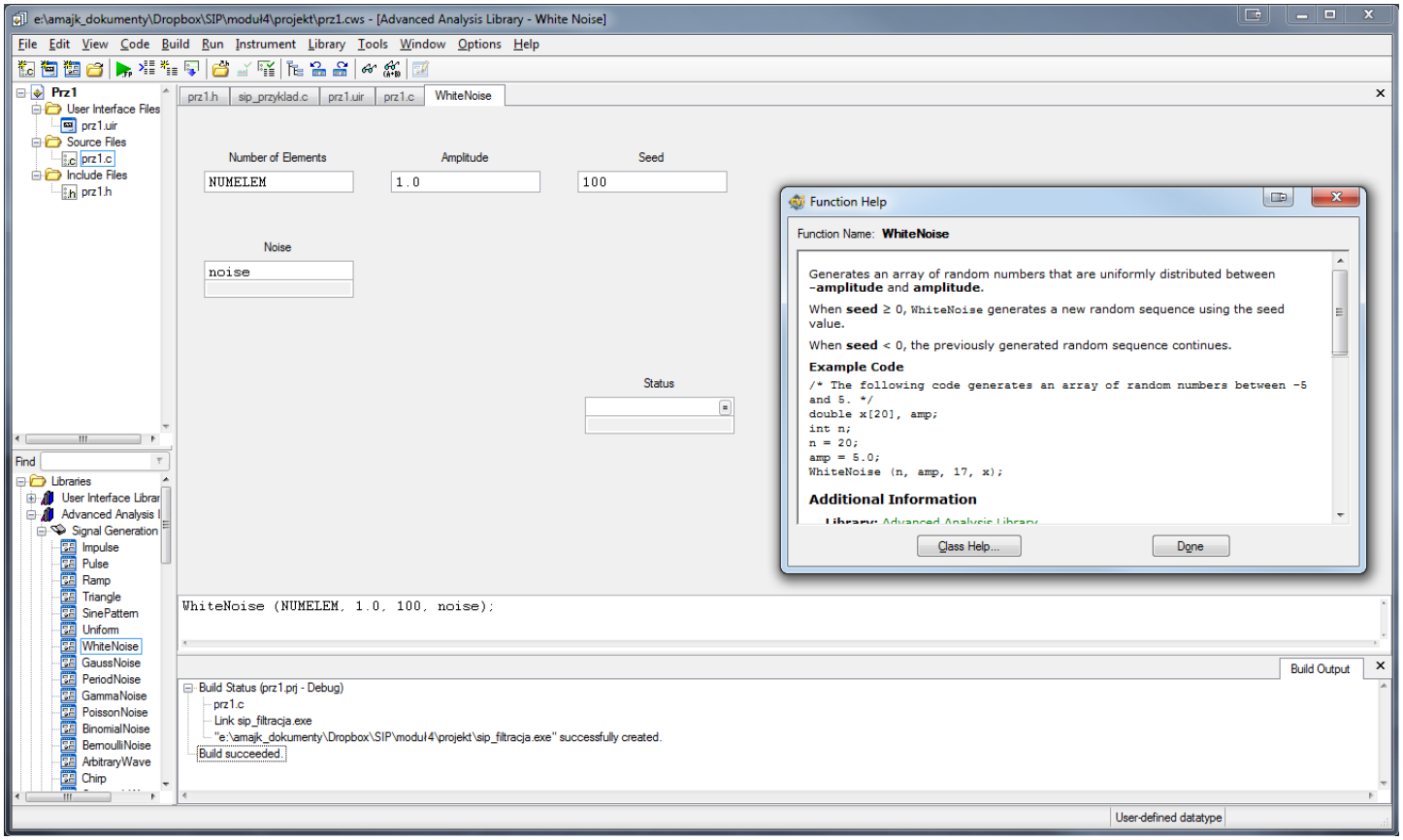
```
#include <analysis.h>
#define NUMELEM 1024
#define ORDER 5
int i=0;
const int order=ORDER;          //rząd filtru
double phase=0.0;               //faza sygnału sinusoidalnego
double noise[NUMELEM];          //szum biały
double clipnoise[NUMELEM];      //szum biały obcięty do <150Hz
double sine[NUMELEM];           //sygnał sinusoidalny
double dirty[NUMELEM];          //zaszumiony sygnał sinusoidalny
double clean[NUMELEM];          //przefiltrowany sygnał sinusoidalny
double x1[ORDER+1];             //warunki graniczne dla filtru IIR
double y1[ORDER+1];             //"
double b[ORDER+1];              //współczynniki filtru IIR
double a[ORDER+1];              //"

int CVICALLBACK Start (int panel, int control, int event,
                                        void *callbackData, int eventData1, int eventData2)
{
        switch (event)
        {
                case EVENT_COMMIT:
                        {

                        WhiteNoise (NUMELEM, 1.0, 100, noise);  //Generacja szumu białego

                        //Obliczenie współczynników BW (tablice a i b) dla filtru IIR górnoprzepustowego 150 Hz
                        Bw_Coef (HIGHPASS, order, 1024, 150, 0, a, order+1, b, order+1);

                        //Warunki graniczne dla filtru IIR, zakładamy, że nie znamy systemu, ustawiamy na 0.0
                        for (i=0;i<(ORDER+1);i++) {
                                y1[i]=0.0;
                                x1[i]=0.0;
                        }

                        //Zastosowanie filtru górnoprzepustowego BW (150 Hz) do szumu
                        IIRFiltering (noise, NUMELEM, a, y1, order+1, b, x1, order+1, clipnoise);

                        SineWave (NUMELEM, 1.0, 7.8125e-3, &phase, sine);  //Generate a Sine wave

                        //Dodanie przefiltrowannego szumu do sinusoidy
                        Add1D (sine, clipnoise, NUMELEM, dirty);

                        //Narysowanie zaszumionej sinusoidy
                        PlotWaveform (panelHandle, PANEL_GRAPH, dirty,
                                                NUMELEM, VAL_DOUBLE, 1.0, 0.0, 0.0,
                                                1, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
                                                VAL_SOLID, 1, VAL_YELLOW);
```

```
                        }
                        break;
        }
        return 0;
}

int CVICALLBACK Filtruj (int panel, int control, int event,
                                                void *callbackData, int eventData1, int eventData2)
{
        switch (event)
        {
                case EVENT_COMMIT:

                        {
                        //Obliczenie współczynników filtru dolnoprzepustowego (25Hz)
                        Bw_Coef (LOWPASS, order, 1024, 25, 0, a, order+1, b, order+1);

                        //x1 i y1 zawierają warunki graniczne dla zastosowanego powyżej filtru IIR
                        for (i=0;i<(ORDER+1);i++) {
                                y1[i]=0.0;
                                x1[i]=0.0;
                        }

                        //Zastosowanie filtru
                        IIRFiltering (dirty, NUMELEM, a, y1, order+1, b, x1, order+1, clean);

                         //Wykreślenie przefiltrowanej sinusoidy
                        PlotWaveform (panelHandle, PANEL_GRAPH_2, clean,
                                                NUMELEM, VAL_DOUBLE, 1.0, 0.0, 0.0,
                                                1, VAL_THIN_LINE, VAL_EMPTY_SQUARE,
                                                VAL_SOLID, 1, VAL_YELLOW);
                        }

                        break;
        }
        return 0;
}
```

Listing 4. Funkcje *callback()* odpowiadające przyciskom *Wyświetl zaszumiony sygnał* i *Filtruj sygnał*

Funkcje przetwarzania danych pochodzą z biblioteki *Advanced Analysis*, stąd trzeba dodać nagłówek #include <analysis.h> . Funkcje do wyświetlania danych pochodzą z biblioteki *User Interface* (wcześniej dodany nagłówek #include <userint.h>). Oczywiście wszystkie potrzebne zmienne należy wcześniej zadeklarować.

