1.2.6. SFC-мова

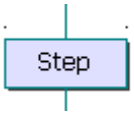
Мова послідовних функціональних схем SFC (Sequential Function Chart) є найважливішою із всього сімейства стандартних мов. Вона використовується для опису алгоритму у вигляді набору зв'язаних пар – крок (step) і перехід (transition). Крок є набором операцій над змінними. Перехід – набір логічних умовних виразів, визначаючих передачу управління до наступної пари крок-перехід. На вигляд опис SFC-мовою нагадує добре відомі логічні блок-схеми алгоритмів. Строго кажучи, SFC-мова не є мовою програмування. Це засіб проектування прикладного програмного забезпечення, що складається з комплексу великої кількості одиниць: програм, функціональних блоків, функцій. Основними перевагами цієї мови є можливість описувати паралельно-послідовні процеси в наочній і компактній формі й автоматичне отримання за цим описом (за наявності транслятора) керуючих програм. SFC-мова дуже зручна для програмування стадійних процесів, систем дозування та бізнес-додатків. Проте SFC-мова не має засобів для опису кроків і переходів. Вони можуть бути виражені тільки засобами інших мов стандарту IEC.

Теоретичною основою SFC-мови є мережі Петрі, за допомогою яких в теорії кінцевих автоматів описується стан складних процесів управління і, які складають основу французької мови програмування контролерів Grafcet.

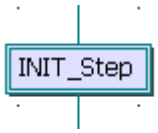
1.2.6.1. Синтаксис SFC-мови

Основними компонентами SFC-мови є кроки і початкові кроки, переходи, орієнтовані сполучення, стрибок на крок, сходження і розбіжності.

Крок зображається прямокутником. Кожному кроку привласнюється ім'я, написане усередині прямокутника.

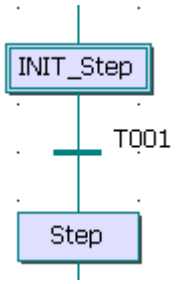


Початковий крок позначається графічним символом з подвійною рамкою.



Кроки не можуть слідувати один за одним. Обов'язково поміж ними має бути перехід.

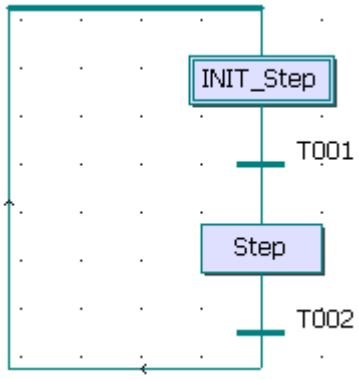
Переходи зображаються маленькими горизонтальними смужками, які перетинають лінії сполучення. Кожному переходу присвоюється ім'я, яке записується поряд.



Перехід виконується, коли перехід дозволено (відповідний йому крок активний) и при цьому умова переходу має значення TRUE.

Для сполучення кроків і переходів використовуються *орієнтовані лінії*. Коли орієнтація не задана явно, сполучення орієнтовано зверху вниз.

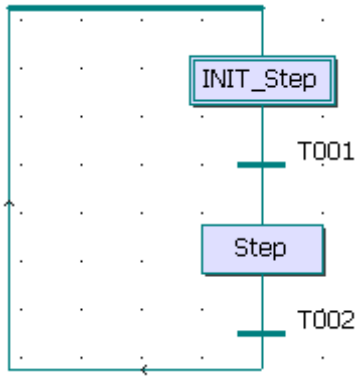
Явна орієнтація від переходу T002 до кроку INIT_Step

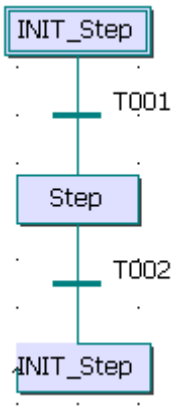


Неявна орієнтація від кроку INIT_Step до переходу T001

Щоб позначити лінію сполучення від переходу до будь-якого кроку, не малюючи лінії, використовується символ *стрибка* з іменем кроку призначення. У той же час за допомогою символу стрибка не можна позначити лінію сполучення від кроку до переходу.

Наступні схеми еквівалентні:





Дії у SFC- програмі зображуються у вигляді прямокутників, розташованих праворуч від символу кроку і приєднаних до них лінією. У лівій частині прямокутника знаходиться класифікатор, можливо, з константою часу, а права містить ім'я дії або логічній змінній. Кроки можуть мати декілька дій.

