6. Sterowniki PLC – wstęp do programowania

Część 3 normy IEC 61131 [17] omawia metody programowania sterowników PLC, metody te opisuje również wiele pozycji zarówno w języku polskim [2], [7], [11], [21], jak i w językach obcych [1], [6]. Norma definiuje dwie grupy języków programowania:

języki tekstowe

- język *listy instrukcji IL* (ang. Instruction List) – jest odpowiednikiem języka niskiego poziomu jakim jest *asembler*, którego instrukcje bazują na *liście rozkazów* procesora, w skład zbioru instrukcji *IL* wchodzą operacje logiczne, arytmetyczne, operacje relacji (porównań) oraz funkcje przerzutników, timerów itp.
- język *tekstu strukturalnego ST* (ang. Structured Text) – jest odpowiednikiem języka wysokiego poziomu, zawiera podobny zestaw instrukcji jak Pascal czy C.

języki graficzne

- język *schematów drabinkowych LD* (ang. Ladder Diagram) – jest odpowiednikiem schematów (obwodowych, drabinkowych) układów cyfrowych realizowanych w technologii stykowo-przekaźnikowej, dodatkowo język pozwala na wykorzystywanie funkcji realizujących operacje arytmetyczne, logiczne, porównania oraz bloków funkcjonalnych realizujących zadania przerzutników, timerów, liczników.
- język *funkcjonalnych schematów blokowych FBD* (ang. Function Block Diagram) – jest odpowiednikiem schematów stosowanych do opisu układów logicznych wykorzystujących bramki logiczne, przerzutniki, timery, liczniki.

Dodatkowo norma definiuje graficzną metodę organizacji programu w postaci *sekwencyjnego schemat funkcjonalnego SFC* (ang. Sequential Function Chart). Zadania sterowania sekwencyjnego reprezentowane są w tym przypadku w postaci tzw. *grafu sekwencji* zbudowanego z *kroków* i *tranzycji*, które mogą być oprogramowywane w jednym z tekstowych lub graficznych języków normy.

Zgodnie z normą, każde zadanie sterowania stanowi tzw. *projekt*. Program użytkowy rozwiązujący określone zadanie projektowe może składać się z oddzielnych modułów oprogramowania. Moduły te nazywane są *jednostkami organizacyjnymi oprogramowania* (POU, ang. Program Organization Units). Rozbicie programu na jednostki organizacyjne poprawia czytelność programu i umożliwia przenoszenie jednostek organizacyjnych pomiędzy różnymi projektami. Jednostkami organizacyjnymi oprogramowania są:

- *funkcje* – na podstawie argumentów wejściowych wyznaczają wartość argumentu wyjściowego (argumenty wejściowe jednoznacznie wyznaczają wartość argumentu wyjściowego),
- *bloki funkcjonalne* – posiadają „pamięć", wartość argumentu wyjściowego nie wynika wyłącznie z wartości argumentów wejściowych, zależy również od stanu „pamięci",
- *programy* – podstawowe jednostki organizacyjne programu użytkowego, odpowiadają „programowi głównemu" w klasycznych językach programowania, programy mają dostęp do danych z modułów wejściowych i wyjściowych sterownika.

Norma zakłada, że pojedyncza jednostka organizacyjna musi być zaprogramowana w jednym języku programowania.

6.1. Elementarne typy danych

Jednostki organizacyjne wykorzystują *zmienne* do przechowywania informacji (danych). Zmienne przechowują wartości danych wejściowych i wyjściowych oraz wartości danych pomocniczych. Przechowywane wartości mogą być wartościami różnych *typów*. Norma definiuje *typy elementarne*, przedstawia także sposoby definiowania nowych typów danych tzw. *typów pochodnych*. Typy elementarne definiowane w normie IEC 61131-3 zostały zestawione w tab. 6.1.

Norma przewiduje różne metody zapisu wartości. Zgodnie z opisem w tab. 6.1 dane typu `BOOL` można zapisywać albo jako `TRUE` lub `FALSE`, albo jako `1` lub `0`. W przypadku danych liczbowych możliwe jest, dla zwiększenia czytelności zapisu, użycie symbolu podkreślenia (który nie jest w żaden sposób interpretowany). Ponadto liczby całkowite mogą być podawane w formacie:

- dziesiętnym,
- dwójkowym (z prefiksem `2#`),
- ósemkowym (z prefiksem `8#`),
- szesnastkowym (z prefiksem `16#`).

Liczby rzeczywiste zapisywane są z kropką i mogą być podane w formacie:

- dziesiętnym,
- wykładniczym (z użyciem symbolu: `e` lub `E`).