Poniżej zamieszczono opis kodu związanego z przyciskiem *Wyświetl zaszumiony sygnał*. Na początku generowany jest szum biały (funkcja *WhiteNoise()*). Opis parametrów funkcji można uzyskać przez panel funkcji - *Recall Function Panel* (rys. 6.3). Aby go wywołać klikamy na nazwie funkcji lewym przyciskiem myszy i wybieramy opcję *Recall Function Panel*. Dodatkową pomoc uzyskujemy przez kliknięcie prawym przyciskiem muszy gdziekolwiek na panelu.



Rys. 6.3 Panel funkcji *WhiteNoise()*

Klikając prawym przyciskiem myszy ma nazwie zmiennej otrzymujemy jej opis. Funkcja *WhiteNoise()* generuje 1024 próbki szumu i zapisuje je w zmiennej *noise*. Następnie szum jest przepuszczany przez filtr NOI, tak by pozostawić tylko częstotliwości wyższe od 150 Hz. Filtr cyfrowy Buterwortha jest zaprojektowany z użyciem funkcji *Bw_Coef()*. Filtracja jest przeprowadzona z użyciem funkcji *IIRFiltering()*. Następnie generowany jest sygnał sinusoidalny (funkcja *SineWave()*). Szum jest dodawany do sygnału sinusoidalnego (funkcja *Add1D()*). Wynikowy sygnał jest wykreślany na wykresie na panelu (funkcja *PlotWaveform()*).

Kod związany z przyciskiem *Filtruj sygnał* jest następujący. Funkcja *Bw_Coef()* jest użyta do zaprojektowania filtru Butterwortha dolnoprzepustowego o częstotliwości odcięcia 25 Hz. Następnie zaszumiony sygnał jest przepuszczany przez ten filtr (funkcja *IIRFiltering()*). Przefiltrowany sygnał jest wyświetlany na wykresie na panelu (funkcja *PlotWaveform()*).

W ten sposób zakończyliśmy projekt. Można go oczywiście modyfikować na różne sposoby.

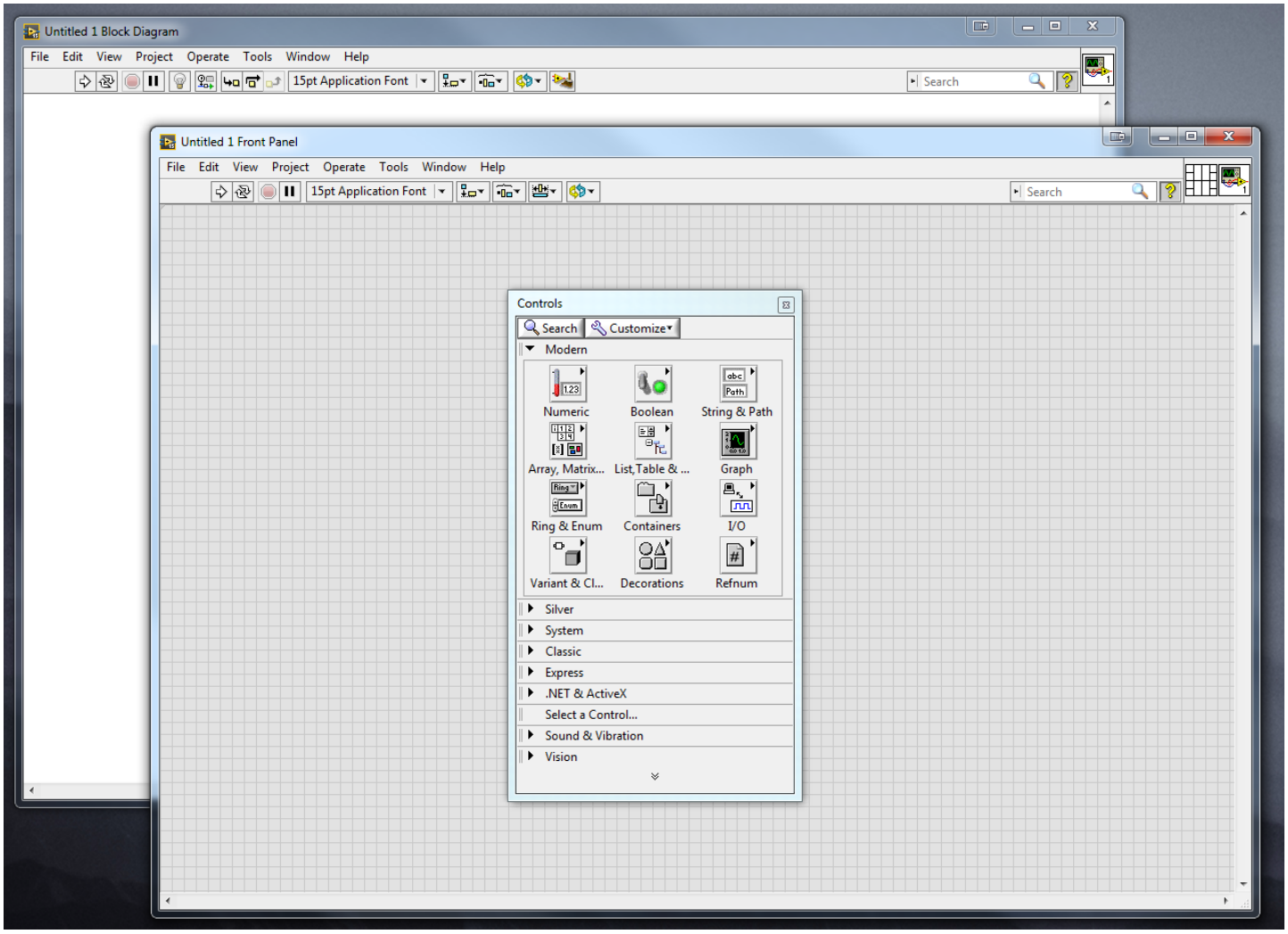
6.2 Prosty analizator widmowy napisany w środowisku LabView

W LabView programuje się używając języka graficznego. W zasadzie rysujemy program. W tym przypadku również zaprezentowany przykład ma pokazać raczej metodykę tworzenia programu, niż nauczyć samego programowania.

Tworzymy nowy projekt w LabView (blank.vi). Pojawiają się dwa panele. Panel z kratką (podziałką) będzie zawierał elementy interfejsu graficznego, panel z białym tłem będzie zawierał nasz program (rys. 6.4).

