3. Бібліотека елементарних функцій EF/блоків EFB, та похідних блоків DFB.

3.1 Основні поняття.

В програмі користувача можна використовувати різні типи програмних блоків, в які можна передавати фактичні параметри та отримувати на виході результати розрахунку. Можна користуватися як бібліотечними готовими блоками (функції, процедури,елементарні функціональні блоки) так і розробити власні (похідні функціональні блоки).

Функція (Elementary Functions - EF) - бібліотечний програмний блок, який може мати декілька входів і тільки один вихід. Функція не має внутрішньої пам'яті,тобто вона не може між викликами "всередині себе" зберігати розраховані значення.

Це значить, що вихід функції завжди однозначно залежить тільки від станів її входів.

Процедура (Procedure) – бібліотечний програмний блок, який може мати декілька входів і декілька виходів. Крім наявності декількох виходів, відмінність процедур від функцій є в можливості використання параметрів типу VAR_IN_OUT(вхід/вихід).

Елементарний функціональний блок (Elementary Function Block, ***EFB***) – готовий бібліотечний програмний блок, який має внутрішню пам'ять, тобто може зберігати проміжні розрахункові значення між викликами. Перед використанням функціонального блоку у програмі користувача, спочатку створюють ***екземпляр функціонального блоку*** (FB instance), якому дають унікальне ім'я. Перегляд,створення та видалення функціональних блоків в UNITY PRO виконується у розділі"Variables & FB Instances" у підрозділі "Elementary FB Instances". Екземпляр функціонального блоку буде вміщувати всі внутрішні дані, тобто його пам'ять. Данідля екземплярів функціональних блоків розміщуються в нелокалізованій області пам'яті, тому за адресою до них звернутися неможливо.

Похідний функціональний блок (Derived Function Block, ***DFB***) – функціональний блок, який розроблений користувачем засобами UNITY PRO.

Спочатку в UNITY PRO у розділі проекту Derived FB Types розробляється ***DFB тип***(DFB Type), на основі якого потім створюються екземпляри. При створенні DFB типу визначають: ім'я типу; інтерфейс блоку (вказують ім'я та тип вхідних та вихідних параметрів функціонального блоку); внутрішні змінні та внутрішні екземпляри функціональних блоків, в яких будуть зберігатися проміжні результати між викликами; створюють програму для функціонального блоку на мовах ST, IL,LD або FBD. Використання екземплярів DFB аналогічно як і EFB.

Виходи EFB та DFB є змінними екземпляру, тобто до них можна звертатись як до інших змінних. Для цього вказується ім'я екземпляру а потім через крапку назва виходу. Так наприклад запис "Timer1.Q" означає, що йде звернення до виходу Q екземпляру з іменем "Timer1".

3. 2 Використання бібліотечних блоків.

В ***бібліотеці типів*** UNITY (Types Library) доступна велика кількість функцій, процедур та елементарних функціональних блоків. Вони можуть бути використані в будь-якій із мов програмування. Так наприклад функція ADD призначена для

додавання в мові FBD,однак її можна викликати і в ST, LD та IL, хоча для додавання там зручніше використовувати відповідний оператор "+".

Найбільш загальні програмні блоки зведені у розділі стандартної бібліотеки. Частину з них наведено в таб.3.1. Ряд блоків з бібліотеки управління, які використовуються при реалізації алгоритмів регулювання наведені у таб.3.2.

Таблиця 3.1.Стандартна бібліотека (Standard Library, часткова вибірка)

Математичні (сімейство Mathematics)		
ADD – додавання	SUB – віднімання	SQRT – квадратний корінь
MUL – множення	MOVE – присвоєння	
DIV – ділення	NEG – зміна знаку	
Порівняння (сімейство Comparison)		
EQ – дорівнює (=)	GE – більше або дорівнює (≥)	LE – менше або дорівнює (≤)
NE – не дорівнює (≠)	GT – більше (>)	LT – менше (<)
Логічні (сімейство LOGIC)		
AND – логічне ТА	SR – SR тригер	FE – визначення переднього фронту
OR – логічне АБО	RS – RS тригер	RE – визначення заднього фронту
XOR – виключне АБО	SET – встановлення в лог. "1"	
NOT – логічне НІ	RESET – встановлення в лог. "0"	
Статистичні (сімейство Statistical)		
MUX – мультиплексор	LIMIT – обмеження, LIMIT_IND – обмеження з індикацією	MAX – вибір максимального
SEL – бінарний вибір	AVE – середнє значення	MIN – вибір мінімального
Таймери та лічильники (сімейство Timers & Counters)		
CTD – лічильник вниз	CTUD – лічильник вгору/вниз	TOF – таймер з затримкою на виключення
CTU – лічильник вгору	TP – таймер формування імпульсу	TON – таймер з затримкою на включення
Перетворення типів (сімейство Type to Type)		
INT_TO_REAL	WORD_TO_BIT	WORD_TO_INT
REAL_TO_INT	BIT_TO_WORD	INT_TO_WORD
Цілочисельне регулювання (сімейство CLC_INT)		
PID_INT – ПІД регулятор	PWM_INT – широтно-імпульсне перетворення	SERVO_INT – управління серводвигунами

