PROGRAMOWALNE STEROWNIKI LOGICZNE

Prowadzący:
dr inż. Michał Krystkowiak
Michal.Krystkowiak@put.poznan.pl

PROGRAMOWALNE STEROWNIKI LOGICZNE

WYKŁAD nr 1 i 2

- **Pojęcie systemu czasu rzeczywistego oraz programowalnych sterowników logicznych**
- **Klasyfikacja programowalnych systemów sterowania logicznego**
- **Architektura programowalnych sterowników logicznych**
- **Charakterystyka środowiska programistycznego Step7-Micro/WIN**
- **Języki programowania programowalnych sterowników logicznych**

System czasu rzeczywistego

System czasu rzeczywistego (*real time system*) to system komputerowy, w którym obliczenia prowadzone są równolegle z przebiegiem procesu i mają na celu: nadzorowanie, sterowanie oraz terminowe reagowanie na zdarzenia zachodzące w procesie.

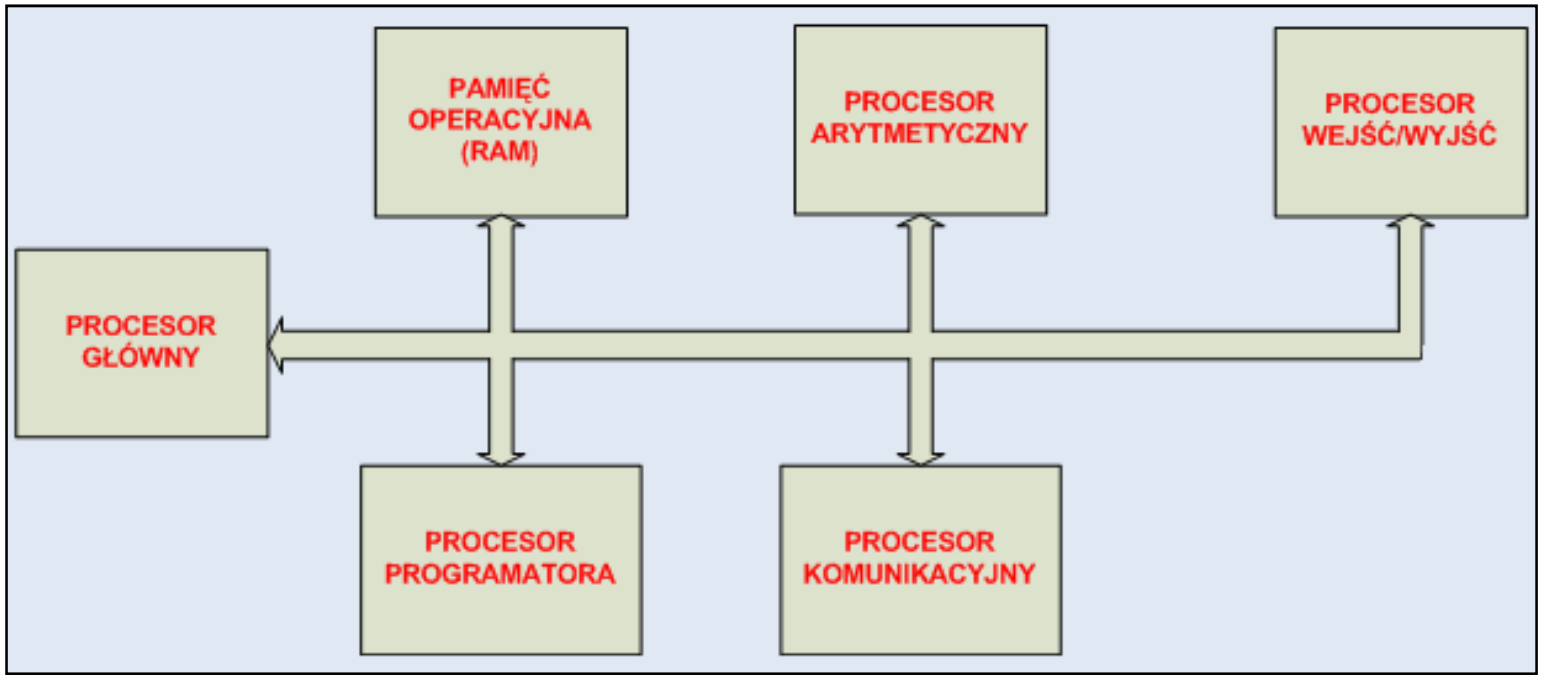
Programowalne sterowniki logiczne (PLC)

Programowalne sterowniki logiczne (*programmable logic controller*) to urządzenia przeznaczone przede wszystkim do przetwarzania sygnałów dwustanowych. Do podstawowych dziedzin ich zastosowania należą: systemy sterowania procesami dyskretnymi oraz maszynami, układy zabezpieczeń i kontroli.

Klasyfikacja programowalnych systemów sterowania logicznego

Przetwarzanie sygnałów ciągłych	Rodzaj systemu	Liczba wej/wyj binarnych	liczba generatorów interwałów czasowych
wyjątkowo	**mały**	24…512	16…256
często	**średni**	768…4096	128…2048
zawsze	**duży**	do 10240	do 5000

Architektura programowalnych sterowników logicznych



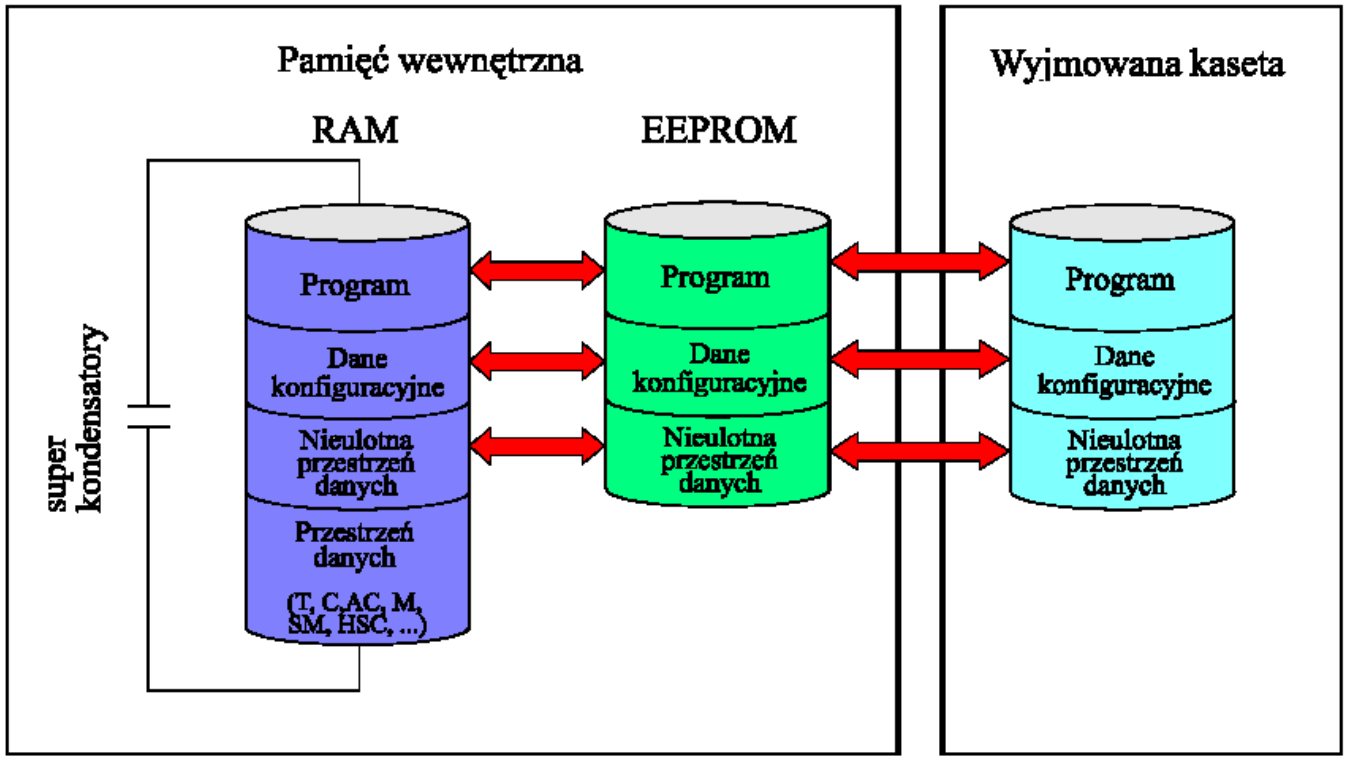
Stany pracy procesora głównego:

a) stan wejścia (*load state*) – odczyt stanu wejść i aktualizacja ich wartości w pamięci operacyjnej,

b) stan przetwarzania *(solve state*) – rozwiązywanie równań oraz relacji logicznych dla elementów wyjściowych,

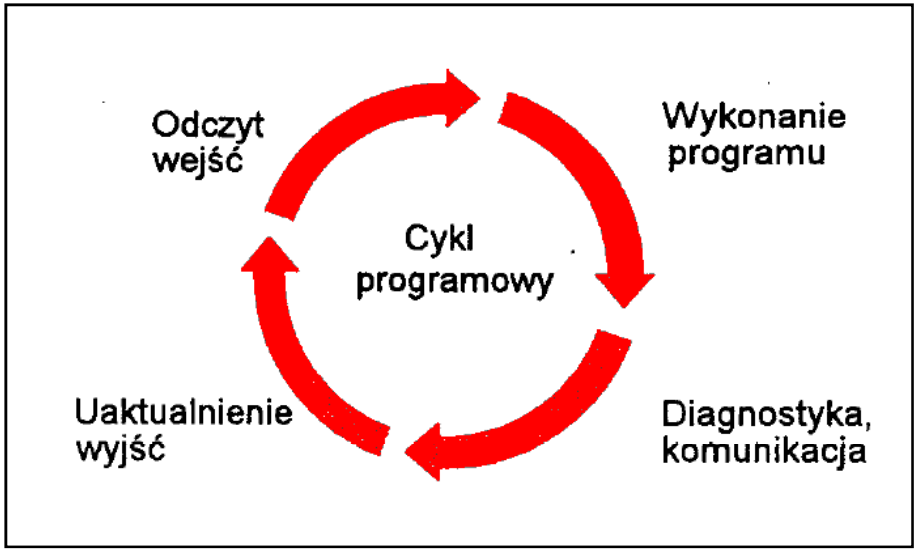
c) stan wyjścia (*dump state*) – uaktualnienie stanów: wyjść, liczników oraz zegarów.

Organizacja pamięci wewnętrznej sterownika PLC



Po doprowadzeniu napięcia zasilającego do sterownika zmienne bez pamięci stanu (*nonretentive*) umieszczone w pamięci EEPROM są z niej kopiowane do pamięci RAM.
Jeżeli w pamięci RAM umieszczone były zmienne z pamięcią (*retentive*) oraz w chwili doprowadzenia napięcia zawartość pamięci RAM była podtrzymana przez superkondensatory, zawartość tej pamięci nie ulega zmianie, natomiast jeżeli pamięć RAM utraciła te dane, to zmienne z pamięcią (*retentive*) są kopiowane do niej z pamięci EEPROM.

Cykl programowy programowalnych sterowników logicznych



Każdy cykl programowy rozpoczyna się obsługą wejść polegającą na odczytaniu aktualnych stanów na wejściach sterownika i wpisaniu ich do rejestrów wejściowych. Na tej podstawie rozpoczyna się proces wykonania części logicznej programu sterującego. Po wykonaniu programu PLC realizuje proces komunikacji poprzez port komunikacyjny z programatorem lub modułami zewnętrznymi oraz przeprowadza samodiagnostykę. Ostatnią fazą cyklu programowego jest obsługa wyjść polegająca na uaktualnieniu stanu wyjść, gdyż efektem wykonania programu może być zmiana wartości rejestrów wyjściowych, którym przypisuje się fizyczne wyjścia sterownika. *Czas wykonania cyklu zależy od rozmiaru programu użytkowego, ilości użytych wejść i wyjść sterownika i ilości danych wymienianych podczas komunikacji*.

Cykl programowy programowalnych sterowników logicznych

1. **Czytanie wejść**

***Wejścia cyfrowe*: każdy cykl programu zaczyna się od odczytu bieżącej wartości wejść cyfrowych a następnie zapisania tych wartości do tzw. ”obrazu wejść procesu”.**

***Wejścia analogowe*: sterownik odczytuje na początku każdego cyklu wartości analogowych wejść z modułów dodatkowych, o ile filtrowanie danego wejścia analogowego jest wyłączone. Filtr analogowy jest używany w celu poprawienia stabilności sygnału.**

Kiedy funkcja analogowego filtrowania jest aktywna dla danego wejścia, to sterownik odczytuje jego wartość w każdym cyklu i zapisuje przefiltrowaną wartość do zmiennej wewnętrznej. CPU pobiera tą wartość do dalszego przetwarzania programu.

Cykl programowy programowalnych sterowników logicznych

2. Wykonywanie programu

Podczas wykonywania cyklu programu sterownik przetwarza instrukcje programu począwszy od pierwszej linii, aż do ostatniej. Instrukcje odczytu *bezpośredniego* umożliwiają odczyt lub zapis wejść/wyjść binarnych z pominięciem cyklu programu lub procedury przerwań.

Procedury obsługi przerwań nie są wykonywane jako część cyklu programu lecz egzekwowane są dopiero w przypadku wystąpienia zdarzenia wywołującego przerwanie.

Cykl programowy programowalnych sterowników logicznych

3. Obsługa komunikacji

Podczas przetwarzania programu (cyklu programu) sterownik przetwarza każdy komunikat odebrany z portu komunikacyjnego lub inteligentnego modułu wejść/wyjść.