Дії кроків описуються окремо від них і не належать певному кроку, тому та сама дія може багаторазово використовуватися в межах одного POU.

Класифікатор визначає спосіб впливу активного кроку на дію, табл.1.6.

Таблиця 1.6. Класифікатори SFC-мови

Класифі-катор	Дія	Коментар
N (Non-stored)	Що не зберіга-ється	Дія виконується у кожному робочому циклі протягом активності кроку
R (Reset)	Позачерговий скид	Дія деактивується
S	Що зберігається	Дія активна доки немає скиду. Дія

(Stored)		продовжує виконання у кожному циклі, навіть тоді, коли крок вже не активний
L (time Limited)	Що обмежена за часом	Дія активна протягом вказаного часу, але не триваліше часу активності кроку
D (Delayed)	Що відкладена	Дія активується через заданий час після активації кроку і залишається бути активною, доки крок є активним
P (Pulse)	Імпульс	Дія виконується один раз при активації і другий раз після деактивації кроку
SD (Stored and timeDelayed)	Що зберігається і відкладена	Дія активується через заданий час після активації кроку, навіть якщо крок вже не активний
DS (Delayed and Stored)	Що відкладена і зберігається	Дія активується через заданий час після активації кроку і залишається активною до скиду
SL (Stored and Limited)	Що зберігається і обмежена за часом	Дія активується разом з кроком і залишається бути активною заданий час незалежно від активності кроку

Класифікатори L, D, SD, DS і SL потребують установлення константи у форматі TIME.

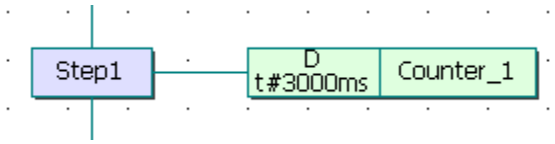
Синтаксис дій на ST та IL- мовах має вигляд:

Action(Класифікатор):

*(*Оператори ST-мови або блок команд IL-мовою*)*

End _Action;

Приклад:



Тут дія Counter_1 написана IL-мовою :

Action(D):

LD Count

ADD INT#1

ST Count

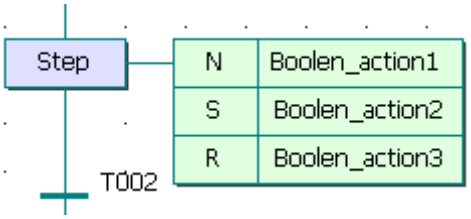
End_Action;

Опис дій, що виконуються протягом активності кроку, здійснюється з використанням текстових властивостей мови SFC і інших мов міжнародного стандарту. Основні типи дій:

- булеві дії із класифікаторами: 'Set', 'Reset' або 'Non-stored';
- імпульсні дії або дії, що не зберігаються із класифікаторами відповідно 'Pulse' і 'Non-Stored';
- SFC дії (управління дочірніми SFC програмами) із класифікаторами 'Set', 'Reset' або 'Non-Stored'.
- Виклик функцій, функціональних блоків і підпрограм, створених будь- якою мовою, окрім SFC.

Булеві дії присвоюють вихідним (OUTPUT) або внутрішнім (MEMORY) логічним змінним значення активності кроку кожного разу, коли поточний крок стає активним. Синтаксис основних логічних дій такий:

N	boolen var	присвоює змінній boolen var сигнал активності кроку.
S	boolen vari	присвоює змінній boolen var значення TRUE, коли сигнал активності кроку стає TRUE.
R	boolen vari	присвоює змінній boolen var значення FALSE, коли сигнал активності кроку стає TRUE.



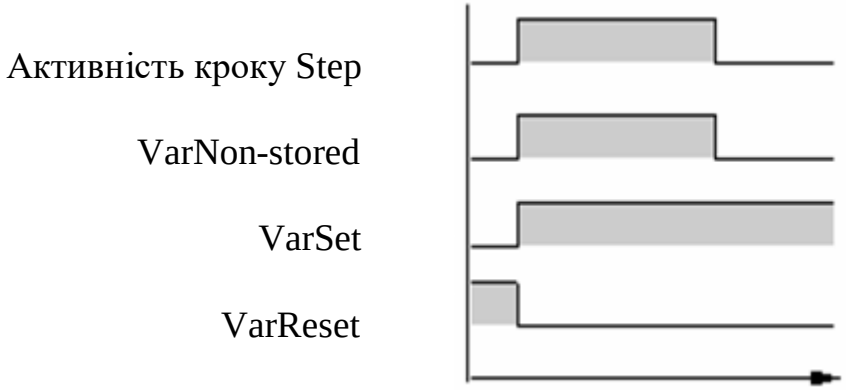
Boolen_actions:

Action(N): VarNon-stored; *End_Action;*

Action(S): VarSet; *End_Action;*

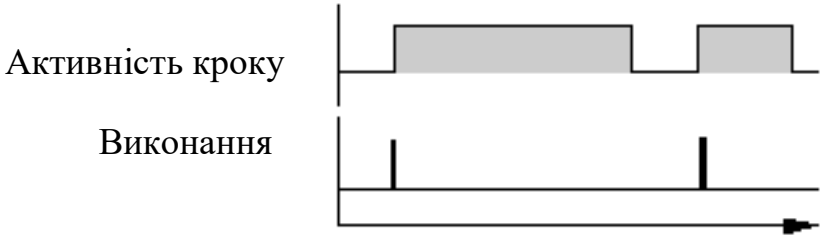
Action(R): VarReset; *End_Action;*

Часова діаграма зміни значень булевих змінних VarNon-stored; VarSet; VarReset; має вигляд:

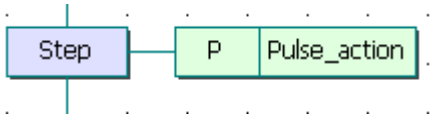


Імпульсні дії **(**Pulse) - це список ST, IL – інструкцій або LD- схем , які виконуються тільки одного разу при активізації кроку .

Часова діаграма імпульсної дії із класифікатором “P” має вигляд:



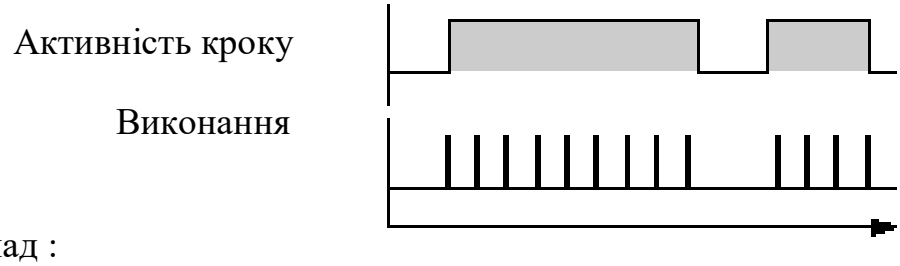
Приклад:



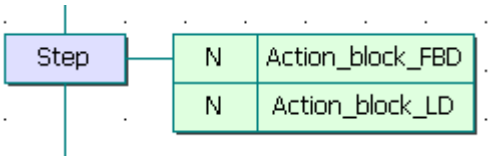
Pulse_action:
Action(P):
Lamp1: TRUE
End _Action;

Дія, що не зберігається (Non-Stored), - це список ST-, IL- інструкцій або LD-, FBD- схем, які виконуються в кожному циклі протягом усього періоду активності кроку. Інструкції пишуться відповідно до синтаксису використовуваної мови.

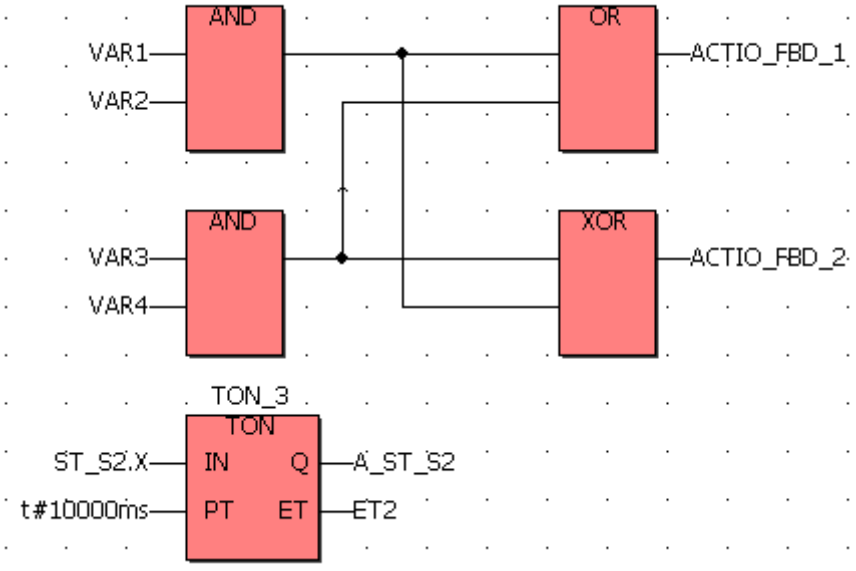
Нижче показаний результат дії, що не зберігається із класифікатором “N”:



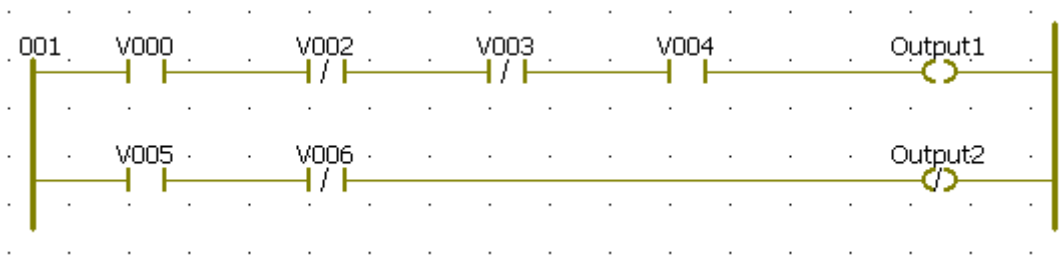
Приклад :



Action_block_FBD-мовою:



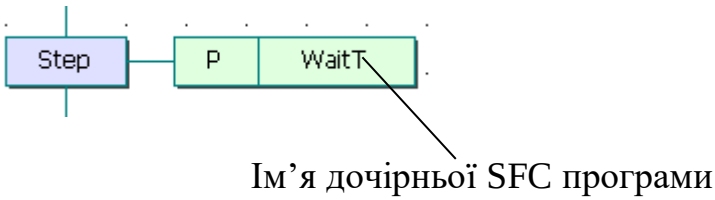
Action_block_LD-мовою:



SFC дія - це дочірня послідовність SFC, що стартує і знищується відповідно до зміни сигналу активності кроку. SFC дія може мати класифікатор N (що не зберігається), S (установка), або R (скидання). Нижче приведений синтаксис основних SFC дій:

N on a child	запустити дочірню послідовність, коли крок стає активним, і знищити її, коли крок стає пасивним.
S on a child	запустити дочірню послідовність, коли крок стає активним, і нічого не робити, коли крок стає пасивним.
R on a child	знищити дочірню послідовність, коли крок стає активним, і нічого не робити, коли крок стає пасивним.

SFC- послідовність, що визначена як дія, повинна бути дочірньою SFC-програмою поточної редагованої програми. Використання класифікаторів **S** (установка) або **R** (скидання) для SFC дій має той же самий ефект, що і використання операторів **GSTART** і **GKILL** в імпульсній дії на мові **ST**.



Ім’я дочірньої SFC програми

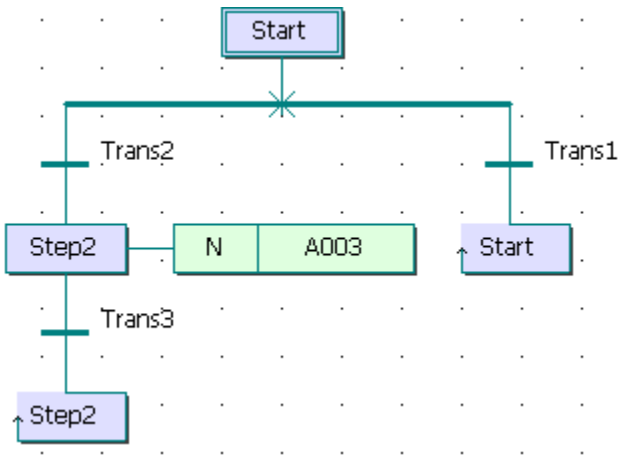
Дія WaitT:

Action (P):

GKILL(WaitT);

End_Action;

Дочірня програма WaitT:



Дія A003 :

```
(*Перемикання режимів*)
    Action (N) :
        Mode:=NewMode;
    End_Action
```

Підпрограми, функції або функціональні блоки (написані ST-, IL-, LD- або FBD- мовами), а також 'C' функції і 'C' функціональні блоки можуть бути безпосередньо викликані з блока дій SFC на основі синтаксису мови, що використовується .