Таблиця 3.2Бібліотека управління (Control Library, часткова вибірка).

PI_B – базовий ПІ регулятор	PWM1 – широтно-імпульсна модуляція
PIDFF – повний ПІД регулятор	SERVO – управління серводвигунами
LAG_FILTER – фільтрація (аперіодична ланка)	SAMPLETM – шаблон часу для періодичного виклику блоків регулювання
SCALING – маштабування	DTIME – транспортне запізнення

Математичні функції. Математичні функції призначені для виконання таких базових операцій як додавання, віднімання, множення, ділення, присвоєння, а також тригонометричних, експоненційних та інших математичних функцій. Базові математичні операції в основному призначені для FBD, оскільки в інших мовах для цього використовуються відповідні оператори.

Більшість функцій у бібліотеці представлені для різних типів даних таких як INT, DINT, UINT, UDINT та REAL. Так для додавання цілих чисел можна використати функцію ADD_INT, або універсальну функцію ADD. Тим не менше, використання універсальної (за типом даних) функції не дозволяє використовувати в якості її параметрів різні за типом дані. Надалі ми будемо розглядати тільки

універсальні за типом даних функції.

Всі розглянуті в таб.3.1 функції мають один вихід – результат виконання математичної функції. Функції SUB (віднімання) та DIV(ділення) мають по два входи, функції ADD (додавання) та MUL (множення) можуть мати від 2-х до 32-х входів (вибирається після встановлення), всі інші – один вхід. На рис.3.1 показаний приклад використання математичних функцій для реалізації залежностей:

$$X = \frac{A \cdot B \cdot C + D - E + \sqrt{F} + G}{H};$$

$$Y = -(A \cdot B \cdot C + D - E + \sqrt{F} + G);$$

$$Z = A \cdot B \cdot C + D - E + \sqrt{F} + G;$$

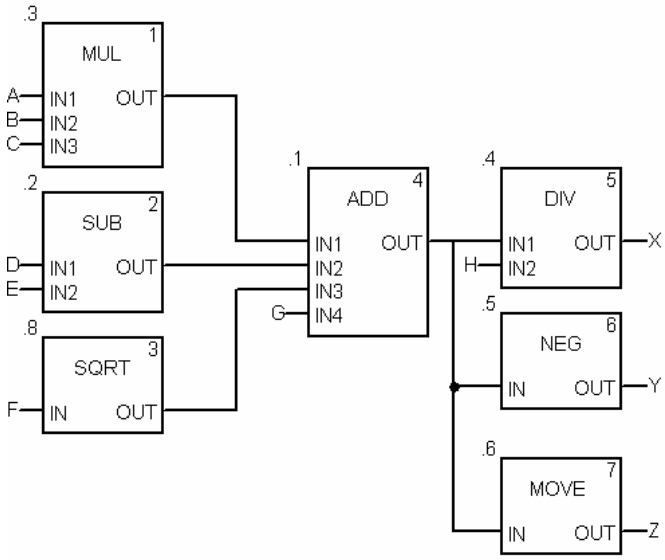

Рис.3.1. Приклад використання математичних функцій в FBD

Функції порівняння. Дані функції використовуються для порівняння двох або більше вхідних величин в мові FBD та LAD (однак дозволяється їх використання і в інших мовах). Результат порівняння є булевою величиною ("0" або "1"). За допомогою функцій порівняння та входів EN зручно проводити управління виконання логікою програми.