4. Obsługa diagnostyki auto-testu jednostki CPU

Podczas trwania tej fazy cyklu programowego sterownik sprawdza prawidłowość funkcjonowania CPU oraz status każdego z modułu rozszerzeń.

Cykl programowy programowalnych sterowników logicznych

5. Zapisywanie do wyjść

Pod koniec każdego cyklu programu sterownik wpisuje wartości zachowane w *wyjściowym obrazie procesu* do fizycznych wyjść cyfrowych (wyjścia analogowe są uaktualniane natychmiastowo, niezależnie od cyklu programu).

Specyfikacja techniczna sterowników rodziny Simatic S7-200 firmy Siemens

Część 2

Table 1-1 Comparison of the S7-200 CPU Models

Feature	CPU 221	CPU 222	CPU 224	CPU 224XP	CPU 226
Physical size (mm)	90 x 80 x 62	90 x 80 x 62	120.5 x 80 x 62	140 x 80 x 62	190 x 80 x 62
Program memory: with run mode edit without run mode edit	 4096 bytes 4096 bytes	 4096 bytes 4096 bytes	 8192 bytes 12288 bytes	 12288 bytes 16384 bytes	 16384 bytes 24576 bytes
Data memory	2048 bytes	2048 bytes	8192 bytes	10240 bytes	10240 bytes
Memory backup	50 hours typical	50 hours typical	100 hours typical	100 hours typical	100 hours typical
Local on-board I/O Digital Analog	 6 In/4 Out -	 8 In/6 Out -	 14 In/10 Out -	 14 In/10 Out 2 In/1 Out	 24 In/16 Out -
Expansion modules	0 modules	2 modules[1]	7 modules[1]	7 modules[1]	7 modules[1]
High-speed counters Single phase Two phase	 4 at 30 kHz 2 at 20 kHz	 4 at 30 kHz 2 at 20 kHz	 6 at 30 kHz 4 at 20 kHz	 4 at 30 kHz 2 at 200 kHz 3 at 20 kHz 1 at 100 kHz	 6 at 30 kHz 4 at 20 kHz
Pulse outputs (DC)	2 at 20 kHz	2 at 20 kHz	2 at 20 kHz	2 at 100 kHz	2 at 20 kHz
Analog adjustments	1	1	2	2	2
Real-time clock	Cartridge	Cartridge	Built-in	Built-in	Built-in
Communications ports	1 RS-485	1 RS-485	1 RS-485	2 RS-485	2 RS-485
Floating-point math	Yes				
Digital I/O image size	256 (128 in, 128 out)				
Boolean execution speed	0.22 microseconds/instruction				

Cykl programu w S7-200

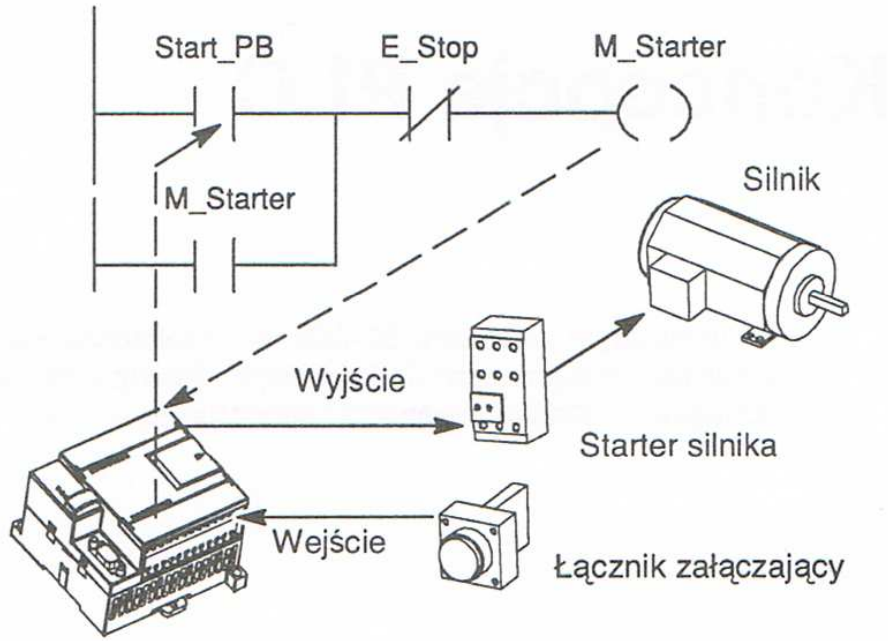
S7-200 w sposób cykliczny wykonuje pętle programowe odczytując i zapisując dane.

S7-200 odnosi program do fizycznych wejść i wyjść

Podstawowe działanie S7-200 jest bardzo proste:

- S7-200 czyta status wejść.
- Program, który jest zapisany w S7-200 używa tych wejść aby realizować logikę sterującą. Podczas pracy programu S7-200 uaktualnia dane.
- S7-200 zapisuje dane do wyjść.

Rysunek pokazuje prosty diagram działania elektrycznego przekaźnika przeniesionego na logikę S7-200. W przykładzie tym, stan łącznika załączającego silnik jest wpleciony w logikę stanów innych wejść sterownika. Przetworzenie wg logiki programu stanów wejść, określa stan wyjścia, które ma bezpośredni wpływ na urządzenie wykonawcze załączające silnik (przekaźnik, stycznik).

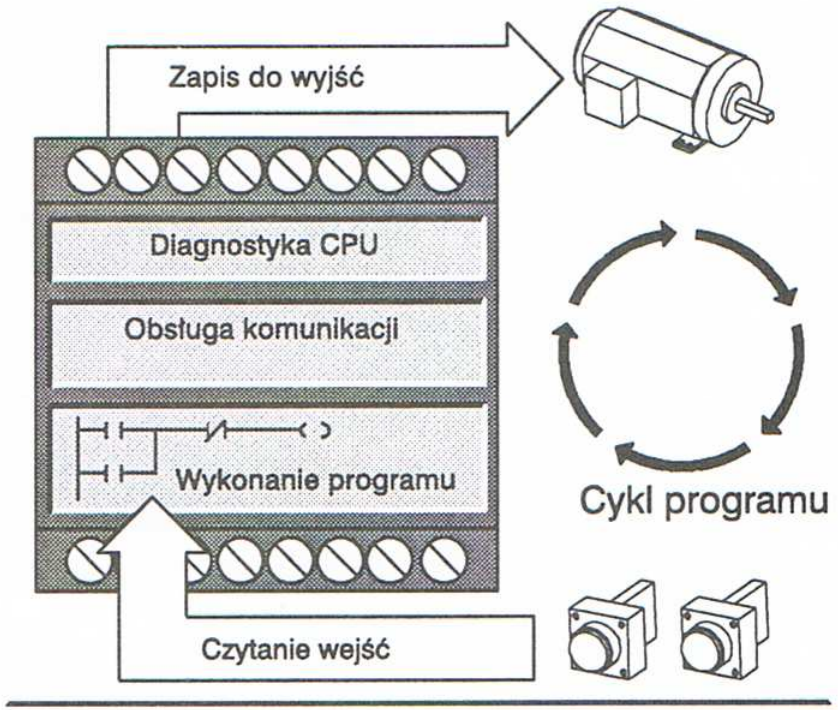


Sterowanie wejściami i wyjściami

S7-200 wykonuje zadania w pętlach programowych (cykl programu)

S7-200 wykonuje serię zadań w sposób cykliczny. Jak pokazano na rysunku 4-2, S7-200 wykonuje większość lub wszystkie wymienione zadania w trakcie trwania cyklu programu:

- Czytanie wejść: S7-200 kopiuje stan wejść fizycznych do obrazu wejść procesu
- Realizuje logikę sterującą w programie: S7-200 wykonuje instrukcje programu i zapisuje wartości w różnych obszarach pamięci.
- Realizuje żądania komunikacji: S7-200 wykonuje wszystkie żądania związane z komunikacją.
- Wykonuje diagnostykę CPU: S7-200 sprawdza, czy oprogramowanie systemowe (firmware), pamięć programu i moduły rozszerzeń pracują poprawnie.
- Zapisywanie do wyjść: wartości zapisane w obrazie wyjść procesu, są przekazywane na wyjścia fizyczne sterownika.



Rysunek 4-2 Cykl programu S7-200

Wykonanie programu użytkownika zależy od trybu pracy S7-200. W trybie RUN program jest wykonywany cyklicznie a w trybie STOP sterownik nie przetwarza żadnych instrukcji.

Czytanie wejść – ogólne uwagi (S7-200)

Czytanie wejść

Wejścia cyfrowe: Każdy cykl programu zaczyna się od odczytu bieżącej wartości wejść cyfrowych a następnie zapisania tych wartości do tzw. "obrazu wejść procesu" .

Wejścia analogowe: S7-200 odczytuje na początku każdego cyklu wartość analogowych wejść z modułów dodatkowych (rozszerzających - SMxxx) o ile filtrowanie danego wejścia analogowego jest wyłączone. Filtr analogowy jest używany aby poprawić stabilność sygnału. Filtr analogowy może być uaktywniony dla każdego z wejść analogowych.

Kiedy funkcja analogowego filtrowania jest wybrana dla danego wejścia, S7-200 odczytuje jego wartość w każdym cyklu i zapisuje przefiltrowaną wartość do zmiennej wewnętrznej. CPU pobiera tą wartość do dalszego przetwarzania programu.

Kiedy filtracja analogowa nie jest aktywna, S7-200 czyta wartości z wejścia analogowego modułu rozszerzenia za każdym razem kiedy program żąda dostępu do danego wejścia analogowego.

Wskazówka

Filtracja wejść analogowych zapewnia odczyt znacznie bardziej stabilnego sygnału wejściowego. Zaleca się stosowanie filtracji analogowej w aplikacjach wykorzystujących wolnozmienne sygnały. Jeśli sygnał jest typu szybkozmiennego, nie powinno się uaktywniać filtracji analogowej.

Nie używaj filtru analogowego z modułami które przenoszą informację cyfrową lub wywołują alarmy w słowach analogowych. Zawsze wyłączaj filtrację analogową dla RTD, termopar, modułów AS-Interface Master.

Środowisko Step7-Micro/Win jako narzędzie uruchomieniowe systemów bazujących na sterownikach PLC z rodziny SIMATIC S7-200 firmy Siemens

Charakterystyka środowiska Step7-Micro/Win:

- zintegrowane funkcje On-line,
- pomoc kontekstowa On-line dla wszystkich funkcji,
- przejrzysta i użyteczna symbolika w tabeli symboli,
- strukturalne oprogramowanie z wykorzystaniem bibliotek,
- strukturalne programowanie z wykorzystaniem podprogramów.

Dostępne edytory pozwalające na tworzenie aplikacji użytkownika:

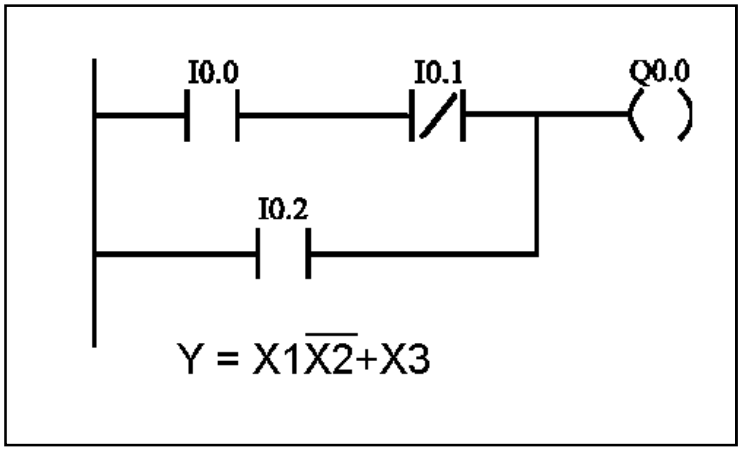
STL - lista instrukcji,

LAD - drabinka instrukcji,

FBD - schemat blokowy.

Język drabinkowy (LAD)

Część logiczna programu sterującego składa się z umieszczonych jeden pod drugim tzw. szczebli programowych. Przypominają one typowy elektryczny schemat połączeń. W skład szczebla wchodzą: *elementy logiczne* (*styki*), *przekaźniki*, jak i bardziej złożone *bloki funkcyjne*. Schemat drabinkowy posiada symboliczne *źródło zasilania*. Zakłada się przepływ sygnału od szyny umieszczonej po lewej stronie schematu do przekaźników lub bloków funkcyjnych umieszczonych po prawej stronie danego szczebla. Kolejne szczeble drabiny odczytywane są kolejno od góry do dołu. Po dojściu do ostatniego szczebla proces śledzenia programu rozpoczyna się od początku



Struktura szczebla drabiny logicznej

Szczebel drabiny logicznej (*Network*) musi mieć odpowiedni format i składnię;

- każdy szczebel może zawierać do 16 linii równoległych, każda linia może natomiast zawierać do 16 elementów logicznych połączonych szeregowo,

- ostatnim elementem szeregowego połączenia w danym szczeblu musi być jeden z przekaźników lub blok funkcyjny,

- szczebel może zawierać maksymalnie do 16 przekaźników,

- szczebel musi zawierać przynajmniej jeden styk przed wystąpieniem przekaźnika, bloku funkcyjnego lub połączenia pionowego,

- nie może wystąpić rozgałęzienie mające początek lub koniec wewnątrz innego odgałęzienia.

