Document de formation
pour une solution complète d'automatisation
Totally Integrated Automation (T I A)

ANEXE II

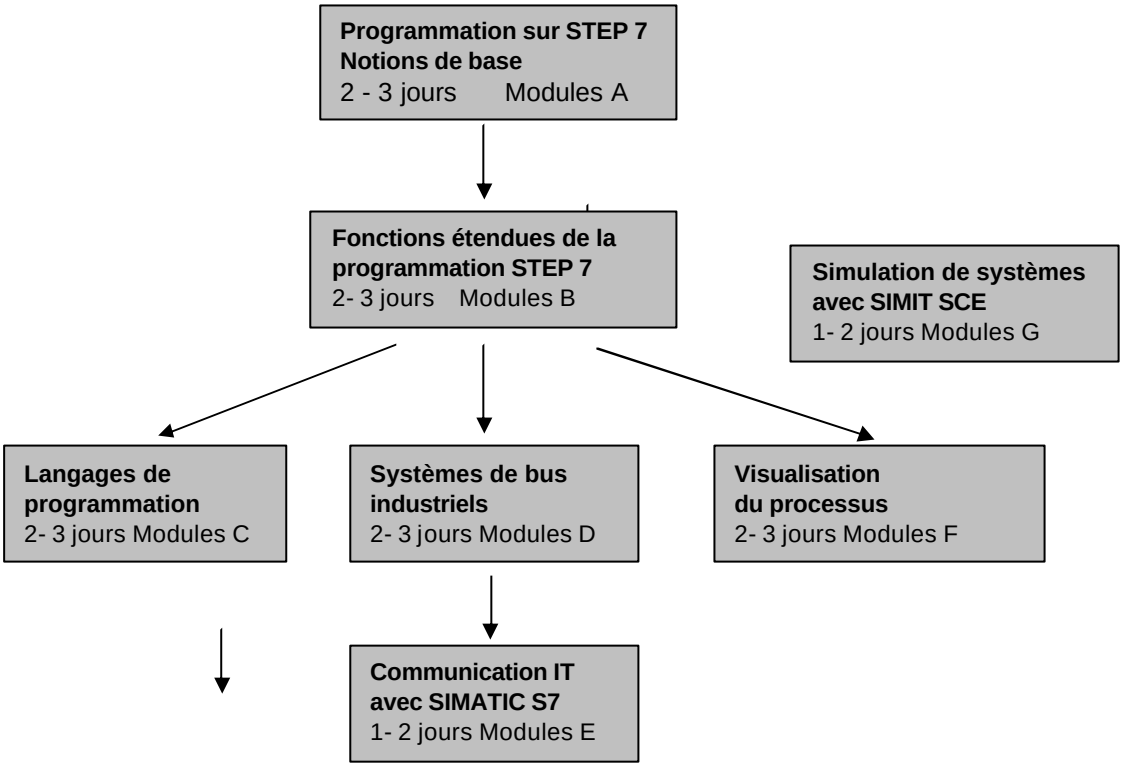
CEI 61131

Nous remercions l'entreprise Michael Dziallas Engineering et les enseignants d'écoles professionnelles ainsi que tous ceux qui ont participé à l'élaboration de ce document.

PAGE :

1. AVANT-PROPOS

L'annexe II présente les fondements théoriques pour l'étude de l'ensemble des modules.



Objectif :

Dans cette annexe, on vous présentera des informations sur la norme internationale IEC 61131.

Pré-requis :

Aucun pré-requis particulier n'est nécessaire pour l'étude de ce module, car il concerne les fondements théoriques.

2. INDICATIONS POUR LA NORME IEC 61131

La norme EN 661131-3 établit la syntaxe et la sémantique d'une suite homogène de langages de programmation pour les commandes à mémoires programmables (SPS).
Dans ce module, le lecteur sera amené à prendre connaissance des conventions de cette norme.
Ce document n'est pas une documentation exhaustive de la norme avec des explications concernant chaque symbole de circuits, mais plutôt une description générale des éléments les plus importants des langages.

- Liste d'instructions (AWL)
- Langage de bloc de fonction (FBS)
- Schéma à contacts (KOP)
- Langage de déroulement (AS)
- Texte structuré (ST)

Ce document se base sur des extraits et des interprétations de la norme DIN EN 661131 : 1993. Vous trouverez une description plus précise des définitions des langages de programmation dans la documentation de cette norme.

Indications vis-à-vis du respect de la norme

Les langages de programmation CONT et LOG correspondent aux langages « Schéma de contacts » et « langage de bloc de fonction » fixés dans la norme DIN EN 661131-3 (IEC 61131-3). Vous trouverez des informations précises dans le tableau de respect de la norme „NORM.TAB“. Le fichier se trouve dans le sous-répertoire „Siemens“ du répertoire „STEP 7“.

LIST correspond au langage de programmation « Liste d'instructions » de la norme DIN EN 661131-3 (IEC 61131-3), avec toutefois des différences essentielles au niveau d'opérations.
Vous trouverez des informations précises dans le tableau de respect de la norme „NORM.TAB“. Le fichier se trouve dans le sous-répertoire „Siemens“ du répertoire „STEP 7“.

Le langage de déroulement Graph S7 correspond au langage de programmation graph-7 ou encore Diagramme Fonctionnel en Séquence („Sequential Function Chart“) de la norme DIN EN 661131-3 (IEC 61131-3).

3. INTRODUCTION A LA NORME DIN 661131

Ce document est un guide concernant l'utilisation des automates à mémoire programmable (SPS) et des outils périphériques correspondants.
Cette norme internationale est appliquée suivant les règles de l'Union Européenne, et est désignée sous le nom de DIN EN 661131 en Allemagne, de NF EN 661131 en France et de BS EN 661131 en Angleterre.

