1.2.5. ST- мова

Мова структурованого тексту ST (Structured Text) - це мова, яка відноситься до класу мов високого рівня, схожа на Паскаль і орієнтована на програмістів. У цій мові підтримуються масиви (також і багатовимірні), контроль перетворення типів, є такі конструкції, як DO-WHILE, REPEAT-UNTIL, FOR-TO-DO, IF-THEN-ELSE, CASE-OF, а також інші зрозумілі будь-якому програмісту оператори. Самостийно ця мова використовується лише для створення складних процедур, які не можуть бути легко виражені за допомогою графічних мов.

Зазвичай мова структурованого тексту (ST) використовується тільки для програмування послідовних кроків і переходів SFC-мови. Ця мова має походження від мови програмування Графсет (Grafcet) французької фірми Telemechanique - Groupe Schneider.

1.2.5.1. Синтаксис ST-мови

Основою ST-програми є вирази. Вираз складається із операндів (змінних, констант, функцій і функціональних блоків), розділених операторами. Результат обчислення виразу, як і у Паскалі, присвоюється змінній за допомогою оператора «:=». Кожний вираз обов'язково закінчується крапкою з комою «;». Стандартні оператори у ST- виразах мають символьний вигляд, наприклад математичні дії: +, −, *, /, операції порівняння: <, >, <=, >= і т. ін. Окрім операторів, елементи виразу для кращого сприйняття можна відділяти пропусками і табуляціями . У текст можуть бути введені коментарі. Коментар повинен починатися символами "(*",а закінчуватися символами"*)". Коментарі можна вставляти там, де допустимі пасивні роздільники (пропуски і табуляції).

Обчислення виразів виконується відповідно до правил пріоритету. Оператор з найвищим приоритетом виконується першим, оператор з нижчим пріоритетом –другим і т.д., доки не будуть виконані усі оператори. Оператори з однаковим пріоритетом виконуються зліва направо. У табл. 1.5 приведено список ST- операторів, розташованих у порядку пріоритету.

Таблиця 1.5. Список ST- операторів

Оператор	Приклад	Значення прикладу	Опис	Пріоритет
( )	(1+2)*(4+5)	45	Круглі дужки	Най-вищий
**	3.0**4	81.0	Піднесення	
-	-10	-10	Заміна знаку	
NOT	NOT TRUE	False	Числове доповнення	
*	10*3	30	Множення	
/	6/2	3	Ділення	
MOD	17MOD 10	7	За модулем	
+	2+3	5	Додавання	
-	4-2	2	Віднімання	
<,>,<=,>=	4>12	False	Порівняння	

=	T#26h=T#1d2h	True	Рівність	
<>	8<>16	True	Нерівність	
&, AND	TRUE&FALSE	False	Логічне „І”	
OR	TRUE OR FALSE	True	Логічне „АБО”	
XOR	TRUE XOR FALSE	True	Логічне„ВиключнеАБО”	Най-нижчий

Для кожного окремого виразу, об'єднуючого операнди з одним оператором тип операндів має бути одним і тим же. При цьому цей окремий вираз має той самий тип, що і його операнди, і може бути використаним у складнішому виразі.

Дужки використовуються для того, щоб відділити складові виразу і явно визначити пріоритетність операцій. Коли в складному виразі немає дужок, послідовність операцій задається неявно пріоритетністю операторів ST.

Основні оператори ST-мови :

- присвоєння;
- виклик підпрограми або функції;
- виклик функціонального блока;
- оператори вибору (IF, THEN, ELSE, CASE...);
- оператори циклу (FOR, WHILE, REPEAT...);
- управляючі оператори (RETURN, EXIT...);
- спеціальні оператори для зв'язку з такими мовами як SFC.

Оператор присвоєння є найпростішим оператором, який частіше всього застосовується при програмуванні. Він призначений для обчислення нового значення певної змінної, а також визначення значення функції, що повертається.

Перед оператором присвоєння «:=» знаходиться операнд (змінна або адреса), якому присвоюється значення виразу, що стоїть після оператора присвоєння. Синтаксис оператора присвоєння має традиційний для мов програмування вигляд:

<змінна> := <будь- який вираз >;

Змінна має бути внутрішньою або вихідною.

Приклад:

*Var1: = Var2 * 10;*

Після виконання цієї операції Var1 приймає значення вдесятеро більше, ніж Var2.