Numeracja wejść/wyjść sterownika

Wybierając w programie sterującym określone wejście (I) lub wyjście (Q) sterownika należy podać numer identyfikacyjny (I/O) określający jednocześnie adres w pamięci wewnętrznej CPU/ W poniższej tabeli przedstawiono sposób adresowania wejść i wyjść dla sterowników S7-200 z CPU 212 i 214

numer wejścia/wyjścia	Rodzaj jednostki centralnej sterownika S7			
	CPU - 212		CPU - 214	
	wejście	wyjście	wejście	wyjście
pierwsze	I0.0	Q0.0	I0.0	Q0.0
drugie	I0.1	Q0.1	I0.1	Q0.1
trzecie	I0.2	Q0.2	I0.2	Q0.2
czwarte	I0.3	Q0.3	I0.3	Q0.3
piąte	I0.4	Q0.4	I0.4	Q0.4
szóste	I0.5	Q0.5	I0.5	Q0.5
siódme	I0.6		I0.6	Q0.6
ósme	I0.7		I0.7	Q0.7
dziewiąte			I1.0	Q1.0
dziesiąte			I1.1	Q1.1
Jedenaste			I1.2	
Dwunaste			I1.3	
Trzynaste			I1.4	
Czternaste			I1.5	
Adresy wyjść do wykorzystania jako zmienne wewnętrzne (bity)				
		Q0.6		Q1.2
		Q0.7		Q1.3
				Q1.4
				Q1.5
				Q1.6
				Q1.7

Oznaczenia literowe identyfikatorów

Identyfikator	
oznaczenie	nazwa
I	zmienna wejściowa
Q	zmienna wyjściowa
M	wewnętrzna zmienna dyskretna
SM	wewnętrzna zmienna specjalna (zmienna systemowa)
V	zmienne pamięciowe
T	timer
C	licznik
AI	zmienna wejściowa analogowa
AQ	zmienna wyjściowa analogowa
AC	akumulator
HC	szybki licznik
K	stała

Obraz wejść procesu: I
S7-200 próbkuje fizyczne wejścia na początku cyklu programu i zapisuje je do rejestru obrazu wejść procesu.

Obraz wyjść procesu: Q
Na końcu cyklu programu S7-200 kopiuje wartości zapisane w rejestrze obrazu wyjść procesu do wyjść fizycznych.

Obszar pamięci danych: V
Pamięć typu V może być użyta do przechowywania pośrednich wyników operacji programu wykonywanych przez CPU.

Obszar pamięci o dostępie bitowym: M
Pamięć typu M może być użyta jako obszar znaczników do zapamiętania stanów binarnych wyników operacji logicznych.

Timerowy obszar pamięci: T
S7-200 udostępnia timery, które zliczają impulsy czasowe z rozdzielczością 1ms, 10ms oraz 100ms.

Z timerem związane są dwie zmienne:
- wartość bieżąca: 16-bitowa wartość całkowita ze znakiem zawierająca ilość czasu zliczonego przez timer,
- bit timera: bit ten jest ustawiany lub zerowany jako wynik porównania wartości bieżącej z wartością odniesienia timera. Wartość odniesienia jest wprowadzana jako parametr instrukcji timera (PT) i stanowi krotność bazowego czasu, który odmierza timer.

Licznikowy obszar pamięci: C
S7-200 udostępnia trzy typy liczników, które zliczają zmiany stanów (0 na 1) na ich wejściach. Są
Trzy rodzaje liczników:
- zliczające tylko w górę,
- zliczające tylko w dół,
- zliczające w obie strony (rewersyjne).

Z licznikiem związane są dwie zmienne:
- wartość bieżąca: 16-bitowa wartość całkowita ze znakiem zawierająca zliczoną wartość,
- bit licznika: bit ten jest ustawiany lub zerowany jako wynik porównania wartości bieżącej z wartością odniesienia licznika. Wartość odniesienia jest wprowadzana jako parametr instrukcji licznika (PV)

Akumulatory: AC
Akumulatory są uniwersalnymi rejestrami odczytu i zapisu, które mogą być użyte podobnie jak pamięć. Przykładowo akumulatorów można użyć do przekazania parametrów do i z programu oraz zachowania pośrednich wartości użytych w obliczeniach.

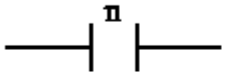
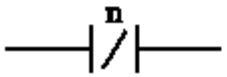
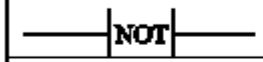
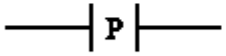
Pamięć specjalna: SM
Bity SM są bitami systemowymi sterownika S7-200 i ułatwiają komunikację pomiędzy CPU a programem użytkownika. Można tych bitów użyć do diagnostyki lub kontroli pracy sterownika.

Typy zmiennych

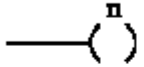
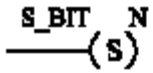
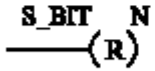
zmienna rejestrowa	liczba całkowita bez znaku		liczba całkowita ze znakiem	
	zapis dziesiętny	zapis szesnastkowy	zapis dziesiętny	zapis szesnastkowy
B (bajt - dana zawierająca 8 bitów)	0 do 255	0 do FF	-128 do +127	80 do 7F
W (słowo - dana awierająca 16 kolejnych bitów)	0 do 65535	0 do FFFF	-32.768 do +32.767	8000 do 7FFF
D (podwójne słowo - dana zawierająca 32 kolejne bity)	0 do 4.294.967.295	0 do FFFF FFFF	-2.147.483.648 do +2.147.483.647	8000 0000 do 7FFF FFFF

Nazwa kategorii	Zawartość
Elementy stykowe *(Contacts)*	styki normalnie otwarte/zamknięte, komparatory, styk negacji, styki impulsowe.
Przekaźniki wyjściowe *(Output Coils)*	przekaźnik o stykach otwartych, przekaźniki ustawialne SET/RESET.
Timery i liczniki *(Timers/Counters)*	timery z i bez pamięci, liczniki zliczające w górę oraz w górę i w dół.
Operacje matematyczne *(Math/Inc/Dec)*	funkcje dodawania, odejmowania, mnożenia i dzielenia liczb, funkcja pierwiastka kwadratowego, zwiększanie/ zmniejszanie wartości o 1.
Kopiowanie, przesuwanie, obrót *(Move/Shift/Rotate/Fill)*	kopiowanie zmiennych, przesuwanie zmiennych w prawo/lewo, obrót (rotacja) zmiennych w prawo/lewo.
Funkcje sterujące *(Program Control)*	funkcje końca lub zatrzymania programu, funkcje obsługi podprogramów funkcje skoków programowych
Operacje logiczne *(Logical Operations)*	iloczyn, suma logiczna słów, alternatywa wyłączająca słów (XOR - albo), inwersja słów.
Konwersja *(Conversion)*	zamiana danych BCD-4/liczbę całkowitą i odwrotnie, zamiana kodu ASCII na Hex i odwrotnie, moduł wyświetlacza 7 - segmentowego, inne.
Szybkie operacje *(High Speed Operations)*	definiowanie parametrów szybkich liczników, wyjście impulsowe (tylko dla CPU 214).
Zegar czasu rzeczywistego *(Real Time Clock)*	odczyt aktualnej daty i czasu (rejestr 8-bajtowy), ustawianie powyższych parametrów.
Linie *(Lines)*	linia pozioma, linia pionowa.
Operacje tablicowe *(Table/Find)*	Wpisywanie do tablicy danych, wyprowadzanie danych z tablicy, wyszukiwanie w tablicy określonych danych
Przerwania i komunikacja *(Interrupt/Comunications)*	Bloki funkcyjne i przekaźniki obsługujące procedury przerwań programowych, bloki funkcyjne obsługujące pracę sieciową sterowników
Wszystkie kategorie *(All Categories)*	wszystkie elementy i bloki funkcjonalne poszczególnych kategorii zgrupowane w jedną w porządku alfabetycznym.

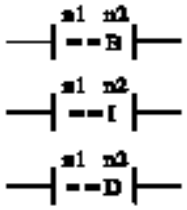
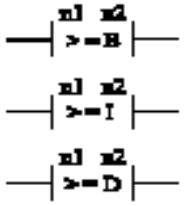
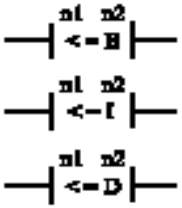
Rodzaje styków

LAD	Opis	Zmienna
	Styk normalnie otwarty. Przewodzi sygnał (zwiera styki), gdy wartość logiczna przypisanej zmiennej wynosi "1". **(Normally Open)**	n: I, Q, M, SM, T, C, V (bit)
	Styk normalnie zamknięty. Przewodzi sygnał (zwiera styki), gdy wartość logiczna przypisanej zmiennej wynosi "0". **(Normally Closed)**	
	Styk negacji. Gdy dochodzi do niego sygnał jego styk jest otwarty. **(NOT)**	bez dodatkowych oznaczeń
	Styk zwierny na czas jednego cyklu pracy sterownika, gdy sygnał podany do tego styku zmienia wartość z "0" na "1". (styk impulsowy) **(Positiv Transition)**	
	Styk zwierny na czas jednego cyklu pracy sterownika, gdy sygnał podany do tego styku zmienia wartość z "1" na "0". (styk impulsowy) **(Negative Transition)**	

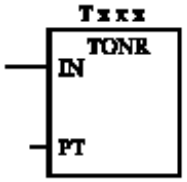
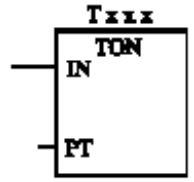
Rodzaje przekaźników

LAD	Opis	Zmienna
	Przekaźnik ustawia wartość przypisanej zmiennej na "1", gdy podany zostanie do niego sygnał. Jest to przekaźnik o stykach otwartych. **(Output)**	n: I, Q, M, SM, T, C, V (bit)
	Przekaźnik ustawialny "SET". Przekaźnik ustawia na "1" kilka przypisanych mu zmiennych dyskretnych, których liczbę określa parametr "N". Zmienna S_BIT określa adres początkowy tych zmiennych. Jest to przekaźnik z pamięcią, gdyż jego stan jest podtrzymywany w przypadku wyłączenia zasilania sterownika lub po przejściu sterownika w tryb STOP. Wartość "1" jest utrzymywana do momentu, aż sygnał podany zostanie do przekaźnika "RESET". **(Set)**	S_BIT:I, Q, M, SM, T, (bit) C, V N:IB, QB, MB, SMB, (bajt) VB, AC, K
	Przekaźnik ustawialny "RESET" przystosowany do współpracy z przekaźnikiem "SET". Gdy do przekaźnika podany zostanie sygnał wartość przypisanej mu zmiennej S_BIT oraz kilka kolejnych zmiennych dyskretnych określonych parametrem "N" ustawiona zostanie na "0". Jest to przekaźnik z pamięcią. **(Reset)**	

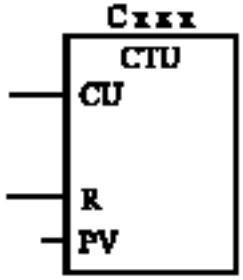
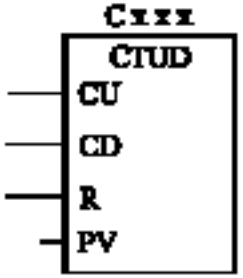
Styki komparatorów

LAD	Opis	Zmienna
	Zestyk jest zwarty (przewodzi sygnał), gdy: n1 = n2 B = bajt (zmienna 8 bitowa) I = liczba całkowita ze znakiem (zmienna 16 bitowa) D = liczba całkowita ze znakiem (zmienna 32 bitowa) R = wartość rzeczywista **(= = Byte), (= = Word Integer), (= = Double Word Integer)**	n1, n2: VB,IB, QB, MB, (bajt) SMB, AC, K n1, n2: VW, T, C, IW, QW, (słowo) MW, AC, SMW, AIW, K n1, n2: VD, ID, QD, K, (32bity) MD, SMD, AC, HC
	Zestyk jest zwarty (przewodzi sygnał), gdy: n1 > = n2 B = bajt (zmienna 8 bitowa) I = liczba całkowita ze znakiem (zmienna 16 bitowa) D = liczba całkowita ze znakiem (zmienna 32 bitowa) R = wartość rzeczywista **(> = Byte), (> = Word Integer), (> = Double Word Integer)**	
	Zestyk jest zwarty (przewodzi sygnał), gdy: n1 < = n2 B = bajt (zmienna 8 bitowa) I = liczba całkowita ze znakiem (zmienna 16 bitowa) D = liczba całkowita ze znakiem (zmienna 32 bitowa) R = wartość rzeczywista **(< = Byte), (< = Word Integer), (< = Double Word Integer)**	

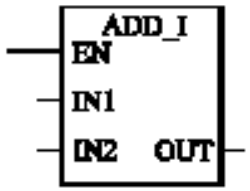
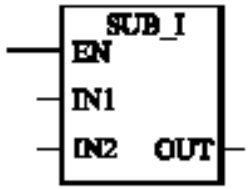
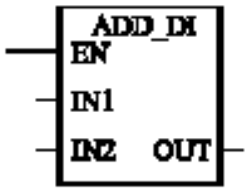
Timery

LAD	Opis	Zmienna
	Timer zlicza czas, gdy podany zostanie do niego sygnał "IN" i zatrzymuje naliczoną wartość, gdy sygnał jest nieaktywny. Po ponownym pojawieniu się tego sygnału zliczanie czasu jest kontynuowane. Może zliczyć maksymalnie +32767 jednostek czasu. Gdy wartość bieżąca zrówna się z wartością zadaną "PT" timer zostaje uaktywniony, tzn. bit wyjściowy timera "Txxx" ustawia się na "1". **(Timer - Retentive On Delay)** CPU 212/214 — CPU 214 1ms T0 — T0 - T64 10ms T1 - T4 — T65 - T68 100ms T5 - T31 — T69 - T95	Txxx: CPU 212: 0 - 31 (słowo)CPU 214: 0 -1, 64 - 95 PT: VW, T, C, IW, QW, (słowo) MW, SMW, AC, AIW, K
	Timer zlicza czas, gdy podany zostanie do niego sygnał "IN" i zostaje wyzerowany, gdy sygnał przestaje być aktywny. Po pojawieniu się tego sygnału zliczanie czasu rozpoczyna się od początku. Zakres zliczanych jednostek czasu: +32767. Gdy wartość bieżąca zrówna się z wartością zadaną "PT" timer zostaje uaktywniony, tzn. bit wyjściowy timera "Txxx" ustawia się na "1". **(Timer - On Delay)** CPU 212/214 — CPU 214 1ms T32 — T96 10ms T33 - T36 — T97 - T100 100ms T37 - T63 — T101 - T127	Txxx: CPU 212: 32 - 63 (słowo)CPU 214: 32 - 63, 96 - 127 PT:VW, T, C, IW, QW, (słowo) MW, SMW, AC, AIW, K