3.1. DIN EN 661131 PARTIE 1, INFORMATIONS GENERALES

La partie 1 de la norme DIN EN 661131 (IEC 61131-1) s'applique pour les commandes à mémoire programmable. Celles-ci doivent être employées dans des dispositifs de basse tension, la tension nominative du réseau ne doit pas dépasser 1000V AC (50/60Hz) ou 1500V DC. Elles sont mises en place pour commander des machines et des processus discontinus. Dans cette partie 1, des notions et des concepts utilisés dans les autres parties de la norme sont définis.

3.2. DIN EN 661131 PARTIE 2, CAHIER DES CHARGES DES MOYENS DE PRODUCTION

Cette partie 2 de la norme DIN EN 661131 (IEC 61131-2) spécifie

- Les cahiers des charges électriques, mécaniques et fonctionnelles des automates à mémoire programmable et de leurs appareils périphériques associés ainsi que les conditions d'emploi et de fonctionnement, d'état et de transport;
- les documentations qui ont dû être livrées avec par le fabricant ;
- les méthodes de contrôle et les manières de procédé pour la vérification de la correspondance avec les exigences des commandes à mémoire et des outils périphériques associés.

3.3. DIN EN 661131 PARTIE 3, LANGAGES DE PROGRAMMATION (IEC 61131-3)

Le document traite des langages de programmation pour les commandes à mémoire programmable comme ils le sont définis dans la norme DIN EN 661131-1. La représentation sur imprimante et sur écran doit employer la fonte ISO 646. Les représentations graphiques et semi-graphiques des éléments de langage, définies dans cette partie, sont certes recevables, mais elles ne sont pas définies officiellement par cette partie.
Aucun nouveau langage de programmation n'a été défini, mais les langages les plus utilisés, LIST, CONT, FBS, AS et ST ont été simplement harmonisés.

3.4. DIN EN 661131 PARTIE 4, DIRECTIVES UTILISATEUR (EN TRAITEMENT DANS IEC)

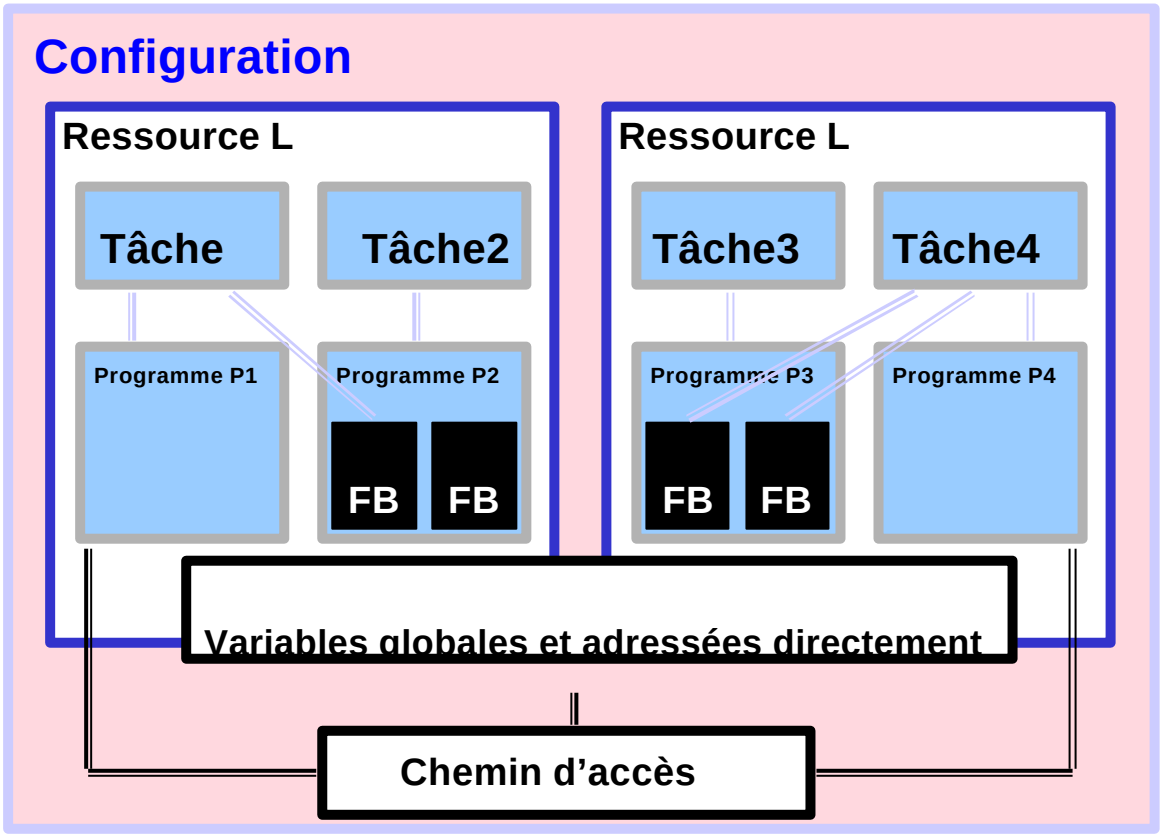
Cette partie traite des directives d'utilisation des systèmes SPS. On y trouvera des indications sur toutes les phases d'un projet, depuis l'analyse du système jusqu'à l'utilisation et l'entretien des appareils, en passant par les phases de spécifications et du choix des appareils.

3.5. DIN EN 661131 PARTIE 5, COMMUNICATION (EN TRAITEMENT DANS IEC)

Cette partie traite de la communication entre SPS de différents constructeurs et la communication d'appareils quelconques avec le SPS.
Les services de communication d'un SPS sont fixés en complément de la norme ISO/IEC 9506-1/2, en se basant sur la standardisation MAP. Les blocs de communication seront décrits pour des accès normés en lecture et en écriture.

