ST教學

Structured text 結構化編程

此筆記是使用根據三菱電機**GX Works**及**FX-3U**所製作，其它種類的PLC不一定適用。

概論

ST全稱Structured text(結構化編程)，此語言主要是使用在PLC上的開發，相比於階梯圖這門語言也更接近如C、Pascal，所使用的指令式程式設計，因此在數學與邏輯上也會更為容易的表達，但也因此學習成本與難度也相較於其它方式較高。

由於ST為編譯式語言，因此在燒錄時需要先進行編譯。

使用時只需要在PLC所對應的平台上選擇**結構化編程**，即可看到**ST**語言。

一份檔案包含兩的部份，分別為**程式區**與**變數表**，主要的程式都是在程式區編輯，有一個不同於其它程式語言的地方，ST的變數需要在變數表內先宣告，這也代表變數的作用範圍是全域的。

資料型態

ST的資料型態和其它語言並無不同，但需注意要事先在變數表宣告。

賦值使用 := 而不是 = ，因為在ST裡面 = 為關係運算

舉例：

```
a := 50;
b := TRUE;
Y0 := TRUE;
Y1 := X0;
```

> Bit

如其本身的名稱，這個型態只會用到一個bit，也就是說可以表達TRUE或FALSE ，用於儲存布林值。

> Word[Signed]

word佔用16的bit，可以表達 -2^{16} ~ $2^{16-1} - 1$ 有號數，常用的變數型態。

> Double Word[Signed]

兩倍的word也就是說有 -2^{32} ~ 2^{32-1} - 1 的有號數。

> Time

專門用於表達時間，常與TON配合。
時間符號使用**T#並可以搭配d、h、m、s、ms**：

符號	現實單位
d	天
h	小時
m	分鐘
s	秒
ms	毫秒

特殊物件

在ST有兩個最常用的功能被封裝成物件。

> TON

```
ton1(IN := <條件式>, PT := <Time>);
```

只有在 IN 為 TRUE 的時候才會開始計時，如果在計時到一半的時候變為 FALSE 會中斷計時。

因為這個特性，可以使用下列的用法重複使用同樣的計時器：

```
ton1(IN := NOT ton1.Q, PT := <Time>);
```

PT 需要輸入Time，如：T#5s、T#50ms，或宣告一個Time在進行輸入也可以。

在讀取TON的狀態需要使用：

```
ton1.Q
```

> R_TRIG

```
rt1(_CLK := <條件式>);
```

在階梯圖的上緣觸發在ST裡被封裝成物件，因此使用的時候需要先進行宣告。

如果要讀取狀態需要使用：

```
rt1.Q
```

運算子

ST的運算子雖然沒有不同，但在使用上還是有一些與其它語言的不同之處。

算術運算

令
$a = 17$
$b = 5$

運算子	動作	動作
+	回傳 **左邊的數值** 加上 **右邊的數值**	$a + b = 17 + 5 = 22$
-	回傳 **左邊的數值** 減掉 **右邊的數值**	$a - b = 17 - 5 = 12$
*	回傳 **左邊的數值** 減掉 **右邊的數值**	$a * b = 17 * 5 = 85$
/	回傳 **左邊的數值** 除於 **右邊的數值**	$a/b = 17/5 = 3$
MOD	回傳 **左邊的數值** 除於 **右邊的數值** 的餘數	$a\ MOD\ b = 17\ MOD\ 5 = 2$

關係運算

令
$a = 20$

$b = 20$
$c = 15$

運算子	動作	舉例
>	回傳 **左邊的數值** 是否大於 **右邊的數值**	$a > b = 0$ $a > c = 1$ $c > a = 0$
<	回傳 **左邊的數值** 是否小於 **右邊的數值**	$a < b = 0$ $c < a = 1$ $a < c = 0$
>=	回傳 **左邊的數值** 是否大於或等於 **右邊的數值**	$\underline{a >= b} = 1$ $\underline{a >= c} = 1$ $\underline{c >= a} = 0$
<=	回傳 **左邊的數值** 是否小於或等於 **右邊的數值**	$\underline{a <= b} = 1$ $\underline{c <= a} = 1$ $\underline{a <= c} = 0$
=	回傳 **左邊的數值** 是否等於 **右邊的數值**	$\underline{a = b} = 1$ $\underline{a = c} = 0$
<>	回傳 **左邊的數值** 是否不等於 **右邊的數值**	$a <> b = 0$ $a <> c = 1$

邏輯運算

令
$a = 1 \;\; b = 1$
$c = 0 \;\; d = 0$

運算子	動作	舉例
NOT	回傳 **右邊** 的布林值取**反**	$NOT\ a = 0$
AND	回傳 **左右兩邊** 的布林值是否皆為 **正**	$a\ AND\ b = 1$ $a\ AND\ c = 0$
OR	回傳 **左右兩邊** 的布林值是否有其中一者為 **正**	$a\ OR\ b = 1$ $a\ OR\ c = 1$ $c\ OR\ d = 0$
XOR	回傳 **左右兩邊** 的布林值為 正 的數量是否為 **奇數**	$a\ XOR\ b = 0$ $a\ XOR\ c = 1$ $c\ XOR\ d = 0$