Виклик функції може бути використаний у будь-якому виразі. Щоб викликати функцію, необхідно знати її ім’я і параметри, якими вона характеризується. Функції, що викликаються, мають бути написані мовами IEC-стандарту або мовою "C".

Синтаксис виклику функції такий:

<змінна> := *<ім’я функції> (<пар1>, ... <парN>*);

Тут *(<пар1>, ... <парN>*)- параметри функції.

Змінна набуває значення функції, що повертається. Тип значення і параметри виклику повинні відповідати інтерфейсу, визначеному для функції.

Приклад : виклик IEC функцій

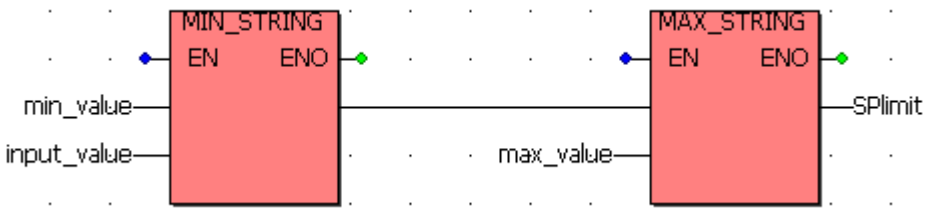
(Головна ST програма *)*

(набуває ціле значення і перетворює його на обмежене часове значення *)*

ana_timeprog := SPlimit (tprog_cmd);

*appl_timer := ANY_TO_TIME (ana_timeprog * 100);*

(FBD функція, що викликається, з ім'ям 'SPlimit' *)*



Виклик функціонального блока із стандартної бібліотеки або бібліотеки користувача здійснюється після оголошення його унікальної копії (екземпляра), визначення імені копії і типу блока. Функціональний блок викликається за допомогою імені екземпляра і списку вхідних параметрів з присвоюванням даних у круглих дужках.

Синтаксис виклику функціонального блока:

(* викликати функціональний блок *)

< ім'я блока> (<пар1>, <пар2> ...);

(* одержати параметри, що повертаються *)

<результат> := *<ім'я блока>*. *<возвр_пар1>*;

...

<результат> := *<ім'я блока>*. *<возвр_парN>*;

Приклад:

(Виклик функціонального блоку ST- програмою *)*

(оголосити екземпляр функціонального блоку в словнику: *)*

(trigb1 : блок R_TRIG - визначення переднього фронту *)*

(активізація функціонального блоку з мови ST *)*

trigb1 (b1);

(доступ до параметрів, що повертаються *)*

If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;

Оператор циклу з передумовою **WHILE** (*ДОКИ, У ТОЙ ЧАС ЯК*) забезпечує повторне виконання виразу, доки булева умова ***перед*** виконанням кожного циклу має значення TRUE. У протилежному випадку, коли значенням умови є FALSE, виконання оператора циклу закінчується. Тіло циклу зовсім не виконується, якщо булева умова із самого початку має значення FALSE. Якщо ж булева умова ніколи не приймає значення FALSE, тіло циклу виконується безкінцево. Щоб такого не трапилося, значення змінної булевої умови в тілі

циклу обов'язково має змінюватися, наприклад, шляхом інкременту або декременту лічильника.

Синтаксис WHILE має вигляд:

WHILE *<булева умова>* ***DO***

<вираз>;

<вираз>;

... ;

END_WHILE

Приклад:

```
WHILE Counter<>0 DO
   Var1:=Var1*2;
   Counter:=Counter-1;
END_WHILE
```

У цьому прикладі доки Counter <>0 змінна Var1 подвоюється.

Оператор циклу з постумовою **REPEAT (***ПОВТОРИТИ***)** аналогічний попередньому оператору WHILE тільки його умова перевіряється ***після*** виконання чергової ітерації, тобто гарантується хоча б одноразове виконання циклу. Окрім того, критерієм завершення циклу є рівність виразу константі TRUE. Якщо вираз дорівнює FALSE, то цикл повторюється Щоб не було безкінцевого виконання циклу, значення змінній булевої умови у тілі циклу, як і у попередньому випадку, обов'язково має змінюватися.

Синтаксис REPEAT:

REPEAT

<вираз>;

<вираз>;

... ;

UNTIL *<булева умова>****;***

END_REPEAT;

Приклад:

```
REPEAT
Proc1(x, y+1);
I:=i-1
UNTIL
   I=0
END_REPEAT;
```

Цикл виконується до тих пір, доки "і" не стане більше нуля.