Tab. 6.1. Typy elementarne definiowane w normie IEC 61131–3

Typ	*Opis*
`BOOL`	dane boolowskie o rozmiarze 1 bitu, przyjmują wartości `TRUE` i `FALSE` (`1` i `0`)
`BYTE`, `WORD`, `DWORD`, `LWORD`	ciągi bitowe o rozmiarze: 8, 16, 32, 64 bitów
`SINT`, `INT`, `DINT`, `LINT`	liczby całkowite, różnią się zakresem wartości (zajmują w pamięci odpowiednio 1, 2, 4 i 8 bajtów)
`USINT`, `UINT`, `UDINT`, `ULINT`	jw., ale dane reprezentują liczby całkowite bez znaku
`REAL`, `LREAL`	liczby rzeczywiste, typy różnią się zakresem (zajmują w pamięci 4 i 8 bajtów)
`TIME`	czas trwania, wykorzystywany np. przez timery
`DATE`	daty
`TIME_OF_DAY` w skrócie: `TOD`	czas zegarowy
`DATE_AND_TIME` w skrócie: `DT`	data połączona z czasem
`STRING`, `WSTRING`	ciągi znaków, różnią się liczbą bajtów potrzebną do przechowania pojedynczego znaku (1 lub w 2 bajty)

W tab. 6.2 podano kilka przykładowych wartości liczbowych z odpowiadającymi wartościami w formacie dziesiętnym.

Tab. 6.2. Przykładowe literały liczbowe

Przykład	*Wartość w formacie dziesiętnym*
`-12`	–12
`123_456`	123456
`2#0000_0100`	4
`8#10`	8
`16#0F`	15
`3.14159_26`	3,1415926
`1.234E4`	12340

Dane odpowiadające czasowi trwania (typ `TIME`) należy podawać, poprzedzając je symbolami `T#` lub `TIME#` (lub `t#`, `time#`). Po podaniu wartości liczbowej należy określić jej jednostkę: dni (litera `d` lub `D`), godziny (litera `h` lub `H`), minuty (litera `m` lub `M`), sekundy (litera `s` lub `S`), milisekundy (litery `ms` lub `MS`). W przypadku określania kilku różnych jednostek czasu należy je podawać w kolejności malejącej, używając ewentualnie symbolu podkreślenia do oddzielenia wartości w różnych jednostkach czasu. Poniżej podano kilka przykładowych czasów trwania:

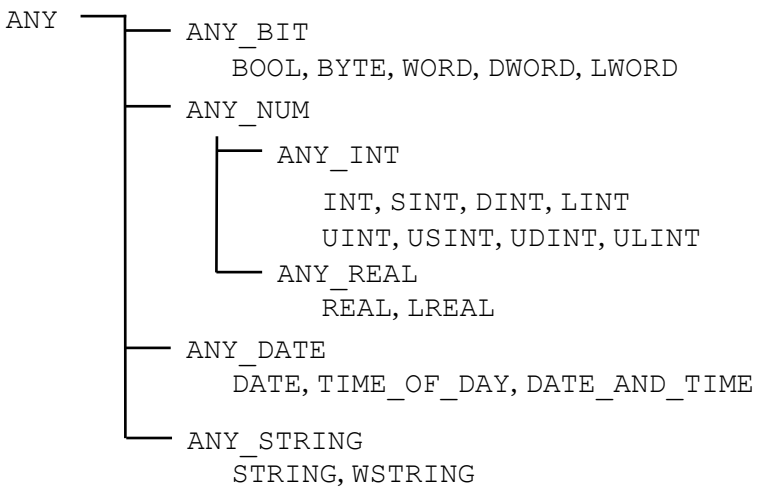
- `t#5s`,
- `t#5d14h_12m_18s_3.5ms`.

Dane odpowiadające datom (typ `DATE`) należy podawać, poprzedzając je symbolami `D#` lub `DATE#` (lub `d#`, `date#`), podając kolejno 4 cyfry roku, 2 cyfry miesiąca i 2 cyfry dnia, oddzielając je symbolem myślnika. Podobnie powinny być zapisywane dane odpowiadające czasowi zegarowemu. W tym przypadku z przodu należy podać przedrostek `TOD#` lub `TIME_OF_DAY#` (lub `tod#` lub `time_of_day#`), następnie 2 cyfry z godziną, minutą i sekundą oddzielone symbolem dwukropka. Czas może również zawierać określoną liczbę milisekund – muszą być one podawane po kropce jako część pola z sekundami. Wartości typu `DATE_AND_TIME` należy podawać, poprzedzając je przedrostkiem `DT#` lub `DATE_AND_TIME#` (`dt#` lub `date_and_time#`), stosując omówione już zasady dotyczące podawania daty i czasu, oddzielając datę od czasu symbolem myślnika. Poniżej podano kilka przykładowych dat i czasów:

- `d#1999-12-31`,
- `tod#23:59:59.5`,
- `dt#1999-12-31-23:59:59.5`.

Ciągi znaków typu `STRING` należy zapisywać, umieszczając na początku i końcu ciągu symbol apostrofu. Ciągi typu `WSTRING` należy zapisywać, używając symbol cudzysłowu.

Typy w normie zostały pogrupowane zgodnie ze schematem przedstawionym na rys. 6.1. Nazwy rozpoczynające się od przedrostka `ANY` nie są właściwie nazwami typów, chociaż są nazywane typami uniwersalnymi. Użytkownik nie może używać tych typów do tworzenia zmiennych (zob. p. 0), są one używane do specyfikacji wejść i wyjść standardowych funkcji i bloków funkcjonalnych (zob. p. 6.3 i 6.4).