優先順序

運算子	順序

運算子	順序
()	1
函式	2
NOT、負號	3
* 、/、MOD	4
+、-	5
>、<、>=、<=	6
=、<>	7
AND	8
XOR	9
OR	10

流程控制

常用的迴圈在ST裡面都有，但語法不盡相同。

在PLC內如果使用過多的迴圈會導致看門狗被觸發，若發生此狀況可能要改變算法或更改看門狗的觸發時間。

> IF 條件判斷

```
IF <條件式> THEN

END_IF;
```

```
IF <條件式> THEN

ELSIF <條件式2> THEN

ELSE

END_IF;
```

> CASE 多重選擇

```
CASE <變數> OF
    <狀態>:

    <狀態2>:
```

```
    <狀態3>:
ELSE
    <其它狀態>
END_CASE;
```

> FOR 變數迴圈

使用前在要先於變數表內宣告要使用的變數

```
FOR <變數> := <初始值> TO <目標值> BY <遞增值> DO

END_FOR;
```

> WHILE 條件迴圈

```
WHILE <條件式> DO

END_WHILE;
```

> EXIT、RETURN 中斷與回傳

當迴圈需要在執行時中斷就可使用EXIT
舉例：

```
Y0 := FALSE;

cnt := 0;
WHILE TRUE DO
    cnt := cnt + 1;

    IF cnt = 50 THEN
        EXIT;
    END_IF;
END_WHILE;

Y0 := TRUE; (*Y0會在cnt計數到50的時候被執行*)
```

當使用RETURN時，程式會無條件跳轉到程式的最後一行。
舉例：

```
Y0 := FALSE;

cnt := 0;
WHILE TRUE DO
    cnt := cnt + 1;
```

```
    IF cnt = 50 THEN
        EXIT;
    END_IF;
END_WHILE;

Y0 := TRUE; (*Y0不會被執行到*)
```

進階

ST可以使用一些在階梯圖難以實現的用法，且在ST有些用法也與階梯圖不同。

陣列

陣列常用於大量的變數控制，在階梯圖這是比較難實現的，並常與FOR搭配使用。陣列也可以用在實作一些較難的資料結構。

在變數表宣告時使用 [1..?]，問號內填入要的格數。

（如果將ST編譯後在反編譯成階梯圖，會發現變數都是由D、M等來構成的，只是在ST內我們不需在特別去使用，也可以取成有意義的命名。）

索引暫存器

因為FOR迴圈的特性非常適合用來控制大量的IO，但因為我們沒有辦法直接使用變數做為後綴，因此才有了索引暫存器。

索引暫存器可以使用Z跟V，兩者在用法上是完全相同的。

使用時先將要使用的索引暫存器指定為要讀取的編號，在將索引暫存器做為IO的後綴。

舉例：

```
IF M8013 THEN
    FOR ii := 0 TO 7 BY 1 DO
        Z0 := ii;
        Y0Z0 := TRUE;
    END_FOR;
ELSE
    FOR ii := 0 TO 7 BY 1 DO
        Z0 := ii;
        Y0Z0 := FALSE;
    END_FOR;
END_IF;
```

狀態機

在階梯圖有一個經典的用法就是狀態機，在階梯圖在使用S來表達步驟，但在ST裡我們使用一個變數來當作步數，並使用CASE來作掃描。

如要用這種用法有一個重點是，在每次程式的開頭要使用M8002進行歸零，因為在PLC從 STOP 到 RUN 的過程中，變數表並不會被歸零。

舉例：

```
IF M8002 THEN
    wstep := 0;
END_IF;

CASE wstep OF
    0:
    Y1 := FALSE;
    Y0 := TRUE;
    IF X0 THEN
        wstep := 20;
    END_IF;

    20:
    Y0 := FALSE;
    Y1 := TRUE;
    IF X1 THEN
        wstep := 0;
    END_IF;
END_CASE;
```

函式

在程式設計的時候有一個準則是**事不過三***，因為如果同樣的事不停的重複的話，一但遇到要修改就會變得相當的麻煩。

函式在使用上有一個要注意的點的是，函式的回傳值並**不是用RETURN**，是將函式的名稱當成變數，直接改變就行了。

有些時候可能會用到遞迴的設計模式，但在ST裡並不支援這樣的設計，因此在設計上應使用迭代來取代。

物件

一般程式設計在使用上在較短的程式上可能沒有問題，一旦程式大了起來，就會發現有很多的問題在處理上變得麻煩，因此這個時候就需要使用物件導向的設計模式，簡化主程式並可以讓擴充性與可讀性提升。

物件在使用是只需要在變數表內宣即可直接使用，在使用之前也可以設定初始化並進行歸零。

有一點需要注意的是，ST不能使用繼承，也不能在物件內宣告函式，但在一般的PLC程式設計上這樣也足夠了。

看門狗計時器

英文稱watchdog，其功能就像其名子一樣，會不斷的檢測PLC是否沒有回應，但有時候程式並沒有卡住，而是計算量過大，這個時候我們就需要調整看門狗計時器，在FX-3U這顆PLC中的看門狗變數為D8000，單位為毫秒。

```
D8000 := 1000; (*設為一秒*)
```

若程式本身的算法效率不高，這並不是正確的處理方法，因此在修改之請先檢視程式是否有辦法達到更高的效率。