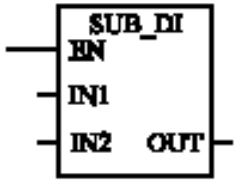
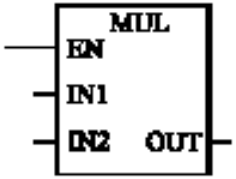
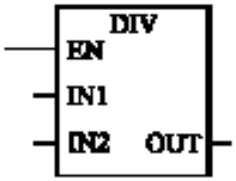
Liczniki

LAD	Opis	Zmienna
	Licznik zliczający w górę służy do zliczania impulsów (zbocze narastające) doprowadzonych do wejścia "CU". Zakres licznika wynosi 32767imp. Licznik jest wyzerowany tak długo, jak długo do wejścia kasującego "R" podany zostaje sygnał. Przy zrównaniu się liczby zliczanych impulsów z wartością zadaną "PV" licznik zostaje uaktywniony, tzn. bit wyjściowy licznika "Cxxx" ustawia się na "1". **(Count Up)**	Cxxx:CPU 212: 0 - 63 (słowo)CPU 214:0-127 PV:VW, T, C, IW, QW, (słowo) MW, SMW, AC, AIW, K
	Licznik umożliwia zliczanie w górę impulsów (zbocze narastające) doprowadzonych do wejścia "CU" lub w dół - wejście "CD". Zakres zliczanych impulsów wynosi: -32768 do +32767. Licznik jest wyzerowany tak długo, jak długo do wejścia kasującego "R" podany zostaje sygnał. Przy zrównaniu się liczby zliczanych impulsów z wartością zadaną "PV" licznik zostaje uaktywniony, tzn. bit wyjściowy licznika "Cxxx" ustawia się na "1". **(Count Up/Down)**	Cxxx: CPU 212: 0 - 63 (słowo)CPU 214: 0 -27, PV: VW, T, C, IW, QW, (słowo) MW, SMW, AC, AIW, K

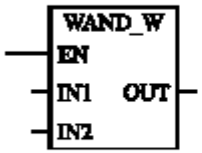
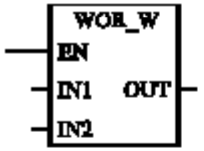
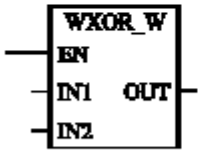
Wybrane operacje matematyczne

LAD	Opis	Zmienna
	Blok funkcyjny umożliwiający dodawanie dwóch liczb całkowitych 16 bitowych ze znakiem (Signet Integer) w zakresie: -32768 do +32767 doprowadzonych do wejść: "IN1" oraz "IN2". Wynik działania , jako zmienna 16 bitowa, wyprowadzony jest na wyjście "OUT". Operacja zostanie wykonana, gdy do wejścia "EN" podany zostaje sygnał./[1,2,3] **(Add Integer)**: IN1 + IN2 = OUT	I N1, IN2: VW, T, C, IW, (słowo) QW, MW, SMW, AC, AIW, K OUT: VW, T, C, IW, (słowo) QW, MW, SMW, AC
	Blok funkcyjny umożliwiający odejmowanie dwóch liczb całkowitych 16 bitowych ze znakiem (Signet Integer) w zakresie: -32768 do +32767 doprowadzonych do wejść: "IN1" oraz "IN2". Wynik działania , jako zmienna 16 bitowa, wyprowadzony jest na wyjście "OUT". Operacja zostanie wykonana, gdy do wejścia "EN" podany zostaje sygnał./[1,2,3] **(Subtract Integer)**: IN1 - IN2 = OUT	
	Blok funkcyjny umożliwiający dodawanie dwóch liczb całkowitych 32 bitowych ze znakiem (Signet Integer) w zakresie: -2147483648 do +2147483647 doprowadzonych do wejść: "IN1"oraz "IN2". Wynik działania , jako zmienna 32 bitowa, wyprowadzony jest na wyjście "OUT". Operacja zostanie wykonana, gdy do wejścia "EN" podany zostaje sygnał./[1,2,3] **(Add Double Integer)**: IN1 + IN2 = OUT	IN1, IN2: VD, ID, QD, MD, (32 bity) SMD, AC, HC, K OUT: VD, ID, QD, MD, (32 bity) SMD, AC

Wybrane operacje matematyczne

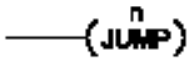
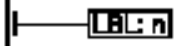
	Blok funkcyjny umożliwiający odejmowanie dwóch liczb całkowitych 32 bitowych ze znakiem (Signet Integer) w zakresie: -2147483648 do +2147483647 doprowadzonych do wejść: "IN1"oraz "IN2". Wynik działania , jako zmienna 32 bitowa, wyprowadzony jest na wyjście "OUT". Operacja zostanie wykonana, gdy do wejścia "EN" podany zostaje sygnał./1,2,3 **(Subtract Double Integer)**: IN1 - IN2 = OUT	IN1, IN2: VD, ID, QD, MD, (32 bity) SMD, AC, HC, K OUT: VD, ID, QD, MD, (32 bity) SMD, AC
	Blok funkcyjny umożliwiający mnożenie dwóch liczb całkowitych 16 bitowych ze znakiem (Signet Integer) w zakresie: -32768 do +32767 doprowadzonych do wejść: "IN1"oraz "IN2". Wynik działania , jako zmienna 32 bitowa, wyprowadzony jest na wyjście "OUT". Operacja zostanie wykonana, gdy do wejścia "EN" podany zostaje sygnał./2,3 **(Multiply Integer)**: IN1 * IN2 = OUT	IN1, IN2: VW, T, C, IW, (słowo) QW, MW, SMW, AC, AIW, K OUT: VD, ID, QD, MD, (32 bity) SMD, AC <u>Uwaga</u>: *Część całkowita ilorazu przechowywana jest w pierwszych 16 mniej znaczących bitach, a reszta w kolejnych 16 bar- dziej znaczących bitach zmiennej wyjściowej 32 bitowej.*
	Blok funkcyjny umożliwiający dzielenie dwóch liczb całkowitych 16 bitowych ze znakiem (Signet Integer) w zakresie: -32768 do +32767 doprowadzonych do wejść: "IN1"oraz "IN2". Wynik działania , jako zmienna 32 bitowa, wyprowadzony jest na wyjście "OUT". Operacja zostanie wykonana, gdy do wejścia "EN" podany zostaje sygnał. /1,2,3 **(Divide Integer)**: IN1 / IN2 = OUT	

Wybrane operacje logiczne

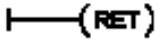
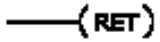
LAD	Opis	Zmienna
	AND (I) - koniunkcja dwóch słów 16 bitowych bez znaku (zakres od 0 do 65535) doprowadzonych do wejść: "IN1" oraz "IN2". Operacja zostaje wykonana po doprowadzeniu sygnału do wejścia "EN", a wynik przypisany do wyjścia "OUT". Operacja ta jest wykonywana dla każdej pary bitów o tych samych wagach począwszy od najmniej znaczących bitów./1 **(AND Word)**	IN1, IN2: VW, T, C, IW, (słowo) QW, MW, SMW, AC, AIW, K OUT:VW, T, C, IW, (słowo) QW, MW, SMW, AC
	OR (LUB) - alternatywa dwóch słów 16 bitowych bez znaku (zakres od 0 do 65535) doprowadzonych do wejść: "IN1" oraz "IN2". Operacja zostaje wykonana po doprowadzeniu sygnału do wejścia "EN", a wynik przypisany do wyjścia "OUT". Operacja ta jest wykonywana dla każdej pary bitów o tych samych wagach począwszy od najmniej znaczących bitów./1 **(OR Word)**	
	XOR (WYŁĄCZNIE - LUB) - suma modulo 2 dwóch słów 16 bitowych bez znaku (zakres od 0do 65535) doprowadzonych do wejść: "IN1" oraz "IN2". Operacja zostaje wykonana po doprowadzeniu sygnału do wejścia "EN", a wynik przypisany do wyjścia "OUT". Operacja ta jest wykonywana dla każdej pary bitów o tych samych wagach począwszy od najmniej znaczących bitów./1 **(XOR Word)**	IN1, IN2: VW, T, C, IW, (słowo) QW, MW, SMW, AC, AIW, K OUT: VW, T, C, IW, (słowo) QW, MW, SMW, AC

Wybrane funkcje sterujące

Funkcje związane ze strukturą programu

STL	Opis	Zmienna
	Przekaźnik skoku warunkowego do podprogramu o adresie "n". Powoduje ona pominięcie części programu sterującego umiesz- czonego pomiędzy instrukcją "JUMP n", a etykietą "LABEL n". Instrukcja zostanie wykonana, gdy do przekaźnika podany zostaje sygnał. **Jump to Label (JMP)**	n = (0 do 63) dla CPU 212 n = (0 do 255) dla CPU 214
	Przekaźnik określający miejsce docelowe "n" skoku wywołanego przekaźnikiem "JUMP n" z tą samą etykietą i powoduje dalszą kontynuację wykonywania programu począwszy od tej etykiety. **Label (LBL)**	n = (0 do 63) dla CPU 212 n = (0 do 255) dla CPU 214

Wybrane funkcje sterujące

	Przekaźnik bezwarunkowego powrotu z podprogramu. Jest ostatnim elementem każdego podprogramu kończąc jego wykonywanie. **Return from Subroutine (RET)**	bez oznaczeń
	Przekaźnik warunkowego powrotu z podprogramu. Przekaźnik może być wykorzystany do opuszczania danego podprogramu. Może również wystąpić przed przekaźnikiem bezwarunkowego zakończenia podprogramu "RET". Instrukcja zostanie wykonana, gdy do przekaźnika podany zostaje sygnał. **Return from Subroutine (RET)**	bez oznaczeń
	Przekaźnik zakończenia wykonywania części logicznej programu. Przekaźnik spowoduje zatrzymanie wykonywania programu w miejscu, w którym występuje i rozpoczęcie cyklu skanowania od początku. Instrukcja zostanie wykonana, gdy do przekaźnika podany zostaje sygnał. **(END)**	bez oznaczeń
	Przekaźnik o działaniu bezwarunkowym. Jest ostatnim elementem programu sterującego. Powoduje rozpoczęcie przez sterownik skanowania programu od początku. **(END)**	bez oznaczeń
	Przekaźnik ten kończy wykonywanie programu i powoduje natychmiastowe przejście sterownika w tryb STOP. Instrukcja zostanie wykonana, gdy do przekaźnika podany zostaje sygnał. **(STOP)**	bez oznaczeń

Tryby pracy sterownika S7-200

S7-200 posiada dwa tryby pracy: STOP i RUN. Statusowe diody LED na przednim panelu CPU wskazują bieżący tryb pracy. W trybie STOP, S7-200 nie wykonuje programu i wtedy można załadować program lub konfiguracje CPU. W trybie RUN, S7-200 wykonuje instrukcje programu.

- S7-200 posiada przełącznik trybu pracy. Przełącznika trybu pracy (umieszczonego pod frontowa klapką S7-200) można użyć do ręcznej zmiany trybu pracy: ustawiając przełącznik na STOP zatrzymuje się wykonywanie programu; ustawiając przełącznik na RUN rozpoczyna się wykonywanie programu a ustawiając przełącznik na TERM (terminal) nie zmienia się trybu pracy.

  Jeśli występuje zanik zasilania kiedy przełącznik trybu pracy ustawiony jest na STOP lub TERM, S7-200 automatycznie przechodzi do trybu STOP po przywróceniu zasilania. Jeśli występuje zanik zasilania kiedy przełącznik trybu pracy ustawiony jest na RUN, S7-200 automatycznie wchodzi do trybu RUN po przywróceniu zasilania.

- STEP 7-Micro/WIN pozwala na zmianę trybu pracy on-line. Aby uaktywnić programową zmianę trybu pracy należy ręcznie ustawić przełącznik trybu pracy S7-200 na TERM lub RUN. Następnie używając **PLC > STOP** lub **PLC > RUN** z menu poleceń oprogramowania STEP7-Micro/WIN lub odpowiednich przycisków na pasku narzędziowym można zmieniać tryb pracy sterownika.

- Można wstawić w programie instrukcję STOP aby wprowadzić S7-200 w tryb STOP. Pozwala to na zatrzymanie wykonywania instrukcji programu.

PROGRAMOWALNE STEROWNIKI LOGICZNE

WYKŁAD nr 3

- **Tekstowe języki programowania:**
  - **STL – język instrukcji (IL),**
  - **ST – język srtukturalny**
- **Przykładowe aplikacje – test znajomości podstaw języka drabinkowego**

STL – lista instrukcji

Cechą charakterystyczną tego języka jest sekwencja instrukcji podobnych do kodów programowania mikroprocesora w języku niskiego poziomu (assembler).

Instrukcja zawiera:

- operator,
- modyfikator (opcjonalny),
- jeden lub kilka operandów.