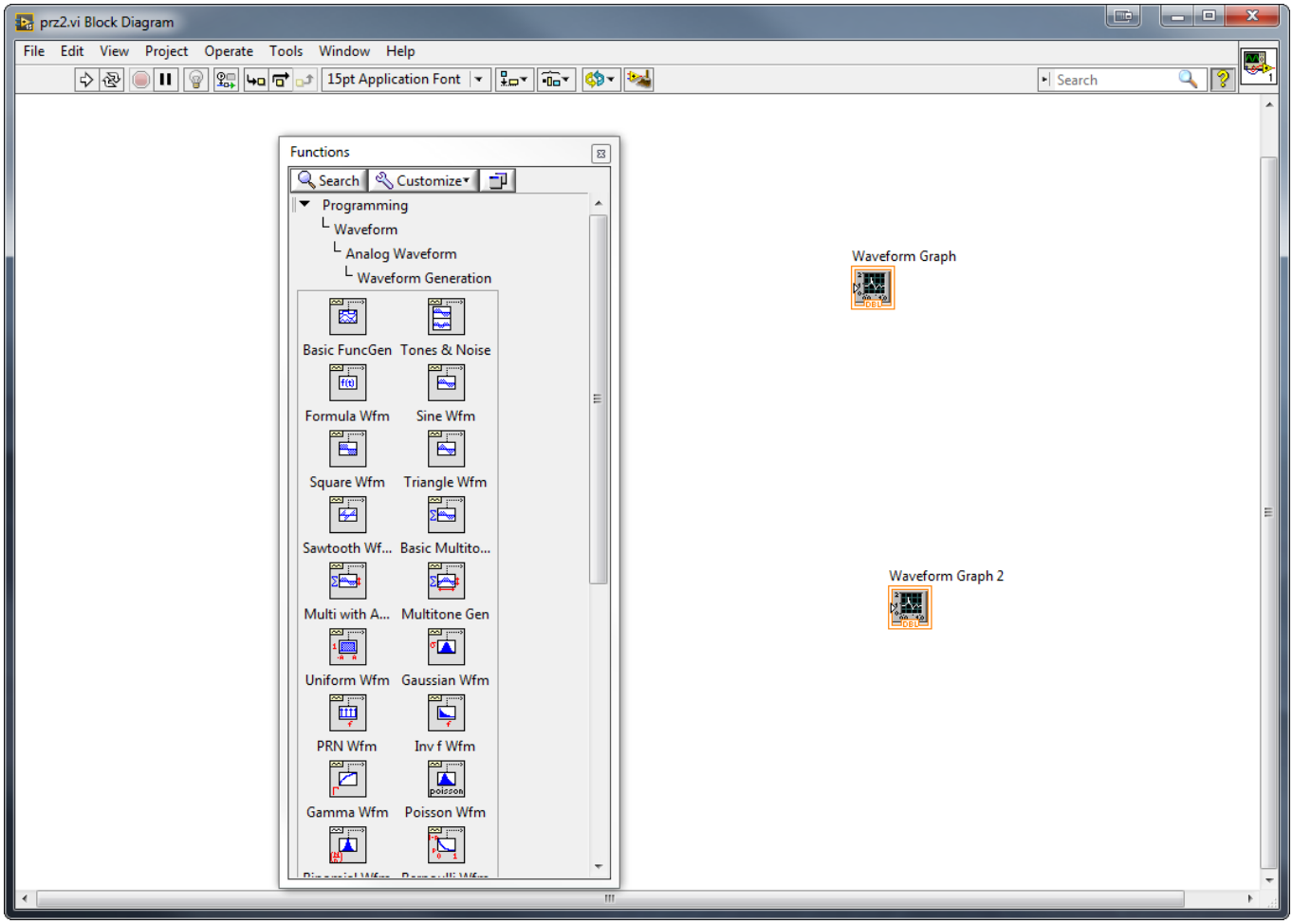
Napiszemy program, który policzy i wyświetli na wykresie widma amplitudowe sygnałów sinusoidalnego, prostokątnego, trójkątnego i piłokształtnego. Zacznijmy od interfejsu użytkownika. Będą potrzebne dwa

wykresy. Jeden będzie przedstawiał sygnał, a drugi jego widmo. Wstawmy te wykresy klikając lewym przyciskiem myszy na panelu interfejsu i wybierając dwa razy element *Graph -> WaveformGraph* (rys. 6.4).

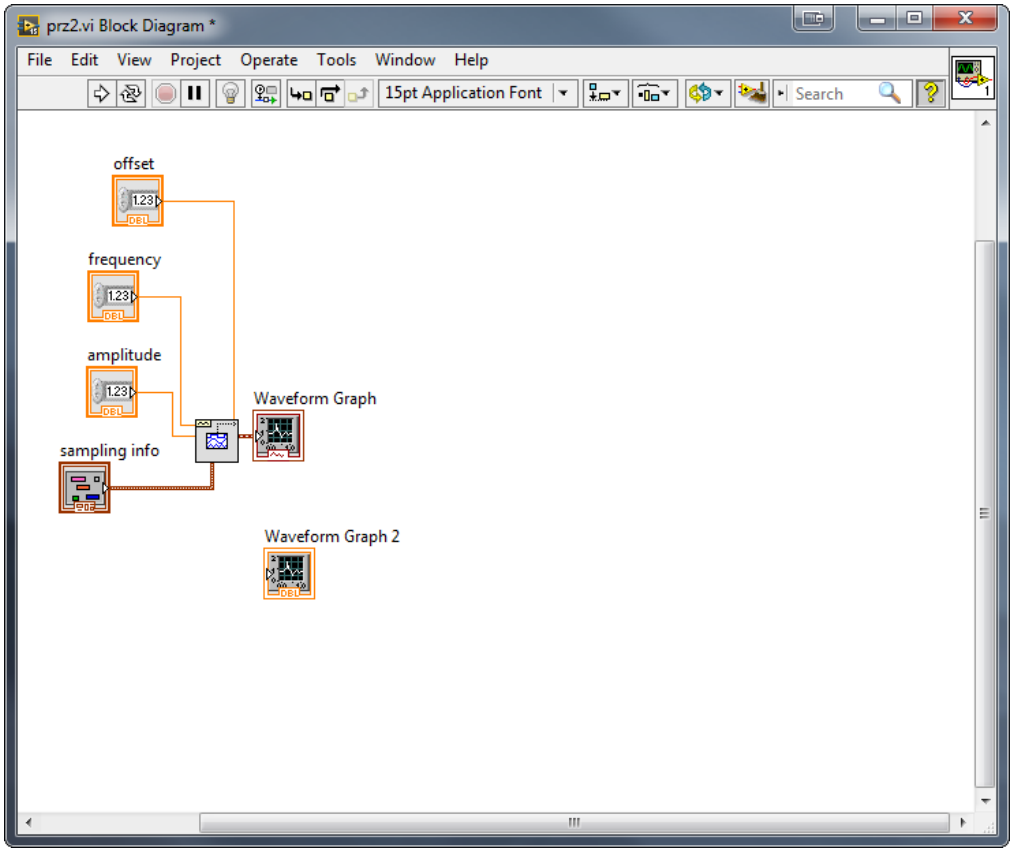


Rys. 6.4 Okna programu i interfejsu użytkownika

Zauważmy, że odpowiedni kod programu (ikony) pojawia się z drugim okienku. Przejdźmy teraz do okna programu. Chcielibyśmy wygenerować sygnały sinusoidalny, prostokątny, trójkątny i piłokształtny. Skorzystamy z bloku *Basic function Generator.vi*. Po kliknięciu lewym przyciskiem myszy w okienku programu wybieramy opcje *Programming -> Waveform -> Analog Waveform -> Waweform Generation* i wybieramy blok *Basic function Generator* (rys. 6.5). Można też wpisać w pole *search* nazwę funkcji czyli *basic function generator*. Przeciągamy odpowiednią ikonę na panel programu. Z każdym elementem związana jest pomoc kontekstowa (polecenie *Help -> Show Context Help*). Najeżdżając myszką na ikonę generatora funkcji otrzymujemy opis wprowadzeń i wyprowadzeń (zmiennych). Każdy kolor oznacza inny typ zmiennej. Łączymy narzędziem szpulka wyjście generatora (*Signal Out*) z wejściem pierwszego wykresu. Chcielibyśmy też mieć kontrolę nad generowanym sygnałem. W tym celu tworzymy odpowiednie pola do wprowadzania zmiennych. Wybieramy wejście *Amplitude* (musi migać) następnie klikamy na nim prawym przyciskiem myszy i wybieramy opcję *Create Control*. Pojawi się ikona kontrolki i odpowiednie pole na panelu interfejsu użytkownika. Analogicznie robimy dla wejść generatora sygnałów *Frequency* i *Offset*. Powstałe na interfejsie użytkownika elementy można dowolnie rozmieszczać i skalować. Utwórzmy jeszcze kontrolkę dla wejścia generatora *Sampling info*. Kontrolka umożliwia wpisanie częstotliwości próbkowania i liczby próbek generowanego sygnału. Oba te parametry są połączone w jedną zmienną. Dotychczas stworzony kod programu jest przedstawiony na rys. 6.6.