4. DIN EN 661131 PARTIE 3, LANGAGES DE PROGRAMMATION

4.1. LE MODELE LOGICIEL IEC 61131-3



Aperçu	Un des aspects les plus importants dans un système de programmation est celui d'être apte à répartir un programme volumineux en petites sous-unités. Ces unités doivent encapsuler leurs propres données et ne doivent communiquer avec d'autres composants logiciel que par des interfaces définies de façon claire. Dans ce but là, un programme SPS a été inclus lors du développement de la norme IEC 61131-3 et un modèle logiciel abstrait y a été développé avec des niveaux de hiérarchie.
Configuration (Configuration en anglais)	Le niveau le plus élevé dans un programme utilisateur PLC est contenu dans une configuration. Une configuration correspond en général à une commande SPS. Les systèmes d'automatisation réels sont en général constitués de plusieurs SPS c'est-à-dire de plusieurs configurations IEC. C'est pourquoi une configuration IEC peut communiquer avec d'autres configurations IEC par des interfaces clairement définies, devant bien sûr être définies au préalable.
Ressource (Resource en anglais)	A l'intérieur d'une configuration, il y a une ou plusieurs ressources. Une ressource apporte le support nécessaire pour l'exécution de programmes utilisateurs SPS. Dans un système de commande, une ressource IEV correspond par ex. à un module CPU réel ou à une CPU SPS virtuelle sur un PC (SOFT-PLC).

4.2. PROGRAMME

La définition d'un programme d'après IEC 61131 est « un agencement logique de tous les éléments et de toutes les constructions de langage de programmation exigés pour le traitement de signal souhaité afin de commander une machine ou un processus d'un système SPS ».

Un programme qui respecte la norme

1. n'a le droit d'utiliser que les propriétés fixées dans cette partie pour chacun des langages,
2. ne peut pas utiliser des propriétés qui sont des extensions du langage,
3. ne peut pas dépendre d'interprétation de propriétés dépendant de l'implémentation.

Un programme respectant la norme doit conduire aux mêmes résultats sur n'importe quel système respectant la norme.
Les exceptions sont seulement constitées des programmes qui sont dépendants :

- du comportement temporel de l'exécution du programme et de son utilisation
- des propriétés dépendant de l'implémentation du programme lui-même
- de l'exécution des procédures dans le traitement des erreurs

Extrait de la déclaration du programme :

1. Les mots clé délimitant la déclaration d'un programme doivent être PROGRAMME...END-PROGRAMME.
2. Un programme peut contenir une construction VAR_ACCESS...END_VAR qui offre un moyen de fixer des variables désignées auxquelles on peut avoir accès par des services de communication. Un chemin d'accès relie de telles variables avec une entrée, une sortie et une variable interne d'un programme. Le format et l'utilisation de cette construction doivent respecter la norme IEC 61131-5.
3. Les programmes ne peuvent être instanciés qu'à l'intérieur de ressources, tandis que les blocs de fonction ne peuvent être instanciés qu'à l'intérieur des programmes.

Récapitulatif

Un programme respectant la norme

- Doit être créé dans un langage respectant la norme, sans extensions de langage
- Doit disposer d'une partie de déclaration de variables
- est délimité par les mots clés PROGRAMME... END PROGRAMME
- n'est exécutable qu'à l'intérieur de ressources
- doit rendre en sortie les mêmes résultats sur tout système respectant la norme, les exceptions étant toutefois possibles et devant être documentées

4.3. VARIABLES

Une variable sert à l'identification d'objets de données. Le contenu des variables est modifiable, il est lié à des données, des entrées, des sorties ou des cases mémoire du SPS. Les variables peuvent être déclarées comme élémentaires ou comme dérivées.

Représentation des variables à élément individuel

Une variable à élément individuel est l'élément de données de base d'un type de données élémentaires, par ex. une entrée ou une sortie. La représentation directe de variables à élément individuel est réalisée par les signes suivants :

- un signe de pourcentage
- un préfixe pour l'endroit mémoire
- un préfixe pour la taille
- un ou plusieurs nombres entiers non signés

Exemples :

%QX75 et %Q75	Bit de sortie 75
%IW215	Zone mémoire mot d'entrée 15

Propriétés des préfixes

N°	Préfixe	Signification
1	I	Zone mémoire entrée
2	Q	Zone mémoire sortie
3	M	Zone mémoire interne
4	X	Taille de bit (individuel)
5	aucun	Taille de bit (individuel)
6	B	Taille d'octet (8 bits)
7	W	Taille de mot (16 bits)
8	D	Taille de mot double (32 bits)
9	L	Taille de mot long (64 bits)

Remarque : Dans le cas où le type de données d'une variable directement adressée n'est pas déclaré ailleurs, il doit avoir une taille d'un « bit unique » de type booléen. Les organisations de normalisation nationales peuvent éditer des tables d'interprétations des préfixes.

Représentation de variables multi élément

La variable multi-élément se compose de tableaux (Arrays en angl.) et de structures (Structures en angl.).
Un tableau est une collection d'éléments de données du même type, qui sont accessibles par un ou plusieurs indices de tableau. Les indices sont entre crochets, et séparés par des virgules.

Exemple :

```
OUTARY[%MB6,SYM] := INARY[0] + INARY[7] - INARY[%MB6] * %IW62;
```

Une variable structurée est une variable dont le type est une structure de données. Cette structure se compose d'une collection d'éléments marqueurs. Un élément d'une variable structurée doit être représenté par un ou plusieurs marqueurs ou par des champs d'accès. Les marqueurs sont séparés par des points, le premier marqueur représentant le nom de l'élément structuré. Les marqueurs suivants représentent l'accès à un certain élément à l'intérieur de la structure de données.