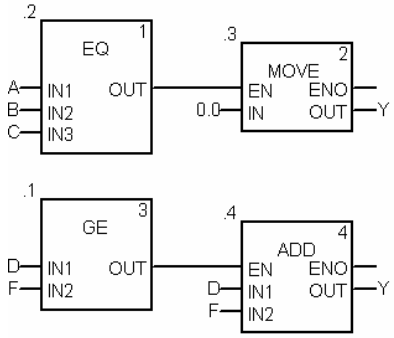

Рис.3.2. Приклад використання функцій порівняння в FBD

На рис.3.2 показаний приклад використання функцій порівняння для виконання логічного управління. При умові що A=B=C, змінній Y буде присвоєне значення 0. При умові, що D≥F, виконається дія Y:=D+F.

Альтернативою функціям порівняння в FBD може бути ST вираз, який повертає булевий результат.

Логічні блоки. Логічні функції AND, OR, XOR, NOT використовуються для логічних операцій переважно в мові FBD, так як у мові ST для цього використовуються оператори, а в LD – спеціальні графічні оператори. Функції SET та RESET, не мають входів (за винятком EN). При виклику функції SET, вихідний параметр виставляється у логічну одиницю. Аналогічно, при виклику функції RESET вихідний параметр виставляється в логічний нуль.

Функції FE та RE служать для відлову відповідно заднього та переднього фронтів у мовах FBD, ST та IL. У якості вхідного параметру використовується змінні типу EBOOL.

Функціональні блоки SR та RS є тригерами. Тобто при поданні логічної "1" на вхід S на виході по передньому фронту встановлюється логічна "1", а на R – логічний "0". SR та RS тригери відрізняються пріоритетністю входів S та R при одночасному поданні на них логічних "1". У RS тригері пріоритетний вхід R, а у SR, – вхід S. Нагадаємо, що функціональні блоки потребують створення екземплярів.

На рис.3.3 показаний приклад використання логічних блоків в FBD. Змінна Bool1 встановиться в "1" в тому випадку, коли A≥B або C>D. В іншому випадку, ніяких дій зі змінною Bool1 проводитись не буде. Змінна Bool3 виставиться в "1", коли C>D і встановиться у "0" по передньому фронту змінної ebool2, навіть якщо C буде більше D, оскільки вхід R1 має вищий пріоритет. Однак, якщо на наступний цикл після виникнення логічної "1" в ebool2, C буде більше ніж D, умова фронту сигналу вже не буде працювати, тому Bool3 знову переведеться в логічну"1". У разі виникнення переднього фронту у змінній ebool2 змінна B буде збільшуватися на 1.0.

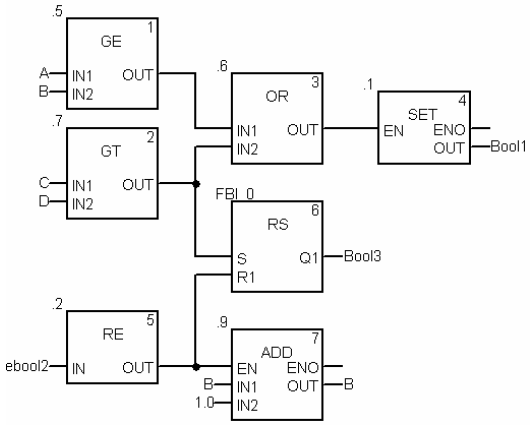

Рис.3.3. Приклад використання логічних блоків в FBD

Статистичні функції. Функції MIN та MAX призначені для запису відповідно мінімального та максимального значення серед вхідних сигналів у вихідний сигнал. Кількість входів може бути змінена до 32.

Функція MUX є мультиплексором, вихід OUT якої перемикається на один із входів IN0-IN30, по номеру що подається на вхід K. Тобто якщо K=2 OUT:=IN2.

Функція SEL перемикає вихід на один із двох входів IN, в залежності від значення вхідного параметру G. Тобто OUT:=IN0 при G=FALSE, OUT:=IN1 при G=TRUE.

Функція AVE реалізує розрахунок зваженого середнього за формулою:

$$Y = \frac{\sum(Ki \cdot Xi)}{\sum(Ki)}$$

де Ki – коефіцієнт і-го вхідного значення; Xi – і-те вхідне значення.

Коефіцієнт і значення вхідного сигналу являються парою вхідних параметрів. Тобто три пари сигналів потребують шість входів (K_X1…K_X6). Непарні входи є коефіцієнтами, парні – значеннями сигналів.