Najważniejsze operatory języka STL

oznaczenie	typy zmiennych	opis
LD	Bool, word , integer	"**Załaduj**"Załadowanie wartości zmiennej do aktywnego rejestru (z informatyki odpowiednikiem byłoby ładowanie na stos) .Zmienna występująca zwykle po prawej od wyrażenia może być właściwie dowolnego typu , uzależnione to jest od operatora , który wystąpi po niej.
ST	Bool, word , integer	"**Ustaw**&quotZapamiętanie (przepisanie) wartości z aktywnego rejestru do zmiennej.
S	Bool	"**Ustaw**" Zapamiętanie tzw zatrzaskowe wartości zmiennej "**1**", zmiana stanu możliwa jest tylko przez funkcję Kasuj -> patrz dalej
R	Bool	"**Kasuj**" stan zmiennej jest resetowny do "**0**"
AND	Bool	Operator logiczny "**I**"
OR	Bool	Operator logiczny "**LUB**"
ADD	Bool,word, integer,	Dodawanie
SUB	Bool,word, integer,	Odejmowanie
MUL	Bool,word, integer,	Mnożenie
DIV	Bool,word, integer,	Dzielenie
GT	Bool,word, integer,	Większe niż : ">"
GE	Bool,word, integer,	Większe lub równe : " ≥ "
EQ	Bool,word, integer,	Równe : " = "
NE	Bool,word, integer,	Nie równe : " ≠ "
LE	Bool,word, integer,	Mniejsze lub równe : " ≤ "
LT	Bool,word, integer,	Mniejsze : " < "
( )		Operator zaięgu

Wszystkie użyte symbole i ozanczenia są zgodne z normą IEC 61131-3 dotyczącą normalizacji symboliki i funkcji języków programowania sterowników PLC

Najważniejsze modyfikatory języka STL

- **N (not)** – oznacza logiczną negację operandu logicznego**,**

- **(** – otwarcie nawiasu oznacza, że jako operand użyty jest wynik wykonania wszystkich instrukcji aż do operatora zasięgu, tj. zamknięcia nawiasu,

- **C** - oznacza, że instrukcja jest wykonywana tylko wtedy, gdy wartość bieżąca jest równa logicznej jedynce.

Operandy języka STL

Operandami języka tekstowego STL mogą być stałe lub zmienne różnych typów (logiczne, arytmetyczne, bitowe, łancuchy znakowe).

Fragment programu napisanego w języku STL (sterownik TSX MIRO)

Operator z modyfikatorem	Operand	Komentarz
LD	**%I2.2**	**załadowanie wartości z wejścia I2.2**
AND (	**%I2.1**	**iloczyn logiczny bieżącej wartości załadowanej z wejścia I2.2 z wynikiem wyrażenia objętego nawiasami zewnętrznymi** (pierwszy element tego wyrażenia to: I2.1)
OR (N	**%I1.3**	suma logiczna stanu wejścia I2.1 z wynikiem wyrażenia objętego nawiasami wewnętrznymi
AND	**%M8**	iloczyn logiczny negacji wejścia I1.3 z bitem pamięci M8
)		zasięg wewnętrzny nawiasu
)		zasięg zewnętrzny nawiasu
ST	**%Q1.2**	wysłanie wyniku na wyjście Q1.2

Q1.2 = I2.2AND[I2.1OR(NOTI1.3ANDM8)]

Realizacja wybranej funkcji logicznej za pomocą języka drabinkowego

Q1.2 = I2.2AND[I2.1OR(NOTI1.3ANDM8)]

?

STL – uwaga końcowa

Język STL nie jest wygodny oraz przejrzysty. Daje jednak ogromne możliwości optymalizacji w celu ograniczenia czasu realizacji cyklu programu (właściwość języka niskiego poziomu).

ST – język strukturalny

Język strukturalny opiera się na strukturach języków algorytmicznych wyższego poziomu. Podstawowymi elementami są tutaj zdania logiczne zawierające warunki i polecenia. Warunki i polecenia są natomiast związane są z wyrażeniami. Wyrażenia z kolei składają się z operatorów i operandów.

Pojęcia:
-zdanie logiczne,
-warunek,
-polecenie,
-operator,
-operand.

Podstawowe struktury zdań języka ST

IF (**warunek**) **THEN** **polecenia1** **ELSE** **polecenia2** – w czasie wykonywania programu sprawdzane jest spełnienie warunku występującego po słowie kluczowym **IF**. Jeżeli ten warunek jest spełniony (tzn. skojarzone z warunkiem wyrażenie logiczne przyjmuje wartość 1), to wykonywane są **polecenia1** (występujące po słowie kluczowym **THEN**). W przeciwnym wypadku wykonywane są **polecenia2** (występujące po słowie kluczowym **ELSE**).

CASE (**wybierak**) **OF** (**lista poleceń z etykietami**) **ELSE** (**inne polecenia**) – zdanie to umożliwia wykonywanie różnych poleceń w zależności od wartości, jaką przyjmuje **wybierak**. **Wybierak** jest wyrażeniem typu INTEGER (może przyjmować wartości całkowite). **Lista poleceń** zawiera różne grupy poleceń. Każda z grup ma natomiast swoją **etykietę**. Etykietą może być jedna lub więcej liczb całkowitych albo pewien zakres tych liczb, np..:

liczba 2 może być etykietą,
5,7,9 – kilka liczb może być etykietą,
1do 4 – zakres liczb może być etykietą

W czasie realizacji programu wykonywana będzie ta grupa poleceń, która ma **etykietę** zgodną z wartością **wybieraka**. W przypadku, kiedy wartość wybieraka nie odpowiada żadnej etykiecie wykonywane będą **inne polecenia** (występujące po słowie kluczowym **ELSE**)

Podstawowe struktury zdań języka ST

FOR (**wartość początkowa**) **TO** (**wartość końcowa**) **BY** (**krok**) **DO** (**polecenia**) – struktura służy do zaprogramowania pętli, w której iteracyjne wykonywane są polecenia przy znanej liczbie iteracji. Liczba iteracji jest ściśle określona przez **wartość początkową** jej **wartość końcową** oraz **krok**.

WHILE (**warunek**) **DO** (**polecenia**)
REPEAT (**polecenia**) **UNTIL (warunek)** – zdania te służą do zaprogramowania cyklicznego wykonywania **poleceń** w przypadku, kiedy liczba iteracji nie jest z góry znana. O liczbie iteracji decyduje w tym przypadku spełnienie **warunku** (występujące po słowach odpowiednio: **WHILE** oraz **UNTIL**.

Język strukturalny – ciąg dalszy

W języku strukturalnym oprócz przedstawionych wcześniej struktur zdań wykorzystuje się również elementy innych języków programowania – zwłaszcza służące do definiowania pewnych fragmentów programu, np..:

FUNCTION … END – sposób definiowania funkcji

FUNCTION BLOCK … END – sposób definiowania bloku funkcyjnego

PROGRAM … END – sposób definiowania podprogramu

STEP … END – definiowanie zbioru zdań

TRANSITION … END – definiowanie warunków przejścia między etapami programu

ACTION … END – definiowanie zbioru poleceń

Fragment programu napisanego w języku ST dla sterownika FESTO FPC 202C

```
IF              I1.3
        EXOR    I1.1
THEN    SET     O1.4
ELSE JUMP TO SETUP
```

Komentarz:
Jeżeli wartość I1.3 EXOR I1.1 przyjmuje wartość 1, to wyjście O1.4 ustawione jest w stan logicznej jedynki. W przeciwnym wypadku następuję skok do fragmentu programu o nazwie SETUP.

TRANSMISJA DANYCH

WYKŁAD nr 4

- Standard RS232,
- Przemysłowe protokoły komunikacyjne

Standard RS-232

Standard RS-232 został wprowadzony w 1962 roku w celu normalizacji interfejsu pomiędzy **urządzeniem końcowym dla danych** (**DTE** – Data Terminal Equipment), a **urządzeniem komunikacyjnym dla danych** (**DCE** – Data Communication Equipment). Jakkolwiek główny nacisk położono na zdefiniowanie interfejsu pomiędzy terminalem (urządzeniem DTE) a modemem (urządzeniem DCE), to standard ten jest bardzo często stosowany również przy szeregowej transmisji danych pomiędzy różnymi typami urządzeń DTE.

W standardzie RS-232 transmisja danych odbywa się szeregowo bit po bicie, przy czym definiuje się dwa rodzaje transmisji:

- **asynchroniczna transmisja znakowa**,
- **transmisja synchroniczna**.

Asynchroniczna transmisja znakowa

Asynchroniczna transmisja znakowa polega na przesyłaniu pojedynczych znaków, które posiadają ściśle określony format. Początek znaku stanowi **bit startu**, jałowy z punktu widzenia przesyłanej informacji i służący jedynie celom synchronizacyjnym. Dalej następuje **pole danych**, na które wprowadza się kolejne bity stanowiące treść znaku (począwszy od bitu najmniej znaczącego LSB). Bezpośrednio za polem danych przewidziano **bit kontrolny**, służący zabezpieczeniu informacji znajdującej się na polu danych. Jest to bit, który może ale nie musi wystąpić. Nie należy tego rozumieć tak, że w jednym znaku bit kontrolny zostanie umieszczony, a w następnym nie. Decyzja o jego stosowaniu (lub nie stosowaniu) ma charakter globalny i dotyczy wszystkich znaków. Transmitowany znak kończy jeden lub dwa **bity stopu**.

Zdefiniowany powyżej zespół bitów tworzy „jednostkę informacyjną”. **W ramach jednostki informacyjnej bity przesyłane są synchronicznie**, tzn. zgodnie z taktem nadajnika. Natomiast **poszczególne jednostki są przesyłane asynchronicznie** – ich wyprowadzanie nie jest synchronizowane żadnym sygnałem, a więc odstęp pomiędzy nimi jest dowolny.

Asynchroniczna transmisja znakowa

Czas trwania bitu w jednostce informacyjnej nazywa się odstępem jednostkowym i jest oznaczany przez $\mathbf{t_b}$. Jego odwrotność określa szybkość transmisji **(1bd = 1 bit/s).** Typowe szybkości transmisji przy asynchronicznej transmisji znakowej wynoszą: 1200, 2400, 4800, 9600 bd.

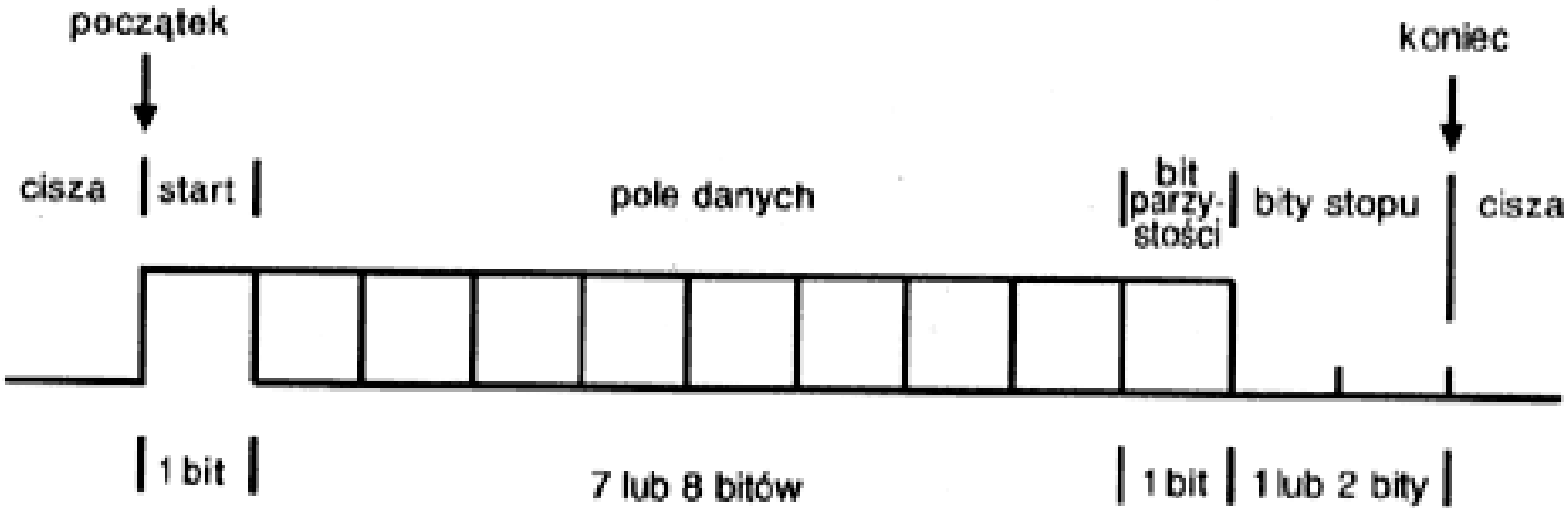
Przy asynchronicznej transmisji znakowej przyjmuje się, że zarówno odbiornik, jak i nadajnik pracują z tą samą częstotliwością, jakkolwiek takty nadawania i odbioru nie są zsynchronizowane. Ze względu na małą długość jednostki informacyjnej, niewielka różnica częstotliwości generatorów taktu w nadajniku i odbiorniku nie powoduje błędnego odbioru znaków.