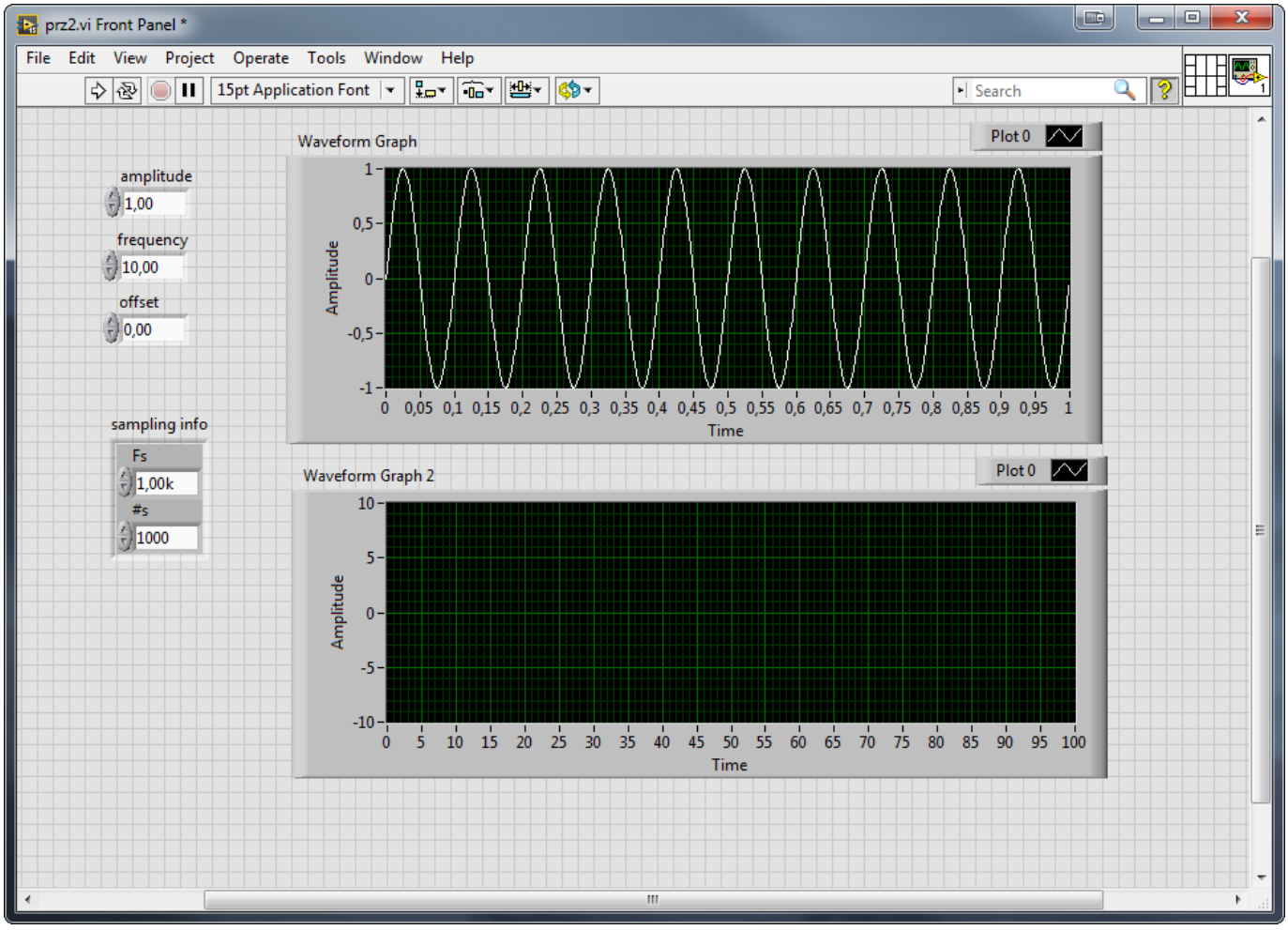


Rys. 6.5 Okno programu i sposób wybierania elementów programu

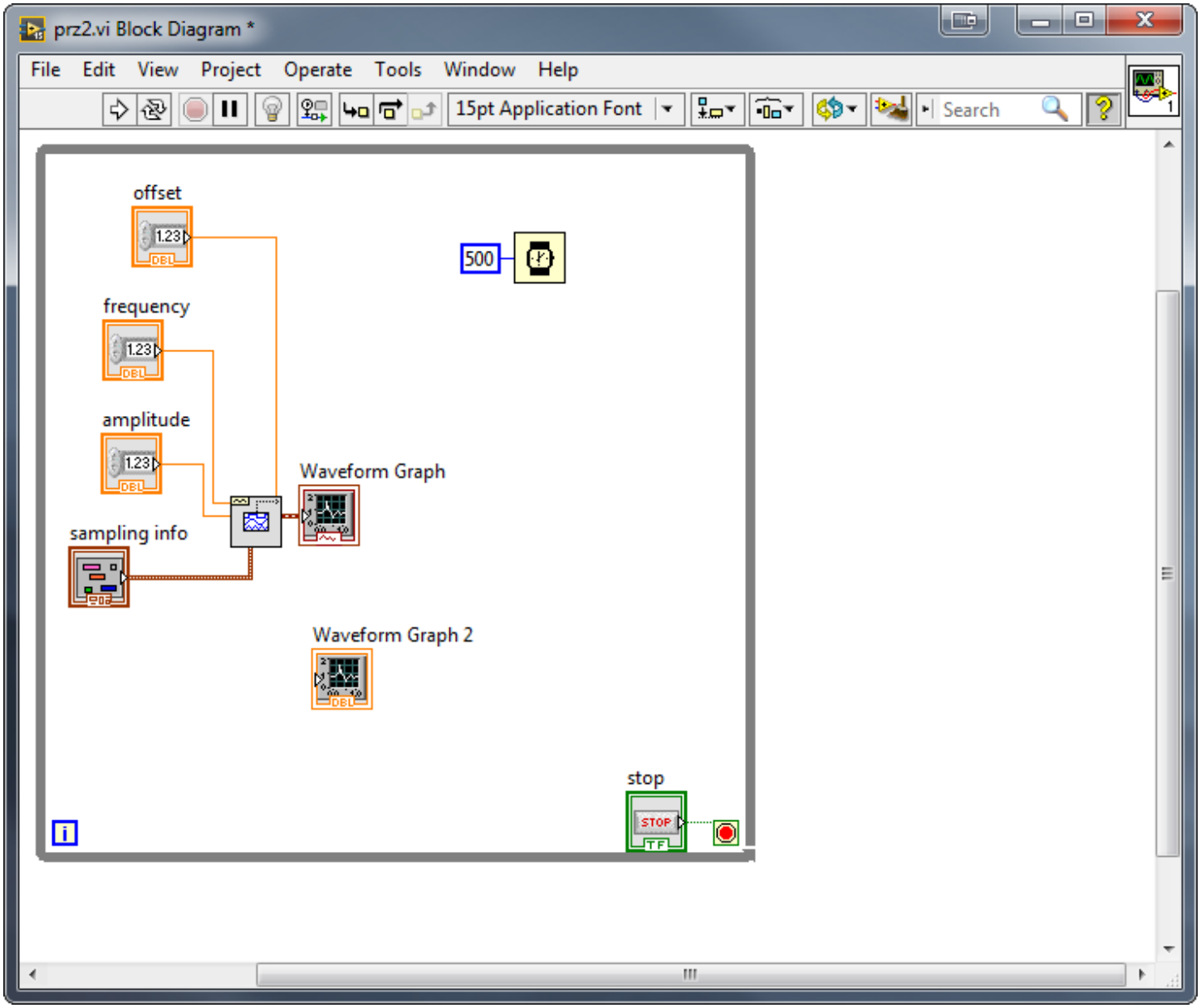


Rys. 6.6 Program generujący sygnał

Program można już uruchomić. Program uruchamiamy klikając ikonę strzałki pod menu głównym środowiska LabView. Można również zmieniać parametry generowanego sygnału wpisując różne liczby w odpowiednie pola. Za każdym razem program uruchamia się jednokrotnie i kończy działanie. Nie jest to wygodne. Zastosujmy pętlę *while* (wybieramy *Programing Structures* -> *While loop*). Przeciągamy pętlę do okna programu i otaczamy nią wszystkie elementy programu. Do czerwonej ikonki na dole pętli tworzymy kontrolkę (podobnie jak poprzednio klikamy prawym przyciskiem myszy na ikonce i wybieramy opcję *Create Control*). Powstaje kontrolka *stop* i jej odpowiednik