Exemple :

MODULE_5_CONFIG.SIGNAL_TYP := SINGLE_ENDED;

Initialisation de variables

A l'initialisation, une variable peut accepter différentes valeurs :

- la valeur que la variable avait lorsque l'élément de configuration a été arrêté (mémoire tampon)
- une valeur initiale spécifique à l'utilisateur
- une valeur de démarrage prédéfinie pour laquelle le type de démarrage correspond à la variable

L'utilisateur peut choisir si la variable doit être mise en mémoire tampon en employant le signe de convention RETAIN. La valeur de démarrage doit être déterminée au démarrage suivant les règles suivantes :

- lors d'un démarrage à chaud, les variables mémoire tampon (rémanentes) doivent prendre les valeurs enregistrées
- lors d'un démarrage à froid, les variables doivent prendre les valeurs précédentes ou les valeurs initiales spécifiques du type de données correspondant
- les variables non mémoire tampon doivent être initialisées avec une valeur de démarrage prédéfinie ou avec une valeur initiale spécifique au type de données

Déclaration

Chaque déclaration de type d'une unité d'organisation d'un programme SPS doit contenir à son début au moins une partie déclaration, qui fixe les types des variables devant être utilisées dans cette unité d'organisation. La partie déclaration doit posséder le texte des mots clés VAR, VAR_INPUT ou VAR_OUTPUT. La déclaration est terminée par VAR_END.

Exemple :

Stance du programme tâche 2

```
VAR
    B1 AT %IX0.0        :BOOL;  (*  Détecteur de proximité B1   *)
    B2 AT %IX0.1        :BOOL;  (*  Détecteur de proximité B2   *)
    B3 AT %IX0.2        :BOOL;  (*  Détecteur de proximité B3   *)
    B4 AT %IX0.3        :BOOL;  (*  Détecteur de proximité B4   *)
    Y1 AT %QX4.0        :BOOL;  (*  Sortir vérin 1.0  *)
END_VAR
```

4.4. LA LISTE D'INSTRUCTIONS (LIST)

La liste d'instructions est composée d'une suite d'instructions (de commandes). Les conventions suivantes sont à respecter :

- Chaque instruction doit commencer à une nouvelle ligne
- Chaque instruction contient un opérateur avec des modificateurs supplémentaires (par ex. I 0.0 pour le bit d'entrée 0 dans l'octet d'entrée 0)
- On sépare plusieurs opérandes par des virgules (par ex. cal FB1, DB2)
- L'instruction peut suivre une étiquette, qui est suivie de deux points.
- Un commentaire est le dernier élément d'une ligne
- Des lignes vides peuvent être utilisées

Operateurs, Modificateur et opérandes

La valeur de l'expression allant être calculée va être remplacée par la valeur effectivement calculée.
Résultat := Résultat OPERATEUR Opérande

Exemple : Résultat := Résultat AND %IX1

Le modificateur „N" correspond à la négation booléenne de l'opérande.
L'instruction ANDN%IX2 signifie :

Résultat := Résultat AND NOT %IX2

Le modificateur parenthèse gauche „(" indique que l'évaluation de l'opérateur est repoussée jusqu'à ce que la parenthèse droite „)" apparaisse.
La suite d'instructions

```
AND( %IX1
OR %IX2
)
```

Signifie Résultat:= Résultat AND (%IX1 OR %IX2)

Le modificateur „C" indique que l'instruction correspondante n'a le droit d'être accomplie que si la valeur du résultat déjà évalué est un 1 booléen. Si le modificateur „N" est relié avec l'opérateur, le résultat doit être un 0 booléen.

Opérandes de la liste d'instructions

N°	Opérateur	Modificateur	Opérande	Signification
1	LD	N	Rem. 1	Met l'opérande à la valeur du résultat actuel
2	ST	N	Rem. 1	Sauve le résultat actuel à l'adresse de l'opérande
3	S	Rem. 2	**BOOL**	Met l'opérateur booléen à 1
	R	Rem. 2	BOOL	Remet l'opérateur booléen à 0
4	AND	N, (	BOOL	ET booléen
5	&	N, (	BOOL	ET booléen
6	OR	N, (	BOOL	OU booléen
7	XOR	N, (	BOOL	OU exclusif booléen
8	ADD	(	Rem. 1	Addition
9	SUB	(	Rem. 1	Soustraction
10	MUL	(	Rem. 1	Multiplication
11	DIV	(	Rem. 1	Division
12	GT	(	Rem. 1	Comparaison : >
13	GE	(	Rem. 1	Comparaison : >=
14	EQ	(	Rem. 1	Comparaison : =
15	NE	(	Rem. 1	Comparaison : <>
16	LE	(	Rem. 1	Comparaison : <=
17	LT	(	Rem. 1	Comparaison : <
18	JMP	C,N	**INDEX**	Saut à l'étiquette
19	CAL	C,N	NOM	Appel du bloc de fonction (Rem. 3)
20	RET	C,N		Retour de fonction ou de bloc de fonction
21	)			Traitement des opérations entre parenthèses (Rem. 4)

Remarque 1 : Les opérateurs doivent être soient surchargés soit déclarés avec un type. Le résultat actuel et l'opérande doivent avoir le même type.

Remarque 2 : Les opérations sont seulement exécutées si la valeur du résultat actuel est un 1 booléen.

Remarque 4 : Le nom d'une fonction est suivie d'une liste d'arguments entre parenthèses.

Propriétés de l'appel des blocs de fonction

N°	Description/Exemple
1	CAL avec liste des paramètres d'entrée : CAL C10(CU:=%IX10, PV:=15)
2	CAL avec chargement/enregistrement des paramètres d'entrée : LD 15 ST C10.PV LD %IX10 ST C10.CU CAL C10
3	Usage de l'opérateur d'entrée : LD 15 PV C10 LD %IX10 CU C10

Remarque : Une déclaration telle que VAR C10: CTU; END_VAR sera reprise dans les exemples.