Wspomniany bit kontrolny jest najczęściej bitem parzystości, którego stan określa się według jednej z dwóch zasad:

- **kontrola parzystości (even parity),**
- **kontrola nieparzystości (odd parity).**

Kontrola parzystości polega na sprawdzeniu ilości jedynek logicznych w polu danych i ustawieniu bitu kontrolnego na „1”, w przypadku nieparzystej ilości jedynek, lub na „0” w przypadku parzystej ilości jedynek (jest to więc uzupełnienie do parzystości). **Bit kontroli parzystości pozwala wykryć fakt przekłamania znaku na polu danych, pod warunkiem, że ilość przekłamań jest nieparzysta.**

Asynchroniczna transmisja znakowa



Format jednostki informacyjnej dla transmisji asynchronicznej

Asynchroniczna transmisja znakowa

Wszystkie te charakterystyczne cechy ramki, tj. czas trwania (odpowiadający prędkości transmisji), liczba bitów danych, liczba bitów stopu i tryb kontroli parzystości są uzgadniane między nadajnikiem i odbiornikiem jako format przekazywanych danych jeszcze przed nawiązaniem połączenia (ponieważ parametry transmisji nie są wówczas ustalone, uzgodnienia należy na ogół dokonać „ręcznie", za pomocą programów konfigurujących). Tylko w ten sposób możliwe jest prawidłowe odczytanie napływających do odbiornika informacji. Generator odbiornika pracuje więc z założenia z taką samą częstotliwością z jaką pracował generator nadawczy, formujący dane do postaci szeregowej. Pozostaje jedynie synchronizacja fazy, tzn. uruchomienie układu dekodującego odbiornika precyzyjnie na początku ramki.

Bit startu przyjmuje zawsze wartość zero. Linia danych łącza w stanie spoczynku utrzymywana jest w stanie logicznej „1". Bit startu jest więc wyraźnie zaznaczonym początkiem transmisji. Bit stopu zamyka ramkę i łącze wraca do stanu początkowego lub realizuje transmisję następnej porcji danych.

Asynchroniczna transmisja znakowa

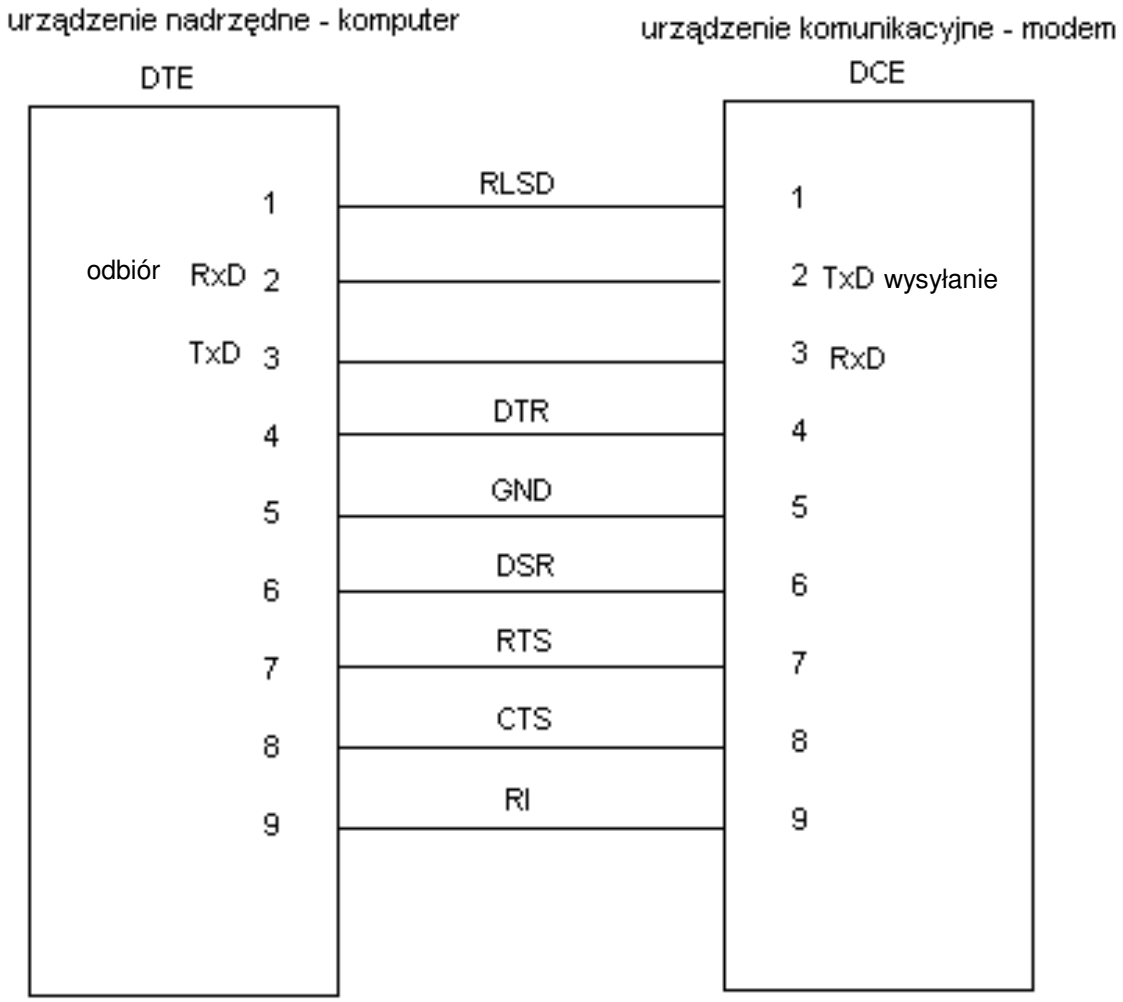
W warunkach rzeczywistych nie występują oczywiście tak wyidealizowane przebiegi prostokątne jak przedstawione na wcześniejszym rysunku. Przy dużych prędkościach transmisji zaczynają odgrywać rolę pojemności przewodów łączących. Odległość dzieląca nadajnik i odbiornik może dochodzić do 200 metrów, i to w warunkach przemysłowych. Wszystko to powoduje zniekształcenia zboczy sygnałów oraz nakładanie się na nie dodatkowych przebiegów zakłócających.

Przetwarzanie szeregowego strumienia danych napływających do odbiornika musi więc często odbywać się w sposób bardziej skomplikowany. **Odbiornik pracuje z częstotliwością 16-krotnie większą niż wynikałoby to z ustalonej szybkości pracy łącza, a każdy bit ramki jest badany (próbkowany) 16 razy.** Wynik pobrania tych 16 próbek jest uśredniany, a na podstawie tej operacji zapada decyzja, czy badanemu wycinkowi czasowemu przyporządkować niski czy wysoki poziom logiczny. Metoda ta podnosi znacznie odporność łącza na zakłócenia impulsowe i eliminuje niekorzystny wpływ zniekształconych zboczy.

Asynchroniczna transmisja znakowa

Połączenie interfejsów RS-232C od strony mechanicznej stanowi **25-żyłowy** przewód zakończony zdefiniowanymi przez standard wtykami. Większość z 25 linii przeznaczona została dla potrzeb szeregowej transmisji synchronicznej. W standardzie IBM PC wykorzystywane jest tylko 9 z tych sygnałów, dlatego tez często zamiast wtyku 25 - końcówkowego (DB-25) stosuje się wtyk 9 - końcówkowy **DB-9**.Ta liczba linii w zupełności wystarcza do obsłużenia transmisji asynchronicznej w standardzie RS-232C.

Asynchroniczna transmisja znakowa



Układ połączeń miedzy gniazdami łącz szeregowych urządzeń DTE i DCE

Asynchroniczna transmisja znakowa

Linie TxD (Transmitted Data) i RxD (Received Data) są właściwymi przewodami służącymi wymianie danych. Pozostałe są liniami sterującymi.

RTS (Request To Send) - urządzenie **DTE** (terminal, PC) sygnalizuje tą linią zamiar przekazywania danych do **DCE** (modemu). Modem przygotowuje się do odbioru danych.

CTS (Clear To Send) - linią tą przesyłane jest potwierdzenie przyjęcia sygnału **RTS** przez DCE (modem) i stwierdzenie gotowości do odbioru danych od DTE. Komputer może rozpocząć przekazywanie danych.

Para sygnałów sterujących **RTS**/**CTS** może przy półdupleksowym trybie pracy łącza (tzn. takim, w którym dopuszczalna jest niejednoczesna transmisja w obu kierunkach) sterować kierunkiem transmisji, przydzielając połączonym korespondentom na przemian rolę nadawcy i odbiorcy.

Asynchroniczna transmisja znakowa

DSR (Data Set Ready) - specyfikacja RS-232C określa ten sygnał jako meldunek urządzenia DCE (zwykle modemu), że zostało nawiązane połączenie i układ jest gotów do przyjęcia danych od DTE (zwykle komputera). W praktyce większość modemów nie wykorzystuje tej linii i jest ona sztucznie utrzymywana w stanie aktywnym, nie mówiąc nic o istnieniu połączenia z korespondentem. Po poziomie tego sygnału można co najwyżej wnioskować, czy modem jest w ogóle włączony.

DTR (Data Terminal Ready) - sygnał ten wskazuje w ogólności na gotowość urządzenia DTE (komputera). Musi on pozostawać aktywny przez cały czas trwania połączenia.

Uwaga:

*para sygnałów **DTR** i **DSR** odpowiada za utrzymanie połączenia, podczas gdy sygnały **RTS** i **CTS** są odpowiedzialne za przekazywanie danych i ewentualne sterowanie kierunkiem przepływu (w trybie półdupleksowym).*

Asynchroniczna transmisja znakowa

DCD (Data Carrier Defect) - modem (DCE) sygnalizuje tą linią odbiór fali nośnej, co oznacza, że druga strona jest w trakcie nawiązywania połączenia. Sygnał DCD pozostaje aktywny przez cały czas trwania transmisji.

Uwaga: DCD oznacza się także jako RLSD

RI (Ring Indicator) - w przypadku połączenia modemów przez sieć telefoniczną urządzenie DTE (komputer) informowane jest o odebraniu sygnału odpowiadającego wywołaniu abonenta (dzwonieniu).

DSRD (Data Signal Rate Detector) - linia ta występuje tylko w 25-końcówkowej wersji łącza. Umożliwia dostosowanie się korespondentów do jednej z dwóch możliwych prędkości transmisji. Z sygnału tego mogą korzystać obie strony połączenia.

Asynchroniczna transmisja znakowa

Uwaga ogólna

Jeżeli komunikacja ma zachodzić między dwoma urządzeniami klasy **DTE** (np. dwa komputery PC), należy wykorzystać tak zwany kabel modemu zerowego. W wyniku zastosowania takiego rozwiązania urządzenie nr 1 postrzega urządzenie nr 2 jako modem i odwrotnie.

Tryby pracy

Tryb simpleksowy

Łącze skonfigurowane jest na stałe na jeden z możliwych kierunków transmisji: **DTE -> DCE** lub **DCE -> DTE**. Rozpatrzmy dla przykładu pierwszą z tych możliwości. DTE wykorzystuje wówczas wyłącznie linię TxD, zaś linia RxD nie jest podłączona. Modem (DCE) nie bierze pod uwagę stanu sygnału RTS, bądź tez sygnał ten jest utrzymywany przez DTE stale w stanie aktywnym. Podobnie, komputer (DTE) nie uwzględnia sygnału CTS od modemu (DCE) lub też modem utrzymuje ten sygnał na poziomie aktywnym przez cały czas trwania połączenia. Sygnał DCD modemu z założenia nigdy nie może być aktywny. Sygnał DSR jest albo stale aktywny, albo aktywowany w momencie nawiązania kontaktu z korespondentem (drugim modemem). DTE (komputer) może poprzez sygnał DTR zgłaszać swoją gotowość modemowi (DCE); w rozwiązaniach praktycznych sygnał ten włącza i wyłącza modem. Linia RI z oczywistych względów nie ma tutaj znaczenia.

Tryby pracy

Tryb półdupleksowy

W trybie tym zarówno **DTE** jak i **DCE** mogą być stroną **nadającą jak i odbierającą**, jednak do dyspozycji jest **tylko jeden logiczny kanał danych**, który można naprzemiennie wykorzystywać w obydwu kierunkach. Wyjścia TxD każdego z urządzeń połączone są z wejściami RxD partnera. Wybór aktualnego kierunku transmisji dokonywany jest za pomoce sygnałów RTS-CTS, a prawo tego wyboru przysługuje w jednakowym stopniu obydwu stronom. Przykładowo, DCE (modem) chcąc przesłać dane do DTE (PC) aktywuje swój sygnał RTS i czeka na potwierdzenie na linii CTS. Uzgodnienie to upoważnia w tym przypadku modem do wysyłania danych a komputer do ich odbioru. Pozostałe sygnały zachowują swoje naturalne znaczenie, tzn. modem może aktywować linię DCD chcąc przekazywać dane do komputera, DSR oznacza trwały kontakt z korespondentem (drugim modemem), zaś DTR może służyć do włączania i wyłączania modemu. Urządzenie DCE może też robić użytek z linii RI, sygnalizując nią chęć nawiązania kontaktu przez partnera modemu znajdującego się po drugiej stronie łącza (zwykle telefonicznego).

Tryby pracy

Tryb dupleksowy

Dane mogą być przekazywane pomiędzy **DTE** i **DCE** jednocześnie w obydwu kierunkach. Większość nowoczesnych modemów może pracować w tym trybie. Nie jest wymagane uzyskiwanie przez żadną ze stron zezwolenia na nadawanie. Kanał logiczny połączenia otwarty jest stale w obie strony, a sygnały RTS/CTS nie mają znaczenia; są one albo niepodłączone albo stale aktywne. Stale aktywna jest też na ogół linia DSR, chyba że sygnalizuje ona nawiązywanie połączenia z drugim modemem. Sygnał DCD aktywowany jest w naturalny sposób jako odpowiedź na wykrycie fali nośnej, a linia DTR można sterować włączaniem i wyłączaniem modemu.

Transmisja synchroniczna

Transmisja synchroniczna polega na przesyłaniu dużych bloków danych (ramek – ang. frame) przy rezygnacji z bitów określających początek i koniec znaku. Poszczególne bity wyprowadzane są zgodnie z taktem nadawania. Po ostatnim bicie znaku poprzedniego wysyłany jest natychmiast pierwszy bit znaku następnego. Grupowanie bitów w znaki (bajty) po stronie odbiorczej umożliwia specjalny znak synchronizacyjny umieszczany na początku bloku (**znacznik początku**). Blok kończy inny wyróżniony znak (**znacznik końca**).