na interfejsie użytkownika. Jest to warunek zakończenia działania pętli i tym samym całego programu. Jak naciśniemy przycisk *stop* program zakończy działanie. Pętle są wykonywane z maksymalną szybkością. Niepotrzebnie zużywa to moc komputera. Możemy uruchamiać pętlę co jakiś czas. Służy do tego funkcja *Wait()* (wybieramy *Programing* -> *Timing* -> *Wait*). Na wejściu bloku *Wait() miliseconds to wait* tworzymy stałą (klikamy prawym przyciskiem myszy i wybieramy opcję *Create* -> *Constant*, następnie zamieniamy 0 na 500. Pętla będzie uruchamiana co 500 ms. Kod stworzonego programu zaprezentowany jest na rys. 6.8. Teraz program działa w pętli. Uruchamiamy go raz. Aby wyjść z programu należy wcisnąć przycisk STOP.



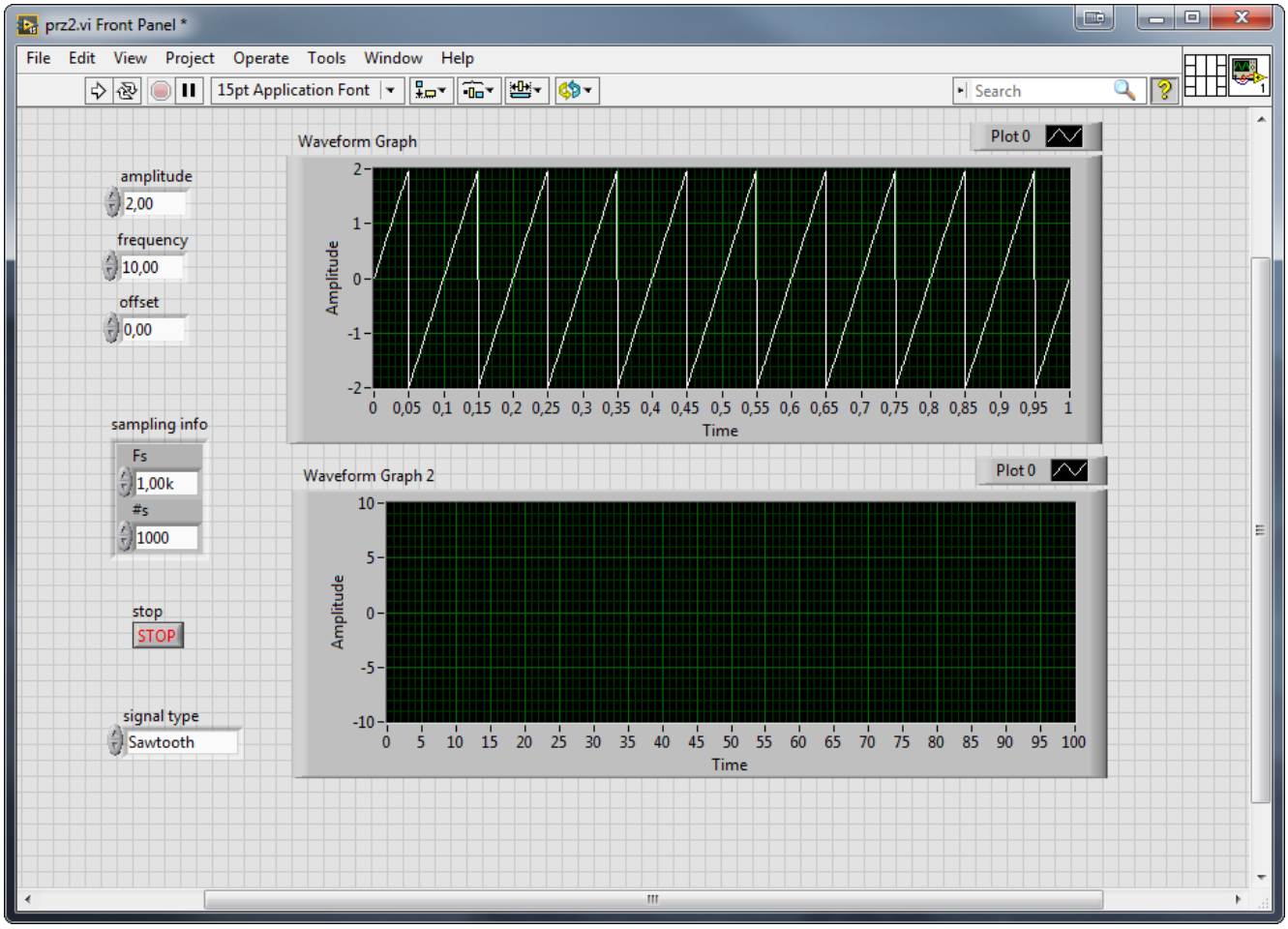
Rys. 6.7 Wynik działania programu zaprezentowanego na rysunku 6.6