Représentation de la liste d'instructions en STEP 7

FC2 : Bloc exemple
Réseau 1 : boîte logique ET en schéma à contacts

```
A     I     0.0          "B1"          --  Détecteur de proximité B1
A     I     0.1          "B2"          --  Détecteur de proximité B2
=     Q     4.0          "Y1"          --  Sortir vérin 1
```

Réseau 2 : Boîte logique OU en schéma à contacts

```
O     I     0.2          "B3"          --  Détecteur de proximité B3
O     I     0.3          "B4"          --  Détecteur de proximité B4
=     Q     4.0          "Y1"          --  Sortir vérin 1
```

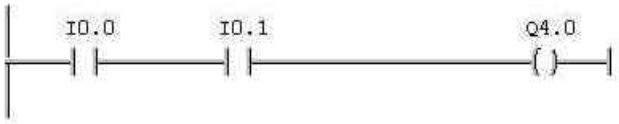
4.5. SCHEMA A CONTACTS (CONT)

Définition du réseau

- Un réseau CONT est délimité à gauche et à droite par une barre conductrice verticale.
- Un élément de liaison est identifié par une ligne horizontale.
- L'état de l'élément de liaison est désigné comme „MARCHE" et. „ARRET" conformément au flux de courant et correspond aux valeurs booléennes 0 et 1.
- L'état de la ligne de liaison gauche est toujours sur „MARCHE".
- L'état de la ligne de liaison droite est indéfinie
- L'élément de liaison horizontal transmet l'état du côté gauche au côté droit de l'élément
- L'élément de liaison vertical se croise avec un ou plusieurs éléments horizontaux. L'état de la liaison verticale doit être à „ARRET", si toutes les liaisons horizontales à gauche de l'élément sont sur „ARRET". Son état doit être „MARCHE", si une ou plusieurs des liaisons horizontales à gauche de l'élément sont sur „MARCHE".
- L'état de la liaison verticale doit être copié sur toutes les liaisons horizontales du côté droit de l'élément. Il n'est pas autorisé de faire une copie de l'état sur le côté gauche.

Représentation du schéma à contacts en STEP 7

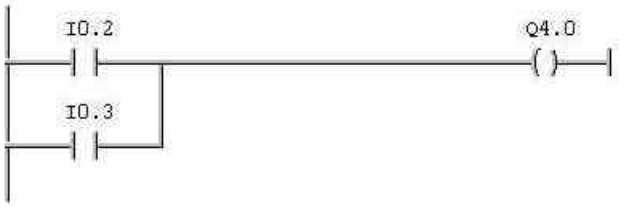
FC2 : Bloc exemple
Réseau 1 : boîte logique ET en schéma à contacts



Information mnémonique :

I0.0	B1	Détecteur de proximité B1, Détecteur de proximité B2, Sortir vérin 1
I0.1	B2	
Q4.0	Y1	

Réseau 2 : Boîte logique OU en schéma à contacts



Information mnémonique :

I0.2	B3	Détecteur de proximité B3, Détecteur de proximité B4, Sortir vérin 1
I0.3	B4	
Q4.0	Y1	

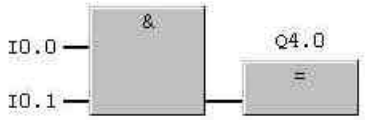
4.6. LANGAGE BLOC DE FONCTION (FBS)

Definition du réseau

- Les éléments du langage FBS doivent être délimités par des lignes de flux de signaux
- Les sorties des blocs de fonction ne doivent pas être reliées entre elles
- Le traitement d'un réseau doit être entièrement terminé, avant de pouvoir traiter le réseau suivant, si le réseau suivant utilise des sorties du réseau précédent.

Représentation du FBS en STEP 7

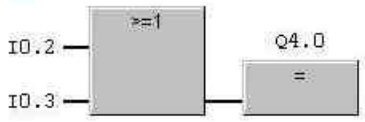
FC2 : Bloc exemple
Réseau 1 : boîte logique ET en schéma à contacts



Information mnémonique :

I0.0	B1	Détecteur de proximité B1, Détecteur de proximité B2,
I0.1	B2	Sortir vérin 1
Q4.0	Y1	

Réseau 2 : Boîte logique OU en schéma à contacts



Information mnémonique :

I0.2	B3	Détecteur de proximité B3, Détecteur de proximité B4,
I0.3	B4	Sortir vérin 1
Q4.0	Y1	

4.7. LE LANGAGE DE CHAINE (AS)

Les éléments d'une langue de chaîne offrent une ressource pour un plan d'une unité d'organisation programme décliné en une série de transitions elles-mêmes reliées à des étapes. A chaque étape appartient un certain nombre d'actions et chaque transition contient une condition de transition. Le programme créé est enregistré dans des blocs de fonction avec une instance de blocs de fonction.

Etape

Une étape est représentée graphiquement par un bloc. Le bloc contient le nom de l'étape sous forme d'identifiant ou de texte (ex. Etape 1 ou S1). Les liaisons directionnelles sont représentées par des lignes verticales reliées par le haut à une étape. Les liaisons peuvent être aussi représentées sous la forme de transitions.
A une étape correspondent un certain nombre d'actions différentes comme par exemple l'initialisation / la réinitialisation d'entrées ou de sorties.
Un marqueur d'étape présente l'état de l'étape, actif ou inactif, il peut être représenté par une variable booléenne. La variable présente un '1' booléen lorsque l'étape est active et un '0' booléen lorsque celle-ci n'est pas active.
La durée „Time" commence à la valeur 0 lors de l'activation d'une étape et s'arrête lors de son inactivation. La valeur temporelle atteinte reste alors inchangée jusqu'à la prochaine activation.
Chaque commande de déroulement possède une unité organisationnelle de programme. Dans celle-ci se trouvent les valeurs initiales des variables internes et de sortie. De plus, il y a une définition des étapes initiales actives, et chaque réseau AS doit avoir exactement une étape initiale.
L'initialisation du système est définie par une durée initiale prédéfinie pour les étapes. L'activation de l'étape suivante doit avoir lieu en un temps prédéfini. Dans le cas contraire, le système passe dans un état de dérangement. L'initialisation du système peut être aussi définie par un état initial pour les étapes habituelles dont l'activation s'effectue par la validation de la condition de poursuite.