Функція LIMIT забезпечує обмеження вхідної величини IN по мінімуму (віх MN) та по максимуму (вхід MX). Процедура LIMIT_IND працює аналогічно LIMIT однак має додаткові виходи Y_MAX та Y_MIN, який виставляється в логічну "1" при досягненні відповідно максимального та мінімального значень на вході.

Приклад використання статистичних функцій в FBD наведені на рис.3.4. Для кращого розуміння приклад показаний як в режимі редагування та і в режимі виконання програми з анімацією, де для всіх вхідних та вихідних параметрів показані числові значення. На виході блоку ".4" формується зважене середнє 3-х сигналів (K_X2:=D, K_X4:=E, K_X8:=F) з коефіцієнтами (K_X1:=0.1, K_X3:=0.25, K_X5:=0.5). Блок ".1" вибирає максимальне значення серед 4-рьох входів (OUT:=98.0), блок ".5" переключає вихід на перший вхід (OUT:=IN1). Серед двох вхідних сигналів блоку ".2" вибирається 0-вий (OUT:=IN0), оскільки G=FALSE (для блока ".6" не справджується умова IN1<IN2). Блок ".7" обмежує по максимуму вхідну величину (MX:=95), тому на виході OUT:=95, а на виходах MN_IND:=FLASE та MX_IND:=TRUE.

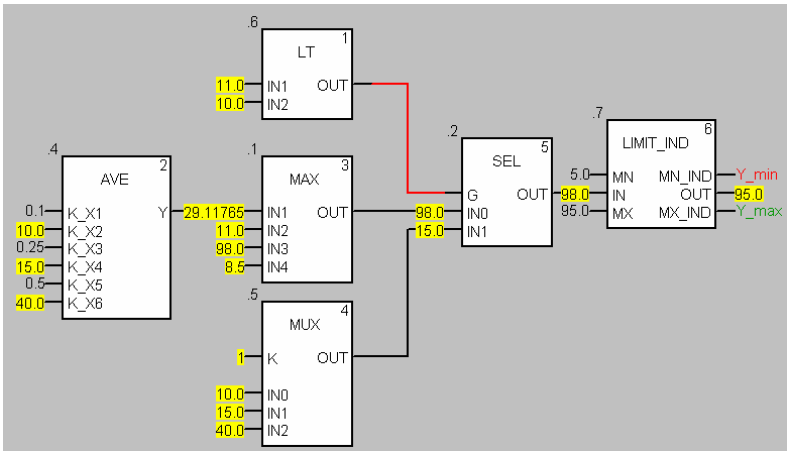

Рис.3.4. Приклад використання статистичних функцій в FBD в режимі анімації

Таймери та лічильники. Таймери та лічильники в UNITY реалізовані у вигляді функціональних блоків, тому потребують попереднього створення екземплярів.

Лічильники доступні у вигляді 3-х функціональних блоків: CTU – підрахунок імпульсів зі збільшенням, CTD – підрахунок імпульсів зі зменшенням, CTUD – реверсивний лічильник.

Лічильник CTU збільшує плинне значення CV, по передньому фронту сигналу на вході CU. На вході PV задається уставка. При досягненні плинного значення CV≥PV, вихід Q:=TRUE. При подачі на вхід R:=TRUE, скидає плинне значення в нуль.

Лічильник CTD зменшує плинне значення CV, по передньому фронту сигналу на вході CD. На вході PV задається уставка. При досягненні плинного значення CV≤0, вихід Q:=TRUE. При подачі на вхід LD:=TRUE, у плинне значення записується значення уставки, тобто CV:=PV.

Лічильник CTUD, об’єднує в собі функції двох лічильників CTU та CTD. Входи CU та CD призначені відповідно для збільшення та зменшення плинного значення лічильника CV. При CV≥PV, вихід QU:=TRUE. При CV≤0, вихід QD:=TRUE. Якщо LD=TRUE, CV:=PV. Якщо R=TRUE, CV:=0.

Таймери TON, TOF та TP мають вхід IN – сигнал запуску таймеру, PT – часова уставка таймеру, вихід таймера Q, та плинне значення таймеру ET. Діаграми роботи даних таймерів показані на рис.3.5.