Наприклад, підпрограма, що створена ST-мовою, викликається з використанням наступного синтаксису:

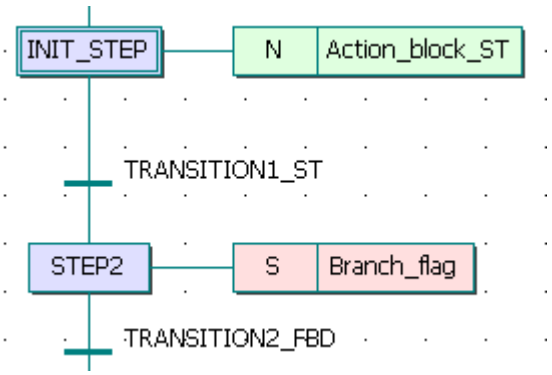
```
Action (P):                          (* або Action (N):*)
    < результат > : = < імя підпрограми >( );
End_Action.
```

Тобто

```
Action (P):
    Init: = SPinit( ) ;
    Run: = SPRun ( );
    Err: = Error ( );
End_Action
```

Ім'я кожного переходу має логічний вираз, який є умовою переходу. Якщо до переходу вираз не приєднаний, то за умовчанням умова є істиною (TRUE).

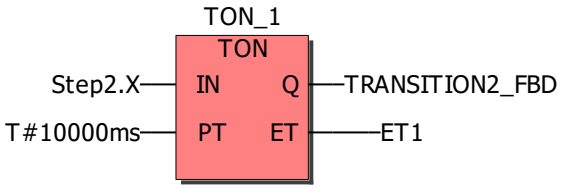
Умови переходів створюються ST-, IL-, LD- і FBD- мовами відповідно до їх синтаксису. Наприклад:



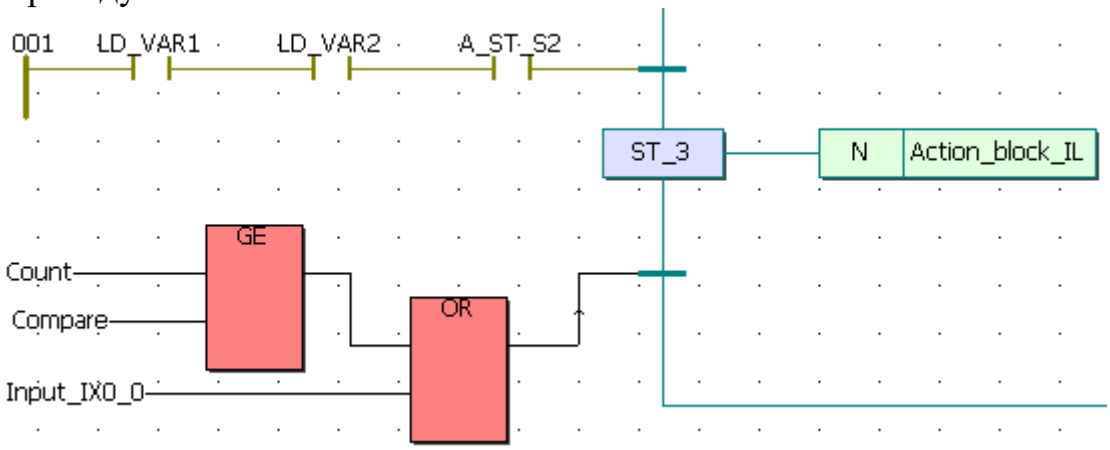
Тут умови переходів TRANSITION_ST і TRANSITION_FBD створені відповідно ST- і FBD – мовами:

```
TRANSITION_ST:
TON_SFC(IN:=INIT_STEP.X,PT:=T#10000ms);
Out := TON_SFC.ET;
Transition1_ST := TON_SFC.Q;

TRANSITION_FBD:
```



LD - діаграма, що описує умови переходів, складається з одиничної мережі з котушкою, значення якої представляє значення переходу.
Умови переходів LD - і FBD- мовами можуть бути прив'язані до точки переходу:



Опис умов, приєднаних до переходів, які зроблені IL-мовою, здійснюється згідно її синтаксису:

Action:
*(*блок команд IL-мовою*);*
End _Action.