Transmisja synchroniczna jest szybsza od asynchronicznej ze względu na brak „jałowych" bitów startu i stopu oraz brak przerw między poszczególnymi znakami.

Przesyłanie znaków w większych blokach jest zadaniem bardziej złożonym niż transmisja pojedynczego znaku i wymaga buforowania, rozumianego jako grupowanie znaków przed ich wysłaniem. Przygotowany do nadania blok trzeba ograniczyć na początku i na końcu wyróżnionymi znakami (znaczniki początku i końca), co umożliwia stacji odbierającej podział odbieranych bitów na znaki i naturalne zakończenie odbioru. Poszczególne bity przesyłanego bloku są wyprowadzane ze stacji początkowej zgodnie z taktem nadawania, nazywanym również **„podstawą czasu przy nadawaniu"**

Szybkość transmisji synchronicznej w standardzie RS-232 wynosi do 20000 bodów, przy czym powszechnie stosowane wartości to: 1200, 2400, 9600 i 19200.

Podstawowym zadaniem stacji odbierającej dane jest prawidłowe rozpoznanie bitów w sygnale odebranym (synchronizacja bitowa) oraz połączenie bitów w znaki (synchronizacja znakowa).

Kabel modemu zerowego

Rozpatrzymy przypadek komunikacji między dwoma urządzeniami DTE (komputerami) ale bez pośrednictwa modemów. Zadanie polega na takim połączeniu ich wyprowadzeń, aby komputer 1 postrzegał komputer 2 jako modem i odwrotnie. Kabel służący połączeniu dwóch stacji typu DTE nazywamy kablem modemu zerowego (**null-modem cable**) ponieważ eliminuje on konieczność zastosowania modemów.

Najmniejszy problem stanowi oczywiście połączenie mas, po prostu łączy się ze sobą odpowiadające sobie masy obu urządzeń DTE. Sygnał TxD z komputera 1 trzeba wprowadzić do wejścia RxD komputera 2 i odwrotnie, sygnał TxD z komputera 2 do wejścia RxD komputera 1. Pozostaje oczywiście odpowiednie połączenie linii sterujących transmisją. Do dyspozycji są tylko sygnały DTE, trzeba więc je wykorzystać do emulacji sygnałów DCE. DTR jest sygnałem świadczącym o załączeniu terminala oraz wykorzystywanym do utrzymywania połączenia. Natomiast DSR (sygnał modemu) informuje o istnieniu połączenia między stacjami. Można więc sygnał DTR komputera 1 wprowadzić na wejście DSR komputera 2, a sygnał DTR komputera 2 na wejście DSR komputera 1. **W takim przypadku znaczenie DTR jest szersze , bowiem spełnia ono dodatkowo rolę DSR**. Jeżeli stacja zostanie wyłączona, to jej sygnał DTR zaniknie i na drugim końcu linii zabraknie sygnału DSR, co świadczy o przerwaniu połączenia.

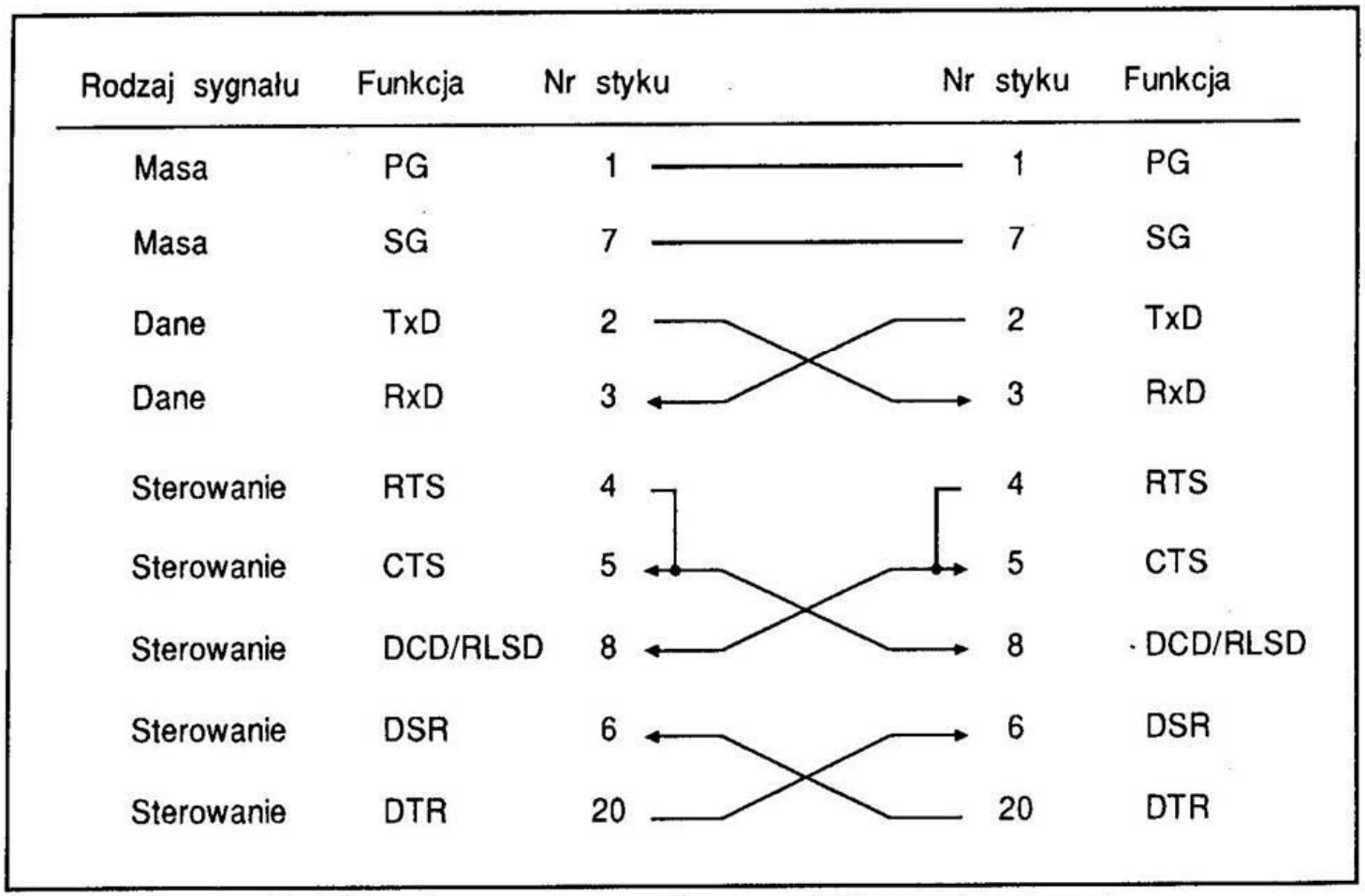
Kabel modemu zerowego

Za sterowanie transmisją danych w systemie z obecnym modemem odpowiedzialne są sygnały RTS, CTS i DCD. Z tej trójki tylko RTS jest sygnałem DTE, pozostałe dwa pochodzą z DCE i muszą być emulowane, jeżeli oczywiście połączenie dotyczy dwóch urządzeń DTE. Ponieważ RTS oznacza gotowość do nadawania danych, a CTS zezwolenie na transmisję, to **należy połączyć RTS z CTS tej samej stacji** uzyskując tym samym natychmiastowy do łącza po załączeniu RTS. Pozostaje jeszcze o tym fakcie powiadomić drugą stację, na przykład poprzez wprowadzenie sygnału RTS na jej wejście DCD. Połączenie takie umożliwia sprawdzenie zajętości łącza przed uaktywnieniem sygnału RTS. Podobnie zwarte wyprowadzenia RTS i CTS na złączu komputera 2 należy połączyć z wyprowadzeniem złącza DCD złącza komputera 1. Podane połączenia spełniają wymagania związane ze sterowaniem wymianą danych zarówno przy transmisji półdupleksowej jak i dupleksowej. Na następnym slajdzie pokazana została organizacja kabla modemu zerowego przy transmisji asynchronicznej.

Przypomnienie:

DCD (Data Carrier Defect) modem (DCE) sygnalizuje tą linią odbiór fali nośnej, co oznacza, że druga strona jest w trakcie nawiązywania połączenia. Sygnał DCD pozostaje aktywny przez cały czas trwania transmisji. DCD wyłącza się jeśli nie został odebrany żaden sygnał. Sygnał DCD zatem świadczy o zajętości łącza.

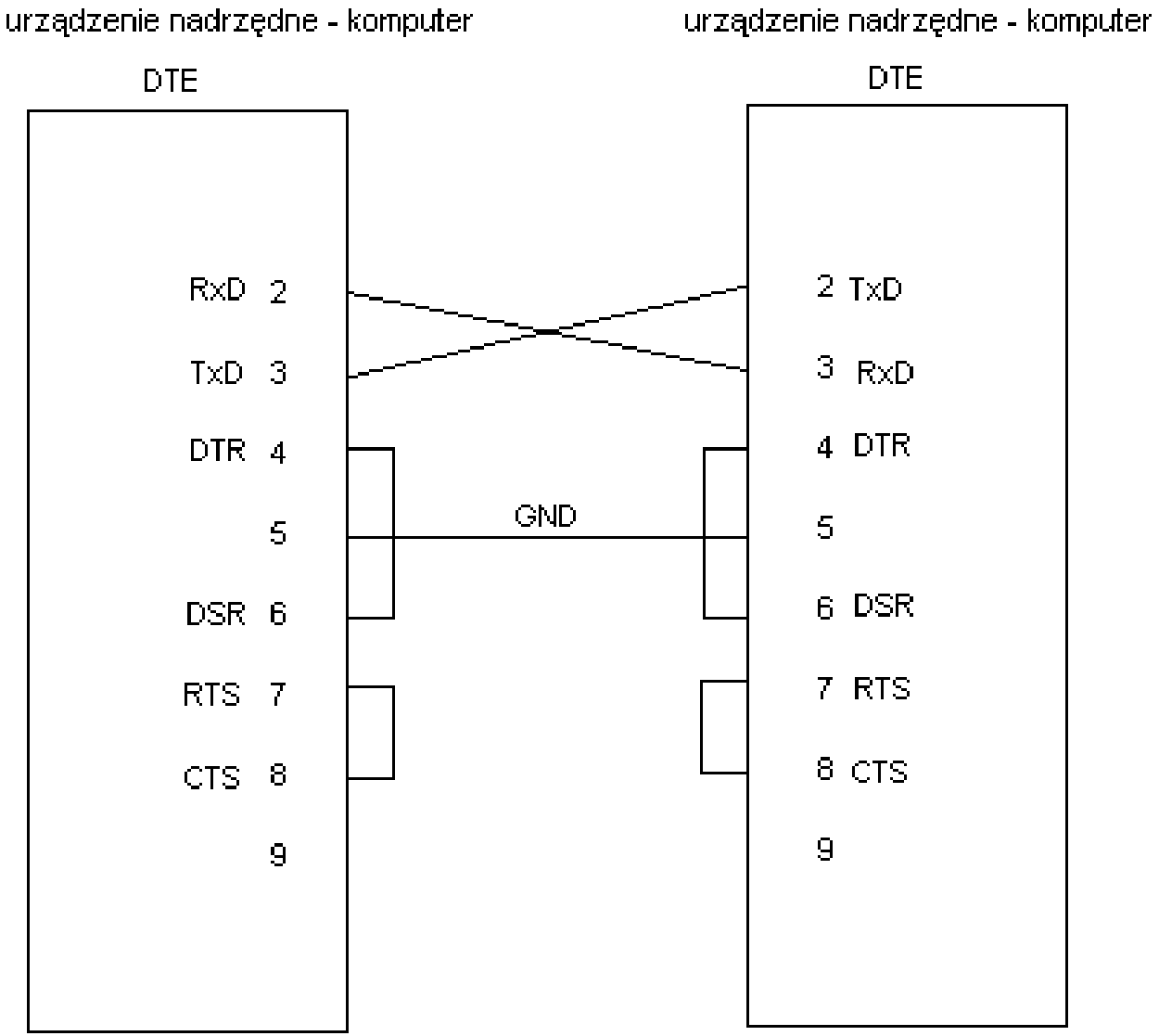
Kabel modemu zerowego



Rodzaj sygnału	Funkcja	Nr styku	Nr styku	Funkcja
Masa	PG	1	1	PG
Masa	SG	7	7	SG
Dane	TxD	2	2	TxD
Dane	RxD	3	3	RxD
Sterowanie	RTS	4	4	RTS
Sterowanie	CTS	5	5	CTS
Sterowanie	DCD/RLSD	8	8	DCD/RLSD
Sterowanie	DSR	6	6	DSR
Sterowanie	DTR	20	20	DTR

Kabel modemu zerowego dla transmisji asynchronicznej

Kabel modemu zerowego – wersja zminimalizowana



TRANSMISJA DANYCH

WYKŁAD nr 5

LABVIEW

Zintegrowane środowisko programowania LabVIEW jest narzędziem wykorzystującym **graficzny język programowania G**, który umożliwia konstruowanie urządzeń wirtualnych przeznaczonych do zbierania danych, ich przetwarzania, analizy i wizualizacji oraz sterowania procesami i pomiarami.

Urządzenia wirtualne to urządzenia, które umożliwiają oddziaływanie na rzeczywiste urządzenia pomiarowe za pomocą komputera (dane pomiarowe udostępniane są użytkownikowi za pośrednictwem monitora).