Оператор циклу з параметром **FOR** *(ДЛЯ)* використовується, коли кількість повторювань циклу може бути визначеною перед його початком. Таким чином, перед виконанням циклу лічильник отримує початкове значення і

тіло циклу повторюється, доки значення лічильника збільшуючись не перевищить кінцеве значення.

Синтаксис FOR:

FOR<*лічильник*>*:=*<*початкове значення*>***TO***<*кінцеве значення*>
{BY<*крок*>***}DO***
<*вираз*>*;*
<*вираз*>***;***
… ***;***
END_FOR;

Частина конструкції у фігурних дужках не обов'язкова, вона визначає крок приросту лічильника. За умовчанням лічильник збільшується на одиницю у кожній ітерації. Лічильником може бути змінна будь-якого цілого типу.

Крок зміни лічильника ітерацій може бути і від'ємний. У цьому випадку початкове значення має бути більше за кінцеве. Тоді цикл закінчується, коли значення лічильника стає менше кінцевого.

Приклад:

```
Varl := 0
FOR Counter := 1 TO 5 DO
   Varl := Varl *2;
END_FOR
Erg:= Var1
```

Даний цикл буде виконуватися 5 разів і Varl матиме значення 32.

Використовуючи умовний оператор **IF** *(ЯКЩО)*, можна перевірити умову і залежно від цієї умови виконати будь-які дії. У якості умови використовується значення логічного виразу.

Синтаксис IF:

IF <*булева умоваIF*> **THEN**
<*виразIF*>*;*
{ELSIF <*булева умова ELSIF1*> **THEN**
<ELSIF- вираз1>;
… ;
ELSIF <*булева умоваELSIF n*> **THEN**
<ELSIF_вираз n>
ELSE
<ELSE-вираз>**}**
END_IF;

Якщо <*булева умоваIF*> TRUE, тоді виконується вираз першої групи <*виразIF*>. Інші вирази пропускаються, альтернативні умови не перевіряються.

У випадку, коли <*булева умоваIF*> є FALSE, послідовно перевіряється решта логічних виразів. Перша знайдена умова TRUE призведе до виконання виразів, що стоять після цієї логічної умови, до наступного ELSIF або ELSE.

ELSIF може бути декілька або зовсім не бути.

Коли усі логічні вирази помилкові, то виконуються інструкції, що стоять після ELSE, якщо вона є. Якщо ELSE немає, то нічого не виконується.

Частина конструкції у фігурних дужках не обов'язкова.
У найпростішому випадку оператор IF містить тільки одну умову.
Приклад:
IF temp < 20
THEN heating_on: = TRUE;
ELSE heating_on: = FALSE;
END_IF

У даному прикладі нагрівання (heating) вмикається, коли температура знизиться нижче 20 градусів, інакше воно залишиться вимкненим.

За допомогою оператора множинного вибору CASE *(У ВИПАДКУ)* можна виконати один з декількох виразів. Вибір здійснюється відповідно до значення поточного обчислення цілочислової змінної або виразу, що записані після оператора CASE .

Синтаксис оператора множинного вибору має вигляд:

CASE < *цілочислова змінна* > **OF**
< *значення 1*>:< *вираз 1*> ;
< *значення 2* > , < *значення 3* >,< *значення4* >:< *вираз 2* > ;
< *значення 5* > .. < *значення10* >:< *вираз 3* >;
...
< *значення n* >:<*вираз n* >
ELSE
< *вираз* ELSE> ;
END_CASE

Оператор CASE виконується відповідно до наступних правил:

- У випадку <цілочислова змінна > має < значення і >, то виконується < вираз і>, тобто <значення і>:< вираз і> . Інші умови не аналізуються.
- У випадку <цілочислова змінна > не дорівнює жодному з вказаних значень, то виконується <вираз ELSE>. Цей вираз не є обов'язковим.
- Щоб той самий вираз виконувався при різних значеннях <цілочислової змінної >, необхідно перелічити ці значення через кому: < значення 2 >,< значення 3 >,< значення4 >:< вираз2 > ;
- Щоб той самий вираз виконувався для певного діапазону значень <цілочислової змінної >, необхідно вказати початкове та кінцеве значення і відокремити їх двома крапками: < значення 5 > .. < значення10 >:< вираз 3 >.

Значеннями оператору множинного вибору CASE можуть бути тільки цілі константи, змінні використовувати не можна. Однакові значення в альтернативах вибору задавати не можна , навидь у діапазонах.

