

Tổng quan về SCL

1. Giới thiệu về SCL:

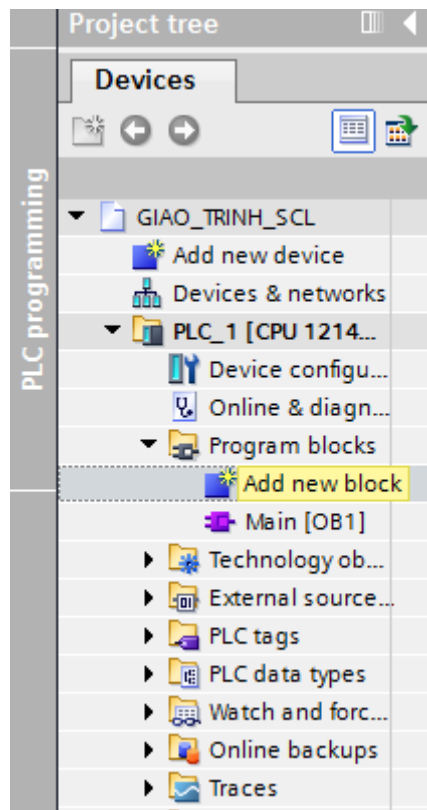
SCL (Ngôn ngữ điều khiển có cấu trúc) là ngôn ngữ lập trình cấp cao dành cho SIMATIC S7 dựa trên Pascal. Ngôn ngữ SCL thuận tiện cho việc lập trình các thuật toán phức tạp và các tác vụ liên quan đến việc quản lý dữ liệu vùng nhớ. Là ngôn ngữ cấp cao nên SCL gần gũi với tư duy của người lập trình, hỗ trợ bảo mật tốt. Bên cạnh đó SCL cũng hỗ trợ việc lập trình cấu trúc theo dạng khối nên dễ dàng kết hợp các ngôn ngữ khác khi viết chương trình (LAD, STL, FBD). Trong các đời PLC mới như S7-1200 không còn sử dụng STL mà thay thế bằng SCL

2. Trình biên dịch SCL

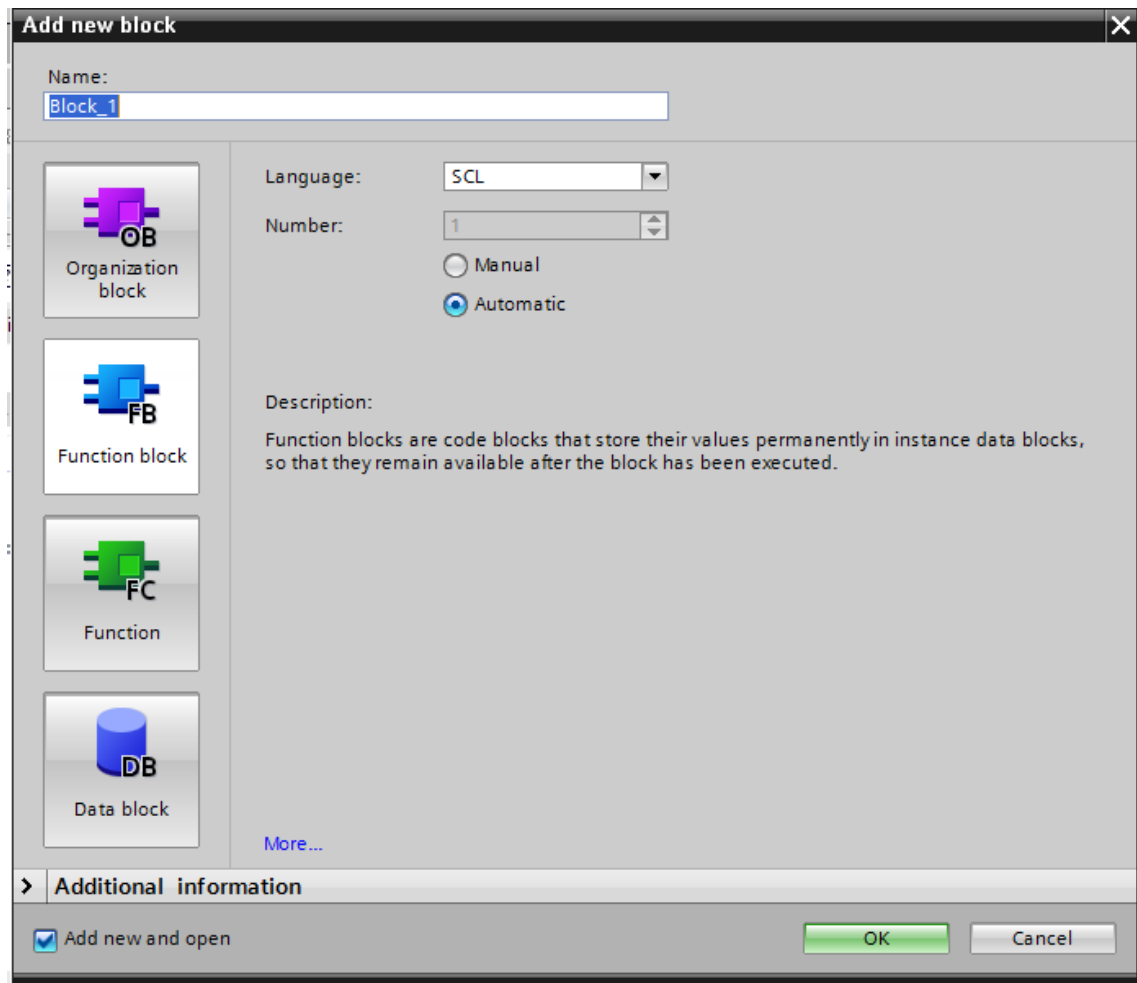
Các khối chương trình có thể viết trong Source file bao gồm Function (FC), Function Block (FB), Data Block (DB) và Organization Block (OB). Thông thường SCL không dùng để viết cho OB mà chỉ dùng viết cho FC, FB và DB. Organization Block đóng vai trò như là giao diện giữa người lập trình và các khối đã viết sẵn, người lập trình chỉ việc gọi các khối có sẵn chèn vào OB.