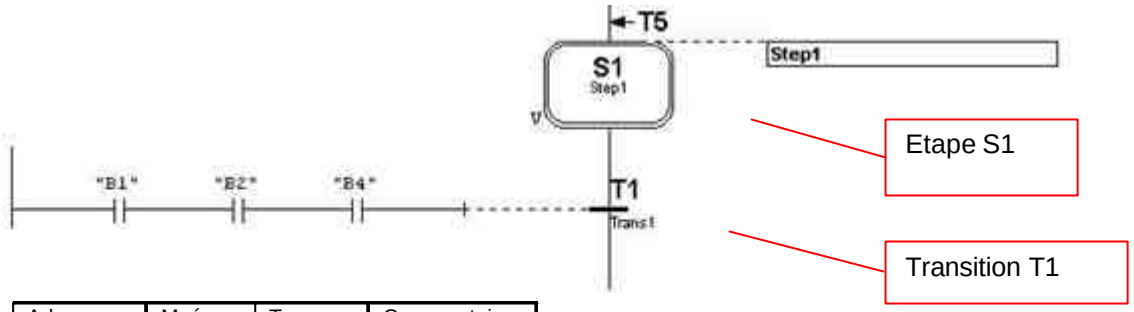
Transitions

La transition est représentée par un trait horizontal et se trouve entre 2 étapes elles-mêmes reliées par un trait vertical. Une transition est une condition de poursuite de la chaîne, condition composée elle-même de plusieurs conditions à remplir avant que l'étape suivante ne puisse être activée. Si aucune condition n'est définie dans une transition, alors celle-ci est considérée comme valide.
La condition de transition peut être définie dans les langages CONT, ST, LIST ou FBS.

Actions

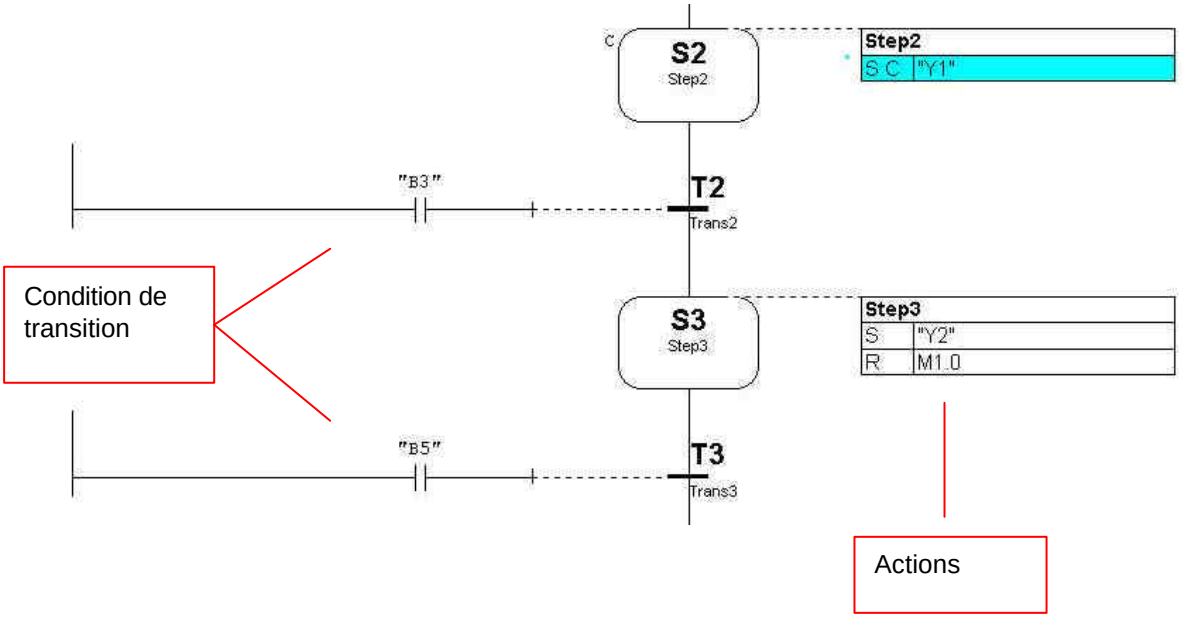
Une étape contient de 0 à plusieurs actions. Une action peut être une variable booléenne ou également une suite d'instructions dans un langage de programmation normé. La fonction de l'action doit être facilement reconnaissable par des identificateurs de commande.

Représentation d'un AS en GRAPH S7



Adresse	Mnémonique	Type	Commentaire
I0.0	B1	BOOL	Détecteur de proximité B1
II0.1	B2	BOOL	Détecteur de proximité B2
I0.3	B4	BOOL	Détecteur de proximité B4
S001.T	Step1.T	DWORD	

Extrait d'un graph-7



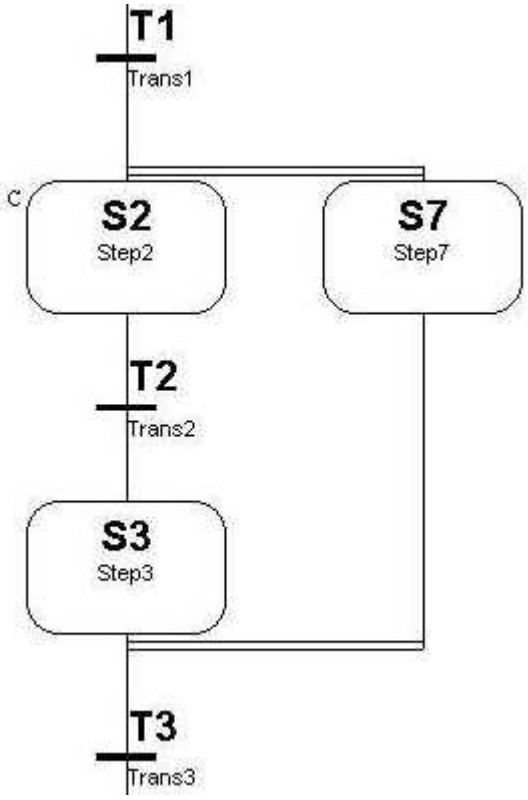
Représentation des signes de convention pour les actions