Значення, яке містить поточний результат (IL регістр) в кінці IL послідовності, буде умовою, приєднаною до переходу:

Поточний результат = 0 або FALSE > Умова FALSE
Поточний результат <> 0 або TRUE > Умова TRUE

Наприклад:

Action:
LD Run
ANDN Error
End _Action.

Будь-яка функція , що написана FBD-, LD-, ST- або IL-мовою, а також "C"- функція можуть бути викликані для обчислення значення умови, приєднаної до переходу, згідно з таким синтаксисом на ST і IL:

Action:
< function > ( ) ;
End _Action.

Значення, що повертається функцією, повинне бути логічним і давати результуючу умову:

значення, що повертається = **FALSE** -> умова **FALSE;**
значення, що повертається = **TRUE** -> умова **TRUE** .

SFC-програма може мати розбіжності і сходження.

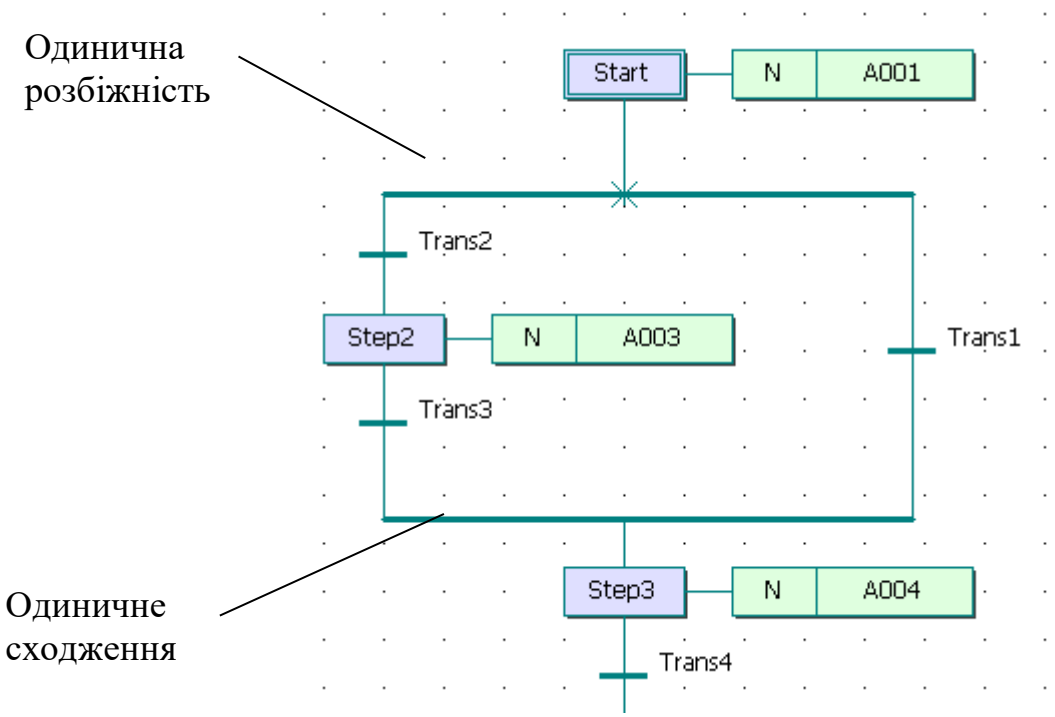
Розбіжності - це множинні сполучення від одного символу SFC (Кроку або Переходу) до багатьох інших символів.

Сходження - це множинні сполучення від більш ніж одного символу SFC до одного іншого символу. Сходження і розбіжності можуть бути одиночними або подвійними.

Одинична розбіжність (альтернативні гілки) - це множинний зв'язок від одного кроку до декількох переходів. Вона дозволяє маркеру активності пройти по одній з безлічі гілок. Перевірка альтернативних умов виконується зліва направо. Якщо вірна умова знайдена, то інші альтернативи не розглядаються. У альтернативних гілках завжди працює тільки одна, тому, коли вона закінчується, здійснюється перехід до наступного за альтернативною групою кроку.

Одиничне сходження - це множинний зв'язок від декількох переходів до одного і того ж кроку. Одиничне сходження звичайно використовується для того, щоб об'єднати декілька гілок SFC, що почалися з одиничної розбіжності.

Одиничні розбіжності і сходження позначаються одиничними горизонтальними лініями.