3. Tạo một FB, FC hoặc OB sử dụng SCL

- Click chọn Add new block



- Click chọn khối FC, FB hoặc OB muốn tạo



- Trong mục language, click chọn SCL -> nhấn OK

4. Các cấu trúc cơ bản trong SCL:

Các cấu trúc cơ bản trong SCL khá giống với Pascal, nó cũng bao gồm các câu lệnh điều kiện, rẽ nhánh, vòng lặp. Lập trình SCL thường được sử dụng để tính toán các biểu thức phức tạp hay những công việc mà thực hiện bằng Ladder tốn rất nhiều thời gian








Một số kí hiệu:

Type	Operation	Operator	Priority
Parentheses	(<i>Expression</i>)	(,)	1
Math	Power	**	2
	Sign (unary plus)	+	3
	Sign (unary minus)	-	3
	Multiplication	*	4
	Division	/	4
	Modulo	MOD	4
	Addition	+	5
	Subtraction	-	5
Comparison	Less than	<	6
	Less than or equal to	<=	6
	Greater than	>	6
	Greater than or equal to	>=	6
	Equal to	=	7
	Not equal to	<>	7
Bit logic	Negation (unary)	NOT	3
	AND logic operation	AND or &	8
	Exclusive OR logic operation	XOR	9
	OR logic operation	OR	10
Assignment	Assignment	:=	11

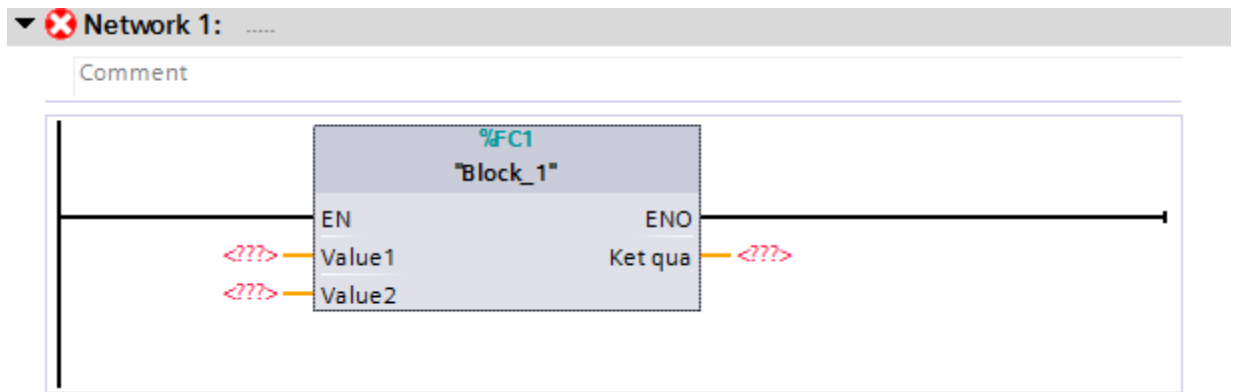
4.1. Tab khai báo biến

Ở tab này cho phép ta mô tả các biến input, output, in/output và khai báo các biến tạm sử dụng trong chương trình,...