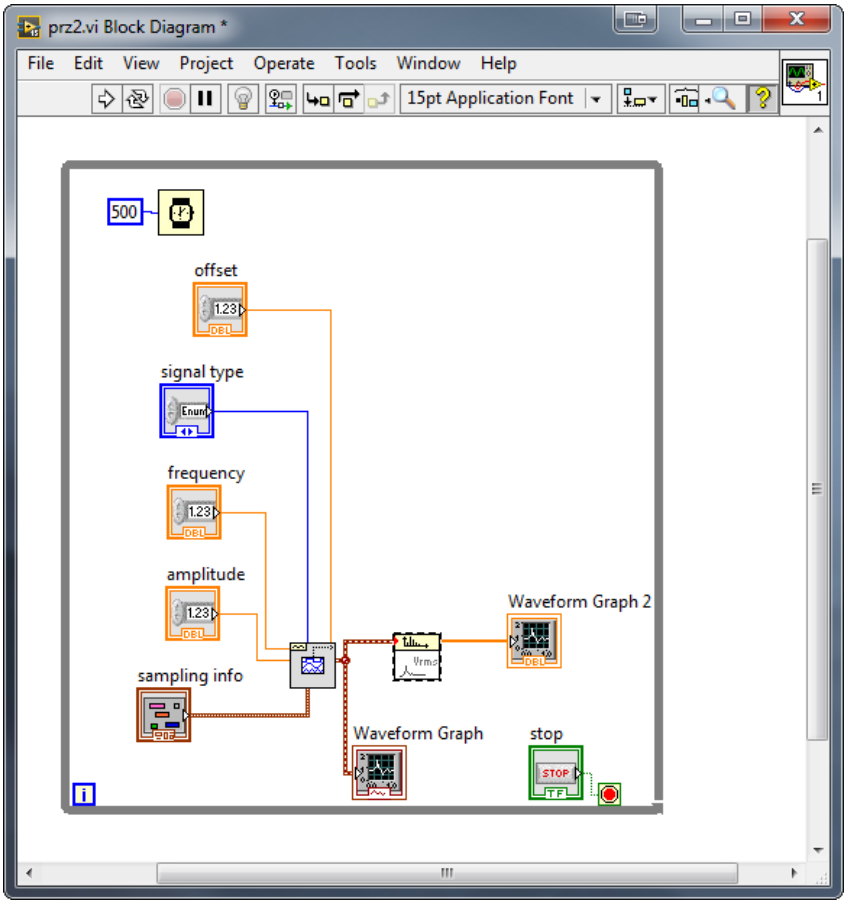
Rys. 6.8 Kod programu z dodaną pętlą *while*

Do tej pory wyświetlany był tylko jeden typ sygnału – sinusoida. Wróćmy do ikony generatora w programie i utwórzmy jeszcze jedną kontrolkę wyboru typu sygnału (klikamy prawym przyciskiem myszy na wejściu generatora sygnałów *signal type* i wybieramy opcję *Creare control*). Pojawia się jeszcze jedna kontrolka. Na interfejsie użytkownika dostajemy pole wyboru, z którego możemy wybrać typ sygnału (rys. 6.9). Ikony programu można w każdej chwili automatycznie uporządkować wciskając sekwencję Ctrl-u z klawiatury komputera. W zasadzie mamy już wszystkie elementy dotyczące generacji sygnału.

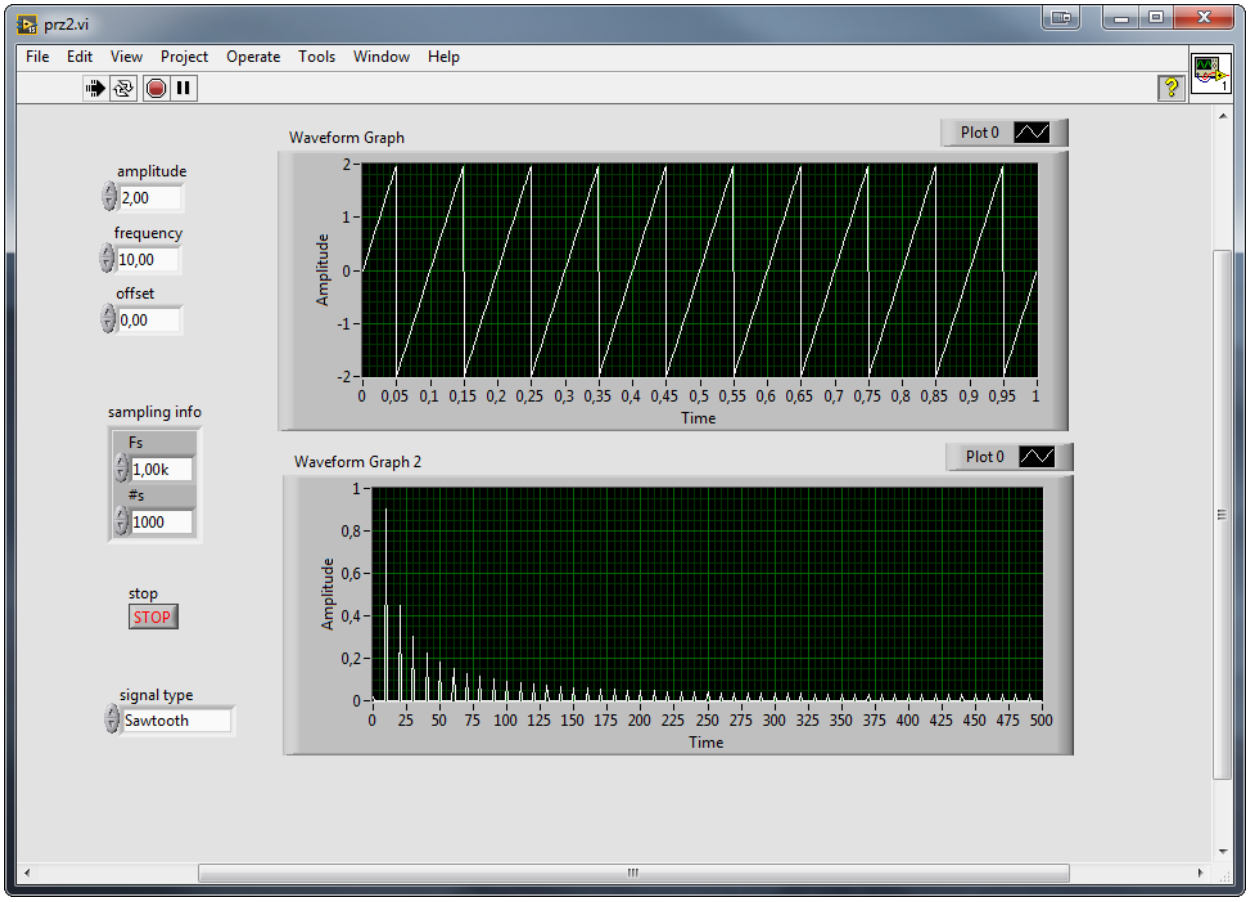
Przejdźmy teraz do policzenia widma sygnału. Wykorzystamy do tego celu blok *Amplitude and Phase Spectrum* (w oknie program wybieramy *Signal Processing -> Spectral Analysis* i blok *Amplitude and Phase Spectrum*). Łączymy wejście bloku *Amplitude and Phase Spectrum* z wyjściem generatora sygnałów, a następnie wyjście bloku *Amplitude and Phase Spectrum* (*Mag. Spectrum*) z wejściem drugiego wykresu z użyciem narzędzia szpulka (rys. 6.10). Po uruchomieniu programu mamy już widmo sygnału (rys. 6.11).