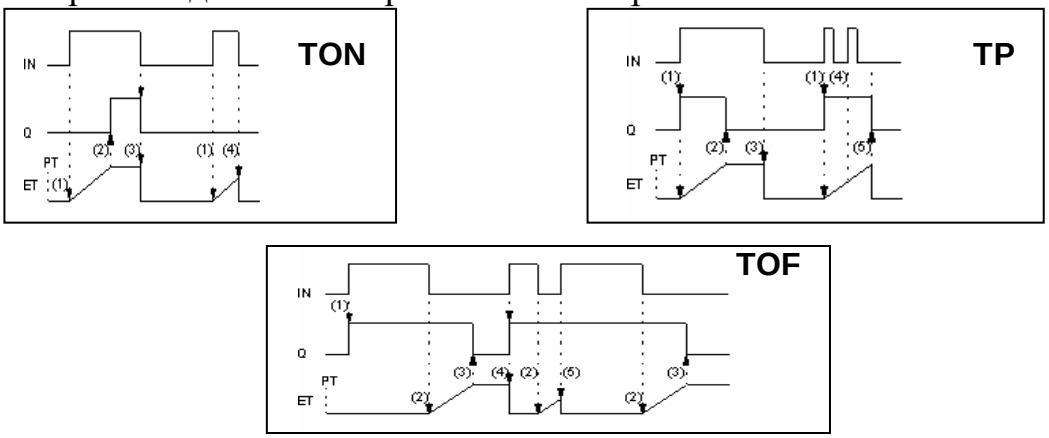

Рис.3.5. Графіки режимів роботи таймерів TON, TOF та TP.

Приклад використання лічильника CTUD та таймера TON в FBD показаний на рис.3.6. Вихід QU лічильника "CounterUD" спрацює при досягненні CV уставки (CV=20). Після досягнення уставки лічильника змінна Y_bool:=TRUE і запуститься таймер Timer1. Через 2с 100 мс після запуску таймера скинеться плинне значення лічильника "CounterUD", тобто CounterUD.CV:=0. Слід звернути увагу на використання виходу EFB "Timer1.Q" в якості фактичного параметру на вході R блоку "CounterUD".

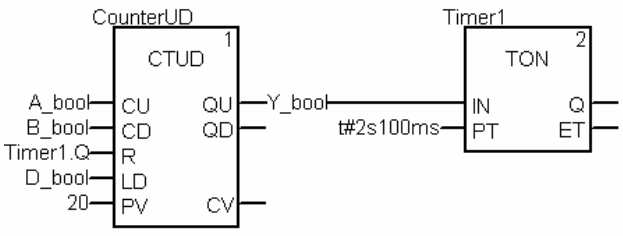

Рис.3.6. Приклад використання функціональних блоків TON та CTUD

Функції перетворення типів. Функція INT_TO_REAL перетворює ціле значення в дійсне, а REAL_TO_INT – навпаки. Дані функції необхідні у випадках, коли над одними даними треба проводити як цілочисельні операції так і операції з плаваючою комою. На рис.3.7 показаний приклад, де аналогову вхідну змінну необхідно помножити на коефіцієнт A (дійсне число) після чого результат подати на аналоговий вихід. Враховуючи, що значення аналогових входів записується у цілочисельні змінні %IW, необхідно спочатку його перетворити в тип REAL (функція INT_TO_REAL), після чого помножити на A. Враховуючи, що значення аналогових виходів зберігається в цілочисельних змінних %QW, розрахований добуток попередньо перетворюється в цілочисельний формат (функція REAL_TO_INT).

Функції INT_TO_WORD та WORD_TO_INT забезпечують перетворення відповідно цілочисельних даних INT в формат слова WORD та навпаки. Формат

WORD використовується для побітових операцій над змінними.

Функція WORD_TO_BIT призначена для побітового розкладення входу IN типу WORD. Виходи BIT0…BIT15 виставляються в TRUE або в FALSE відповідно до значення бітів у вхідному слові. Біти у слові рахуються від молодшого (0-й) до старшого (15-й). Функція BIT_TO_WORD – навпаки, по значенням входів BIT0…BIT15 формує вихідне значення OUT типу WORD. На рис.2.14 показаний приклад, де значення змінної %IW0.1.1 інтерпретується як набір бітів. Спочатку значення перетворюється в тип WORD (функція INT_TO_WORD), після чого значення 0-го біту записується в a_bool, 1-го в b_bool, 7-го в c_bool.