Rys.6.1. Uniwersalne typy danych

6.2. Zmienne

Zmienne są używane przez jednostki organizacyjne oprogramowania (POU) do przechowywania i przetwarzania danych. Każda zmienna odpowiada pewnej informacji umieszczonej w strukturze wejść, wyjść lub w pamięci sterownika. Zmienne reprezentujące pojedynczą wartość typu elementarnego nazywane są *zmiennymi prostymi* (*skalarnymi*, *jednoelementowymi*). Zmienne proste mogą być przedstawione symbolicznie (poprzez nazwę), mogą też być deklarowane bez nazwy przez bezpośrednie wskazanie lokalizacji (odwołanie do wejścia/wyjścia lub komórki pamięci). Lokalizacja może być również wskazana w przypadku definiowania zmiennej symbolicznej, jeżeli lokalizacja nie jest wskazana, zmienna jest automatycznie lokowana w pamięci sterownika.

Lokalizację zmiennej wskazuje się zgodnie z następującym schematem:

```
%<położenie><rozmiar><adres>
```

gdzie:

`<położenie>` określa obszar pamięci, w którym jest umieszczona zmienna, symbole: `I`, `Q`, `M` są stosowane do wskazania odwołania odpowiednio do obszaru pamięci danych wejściowych, wyjściowych i obszaru pamięci pomocniczej przeznaczonej na dane użytkownika,

`<rozmiar>` określa liczbę bitów, które zmienna zajmuje, symbol `X` lub brak symbolu oznacza, że zmienna ma rozmiar jednego bitu, symbole `B`, `W`, `D` i `L` są używane do wskazania rozmiaru o wielkości odpowiednio jednego: bajtu (8 bitów), słowa (16 bitów), podwójnego słowa (32 bity) i poczwórnego słowa (64 bity),

`<adres>` to jedna liczba całkowita lub ciąg liczb całkowitych rozdzielonych kropkami, adres podany w postaci ciągu liczb jest interpretowany jako hierarchiczny sposób adresacji, pierwsze pole od lewej oznacza poziom najwyższy (adresowanie jest zależne od urządzenia i jego konfiguracji).

Symbole wykorzystywane podczas deklarowania zmiennej z zadaną lokalizacją zostały zebrane w tab. 6.3.

Tab. 6.3. Symbole w deklaracjach zmiennych z lokalizacją

Symbol	*Znaczenie*
położenie	
`I`	pamięć danych wejściowych
`Q`	pamięć danych wyjściowych
`M`	pamięć danych użytkownika
rozmiar	
`X` lub brak	bit
`B`	bajt (8 bitów)
`W`	słowo (16 bitów)
`D`	podwójne słowo (32 bity)
`L`	poczwórne słowo (64 bity)

Zgodnie z normą każda jednostka organizacyjna oprogramowania powinna na początku zawierać co najmniej jeden *blok deklaracji zmiennych*. Zmienne wewnętrzne (lokalne) definiowane za pomocą bloku:

```
VAR

END_VAR
```

Zmienne deklarowane są wewnątrz takiego bloku zgodnie ze schematem:

```
<nazwa> AT<lokalizacja>: <typ> := <wartość>;
```

gdzie:

<nazwa> to unikalny ciąg znaków identyfikujący zmienną (litery alfabetu angielskiego, cyfry, podkreślenie, nie może zaczynać się od cyfry, wielkość znaków nie ma znaczenia, więc nazwy ABCD, abcd i aBCD odnoszą się do tej samej zmiennej), nazwa jest pomijana w przypadku deklarowania zmiennej przez bezpośrednie wskazanie lokalizacji,

AT<lokalizacja> określa lokalizację zmiennej zgodnie z zasadami opisanymi wcześniej, parametr ten jest pomijany w przypadku definiowania zmiennej automatycznie lokowanej w pamięci sterownika,

<typ> określa typ zmiennej (jeden z typów elementarnych lub typ pochodny),

<wartość> określa początkową wartość zmiennej, nadawanie zmiennym wartości początkowych jest opcjonalne.

Poniżej podano kilka przykładowych deklaracji zmiennych.

```
VAR
1 AT %QX5.1: BOOL := 1;
2 AT %MW6: INT := 8;
3 x AT %QW3: INT := 100;
4 y AT %QX1: BOOL;
5 z AT %Q2: BOOL;
6 a: BOOL;
7 b: BOOL := TRUE;
END_VAR
```

W pierwszych dwóch deklaracjach tworzone są zmienne reprezentowane bezpośrednio. Zmienna pierwsza jest zmienną boolowską i odpowiada pierwszemu bitowi modułu wyjściowego umieszczonego w piątym gnieździe, zmienna jest inicjalizowana wartością 1 (TRUE). Druga zmienna to liczba całkowita inicjowana wartością 8, zmiennej przydzielana jest pamięć o wielkości jednego słowa w pamięci danych użytkownika pod adresem 6. Zmienne od trzeciej do piątej to zmienne o nazwach: x, y i z kojarzone z wyjściami sterownika. Zmienna x odpowiada słowu wyjściowemu o adresie 3, które jest inicjalizowane wartością 100. Zmienne y i z są skojarzone z pierwszym i drugim bitem na wyjściu. Zmienne a i b są automatycznie lokowane w pamięci sterownika, zmienna b otrzymuje wartość początkową TRUE (1).