Ví dụ: khai báo input và output như sau

Block_1				
	Name	Data type	Default value	Comment
1	 Input			
2	 Value1	Int		
3	 Value2	Int		
4	 <Add new>			
5	 Output			
6	 Ket qua	Int		

➔ Khi ta kéo thả khối FC vừa mới tạo ra chương trình chính OB1 thì khối FC sẽ có cấu trúc như sau

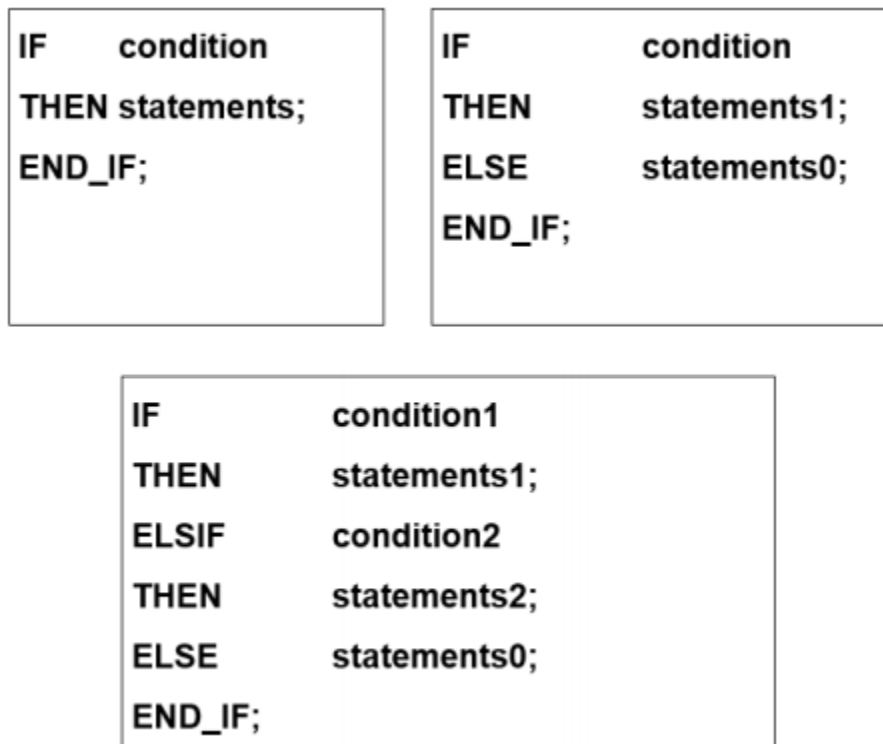


Trong <???' ta điền các giá trị, vùng nhớ thỏa mãn kiểu giá trị mà đã khai báo trong FC/FB

4.2. Các cấu trúc cơ bản

4.2.1 IF...THEN...ELSE...END_IF

Cấu trúc:



Ví dụ:

```
IF (#Value1 = 10) THEN
```

```
    # "Ket qua" := 100;
```

```
ELSE
```

```
    # "Ket qua" := 10;
```

```
END_IF;
```

➔ Lưu ý: sau mỗi câu lệnh phải có dấu “;”, nếu không có dấu này trình biên dịch sẽ báo lỗi

➔ Muốn gán một giá trị cho một biến ta dùng cú pháp

```
    Biến := Biến/Giá trị ;
```

➔ Muốn tổ hợp các phép logic

4.2.2 CASE ... OF

Cấu trúc:

```
CASE _variable_name_ OF
```

```
    1: // Statement section case 1
```

```
    ;
```

```
    2..4: // Statement section case 2 to 4
```

```
    ;
```

```
    ELSE // Statement section ELSE
```

```
    ;
```

```
END_CASE;
```

➔ Cấu trúc sẽ nhánh tương tự như IF nhưng rẽ nhánh với một giá trị cụ thể

Ví dụ:

```
CASE (#Value1) OF
```

```
    1:
```

```
#"Ket qua" := 10;
```

2..4:

```
#"Ket qua" := 20;
```

ELSE

```
#"Ket qua" := 100;
```

```
END_CASE;
```

4.2.3 Vòng lặp FOR

Cấu trúc:

```
FOR _counter_ := _start_count_ TO _end_count_ DO
```

```
    // Statement section FOR
```

```
;
```

```
END_FOR;
```

```
FOR Runtime_variable := Starting_value  
    TO End_value  
    BY Step_width  
    DO Statements;  
END_FOR;
```