Rys. 6.9 Interfejs użytkownika po dodaniu pętli while i pola wyboru typu sygnału



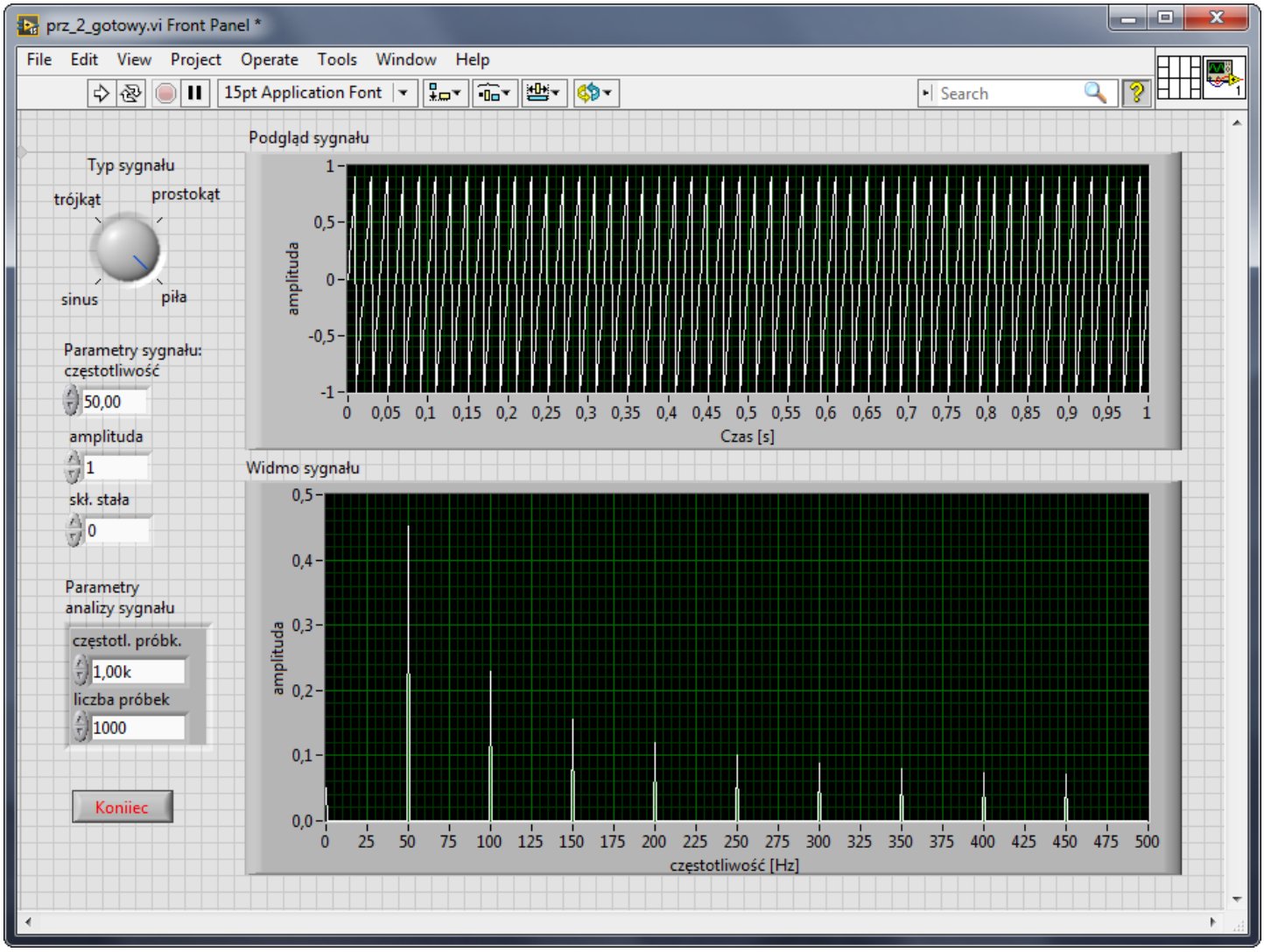
Rys. 6.10 Program z dołączonym blokiem do liczenia widma



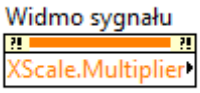
Rys. 6.11 Interfejs użytkownika w policzonym widmem

Warto dokonać jeszcze kilku rzeczy. Po pierwsze należy zoptymalizować rozkład wykresów i kontrolek. Można też pozmieniać opisy kontrolek, przycisków i wykresów. W tym celu należy kliknąć myszką na odpowiedni napis i go po prostu zmienić. Generalnie z każdym elementem interfejsu graficznego jest związany panel *Properties* (klikamy prawym przyciskiem myszy na elemencie i wybieramy ostatnią opcję *Properties*). Poprzez *Properties* można dostosować do swoich potrzeb niemal każdy aspekt danego elementu. Po zmianach panel może wyglądać np. tak jak na rys. 6.12. Można zauważyć, że inaczej niż poprzednio wygląda pole wyboru sygnału. Sposób wyświetlania tego (i innych) elementów można łatwo zmieniać. W tym celu klikamy prawym przyciskiem myszy na konkretnym elemencie i wybieramy opcję *Replace* i następnie element, na który chcemy zmienić obecny (w tym przypadku zamiana nastąpiła na element typu *Knob*).

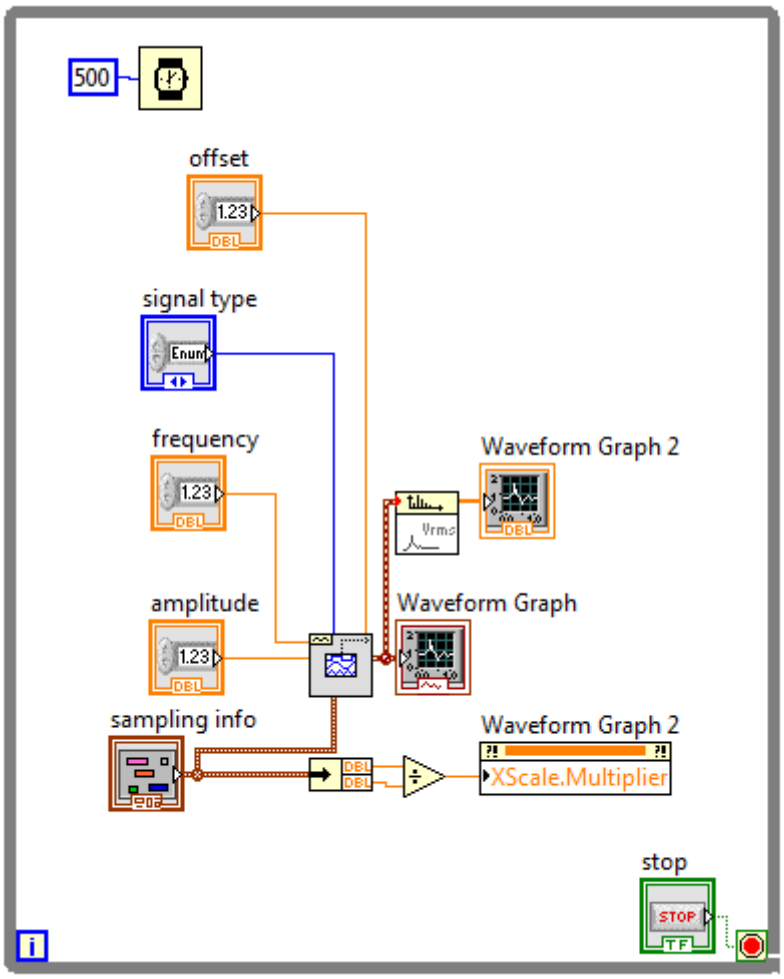
Pozostaje jeszcze jedna rzecz, którą należy oprogramować, a mianowicie skalowanie osi wykresu widma. Jeśli zmieni się parametry analizy sygnału widmo nie będzie poprawnie wyskalowane. Dzieje się tak dlatego, że widmo jest wykreślane w funkcji indeksu próbki. Natomiast odległość między próbkami powinna wynosić *df=częstotliwość próbkowania/liczba próbek*. Odległość taką można co prawda ustawić w *Properties* wykresu. Problem w tym, że będzie się ona zmieniała wraz ze zmianą parametrów analizy sygnału. Można ten problem rozwiązać wyciągając odpowiednią właściwość wykresu na panel programu. W tym celu klikamy prawym przyciskiem myszy na wykresie i wybieramy *Create –> Property Node –> X Scale –> Offset and Multiplyer –> Multiplier*. Pojawi się ikonka jak na Rys 6.13. Klikamy na nią prawym przyciskiem myszy i wybieramy opcję *Change to Write*. Teraz możemy doprowadzić do niej właściwy współczynnik *df*. Musimy go policzyć. Problemem jest to, że parametry *częstotliwość próbkowania* i *liczba próbek* są połączone w jedną zmienną. Trzeba je najpierw rozdzielić. Służy do tego blok U*nbundle* (w okienku programu wybieramy *Programming –> Cluster, Class & Variant* i blok *Unbundle*). Do bloku doprowadzamy zmienną *sampling info* (przy użyciu narzędzia szpulka) jak na rys. 6.14. Doprowadzamy zmienne *częstotliwość próbkowania* i *liczba próbek* do bloku dzielenia (*Programming –> Numeric -> Divide*) i wyjście bloku do kontrolki właściwości wykresu widma (rys. 6.14). Ta operacja kończy program.