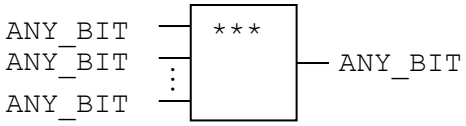
6.3. Funkcje standardowe

Funkcja to jednostka organizacyjna oprogramowania, która na podstawie argumentów wejściowych wyznacza wartość argumentu wyjściowego (argumenty wejściowe jednoznacznie wyznaczają wartość argumentu wyjściowego). Użytkownik może pisać własne funkcje zarówno w językach tekstowych, jak i graficznych. Dodatkowo norma wprowadza szereg funkcji standardowych, które powinny być implementowane przez producentów w sterownikach PLC zgodnych z normą. Funkcje standardowe wykonują operacje konwersji pomiędzy typami, obliczenia arytmetyczne, wyznaczają wartości funkcji trygonometrycznych, logarytmicznych, realizują porównania, operują na ciągach znaków, datach i czasie. W tym punkcie zostaną omówione wyłącznie *standardowe funkcje boolowskie* i *arytmetyczne* (zob. tab. 6.4 i 6.6).

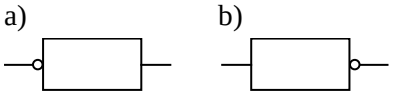
6.3.1. Standardowe funkcje boolowskie

Funkcje boolowskie realizują operacje logiczne na argumentach typu boolowskiego lub na ciągach bitowych (w przypadku ciągów bitowych funkcja logiczna działa kolejno na wszystkich bitach argumentów wejściowych). Standardowe funkcje boolowskie zostały zestawione w tab. 6.4.

Tab. 6.4. Standardowe funkcje boolowskie

Symbol graficzny		
		
(***) to nazwa albo symbol funkcji		
nazwa	*symbol*	*opis*
AND	&	koniunkcja
OR	>=1	alternatywa
XOR	=2k+1	alternatywa wykluczająca
NOT		negacja

Funkcja NOT jest funkcją jednoargumentową. W przypadku funkcji i bloków funkcjonalnych wykorzystywanych w językach graficznych norma umożliwia negowanie wejścia lub wyjścia w sposób przedstawiony na rys. 6.2, pozwala to na wyeliminowanie konieczności użycia funkcji NOT.

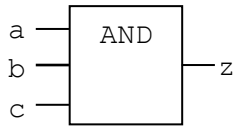


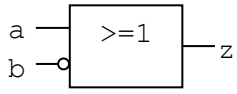
Rys.6.2. Negacja a) wejścia i b) wyjścia

Funkcje AND, OR i XOR mogą mieć dowolną liczbę argumentów (wieloargumentowa koniunkcja jest prawdziwa, jeśli wszystkiej jej argumenty są prawdziwe, wieloargumentowa alternatywa jest fałszywa, jeśli wszystkie jej argumenty są fałszywe, wieloargumentowa alternatywa wykluczająca jest fałszywa, jeśli parzysta liczba jej argumentów jest prawdziwa). W językach graficznych funkcja może być opisana nazwą i symbolem, w językach tekstowych tylko funkcja AND może być tak opisywana, pozostałe funkcje należy opisywać nazwą. W tab. 6.5 została pokazana realizacja wybranych wyrażeń logicznych w języku *FBD* i *ST*.

Tab. 6.5. Przykładowe wyrażenia logiczne w językach: FBD i ST

FBD	*ST*

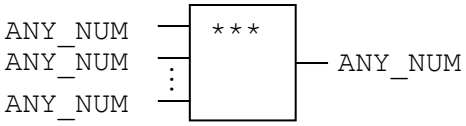
	`z := AND(a, b, c);`
	`z := a AND b AND c;`
	`z := a & b & c;`

	`z := a OR NOT b;`

6.3.2. Standardowe funkcje arytmetyczne

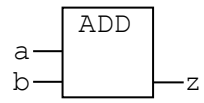
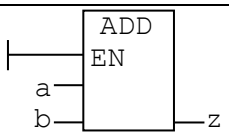
Kolejną grupą funkcji standardowych są funkcje arytmetyczne, które operują na liczbach tzn. na argumentach typu: ANY_NUM, ANY_INT i ANY_REAL. Funkcje te zostały zestawione w tab. 6.6.

Tab. 6.6. Standardowe funkcje arytmetyczne

Symbol graficzny		
		
(***) to nazwa albo symbol funkcji		
nazwa	*symbol*	*opis*
funkcje o zmiennej liczbie wejść		
ADD	+	dodawanie
MUL	*	mnożenie
funkcje o stałej liczbie wejść		
SUB	-	odejmowanie
DIV	/	dzielenie całkowite
MOD		reszta z dzielenia
EXPT	**	potęgowanie
MOVE	:=	przepisanie

Funkcje ADD i MUL mogą mieć dowolną liczbę argumentów, funkcje SUB, DIV, MOD i EXPT są dwuargumentowe, funkcja MOVE jest jednoargumentowa i może być wykorzystana do przepisania wartości z jednej zmiennej do drugiej. Norma dopuszcza opisywanie funkcji nazwą i symbolem. W tab. 6.7 pokazano realizację dodawania w języku *FBD*, *LD* i *ST*.