- ➔ Lưu ý: biến counter thường khai báo trong mục Temp của FC/FB
- ➔ Ngoài ra còn có một cấu trúc khác, ví dụ muốn thực thi lệnh với các giá trị của counter là 0,2,4,6,8,... thì ta có thể dùng
FOR #counter := 0 TO 10 BY 2 DO

Ví dụ:

```
FOR #var := #upper TO #lower BY -2 DO  
  
END_FOR;
```

4.2.4 Vòng lặp WHILE... DO

Cấu trúc:

SCL	Description
<pre>WHILE "condition" DO Statement; Statement; ...; END_WHILE;</pre>	<p>The WHILE statement performs a series of statements until a given condition is TRUE.</p> <p>You can nest WHILE loops. The END_WHILE statement refers to the last executed WHILE instruction.</p>

4.3 Các lệnh truy xuất dữ liệu địa chỉ trực tiếp:

Trong các chương trình lập trình PLC Siemens đời cũ, ta có thể gán trực tiếp giá trị của một vùng nhớ M, input IB, output QB bằng phép gán, ví dụ:

```
Result := IB2;
```

Nhưng trong TIA Portal phiên bản mới, các lệnh này không hợp lệ, thay vào đó là các cú pháp PEEK, PEEK BOOL, POKE, POKE BOOL

- PEEK: đọc một vùng nhớ, có thể kết hợp với dạng dữ liệu như: PEEK_WORD, PEEK_DWORD

```
PEEK (AREA:= <Operand>,  
      DBNUMBER := <Operand>,  
      BYTEOFFSET:= <Operand>)
```

- PEEK BOOL: đọc một bit

```
PEEK_BOOL (AREA := <Operand>,  
            DBNUMBER := <Operand>,  
            BYTEOFFSET := <Operand>,  
            BITOFFSET := <Operand>)
```

- POKE: Xuất giá trị VALUE ra một vùng nhớ

```
POKE (AREA := <Operand>,  
      DBNUMBER := <Operand>,  
      BYTEOFFSET := <Operand>,  
      VALUE := <Operand>)
```

- POKE BOOL: Xuất giá trị VALUE(Bool) ra một bit

```
POKE_BOOL (AREA := <Operand>,  
            DBNUMBER := <Operand>,  
            BYTEOFFSET := <Operand>,  
            BITOFFSET := <Operand>,  
            VALUE := <Operand>)
```

Trong đó:

AREA: gán các giá trị tương ứng sẽ truy xuất đến vùng nhớ tương ứng

16#81	I
16#82	Q
16#83	M
16#84	DB

DBNUMBER: Số thứ tự của Data Block ta muốn truy xuất đến, nếu AREA là 16#84 tương đương với vùng nhớ DB. Nếu giá trị AREA là các giá trị khác(16#81, 16#82,16#83) thì mặc định set giá trị DBNUMBER bằng 0

BYTEOFFSET: Vị trí byte muốn đọc, xuất giá trị

BITOFFSET: Vị trí bit muốn đọc, xuất giá trị

Ví dụ:

```
%MB100 := PEEK(area:=16#81,  
dbNumber:=0, byteOffset:=#i); // when  
#i = 3
```

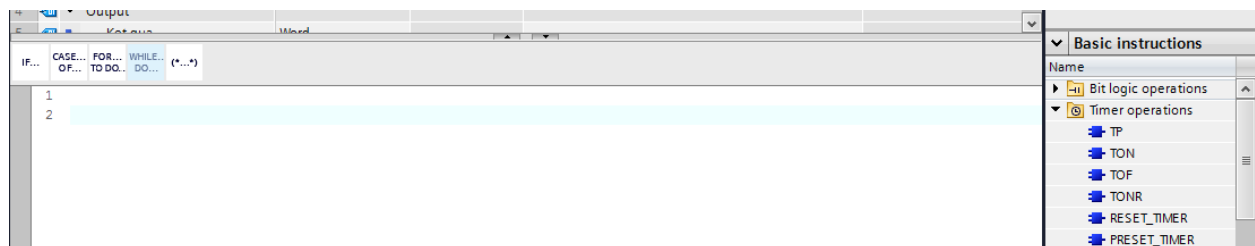
➔ Đọc giá trị IB3 và gán vào MB100

```
POKE(area:=16#82, dbNumber:=0,  
byteOffset:=3, value:="Tag_1");
```