Pojęcie urządzenia wirtualnego *(VI)*

Programy LabVIEW traktowane są jako urządzenia wirtualne, ponieważ ich stosowanie przypomina pracę z rzeczywistymi urządzeniami. Do obsługi programu stosuje się wirtualne „elementy konstrukcyjne”, rozmieszczone na płycie czołowej panelu sterowania urządzenia wirtualnego, które stanowią graficzną reprezentację rzeczywistych elementów konstrukcyjnych, natomiast samo urządzenie funkcjonuje zgodnie z instrukcjami otrzymywanymi z programu graficznego.

Urządzenia wirtualne vi – panel sterowania i program graficzny

Urządzenie wirtualne określać będziemy nazwą **VI**, zgodnie z terminologią przyjętą przez autorów LabVIEW.

Projektowanie panelu dotyczy operacji projektowania płyty czołowej panelu sterowania urządzenia wirtualnego.

Programowanie natomiast należy interpretować jako tworzenie blokowej reprezentacji programu przedstawionego graficznie.

Wśród graficznych reprezentacji elementów konstrukcyjnych umieszczonych na panelu rozróżnia się takie, które umożliwiają wprowadzanie danych do programu czyli ***obiekty wejściowe – Controls***,
i takie, które służą do wizualizacji danych i rezultatów otrzymanych w wyniku ich obróbki, a więc ***obiekty wyjściowe – Indicators***.
Z obiektami wejściowymi i wyjściowymi skojarzone są odpowiednie ikony, umieszczane automatycznie w programie w czasie projektowania panelu, odpowiednio – ***Control Terminals*** i ***Indicator Terminals.***

Typ danych dostarczanych przez Control Terminals lub akceptowanych przez Indicator Terminals jest określony przez kolor konturów i ich ikon, przy czym grubość konturów umożliwia w sposób jednoznaczny określenie funkcji obiektu. Kontury pogrubione są charakterystyczne dla obiektów wejściowych, natomiast cienkie dla obiektów wyjściowych.

Przepływ danych w środowisku LabVIEW

Ikony są obiektami niskiego poziomu, które w czasie programowania łączy się między sobą, tak aby zapewnić przepływ danych. Dzięki temu, realizacja programu jest wyznaczana **przepływem danych**. Dane wprowadzane przez użytkownika za pomocą obiektów wejściowych umieszczonych na panelu są przekazywane do programu graficznego za pośrednictwem terminali Control Terminals. Proces odwrotny, tj. przekazywanie danych z programu graficznego do obiektów wyjściowych umieszczonych na panelu, odbywa się za pośrednictwem terminali Indicator Terminals. Control i Indicator Terminals pełnią zatem funkcję portów, za pośrednictwem których dane są przesyłane między panelem i programem.

Przepływ danych w środowisku LabVIEW

Realizacja programu **wymuszona przepływem danych** polega na tym, że w momencie pojawienia się kompletu argumentów na terminalach wejściowych odpowiedniej ikony przystępuje ona do wykonania przypisanego jej zadania. Po przetworzeniu danych są one natychmiast udostępniane przez tą ikonę innym ikonom za pośrednictwem trerminali wyjściowych.

Typy danych

Uwaga: kolor i rodzaj linii łączących poszczególne ikony programu informuje o typie przesyłanych danych.

- liczby całkowite – połączenia w kolorze niebieskim,
- liczby rzeczywiste – połączenia w kolorze pomarańczowym,
- zmienne logiczne – połączenia w kolorze zielonym,
- łańcuchy znaków – stringi są przesyłane połączeniami w kolorze purpurowym podobnie jak klastry, z tym, że wymienione rodzaje linii różnią się między sobą fakturą (klastrami będą nazywane grupy obiektów różnego typu traktowane jako jeden obiekt).

Pogrubiona linia w odpowiednim kolorze symbolizuje przesyłanie jednowymiarowych tablic zmiennych określonego typu.

Podwójna linia lub znacznie pogrubiona symbolizuje przesłanie tablic dwuwymiarowych ze zmiennymi określonego typu.

Kolor brązowy jest charakterystyczny dla klastrów utworzonych tylko z udziałem zmiennych numerycznych.

Programowanie w języku G

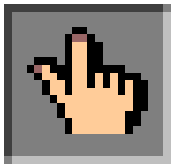
Programowanie w **języku graficznym G** składa się z trzech części: projektu panelu wirtualnego, opracowania graficznej reprezentacji programu oraz projektu ikony/złącza.

W pierwszym etapie projektuje się panel sterowania urządzenia wirtualnego, następnie tworzy się **program graficzny w oknie diagramu** i na końcu projektuje się ikonę/złącze reprezentującą program i umożliwiającą jego dołączenie do innych programów.

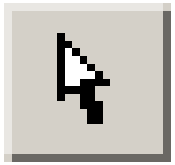
Programowanie w języku G – pole narzędzi

Pole narzędzi – Tools Palette

Do projektowania/programowania wykorzystuje się narzędzia **Tools Palette**, dzięki którym możliwy jest, w pierwszej kolejności, wybór obiektów oraz ich umieszczanie na panelu lub wstawianie do programu graficznego. W czasie projektowania/ programowania niezbędne są takie operacje jak: łączenie, przemieszczanie, uszeregowywanie i rozmieszczanie poszczególnych obiektów lub całych grup, usuwanie zbędnych obiektów, edycja tekstów, zadawanie parametrów, uruchamianie, testowanie i zatrzymywanie programu. Omawiane narzędzia spełniają powyżej wymienione funkcje.



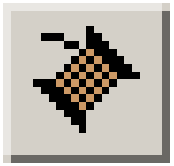
Wskaźnik **manipulatora** służy do obsługi panelu sterowania, umożliwiając włączanie lub wyłączanie przełączników, przesuwanie suwaków, potencjometrów, inkrementację i dekrementację wartości liczbowych zapisanych w obiektach numerycznych, itp.



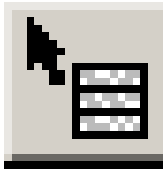
Wskaźnik **selektor** służy do zaznaczania obiektów lub grup obiektów w celu ich przesunięcia, zmiany rozmiarów, usunięcia zarówno w czasie projektowania panelu, jak i programowania.

Uwaga:

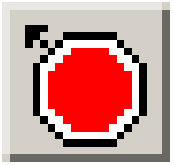
Dowolną stałą można przeciągnąć z okna diagramu do okna panelu i utworzyć w ten sposób w oknie panelu odpowiedni obiekt wejściowy. Możliwa jest również operacja odwrotna



Wskaźnik **szpulka** służy wykonywaniu połączeń między ikonami podczas tworzenia programu, jak również przyporządkowywania elementów panelu sterowania wybranym terminalom w złączu.



Wskaźnik **menu** rozwija podręczne menu odnoszące się do wskazanego obiektu i umożliwia jego modyfikację lub zmianę parametrów.



Wskaźnik **przerwań** powoduje zatrzymanie programu w miejscu zaznaczenia i umożliwia np. wybiórcze uruchamianie podprogramów dołączonych do programu głównego. Przerwania mogą dotyczyć:

- konkretnego obiektu przez jego zaznaczenie wskaźnikiem przerwań lub umieszczenie przerwania na połączeniu doprowadzającym dane do któregokolwiek terminalu tego obiektu,
- wybranej struktury przez kliknięcie wskaźnikiem przerwań w wolnym polu danej struktury,
- całego programu przez kliknięcie wskaźnikiem przerwań w wolnym polu programu.

Przerwanie jest sygnalizowane migotaniem czerwonej ramki wokół programu, struktury lub obiektu.

Pasek narzędziowy w oknie panelu

 - uruchamianie vi

 - zatrzymywanie vi

 -czasowe zatrzymanie programu

 - ciągły tryb pracy

Pasek narzędziowy w oknie panelu

przesuwanie obiektów jeden na drugi, pod spód, itp.

szeregowanie obiektów osiowo w pionie/poziomie lub wzdłuż wybranej krawędzi, wyrównywanie odstępów

Pasek narzędziowy oknie diagramu

uruchomienie spowolnionego trybu wykonywania programu

Zestaw trzech przycisków umożliwia przeglądanie programu krok po kroku, przy czym lewy przycisk uruchamia kolejne kroki, środkowym można ominąć napotkany obiekt funkcyjny, a prawy przycisk powoduje wyjście z danego trybu pracy.

Pole z grupami obiektów wejściowych i wyjściowych

Pole z pogrupowanymi obiektami wejściowymi i wyjściowymi jest dostępne tylko wtedy, gdy okno do projektowania panelu jest aktywne. Udostępnia ono grupy bibliotek oraz rozwijane kaskadowo pola, które zawierają obiekty umożliwiające sterowanie procesami, wpisywanie i wizualizację danych oraz wczytywanie plików użytkownika.

Obiekty wejściowe/wyjściowe udostępnione przez poszczególne biblioteki:

- Numeric – przełączniki i wyświetlacze numeryczne,
- Boolean – przełączniki i wyświetlacze dwustanowe,
- String & Table – obiekty z polami do wpisywania i odczytu tekstów oraz tabel,
- List & Ring – tekstowo – numeryczne obiekty wyboru,
- Array & Cluster – obiekty do konstruowania tablic i klastrów,
- Graph – wyświetlacze graficzne,
- Path & Refnum – obiekty do wpisywania ścieżki dostępu oraz identyfikacji operacji wejścia/wyjścia,
- ActiveX – obiekty do transferu danych,
- Dialog – obiekty dialogowe,
- Decorations – elementy dekoracyjne,
- User Controls – obiekty wejściowe i wyjściowe użytkownika,
- Select a Control… - polecenie otwarcia okna dialogu.

Pole z grupami obiektów funkcyjnych i procedurami

Pole z pogrupowanymi obiektami funkcyjnymi i procedurami jest dostępne tylko wtedy, kiedy aktywne jest okno do programowania graficznego. Udostępnia ono grupy bibliotek oraz kaskadowo rozwijane pola, które zawierają obiekty funkcyjne, umożliwiające przetwarzanie danych oraz kontrolowanie ich przepływu w programie graficznym. Obiekty funkcyjne i procedury można podzielić na dwie grupy: obiekty niskiego poziomu – podstawowe (operatory operacji arytmetycznych, funkcji trygonometrycznych) oraz obiekty wysokiego poziomu – zaawansowane (sterowniki urządzeń zewnętrznych, obiekty umożliwiające analizę danych).

Podstawowe obiekty funkcyjne i procedury

- Structures – struktury programistyczne,
- Numeric – obiekty udostępniające operacje numeryczne,
- Boolean – obiekty udostępniające operacje logiczne,
- String – obiekty udostępniające operacje z łańcuchami znaków,
- Array – obiekty udostępniające operacje na tablicach,
- Cluster – obiekty udostępniające operacje na klastrach,
- Comparison – obiekty udostępniające operacje porównania,
- Time&Dialog – obiekty do taktowania i wprowadzania opóźnień,
- File I/O – obiekty do zapisu i odczytu plików.

Zaawansowane obiekty funkcyjne

Do zaawansowanych obiektów funkcyjnych zalicza się między innymi obiekty funkcyjne i procedury wymienione poniżej:

- Instrument I/O – obiekty do obsługi urządzeń zewnętrznych (np. obsługa portu szeregowego RS232),
- Instrument Drivers – sterowniki urządzeń,
- Signal Processing – obiekty do obróbki sygnałów,
- Mathematics – obiekty z operacjami matematycznymi,
- Graphics & Sound – obiekty udostępniające operacje graficzne i dźwiękowe,
- Communication – obiekt do komunikacji komputera z otoczeniem,
- Application Control – obiekty służące obsłudze aplikacji,
- Advanced – obiekty udostępniające zaawansowane operacje,
- User Libraries – biblioteki użytkownika,
- Select a VI… - polecenie otwarcia okna dialogu.

Ikony i złącza

Programy vi mają strukturę modułową i hierarchiczną, co oznacza, że każdy program realizujący konkretne zadanie może zawierać podprogramy (procedury) przeznaczone do wykonywania mniej skomplikowanych operacji lub zostać wykorzystany jako podprogram (***SubVI***) w innej aplikacji. Jest to możliwe dzięki ikonie i złączu.

Ikona reprezentuje konkretny podprogram w innym programie graficznym, natomiast złącze z zaciskami umożliwia dołączenie tego podprogramu do innych lub dołączenie terminali reprezentujących obiekty wejściowe i wyjściowe umieszczone na panelu sterowania.

Czynnością kończącą programowanie graficzne jest zaprojektowanie złącza. Dowolny obiekt wejściowy lub wyjściowy umieszczony na panelu sterowania można dołączyć do wolnego zacisku w złączu towarzyszącym ikonie. Typ złącza można wybrać po skorzystaniu z biblioteki Patterns zawierającej wzorce złącz, do której dostęp umożliwia podręczne menu rozwijane po zaznaczeniu Show Connector. Połączenie obiektu z zaciskiem uzyskuje się za pomocą wskaźnika szpulki kliknięciem kolejno zacisku w złączu, a następnie wyselekcjonowaniu obiektu na panelu.

Po zaprojektowaniu złącza możliwe jest dzięki niemu wprowadzenie danych do danego podprogramu lub ich wyprowadzenie z podprogramu w celu udostępnienia innym obiektom w programie nadrzędnym.

Struktura wyboru - Case

Jeżeli te same dane wprowadzane do programu mają być przetwarzane w różny sposób, to niezbędna jest struktura, która umożliwi wybór jednego z dopuszczalnych wariantów. Taką możliwość oferuje **struktura Case.** Po zaznaczeniu struktury **Case** przesuwamy kursor w odpowiednie miejsce w oknie diagramu i po przyciśnięciu i przytrzymaniu lewego klawisza myszy otaczamy ramką tę część programu, która ma podlegać operacji wyboru. Po otoczeniu programu strukturą wyboru na jej ramce pojawią się **tunele służące do wprowadzania danych lub ich wyprowadzania. Obiekt wejściowy decydujący o wyborze jednego z wariantów dołączamy do węzła selektora oznaczonego znakiem zapytania.**