Tab. 6.7. Przykładowa operacja arytmetyczna w językach: FBD, LD i ST

FBD	*LD*	*ST*
		`z := ADD(a, b);` `z := a + b;`

Funkcje ADD, SUB, MUL i DIV wymieniane są również w normie w grupie funkcji operujących na danych związanych z czasem, tzn. na argumentach typu: DATE, TIME, i TIME_OF_DAY.

6.3.3. Wykorzystanie wejścia i wyjścia EN i ENO

W językach graficznych *FBD* i *LD* funkcje mogą być wywoływane z dodatkowym wejściem EN (ang. Enable), wyjściem ENO (ang. Enable Out) lub jednocześnie z wejściem EN i wyjściem ENO. W języku *LD* użycie wejścia EN jest konieczne, w przypadku gdy funkcja nie ma wejścia boolowskiego (tab. 6.7). Wejście EN i wyjście ENO umożliwiają przepływ prądu przez blok.

Wejście EN wprowadza pozwolenie na wykonanie funkcji, jeżeli na tym wejściu pojawi się sygnał 0 (FALSE), funkcja nie jest wywoływana, a na wyjściu ENO generowany jest również sygnał 0 (FALSE). Obliczanie wartości funkcji odbywa się tylko w przypadku gdy na wejściu EN jest sygnał 1 (TRUE), wartość ta jest domyślną wartością dla tego wejścia. Jeżeli funkcja zostanie wykonana bez błędów, na wyjściu ENO jest ustawiana wartość 1 (TRUE), w przeciwnym przypadku wartość 0 (FALSE).

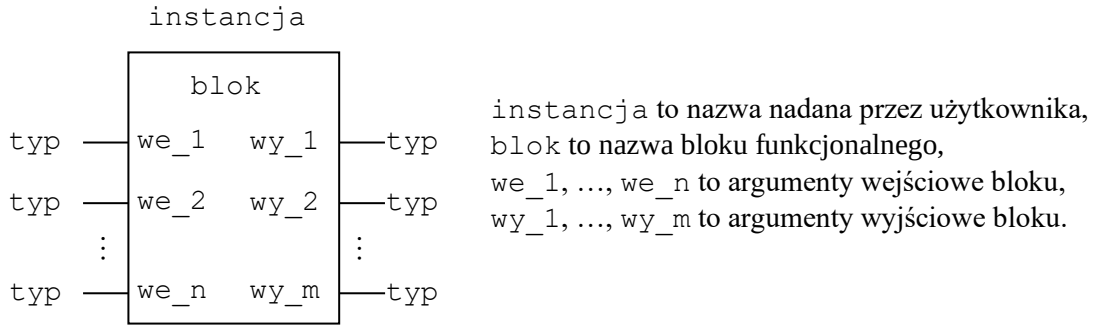
6.4. Standardowe bloki funkcjonalne

Blok funkcjonalny to jednostka organizacyjna oprogramowania, która na podstawie argumentów wejściowych wyznacza wartość argumentu wyjściowego (argumenty wejściowe nie wyznaczają jednoznacznie wartości argumentu wyjściowego, ponieważ jego wartość zależy również od stanu „pamięci" konkretnej instancji danego bloku). Użytkownik może pisać własne bloki funkcjonalne, podobnie jak zwykłe funkcje zarówno w językach tekstowych, jak i graficznych.

Norma definiuje szereg standardowych bloków funkcjonalnych. Bloki te można podzielić na:

- elementy bistabilne,
- detektory zbocza,
- liczniki,
- timery.

W przypadku korzystania z bloków funkcjonalnych konieczna jest wiedza o nazwie bloku oraz nazwach i typach jego argumentów wejściowych i wyjściowych. Nazwa bloku i nazwy argumentów oraz nazwa instancji nadana przez użytkownika są elementami symbolu graficznego bloku (rys. 6.3).



Rys.6.3. Symbol graficzny bloku funkcjonalnego

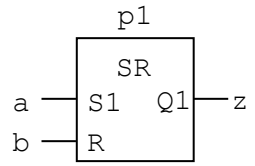
W językach graficznych bloki funkcjonalne można wywoływać z wejściem `EN` i wyjściem `ENO` (zob. p. 0). Niezależnie od tego czy blok funkcjonalny wykorzystywany jest w języku tekstowym czy graficznym, konieczne jest tekstowe utworzenie tzw. instancji bloku. Instancję (inaczej wystąpienie czy egzemplarz) bloku tworzy się, deklarując zmienną o typie odpowiadającym danemu blokowi. Zakładając, że użytkownik chce wykorzystać przerzutnik z dominującym wejściem ustawiającym, który zgodnie z normą realizowany jest za pomocą bloku `SR`, deklaracja taka może wyglądać następująco:

```
VAR
  p1: SR;
END_VAR
```

gdzie: `p1` to nazwa instancji bloku ustalona przez użytkownika, a `SR` to nazwa typu odpowiadająca przerzutnikowi SR. Dopiero po zadeklarowaniu instancji blok może być wykorzystany w programie.