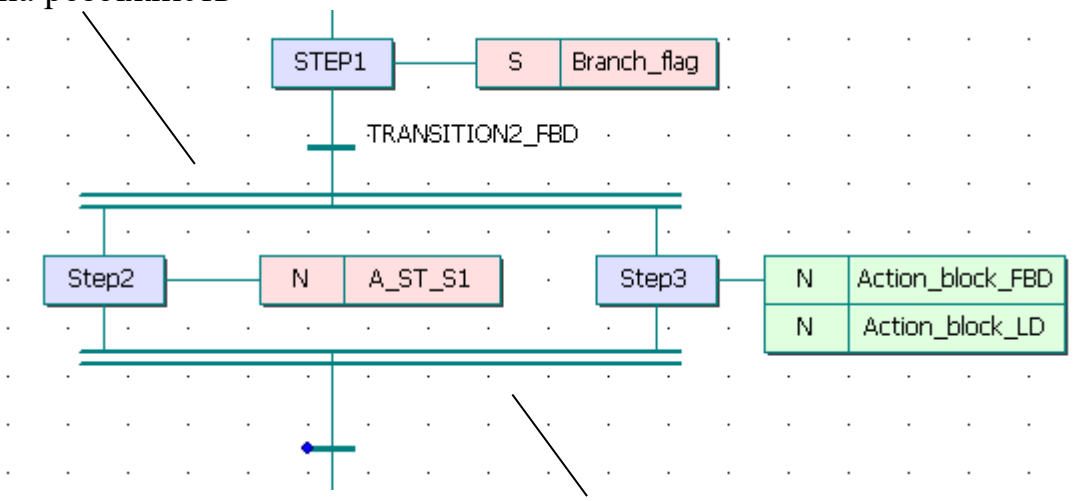
Умови, що приєднані до переходів, не є взаємовиключними. Для того, щоб програма пішла по одній гілці, треба явно визначити винятковість умови переходу.

Подвійна розбіжність - це множинний зв'язок від одного переходу до декількох кроків. Вона відповідає паралельній роботі процесу.

Подвійне сходження - це множинний зв'язок від декількох кроків до одного і того ж переходу. Подвійне сходження використовується для того, щоб об'єднати декілька SFC-гілок , що почалися в подвійній розбіжності.

Подвійні розбіжності і сходження позначаються подвійними горизонтальними лініями.

Подвійна розбіжність



Подвійне сходження

Кожна паралельна гілка починається і закінчується кроком, тобто умова входу в паралельність завжди єдина, як і єдина на всіх умова виходу.
Паралельні гілки виконуються теоретично одночасно – в одному робочому циклі, зліва направо.

При програмуванні існують п'ять динамічних поведінок SFC-мови **:**

- ***Початкова ситуація***

Початкова ситуація характеризується початковими кроками, які, за визначенням, знаходяться в активному стані на початку роботи. Принаймні, один початковий крок повинен бути в кожній SFC -програмі.

- ***Очищення переходу***

Перехід або дозволений, або заборонений. Перехід дозволений, якщо усі попередні кроки, пов'язані з відповідним символом переходу- активні, інакше, перехід заборонений. Перехід не може бути очищений, поки він не дозволений і відповідна умова переходу не є TRUE.

- ***Зміна стану активних кроків***

Очищення переходу одночасно веде до активного стану безпосередньо наступних кроків і пасивного стану безпосередньо попередніх кроків.