N°.	Signes de convention	Explication
1	Aucun	Non enregistré (pas de signes)
2	N	Non enregistré
3	R	Réinitialisation prioritaire
4	S	Initialisation (enregistré)
5	L	Limité dans le temps
6	D	Retardé
7	P	Impulsion (front)
8	SD	Enregistré et retardé
9	DS	Retardé et enregistré
10	SL	Enregistré et limité dans le temps

Règle de déroulement

- L'état de démarrage d'un réseau AS est caractérisé par une étape initiale, dans laquelle l'iniT I A lisation du programme est active.
- Une étape est débloquée quand l'étape précédente et les conditions de transition nécessaires sont remplies.
- Le déclenchement d'une transition désactive toutes les étapes précédentes
- Deux étapes ne sont jamais reliées directement, elles sont toujours séparées de transistions
- Une activation de plusieurs étapes passe par un branchement OU. Elle est signalisée par une ligne double horizontale à partir de laquelle s'exécutent simultanément les chaînes.

Représentation d'un branchement ET



4.8. TEXTE STRUCTURE (ST)

4.8.1. EXPRESSIONS

Les expressions sont composées d'opérateurs et d'opérandes. Les opérateurs du langage ST sont représentés dans le tableau 5.

N°	Opération	Symbole	Hiérarchie
1	Parenthèses	(Expression)	La plus haute
2	Evaluation de fonction Exemples :	Qualificateur (Liste d'arguments) LN(A), MAX(X,Y) etc..	
3	Puissance	**	
4 5	Négation Complément	- NOT	
6 7 8	Multiplication Division Modulo	* / MOD	
9 10	Addition Soustraction	+ -	
11	Comparaison	<,>,<=,>=	
12 13	Egalité Différence	= <>	
14 15	ET booléen ET booléen	& AND	
16	OU exclusif booléen	XOR	
17	OU booléen	OR	La plus basse

L'évaluation d'une expression consiste en l'application des opérateurs sur les opérandes, dans un ordre bien précis qui est défini par les priorités entre les opérateurs. L'opérateur à la priorité la plus haute dans une expression est utilisé en premier, suivi de l'opérateur à la deuxième priorité la plus haute… etc jusqu'à ce que l'évaluation soit achevée. Les opérateurs de même priorité sont utilisés dans l'ordre de l'expression, de la droite vers la gauche.
Par exemple, si A,B,C et D ont le type INT (Entier (Integer)) et respectivement les valeurs 1,2,3 et 4 alors

A+B-C*ABS(D) calculé comme –9 ((1+2)-(3*4)= -9)

et

(A+B-C)*ABS(D) donne 0. **((1+2-3)*4=0)**

Si un opérateur a deux opérandes, l'opérande de gauche doit être tout d'abord évalué. Par exemple, dans l'expression

SIN(A)*COS(B)

l'expression SIN(A) est d'abord évaluée, puis COS(B), et enfin le produit des deux est effectué.

Les expressions booléennes ont seulement besoin d'être évaluées si cela a une incidence sur la suite du résultat, comme nous allons le voir dans l'exemple suivant.

Si par exemple on a A<=B, et qu'on évalue tout d'abord (A>B), cela suffit pour dire que le résultat de l'expression (A>B)&(C<D) vaut un 0 booléen, sans avoir même à évaluer (C<D).

Les fonctions doivent être appelées comme éléments d'expressions constitués du nom de la fonction suivi d'une liste d'arguments entre parenthèses.

4.8.2. INSTRUCTIONS

Les instructions du langage ST sont récapitulées dans le tableau 6. Les instructions doivent être terminées par un point virgule.

N°	Type d'instruction/Référence	Exemples
1	Attribution	A:=B; CV:=CV+1; C:=SIN(X);
2	Appel de bloc de fonction et usage de la sortie FB	CMD_TMR(IN:=%IX5, PT:=T#300ms); A:=CMD_TMR.Q;
3	RETURN	RETURN;
4	IF	D:=B*B-4*A*C; IF D< 0.0 THEN NROOTS:=0; ELSEIF D=0.0 THEN NROOTS:= 1; X1:=-B/(2.0*A); ELSE NROOTS:= 2; X1:=(-B+SQRT(D))/(2.0*A); X1:=(-B-SQRT(D))/(2.0*A); END_IF;
5	CASE	TW:=BCD_TO_INT(THUMBWHEEL); TW_ERROR:=0; CASE TW OF 1,5:DISPLAY:=OVEN_TEMP; 2: DISPLAY:=MOTOR_SPEED; 3: DISPLAY:=GROSS_TARE; 4,6..10:DISPLAY:=STATUS(TW-4); ELSE DISPLAY:=0; TW_ERROR:=1; END_CASE; QW100:=INT_TO_BCD(DISPLAY);

6	FOR	J:=101;
		FOR i:=1TO 100BY 2 DO
		IF WORDS[I]='KEY' THEN
		J:=I;
		EXIT;
		END_IF
		END_FOR;

7	WHILE	J:=1; WHILE J<= 100 & WORDS[J] <> ,KEY' DO J:= J+2; END_WHILE;
8	Répétition REPEAT	J:= -1; REPEAT J:= J+2; UNTIL J= 101 OR WORDS[J]= ,KEY' END_REPEAT;
9	Sortie EXIT	EXIT;
10	Instruction vide	;

.

4.8.2. ATTRIBUTIONS

L'attribution remplace la valeur actuelle d'une variable unique ou multi élément par le résultat de l'évaluation d'une expression. Une instruction doit être constituée à gauche d'une variable suivie de l'opérateur d'attribution „:=", lui-même suivi de l'expression à évaluer. Par exemple l'instruction A:=B; est utilisé pour remplacer la valeur de la variable A par la valeur stockée actuellement dans B si les deux variables ont effectivement le type INT.