W tab. 6.8 pokazano przykładowe użycie bloku `p1` w językach *FBD* i *ST*. Wejściom bloku `S1` i `R` przypisywane są wartości zmiennych `a` i `b`. Blok wykonuje operację ustawiania, w wyniku czego odpowiednia wartość pojawia się na jego wyjściu `Q1`. Wartość z tego wyjścia jest ostatecznie przypisywana zmiennej `z`. W języku *ST* w celu skorzystania z bloku funkcjonalnego należy napisać dwie linie kodu, pierwsza linia zawiera wywołanie bloku z odpowiednimi wartościami argumentów wejściowych, w linii drugiej odbywa się przypisanie wybranej zmiennej wartości pobranej z wyjścia bloku.

Tab. 6.8. Przykładowe wykorzystanie bloku funkcjonalnego

FBD	*ST*
	`p1(S1:=a, R:=b);` `z := p1.Q1;`

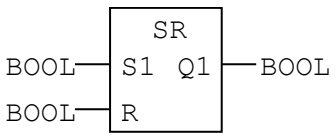
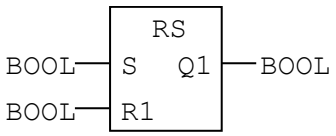
6.4.1. Elementy bistabilne

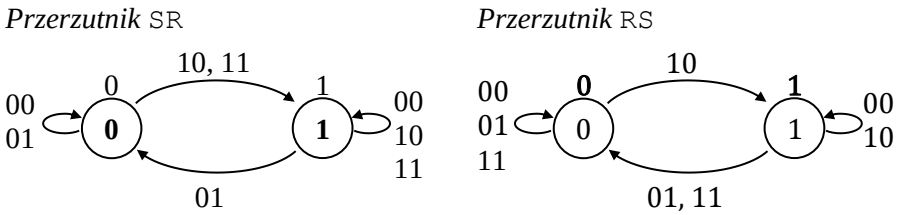
Przerzutniki są elementarnymi układami sekwencyjnymi zdolnymi do zapamiętania jednego bitu informacji, zostały omówione szerzej w punkcie 4.5. Norma definiuje dwa przerzutniki SR:

- *przerzutnik z dominującym wejściem ustawiającym* oznaczany jako `SR` (ang. Set Reset),
- *przerzutnik z dominującym wejściem zerującym* oznaczany jako `RS` (ang. Reset Set).

Symbole graficzne, opis wejść i wyjść przerzutników zestawiono w tab. 6.9, zasadę działania przedstawiono w formie grafów przejść na rys. 6.4. Rozwiązania przykładowych zadań z wykorzystaniem przerzutników zostały przedstawione w punktach: 4.5, 7.2, 8.2.

Tab. 6.9. Standardowe elementy bistabilne

Przerzutnik `SR`	
	`S1` – dominujące wejście ustawiające `R` – wejście zerujące `Q1` – wyjście
Przerzutnik `RS`	
	`S` – wejście ustawiające `R1` – dominujące wejście zerujące `Q1` – wyjście



Rys.6.4. Zasady działania przerzutników SR i RS

6.4.2. Detektory zbocza

Detekcję zbocza (rys. 6.5), tzn. zmiany wartości sygnału cyfrowego pomiędzy dwoma kolejnymi wykonaniami bloku detekcji, umożliwiają:

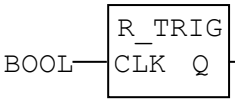
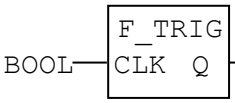
- *detektor zbocza narastającego* `R_TRIG` (ang. Rising Trigger),
- *detektor zbocza opadającego* `F_TRIG` (ang. Falling Trigger).

Detektor zbocza narastającego wykrywa zmianę sygnału wejściowego z wartości `0` na `1` i wtedy jego wyjście jest ustawiane na `1`, jeżeli zmiana sygnału wejściowego nie nastąpi, wyjście bloku jest zerowane. Podobnie działa detektor zbocza opadającego, ale wykrywa zmianę sygnału z wartości `1` na

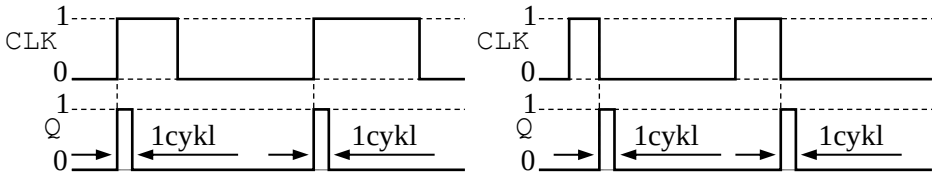
0. Symbole graficzne, opis wejść i wyjść detektorów zestawiono w tab. 6.10. Zasada działania detektorów w formie wykresów czasowych została przedstawiona na rys. 6.6, a przykładowe zastosowania opisano w punktach 7.3, 7.5.3, 8.3, 8.5.3, 10.5.3 i 10.5.4.