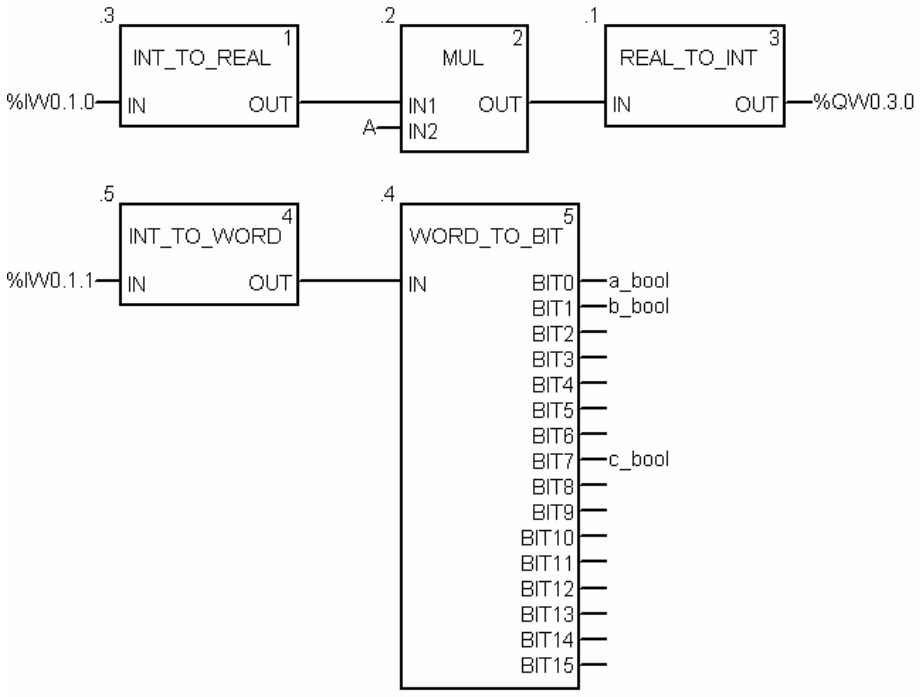

Рис.3.7. Приклад використання функцій перетворення типів в FBD

3.3 Створення функціональних блоків користувача

Часто виникають задачі, які передбачають обробку даних за однаковим алгоритмом. Для того, щоб не повторювати однаковий код програми з різними даними декілька раз, є сенс викликати один і той самий код, однак з різними фактичними параметрами. За таким самим принципом реалізовані функції, процедури та функціональні блоки, які зберігаються у бібліотеці.

При необхідності виклику частин програм декілька раз можна використати підпрограми (SR). Однак в Unity Pro в підпрограми не можна безпосередньо передавати параметри, тобто вони завжди викликаються як частина коду.

Для можливості передачі параметрів до частини програми у багатьох програмних середовищах сучасних ПЛК існують функції та функціональні блоки користувача. В Unity Pro немає можливості створення функцій користувача, однак є ймовірність створення функціональних блоків користувача, які носять назву Derived Function Block (DFB). У принципі якщо DFB-типі не описати змінні, що організовують внутрішню пам'ять то в першому наближенні він може бути використаний як функція.

Для використання функціональних блоків користувача в Unity Pro, спочатку у розділі проекту Derived FB Types розробляється DFB-тип, на основі якого потім створюються екземпляри. Екземпляри функціональних блоків користувача створюються та використовуються аналогічна EFB.

Процес створення DFB типу схожий на створення структурного DDT. При цьому вказується ім'я типу, описується інтерфейс та внутрішня структура типу (рис.3.8). Інтерфейс блока (формальні параметри) описується входами (Inputs) виходами (Outputs) та входами і виходами (Inputs/Outputs), використання яких у програмі аналогічно як для елементарних функціональних блоків. На рис.3.8 показано створення DFB-типу, для якого описані 4 вхідні формальні параметри (Start, LS1, LS2, ManPump) та один вихідний (M1). Порядок розміщення інтерфейсних параметрів вказується у властивості «no».