Rys. 6.12 Interfejs użytkownika po zmianach nazw domyślnych i optymalizacji położenia kontrolek



Rys. 6.13 Ikona właściwości wykresu



Rys. 6.14 Skalowanie osi wykresu widma i jednocześnie kompletny program analizatora widma

7 Pytania kontrolne

Proszę zaznaczyć poprawne odpowiedzi. Może być jedna bądź dwie poprawne odpowiedzi w każdym pytaniu.

1) W obsłudze modułów sprzętowych najbliżej sprzętu znajduje się:
    a) Poziom rozkazowy
    b) Interfejs użytkownika
    c) *Poziom rejestrowy
    d) Oprogramowanie narzędziowe

2) Sterownik urządzenia:
    a) *Jest to program zawierający instrukcje realizujące pełny dostęp do złożonych funkcji pomiarowych konkretnego urządzenia
    b) Wymaga od użytkownika programowania na poziomie rejestrowym
    c) Ułatwia stworzenie graficznego interfejsu urządzenia
    d) Wymaga od użytkownika znajomości wewnętrznej struktury urządzenia

3) Do specjalistycznego oprogramowania narzędziowego wspomagającego projektowanie wirtualnych przyrządów pomiarowych można zaliczyć:
    a) Basic
    b) * LabVIEW
    c) *TestPoint
    d) Pascal

4) Język SCIPI to:
    a) *Uniwersalny język programowania przyrządów pomiarowych
    b) Uniwersalny język programowania sterowników do przyrządów pomiarowych
    c) Specjalizowane środowisko do tworzenia przyrządów wirtualnych
    d) *Zawiera zestaw instrukcji, które niezależnie od rodzaju przyrządu i typu interfejsu, wysłane z kontrolera pozwalają na pełne zaprogramowanie jego pracy

5) O składni języka SCPI można powiedzieć, że:
    a) Jest to język obiektowy
    b) Jest to język graficzny
    c) *Definiuje zestaw komend do obsługi urządzenia
    d) *Ma strukturę hierarchiczną

6) Przy stosowaniu języka SCPI prawdą jest, że:
    a) Kontroler urządzenia może wysyłać komunikaty w tylko w ściśle ustalonej chwili
    b) *Urządzenie wysyła odpowiedź na życzenie
    c) Moduły informacji programowej (w tym samym komunikacie programowym), które nie zostały poprzedzone dwukropkiem są traktowane jako polecenia innego poziomu niż je poprzedzające
    d) *Jedynie komunikaty zakończone znakiem ? upoważniają urządzenie do umieszczania odpowiedzi w kolejce wyjściowej

7) Środowisko programistyczne LabWindows CVI:
    a) *Umożliwia programowanie w języku C
    b) Umożliwia programowanie w języku C#
    c) Umożliwia programowanie w specjalnym języku graficznym
    d) Można programować w C lub języku graficznym

8) Środowisko programistyczne LabView:
    a) Umożliwia programowanie w języku C#
    b) W środowisku tym można jedynie zaprojektować graficzny interfejs użytkownika
    c) *Umożliwia programowanie w specjalnym języku graficznym
    d) W środowisku tym nie można przeprowadzić analizy i przetwarzania sygnału pomiarowego

9) Do typowych sieciowych protokołów komunikacyjnych można zaliczyć:
    a) *UDP - (ang. User Datagram Protocol)
    b) USB
    c) *NTP - (ang. *Network Time Protocol)*
    d) HDMI

10) System SCADA to

a) *Zdalny system monitorowania i nadzoru procesów technologicznych
b) Środowisko programistyczne, w którym można używać wielu języków programowania
c) Jest bardzo prostym systemem komputerowym
d) *System, który pracuje zawsze w czasie rzeczywistym

11) W architekturze peer-to-peer (P2P):

a) *Każda strona ma równorzędne prawa
b) Występuje dedykowany kontroler systemu
c) *Każdy komputer może jednocześnie pełnić zarówno funkcję klienta, jak i serwera
d) Czas obsługi systemu jest bardzo długi

12) Do funkcji systemu SCADA nie należy:

a) Wizualizacja pracy urządzenia/instalacji
b) *Zastąpienie człowieka
c) Wykrywanie i sygnalizacja stanów awaryjnych
d) Sterowanie zdalne (automatyczne lub ręczne)

13) Komponenty sprzętowe SCADA nigdy nie zawierają:

a) Serwera do zbierania danych z procesów pomiarowych
b) *Symulacji różnych przyrządów pomiarowych
c) Systemu komunikacyjnego - sieci lub systemu łączności bezprzewodowej
d) Stacji roboczych realizujących różne funkcje

14) System Wizcon:

a) *Jest zaawansowanym systemem sterowania nadrzędnego i akwizycji danych (SCADA)
b) Aplikacje systemu Wizcon nigdy nie komunikują się z urządzeniami, takimi jak sterowniki PLC, przyrządy pomiarowe i inne
c) *Praca z aplikacjami Wizcon dla Internetu nie wymaga żadnego dedykowanego oprogramowania
d) Wszystkie funkcje sterowania i monitorowania w tym systemie muszą być opracowane przez jego projektanta

8 Bibliografia

[1] P. Lesiak, D. Świsulski, *Komputerowa technika pomiarowa – w przykładach*, Agenda Wydawnicza PAK, Warszawa 2002.

[2] W. Nawrocki, *Komputerowe systemy pomiarowe*, Wydawnictwa Komunikacji i Łączności, Warszawa 2002.

[3] R. Rak, *Przyrządy wirtualny – realne narzędzie współczesnej metrologi*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2003.

[4] R. Rak, *Systemy informacyjno-pomiarowe*, podręcznik multimedialny, Ośrodek Kształcenia na Odległość Politechniki Warszawskiej – OKNO, Warszawa 2005.

[5] W. Tłaczała, *Środowisko LabVIEW w eksperymencie wspomaganym komputerowo*, Wydawnictwa Naukowo-Techniczne, Warszawa 2002.

[6] W. Winiecki, *Organizacja mikrokomputerowych systemów pomiarowych*, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 1997.

[7] M.Chruściel, LabView w praktyce, BTC, 2008.

[8] W. Mielczarek, Urządzenia pomiarowe i systemy kompatybilne ze standardem SCPI, Helion, 1999.