Rys.6.5. Zbocze narastające i opadające

Tab. 6.10. Standardowe detektory zbocza

Detektor zbocza narastającego	
	CLK – analizowany sygnał Q – wyjście
Detektor zbocza opadającego	
	CLK – analizowany sygnał Q – wyjście

Detektor zbocza narastającego *Detektor zbocza opadającego*

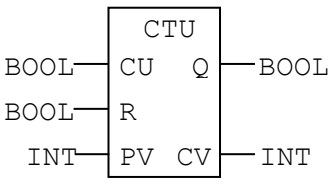
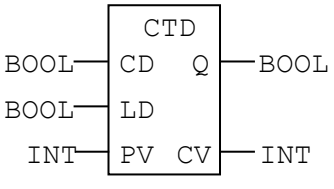


Rys.6.6. Zasady działania detektorów zbocza

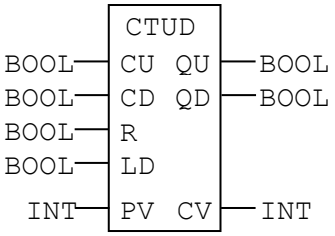
6.4.3. Liczniki

Zliczanie zboczy narastających umożliwiają bloki liczników. Symbole graficzne, opis wejść i wyjść liczników zestawiono w tab. 6.11.

Tab. 6.11. Standardowe liczniki

Licznik zliczający w górę	
	CU – sygnał o zboczach zliczanych w górę R – wejście zerujące licznik PV – wartość zadana Q – wyjście CV – wartość bieżąca
Licznik zliczający w dół	
	CD – sygnał o zboczach zliczanych w dół LD – wejście ustawiające licznik PV – wartość zadana Q – wyjście CV – wartość bieżąca

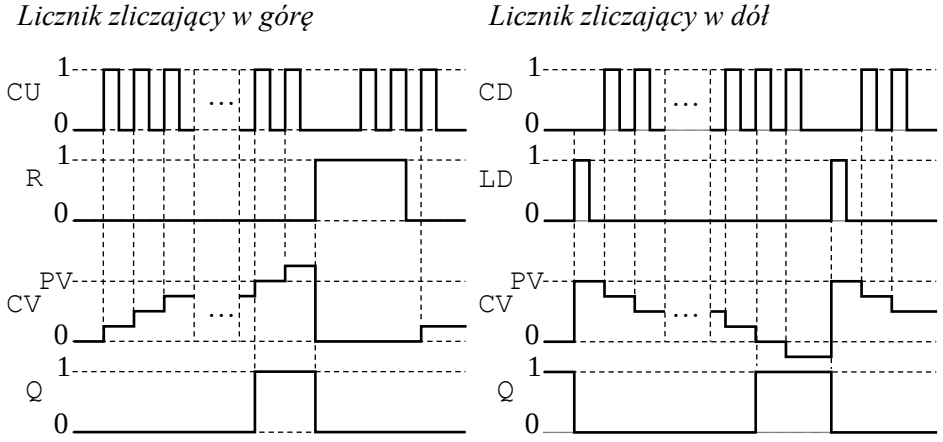
Tab. 6.11. c.d.

Licznik zliczający w górę i w dół	
	CU – sygnał o zboczach zliczanych w górę CD – sygnał o zboczach zliczanych w dół R – wejście zerujące licznik LD – wejście ustawiające licznik PV – wartość zadana QU – wyjście włączane po przekroczeniu PV QD – wyjście włączane po przekroczeniu 0 CV – wartość bieżąca

Norma definiuje:

- *liczniki zliczające w górę* CTU (ang. Counter Up),
- *liczniki zliczające w dół* CTD (ang. Counter Down),
- *liczniki zliczające w górę lub w dół* CTUD (ang. Counter Up Down).

Wszystkie liczniki zliczają zbocza narastające, tzn. wykrywają zmianę sygnału z wartości 0 na 1. Na wyjściu licznika można odczytać informację czy został on już włączony (tzn. czy osiągnął określoną wartość), można również odczytać zliczoną przez licznik liczbę wykrytych zboczy narastających. Zasada działania liczników w formie wykresów czasowych została przedstawiona na rys. 6.7, a przykładowe zastosowania zostały pokazane w punktach 7.4, 8.4 i 10.5.4.



Rys.6.7. Zasady działania liczników

Z wykresu czasowego licznika zliczającego w górę widać, że każde zbocze narastające sygnału CU powoduje zwiększenie wartości wyjścia CV (ang. Current Value). Na wyjściu Q pojawia się sygnał o wartości 1, dopiero gdy wartość CV osiągnie wartość zadaną PV (ang. Preset Value). Licznik można wyzerować, podając na wejście R sygnał o wartości 1.

Z wykresu czasowego licznika zliczającego w dół widać, że dla bieżącej wartości licznika mniejszej lub równej 0 wyjście Q jest ustawiane na 1. Wartość początkową określoną na wejściu PV ustawia się, wprowadzając na wejście LD sygnał o wartości 1. Każdorazowe wystąpienie zbocza narastającego na wejściu CD powoduje zmniejszenie bieżącej wartości licznika CV.