L'attribution doit être aussi utilisée pour récupérer la valeur renvoyée par une fonction. Il est possible de réaliser cela en plaçant le nom de la fonction à gauche de l'opérateur d'attribution dans le prototype de la déclaration de fonction. La valeur, retournée par la fonction, doit être le résultat de la dernière évaluation effectuée par cette fonction. C'est une erreur de retourner une valeur différente de 0 pour l'évaluation d'une fonction avec la sortie « ENO », si au moins une telle attribution n'a pas été effectuée.

4.8.3. INSTRUCTIONS DE COMMANDES POUR LES FONCTIONS ET LES BLOCS DE FONCTIONS

Les instructions de commandes pour les fonctions et les blocs de fonctions se composent de mécanismes pour appeler des blocs de fonctions et pour commander les sauts retours aux unités appelantes, avant d'atteindre la fin physique d'une fonction ou d'un bloc de fonctions.
Les blocs de fonction doivent être appelés par une instruction, constituée du nom du bloc de fonction, suivie d'une liste (les arguments) entre parenthèses de valeurs à attribuer et/ou de paramètres d'entrée, comme défini dans le tableau 6. Il n'est pas obligatoire que tous les paramètres d'entrée soient modifiés (attribués) à tous les appels de bloc de fonctions. Dans le cas où un certain paramètre n'est pas attribué lors de l'appel d'un bloc de fonctions, la valeur précédemment attribuée est utilisée (ou alors la valeur initiale si aucune attribuation n'a été réalisée précédemment). L'instruction RETURN (Saut retour) provoque une interruption prématurée de la fonction ou du bloc de fonctions.

4.8.4. INSTRUCTIONS DE SELECTION

Les instructions de sélection regroupent les instructions IF et CASE. Une instruction de sélection choisit son instruction ou son groupe d'instructions suivant une condition associée. Des exemples d'instructions de sélection sont donnés dans le tableau 6.
L'instruction IF impose qu'un groupe d'instructions soit seulement exécuté si l'expression booléenne correspondante délivre la valeur 1 (vrai). Si cette condition n'est pas remplie, soit aucune instruction n'est exécutée, soit le groupe d'instructions suivant le mot clé ELSE (ou encore le mot clé ELSIF, dans le cas où la condition booléenne correspondante est vraie) est exécutée.

L'instruction CASE est constituée d'une expression qui doit être évaluée sous la forme d'une variable de type INT, et d'une liste de groupes d'instructions, chacun étant assorti d'une étiquette. Cette étiquette est composée d'un ou plusieurs entiers voire d'intervalle d'entiers. Cela impose que le premier groupe d'instructions soit seulement exécuté s'il suit le mot clé ELSE. Sinon, aucune autre suite d'instructions ne peut être exécutée.

4.8.5. INSTRUCTIONS DE REPETITION

La répétition d'instructions consiste en la répétition d'un groupe d'instructions. L'instruction FOR est utilisée dans le cas où le nombre d'itérations est connu à l'avance, sinon on utilise WHILE et REPEAT.

L'instruction EXIT doit être utilisée pour arrêter les boucles, avant même que ne soit remplie la condition de terminaison.
Si l'instruction EXIT se trouve à l'intérieur d'une boucle imbriquée, c'est la boucle la plus interne qui se trouve interrompue. L'exécution passe alors directement à la première instruction suivant la fin de la boucle interrompue.

L'instruction FOR indique qu'une série d'instructions doit être répétée jusqu'au mot clé END_FOR. Une variable de commande (l'itérateur) est incrémentée/décrémentée à chaque itération. La variable de commande, la valeur initiale et la valeur finale doivent être des expressions du même type de données entier et ne doivent pas être changées par une instruction se trouvant à l'intérieur de la boucle. L'instruction FOR incrémente ou décrémente l'itérateur, depuis sa valeur initiale jusqu'à sa valeur finale, évaluée par une expression. L'incrémentation/décrémentation par défaut vaut 1.
Le contrôle de condition de fin est effectué au début de chaque itération. Ainsi, la suite d'instructions n'est pas du tout exécutée si la valeur initiale dépasse la valeur finale. La valeur de la variable de contrôle après la sortie de la boucle FOR dépend de l'implémentation.

Le groupe d'instructions compris entre un WHILE et un END_WHILE est répété tant que la condition booléenne de sortie n'est pas remplie. Si elle est remplie dès le début, le groupe d'instructions n'est jamais exécuté.

Le groupe d'instruction au niveau d'un REPEAT est exécuté au moins une fois. Il est ensuite exécuté jusqu'au mot clé UNTIL, et répété tant que la condition booléenne de sortie n'est pas remplie.

Les instructions WHILE et REPEAT ne doivent pas être utilisées pour établir une synchronisation entre des processus, par exemple comme une « boucle d'attente » à condition finale externe. Dans ce cas-là, il faut utiliser les éléments AS spécialement dédiés.
C'est une erreur d'utiliser les instructions WHILE ou REPEAT dans un algorithme lorsque les conditions de sortie de la boucle (soit par validation de la condition d'arrêt, soit par l'utilisation du mot clé EXIT) ne peuvent pas être systématiquement garanties.

4.9. DOCUMENTATION

On devra tenir compte des points suivants pour la documentation d'un projet :

- Description de la configuration matérielle (Hardware) à l'aide de spécifications orientées projet
- La documentation du programme utilisateur se compose de :
- Listage du programme utilisateur avec si possible des qualificateurs pour les signaux traités et les données
- Listes croisées de toutes les données traitées (Entrée/Sortie, fonctions internes comme les données mémorisées en interne, temporisateurs, compteurs etc.)
- Commentaires
- Description des modifications
- Manuel de maintenance