➔ Xuất giá trị value “Tag_1” ra vùng output QB3

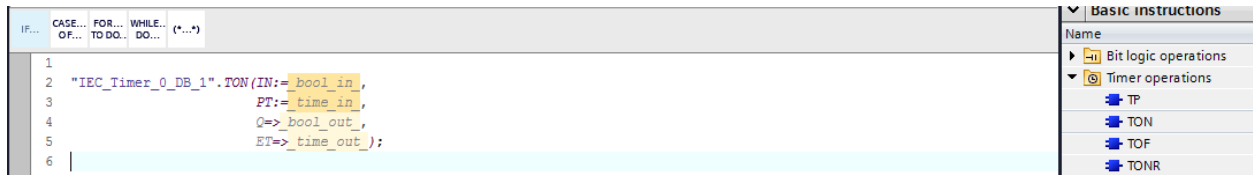
4.4 Sử dụng các khối chức năng

4.4.1 Timer



Để sử dụng Timer trong SCL, click chọn mục Basic Instructions -> Timer operations

Kéo thả timer tương ứng vào chỗ muốn sử dụng



▪ Cấu trúc của TimerOn

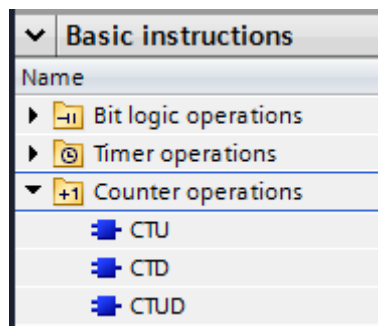
```
"IEC_Timer_0_DB_1".TON(IN:=_bool_in_,
                        PT:=_time_in_,
                        Q=>_bool_out_,
                        ET=>_time_out_);
```

Lưu ý: ta có thể không cần thiết sử dụng thành phần Q và ET nên có thể xóa nó đi, dấu phẩy là dấu ngăn cách giữa mỗi thành phần(bắt buộc), kết thúc khai báo timer bằng dấu đóng ngắt tròn và chấm phẩy “); “

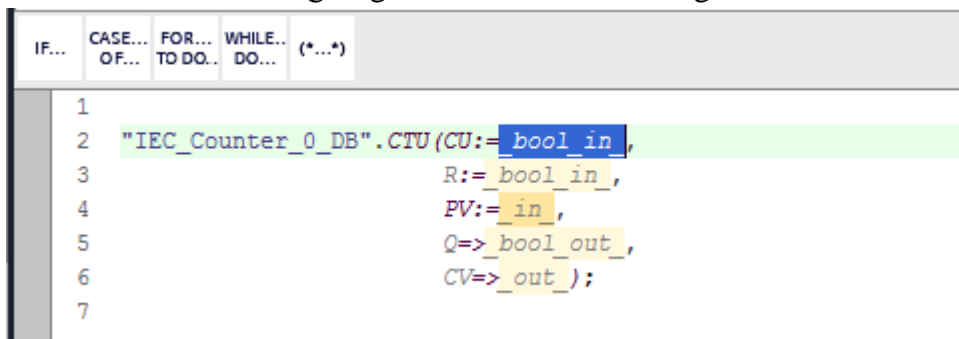
▪ Tương tự với các Timer khác

4.4.2 Counter

Để sử dụng Timer trong SCL, click chọn mục Basic Instructions -> Counter operations



Kéo thả Counter tương ứng vào chỗ muốn sử dụng



- Tương tự như Timer, các thành phần như Q, trong trường hợp không sử dụng đến ta có thể xóa đi

4.4.3 Các lệnh số học

Các lệnh số học có thể tìm kiếm trong thẻ Math Function



Click và giữ chuột trái vào hàm ta muốn sử dụng, sau đó kéo thả ra cửa sổ soạn thảo

- Ví dụ:
 - Hàm MAX, MIN có cấu trúc như sau:

MIN(IN1:=_in_, IN2:=_in_)

```
#"Ket qua" := MIN(IN1 := #Value1, IN2 := #Value2);
```

Kết quả các hàm MIN sẽ là một trong 2 giá trị Value1, Value2, giá trị nào nhỏ hơn sẽ được gán vào biến “Ket qua”

➔ Hàm MAX tương tự

- Hàm SQR

```
#"Ket qua" := SQR(#Value1);
```

Cấu trúc trong tự hàm MAX, MIN, kết quả trả về của hàm sẽ được gán vào biến “Ket qua”