Licznik zliczający w górę i w dół stanowi właściwie połączenie liczników zliczających wyłącznie w górę i wyłącznie w dół. Licznik ten ma zarówno wejścia pochodzące od licznika CTU (wejścia CU i R), jak i pochodzące od licznika CTD (wejścia CD i LD). Licznik inkrementuje swoją wartość po napotkaniu zbocza narastającego na wejściu CU i dekrementuje swoją wartość po napotkaniu zbocza narastającego

na wejściu CD. Przekroczenie górnego zakresu zliczania jest sygnalizowane na wyjściu QU, a przekroczenie dolnego zakresu (zera) na wyjściu QD. Bieżącą wartość licznika można odczytać z wyjścia CV, a jego stan można zmienić, ustawiając wartość 1 na wejściach R (zerowanie stanu licznika) lub LD (ustawianie wartości z wejścia PV).

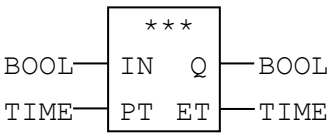
6.4.4. Timery

Timery są blokami funkcjonalnymi, które generują sygnał wyjściowy, uwzględniając czas, który upłynął od pojawienia się określonego zdarzenia. W normie zdefiniowane zostały następujące podstawowe timery:

- *timer ON-delay* TON, który realizuje włączanie z opóźnieniem,
- *timer OFF-delay* TOF, który realizuje wyłączanie z opóźnieniem,
- *timer Pulse* TP, który generuje sygnał o określonym czasie trwania.

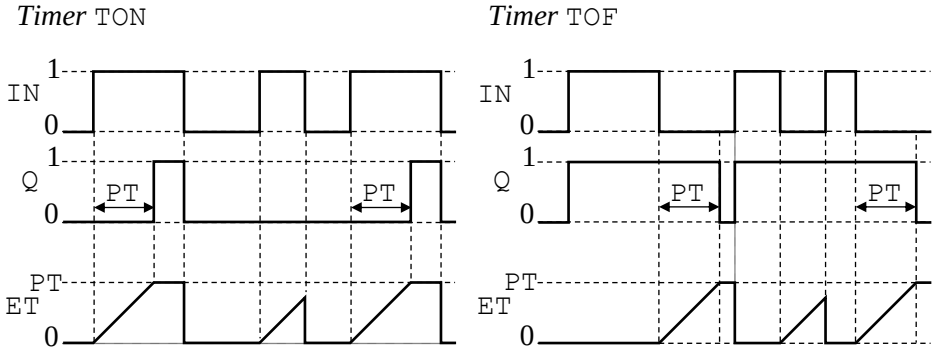
Symbole graficzne timerów różnią się właściwie tylko nazwą timera – zostały one opisane w tab. 6.12. Zasadę działania timerów przedstawiono w formie wykresów czasowych na rys. 6.8 i 6.9. Przykładowe zastosowania timerów zostały pokazane w punktach 7.5 i 8.5.

Tab. 6.12. Standardowe timery

Timery: TON, TOF i TP	
 (***) to nazwa timera	IN – wejście uruchamiające timer PT – czas trwania (ang. Preset Time), znaczenie uzależnione ot typu timera Q – wyjście ET – czas od uruchomienia timera (ang. Elapsed Time)

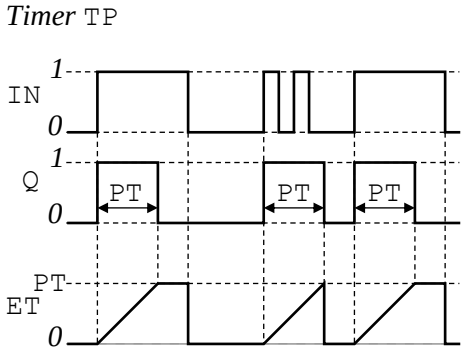
Z analizy przebiegów na rys. 6.8 wynika, że:

- timer TON włącza wyjście Q po czasie PT od włączenia sygnału IN, sygnał 0 na wejściu IN zeruje timer i wyłącza wyjście Q, jeśli sygnał na wejściu IN jest włączony na czas krótszy niż PT wyjście Q nie jest włączane,
- timer TOF wyłącza wyjście Q po czasie PT od wyłączenia sygnału IN, sygnał 1 na wejściu IN zeruje timer i włącza wyjście Q, jeśli sygnał na wejściu IN jest wyłączony na czas krótszy niż PT wyjście Q nie jest wyłączane,



Rys.6.8. Zasady działania timerów TON i TOF

Z wykresów czasowych timera TP (rys. 6.9) wynika, że włącza on wyjście Q na czas PT z chwilą włączenia sygnału IN. Zmiany sygnału na wejściu IN w czasie pracy timera są ignorowane.



Rys.6.9. Zasada działania timera TP

6.5. Zadania

6.1. Zapisz blok deklaracji zmiennych, definiując w nim zmienne boolowskie:
   a) `x1` i `x2` kojarzone z pierwszym i drugim wejściem modułu wejściowego umieszczonego w pierwszym gnieździe sterownika,
   b) `y1` i `y2` kojarzone dopowiednio z pierwszym i drugim wyjściem modułu wyjściowego umieszczonego w drugim gnieździe sterownika,
   c) `m1` i `m2` automatycznie lokowane w pamięci danych użytkownika, zmienna `m2` powinna być zainicjalizowana wartością `1`.

6.2. Dla podanych poniżej bloków narysuj przebieg zmian zmiennej `q`, zakładając, że sygnały wejściowe bloku zmieniają się tak jak pokazano na dołączonych wykresach czasowych.