- ***Одночасне очищення переходів***

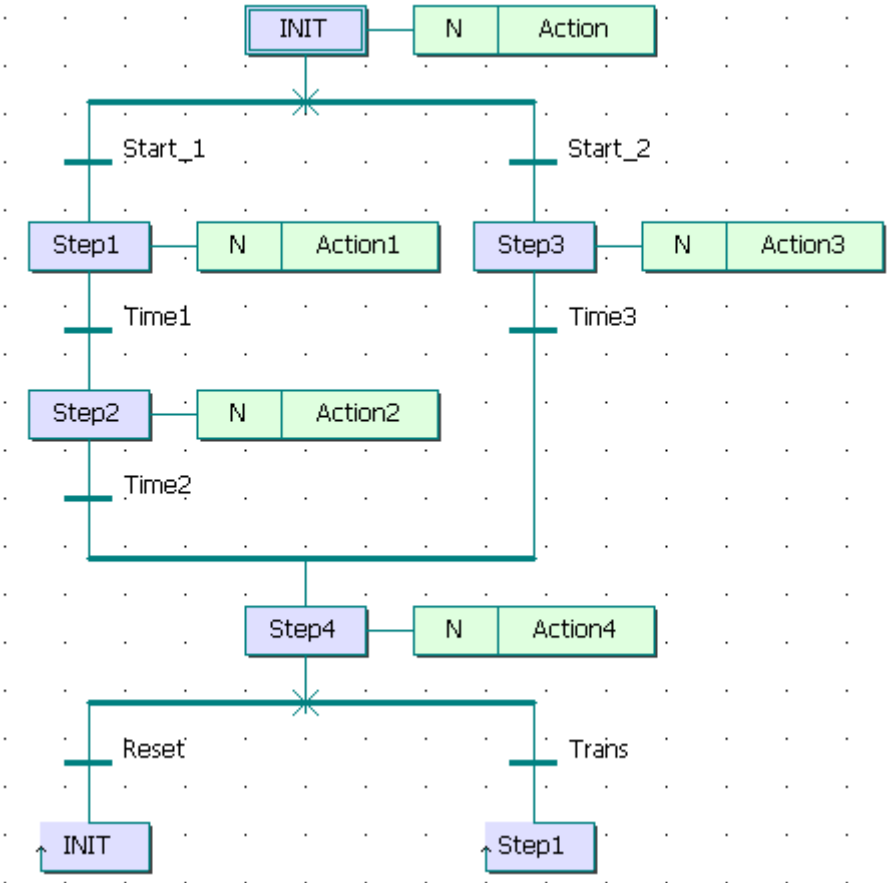
Всі переходи (всіх SFC програм), які можуть бути очищені (тобто вони дозволені і їх умова true), очищаються одночасно.

- ***Одночасна активація і деактивація кроку***

Якщо під час роботи крок одночасно активується і деактивується, пріоритет віддається активізації[4-9].

1.2.6.2. Приклад програмування SFC – мовою

Як приклад програмування SFC – мовою розробимо програму керування роботою двох двигунів згідно з алгоритмом наведеним у параграфі 1.2.2.2.
Проектний код SFC-мовою має вигляд:



Тут:

Action (ST-мовою) - motor_1:=FALSE; motor_2:=FALSE; Cycle_Count:=0;

Start_1 (ST-мовою) - Motor_Counter1(CU:=Motor_Start1(* BOOL *),
RESET:=start_1(*BOOL *),PV:=int#1(* INT *));
Start_1(* BOOL *):=Motor_Counter1.Q;
Pressed1(* INT *):=Motor_Counter1.CV;

Start_2(ST-мовою) - Motor_Counter2(CU:=Motor_Start2(* BOOL *),
RESET:=start_2(* BOOL *),PV:=int#2(* INT *));
Start_2(* BOOL *):=Motor_Counter2.Q;
Pressed2(* INT *):=Motor_Counter2.CV;

Action1(ST-мовою) - Motor_1:=TRUE; Motor_2:=FALSE;

Action3(ST-мовою) - Motor_2:=TRUE; Motor_1:=FALSE;

Time1(ST-мовою) - TON_1(IN:=step1.X(* BOOL *),PT:=t#5s(* TIME *));
time1(* BOOL *):=TON_1.Q;
A_time1(* TIME *):=TON_1.ET;

Time3(ST-мовою) - TON_3(IN:=step3.X(* BOOL *),PT:=t#10s(* TIME *));
time3(* BOOL *):=TON_3.Q;
A_time3(* TIME *):=TON_3.ET;

Action2(ST-мовою) - Motor_2:=TRUE; Motor_1:=FALSE;

Time2(ST-мовою) - TON_2(IN:=step2.X(* BOOL *),PT:=t#10s(* TIME *));

```
                    Time2(* BOOL *):=TON_2.Q;
                    A_time2(* TIME *):=TON_2.ET;
Action4(ST-мовою) - IF  Time1 or Time2 or Time3 (*EXPRESSION (must return
                    a  boolean  value)*)
                    THEN  Cycle_count:=Cycle_Count+1 (*If returned  value of
                    EXPRESSION = TRUE*)
                    (*STATEMENT*);
                    END_IF;

Reset(IL-мовою) -   LD Cycle_Count(*N*)     (*IN as ANY*)
                    GT 10
                    ST Reset

Trans(IL-мовою) -   LD Cycle_Count(*N*)     (*IN as ANY*)
                    LE 5
                    ST Trans
```