Приклад:

```
CASE INT1 OF
1,5: BOOL1:=TRUE;
        BOOL3:=FALSE;
2: BOOL2:=FALSE;
        BOOL3:=TRUE;
10..20: BOOL1:=TRUE;
```

```
ELSE
        BOOL1:= NOT BOOL1;
        BOOL2:= BOOL1 OR BOOL2;
END_CASE;
```

Якщо INT1 дорівнює 1 або 5, то BOOL1:=TRUE; і BOOL3:=FALSE. Якщо INT1 дорівнює 2, то BOOL2:=FALSE; BOOL3:=TRUE. Якщо INT1 дорівнює значенню, що знаходиться поміж 10 і 20 включно, то BOOL1:=TRUE. Якщо ж жодної умови не виконується, то BOOL1:= NOT BOOL1; BOOL2:= BOOL1 OR BOOL2;

Якщо в циклах FOR, WHILE, REPEAT є оператор **EXIT** *(ВИХІД)*, то цикл закінчує свою роботу незалежно від значення умови виходу. EXIT звичайно використовується з оператором IF усередині блоку FOR, WHILE або REPEAT.

Приклад:

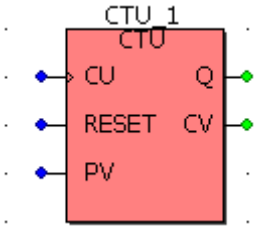
```
length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code = searched_char) THEN
        found := YES;
        EXIT;
    END_IF;
END_FOR;
```

У наведеній програмі здійснюється пошук символу в рядку.

Оператор **RETURN** *(ПОВЕРНЕННЯ)* припиняє виконання поточної програми і здійснює негайне повернення з POU. У блоці дій SFC-програми оператор RETURN позначає кінець виконання тільки даного блоку[4,7-9].

Приклад:

Програма, яка реалізує стандартний функціональний блок лічильника,



має вигляд:

```
If NOT (CU) then
    Q := false;
    CV := 0;
    RETURN; (* закінчити програму *)
end_if;
```

```
    if RESET then
        CV := 0;
    else
        if (CV < PV) then
            CV := CV + 1;
        end_if;
end_if;
Q := (CV >= PV);
```

1.2.5.2. Приклад програмування ST – мовою

Як приклад програмування ST – мовою розробимо програму керування роботою двох двигунів згідно з алгоритмом, наведеним у параграфі 1.2.2.2.

Проектний код ST-мовою має вигляд:

```
(*Програмування запуску двигуна_1*)
RS_1(SET:=(start_1 AND NOT motor_2)OR time_2,
RESET1:=stop OR time_1 OR autostop);
motor_1:=RS_1.Q1;
(*Програмування таймера роботи двигуна_1*)
M_TIME_1(IN:=motor_1,PT:=t#5s);
time_1:=M_TIME_1.Q;
act_time_1:=M_TIME_1.ET;
(*Програмування лічильника кількості натисків кнопки запуску
двигуна_2*)
MOT_COUNT(CU:=start_2,RESET:=motor_2,PV:=int#2);
out:=MOT_COUNT.Q;
Pressed:=MOT_COUNT.CV;
RS_2(SET:=(out AND NOT motor_1)OR time_1,
RESET1:=stop OR time_2 OR autostop);
motor_2:=RS_2.Q1;
(*Програмування таймера роботи двигуна_2*)
M_TIME_2(IN:=motor_2,PT:=t#10s);
time_2:=M_TIME_2.Q;
act_time_2:=M_TIME_2.ET;
(*Програмування лічильника циклів роботи двигуна та його
зупинення*)
if time_1 OR time_2 then
COUNTER:=COUNTER+1;
end_if;
if COUNTER>=10 then
autostop:=INT_TO_BOOL (COUNTER);
end_if;
```

Тут:

START_1 і START_2 - кнопки початкового пуску двигунів;

MOTOR_1 і MOTOR_2 – перший і другий двигуни;

M_TIME_1 і M_TIME_2 – екземпляри функціонального блока таймера TON, які відлічують час роботи двигунів;

MOT_COUNT – екземпляр функціонального блока лічильника CTU, який підраховує кількість натисків на кнопку START_2;

RS_1 і RS_2 – екземпляри функціонального блока детектора імпульсу переднього фронту R_TRIG.

COUNTER– лічильник циклів роботи двигунів;

STOP – кнопка непередбаченої зупинки двигунів.