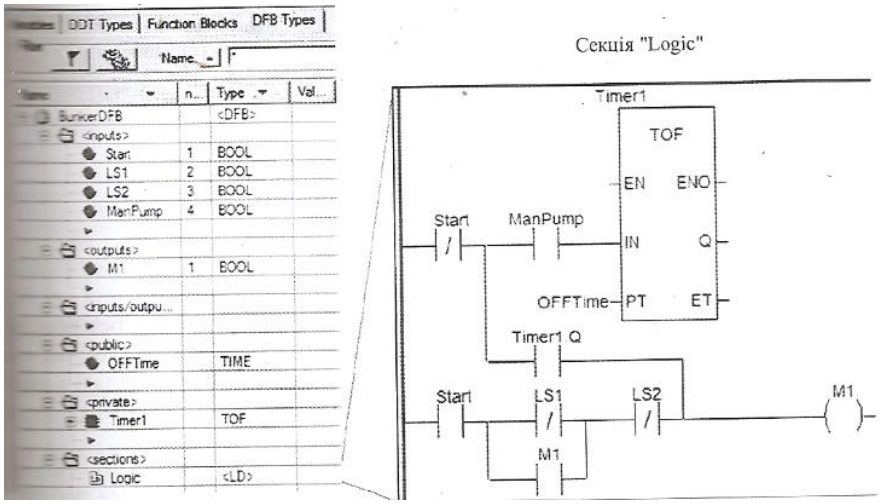

Рис 3.8 Створення типу функціонального блока користувача DFB.

Програма функціонального блока описуються в розділі Sectins. Секції можуть бути створені з використанням мов LD, ST, IL або FBD. У секціях можна використовувати тільки входи, виходи, входи/виходи, локальні змінні або екземпляри та глобальні змінні функціонального блока, бібліотечні функції і процедури. Дозволяється також використовувати системні біти та слова. На рис. 3.8 функціональний блок вміщує тільки одну програмну секцію – «Logic».

Крім інтерфейсної частини, тип може вміщувати ***локальні дані,*** які розташовуються в розділі «private». Локальні дані по суті є пам'яттю функціонального блока, яка інкапсульована в його екземпляр і недосяжна із зовнішньої програми. У прикладі на рис.3.8 в якості локальних даних використовується екземпляр таймера з іменем *Timer1*. Слід звернути увагу, що налаштування *Timer1* і доступ до його полів можливий тільки із середини функціонального блока, тобто з його програмних секцій.

Для можливості доступу до змінних функціонального блока не описуючи їх в інтерфейсі, ці змінні визначають як глобальні в розділі *Public*. У прикладі на рис.3.8 визначена глобальна змінна *OFFTime,* за допомогою якої індивідуально можна налаштувати таймер *Timer1* у програмній секції функціонального блока. При створенні екземпляра значення глобальної змінної при ініціалізації можна прописати в поле *Value*. У програмі на рис. 3.9 під час ініціалізації блока глобальна

змінна *OFFTime* буде отримувати значення T#5s. До глобальної змінної блока в програмі можна також звернутися як до виходу блока тобто через крапку (Рис.3.9).

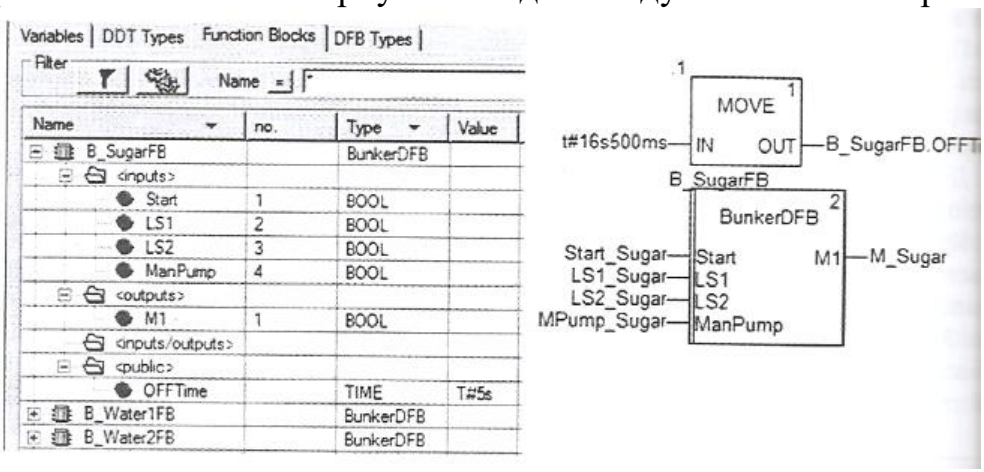

Рис.3.9 Створеення екземплярів функціонального блока користувача DFB та їх використання в програмі